UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence
Master Thesis

Neural Style Transfer on Audio

by

Baris Demirdelen

11104082

August 2017

36EC
2016-2017

Supervisor:
MSc Karen Ullrich

Assessor:
Dr. Zeynep Akata

Informatics Institute, Graduate Schools of Science

"Let's think the unthinkable, let's do the undoable. Let us prepare to grapple with the ineffable itself, and see if we may not eff it after all."

Douglas Noel Adams

UNIVERSITY OF AMSTERDAM

Abstract

Informatics Institute
Graduate Schools of Science

Msc Artificial Intelligence

Neural Style Transfer on Audio

by Baris Demirdelen

Generative stylistic image models have flourished with the findings of style transfer methods, but converting the findings in image domain to audio domain turned out not to be trivial. In this thesis, approaches to apply style transfer methods to fixed length audio signals are researched. Moreover, a new style transfer based algorithm is presented to apply style transfer to indefinitely long audio signals.

Acknowledgements

I would like to thank to my great advisor and supervisor Karen Ullrich for all the mentorship. I would also like to thank Zeynep Akata for agreeing to be my assessor. Finally i would like to thank to Iclal Elgun Sarp, Merter Demirdelen, Doga Demirdelen, Umit Sarp and Gizem Yukcu for all the support and encouragement during this period.

Contents

List of Figures

List of Tables

Abbreviations

FFT	**F**ast **F**ourier **T**ransform
STFT	**S**hort **T**ime **F**ourier **T**ransform

Dedicated to my late cats, Mavis and Ekuri...

Chapter 1

Introduction

In machine learning and computer vision, there have been constant research about generative methods. One generative method that worked so well on images is the so called neural style transfer method, which creates images based on a content image, but having the style of a given style image.

Style transfer methods have flourished with the discoveries of Gatys et al.[9][8], and after the initial discoveries many have researched additional ways to improve the style transfer method.

In this thesis, the objective is to implement a feasible style transfer method, that would work on audio signals. The resulting method would generate an audio signal, in the same structure as a given content signal. An additional constraint for this generated signal would be to have the stylistic characters of another given style signal.

This thesis will be structured as follows: First, some background information about existing style transfer models and audio representations will be given. After that, parameter analysis for style transfer on audio will be conducted. Following, created models for this thesis will be explained. Later, experiments will be conducted on the created models. Discussion will include justifications and remarks for the experiment results. Before concluding, the windowed model will be showcased with different musical style transfers. Conclusion will provide further research topics and finishing thoughts about the thesis.

Since this is an audio heavy thesis, it is encouraged to listen to the audio recordings along the way. All the audio clips for analysis and experiments can be found in the website of this thesis[6].

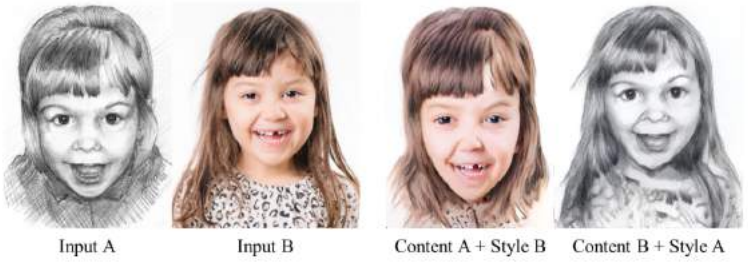

FIGURE 1.1: Illustration of style transfer on images taken from [11]

Chapter 2

Background

2.1 Style Transfer on Images

For images, Gatys et al.[8] proposed a way to separate content and style, and transfer one images style to another using 2D convolutional networks and called this process style transfer. Images are fed into a pretrained convolutional neural network, and then from the output of a layer, style loss and content loss of that layer can be calculated.

Let $\vec{p}$ the content input and $\vec{x}$ the generated image, P^l and F^l be the feature representation in layer l. Content loss for layer l is calculated as follows:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 \tag{2.1}$$

One layer is chosen for content loss and total content loss is the content loss of that layer.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}) = \mathcal{L}_{content}(\vec{p}, \vec{x}, l) \tag{2.2}$$

Style loss of a layer is calculated as follows:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}, l) = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \tag{2.3}$$

where A and G are gram matrices defined as:

$$G_{ij}^l = \sum_k Fik^l Fjk^l \tag{2.4}$$

More than one layer can be chosen for style loss. Total style loss is:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l \mathcal{L}_{style}(\vec{a}, \vec{x}, l) \tag{2.5}$$

Then the total loss is calculated as follows:

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x}) \tag{2.6}$$

where α is content factor and β is style factor. Increasing the content factor makes the generated image resemble more of the content and increasing the style factor makes the generated image resemble more of the style.

A random image $\vec{x}$ can then be optimized using L-BFGS-B[18] minimizing $\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x})$. After a few hundred iterations, $\vec{x}$ starts to have the content of $\vec{p}$ and the style of $\vec{a}$. This process is therefore called style transfer.

2.2 Texture Synthesis on Images

If in Equation 2.6 content factor α is set to 0, then the generated image becomes independent from content image and only has the style of the style image. This process is then called texture generation or texture synthesis.

Prior to the style transfer model, Gatys et al. [9] proposed the texture synthesis model and used deep convolutional VGG Networks[12] with successful results.

Champandard[1] discovered that random weights can also be used for texture synthesis, and he produced excellent quality results with the random model.

Ustyuzhaninov et al.[16] also proposed shallow models for texture synthesis. They also used random networks instead of a pretrained network and showed comparable results to deep pretrained networks.

2.3 Audio representation

Two types of audio representation will be used.

2.3.1 Raw Audio

Raw audio stores uncompressed audio in raw form. In this thesis, it consists of a single channel(mono) signal, each sample is 16-bit integer and sampling rate is 16 khz, meaning every second of audio correspond to 16000 samples. Raw audio representation is the simplest representation and is successfully used in discrimination and generation tasks.

Dai et al. [4] used deep convolutional networks of raw audio for urban sound classification. Their largest network is a 34 layered strictly convolutional network and classifies on a global average pooling layer, without using any fully connected layers.

Oord et al. [17] proposed the WaveNet structure which is a highly successful generative model for raw audio. The innovative idea is to use causal time dilated convolutions instead of traditional convolutions. They use the output of causal layers to generate one sample at a time. The use of time dilated layers in 1D signals allows capturing deep features in shallow layers. In this thesis, most neural network architectures will have time dilated convolutional layers.

As of the date of this thesis, no successful style transfer experiments on raw audio exist.

2.3.2 Spectrogram

To convert raw audio to spectrogram, short time fourier transform(STFT) is calculated first. For this, audio is divided into overlapping windows and each window is fourier transformed.

$$STFT\{x[n]\}(m,w) \equiv X(m,w) = \sum_{n=0}^{N} x[n]w[n-m]e^{-jwn} \tag{2.7}$$

Then spectrogram can be calculated as:

$$spectrogram\{x(t)\}(\tau,w) \equiv |X(\tau,w)|^2 \tag{2.8}$$

Spectrogram captures the time frequency domain as opposed to raw audio, which only captures the time domain. For audio, frequency information is really useful and spectrogram representation helps the models capture this information easily.

Costa et al. [2] used spectrogram representation for classification of music with many datasets. They treat spectrograms as 2D images with time and frequency as their axis so 2D convolutional layers are a perfect fit.

Ulyanov [15] tried style transfer on spectrograms with limited success. One interesting idea they used is that instead of using 2D convolutions, 1D convolutions are used. In this form, spectrogram is treated as multiple time signals and frequency bins are treated as different channels for the signal. Their model also differs from Gatys et al. with respect to model depth. While Gatys et al. uses pretrained very deep VGG networks, Ulyanov uses a one layered shallow network with random weights.

Chapter 3

Parameter Analysis

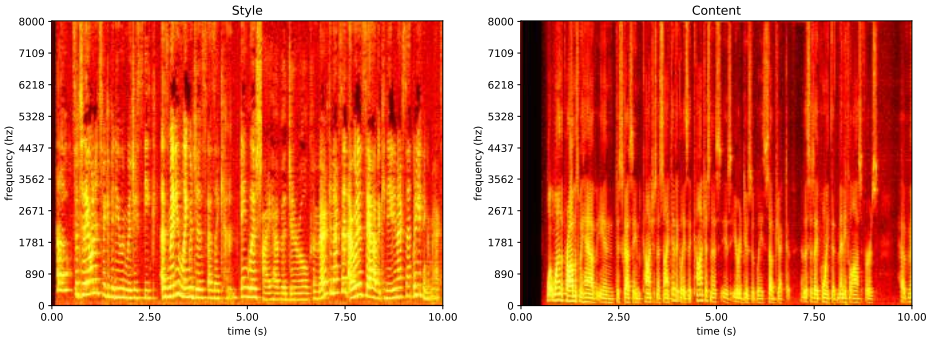

FIGURE 3.1: Inputs for parameter experiments

3.1 Layer Depth

Style transfer process requires style layers and a content layer to be chosen from a network. In images, shallow content layers and deep style layers are shown to work well by Gatys et al[12], although Ustyuzhaninov et al.[16] proposed shallow layers for both style and content.

In this section, layer depth is experimented for audio domain. For the experiment, neural network in Table 3.1 is used. In the convolutional layers, kernel describes the convolution kernel size for each layer, filters describes how many convolutional filters are in in that layer, dilation describes the time dilation of the layer.

For simplicity, in each experiment a single style layer s and a content layer c is chosen. Results are shown in Figure 3.2.

It is observed that a shallow style layer produces best results. As the style layer goes deeper, results become noisier. Results also start to fail at capturing style as the style layer go deeper. For the content layer, a shallow layer is also the best. There isn't a drastic difference between shallow and deep content layers, but shallow layers produce less noise.

layer	input	properties
conv1_1	$\vec{x}$	kernel: 3, filters: 2048, dilation: 2
conv2_1	*conv*1_1	kernel: 3, filters: 2048, dilation: 2
conv3_1	*conv*2_1	kernel: 3, filters: 2048, dilation: 2

TABLE 3.1: Layer depth experiment architecture

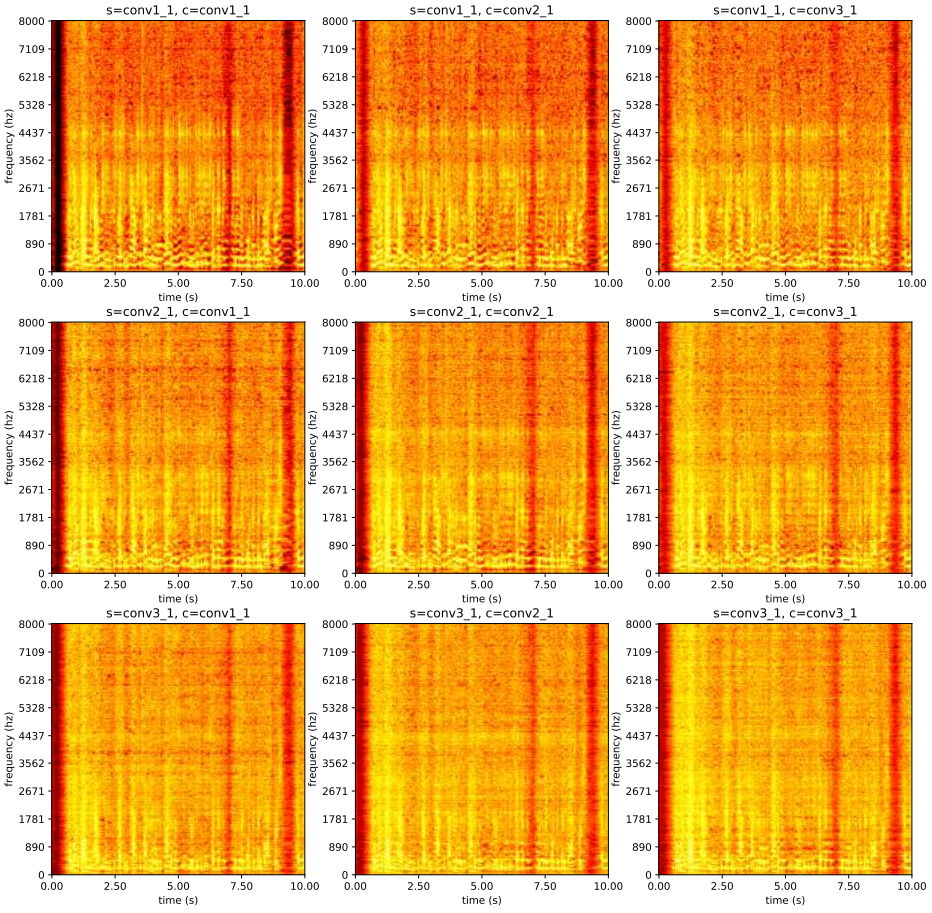

FIGURE 3.2: Effect of Layer Depth: Each row represents a style layer and each column represents a content layer.

3.2 Merged Style Layer

Style transfer process allows for choosing multiple style layers, but those layers can also be combined into one large layer if they all have the same size, thus having a larger gram matrix. In this section, the effect of choosing multiple style layers versus merging style layers into one single layer is experimented. For the experiment, neural network architecture from Table 3.4 is used. For multiple style layers experiment, style layers are chosen as conv1_1, conv1_2, conv1_3, conv1_4. For merged style layers experiment, style layer is chosen as conv1_merge. For both experiments content layer is chosen as conv1_merge. Results are shown in Figure 3.3.

It is observed that merging style layers results in higher quality style transfer overall. Using multiple style layers produces multiple style losses as per Equation 2.3 which are then summed up for total style loss. On the other hand using one large layer produces one single style loss, but since that loss is generated from the difference of larger Gram matrices, minimizing that loss produces a better result.

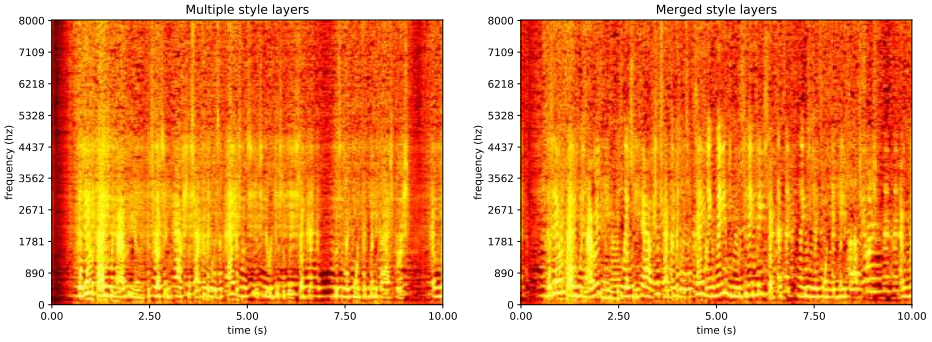

FIGURE 3.3: Effect of multiple style layers versus single merged style layer

3.3 Filter Size

In this section, the effect of different filter sizes on the style layers are experimented. For the experiment, neural network architecture from Table 3.2 is used. Both style layer and content layer is conv1_merge. Results with different values of ff are shown in Figure 3.4.

It can be seen that for $ff = 4$ the result is noisy and low quality. For results of $ff = 16$ and $ff = 32$ there isn't much difference; they both are less noisy and of high quality. The main difference between $ff = 16$ and $ff = 32$ is the model with $ff = 16$ produces the results in ≈ 30 seconds and model with $ff = 32$ produces the model in ≈ 80 seconds.

It is inferred that since the filter size determines the size of the Gram matrix, lesser sizes directly reduces the quality of the style transfer. Therefore in general, higher filter sizes are preferred in the style layers, but the benefits diminish with greater filter sizes.

layer	input	properties
conv1_1	$\vec{x}$	kernel: 3, filters: $128 \times ff$, dilation: 1
conv1_2	$\vec{x}$	kernel: 3, filters: $64 \times ff$, dilation: 2
conv1_3	$\vec{x}$	kernel: 3, filters: $32 \times ff$, dilation: 4
conv1_4	$\vec{x}$	kernel: 3, filters: $16 \times ff$, dilation: 8
conv1_merge	conv1_[1..4]	filters: $240 \times ff$

TABLE 3.2: Filter size experiment architecture

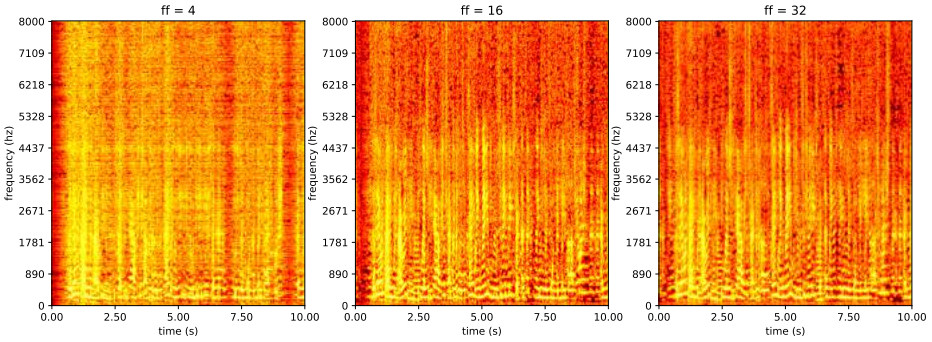

FIGURE 3.4: Effect of Filter Size

3.4 Kernel Size

In this section, the effect of different kernel sizes on convolutional layers are experimented. For the experiment, neural network architecture from Table 3.3 is used. Both style layer and content layer is conv1_merge. Results with different values of k are shown in Figure 3.5.

It is observed that as kernel size k increases, resulting sound bits starts to spread over time axis. This introduces noise around the sound bits, so lower kernel size is preferred.

layer	input	properties
conv1_1	$\vec{x}$	kernel: k, filters: 2048, dilation: 1
conv1_2	$\vec{x}$	kernel: k, filters: 1024, dilation: 2
conv1_3	$\vec{x}$	kernel: k, filters: 512, dilation: 4
conv1_4	$\vec{x}$	kernel: k, filters: 256, dilation: 8
conv1_merge	conv1_[1..4]	filters: 3840

TABLE 3.3: Kernel size experiment architecture

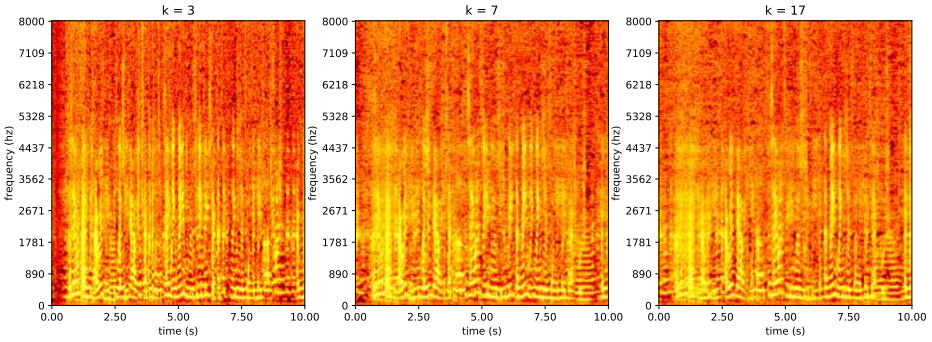

FIGURE 3.5: Effect of Kernel Size

3.5 FFT Size

In this section, the effect of different FFT sizes for spectrograms are experimented. For the experiment, neural network architecture from Table 3.4 is used. Both style layer and content layer is conv1_merge. Results with different values of fft are shown in Figure 3.6.

With $fft = 2048$, the result is very noisy, and frequencies are a little bit mixed. $fft = 512$ and $fft = 1024$ are both good in quality, but there is a slight difference. With $fft = 512$ it is observed that sounds are more condensed, the difference with $fft = 1024$ can be observed in silence bits, especially in 9.5 seconds mark that $fft = 512$ ends the sound immediately, where $fft = 1024$ provides a more natural continuation. It is found that both FFT sizes are feasible in style transfers and selecting one is likely a matter of taste.

layer	input	properties
conv1_1	$\vec{x}$	kernel 3 filters 2048 stride 1 dilation 1
conv1_2	$\vec{x}$	kernel 3 filters 1024 stride 1 dilation 2
conv1_3	$\vec{x}$	kernel 3 filters 512 stride 1 dilation 4
conv1_4	$\vec{x}$	kernel 3 filters 256 stride 1 dilation 8
conv1_merge	conv1_[1..4]	filters 3840

TABLE 3.4: FFT size model architecture

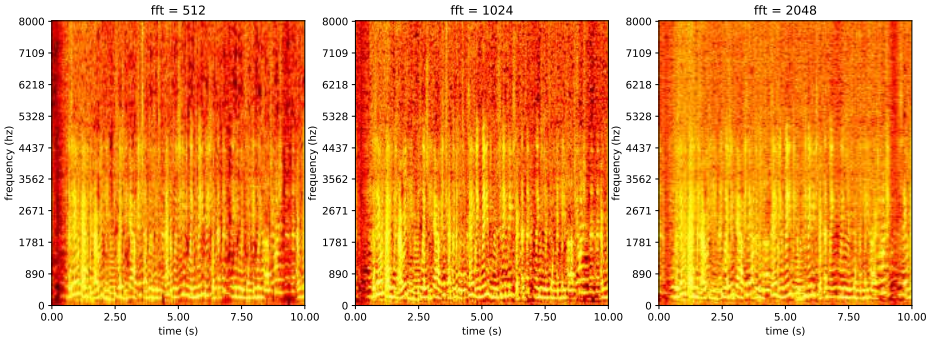

FIGURE 3.6: Effect of FFT Size

Chapter 4

Models

4.1 Style transfer for audio

The findings in image domain can be converted into audio and style transfer can be done with similar models. In this case, the objective is to separate content and style of an audio, and transfer the style while keeping the content structure same. This chapter contains models experimented.

4.2 Raw audio transfer

Raw audio is the most intuitive representation for audio. The signal can be fed into the neural network without any processing and a raw audio signal can be generated without any reverse processing.

To gather useful information about audio, the model should gather at least simple sounds. Since the audio is of high sample rate, even the simple sounds have high number samples. Comparing to networks working on images, a small kernel size for the network is usually 3x3 pixels. In audio domain though, it should be at least a couple hundred milliseconds. Sixteenth of a second is a good point to capture simple sounds. Since the sample rate of the audio is chosen as 16000, the smallest kernel size is therefore

$$sample\_rate/16 + 1 = 1001$$

One shortcoming of this model is that since the kernel size is very large, it is very slow and memory intensive.

Content audio $\vec{p}$ and style audio $\vec{a}$ are given as inputs. $\vec{x}$ is initialized equal to $\vec{p}$. Neural network shown in Table 4.1 is created.

layer	input	properties
conv1_1	$\vec{x}$	kernel 4001 filters 128 stride 1
conv1_2	$\vec{x}$	kernel 2001 filters 128 stride 1
conv1_3	$\vec{x}$	kernel 1001 filters 128 stride 1
conv1_merge	conv1_[1..3]	filters 384

TABLE 4.1: Raw model architecture

Style layers set L initialized as $L = \{conv1\_merge\}$. Content layer is also set to $conv1\_merge$. Total loss can then be calculated as Equation 2.6. This loss is minimized with respect to $\vec{x}$ over 300 iterations of L-BFGS-B and the resulting $\vec{x}$ is outputted.

4.3 Spectrogram transfer

For the following models, the spectrogram of the input style and content is calculated with Equation 2.8. Generated audio is not a raw audio signal anymore, but a spectrogram. For style transfer, initial spectrogram is set equal to content spectrogram and fed into the network. Minimizing the style transfer loss, the spectrogram is optimized directly.

For the neural network, spectrograms are interpreted as 1D signals with $fft\_size/2$ channels. All convolutional layers in the models are therefore 1D convolutional layers.

After the optimization is done, spectrogram is reverted back to audio using LSEE-MSTFTM algorithm provided by Griffin and Lim [10]. It is an iterative algorithm to estimate signal from stft magnitude, as shown in Figure 4.1.

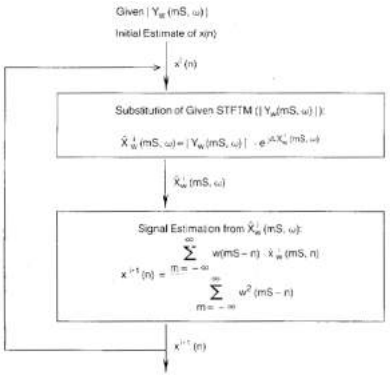

FIGURE 4.1: LSEE-MSTFTM algorithm taken from [10]

4.3.1 Denoiser Model

An audio denoiser model is created in the auto-encoder form as shown in Table 4.2. Input $\vec{x}$ is a spectrogram with FFT size 512. Output of conv_out $\vec{y}$ is a same shaped spectrogram as input. Inverse convolution layers are implemented using Tensorflow's conv2d_transpose function [14].

Model is trained using free audio books compiled by Open Culture [3]. Audio books are split into 30 second segments and converted into spectrograms. Training is done with adam optimizer with learning rate 1.0×10^{-5} and batch size of 8. Loss function is mean squared errors of $\vec{x}$ and $\vec{y}$. A small gaussian noise is added to input in spectrogram form for training. The model then trained until convergence which took 2 weeks. Resulting model showed good performance in reconstructing the spectrogram given as input. Figure 4.2 shows a sample input and corresponding reconstructed output by the trained model. It can be seen as though the output is a bit blurry, all the defining features of input are captured in the output.

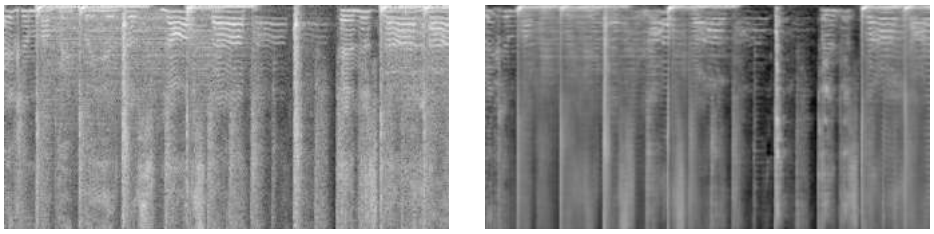

FIGURE 4.2: Trained denoiser sample input-output. Input spectrogram on the left and corresponding reconstructed output spectrogram on the right.

For style transfer, style layers are chosen as conv1_1, conv2_1 and conv3_1 and content layer is chosen as conv3_1.

layer	input	properties
conv1_1	$\vec{x}$	kernel 1 filters 512 stride 1
pool1	conv1_1	pool size 2
conv2_1	pool1	kernel 5 filters 1024 stride 1
pool2	conv2_1	pool size 2
conv3_1	pool2	kernel 5 filters 2048 stride 1
pool3	conv3_1	pool size 2
inv_conv3_1	pool3	kernel 5 output filters 2018
inv_conv2_1	inv_conv3_1	kernel 5 output filters 1024
inv_conv1_1	inv_conv2_1	kernel 5 output filters 512
conv_out	inv_conv1_1	kernel 5 filters 257

TABLE 4.2: Denoiser model architecture

4.3.2 Classifier Model

Another spectrogram model is created to classify some properties of instruments using the NSynth dataset [13]. NSynth is a large scale dataset of annotated musical notes recently published by Magenta Team. It contains 305,979 musical notes, each with a unique pitch, timbre, and envelope. The dataset is used to classify following properties of the recordings:

- Pitch: Frequency ordering value of the instrument, a value between [0,127] in the dataset, normalized into [0,1] for neural network.
- Velocity: A measure of how forcefully the note is played on the instrument, a value between [0,127] in the dataset, normalized into [0,1] for neural network.
- Qualities: 10 binary qualities of the sound, including following qualities: bright, dark, distortion, fast decay, long release, multiphonic, nonlinear envelope, percussive, reverb, tempo synced.
- Instrument family: Class of the instrument, one hot encoded as 11 values into output. Must be one of: bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, synth lead, vocal.
- Instrument source: Class of the source, one hot encoded as 3 values into output. Must be one of: acoustic, electronic, synthetic.

These properties combined makes a total of 26 features in the output layer of the network. Loss function to minimize when training is the sum of the following:

- Mean squared error for pitch and velocity
- Sigmoid cross entropy for qualities
- Softmax cross entropy for instrument family and instrument source.

Training is done with adam optimizer with learning rate 1.0×10^{-4} and batch size of 100. 3 epochs were enough for convergence and training took 2 days. In the end the model showed 68.85% accuracy on the test set.

The architecture of the model is shown in Table 4.3. Input $\vec{x}$ is a spectrogram with FFT size 1024. conv1_merge layer merges all the layers into one layer with 4080 filters. fc3 is the fully connected output layer with 26 features.

For style transfer, style loss is calculated on conv1_merge and content loss is calculated on conv3_1.

layer	input	properties
conv1_1	$\vec{x}$	kernel 3 filters 2048 stride 1 dilation 1
conv1_2	$\vec{x}$	kernel 3 filters 1024 stride 1 dilation 2
conv1_3	$\vec{x}$	kernel 3 filters 512 stride 1 dilation 4
conv1_4	$\vec{x}$	kernel 3 filters 256 stride 1 dilation 8
conv1_5	$\vec{x}$	kernel 3 filters 128 stride 1 dilation 16
conv1_6	$\vec{x}$	kernel 3 filters 64 stride 1 dilation 32
conv1_7	$\vec{x}$	kernel 3 filters 32 stride 1 dilation 64
conv1_8	$\vec{x}$	kernel 3 filters 16 stride 1 dilation 128
conv1_merge	conv1_[1..8]	filters 4080
conv2_1	conv1_merge	kernel 3 filters 512 stride 1 dilation 2
conv3_1	conv2_1	kernel 3 filters 256 stride 1 dilation 2
conv4_1	conv3_1	kernel 3 filters 128 stride 1 dilation 2
conv5_1	conv4_1	kernel 3 filters 64 stride 1 dilation 2
conv6_1	conv5_1	kernel 3 filters 32 stride 1 dilation 2
fc1	conv6_1	units 512
fc2	fc1	units 512
fc3	fc2	units 26

TABLE 4.3: Classifier model architecture

4.4 Windowed Style Transfer

Models discussed in previous section all work on short, fixed size style and content inputs and they are not good at generating longer audio clips. There is an inherent problem with longer clips; longer clips reduce style transfer quality.

Style loss is calculated from Gram matrices of the feature layers with Equation 2.4. One quality of these gram matrices is that they are always a square matrix with the size of the number of filters in that layer. Their size is independent of the clips size. Therefore with short clips, more information can be stored in the same sized matrix about the style, whereas longer clips need larger gram matrices to store the same density of information.

One possible solution to this problem would be to increase the filter size of the style layers, but this size is limited by GPU memory. Moreover, even if the filter sizes are increased, shorter style transfers usually still have more style quality than longer style transfers because of gram matrix density, so shorter style clips are always preferred.

Another solution would be to keep the content clip size long while keeping the style clip size short. This would mean that longer clips can also have style transferred, but only from a short style. This may mean that since the style clip is short, transfer results may be very repetitive in terms of style.

On the other hand, to transfer a full piece of music, long content clips with long style clips are required. To overcome this issue, following model is proposed.

Content clip is split into overlapping time-frames, and for each content time-frame, the best style time-window is selected to transfer the style to it. This idea resembles Li an Wand's markov random fields style transfer algorithm for images [11]. Their algorithm also select small content patches and select the best corresponding style patch for transfer, but the main difference is, with their algorithm even though small patches are selected, the whole operation graph is stored in GPU memory and gradients are extracted for whole style and content images. On the other hand with this proposed model, Only the small patches are optimized, one patch at a time, so the operations are fast and not memory intensive.

For the style transfer neural network, a small extract from Classifier Model is selected. Neural network architecture is shown in Table 3.4. For both style and content layers, conv1_merge is selected.

Pseudocode for the windowed algorithm is shown in Algorithm 1. Function *get_best_style*() loops through style clip time-frames and returns the time-frame where style loss in Equation 2.5 is minimal. Function $style\_transfer()$ is the actual style transfer function, which transfers the style of content and style patches and returns the result.

Algorithm 1 Time windowed style transfer

function TIME_WINDOWED_SPECTROGRAM_TRANSFER($\vec{a}, \vec{p}, t_h, w$)
    $\vec{x} \leftarrow \vec{p}$
    for t *hops by* t_h *from* 0 *to* $length(\vec{p})$ **do**
        $\vec{p_t} \leftarrow \vec{p}[t : t + w]$
        $\vec{x_t} \leftarrow \vec{x}[t : t + w]$
        $\vec{a_t} \leftarrow get\_best\_style(\vec{a}, \vec{x_t}, t_h, w)$
        $\vec{x}[t : t + w] \leftarrow style\_transfer(\vec{a_t}, \vec{p_t}, \vec{x_t})$

Chapter 5

Experiments

As audio can represent a lot of different types, three sets of experiments are made with the models. These experiments contain different types of audio as style and content inputs. They are classified as chromatic scale, speech and music experiments. All of the experiments contains a style input and a content input of 10 seconds except time windowed model, which has a content input of 10 seconds but a style input of 40 seconds. All the clips are with sampling rate of 16000. Each experiment contains an output optimized over 300 iterations of L-BFGS-B. In the experiments content factor $\alpha = 5.0 \times 10^{-9}$ and style factor $\beta = 1.0$

5.1 Chromatic Scale

The chromatic scale is a musical scale with twelve pitches, each a semitone above or below another. A chromatic scale contains all the semitones in modern western music when played fully.

In this experiment a violin playing chromatic scale is chosen for content and a piano playing chromatic scale is chosen for style. The simplest form of style transfer should play the same chromatic scale the violin is playing, but with piano sounds. Figure 5.1 contains spectrograms of the piano as style, violin as content, and style transfer outputs of all four models.

Raw audio model spectrogram is very blurry, and when listened, although the piano sounds are present, they are very noisy. The blurry spectrogram tells the signs of needing more style filters, although it was not possible because of GPU memory limitations.

Denoiser with random weights capture both content and style pretty well, but the piano sounds are incredibly dense. This can usually be overcome by increasing the filter size in layers, with a trade-off of adding noise.

Denoiser with trained weights capture neither the content nor the style.

Both classifier models emphasize the starting points of content scales. Untrained model is inaudible besides the emphasized points, trained model captures the content in low volume but no style is present.

Time windowed model transfers the style quite good. Violin sound is not present anymore. I can be seen in the spectrogram that violins continuity vanished and instead strictly divided piano notes follow the content. Although there are some piano notes missing and some notes are an octave higher than the intended notes.

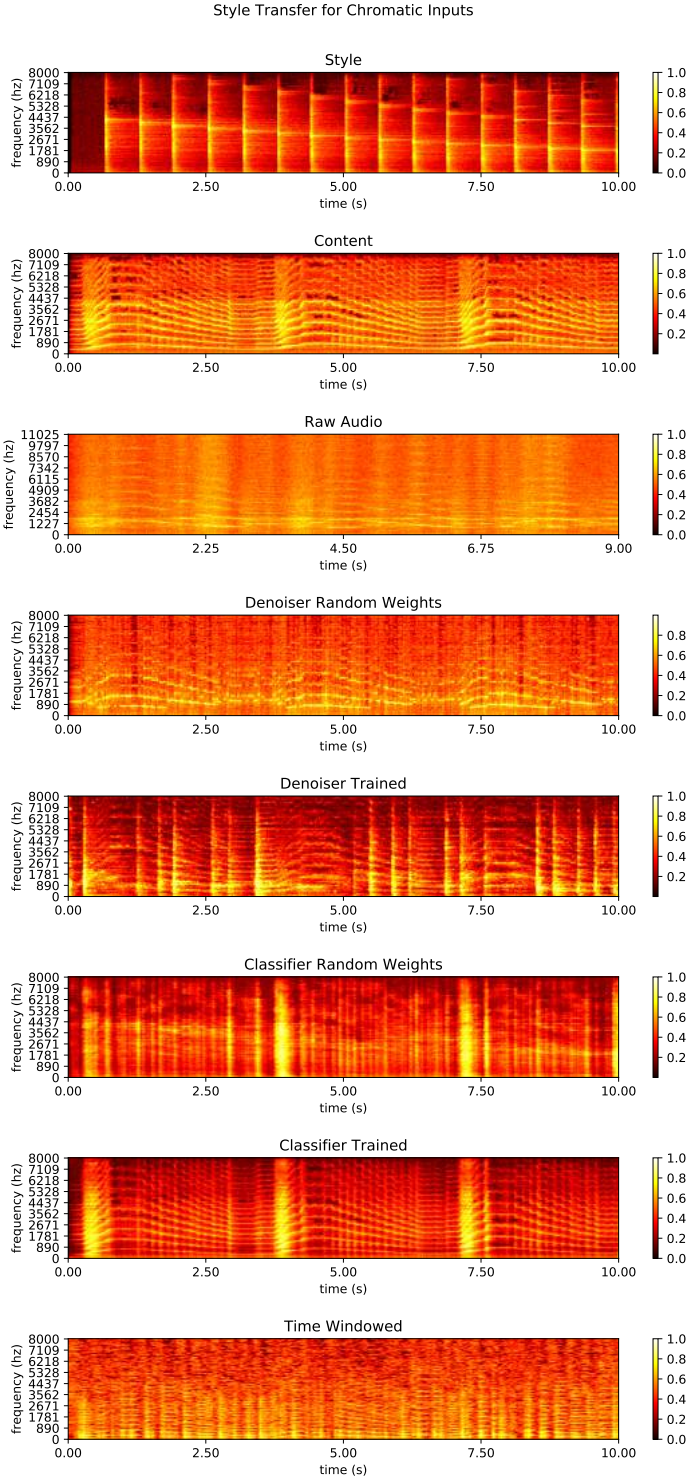

FIGURE 5.1: Style transfer for chromatic inputs

5.2 Speech

These set of experiments focus on transferring the speech from one voice to another. Speech is very different from chromatic scale, because it contains many small frequency changes in short time, compared to one long note at a time. As inputs, two free audio books are chosen. As style input, "As I Walked Out One Evening" poem by W. H. Auden is cited by a male voice. As content input the book "The Giving Tree" by Shel Silverstein is read by a female voice. The expectation is that the output would be a male voice reading "The Giving Tree". Figure 5.2 contains spectrograms of the male voice as style, female voice as content, and style transfer outputs of all four models.

Raw model is again very noisy, the female voice is still heard, and there are some male textures but not clear at all.

Denoiser with random weights shows good style transfer. The male voice from the style is prominent and the words from content can be heard although not very clearly.

With denoiser with trained weights, the words can be heard very clearly, and although the voice is more masculine, it is not of the voice of the style. It instead feels more like a low pass filtered content voice.

Classifier with random weights is even more low pass filtered than denoiser with trained weights, but it also resembles some voices from the style. The words are not clear.

Classifier with trained weights is just like the original content, with low frequencies boosted a bit, although it is nowhere near the style voice.

Time windowed model again shows the best results. Transfer quality is along the lines of denoiser with random weights, but the words are clearer. In silence bits slight artifacts of male voice can be heard.

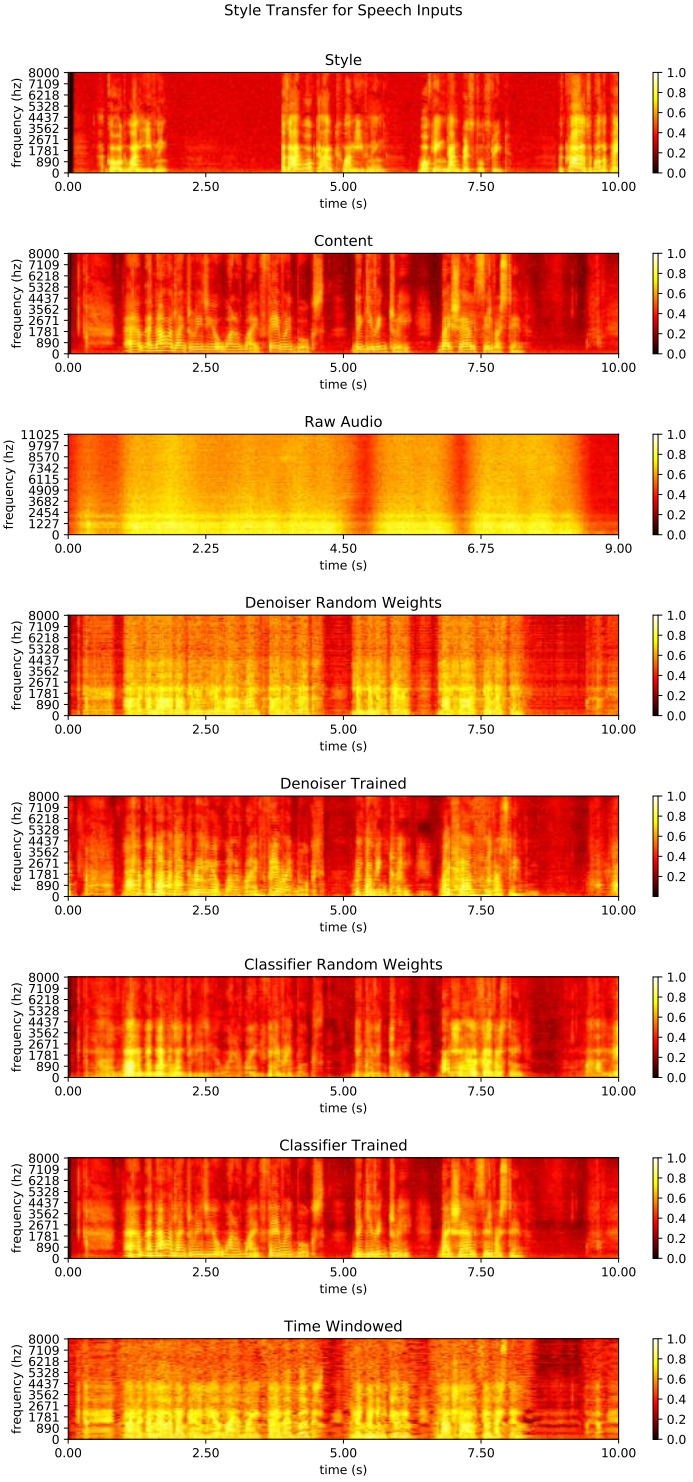

FIGURE 5.2: Style transfer for speech inputs

5.3 Music

In this set of experiments, two musical inputs are given as style and content inputs. In contrast to chromatic scale experiments, these audio recordings contain multiple instruments with rhythm, as well as vocals. Therefore the recordings in this set is more complex than both chromatic scale and speech recordings. Style input is "Green Onions" by Booker T. & The M.G.'s, a 60's soul and blues song, and content input is the popular song "Get Lucky" by Daft Punk. Ideal output would be "Get Lucky" played in a blues tune. Figure 5.3 contains spectrograms of the "Green Onions" as style, "Get Lucky" as content, and style transfer outputs of all four models.

Raw model mixes style and content but the output is still very noisy and the spectrogram looks blurry.

Denoiser with random weights transfers the style mediocrely, the guitar of "Get Lucky" is present on top of the "Green Onions" bass-drum-electric piano combo, but the guitar sounds are very scratchy and the overall quality of the sound isn't very good.

Trained denoiser contains most of the content intact, although emphasizing the start of the beats a bit more. There are some textures from style present but not enough, and the whole audio sounds like it has been high pass filtered.

Classifier with random weights has a low pass feel overall, with one occurrence of high frequency emphasis. Despite of low quality audio, the transfer seems quite good, as a blues tuned "Get Lucky" can be heard as intended.

Trained classifier has the guitar of content intact, backed up with style textures. Audio quality is good and listenable.

Windowed model is more noisy than the trained classifier, but the audio is also more alive and enjoyable. Guitar of "Get Lucky" blends in almost perfectly with the electric piano of "Green Onions". Some texture variance is available and keeps the audio clip interesting.

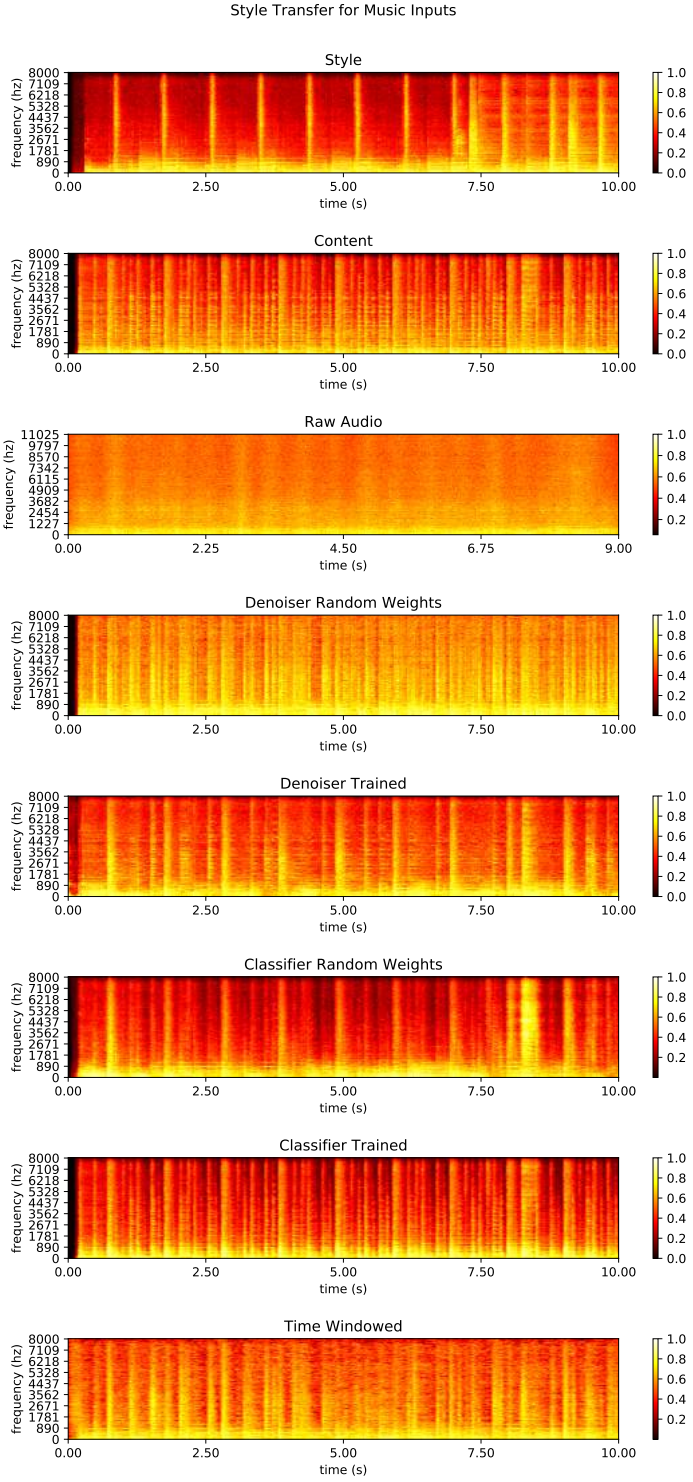

FIGURE 5.3: Style transfer for music inputs

Chapter 6

Discussion

To summarize the experiments, raw model always performed worst and windowed model usually performed best. Trained models usually produced worse result than random weighted counterparts. In this chapter these findings are discussed more in depth.

First of all, raw model was always the slowest and its results were the noisiest. This can be explained by lack of style features for the transfer. Because initial audio is always set to content audio, the noise comes from style loss. More style features are required for a less noisy style transfer, but GPU memory is not enough to hold more. On the other hand spectrogram models reduce the space complexity greatly therefore can hold more style layers, which results in them performing better than the raw model.

Secondly, trained models needs a bit investigation since they don't perform better than the random weighted models. The main observation in common with all the experiments is that trained models produces filtered down results, especially low pass or high pass filters. In theory, training makes convolutional layers converge into different filters, but it seems in practice, they mostly converged into similar low pass or high pass filters. It is also apparent in the test output of denoiser network in Figure 4.2 that the output looks filtered down.

It is also interesting to note that even though denoiser model was especially trained for speech, denoiser with random weights produce better results in speech than the trained one. Similarly, even though the classifier model was trained on singular notes, which resemble more the chromatic experiment, classifier model performed very poorly on the chromatic experiment.

To compare the denoiser model with the state of art, Engel et al.[7], trained two auto-encoder models with NSynth dataset, a spectrogram auto-encoder as baseline, and a WaveNet auto-encoder. WaveNet auto-encoder performed significantly better, but the

reconstruction is still audibly very different. Even though the denoiser model in this thesis works on spectrograms, structure-wise it is more similar to the WaveNet auto-encoder of Engel et al., since both models use time dilated 1D convolutions. The results are also very similar of both models, with both outputting a filtered down version of the original audio.

For the classifier model, main intuition is that classifier model convolutions are allowed to emphasize on some features, and nullify some features on the spectrogram, since a reconstruction of original spectrogram is not necessary for training. This explains why we see classifier model emphasizing some style features too much, and be inaudible at some features.

Moreover, style transfer using random networks such as Champandard[1], Ustyuzhaninov et al.[16] and Ulyanov[15] were already discussed before and since their random models produce on par results with pretrained models for style transfer and texture synthesis, in accordance to the experiments, it can be concluded that random models tend to perform better in audio domain than the trained models, because of the drawbacks of trained models in audio.

On the other hand, time windowed model performed well in all experiments, but excelled in style transfer for music. To be able to transfer the style from different patches to the content proved to keep the music interesting enough to be listenable and enjoyable. To further demonstrate the time windowed model, a longer style transfer experiment for music recordings will be showcased in the next chapter.

Chapter 7

Style Transfer on Long Musical Pieces

In this chapter, time windowed model is used to generate full music recordings of high quality. These experiments show the true potential of the model, not only it can generate indefinitely long musical pieces, generated results are also of high quality and enjoyable.

Since the audio clips in this section are very long, it may be hard to see the style transfer in the spectrograms, but all the clips in this section can also be listened from the thesis website[6].

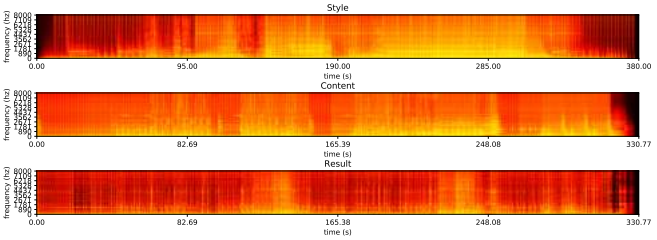

FIGURE 7.1: Showcase 1: "Teardrop" from Massive Attack stylized as "Angel" from Massive Attack.

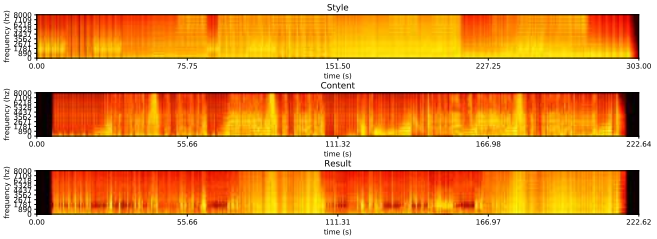

FIGURE 7.2: Showcase 2: "Another One Bites The Dust" from Queen stylized as "Freak" from LFO.

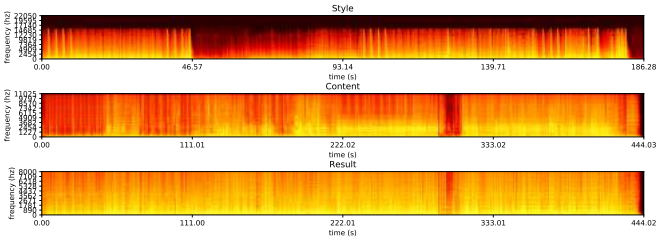

FIGURE 7.3: Showcase 3: "Since I've Been Loving You" from Led Zeppelin stylized as "The Imperial March" from John Williams.

Chapter 8

Conclusion

Generative stylistic image models have flourished with the findings of style transfer methods, but to converting the findings in image domain to audio domain turned out not to be trivial. In this thesis, approaches to apply style transfer methods to different kinds of audio signals are researched. Parameter analysis showed a path towards creating a good model, and proposition of the time windowed model enabled longer recordings to be style transferred in high quality.

Even though the time windowed model works well on musical pieces, improvements can be made. Sometimes the dynamic of the generated audio doesn't match the dynamics of the content audio, sometimes generated audio gets unnaturally loud. To produce production quality recordings, good mixing and audio balance is essential. Therefore a way to balance the generated audios levels should be researched.

This thesis proved that style transfer on audio can definitely be done, and it can produce enjoyable records. Hopefully this thesis will pave path towards full piece style transfers with production quality.

Appendix A

Code Release

The code for this thesis is released under MIT license in github link https://github.com/barisdemirdelen/generative_audio [5].

The code only works on: Python 3.6.

Dependencies:

- python >= 3.6
- tensorflow
- librosa
- scipy

Usage:

In src folder, type in terminal.

python windowed_style_transfer.py -c "content_wav_file" -s "style_wav_file"

The result will be produced in the output/result.wav

Appendix B

Website Structure

The audio for thesis is released in the website
http://barisdemirdelen.com/audio-style-transfer [6].

The directory structure goes along the same structure as the thesis.

In every experiment directory, there are following files:

- style.wav: Style input for the experiment
- content.wav: Content input for the experiment
- x.wav: Initial audio for generation, usually same as content.wav
- result.wav or output.wav: Style transfer result of the experiment

Appendix C

Computer Specifications for the Experiments

All of the experiments in this thesis are done in a computer with following specifications:

- CPU: Intel i5-4690K
- GPU: GeForce GTX 980 Ti
- RAM: 16GB
- OS: Arch Linux
- Kernel: x86_64 Linux 4.12.6-1-ARCH
- Python: 3.6
- Tensorflow: r1.2

Bibliography

[1] Alex J. Champandard. Extreme style machines. https://nucl.ai/blog/extreme-style-machines/, 2016. [Online; accessed 01-August-2017].

[2] Yandre MG Costa, Luiz S Oliveira, and Carlos N Silla. An evaluation of convolutional neural networks for music classification using spectrograms. *Applied Soft Computing*, 52:28–38, 2017.

[3] Open Culture. 900 free audio books. http://www.openculture.com/freeaudiobooks, 2017. [Online; accessed 01-August-2017].

[4] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. *CoRR*, abs/1610.00087, 2016.

[5] Baris Demirdelen. Code reposityory for neural style transfer on audio. https://github.com/barisdemirdelen/generative_audio, 2017. [Online; accessed 16-August-2017].

[6] Baris Demirdelen. Neural style transfer on audio. http://barisdemirdelen.com/audio-style-transfer, 2017. [Online; accessed 16-August-2017].

[7] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. *CoRR*, abs/1704.01279, 2017.

[8] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[9] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *CoRR*, abs/1505.07376, 2015.

[10] D. Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, Apr 1984.

[11] Chuan Li and Michael Wand. Combining markov random fields and convolutional neural networks for image synthesis. *CoRR*, abs/1601.04589, 2016.

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[13] Magenta Team. The nsynth dataset. https://magenta.tensorflow.org/datasets/nsynth, 2017. [Online; accessed 01-August-2017].

[14] Tensorflow. tf.nn.conv2d_transpose. https://www.tensorflow.org/api_docs/python/tf/nn/conv2d_transpose, 2017. [Online; accessed 01-August-2017].

[15] Dmitry Ulyanov. Audio texture synthesis and style transfer. https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/, 2016. [Online; accessed 01-June-2017].

[16] Ivan Ustyuzhaninov, Wieland Brendel, Leon A. Gatys, and Matthias Bethge. Texture synthesis using shallow convolutional networks with random filters. *CoRR*, abs/1606.00021, 2016.

[17] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[18] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997.