Music Style Transfer Using Pre-Trained Audio Autoencoders

Beril Besbinar Jade Maï Cock Paola Mejia Baran Özaydin Ehsan Pajouheshgar

***Abstract*—Convolutional Neural Networks (CNNs) are able to extract features that can encode both the style and the content information of an image. This particular property has made them the perfect candidates to transfer the style of one image onto another one. The quality of the resulting pictures motivates us to adopt the audio equivalent of image style transfer methods based on CNN representations to perform music style transfer. Most existing methods for audio style transfer operate on spectral representations instead of the raw waveform. Consequently, unlike current methods, we propose using auto-encoders trained in a self-supervised manner to transfer the timbre of one instrument to an audio sample played by another instrument. To that effect, we first experiment in the image domain and then apply the methods whose output were promising onto the audio domain. Our results show that the adoption of well-established methods for image style transfer to timbre transfer for audio signals is not straightforward.**

***Index Terms*—Style Transfer, Timbre transfer, JukeBox, CNN, Autoencoder**

I. Introduction and Related Work

The emergence of Deep Convolutional Neural Networks (CNNs) has opened the doors to many artistic applications, one of which is neural style transfer. As the pioneer work in the field, Gatys et al. [4] showed how representations encoded in the intermediate layers of a CNN, such as VGG19 [11], which is trained for image classification tasks, can be used to define the ”content” and the ”style” of an image. In that regard, they formulate the problem of neural style transfer as the generation of a new image guided by two input images, one for content and the other one for style. Their approach does not require training any neural networks. On the contrary, the target image is optimized to match the relevant characteristics of input images that are defined by featuremaps encoded by the pre-trained CNN via an iterative optimization process, which limits the application of their algorithm to real-time applications.

To overcome this problem, the authors of [7], [9] propose to use CNN decoders to directly decode the intermediate featuremaps to the output image. Then the idea is to perform some transformations on this intermediate representation of the content image to change its stylistic aspects. Li et al. [9] propose Whitening-Coloring Transform (WCT) that matches the Gram matrix[1] of the content image with the Gram matrix of the style image. Huang et al. [7] suggest a simpler method called Adaptive Instance Normalization (AdaIN) to simply swap the channel means and variances of the featuremaps[2]. However, both of these methods train CNN decoders on top of CNNs pre-trained for image classifications such as VGG19 [11].

Recently, there has been an increasing interest in adopting neural style transfer to audio signals [2], [5]. As an early attempt, Eric *et al.* experimented with different approaches to apply neural style transfer to audio signals based on the Short-Time Fourier Transform (STFT) representations. Their findings show that rather than initializing the target signal with random noise, modifying the content audio by a shallow neural network that is trained from scratch to match the style with the second input performs the best [5]. Similarly, Huang *et al.* [6] use Constant Q Transform (CQT) to map the audio signals to a two-dimensional space, where the timbre transfer is achieved with CycleGAN, and the target audio is generated by a conditional WaveNet from the resultant CQT representation. However, mapping the spectrogram representations back to audio signals pose its own challenges, mainly due to the ambiguity in the phase reconstruction step.

In contrast, few other works [1], [2] opted for working with one-dimensional signals by training a neural network that is explicitly designed or trained for audio style transfer. Bitton *et al.* trained an autoencoder with a discrete latent space (Vector-Quantized VAE, also known as VQ-VAE [12]) that model the given timbre distribution using a corpus of audio files recorded for a target sound domain [1]. Cifka *et al.* also train a VQ-VAE; however, in contrast to [1], they bypassed the need for paired training data as a result of their self-supervised learning scheme [2]. More specifically, they achieve the data-driven disentanglement of pitch and timbre by modeling a discrete representation for the content and a continuous representation for the style, where the disentanglement is mostly attributed to enforcing the same style representation for different segments of a given audio signal. Despite the encouraging results, the method involves training a CNN from scratch which require large computational resources.

In this project, we investigate the applicability of music style transfer[3] using pre-trained audio autoencoders that operate on raw audio waveform. We aim to use representations obtained by such an encoder analogous to image style transfer methods and avoid the phase reconstruction problem as the related decoder would map the representation back to raw audio signals.

We would like to thank Paolo Prandoni and Eric Bezzam for the amazing Computers and Music course.

[1]Gram matrix represents the correlations between channels of the featuremap.

[2]The channel variances are represented in the diagonal of the Gram matrix.

[3]We use style and timbre interchangeably in this report.

Using pre-trained autoencoders for audio style transfer has several benefits:

- There is no need to re-train the model for each instrument (zero-shot style transfer).
- There is no need for annotated data to train the model.
- Based on the selected method, we can perform style transfer in real-time without any need for optimization.
- With the help of the decoder module, we can directly generate the output audio and avoid the phase reconstruction problem.

Thus, in this project, we use the autoencoder module of Jukebox, which is a neural network that can generate music based on a variety of input conditions [3]. The Jukebox works with raw audio signals, and to cope with high dimensionality and compress the raw audio to a lower-dimensional space, it employs a hierarchical vector-quantized variational autoencoder. The autoencoder has 2 million parameters and it is trained with 1.2 million songs, which makes it a good candidate for the proposed approach. In particular, we adopt the methods introduced in [4], [7], [9] for music style transfer using the representations obtained by Jukebox autoencoders. To the best of our knowledge, there is no work exploiting autoencoders trained without supervision for music style transfer[4] We use different metrics to quantitatively compare the style transfer quality of different methods. We also provide the generated results for qualitative comparison.

In Section II, we discuss the details of our proposed methods. Then, we present the dataset and provide quantitative and qualitative analysis of the outcomes in Section III-D. Finally, in Section V, we present possible reasons explaining why the image style transfer methods do not perform well on music signals and discuss related challenges. In the supplementary material, we provide some of the experiments we performed to understand Jukebox. We also provide the image equivalent of our music style transfer experiments in support of the claim that autoencoders are capable of style transfer.

II. Proposed Methods

Our main hypothesis is that image style transfer methods such as [4], [7], [9], can be effective for modeling the timbre in music while preserving the harmonic and melodic structure. These methods require rich features of the input signal and we propose using [3] as a feature extractor.

We first briefly review the JukeBox autoencoders in II-A. Then, we present four different methods we tried for style transfer in Sections II-B, II-C, II-D, II-E.

A. Jukebox

As shown in figure 1, the Jukebox autoencoder model uses a three level VQ-VAE architecture with different temporal resolutions. At each level, the raw input audio is encoded into latent vectors. Then, the vectors are quantized to the closest codebook vector. The decoder then takes the sequence of codebook vectors and reconstructs the audio.

In our work, we take advantage of the pretrained VQ-VAE to perform style transfer. In particular, given source and target audio signals $x, y \in \mathbb{R}^{1\times T}$ where T is temporal length of the signal, our aim is to generate a translation $x \rightarrow y$ that preserves the 'content' of x and adopts the 'style' from y.

To that end, we first extract deep features of x and y by choosing a layer inside a particular JukeBox VQ-VAE and split the resultant representation into two parts: $F_{pre} : \mathbb{R}^{1\times T} \rightarrow \mathbb{R}^{C\times T'}$ and $F_{post} : \mathbb{R}^{C\times T'} \rightarrow \mathbb{R}^{1\times T}$ where C denotes the number of feature channels and T' is the temporal dimension of the deep features. F_{pre} then is used to extract features $h_x, h_y \in \mathbb{R}^{C\times T'}$ such that

$$h_s = F_{pre}(s) \tag{1}$$

where $s \in \{x, y\}$ is the input signal to JukeBox.

Extracted features h_s can be directly decoded to reconstruct the input audio signal $\hat{s} = F_{post}(F_{pre}(s)) \sim s$. In our work, we propose transforming the deep features h_x using the statistics of h_y in order to perform style transfer. We denote our transformation function as $F_{trans} : (\mathbb{R}^{C\times T'}, \mathbb{R}^{C\times T'}) \rightarrow \mathbb{R}^{C\times T'}$ and the transformed features as $h_{x\rightarrow y}$.

$$h_{x\rightarrow y} = F_{trans}(h_x, h_y) \tag{2}$$

Finally, we obtain the translation as

$$x \rightarrow y = F_{post}(h_{x\rightarrow y}) \tag{3}$$

In the next sections, we describe different splits we chose for F_{pre} and F_{post}, and transformation functions we use for F_{trans}.

B. Style Transfer using the VQ-VAE codebook

In this section we describe our simplest method that was used as a baseline model. The encoder first transforms the input audio into latent vectors h_s and then the vectors are quantized to the closest codebook vector e_s. The intuition behind this first method is to quantize the latent vectors from content signal x using only the unique vectors from style signal y.

First, we obtain the latent vectors from x and y (h_x and h_y). For y we quantize the vectors using the full codebook to obtain e_y and for x we quantize the vectors using only the unique vectors from e_y. We implement F_{trans} as $F_{codebook}$ in the following manner:

$$F_{codebook}(h_x, h_y) = Codebook_{e_y}(h_x) \tag{4}$$

Where $Codebook_{e_y}$ is the codebook restricted to the quantized vectors e_y obtained by quantizing h_y.

[4]In the supplementary we show that it is possible to perform image style transfer using image autoencoders. This is an encouraging fact since the autoencoder might not have extracted meaningful features for style transfer.

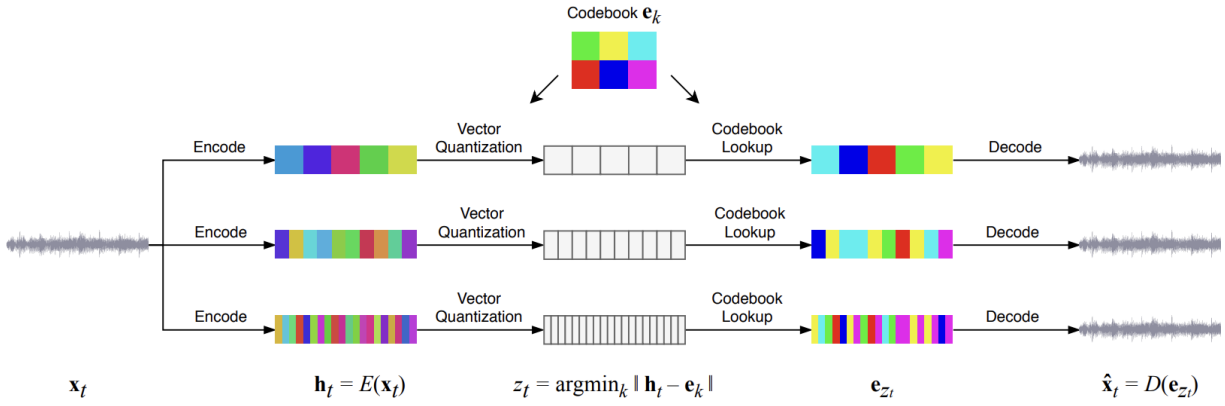


Fig. 1. Jukebox autoencoder architecture. Jukebox has an autoencoder module that is composed of three separate VQ-VAE models that share the same codebook. The raw audio signals are compressed by different factors: 8x, 32x and 128x with increasing level of abstraction. [image taken from [3]]

The translation using the $Codebook_{e_y}$ is defined as:

$$x \xrightarrow{codebook} y = F_{post}(F_{codebook}(F_{pre}(x), F_{pre}(y))) \quad (5)$$
$$= F_{post}(F_{codebook}(h_x, h_y)) \quad (6)$$

C. Style Transfer by Input Optimization

In the nominal work of [4], the style transfer is achieved by optimizing a loss function composed of content of style terms. The content term tries to minimize the $L2$ distance between the embeddings of the content and target images at a certain layer in the pre-trained CNN, whereas the style loss penalizes the discrepancy between the statistical properties of embeddings of the style and target images at different layers expressed as Gram matrix. As aforementioned earlier, the optimization alters only the target image, which is often initialized as random noise.

In this project, as another attempt for audio style transfer, we follow Eric *et al.* [5] and formulate the problem as modification of the content signal x based on the style loss $\mathcal{L}_{style}$ that first constructs the Gram matrix of both signals x, y based on their representations h_x, h_y. More formally, the Gram matrix of $h_x \in \mathbb{R}^{C \times T'}$ is constructed for all channels in h_x by the Equation (7), where i indexes the time in $h_x(i) \in \mathbb{R}^{C' \times 1}$.

$$G(h_x(i)) = h_x(i)h_x(i)^T \in \mathbb{R}^{C' \times C'} \quad (7)$$

The style loss for the autoencoder layer (l) is then defined as in Equation (8):

$$\mathcal{L}^{(l)}_{style} = \frac{1}{C'^{(l)}T'^{(l)}} \sum_{i=1}^{T'^{(l)}} \left(G^{(l)}(h_x(i)) - G^{(l)}(h_y(i)) \right)^2 \quad (8)$$

Following [4], we use L-BFGS algorithm to minimize the style loss in Equation (8).

In our experiments, we try modifying the input based on representations from all levels of encoders, which can be denoted by $h_x^{(0)}, h_x^{(1)}, h_x^{(2)}$, where $h_x^{(0)}$ refers to the features extracted by the lowest-compression (8x) autoencoder while $h_x^{(2)}$ denotes the representation obtained by the highest-compression (128x) counterpart. We also try using a single autoencoder. As this method aims modifying the input itself, we can either consider the modified input as the resultant target signal, or, we can decode the representation of the modified input as our target signal.

D. Style Transfer using AdaIN

In this section, we describe how we use AdaIN from [7] to implement F_{trans} as F_{AdaIN}.

$$F_{AdaIN}(h_x, h_y) = \sigma_{h_y} \frac{h_x - \mu_{h_x}}{\sigma_{h_x}} + \mu_y \quad (9)$$

where $\mu_{h_s}, \sigma_{h_s} \in \mathbb{R}^{C \times 1}$ temporal average and standard deviation of h_s respectively.

The translation using AdaIN is defined as

$$x \xrightarrow{AdaIN} y = F_{post}(F_{AdaIN}(F_{pre}(x), F_{pre}(y))) \quad (10)$$
$$= F_{post}(F_{AdaIN}(h_x, h_y)) \quad (11)$$

AdaIN preserves the temporal information of the source, 'content', signal but changes the relative importance and mean of the channels according to the target, 'style', statistics. Compared to Gram matrix based approach in [4], AdaIN is computationally more efficient as it does not require per image optimization. On the other hand, AdaIN only acts on the diagonal of the Gram matrix, hence, it might have less control over the output image. We refer the readers to [7] for a more detailed analysis on AdaIN.

We experimentally see that the JukeBox encoder with highest downsampling rate has more control over the output signal. Hence, we choose the autoencoder with 128x downsampling rate. We, then, choose F_{pre} as the encoder and quantization blocks of JukeBox, and F_{post} as the decoder. After observing that modifying a single feature layer is not effective, we apply AdaIN in multiple layers of the decoder. More specifically, we split Jukebox into N different blocks instead of 2 and name them as F_i where $i \in 1, ..., N$ and apply AdaIN to the outputs of $F_1, F_2, ..., F_{N-1}$, iteratively. The output of F_N remains unchanged because it is the output wave signal. We

discuss a perceptual normalization for the output wave signals in our experiments section .

E. Style Transfer using WCT

We then use WCT to implement F_{trans} as F_{WCT}. We first extract the normalized features as $\hat{h}_s = h_s - \mu_{h_s}$. Then we apply eigen-decomposition to the normalized features

$$\frac{1}{T'}\hat{h}_s\hat{h}_s^T = E_s A_s E_s^T \tag{12}$$

We perform the whitening on the source features h_x to obtain an uncorrelated feature map.

$$h_x^{white} = E_x A_x^{-1/2} E_x^T \hat{h}_x \tag{13}$$

We then colorize h_x^{white} with $\hat{h}_y$

$$h_x^{colored_y} = E_y A_y^{1/2} E_y^T h_x^{white} \tag{14}$$

Finally, we add the mean of h_y to the colored features

$$h_{x\to y} = h_x^{colored_y} + \mu_y \tag{15}$$

The transformation function for WCT then becomes

$$F_{WCT}(h_x, h_y) = E_y A_y^{1/2} E_y^T (E_x A_x^{-1/2} E_x^T (h_x - \mu_{h_x})) + \mu_{h_y} \tag{16}$$

Notice that similar to AdaIN, we normalized the features along the temporal dimension. However, WCT applies whitening and coloring transforms instead of the standardization used in AdaIN. In that sense, WCT is a stronger transformation compared to AdaIN. Applying stronger transforms is advantageous as it brings the feature statistics of the translation and target signals closer to each other. On the other hand, stronger test time transformations create discrepancy between testing and training, which might result in less realistic outputs. For further analysis on WCT, we refer the readers to [9]

We repeat the choice of F_{pre} and F_{post} in II-D. We also experiment with applying WCT in multiple layers as explained in II-D

III. Experiments

A. Dataset

In order to evaluate the methods presented in this paper, we select 3 different audio samples interpreted by 2 to 6 different instruments. Specifically, we experiment on 2 *Moonlight* samples (piano, guitar), 6 samples of the note *C4* (flute, piano, sawtooth, sine, trumpet, violin), and 2 non-matching samples of music interpreted by a banjo and an organ.

B. Experiment 1

In the first experiment, the *source* and *style* audio samples both interpret the same note (C4), but are played by different instruments (flute, piano, sawtooth, sine wave, trumpet and violin). Each of those 6 instruments is in turn source and style which results in 30 tests for each of our methods.

C. Experiment 2

We raise the difficulty of Experiment 1's set up by using more complex source and target music pieces using the *Moonlight* dataset, played by a guitar or a piano. This experiment results in 2 tests for each method: from guitar to piano, and from piano to guitar.

D. Experiment 3

The last experiment consists in giving a source and target audio samples which do not interpret the same music piece, and which are played by different instruments. Specifically, we use a banjo piece and an organ piece to each be target and style file which results in 2 samples per method.

E. Quantitative Evaluation of the experiments

Objectively quantifying whether a generated sample is closer in timbre to its *source* or *style* sample poses several challenges. Indeed, the three audio pieces may not be aligned, may not be played at the same pace or may not be interpreted in the same way which means that classic losses such as *mse*, *l2 norm* may not work to assess the timbre-distance between the files. Additionally, we do not have any ground truth about what the translated audio files should sound like, as similar instruments can sound very different based on its maker, its material and a plethora of other criteria. This is why a relative metric rather than an absolute one is more suitable in our particular case. Consequently, to palliate some of those issues, we design the *l2-mfcc* loss which consists in taking the l2 distance between the averaged Mel-Frequency Cepstrum Coefficients (mfcc) of the reference sample (source or style) and the generated audio. This choice is motivated by previous work which links the values of those coefficients to the encoding of voice or instrument timbers in music processing. Before applying the *l2-mfcc*, we normalize all synthetic samples to zero mean, and apply the EBU R128 loudness normalization via the `ffmpeg-normalize`[5] package. This normalization technique is a european norm ensuring that the volume of all signals are adjusted to the same *perceived* volume [10]. We then extract the mfcc of the normalized segments, which we average over time before computing the l2 norm between the two sets of averaged coefficients. For each generated signal, we compute a pair of *l2-mfcc loss*: once between the generated signal and the source signal, and once between the generated signal and the target signal. We then analyse whether the timbre transferred files are closer to their source sample or style sample.

IV. Results

A. Experiment 1

When the source file is an audio sample of the note *C4*, all techniques generate samples close to the *source* file than the *style* file no matter the instrument. The only exception is the *codebook* method when transferring piano to violin as observed on figure 2. When listening to the actual audios, it

[5]https://github.com/slhck/ffmpeg-normalize

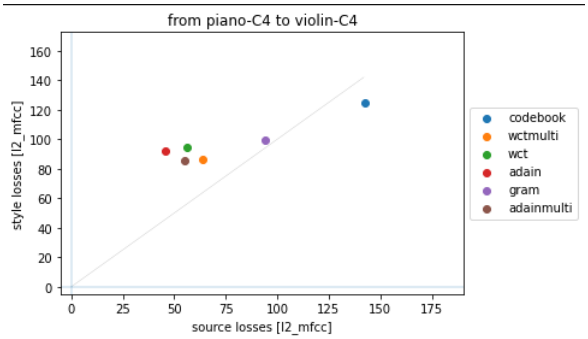


Fig. 2. Piano to violin translation evaluated by our metrics. Ideal translations are expected to have low style loss and higher source loss

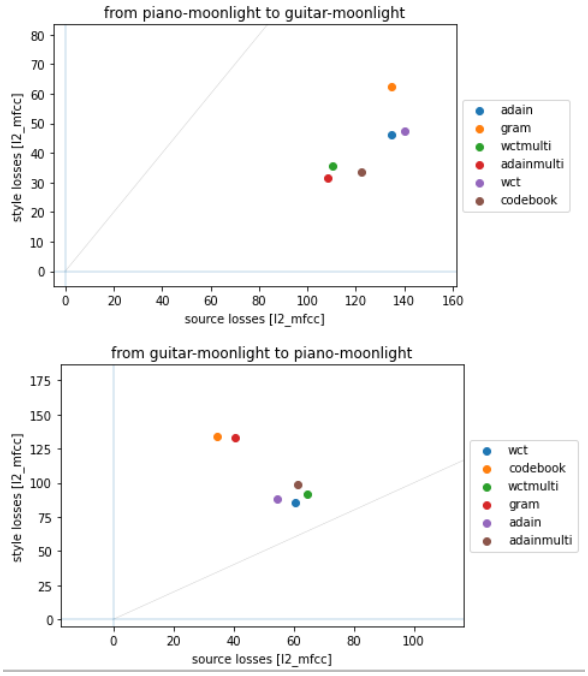


Fig. 3. Piano to guitar and guitar to piano translations evaluated by our metric.

transpires that all techniques closer to the source still modulate the attack which is very characteristic of pianos. The codebook sample, however, has a very pronounced rubbing sound which is closer to the violin than the piano. The sawtooth and trumpet instruments have the inverse tendency: all generated samples are closer to the *style* file no matter the instrument.

Another interesting finding is that when violin is used as a *source* file, all methods tend to generate samples which have very similar losses, no matter whether they are closer to the *style* file or the *source* file.

B. Experiment 2

All techniques generated samples whose *l2-mfcc* loss indicates that they are closer to the guitar sample, no matter whether they are the *source* or the *style* file (Figure 3). This may indicate that guitar strings are easier to modulate than piano ones. From a qualitative point of views, it sounds like the guitar-to-piano files are perceived as generated by a piano with a vibrating noise in the background. This could explain why, though the perceived sound is loyal to a piano, the quantitative metric still places it closer to a guitar.

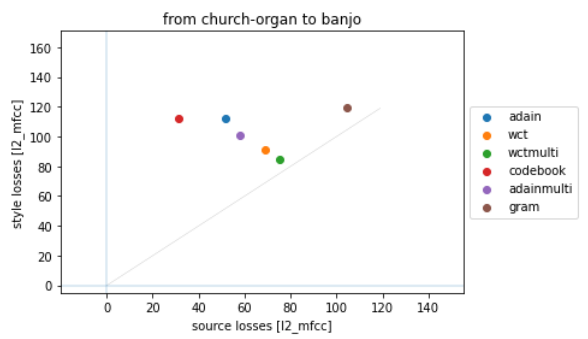


Fig. 4. Church organ to banjo translations evaluated by our metric

C. Experiment 3

All techniques except for the gram method performed similarly, generating samples closer to the source input than the style input as illustrated in Fig. 4. The outlier is perceived as a very noisy, rainy-like audio sample, which may explain why it is seen as neither close to a banjo, nor close to a church organ.

V. Discussion

In this project, we explored multiple methods for music style transfer. To test the methods we ran three different experiments. In the first experiment, we used *source* and *style* samples from the same note but different instruments. We found that in most of the cases the generated samples were closer to the source than to the style. In the second experiment, we used the same song (Moonlight Sonata) for piano and guitar and found that similar to the previous experiment, when using the guitar as the source and piano as the style, all techniques were closer to the guitar sample. In the last experiment using church-organ as the source and banjo as the style, we observed the same result: the generated samples were closer to the source than to the style input.

When listening to the actual audio, the results are unsatisfactory. In comparison to style transfer in images, timber transfer in music is a much more complex problem involving an additional time dimension. Adapting the style from one song to the content from another is not an easy task even for human beings and requires years of professional training. To convincingly perform music style transfer between two specific styles, a musician must first be familiar with the annotation, instrument usage, tonality, harmonization, and theories of the two styles. The task gets harder when we consider more styles. The models have to consider not only the pitch content of the given music but also the pitch range of each instrument as well as the relation between different instruments (e.g., some instrument combination creates more harmonic sounds) [8].

Future work could focus on teaching transfer to one style in particular by training the models on a larger corpus of only two styles. For example, developing an architecture that successfully learns the style of piano and applies it to any guitar audio can be the starting point for a more generic model.

REFERENCES

[1] Adrien Bitton, Philippe Esling, and Tatsuya Harada. Vector-quantized timbre representation. *arXiv preprint arXiv:2007.06349*, 2020.

[2] Ondřej Cífka, Alexey Ozerov, Umut Şimşekli, and Gael Richard. Self-supervised vq-vae for one-shot music style transfer. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 96–100. IEEE, 2021.

[3] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

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

[5] Eric Grinstein, Ngoc QK Duong, Alexey Ozerov, and Patrick Pérez. Audio style transfer. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 586–590. IEEE, 2018.

[6] Sicong Huang, Qiyang Li, Cem Anil, Xuchan Bao, Sageev Oore, and Roger B Grosse. Timbretron: A wavenet (cyclegan (cqt (audio))) pipeline for musical timbre transfer. *arXiv preprint arXiv:1811.09620*, 2018.

[7] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization, 2017.

[8] Yun-Ning Hung, I-Tung Chiang, Yi-An Chen, and Yi-Hsuan Yang. Musical composition style transfer via disentangled timbre representations, 2019.

[9] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. *CoRR*, abs/1705.08086, 2017.

[10] EBU Recommendations. Loudness normalisation and permitted maximum level of audio signals. EBU, 2020.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[12] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Supplementary Material

I. Visualizing Channels in Jukebox CNN Encoder

Trained CNNs extract meaningful features from their input. Olah et al. [2] show that the earlier layers of CNNs extract low-level features such as edges, orientations, and textures while deeper layers extract more meaningful features such as complex patterns, object parts, and objects. In order to find the pattern that each neuron in a trained CNN is sensitive to, they optimize the pixels of the input image such that it highly activates that neuron. This allows us to find the types of patterns that the neuron is sensitive to.

We tried the method proposed in [2] on the neurons in Jukebox autoencoder network and optimized the input audio wave in order to maximize the activation of an arbitrary neuron in the network. We were hoping that this will result in some forms of audio textures. However, the results were extremely noisy and unpleasant to listen to[1]. Hence, we adopted a different approach to understand the role of neurons in Jukebox autoencoder network. We hypothesize that channels in Jukebox autoencoders act as filters with specialized frequency responses. In order to check the validity of this hypothesis, we use a frequency sweep audio as the input to the Jukebox autoencoder and observe the activation of a channel over time. Since for the frequency sweep we know the relationship between time and frequency, the activation of a channel over time will reveal the frequency response of that channel. We developed an application using python to help us visualize the frequency response of all channels in different layers of Jukebox encoder. Figure 2 shows the user interface of our Streamlit application. Our app allows the users to choose the desired configuration to visualize:

- Choosing the Jukebox encoder to visualize (3 different encoders with 8x, 32x, 128x downsampling rates.)
- Choosing type of frequency sweep from either linear, chromatic scale, or major scale.
- Choosing the time length of frequency sweep, the starting frequency, and the ending frequency.
- Choosing the corresponding layer and channel index to visualize.

After choosing the desired configuration from the available options we provide the following outputs:

1) The activation of channel for different frequencies.
2) The Mel Spectogram of the input and output signal.
3) The audio signal of the channel activation. This allows us to listen to the activation of the channel.

[1]We applied audio-equivalent of some of the techniques introduced in [2] to reduce the high frequency part of the audio but the results were still not good.

Figure 3 shows the activation of two channels in two different layers of the 128x Jukebox autoencoder. We use a chromatic scale frequency sweep starting from frequency of 110Hz and going up 7 octaves over 10 seconds. The Mel Spectogram of the input audio can be seen in Figure 3a. Figure 3b, 3c show the activation and Spectogram of channel 2 in layer 0 and channel 2 in layer 47 of the encoder network. We can see that channel 2 in layer 0 acts as a high pass filter while channel 2 in layer 47 acts as a Band-pass filter. Note that because of the downsampling in the encoder, the deeper layers have lower sampling rate than the early layers. This is shown by the red line in Figure 3b which represents the Nyquist frequency in the layer which the chosen channel resides. In Figure 3b-right we can see that the band-select frequency is above the Nyquist rate for this layer. This shows that the encoder network is aware of the aliasing so the decoder can reconstruct the original signal after downsampling by a factor of 128. We provide an audio file which we extracted from the activations of a channel in jukebox encoder given a chromatic scale input signal. We can hear the chromatic scale in the activation of this channel up to some frequency and after that aliasing starts to happen[2].

II. Exploring Jukebox Activations for Different Instruments

To explore Jukebox, we explored the activations of the encoder. In particular, we used as input the same note played in four different instruments (piano, violin, trumpet, and flute) and compared the encoder activations after encoding each separately. An interesting finding was that there were some channels in the layers of the encoder that were only activated for some instruments. In other words, some channels were not being used by the note when changing instruments. This leads us to believe that the model is capable of identifying timber and encoding it. Figure 1 shows the activated and deactivated layers-channels for the note C4.

We visualize the layers that are activated for some instruments but not for others for two notes: C4 and G4.

III. Image Style Transfer Using Autoencoders

In this section we show that we can use self-supervised image autoencoders to perform style transfer. We use VQGAN Autoencoder [1] which is similar to Jukebox autoencoder, as they both use vector-quantized (VQ) latent space. We choose to utilize two different methods for style transfer: AdaIN and WCT. We show the effect of each method when applied to different layers of the encoder network. Figure 4, 5 shows the

[2]The file is in google drive under report_data/sound_of_channels/sample.wav

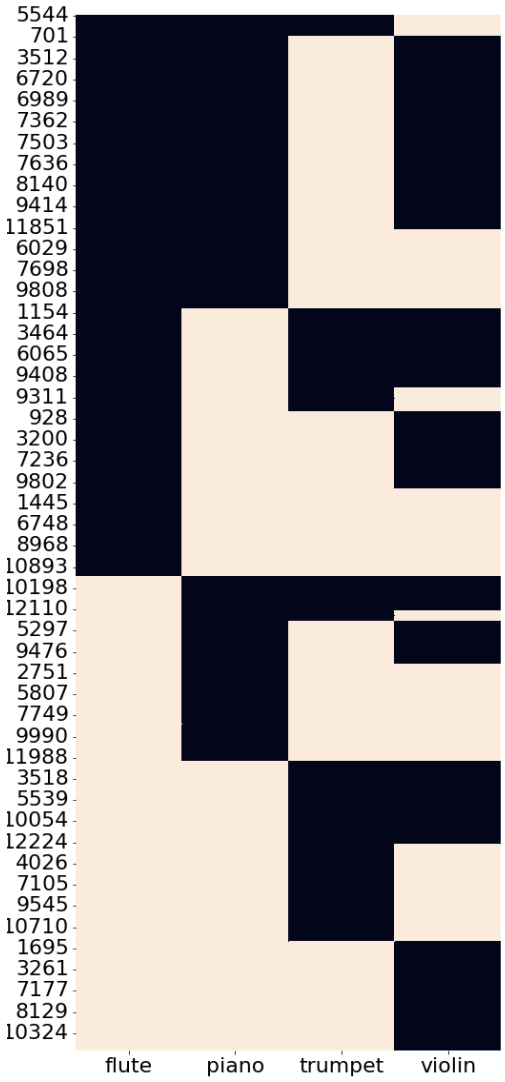


Fig. 1: Layers-channels activates for the same note with different instruments

results on two different content and style images. As can be seen, using deeper layers of the encoder yields better style transfer results. Also the quality of the style transfer is better using WCT method. This results are encouraging because they show that self-supervised autoencoders are capable of extracting meaningful features for image style transfer.

REFERENCES

[1] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis, 2020.

[2] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

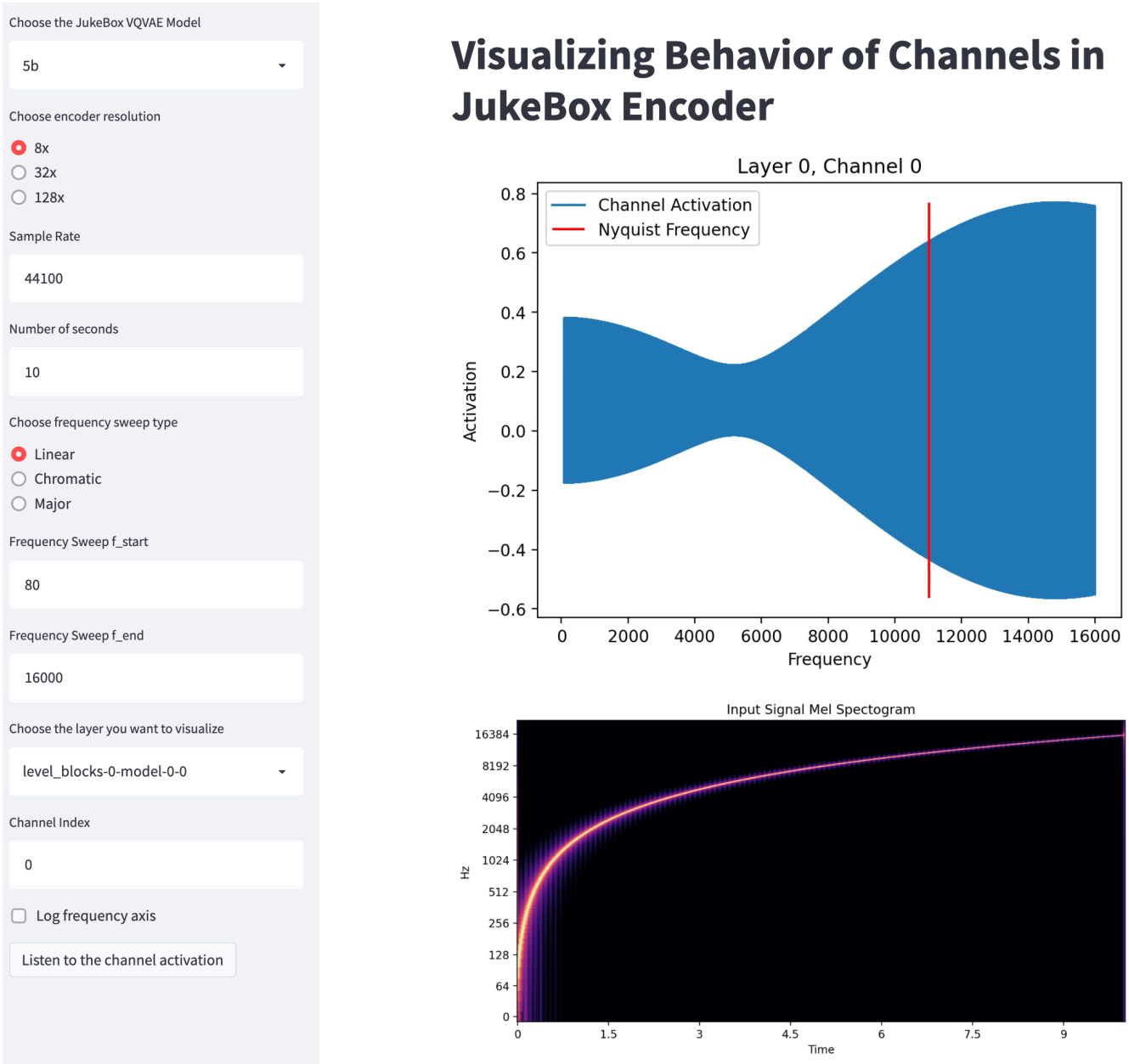
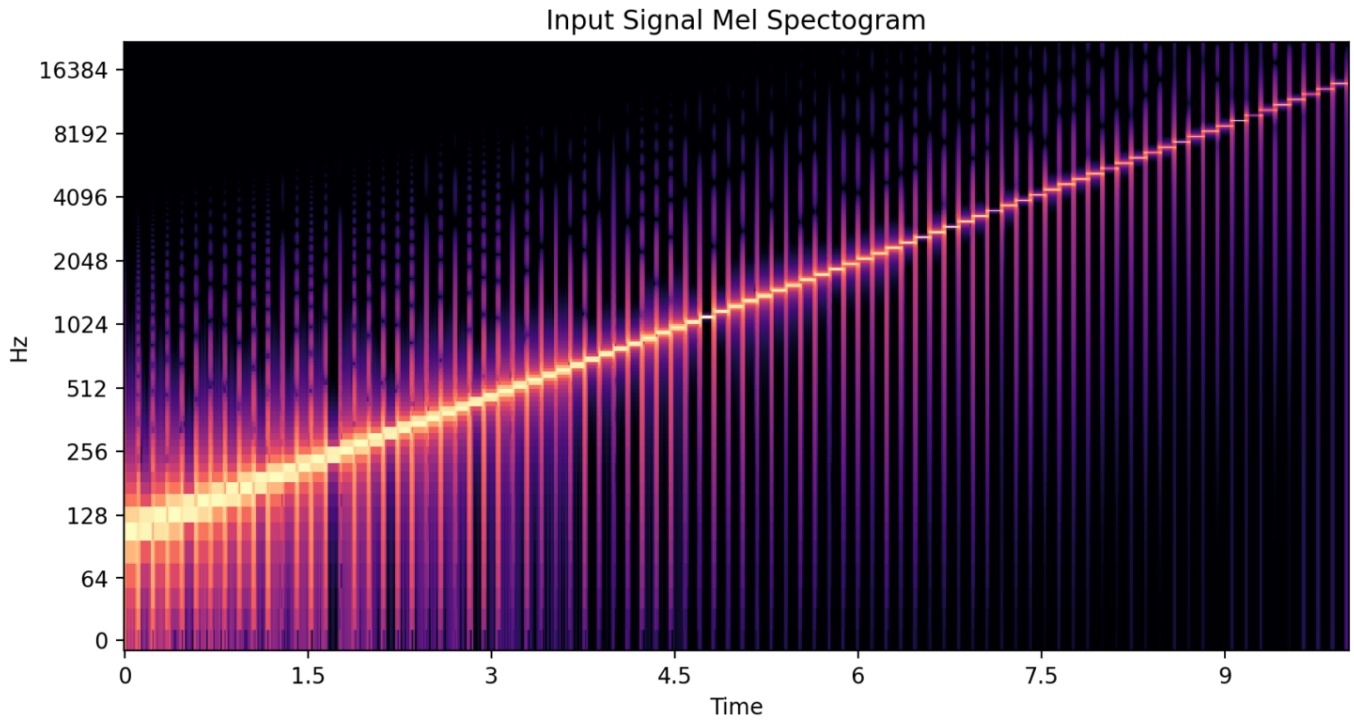
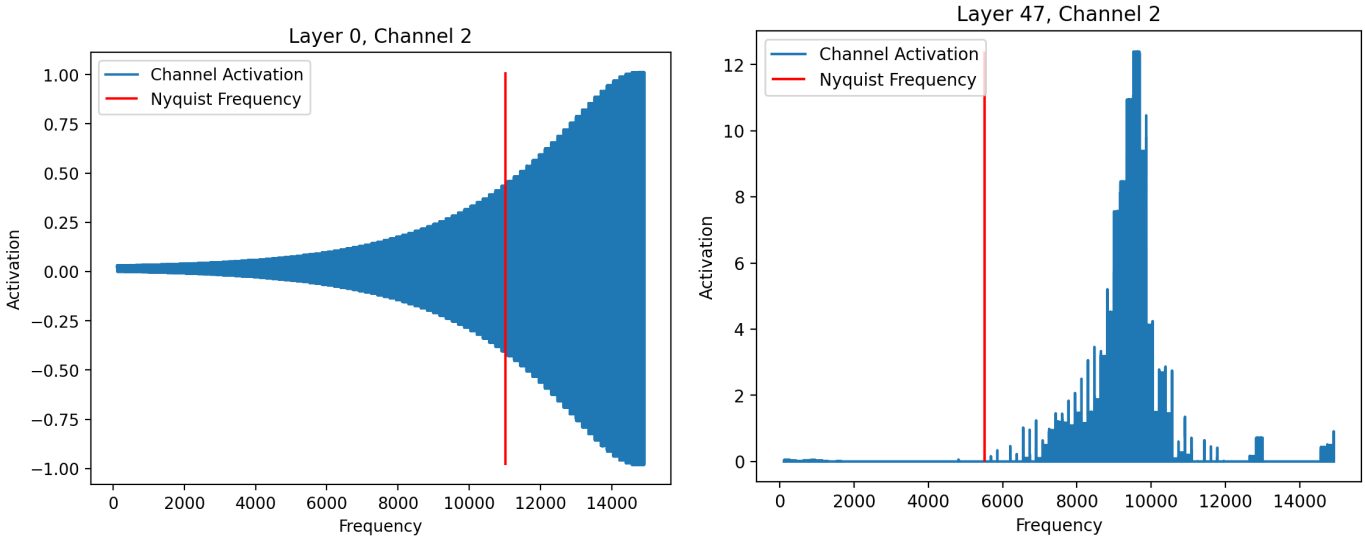


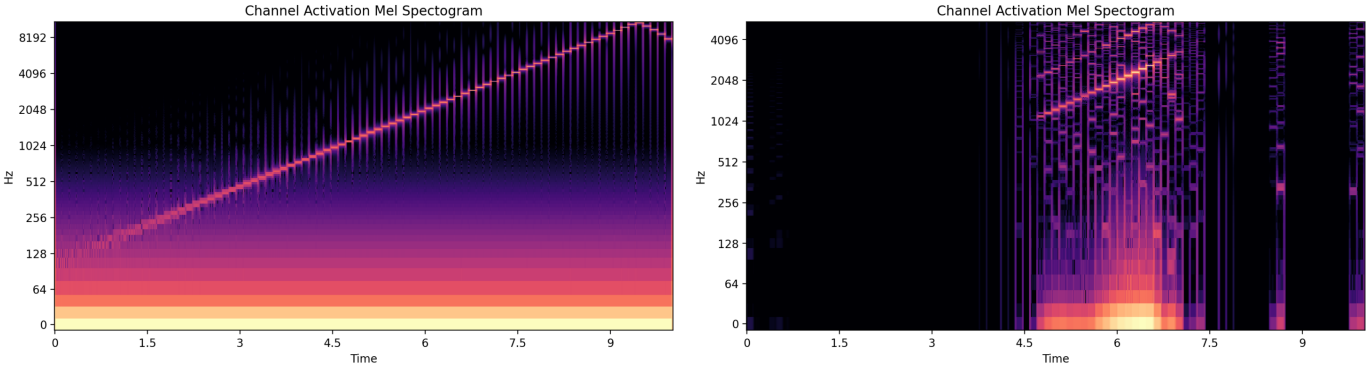
Fig. 2: User interface of our Streamlit application for visualizing the channels of Jukebox encoder networks.



(a) The Mel Spectogram of input signal to the Jukebox encoder.



(b) The activation of two channels in two different layers over through the frequency.



(c) The Mel Spectogram of the activation of two channels in two different layers through the frequency sweep.

Fig. 3: Visualization of two different channels in two different layers in Jukebox encoder.

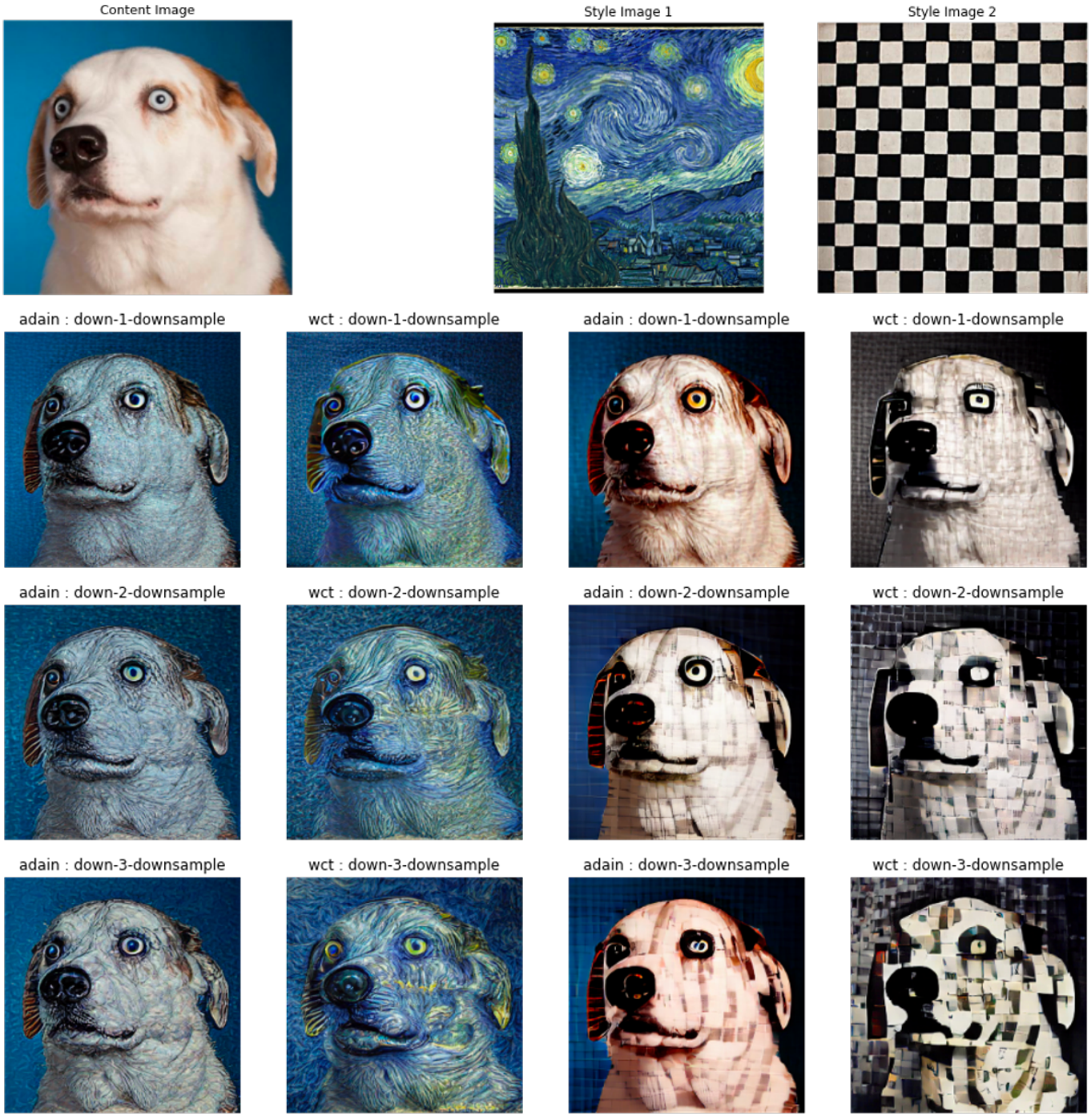


Fig. 4: Result of style transfer using VQGAN Autoencoder, and two different methods: AdaIN, and WCT. Note that deeper layers of the network yield better style transfer quality.

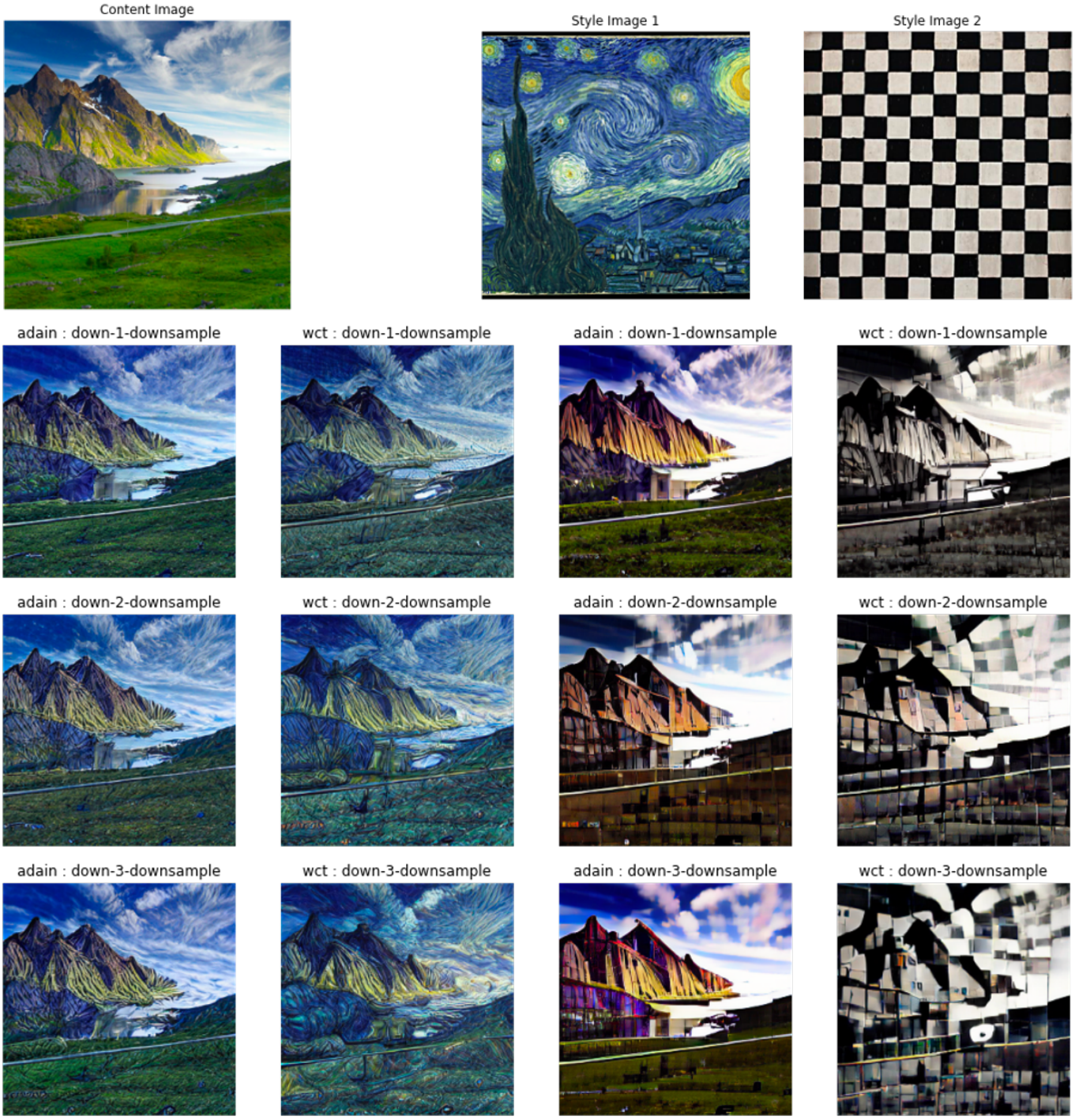


Fig. 5: Result of style transfer using VQGAN Autoencoder, and two different methods: AdaIN, and WCT. Note that deeper layers of the network yield better style transfer quality.