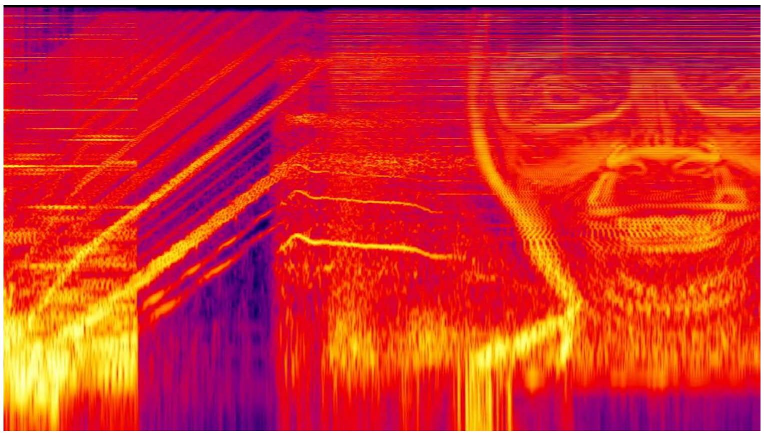
PITCH SHIFTING : A MACHINE LEARNING METHOD FOR MUSIC TRANSFORMATION

CHARLES BRAZIER
SUPERVISED BY HENDRIK PURWINS



Sound and Music Computing Group
Aalborg University Copenhagen
Copenhagen

August 19, 2017

ABSTRACT

In music, sound transformations can be done thanks to signal processing algorithms. However these handcrafted methods are very dependent on their input. In a more general context, Machine Learning algorithms propose to learn abstract mathematical functions which reproduce a sound transformation. Autoregressive models have already shown their efficiency in the field of image generation and are good candidates to automate sound transformations thanks to a simple and stable training process. These end-to-end algorithms allow to take raw audio as input and to give raw audio as output. Among them Wavenet and SampleRNN are two good candidates for this problematic. One of them, Wavenet, has recently been introduced as a very successful methods in multitude of application domains : text-to-speech dependent on individual fo curves and speaker characteristics, speech recognition, speech enhancement, and music generation. It has the particularity to be able to be conditioned by a global bias which can represent a pitch shifting interval. Focused on this network, a relevant training dataset is built step by step, from sine waves to real sounds, to produce a pitch shifting transformation. Showing difficulties to keep long term dependencies between samples due to a small receptive field, a solution using an autoencoder is presented.

RÉSUMÉ

En musique, les transformations sonores peuvent être réalisées au moyen d'algorithmes de traitement de signal. Cependant ces techniques artisanales sont trop dépendantes de leur entrée. Dans un contexte plus général, des algorithmes de Machine Learning proposent d'apprendre des fonctions mathématiques abstraites qui reproduisent une transformation sonore. Les modèles autorégressifs ont déjà montré leur efficacité dans le domaine de la génération d'image et sont de bons candidats pour automatiser des transformations sonores grâce à un procédé d'entrainement simple mais surtout stable. Ces algorithmes appelés bout-à-bout permettent de recevoir directement de l'audio en entrée et de générer de l'audio. Parmi eux, les réseaux Wavenet et SampleRNN font de bons candidats pour cette problématique. L'un d'entre eux, Wavenet, a été introduit récemment comme être une méthode brillante dans de nombreux domaines : la génération de voix à partir d'un texte en fonction de la fréquence fondamentale et des caractéristiques du locuteur, la reconnaissance vocale, la génération de voix de qualité et la génération de musique. En se concentrant particulièrement sur ce réseau, un dataset d'entraînement pertinent est construit étape par étape, à partir de sinusoïdes jusqu'à des sons réels, dans le but de produire des transformations de changement de hauteur. Faisant apparaître des difficultées à conserver des dépendances à long terme entre les échantillons à cause d'un champ réceptif trop faible, une solution utilisant un autoencodeur est présenté.

KEYWORDS

End-to-end algorithm, Machine Learning, Music transformation, Pitch shifting, Wavenet.

ACKNOWLEDGMENTS

I want to thank Hendrik Purwins, my supervisor, and Stefania Serafin for making me welcome in the Sound and Music Computing team. They contributed to the good run of my internship.

I want also thank Andrea, Alex, Jose Luis, Jorge and Mattia for these great discussions about work and other things at the university. I thank warmly Lui for the same interest for music and for his MPC he lends me during this six months. Thank you also to discover me Denmark and its culture. Finally I want to thank Yassine to be with me as a worker at the university during all the summer.

CONTENTS

1 INTRODUCTION

This work is the fruit of six months of working at Aalborg University of Copenhagen. It represents the end-of-year internship of the MSc ATIAM.

The goal of this internship is, by exploring Machine Learning methods, to be able to automate sound transformations and specially pitch-shifting. Thus the network has to learn the mathematical function allowing to have an overall point of view.

Firstly, a state of the art will be exposed to present different existing end-to-end algorithms which can be interesting to use for this problematic. Then we will modify the Wavenet network to be able to learn a pitch-shifting transformation. Finally we will show different approaches to built a strong training dataset and we will expose a recent method, NSynth, which could solve several difficulties.

Part I

STATE OF ART

2 STATE OF ART

2.1 PRESENTATION OF THE HOST ORGANIZATION

The Aalborg University (AAU) of Copenhagen is one of the three universities in the AAU group with Aalborg and Esbjerg. It hosts 3.300 students into fourteen different research departments, proposing 9 bachelor's and 24 master's programs. Among these departments, there is the Department of Architecture, Design, and Media technology where I was into but there are many other fields such as building research, business and management, chemistry and bioscience, clinical medicine, communication and psychology, culture and global studies, development and planning, electronic systems, learning and philosophy, materials and production, mathematical sciences, political science, and sociology and social work.

The Department of Architecture, Design, and Media technology holds 140 researchers and PhD students and participates in a long range of national and international projects. At Media Technology, projects are focused on a mix between technology and creativity, trying to understand how these technologies influence human perception and interaction by applying them in solving real world problems. Interactive sound and audio technology are two research fields present into this department, combining in the Sound and Music Computing team.

The Sound and Music Computing (SMC) research group addresses challenges in the current generation of audio and music technologies such as sound synthesis, analysis and processing, music perception and cognition, music expression, music informatics and intelligence. The issue of this group is to develop technologies for users and also a wide variety of industries, from entertainment to education to social services. With me, there were four other master thesis students working in their projects, two also in Machine Learning applied to music, one in sound synthesis for games and one in sound design.

2.2 STATE OF THE ART

Among the different Machine Learning subjects of other students at the university, there is a subject concerning timbre modification using Variational AutoEncoders (VAE), a subject concerning sound generation with a Generative Adversarial Network (GAN), and my subject concerning music transformation. These problematics introduce different approaches which currently dominate the field of image generation and which can be applied for music generation.

VAEs are generative models which allow to learn a representation of data which is sufficient to reproduce inputs. The network is composed of a bottleneck to reduce the dimensionality of the input to an encoding vector. Then this vector is decoded to reconstruct the input. A trained autoencoder is able to decode only vectors which were trained before. To be more general, a constraint can be added to the network which forces to generate an encoding which follows a unit gaussian distribution. The network is then a variational autoencoder. It allows to have an efficient inference with approximate latent variables. However the network is often trained with a mean square error function which consequently gives blurry results.

GANs were introduced by Ian Goodfellow et al. in 2014 and presented a new unsupervised method used in picture generation. Each GAN is composed of one network to generate candidates, the generator, and one network to evaluate them, the discriminator. His role is to progress towards a good discrimination between input datas and what is created by the generator. For the generator, the aim is to progress towards the generation of efficient false examples which can be interpreted by the discriminator as right datas. Thus the training can be seen as a game between the discriminator and the generator that aims to converge towards his Nash Equilibrium.

Concerning Wavenets, these models were announced by Google's Deepmind team in September 2016 and allow to get good audio generation results. Firstly developed for speech synthesis, this network was able to learn the architecture of a natural pronunciation to generate good speech results as a human can pronounce. Taking audio in input and giving audio in output, this end-to-end algorithm was extended to music to generate pieces of piano which have a good consistency.

Focused on music transformation with machine learning methods, Wavenets seem to be well appropriate to generate song in learning sound constructions. These networks can be conditioned with a local or a global conditioning which could permit to learn transformations.

2.2.1 *Motivation*

With actual technologies, deaf people who have a Cochlear Implant (CI) are able to recognize very well the speech thanks to a group of electrodes put in the cochlea. This device allows to transform the sound from outside into electrical impulses permitting to stimulate the cochlea and thus creating the stimulation of the audio nerve. CI users can perceive rhythm or relearn speech understanding but they cannot enjoy music, due to the fact that the musical signal is more complex and impose to have a better spectral resolution [1]. The difficulty with music is there is a lot of problems like inadequate spectral, fine-temporal, and dynamic range representation.

Music is seen as the most complex auditory stimulus in existence. The mechanism which encodes the music is composed of many band-pass filters where each extract the envelope, low frequencies correspond to the apex and high frequencies correspond to the base. To replace the role of 3500 inner hair cells, we use electrodes placed along the cochlea not by hazard but along the Greenwood's frequency-position function to find ideal positions, and where each electrode stimulates a large band of nerve fibers. When the speech spectral resolution needs four channels, the music spectral resolution needs 64 channels [2]. The first idea is to increase the number of channels with intermediate electrodes into the cochlear. But the spatial channel interaction limits this maximum number of channels. If we increase the number of channels, CI users perform as if they have only eight to ten effective channels.

CI devices have also a problem concerning dynamics which represent the variation in loudness. Dynamics are essential to share excitation, expression and meaning. Contrary to a Normal Hearing (NH) listener who has a dynamic range of 120 dB with 60 to 100 discernable steps, CI users are able to distinguish 6-30 dB with 20 discernable steps. This reduction has an impact on sound quality and thus decreases the ability to detect spectral shape differences.

If the rhythm and the tempo are detected without problem by a CI user, it exists several difficulties to faithfully adapt the pitch and the melody. Whereas NH listeners can easily detect a one semitone pitch direction change, pitch direction thresholds for CI users range anywhere from one to eight semitones or more. Concerning the melody, it is defined as a sequential series of pitches. If the melody is composed of pure tones instead of complex harmonic tones typically found in music, CI users have no difficulty to identify the melody. Indeed pure tones can activate a single electrode to provide clearer pitch contours. Otherwise they have some difficulties to perceive contour shape, interval difference, scale and this difficulty increases if chords are played.

When spectral features of music are not really well translated to a CI user, the combination of spectral and temporal features to recognize the timbre is also problematic. Timbre is a multidimensional percept that depends both on the frequency spectrum and temporal envelope of the incoming sound wave. Normal hearing listeners are able to use both envelope and fine-structure cues interchangeably to identify instruments whereas CI users rely entirely on temporal envelope cues to make instrument judgments, and are unable to use fine structure information to make any judgments of timbre. Timbre can be resumed as a three dimensional space where each dimension is correlated to temporal envelope (log-attack time), spectral envelope (spectral centroid where the amplitude weighted mean of the harmonic peaks) or spectral fine structure (difference in amplitude between adjacent harmonics). For CI users, this timbre space is represented by only two dimensions which are temporal envelope features and spectral envelope features.

Music perception in CI users can be improved with training sessions. Indeed this allows to be better in pitch contour and familiar melody identification. It also works in timbre recognition tasks. However all of these problems and without speaking about auditory nervous system limitations for deaf people, enjoying music seems to be a hard work. To get around this problem we propose a different approach to explore music by transforming it in order to make it more enjoyable and comprehensible.

2.2.2 *Learning automatically general sound transformation*

Before learning transformations applied to music, composers created them by hand to enlarge the scope of feasibility. From a compositional point of view, the musical variety can be multiply to infinity in adding general transformations into each song. Considering a music sheet, each note can be moved from a position to an other with a constant translation or they can also be transformed in changing their duration. Globally all kind of geometric similitudes can propose a new song transformed from a previous one. The Hungarian composer György Ligeti proposed natural timbral metamorphoses in different works and specially in *Cello Concerto, 1996* where the timbre goes continuously from an instrument to an other. Recently the french composer Yann Tiersen underlines a pitch shift in *Comptine d'un autre été : l'après midi* where the melody starts from an octave to one octave higher.

With the coming of electroacoustic music, the potential of sound transformation has grown enormously. All of these modifications naturally used by composers were the work of scientist to automate them. Each instrument can be modeled [3] by a representation with masses, springs and dampers which recreates the same wave propagation until a same sound radiation to hear the proper timbre of an instrument. Passing models from one to an other allows to simulate a timbre modification.

If now modifications are applied directly to the sound wave, a new sound can be expected. Signal processing algorithms are efficient in filtering and audio quality increasing but they can also be used as audio transformers such as echo addition, pitch shifting time stretching and many other applications. The modification of frequency and temporal clues can be concatenate into one tool, the phase vocoder, which has been well described by Dolson in 1986 [4]. This analysis-synthesis technique allows to take an input signal and to give an output signal which is a modified version of the input. These modifications are done in the Fourier area allowing to separate temporal and spectral information. For a time stretching transformation, the idea is to cut the input spectrogram into frames which are overlapped. This ratio of overlapping is now modified before the reconstruction to extend or shorten the input song. Complex datas do not present any problem concerning the amplitude which is preserved but concerning the phase, it linearly changes with the time. Thus frames of the desired output which are not at the same position as the input ones because of this new synthesis scale show phase jumps. Adding this phase incrementation at each frame, the reconstructed sound has a high fidelity. If the phase vocoder can be used to modify temporal clues without changing frequency clues, it is easily to understand that the contrary is conceivable. Pitching a sound by a pitch-change factor corresponds to time-stretch the sound by this factor and listen to the result with a modified sample rate. This new sample rate is equal to the previous one times the pitch-change factor.

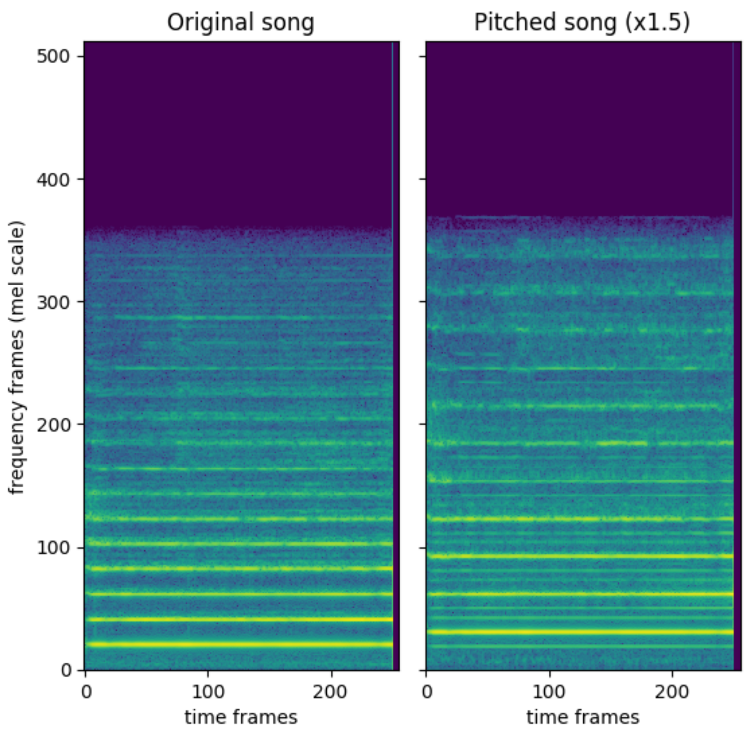


Figure 1: Pitch shifting using Dolson's phase vocoder obtained applying a short-time Fourier transform (window size of 1024 and hop length of 256) into a four second flute note.

The phase vocoder is one of the best methods which exist for the analysis and for sound transformations. Despite the artifacts which are present inside the spectrogram, a signal processing algorithm is handcrafted for each input and requires details to control the value of frequencies and their phase. Learning mathematical functions allows to have a greater abstraction and not to be dependent on inputs. A deep learning approach allows to learn implicit functions of transformation with a high level of abstraction into datas. This domain has already succeed in a lot of different topics [5] such as facial recognition, speech recognition, computer vision, natural language processing, audio recognition, social network filtering, machine translation and bioinformatics. Deep learning is actually an acknowledged method which is also applied in the music field where applications are led by two main studies which are the music information retrieval and the music signal processing.

2.2.3 *Feature extraction with long range dependencies*

Audio applications in deep learning follow previous work in speech recognition where a great interest appeared in the last decade. Since the first major study in speech recognition in 2009, improvements did not stop to progress until the first study for audio and music processing proposed in the paper of Huang et al [6]. For the first time, the aim was not to be focused on creating a single melody but more on creating music which includes harmony and melody, as if a composer created it. Their approach in performing an end-to-end LSTM algorithm with deep neural networks allows to learn and generate music taking into account global structure in the music.

To capture rich features in the frequency domain and increase the quality of music generated, the most common method presented in the literature is the use of fully connected layers, convolutional layers and LSTMs. There was a lot of work concerning a training with musical features such as notes, chords, rhythm, or other information which describes the audio. The Google brain team performed into this way, proposing their project Magenta [7]. This machine learning platform is used for generating art in general. It proposes to mix art and music, allowing to increase music generation by putting some intonation, emotion, attention or surprise into music. During the training each song is labelled with an instrument, a pitch, a velocity, and a rate to describe the sonic quality. All these metadata allow to generate music with high quality. Music has to not to be differentiated with the music created by a composer or played by a musician.

However these informations restrict drastically the field of music which can be generated. Each generated song must be matched with a vector which represents all different metadata. To remove these restrictions these different networks must learn how to generate music by taking directly raw audio. They must learn how to create a logical succession of samples to finally have a melodic and harmonic result which is pleasant to hear. The input of this training is then audio waveforms. They are one dimensional signals and composed of a set of samples where the amplitude of these samples varies with time. The variation is not hazardous but underlines a smooth transition from a sample to the next one. Thus a network which can propose music generation sample by sample must memorize samples which allow him to predict the next one with a good succession.

2.2.4 *Wavenet : an end-to-end algorithm to create music*

In end-to-end algorithms, the model takes raw audio as input and gives also raw audio as output. These networks are trained in learning probabilities of sample succession seeing a frame of a song. Recently the Google Deepmind team developed Wavenet [8] to obtain good audio generation results. It allows to be fed by raw audio and to generate raw audio sample by sample. Building up samples one step at a time is computationally expensive but it is essential for generating complex and realistic-sounding audio. This end-to-end neural network was created at the beginning for speech generation capturing characteristics of many different speakers and returning at the end phoneme recognition. It became a very successful method in multitude of application domains : text-to-speech dependent on individual f_0 curves and speaker characteristics, speech recognition, speech enhancement, and music generation. Therefore it seems that it could serve as a new paradigm for music transformation as well.

This Convolutional Neural Network is composed of several convolutional layers. It uses dilated convolutions instead of convolutions which allow to capture long-term dependency in taking convolutions that initially far from in the time space. In order to extract useful information from audio waveform, the input raw audio has to be quantify thanks to the μ-law algorithm, transforming the problem into a classification task. This network has proven to be an efficient tool in speech generation, learning how to gather different phonemes to create good quality words. It is then realistic to do the same with music. The Google Deepmind team obtained great results in training their network on a dataset of classical piano music. The generation gave fascinating samples with the same sample rate as the input training dataset.

Wavenet has also shown conditioning possibility in speech processing, being able to choose the identity of the speaker such as the sex, the nationality or the age. Choosing these parameters can be transposed into a pitch shifting problem where the pitch shifting can be selected. Thus conditioning answers perfectly the problematic of this subject. The aim of this neural network is now to be able to transform sound. If the dataset is labeled with spectral and temporal features, the algorithm could be able to generate sound imposing a specific transformation. It is actually possible to impose the instrument which must play the generated output audio. With a dataset labeled with either piano or guitar or classical music, the output can be conditioned according to what we want. Thus with a dataset labeled in tempo, we may impose the tempo of the output and thus doing time stretching or labeled in frequencies, we may do pitch shifting. Finally Wavenet can be an excellent candidate for this work where it will learn spectral or temporal transformations or even more complex transformations can be thought like timbre modification or other creative modification.

Part II

METHODS

3 METHODS

Developing an end-to-end for audio is not an easy task. Output song must be produced timestep by timestep with deep neural networks works keeping the same sample rate. So the network has to work with a very small timescale, requiring to its architecture to be ingenious to take into account a memory cell helping to predict the next sample. Different end-to-end algorithms applied to music have already been developed such as the Google Deepmind Wavenet network or SampleRNN which proposes a new solution to implement an end-to-end network. For a better understanding of how the song is processed, these architectures will be developed

3.1 END-TO-END NETWORKS

As said before, end-to-end algorithms have the particularity to process directly raw audio without extracting any features. It allows to expect a better quality in results.

3.1.1 *Wavenet*

Wavenet is a deep neural network, trained on a dataset of classical piano music by the Google team to be able to generate one second of audio on their GPUs.

Increase of the receptive field size by dilated convolutions

This model is fully probabilistic, that means it tries to learn the probability of a sample given a list of previous ones, as an autoregressive model. Each sample is then reliant on previous samples representing the receptive field which has a role of memory. This receptive field allows to create the new sample. To control the size of the receptive field, this network uses deep dilated causal convolutions. Each layer of dilated causal convolutions are causal convolutional layers present in the network but with holes to increase faster the receptive field. Mathematically, a dilated convolution between a signal **f** and a kernel **k** and with a dilated factor **l** can be defined as :

$$(k *_l f)_t = \sum_{\tau=-\infty}^{\infty} k_\tau \cdot f_{t-l\tau} \tag{1}$$

For l=1 we find the equation for a standard convolution. For a kernel size equal to two, the convolution takes into account two neighbor samples of the signal. But for a dilated convolution **l**, it takes two samples away from (l-1) samples between them. In this way the receptive field grew from 2 for a standard convolution to l with a dilated factor equal to **l**. In a whole network, using just causal convolutions imposes to have a receptive field equal to the number of layers present in the network (precisely equal to the number of layers without the input layer plus the size of the kernel -1) whereas with dilated convolutions the receptive field is equal to $2^{nbLayers-1}$, skipping one convolution out of two at each layer. Figure 2 illustrates the phenomenon. Thus the number of layers present into the network can be reduced to reach a precise receptive field size.

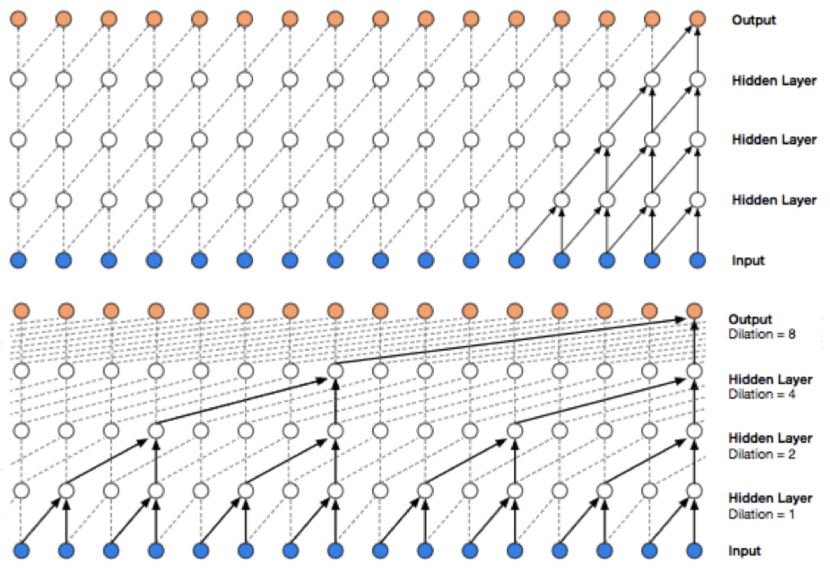


Figure 2: Picture from the Wavenet paper illustrating the comparison of the receptive field size between a stack of causal convolutional layers (top) and a stack of dilated causal convolutional layers (bottom). The receptive field of the orange sample in the output layer corresponds to the number of blue samples present in the input layer which take part in the creation of the output sample.

With a bigger receptive field, the prediction of a sample depends on more input samples. Thus the prediction is less sensitive to fast variations which appear when it depends on the previous two or three samples but the prediction takes into account a larger number of samples, increasing the quality of the result.

To increase again more the receptive field size, the network can be built with several stacks which are on top of each other. A stack is composed of a same number of dilated convolutional layers which can be repeated, increasing the complexity of the network. It allows to have now a very large receptive field with just a few layers, reaching $nbStacks \times 2^{nbLayers-1}$.

A classification problem

Comparing raw audio between the output of the model and the desired output seems not to be an easy task. Values are random integers between -32768 and 32767, coding in 16 bits. In order to decrease computational cost, it is relevant to reduce the number of categories. Works on speech recognition showed a good reconstitution when the audio is coded in 8 bits, ordering to have 256 classes. These separations could be linear but in audio processing, the information is more compacted near the amplitude zero. Consequently classes are built thanks to the μ-law. For each sample, the transformation with $\mu = 255$ is given by :

$$f(x_i) = sgn(x_i) \times \frac{\ln(1+\mu|x_i|)}{\ln(1+\mu)} \tag{2}$$

This function is like a logarithmic function which allows to be more accurate with low amplitudes and less accurate with high amplitudes (with absolute values). The network finally generates a list of samples which can be converted into an audio value thanks to the inverse μ-law :

$$F^{-1}(y) = sgn(y) \times \frac{(1+\mu)^{|y|} - 1}{\mu} \tag{3}$$

Answering to this classification problem requires to have the softmax function as loss function.

Two activation functions : tanh and σ

Each layer present in each stack has a role to do a dilated causal convolution. Before going to the next layer, two activation functions are used as gated activation units. It allows to combine different nonlinearities into the model, thanks to a tanh activation and also a sigmoid activation working as a gate for the tanh function. These two functions are vital for complex problems where the built network is very deep. Then half of the result of the dilated convolution goes to the tanh activation function and the other half goes to the σ activation function. The results of these activation functions are finally multiplied together to give the result of this layer.

Deep residual learning for a deep neural network

It is true that deeper neural networks are very difficult to train seeing the high number of weights and bias present in these networks. Having a very deep network allows to have an excellent learning concerning general clues but experiments show a degradation of results in a deeper network with a higher training error [9] compared to the same network with less layers. Consequently it is relevant to build a building block, called deep residual learning, to improve the training.

The residual learning can be seen as a preconditioned solution. If the function that the network has to learn is closer to the identity function (in the case where the input is quite the same as the output), it is easier for the network to learn a function which is almost a zero mapping than to learn a new function to transform the input into a desired output. In the case of a pitch shifting problem, it is obviously not the identity function that we want to learn but more a function like what the phase vocoder algorithm does. However the input and the desired output have the same nature and therefore is closer to the identity function than a completely random function. Adding this residual learning at each layer allows to have finally a good training, keeping the fact that this network can be trained end-to-end with backpropagation.

Each layer can be then schematized with two trees, a residual tree and a skip tree, as shown in Figure 3. The residual tree allows to conserve the input as the identity function and so the other part has to learn the function h where $h = f - id$ with f the function that the network has to reproduce. The layer does the dilated convolution and pass the result into two activated functions. The result is then convoluted by a 1×1 convolution which is a feature pooling method. Several feature maps go to the skip tree and the rest come back to the residual tree before going to the next layer.

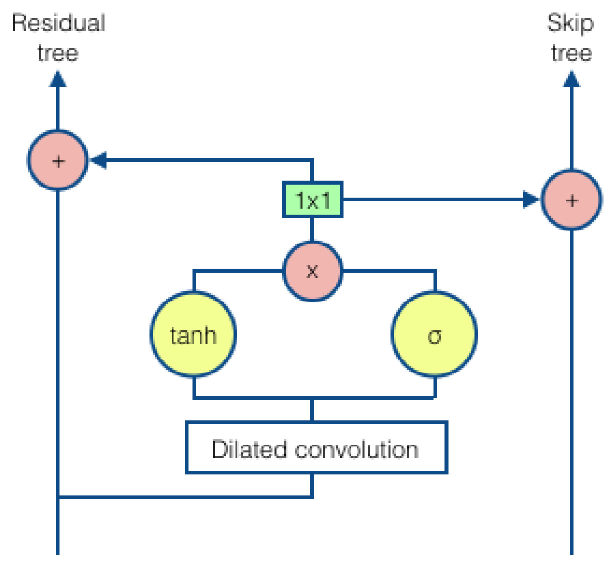


Figure 3: Architecture of a layer doing a dilated causal convolution and applying activation functions.

When finally the input receptive field is passed into the whole network, all results piled into the skip tree are added to become the result h(x) given by the network. To find the desired result, the input is added to this intermediate result. The final result goes to a post-processing task which is composed of steps of ReLU, 1×1 convolution, ReLU, 1×1 convolution and then the Softmax loss function to create the final output.

Conditioning

Conditioning has the role to impose a particular aspect to the result. It was used by Google Deepmind to be flexible regarding the genre (male or female) of the voice or the nationality of the speaker. In the case of a pitch shifting method, conditioning can represent a particular interval of pitch shifting. It can have the role of a fader which allows to choose which pitch the user want.

A probabilistic view of this problem shows the impact of an added condition in the result. In this way a waveform $\mathbf{x} = (x_1, ..., x_N)$ of N samples can be written as a product of conditional probabilities :

$$p(\mathbf{x}) = \prod_{t=1}^{N} p(x_t | x_1, ..., x_{t-1}) \tag{4}$$

If now we want to condition $\mathbf{x}$ by a bias $\mathbf{h}$, the conditional distribution becomes :

$$p(\mathbf{x}|\mathbf{h}) = \prod_{t=1}^{N} p(x_t | x_1, ..., x_{t-1}, \mathbf{h}) \tag{5}$$

The network can be modified to introduce this condition $\mathbf{h}$ inside both activated functions present at each layer. If this condition is the same with the time, we talk about global conditioning. If the condition is also a timeseries, we call it local conditioning.

3.1.2 *SampleRNN*

In finding other possibilities to implement an end-to-end algorithm, there is also SampleRNN [10] which proposes even best results than Wavenet. Still creating music sample by sample, SampleRNN tries to reduce the complexity of the network compared to the Wavenet architecture.

SampleRNN is a parametric model which can model long term dependencies. Contrary to Wavenet which uses dilated convolutions to create a receptive field size, SampleRNN uses a classical method with Recurrent Neural Networks. A RNN is a network where recurrent connexions are present. These networks are very powerful thanks to their compact, shared parametrization of a series of conditional distributions. It is adapted for timeseries and allows to save in memory the effect of the past. An unfolded unity of RNN corresponds to a classic neural network with equality conditions between weights. The output sample at a time t is then linked with the input sample at the time t but also with previous ones. The main problem with a RNN is that after many time steps, the sensibility of the previous samples decreases exponentially with the time until to cannot depend on the first ones. Long Short-Term Memory networks, new RNN architectures, were developed to answer this problem with a cell state which has the role of memory. After that Gated Recurrent Units, varieties of LSTMs, were developed without this cell state. It is particularly these GRUs which are used in this network.

The SampleRNN network is created with several tiers where each tier is built with RNNs allowing to save memory at different scale. The first one is transformed into a serie of Multi Layer Perceptrons. Each MLP takes several samples as input (in the Figure 4, four samples are taken) and also the upsampled output from the second tier seeing as a temporal condition with a wider scale. The second tier is built as a GRU where each iteration takes several generated samples (four in the example) and also the upsampled output from the third tier to enlarge again the temporal scale. Finally the third and last tier is also a GRU where each iteration takes several generated samples (16 in the example) to enlarge again the temporal scale.

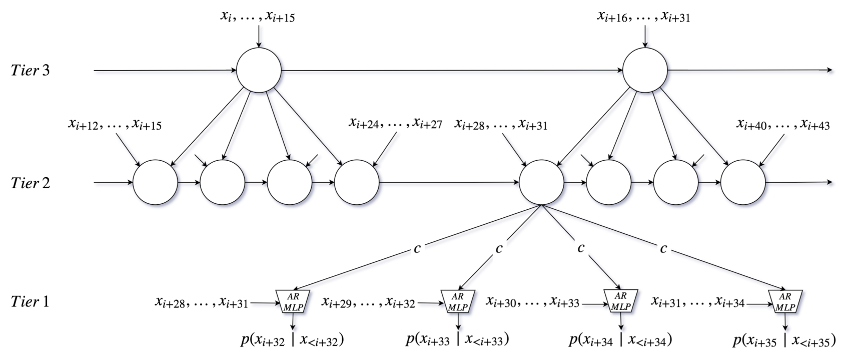

$x_i, \ldots, x_{i+15}$
$x_{i+16}, \ldots, x_{i+31}$
Tier 3
$x_{i+12}, \ldots, x_{i+15}$
$x_{i+24}, \ldots, x_{i+27}$
$x_{i+28}, \ldots, x_{i+31}$
$x_{i+40}, \ldots, x_{i+43}$
Tier 2
c
c
c
c
Tier 1
$x_{i+28}, \ldots, x_{i+31}$
AR MLP
$x_{i+29}, \ldots, x_{i+32}$
AR MLP
$x_{i+30}, \ldots, x_{i+33}$
AR MLP
$x_{i+31}, \ldots, x_{i+34}$
AR MLP
$p(x_{i+32} \mid x_{<i+32})$
$p(x_{i+33} \mid x_{<i+33})$
$p(x_{i+34} \mid x_{<i+34})$
$p(x_{i+35} \mid x_{<i+35})$

Figure 4: Architecture of the SampleRNN algorithm with three tiers. It shows the number of input samples that each tiers takes as input and also the direction of the conditioning, from the top to the bottom allowing to have different time scales.

The network proposes a construction with different modules where each operates at a different temporal resolution. The first one concerns short resolution, links between neighbors, and the other ones increase this resolution. Seeing raw audio contains correlation at different scale, this network can have a good efficiency. RNNs can be trained with a backpropagation. Indeed if the RNN is unrolled, the network is seen as successive layers. However it is computationally expensive because the gradient has to be computed until the beginning of the sequence. To increase the training speed a method called Truncated Backpropagation Through Time [11] is presented. It imposes a limit on the number of steps to run backwards without altering long term dependencies.

As said in the title of the paper, this new network seems to be "unconditional", forbidding to put some conditions concerning which pitch shifting we want to have. A first solution is evoked by the same team in a speech synthesis problematic [12]. It proposes to add an extra input as a condition to a particular time step.

3.1.3 *First experiences with existing networks*

The first tests were to test both networks Wavenet and SampleRNN with sine waves as input, thanks to codes given respectively on GitHub. Concerning Wavenet, the code is written in TensorFlow, an open source library developed by the Google Brain team allowing to use Machine Learning functions. Simulations started with a very small network, composed of two dilation stacks of three different dilations (1, 2 and 4). This configuration imposes a receptive field equal to $2 \times 2^3 = 16$ samples. The learning signal is a sine wave of 440 Hz during one second with a sample rate equal to 16000 Hz. Thus one period is equal to 36 samples which indicates that the receptive field corresponds to a few less that half a period. The network was trained during 100 epochs and results generation were extracted after 30, 60 and 100 training epochs.

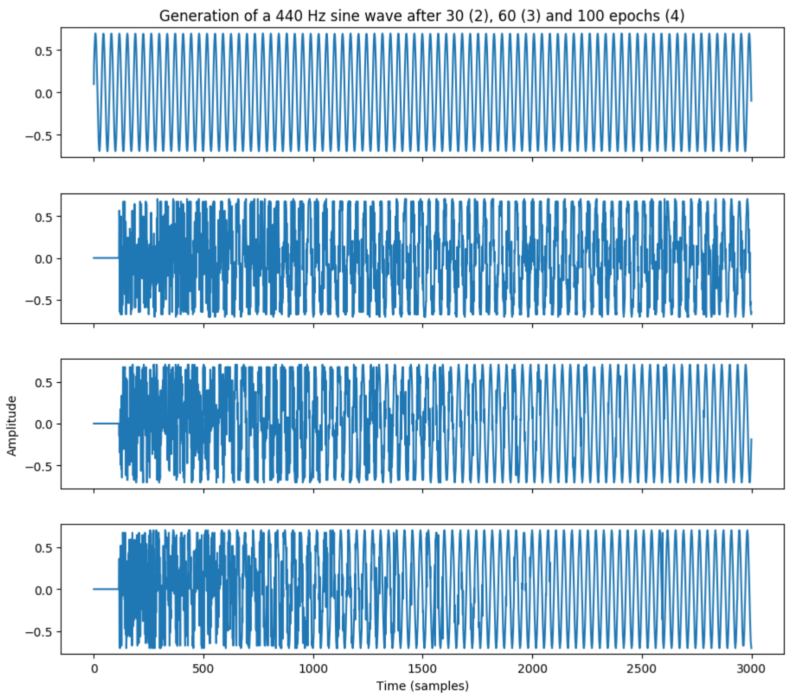


Figure 5: Wavenet trained with a sine wave of 440 Hz (top) and results generation after 30 (2nd), 60 (3rd) and 100 (4th) training epochs.

The final result shown in Figure 5 gives a good audio result even if some artifacts are still present. They can be deleted in increasing the training by doing more training epochs. The network is already able to generate one frequency. The next step is to learn two frequencies in a row. The training song is now 150 ms of a 440 Hz sine wave joined with 150 ms of a 1 kHz sine wave. The result shown in Figure 6 underlines that the network learns just the last frequency with regular transition moments present between samples 500 and 1500 and starting again at sample 3900.

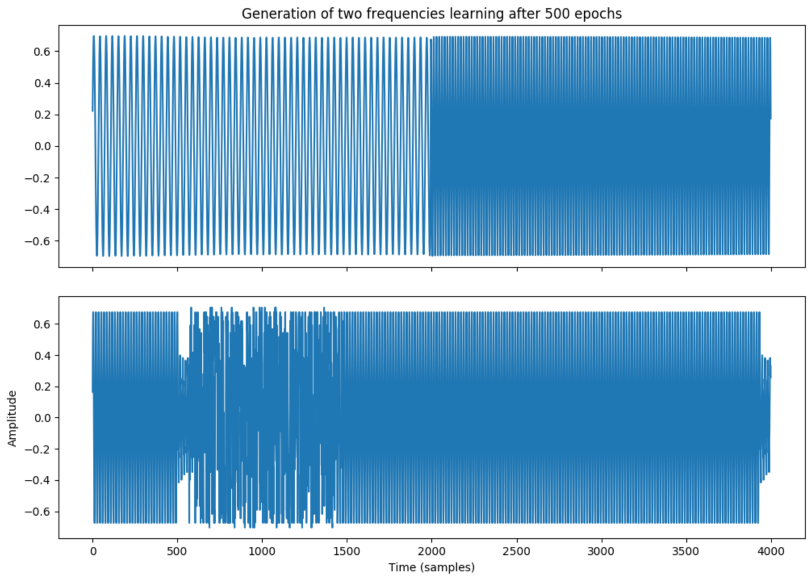


Figure 6: Wavenet trained with a 440 Hz sine wave with a 1 kHz sine wave (top) and results generation after 500 training epochs (bottom).

Learning two frequencies is impossible using this method. A different way to learn different frequencies is to label them thanks to the possibility of conditioning. However the code given by Google is really complex to be modified as I want and that is the occasion to test the second one SampleRNN.

The code of SampleRNN given on GitHub is written in Theano, an other Python library for Machine Learning. Contrary to the Wavenet code, the SampleRNN code is more flexible and easier to understand. The same method is applied to this new algorithm, testing with simple examples.

Results are shown in Figure 7. After few iterations, we obtained the right sine wave. Concerning the learning of several frequencies, it succeeds. There is a transitional mode of around 150 samples, between sample 1450 and sample 1600, to go from the first frequency to the second one but this time could decease with a longer training.

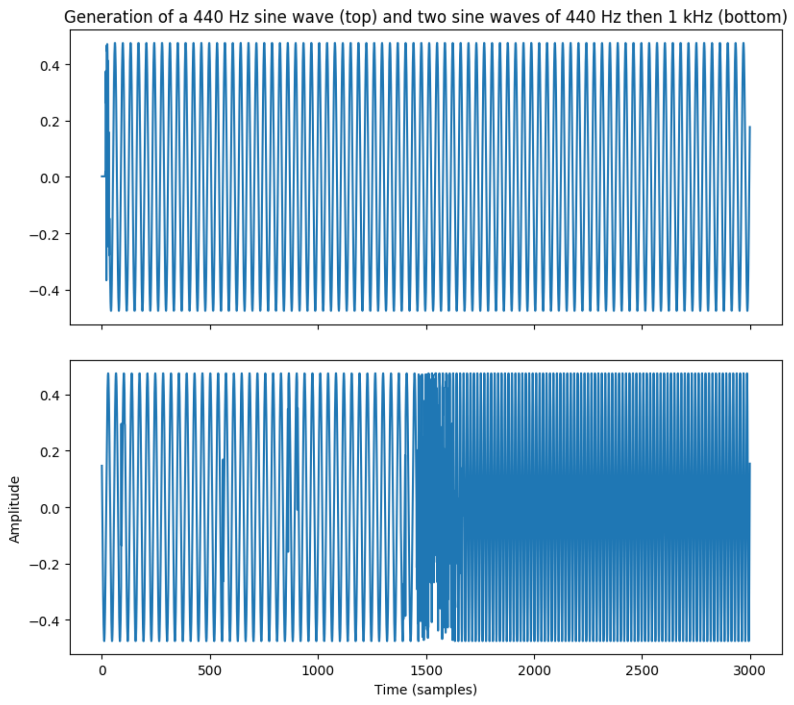


Figure 7: SampleRNN trained with two tiers where each RNN as a dimensionality of 1024, 64 frames in each TBTT with 16 samples per frame. Results are obtained after 80 epochs.

Adding conditioning in SampleRNN is not an easy task. Indeed conditioning is an abstract data, evoked sometimes as a bias, and which must modify the probability of a waveform. In SampleRNN, the different time scales impose to master the timescale of the conditioning. In a pitch shifting point of view, conditioning represents the pitch interval we want to apply to our input song. Therefore it has to take action in the upper tier in the network.

Despite finding a successful way to implement it, I decided to create my own network to be able to modify it as I want. The network I focused is Wavenet which has already shown good applications with conditioning.

3.1.4 *From PixelCNN to Wavenet*

As a lot of Machine Learning networks applied to music, Wavenet comes from a network of picture generation. When Wavenet produces a new song sample by sample, the previous network PixelCNN [13] produces a new image pixel by pixel. In a way a song can be seen as an image in one dimension. To implement the Wavenet network, it is interesting to look at how to implement PixelCNN.

PixelCNN is a deep neural network which allows to predict the pixel values with an unsupervised probability learning. The problem is seen as a sequence problem where each sequence learns to predict the next pixel given all the previous generated pixels. Basically inspired by PixelRNN using LSTM layers, it was observed that Convolutional Neural Networks (CNN) can also be used as a sequence problem offering the best results. Then LSTMs layers are replaced by a fully convolutional network given a conditional distribution at each location.

For the same reasons as Wavenet, values of pixels are rounded into 256 discrete values to answer to a classification problem. The joint distribution $p(\mathbf{x})$ of an image $\mathbf{x} = n \times n$ pixels can be represented like in the Wavenet model :

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i|x_1, ..., x_{i-1}) \tag{6}$$

This joint distribution gather all local probabilities of pixels present in the image. The probability $p(x_i|x_1, ..., x_{i-1})$ can be translated by what is the probability of the i^{th} pixel given the (i-1) previous ones. Adding them to the product step by step, the generation of an image is finally solved by recurrence from the top left hand corner pixel to the bottom right hand corner, row by row and pixel by pixel. Thus the generation costs time because each pixel requests a forward pass. For training, it is not necessary to wait the value of a pixel to predict the next one because the network has initially access to all pixels in the training images. Thus the distributions over the pixel values can be computed in parallel. Also each pixel is composed of three values in the RGB representation. The model presented in the Wavenet paper links these values by an internal probability which will not be developed because this complication does not affect raw audio. Finally to simplify this model, each pictures will be rounded in two discrete values, 0 or 1, seeing images in black and white.

Chainer framework

Finding helpful explications in the blog of Sergei Turukin [14] and an already well explained implementation of this PixelCNN [15], all codes which will contribute to this project will be coded in Chainer. Chainer is a “flexible deep learning framework” for neural network which works with Python. The difference from other famous deep learning frameworks like Tensorflow or Torch is that Chainer constructs neural networks dynamically, which enables to write a neural network in more flexible way. Thus the network can be extended step-by-step into the forward pass and not be prepared before. It can be adapted to GPU computation.

The particularity of Chainer is to put all data into a *Variable* object where are put shape, type, data or gradient. All steps of a network such as Linear, Convolution, LSTM are stored in *Links* where weights and bias are automatically computed. All functions such as activation functions, loss functions or mathematical functions are stored in *Functions*.

Model

The model is constructed similarly to Wavenet. It is composed of many identical layers (the paper proposes to fix it at 15) on top of each others. These layers are used to calculate the probability of a pixel given the previous ones. To be sure that the model is causal, the filters of the convolution need to be masked to not to use future pixels. These masks are binary masks which allow to deactivate the influence of future pixels. Unfortunately applying that in a 2D case creates a blind spot. A blind spot is a group of pixels which are situated before the target pixel but which are not taken into account in the calculation of the target pixel prediction. This missing information appears because of the convolution in two dimensions but this problem can be solved in changing the way of considering the receptive field. The network is then transformed into a Gated PixelCNN where the receptive field is divided into an horizontal stack and a vertical stack and combining them. The vertical stack conditions on all rows above the current row and the horizontal stack conditions on the current row. Figure 8 shows the layer architecture. Each layer is then composed of two trees : a vertical tree on the left and a horizontal tree on the right. To calculate vertical feature maps, $n \times n$ convolutions are applied allowing to take into account all pixels above the target pixel. And then to calculate horizontal feature maps, $1 \times n$ convolutions are applied to take into account just pixels before the target in the same row. Horizontal feature maps are calculated seeing the value of vertical feature maps. Finally both vertical and horizontal feature maps are given to the next layer.

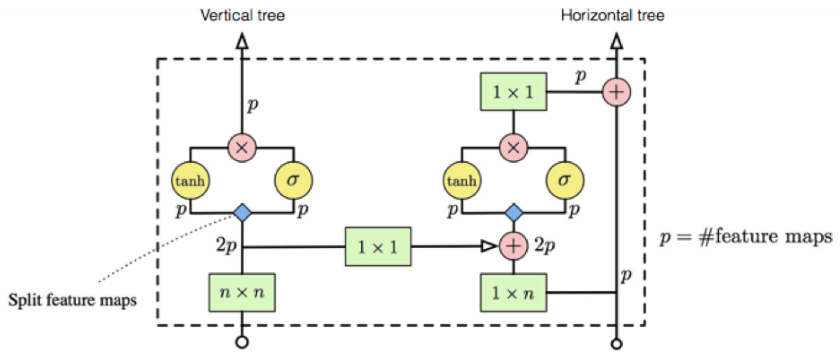


Figure 8: A PixelCNN layer composed of tow trees : the vertical tree on right and the horizontal tree on left.

At the end of these layers is present a post processing step. It is composed of a ReLU function, a 1×1 convolution, a ReLU, a 1×1 convolution and the softmax loss function.

Pre processing

Data can initially take random values. They are transformed from values in zero-one range to categories between 0 and 255, as a classification problem. That means that input values are rounded into the nearest label value.

Generation

The generation is done pixel by pixel. The network is at the beginning fed with a white image and it calculates the first pixel, modifying the input. Given this modification the network is now fed with the new input to calculate the second pixel and so forth, costing a lot of time because of the sequential but non parallel problem.

Conditioning

Conditioning allows to guide the result to a hoped prediction. As the same idea as Wavenet, the condition takes place inside the two activation functions. The condition is multiplied by an embedding matrix and put inside the sigmoid and the tanh function. To generate an example, the network can be trained with the MNIST dataset, a database of images where each image is a digit from 0 to 9 written by hand. This dataset is composed of 60000 training images and 10000 testing images. Each image has 28×28 pixels. To simplify the problem these images are transformed into binary images, composed of 0 and 1 and limiting the problem classification in two classes. Each image is labeled with the matching digit and the label is put in the network as a condition, allowing to learn the bias corresponding to each digit. Figure 9 shows the result obtained in conditioning the network to generate a 0.

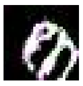


Figure 9: Image generation conditioned to generate a zero digit. The model is composed of 15 stacks, 128 hidden feature maps and 2 levels. It was trained on the MNIST dataset with 20 epochs.

Modifications for Wavenet

With PixelXNN, images which are fed to the network are matrices of integers between 0 and 255. Now with Wavenet, songs are one dimensional vectors with also the same type of values. Converting PixelCNN into Wavenet ask to apply 1D convolutions rather than 2D convolutions, causing the ending of the blind spot problem and the modification of masks. The receptive field is now a vector of samples helping to calculate the prediction of the next one. Masked convolutions are replaced by dilated convolutions, increasing the dilation layer after layer.

The PixelCNN algorithm was used to learn how to generate a song or a picture, conditioned on the number of a digit. The condition was the ten digits and choosing one causes the generation of this digit thanks to learned probabilities. Concerning Wavenets, Google Deepmind proposes to condition it to choose the features of the voice such as the sex or the nationality. Transposing the problem into music, digits can be replaced by frequencies. Thus each label can be used to learn different frequencies. Then the network learns different frequencies which can be generated separately.

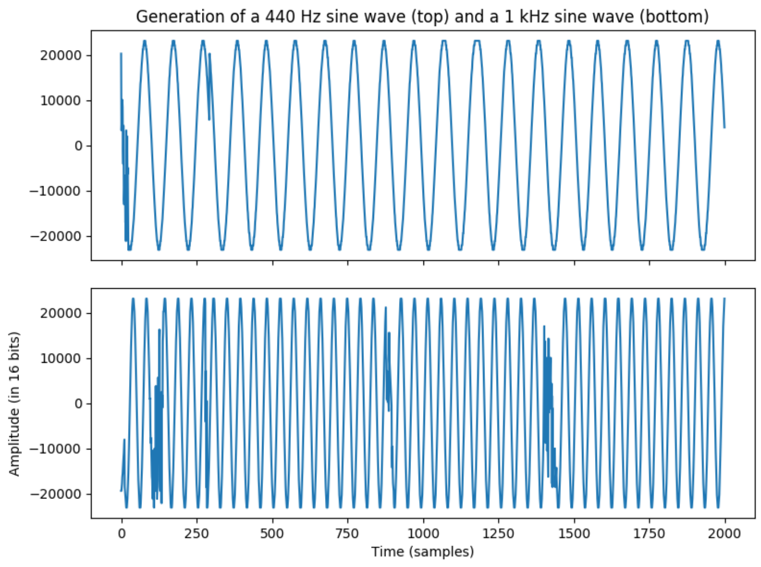


Figure 10: Wavenet generation with a receptive field of 4 samples ($1\ \mathtt{stack} \times 2^{2\ \mathtt{layers}}$). The network was trained during 1000 epochs with 5 seconds of each frequency with a sample rate of 44100.

Figure 10 shows the generation of two different frequencies which were learnt during the training. Several artifacts can be seen in the sine waves. This is not linked with an insufficient training because after 15 epochs, the best loss value was reached (see Figure 11). A solution at this problem can be to increase the receptive field implying that each sample is dependent on more previous samples. It allows to gain in stability but the network becomes more complex.

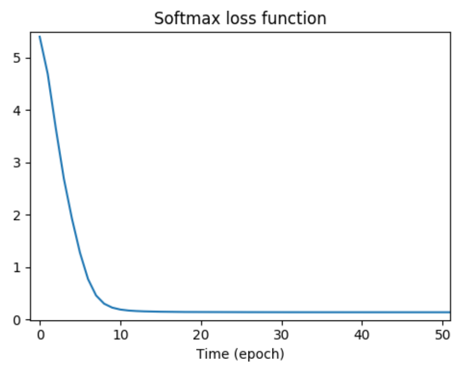


Figure 11: Softmax loss function applied

To generate two different sine waves, each frequency was labeled with a different number. The first sine wave at the top was generated asking the label named "0" and the second sine wave at the bottom was generated asking the label "1". The network has correctly associated the label with the frequency. But to shift the pitch of an entire song with this method, it implies at the end a Fourier series decomposition to have access to the sine wave decomposition which is inappropriate.

A good way of training is to consider these labels as a different pitch shifting interval. For each label, the network does not have to learn predictions to generate the same output than the input. The network has to learn the pitch shifting function which corresponds to a particular pitch shifting. The input of the network is a song and the desired output is this same song but pitched.

Part III

LEARNING THE PITCH SHIFTING FUNCTION

4 LEARNING THE PITCH SHIFTING FUNCTION

Wavenet has shown good capacities to reproduce sine waves. If now the output is modified to become a low or high-pitched version of the input, the learning of a pitch shifting function can be expected.

4.1 DATASET CONSTRUCTION

As a step by step learning, first works concern sine waves. The aim is to modify the pitch of one sine wave, then multiple sine waves with different frequencies and different amplitudes. Finally tests will concern a real sound of a flute.

4.1.1 *Pitch shifting*

Learning the pitch shifting function consists on putting the pitched song at the desired output of the network. The chosen transformation is to increase the pitch to one octave. Working with a doubled frequency allows to avoid phase problems. It forces the network to just have two amplitude solutions for the high-pitched frequency seeing an initial frequency. The training dataset is composed of one 400 Hz sine wave labeled "0" and one 600 Hz sine wave labeled "1". Each frequency has a duration of one second with a sampling frequency of 44100 Hz. Then the desired output for the label "0" is a 800 Hz sine wave and for the label "1" a 1200 Hz sine wave. To obtain fast results, the network is limited to one stack and two layers, given a receptive field of four samples.

The generation of these two labels can be shown in Figure 12. The 400 Hz frequency is well transformed into a 800 Hz frequency and the 600 Hz frequency is well transformed into a 1200 Hz frequency. Some artifacts are present, always in the same place, just behind the zero amplitude line. For example, the sample 250 on the generated sine wave of 400 Hz is higher than expected. It is where the derivative is the highest. To generate a sample knowing the previous one, it exists two possibilities (if we except peaks). Either the amplitude is higher if the derivative is positive or the amplitude is lower if the derivative is negative. Knowing that the size of the receptive field is very small, the prediction is based on the value of four samples. Then the prediction can fail, believing the wrong derivative. Increasing the size of the receptive field will cause a bigger temporal memory and could erase this problem.

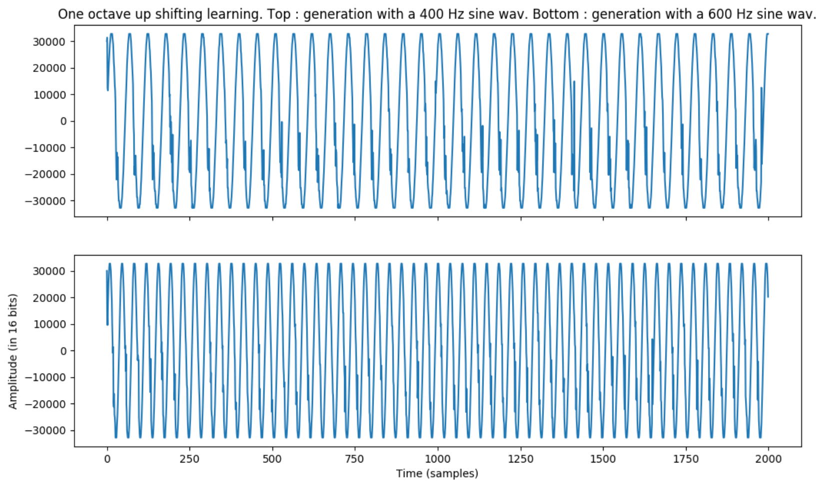


Figure 12: High-pitched transformation learned on a 400 Hz (top) and a 600 Hz (bottom) sine waves.

The pitch transformation can be learned by the network. When the network is fed by a 400 Hz sine wave labeled with 0, it gives a 800 Hz sine wave and when it is fed by a 600 Hz sine wave labeled with 1, it gives a 1200 Hz sine wave. If we expect to shift all frequencies of a sound, the different frequencies do not have to be at different labels but it needs to be in the same, considering labels as pitch shifting intervals. The idea to learn several frequencies is to join them in the input. Thus the learning of the 400 Hz sine wave and the 600 Hz sine wave in the same label is done with a dataset composed of these two joined frequencies where each lasts half a second. The desired output is then composed of two joined frequencies, a 800 Hz and a 1200 Hz sine waves with the same sample rate and the same duration.

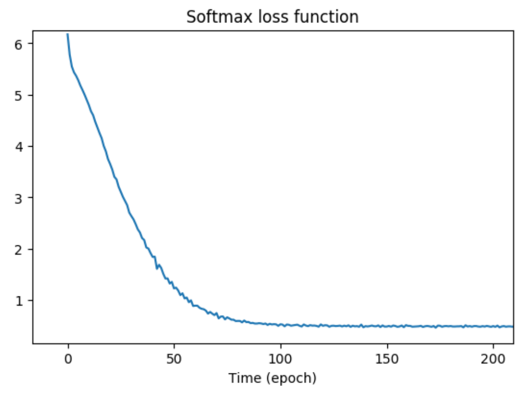


Figure 13: Loss function of the learning of two frequencies in the same time.

The generation consists on modifying the pitch of the two joined frequencies, 400 Hz and then 600 Hz.

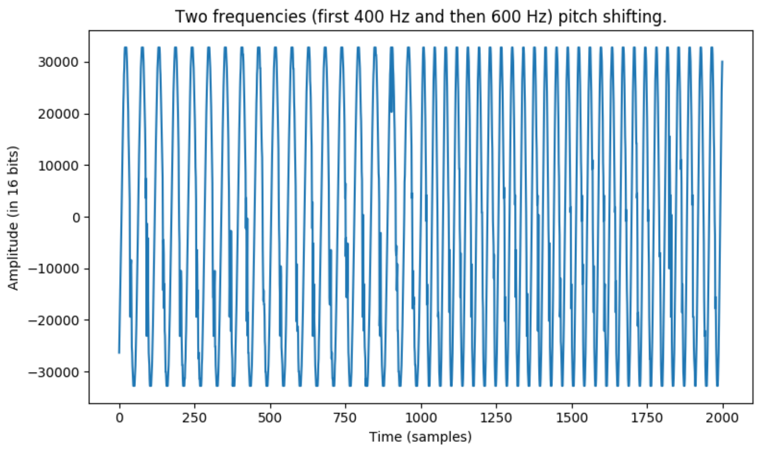


Figure 14: High-pitched transformation of two sine waves learned in the same label.

The result shown in Figure 14 gives a 800 Hz frequency followed by a 1200 Hz frequency which what was expected. Frequencies given during the training had always a constant amplitude. If we try to modify the amplitude, the network does not learn by itself how to generate the high-pitched signal at this new amplitude. The dataset has to be modified to learn all different amplitudes.

4.1.2 *Learning different amplitudes*

In the case of learning different amplitudes, the frequency is fixed at the two previous values, 400 Hz and 600 Hz. The first idea to learn several amplitudes is to transform the training dataset into an amplitude sweep. The sweep can be linear or logarithmic in the amplitude domain and can goes from 0 to $2^{15} - 1$ in amplitude. However the network does not succeed to generate a constant sine wave at a particular amplitude. This is due to the fact that the network has to see at least one period at each amplitude we want to generate. Without that it is impossible to generate expected outputs. Seeing that the amplitude is not linearly quantized by the network but follows the μ-law, a sweep was applied after applying the μ-law at the audio signal but this method did not give good results.

A new idea consists on the creation of steps of amplitudes where at each step is present more that one period of the frequency. To force the network to well understand that it is at a particular step of amplitude, it is important to increase the receptive field. For that, the network is now composed of two stacks and five layers at each stack. It corresponds to a total of $2 * 2^5 = 64$ samples which assures to have more that half a period of the frequency in the receptive field. The number of amplitude steps must be chosen. If this number of steps is too low, the network does not learn amplitudes which are present between two successive amplitudes and generates a noisy signal. Figure 15 shows that for 10 amplitude steps, it is not enough to generate a random amplitude which can be situated in the middle of two steps.

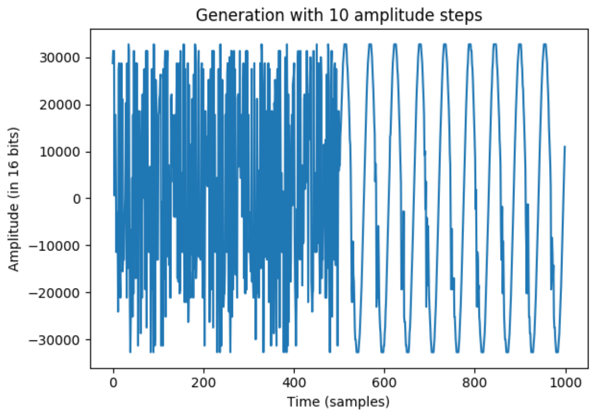


Figure 15: High-pitched transformation of a 600 Hz sine wave with an amplitude in the middle of the 5th and the 6th step and then of a 400 Hz sine wave with the last amplitude step. The learning was done with 10 amplitude steps where each step was composed of 5 periods.

After experiments it was shown that 40 amplitude steps allow to have a good generation even if the desired amplitude is situated in the middle of two steps. The number of periods at each step must be also fixed. I explained that we must have at least one period at each amplitude step but as shown in Figure 16 it seems to be not enough to expect having a good audio generation. A good learning was found for 7 periods.

The dataset to learn the generation of a random amplitude is finally composed of 40 steps where each is composed of 7 periods of the frequency (see Figure 17). For the two training frequencies, 400 Hz and 600 Hz, it is equal to 51400 training samples so a few more than one second.

As shown in Figure 19, the output gives the desired frequencies. We can see in the first one the amplitude is oscillating because the asked amplitude is between two steps but it has not generated noise. However the transition between frequencies is noisy. Samples between 600 and 620 are not well transformed because the associate receptive field composed of 64 samples takes into account more than the half of its samples in the first frequency. Then the network is lost when the receptive field is in the middle of both because it never saw this signal during the training. The problem can be lowered by reducing the size of the receptive field but yet this operation gives a noisy sound.

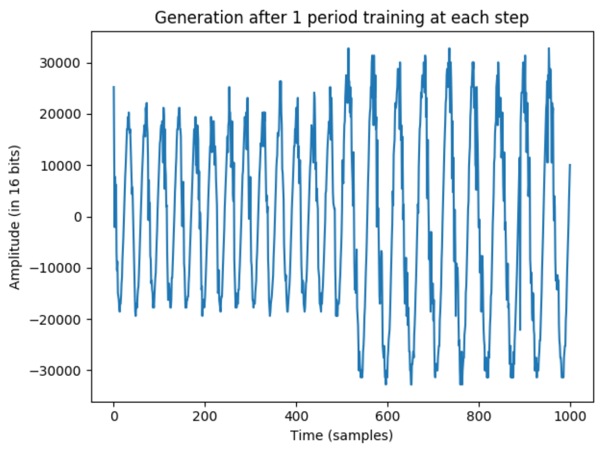


Figure 16: High-pitched transformation of a 600 Hz sine wave with the amplitude of the 23th step and then of a 400 Hz sine wave with the 38th amplitude step. The learning was done with 40 amplitude steps where each step was composed of 1 period.

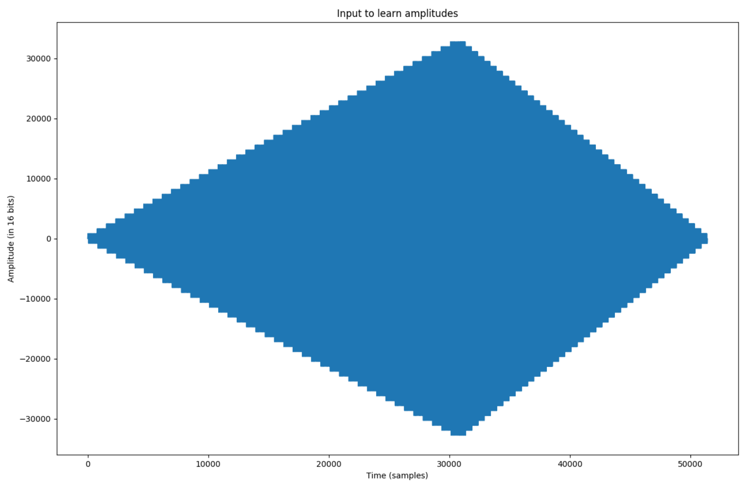


Figure 17: Dataset to learn amplitudes. Il is composed of 40 increasing amplitudes steps in the 400 Hz sine wave and then 40 decreasing amplitude steps in the 600 Hz step.

We now have a dataset allowing to generate a random amplitude in the two studied frequencies. The next step concerns the learning of audible frequencies.

4.1.3 *Learning different frequencies*

Keeping the same approach as for the amplitude learning, the dataset was transformed into a frequency sweep. At constant amplitude, the

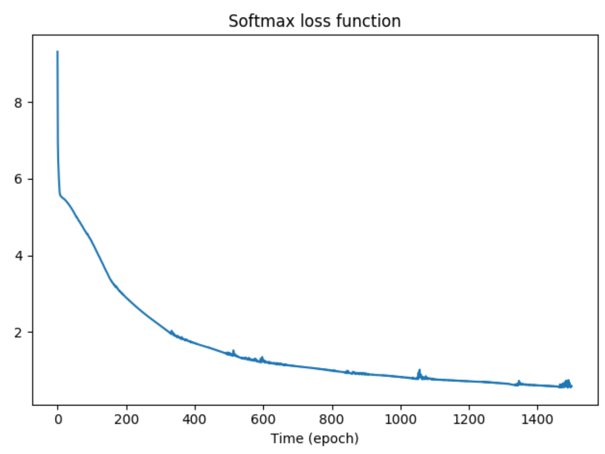


Figure 18: Loss function for the amplitude training.

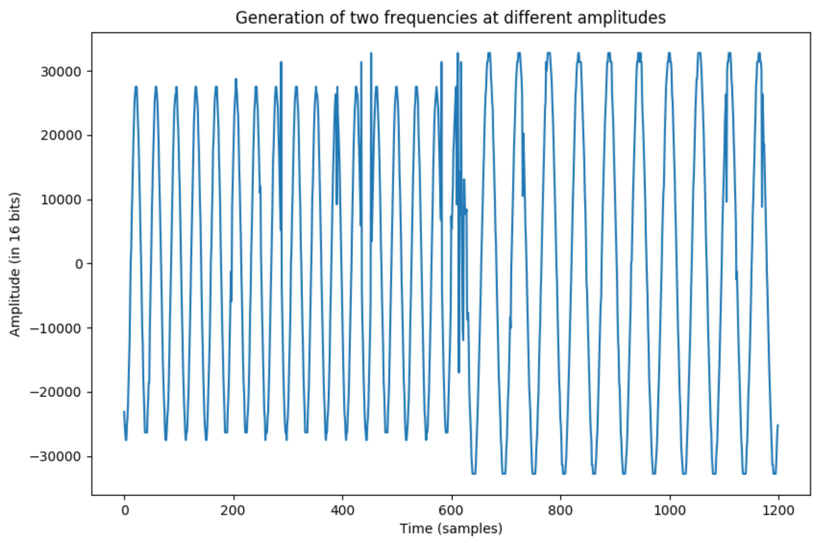


Figure 19: High-pitched transformation of a 600 Hz sine wave with an amplitude in the middle of the 32th and the 33th step and then of a 400 Hz sine wave with the last amplitude step. The learning was done with 40 amplitude steps where each step was composed of 7 periods.

sweep linearly goes from 200 Hz to 800 Hz for the input and so from 400 Hz to 1600 Hz for the desired output. Asking to transform a 400 Hz sine wave joined with a 450 Hz sine wave, the result shown in Figure 20 a good resolution.

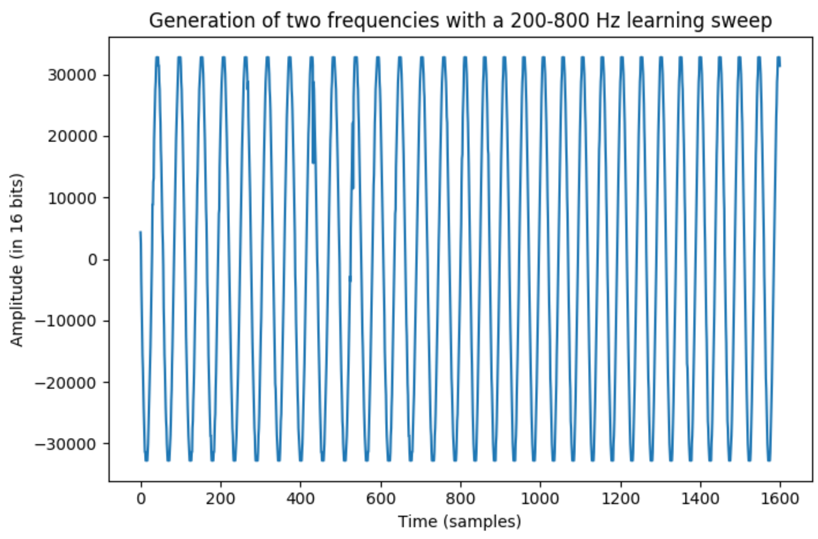


Figure 20: High-pitched transformation of a 400 Hz sine wave and then of a 450 Hz sine wave with the last amplitude step taken into a frequency sweep with a range of 200-800 Hz.

4.1.4 *Dataset generalization*

To generalize sine wave transformations, both ideas of amplitude and frequency can be mixed together. The aim is to be able to transform a sine wave whatever the amplitude and whatever the frequency. Thus the dataset is transformed into 40 steps of amplitudes where in each step there is a frequency sweep from 200 Hz to 800 Hz with a duration of one second (see Figure 21). The input is then composed of 40 seconds of audio. The desired output is composed of 40 steps of amplitudes where in each step there is a frequency sweep from 400 Hz to 1600 Hz.

The training lasts more than five hours on GPU to compute 3000 epochs. The loss function shown in Figure 22 seems to be at the limit of convergence even if the value is still high compared to previous examples.

The result shown in Figure 23 was obtained in asking to transform a 400 Hz sine wave at the 32th amplitude step stuck with a 450 Hz sine wave at the 23th amplitude step. Sometimes the maximal amplitude of the sine wave is higher than normally, as if the network generates the next step of amplitude.

Having good results with sine waves, it is conceivable to test with real sounds. The aim is to transform a note of a flute at 440 Hz into a one octave higher note. As seen in Figure 24, the given result is noisy. We can see that the network has a lot of difficulties to save the shape. The frequencies are quite doubled inside (see Figure 25) but there are always erratic points.

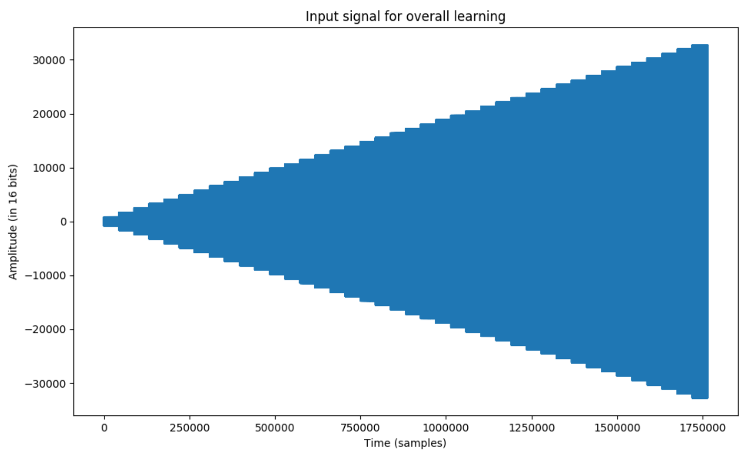


Figure 21: Dataset fed to the network to learn all transformations.

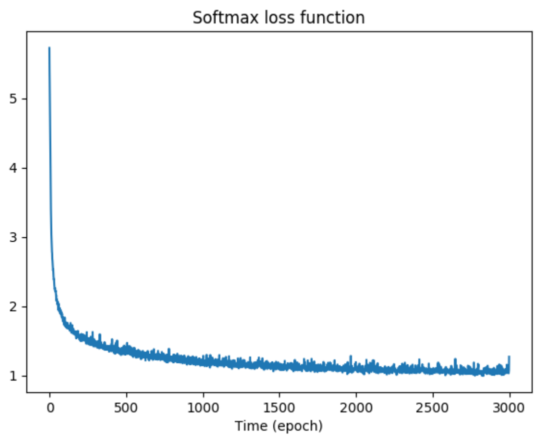


Figure 22: Loss function value.

The noise comes from the fact that all harmonics of a flute placed on top of each other give a non sinusoidal wave. Thus the network does not succeed to link the input with a already learned sine wave. To answer the fact that the network does not succeed to keep the envelope of the input signal, this envelope can be extracted thanks to a Hilbert transform.

The used method illustrated in Figure 26 is to divide the signal by its envelope. It produces a signal which has quite the same amplitude and where harmonics are less visible. Thus the created signal is nearer a sine wave than before. The fact that the amplitude is quite constant gives the possibility to not to be affected by a transition between amplitude steps. This signal is transformed by the network and the output is multiplied by the initial envelope. As shown in Figure 27, we can see that the frequency is quite well doubled inside but it is still noisy.

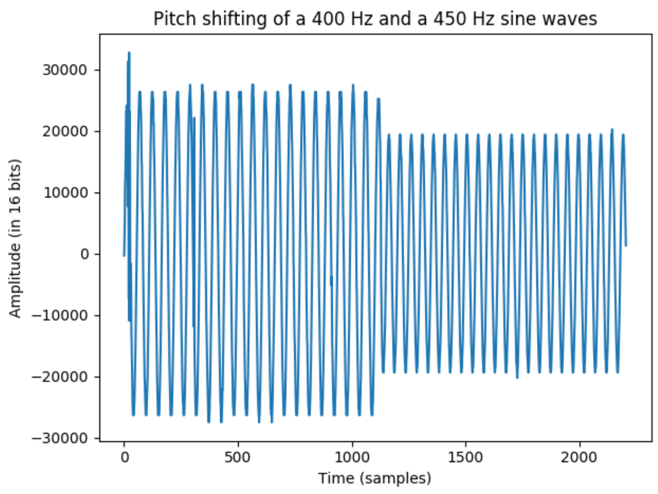


Figure 23: High-pitched transformation of a 400 Hz sine wave at the 32th amplitude step and then of a 450 Hz sine wave at the 23th amplitude step.

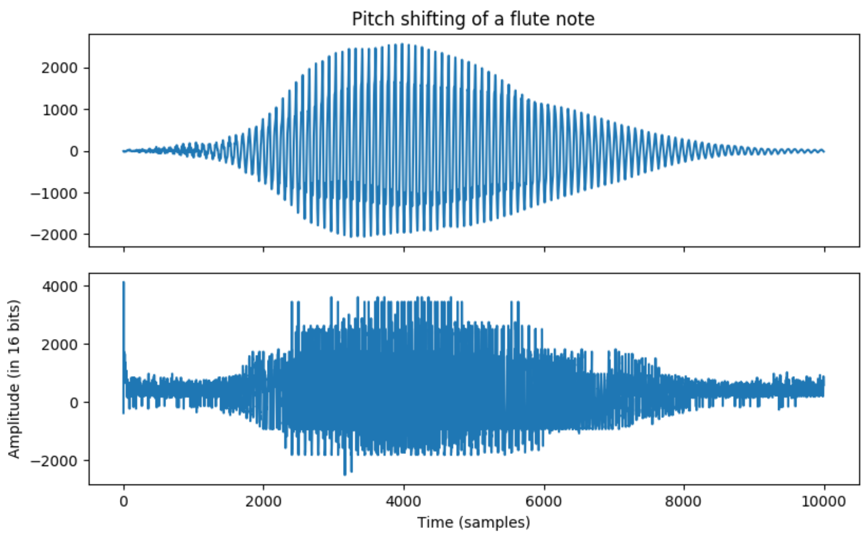


Figure 24: High-pitched transformation of a A note of flute. Top : input, bottom : output.

4.2 NSYNTH : A WAY TO SAVE SEVERAL SOUND FEATURES

Recently the Google Brain team presented a project in audio synthesis named NSynth [16]. This algorithm has the role to generate music copying human expressions, as a musician could do when he plays a song. One of the goals of Magenta is to use Machine Learning to learn these human expressions. Thus they created NSynth (abbreviation of Neural Synthesizer) to propose a new tool for artists to control timbre

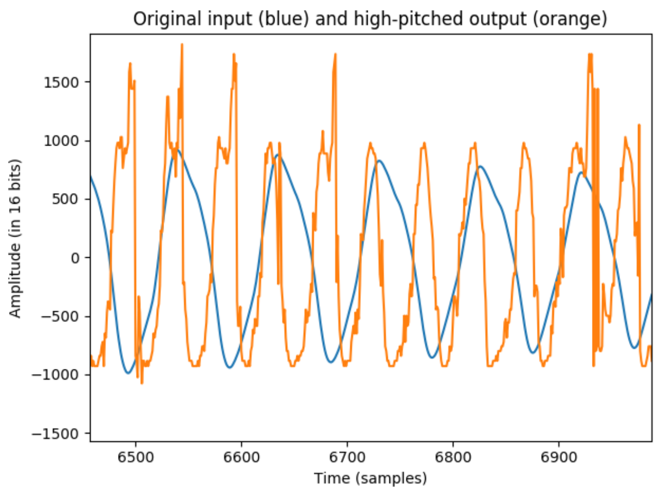


Figure 25: Zoom inside the result. Input in blue and output in orange.

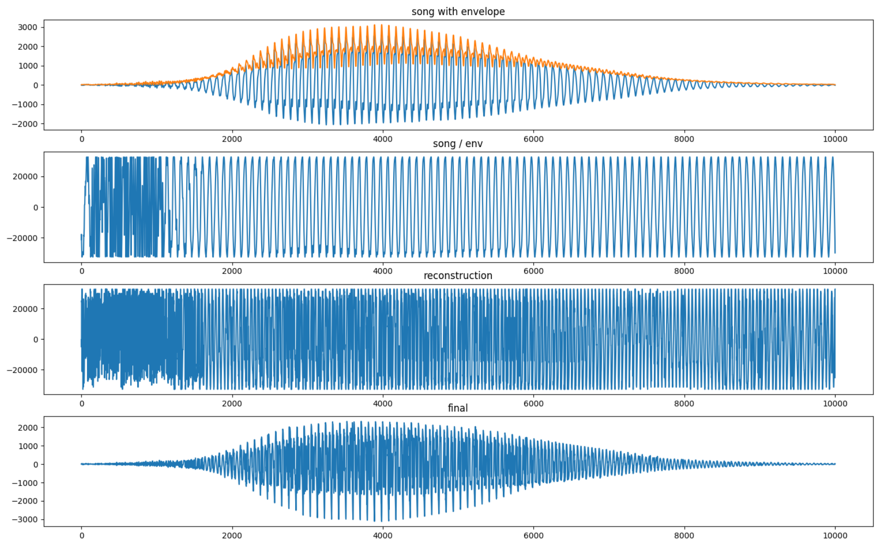


Figure 26: High-pitched transformation using Hilbert transformation to keep the temporal envelope.

and dynamics. With Wavenet it was not possible to do it because the memory of the network was limited to several thousand samples which correspond to the size of the receptive field. In that way this new end-to-end algorithm allows to keep long-term structure.

4.2.1 *The network*

The network is a Wavenet-style autoencoder which allows to generate a song sample-by-sample keeping long term structure. For that, a Wavenet network which was explained before is linked with an

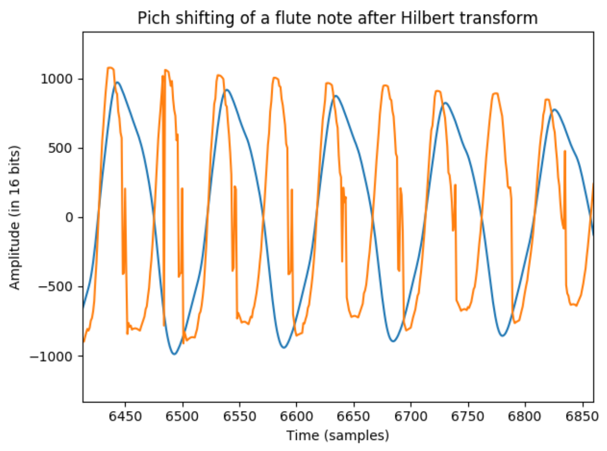


Figure 27: Zoom inside the result. Input in blue and output in orange.

autoencoder. The audio encoding allows to keep long term structure and insert itself in the Wavenet as a global conditioning. The temporal encoder model is composed of 30 layers. Each layer is composed of a dilated convolution, a ReLU activation function to insert non-linearity and a 1×1 convolution to increase the number of feature maps. At the end there is an average pooling layer which allows to reduce the temporal dimensionality.

This association which has already proven its efficiency in image generation [17] is a good way to save global features that our transformation function does not have to modify. This encoding can be made thanks to non causal dilated convolutions which were not present in Wavenet. This network was trained with a large database of musical notes which are labeled with a unique couple of pitch, timbre and envelope.

As an end-to-end algorithm, the network is fed by raw audio and gives raw audio. The raw audio is encoded into a reduce form where the principal features are stored. The encoding is then decoded to generate the output raw audio. The encoder applied by NSynth gives at all 16 channels put in a matrix which have quite the same shape as the envelope of the raw audio, as shown in Figure 28. This matrix can be seen as a picture which can be stretched. If we resize the time axis of this picture and we decode this new picture, it finally causes a time stretching modification into the raw audio during the generation. The algorithm proves that keeping global features could be a solution to learn time stretching and so pitch shifting modification. The pitch shifting result on the note of a flute can be seen in Figure 29.

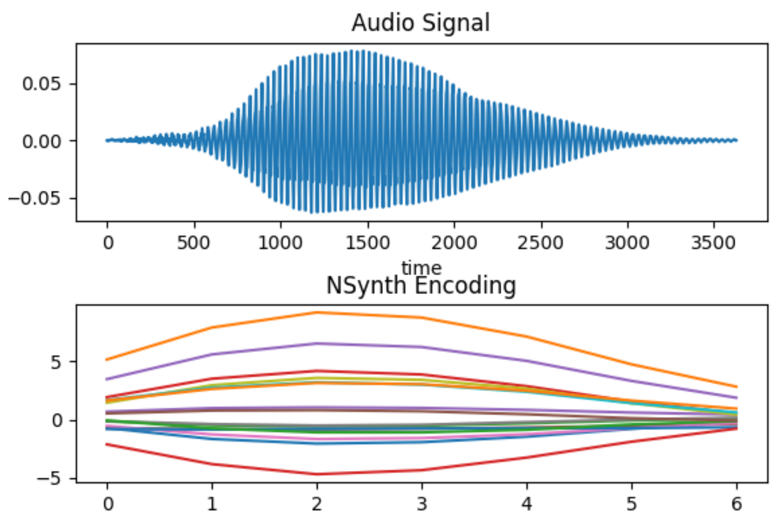


Figure 28: Encoding of a note of flute.

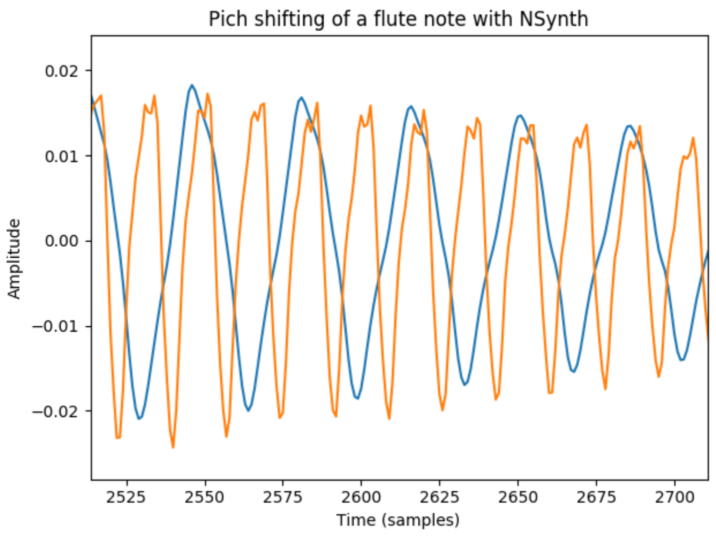


Figure 29: High-pitched transformation of a note of flute. Input in blue and output in orange.

4.2.2 *Experiments*

The previous network used to learn a pitch shifting function is completed by an autoencoder which has the role to keep long term dependencies which were lost before. As explained in the NSynth paper, the Wavenet encoder can be schematize by a convolutional autoencoder. An architecture called Baseline Spectral Autoencoder and shown in Figure 30 was also tested by the team to reproduce the encoding. It is fed by spectrograms which are calculated using a large FFT size of 1024 samples relative to the hop size of 256 samples. Then the network encodes them and decodes the encoding to gives a new spectrogram which must converge to an input copy. The encoder is composed of 10 2D convolutional layers and the decoder is composed of 10 2D deconvolution layers. Deconvolution layers are transposed convolutional layers. To apply this type of convolution, the input is padded by inserting (s-1) zeroes between each samples and doing normal convolutions. The spectrograms dimensions are reduced and then increased thanks to a kernel stride. The loss function is the mean-squared error learned with a learning rate fixed at 10^{-4}.

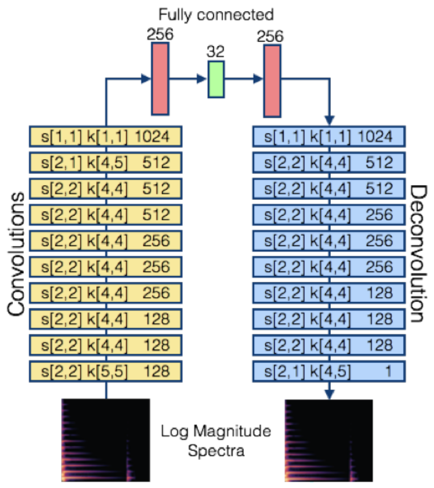


Figure 30: Autoencoder : encoding and decoding are made with 2D convolutions with stride s, kernel size k and channels. Between them, three fully connected layers allow to have a vector of 32 values to encode each input.

This architecture was reproduced and trained with a dataset of flutes. It is composed of 6362 sound files of 4 seconds each. Each spectrogram representing a 512×256 matrix is fed to the network, giving an encoding vector with a length of 32, as shown in Figure 31. The training lasts 56 hours in one GPU, given a loss function value equal to 0.035 (see Figure 32).

The encoding is a short representation of each song. It can be insert as a global conditioning into the previous Wavenet to keep long term structure. This experiment should be the next step of this project.

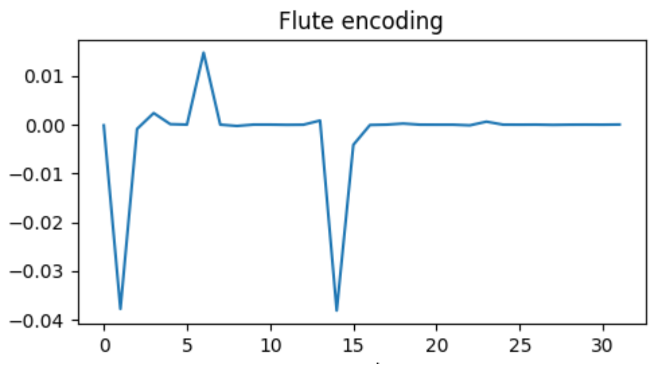


Figure 31: Encoding vector of a note of a flute.

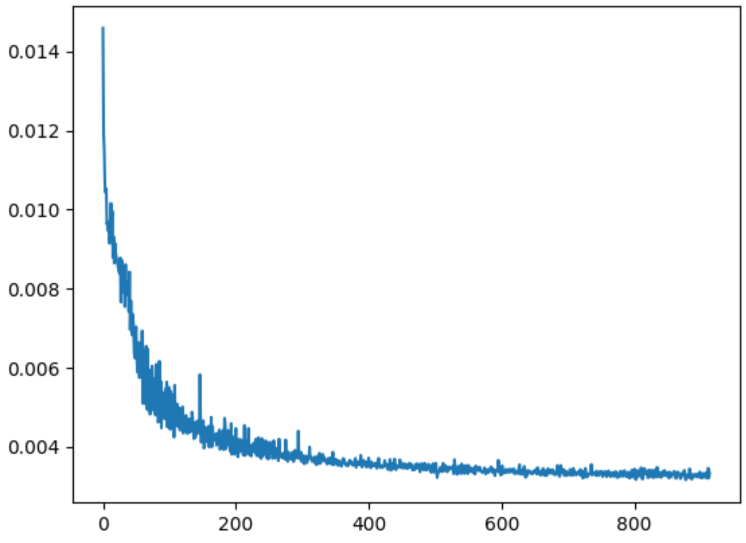


Figure 32: Loss function of the autoencoder.

5 CONCLUSION

The goal of this project was to succeed to build a neural network which was able to learn high pitched transformation. For that the architecture was inspired by Wavenet and transformed to answer the problematic by inserting global conditioning. The training dataset was constructed step by step to learn first how to transform sine waves, being careful about the length of the receptive field.

The receptive field which has the role of memory inside the network is most important variable. It is linked directly to the number of stacks and layers, showing the complexity of the network. But it shows also the number of samples taken in memory to predict the next one. Having causal convolutions inside layers, it forbids keeping long term structure which is important in a high pitched transformation to still have the same input features except the pitch. Indeed each prediction is the result based on the few previous samples.

NSynth allows to keep these temporal features by encoding each input as a bias and putting this bias as a global condition inside the Wavenet. This technic did not have time to be developed at the end and should be a good trail for future work.

This internship was very interesting and was seen as a new Machine Learning application. My previous project on playlist generation was very helpful to build and train a new neural network. It was also the opportunity to use the ssh protocol to launch simulations and to compute these simulations on the GPU Nvidia Titan X.

BIBLIOGRAPHY

[1] Takayuki Nakata. Cochlear implants and music. *Cholesteatoma and Ear Surgery: An Update*, page 155, 2013.

[2] Charles J Limb and Alexis T Roy. Technological, biological, and acoustical constraints to music perception in cochlear implant users. *Hearing research*, 308:13–26, 2014.

[3] Antoine Chaigne and Jean Kergomard. *Acoustique des instruments de musique*. Collection Echelles. Belin, 2008.

[4] Mark Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[5] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[6] Allen Huang and Raymond Wu. Deep learning for music. *arXiv preprint arXiv:1606.04930*, 2016.

[7] Google magenta. `https://magenta.tensorflow.org/`.

[8] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

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

[11] Waybackprop. `https://magenta.tensorflow.org/blog/2017/06/01/waybackprop/`.

[12] Jose Sotelo, Soroush Mehri, Kundan Kumar, Joao Felipe Santos, Kyle Kastner, Aaron Courville, and Yoshua Bengio. Char2wav : End-to-end speech synthesis. *Workshop track - ICLR 2017*, 2017.

[13] Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *CoRR*, abs/1601.06759, 2016.

[14] Sergei Turukin. Pixelcnn. http://sergeiturukin.com/2017/02/22/pixelcnn.html.

[15] Kundan Kumar. Pixelcnn code. https://github.com/kundan2510/pixelCNN.

[16] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. *CoRR*, abs/1704.01279, 2017.

[17] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016.