Neural Style Transfer for Audio

Aryan Kanak

Introduction

The goal of this project will be to achieve neural style transfer for audio files, specifically music. That is, given a content audio and a style audio as input, a new audio file will be generated that has the content of the content audio and the style of the style audio. What exactly "content" and "style" means is subjective, but content is typically thought of to be the melody of a song while style refers to the timbre.

Originally neural style transfer was created to generate images like below.

We will apply similar methods adapted for audio files.

Neural Style Transfer

The original neural style transfer algorithm is described in [1]. First, the generated image is initialized as noise (or in later work, as the content image). At each step, we put the content image, style image, and generated image through a pre-trained convolutional neural network (VGG-19 in this case) and calculate the content loss and style loss using the produced feature maps. The content loss is simply the squared-error loss between the feature maps of the content and generated images. For the style loss, we first compute the Gram matrix for the style and generated images. Each entry (i, j) in the Gram matrix for layer l is the inner product between the vectorized feature map i and j.

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l .$$

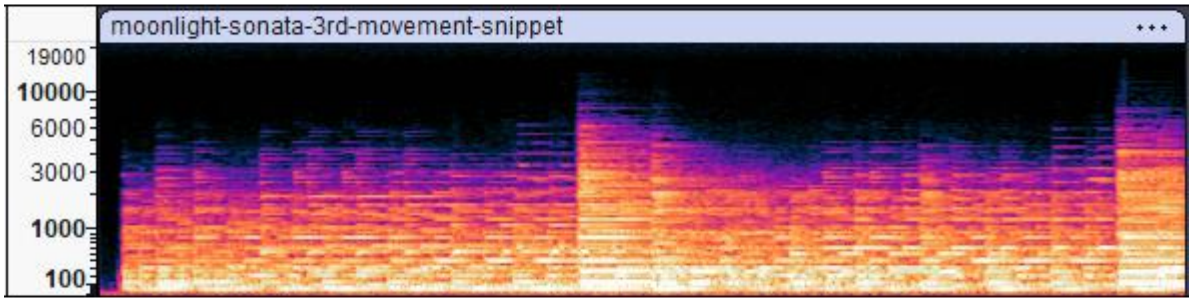
The style loss is then the sum of squared errors between the Gram matrices in each layer. The total loss is then weighted between the content loss and style loss where α and β are weighting factors. Then, we perform backpropagation with respect to the generated image to minimize this loss.

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

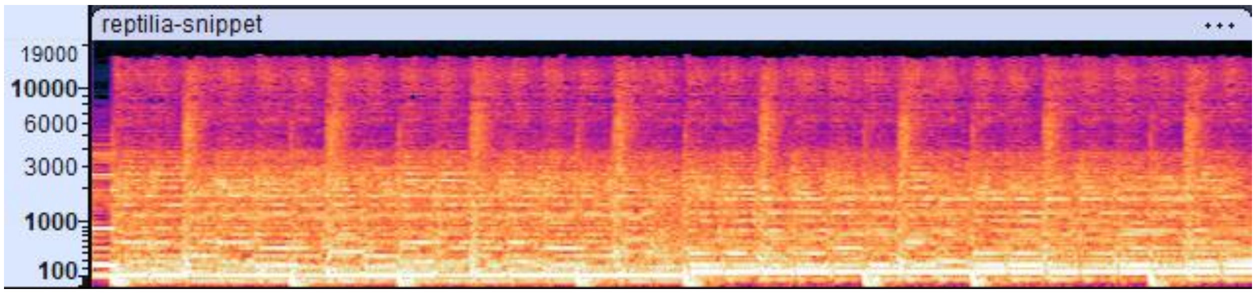
Experiments

Attempt 1: Typical Neural Style Transfer With VGG-19

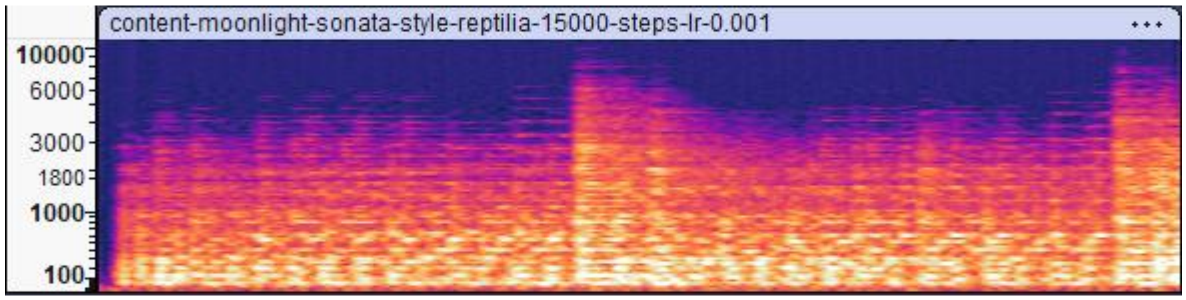
The first technique we'll use is the same one used in [1] but slightly modified for audio. This technique was developed in [2]. First, we extract the spectrograms of the content and style audio as the magnitude of the short-time Fourier transform. As well, we resize the style audio spectrogram to match that of the content audio. Then, each spectrogram is duplicated 3 times to get an object like an RGB image and the generated audio is initialized to the content. Then, we simply apply the original neural style transfer algorithm, putting the audio representations through VGG-19 and trying to minimize the loss function. Finally, we use the Griffin-Lim algorithm to recover the audio from the generated spectrogram. Here are the spectrograms for different hyperparameters.



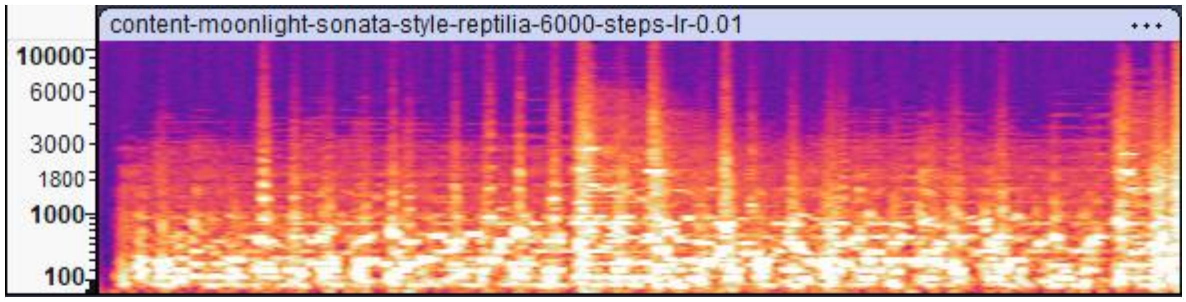
Moonlight Sonata by Ludwig Van Beethoven



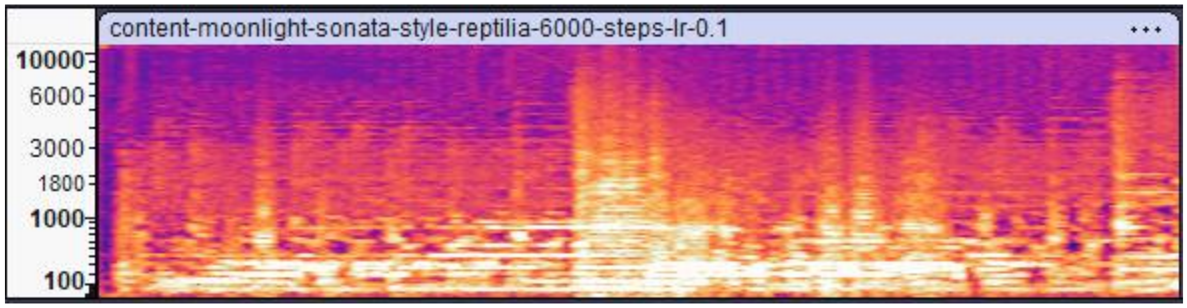
Reptilia by The Strokes



Moonlight Sonata in the style of Reptilia, 15000 steps, learning rate=0.001



Moonlight Sonata in the style of Reptilia, 6000 steps, learning rate=0.01

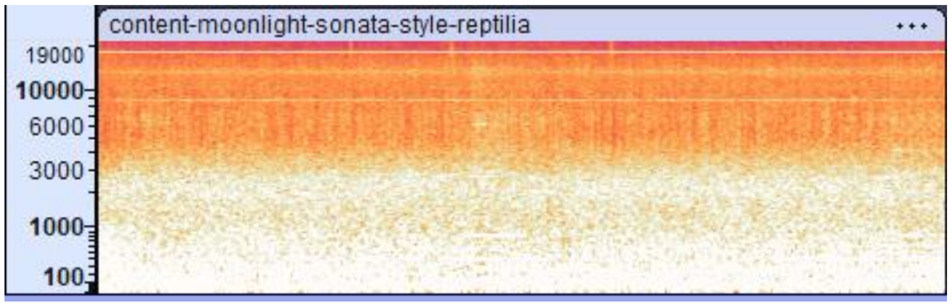


Moonlight Sonata in the style of Reptilia, 6000 steps, learning rate=0.1

The results aren't very pleasing but this is expected and is consistent with the findings in [2]. The generated audio starts to get noisy and doesn't really sound too nice. Note that the program could have been run for more steps to minimize the loss even further but it is not worth doing so as we probably won't get good results either way. This is probably because VGG-19 is trained on images and likely isn't good at extracting the style of audio. So, it makes sense to try a model that is trained on audio instead.

Attempt 2: Using a Model for Music Genre Classification

The model that we'll be trying is a model for music genre classification by Dmytro Iakubovskyi [3]. It just takes a loaded audio file as input so we don't have to do any preprocessing before sending the audio to the model. It's based on the Wav2Vec2 model which uses a CNN to extract features of the audio and then does a bunch of other stuff to classify it. We only care about the feature extractor so we'll ignore everything else and use the feature maps generated by it to calculate the loss.



Moonlight Sonata in the style of Reptilia

The result is just the content audio with a ton of noise in the background. In this case, we used all the layers in the feature extractor to calculate the loss. But even using only certain layers didn't provide better results. Instead of trying random models and hoping for a good result, we should take a smarter approach.

What is Style?

To develop a better model for neural style transfer for audio, we need to look at how style transfer originally came about. Neural style transfer for images developed out of studies of texture synthesis. That is, generating images that have the same style as another but are notably distinct from the input. The authors of the original neural style transfer paper [1] adapted the method from a texture synthesis method they developed [4]. The texture synthesis method was essentially just style transfer but with 0 weighting on the content loss. By studying texture synthesis for audio, we can design a smarter method to conduct neural style transfer.

Attempt 3: Caracalla-Roebel Texture Synthesis

We will develop a new method for audio style transfer based on a texture synthesis algorithm developed by Hugo Caracalla and Axel Roebel [5]. This algorithm works with the log-spectrograms of each audio. Given spectrogram X computed with the short-time Fourier transform, the log-spectrogram is given as

$$S = \frac{\log(1 + C \times X)}{\log(1 + C)}$$

where C is a factor controlling compression. Instead of using a pre-trained network, we use a single-layer CNN with randomized weights. The single layer contains 32 filters of the sizes 3, 5, 7, 11, 15, 19, 23, and 27 each. The reason for the random weights is that previous work has shown that CNNs with randomized weights are quite good at extracting features of audio. The major difference in this method from traditional texture synthesis is that instead of computing Gram matrices, we compute three-dimensional tensors

$$H^k(i,j,x) = \sum_y F_i^k(x,y) \times F_j^k(x,y)$$

More detail can be found in the paper but the main reason for this is that summing over both frequency and time axes implies that parameterization is invariant in both directions. However, audio behaves differently along these two axes so we should only sum over the time axis. The loss is calculated the same way but with these tensors instead of Gram matrices. We adapt this for style transfer by using this new loss as the style loss and keeping the same content loss. Hyperparameters were as follows:

Compression factor – 1000

Frame size – 512

Hop size – 256

Total steps – 1000

Learning rate – 0.1

Alpha – 1

Beta – 0.01

Spectrograms of the generated results are shown in the appendix. All audio files tested were around 5 seconds in length and sampled at 44100 samples/second. This is the most promising method so far. Although it works better for some examples than others, you can start to hear elements of the style audio and the spectrograms start to reflect this. However, there are quite a few improvements that could be made to achieve even better results.

Future Work

In our method, we optimize the log-spectrogram of the generated audio and then at the end, we convert it to audio using the Griffin-Lim algorithm. However, in the work by

Caracalla and Roebel [5], the audio is optimized directly. The reason for this is that the Griffin-Lim algorithm can cause a loss in quality. So, since the short-time Fourier transform is differentiable, it is possible to bypass the lossy conversion and optimize the audio directly.

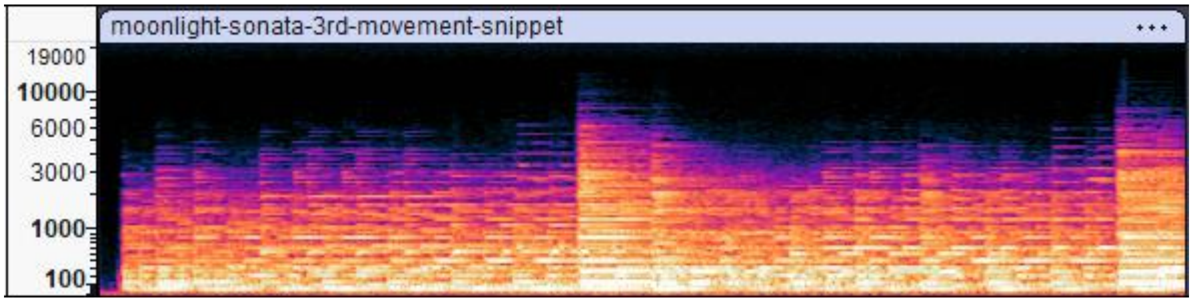
Another improvement that could be made is increasing the number of filters in the CNN. Our method used 32 filters while Caracalla and Roebel used 128. 128 filters were tried, but the computation took too long to complete. Caracalla and Roebel were able to perform the experiment in an hour on a GTX 1080 Ti while our code performing a less intensive computation took much longer on a GTX 4070. Thus, the problem is likely a lack of optimization in the code rather than a hardware limitation.

An improvement that was not explored much was experimenting with different hyperparameters. This alone has the potential to greatly improve the quality of the generated audio. Another potential improvement would be to try increasing the number of layers in the network.

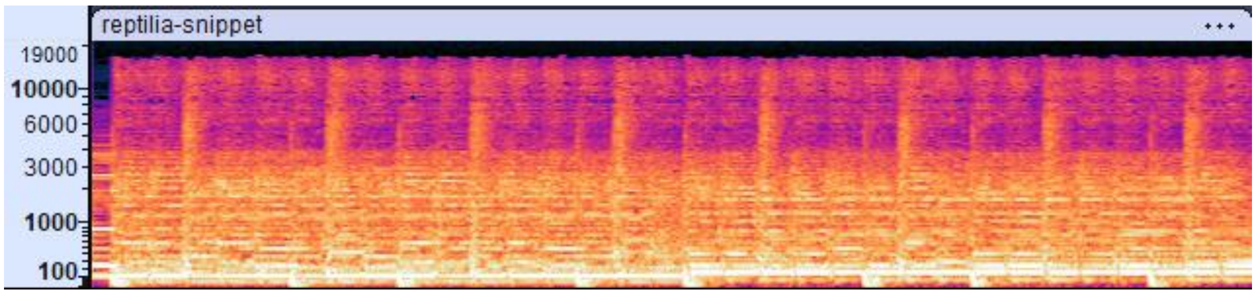
Overall, neural style transfer based on the Caracalla-Roebel texture synthesis method has shown very promising results compared to previous work done on the topic. If further explored, it could potentially achieve results much better than what has been produced before.

All code and generated audio can be found at https://github.com/Aryan-Kanak/audio-style-transfer.

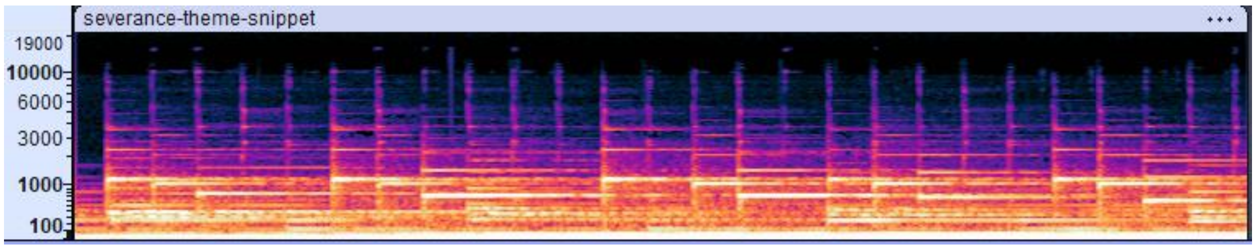
Appendix – Spectrograms of Input and Generated Audio Using Caracalla-Roebel Method



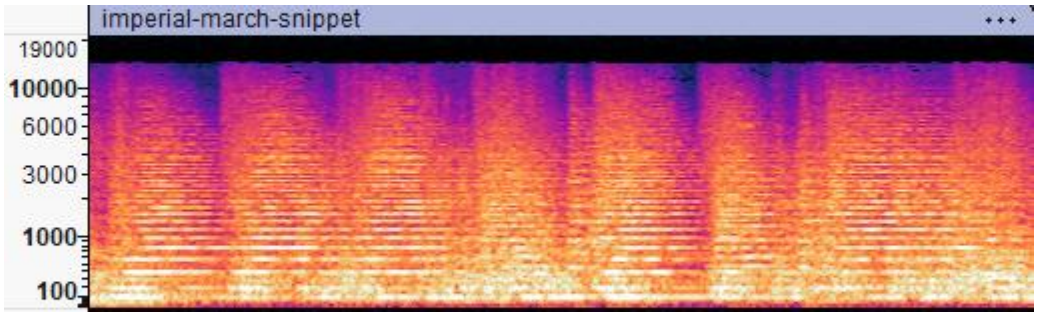
Moonlight Sonata by Ludwig Van Beethoven



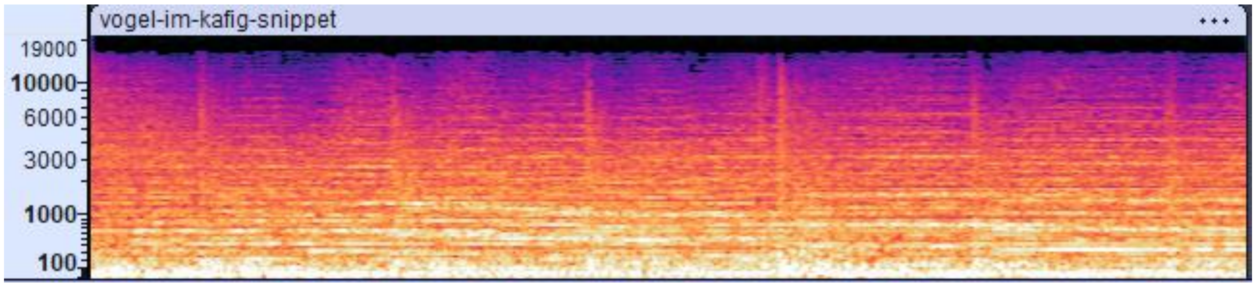
Reptilia by The Strokes



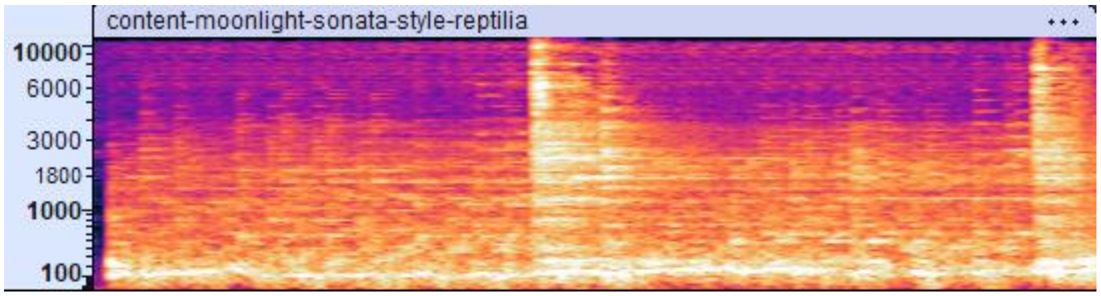
Severance Theme



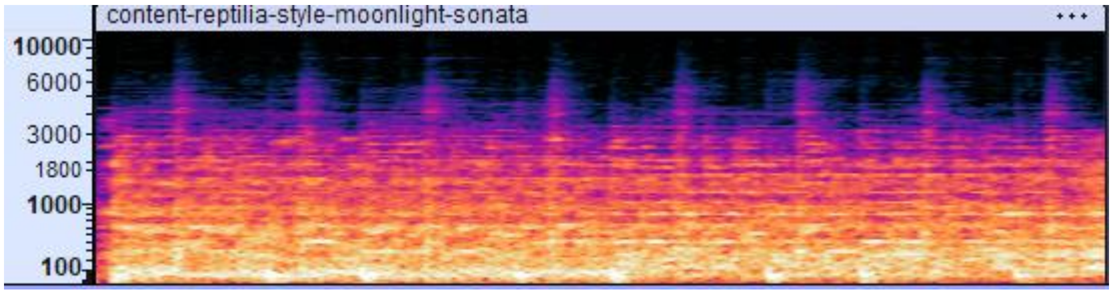
Imperial March



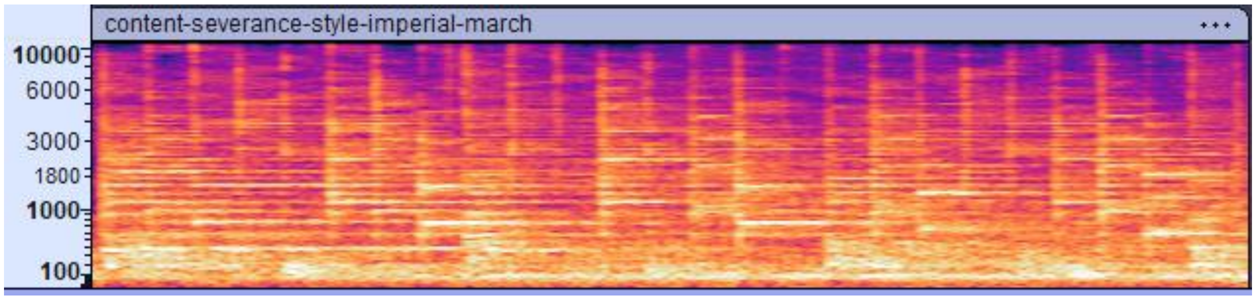
Vogel Im Kafig



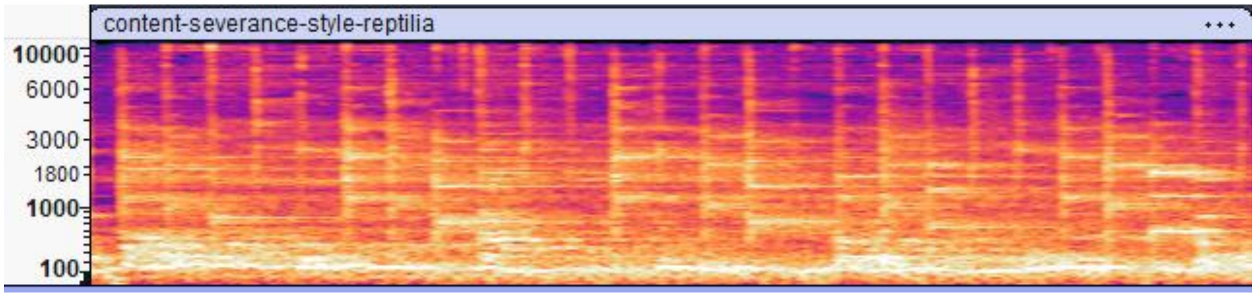
Moonlight Sonata in the style of Reptilia



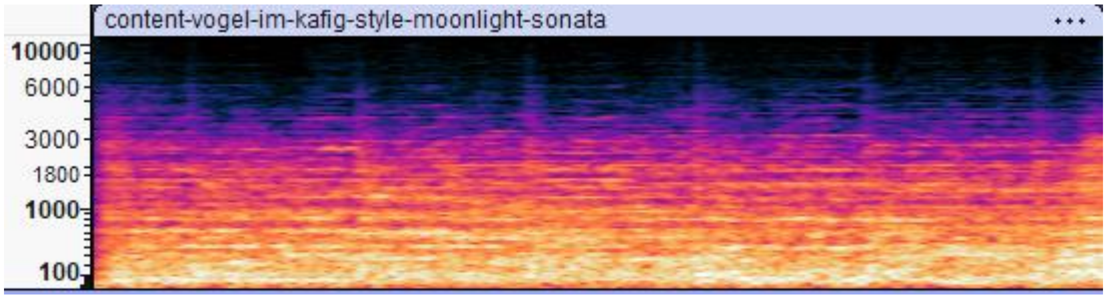
Reptilia in the style of Moonlight Sonata



Severance Theme in the style of Imperial March



Severance Theme in the style of Reptilia



Vogel Im Kafig in the style of Moonlight Sonata

References

1. Gatys, Leon A. "A neural algorithm of artistic style." *arXiv preprint arXiv:1508.06576* (2015).
2. Grinstein, Eric, et al. "Audio style transfer." *arXiv preprint arXiv:1710.11385* (2017).
3. https://huggingface.co/dima806/music_genres_classification
4. Gatys, Leon, Alexander S. Ecker, and Matthias Bethge. "Texture synthesis using convolutional neural networks." *Advances in neural information processing systems* 28 (2015).
5. Caracalla, Hugo, and Axel Roebel. "Sound texture synthesis using convolutional neural networks." *arXiv preprint arXiv:1905.03637* (2019).