Master's Thesis : Audio frame reconstruction from incomplete observations using Deep Learning techniques

Auteur : Schils, Minh
Promoteur(s) : Embrechts, Jean-Jacques
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"
Année académique : 2019-2020
URI/URL : http://hdl.handle.net/2268.2/10138

Glossary

CNN A neural network with mainly convolutional layers. See section 1.2.1.2 and 1.2.2.

GAN Generative Adversarial Network. See section 1.2.3.

ICNN Inpainting Convolution Neural Network. See section 4.4.1.

IGAN Inpainting Generative Adversarial Network. See section 3.3.2.

LPC Linear Predictive Coefficient. See section 1.1.3.

MSE Mean Square Error. See section 1.8.

NN Neural Network. See section 1.2.1.2.

ODG Objective Difference Grade. See section 4.3.2.

ReLU Rectified Linear Unit. See section 1.2.1.

SNR Signal to Noise Ratio. See section 4.3.1.

STFT Short Time Fourier Transform. See section 1.1.1.

tanh The hyperbolic tangent. See section 1.2.1.

Université de Liège - Faculté des Sciences Appliquées
Politecnico di Milano - ISPG Lab

Audio frame reconstruction from incomplete observations using Deep Learning techniques

SCHILS Minh Cédric

Travail de fin d'études réalisé en vue de l'obtention du grade de master "Ingénieur Civil en Informatique - spécialisation en Intelligent system"

1st June 2020

Co-Promoteurs: EMBRECHTS Jean-Jacques, SARTI Augusto
Supervisor:COMANDUCCI Luca

Abstract

In this thesis, we tackle the problem of restoring an audio frame given the preceding and subsequent one, e.g. audio inpainting, and extend our proposed solution to the prediction of an audio frame given the last one. We consider frames of 64 and 128 milliseconds. The proposed solution combines a signal processing pipeline with a Generative adversarial network (GAN). Using as input the absolute value of the STFT of the surrounding frames, the network is able to retrieve the STFT magnitude corresponding to the gap frame. By applying the Griffin-Lim Algorithm, we are then able to estimate also the STFT phase and finally through the inverse STFT to reconstruct the missing audio frame. We compare our method, considering as baseline a Linear predictive coefficient (LPC) technique. The proposed solution shows encouraging results with respect to the baseline both for inpainting and prediction. It outperforms the baseline in term of Signal to noise ratio (SNR) on the magnitude spectrum and performs equally well or better in term of the Objective difference grade (ODG) which is a measure used tu assess the perceived audio quality. Since the phase of the STFT can be only approximately reconstructed through the Griffin-Lim Algorithm, the baseline shows better performances in terms of audio SNR. We further show the model generalization ability, by training and testing on two different types of music datasets.

Acknowledgements

First, I would like to thank Luca Comanducci for his availibility and guidance throughout all the thesis. Then, I would like to thank JJ Embrecht for his support and advices. I am also grateful to Augusto Sarti, who propose me the subject and to the ISPL lab for their welcome.

Contents

List of Figures

List of Tables

Introduction

Nowadays, it is common to listen to music on streaming either for album and live concert. However, the streaming can occasionally lose some packets and audio contents. In this context, the goal of this thesis is to explore and propose a way to recover loss data of music for a duration of around 64ms. In some cases, the missed data can be surrounded by a reliable data. Fill in the gap is called audio inpainting in the literature and is the problem tackled in this work. For short gaps, several algorithms were proposed: [2], [20], ... These algorithms are designed to recover an audio corrupted by clicks, clippings, or other distortions of short duration. Unfortunately, when the gap size increases, the performance of these algorithms dramatically decreases. In order to inpaint a longer gap, 64 ms for example, the author of [23] explores the use of a convolutional auto encoder architecture.

Our solution is designed to inpaint gaps of 64ms. Beside, we also adapt our model to audio prediction. Thus, even in the cases of a limited delay of the network, the listener can still enjoy the music without feeling any gaps. As far as we know, audio inpainting is a quite new research interest and no previous work has been done on predicting the next raw audio frame. However, two recent models called MuseNet [27] and MusicTransformer [14] were developed to predict the next token of music, in a MIDI like language.

In Chapter 1, the concepts needed to understand the thesis are explained. The Chapter 2 is a review of the state of the art of similar problems. In Chapter 3, we define the problem and describe our model. After that, in Chapter 4, we first detail our quality measures and compare our models to a baseline. Finally, we draw the conclusion and suggest tracks to improve the model.

Part I

State of the art and related work

Chapter 1

Background

1.1 Signal processing

In this section, we present two signal processing algorithms used in this thesis. The first one, the Short time fourier transform (STFT) is a well know and widely used tool in signal processing, to go from time-domain to the frequency domain and reversely. It allows to analyze local portion of the signal and thus analyze the frequency content in time. The second one, Griffin-Lim, is a phase recovery algorithm. Given the magnitude spectrum of a signal, it aims to estimate the phase of the complex spectrum.

1.1.1 Short Time Fourier Transform

The STFT is a fourier-related transform used to determine the sinusoidal amplitude and phase content of local sections of a signal. In practice, the long time signal is divided into frames of equal length on which the Fourier transform is applied. In the discrete time case, the STFT of a signal x is defined by:

$$STFT[x](m, f) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-jfn}, \tag{1.1}$$

where w is a window signal m the time and f the frequency. An STFT is provided as example in Figure 1.1

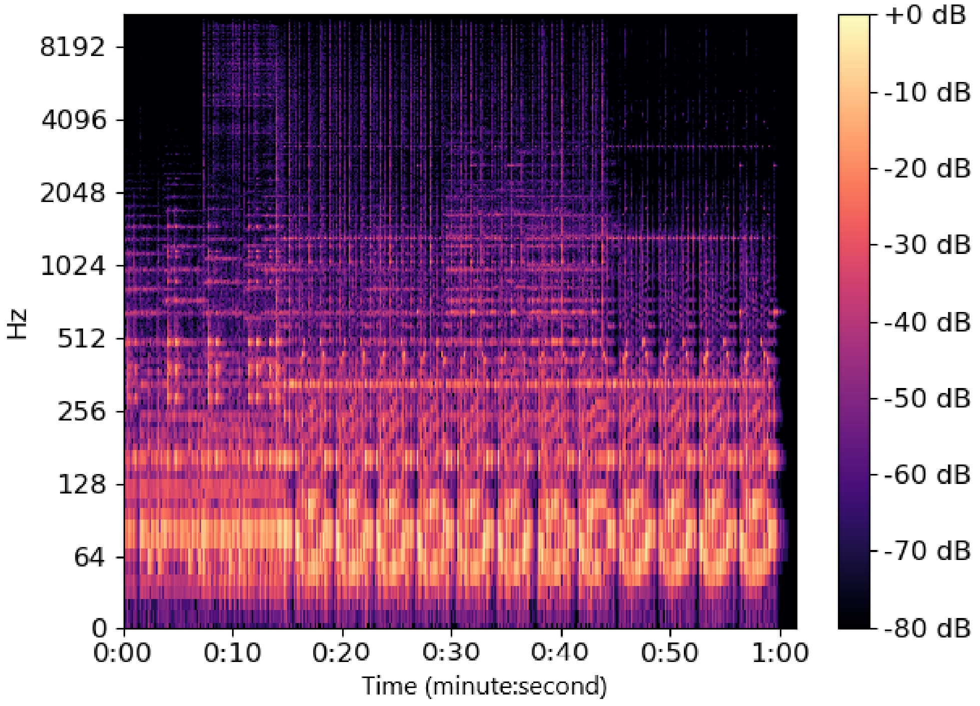


Figure 1.1: STFT magnitude example. The horizontal axis is the time in minute:second and the vertical one is the frequency in Hz

1.1.2 Griffin-Lim

The Griffin-Lim algorithm is able to reconstruct a signal given its magnitude spectrum by estimating the phase. The problem is expressed as finding a vector $x^* \in \mathbb{R}^L$ such that the magnitude of the STFT of x^* : $|Gx|$ is as close as possible to the given spectrum. Given a frame G and real positive coefficient s which is the amplitude of the STFT, x^* is the solution of

$$\min_{x \in \mathbb{R}^L} || \, |Gx| - s||. \tag{1.2}$$

The Griffin-Lim algorithm proceeds by projecting a signal iteratively onto two different set, $\mathcal{C}_1, \mathcal{C}_2$, where $\mathcal{C}_1 = \{c \mid \exists x \in \mathbb{R}^L : c = Gx\}$ and $\mathcal{C}_2 = \{c \in \mathbb{C}^{MN} \mid |c| = s\}$. The Griffin-Lim algorithm was first proposed in [7] and then improve in [29].

1.1.3 Linear predictive coefficient

A conventional linear filter approximates a sample x_n by a combination of the p past samples and is computed using finite impulse response. More formally

$$\hat{x}_n = \sum_{i=1}^{p} a_i x_{n-i}. \tag{1.3}$$

The error to minimize is the square error between the predictions and the signal:

$$e(x, \hat{x}) = \sum_{n=-\infty}^{\infty} (\hat{x}(n) - x(n))^2, \tag{1.4}$$

where a_i are the prediction coefficients, p is the order. The Linear predictive coefficient (LPC) finds these coefficients that minimize the error in the least square sense.

1.2 Deep learning

In this section, we present the main concept of deep learning used in this work. We first introduce the building block of a Neural network (NN), the neuron. Then we explain the principle of NN and how they are trained.

Finally, we will introduce more specific NN layers and architectures: Convolutional Neural network (CNN), transposed convolution and Generative adversarial network (GAN).

1.2.1 Neural network

1.2.1.1 A neuron and its activation function

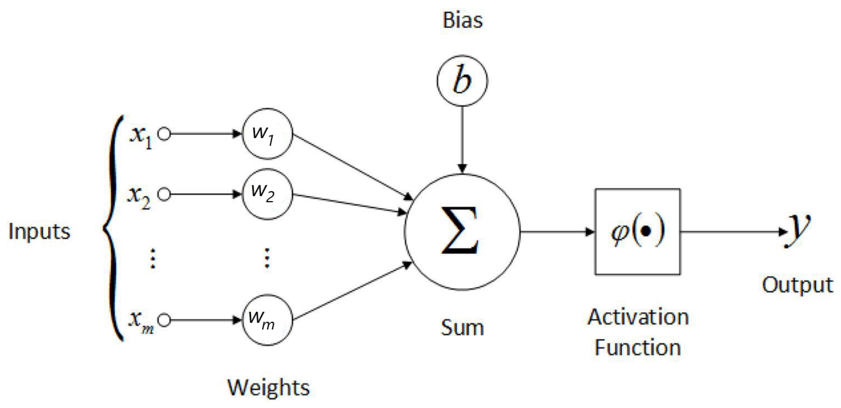


Figure 1.2: A neuron output a function φ of a linear combination of its inputs: $y = \varphi(b + \sum_{i=1}^{m} \omega_i x_i)$

The neuron is the basic unit processing of a NN. A neuron outputs a function of a linear combination of its inputs, as shown in Figure 1.2. The function is called the activation function. The weight of the linear combination , noted w_i, as well as the independent term b, called bias, are learned during training. Example of commonly used activation functions are tanh, ReLU and sigmoid. The tanh takes value between $[-1, 1]$ and is defined by

$$\varphi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{1.5}$$

The ReLU takes value between $[0, \infty]$ and is defined by:

$$\varphi(x) = max(0, x). \tag{1.6}$$

Finally, the sigmoid function takes value between $[0, 1]$ and is defined by:

$$\varphi(x) = \frac{x}{1 + e^{-x}}. \tag{1.7}$$

A draw of these activation function are shown in Figure 1.4.

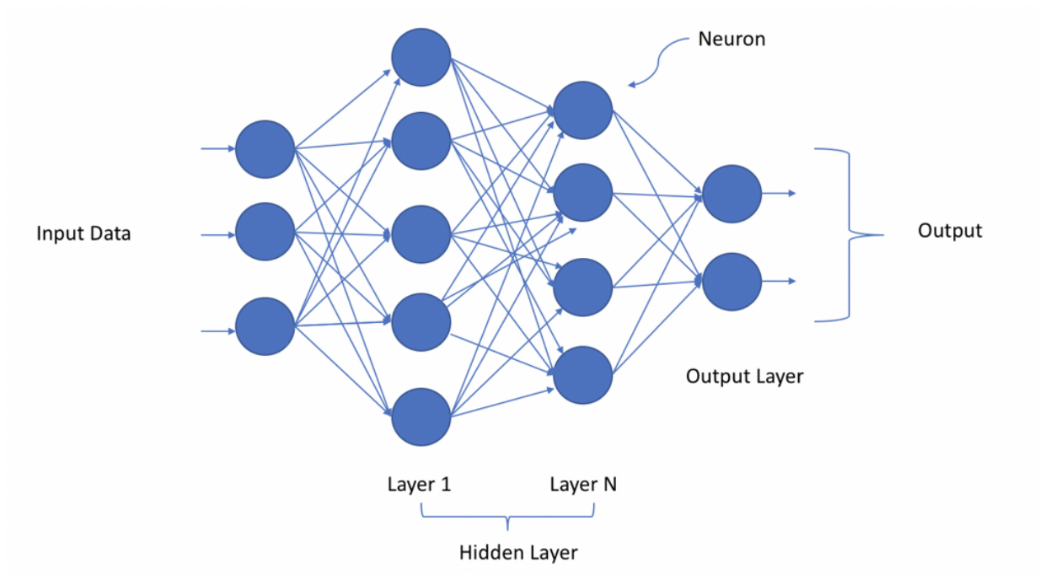


Figure 1.3: An example of a neural network

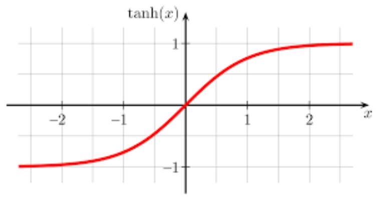
1.2.1.2 A Neural network and its goal

A NN is a computing system vaguely inspired by the human brain. It is a graph of connected neurons organized into layers. The first layer is called the input layer and the last one is the output layer. Each layer receives as input the previous layer output. The network can be viewed as a function $\hat{y} = f(x, \mathbf{W})$ where $\mathbf{W}$ is a matrix containing all the network parameters, i.e. the weight and bias of the neuron. An example NN is provided in Figure 1.3.

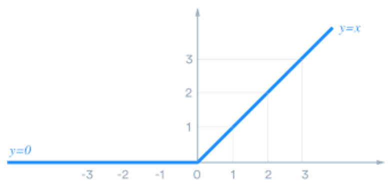
To train a neural network, a dataset D of pairs (x, y) is needed. The goal of the neural network is to minimize a loss $L(\hat{y}, y)$ between the predicted output $\hat{y}$ and the real one y. This optimization problem is resolved, thanks to the back-propagation algorithm, that updates the weights of the NN according to the gradient of the loss with respect to the network weights. Consequently, the loss function has to be derivable. A commonly used loss function in regression is the Mean Square Error (MSE), which is defined by:

$$L(\hat{y}, y) = \frac{\sum_{(x,y)\in D} (\hat{y} - y)^2}{|D|}, \tag{1.8}$$

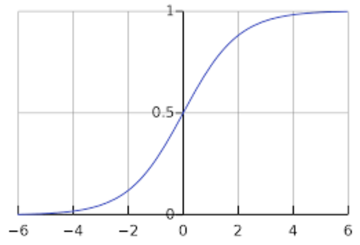
where $|D|$ denote the cardinality of the dataset D, $\hat{y} = f(x)$ denotes the



(a) Hyperbolic tangent (tanh)



(b) Rectified Linear Unit (ReLU)



(c) Sigmoid

Figure 1.4: Common activation functions

network prediction.

1.2.1.3 Neural network training

A NN can be viewed as a function $\hat{y} = f(x, \mathbf{W})$ where $\mathbf{W}$ is a matrix containing all the network parameters, i.e. the weight and bias of the neuron. A NN can be evaluated with a dataset consisting of input/output pairs, and a loss function L that measures the distance between the desired output y and the network output $\hat{y}$. The training goal consist of finding the parameters $\mathbf{W}$ that minimize the average error L on the dataset. To do so, the network parameters are randomly initiated at first and then are iteratively updated in the direction that improves the network loss. More formally, the average loss is:

$$L_{avg}(\mathbf{W}) = \frac{\sum_{(x,y)\in D} L(f(x, \mathbf{W}), y)}{|D|}. \tag{1.9}$$

The derivative of L with respect to $\mathbf{W}$ is noted $\frac{\partial L}{\partial \mathbf{W}}$. The parameters are then updated following:

$$\mathbf{W} = \mathbf{W} - \gamma \frac{\partial L(\mathbf{W})}{\partial \mathbf{W}}, \tag{1.10}$$

where γ is called the learning rate. The learning rate is often between 0 and 1. A too large learning rate may lead to an unstable training and oscillates around a minimum while a too small learning rate will take longer to converge and is more likely to be stuck in a local minimum. All the derivative $\frac{\partial L}{\partial \mathbf{W}}$ computation can be efficiently perform thank to the back-propagation algorithm detailed in [6]. In brief, this algorithm recursively applies the chain rule $\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$ from the output layer to the input one.

1.2.2 Convolutional Neural network

A CNN is a network architecture based on convolution layers. It is mainly used for extracting image features.

1.2.2.1 Convolutional layer

A convolutional layer is commonly used as a feature extractor in image processing. It performs the convolution operation on its inputs and pass the result to the next layer. Each filter (also named kernel) is convolved with the input image, resulting in an activation map. Let us consider the 2D case. The convolution operation is then defined as:

$$S_{i,j} = B_{i,j} + \sum_{m=0}^{w-1} \sum_{n=0}^{h-1} I_{n+i,m+j} K_{n,m} \ , \tag{1.11}$$

where S is the result of the convolution of the input image I of size $W \times H$ with the filter K of size $w \times h$, i and j are index of the output matrix of size $(H - h + 1) \times (W - w + 1)$. More visually the convolution consists to slide a kernel on the input image. For each kernel position on the input image, the sum of the element wise multiplication between the kernel and the input image is done. Finally the bias is added to the result.

A convolutional layer has commonly the following hyper parameters

- **Filter size** is the dimension of the filter, the w and h of the formula 1.11
- **Stride** is how much the kernel slide on the input image. The formula in 1.11 is only correct for a stride of $(1, 1)$. For a stride of $(s1, s2)$, the formula become $S_{i,j} = B_{i,j} + \sum_{m=0}^{w-1} \sum_{n=0}^{h-1} I_{n+i*s1,m+j*s2} * K_{n,m}$

- **Depth** is the number of filters in the layer. For each filter, the convolution operation defined at 1.11 is performed and each output are stacked in a new dimension. The output has thus a dimension of $(H-h+1)\times(W-w+1)\times d$, where d is the depth. The formula can be updatet to $S_{i,j,d} = B_{i,j,d} + \sum_{m=0}^{w-1}\sum_{n=0}^{h-1} I_{n+i,m+j,d} * K_{n,m,d}$.

- **Padding** Padding consists to enlarge the input matrix so that the output will have the size needed. Padding is usually done by adding zeros at the input image borders.

An illustration of a convolution is shown in Figure 1.5. In this example, $w = h = 3$, $W = H = 7$, $s1 = s2 = 1$, $d = 1, B = 0$. There is no padding nor bias nor stride.

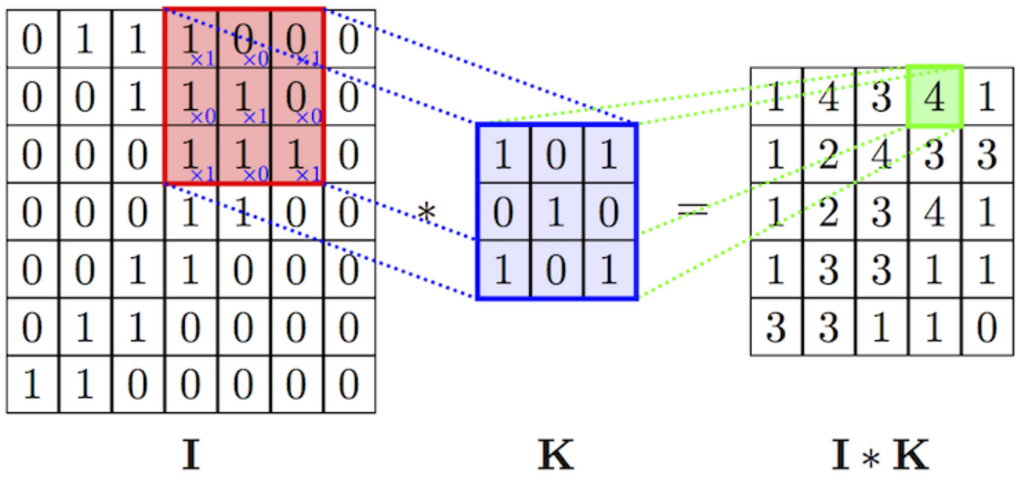


Figure 1.5: A convolution illustration. The input image is on the left, the filter in the middle and the output feature map on the right

1.2.2.2 Transposed convolutional layer

A transposed convolutional layer is usually used to up-sample. Similarly to the convolutional layer, it is made of n filters, which are $M \times N$ matrix. Lets consider only one filter, $n = 1$ and the input image has a size of $W \times H$. Each pixel of the input image multiplies the filters, resulting into WH square of size $M \times N$. Then the WH square are placed according to the pixel location and each superimpose case are summed up. An illustration of the transposed convolution is shown in Figure 1.6.

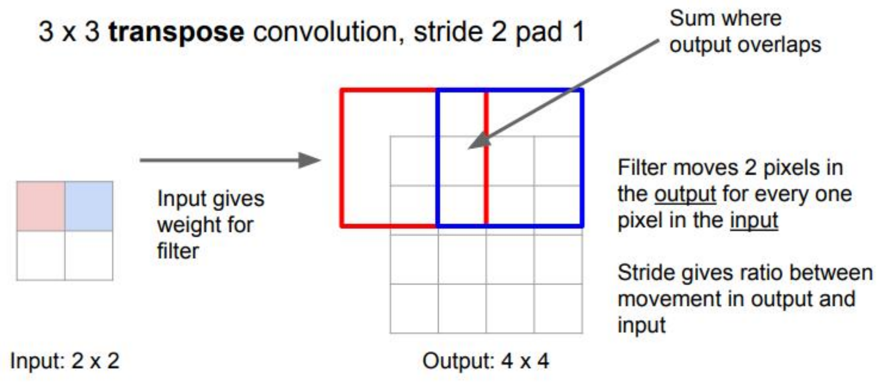

3 x 3 **transpose** convolution, stride 2 pad 1
Sum where output overlaps
Input gives weight for filter
Filter moves 2 pixels in the output for every one pixel in the input
Stride gives ratio between movement in output and input
Input: 2 x 2
Output: 4 x 4

Figure 1.6: A transposed convolution illustration

1.2.3 Generative Adversarial Network

A GAN is usually used in generation tasks. It has the advantage to be able to generate more 'plausible' images than a network trained with a mean squared error, as the later one tends to produce blurred samples. A GAN is a network architecture made of two main component: a generator and a discriminator. The goal of the generator is to generate plausible samples from noise and fool the discriminator, while the goal of the discriminator is to detect whether a sample is fake. Thus, the two component of this network architecture have antagonist objective, making the training of such network often tricky. Let define the following loss:

$$V(Gen, Dis) = E_x[log(Dis(x))] + E_z[log(1 - Dis(Gen(z)))], \qquad (1.12)$$

where Dis is the discriminator, Gen the generator, $Dis(x)$ is the discriminator estimation that the real data x is a real instance, E_x is the expectation over all real instance, E_z is the expectation over all real noise, $Gen(z)$ is the generator output given noise z and $Dis(G(z))$ is the probability estimation of the discriminator that the fake instance is real. For a fixed generator Gen, the discriminator tries to maximise the loss in 1.12 as it aims to be able to

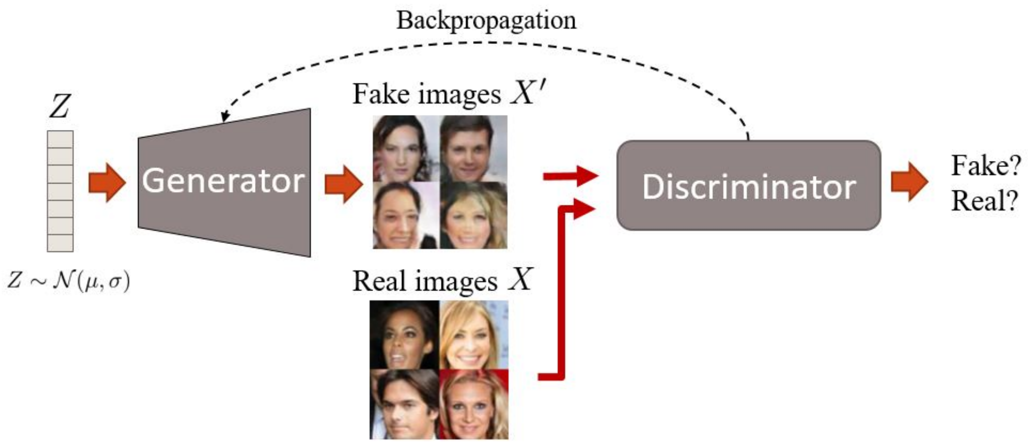


Figure 1.7: A schema of a gan network.

classify correctly a sample as fake or real. For a fixed discriminator Dis, the generator tries to minimize the loss at 1.12. Training consist of finding the generator Gen^* that minimize the loss 1.12 when Dis is the best discriminator, i.e. the one that maximize the loss 1.12. This is formally written in the following equation:

$$Gen^* = \min_{Gen} \max_{Dis} V(Gen, Dis). \tag{1.13}$$

The generator and discriminator are engaged in a zero sum game against each other. It was theoretically proven in [12] that the global optimum of 1.13 happens when the probability distribution defined by the generator is the same than the one of the real data.

To train this model, the discriminator and the generator are alternatively updated. The discriminator is first updated on a mini batch of data in the direction of the ascending gradient of $V()$: $Dis = Dis + \gamma \frac{\partial V(Dis,Gen)}{\partial Dis}$, where γ is the learning rate. Similarly, the generator is then updated in the direction of descent gradient: $Gen = Gen - \gamma \frac{\partial V(Dis,Gen)}{\partial Gen}$. From a practical point of view, it is important during the training that neither the generator nor the discriminator outperforms the other. Otherwise, the gradient for updating will be to low or meaningless. An illustration of this architecture is shown in Figure 1.7.

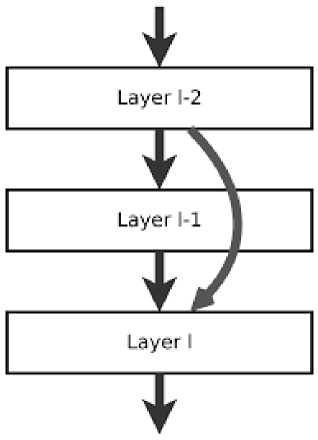


Figure 1.8: A skip connection illustrion. The data going throught the two arrow can either be concatenate or added

1.2.4 Skip connection

During back propagation, the gradient has to go through all the layers before reaching the first one. At each layer, it is multiplied by a number usually smaller than one. Thus if the network is too deep, the gradient at the first layer will be too low or even null. This problem is know as 'vanishing gradient'. A skip connection addresses this problem by providing short-cut between two non adjacent layers. The data coming from the shortcut and the normal path are then either added or concatenate. Thus the gradient can skip some layers and avoid some multiplication by a number smaller than one. An illustration of a skip connection is shown in Figure 1.8.

1.2.5 Binary Cross Entropy

The binary cross entropy is a loss function used in classification task involving two classes. It is defined by the following equation:

$$H(y, \hat{y}) = y \quad log(\hat{y}) + (1 - y) \quad log(1 - \hat{y}), \tag{1.14}$$

where $y \in \{0, 1\}$ is the class label and $\hat{y} \in \;]0, 1[$ is the estimated probability that the sample belong to class 1.

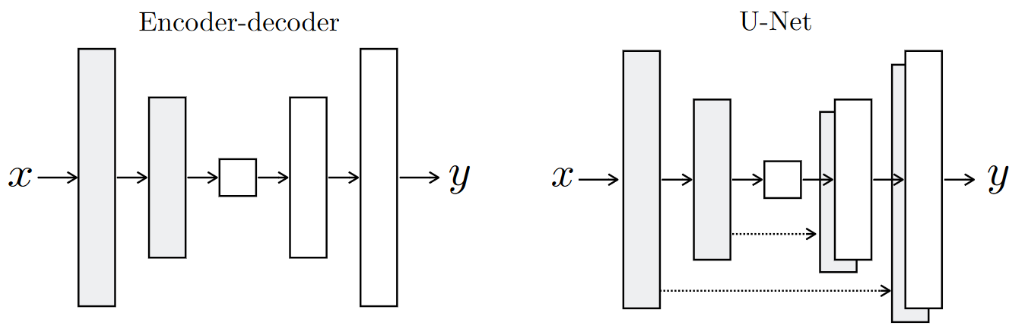


Figure 1.9: U-net architecture. On the left auto-encoder architecture. On the right U-net architecture, autoencoder with skip connection between mirrored layer. Schema taken from [15]

1.2.6 Pix2pix

Pix2pix[15] is an image to image translator. Given a conditioning input image, it generates a new image. A pix2pix model is trained on a dataset of pair of conditioning image and target image. Pix2pix is a GAN composed of a patch discriminator and a U-net generator. Unlike the classical GAN described in the section 1.2.3, the variable z is not taken from noise, but is the input image. A U-net is a encoder-decoder network with skip connection between mirrored layer, as shown in Figure 1.9. In a patch discriminator, the discriminator doesn't try to class the whole image as fake or real but rather try to classify each patch of the image. Last trick, the generator loss doesn't only take into account the discriminator performance but also take care of the reconstruction error between the generated image and the target image. Only used a reconstruction loss leads to blurry image while only used the adversarial loss leads to sharper image but with unwanted visual artifacts. An overview of the model is shown in Figure 1.10.

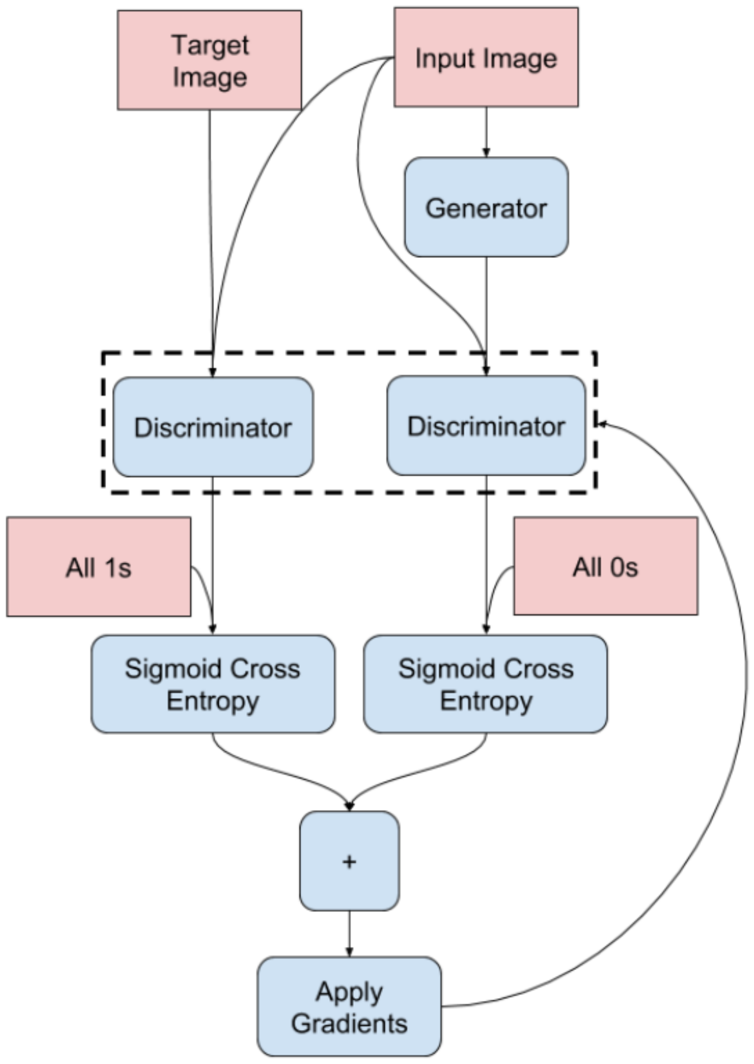


Figure 1.10: Pix2pix overview. The generator loss is made of two part: the adversarial loss which is a sigmoid cross entropy and the reconstruction loss which is a mean absolute error [15]

Chapter 2

Related work

In this section, a review of the state-of-the-art and main works in the field of audio and music generation will be done. The first part of the review is dedicated to the generation of raw waveform. The second one is related to the audio inpainting task, i.e, filling an audio gap given the previous samples and the next ones. The last part is concerned with music prediction. All of these subjects are linked to this thesis topic, as we try to generate an audio frame of a music, given the previous and optionally the next frame.

2.1 Audio generation

In the audio generation task, one tries to generate the waveform of an audio. This can be done directly in time domain. The problem can also be resolved in another domain, such as the time-frequency one, and then map to the time domain. The first subsection describes models that use the first approach, while the second subsection models follow the time-frequency approach.

2.1.1 Audio synthesis in time domain

2.1.1.1 Audio synthesis in time domain using GAN

In the paper [8], the authors applied GAN to generate directly raw audio waveforms on a dataset of bird songs, spoken digits, drum effects, ... The generator is made of one-dimensional transpose convolution layers while the discriminator is composed of 1D convolution layers. For the spoken digits dataset, human subjects were able to recognize the number in the generated

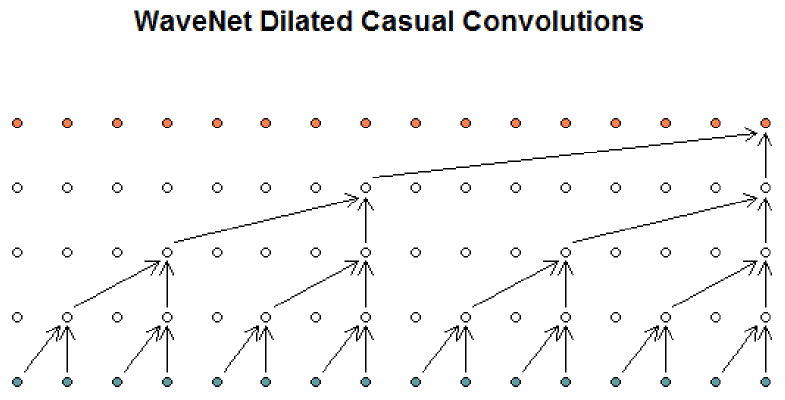


Figure 2.1: Illustration of dilated convolution layers. The bottom layer does not have any dilatation, it is an usual 1D convolution. All the other layers are dilated convolution, one input over 2,4,8 is skipped. [36]

sample in 58% of the cases. On the real data, the recognition percentage reached 95%.

2.1.1.2 Audio synthesis in time domain using an autoregressive model

In an autoregressive model, the output of a sample x at time t depends of the output at time $t-1, t-2, ...$

Recently, two different autoregressive models, WaveNet [36] and [25] are developed and applied on waveform.

The first one, WaveNet is based on dilated convolution layers. A dilated convolution layer is a normal convolution layer where the filter is applied on a larger area than the filter size, by skipping input values with a certain step. The length of the step is called the dilation factor. The dilated convolution layers enable the model to exponentially increases the receptive field of the network without increasing too much the number of parameters of the model. An illustration of a dilated convolution layer is shown in Figure 2.1. The

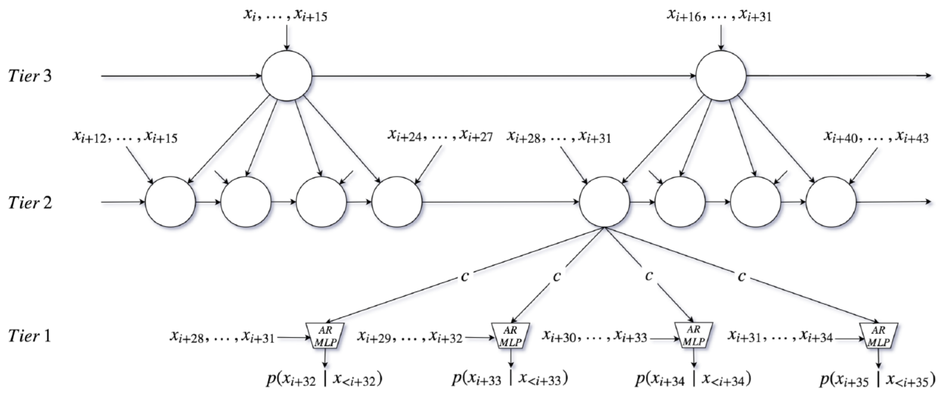

$x_i, \ldots, x_{i+15}$
$x_{i+16}, \ldots, x_{i+31}$
Tier 3
$x_{i+12}, \ldots, x_{i+15}$
$x_{i+24}, \ldots, x_{i+27}$
$x_{i+28}, \ldots, x_{i+31}$
$x_{i+40}, \ldots, x_{i+43}$
Tier 2
c
c
c
c
Tier 1
$x_{i+28}, \ldots, x_{i+31}$
AR MLP
$x_{i+29}, \ldots, x_{i+32}$
AR MLP
$x_{i+30}, \ldots, x_{i+33}$
AR 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 2.2: Illustration of sampleRNN tier organization. The 3rd tier summarizes the past 16 samples and conditions the 2nd tier. The 2nd tier looks at both the last 4 samples and the summary provided by the 3rd tier and conditions accordingly the first tier. The first tier looks at the 4 past samples and the provided conditioning to output the next sample. Schema taken from [25]

wavenet model also uses residual and skip connection. The model can also be conditioned on some other input, for example a text to read. Wavenet is currently used in production by google on its products.

The second one, sampleRNN, is based on recurrent layer working at different time scales organized into hierarchical tier, as shown in Figure 2.2. In that figure, the third tier looks at the past 15 samples and summarizes them in a vector c , which conditions the second tier. The second tier only looks at the past 4 samples and the summary given by the third tier, then conditions the last tiers. The last tier only looks at the past 4 samples and the conditioning vector given by the second tier. Then, it generates the audio sample by sample.

Unlike WaveNet, sampleRNN is able to generate audio in real time. These two models are sequential and can only generate one sample at a time. These models are able to predict an audio frame given the previous samples but only one sample at a time.

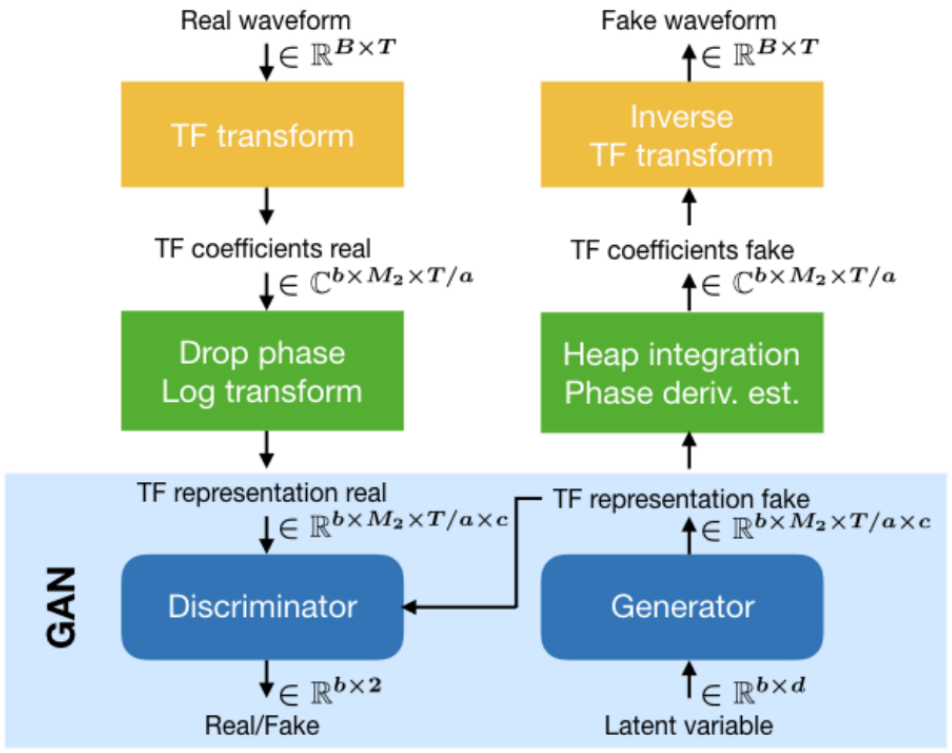

Real waveform
$\in \mathbb{R}^{B\times T}$
TF transform
TF coefficients real
$\in \mathbb{C}^{b\times M_2\times T/a}$
Drop phase
Log transform
TF representation real
$\in \mathbb{R}^{b\times M_2\times T/a\times c}$
Discriminator
$\in \mathbb{R}^{b\times 2}$
Real/Fake
Fake waveform
$\in \mathbb{R}^{B\times T}$
Inverse
TF transform
TF coefficients fake
$\in \mathbb{C}^{b\times M_2\times T/a}$
Heap integration
Phase deriv. est.
TF representation fake
$\in \mathbb{R}^{b\times M_2\times T/a\times c}$
Generator
$\in \mathbb{R}^{b\times d}$
Latent variable
GAN

Figure 2.3: Tifgan overview. The GAN model focus on generating the spectrum of the signal. Then, the phase is estimated thanks to the heap integration algorithm and the audio is finally obtain with an inverse fourier transform. [24]

2.1.2 Audio generation model based on spectrum

Recently, two models made of convolution layers and working with a spectrum representation have been developed. Both model, gansynth [10] and tifgan [24] are GAN, as suggested by their name. They have in common to output the spectrum of the generated audio, and being conditioned on a label.

The tifgan model works with magnitude spectrum and uses a phase estimation algorithm and the inverse fourier transform to generate an audio of a spoken number. The model deals with log magnitude spectrums, as shown in

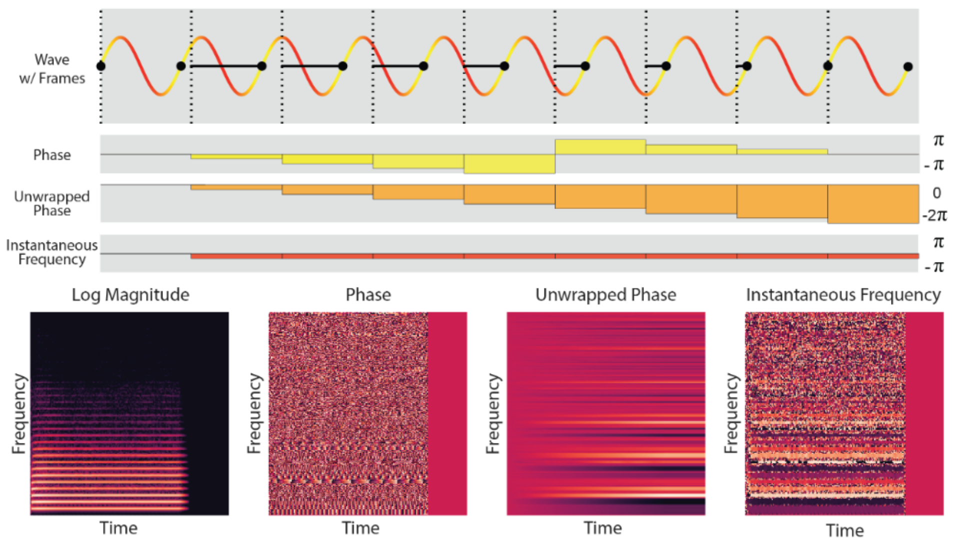


Figure 2.4: Instantaneous phase illustration. The instantatneous frequency is the time derivative of the unwrapped phase [10]

Figure 2.3. The phase is estimated thanks to the heap integration algorithm. The fake waveform is obtained by a final inverse STFT.

The gansynth model generates an audio of a single note given the desired pitch, velocity and instrument. Gansynth is trained using progressive training [18]. The principle of progressive training is to progressively increase the image resolution in the generator and discriminator by adding layers. This incremental nature allows the model to first discover large-scale structure of the image distribution and then focus on details, rather than learning all the scales simultaneously. Unlike tifgan, gansynth represents both the phase and amplitude of a spectrum. The phase is not directly feed into the network, it is first unwrapped and then derived in time, yielding to the instantaneous frequency. An illustration of the phase representation is shown in Figure 2.4

2.2 Audio inpainting

The term audio inpainting describes a large class of inverse problems in audio processing. The general assumption is that the audio are represented in a domain, such as the time domain or time-frequency domain, into frames. Some of these frames are missing or corrupted. The inpainting task consists to recover these frames from the one immediately preceding and following. We will refer as short gap when the corrupted/missing data doesn't exceed a few milliseconds. In contrast, when the missing data has a length exceeding a few hundreds of milliseconds, we will refer to it as long gap. In the middle case, when the missing data has a length of a few dozens of millisecond, we will refer to it as a medium gap.

2.2.1 Inpainting short gap

Short corrupted gaps can occur in the case of clipping or clicking. Clipping is a form of distortion that can occur when an amplifier is overdriven and attempts to output a voltage or current beyond is maximum capability. The clipping phenomenon is illustrated in Figure 2.5. Clicking occurs when a sudden noise is added to the signal, as illustrated in Figure 2.6

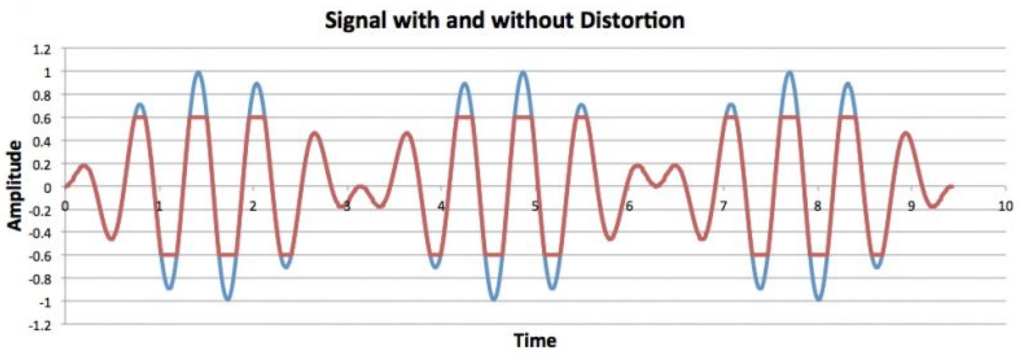
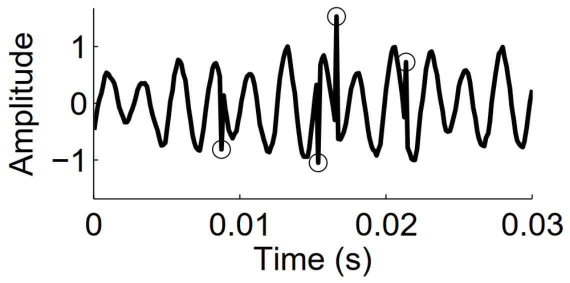


Figure 2.5: Clipping illustration. The red signal is the corrupted one while the blue signal is the original one.

One of the first attempt to audio perform audio inpainting is described in [16]. The signal is modelled as an autoregressive model. The idea is to find the parameters of the autoregressive model and the value of the unknown



(a) Speech signal corrupted by clicks (circles).

Figure 2.6: Clicking illustration. Taken from [2]

samples such that the restored audio is accurately reconstructed by the autoregressive model.

The study presented in [2] proposes a solution to audio inpainting based on a orthogonal matching pursuit algorithm [22]. In this study, the location of the corrupted data is assumed to be known and are treated as missing. It uses sparse representation modelling on audio frame. Sparse audio modelling approximates a signal by a weighted sum of 'atoms' that are taken from a finite ensemble of atoms referred as dictionary. These atoms are the column of the dictionary. Sparse representation modelling aims at finding the sparsest set of coefficient that reconstruct the signal. In the proposed solution, the signal is firstly decomposed into overlapping frames, then the restoration problem is formulated as an inverse problem per each audio frame. Each inverse problem is resolved using the matching pursuit algorithm with a gabor or a discrete cosine transform dictionary. The proposed method works well for short duration gap. However when the gap length increases, the quality of the reconstruction decreases, as shown in Figure 2.7.

The use of sparse model for audio inpainting short gap or denoising short burst of noise has already been extensively explored: in [35] the authors try to find a suitable similarity measure between sparse representation of audio frame which may content unknown sample, [20] is concerned with audio declipping using sparse model, in [26] and [21] the authors are concerned with audio inpainting using a sparse model. Finally, in [11] the authors are concerned by removing white noise.

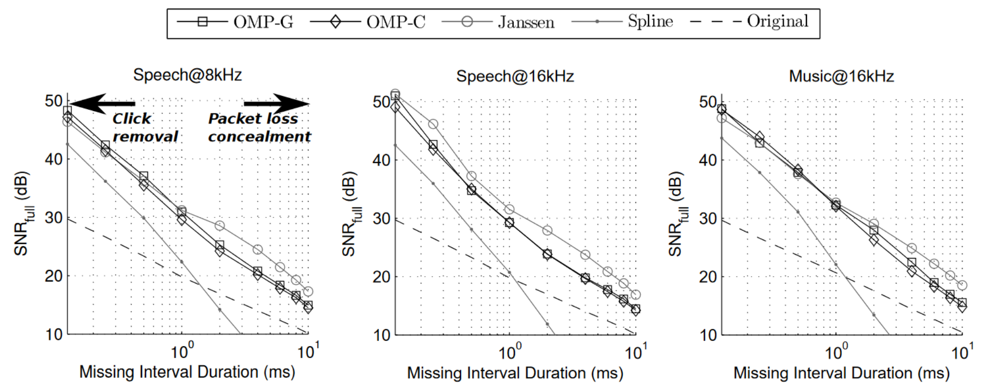


Fig. 2. Performance of inpainting algorithms as a function of the duration of missing intervals for each dataset (subfigures). The missing intervals were generated periodically every 100ms (a total of 50 equal duration missing intervals per signal).

Figure 2.7: Performance of the algorithm proposed in [2]

2.2.2 Inpainting medium gaps

In [23], the author tries to fill in a gap of 64 ms given the 128ms before and after the gap. It addresses the problem by means of a convolutional neural network applied on either the complex spectrum or on the magnitude spectrum. The paper shows that for a dataset made up of a single note, there is no need for a neural network, a linear tool like the lpc is enough. However, for a dataset made of music, the neural network outperforms the lpc based method, even if there is still room for improvement. Another interesting conclusion emerges from the comparison of the complex network and the magnitude network. It appears that use the convolutional network to only predict the magnitude of the spectrum and then use the heap integration algorithm leads to better results than using the convolutional network to also predict the phase. Last result, even if the network is trained for gap of 64ms, it can be used to predict gap of 48ms without performance loss.

2.2.3 Inpainting long gaps

In the case of long gap, most inpainting methods tends to look for repetitions and determine the most promising segment to fill in the gap, like in [28] and [3]. These methods do not pretend to accurately recover the missing samples but rather to fill in with plausible one.

The method presented in [28] addresses the problem of filling a long gap of 1 second, considering a whole music piece. The reliable data is analysed to detect spectro-temporal similarities resulting in a similarity graph representation. Inpainting of the lost data is then achieved by determining which other segments of the data can substitute the gap and is smoothly inserted in it.

The algorithm described in [3] is designed to work with speech and recover data from packet loss. The speech streaming is divided into overlapping frames of constant length, called AB. When there is no packet loss during the whole frame, the AB is an example. Otherwise, it is a query AB. For each query, the most suitable example is picked. An example is said to be suitable if the intact portion of the query has similar features than the example query and if the resulting AB sequence has an high probability to occur. The AB sequence is modelled as a Markov chain. Then pitch and gain modification, offset tuning and cropping are applied to smoothly insert the candidate segment into the gap.

2.3 Music prediction

In this task, a music is represented as a sequence of tokens. The models goal is to find the next token given the previous one. The piano roll representation can be viewed as a 2D graph, where the horizontal axis is the time and the vertical axis is the pitch. When a note is played, an horizontal line is drawn in the graph, starting at the moment the note is played and ending when the note stop. This representation is illustrated in Figure 2.8. The token representation is often derived from a piano roll representation in this task. Two recently developed model, MuseNet [27] and MusicTransformer [14], use a similar representation to piano roll.

Music transformer [14] and Musenet [27] are two models that deal with symbolic representation of music and are able to predict the next token given the previous one. They can also generate a coherent sequence of token without any conditioning. Both models are based on a transformer [37] architecture.

A Transformer model is based on attention layer. An attention layer first transforms the input into three vectors named query, key, and value. Then the query of each input is dot multiplied with the key of each input. All the result will be feed into a softmax function to get a coefficient such that they

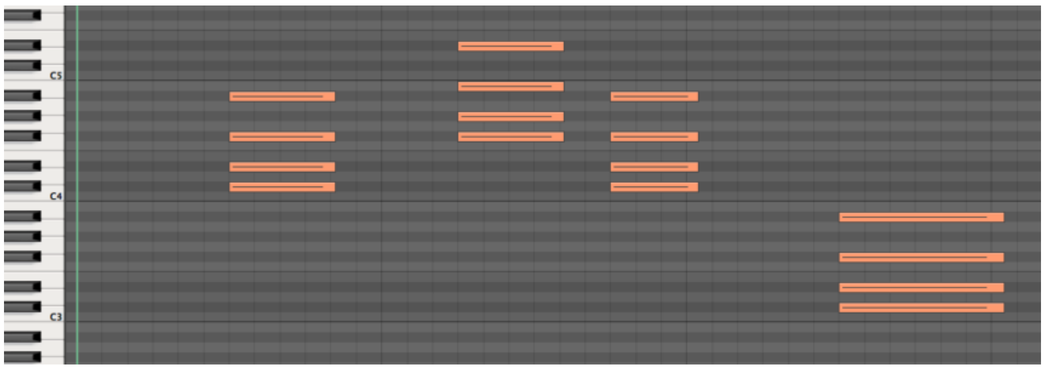


Figure 2.8: Illustration of a piano roll representation. the horizontal axis is the time and the vertical axis the pitch.

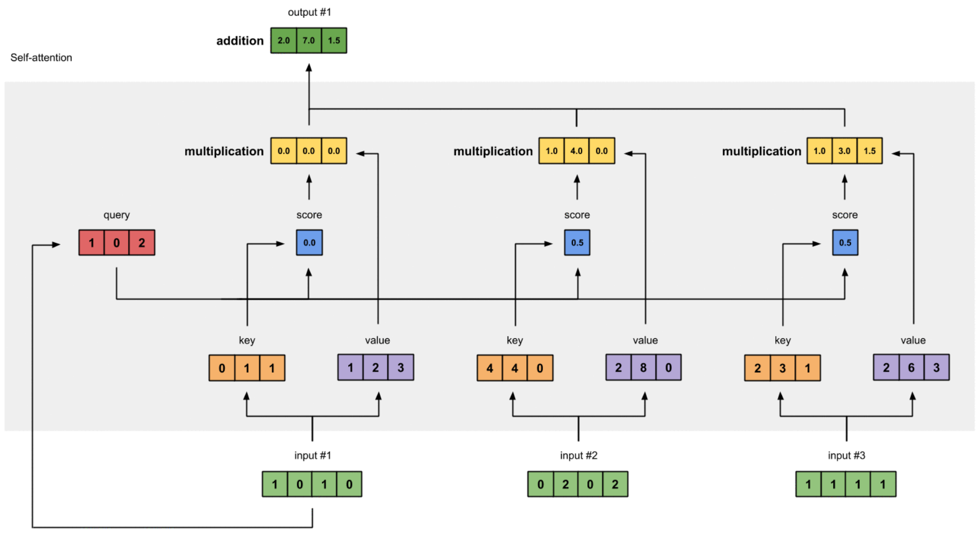


Figure 2.9: Self attention illustration. Each input is projected into 3 vectors named query key and input. Then the dot product of the query and key are applied and feed into a softmax function, to get the score. Then the input value is multiplied by a score. The scores sum up to one. Finally each multiplied value are summed to get the first output. The same processing step is done with the query of the second input to get the second output, as well as with the query of the last input. Schema taken from [1]

sum up to one. Then each value is multiplied by the corresponding coefficient. The idea behind such process is that each input value will be weighted by the importance it has with respect to all the input. An illustration of a self attention layer is available at Figure 2.9.

2.3.1 Music transformer

The paper [14] use a slightly different transformer than the original one present in [37]. It uses relative distance encoding instead of absolute one, like in [31] but improves the memory requirement of the relative encoding computation of [31]. This improvement enables the model to be trained on sequence of 2048 token, yielding to a good consistency up to one minute. Music transformer, focus on piano and is trained on the [13] MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) dataset.

2.3.2 Musenet

The work described in [27] uses the same technology than GPT-2, a large scale transformer, described in [30] and initially trained for language modelling. Musenet can be condition on instrument and on previous tokens. Musenet is able to combine 10 instruments on several styles and is trained on several and various MIDI dataset such as MAESTRO, ClassicalArchive, Bitmidi.

Part II

Proposed solution

Chapter 3

Model description

Two similar problems were considered: audio inpainting and prediction. The first problem consists of inferring an audio frame given the surrounding ones while in the second problem, only the audio frame before the target frame is given. An illustration of the audio inpainting task and the audio prediction task is given in Figure 3.1.

3.1 Audio inpainting

The goal of audio inpainting is to fill a gap in an audio signal given the next and previous frame. A frame is defined as a set of consecutive samples. Let us define the gap size in seconds as T_g corresponding to N_g samples and T_p,T_s and N_p, N_s as the previous and subsequent frames length in seconds and samples, respectively. Let x be the audio signal. Let us call the previous frame, gap frame and subsequent frame by $\mathbf{x}_p$, $\mathbf{x}_g$ and $\mathbf{x}_s$. The audio inpainting problem, can be formalized as:

$$\mathbf{x}_g = f(\mathbf{x}_p, \mathbf{x}_s). \tag{3.1}$$

Inpainting consists in finding a good approximation of $\mathbf{x}_g$, noted $\hat{\mathbf{x}}_g$, given $\mathbf{x}_p$ and $\mathbf{x}_s$, by modelling the function f.

3.2 Audio frame prediction

The goal of audio frame prediction is similar to the one of inpainting described in the last section, excepts that $\mathbf{x}_s$ is not provided any more to the model.

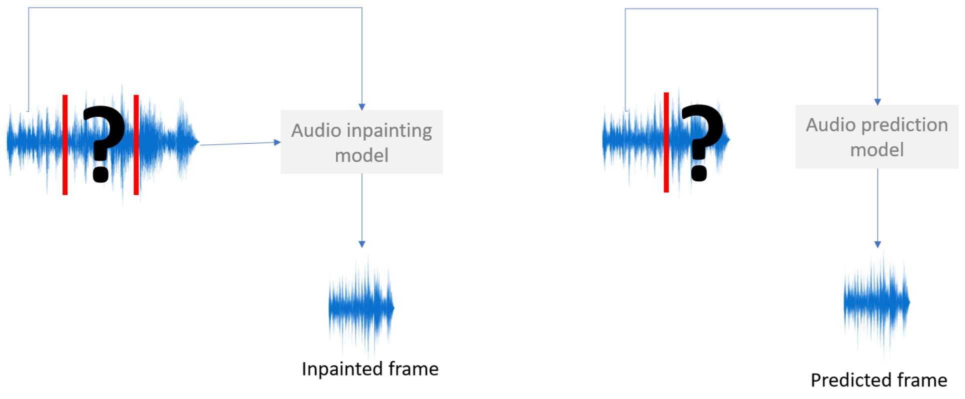


Figure 3.1: Audio inpainting vs audio prediction

More formally, it aims to model the function g in the following equation:

$$\mathbf{x}_g = g(\mathbf{x}_p). \tag{3.2}$$

3.3 Deep learning models

We can divide the audio inpainting process into three main parts. The first part computes the spectrum and feeds it into the NN. The second part is the NN who estimates the magnitude spectrum of $\mathbf{x}_g$. The last part estimates the audio signal from the STFT magnitude output of the network. This task is performed, first by reconstructing the STFT phase, through the Griffin-Lim algorithm explained in Sec 1.1.2 and then by applying the Inverse Short Time Fourier Transform (ISTFT) An overview of the sound reconstruction process is shown in Figure 3.2

3.3.1 Processing

3.3.1.1 Pre-processing

The audio signal is first randomly trimmed to a length of $T_P + T_G + T_N$ ms. Then, it is split into $\mathbf{x}_p$, $\mathbf{x}_g$ and $\mathbf{x}_s$. Afterwards, the STFT of $\mathbf{x}_p$, $\mathbf{x}_g$ and $\mathbf{x}_n$ are computed, with the parameters in Table 3.1. We define $\mathbf{p} = |STFT(\mathbf{x}_p)|$, $\mathbf{g} = |STFT(\mathbf{x}_g)|$ and $\mathbf{s} = |STFT(\mathbf{x}_s)|$. Finally, in the case of

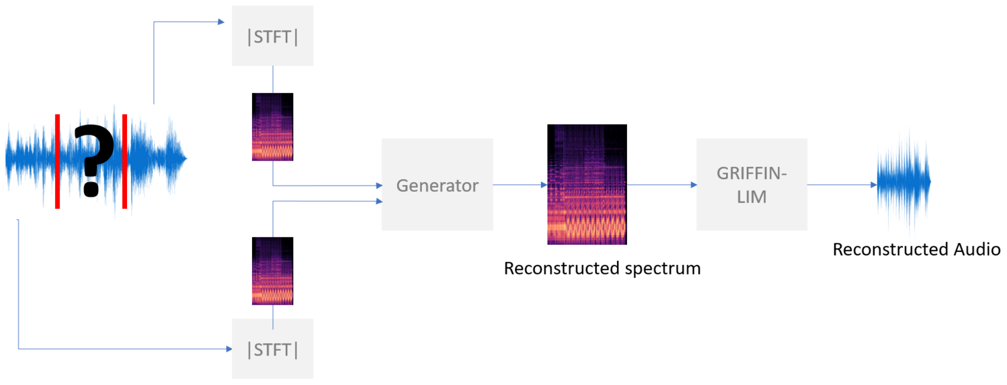


Figure 3.2: Overview of the sound reconstruction. The magnitude of the STFT of the previous frame and the subsequent frame are fed into the generator. After that, the generator outputs the reconstruction of the magnitude of the STFT of the gap frame. Then, the Griffin-Lim algorithm is applied on the generator output to estimate the phase. Afterwards, the complex STFT of the gap frame can be reconstructed from its magnitude and phase estimate. The audio is finally obtained by inverting the STFT

.

hop size	256
window size	1024
window function	hann window
fft length	1024
zero padding left	512
zero padding right	512

Table 3.1: STFT parameter

audio inpainting, the NN is fed with $\mathbf{p}$ and $\mathbf{s}$ as input and $\mathbf{g}$ as target. In the case of prediction, the NN is only fed with $\mathbf{p}$.

3.3.1.2 Post processing

In order to estimate the phase from the magnitude spectrum, 100 iterations of the Griffin-Lim algorithm are performed. Once the phase estimation is finished, the ISTFT is performed. The parameters of the ISTFT are the same than of the STFT shown in Table 3.1. Before inverting the STFT, the padding at the start and the end of the signal is discarded.

3.3.2 Neural network models

Two NN models are developed. One is an adaptation of pix2pix, presented in the paper [15]. The other one is a custom GAN, made of a series of convolutional layers. It is named InpaintingGAN (IGAN). These two models share common characteristics: both follow the GAN framework, optimize the same loss, have the same kind of discriminator and finally are trained in the same way. In the next paragraph, we describe the common characteristics shared by the two models. Afterwards, the IGAN is detailed. Finally, the adapted pix2pix model is described.

The two GAN models have the same kind of discriminator, a patch discriminator with a binary cross entropy loss between fake and real instances. A patch discriminator divides the images into patches and classifies each of these as either fake or real. Both generator are fed with a real image that condition the inpainting or prediction class instead of a gaussian noise as in classical unconditioned GAN. Moreover, the generator loss also includes a "structural term", which is the mean absolute error between $\mathbf{g}$ and $\hat{\mathbf{g}}$. The structural term and the adversarial term are weighted thank to the parameter λ which is a constant. The adversarial term is the second term in the equation 3.4 and implies the discriminator loss. The discriminator loss is the following:

$$L_{discr}(\hat{\mathbf{g}}, \mathbf{g}) = H(\mathbf{1}, D(\mathbf{g}, \mathbf{p}, \mathbf{s})) + H(\mathbf{0}, D(\hat{\mathbf{g}}, \mathbf{p}, \mathbf{s})), \tag{3.3}$$

where: $\hat{\mathbf{g}}$ is the NN estimation of the magnitude spectrum of the gap frame, $D(\mathbf{g}, \mathbf{p}, \mathbf{s})$ is the discriminator output containing the probability that an image patch of the ground truth given the conditioning images come from a real

Layer type and name	Number of filters	Stride	Kernel size	Activation function
conv2d_4	32	(2,2)	(3,3)	ReLU
conv2d_5	64	(2,2)	(3,3)	ReLU
conv2d_6	128	(2,2)	(3,3)	ReLU
conv2d_7	1	(2,2)	(3,3)	sigmoid

Table 3.2: Parameter table of the discriminator. The discriminator is a stack of 4 convolutional layers with the parameters described above. A schema of this network is available in Figure 3.3

instance and $D(\hat{\mathbf{g}}, \mathbf{p}, \mathbf{s})$ is the discriminator output containing the probability that an image patch from a fake instance given the conditioning images come from a real instance, H is the cross entropy function defined in Section 1.2.5.

The generator loss is then defined by:

$$L_{gen}(\hat{\mathbf{g}}, \mathbf{g}) = \lambda|\hat{\mathbf{g}} - \mathbf{g}| + L_{discr}(\hat{\mathbf{g}}, \mathbf{g}). \tag{3.4}$$

The generator and discriminator are trained alternatively.

3.3.2.1 IGAN

The model presented in this section is a GAN, where both the generator and discriminator are made of a series of convolutional layers. A first model is developed for audio inpainting. The second presented model is an adaptation of the first model for audio prediction.

Inpainting model This model follows the GAN framework. It is composed of a generator and discriminator component. The discriminator is a patch discriminator, inspired from the one of the pix2pix model, presented in the paper [15]. It is made of a stack of 2D convolutional layers where each of these is followed by an ReLU activation function and a last 2D convolutional layer followed by a sigmoid activation function. The generator and discriminator loss as well as the training procedure are described in Sec. 3.3.2. An overview of the discriminator and generator architecture are shown in Figure 3.3 and 3.4 respectively. A detailed table of all the parameters is available in Table 3.2, 3.3 and 3.4.

Prediction model The prediction model is a re-adaptation of the audio inpainting architecture. The main difference is that while before IGAN was

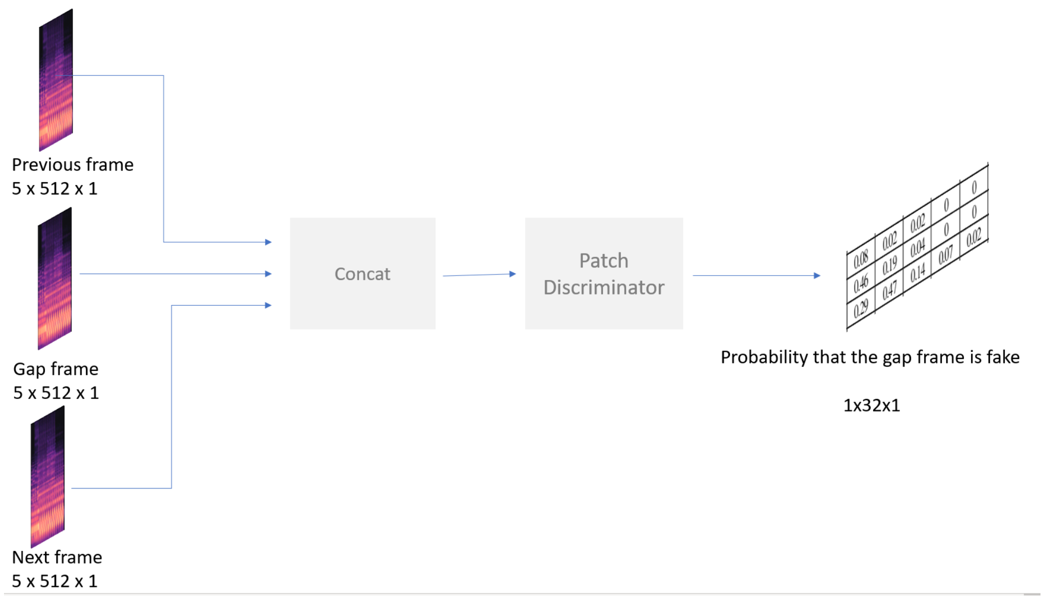


Figure 3.3: Discriminator part of the GAN based on CNN for the inpainting task. The discriminator receives as input $[\mathbf{p}, \mathbf{g} \oplus \hat{\mathbf{g}}, \mathbf{s}]$. It has to estimate whether it has received $\mathbf{g}$ or $\hat{\mathbf{g}}$. The discriminator outputs is an estimate probability whether each patch of the second input are from $\mathbf{g}$. A table containing the parameters of the four convolutional layers is available in Table 3.2

.

Layer name	Number of filters	Stride	Kernel size	Activation function
conv2d	32	(1,2)	(4,6)	ReLU
conv2d_1	64	(1,2)	(4,6)	ReLU
conv2d_2	128	(1,2)	(4,6)	ReLU
conv2d_3	256	(1,2)	(4,6)	ReLU

Table 3.3: Parameter table of the encoder part of the generator. A schema of this network is available in Figure 3.4

.

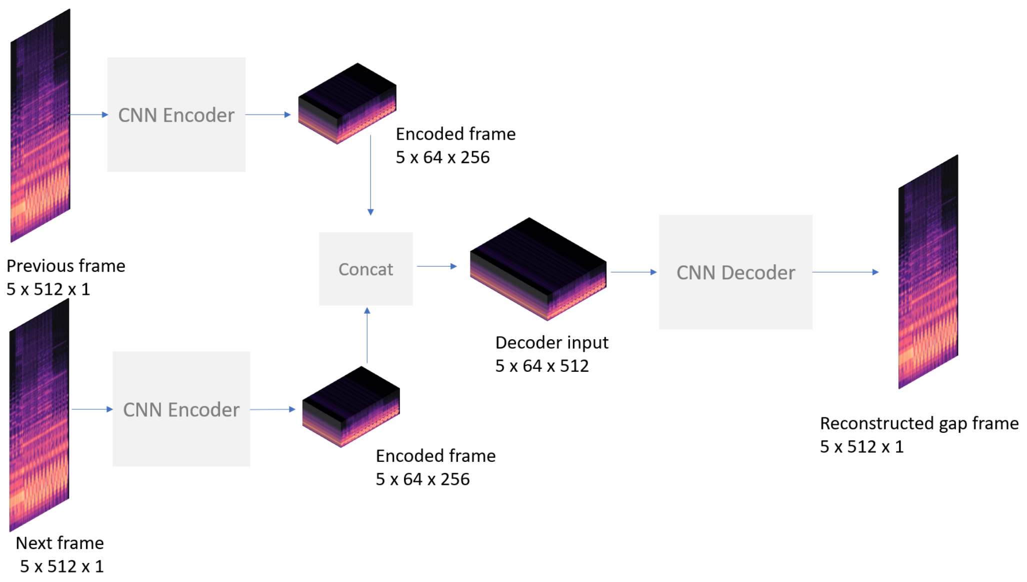


Figure 3.4: Generator part of the GAN based on CNN for the inpainting task. The previous frame and next frame go through the same encoder, which is a stack of convolutional layers. After that, the encoded version of the previous frame and next frame are concatenated and finally fed to the decoder part, which is a stack of transposed convolutional layers. The decoder part output is the model reconstruction of the magnitude spectrum of the gap frame.

Layer name	Number of filters	Stride	Kernel size	Activation function
conv2d_transpose	256	(1,2)	(4,6)	ReLU
conv2d_transpose_1	128	(1,2)	(4,6)	ReLU
conv2d_transpose_2	64	(1,1)	(4,6)	ReLU
conv2d_transpose_3	32	(1,1)	(4,6)	ReLU
conv2d_transpose_4	1	(1,1)	(4,6)	identity

Table 3.4: Parameter table of the decoder part of the generator. A schema of this network is available in Figure 3.4

.

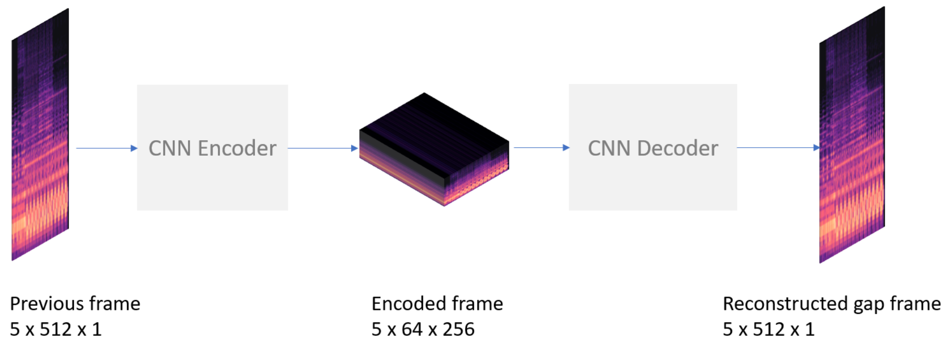


Figure 3.5: Generator part of the GAN based on CNN for the prediction task. The previous frame is first fed into the encoder which is a stack of convolutional layers. Then, it is fed into the decoder which is a stack of transpose convolutional layers. The model outputs the reconstruction of the magnitude spectrum of the gap frame.

fed with both the previous and subsequent frames **p** and **s**, now it is fed only with **p**, thus implementing actual audio prediction. For the sake of completeness, a schema of the generator and discriminator are available at Figure 3.5 and 3.6, respectively. The parameter tables are exactly the same of the one used for inpainting. They are available in Table 3.3 and 3.2.

3.3.2.2 Adapted pix2pix

Pix2pix is a GAN presented in the paper [15]. Given a conditioning input image, it generates a new image. For example, given an aerial picture, it can generate the corresponding map. To train this model, it is thus necessary to have a dataset of pair of conditioning images / possible outputs. In our case, the input image is the concatenation of **p**, a zero matrix which has the dimension of **g** and **s**. The target image is the concatenation of **p**, **g**, and **s** as illustrated in Figure 3.7. We closely follow the model description provide in [15], except that we adapt the kernel shape and the stride of the convolutional layers to our case. A schema of the pix2pix discriminator and generator are shown in Figure 3.8 and 3.9, respectively. The generator has a

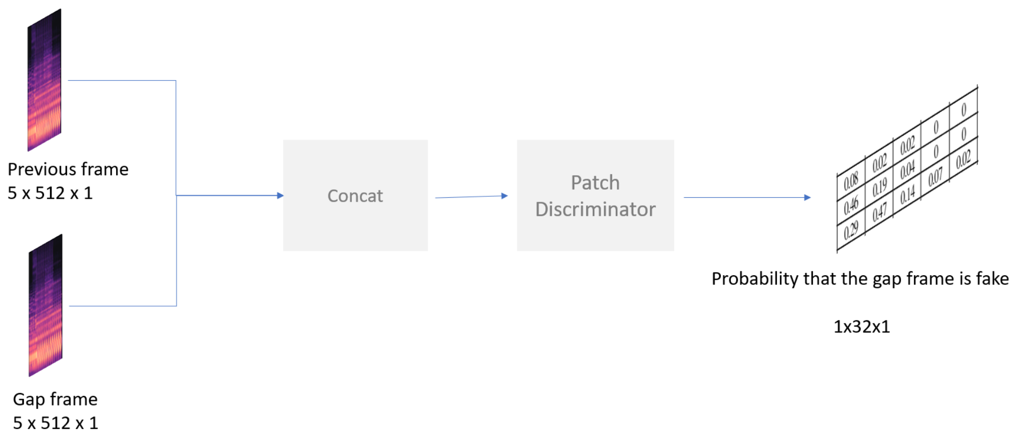


Figure 3.6: Discriminator part of the GAN based on CNN for the prediction task. The discriminator receives the concatenation of the previous frame and gap frame. The gap frame may come from the real data or may be the reconstructed gap frame output by the generator. The discriminator outputs its estimate probability whether the input is made from a fake or real gap frame. The discriminator is a stack of convolutional layer. A table containing the parameters of the 4 convolution layers is available in the annexe at section 3.2

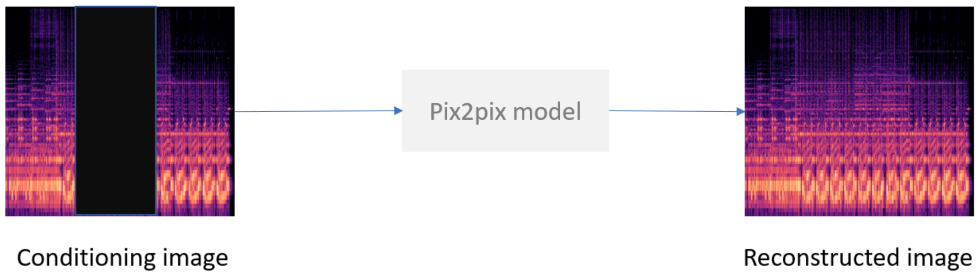


Figure 3.7: Illustration of the input/output of the pix2pix model. The input is the concatenation of the previous frame, a zero matrix and next frame. The output is the concatenation of the previous frame, reconstructed gap frame and next frame.

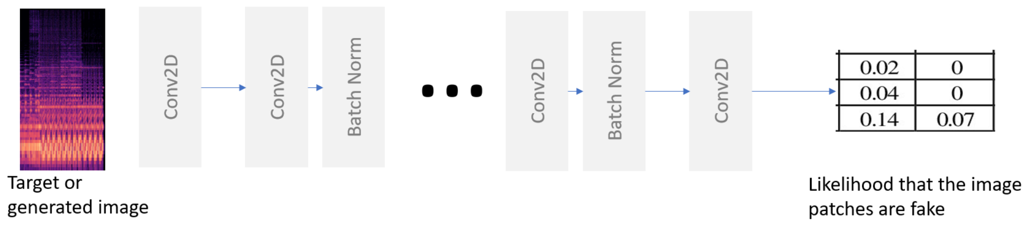

Target or generated image
Conv2D
Conv2D
Batch Norm
Conv2D
Batch Norm
Conv2D
0.02
0
0.04
0
0.14
0.07
Likelihood that the image patches are fake

Figure 3.8: Schema of the pix2pix discriminator. The discriminator takes as input the generated image or the ground truth and outputs its likelihood estimation that the patches of the input image are fake.

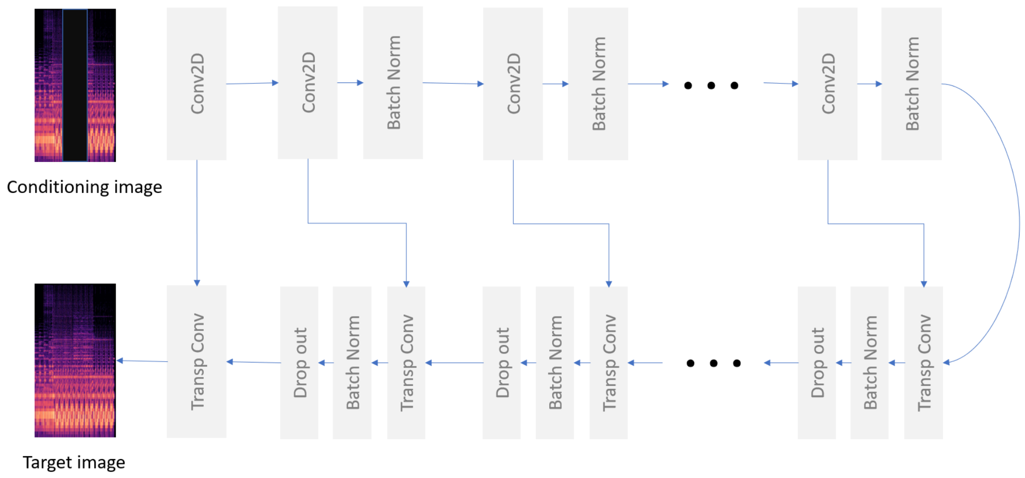


Figure 3.9: Schema of the pix2pix generator. The generator takes as input the concatenation of the previous frame, a zero matrix and the next frame and outputs its reconstruction of the previous frame, gap frame and next frame.

conv2d_0	filters=32	stride=(1,2)	kernel size=(4,6)	activation=ReLU
batch_norm_0				
conv2d_1	filters=64	stride=(1,2)	kernel size=(4,6)	activation=identity
batch_norm_1				
conv2d_2	filters=128	stride=(1,1)	kernel size=(4,6)	activation=ReLU
batch_norm_2				
conv2d_3	filters=256	stride=(1,1)	kernel size=(4,6)	activation=ReLU
conv2d_transpose_0	filters=256	stride=(1,2)	kernel size=(4,6)	activation=ReLU
drop_out_0	rate=0.5			
conv2d_transpose_1	filters=128	stride=(1,2)	kernel size=(4,6)	activation=ReLU
drop_out_1	rate=0.5			
conv2d_transpose_2	filters=64	stride=(1,1)	kernel size=(4,6)	activation=ReLU
drop_out_2	rate=0.5			
conv2d_transpose_3	filters=32	stride=(1,1)	kernel size=(4,6)	activation=ReLU
drop_out_3	rate=0.5			
conv2d_transpose_4	filters=1	stride=(1,1)	kernel size=(4,6)	activation=ReLU

Table 3.5: Parameter table of the generator. A schema of this network is available in Figure 3.4

.

U-net architecture and is made with blocks of convolutional layers followed by a batch normalization layer for the encoder part. The decoder part is made of block of transpose convolutional layer followed by batch normalization and drop out. A summary table of all the parameters of the pix2pix model is provided in the annexe at Table 3.5 and 3.6.

TODO	Number of filters	Stride	Kernel size	Activation function
conv2d_transpose	256	(1,2)	(4,6)	ReLU
conv2d_transpose_1	128	(1,2)	(4,6)	ReLU
conv2d_transpose_2	64	(1,1)	(4,6)	ReLU
conv2d_transpose_3	32	(1,1)	(4,6)	ReLU
conv2d_transpose_4	1	(1,1)	(4,6)	identity

Table 3.6: Parameter table of the decoder part of the generator. A schema of this network is available in Figure 3.4

.

Part III

Result and comparison

Chapter 4

Results

In this section we describe the results aimed at showing the performances of our models. Several audio samples are available on this website. We first describe the baseline in Section 4.1. Afterwards, we describe the dataset used in Section 4.2. Then, we describe the metrics used in Section 4.3. In the next Section 4.4, we describe and discuss the results of a first experiment, used to select the most appropriate model. In Section 4.5, 4.6, 4.7 the best model is further tested on different case to see on which extend it can generalize. Finally, we discuss and conclude all the results.

4.1 LPC baseline

The LPC baseline is a slightly adapted version of the one used in [23]. It uses the Burg's method to find the coefficient of the optimal linear filter in the mean squared error sense. This is done forward with the previous frame and backward with the next frame. Then the two predictions are averaged with a cosine function. We use a filter of order 1000. The matlab code of the lpc based method is available in the annexe at .1.1.

4.2 Datasets

Two different datasets are considered of increasing difficulty : MAESTRO and the small version of the Free Music Archive (FMA) dataset. All audio files were resampled to 16000 Hz if needed.

4.2.1 MAESTRO

The MAESTRO dataset [13] is composed of piano performance recordings of a contest, both in wav and MIDI format. We only use the wave records of the contest. The dataset has a suggested partitioning, each record belongs to one on these sets: train set, validation set and test set. Each record was split into sample of 4 s long. These samples are then written in either train.tfrecord, val.tfrecord, test.tfrecord, according to the suggested partitioning. The tfrecord samples have only one feature, called 'audio', containing the waveform.

4.2.2 FMA

The FMA dataset [4] is composed of music records classified into 161 genres. It was created for music retrieval tasks and weights 879 GB. We use the small version of this dataset which contains 8000 trimmed tracks of 30 seconds of 8 different genres for a total of 7.2 GB. Like the MAESTRO dataset, the FMA dataset has a suggested training set, validation set and test set. Similarly to the MAESTRO dataset, each track was split into several samples of 4s and was stored either in one the following files: train.tfrecord, val.tfrecord or test.tfrecord. The tfrecord samples has only one feature, called 'audio', containing the waveform.

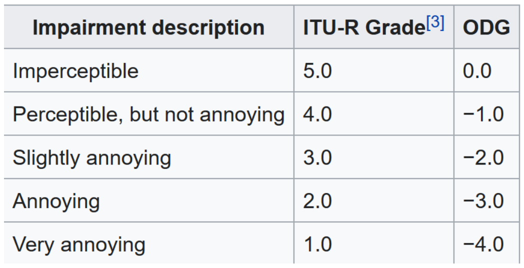
4.3 Evaluation measures

Given a target signal $\mathbf{y}$ and a reconstructed signal $\hat{\mathbf{y}}$, a measure is a function $f(\mathbf{y}, \hat{\mathbf{y}})$ which evaluates how good is the reconstruction with respect to the original. Two measures were used. The two first only take into account the signal value while the third one tries to model the human auditory system so that the measure is perceptually meaningful.

4.3.1 Signal to noise ratio (SNR)

The signal to noise ratio is an usual measures in signal processing. It compare the ratio of a desired signal over the noise. It is defined by:

$$10 \log_{10} \frac{||\mathbf{y}||^2}{||\mathbf{y} - \hat{\mathbf{y}}||^2}, \tag{4.1}$$



Impairment description	ITU-R Grade[3]	ODG
Imperceptible	5.0	0.0
Perceptible, but not annoying	4.0	−1.0
Slightly annoying	3.0	−2.0
Annoying	2.0	−3.0
Very annoying	1.0	−4.0

Figure 4.1: ODG scale

where $\mathbf{y}$ is the reference signal and $\hat{\mathbf{y}}$ is the estimated one. A SNR over 0 dB means there is more signal than noise.

4.3.2 Objective difference grade (ODG)

The ODG is a perceptual measure that has a scale from 0 to -4. It evaluates how different is the tested signal with respect to the reference signal. Zero means that there is no perceptual difference while -4 means that the difference is perceptually very annoying. The ODG Table definition is shown in Figure 4.1. Any algorithm that claims to compute the ODG should be able to fulfil the requirements detailed in [5]. The candidate algorithm must model the human hearing system and should output the same results of a reference dataset of listening tests.

Perceptual Evaluation of Audio Quality [34] (PEAQ) is the name of an algorithm that fulfil the requirement of the ITU standard [5]. However it is protected by patents. We will use instead a matlab open source implementation [17]. It doesn't fully comply from the requirements in [5]. In some cases, the error with the reference dataset is slightly larger than the margin allowed.

4.4 Comparative experiment

The models were first compared on an inpainting test with $T_g = T_p = T_s =$ 64ms. Samples of 64 ms are too short for a perceptual evaluation. To evaluate the model with the Objective difference grade (ODG), samples of 2 seconds are made. The inpainting gap $\mathbf{x}_g$ starts at 0.5 seconds and $\mathbf{x}_g$ is estimated through the model presented in Section 3. The reference signal is the original

learning rate	0.001
beta 1	0.9
beta 2	0.99

Table 4.1: Adam Optimizer parameters

audio of two seconds while the signal under test is the concatenation of the first 0.5 second of the original song with the reconstructed gap frame and the last 1.436 second of the original audio. The Signal to noise ratio (SNR) on spectrum compares the magnitude spectrum of the gap frame of the reconstructed waveform and the original one, SNRs $= \mathrm{SNR}(\mathbf{g}, \hat{\mathbf{g}})$. The SNR on audio compares the reconstructed waveform with respect to the waveform of the gap frame, SNRa $= \mathrm{SNR}(\mathbf{x}_g, \hat{\mathbf{x}}_g)$. The InpaintingGAN (IGAN) was trained with an Adam optimizer [19] with the parameter given in table 4.1. The model was trained for 100 epochs on the Free Music Archive (FMA) dataset.

4.4.1 Discussion

The InpaintingCNN (ICNN) is exactly the same than the generator part of the IGAN. Table 4.2 shows the metric values of the different models on the task of inpainting 64 ms. It appears that the IGAN model performs better than the simpler ICNN one. This shows that the use of the adversarial loss benefits the inpainting task. Indeed, the SNR on audio (SNRa) is increased of almost 1 dB. Beside, the ODG is also improved, from -1 to -0.7. It is also obvious from that table that both the ICNN and IGAN model are better than the pix2pix one. It can be also observe that the generated magnitude spectrums of all deep learning models are better than the one of the Linear predictive coefficient (LPC) based method, even if the audio waveforms of the latter one are closer to the original audio in the SNR sense. This can be explained by the phase estimation step of the deep learning model. In fact, applying the Griffin-Lim algorithm on even the real magnitude spectrum can lead to a different waveform. Moreover, the Griffin-Lim algorithm is not applied on the real magnitude of the gap spectrum, but on an estimation of this one. Thus, it is quite natural that the reconstructed waveforms of the deep learning model have a quite low SNRa. Despite the fact that the SNRa is better for the LPC based method, the perceptual measure of the best deep learning model is slightly better than the one of the LPC baseline. From the results shown in Table 4.2, it is obvious that the IGAN model is the best one among the proposed models. Comparing to the LPC baseline, the results obtained through the IGAN method seem to be perceptually better even if the reconstructed waveform of the LPC model is closer to the original audio in term of SNR. Thus, in the next experiment, we will only focus on the IGAN model and compare it to the LPC baseline.

	ICNN	IGAN	pix2pix	LPC
SNR on spectrum	7.96	8.93	6.97	5.75
SNR on audio	-1.26	-1.15	-1.02	1.038
ODG	-1	-0.71	-1.02	-0.85

Table 4.2: Results for inpainting 64 ms given the 64 ms before and after the gap

4.5 Inpainting longer gap

In this experiment, the IGAN is trained on the FMA dataset with the optimizer described in Table 4.1 for 100 epochs to perform inpainting with $T_p = T_g = T_s = 128$ms. Table 4.3 shows the results of the experiment. From these results, IGAN seems to be better than the LPC baseline. Even if the LPC is better on the SNRa, the IGAN model is better on the SNR on spectrum (SNRs) and also on the ODG, as in the comparative experiment.

	IGAN	LPC
SNR on spectrum	6.98	4.21
SNR on audio	-1.06	-0.52
ODG	-1	-1.13

Table 4.3: Results for inpainting 128 ms given the 128 ms before and after the gap

4.6 Prediction

In this experiment, the proposed model is only fed with the previous frame and tries to deduce the next frame from it. The experiments are done for a gap length of 64 ms and 128 ms. The results are summarized in the table 4.4 and 4.5.

	IGAN	LPC
SNR on spectrum	6.52	4.52
SNR on audio	-0.99	-0.38
ODG	-0.98	-0.94

Table 4.4: Results for predicting 64 ms given the 64 previous ms

	IGAN	LPC
SNR on spectrum	5.32	4.17
SNR on audio	-0.94	- 3.03
ODG	-1.19	-1.01

Table 4.5: Results for predicting 128 ms given the 128 previous ms

As in the inpainting case, the SNRs is better for the IGAN model even if the SNRa is better in the LPC model. In the 128 ms case, the perceptual metric is slightly better for the IGAN model while in the 64 ms case, the LPC baseline and the IGAN model have similar performance.

4.7 Generalization

In this experiment, all the previous models are trained on the FMA dataset and were tested on the MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) dataset. The results are shown in Table 4.6 , 4.7, 4.8 and 4.9.

4.7.1 Discussion

By comparing the figure in Table 4.2 and 4.6, one can see that the SNRs is better on the MAESTRO dataset rather than on the FMA dataset, even if the

	IGAN	LPC
SNR on spectrum	11.08	8.58
SNR on audio	-1.24	4.18
ODG	-1.5	-0.63

Table 4.6: Results for inpainting 64 ms given the 64 next and previous ms. The model is trained on the FMA dataset and test on the MAESTRO dataset

	IGAN	LPC
SNR on spectrum	8.86	7.65
SNR on audio	-1.28	2.96
ODG	-1.75	-0.83

Table 4.7: Results for inpainting 128 ms given the 128 next and previous ms. The model is trained on the FMA dataset and test on the MAESTRO dataset

	IGAN	LPC
SNR on spectrum	8.32	6.44
SNR on audio	-1.64	1.48
ODG	-1.55	-1.12

Table 4.8: Results for predicting 64 ms given the 64 previous ms. The model is trained on the FMA dataset and test on the MAESTRO dataset

	IGAN	LPC
SNR on spectrum	6.81	5.67
SNR on audio	-1.05	-0.31
ODG	-1.83	-1.32

Table 4.9: Results for predicting 128 ms given the 128 previous ms. Model trained on the FMA dataset and test on the MAESTRO dataset

model is only trained on the FMA dataset. Beside, one can also notices that the LPC method works better on the MAESTRO dataset. This observation can also be noticed by comparing Table 4.3 and 4.7. This can be probably explained by the composition of the MAESTRO and FMA dataset. Indeed, the FMA dataset is composed of a lot of different music styles, instruments and sound effects while the MAESTRO dataset only contains records of classical piano. Thus, the waveform and frequency content of the MAESTRO dataset are less diversified and less complex if compared to the FMA ones.

Surprisingly, when comparing Table 4.8 with 4.4 and Table 4.9 with 4.5, the ODG of the LPC on the MAESTRO dataset is lower than on the FMA dataset, even if the two other metrics are better on the MAESTRO dataset than on on the FMA.

From these experiments, it seems that IGAN generalizes rather well with respect to different type of data. Indeed, the SNRs is systematically higher when testing on the MAESTRO dataset despite the fact that the training dataset and the testing one are quite different. However, the ODG on the MAESTRO dataset is worse. That can be probably explained by the composition of the MAESTRO and FMA dataset. The MAESTRO dataset contains simpler sound. Thus, it might be easier for the human hearing system to differentiate between two simple audio than two complex ones. Consequently, the ODG, which models the human hearing system, might be lower for a same SNR.

4.8 General Discussion

Compared to the LPC baseline, all the developed models estimate better the magnitude spectrum of the gap frame. However, even if the magnitude is better estimated, the SNRa is lower than the one of the LPC. These observations are probably explained by the fact that the LPC baseline works on the time domain and tries to minimize the squared error between the original waveform and the reconstructed waveform while the deep learning models tries to minimize the reconstruction error on the magnitude spectrum. Moreover, after computing the magnitude spectrum, the deep learning models use the Griffin-Lim algorithm to estimate the phase given a magnitude spectrum. However, even if the Griffin-Lim algorithm is fed with the real magnitude, the reconstructed audio is not exactly the same than the real one. Also, the Griffin-Lim algorithm is not fed with the real magnitude, but with an estimation of it, given by the Neural network (NN). Thus is quite natural that the reconstructed audio is not the same than the original one. This explains why the SNRa is lower with the developed model while the SNRs is higher comparing to the LPC baseline.

Even if the SNRa is lower with the proposed method than with the LPC baseline, that doesn't heavily impact the perception of the sound. Indeed, the ODG of the IGAN is better in the case of inpaiting 64ms, as shown in Table 4.2. That means that the reconstructed audio is perceptually closer to the original one. Unfortunately, that only happen in this case. On the other experiments, the ODG of the IGAN and the LPC are similar.

As the SNRs is rather good, but the SNRa is low, the weak point of this solution probably lies in the phase estimation. The Griffin-Lim algorithm is an iterative algorithm. Increasing the number of iterations may improve the phase estimation. This immediate solution was tested both on $\mathbf{g}$ and $\hat{\mathbf{g}}$. While increasing the number of iteration effectively improved the phase estimation on $\mathbf{g}$, it does not improved on $\hat{\mathbf{g}}$. Thus, increasing the IGAN performance might also allowed us to improve the phase estimation done from it as well as $\hat{\mathbf{x}}_g$. Another way to improve it might by using the information contained in $\mathbf{x}_p$ and $\mathbf{x}_s$ incorporate them inside the phase estimation step. In a future work, we would like to explore different phase recovery algorithm and improve the phase estimation.

The proposed method is quite general and can be directly applied to any type of audio. It can be also interesting to explore its effectiveness on other types of data. As soon as it is possible to applied the Short time fourier transform (STFT) on the data, it is also possible to apply the proposed method.

Conclusion

In this thesis, we tackled the problem of audio inpainting and extend our solution to audio prediction. The proposed solution combines a signal processing pipeline and a convolutional generative adversarial network. The neural network estimates the magnitude spectrum of the frame to inpaint, given the surrounding magnitude spectrum. The signal processing pipeline computes these spectrums and then estimate the phase of the frame to inpaint before inverting the short time fourier transform. The proposed solution shows encouraging result, it was able to outperform the linear prediction coding baseline in term of SNRs and performs similarly well in term of the perceptual measure ODG. However, the final audio show a low signal to noise ratio with respect to the baseline. Improving the phase estimation can potentially significantly improved the final audio and is let for further work.

The phase estimation can be improved by taking into account the information provided in the previous and subsequent frame. Another way to improve it is by increasing the number of iteration of the Griffin-Lim algorithm. However, the Griffin-Lim algorithm estimates the phase from the magnitude, which is already an estimation made by the neural network. Thus, improving the quality of the magnitude spectrum reconstruction will also allow a better phase estimation and will improve the audio reconstruction.

.1 Matlab code

.1.1 LPC based method code

```
contextLength = 1024;
targetLength = 1024;
contextRatio = ceil(contextLength/targetLength);
maxLag = 1000;
prediction = true;

files = dir('../maestro_dataset/audio/*.wav');

snr = 0;
count = 0;
skipped_value = 0;
for j=1 : size(files)
    audioFilePath = fullfile(files(j).folder, files(j).name)
    [audio, Fs] = audioread(audioFilePath);
    target = 16000;
    audio = resample(audio,target,Fs);

    t = linspace(0, pi/2, targetLength)';
    sqCos = cos(t).^2;


    %for i = ...
       contextRatio:(length(audio)/targetLength)-contextRatio-2

        previous_sig = audio(8000: 8000 + contextLength-1);
        target_sig = audio(8000 + contextLength : 8000 + ...
           contextLength + targetLength-1);
        next_sig = audio(8000 + contextLength + ...
           targetLength : 8000 +  2 * contextLength + ...
           targetLength-1);

        ab = arburg(previous_sig, maxLag);
        Zb = filtic(1,ab,previous_sig(end-(0:(maxLag-1))));
        forw_pred = filter(1,ab,zeros(1,targetLength),Zb)';

        if ¬prediction
            next_sig = flipud(next_sig);
            af = arburg(next_sig, maxLag);
            Zf = filtic(1,af, next_sig(end-(0:(maxLag-1))));
            backw_pred = ...
               flipud(filter(1,af,zeros(1,targetLength),Zf)');
            sigout = sqCos.*forw_pred + ...
               flipud(sqCos).*backw_pred;
        else
            sigout = forw_pred;
        end
        filename1 = ...
           strcat('../reconstruction/maestro/lpc/64ms/prediction/', ...
           'rec_' , int2str(j), '_',int2str(i), '.wav')
        filename2 = ...
           strcat('../reconstruction/maestro/lpc/64ms/prediction/', ...
           'ori_' , int2str(j), '_',int2str(i), '.wav')
        filename3 = ...
           strcat('../reconstruction/maestro/lpc/64ms/prediction/percep_eval/', .
           'rec_' , int2str(j), '_',int2str(i), '.wav')
        filename4 = ...
           strcat('../reconstruction/maestro/lpc/64ms/prediction/percep_eval/', .
           'ori_' , int2str(j), '_',int2str(i), '.wav')

        audiowrite(filename1, sigout, target)
        audiowrite(filename2, target_sig, target)
        rec2s = cat(1, audio(1: 8000 + contextLength), ...
            sigout, audio(8000 + contextLength + ...
            targetLength: 32000));
        audiowrite(filename3, rec2s, target)
        audiowrite(filename4, audio(1:32000), target)
        size(sigout)
        size(target_sig)
        value = mySNR(sigout, target_sig)

        if ¬isfinite(value)
            skipped_value = skipped_value + 1
            continue
        end
        snr = snr + value;
        count = count + 1;



    %end
end
snr = snr / count

skipped_value
```

Bibliography

[1] Illustrated self attention. https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a. Accessed: 2020-02-07.

[2] Amir Adler, Valentin Emiya, Maria G. Jafari, Michael Elad, Rémi Gribonval, and Mark D. Plumbley. Audio Inpainting. *IEEE Transactions on Audio, Speech and Language Processing*, 20(3):922 – 932, March 2012.

[3] Yuval Bahat, Yoav Schechner, and Michael Elad. Self-content-based audio inpainting. *Signal Processing*, 111, 06 2015.

[4] Kirell Benzi, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. FMA: A dataset for music analysis. *CoRR*, abs/1612.01840, 2016.

[5] RECOMMENDATION ITU-R BS.1387-1. Method for objective measurements of perceived audio quality. 2001.

[6] Rumelhart D, Hinton G., and Williams R. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[7] Jae S. Lim Daniel W. Griffin. Signal estimation from modified short-time fourier transform. 1983.

[8] Chris Donahue, Julian J. McAuley, and Miller S. Puckette. Synthesizing audio with generative adversarial networks. *CoRR*, abs/1802.04208, 2018.

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

[10] Jesse H. Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *CoRR*, abs/1902.08710, 2019.

[11] C. Gaultier, S. Kitić, N. Bertin, and R. Gribonval. Audascity: Audio denoising by adaptive social cosparsity. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 1265–1269, 2017.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[13] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.

[14] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. An improved relative self-attention mechanism for transformer with application to music generation. *CoRR*, abs/1809.04281, 2018.

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[16] A. Janssen, R. Veldhuis, and L. Vries. Adaptive interpolation of discrete-time signals that can be modeled as autoregressive processes. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(2):317–330, 1986.

[17] P. Kabal. An examination and interpretation of itu-r bs.1387: Perceptual evaluation of audio quality. 2003.

[18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[19] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[20] Srđan Kitić, Nancy Bertin, and Rémi Gribonval. Sparsity and cosparsity for audio declipping: A flexible non-convex approach. In Emmanuel Vincent, Arie Yeredor, Zbyněk Koldovský, and Petr Tichavský, editors, *Latent Variable Analysis and Signal Separation*, pages 243–250, Cham, 2015. Springer International Publishing.

[21] Florian Lieb and Hans-Georg Stark. Audio inpainting: Evaluation of time-frequency representations and structured sparsity approaches. *Signal Processing*, 153:291 – 299, 2018.

[22] Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.

[23] A. Marafioti, N. Perraudin, N. Holighaus, and P. Majdak. A context encoder for audio inpainting. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2362–2372, 2019.

[24] Andrés Marafioti, Nicki Holighaus, Nathanaël Perraudin, and Piotr Majdak. Adversarial generation of time-frequency features with application in audio synthesis. *CoRR*, abs/1902.04072, 2019.

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

[26] O. Mokrý, P. Záviška, P. Rajmic, and V. Veselý. Introducing spain (sparse audio inpainter). In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, 2019.

[27] Christine Payne. Musenet. http://openai.com/blog/musenet, April 2019. Accessed: 2010-09-30.

[28] N. Perraudin, N. Holighaus, P. Majdak, and P. Balazs. Inpainting of long audio segments with similarity graphs. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(6):1083–1094, 2018.

[29] Nathanaël Perraudin, Peter Balazs, and Peter Søndergaard. A fast griffin–lim algorithm. pages 1–4, 10 2013.

[30] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[31] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018.

[32] Kai Siedenburg, Matthieu Kowalski, and Monika Doerfler. Audio declipping with social sparsity. pages 1577–1581, 05 2014.

[33] Marco Tagliasacchi, Beat Gfeller, Félix de Chaumont Quitry, and Dominik Roblek. Self-supervised audio representation learning for mobile devices, 2019.

[34] Thilo Thiede, William C Treurniet, Roland Bitto, Christian Schmidmer, Thomas Sporer, John G. Beerends, Catherine Colomes, Michael Keyhl, Gerhard Stoll, and Bernhard Feiten Karlheinz Brandenburg. Peaq the itu standard for objective measurement of perceived audio quality. 2000.

[35] Ichrak Toumi and Valentin Emiya. Sparse non-local similarity modeling for audio inpainting. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*, pages 576–580. IEEE, 2018.

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

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances*

in Neural Information Processing Systems 30, pages 5998–6008. Curran Associates, Inc., 2017.