IRCAM ATIAM - Course 2018/ 2019

LATENT SPACE ORGANISATION OF SOUNDS WITH VARIATIONAL AUTOENCODERS

Caillon, Antoine
Dakeyo, Jean-Baptiste
Fouilleul, Martin
Geoffroy, Thibault

Supervised by:
Adrien Bitton
&
Philippe Esling

Contents

Introduction

This project aimed to observe the organisation of sounds encoded in the latent space of an auto-encoder, and to obtain a preferably *real-time* generative model, based on a chosen training set and a carefully designed architecture.
In this paper, we will firstly detail the preliminary work we performed prior to the designing of our architecture, then we will go on and present *Ukulelatent*, the first system we designed, and finally we will present an idea for a second system, to be implemented in the future. All the code relevant to this project is available at this address: https://github.com/nebularnoise/serge.

1 Preliminary work

1.1 Auto-encoders, Variational auto-encoders, Regularization

An auto-encoder is a type of artificial neural network where two applications named *encoder* ($\mathcal{Q}$) and *decoder* ($\mathcal{G}$) are defined as: $\mathcal{Q} : x \longrightarrow z \in \mathbb{Z}$, and $\mathcal{G} : z \in \mathbb{Z} \longrightarrow r$, with $\mathbb{Z}$ the latent space and r the reconstruction of x, with hopefully $r \approx x$.

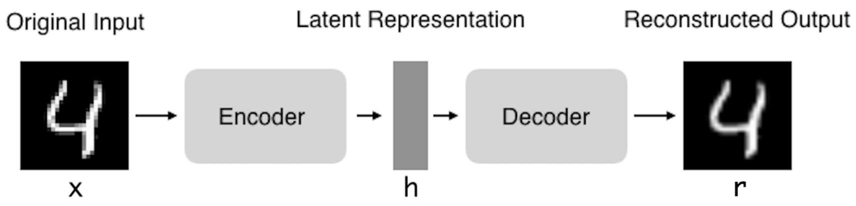


Figure 1: Architecture of a AE applied to the MNIST dataset (*towarddatascience.com*)

Classical auto-encoders can be very powerful in terms of reconstruction, at the expense of having a meaningful latent space. Since we are interested in building a generative model, the first step to consider is to incorporate a regularisation step during the training phase.

Let $\mathcal{Q} : x \longrightarrow (\mu, \sigma)$ be an application encoding an input x into a pair of latent variables (μ, σ), called a *stochastic* encoder. We can then sample the distribution $\mathcal{N}(\mu, \sigma)$ to get our latent variable h. The whole point of having a stochastic encoder is to force it to *generalize* by regularizing the latent space. Classical VAEs define a loss based on the KL divergence between the encoder's output and a prior distribution[1]. The encoder's output is then forced to look like this prior; the choice of a good prior is thus extremely important to have satisfying results.

In our case, we use a slightly different way to regularize our latent space. Instead of forcing (μ, σ) to correspond to our prior, we consider the whole encoder's output $\mathcal{Q}(\underline{x})$ given a batch of inputs $\underline{x}$, and compute the discrepancy between this output and a prior, via the *Maximum Mean Discrepancy* method [1]. This method requires us to choose between different kernels in order to estimate this discrepancy (RBF, IMQ).

[1]More detail in appendix A

1.2 Divergence Benchmark

The aim of this part is to compare the efficiency of different divergence functions, namely:

1. Kullback - Leibler Divergence (KLD)
2. Maximum Mean Discrepancy for a Radial Basis Function Kernel ($\text{MMD}_{(\text{RBF})}$)
3. Maximum Mean Discrepency for a Inverse Multi-Quadratic Kernel ($\text{MMD}_{(\text{IMQ})}$)

Each colored line displayed on Figure 2 gives the corresponding divergence between our $\mathcal{N}(0, 1)$ prior and a set of points sampled from a normal distribution, which mean is between $[0, 10]$ and is defined by the color of the curve, depending on the variance σ^2.

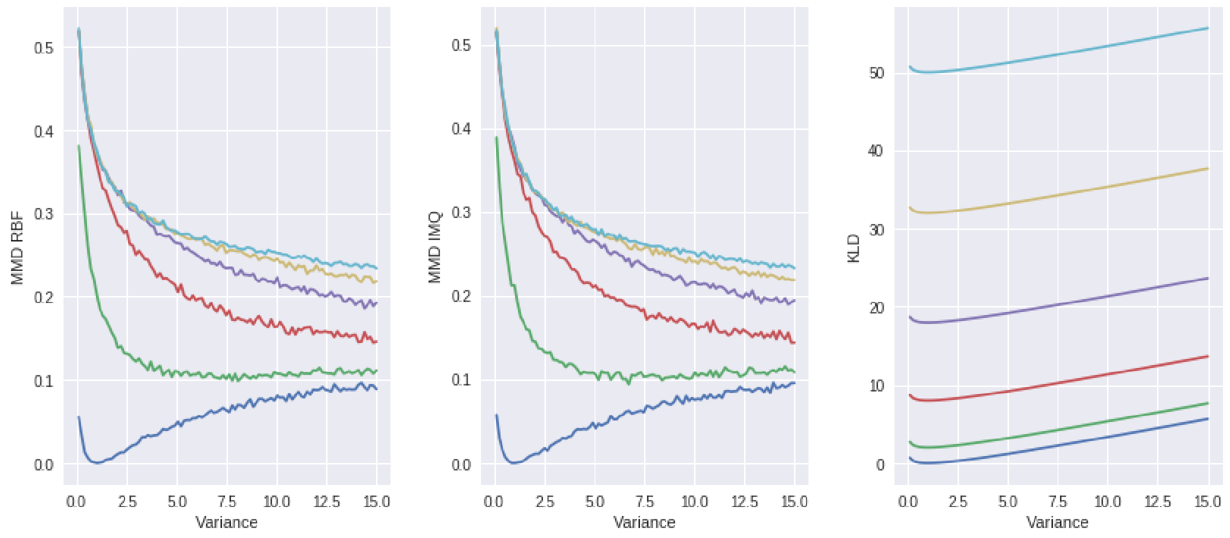


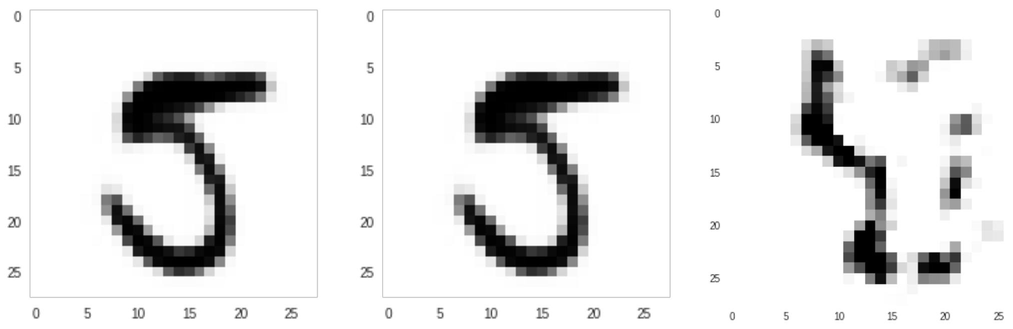
Figure 2: Benchmark of $\text{MMD}_{(\text{IMQ})}$, $\text{MMD}_{(\text{RBF})}$ and KL for a standard normal prior

All those divergences reach a global minimum for the distribution $\mathcal{N}(0, 1)$ identical to our prior, which was expected, but still reassuring. The main differences between those three are their behaviour when the variance goes up. The KL divergence, being analytic, doesn't suffer from a high variance and is still able to "measure" how far the distribution is from the prior. For both $\text{MMD}_{(\text{RBF})}$ and $\text{MMD}_{(\text{IMQ})}$, when the mean is not 0, they decrease as the variance increase, which is not what we would have liked them to do.

In practice however, the $\text{MMD}_{(\text{RBF})}$ outperforms the KL divergence in terms of organization of the latent space. While the KL divergence tends to force the encoder to produce $\mathcal{N}(0, 1)$ samples, the $\text{MMD}_{(\text{RBF})}$ allows it to yield any value it wants, *as long as* the continuous mixture of latent variables looks like the prior.

1.3 Toward a MNIST VAE

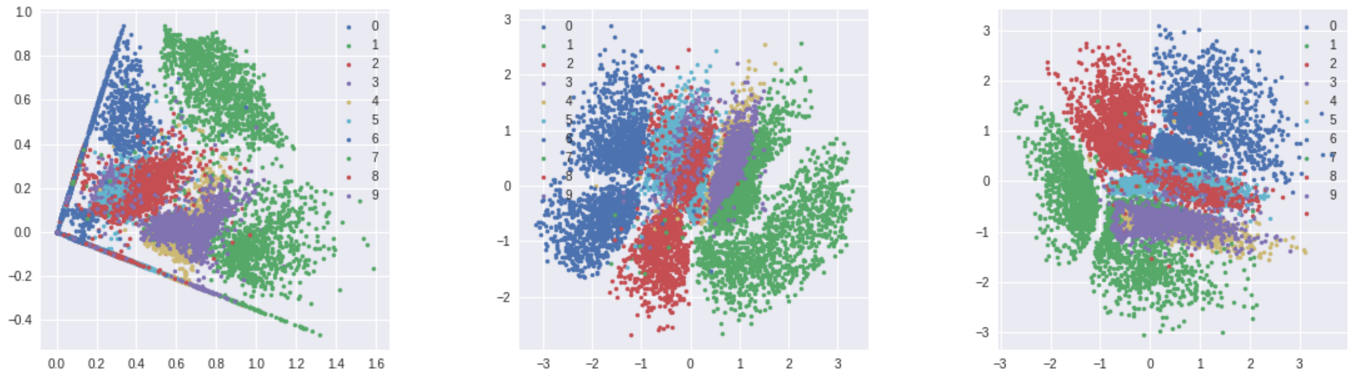
In order to get used to the AE, VAE and WAE, we played with the **MNIST** dataset, which is made of 60000 28x28 pixels handwritten digits. We first implemented a non-regularized auto-encoder and trained it with an Adam optimizer on this dataset. The encoding / decoding pass of a sample from the train set produced good reconstructions, but a small variation in latent space coordinates produces disastrous results.



(a) Sample from the train set (b) Reconstruction (c) Reconstruction with a small variation of the latent point

Figure 3: Reconstruction on a non regularized auto-encoder

We then implemented both a variational auto-encoder(VAE) and Wasserstein auto-encoder(WAE), which yielded much better latent space organization (see Figure 4). While the VAE's reconstructions were much blurrier than the AE's, the WAE's latent space had a good structure, and still produced satisfying results.



(a) AE latent space (b) VAE latent space (c) WAE latent space

Figure 4: 2D latent space structural differences for an AE, VAE and WAE trained on the same MNIST dataset

2 Ukulelatent: a latent ukulele sampler

In this project we use regularized auto-encoders in order to reconstruct audio spectrograms from a latent space. Given a spectrogram S, we are learning two applications $\mathcal{Q}$ and $\mathcal{G}$ such as

$$S \approx \mathcal{G}((Q(S))). \tag{1}$$

The final aim is to study how the latent space is organized, hence we need a way to actually explore and *hear* this space. We decided to achieve this by embedding a pre-trained **PyTorch** model inside a **PureData** external, connected to a MIDI controller.

This presents some challenges: we need to make our sampler *real-time*, find a way to get audio from mel-spectrograms, and design a way to *see* and *control* our position inside the latent space intuitively.

2.1 Dataset

In order to get audio files to work on, we recorded about ≈ 500 plucked strings sounds from two instruments: a *motu*[2] and a classical guitar. Every sound is normalized and cropped to 34560 samples. The only two things left to compute are the **mel-spectrogram** and an estimation of the fundamental frequency of each audio files. Each mel-spectrogram's amplitude is brought between -1 and 1, log scale.

2.2 Model

We define $\mathcal{Q}$ and $\mathcal{G}$ as encoder and decoder respectively. Both are using several convolutive and fully-connected layers, as long as a some regularization applied to the latent space.

Architecture

The following architecture is implemented using the `PyTorch` framework.

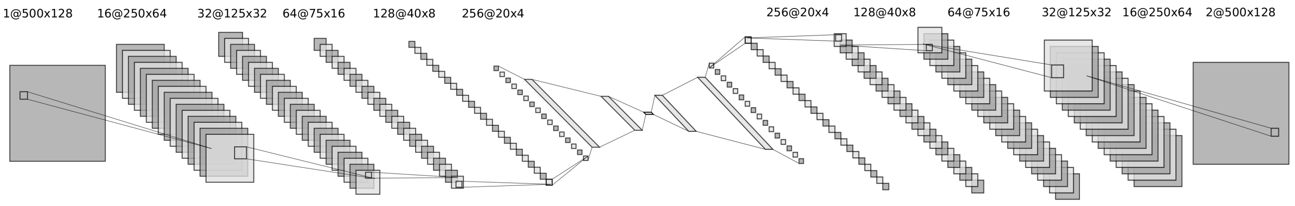


Figure 5: Architecture of the WAE used for the Ukulelatent

This architecture was subject to many modifications during the course of this project. The one presented in Figure 5 gave satisfying results reconstruction-wise. Both encoders and decoders are stochastic (they both yield a mean and a variance from which we can sample a value), and a regularization is applied to the latent space (see **Losses**).

[2]A traditional instrument from the Oleron Island, akin to a small ukulele

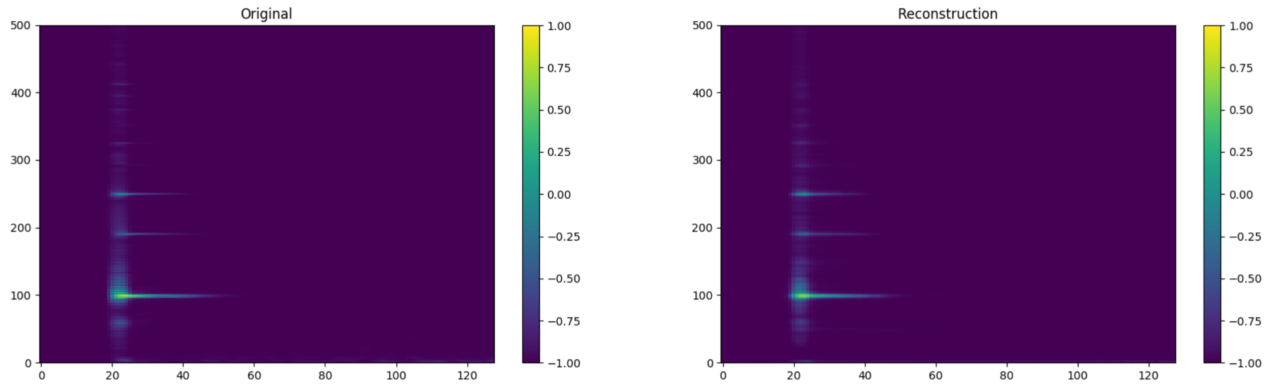


Figure 6: Comparison between a sample from the test set and its reconstruction from the WAE

At first we used deconvolutive layers in the decoder, to invert the convolutive layers of the encoder. We managed to make some reconstructions, but had problems with the well-known checkerboard effect, so we replaced them with upsample - convolutive layers as described in [7], resulting in smoother results (see figure 6).

Losses

We define the following WAE objective in equation (2) :

$$\mathcal{L}_{\text{WAE}}(x, \hat{x}, z) = \text{MSE}(x, \hat{x}) + \alpha.\text{MMD}_{\text{RBF}}(z), \tag{2}$$

where $x, \hat{x}$ are the original and reconstructed mel-spectograms, $z = \mathcal{Q}(x)$, MSE the mean square error function, and MMD_{RBF} an estimation of the maximum mean discrepancy between z and a normal distribution using a standard normal kernel ([8], [1]). We then train our model using an Adam optimizer, with an initial learning rate of 10^{-5} which we divide by two every 100 epoch.

Signal reconstruction

We first tried to use an Multi-Scale Convolutional Neural Networks (MCNN) to reconstruct the signal from the magnitude spectrogram produced by our model. After having acknowledged the fact that we had not enough training data to produce good reconstruction, especially on data outside its train set, we settled for the classical Griffin-Lim algorithm[3].

Supervising

While the latent space may be well organized, it is still unclear how to properly explore it. In order to finally get a sampler that we can play, we need to make the model map the audio samples to a certain midi note number. We first implemented a function estimating the fundamental frequency f_0 of a signal, following those steps:

1. Compute the biased auto-correlation estimator R_{xx} of the input signal x
2. Define a frequency range $[f_{\min}, f_{\max}]$ in which f_0 is included
3. Compute the corresponding period range $[\tau_{\min} = f_e/f_{\max}, \tau_{\max} = f_e/f_{\min}]$
4. Compute $\tau_0 = \text{argmax}(R_{xx}[\tau_{\min} : \tau_{\max}]) + \tau_{\min}$
5. Yield $f_0 = f_e/\tau_0$.

This fundamental frequency method is quite cheap to compute and robust in term of identification, thus it is added to the audio dataset pre-processing pass.

Next, we need to tell the model what is the fundamental frequency of the audio sample, which we tried to do in many different ways.
At first we simply mapped the frequency to an extra dimension on both the encoder and decoder, hoping the system would see that there was a correlation between a vertical shift in the mel spectrogram and the given parameter. It failed by bypassing completely our parameter.
We then tried to manually shift the input mel-spectrogram according to its estimated fundamental frequency. The encoder would therefore only ever encode sounds of the exact same fundamental frequency. On the decoder side, the system is expected to reconstruct the unshifted spectrogram by using our MIDI-note parameter. This was a failure too, as the model would just fail to reconstruct the spectrograms.

Finally, we tried to pass f_0 as two one-hot vectors, one of size 7 (giving the octave number), and the other of size 12 (giving the semitone number), and it worked! The model was learning to map data accordingly to those two vectors, thus giving us the ability to choose the spectrogram's pitch we wanted to reconstruct.

What is now interesting is to guess the purpose of the latent space's other dimensions given by the encoder, now that the pitch related information is manually encoded into two one-hots. We fixed the encoder's output dimension to 4, as we could explore it with two XY pads. By playing around with a midi keyboard, we estimated that mainly three hyper parameters are encoded in this 4-dimensional space:

- Amount of low frequency in the spectrum
- Damping
- Metallic-ish sounding instrument.

This intuitively sounds like an accurate way to organize the dataset, as those are parts of the main differences between a motu and a guitar. Maybe a 4-dimensional space is overscaled for this dataset, and should be upscaled for more complex ones. Using the T-SNE algorithm, we are able to *unfold* the latent space in order to visualise the inner separation between samples (see fgure 7).

2.3 Puredata external

In order to allow the real-time exploration of the latent space of our pre-trained model, we embedded our model inside a `Pd` external called `vae_sampler~`. It can then be integrated within a patch with an adapted GUI and MIDI control.

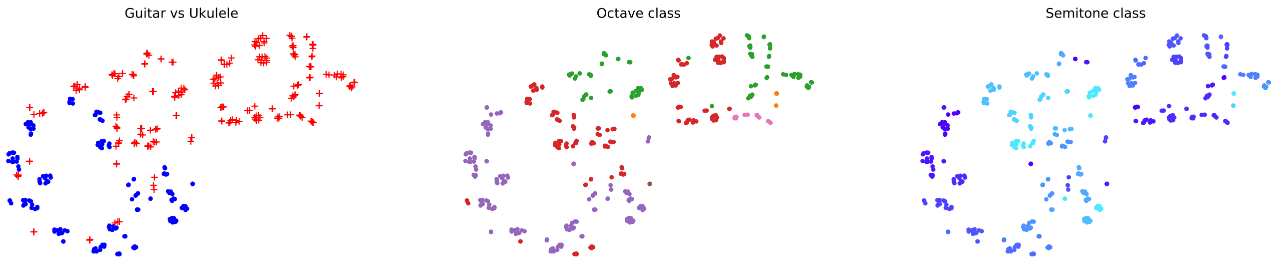


Figure 7: 2D representation of the latent space via the T-SNE algorithm

The external has one inlet that can receive two messages :

- When receiving a `load` message with a file name as argument, the external loads the model from a file.
- When receiving a `play` message with one int (corresponding to a MIDI note) and four floats (corresponding to coordinates in the latent space) as arguments, the external uses the loaded model to generate a spectrogram. It then reconstructs an audio signal from this spectrogram and outputs this signal from its audio outlet.

2.3.1 Wrapping `libtorch`

Using a `PyTorch` model from `C++` is made possible by leveraging the `LibTorch` API. The `PyTorch` model is first converted to `TorchScript` by tracing its `forward()` function with example data from the train-set, and serialized to a file. This file can then be loaded by a `C++` program linked to the `LibTorch` library.
The `vae_sampler~` external is coded in plain `C`, whereas `LibTorch` exposes a `C++11` API. Hence in order to use it in our external, we wrote a wrapper that is responsible for loading the model, calling its `forward()` function and converting back and forth `LibTorch` tensor types.

2.3.2 Real-time signal reconstruction

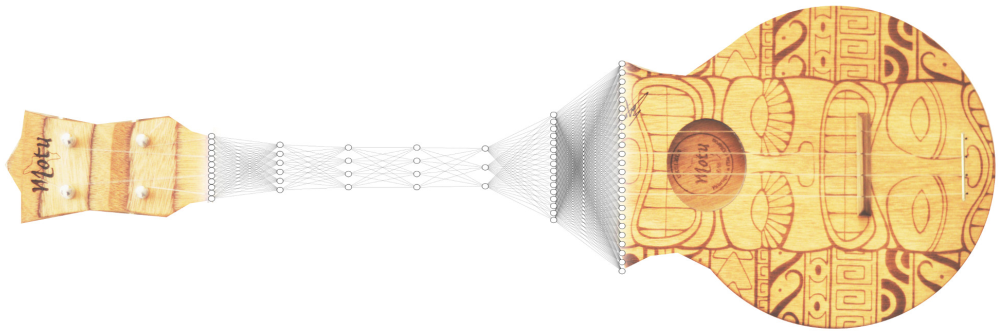
The external has a number of polyphony voices each associated with a worker thread and an audio buffer. When requested to play a note with a given set of latent space coordinates, the external allocates a voice of polyphony and fires up its worker thread. This thread calls into our wrapper's functions to retrieve a spectrogram. It then proceeds to reconstruct the signal from the spectrogram and stream it to the voice's audio buffer, staying ahead of `Pd`'s audio callback.
In order to reconstruct the signal inside the external with sufficient performance, we re-implemented the Griffin-Lim algorithm in `C`. Our implementation is based on the state-of-the-art `FFTW` library for fast discrete Fourier transforms. With compiler optimizations on, on our test machine, our implementation of the Griffin-Lim algorithm took 0.7s to reconstruct a 1s sample with 100 iterations. In order to eliminate most of this latency, our reconstruction function divides the spectrogram into batches of 32 DFT slices, streams the batches into the Griffin-Lim algorithm and reconstructs the signal step-by-step. We were able to reach a real time throughput using this technique.

The output buffers of each voice are then read and combined by the `perform()` routine of the external, and finally sent through the signal outlet of the external.

2.3.3 Handling the model's latency

The model itself took 0.8s to compute a complete spectrogram on our test machine without `CUDA` support. We tried to build a new model which could stream the spectrogram in small batches, but we found it was much more difficult to train, probably due to the loss of internal coherence of those small batches compared to a full spectrogram. Thus the model imposes an initial latency on the audio generation.

We then wrote a wrapper around our `C` Griffin-Lim implementation, and used it inside a pure `Python` audio engine with `CUDA` support (see `explore_space.py`). On a machine with an nvidia MX130 GPU, we were able to reconstruct a full mel-spectrogram in less than 60ms. A demo video is available at https://youtu.be/yjcfi6fvs3o. `CUDA` support in the `PureData` external is still to be improved.



3 Latent audio effect organisation on synth samples : FX-Coast

One objective of this project was to explore the possibilities of latent space organization of sounds processed by effects. This would require to train the model with a dataset, augmented by DSP routines. We decided to try and have a bigger variety of sounds in the core dataset than in the previous system.

3.1 Dataset

For this new exploration of latent space organisation of sounds, we first developed a new core dataset. We recorded more than 2000 synth hits made with a modular system, and a Make Noise 0-coast. After developing the core-dataset, the idea was to augment it by processing it with DSP routines. We needed to be able to process our dataset in batches, so the ideal solution was one-liner commands. After attempting to use CSound, we decided to use Faust, out of familiarity with its ecosystem.

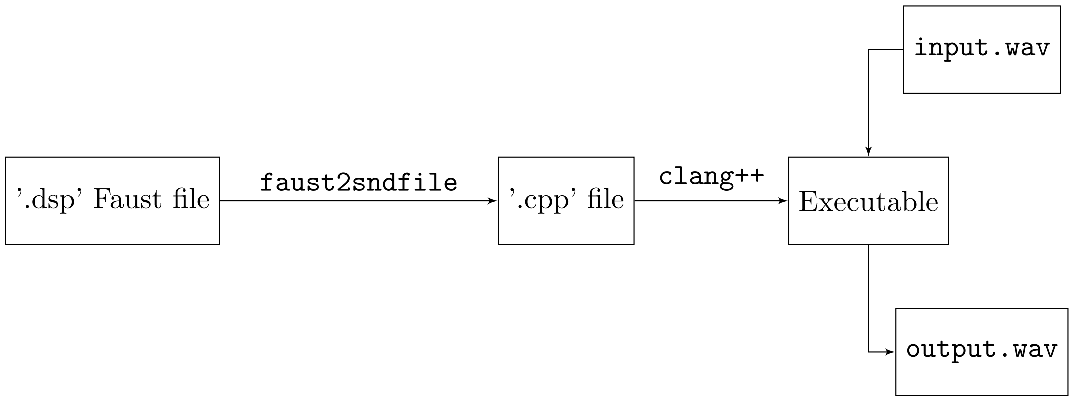


Figure 8: Principle of the dataset augmentation pipeline

We needed to keep the time response relatively short for the model to be able to process the dataset. This implied that long delays and reverb sounds were out of the question, so the effects are mainly filtering and distortion effects.
After processing the dataset, we had multiplied the size of our dataset by a factor of 72, which is to say the amount of DSP routines we developed.

3.2 The issue of time

The idea was to reuse the architecture of the ukulelatent network, and process the FX-Coast dataset. However, due to the sheer size of the resulting FX-Coast Dataset ($\approx$ 120 GiB), one epoch would have likely taken several hours. Given that the deadline for the project was only less than a week away, we gave up on this endeavour.

Conclusion

Using a WAE with multiple convolutional and fully connected layers, we were able to properly reconstruct and generate mel-spectrograms, after constraining our latent space to a normal standard prior with the MMD_{RBF} regularisation. Supervision using one-hot vectors succeeded in shifting the spectrograms in frequency. After wrapping the **PyTorch** model in a **PureData** external, we managed to reconstruct audio in real-time using an efficient C implementation of the Griffin-Lim algorithm, and CUDA support in the external.
The resulting sampler gave satisfying results on a plucked string dataset, which was developed for the occasion.
In an effort to diversify the dataset and explore the organisation of audio effects in the latent space, we set out to develop a new system, with a larger dataset, augmented by the use of DSP routines. While the groundwork was performed, the training had to be shut down due to a lack of time. Further work could allow for a second real-time sampler using this new DSP-augmented dataset.

References

[1] Kernel functions for machine learning applications. http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/#gaussian.

[2] Cyran Aouameur. Learning latent spaces for real-time synthesis of audio waveforms. 2017.

[3] D. W. Griffin and J. S. Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on acoustics, speech, and signal processing*, ASSP-32(2):236–243, 1984.

[4] W. K Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[5] S. Kullback. *Information Theory and Statistics*. John Wiley & Sons, Inc, 1959.

[6] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[7] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[8] Alexander J. Smola. Maximum mean discrepancy. *National ICT Australia*, 2006.

A Divergences

A.1 Kullback-Leibler Divergence

The Kullback–Leibler divergence (KLD) measure the difference between two probability distribution ([6], [5]). The more two distributions are similar, the closer the KLD will be to 0, and the further the distributions are, the more the divergence increases. The KLD is defined as follows :

$$D_{KL}(P\,||\,Q) = \sum_i P(i) \ln \left(\frac{P(i)}{Q(i)} \right), \tag{3}$$

with P and Q two probability distributions of a discrete random variable. We can also see it like the average of the logarithmic difference between P and Q.
However, the KLD is only defined if $P(i) > 0$, $Q(i) > 0$ for any i, and if P and Q both sum to 1. A simplification in the formula is possible in the case of multivariate distributions. Suppose that we have two multivariate normal distributions, with means μ_p , μ_q and with (non-singular) covariance matrices Σ_p and Σ_q. If the two distributions have the same dimension, k, then the KLD is :

$$D_{\mathrm{KL}}(\mathcal{N}_p \parallel \mathcal{N}_q) = \frac{1}{2} \left(\operatorname{tr}\left(\Sigma_q^{-1}\Sigma_p\right) + (\mu_q - \mu_p)^{\mathsf{T}} \Sigma_q^{-1} (\mu_q - \mu_p) - k + \ln \left(\frac{\det \Sigma_p}{\det \Sigma_q} \right) \right). \tag{4}$$

In our case, we consider a dependence to z the latent space, from P and Q. We define $p(z)$ a normal prior and $q(z)$ another normal distribution. The distribution p(z) will be later sampled to generate some data, the easiest choice would be to choose $p(z) \sim \mathcal{N}(0, 1)$. It is possible to write all the latent variables mutually independent and each parametized by a distinct variational parameter [2]:

$$q(z) = \prod_{j-1}^{m} q_j(z_j). \tag{5}$$

Here, each latent space z_j follows an independent variational factor, defined by $q_j(z_j)$. We have made the choice to use normal factors. Thus each dimension of the latent space will be governed by an independent normal distribution with its own mean and variance, and in the VAE this is especially $q_j(z_j) = \mathcal{N}(\mu_j(\mathbf{x}), \Sigma_j(\mathbf{x}))$, where the parameters depend on the input data. The KLD can be computed for our VAE like :

$$D_{\mathrm{KL}}\left[P\,||\,Q\right] = D_{\mathrm{KL}}\left[\mathcal{N}(0,1)\,||\,\mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x}))\right]. \tag{6}$$

If the two distributions have the same dimension, k, then the KLD between them is as follows :

$$D_{\mathrm{KL}}\left[\mathcal{N}(0,1)\,||\,\mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x}))\right] = \frac{1}{2} \sum_k \left(\Sigma_q(x_k) + \mu_q^2(x_k) - \ln(\Sigma_q(x_k)) - 1 \right), \tag{7}$$

where $\Sigma_q(x_k)$ and $\mu_q(x_k)$ are respectively the variance and the mean of the k^{th} dimension of Q. This is the formula we used to measure the KLD in our VAE.

A.2 Maximum Mean Discrepancy

To set the Hilbert Space, we consider a reproducing Kernel Hilbert Space $\mathcal{H}$ with kernel k [8]. To evaluate the function f depending to the kernel k, we define : $f(x) = \langle k(x,.), f\rangle$.
And computing means via linearity gives :

$$\begin{aligned} E_p[f(x)] &= E_p[\langle k(x,.), f\rangle] && (8)\\ &= \langle E_p[k(x,.)], f\rangle, && (9) \end{aligned}$$

where $Ep[f(x)]$ is the expected value of the function f. To define the expected value, we use the Monte-Carlo method. It consist on generate samples randomly from a probability distribution [4]. It is possible to represent the optimization problem as follows :

$$\begin{aligned} sup_{||f||\leq 1} E_p[f(x)] - E_q[f(y)] &= sup_{||f||\leq 1}\langle \mu_p - \mu_q, f\rangle && (10)\\ &= ||\mu_q - \mu_q||_{\mathcal{H}}. && (11) \end{aligned}$$

And if the expression is written in terms of expected values, we get :

$$\begin{aligned} ||\mu_p - \mu_q||_H^2 &= \langle \mu_p - \mu_q, \mu_p - \mu_q\rangle && (12)\\ &= E_{p,p}k(x,x') + E_{q,q}k(y,y') - 2E_{p,q}k(x,x'). && (13) \end{aligned}$$

The variable $E_{i,j}$ is the expected value between two distribution i and j. The equation 13 express how to calculate the Maximum Mean Discrepancy.
For a Radial Basic Function (RBF), or a Inverse Multi-Quadratic kernel (IMQ) [1], the kernel k for two sets of real values x and y is equals to :

$$\text{MMD}_{(\text{RBF})}: \quad k_{(\text{RBF})} = \exp\left(\frac{||x^2+y^2||}{-2\Sigma^2}\right) \quad ; \quad \text{MMD}_{(\text{IMQ})}: \quad k_{(\text{IMQ})} = \frac{1}{\sqrt{||x^2+y^2|| + c^2}}. \tag{14}$$

where c is an adjustable constant.

B Repository organization

The github repository has the following structure:

```
/
├── README.md
├── faust
│   └── *.dsp ............................................ Thibault Geoffroy
├── faust_cpp
├── notebooks
│   ├── MMDs.py ............................................. Adrien Bitton
│   ├── compa DIV.ipynb ....................... JB Dakeyo / Antoine Caillon
│   ├── *.ipynb ............................................ Antoine Caillon
│   ├── VAE_MNIST.py ....................................... Antoine Caillon
│   ├── Dataset_processor.ipynb ........................... Thibault Geoffroy
│   └── refactor_WAE.ipynb ............... Thibault Geoffroy / Antoine Caillon
├── pd-external
│   ├── README.txt ......................................... Martin Fouilleul
│   ├── build .............................................. Martin Fouilleul
│   ├── patches
│   │   ├── *.pd ........................................... Martin Fouilleul
│   │   ├── visu_latent.pd ................ Antoine Caillon / Martin Fouilleul
│   │   ├── keyboard.mmb.pd ...................... puredata.info/downloads/mmb
│   │   └── xyslider.pd ...................... forum.pdpatchrepo.info/user/liamg
│   └── src
│       ├── m_pd.h ............................................ Miller Puckette
│       └── *.c, *.cpp, *.h .................................. Martin Fouilleul
├── pyglim
│   ├── README.txt ......................................... Martin Fouilleul
│   ├── build .............................................. Martin Fouilleul
│   ├── *.py ............................................... Martin Fouilleul
│   └── src
│       └── *.cpp ............................................ Martin Fouilleul
└── src
    ├── train
    │   └── *.py ........................ Antoine Caillon / Thibault Geoffroy
    ├── util
    │   ├── MCNN.py .......................................... Adrien Bitton
    │   ├── audio utilities.py ............................... github/bkvogel
    │   └── *.py ............................................. Martin Fouilleul
    └── *.py, *.ipynb ........................................ Antoine Caillon
```

The **faust** directory contains the .dsp code describing the effects used to augment the second dataset.

The **notebooks** directory is where the first AE - VAE - WAE experiments were made. Inside **pd-external** is the source code for compiling the **PureData** external **vae_sampler**, alongside some test patches. The **src** directory contains **final_model_log_scale.py** which is the Python script that has been extensively used to train a WAE on spectrograms from our plucked strings dataset[3]. The **explore_space.py** loads a pretrained model and make you listen to it interactively.

The trained model used in the demo video is available here WVAE_500.torchscript.

[3]Note that this dataset is available under `/u/atiam.1819/caillon/dev/serge/dataset/string.tar.gz`.