Reimplementation, validation and analysis of the Differentiable Digital Signal Processing model
ATIAM Machine Learning Project

BIROT Olivier
CHAUMENY Lucas
DE HILLERIN André
PARADES Victor
ROMERO Gonzalo

January 2020

Abstract

While today's end-to-end learning methods seem to outperform many of the state of the art algorithms in the field of signal processing, their use rises two major concerns. With less constraints the convergence may be better in the end, but the complexity of the relationships between the parameters does not further our understanding of the modeled behavior. Furthermore, the improved convergence is superior in the final state of training but because of the sheer size of the models required by this approach, end-to-end methods also need a much greater computing power to attain the same degree of precision. Nowadays these observations give rise to a new class of hybrid approaches where some known essential priors or constraints are put onto the model and the detailed modeling is handled by more standard learning methods. This is the idea behind the DDSP library [aut20] which aims at generating the audio output of a specified and learned instrument, given the temporal evolution of the frequency and loudness to be played. The structural constraints applied to the model have the advantage of being close to the way sound is physically emitted while remaining entirely modular and, above all, differentiable. The main goal of this project is to evaluate the use and potential of such an approach. Four types of instruments are chosen to be the focus of the audio synthesis achieved by the neural network: a violin, a saxophone, a standard synth and a piano.

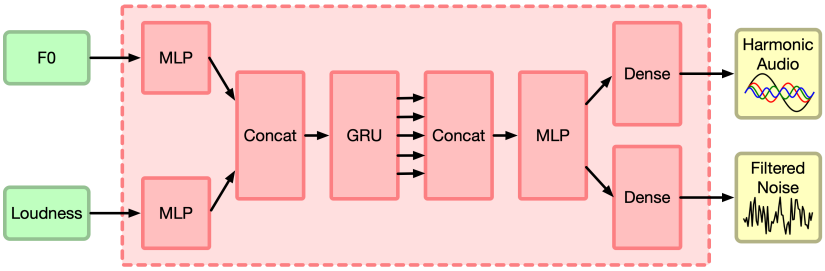


Figure 1: Architecture of the DDSP model [aut20]

1 Introduction

1.1 State of the art

1.1.1 Machine learning and autoregressive models

Due to the high dimensionality of the problem of audio synthesis, deep learning has been very popular as a tool and end-to-end model for generating sound. For example, WaveNet [AvdOK16], WaveRNN [NKK18] and SampleRNN [SMB16] are powerful autoregressive waveform models in that they can be used to generate a waveform a sample at a time, thus avoiding phase-alignment and spectral leakage problems. Nonetheless, these models also present undesired unreliability, as they conclude in incongruent losses since they do take into account specific qualities such as phase alignment which are not perceptualy relevant [aut20].

1.2 The DDSP model

With the goal to generate sounds in a more realistic and perceptually closer way, some "physical" models use signal processing techniques like additive synthesis and analysis in order to extract precise and useful data, such as frequency and loudness. The Differentiable Digital Signal Processing (DDSP) [aut20] model offers a harmonic and noise additive architecture to generate the raw waveform. It follows the heuristic of using priors as structural constraints to enforce our knowledge on the modeled behavior and give the neural network a modular architecture where the different parts correspond to physical phenomena. This not only gives us a better understanding of the learning process that takes place, it also guides the network via biases and therefore gives a better learning efficiency.
Nevertheless, the DDSP model retains a great expressivity in the space of harmonic sounds through the Fourier decomposition of signals : additive synthesis can, in theory, express any signal. As the desired result is perceptual, we can limit the model to a finite number of oscillators.

The initial DDSP model does include an encoder network which outputs f_0 over time, the frequency of the analysed frames, their loudness and an optional latent variable. The decoder part is our point of interest as it is the part of the model which is used to synthesize the sound of a learned timbre given the temporal evolution of f_0 and the loudness. We therefore used CREPE [KSLB18] for the pitch estimation and data gathering while the loudness is computed as a sum of the power spectrum over the frequencies, based on a STFT of size F_s/F_w with 25% overlap. The model is structured as follows: two MLP are fed with the f_0 and loudness values. Then, we concatenate the outputs and we rout them in a GRU cell, giving the network temporal expressiveness. After another MPL layer, two dense layers generate on one hand the global amplitude and the harmonic profile of the signal to be generated, and on the other hand the coefficient of the filter that will be filtering the noise component. A scheme of this architecture is shown in the figure 1
In order to have positive amplitudes we apply a sigmoid to the outputs. This constrains the tensors used for synthesis

to be between 0 and 1. However, the original paper mentions that they had better results using the modified sigmoid

$$s(x) = 2 \cdot \text{sigmoid}(x)^{\log(10)} + 10^{-7} \ .$$

2 Sound synthesis

The output of the model is a tensor y of dimension $[*, F_w \cdot T, 1 + N_h + N_n]$ where $*$ is the batch dimension, F_w is the f_0 and loudness rate (in Hz), T is the duration of the snippet (in s), $1 + N_h$ is the dimension of the harmonic part (we take 1 dimension for the amplitude and N_h harmonics) and N_n is the dimension of the noise part (the number of noise bands). The values we have chosen for these parameters are shown in table 1.
We can then name the tensors $a_i(N)$, $b_j(N)$ as follows

$$\begin{aligned} a_i(N) &= y(*, N, i) \ , \quad i = 0, 1, ..., N_h \\ b_j(N) &= y(*, N, N_h + j) \ , \quad j = 1, 2, ..., N_n \end{aligned}$$

where $N = 1, 2, ..., F_w \cdot T$.
The tensors $a_i(n)$ and $b_j(n)$ will be used for creating respectively the additive and the noise part of the sound. We explain how this is done in the following sections.

2.1 Additive synthesis

In order to have frequencies and amplitudes going at the sample rate, we have to interpolate them from F_w to F_s. We chose to achieve this by linear interpolation. We shall rename the variable N which goes from 1 to $F_w \cdot T$ to n which is expressed in samples and goes from 1 to $F_s * T$. Then, all the tensors are now expressed *via* the time variable n.
So we have the $f_0(n)$ tensor which is the fundamental frequency tensor[1]. We then create the tensor $f_i(n) := i \cdot f_0(n)$, $i = 1, 2, ..., N_h$. Then, we calculate the phase increment tensor $\phi_i(n) = 2\pi \frac{f_i(n)}{F_s}$. For getting the phase tensor $\Phi_i(n)$ we have to accumulate the phase so we define $\Phi_i(n) = \sum_{m=1}^{n} \phi_i(m)$.
We will use the tensor $a_i(n)$ as amplitude. We first separate $a_0(n)$ which will take account of the global amplitude from $a_i(n)$ which will be the harmonic distribution. Then, in order to prevent aliasing, we set the amplitudes of the frequencies over $\frac{F_s}{2}$ to be equal to zero. We normalize them so we have $\tilde{a}_i(n) = \frac{a_i(n)}{\sum_{i=1}^{N_h} a_i(n)}$. Finally, we use all this tensors to create the additive part of the signal

$$s_h(n) = a_0(n) \sum_{i=1}^{N_h} \tilde{a}_i(n) \sin(\Phi_i(n)) \ .$$

2.2 Noise synthesis

In order to generate the noise part of the synthesis, we filter a white noise. The filtering parameters are encoded in the $b_j(N)$ tensor as follows.
The idea is to replace a time-domain convolution by a frequency-domain complex product. So, we first create a white noise $w(n)$, $n = 1, 2, ..., F_s \cdot T$. We split it in F_w snippets $w_N(m)$, $N = 1, 2, ..., F_w$, $m = 1, 2, ..., \frac{F_s}{F_w}$ and we perform a DFT[2] of size N_f on each, obtaining a tensor $W_N(\omega) = DFT[w_N(n)](\omega)$, $\omega = 1, 2, ..., N_f$. Since this tensor is a Hermitian, symmetrical tensor, we can only keep the first $\left\lfloor \frac{N_f}{2} \right\rfloor + 1$ values.
We now have the $b_j(N)$ values taken from the output vector and we interpret them as the amplitudes of the frequency bands of the filter. However, in order to follow a most common filter design and match the size of $W_N(\omega)$, we perform a zero-padding and apply a Hamming window in the time domain, before heading back to the frequency

[1] This fundamental frequency is calculated by CREPE [KSLB18] and is both an input for the model and a tool for the re-synthesis.
[2] We use the FFT algorithm in order to compute it fast. We set then N_f the first power of 2 bigger than $\frac{F_s}{F_w}$.

Parameter	symbol	value
Batch size	$*$	6
f_0 and loudness rate	F_w	100 Hz
Sample rate	F_s	16000 Hz
Snippet duration	T	2 s
Number of harmonics[3]	N_h	64
Number of noise bands	N_n	65

Table 1: Choice of parameters

domain. We therefore obtain a tensor $\tilde{b}_\omega(N)$ and we multiply it by $W_N(\omega)$ obtaining $H_N(\omega) = \tilde{b}_\omega(N) \cdot W_N(\omega)$. This result is sent back to the time domain, which results in $h_N(m)$, which is then itself reshaped to have the noise part of the signal, called $s_n(n)$.
The final synthesis sound $s(n)$ is obtained by adding these two signals, which gives us:

$$s(n) = s_h(n) + s_n(n) \ .$$

2.3 Inharmonic synthesis

While experimenting with the DDSP architecture and the heuristic of integrating our understanding of the physical synthesis into the structure of the model, we naturally wanted to apply this to the inharmonic sounds. Most string instruments and, of course, the piano, are inharmonic.
We therefore applied the following acoustic model of the inharmonicity of the piano, taken from [FR13] :

$$f_n = n f_0 \sqrt{1 + Bn^2}$$

where B is a coefficient that depends on the physical characteristics of the played string and which could be learned by the neural network.

Thus, B only depends on the frequency of the sound that is played. For the implementation, we computed B as the $(n+1)$th harmonic, extracted it for the synthesis and applied the given formula. It is therefore calculated by the network, as deeply as the other harmonics or the noise filter.
The experience showed that the runtime drastically increased for the back-propagation and that it is not affordable for our computing power. Our hypothesis is that this is not caused by the square root or power chain-rule computation, but more by the added complexity of relationships between the outputs of the model as B influences every frequency and therefore every amplitude and, eventually, the noise filter. We did not foresee this, it is a fundamental constraint on the DDSP heuristic and will be discussed in section 4.3. For future work, we consider to take the inharmonicity factor B directly from the output of an additional MLP whose inputs are the f_0 of the input. While this does not tackle the underlying problem, it could reduce the backpropagation time.
For the testing of this approach we recorded and used our own monophonic piano dataset which is presented in section 3.4.

2.4 Reverberation synthesis

Amongst the audio features that a simple harmonic plus noise model cannot reproduce, the acoustics of the room are of upmost importance to reproduce a recorded instrument. It can come from a real room or being virtually added, through a plugin or an external module. Thus, in [aut20], the authors suggest to learn an impulse response

[3] In the original article, they take 100 harmonics. However, since our data set consists of saxophone and violin sounds with all fundamental frequencies over $125\text{Hz} = (\frac{F_s}{2}/64)$ Hz, then the prevent of aliasing makes useless considering more than 64 harmonics. This way, we gain in speed and in memory while training.

as well as the synthesis. This impulse response is convolved with the synthesis sound before computation loss.
To first grasp the actual role of the impulse response (IR) we have decided to implement it only as an *a priori* during the training phase: an appropriate IR is designed before training and convolved with the synthetic sound before each loss computation. This test is only made with the violin dataset, in which the room is clearly audible. We suppose that the IR will have two important impacts : filtering the sound to match the room color, and add the echo and the diffuse sound present in the room. Removing those two factors from the synthesis training should allow the synthesis parameters to be dedicated to the harmonic and noise features of the violin.

It must be noted that the convolution implemented in PyTorch, torch.conv1d, is meant for small sized kernels as in image processing. The IR here is 15696 samples long. Each epoch computation time has consequently increased from approximetaly 15 seconds to more than 70 seconds.

3 Datasets

3.1 Synthesized dataset

To evaluate the training abilities of DDSP model, we synthesized a dataset of signals with specifics characteristics regarding their harmonic profile or noise spectrum. This dataset will be described with more detail in section 4 . https://www.overleaf.com/1264826417jtnwkdvmkyrc

3.2 Saxophone

A good instrument that fits the harmonic plus noise synthesis technique is the saxophone. When recorded, the breath of the musician through the instrument is heard as well as the sound produced by small amounts of saliva present between the reed and the mouthpiece. For instance, it can be heard in the first saxophone file *sax_1.wav*. It seems likely that this can be reproduced by a filtered noise.
The saxophone dataset consists of 10 different recordings : the Ave Maria and 9 jazz standards. The time played, the evolution of the humidity level and the temperature in the room affects the sound of the saxophone. In order to minimize this change, those recordings have been done in one shot. Thus, some false notes are present in those files. However, it will not impact the training at all, they are still saxophone notes independently.
Material used for recording:

- Yamaha YAS-270 saxophone + Vandoren Optimum AL3 mouthpiece + Vandoren Classic 3 reed;
- Shure Beta 57a microphone (cardioid), close position (around 50cm from horn);
- Steinberg UR22mkII USB audio interface.

No post recording treatment.
To get rid of the room acoustics, the microphone has been positioned at close range. It was easily achieved thanks to its high rejection level.

3.3 Violin

The violin is also a great example of harmonic plus noise instrument. The partials of the sound are almost perfectly harmonic and the bow creates a very specific background noise among them. For instance, it can be heard in the first violin file *violin_1.wav*. The violin dataset consists of several recordings including *Once upon a December* from Disney's Anastasia film, a Mozart concerto and Bach Sonatas & Partitas. We recorded the violin in a room who's acoustic we model thanks to an impulse response. This impulse response was created by means of the deconvolution of a chirp recorded inside.
Material used for recording:

- Antonio Monzino violin from Milan, 18th century;

- Rode NT1 microphone (cardioid), medium range (around 1m from the instrument);

No post recording treatment.

3.4 Piano

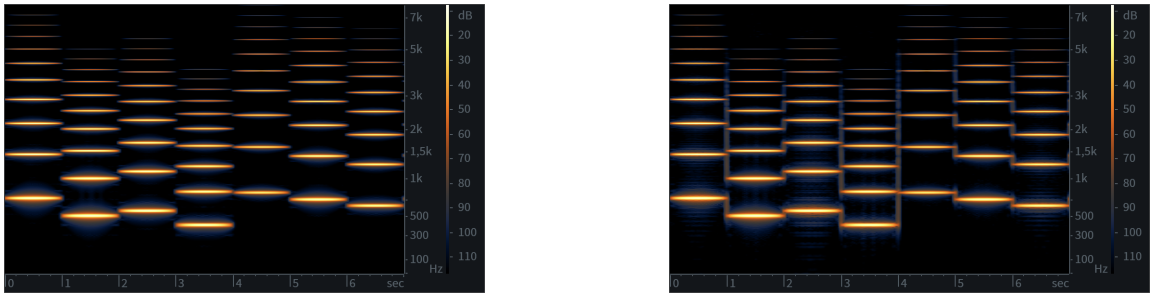
For the testing of our model of inharmonic sound synthesis as described in 2.3, we assembled our own dataset of monodic piano improvisations.
While we could not implement a model that learns the inharmonicity factor because of the computing time this task implies, we tested our network on this dataset with a constant factor $B = 10^{-5}$. The results are exposed in 4.
The sounds where recorded at 16 kHz, monophonically and can be used for various learning tasks.

4 Results

In the folder *Results*, each model trained has its own folder containing the model itself, the *parameter.py* file used for training, a log file containing each epoch computation time and loss and a sound abstract generated by this model.

4.1 Synthesized dataset

In order to evaluate the training abilities of the DDSP model, we synthesized a dataset of signals with specifics characteristics, in order to isolate what may or may not be learnt and reproduced by the model.
We generated purely harmonic signals at random f_0 with a specific harmonic distribution, to verify that the distribution was reproduced in the reconstructed signal (figure 2). We also created other signals with variations in the distribution correlated with the f_0 (as is often the case with real acoustic instruments), in order to know if the model could infer the relation between the f_0 and the harmonic distribution (figure 3). It turns out the model was not able to reproduce this relation and outputs a signal with a static harmonic profile, independent of the f_0.

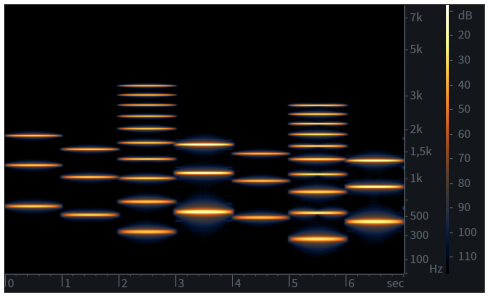


(a) Original

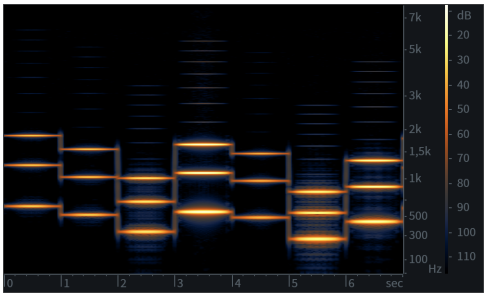
(b) Reconstruction

Figure 2: Harmonic signal of random frequencies and amplitude with constant harmonic profile

The same method was used with the noise spectrum. We generated noise signals with a constant spectral distribution (figure 4), and noise signals where the spectral distribution was correlated with the signal loudness (figure 5). Surprisingly, the network was this time successful in both cases and did infer the link between loudness and noise spectrum.
Finally, an interesting sonic property of acoustics instruments is that the damping of each harmonic increases with the harmonic rank, giving an harmonic distribution that varies with time. Trained on signals where the decay of each harmonics steepened, the DDSP model did manage to reconstruct signals with a similar decay properties, as can be seen in figure 6.
This synthesized dataset was helpful in making evident the ability of the decoder to make inferences about spectral properties of the signal, based on the input loudness. It would be interesting to expand it with signal focusing more

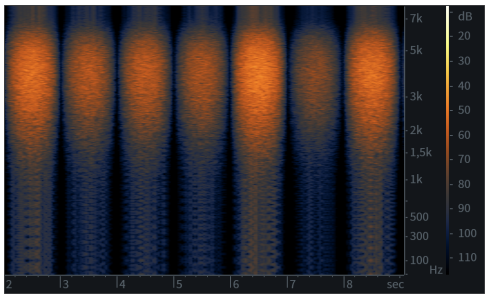


(a) Original

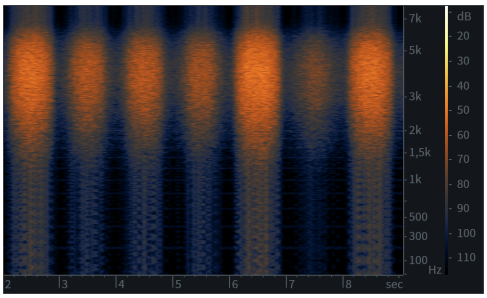


(b) Reconstruction

Figure 3: Harmonic signal of random frequencies and amplitude with time-varying harmonic profile correlated to the frequency : above a certain frequency, only the lower third of the harmonics are preserved



(a) Original



(b) Reconstruction

Figure 4: Noise signal of random amplitude with constant filtering

on the time domain, for instance on the ability of the network to fully recreate attacks despite the undersampling of the loudness.

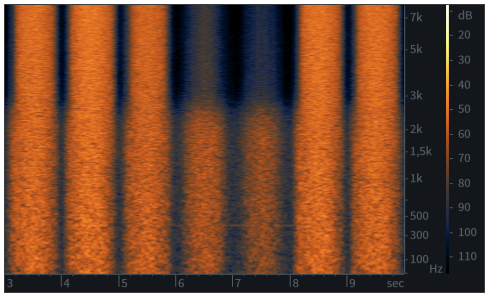
4.2 Saxophone synthesis

After training the saxophone on the net we see that it achieves reproduce both the harmonic part and the noise part. We give and example of spectrograms produced in the figure 7.
After checking the reconstruction we noticed some rays in the spectrum (see figure 8). This rays were produced by a oscillation in the loudness at a determined frequency (33 Hz). This induced an amplitude modulation that created the rays in the spectrum. This shows to us that the variation in the loudness may influence the reconstruction in a way that creates artifacts. In order to suppress them, we low-passed the loudness.

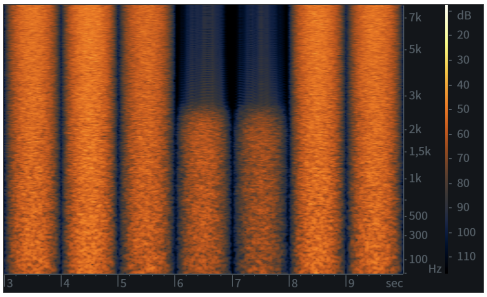
4.3 Discussion

From an acoustical standpoint, there are two phenomena that influence the harmonically produced oscillations of the system. First, the sound box of the instrument lets the frequencies resonate and overlap in time to give the recognizable sound of a violin for example. After that, the sound is diffused in the room by its reverberation. Therefore, the reverberation module of the model tries in fact to model both of these effects and will not be able to learn one without the other. Thus, the reverberation transfer described in [aut20] cannot be as precise and absolute. This yields the question of separation of the module in reverberation and internal resonance for future work.

The issue raised in section 2.3 shows that the modular architecture can find its limitations in efficiency as additional parameters can vastly increase the complexity of the backpropagation due the relations between the coefficients.
On another point, we noticed that an initial bias had to be applied on the noise level in order to make the model converge to a good reconstruction. Otherwise, the coefficients of the noise filter had trouble reaching values low

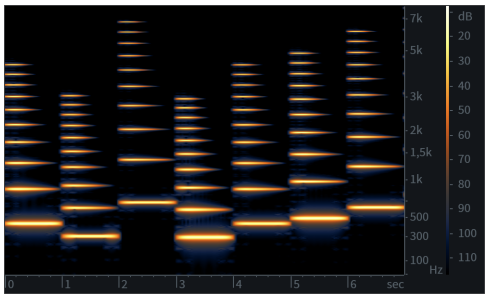


(a) Original

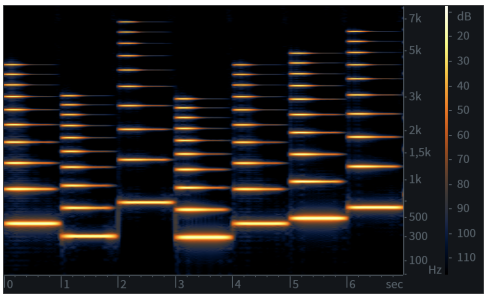


(b) Reconstruction

Figure 5: Noise signal of random amplitude with time-varying filtering depending on the frequency : under a certain amplitude, only the lower third of the filter bands is left unattenuated



(a) Original



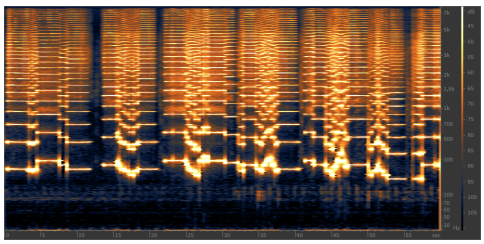
(b) Reconstruction

Figure 6: Harmonic signal of random amplitude with time-varying harmonic profile, the decay steepening with the harmonic rank

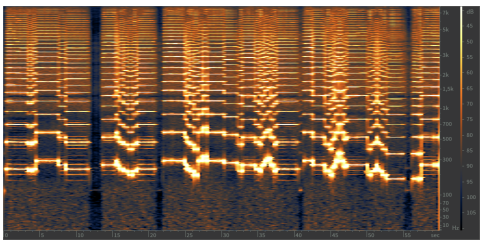
enough to properly reduce the volume, because of the sigmoid function applied to the outputs of the network. More globally speaking, the model hyperparameters such as the number of harmonics and the number of noise bands sometimes have to be carefully chosen to fit the sound being reconstructed in order to reach good results.
As regards the possibility of timbre transfer, it should be noted that in some cases the f_0 is itself a strong component of the sonic identify that will not be transfered, or that will persist : feeding the f_0 and loudness of a violin to a model trained with saxophone recordings produces sounds that are still highly reminiscent of the violin, because of the distinguishing tremolo contained in the f_0.

4.4 Conclusion

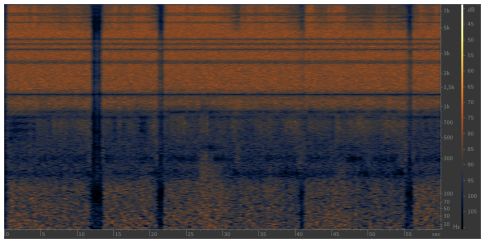
DDSP is a powerful model that, through strong priors, enables the resynthesis of instrument sound through harmonic plus noise synthesis with convincing results. After training on datasets of only a dozen of minutes we can get a very realistic reconstruction of the original audio.
This reimplementation confirmed the efficiency of the model but also showed that it was very sensitive to the adaptation of the synthesis model with the instrument chosen. Without mentioning polyphonic sources, even inharmonic instruments and instruments with long sustain can turn out to be a poor fit for the model.
We can finally mention that the values that feed the model are very expressive : the f_0 of a signal takes account of the vibrato and the loudness plays a big role in the expressivity. This makes possible a very good reconstruction, but also restricts the field of application to reconstruction of existing signals.
A way to make this model more useful would be to try to generate audio from scores. The f_0 will be the note in the score and the loudness may be the dynamics.



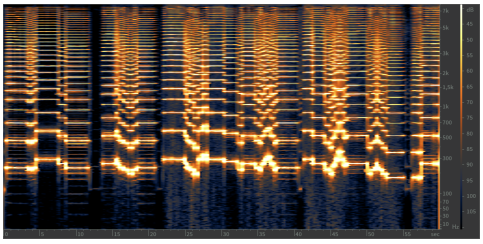
(a) Original



(b) Reconstruction

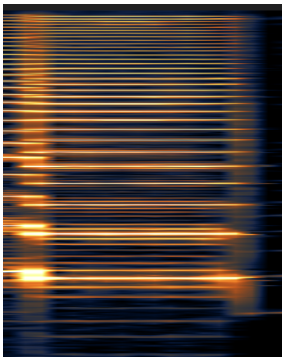


(c) Noise part

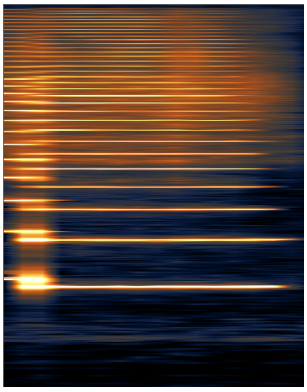


(d) Additive part

Figure 7: Saxophone spectrums



(a) Rays in the spectrum



(b) Original spectrum

Figure 8: Spectrum rays

References

[aut20] Anonymous authors. Ddsp: Differentiable digital signal processing. *Paper under double-bind review as a conference paper at ICLR 2020*, 2020.

[AvdOK16] Heiga Zen Karen Simonyan Oriol Vinyals Alex Graves Nal Kalchbrenner Andrew Senior Aaron van den Oord, Sander Dieleman and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[FR13] Laurent Daudet Francois Rigaud, Bertrand David. Piano: A parametric model and estimation techniques for the inharmonicity and tuning of the piano. *J. Acoust. Soc. Am.*, 2013.

[KSLB18] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation, 2018.

[NKK18] Karen Simonyan Seb Noury Norman Casagrande Edward Lockhart Florian Stimberg Aaron van den Oord Sander Dieleman Nal Kalchbrenner, Erich Elsen and Koray Kavukcuoglu. Efficient neural audio synthesis. *arXiv preprint arXiv:1802.08435*, 2018.

[SMB16] Ishaan Gulrajani Rithesh Kumar Shubham Jain Jose Sotelo Aaron Courville Soroush Mehri, Kundan Kumar and Yoshua Bengio. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

Appendix

A Acknowledgements and attribution

This project took place in the context of the ATIAM Master's program at the IRCAM and concludes the teaching given in the computer science module.
Our team is composed of five ATIAM students, each of them having been assigned on specific parts of the project: Gonzalo Romero developed the Noise filtering, Training and structure of the project while Olivier Birot worked on the Synthesis and Dataloader part. Victor Paredes and Lucas Chaumeny worked on the reverberation and on the inharmonic approach. André de Hillerin helped on strategy design and the Loss computation.
We wish to thank our professor Dr. Philippe Esling for his efforts to teach us machine learning for quite a while, and to address many thanks to supporting fellow experts: Antoine Caillon and Théis Bazin. They helped us grasp the main concepts of this project and understand in-depth machine learning mechanisms.

B Architecture of the DDSP Model

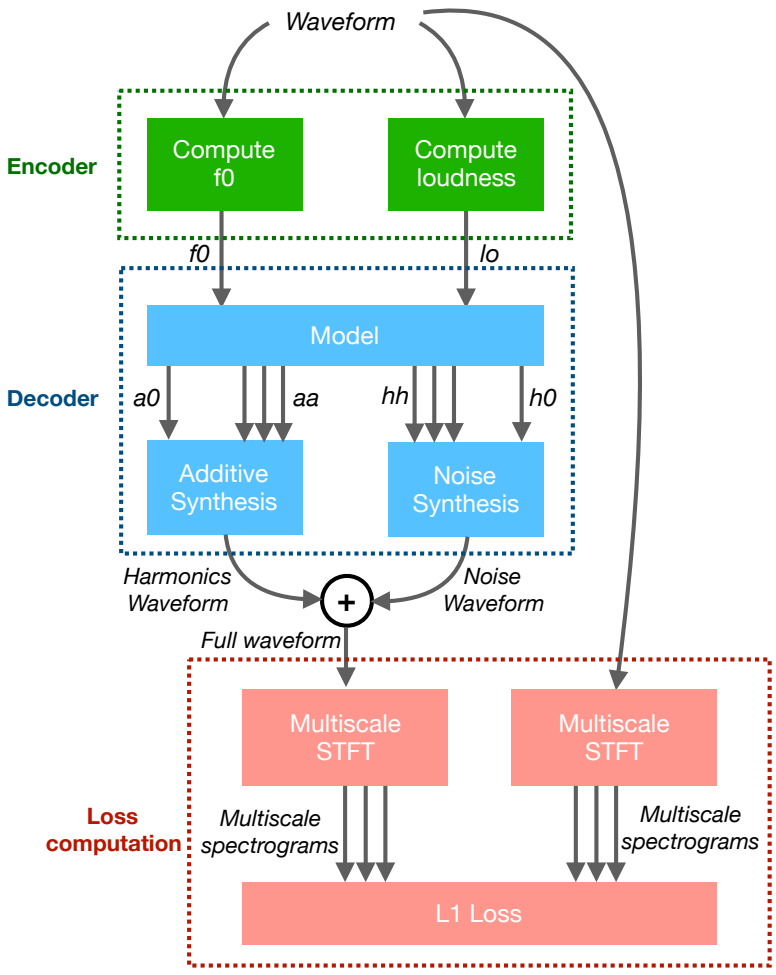


Figure 9: Full architecture of our implementation of the DDSP model