Audio Style Transfer: a review on classical methods and evaluations

Jean-Baptiste Conan
jconan@sutdent.ethz.ch jeanbaptiste.conan.jbc@gmail.com
Swiss Data Science Center, ETH Zurich, Switzerland

***Abstract*—This report looks at audio style transfer, an emerging area within the broader field of style transfer. While style transfer traditionally involves transposing style from one image to another, audio style transfer extends this concept to audio clips, with the aim of imbuing one auditory source with the stylistic attributes of another. These stylistic attributes encompass various elements of sound texture, including tonal nuances, variations in timbre, rhythmic patterns, spatial effects and overall ambience.**

I. Introduction

Style Transfer (ST) is a popular field of research in computer vision and Deep Learning, which aims to apply the style of one image or audio clip, to another image or audio clip. The main goal is to create a new output that preserves the content of the original input while adopting the style of a reference image or audio clip. In this report, we focus on Audio Style Transfer (AST), which is a relatively less explored area of ST.

AST enhances audio clips with another audio source's stylistic attributes. An audio clip comprises voices, music, noises, etc., conveying emotions, stories, or sensory engagement. A stylistic attributes comprises elements of sonic texture such as tonal characteristics, timbre, rhythm patterns, spatial effects, and overall mood. This audio-style interaction uses new techniques to create harmonious auditory mixes.

Our paper's contribution is twofold, and summarized as follows:

1) ***Comprehensive review****:* : We provide a comprehensive review focused on the classical art of style transfer, that is with CNNs. We also provide an overview of the art for Autoencoders, like VAEs, and Generative models, such as GANs.

2) ***New method of evaluation and new benchmark****:* We also propose an new and innovative evaluation method to assess the performance of a style transfer algorithm. This new method was made possible by the creation of a new benchmark for audio style transfer algorithms. This paper indeed presents a new dataset that could be useful in style transfer tasks.

This project was conducted for the Swiss Data Science Center, a joint lab of the ETH Zürich and EPFL; as a semester project worth 8 ECTS, under the supervision of Giulio Romanelli (giulio.romanelli@epfl.ch) and Prof. Dr. Fernando Perez Cruz (fernando.perezcruz@sdsc.ethz.ch).

II. Preliminary works

Disclaimer: The research stopped in April 2023 and is certainly not up to date.

The field of ST, including Audio ST, is particularly rich, and extensive researches, ideas and knowledge have been conducted and brought throughout the years. In addition to a comprehensive review of the state of the art, we decided to concentrate on missing blocks, that would consolidate our understanding of the neural task of ST.

A. Gatys et al.

In 2015, Leon A. Gatys introduced a new innovative method for *Image Style Transfer Using Convolutional Neural Networks* ([1]). It consisted of a neural algorithm based on image representations derived from Convolutional Neural Networks. It is based on another of Gatys' research on audio texture ([2]). Their method allows for high resolution ST, as shown in the images below (Figure 1).

Figure 1. Example of Gatys' algorithm (source: github.com)

Their method is simple, so we will assume that the reader is aware of it, and we will therefore explain it briefly in this review. For more precision on the technical details, please refer to the original paper ([1]).

Take your two input images, a content and a style one, and pass both through an trained image classifier, like VGG-19 [3] as used in the original paper. This will give you some feature maps of the images, results of the data transformation through the different CNN layers. Intuitively, at each optimization step, we are then going to watch and compare the optimizing image's feature maps with the others' feature maps. So this is the original paper's general idea: all information relating to content and style is contained in these maps, and one can define loss functions in

[0]Code and Information on the dataset available at: https://github.com/jbccc/audio_style_transfer

order to optimize the desired output. Different feature maps can be visualized in the schema below (Figure 2).

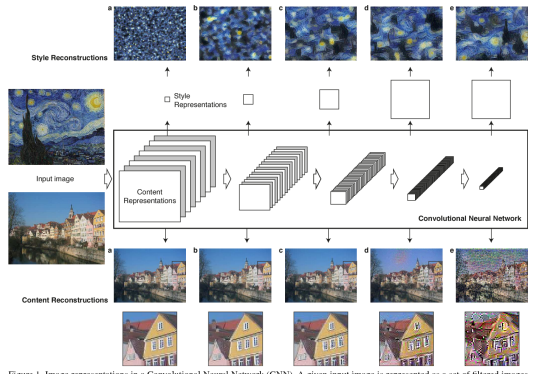

Figure 2. Image feature map (source: Gatys' original paper)

Let's look at the aforementioned loss function. It naturally decomposes into two sub-losses: on the one hand, we want to optimize with respect to the content, and on the other hand with respect to the style.

Content-wise, it is easy. We decide what kind of feature map(s) is(are) the most relevant and we do a pixel-by-pixel loss, generally a mean squared error (MSE - squared L2 norm) loss between the content image and the optimizing image. For the layer l of the CNN, if $\vec{p}$ is the content vector, $\vec{x}$ the optimizing image vector, we can define:

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

and the final content loss function is taken by defining a subset L of chosen layers. No further information were given in the original paper, giving the liberty of choice to the readers.

Style-wise, it is a whole other story. Indeed, the pixel-by-pixel approach is too simple to encompass and grasp a complex concept such as style. Therefore, they use a concept they introduced in their previous research paper ([2]): the Gram matrix. This tool is said to "to capture texture information" in the image, and is basically the correlation matrix. In our case, it is the correlation matrix of the different feature maps of an image. For the layer l of the CNN, if $\vec{a}$ is the gram matrix of the style image, $\vec{x}$ the gram matrix of the optimizing image vector, we can define:

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

and now, contrarily to the content part, we need to have information for each layer, so to take a weighted sum of each layer loss to account for the total style loss:

$$L_{style}((\vec{a}, \vec{x}) = \sum_l w_l \mathcal{L}_{style}(\vec{a}, \vec{x}, l)$$

In the equations above, N_l is the number of feature maps, each of size M_l, i.e. the height times the width of the feature map. We have now our objective function to optimize in order to perform ST: $\mathcal{L} = \alpha L_{content} + \beta L_{style}$, with α, β being weighting factors, which ratio α/β is between 10^{-1} and 10^{-4}, the lower meaning the more emphasize on the style, and vice versa.

Now that we have introduced Gatys' method for images, we can switch to audio.

B. Adaptation to Audio and Ulyanov's work

Raw signals, as pictured in Figure 3, are the way audio appears in the nature. Contrarily to picture and images, audio is transparent, i.e. two different instruments playing simultaneously will appear as one and unique signal, whereas each pixel of a picture can be assigned to one and only one object. This is challenging to do ST, in the sense that the necessary information is mixed and need to be separated or retrieved in some way.

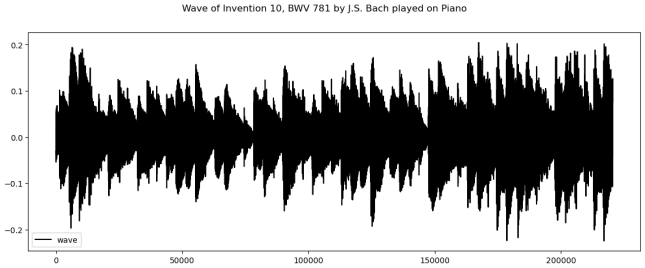

Figure 3. Example of a soundwave (Amplitude per ms)

Fourier, a French Mathematician, discovered that any sound waves can be decomposed into single atomic signals, each representing a frequency: it is called the Fourier Transform.

Now, let's delve into some related concepts. The Fourier Transform (FT) is a fundamental mathematical tool that allows us to analyze a signal in the frequency domain, breaking it down into its constituent frequencies. The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Fourier Transform, especially for discrete data points. It's a crucial technique used in various fields, from signal processing to image analysis.

Moving forward, we encounter the Short-Time Fourier Transform (STFT), which is a modification of the FT that enables us to analyze how the frequency components of a signal change over time. This is particularly useful for signals that vary in frequency content over different intervals.

It's here that we introduce the concept of a spectrogram, a visual representation derived from the FT. A spectrogram presents a signal's frequency content over time, providing insights into its changing characteristics.

visual representation crafted through the application of the Fourier Transform. A spectrogram displays a signal's

dynamic frequency content across time, capturing the interplay of its constituent frequencies. This technique unveils a comprehensive view of how a signal's characteristics evolve, providing a two dimensional analysis and representation of the signal. The Figure 4 shows an example of the transformed signal from Figure 3.

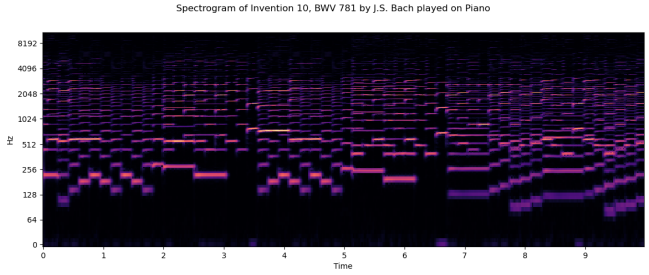

Figure 4. Example of a spectrogram (Hz per s)

With this foundation in place, let's explore how we can leverage this new 2D representation, offered by techniques like the STFT, to perform ST.

In a blog post ([4]), Dmitry Ulyanov and Vadim Lebedev talked about the limits of Gatys' method for audio. Indeed, first of all, classical image have a 3-channel representation, one for each primary color (RGB), whereas for spectrograms, it is a one-channel representation. Second, the neural net used with Gatys' method (VGG-19) is suited for real world objects' representation, and not for spectrogram, yielding bizarre feature maps.

They, therefore, proposed a new method to replace images by audio in Gatys' original paper. Their base idea is, because we can treat signals as $1 \times T$ (T being the max time in ms) matrices with N_{FFT} (the number of Fast Fourier Transform) channels (i.e. each frequency is considered as its own channel), to use this format as input. In the same way, they used a specific

Eventually, it is not enough to generate a spectrogram, we need to convert it back to waveform. This is done thanks to the Griffin-Lim algorithm ([5]). And there we go, having adapted the original paper's idea to audio and spectrogram, their algorithm produced relatively good audio[1].

C. Comparing Time Domain-based and Frequency-Time Domain-based Approaches

When it comes to diving into the world of audio style transfer, there are two main avenues of exploration: the time domain-based approach and the frequency-time domain-based approach. These two methods offer unique perspectives on how we can manipulate and mold sound to achieve stylistic transformations.

Imagine looking at sound in two different ways: one, as a wave evolving through time with different frequency components attached to it, and the other, as a collection of different frequencies that come together to create complex temporal patterns. These viewpoints are encapsulated by the concepts of time domain and frequency-time domain, respectively.

In the frequency-time domain-based approach, we utilize a tool called spectrograms. These nifty visualizations break down sound into its constituent frequencies over time, providing us with a rich representation of its spectral evolution. This approach is similar to what Ulyanov and Lebedev ([4]) or Wyse ([6]) did in their work, where they used short-time Fourier transform (STFT) to transform sound into a series of spectrogram images. These images are then processed by convolutional neural networks (CNNs), allowing us to capture and manipulate stylistic elements present in different frequency ranges.

On the other hand, we have the time domain-based approach, which takes a more direct route. Instead of fussing with the complexities of spectrograms, this method deals with the raw audio signal itself. Think of it as working directly with the sound wave, without breaking it down into its frequency components. Mital PK [17] took this path and worked with the real and imaginary parts of a Discrete Fourier Transform to achieve style transfer. The advantage here is that we sidestep the challenges of handling phase information and can potentially achieve real-time applications and better sound quality.

Let's not forget about the concept of audio texture. This is all about the finer details that make sound unique and recognizable. It's the subtle variations in timbre, rhythm, and harmonics that define a particular style. Both approaches, whether we're looking at spectrograms or working directly with the raw audio, need to account for and preserve these textural intricacies. It's the secret sauce that makes style transfer convincing and captivating.

D. Evaluation Methods

In the realm of Deep Learning research papers, the evaluation of results and model predictions often entails conducting specific tasks and performing on established benchmarks. This practice enables the research community to make comparisons between various models and methodologies. However, in the context of style transfer, there has been a noticeable absence of well-defined evaluation methods to assess different models for a given task. Even more concerning, a significant portion of papers on this subject lack comprehensive model evaluation or offer only a "qualitative" assessment.

A few papers have introduced inventive objective evaluation techniques for style transfer, encompassing essential musical attributes. For instance, "groove2groove" ([7]) employs an approach that quantifies style fit through the comparison of harmonic structures and musical event statistics. Notably, these evaluation methods delve beyond traditional

[1]For examples, see here: https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/.

spectrogram analysis, extending to structured data representations - they are performed on MIDI files.

In contrast, and given this landscape, we have determined that the direction of our project will involve establishing a benchmark against which models can be rigorously evaluated. This concept will be further elaborated upon in Section III.

E. Autoencoders

Autoencoders represent a class of neural network architectures used for unsupervised learning tasks, particularly in the domain of feature learning and data compression. The fundamental principle of autoencoders is to transform input data into a lower-dimensional representation, known as the latent space, and then reconstruct the original input from this representation. This process entails two main components: an encoder that maps input data to the latent space and a decoder that reconstructs data from the latent representation. A visual representation of the architecture is shown in Figure 5.

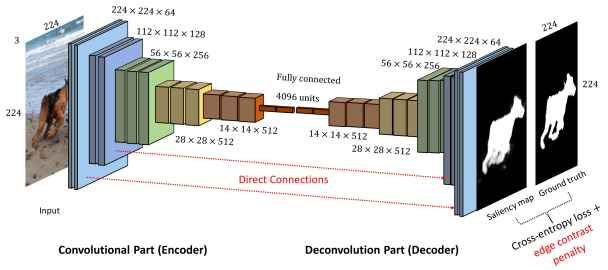

Figure 5. Example of an Autoencoder architecture (source: Gabor Melli Research knowledge graph)

Variational Autoencoders (VAEs) are an extension of autoencoders that introduce probabilistic modeling. VAEs encode data into a latent distribution, allowing for more flexible and expressive feature extraction. This is achieved by modeling the latent space as a probability distribution, typically Gaussian. VAEs enable the generation of new data samples by sampling from the learned latent distribution, making them useful for tasks like image synthesis and data generation.

Autoencoders and VAEs have found applications in diverse areas, such as image denoising, dimensionality reduction, and anomaly detection. In the realm of style transfer, while autoencoders have shown potential in capturing and transferring certain stylistic aspects of input data, the state-of-the-art methods have shifted towards more advanced techniques, such as GANs and neural style transfer networks.

For the scope of our research, we have chosen not to focus on autoencoders or VAEs. Instead, we are exploring alternative methodologies that align with our specific objectives and contribute to the advancement of our chosen direction.

F. Generative Models

Generative models have significantly impacted the realm of audio generation, particularly with the advent of groundbreaking techniques like Generative Adversarial Networks (GANs) ([8]) and more recently, Denoising Diffusion Probabilistic Models (DDPM) ([9]). These models have demonstrated remarkable capabilities in generating high-quality audio samples, contributing to the evolution of creative audio synthesis.

However, our research focus has steered us away from delving into this field. While generative models offer promising avenues for audio generation, our decision not to explore this avenue is driven by considerations of computational resources and our primary objective of advancing the chosen direction of our research. Notably, certain noteworthy models, such as TimbreTron ([10]), have harnessed the power of the WaveNet model ([11]) to achieve timbre transfer in audio, showcasing the potential for generative models in this domain.

Given our research scope and goals, we have chosen to channel our efforts toward a distinct approach that we believe holds promise for the enhancement of the field.

G. Conclusion and Discussions

The field of Audio Style Transfer is, as of today, shifting mainly to Generative AI. We cannot count up the number of voice deepfakes, sound isolation, etc. available online that truly demonstrate how advanced the field is. Nonetheless, there is always room for innovation, and we are going to contribute to this field.

III. Dataset

To achieve our goal and classify different instruments/genres, we needed a dataset with the following specificity: a large variety of tracks, with a version of each one for every class. We did not find such dataset during our research, so we decided to come up with our one - this is our first contribution.

One of the first styles of music that comes to mind, beside the common definition of music genre like Hip Hop, Jazz, etc., is instrument. When you have a track, we can think of the melody

A. MIDI format and Lakh Dataset

The easier way to create sounds is through a synthesizer, which automatically leads to MIDI files, a file format that represents musical information such as notes, timing, and control data. It is then very simple to generate tracks from various instruments.

The MIDI dataset we ended up choosing the ADL Piano MIDI [12], a dataset of 11,086 piano pieces from different genres. It is based on the Lakh dataset [13]. The ADL dataset was already a convenient subpart of the Lakh dataset we chose to create our own.

B. Transformation from MIDI files to Spectrograms

Once the MIDI file were chosen, we needed to transform them to audio file and spectrogram

1) ***MIDI to WAV****:* This step is programatically performed thanks to an open-source free of use software synthesizer, *Fluidsynth*, making use of soundfonts to create the sound texture. Here are the references and links (accessed on 23 March 2023) to the different soundfonts used for the track generation:

- Piano: Nice-Steinway-Lite-v3.0
- Guitar: Kona K2 Series K2T Acoustic-Electric Guitar
- Saxophone: Trumpet collection
- Trumpet: Maestro Clarinet Base
- Clarinet: Pseudo-Realistic Tenor Sax (Expressive)
- Organ: Hammond Organ 1Tr-B3

This step resulted in the creation of 6 times 226 single WAV files, that is 1356 track versions of the six instruments.

2) ***MP3 compression****:* This step was made necessary for storage purposes. Indeed, after the previous step, some of the individual WAV files took around 30 Mb of storage. This is not normal behavior for such file, usually taking not more than 5 Mb knowing that the mean length is . We stress it by supposing the soundfonts used for the generation were responsible in some way, them taking up to 70 Mb of storage.

To overcome this problem, we decided to compress every file, thanks to the open-source free of use software *FFMPEG*, to MP3 format. The size of the folder drastically dropped to 3.4 Gb, yielding an average size 2.55 Mb a file. The graph below (Figure 6) displays the boxplot of duration of our dataset:

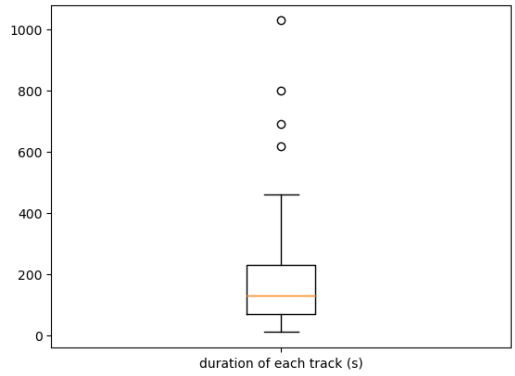

Figure 6. Boxplot of the duration of the 226 tracks

By the way, for more convenient training, we truncated every audio file into 10s chunks, using Librosa.

3) ***Spectrogram generation****:* This steps is the final of the transformation pipeline: it produces ready-to use spectrograms of the audio tracks. We will take a closer look at the choice of spectrogram type in Section III-C.

To transform our audio files into spectrograms, we use the python library Librosa ([14]), a versatile tool for audio analysis. Let's break down the code below to explain how we construct our spectrogram:

`y, sr = librosa.load(mp3_path)`: We load the MP3 audio file using Librosa, capturing the audio waveform y and the sampling rate sr.

`D = librosa.stft(y, n_fft=N_FFT)`: We compute the STFT of the audio waveform y. The parameter n_fft determines the window size for the analysis.

`S, phase = librosa.magphase(D)`: From the STFT representation D, we separate the magnitude spectrogram S and the phase information. The magnitude spectrogram encapsulates the amplitude of different frequencies over time. The phase information, on the other hand, conveys the cyclical changes and temporal relationships between these frequency components (it is used for a better signal reconstruction).

`S = librosa.power\_to\_db(S, ref=np.max)`: The magnitude spectrogram S is transformed into the decibel (dB) scale using the power_to_db function. This conversion enhances our ability to perceive a wider dynamic range of amplitudes.

`np.save(spectrogram_path, S)`: Finally, the resulting spectrogram is saved as a NumPy file at the specified location.

4) ***Data accessibility****:* Finally, the files were saved in a dedicated directory on the server, and will be made publicly available.

C. Types of Spectrogram

This section discusses the different types of spectrogram that can be used for classification and ST. We will also motivate our choice of spectrogram for the rest of the paper.

There are different spectrograms that we will take into account, belonging to two families:

- Mel Spectrograms
- Short-time-fourier-transform spectrograms

1) ***Mel Spectrogram****:* The Mel spectrogram is a type of spectrogram that utilizes the mel-scale to partition the frequency domain. The mel-scale is designed to better align with human auditory perception, emphasizing perceptually relevant frequency regions. In a Mel spectrogram, the vertical axis corresponds to mel-frequency bins, and the horizontal axis represents time. The spectrogram values in each bin reflect the energy or magnitude of the signal at specific mel-frequency bands. Mel spectrograms are particularly useful in tasks involving audio classification and style transfer due to their enhanced representation of human auditory characteristics.

2) ***STFT Spectrogram****:* The STFT spectrogram is another common representation of audio signals. It captures the frequency content of a signal over short time intervals, offering insights into the changing spectral characteristics. In

an STFT spectrogram, the vertical axis represents frequency, and the horizontal axis denotes time. Each point in the spectrogram illustrates the magnitude or energy of a specific frequency component at a given time. STFT spectrograms are valuable for various audio processing tasks, including time-frequency analysis, pitch detection, and audio transformation.

3) ***Log representation****:* When working with spectrogram data, it's common to apply mathematical transformations, such as logarithm, to enhance the representation. Here, two transformations particularly interests us: the `powerToDB` and the `log1p`.

The `log1p` function computes the natural logarithm of the input values plus one. It is often used to compress the dynamic range of spectrogram magnitudes, making the data more amenable to various analyses and algorithms. The `log1p` transformation is particularly useful when dealing with data that spans a wide range of magnitudes, as it can mitigate the effects of extremely high or low values.

On the other hand, the `powerToDB` transformation converts spectrogram values from power units to decibels (dB), which is a logarithmic scale. This transformation is commonly used to represent magnitudes in a way that aligns with human perception of loudness and to emphasize relative differences between magnitudes.

In the context of audio signal processing, the choice between `powerToDB` and `log1p` depends on the specific goals of the analysis. `powerToDB` is well-suited for situations where the emphasis is on perceptual loudness differences and relative comparisons. On the other hand, `log1p` is valuable when managing the dynamic range of spectrogram values while preserving the underlying linear relationships.

4) ***Conclusion and Decision****:* In order to decide which spectrogram to go with, let's visualize each of them for different audio in the figure below (Figure 7).

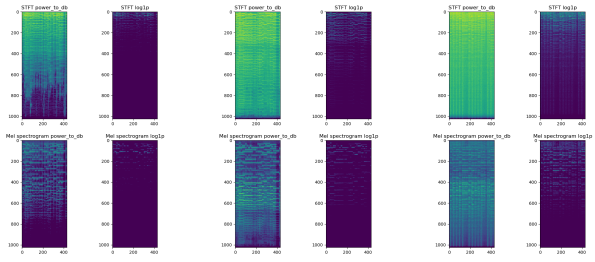

Figure 7. Difference between spectrograms of the 4 methods. For each, the first line is STFT while the second is Mel Spectrogram, and the first column is `powerToDB` while the second is `log1p`. From Left to Right: Empty Chairs at Empty Tables [6] - Piano, Empty Chairs at Empty Tables [6] - Saxophone, USA National Anthem

D. Conclusion and Discussions

In a nutshell, we have created a brand new dataset consisting of a same piece played on different instruments. We cherry-picked Piano, Guitar, Saxophone, Trumpet, Organ and Clarinet, and it would be interesting to grow such datasets with other instruments.

IV. Classifier

A. The objective function

In order to "objectively" classify the samples generated by the ST algorithm, we need to define an objective function. As stated in the Section II-D, it is very difficult to come up with an effective way to objectively assess a transfer of style. It is possible though, as as demonstrated in [15] and as we will see in the next section, we can classify instruments with a high accuracy using CNNs. Therefore, we are going to train a network that will serve as a classifier of style transfer.

This objective function could also addresses another problem. Ulyanov and Lebedev ([4]) stated that there were no such pretrained network classifier they could use for their ST algorithm. Although Shallow Networks with Random Filters are sufficient for Texture Synthesis ([16]), providing such a Network is still advancing the field.

Throughout this section, we will talk about our Multilayer CNN, the choices made in its architecture and the results of the training. This is our second contribution.

B. Convolutional Neural Network (CNN) and Model Architecture

Classifying spectrograms is a task that has garnered considerable attention and has been addressed by a multitude of models. While the architectures of these models exhibit some variations, a unifying component among them is the utilization of Convolutional Neural Networks (CNNs). CNNs represent a pivotal advancement in the field of computer vision. These networks have since found extensive application in classification tasks due to their inherent ability to capture and learn intricate patterns within data.

At its core, a Convolutional Neural Network is designed to automatically learn and extract relevant features from raw input data, such as images or, in our case, spectrograms. This is achieved through a hierarchical arrangement of layers, each of which performs specific operations on the input data. Convolutional layers apply a set of learnable filters or kernels to the input. These filters systematically slide across the input, performing convolutions that accentuate distinctive features like edges, textures, and shapes present in the spectrogram. In our case, we are most definitely interested in the texture of the spectrogram, but the training of the network will learn which features are the most relevant to classify the instruments.

Subsequent to the convolutional layers, CNNs often incorporate pooling layers, such as max pooling or average

pooling, which serve to reduce the spatial dimensions of the data while retaining essential information. This aids in creating a more compact and manageable representation of the spectrogram's features.

To further enhance the network's ability to capture intricate patterns, non-linear activation functions like Rectified Linear Units (ReLU) or Leaky ReLU are employed after each convolutional or pooling layer. These activation functions introduce non-linearity to the model, enabling it to capture complex relationships and variations within the data.

In the context of spectrogram classification, CNNs exploit their hierarchical architecture to progressively learn and combine increasingly abstract features. This process allows the network to discriminate between subtle differences in spectrogram patterns that might correspond to different classes, ultimately leading to accurate classification results. By leveraging the power of CNNs, researchers and practitioners have been able to achieve remarkable performance in classifying spectrograms, thereby contributing significantly to the advancement of various fields reliant on this form of data analysis.

C. Model Architecture

In this section, we describe the layers of our neural network architecture, and depicted below in Figure 8.

1) Convolutional Block 1: The first convolutional block (`conv_block1`) consists of the following layers:

- Convolutional Layer: Input channel = 1, Output channel = 8, Kernel size = 5×5, Stride = 2, Padding = 2.
- Batch Normalization: Normalizes the output of the convolutional layer.
- ReLU Activation: Applies the rectified linear unit activation function.

The first convolutional block (`conv_block1`) performs initial feature extraction. It takes the input spectrogram and applies a convolutional layer with 8 filters, each capturing different patterns in the data. Batch normalization ensures stable training, and the ReLU activation function introduces non-linearity.

2) Convolutional Block 2: The second convolutional block (`conv_block2`) consists of the following layers:

- Convolutional Layer: Input channel = 8, Output channel = 16, Kernel size = 3×3, Stride = 2, Padding = 1.
- Batch Normalization: Normalizes the output of the convolutional layer.
- ReLU Activation: Applies the rectified linear unit activation function.

The second convolutional block (`conv_block2`) continues feature extraction with 16 filters. By using smaller 3×3 kernels and a stride of 2, it captures more intricate patterns. Batch normalization maintains the network's stability, and ReLU activation fosters complex feature learning.

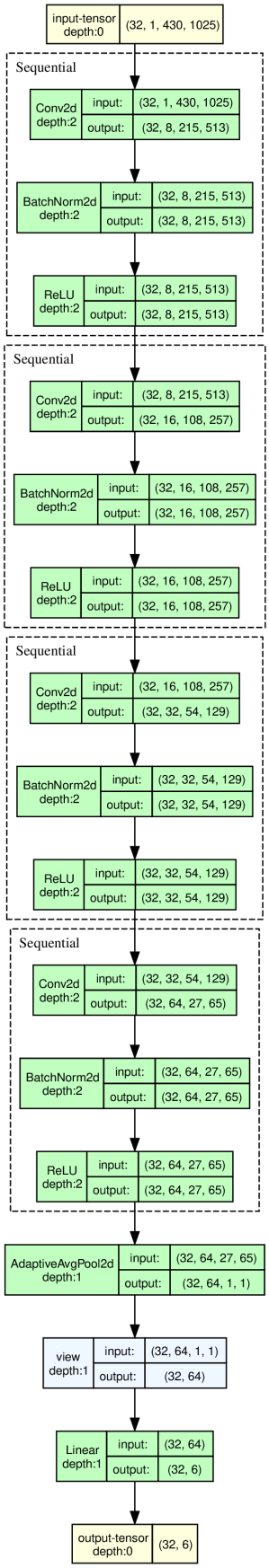

Figure 8. Architecture of the model (sizes for a 32 batch)

3) Convolutional Block 3: The third convolutional block (`conv_block3`) consists of the following layers:

- Convolutional Layer: Input channel = 16, Output channel = 32, Kernel size = 3×3, Stride = 2, Padding = 1.
- Batch Normalization: Normalizes the output of the convolutional layer.
- ReLU Activation: Applies the rectified linear unit activation function.

The third convolutional block (`conv_block3`) further refines features using 32 filters. The 3×3 kernels with a stride of 2 continue to reduce spatial dimensions while enhancing abstraction. Batch normalization ensures effective training, and ReLU activation introduces non-linearity.

4) Convolutional Block 4: The fourth convolutional block (`conv_block4`) consists of the following layers:

- Convolutional Layer: Input channel = 32, Output channel = 64, Kernel size = 3×3, Stride = 2, Padding = 1.
- Batch Normalization: Normalizes the output of the convolutional layer.
- ReLU Activation: Applies the rectified linear unit activation function.

The fourth convolutional block (`conv_block4`) intensifies feature extraction with 64 filters. It employs the same 3×3 kernel and stride of 2, emphasizing high-level feature capture. Batch normalization maintains learning stability, and ReLU activation facilitates complex feature representation.

5) Adaptive Average Pooling: The output from the last convolutional block is fed into an Adaptive Average Pooling layer, which reduces the spatial dimensions of the feature map to 1×1 while preserving channel information.

Adaptive Average Pooling transforms the feature map into a fixed-size representation (1×1), summarizing learned features. This step captures the most crucial information from the previous blocks while discarding spatial details.

6) Fully Connected Layer: Following the pooling layer, a fully connected (`fc`) layer is applied. It takes the flattened feature map from the previous layer and produces an output of size 6, corresponding to the number of classes in the classification task.

The fully connected (`fc`) layer acts as a classifier. It takes the flattened features from the previous layer and produces a vector of size 6, representing the probability distribution over the classes. This step translates learned features into class-related information.

7) Softmax Activation: Finally, the output of the fully connected layer is passed through a softmax activation function, producing class probabilities that sum up to 1.

The Softmax activation converts the output of the fully connected layer into a probability distribution across classes. Each value indicates the likelihood of the input belonging to a specific class, enabling confident classification decisions.

Now that we set up the network architecture, we can begin training the model.

D. Training

*1) **Training Framework and Hardware**:* Our training process benefits from Nvidia GPUs. We trained our model on two NVIDIA TITAN Xp of 12196MiB of storage. The model is coded using the PyTorch framework. We also make use of PyTorch Lightning, simplifying and optimizing the training workflow.

*2) **Loss Function and Optimizer**:* The loss function used to train the model is the Cross Entropy Loss. This loss function is commonly used in classification tasks, and it is defined as follows:

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \cdot \log(p_i) \tag{1}$$

where N is the number of classes, y_i is the ground truth label, and p_i is the predicted probability for class i.

We encoded our ground truth labels as one-hot vectors, and we used the Softmax activation function in the last layer of our model. This allows us to interpret the output of the model as a probability distribution over the classes. The Cross Entropy Loss is then used to compute the loss between the predicted probability distribution and the ground truth labels (which can be thought of as the scalar product between two vectors). We chose this particular loss function because it penalizes greatly completely wrong predictions: the closer to 0 p_i is, the greater the loss is.

With no surprise, the optimizer used to train the model is Adam from PyTorch ([17]), as it is commonly used in deep learning tasks. We did not tune its parameters, in particular, we used the default learning rate value of 0.001.

*3) **Monitoring Performance and Epoch Analysis**:* We monitored the performance of our model using two metrics: loss and accuracy. We let our model run for 100 epochs, and we used a batch size of 512 (the max before running CUDA out of memory).

The first metric, the loss, is computed using the Cross Entropy Loss function defined just before (Section IV-D2). It is computed on the training set after each epoch and provides insight into the training process. The following graph (Figure 9) shows its evolution over the course of the training.

The overall train loss is indeed decreasing in the number of epochs. We observe a similar behaviour for the validation loss, although some spikes which is expected.

The second metric, the accuracy, is defined as the number of correct predictions divided by the total number of predictions. It is computed on the validation set after each epoch and we used it to analyze the training process. The following graph (Figure 10) shows its evolution over the course of the training.

The same analysis as for loss can be made here, since both accuracies increase overall with the number of epochs,

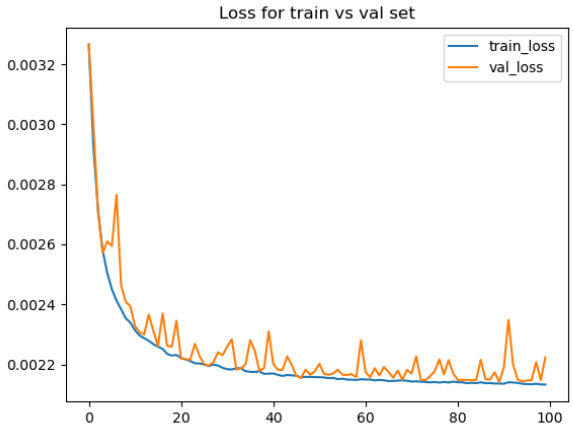

Figure 9. Loss evaluation on 30 epochs of training

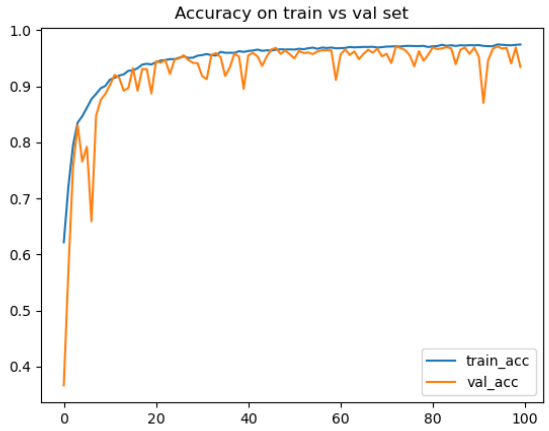

Figure 10. Accuracy evaluation on 30 epochs of training

with some peaks for the validation accuracy. It is also worth noting that the spikes in the validation accuracy are correlated to the ones in the validation loss (which makes sense).

In the end, we can see that both loss and accuracy converge, which is a good sign. The following section provides deeper insights on the performance of our model.

E. Testing and Evaluation

Once the model is trained, it is crucial to evaluate its performance using various techniques to ensure its effectiveness. Here are three key ways we implemented to evaluate our model's performances:

*1) **Overfitting Analysis**:* To determine if the model has overfit the training data, it is essential to evaluate its performance on both the training and validation datasets. By comparing the training and validation loss, we can identify any significant gaps between them. If the training loss is much lower than the validation loss, it could be an indication of overfitting.

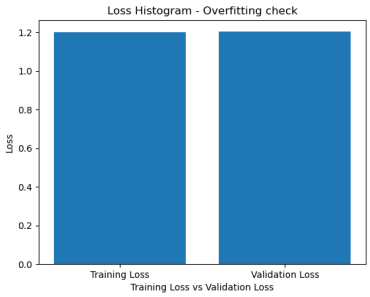

Figure 11. Comparison of Training and Validation Loss

And as reassuring as it seems in Figure 11, both loss seems very close. Indeed, they only differ on the thousandth (1.2012 vs 1.2034!). This ensures our model does not overfit the train data.

*2) **Cross-Validation**:* Cross-validation is a robust technique that involves splitting the dataset into multiple subsets (folds) and training the model on different combinations of training and validation folds. This helps to assess the model's generalization ability and ensures that it is not biased by a specific dataset split. The average performance across all folds provides a more accurate measure of the model's performance. We divided our dataset into 4 folds. Because precision and recall are originally defined for binary classification, and that recall does not make sense in the context of multi-class classification, we computed a modified version of the precision that is averaged along every class. This is commonly called the "micro" average.

Fold	Precision
1	0.602
2	0.916
3	0.603
4	0.919
Average	0.760

Table I
CROSS-VALIDATION RESULTS

The cross-validation results (I) reveal moderate variance in precision across folds. Although variations exist, they are not excessive, indicating reasonable consistency in model performance across different validations. Considering the complexity and inherent variability of the data, this suggests a reliable model performance.

*3) **Confusion Matrix**:* The confusion matrix (Figure 12) is a valuable tool for visualizing the model's performance across different classes. It displays the true positive, true negative, false positive, and false negative counts, enabling a deeper understanding of the model's strengths and weaknesses.

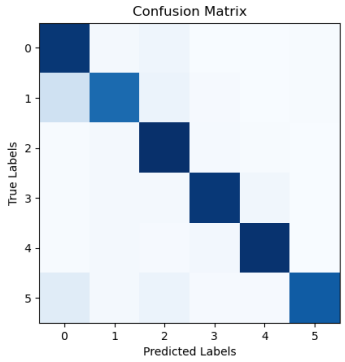

Figure 12. Confusion Matrix Heat-map (Labels from 0 to 5: Guitar, Piano, Clarinet, Sax, Trumpet, Organ.

It confirms a strong ability of the model to classify. Indeed, most of the classes are strongly correlated with themselves only, especially the 0, 2, 3, and 4, namely Guitar, Clarinet, Sax and Trumpet. The model misclassifies mainly, but rarely, Piano for Guitar, and Organ for Guitar.

F. Conclusion and Discussions

By utilizing the different evaluation techniques, we have gained important insights into the model's performance. Indeed, it is doing well on our dataset, and sanity checks related to common problems showed no further parameter change were needed.

V. Experiments and Results

A. Implementing Ulyanov's method

Using the same environment as for the training and testing of the model, we designed a pipeline that would take two spectrogram inputs, namely one for the content and one for the style, and that would output the generated audio, combining the style and content of the respective inputs. We took inspiration from other repositories' code. First of all, of course, we reused some of Ulyanov's code ([18]). We also reused and aggregated code from other repositories ([19], [20] and [21]). We also used Github Copilot ([22]) as a helper to write code. Our repository architecture, though, was designed by us.

1) The network: we import the previously trained network in the pipeline. It is what we will use to compute the feature maps of the different inputs.

2) The optimizing image: we initialize the optimizing image with a normal distribution of mean 0 and standard deviation 90 (inspired by [19]).

3) The loss function: we defined the loss function as in the original paper ([1]) or in Ulyanov's work ([4]). We started with weights $\alpha = 1e-3$ and $\beta = 1$, keeping the desired ratio but emphasizing more on the style loss than the content.

4) The optimizer: we decided to go first with Adam instead of a L-BFGS optimizer, as opposed to Ulyanov. This choice is not motivated by anything other than popularity, as both method seems to perform sufficiently on ST tasks (as showcased in this Medium blog post [23]). The image is then optimized using PyTorch.

5) The evaluation: the resulting spectrogram is then evaluate in two different way: first, we classify it using out pretrained network (Section IV) and second we reconstruct the sound wave using Griffin-Lim algorithm ([5]).

B. Experimenting with our method

As everything is set, we can begin transferring style, and this is were troubles began: our implementation wouldn't work. Despite a very high emphasize on the style loss, the algorithm couldn't find the desired optimization point, that is a spectrogram with the style transferred. We even tried to let the algorithm run for thousands and thousands of step, but still the resulting spectrogram looked like random noise, was classified as the content one, and sounded like the content one. We did reiterate the experience with different combinations of instruments and tracks, but the problem persisted.

This has two ways to go: either our pipeline is wrong, or our data is problematic. Either cases, we need to investigate the matter. So, we searched for similar problems in the literature, but, to the best of our knowledge, nothing comparable have been found. We also looked at different variation of the method and tested out things that could help us solve our problem.

1) More iterations: As seen in the images below 1516, augmenting the number of iteration does not improve the result and the optimization seems to find some local minimum even if the output is not the desired one (see evolution in Figures 1314).

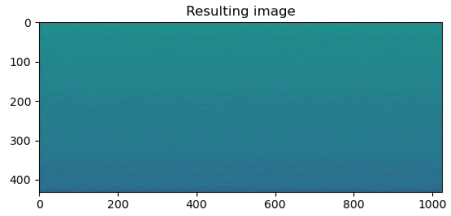

Figure 13. Evolution of the resulting spectrogram at the 300th iteration.

The resulting spectrogram at iteration 3000 really looks like a spectrograms, but isn't one. We can see in Figure 16 that this image were unlikely to move since a while (around iteration 500) and is indeed a local minima.

2) Sanity checks: We carried out multiple checks in order to verify the method we implemented was good. We tried to run the algorithm to minimize the image on the content only, so that we would be sure it optimizes, and it did. We

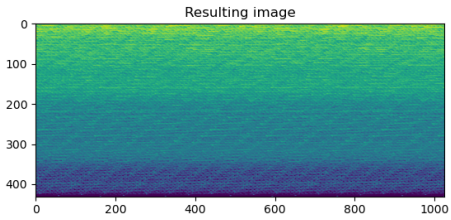

Figure 14. Evolution of the resulting spectrogram at the 3000th iteration.

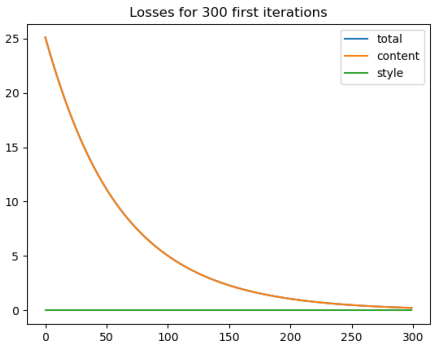

Figure 15. Evolution of the $loss = \alpha \times contentLoss + \beta \times styleLoss$ between 0 and 300 iterations.

checked if the model was altered during the procedure, it wasn't.

3) Other Starting Points: Starting from the content or the style images did not solve our problem. Indeed, on the content case, we ended up just as in the random case, and in the style case, we stayed at the starting point.

4) Total Variation: We tried adding a total variation loss to the loss, but it did not help either. The image below (Figure 17) shows us how the TV loss influenced the algorithm (and it is not in a good way).

5) Change HP: In order to assess whether the hyper-parameters α and β play a role in our settings, we propose

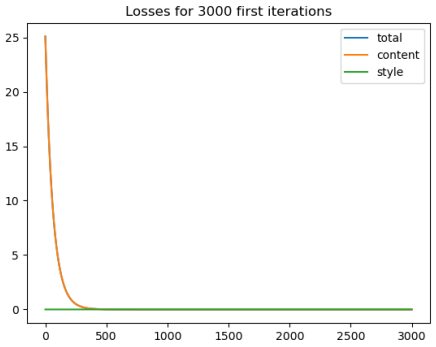

Figure 16. Evolution of the $loss = \alpha \times contentLoss + \beta \times styleLoss$ between 0 and 3000 iterations.

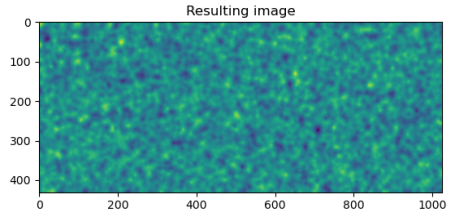

Figure 17. Optimization with TV loss

to conduct a grid search analysis (see Figure 18). It did not reveal major flaws in the implementation, as lower values of alpha and beta automatically yield a lower loss.

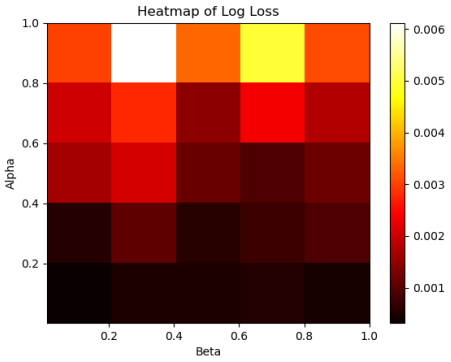

Figure 18. Heat-map of the loss value after 1000 iterations (value of α/value of β)

6) Use Ulyanov's implementation: Even when using Ulyanov's implementation (available online [18]), while its samples would give nice results, it wouldn't converge for ours, as if something was truly wrong with our data...

C. Why it doesn't work

As of today, we cannot clearly state nor rigorously demonstrate why no algorithm we tested could orchestrate the desired style transfer. It has been very frustrating and time-consuming. The only explication we can come up with is the way the dataset were generated. Indeed, to the best of our knowledge, we have not seen anything like MIDI2Wav file conversion for ST. Research either concentrated on the structure of the MIDI format or on spectrograms of sound waves. In our settings, the fact that the audio files contain "perfect" audio, could be the root of our problems.

As for the future, this could lead to another new metric: the amount of information contained in a spectrogram. We can informally know whether a spectrogram contains information or not by looking at it, but it does not seem to be a way to formally and uniformally assess that. One idea that arose during the project was to create a norm based on the Gram Matrix.

VI. Conclusion and Future Work

Throughout this paper and our research, we have had the occasion to review the literature on the subject of Audio Style Transfer. We found that no real objective evaluation were made on the results of style transfer algorithm and decided to investigate this matter. We created a new dataset and successfully trained a classifier on it. The idea was to evaluate results of style transfers using our estimator. However, everything did not go as planned, as we did not achieved to transfer the style of one of our generated samples. This is almost surely due to the way the audio file were generated, as they are not "real world" audios.

For the future of this field, generate a truly usable benchmark to perform audio style transfer could be a great advance. This dataset should combine a great diversity, as well as multiple same content for different style examples. This is what we tried to do.

VII. Acknowledgement

I would like to dedicate this section to Giulio, who has been a dedicated helper and guide throughout this entire project. Your support and insights have been invaluable, and I am truly grateful for your contributions. Thank you for being an essential part of this journey.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[2] ——, "Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks," *CoRR*, vol. abs/1505.07376, 2015. [Online]. Available: http://arxiv.org/abs/1505.07376

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[4] D. Ulyanov and V. Lebedev, "Audio texture synthesis and style transfer," Dec 2016. [Online]. Available: https://dmitryulyanov.github.io/audio-texture-synthesis-and-style-transfer/

[5] D. Griffin and J. Lim, "Signal estimation from modified short-time fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984.

[6] L. Wyse, "Audio spectrogram representations for processing with convolutional neural networks," 2017.

[7] O. Cífka, U. Şimşekli, and G. Richard, "Groove2groove: One-shot music style transfer with supervision from synthetic data," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 2638–2650, 2020.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[9] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[10] S. Huang, Q. Li, C. Anil, X. Bao, S. Oore, and R. B. Grosse, "Timbretron: A wavenet(cyclegan(cqt(audio))) pipeline for musical timbre transfer," 2019.

[11] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016.

[12] L. N. Ferreira, L. H. S. Lelis, and J. Whitehead, "Computer-generated music for tabletop role-playing games," 2020. [Online]. Available: https://arxiv.org/abs/2008.07009

[13] C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching," 2016. [Online]. Available: https://academiccommons.columbia.edu/doi/10.7916/D8N58MHV

[14] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015.

[15] A. Solanki and S. Pandey, "Music instrument recognition using deep convolutional neural networks," *International Journal of Information Technology*, vol. 14, no. 3, pp. 1659–1668, Jan. 2019. [Online]. Available: https://doi.org/10.1007/s41870-019-00285-y

[16] I. Ustyuzhaninov, W. Brendel, L. A. Gatys, and M. Bethge, "Texture synthesis using shallow convolutional networks with random filters," 2016.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[18] D. Ulyanov, "neural-style-transfer-tf," 2016. [Online]. Available: https://github.com/DmitryUlyanov/neural-style-audio-tf

[19] A. Gordić, "pytorch-neural-style-transfer," 2020. [Online]. Available: https://github.com/gordicaleksa/pytorch-neural-style-transfer

[20] A. Dipani, "Neural-style-transfer-audio," 2021. [Online]. Available: https://github.com/alishdipani/Neural-Style-Transfer-Audio

[21] inzva Sanctuary of The Turkish Hacker Community, "Audio style transfer," 2020. [Online]. Available: https://github.com/inzva/Audio-Style-Transfer

[22] GitHub Development Core Team, *GitHub Copilot*, GitHub, 2021. [Online]. Available: http://copilot.github.com

[23] S. Ivanov, Mar 2017. [Online]. Available: https://blog.slavv.com/picking-an-optimizer-for-style-transfer-86e7b8cba84b