Short-time Fourier transform hyperparameters matter: Timbre translation between musical instruments

José Antonio Betances
ancesdotai@gmail.com

Abstract

I propose a methodology to perform a timbre translation from an instrument that is playing a piece of music, to another different instrument, preserving the original music. Such a procedure uses the Short Time Fourier Transform (STFT) to convert an audio signal into spectrograms, which are processed by a Recurrent neural network, which generate other spectrogram that are reversed to audio signals by the Inverse Short Time Fourier Transform (ISTFT). The effect of different *Frame length* and *Frame step* is studied, obtaining very promising results.

1. Introduction

For the sake of simplicity, we can define the timbre of a sound as a set of characteristics (harmonic patterns, rhythmics, noise, transitions, among other things), which make up an identity that enables the mind to make a distinction of such sound over others [1,2].

If the timbre were studied with numerical methods, the problem could be addressed, initially, by plotting a digital signal on an amplitude vs time graph (Figure 1). This would result in images that could reveal information about the sound envelope, which can help distinguish at least one aspect of the timbre: The intensity variation.

But going further, the *Short Time Fourier Transform* (STFT) function could be used to obtain a digital spectrogram (Time-Frequency representation) that is going to be complex-valued [3]. If it were drawn by assigning a color that varies its intensity depending on the amplitude of such numbers, this would reveal more complete information about the timbre, being possible to see, according to time, not only the intensity variations, but the harmonic components of the signal (Figure 2).

Such spectrograms can be reversed to amplitude signals vs. time (Figure 1), which can be easily audible through a digital-to-analog converter, so if the spectrogram were altered, this is going to be translated in alterations in the converted signal.

Thus, it is tempting to think that machine learning models could be developed to modify the timbre of a signal, by processing its spectrograms.

2. Objective

Let's assume two different musical instruments, which produce sounds with different timbres. The intention is to find a method that allows translating the timbre between both instruments with high fidelity, processing the Time-Frequency representation of them. Such a translation should preserve musical aspects such as rhythm and tone.

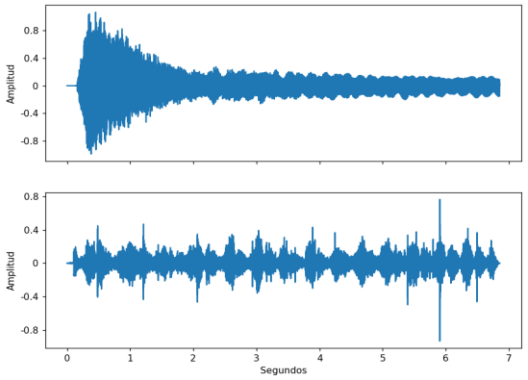

Figure 1: Plot signals from a bell (top) and a steam train (bottom). Note the difference in their forms

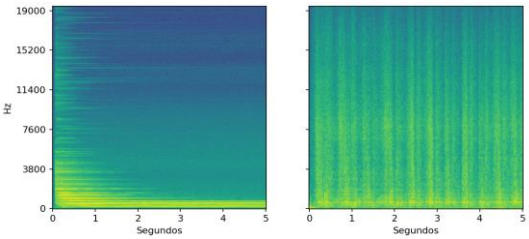

Figure 2: Time-frequency representation (spectrogram) of a bell (left) and a steam train (right), the same sounds as Figure 1.

3. The Architecture

The diagram of the architecture can be seen in Figure 3:

We have a real-valued digital signal S, represented in a vector of shape (N), where each number represents a sample of amplitude n. S is divided into M chunks of equal length. Each chunk represents an example m_i.

The *STFT* function is applied to each m_i, which divides the example into A windows a_i of equal length, *followed by the window function* W, and then the Fast Fourier Transform (*FFT).*

Thus, the audio signal S is transformed to a vector X, with the shape (M,T,U), where T corresponds to time bands t, and U, to complex-valued frequency bins u_i.

X goes into the Machine learning model, which have an output $\hat{y}$, to which the *Inverse Short-time Fourier Transform (ISTFT)* function is applied. For this, the Inverse fast Fourier transform *(IFFT)* in applied over $\hat{U}_i$. Later, the same *function* W is used, and them are added to build each chunk $\hat{m}_i$. Finally, all $\hat{m}_i$ are concatenated to form the output signal $\hat{S}$, with the same shape and type of S.

The shape of X and $\hat{y}$ are equal and depend essentially on three *hyperparameters*: The size of m_i, the *Frame length* (*Fl*) and the *Frame step* (*Fs).*

The Machine Learning model is sketched in Figure 4:

1. Complex-valued input X with the shape (M,T,U)
2. Bifurcation into real and imaginary parts
3. Concatenation c of the real and imaginary part
4. Bidirectional *LSTM1*, 1000 hidden units [4]
5. Bidirectional *LSTM2*, 1000 hidden units
6. Bifurcation by copy (*c1* and *c2*)
7. Densely connected layer for each part of the fork. Linear activation function (D1 and D2)
8. Complex-valued output ŷ=D1+iD2

With this method, the model must learn how to convert the input signal S into the output signal $\hat{S}$.

4. The dataset

For the generation of the dataset, a MIDI sequence was made that swaps each note between a D#3 and an F5 (26 notes) for 6 repetitions, each, at twice the speed of the previous one. The reason for such decisions is to provide the model with enough examples of notes, both short and long, as well as relationships between them.

All notes have the same intensity. The same sequence was *rendered* for both piano and harpsichord, at a sample rate of 44100 Hz, 16-bit depth, *wav* format, monaural, no added effects, generating two files of 40'00" each.

Additionally, a sequence of the first notes of a J.S. Bach fugue was rendered under the same conditions (BWV 846, Fugue 2 in C minor), which is used to inspect by hearing, the performance of the architecture.

The sound of both instruments comes from Garritan Personal Orchestra 5 [6]

4. Training

The model was trained with the exported *wav* files, using a *Hanning* window function [3], and varying the *Fs* for the values 1,2,4,8,16,32,64 and 128, and for each of these, the *Fl* was varied for the values 258,512,1024,2048,4096 and 8192. The length of the timestep was intended to be less than 1000 steps, to take full advantage of training LSTM models in CUDA.

The spectrogram of the piano was used as an input and the harpsichord as the output. Mean square error was used as a loss function, *Adam* as an optimizer at a learning rate of 0.0001, with

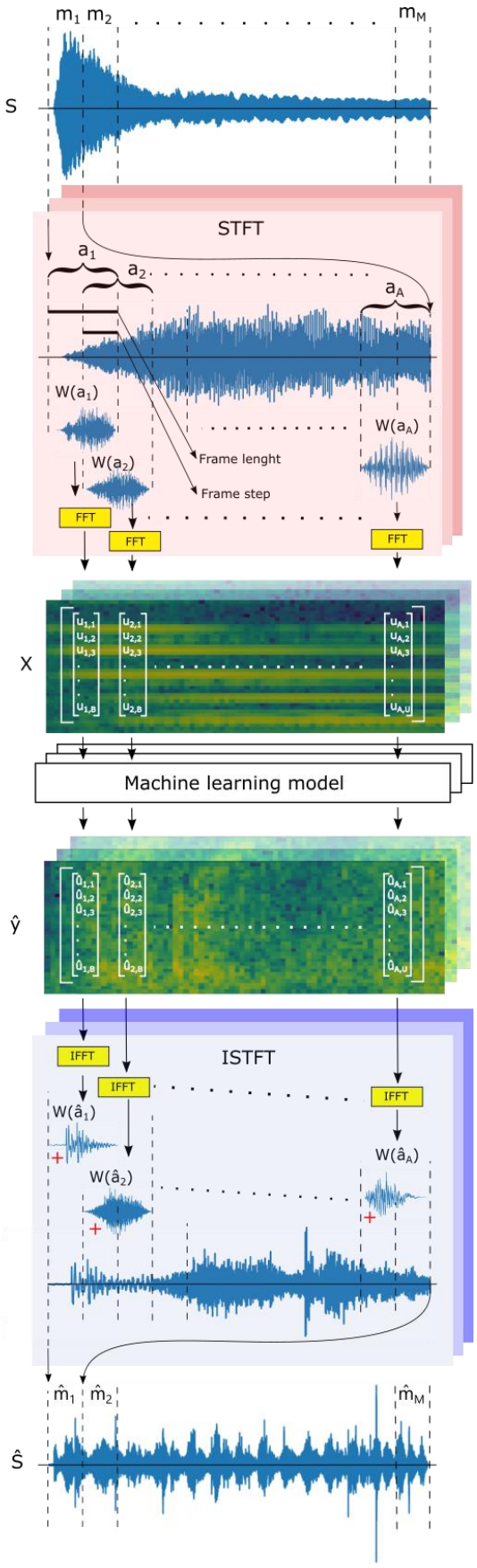

Figure 3: Diagram of the architecture

a *batch* size that varied between 4,8,16,32,64 and 128 depending on GPU memory consumption.

Every 10 epochs, a test translation was automatically saved. The *dataset* was divided into 95-5% for training and testing and allowed to train until the test sample stopped improving significantly with each cycle.

A PC with an Nvidia RTX 2080 Ti GPU was used, and Tensorflow 2.8 as the gradient framework.

5. Results

When inspecting by hearing the test sounds, it was noticeable that for large *Fs*, there was a very annoying noise, that decreases as the *Fs decreases*. For *Fs*=64 and *Fs*=32, certain *grains* were noticeable, as if there were no uniformity in the sound. These *grains* began to be imperceptible for very small values, disappearing completely for *Fs*= 2 and *Fs*=1, although with this, a loss in the higher frequencies was noticeable, which was evident in a frequency analyzer (Figure 5). This behavior was similar for all *Fs* that were tested.

On the other hand, very small *Fl* (258,512) made the sound very artificial and irregular with noticeable damage to the transients, which was improved for *Fl*=1024 and *Fl*=2048. For *Fl*>2048, the sound lost definition in the notes attack, which became even more evident for *Fl*=8193.

The test with the best sound was obtained with *Fs*=4 and *Fl*=1024, which gives 513 frequency bins. When this was compared to the true test sample, it was evident that it approaches very well the ideal to be reached, although it does not have a pristine sound.

6. Discussion

The behavior of *Fl* is due to a problem inherent in the *STFT*: High frequency resolutions lead to low resolutions over time, and high resolutions over time mean low resolutions in frequency.

When you take a window a_i with length *N* of a discrete signal with a sampling rate *f*, and apply an *FFT* to it, you get a complex-valued signal with the same length *N* and only half a part plus one is useful, because *FFT* generates a conjugate of the first half of a_i in reverse order.

Such frequency bins range from 0 to *f/2 Hz*, and each bin has a size of *f/N Hz*. Extending *N* increases the resolution by having more frequency bins. Extending it too many causes the transient to end up *within* the frequency bands, rather than being a consequence of time variations between windows.

A simple way to understand it might be to take the hyperparameters to extremes: Let's assume to apply the *STFT* with 8 seconds of frame length over a 10 second audio signal. This would imply almost the entire signal would enter in each time slot, which would result in few time slots, of many frequency bins.

Regarding the *Fs*, the fact that it works well with low values could be due to what happens with the phase. In Figure 6, it is seen as a pattern that is revealed *random* to the naked eye (*Fs*=512), begins to gain coherence and be predictable as *Fs* decreases, which helps the model a lot, since obviously, it is easier to predict deterministic events than random ones.

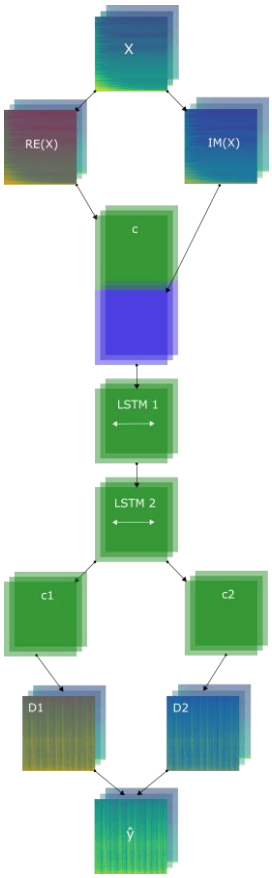

Figure 4: Machine learning model

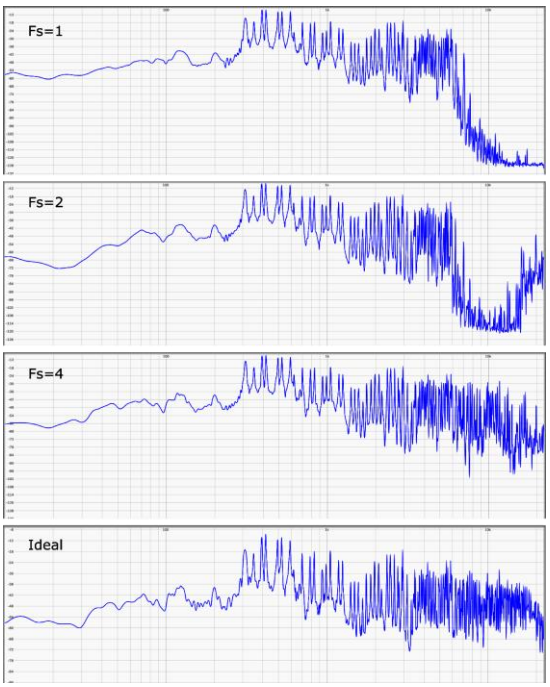

Figure 5: Frequency analysis of different Fs with Fl=1025.

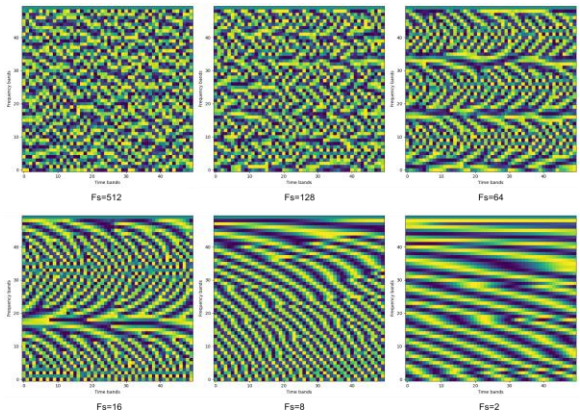

Figure 6: Comparison of what the phase looks like for different Fs

The reason for the deterioration of high frequencies that occurred when $Fs < 4$ is still unknown.

On the other hand, it was observed that the architecture proposed in this research has an important limitation: The output is very sensitive to the input. A small variation in input sound (amplitude, or some reverb), if the model has not been trained to handle such variation, distortions in the result of the translation would be appreciated. I speculate that this could be solved with a more elaborated machine learning model.

6. Conclusions and future work

This research paper has addressed the problem of timbre translation between musical instruments by applying *STFT* and *RNR* machine learning model.

It was observed that the correct choice of *Fl* and *Fs* is fundamental for this type of problem, although the optimal values for such *hyperparameters* (if there were) have not yet been discovered.

The approach proposed in this research paper is interesting and promising for the following reasons:

a. Real-valued Neural networks are used to process and generate complex-valued spectrograms. This eliminates the need to use phase reconstruction algorithms [7]. It would be interesting to develop the complex-valued LSTM equations, as well as GRU and RNR [8], taking advantage of the universal approximators property of complex-valued neural networks have. [9]

b. The architecture is very simple and intuitive. With more elaboration, it could become a good alternative to the current state of the art, in terms of sound processing and generation [10,11] as well as other signals in general.

Acknowledgments

This research has been inspired by Pix2Pix Timbre Transfer [12], which made me wonder if a neural network, being a universal approximator [13], should be able to approximate the phase of a spectrogram.

References

[1] McAdams, S., & Giordano, B. L. (2016, enero 1). The Perception of Musical Timbre. The Oxford Handbook of Music Psychology. https://doi.org/10.1093/oxfordhb/9780198722946.013.12

[2] Smalley, D. (2009). Defining timbre—Refining timbre. Contemporary Music Review. https://doi.org/10.1080/07494469400640281

[3] Allen, J. B., & Rabiner, L. R. (1977). A unified approach to short-time Fourier analysis and synthesis. Proceedings of the IEEE, 65(11), 1558-1564. https://doi.org/10.1109/PROC.1977.10770

[4] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. Neural computation, 9, 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

[5] Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. arXiv:1609.03499 [cs]. http://arxiv.org/abs/1609.03499

[6] Personal Orchestra 5: Symphony Orchestra Sample Library - Garritan GPO5. (s. f.). Garritan. Retrieved May 27, 2022, from https://www.garritan.com/products/personal-orchestra-5

[7] Masuyama, Y., Yatabe, K., Koizumi, Y., Oikawa, Y., & Harada, N. (2019). Deep Griffin-Lim Iteration. arXiv:1903.03971 [cs, eess]. http://arxiv.org/abs/1903.03971

[8] Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. arXiv:1409.1259 [cs, stat]. http://arxiv.org/abs/1409.1259

[9] Voigtlaender, F. (2020). The universal approximation theorem for complex-valued neural networks. arXiv:2012.03351 [cs, math, stat]. https://link.springer.com/article/10.1007%2FBF02551274

[10] Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. arXiv:1609.03499 [cs]. http://arxiv.org/abs/1609.03499

[11] Engel, J., Hantrakul, L. (Hanoi), Gu, C., & Roberts, A. (2019, septiembre 25). DDSP: Differentiable Digital Signal Processing. International Conference on Learning Representations. https://openreview.net/forum?id=B1x1ma4tDr

[12] Martel, H. (2022). Pix2Pix Timbre Transfer [Python]. https://github.com/hmartelb/Pix2Pix-Timbre-Transfer (Original work published 2019)

[13] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2(4), 303-314. https://doi.org/10.1007/BF02551274