

Imperial College London
Department of Mathematics

Audio Source Separation

Samuel Argouet

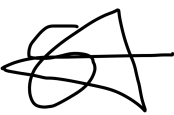
CID: 01786079

Supervised by Kevin Webster

4th September 2020

Submitted in partial fulfilment of the requirements for the MSc in Statistics of
Imperial College London

The work contained in this thesis is my own work unless otherwise stated.

Signed: 

Date: 04/09/2020

Abstract

Audio Source Separation aims to separate an audio mixture of different sources, for instance to extract a voice from a noisy background or to obtain the different instrument tracks from a song. Source Separation is widely studied, and many methods and models exist. Each class of models comes with limitations and advantages. One limitation of the currently best performing models, which are based on supervised learning, is that they require large amount of labeled data, and they can't generalize well to different types of mixtures.

In this project, we adopt a Bayesian approach for source separation. The process is divided into a training process and a separation process. The training process consists in training a generative model for each source. The separation process is based on a Bayesian sampling procedure. This novel method is more flexible for generalization than the supervised model since the separation process does not depend directly on the type of mixture. Indeed, once multiple generative models for different sources are trained, they can be added or removed in the separation process depending on what the mixture is made of. In addition, this method does not require labeled data, since the training and the separation processes are independent.

Using state-of-the-art deep generative models that we trained on melspectrograms, we were able to separate audio mixture in the time-frequency domain effectively and without the need for large amount of data. The results can be listened to on our website: <https://samargt.github.io/AudioSourceSep/>.

Acknowledgements

I would like to thank my supervisor Dr. Kevin Webster for his precious help and guidance, useful critiques and suggestions and his availability through all the stages of this research project. I would like to thank Shelley Katz for the production and collection of the data which was essential for this project. Finally, I wish to thank my parents and my sisters for their support and patience.

Table of Contents

1. Introduction	7
1.1. The Audio Source Separation Problem	7
1.2. Background	8
1.2.1. Blind Source Separation	8
1.2.2. Supervised Source Separation with Deep Neural Networks	9
1.2.3. Bayesian and Generative Source Separation	9
1.3. Our Method	10
1.3.1. Principle	10
1.3.2. Overview of Deep Generative Models	11
1.3.3. Challenges and Discussion	13
2. Unsupervised Audio Source Separation with Deep Generative Priors	14
2.1. Source Separation with Langevin Dynamics	14
2.1.1. Stochastic Gradient Langevin Dynamics	14
2.1.2. Simulated Annealing	15
2.1.3. Bayesian Annealed Signal Separation	15
2.2. Deep Generative Priors	18
2.2.1. The Glow model	18
2.2.2. Noise Conditioned Score Network	23
2.3. Signal Processing for Generative Models	25
3. Experiments	27
3.1. Results on Toy Datasets	27
3.2. The Audio Data	30
3.2.1. Data Preprocessing and Postprocessing	30
3.2.2. Inverting the Spectrograms to the Time Domain	31
3.3. Results on Spectrograms	31
3.3.1. Training the Generative Priors	31
3.3.2. Separation Performance	34
4. Conclusion	39
4.1. Synthesis of our Findings	39
4.2. Further Work	39
Bibliography	40

A. NCSN Architecture	43
A.1. Multi-path Refinement	43
A.2. Conditional Residual Block	44
A.3. Refine Block	44
B. Generated Spectrograms	46
C. BASIS convergence examples	48

1. Introduction

1.1. The Audio Source Separation Problem

Source Separation has been an area of intensive research these last few years [27] especially with the development of Deep Learning. The Source Separation task aims to separate a mix signal coming from different sources. The signal could be recorded with several microphones, but we will be interested in the simplest case where there is only one: the single channel source separation problem [3].

Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)^T$ be a mix signal, where n is the number of channel. Let $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_K)^T$ be the source components where K is the number of components. The common signal representations are time vector (1-D vector) or spectrograms (2-D vectors) (see section 2.3). The relation between the mix signal and the source components is given by the mixing process: $\mathbf{x} = g(\mathbf{s})$. Depending on the model used, the mixing process needs to be known or constrained to some class of functions but it is not always the case. The objective is then to retrieve \mathbf{s} given the mix signal \mathbf{x} and some other background knowledge on the mixing process or the type of sources.

Source Separation is a general applied mathematics problem with various domain of applications such as Neuroscience, Telecommunication and Audio. In music, Source Separation can be applied to separate instruments from music recording. Music source separation can be used for up-mixing, creative music production or karaoke. The "cocktail party problem" is a famous example of motivation for the Audio Source Separation task: the goal being to isolate a speaker from an audio mixture recorded during a crowded party. More precisely, it is used in speech enhancement [41] which aims to obtain clean speech from noisy audio and is widely used in speech recognition software (Siri...).

Music Source Separation is a difficult task since the instrument tracks are often highly correlated and the mixing process is in general non-linear. In the rest of this section, we present an overview of the different approaches that exist to tackle the music source separation problem, and we give a first overview of the approach we adopted and explain why this approach is novel and interesting. In Section II we detail the method we chose and the different algorithms and neural networks we used. Section III presents the experiments we did as well as the results we obtained. Finally, Section IV synthesizes our work and proposes ways to improve our approach.

1.2. Background

Many kinds of models have been developed in order to address the source separation problem. Most of them can be classified into 3 categories.

1.2.1. Blind Source Separation

The first approach is named **Blind Source Separation** and is the traditional approach to source separation. It defines a family of model which does not rely on prior data to separate the mixture. The three main models of this category are Independent Component Analysis [13] (ICA), Non-Negative Matrix Factorization (NMF) [28] [43] and Slow Feature Analysis (SFA) [31].

Independent Component Analysis: ICA models a linear relationship between the mixture \mathbf{x} and the components \mathbf{s} : $\mathbf{x} = \mathbf{A}\mathbf{s}$. The model assumes that there is the same number of channel and components. The components are latent random variables (un-observed) and are considered to be independent. In addition, they are supposed to be non-Gaussian. Various methods are used to estimate the matrix \mathbf{A} such as minimizing the Mutual Information or maximizing the non-Gaussianity via kurtosis. The sources are then estimated by inverting \mathbf{A} .

Non-negative Matrix Factorization: NMF models the input \mathbf{V} , a non-negative matrix of size $F \times T$ as the product of two non-negative matrices \mathbf{W} and \mathbf{H} of size $F \times K$ and $K \times T$. The problems to solve is $\min_{\mathbf{H}, \mathbf{W}} D(\mathbf{V} | \mathbf{W}\mathbf{H})$ where D is a divergence. NMF can be applied to source separation by modelling a linear relationship between the mixture and the components in the same way as for ICA. NMF is in particular suited when the signal is represented in the frequency domain such as power spectrogram since the values are non-negative. Moreover, sparse spectrograms such as music spectrograms often have a unique decomposition. Estimation of \mathbf{W} and \mathbf{H} is usually done with a multiplicative update rule which alternates between \mathbf{W} and \mathbf{H} by majorization-minimization.

Slow Feature Analysis: Given a multi-variate time signal $\mathbf{x}(t)$, the aim of SFA is to find a set of scalar functions $\{g_j\}_{j=1,\dots,J}$ such that the output signals $\{y_j(t)\}_{j=1,\dots,J}$ of theses functions vary as slowly as possible, i.e. minimizing $\Delta(y_j) = \langle \frac{dy_j}{dt} \rangle_t$ with the constraints that the outputs are uncorrelated, have zero mean and unit variance. When the function space is finite, the SFA algorithm can solve the optimization problem and generate a sequence of functions which are ordered according to the slowness of their outputs, g_1 being the slowest function. SFA can be applied to blind source separation where $\mathbf{x}(t)$ is the mixture signal and the output signals are the sources. The theory shows that the output signals of the SFA are similar to the true sources. Sprekeler et al. [31] develops the eXtended SFA algorithm (xSFA) to extract with high reliability the sources from a non-linear mixture.

Blind Source Separation models have the advantage of being data agnostics and have been widely studied. Many more powerful and complex models have been developed based on ICA and NMF. However, they are limited by the assumptions they make and often remain too simple to learn the complexity of the mixing process.

1.2.2. Supervised Source Separation with Deep Neural Networks

The development of deep learning techniques allows the appearance of supervised deep neural networks (DNN) for source separation. The networks are simply trained to generate sources as close as possible to the true sources given the mixture. These models quickly outperformed previous techniques. Spectrograms based DNN [37] performed the best during the Signal Separation Evaluation Campaign (SiSec) in 2018 [34]. Spectrograms based models are either trained to model the source spectrograms directly [37] or trained to learn a mask which is then apply to the mixture spectrograms as a filter [35].

The main limitation of frequency domain models is the conversion of the spectrograms back to the time domain which is flawed by approximations. More recently, DNN operating in the waveform domain have been developed, and they now outperform spectrograms based models [21].

Supervised DNN require a large amount of data in order to be trained efficiently and therefore it is their main limitation. Indeed, gathering large amount of data is difficult and not scalable since each model is trained to separate a specific type of mixture. Moreover, DNN are not easily interpretable which makes them difficult to use in a real world application. Music engineering often requires a total control of the inputs and outputs which is not attainable with a black-box model.

1.2.3. Bayesian and Generative Source Separation

The last approach lies between the blind source separation and the supervised regression. The idea is to use the available knowledge about the sources as priors to separate the mixture. The data about the sources is not required to be the true sources of the mixture, hence the data limitation issue is diminished. In addition, this technique allows a better generalization capacity.

NMF algorithms can be used in that framework by learning a dictionary of each source with a NMF model and then separate the mixture with another NMF algorithm [28].

Bayesian Source Separation: The single-channel source separation problem can be also formulated in the Bayesian setting [2]. Let consider \mathbf{x} the mixture to separate and \mathbf{s} , the sources as random variables. The objective is to be able to sample from the posterior:

$$p(\mathbf{s}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{s})p(\mathbf{s})}{p(\mathbf{x})} \quad (1.1)$$

$p(\mathbf{x}|\mathbf{s})$ is the observation model and characterizes the mixing process, it is often set as a normal distribution. $p(\mathbf{s})$ is the source model. The model can easily be made hierarchical

by giving parameters to the observation and source models.

Cemgil et al.[2] created a Bayesian hierarchical model for source separation. The model leads to a posterior distribution of the form:

$$p(\mathbf{s}|\mathbf{x}) = \frac{1}{Z_x} \int p(\mathbf{x}|\mathbf{s}, \Theta)p(\Theta)d\Theta$$

where Θ is a set of parameters (not detailed here) and Z_x is the scaling parameter. The inference is then made using Markov-Chain Monte Carlo methods or Variational Bayes to sample from the joint posterior distribution $p(\mathbf{s}, \Theta)$ and by estimating $\hat{\mathbf{s}} = \sum_i \mathbf{s}^{(i)}$, where $\mathbf{s}^{(i)}$ are the samples from the joint posterior ($\Theta^{(i)}$ are discarded).

Source Separation with Generative Adversarial Network: Recently, generative adversarial networks (GAN) have been used in this framework [36][18].

GAN are composed of a generative network and a discriminative network which are trained jointly. The generator tries to sample from the wanted distribution starting from a latent variable \mathbf{h} with a tractable distribution, while the discriminator tries to discriminate the generated samples from the true samples.

In the same Bayesian setting as Equation 1.1, the source vector \mathbf{s} is replaced by $G(\mathbf{h})$, where G is the generative network of the pre-trained GAN. But contrary to the Bayesian source separation method, this method does not try to sample from the posterior distribution. Indeed, the separation is performed with a gradient-based iterative approach to minimize the reconstruction error by finding the optimal latent variable. Depending on the derivation, this is equivalent to maximizing the conditional likelihood $p(\mathbf{x}|G(\mathbf{h}))$ [36] or maximizing the posterior distribution $p(\mathbf{x}|G(\mathbf{h}))p(\mathbf{s})$ [18] where the prior distribution is approximated.

1.3. Our Method

In this project, we have decided to adopt the Bayesian Source Separation approach and to make use of state-of-the-art deep generative models at the same time.

1.3.1. Principle

Our method is based on the work of Jayaram & Thickstun [15]. We aim to sample from the posterior distribution (Equation 1.1) in order to estimate the source components.

The observation model is set as a Gaussian Distribution $\mathcal{N}(\mathbf{x}; g(\mathbf{s}), \sigma^2 \mathbf{I})$ where the mixing process $g(\cdot)$ is chosen to be the average of the components (in the time domain) while the priors for the sources are Deep Generative Models.

The sampling procedure is a variant of MCMC called Langevin Dynamics [24], which we detail in the next chapter. This procedure requires explicit computation of the gradient of the log-probability of the source model: $\nabla_{\mathbf{s}} \log p(\mathbf{s})$, which impose a constraint for the choice of the deep generative priors.

1.3.2. Overview of Deep Generative Models

We present now briefly some of the most popular generative models, and we justify our choices of models for the separation.

GAN: As mentioned above, the GAN framework is the following: a first deep neural network creates samples that are intended to come from the training distribution by mapping an input noise variable $\mathbf{z} \sim p_z$ with tractable distribution to the data space: $G_{\theta_g}(\mathbf{z})$. A second deep neural network takes as input both training samples and examples from G_{θ_g} and output a scalar $D_{\theta_d}(\mathbf{x})$ which represents the probability that \mathbf{x} is a sample from the dataset rather than an example from G_{θ_g} . D_{θ_d} and G_{θ_g} are playing a minimax game:

$$J(\theta_d, \theta_g) = -\frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log D_{\theta_d}(\mathbf{x})] - \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_z} [\log(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z})))]$$

D_{θ_d} tries to minimize J while G_{θ_g} tries to maximize it.

GAN are currently the best performing generative model in terms of the quality of the samples generated. However, they have some pitfalls and limitations: GANs are trained for only one purpose which is to generate samples. They do not have any encoder to infer the latents hence they can not accomplish other tasks such as interpolation between datapoints or simply evaluate the log-likelihood.

VAE [16]: Variational Auto-Encoder is the second most powerful deep generative model. It assumes that the data are generated by a random process involving a latent variable \mathbf{z} . It introduces a probabilistic encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and a probabilistic decoder $p_\theta(\mathbf{x}|\mathbf{z})$ and tries to maximize a lower bound on the marginal log-likelihood $\log p_\theta(\mathbf{x})$:

$$\mathcal{L}(\theta; \phi; \mathbf{x}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_z(\mathbf{z})) + \mathbf{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]$$

where the prior $p_z(\mathbf{z})$ is set as a standard gaussian. The latent random variable \mathbf{z} is expressed thanks to the reparametrization trick: $\mathbf{z} = g_\phi(\mathbf{x}, \epsilon) = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$ and $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are encoded by a neural network $\mathbf{E}_\phi(\mathbf{x}) = (\boldsymbol{\mu}, \boldsymbol{\sigma})$. Therefore, we have $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}\mathbf{I})$. For continuous data, the probabilistic decoder is also gaussian and a decoder network $D_\theta(\mathbf{z}) = (\boldsymbol{\mu}, \boldsymbol{\sigma})$ gives the parameters to sample from $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma})$. The variational lower bound is maximized via stochastic gradient ascent on the parameters θ and ϕ . Once the encoder and decoder networks are trained the generative process is the following:

$$\begin{aligned} \mathbf{z} &\sim p_z = \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ \boldsymbol{\mu}, \boldsymbol{\sigma} &= D_\theta(\mathbf{z}) \\ \mathbf{x} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}\mathbf{I}) \end{aligned}$$

VAE is efficient for inferences of the latent variable and the variable \mathbf{x} and therefore can perform downstream tasks. However, the inferences are approximated because of the variational lower bound which can be seen as a major limitation.

Auto-Regressive Models: Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$, a sample from the data distribution. Auto-regressive models are based on the chain rule of probability:

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

Each conditional distribution $p(x_i | x_1, \dots, x_{i-1})$ is modeled as a parametric density whose parameters are functions of the previous variables $x_{1:i-1}$ (often indirectly via a hidden state). For instance, the Pixel Recurrent Neural Networks [25], which aims to generate image pixel by pixel, models the conditional distributions as multinomial distributions. Each variable can take one of the 256 possible values. They define a neural network where each output variable $\mathbf{y}_i = (y_{i,0}, \dots, y_{i,255})$ represents the value taken by the multinomial density and depends only on the variables $x_{1:i-1}$. This architecture allows to evaluate the conditional distributions in parallel.

However, there is a major limitation of all auto-regressive models: they have to generate the samples sequentially, which is troublesome when the size of the samples is large. In addition, auto-regressive models are sensitive to the order of the variables.

Normalizing Flows [7]: In this kind of models, the objective distribution p_x is represented by an invertible transformation $g_\theta = f_\theta^{-1}$ of a tractable distribution p_z . Hence, $\mathbf{x} = f_\theta^{-1}(\mathbf{z})$, where $\mathbf{z} \sim p_z$. The invertible transformation has to be differentiable and its determinant has to be tractable since its parameters will be optimized during training in order to maximize the log-likelihood of the data distribution:

$$\log(p_x(\mathbf{x})) = \log(p_z(f_\theta(\mathbf{x}))) + \log\left(\left|\det \frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}}\right|\right)$$

Score-Based Generative Models [29]: In this last type of model, a neural network is trained to approximate the score of the unknown data distribution $\nabla_x \log p_x(\mathbf{x})$. To do so, Song & Ermon [29] make use of denoising score matching [42] which consists in perturbing the data samples with random noise in order to obtain a tractable objective function (see section 2.2.2). The generating process is then based on Langevin Dynamics coupled with a simulated annealing schedule (see section 2.1.2).

Score-based generative model is very unique in the deep generative model field, notably because the training and the generative processes are not explicitly dependent. Another model which is worthy to mention is the denoising diffusion probabilistic model [12]. This model parametrized a Markov Chain trained to reverse a diffusion process in order to produce samples matching the data.

Choices of the priors: The sampling procedure requires exact calculation of the score: $\nabla_s \log p(\mathbf{s})$. Hence, GAN are not suited for our method, and VAE is not the best fit since it approximates the log-likelihood by a lower-bound. Normalizing Flows, Auto-Regressive Models and Score Based Generative Models are well suited for our method. The latter seems to be the ideal candidate since it directly estimates the score. For the

sake of comparison, we chose another type of generative model. We opted for normalizing flow models for their simplicity and efficiency over Auto-Regressive models (although they perform better in general). We describe in details these 2 models in the next section.

1.3.3. Challenges and Discussion

Our method has never been applied to audio mixture. The first challenge is to be able to train state-of-the art generative models on audio samples, which is difficult due to the high dimensionality of audio data. The second, is to be able to take advantage of these models to separate a mixture using the Langevin Dynamics sampling procedure. These two stages involve many hyperparameters, from the preprocessing of the audio samples to the settings of the sampling procedure itself. Therefore, we had to conduct many experiments to find a set of hyperparameters which was working.

This method is not widely studied but presents many advantages compared to supervised learning as mentioned in section 1.2.3. Its great generalization capacity and its flexibility could make it the next reference model for the Audio Source Separation task.

2. Unsupervised Audio Source Separation with Deep Generative Priors

2.1. Source Separation with Langevin Dynamics

2.1.1. Stochastic Gradient Langevin Dynamics

Stochastic Gradient Langevin Dynamics is inspired by Langevin Monte Carlo (LMC), a class of Markov Chain Monte Carlo techniques and Stochastic Optimization [44].

Langevin Monte Carlo: LMC [24] makes use of Langevin Dynamics, an approach for modelling Molecular systems dynamics, to sample from a complex distribution. Let's suppose we want to sample from an unnormalized posterior distribution:

$$p(\theta|X) \propto p(X|\theta)p(\theta)$$

where X are the observed variables, $p(X|\theta)$ is the likelihood and $p(\theta)$ is the prior over the parameter θ . LMC defines a new update rule for the Metropolis-Hasting algorithm:

$$\begin{aligned}\theta^* &= \theta^{(t)} + \frac{\epsilon^2}{2}(\nabla \log p(\theta^{(t)}|X) + \nabla \log p(\theta^{(t)})) + \eta_t \\ &= \theta^{(t)} + \frac{\epsilon^2}{2}(\nabla \log p(X|\theta^{(t)}) + \nabla \log p(\theta^{(t)})) + \eta_t\end{aligned}$$

where $\eta_t \sim \mathcal{N}(0, \epsilon^2)$, ϵ is the step size and the gradient is taken with respect to θ^t . The proposal distribution $q(\cdot, \theta^t)$ is therefore a gaussian distribution with mean $\theta^{(t)} + \frac{\epsilon^2}{2}(\nabla \log p(X|\theta^{(t)}) + \nabla \log p(\theta^{(t)}))$ and variance ϵ^2 . The acceptance probability is:

$$\min \left[1, \frac{p(\theta^*|X)q(\theta^*, \theta^{(t)})}{p(\theta^{(t)}|X)q(\theta^{(t)}, \theta^*)} \right]$$

Stochastic Optimization: is an optimization method which operates with random subsets of the data to optimize a parameter. For instance, the maximum a posteriori parameter θ in the previous model can be found with stochastic optimization as follows: at each step, a subset of the data X_t is chosen at random and the parameter is updated with:

$$\theta^{(t+1)} = \theta^t + \frac{\epsilon_t}{2} \left(\frac{N}{n} \nabla \log p(X_t|\theta^{(t)}) + \nabla \log p(\theta^{(t)}) \right)$$

where N is the total size of the data and n the size of the subset. ϵ is the step size at step t , the sequence of step sizes decreases after each iteration with the following properties:

$$\sum_{t=1}^{\infty} \epsilon_t = \infty \quad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty$$

The gradient is computed over only a subset of the whole dataset which is more computationally efficient especially with big data set. However, it does not capture parameter uncertainty and hence can overfit.

Stochastic Langevin Dynamics combines the two approaches to sample from a posterior distribution. It takes both the advantage of the Bayesian setting of LMC and the computational efficiency of Stochastic Optimization. Stochastic Langevin Dynamics is an iterative process similar to LMC but without the acceptance/rejection step:

$$\theta^{(t+1)} = \theta^{(t)} + \frac{\epsilon_t}{2} \left(\frac{N}{n} \nabla \log p(X_t | \theta^{(t)}) + \nabla \log p(\theta^{(t)}) \right) + \eta_t \quad (2.1)$$

where $\eta_t \sim \mathcal{N}(0, \epsilon_t)$ and ϵ_t follows the same properties as in stochastic optimization. Therefore, the rejection rate in a Metropolis Hasting Algorithm would go toward 0 as $t \rightarrow \infty$. It can be shown that the Markov Chain defines above will effectively converge toward the wanted distribution. At first, the iterative process will imitate a stochastic gradient ascent algorithm because of the noise coming from the random subset. At the end, the injected noise coming from η_t will take over and the algorithm will imitate the Langevin MH algorithm.

2.1.2. Simulated Annealing

Simulated Annealing [23] is a technique which is used to improve the performance of a Markov Chain sampler and to accelerate the mixing process. A sequence of distributions p_0, p_1, \dots, p_n is defined where p_0 is the distribution we want to sample from and consecutive distributions differ only slightly from each other. p_n is a distribution from which it is easy to sample using a Markov Chain sampler and that allows movement between all the regions in the data space.

We start by running a Markov Chain designed to converge to p_n , for some steps, starting at a random initial state. We continue by running a Markov Chain designed to converge to p_{n-1} , for some steps, starting from the last state of the previous Markov Chain. We repeat the processus until p_0 . The idea is to avoid the Markov Chain to be stuck in a mode of p_0 thanks to the freer movement in the earlier distributions.

2.1.3. Bayesian Annealed Signal Separation

The Bayesian Annealed Signal Separation (BASIS) algorithm was introduced by Jayaram et al. [15]. We consider the single channel source separation problem in the

Bayesian formulation as in 1.2.3. and we would like to sample from the posterior distribution $p(\mathbf{s}|\mathbf{x})$. The observation model is:

$$\mathbf{x} \sim \mathcal{N}(g(\mathbf{s}), \gamma^2 I)$$

where $g(\mathbf{s})$ defines the mixing process of the sources and it is considered to be known. The sources follow a distribution $p(\cdot)$ which will need to be specified.

Applying equation 2.1 with a single mix signal \mathbf{x} gives:

$$\mathbf{s}^{(t+1)} = \mathbf{s}^{(t)} + \frac{\epsilon_t}{2} (\nabla \log p(\mathbf{x}|\mathbf{s}^{(t)}) + \nabla \log p(\mathbf{s}^{(t)}) + \eta_t) \quad (2.2)$$

$$= \mathbf{s}^{(t)} + \frac{\epsilon_t}{2} \left(\frac{1}{2\gamma^2} \nabla \|\mathbf{x} - g(\mathbf{s}^{(t)})\|^2 + \nabla \log p(\mathbf{s}^{(t)}) \right) + \eta_t \quad (2.3)$$

$$= \mathbf{s}^{(t)} + \frac{\epsilon_t}{2} \left(\nabla \log p(\mathbf{s}^{(t)}) - \frac{1}{\gamma^2} \nabla g(\mathbf{s}) \mathbb{1}_c [\mathbf{x} - g(\mathbf{s}^{(t)})] \right) + \eta_t \quad (2.4)$$

where $\eta_t \sim \mathcal{N}(0, \epsilon_t \mathbf{I})$ and the gradients are taken with respect to $\mathbf{s}^{(t)}$ and $\mathbb{1}_c$ is a column vector of ones.

Annealed Langevin Dynamics: For better performance, Jayaram et al. used simulated annealing by perturbing the estimated sources with random noise. They define a sequence of prior distributions $p_{\sigma_1}, p_{\sigma_2}, \dots, p_{\sigma_L}$ where $p_{\sigma_i}(\mathbf{s})$ models the distribution of $\mathbf{s} + \epsilon_{\sigma_i}$, with $\epsilon_{\sigma_i} \sim \mathcal{N}(0, \sigma_i)$ and $\sigma_1 > \sigma_2 > \dots > \sigma_L$. This sequence induces the sequence of posterior distributions we want to sample from: $p_{\sigma_i}(\mathbf{s}|\mathbf{x}) \propto p(\mathbf{x}|\mathbf{s})p_{\sigma_i}(\mathbf{s})$. Intuitively, at high noise level, $p_{\sigma_i}(\mathbf{s})$ will be closed to a gaussian distribution hence easy to sample from, and for low noise level, $p_{\sigma_i}(\mathbf{s})$ will approximate the noiseless distribution. The resulting Langevin Dynamics is then:

$$\mathbf{s}^{(t+1)} = \mathbf{s}^{(t)} + \frac{\epsilon_i}{2} \left(\nabla \log p_{\sigma_i}(\mathbf{s}^{(t)}) - \frac{1}{\gamma^2} \nabla g(\mathbf{s}) \mathbb{1}_c [\mathbf{x} - g(\mathbf{s}^{(t)})] \right) + \eta_t$$

where $\eta_t \sim \mathcal{N}(0, \epsilon_i \mathbf{I})$

The Markov Chain above converges to $p(\mathbf{s}|\mathbf{x})$ as $\sigma_i \rightarrow 0$, $\epsilon_i \rightarrow 0$, $\gamma^2 \rightarrow 0$ and $t \rightarrow \infty$. Jayaram et al. chose the annealing parameters such that the Signal-to-noise ratio (SNR) between the expected size of the gradient of the log posterior term and the expected size of the Langevin Dynamics noise is kept constant:

$$\begin{aligned} \frac{\mathbb{E}_{\mathbf{s} \sim p_{\sigma_i}} \left[\left\| \frac{\epsilon_i}{2} \nabla \log p_{\sigma_i}(\mathbf{s}|\mathbf{x}) \right\|^2 \right]}{\mathbb{E} \left[\|\eta_t\|^2 \right]} &= \frac{\epsilon_i}{4} \mathbb{E}_{\mathbf{s} \sim p_{\sigma_i}} \left[\|\nabla \log p(\mathbf{x}|\mathbf{s}) + \nabla \log p_{\sigma_i}(\mathbf{s})\|^2 \right] \\ &\approx \frac{\epsilon_i}{4} \mathbb{E}_{\mathbf{s} \sim p_{\sigma_i}} \left[\|\nabla \log p(\mathbf{x}|\mathbf{s})\|^2 \right] + \frac{\epsilon_i}{4} \mathbb{E}_{\mathbf{s} \sim p_{\sigma_i}} \left[\|\nabla \log p_{\sigma_i}(\mathbf{s})\|^2 \right] \end{aligned}$$

by assuming the gradients w.r.t. to the likelihood and the prior are uncorrelated.

$$E_{\mathbf{s} \sim p_{\sigma_i}} \left[\|\nabla \log p(\mathbf{x}|\mathbf{s})\|^2 \right] = \frac{1}{\gamma^4} \|\nabla g(\mathbf{s})\|^2 \mathbb{E} \left[\|\mathbf{x} - g(\mathbf{s})\|^2 \right] \quad (2.5)$$

$$= \propto \frac{1}{\gamma^2} \quad (2.6)$$

Song & Ermon observed empirically in their work on Noise Conditioned Score Network (NCSN) [29] that $\mathbb{E}_{\mathbf{s} \sim p_{\sigma_i}} [\|\nabla \log p(\mathbf{s})\|^2] \propto \frac{1}{\sigma_i^2}$.

We have shown that in order to keep the SNR constant, γ^2 and σ^2 have to be chosen proportional to ϵ_i . Experimentally, ϵ_i is set equal to $2 \times 10^{-5} \frac{\sigma_i^2}{\sigma_L^2}$ and $\gamma^2 = \sigma_i^2$. The number of iterations for each noise is set to 100. The resulting BASIS Separation is presented in Algorithm 1.

Algorithm 1: BASIS Algorithm

Input: Mix Signal \mathbf{x} , $\{\sigma_i\}_{i=1}^L$, T
 $\mathbf{s}^{(0)} \sim \text{Independent } \mathcal{U}(0, 1)$
for $i = 1, 2, \dots, L$ **do**
 $\epsilon_i = 2 \times 10^{-5} \frac{\sigma_i^2}{\sigma_L^2}$
 for $t = 1, 2, \dots, T$ **do**
 $\eta_t \sim \mathcal{N}(0, \epsilon_i \mathbf{I})$
 $\mathbf{s}^{(t+1)} = \mathbf{s}^{(t)} + \frac{\epsilon_i}{2} \left(\nabla \log p_{\sigma_i}(\mathbf{s}^{(t)}) - \frac{1}{\sigma_i^2} \nabla g(\mathbf{s}) \mathbb{1}_c [\mathbf{x} - g(\mathbf{s}^{(t)})] \right) + \eta_t$

The Source Model: We assume the source components are independent:

$$\log p(\mathbf{s}) = \sum_{k=1}^K \log p(s_k)$$

$$\nabla_{\mathbf{s}} \log p(\mathbf{s}) = (\nabla_{s_k} \log p(s_k))_{k=1, \dots, K}^T$$

where s_k is the k th component of the source and T is the transpose transformation.

Deep Generative Priors: The prior distributions $\{p(s_k)\}_{k=1}^K$ are modelled with deep generative models: the Glow model [17] which is a Normalizing Flow model and the NCSN model [29], which is a score-based model.

Music mixing process: We assume a basic mixing process which is the average of the sources in the waveform domain (\mathbf{w}):

$$g(\mathbf{s}_w) = \frac{1}{K} \sum_{k=1}^K \mathbf{s}_{w,k}$$

The STFT is a linear transformation. We suppose the mel-spectrogram in amplitude (\mathbf{a}) is as well a linear transformation:

$$g(\mathbf{s}_a) = \frac{1}{K} \sum_{k=1}^K \mathbf{s}_{a,k}$$

We trained our models on the decibel scale (dB), the transformation is the following:

$$\begin{aligned}x_{dB} &= 10\log_{10}(x_a^2) = \frac{20}{\log(10)} \log(x_a) \\x_a &= \sqrt{10^{x_{dB}/10}} = \exp(\log(10)x_{dB}/20)\end{aligned}$$

Therefore, the mixing function becomes:

$$g(\mathbf{s}_{dB}) = \frac{20}{\log(10)} \log \left[\frac{1}{K} \sum_{k=1}^K \exp(\log(10)\mathbf{s}_{dB,k}/20) \right]$$

The gradient of that function is:

$$\begin{aligned}\nabla_s g(\mathbf{s}) &= \left(\frac{\exp(s_k \log(10)/20)}{\sum_{k=1}^K \exp(s_k \log(10)/20)} \right)_{k=1,2,\dots,K}^T \\&= \text{softmax}(\mathbf{s} \log(10)/20)\end{aligned}$$

2.2. Deep Generative Priors

2.2.1. The Glow model

Normalizing Flows: The Glow model belongs to a class of model called Normalizing Flow. They are likelihood-based generative models that are asked to learn a bijective transformation of the data into latent variables which are modelled with a tractable distribution. Normalizing flow models present advantages compare to Generative Adversarial Networks (GANs) [10], Auto-regressive models [25] and Variational Auto-Encoders (VAE) [16] such as exact log-likelihood evaluation, efficient inference and potential memory savings. Due to the high performances of VAE and GANs models, normalizing flow models had gained little attention. However, recent advances in normalizing flow models starting with the NICE model [7], then the Real NVP model [8] and finally the Glow Model [17] start to made them competitive to the best generative models.

Let $D = \{\mathbf{x}^{(i)}\}_{i=1,\dots,N}$ be the dataset of i.i.d random vectors $\mathbf{x}^{(i)} \sim p$. We model the distribution with parameters θ : p_θ . The objective is to minimize the negative log-likelihood:

$$\mathcal{L}(\mathcal{D}) = \frac{1}{N} \sum_{i=1}^N -\log p_\theta(\mathbf{x}^{(i)})$$

The learning process is done with regular stochastic gradient descent on the parameters of the model.

In normalizing flow model, the model is the following:

A bijective function f_θ transforms the data \mathbf{x} into a latent variable \mathbf{z} which is modelled with a tractable distribution p_z . The model distribution is therefore:

$$\begin{aligned} p_\theta(\mathbf{x}) &= p_z(\mathbf{z}) \left| \det \frac{\partial(\mathbf{z})}{\partial \mathbf{x}^T} \right| \\ &= p_z(f_\theta(\mathbf{x})) \left| \det \left(\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}^T} \right) \right| \\ \log(p_\theta(\mathbf{x})) &= \log(p_z(f_\theta(\mathbf{x}))) + \log \left(\left| \det \left(\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}^T} \right) \right| \right) \end{aligned}$$

where $\det \left(\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}^T} \right)$ is the determinant of the Jacobian matrix of f_θ

The inference (generating) process is then simply:

$$\begin{aligned} \mathbf{z} &\sim p_z \\ \mathbf{x} &= f_\theta^{-1}(\mathbf{z}) \end{aligned}$$

The function f_θ is often defined as a sequence of bijective functions which is called a normalizing flow:

$f_\theta = f_{\theta_K} \circ f_{\theta_{K-1}} \circ \dots \circ f_{\theta_1}$. The model becomes:

$$\log(p_\theta(\mathbf{x})) = \log(p_z(f_\theta(\mathbf{x}))) + \sum_{i=1}^K \log \left(\left| \det \left(\frac{\partial f_{\theta_i}(\mathbf{h}_i)}{\partial \mathbf{h}_{i-1}^T} \right) \right| \right)$$

where $\{\mathbf{h}_i\}_{i=1,\dots,K}$ is the sequence of transformed variables given by the sequence of transformations $\{f_{\theta_i}\}_{i=1,\dots,K}$ with $\mathbf{h}_0 = \mathbf{x}$ and $\mathbf{h}_K = \mathbf{z}$.

Once this model defined, the goal is now to be able to define a class of powerful bijective transformations with tractable and efficient computation of the determinant of the Jacobian.

The Glow Model Architecture

We describe now the flow of the Glow model which was the first normalizing flow model to demonstrate the ability to generate high quality samples. The Glow model has been developed for images and therefore the input data is a 3-dimensional vector: Height \times Width \times Channel.

The Glow model is built upon 3 types of layers (i.e. bijective transformations) which are assembled into a multi-scale architecture.

The Affine Coupling Layer: was introduced by Dinh et al. in the NICE model [7] and further improve in the Real NVP model [8]. Let \mathbf{x} be a 3-dimensional vectors $h \cdot w \cdot 2c$, we define the split(.) operation which splits the vector in 2 halves along the channel

dimension and the $\text{concat}(\cdot, \cdot)$ operation which concatenates two vectors on the channel dimension. The transformation of the Affine Coupling Layer is the following:

$$\begin{aligned}\mathbf{x}_1, \mathbf{x}_2 &= \text{split}(\mathbf{x}) \\ s_\theta, t_\theta &= f_\theta(\mathbf{x}_1) \\ \mathbf{h}_2 &= s_\theta \odot \mathbf{x}_2 + t_\theta \\ \mathbf{h}_1 &= \mathbf{x}_1 \\ \mathbf{h} &= \text{concat}(\mathbf{h}_1, \mathbf{h}_2)\end{aligned}$$

where $f_\theta(\cdot)$ is a Neural Network. We used the following architecture for f_θ : 3 convolutional layers with 2 batch normalization layers after the first and second ones. The filters size is 3×3 , the input is padded with zeros so the output of the layer has the same size as the input (in the spatial dimensions). The last convolutional layer has $2 \cdot c$ filters. Its output is split along the channel dimension to give $\log(s_\theta)$ and t_θ , 2 vectors of the same size as \mathbf{x}_2 (and \mathbf{x}_1). $\log s_\theta$ is transformed with a hyperbolic tangent function before being fed to the affine coupling layer. The log-determinant of this transformation is simply $\sum_{i,j,k} \log(s_\theta)$. The inverse transformation is:

$$\begin{aligned}\mathbf{h}_1, \mathbf{h}_2 &= \text{split}(\mathbf{h}) \\ s_\theta, t_\theta &= f_\theta(\mathbf{h}_1) \\ \mathbf{x}_2 &= (\mathbf{x}_2 - t_\theta) / s_\theta \\ \mathbf{x}_1 &= \mathbf{h}_1 \\ \mathbf{x} &= \text{concat}(\mathbf{x}_1, \mathbf{x}_2)\end{aligned}$$

The Invertible 1×1 convolution: Convolution Layers in Deep Neural Networks are wildly used, especially when the input data are images. Let consider a 3-dimensional input vector \mathbf{h} of size $h \times w \times c$. The convolution layer consists in a set of learnable filters, 3-dimensional vectors with are small in the spatial dimension (height and width) compared to the input and have the same channel dimension as the input. Each filter slides along the spatial dimension of the input and the dot product between the input values and the filters values are computed at each position. Figure 2.1 shows the convolution (on a single channel) of a filter of size 3×3 resulting in an output of size 2×2 .

1×1 is a convolution with filters of size $1 \times 1 \times c$. The action of a $1 \times 1 \times c$ filter on the input gives a 2-dimensional vector of size $h \times w$. Since, we want our transformation to be invertible, the output needs to have the same dimension as the input, hence we need c filters.

The convolution transformation can be implemented as a matrix multiplication. The filters can be stretched out into rows, giving a matrix \mathbf{W} of size (c, c) . The spatial dimensions of the input can be also stretched out into rows giving a matrix \mathbf{X} of size $(c, h \cdot w)$. The convolution of the c filters is then equivalent to \mathbf{WX} which can then be

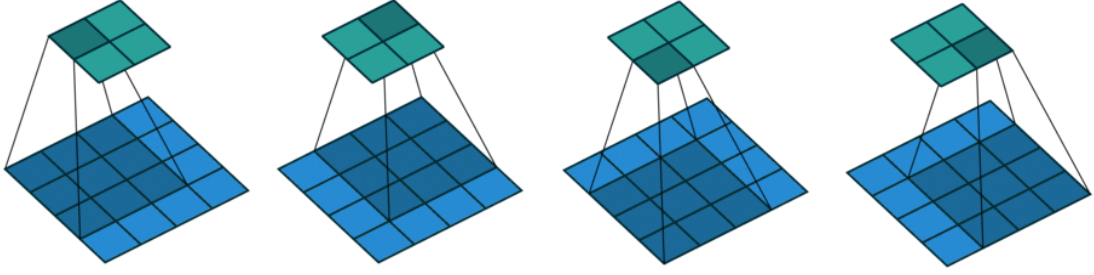


Figure 2.1.: Convolution Animation (from left to right) of a single filter of size 3×3 on an input of size 4×4 (Dumoulin et al., 2016 [9])

reshaped into a 3 dimensional vectors. The Jacobian is:

$$\frac{\partial \text{conv}(\mathbf{h}, \mathbf{W})_{i,j,k}}{\partial \mathbf{h}_{i',j',k'}} = \begin{cases} W_{c,c'} & \text{if } i, j = i', j' \\ 0 & \text{otherwise} \end{cases}$$

So the Jacobian matrix is a diagonal block matrix with $h \cdot w$ matrices \mathbf{W} on the diagonal. Therefore the Jacobian determinant is $\det(\mathbf{W})^{h \cdot w}$. The inverse transformation given an input \mathbf{Y} is $\mathbf{W}^{-1}\mathbf{Y}$.

To ensure the invertibility of \mathbf{W} and to reduce the cost of computing $\det(\mathbf{W})$, \mathbf{W} can be parametrized in its LU decomposition:

$$\mathbf{W} = \mathbf{P}\mathbf{L}(\mathbf{U} + \text{diag}(\mathbf{s}))$$

where \mathbf{P} is a permutation matrix, \mathbf{L} is a lower triangular matrix with ones on the diagonal, \mathbf{U} is an upper triangular matrix with zeros on the diagonal and \mathbf{s} is a vector. We have then $\det(\mathbf{W}) = \prod_{k=1}^c s_k$. $\mathbf{P}, \mathbf{L}, \mathbf{U}$ and \mathbf{s} are computed from an initial random rotation matrix, and only the coefficients of \mathbf{L}, \mathbf{U} and \mathbf{s} are optimized during training (with the constraint $s_k > 0 \ \forall \ k = 1, \dots, c$).

The idea behind the 1×1 invertible convolution is to create a general learnable permutation operation in the channel dimension. It makes sense in the case of the glow architecture, as you will see later in the multi-scale architecture description, because the spatial dimensions are traded for the channel dimension.

The ActNorm Layer: is a normalization layer with data dependent optimization. The normalization operates on the channel dimension. Let $\mathbf{x} = (x_{h,w,c})_{h,w,c}$ be a 3-dimensional vector of dimension $H \times W \times C$. The layer has the following parameters: \mathbf{s} and \mathbf{b} which are 1-dimensional vectors of size C . The layer operates the following transformation:

$$\mathbf{h} = f_{s,b}(\mathbf{x}) = \mathbf{x} \odot \mathbf{s} + \mathbf{b}$$

where the addition and multiplication operate on the channel dimension. The parameters \mathbf{s} and \mathbf{b} can also be seen as 3-dimensional vectors with all the elements in the channel c having values s_c and b_c respectively and \odot is the term to term multiplication.

The specificity of that layer is that the parameters are initialized with a random minibatch of the dataset $\{\mathbf{x}^{(i)}\}_{i=1,\dots,n}$ such as the post-transformation variables have unit mean and variance per channel. We have then:

$$\begin{aligned}\boldsymbol{\mu}_{init} &= \frac{1}{n \times H \times W} \sum_{i=1}^n \sum_{h=1}^H \sum_{w=1}^W \mathbf{x}_{k,w}^{(i)} \\ \boldsymbol{\sigma}_{init}^2 &= \frac{1}{n} \sum_{i=1}^n \left(\frac{1}{H \times W} \sum_{h=1}^H \sum_{w=1}^W \mathbf{x}_{k,w}^{(i)} - \boldsymbol{\mu}_{init} \right)^2 \\ \mathbf{s}_{init} &= \frac{1}{\boldsymbol{\sigma}_{init}} \\ \mathbf{b}_{init} &= -\frac{\boldsymbol{\mu}_{init}}{\boldsymbol{\sigma}_{init}}\end{aligned}$$

The inverse transformation is simply: $\mathbf{x} = (\mathbf{h} - \mathbf{b})/\mathbf{s}$. The jacobian matrix is also simple:

$$\frac{\partial h_{h,w,c}}{\partial x_{h',w',c'}} = \begin{cases} s_c & \text{if } h, w, c = h', w', c' \\ 0 & \text{otherwise} \end{cases}$$

The Jacobian matrix is diagonal. Its determinant is the product of its diagonal elements and the log determinant of the Jacobian is finally: $H \times W \times \sum_{c=1}^C \log(|s_c|)$.

This kind of data dependent initialization allows building deeper model and accelerates the training.

The Multi-Scale Architecture: These 3 layers are stacked together into a multi-scale architecture shown on Figure 2.2. A step of flow is defined as a sequence of an Actnorm layer, an Invertible 1×1 Convolution layer and an Affine Coupling Layer. K steps of flow are stacked to form a level of flow, the flow is built with L levels of flow. Before each level of flow, a squeezing operation is applied. The squeezing operation divides the spatial dimensions of the input into sub-squares of size $2 \times 2 \times c$ and reshapes them into $1 \times 1 \times 4c$ vectors. The input vector is therefore transformed into a vector of size $\frac{h}{2} \times \frac{w}{2} \times 4 \cdot c$, trading the spatial size for the channel dimension. After each level of flow (except the last), the output is split in halves along the channel dimension and one half is factored out from the flow that will be concatenated to the final output.

Fine-tuning the model for the BASIS algorithm: Once trained, the gradient of the log density distribution can be automatically computed thanks to the automatic differentiation of the network. However, we need to estimate the gradient of the log-density at different noise level and therefore we need to train a model for each noise level. We can take advantage of fine-tuning to accelerate the training. We first train a model on the perturbed dataset at noise σ_L (the smallest one) which requires few hundred epochs. Then we train the model with the perturbed dataset at noise σ_{L-1} by initializing the weights of the network with the ones from the first model. We can repeat the same

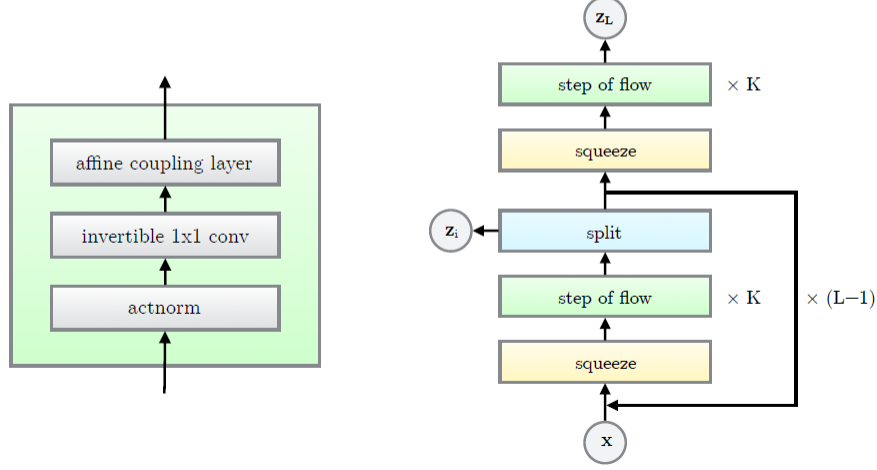


Figure 2.2.: The Multi-scale Architecture of the Glow model (Kingma et al., 2018 [17])

process with σ_{L-2} using the weight of the σ_{L-1} model, until σ_1 . Using this technique, the models converge in a few dozen epochs.

2.2.2. Noise Conditioned Score Network

The NCSN [29] is a score-based generative model which aims to model with a neural network s_θ , the score of a probability density $p(x)$, which is $\nabla_x \log(p(x))$, at different noise level.

Given a data sample \mathbf{x} and a noise σ , the sample is perturbed with a random noise: $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon_\sigma$, where $\epsilon_\sigma \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. The score network s_θ is trained to estimate $\nabla_{\tilde{\mathbf{x}}} \log(p_\sigma(\tilde{\mathbf{x}}))$ given $\tilde{\mathbf{x}}$ and σ . By using a decreasing sequence of noise $\sigma_1, \sigma_2, \dots, \sigma_L$, we can sample from the true distribution with Anneal Langevin Dynamics, i.e. equation 2.1 and Algorithm 1 except that there is no Likelihood term:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \frac{\epsilon_i}{2} s_\theta(\mathbf{x}^{(t)}, \sigma_i) + \eta_t$$

Score Matching: is a technique which allows estimating the score without training a model to estimate the true density probability. The function to minimize is $\frac{1}{2} \mathbb{E}_{p(\mathbf{x})} [\|s_\theta(x) - \nabla_x \log(p(x))\|^2]$ which is proved to be equivalent to $\mathbb{E}_{p(\mathbf{x})} \left[\text{tr}(\nabla_{\mathbf{x}} s_\theta(\mathbf{x})) + \frac{1}{2} \|s_\theta(\mathbf{x})\|^2 \right]$. By perturbing the samples with random noise, the conditional distribution $q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ is tractable and the objective function to estimate the score of $q_\sigma(\tilde{\mathbf{x}}) = \int q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$ is proved to be

equivalent to:

$$\frac{1}{2} \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})} \left[\|s_\theta(\mathbf{x}, \sigma) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})\|^2 \right] \quad (2.7)$$

The optimal score network s_{θ^*} that minimize the equation 2.7 satisfies $s_{\theta^*}(\tilde{\mathbf{x}}, \sigma) = \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})$. When the noise is sufficiently small, $q_\sigma \approx p$ and $s_{\theta^*}(\mathbf{x}, \sigma) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

There are some issues with score estimation that can cause the model to be inconsistent. Score estimation works well when the support of the dataset is the whole space but will be inconsistent if the data is constrained to a low dimensional manifold. Many datasets happen to be constrained in a lower dimensional space. In addition, the score estimation tends to be inaccurate in low density regions due to lack of data samples. Perturbing the data with random noise enables to circumvent these 2 issues. The perturbed data is not confined to a low dimensional manifold anymore and large noise can fill the low density regions.

The data is perturbed with normal noise so $q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I})$. The objective function for a given noise σ is then:

$$l(\theta, \sigma) = \frac{1}{2} \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{\mathcal{N}(\mathbf{x}; \tilde{\mathbf{x}}, \sigma^2 \mathbf{I})} \left[\left\| s_\theta(\tilde{\mathbf{x}}, \sigma) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} \right\|^2 \right]$$

And finally, the global objective is:

$$\mathcal{L}(\theta, \{\sigma_i\}_{i=1}^L) = \frac{1}{L} \sum_{i=1}^L \lambda(\sigma_i) l(\theta, \sigma_i)$$

$\lambda(\sigma)$ is chosen to be equal to $\frac{1}{\sigma^2}$ because empirically, $\|s_\theta(\mathbf{x}, \sigma)\|^2 \propto \frac{1}{\sigma}$ and so $\lambda(\sigma)l(\theta, \sigma)$ does not depend on σ .

For a minibatch of n samples $\{\mathbf{x}^{(i)}\}_{i=1}^n$, we chose n noises randomly $\{\sigma^{(i)}\}_{i=1}^n$ among the L levels. We then compute the perturbed samples with random noise: $\tilde{\mathbf{x}}^{(i)} = \mathbf{x}^{(i)} + \epsilon_{\sigma^{(i)}}$. The loss function to minimize during training via gradient descent is then:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \frac{1}{(\sigma^{(i)})^2} \left\| s_\theta(\tilde{\mathbf{x}}^{(i)}, \sigma^{(i)}) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{(\sigma^{(i)})^2} \right\|^2$$

The NCSN model estimates directly the gradient of the log density at different noise level. Therefore, the model can be used directly in the BASIS algorithm and there is no need for-fine tuning.

The Score Network: There is no constraint for the architecture of the score network. We used the RefineNet design [20] with Dilated Convolution (see Figure 2.3) instead of regular Convolution. We detail the exact architecture of the network in the Appendix.

To condition the score with the noise level, we used a modified Instance Normalization Layer. Let \mathbf{x} be an input vector of size $h \cdot w \cdot c$, we perturb it with the i th noise σ_i giving $\tilde{\mathbf{x}}$.

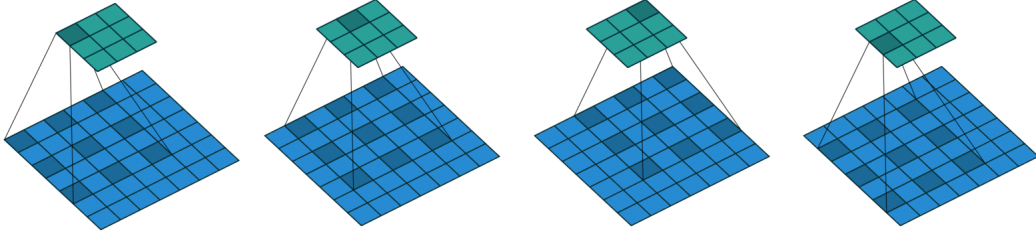


Figure 2.3.: Dilated Convolution (Dumoulin et al., 2016 [9])

$\tilde{\mathbf{x}}_k$ is the vector of the k th channel of $\tilde{\mathbf{x}}$, μ_k and s_k are the mean and standard deviation of $\tilde{\mathbf{x}}_k$ and m and ν are the mean and standard deviation of $\{\mu_k\}_{k=1}^c$. Let $\gamma \in \mathbb{R}^{L \times c}$, $\beta \in \mathbb{R}^{L \times c}$ and $\alpha \in \mathbb{R}^{L \times c}$ be learnable parameters which map each of the L noise levels to vectors of size c . The modified instance normalization conditioned by the i th noise is:

$$\mathbf{z}_k = \gamma_{i,k} \frac{\tilde{\mathbf{x}}_k - \mu_k}{s_k} + \beta_{i,k} + \alpha_{i,k} \frac{\mu_k - m}{\nu}$$

2.3. Signal Processing for Generative Models

Supervised state-of-the art source separation models are operating directly in the time domain. However, modelling long range time dependencies still remains a challenge, which is the case for high sampling rate audio where a few seconds extract represents hundreds of thousands of time steps. For unsupervised and generative audio source separation, time-frequency representation can be an advantage. Two-dimensional representation are easier to model than long-range time signal and one can leverage the recent advances in deep learning on images.

Time-frequency representation of a discrete-time signal is called a spectrogram. A spectrogram is obtained by taking the Short Time Fourier Transform (STFT) of the signal. The STFT divides the time signal into frames of equal length and compute the Discrete Fourier Transform (DFT) of the frames. The frames can overlap and are usually windowed by a window function such as the hann window. The Fourier transform of each segment is then computed. The input of the STFT is a real value time signal $\mathbf{y} = \{y_1, y_2, \dots, y_T\}$ and the output is a complex matrix $X = (x_{f,t})_{f,t}$ where f is the frequency bin and t is the frame index. The number of frames and frequency bins depend on the total length of the signal and on the choice of the frame size and the overlapping size between each frame. The DFT is usually done using a Fast Fourier Transform (FFT) algorithm or a Sliding DFT [14]. Finally, the magnitude or power magnitude of the resulting matrix is taken and it forms a spectrogram.

Stevens, Volkman, and Newman introduced the mel scale [32], which match our perception of sounds at different frequencies. It is therefore common to project the frequency bins of a power STFT spectrogram onto the mel-scale using a mel-filter bank. The result is simply called a Mel-spectrogram.

Inversion of magnitude spectrograms is obviously not perfect since the phase of the STFT is missing. However, various algorithms such as the griffin algorithm [11] or gradient based approach [4] can recover time signal from spectrograms with reasonable approximations and artifacts. In the context of Source Separation, it is common to use the phase of the mixture with the estimated magnitude spectrogram of the source to inverse the STFT. In addition, Single-channel Wiener Filter (SWF) or multi-channel Wiener Filter (MWF) are two popular techniques to further enhance the separation performance [38]. Wiener filtering ensures the estimated sources are the sum of the mixture. In our case, we will use the SWF which is simply:

$$\hat{s}_j(f, t) = \frac{|\hat{s}_j(f, t)|^2}{\sum_{j=1}^K |\hat{s}_j(f, t)|^2} x(f, t)$$

where $|\hat{s}_j(f, t)|^2$ is the estimated STFT power spectrogram and $x(f, t)$ is the STFT of the mixture signal.

3. Experiments

We implemented the Glow and NCSN models and the BASIS Algorithm using Tensorflow 2 [1]. We also implemented a preprocessing pipeline for the data using the librosa library [22]. All the code is in our github repository. ¹.

3.1. Results on Toy Datasets

We first tried to train the generative models on Toy Datasets: the mnist database [6] (handwritten digit) and Cifar-10 [19], colour images in 10 classes. The first dataset is 32×32 black and white images, the second is $32 \times 32 \times 3$ colour (3 channels) images.

These experiments were used to make sure our implementation was working as desired. We could also test which kind of preprocessing was working the best. For instance, for the glow model, rescaling the data values between -0.5 and 0.5 gave the best results. For the NCSN model, the $[0, 1]$ range was the best performing.

For likelihood-based model, a popular metric to measure the quality of the generated samples is the bits-per-pixels negative likelihood. Let $p(\mathbf{x})$ be the density probability of the image \mathbf{x} , in the $[0, 255]$ range, the bits-per-pixels is simply: $b(\mathbf{x}) = \frac{-\log p(\mathbf{x})}{D}$ where D is the number of pixels of the image. We can then compute the average bits-per-pixel returned by the model. Table 3.1 reports the average bits-per-pixels on the testing set for different flow models.

	MNIST	CIFAR-10
Glow (author's implementation)	1.05	3.35
Glow (ours implementation)	1.05	3.35
Real NVP	1.06	3.49
NICE	4.36	—

Table 3.1.: Bits-per-dimension for different normalizing flow models including our implementation on MNIST and CIFAR-10 testing set

We were able to reproduce the results of the articles using the exact same hyperparameters, proving that our implementation was working as intended.

The NCSN does not return the likelihood of the images, hence it is more difficult to evaluate quantitatively. A basic but practical method is to simply look at some generated samples to see if they are realistic. Figure 3.1 shows some generated samples of the

¹<https://github.com/SamArgT/AudioSourceSep>

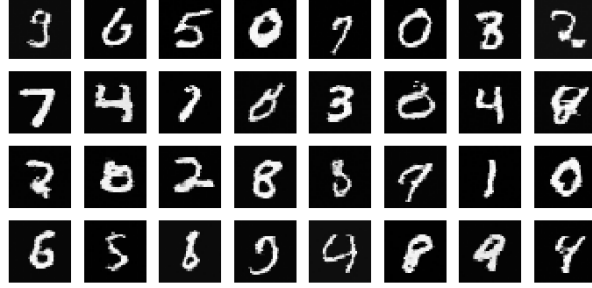


Figure 3.1.: Generated Samples from the Glow model trained on the MNIST model

Glow model trained on the MNIST dataset.

In order to show that the models are learning to generate new samples and not simply memorizing the training images, we can look at the nearest neighbours of the generated samples in the training set. If the nearest neighbours are too close to the generated samples, we might have a generalization issue (overfitting).

We then tried to separate mixture of the images using the pre-trained generative models and the BASIS algorithm. We used 10 noises levels, from $\sigma_1 = 1$ to $\sigma_{10} = 0.01$ logarithmically. The mixture is simply the average of two random images from the testing set. The Glow model is fine-tuned at each noise level. We report in Table 3.2 reports the average Peak signal-to-noise ratio (PSNR) of 1000 pairs of mixed images. For an image I and its estimate K , the PSNR is defined by:

$$\text{PSNR} = 20 \log_{10}(\text{MAX}_I) - 20 \log_{10}(\text{MSE}(I, K))$$

where $\text{MSE}(I, K)$ is the mean square error between I and K and MAX_I is the maximum possible value of I (usually 255 or 1).

Examples of the separation for the MNIST dataset are presented in Figure 3.2

	MNIST
Glow	21.2
NCSN	28.5

Table 3.2.: PSNR of the BASIS on mixture of MNIST images with Glow and NCSN as priors (our implementation). We were able to get similar results as in [15]

True x1	True x2	Mix: $(x1 + x2) / 2$	Separated x1	Separated x2
9	4	9	4	9
7	3	7	7	3
7	6	7	7	6
3	3	3	3	3
2	1	2	2	1
0	8	0	8	0
6	1	6	1	6
3	9	3	3	9
8	6	8	8	6
7	2	7	7	2

Figure 3.2.: Separated mixture from the Glow model trained on MNIST. The first two columns are the ground truth images. The 3rd column contains the mixture and the last two columns are the estimates.

	Song Title	Duration
Training Set	Bach Sonata 2	12'59"
	Bach Sonata 3	16'17"
	Bach Sonata 4	18'01"
	Beethoven Sonata 4 Op 23	26'07"
	Beethoven Sonata 5 Op 24	25'41"
	Beethoven Sonata 9 Op 47	42'16"
	Beethoven Sonata 10 Op 96	24'05"
Training Set Total Duration		165'26"
Testing Set	Bach Sonata 5	16'33"
	Beethoven Sonata	24'33"
Testing Set Total Duration		41'06"
Dataset Total Duration		206'32"

Table 3.3.: Dataset

3.2. The Audio Data

We chose to use music data that was generated only for this project. The data is being generated by Shelley Katz ².

He has developed software to render audio performances from MIDI data, and has provided a selection of Bach and Beethoven violin Sonata for the project. We have 9 songs of Beethoven and Bach Sonata which represent 1 hour 40 minutes of audio, consisting of mixtures and separate audio files for violin and piano, as well as the MIDI files for these instruments (which will not be used in this project). Table 3.3 presents how we separate the dataset between the Training Set and the Testing Set.

3.2.1. Data Preprocessing and Postprocessing

We used the following parameters to build the melspectrograms:

- Time duration of each extract: 2.04 seconds
- Sampling Rate: 16 KHz
- FFT windows size: 2048
- hop length: 512
- number of mel bins: 96

These parameters yield 6072 spectrograms of size 96×64 . There are 4863 spectrograms in the training set and 1209 in the testing set. Bigger spectrograms would give better

²<https://www.symphonova.com/biography/>

quality samples when inverting to the waveform domain but it would have required longer training and potentially more computational power.

In addition, we constrained the spectrograms to a frequency range [125Hz, 7600Hz] and an amplitude range $[-100\text{dB}, 20\text{dB}]$ by clipping higher and smaller values. Controlling the amplitude range allows to scale the spectrograms to a $[0, 1]$ range or to use some preprocessing function such as the logit function. Consequently, we can also control the range of the output by clipping and rescaling.

We have tried many preprocessing techniques: power spectrograms and Decibel spectrograms, in $[0, 1]$ range, in $[-0.5, 0.5]$ range, and logit function applied after rescaling between 0 and 1.

3.2.2. Inverting the Spectrograms to the Time Domain

To convert the spectrograms to the waveform domain we tried 3 techniques:

- The griffin algorithm
- Use the phase of the mixture to inverse back to the time domain (iSTFT).
- The Single Channel Wiener Filter (SWF), coupled with iSTFT.

For the last 2 methods, we need first to approximate the STFT amplitudes from the mel-spectrograms.

As the pre-trained NCSN model takes as input 2.04 seconds long spectrograms, we need to cut the mixture signal into 2.04 seconds extracts and separate each of these extracts separately. Then, we tried two different methods to stick back together the estimated components:

- First inverting the resulting spectrograms and then concatenating them in the time domain. We called this method griffin/iSTFT/SWF per frame.
- First concatenating the spectrograms in the frequency domain into one big spectrogram and then inverting it. We called this method griffin/iSTFT/SWF whole.

3.3. Results on Spectrograms

We present in this section the different experiments we made and results we obtained with the generative priors and the BASIS algorithm on our Audio dataset. Some results can be listened on to the git-hub page of the project:

<https://samargt.github.io/AudioSourceSep/>

3.3.1. Training the Generative Priors

The models are trained until the validation loss reaches a plateau which was around 500 epochs for both models. We saved the weights regularly and took the model with the smallest validation loss.

Glow: We start by training the Glow model on the Piano Spectrograms. Table 3.4 presents the bits-per-pixel for different hyperparameters and preprocessing functions. Figure 3.3 compare spectrograms from the dataset and generated spectrograms from the Glow model.

Hyperparameters	Preprocessing	Bits-per-pixel
$L = 3, K = 32, \text{filters} = 512$	Power + Rescale	2.54
$L = 3, K = 32, \text{filters} = 512$	Power + logit	2.70
$L = 3, K = 48, \text{filters} = 512$	Power + Rescale	2.33
$L = 3, K = 48, \text{filters} = 512$	Power + logit	2.66
$L = 4, K = 48, \text{filters} = 512$	dB + Rescale	1.96
$L = 3, K = 40, \text{filters} = 512$	dB + Rescale	1.89

Table 3.4.: Average bits-per-pixel on the testing set of the piano spectrograms. K , L and filters are respectively the number of steps of flow, the number of levels and the number of filters in the Glow architecture. For the Preprocessing: power or dB are the scales of the magnitude of the spectrograms (power of 2 or decibels respectively). Rescale means the spectrograms are scaled between $[0, 1]$, logit means the logit function: $\log \frac{x}{1-x}$ is applied after rescaling.

The model needs to be deep enough to be able to learn how to generate spectrograms but too deep network leads to an unstable training. Moreover, the Decibel scale is clearly a better preprocessing choice than the power scale, as well as just rescaling the magnitudes between $[0, 1]$.

Training instability of the Glow model We noticed that the training of the Glow model can be unstable, especially when we increased the depth of the architecture. We encountered spikes in the loss after it has reached a plateau as show in Figure 3.4. We didn’t find the solution to this issue except reducing the depth and doing early stopping.

NCSN We then trained NCSN models with different hyperparameters and preprocessing. We fixed the number of noise levels to $L = 10$, $\sigma_1 = 0.01$, $\sigma_L = 1.0$ and $T = 100$. Table 3.5 reports the validation loss of the models for different number of filters. We can not directly compare the loss for different preprocessing technique contrary to the Glow model. After some experiments, we clearly saw that the best preprocessing method for the NCSN model was using the Decibel scale and rescaling the spectrograms between 0 and 1.

The NCSN performance does not depend too much on the number of filters chosen. We decided to take the biggest model with 192 filters for the first Refine Layer.

Better Performance and the Decibel Scale: We noticed the decibel scale is better suited for training the generative models. On Figure 3.5 we show the distributions of the magnitudes of the spectrograms. We see clearly that for power spectrograms, the

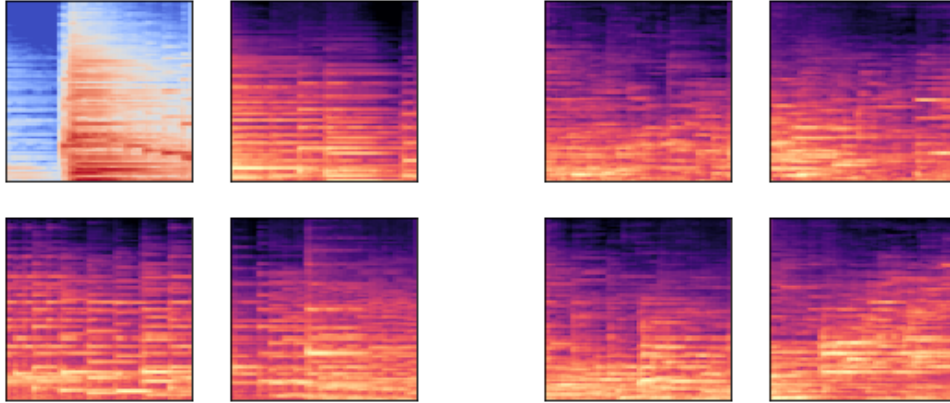


Figure 3.3.: Comparison of the spectrograms from the dataset (left) and generated spectrograms (right) from the glow model with hyperparameters: $K = 40$, $L = 3$, filters= 512, dB + rescale to $[-0.5, 0.5]$. If we look carefully, you can notice that the generated spectrograms are a bit more blurry and less sharp. When listening to it, you clearly can hear the difference;

Hyperparameters	Validation Loss
filters = 192	343
filters = 96	375
filters = 64	384

Table 3.5.: Average loss on the testing set of the piano spectrograms for the NCSN model.

magnitudes are concentrated in a subspace and there are many low density regions. It is not the case for decibel spectrograms whose magnitudes are more evenly distributed. This observation can explain why the decibel scale works better than the power scale.

Inefficiency of the Logit Transformation: Applying the logit function after having rescaling the spectrograms does not give any good results. The logit function enlarges the range of data and intuitively for the NCSN model, it makes the Langevin Dynamics harder to mix.

Computational power limitation and Batch size: The size of the models (number of parameters) was limited by the computational power available, and we had to trade off the batch size with the model size. To train the best performing NCSN model, we had to reduce the batch size to 32. We couldn't overfit the training set with the Glow model, but we couldn't either increase the size of the model due to computational power

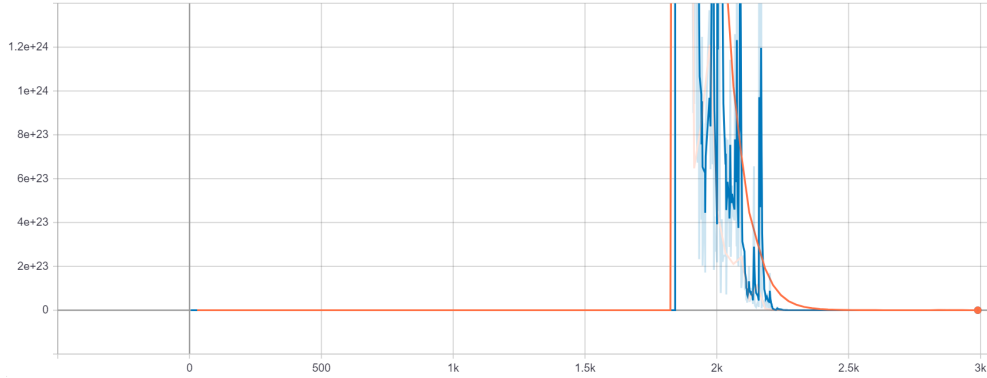


Figure 3.4.: Spikes in the loss during training of the Glow model. The Orange line is the validation loss and the blue line is the training loss.

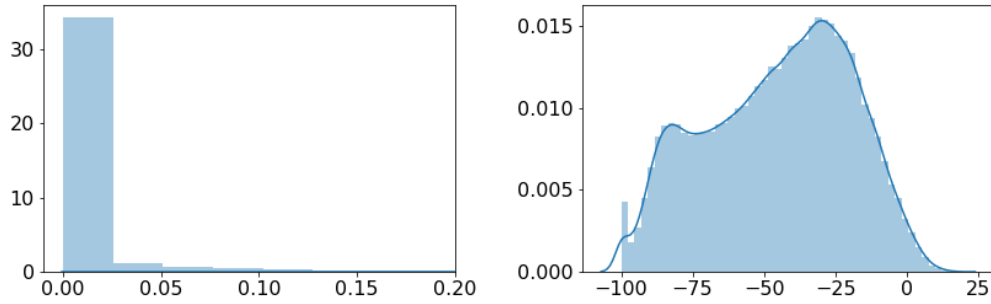


Figure 3.5.: Distribution of the power spectrograms magnitudes (left) and the decibel spectrograms magnitudes

limitation.

Comparison of the Glow and the NCSN models: We can compare only qualitatively the two models by looking and listening to the generated samples. The NCSN model yields clearly better generated samples than the Glow model. Moreover, the NCSN converges faster and requires fewer parameters than the Glow model. Figure 3.6 shows 9 spectrograms generated by the two models. The NCSN ones look clearly sharper and when listening to them, they sound more like a real piano than the Glow ones. Therefore, we decided to focus our effort on the NCSN model for the separation task.

3.3.2. Separation Performance

Vincent et al. [40] developed the metrics to evaluate the quality of source separation models. Initially developed for blind source separation models, these metrics are now used for almost every kind of models. Popular metrics are the Signal to Distortion Ratio

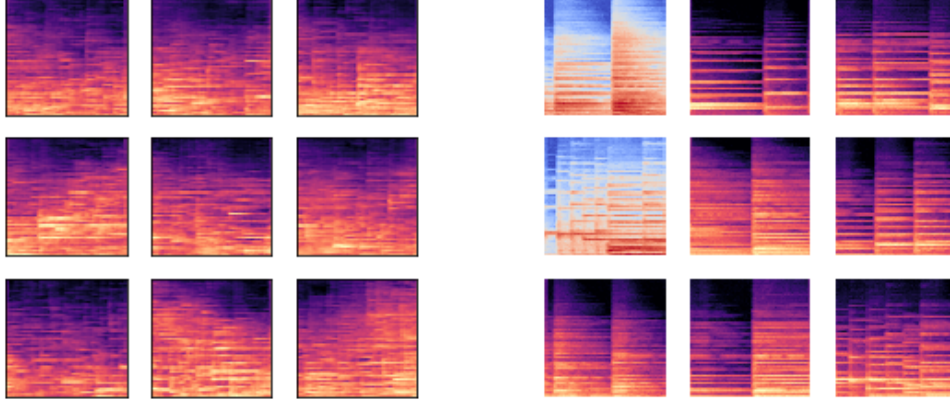


Figure 3.6.: Comparison between generated samples from the Glow model (left) and the NCSN model (right)

(SDR), the Signal to Noise Ratio (SNR), the Signal to Inference Ratio (SIR) and the Signal to Artifacts Ratio (SAR). These metrics aim to be independent and to measure different aspect of the separation quality: the absence of interferences for the SIR, the absence of sensor noises for the SNR, the absence of artifacts for the SAR and the absence of forbidden distortions for the SDR.

The computation of these metrics require to know the true sources and the noise signal (if there is one). One should choose also a family of allowed distortions. The estimated source \hat{s}_j , a time vector, is then decomposed into:

$$\hat{s}_j = s_{target} + e_{interf} + e_{noise} + e_{artif} \quad (3.1)$$

- $s_{target} = f(s_j)$, with f an allowed distortion and s_j is the true source
- $e_{interf}, e_{noise}, e_{artif}$ are the interference, noise and artifacts errors terms coming from the unwanted sources, the sensors noises and forbidden distortions respectively.

The decomposition into the 4 terms depends on the family of allowed distortion chosen. Each of them are computed using the true source and the estimated source. For Time-invariant gains allowed distortions, the decomposition is based on orthogonal projection of the estimated source onto the true sources sub-space. For instance, s_{target} is easy to compute: $s_{target} = \langle \hat{s}_j, s_j \rangle s_j / \|s_j\|^2$ For time-invariant allowed distortions, the projectors have to be modified to take into account the delay between the s_{target} and s_j . In a similar fashion projectors for time-varying gains or filters allowed distortions have to be adapted.

Once the estimated source decomposed, we can compute the metrics as follows:

$$\begin{aligned}
SDR &:= 10 \log_{10} \frac{\|s_{target}\|^2}{\|e_{interf} + e_{noise} + e_{artif}\|^2} \\
SIR &:= 10 \log_{10} \frac{\|s_{target}\|^2}{\|e_{interf}\|^2} \\
SNR &:= 10 \log_{10} \frac{\|s_{target} + e_{interf}\|^2}{\|e_{noise}\|^2} \\
SAR &:= 10 \log_{10} \frac{\|s_{target} + e_{interf} + e_{noise}\|^2}{\|e_{artif}\|^2}
\end{aligned}$$

Although these metrics are widely used, they have some pitfalls. Stoller et al. [33] remarked that when the source is near silent, the SDR is very low which causes a drop in the overall performance. Jayaram et al. [15] noticed that the true sources of a mixture are not always identifiable, i.e., different separation can look plausible and therefore the SDR is not always the good measure of the quality of a separation.

In Audio Source Separation, Human evaluation is often used through a Mean Opinion Score (MOS) survey. Although MOS is subjective and more difficult to reproduce, it can be really useful to detect artefacts or to make a direct comparison between different models. It demonstrates also the limitation of the SDR or other metrics. In Defossez et al. [5], the MOS survey gave a better quality rating to the DEMUCS model than to the Conv-Tasnet, despite a lower overall SDR.

Various datasets exist to evaluate the performances of a model and compare them with others. A popular is the MUSDB18 dataset [34], which gather 10 hours music tracks of different genres along with their isolated drums, bass, vocals and others stems.

Oracle Systems for Source Separation: Oracle performances are useful references to compare with any separation method. We describe the Ideal Binary Mask (IBM) and the Ideal Ratio Mask (IRM). These two oracles operate in the time-frequency domain, given the true sources and the mixture, they output a mask for each source to apply to the mixture to obtain the estimated sources.

The **Ideal Binary Mask** for source j is the following:

$$M_j(f, t) = \begin{cases} 1 & \text{if } \frac{|s_j(f, t)|^\alpha}{|x(f, t)|^\alpha} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

where α is the fractional power of the spectrograms, usually 1 for energy spectrogram and 2 for power spectrograms.

The **Ideal Ratio Mask** for source j is the following:

$$M_j(f, t) = \frac{|s_j(f, t)|^\alpha}{\sum_{k=1}^K |s_k(f, t)|^\alpha}$$

Results: Once the NCSN models are trained on each source, we can run the BASIS algorithm. We separated the first minute of the Beethoven Sonata 1 from the testing set. Figure 3.7 compares the performance of the separation using different inversion method. Figure 3.8 compares the true spectrograms and the estimated ones for the piano and violin components.

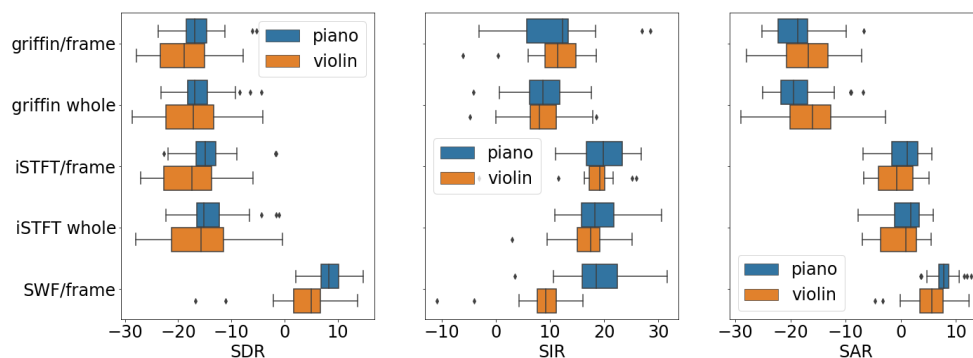


Figure 3.7.: Inversion Methods Performance

For the Signal to Distortions Ratio, the SWF technique is clearly the best one. For the Signal to Interferences Ratio and the Signal to Artifacts Ratio, using the phase of the mixture (iSTFT or SWF) improve clearly the performance of the separation. However, one should note that the SWF technique gives worst result than the simple iSTFT for the SIR metric, and especially for the violin track. This appears clearly when listening to the estimated sources: there are many interferences (i.e., piano interferences) in the violin SWF estimate compare to the iSTFT estimate. Finally, it is not clear which gives the best separation between inverting frame by frame or the whole extract, and we chose arbitrarily to use the frame method for further analysis.

We then compared our separation with Oracle Systems (IBM and IRM), obtained with the iSTFT/frame and SWF methods (see Figure 3.9). Clearly, the Oracle Systems perform better than our method, especially for the SDR and the SAR. However, the SIR of our method is approaching the SIR of IBM.

Discussion: Qualitatively, the separated components match the mix signal and the ground truth components when listening to them. However, there are still many artifacts in the estimated sources. In addition, the violin component can sometimes be heard in the piano component estimate and vice versa. In average, the estimated components are louder than the true ones. The one minute long estimated components are obtained by stacking together the 2.04 seconds estimated components. We could have developed other methods such as using a sliding window of 2.04 seconds and averaging each feature.

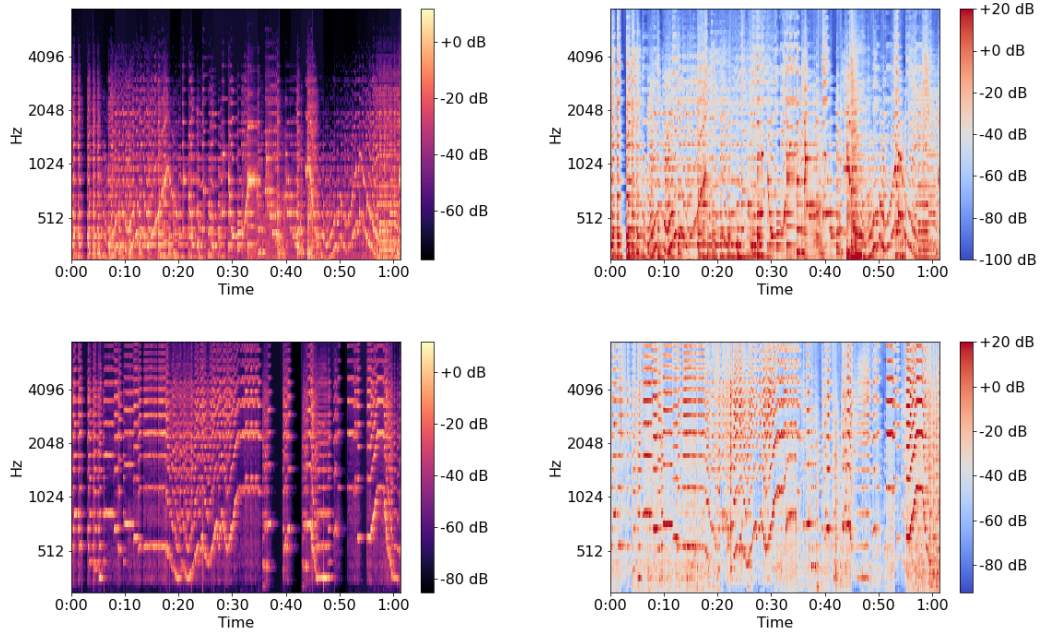


Figure 3.8.: True Spectrograms of each component on the left and Estimated ones on the right. The piano spectrograms are on the top and the violin on the bottom.

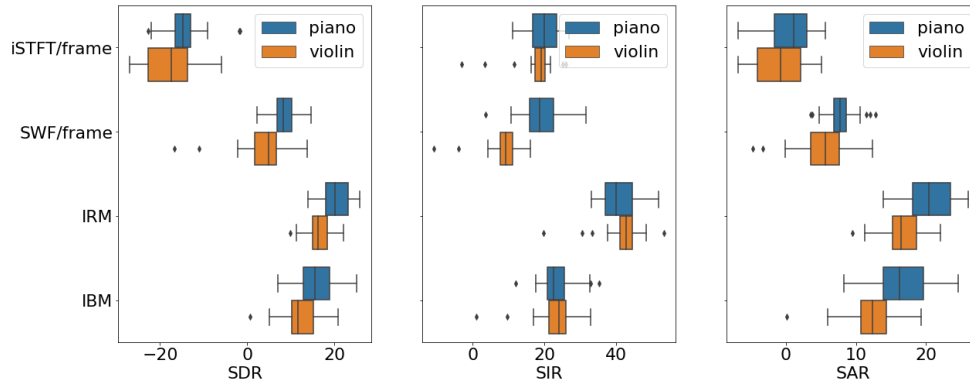


Figure 3.9.: Comparison with Oracle Systems

4. Conclusion

4.1. Synthesis of our Findings

In this research project, we explore the audio source separation problem in a Bayesian Setting, using state-of-the art Deep Generative Models coupled with Langevin Dynamics.

We have shown that the NCSN model was able to learn how to generate spectrograms and that we could effectively separate a mixture of audio in the frequency domain using Langevin Dynamics. Our method has the advantage to generalize well to different mixture of audio since each prior for each source can be trained separately.

4.2. Further Work

For future work, we could explore if this method is capable of separating high quality spectrograms. We would need to train the priors on higher quality spectrograms. To further enhance the performance of the separation, one could use Auto-Regressive models for the priors since this kind of model are known to generate high quality samples. One potential model is MelNet [39] who has proved to generate high quality melspectrograms.

We can also improve the Langevin Dynamics mixing by following the techniques of Song and Ermon [30].

Another way of improvement is to develop an effective method to cut the mixture into shorter extracts, estimate the components from each extract and then stick together the estimated components.

Finally, we could try to use the method directly in the waveform domain with state-of-the art generative models such as WaveNet [26].

Bibliography

- [1] Martín Abadi et al. ‘Tensorflow: A system for large-scale machine learning’. In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.
- [2] A Taylan Cemgil, Cédric Févotte and Simon J Godsill. ‘Variational and stochastic inference for Bayesian source separation’. In: *Digital Signal Processing* 17.5 (2007), pp. 891–913.
- [3] Mike E Davies and Christopher J James. ‘Source separation using single channel ICA’. In: *Signal Processing* 87.8 (2007), pp. 1819–1832.
- [4] R. Decorsière et al. ‘Inversion of Auditory Spectrograms, Traditional Spectrograms, and Other Envelope Representations’. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.1 (2015), pp. 46–56.
- [5] Alexandre Défossez et al. ‘Music source separation in the waveform domain’. In: *arXiv preprint arXiv:1911.13254* (2019).
- [6] Li Deng. ‘The mnist database of handwritten digit images for machine learning research [best of the web]’. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.
- [7] Laurent Dinh, David Krueger and Yoshua Bengio. ‘Nice: Non-linear independent components estimation’. In: *arXiv preprint arXiv:1410.8516* (2014).
- [8] Laurent Dinh, Jascha Sohl-Dickstein and Samy Bengio. ‘Density estimation using real nvp’. In: *arXiv preprint arXiv:1605.08803* (2016).
- [9] Vincent Dumoulin and Francesco Visin. ‘A guide to convolution arithmetic for deep learning’. In: *ArXiv e-prints* (Mar. 2016). eprint: 1603.07285.
- [10] Ian Goodfellow. ‘NIPS 2016 tutorial: Generative adversarial networks’. In: *arXiv preprint arXiv:1701.00160* (2016).
- [11] Daniel Griffin and Jae Lim. ‘Signal estimation from modified short-time Fourier transform’. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.2 (1984), pp. 236–243.
- [12] Jonathan Ho, Ajay Jain and Pieter Abbeel. ‘Denoising Diffusion Probabilistic Models’. In: *arXiv preprint arXiv:2006.11239* (2020).
- [13] Aapo Hyvärinen and Erkki Oja. ‘Independent component analysis: algorithms and applications’. In: *Neural networks* 13.4-5 (2000), pp. 411–430.
- [14] Eric Jacobsen and Richard Lyons. ‘The sliding DFT’. In: *IEEE Signal Processing Magazine* 20.2 (2003), pp. 74–80.

- [15] Vivek Jayaram and John Thickstun. ‘Source Separation with Deep Generative Priors’. In: *arXiv preprint arXiv:2002.07942* (2020).
- [16] Diederik P Kingma and Max Welling. ‘Auto-encoding variational bayes’. In: *arXiv preprint arXiv:1312.6114* (2013).
- [17] Durk P Kingma and Prafulla Dhariwal. ‘Glow: Generative flow with invertible 1x1 convolutions’. In: *Advances in neural information processing systems*. 2018, pp. 10215–10224.
- [18] Qiuqiang Kong et al. ‘Single-channel signal separation and deconvolution with generative adversarial networks’. In: *arXiv preprint arXiv:1906.07552* (2019).
- [19] Alex Krizhevsky, Geoffrey Hinton et al. ‘Learning multiple layers of features from tiny images’. In: (2009).
- [20] Guosheng Lin et al. ‘Refinenet: Multi-path refinement networks for high-resolution semantic segmentation’. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1925–1934.
- [21] Yi Luo and Nima Mesgarani. ‘Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation’. In: *IEEE/ACM transactions on audio, speech, and language processing* 27.8 (2019), pp. 1256–1266.
- [22] Brian McFee et al. ‘librosa: Audio and music signal analysis in python’. In: *Proceedings of the 14th python in science conference*. Vol. 8. 2015, pp. 18–25.
- [23] Radford M Neal. ‘Annealed importance sampling’. In: *Statistics and computing* 11.2 (2001), pp. 125–139.
- [24] Radford M Neal et al. ‘MCMC using Hamiltonian dynamics’. In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.
- [25] Aaron van den Oord, Nal Kalchbrenner and Koray Kavukcuoglu. ‘Pixel recurrent neural networks’. In: *arXiv preprint arXiv:1601.06759* (2016).
- [26] Aaron van den Oord et al. ‘Wavenet: A generative model for raw audio’. In: *arXiv preprint arXiv:1609.03499* (2016).
- [27] Zafar Rafii et al. ‘An overview of lead and accompaniment separation in music’. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.8 (2018), pp. 1307–1335.
- [28] Paris Smaragdis et al. ‘Static and dynamic source separation using nonnegative factorizations: A unified view’. In: *IEEE Signal Processing Magazine* 31.3 (2014), pp. 66–75.
- [29] Yang Song and Stefano Ermon. ‘Generative modeling by estimating gradients of the data distribution’. In: *Advances in Neural Information Processing Systems*. 2019, pp. 11918–11930.
- [30] Yang Song and Stefano Ermon. ‘Improved Techniques for Training Score-Based Generative Models’. In: *arXiv preprint arXiv:2006.09011* (2020).

- [31] Henning Sprekeler, Tiziano Zito and Laurenz Wiskott. ‘An extension of slow feature analysis for nonlinear blind source separation’. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 921–947.
- [32] Stanley Smith Stevens, John Volkman and Edwin B Newman. ‘A scale for the measurement of the psychological magnitude pitch’. In: *The Journal of the Acoustical Society of America* 8.3 (1937), pp. 185–190.
- [33] Daniel Stoller, Sebastian Ewert and Simon Dixon. ‘Wave-u-net: A multi-scale neural network for end-to-end audio source separation’. In: *arXiv preprint arXiv:1806.03185* (2018).
- [34] Fabian-Robert Stöter, Antoine Liutkus and Nobutaka Ito. ‘The 2018 signal separation evaluation campaign’. In: *International Conference on Latent Variable Analysis and Signal Separation*. Springer. 2018, pp. 293–305.
- [35] Fabian-Robert Stöter et al. ‘Open-unmix-a reference implementation for music source separation’. In: (2019).
- [36] Y Cem Subakan and Paris Smaragdis. ‘Generative adversarial source separation’. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 26–30.
- [37] Naoya Takahashi, Nabarun Goswami and Yuki Mitsufuji. *MMDenseLSTM: An efficient combination of convolutional and recurrent neural networks for audio source separation*. 2018. arXiv: 1805.02410 [cs.SD].
- [38] Stefan Uhlich et al. ‘Improving music source separation based on deep neural networks through data augmentation and network blending’. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 261–265.
- [39] Sean Vasequez and Mike Lewis. ‘Melnet: A generative model for audio in the frequency domain’. In: *arXiv preprint arXiv:1906.01083* (2019).
- [40] Emmanuel Vincent, Rémi Gribonval and Cédric Févotte. ‘Performance measurement in blind audio source separation’. In: *IEEE transactions on audio, speech, and language processing* 14.4 (2006), pp. 1462–1469.
- [41] Emmanuel Vincent, Tuomas Virtanen and Sharon Gannot. *Audio source separation and speech enhancement*. John Wiley & Sons, 2018.
- [42] Pascal Vincent. ‘A connection between score matching and denoising autoencoders’. In: *Neural computation* 23.7 (2011), pp. 1661–1674.
- [43] T. Virtanen. ‘Monaural Sound Source Separation by Nonnegative Matrix Factorization With Temporal Continuity and Sparseness Criteria’. In: *IEEE Transactions on Audio, Speech, and Language Processing* 15.3 (2007), pp. 1066–1074.
- [44] Max Welling and Yee W Teh. ‘Bayesian learning via stochastic gradient Langevin dynamics’. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 681–688.

A. NCSN Architecture

A.1. Multi-path Refinement

The architecture of the network is the RefineNet architecture [20]. The figures below are taken and adapted from the RefineNet article to match our network. This network makes use of residual connections to propagate low-level features and hence refine coarse high level features. The RefineNet is composed of a sequence of 4 pairs of Residual Blocks and a cascade of 4 RefineNet blocks. Each pair of Residual blocks is connected to the previous one. Each RefineNet block is connected to the previous one as well as the output of the Residual blocks at the same level. A first Convolutional Layer is used as pre-activation. A last Instance Normalization Layer and Convolutional Layer are used at the output of the RefineNet Cascade. Figure A.1 illustrates this multi-path refinement architecture.

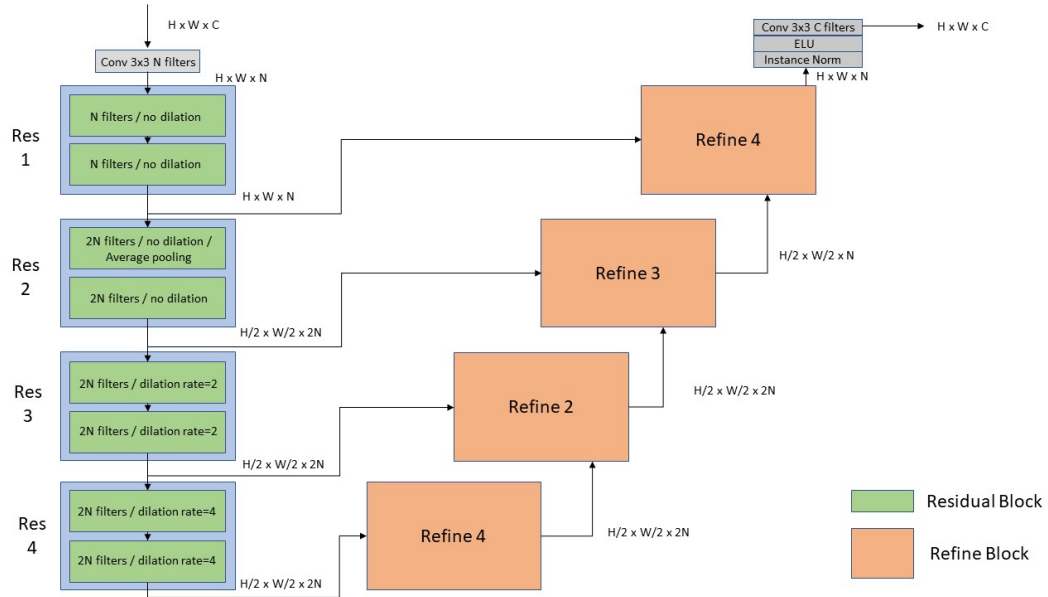


Figure A.1.: Multi-Path Refinement Architecture

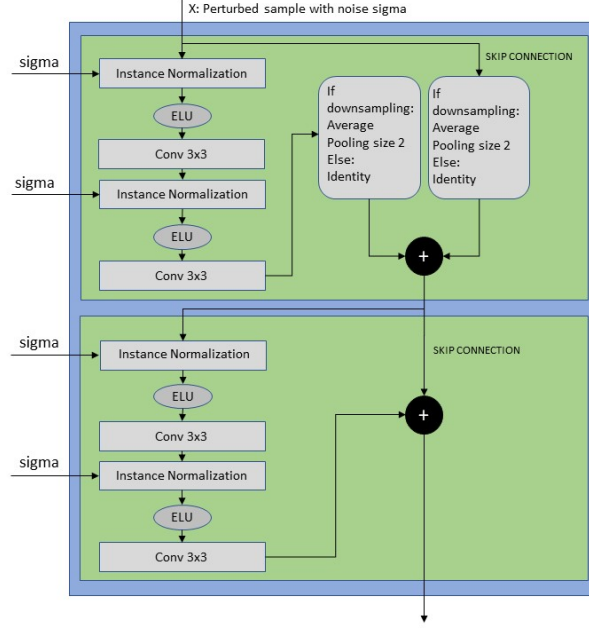


Figure A.2.: A pair of Residual Blocks

A.2. Conditional Residual Block

Each Residual Block is composed of 2 sequences of Instance Normalization - Convolutional Layer with an ELU activation after each normalization: $\text{ELU}(x) = x$ if $x > 0$ and $\exp(x) - 1$ otherwise. The output is added to the input via a skip connection. The first pair of residual blocks uses regular Convolutional Layers with n filters while the other pairs use $2n$ filters. The first Residual Block of the 2nd pair downsamples its input with Average pooling to reduce the spatial dimensions by a factor of 2. The 3rd and 4th pairs use dilated Convolution of size 2 and 4 respectively. Figure A.2 shows a pair of Residual Block.

A.3. Refine Block

The Refine Block is composed of a sequence of 3 kind of layers: the Residual Convolution Unit (RCU), the Multi-Resolution Fusion layer (MRF) and the Chained Residual Pooling layer (CRP) as shown in Figure A.3. If the two inputs of the Refine Block do not have the same size, the smallest one (from the previous Refine Block) is upsampled in the MRF layer using Bilinear Interpolation. In the CRP layer, the "Mean Pool 5x5" blocks are Average Pooling Layer with pooling size 5 and stride 1: the average of each 5×5 square in the spatial dimensions is taken to build the output (the input is padded with zeros so the dimension is kept the same). The last Refine Block has 3 RCU after the CRP layer while the others have only one. The first Refine Block does not have a MRF

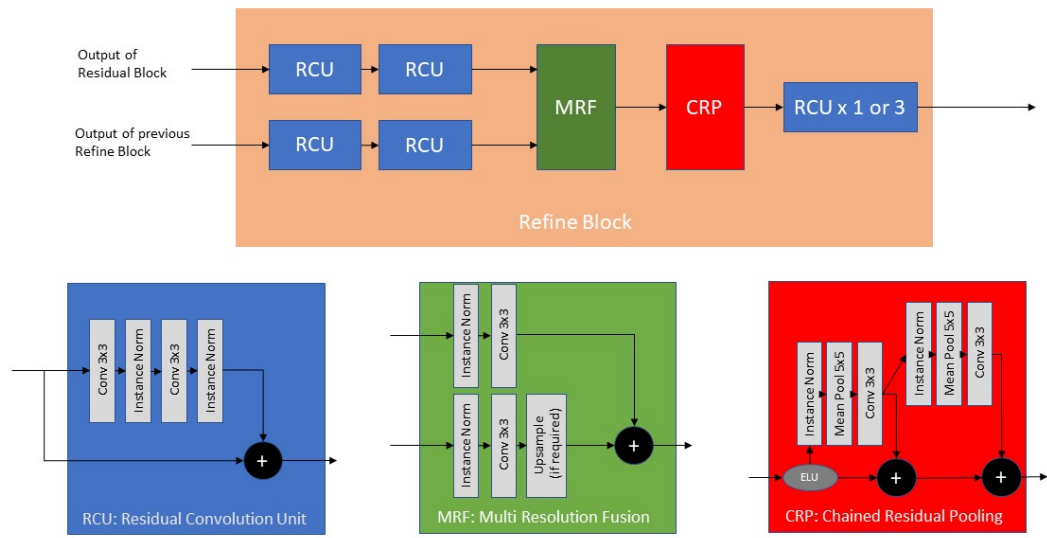


Figure A.3.: Refine Block

layer since it has only one input.

B. Generated Spectrograms

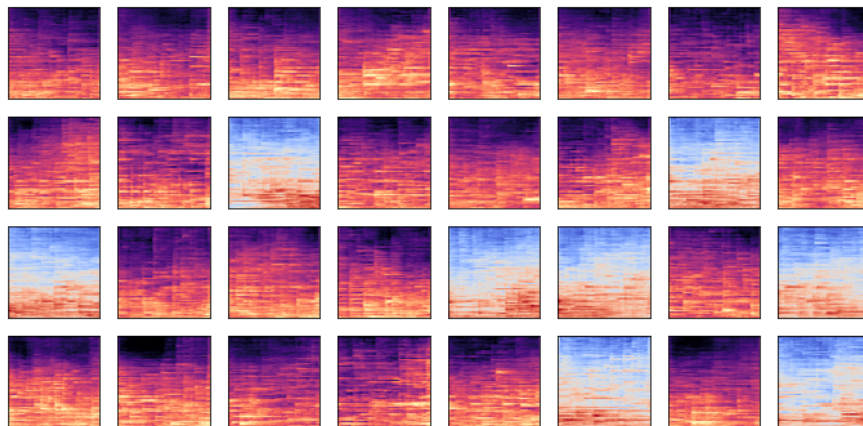


Figure B.1.: Generated Piano Spectrograms using the Glow model: $L=3$, $K=40$, filters=512

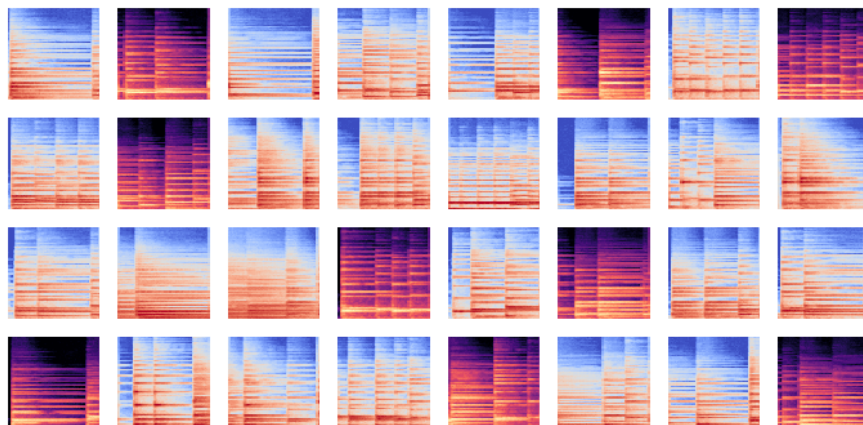


Figure B.2.: Generated Piano Spectrograms using the NCSN model: filters=192

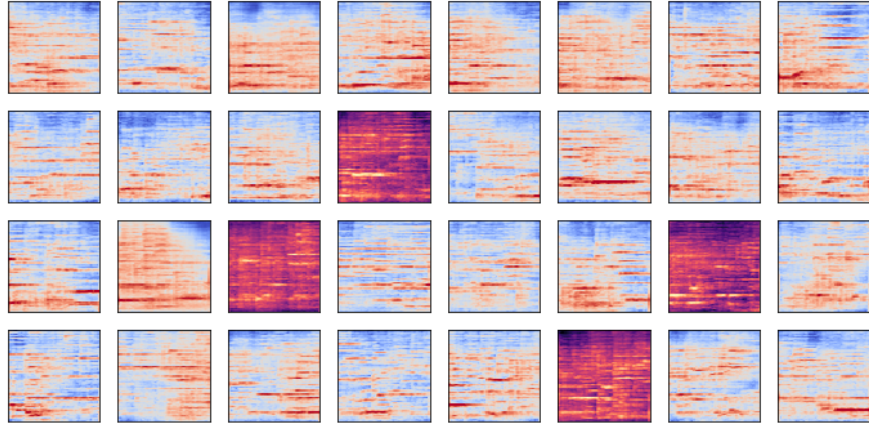


Figure B.3.: Generated Violin Spectrograms using the Glow model: $L=3$, $K=40$, filters=512

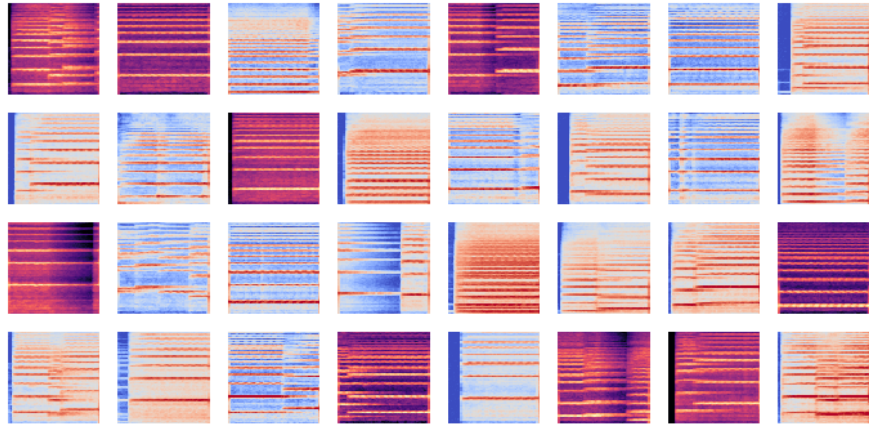


Figure B.4.: Generated Violin Spectrograms using the NCSN model: filters=192

C. BASIS convergence examples

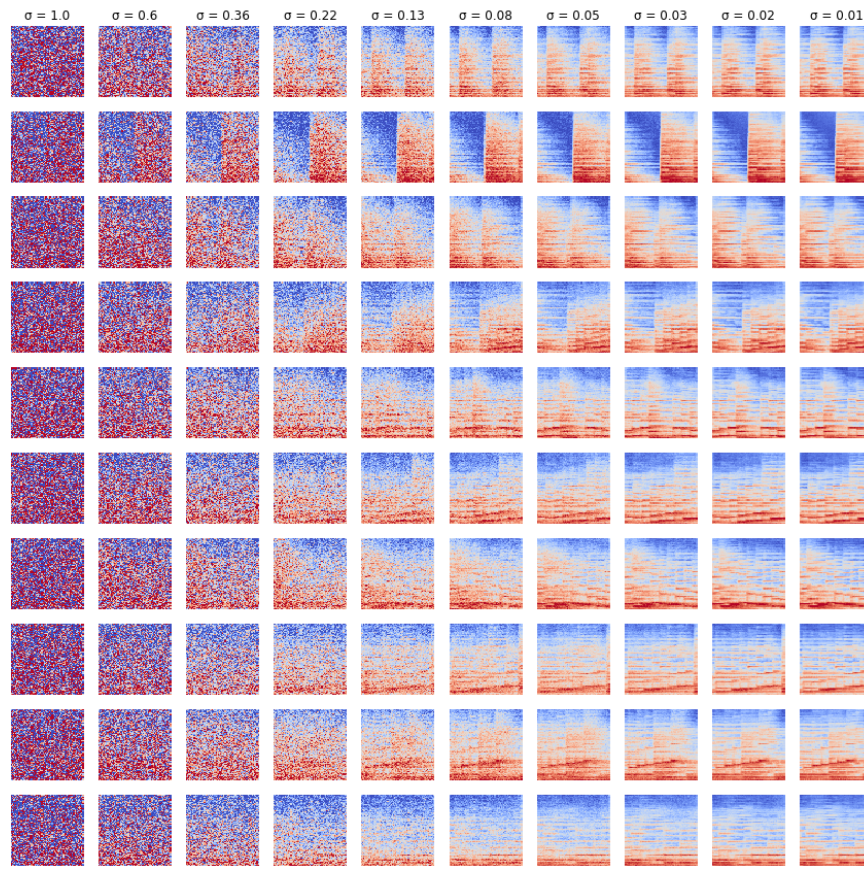


Figure C.1.: Convergence of the BASIS algorithm for the piano component

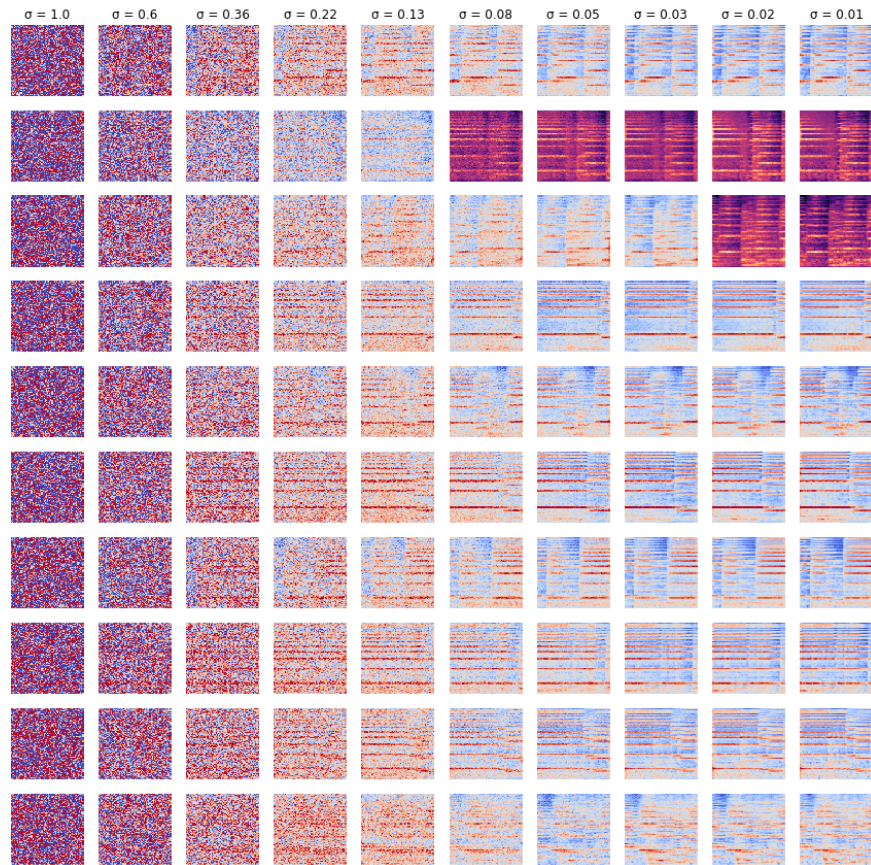


Figure C.2.: Convergence of the BASIS algorithm for the violin component