MASTER ATIAM

MASTER THESIS

Supervised Singing Voice Separation: Designing a data pipeline for supervised learning

Author:
Laure PRÉTET

Supervisors:
Romain HENNEQUIN
Jimena ROYO-LETELIER

DEEZER R&D

February 6th - August 3rd 2018

Master ATIAM

Abstract

Supervised Singing Voice Separation: Designing a data pipeline for supervised learning

by Laure PRÉTET

In this research internship, our topic is to design and evaluate algorithms that are able to separate the vocal part from the accompaniment in songs. Given any song recording, we try to estimate how the singing voice and the accompaniment would sound when isolated from each other. The emphasis is set on supervised, neural network-based systems that address this task. Two up-to-date singing voice separation systems from the literature are implemented. We explain the issue of finding good training datasets for supervised learning and we adapt an existing method to design one from Deezer's catalog. Then, we study the impact of the training databases on separation performances for each system. Additionally, we studied other data-related settings in training and reconstruction, such as data augmentation, post-processing and the training loss. Good performances were achieved using a U-Net that was able able to model large time and frequency contexts. Other results show that the training loss and post-processing must be carefully chosen depending on the final application, while data augmentation has little effect on performances.

Notre sujet d'étude, dans ce stage de recherche, est de produire et d'évaluer des algorithmes capables de séparer la partie vocale d'une chanson de son accompagnement. Étant donné l'enregistrement d'une chanson, nous tentons de produire des estimés des parties vocales et instrumentales telles qu'elles sonneraient si elles étaient isolées l'une de l'autre. Nous nous intéressons en particulier aux méthodes supervisées qui s'appuient sur des réseaux de neurones pour aborder cette tâche. Nous implémentons deux systèmes de l'état de l'art pour la séparation de voix chantée. Nous expliquons le défi qui consiste à trouver de bonnes bases de données d'entraînement pour l'apprentissage supervisé et nous adaptons une méthode existante pour en créer une à partir du catalogue de Deezer. Puis, pour chaque système, nous étudions l'impact de la base de données d'entraînement sur les performances de séparation. Nous nous intéressons également à d'autres modalités pour l'entraînement et la reconstruction, à savoir l'augmentation de données, le post-traitement et la fonction de coût pour l'entraînement. À l'aide d'un U-Net, capable de modéliser de longs contextes temporels et fréquentiels, nous obtenons de bonnes performances de séparation. Nos autres expériences montrent qu'il est crucial de bien choisir la fonction de coût et le post-traitement en fonction de l'application finale de la séparation, tandis que l'augmentation de données a seulement un impact limité sur les performances.

Keywords: Music Information Retrieval, Singing voice separation, Deep learning, Training datasets, Data augmentation.

Contents

1 Introduction to Singing Voice Separation **1**
1.1 Internship context . . . 1
1.2 Practical applications . . . 2
1.3 The singing voice separation task . . . 2
1.4 Performances evaluation . . . 5
1.5 Unsupervised systems . . . 6

2 Deep neural networks for singing voice separation **9**
2.1 Supervised learning and regression problems . . . 9
2.2 Fully-connected neural networks . . . 10
2.3 Convolutional neural networks . . . 10
2.4 How to train a neural network . . . 12
2.5 Deep neural networks applied to singing voice separation . . . 13

3 Multi-tracks datasets for supervised learning and evaluation **17**
3.1 Standard databases . . . 17
3.2 Building a singing voice separation dataset . . . 17
3.3 The dataset problem in supervised machine learning . . . 19
3.4 Evaluation campaigns . . . 20

4 Experiments and results: the impact of various data-related parameters on performances **23**
4.1 Baseline systems . . . 23
4.2 Experimental framework and methodology . . . 24
4.3 The selected network architectures . . . 25
4.4 Experiments . . . 28
4.4.1 Experiment 1: different training datasets . . . 29
4.4.2 Experiment 2: different losses . . . 31
4.4.3 Experiment 3: different post-processing methods . . . 34
4.4.4 Experiment 4: different data augmentation methods . . . 37

5 Discussions and future work **41**
5.1 Discussion . . . 41
5.2 Future work . . . 41

A Code snippets **43**
A.1 An example of a score file produced by the Museval package for a short song. 43

B Details of network architectures **45**
B.1 The transposed convolution used in the upsampling path of the networks. . . 45

Bibliography **47**

Chapter 1

Introduction to Singing Voice Separation

In this chapter, after setting the context of the internship, we explain the general framework needed for singing voice separation. We also present applications of this task and examples of unsupervised systems that can be used to achieve it.

1.1 Internship context

This internship was hosted by the Research and Development team at Deezer headquarter, Paris. Deezer is an international music streaming company which serves millions of users in 180 countries. Its catalog features about 55 million audio tracks, including music from all genres, quality and geographic origin, as well as spoken audiobooks and podcasts.

One of the missions of the Research and Development team is to design, experiment and prototype systems to better characterize and organize the content of the catalog. Indeed, the genre, performing artist and other attributes of the tracks are often ambiguous, corrupted or undefined. Fixing these issues is crucial to propose better music recommendations and better services to the user in the long term. These projects can rely on both metadata (when available) and audio data (always available), as well as external data in some cases (public domain datasets, external ontologies). In particular, to quickly and extensively characterize musical content, the team uses audio-based Music Information Retrieval methods which proved efficiency, as exposed in Royo-Letelier, 2015.

The Research and Development team is composed of Manuel Moussalam (head of the team), Jimena Royo-Letelier, Viet-Anh Tran and Romain Hennequin (research scientists), Mickaël Arcos and Anis Khlif (software engineers), Andrea Vaglio (PhD student) and four interns: Yousri Sellami, Agnès Mustar, Paul Vernhet and myself.

My mission during six months was to study singing voice separation methods, in order to apply them later to a large number of songs present in the catalog. During the last two months, I worked in tandem with Andrea Vaglio, who started his PhD on automatic lyrics transcription.

Our main contributions are the following: we re-implemented two systems presented in recent literature that perform well on singing voice separation (according to the last evaluation campaign: Stöter, Liutkus, and Ito, 2018). We then trained and evaluated them using different datasets. This has two main interests: study first the impact of the network architecture and second, the impact of the training dataset, in terms or performances. In the literature, it is indeed difficult to evaluate the different elements of the systems, since the papers tend to present a full procedure, including dataset building, data pre-processing, architecture design, post-processing and sometimes data augmentation. In addition, we studied the impact of different post-processing methods, different training losses and data augmentation techniques.

This report is organized in five parts. In Chapter 1, we present the topic of singing voice separation. In Chapter 2, we give a short summary of the deep network architectures, and present some of them that were used for singing voice separation. In Chapter 3, we explain the issue of finding good training datasets for supervised learning, present public datasets and expose a method to design one from Deezer's catalog. In Chapter 4, we present our experiments with different system architectures and training datasets. Finally, Chapter 5 is dedicated to the analysis of the results and the research perspectives raised by this work.

1.2 Practical applications

Singing voice separation can be used as a preprocessing step for a variety of tasks, such as remixing, pitch tracking (Pollastri, 2002) or singer identification (Mesaros, Virtanen, and Klapuri, 2007). Furthermore, it is a widely used pre-processing step for lyrics transcription (Mesaros and Virtanen, 2010).

As a first application case, the karaoke industry can take great advantage of singing voice separation algorithms: they allow one to create instrumental tracks automatically from any song recording. The karaoke industry has been estimated to be worth several billions of dollars globally (source: https://en.wikipedia.org/wiki/Karaoke).

For the streaming industry, a transcription of the lyrics of any song, even an approximate one, can be very useful to characterize large catalogs. One could more easily perform tasks such as cover detection, language detection (Mesaros, 2012), and explicit content detection (Kruspe and Fraunhofer, 2014). Since Deezer's Research and Development team just hired Andrea Vaglio as a PhD student to work on lyrics transcription, one can consider my internship as a preparatory study on this topic.

1.3 The singing voice separation task

Singing voice separation is a sub-domain of music source separation, a Music Information Retrieval (MIR) task. The objective of MIR is to develop automatic systems that are able to solve various tasks in order to characterize music in terms of genre, timbre, rhythm and other attributes (Orio, 2006). Typical MIR tasks are beat tracking, pitch tracking, or genre retrieval. The systems usually start directly from the audio files and rely on signal processing and machine learning techniques.

Music source separation has been a popular topic within the MIR community for several decades. In this task, the objective is to decompose a music recording into several tracks, each of one corresponding to a single instrument or a group of instruments. While this seems intuitive for the human auditory system, it is an underdetermined problem that is still unsolved by computational means: several solutions exist for the same recording.

Singing voice separation is a particular case of music source separation. Its specificity is to consider only two sources to be separated: the singing voice on one side, and the instrumental accompaniment on the other. The first source, the **vocal** part, gathers the leading voice (melody), the choirs, distorted voice and voice reverberation effects, while all music instruments are grouped together into one single source: the **instrumental** part (see Figure 1.1). The original recording that contains all the sources is called the **mix**, or **mixture**.

This problem is tailored for western popular music, where there is often a lead vocalist and an instrumental accompaniment. It is one of the most studied cases of music source separation, and it has a wide range of applications. New papers are released each month,

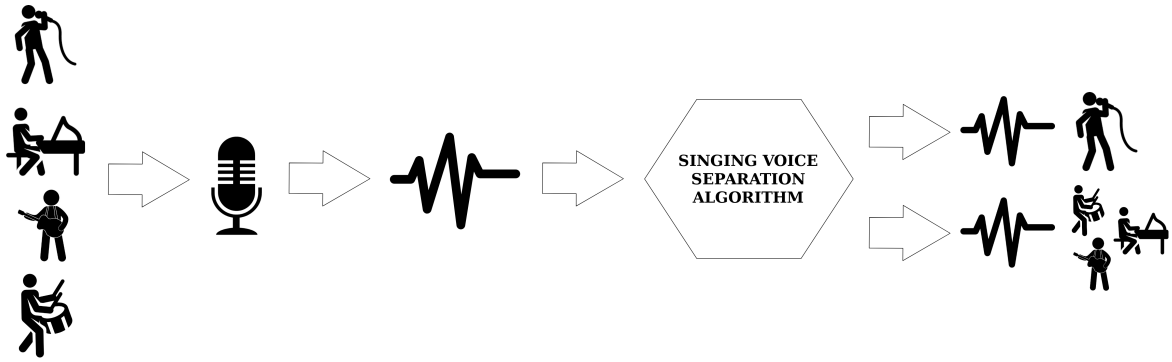


FIGURE 1.1: An illustration of the singing voice separation framework.

providing a large amount of existing work to support research activities. Yearly competitions such as the SiSec MUS challenge[1] gather an increasing number of teams (24 systems evaluated in 2016, 30 in 2018).

In the following paragraphs, we will describe the data pipeline required to perform the separation. We will consider the separation system itself as a black box that will be detailed later, and focus on all the processing steps that happen before and after. This pipeline is illustrated in Figure 1.2.

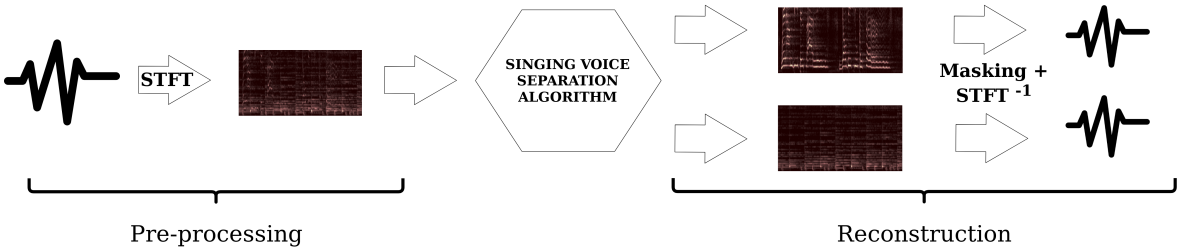


FIGURE 1.2: An illustration of the singing voice separation framework.

Data pre-processing: STFT

To perform singing voice separation, the majority of systems rely only on the audio signals, without additional information or metadata. First, the raw audio must be transformed into a more adapted representation: a time-frequency representation. Several transforms are possible, including the Short-Time Fourier Transform (STFT), the Mel Spectrogram and the Constant-Q transform (CQT). In practice, the separation frameworks that we will study operate in the STFT domain. They take as input the spectrogram of the mixture and try to estimate the vocal and instrumental spectrograms, as illustrated on 1.3. The main advantage of the STFT over other transforms is to be easily revertible to an audio format. Moreover, in Dieleman and Schrauwen, 2014, the authors found that using the waveform as input of a tag classifier usually leads to learning a sinusoid-based representation in the first layer. Consequently, it is pragmatic to use directly a Fourier-base transform as input of the separation systems.

As a convention, in the following of this report, we will note S the complex spectrograms, obtained by STFT, and V their magnitude.

[1] https://sisec.inria.fr/

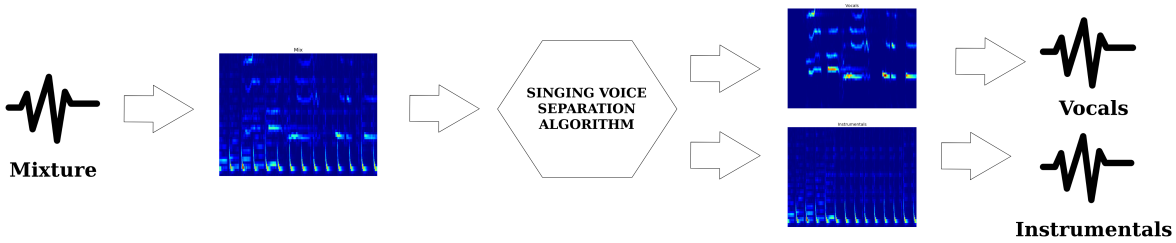


FIGURE 1.3: An illustration of a singing voice separation framework that operates in the time-frequency domain.

Data reconstruction

For the algorithms that operate in the STFT domain, a specific estimation procedure has been developed, which relies on the concept of **spectrogram masks**. In this process, only the magnitude of the mixture spectrogram is used as the input of the system. Removing the phase, which varies faster in time, has a smoothing and denoising effect. From the mixture's magnitude spectrogram V_{mix}, the separation algorithm estimates the magnitude spectrogram of both target sources: $\hat{V}_{vocals}$ and $\hat{V}_{instrumentals}$. These magnitude spectrograms are used to compute two ratio masks: M_{vocals} and $M_{instrumentals}$. The masks are matrices of weights (all coefficients have values between 0 and 1). When applied to the mixture's original spectrogram S_{mix} by element-wise multiplication, they keep only the bins containing energy from the desired source (the vocals, for example). The other sources (the instrumentals) are filtered out from the remaining spectrogram (see Figure 1.4).

$$M_{vocals} = \frac{\hat{V}_{vocals}}{\hat{V}_{vocals} + \hat{V}_{instrumentals}} \tag{1.1}$$

$$M_{instrumentals} = \frac{\hat{V}_{instrumentals}}{\hat{V}_{vocals} + \hat{V}_{instrumentals}} \tag{1.2}$$

$$\forall i, j \quad M_{instrumentals_{i,j}} + M_{vocals_{i,j}} = 1 \tag{1.3}$$

$$\hat{S}_{vocals} = M_{vocals} * S_{mix}. \tag{1.4}$$

$$\hat{S}_{instrumentals} = M_{instrumentals} * S_{mix}. \tag{1.5}$$

This way, full spectrograms are built for each source, using the estimated magnitude and the original mixture's phase. The obtained spectrograms ($\hat{S}_{vocals}$, $\hat{S}_{instrumentals}$) can eventually be reverted to estimated source audio signals. In the literature, some systems operate on squared magnitude spectrograms to produce estimates of the power spectral density of the sources. In this case, the masking method is referred to as *Wiener filtering*.

Systems using masking tend to be more robust and to produce limited artifacts (Huang et al., 2014a). Masking also enforces the fact that both estimates sum to the original mixture in the time domain (reverse STFT being a linear operation). That's why it is almost systematically used for reconstruction of the audio after separation.

Dominance assumption

When using a masking technique, the underlying assumption is that for each time-frequency bin, one of both sources substantially dominates the other. Therefore, we can distribute the spectrogram's energy among the sources for each bin. In this model, the isolated sources are

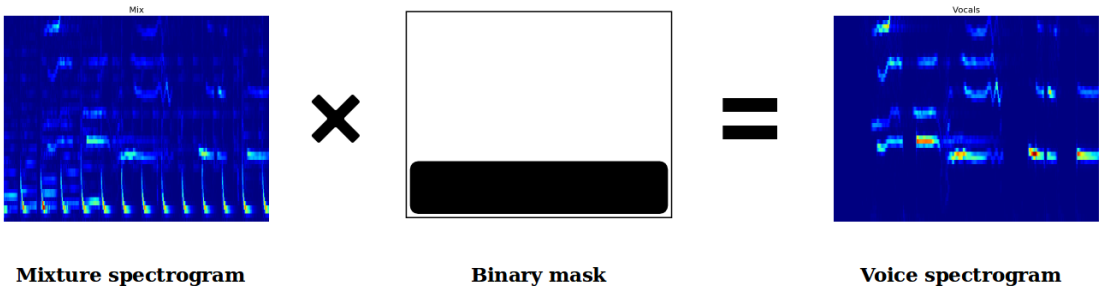


FIGURE 1.4: Illustration of binary masking with clearly separated parts.

assumed to be more or less uncorrelated in time and frequency. This is of course debatable in music. However, most of the separation systems use masking[2] , and some of them try to directly estimate time-frequency masks (Huang et al., 2014b). Masking is also used to produce *oracle* systems (Stöter, Liutkus, and Ito, 2018). As we will show in 4.4.1, oracle masks generally produce very convincing estimates with very few artifacts. For simplification purposes, we will use the dominance hypothesis as well.

1.4 Performances evaluation

To evaluate the performance of a singing voice separation algorithm, the usual procedure is the following. From a standard dataset of songs, the systems must estimate the contributions of both target sources in the form of two audio files, using only the mix. Then, the performance is estimated using either automatic tools or human perception.

Objective metrics

In this type of evaluation, a dedicated toolbox uses the estimated tracks and the reference ground truth tracks to compute separation scores. For the most common evaluation toolbox, BSS_EVAL, these scores take the form of three measures, in decibels, that must be as high as possible. Following the original paper of Févotte, Gribonval, and Vincent, 2005, the estimated signal is first decomposed as the sum of its target signal and a three perturbation signals:

$$S_{estim}(t) = s_{target}(t) + e_{interf}(t) + e_{noise}(t) + e_{artif}(t), \tag{1.6}$$

where e_{interf}, e_{noise} and $e_{artif}(t)$ correspond respectively to the interferences, noise and artifacts error terms. Then the metrics are computed as follows:

- The **Signal to Distortion ratio** (SDR) measures the general quality of the separation;

$$SDR = 10\log_{10}\frac{||s_{target}||^2}{||e_{interf} + e_{noise} + e_{artif}||^2} \tag{1.7}$$

- The **Signal to Interference ratio** (SIR) measures how much of the other sources remain in the separated signal;

[2] As examples of alternative approaches, we can cite Nugraha, Liutkus, and Vincent, 2016 and Stoller, Ewert, and Dixon, 2018. The first one exploits multi-channel information to directly output complex spectrograms, while the second one exclusively works in the time domain.

$$SIR = 10\log_{10}\frac{||s_{target}||^2}{||e_{interf}||^2} \tag{1.8}$$

- The **Signal to Artifacts ratio** (SAR) measures the artifacts induced by the separation algorithm that are not residuals of other sources.

$$SAR = 10\log_{10}\frac{||s_{target} + e_{interf} + e_{noise}||^2}{||e_{artif}||^2} \tag{1.9}$$

The main aspect to note on these metrics is that they are computed based on the energy of the signal, and no difference is made between low and high frequencies. This means that in low-energy frequencies, namely high frequencies in music, more important relative error can be tolerated. We should also notice that the metrics are not defined if the target signal is silent. This happens regularly for voice estimation, so some parts of the signal must be ignored. We will keep these specificities in mind to interpret our results.

The relevance of each metric depends on the application. Empirically, it was found that for remixing, the interferences are more problematic than the artifacts, so separation systems designed for this goal will try to optimize the SIR prior to the SAR. In a similar way, we expect to require such compromises for lyrics transcription.

While the original toolbox was released in Matlab, different re-implementations exist for Python. We will mainly use Museval (Rafii et al., 2017). The main advantage of these evaluation algorithms is that they are deterministic and thus allow to objectively and fairly compare the separation system with each other. The main limitation of objective metrics are that despite their logarithmic scale, they often not correlate to human perception (Cano, FitzGerald, and Brandenburg, 2016).

Perceptive measures

In addition to the objective metrics, one can judge the quality of the separation by asking directly for the opinion of human listeners. Specific procedures exist for this perceptive evaluation (see Cano, FitzGerald, and Brandenburg, 2016 and Jansson et al., 2017). Such measures would in theory be the best possible, but they remain out of reach for most research teams, because of financial and time constraints. This is why, in the context of this internship, we restricted ourselves to informal listening tests, as a complement to the objective measures.

1.5 Unsupervised systems

Over time, researchers have developed a wide variety of methods to separate vocals from instrumentals in a mixture. REPET, an algorithm released by Rafii and Pardo, 2013, proposed to model the typical repeating structure of the instrumental accompaniment in pop songs. Using auto-correlation, the repeating background (guitar riffs, drum loops) can be extracted from the foreground (vocal line). This technique works mostly on short segments, where it gives better results than a simple high-pass filtering, used as baseline. Such a method exploits prior knowledge about the spectrograms and deduces a set of rules to be applied directly to any audio to perform separation. We say that it is **model-based**. Other model-based systems include Li and Wang, 2005 and Ryynanen et al., 2008, in which a first pass of pitch contour extraction with F0 tracking is performed before estimating the signal without the leading melody.

Other approaches rely on matrix decomposition techniques, in order to split the spectrogram into several parts. Statistically driven methods (Raj et al., 2007), bayesian methods

(Ozerov et al., 2007) and independent components analysis (Vembu and Baumann, 2005) were successfully used for singing voice separation. Another popular decomposition is robust principal components analysis (RPCA), in which one assumes that the instrumentals are repetitive and the vocals are sparse. The spectrogram of the mix is thus estimated as the sum of a low-rank and a sparse matrix: $X = A + E$, and the optimization problem to solve is:

$$\min_{A,E} ||A||_* + \lambda||E||_1$$

where $||A||_* = Tr(\sqrt{AA^*})$ is the trace norm of the matrix. Variations of this technique, such as low-rank and group-sparse representations, are benchmarked in Chan and Yang, 2017. But, as shown by the benchmark, these systems only have limited performances, unless they are informed with chords or pitch annotations.

Another way to decompose the audio is to apply matrix factorization to the spectrograms. Historically, matrix factorization has been a privileged mean to solve source separation problems. In particular, for audio, the Non-negative Matrix Factorization (NMF) is an unsupervised method for the factorization of magnitude spectrograms (Weninger et al., 2014). In NMF, an iterative algorithm estimates the mixture spectrogram V as the product of two positive matrix : the first one is called the *dictionary* (W) and contains spectrogram patterns, while the other is the *activation* (H) matrix and specifies when the patterns should be activated (see Figure 1.5). Grouping the patterns into clusters allows to create an estimation of the magnitude spectrogram of each separated source.

Along with the history of singing voice separation, many refinements have been brought to NMF. A first way to inform the NMF decomposition for separation is to use the pitch information. The pitch of the leading voice can be estimated with F0 tracking. The estimated melody then informs the NMF, as in Virtanen, Mesaros, and Ryynänen, 2008. The limitations of such a system is that it only guarantees that the main melody will be separated, and it actually transcribes any harmonic instrument that plays it. Another way to improve NMF and to be more vocals-specific is Durrieu's model, as used in Durrieu et al., 2009. The idea is to integrate a source-filter model for the voice into the NMF algorithm. The *vocal* part is decomposed into a generic source (for pitch) and a filter (for formants): $V_{voice} = (W_F H_F) * (W_S H_S)$. The source matrix W_S is known: it follows the KLGLOTT88 model, which relates to a "cumb" source with peaks at every multiple frequency of the fundamental frequency (Klatt and Klatt, 1990). The accompaniment is modeled as the product of two unknown matrices: $W_{acc} H_{acc}$. The mix spectrogram model that the NMF will try to fit is: $V_{mix} = (W_F H_F).*(W_S H_S) + W_{acc} H_{acc}$.

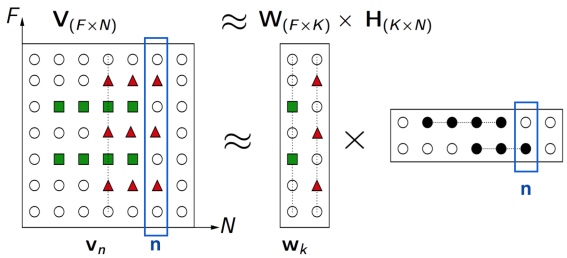


FIGURE 1.5: The NMF algorithm (Image: Cédric Févotte)

The main advantage of NMF-based method is their interpretability, since the elements of the dictionary are spectrograms that are directly interpretable as audio components of the signal (one note or one drum stroke, for example). But they tend to create artifacts in the reconstructed signals, depending on the type of model chosen (Ozerov and Févotte, 2010).

Chapter 2

Deep neural networks for singing voice separation

We now present the general framework to perform separation using supervised, neural-network based methods. We explain the structure of standard feed-forward networks and how to train them. Then, we focus on convolutional neural networks. Finally, we present state-of-the-art methods that use convolutional neural networks for singing voice separation.

2.1 Supervised learning and regression problems

Singing voice separation can be thought as a **regression problem**. This means that from a given **input** (the mixture's spectrogram), we will produce an estimate of a **target** (the source's spectrogram), and this estimate belongs to a continuous output space. The target y is sometimes called the **ground truth**. Let x be our input, y our target and $\hat{y}$ our target's estimate. The problem consists in finding a transformation f_θ:

$$\hat{y} = f_\theta(x). \tag{2.1}$$

The function f_θ is called the **model**. It is parametric such that, by tuning θ, we can minimize a certain distance d between the estimate and the target:

$$\min_\theta d(y, \hat{y}). \tag{2.2}$$

The distance d is called the **loss function**. For our problem, it is often the L_1 norm of the difference between y and $\hat{y}$, or their mean squared error (MSE). The form of the function f_θ depends on the type of model that we use (neural network, support vector machine...), while the parameters θ are adjusted during a process called the **training**. During the training, the parameters of the model are iteratively refined until a satisfying performance on the regression task is reached. The training is performed using a dataset of examples: the **training dataset**. In our case, the training is said **supervised**, as the training dataset contains the ground truth for each example (the separated instrumental and vocal tracks). Hopefully, at the end of the training process, the model fits the training data and is able to produce good estimates for new data: this is the **inference** step. In addition to θ, the systems often have other parameters that cannot be tuned during the training. They typically affect the architecture of the model and the training algorithm itself. They are called **hyper-parameters**. Hyper-parameters are generally guessed and tuned manually, sometimes using a grid search algorithm.

2.2 Fully-connected neural networks

Neural networks are a family of models that has become increasingly popular during the last decades (Isola et al., 2017). A neural network is composed of **units**, that are organized in **layers**. The units are connected to each other and each connection is affected a **weight**. The weights are the parameters θ that are tuned during the training.

The most elementary neural network is composed of one single unit: the **perceptron**. It has n numerical inputs $(x_1, ..., x_n)$, $n+1$ weights $(b, w_1, ..., w_n)$ and one output $\hat{y}$. The output of a unit is called the **activation**.

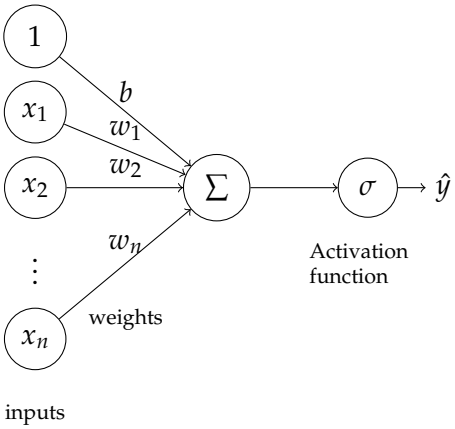


FIGURE 2.1: A perceptron with n inputs and one output.

In the perceptron model (Figure 2.1), the inputs first go through a weighted sum to which a **bias** b is added. Then the results goes though an **activation function**. The final output is:

$$\hat{y} = \sigma(w^T x + b). \tag{2.3}$$

In singing voice separation, we will mainly use the ReLu activation function (Nair and Hinton, 2010):

$$\sigma(x) = \max(x, 0). \tag{2.4}$$

In more complex models, the units can be grouped into layers, which are stacked into networks by following a given architecture. By extension, we also call *activations* the tensor composed of all the outputs of a layer. We call the **depth** of the network its number of layers, excluding the input layer. We say that a layer is **fully-connected** if each unit of each layer is connected to all the units of the next layer (Figure 2.2).

2.3 Convolutional neural networks

Convolutional networks are a special case of neural networks which showed excellent performances in image recognition and in MIR during the recent years (Russakovsky et al., 2015; Stöter, Liutkus, and Ito, 2018). This type of architecture rely on two special types of layers: the **convolutional layers**, and the **pooling layers**.

Convolutional layers

Most convolutional networks take as input real 3D tensors with a width M, a height N and a certain number of channels K. For stereo magnitude spectrograms, the initial number of

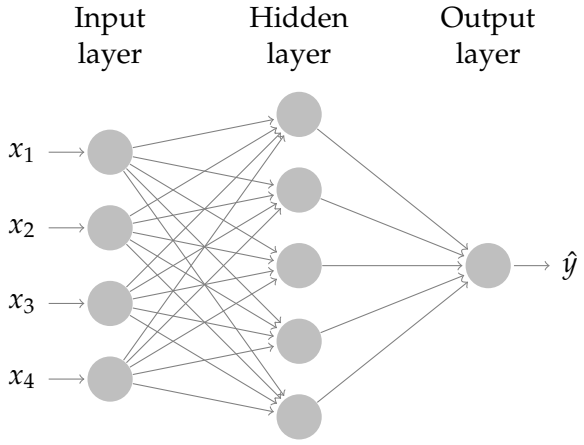


FIGURE 2.2: A fully-connected neural network with one hidden layer.

channels is 2. In a convolutional layer, the input activation tensors are filtered by a set of filters, which are small matrices of weights (see Figure 2.3).

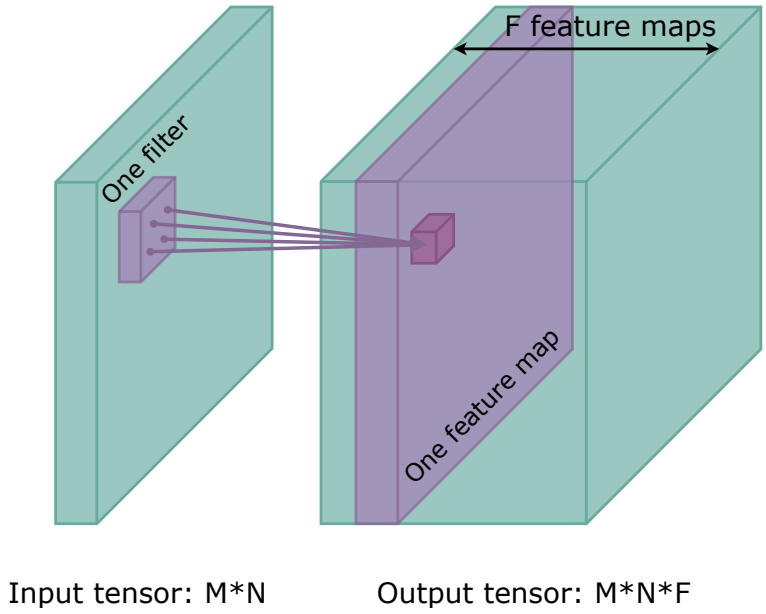


FIGURE 2.3: A schematic view of a convolutional layer.

The convolution operation between a filter W of shape (U, V) and an input tensor A of shape (M, N, K) is defined as:

$$\forall i, j \in [\![1, M]\!] \times [\![1, N]\!] \quad (A \star W)_{i,j} = \sum_{k=1}^{K} \sum_{u=1}^{U} \sum_{v=1}^{V} A_{i-u+1,j-v+1,k} W_{u,v}. \tag{2.5}$$

After filtering, a bias specific to the filter is added, and the result goes through the activation function:

$$\forall i, j \in [\![1, M]\!] \times [\![1, N]\!] \quad (H)_{i,j} = \sigma((A \star W)_{i,j} + b_W). \tag{2.6}$$

Each filter processes sequentially the whole tensor and produces a **feature map** H of the same shape: (M, N). If the layer contains F filters, then F feature maps are produced and stacked together. This way, the output tensor's shape is (M, N, F).

In some architectures, convolutional filters of shape (1,1) are used in order to reduce the number of feature maps. Such filters can be reduced to a single coefficient w. When applied to an input tensor, all its coefficients are collapsed along the channels dimension by a summation operation.

$$\forall i, j \in [\![1, M]\!] \times [\![1, N]\!] \quad (A \star W)_{i,j} = \sum_{k=1}^{K} A_{i,j,k} w. \tag{2.7}$$

Pooling layers

In a pooling layer, the number of feature maps is preserved, while the size of each filter map is reduced. The feature maps are segmented into patches and only one value is kept for each patch. This way, the values that are too low or not significant in each patch are removed. A typical size for the patches is $(2, 2)$, which reduces the size of the map in each dimension by a factor of 2. There are several types of pooling. We will mainly use **average pooling**, where the value that is kept for each patch is the average of its coefficients.

Convolutional networks

A convolutional neural network (as presented on Figure 2.4) is usually composed of a series of **convolutional layers**, alternated with **pooling layers**. The idea of such architectures is to progressively transform the input data into a compact representation by breaking it into small, simple parts.

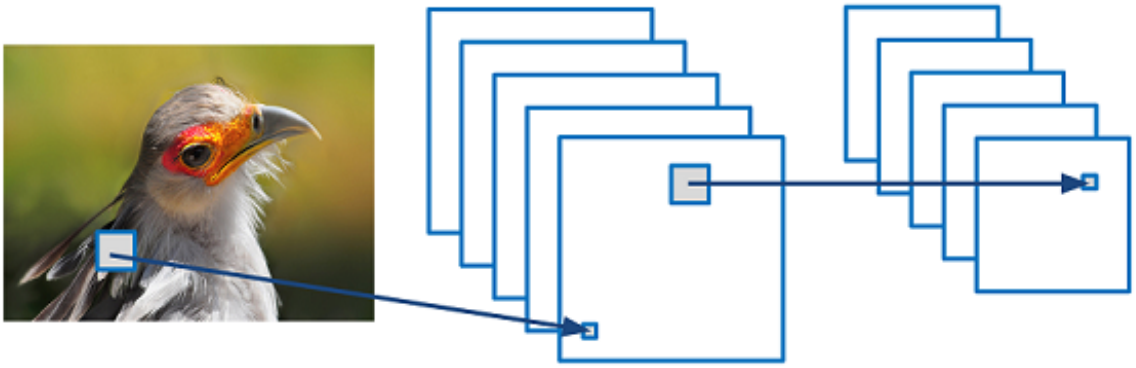


FIGURE 2.4: A convolutional neural network with two convolutional layers.
Image: Adit Deshpande

Convolutional neural networks (CNNs) have good properties for singing voice separation: because they model only local interactions, they have less parameters than fully-connected networks. This results in a faster training and a reduced risk of overfitting. What's more, they are robust to spatial translations (Humphrey, Bello, and LeCun, 2012). In our case, it is useful that the detection of singing voice does not depend on the time axis.

2.4 How to train a neural network

The goal of the training process is to reach a value of θ for which the loss $d(y, \hat{y})$ is as small as possible, since in general, no analytical formula gives the optimal result. In order to do so, we start from a random value of θ and perform successive iterations to change its value into another one, which hopefully reduces the loss.

This update rule for θ is called the **gradient descent**:

$$\theta \leftarrow \theta - \lambda * \frac{\partial d(y, \hat{y})}{\partial \theta}. \tag{2.8}$$

In the case of neural networks, the gradient is computed in two steps:

- First, a **feed-forward** step, in which the input data goes through the series of transformations defined by the network. The distance between the corresponding output and the target is computed via the loss function: it is the **error** of this iteration.
- Then, during the **back-propagation** step, the gradient of the error is back-propagated through the network: according to this value, the weights are updated, and a new iteration can start.

The parameter λ defines the *speed* at which the parameters will be updated. We call it the **learning rate**. There exists various gradient descent algorithms, according to the proportion of the training dataset that is used at each iteration. We will use the **mini-batch** gradient descent, in which the weights are updated after a small number of samples from the dataset have gone through the forward pass: $x = (x_1, ..., x_n)$. This is a compromise between an optimization on the full dataset, too computationally heavy, and an optimization sample by sample, too unstable (Hinton, Srivastava, and Swersky, 2012). We refer to an **epoch** as a constant number of iterations. The duration of the training depends on the number of epochs that must be performed in order to reach a satisfying accuracy.

However, reaching good performances on the training dataset gives no guarantee that the model will be able to give perform well on unseen data. We could have built a model that gives optimal performances only on the training data, and which is useless in real life situations: in other words, a model which lacks **generalization** power. It is the **overfitting** phenomenon, and there are several ways to avoid it. First, we can use a **validation** dataset. The validation dataset is not used for training and must have no sample in common with the training dataset. After each epoch, the loss on the validation dataset is computed. During the training, we can monitor the loss on the validation dataset. If the network looses its ability to generalize, we will observe an increase in the validation loss. We will hence know that the training has lasted long enough and that the network is starting to overfit. We can interrupt the training before this happens, and this method is called **early stopping**. We can also introduce **regularization** within the model. In neural networks, specific layers such as the Dropout layer (Srivastava et al., 2014) prevent the model to learn only a mapping between the inputs and the targets on the training set.

2.5 Deep neural networks applied to singing voice separation

Deep neural networks

We call a neural network **deep** if it has a large number of hidden layers (typically superior to 3, and reaching to more than 100 in the largest models). Such networks can handle complex and non-linear relationships and operations. In this section, we will present recent deep network-based systems that perform singing voice separation.

State of the art systems

Recently, supervised systems based on deep learning have gathered the interest of the audio source separation community.

By learning time-frequency filters to be applied on spectrograms, CNNs are able to exploit spatial and temporal dependencies within the musical data. A very popular architecture for CNN-based singing voice separation is the **convolutional denoising auto-encoder** (CDAE, see Figure 2.5). It consists in a **downsampling path** (an alternance of convolutional and pooling layers), which compress the data into a compact representation. Then, an **upsampling path** restores the original shape of the inputs. In Chandna et al., 2017 and Grais and Plumbley, 2017, a CDAE learns to estimate source spectrograms or directly the spectrogram masks from the mixture's spectrogram. Though CNNs did not systematically oupterform standard fully-connected networks, they have less parameters and train much faster.

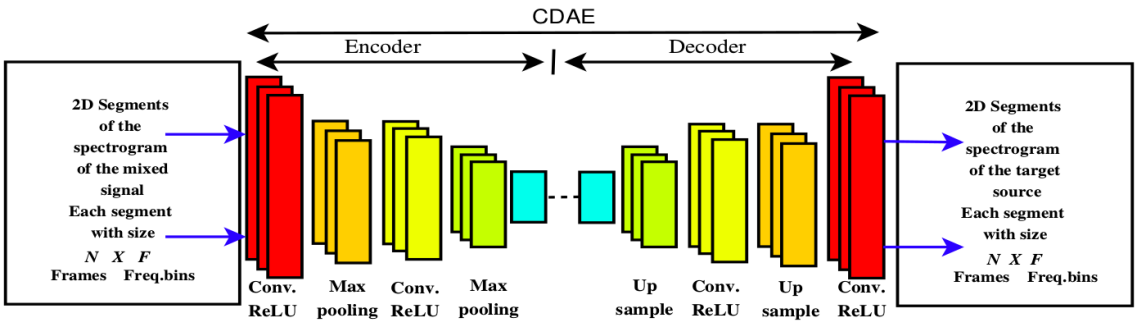


FIGURE 2.5: A convolutional auto-encoder for singing voice separation. Image from Grais and Plumbley, 2017.

In parallel, other CDAE-oriented approaches (Nugraha, Liutkus, and Vincent, 2016) have tried to take advantage of both channels of the audio file by jointly estimating source magnitude spectrograms and spatial covariance matrix. They gave promising results, showing that the stereo information was relevant to use for separation. This confirms our intuition, since the voice is almost always mixed at the center in professional recordings. A more recent paper compared the mono-channel and stereo approach for the same systems (Stoller, Ewert, and Dixon, 2018) and concluded that stereo modeling was mostly benefiting the accompaniment estimations (+3 dB in SDR).

Recently, a U-Net proposed by Jansson et al., 2017 achieved good results. U-Nets are variants of CDAEs featuring skip-connections to preserve precise details at reconstruction (See Figure 2.6. More details will be provided in 4.3). The specificity of this system is that the authors trained the network on a custom dataset built from Spotify's music streaming catalog (see 3.2). We found this setup very interesting and selected it as our starting point.

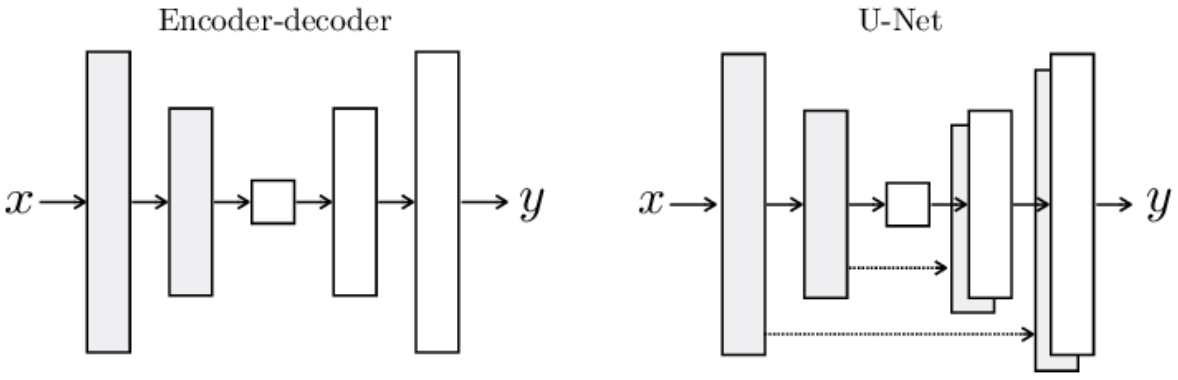


FIGURE 2.6: The U-Net architecture. Image from Isola et al., 2017.

Then, a series of systems proposed by the same team was released too (Uhlich et al., 2017, Takahashi and Mitsufuji, 2017, Takahashi, Goswami, and Mitsufuji, 2018). These systems performed particularly well at the last music source separation challenge (SiSec 2018). In Uhlich et al., 2017, the authors suggests to use data augmentation to enhance performances; hence, we decided to explore this option too. In Takahashi and Mitsufuji, 2017, an architecture supposed to improve the U-Net by using dense blocks instead of simple convolutional layers was introduced. We also implemented a similar system to test it in various situations.

All of these systems operate in the magnitude STFT domain and use masking for reconstruction. Sometimes, another post-processing, like the multi-channel Wiener filtering, is applied (see 4.4.3).

Chapter 3

Multi-tracks datasets for supervised learning and evaluation

In this chapter, we will present the SiSec MUS challenge, which was used as a reference in our work, as well as a some insights about training datasets in machine learning.

3.1 Standard databases

To train supervised systems and evaluate the algorithms, singing voice separation requires **multi-tracks datasets**, where the original mixture, vocal and instrumental tracks are available for each song. For the purpose of MIR research, a few of such datasets were created and made available. A rather extensive list with descriptions and references is available at: `https://sigsep.github.io/datasets/`. Each dataset has a duration comprised between 2 and 10 hours and a production quality ranging from amateur to professional. In our study, we will focus on MUSDB, which is the largest and most up-to-date public dataset.

MUSDB (Rafii et al., 2017) is composed of 150 professionally produced songs, mainly taken from the DSD100 (Liutkus et al., 2017) and MedleyDB (Bittner et al., 2014) datasets. The genres represented are only western music genres, with a majority of pop/rock songs, along with some hip-hop, rap or metal songs. 100 tracks belong to the training set and the 50 remaining ones belong to the test set. They sum up to 10 hours of audio, which occupy together 27G of storage in `.wav` format. The songs are full-length (4 minutes on average), stereo and sampled at 44100 Hz. For each song, five tracks are available: the mix, and four separated sources ("drums", "bass", "vocals" and "others"). There is no mixing effect in the "mix" track (the different sources sum up exactly to the original track). To create the "instrumentals" source, we add up the tracks corresponding to "drums", "bass" and "others". Then, we focus exclusively on these two sources: "instrumentals" and "vocals".

These public datasets are very convenient for evaluation, but are rather small to train supervised systems, and often lack representativeness. Overall, there are very few publicly available multi-track recordings. In the literature, this data scarcity is often cited as one of the main limits to building efficient and scalable supervised singing voice separation algorithms.

3.2 Building a singing voice separation dataset

Since a large and diverse training dataset is likely to enhance separation performances, we figured out a way to build one. To do so, we inspired ourselves from a paper published recently: Humphrey et al., 2017. It presents a method to build a dataset based on a music streaming catalog. To take advantage of the catalog, the idea is to exploit the **instrumental versions** that are released by some artists along with the original songs.

The procedure, schematized on Figure 3.1, is the following. The first step is to find all possible *instrumental/mix* track pairs within the catalog. This matching is done using metadata (the song's title and the *version* field) and audio fingerprints. The fingerprinting algorithm used is described in Wang, 2003 and is known for being used by the *Shazam* application. At first, 62 048 pairs were found in the catalog with this process. Then, we perform a few filtering and homogenization operations:

- If both tracks have a duration difference greater than 5 seconds, the pair is removed. In this case, we assume the metadata were mismatch and both tracks did not represent exactly the same song (it could be a remastered edition, for example).
- Songs longer than 5 minutes are also filtered out. This makes it easier to compute the signal correlations in the next step.
- Both tracks are temporally re-aligned using autocorrelation.
- Finally, the loudness of both tracks are equalized.

To produce a triplet **(mix, instrumentals, vocals)** from the pair **(mix, instrumental)**, we perform a half-wave rectified difference of both spectrograms. Eventually, 43,391 triplets were created, which represent 69% of all original matches. The full filtering and re-alignment procedure took about 24 hours on a 32-cores machine. The vocal estimates occupy 1.7T of storage in `.wav` format. Using metadata, we found out that a great variety of genres was represented in this corpus, which contains a majority of pop songs. Other genres include rap, kid music and movie soundtracks.

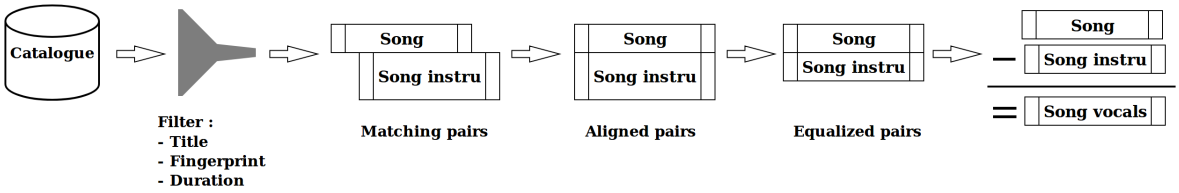


FIGURE 3.1: A schematic view on the creation of the Catalog dataset.

This database will be named "Catalog" database in the following of the report. Despite its very large volume compared to the standard datasets described in 3.1, we must keep in mind that it was not professionally produced for separation purposes and is necessarily of a lower quality. The main issues that we found in the dataset are:

- The half-wave rectified difference between the mix and the instrumental version should theoretically correspond to the vocal part. But in practice, even a small misalignment between both tracks can produce residuals in the vocals. A listening test on a small subset (40 tracks) reveals that this happens in almost 50% of the tracks.
- Even with a good alignment, there might be other sources of residual energy; typically, non-linearity effects introduced during production (mastering, compression ...). The original article equally mentions a "sensibility to mixing quality" in the vocal estimates.
- Duplicates are present in the dataset. For example, several pairs have the same mix identifier, while having different instrumental versions. The total number of different instrumental version identifiers reveals that such pairs account for 10% of the dataset.
- We noticed genre bias in the dataset. For example, kids music, which is very often released along with an instrumental version, is over-represented in the dataset compared to its actual proportion in the catalog.

Music Genre	Mix URL	Instrumentals URL
Irish music	deezer.com/en/track/67468121	deezer.com/en/track/67468122
Religious song	deezer.com/en/track/115447012	deezer.com/en/track/115447094
Kids song	deezer.com/en/track/136839184	deezer.com/en/track/136839186
Pop	deezer.com/en/track/67469788	deezer.com/en/track/67469790
Rap	deezer.com/en/track/127593433	deezer.com/en/track/127593409

TABLE 3.1: Examples of (instrumental, mix) pairs that are available in Deezer's catalog.

- Lastly, if the metadata matching is not perfect, there might be tracks that feature no singing voice at all from the beginning. In this case, the "vocals" part is only residual noise. Reversely, some "instrumentals" tracks contain choirs. These cases are difficult to point out by automatic systems.

Accordingly, we can say that the Catalog database provides us with a large amount of weakly labeled data. The instrumental part is professionally-produced, while the vocals are only estimates. We summed up in table 3.1 some examples of satisfying **(instrumental, mix)** pairs that were available in the catalog.

To train the models, we selected one segment of 11.88 seconds from each song, avoiding the intro (the 20 first seconds) and the outro (20 last seconds), where lyrics are often missing. The duration of the segments was chosen according to Jansson et al., 2017. All samples are stereo and resampled to 22050 Hz. They add up to a total training material of almost 6 days. Pilot experiments with 2 segments per song did not result in any significant improvement in performances, and for storage reasons, they are difficult to conduct, so we did not experiment further on huge datasets.

In addition to the Catalog dataset, we also used an internal multi-track dataset, from which we were able to extract 19,502 segment of 11.88 seconds each. We will name this dataset "Bean". The Bean dataset contains a majority of pop/rock songs and its quality is higher than the one of the Catalog. Once again, the segments are stereo and are resampled to 22050 Hz. The total training material in this dataset is around 64 hours.

3.3 The dataset problem in supervised machine learning

The characteristics of the ideal training dataset: a vague concept

Machine learning is used to solve very various tasks, and if used in a supervised approach, one must first find or build a suitable training dataset. The quality of the training dataset impacts directly the final inference capacity of the algorithm (Japkowicz and Stephen, 2002). Ideally, we would like the training dataset to be as representative as possible of any situation that can be encountered during the inference task. But a certain number of papers point out bias in the training datasets that limit performance, such as: Dupin et al., 2011, Hansen et al., 2004 and Torralba and Efros, 2011. We will present here these papers that studied the impact of training datasets on performances.

A first paper that studied the impact of the training dataset for regression is Dupin et al., 2011. The domain is botanics and the task is to predict the evolution of invasive species. Different models including a SVM, a PCA and other botanics-specific models are trained with various features and dataset sizes. The authors concluded that providing the models with more training data leads to a higher accuracy in predictions. But this expected result does not seem to generalize to any other domain. For example, in cardiac imagery, where no

error can be tolerated, the authors in Hansen et al., 2004 recommend to favor quality over quantity in the training dataset. Their goal is to study an intelligent image compression algorithm called k-t BLAST, and to minimize its reconstruction error. For this task, they found that "an increased amount of training data increases susceptibility to misregistration of the training data".

Our intuition after this short literature review is that the correct amount and quality of training data varies according to the task, the model and the domain. To our knowledge, no such study was found in audio source separation. What's more, in singing voice processing, according to Gómez et al., 2018, "there is no agreed-upon methodology to document and evaluate datasets in terms of coverage, e.g. with respect to singer, gender, race, language or singing style".

Bias are present in any dataset

In image recognition, where large training datasets are more often available, datasets bias is a very known issue. According to Torralba and Efros, 2011, most datasets have bias, that give them a distinct *signature*. Over time, they became "closed worlds", since the systems generally try to reach the best performances for a given dataset. This is why some systems end up overfitting on a train/test data setup, even if this means modeling directly the bias. The author's recommendations to such issues are:

- Perform *cross-datasets generalization*.
- To avoid any selection bias, use an automatically collected dataset.
- Perform data augmentation.

In our setup, the Catalog dataset would theoretically be the least biased one, since it has been collected automatically. However, we noticed that some genres, like kids songs, are over-represented. We will make sure that systems that were trained on a given dataset can be tested on another dataset (for example, training on Catalog and testing on MUSDB). Data augmentation will also be studied in 4.4.4.

3.4 Evaluation campaigns

To evaluate and compare singing voice separation systems, evaluation campaigns are regularly organized. The main ones are SiSec MUS[1] and MIREX[2]. For this study, we will focus on SiSec, which is more up-to-date (Stöter, Liutkus, and Ito, 2018). The organizing team of SiSec has released standard datasets, along with tools to manipulate them. The last two datasets that were released are DSD100 in 2016 and MUSDB in 2017. They come with a split in a **Dev part** (to train supervised systems) and a **Test part** (to compare the performances of the systems).

The standard procedures for the singing voice separation task is the following:

- Each team can present one or several systems, supervised or not, trained with additional data or not;
- The systems take as input the audio files of the mixture of the test dataset and output estimates of each source (vocals and accompaniment in our case);

[1] https://github.com/sigsep/sigsep-mus-2018
[2] http://www.music-ir.org/mirex/wiki/2016:Singing_Voice_Separation

- The provided toolbox outputs scores that are published on the official website: https://github.com/sigsep/sigsep-mus-2018

A few salient aspects of the results of the last two campaigns are worth noticing. First, all model-based methods (such as REPET or RPCA, presented in 1.5) are outperformed by supervised systems. The best systems are now good enough for automatic karaoke generation (separation of the instruments): tracks without distortion can be separated almost as well as an oracle would do. Oracles are systems that use ideal masks, computed from the ground truth sources, to perform the separation. Oracles based on ideal ratio masks yield very good results in terms of both perception and objective metrics. Still, no system was found to be convincing enough to generate realistic separated vocals. The audio demonstrations can be found at: https://sisec18.unmix.app/#/.

By analyzing the systems, the authors of the campaign found that state-of-the-art algorithms ought to feature multichannel modeling, data augmentation or a fusion of two complementary systems. The authors of the campaign also find it interesting to investigate the effect of the training data on the performances.

Finally, we must keep in mind that the SiSec performance evaluation has limitations. In particular, the Test set is publicly available. This is very convenient to compare a system to the state-of-the-art without waiting for the next campaign, but it also makes it possible for all teams to optimize the model's hyper-parameters using the test set. It is a form of overfitting that cannot be avoided in the SiSec's challenge.

Chapter 4

Experiments and results: the impact of various data-related parameters on performances

During this internship, we studied uninformed systems, meaning that we worked exclusively on audio data, without using metadata. We also focused our efforts towards supervised models which rely on deep network architectures. The reason for this it that these methods achieve the current state-of-the-art results, and they are likely to better take advantage of the different datasets. In this chapter, we will present our experimental setup and our results. The comparison between the different setups constitutes our main contribution.

4.1 Baseline systems

In our test set (the 50 songs from MUSDB Test), the ground truth signals are available for evaluation. We can also use these reference signals to estimate an "ideal mask" as a high baseline for all the systems that use masking. The ideal mask is applied to the mixture spectrograms and then, "ideal" audio estimates can be computed and evaluated. This oracle system is called **IRM1** in the SiSec 2018 challenge. It will be used as our upper baseline.

$$M_{vocals}^{IRM1} = \frac{V_{vocals}}{V_{instrumentals} + V_{vocals}} \tag{4.1}$$

$$M_{instrumentals}^{IRM1} = \frac{V_{instrumentals}}{V_{instrumentals} + V_{vocals}} \tag{4.2}$$

A low baseline is also computed: it corresponds to the case when the mixture spectrogram is used directly as estimates of each source. This system is called **MIX** in the SiSec 2018 challenge.

$$\forall i, j \quad M_{vocals_{i,j}}^{MIX} = M_{instrumentals_{i,j}}^{MIX} = 0.5 \tag{4.3}$$

We implemented both baselines using the provided code that can be found at: https://github.com/sigsep/sigsep-mus-oracle. Then, we evaluated them using the Museval toolbox. The toolbox takes as input a directory containing the estimates of the separated sources for each song and the directory containing the reference sources for each song. It returns a series of `.json` files, one for each song. The Figure 4.1 shows the structure of the input and output folders for this step.

Each score file contains the SDR, SIR and SAR metrics for each source and for each *frame* of the song. By default, the frame length is 1 second. A snippet of a typical output of Museval for a given song can be found in Appendix A.

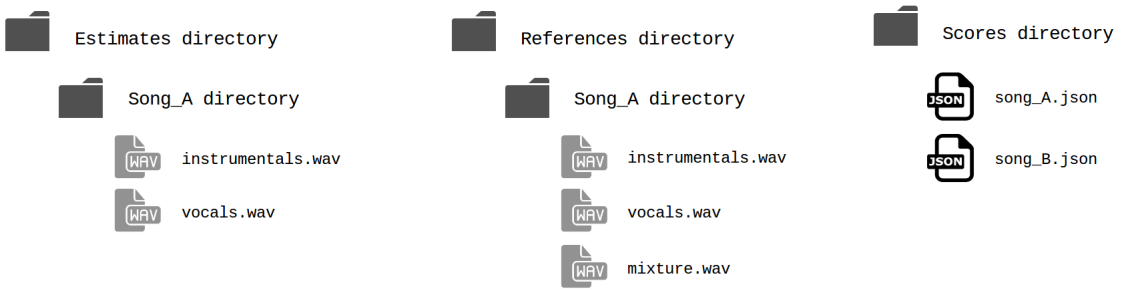


FIGURE 4.1: The required architecture for evaluation.

The last step is to aggregate the metrics over the dataset. First, we select a metric and a source. For each song, we compute the median of the metric over all the frames of the song. Then, we compute simple statistics (median, percentiles) over the score of all the songs in the test dataset. This methodology is inspired from the one used by SiSec 2018 for the publication of the results[1].

4.2 Experimental framework and methodology

In order to study the impact of data-related aspects on performances, we have set up a series of experiments. Each experiment consists in a combination of a training dataset, a network architecture and some training and inference settings, like the post-processing technique. The experiments consist in three steps. First, two identical network, one for each source, are trained on the chosen dataset. Then, the estimates of the separated sources of each track of the test dataset (generally MusDB Test) are inferred. Finally, the Museval toolbox produces separation scores for the experiment that can be analyzed. We will detail the different steps below.

Data preparation

The preparation of a dataset for training is rather simple. A dataset is a collection of samples, each of one representing a segment of a song. For each sample, three audio files are referenced: the mix, the instrumental and the vocal part. The mix serves as input to the separation system and the two others as targets. Each segment of audio must be transformed into a magnitude spectrogram before use. We chose a simple STFT representation because it offers meaningful time-frequency information at a low computational cost and is easy to reverse for inference.

The STFTs are computed using the `Librosa` Python library (McFee et al., 2015). The window size is 2048 and the step size is 512 (75% overlap). The temporal length of each segment is 11.88 seconds. We chose these settings such that the dimensions of the spectrograms, after removing the highest frequency band, are a power of two: (channels, time steps, frequency bins) = (2,512,1024). This is important, because the network architectures that we will use reduce the dimensions of the spectrogram by a factor which is a power of two (see 4.3). The features (the magnitude spectrograms) of each sample are pre-computed using an internal micro-service and stored in cache. The computation of the features is heavily parallelized to reduce its duration. They represent 41GB for MUSBD, 283GB for Bean and 478GB for the Catalog dataset, stored in 32 bits floating-point.

[1] https://github.com/sigsep/sigsep-mus-2018-analysis

Training

Even if it is possible to train a two-headed network to simultaneously produce estimates of the vocals and instrumentals, we chose to train one network for the vocals and another one for the instruments. This allows each network to specialize and learn audio features that are specific to each type of source. This type of approach is almost systematically used in recent literature (Uhlich et al., 2017, Jansson et al., 2017).

The design of the training part of the experiment is made using the internal audio processing and machine learning librairies from Deezer. These libraries are based on Keras[2] with Tensorflow backend[3]. For training, we define an epoch as 800 gradient descent steps. We define a validation step as a complete forward pass over the validation dataset. We train each network during 500 epochs and monitor the loss over the training and validation dataset. At the end, we keep the configuration that gave the best results on the validation dataset (**early stopping**). The layers are initialized using the `he_uniform` distribution (He et al., 2015) and a fixed seed for reproducibility. The batch size is set to 1, because larger batches do not fit in memory. The training is done using CUDA parallel computing on a NVIDIA GTX 1080 Ti card with 11GB of RAM.

Inference and evaluation

Once the training is finished, we performed a forward pass on the test dataset, after dividing each track into segments. The complex spectrograms of each source are computed using magnitude masks and the phase of the original mix, as presented in 1.3. Optionally, a post-processing operation is applied. The STFTs are reversed to audio signals and the segments are finally re-assembled into full-length songs. We did not find any paper mentioning a particular way to re-assemble the separated segments together, so we simply concatenated them with each other chronologically. We checked that no artifact was created at the borders by looking at the waveform on Audacity on some examples. We also plotted the SDR score of the test songs as a function of time, using the framewise scores. If the concatenation was not precise enough, we would expect periodical drops in scores around the borders of the segments. But no such pattern was visible, so we deduced that this reconstruction method was viable.

Finally, the songs are upsampled back to 44100 Hz and the scores are computed using the Museval toolbox.

Training networks for singing voice separation is not obvious, since the performances are not directly accessible as an error rate produced by the network. One must first perform inference and evaluation, which last respectively 27 and 25 minutes on a 32-cores machine. This makes the tuning of the hyper-parameters tricky.

4.3 The selected network architectures

We focused our efforts towards the implementation of two fully-convolutional neural networks.

U-Net

This network architecture was taken from Jansson et al., 2017 (Figure 4.2). The U-Net looks like a convolutional auto-encoder, except that it features skip-connections that bring back

[2] https://keras.io/
[3] https://www.tensorflow.org/

detailed information lost during the encoding stage to the decoding stage. During the encoding path, each convolutional layer halves the size of the feature maps thanks to a stride equal to 2. It also doubles the number of feature maps, creating a small and deep representation. The upsampling operation during the decoding path is done using **transposed convolutions**, as presented in Appendix B. All the information provided in the paper was kept as it. Only the initial shape of the features is slightly different. The re-implemented network has the same number of layers, the same activation functions, and it was trained using the same optimizer (Adam).

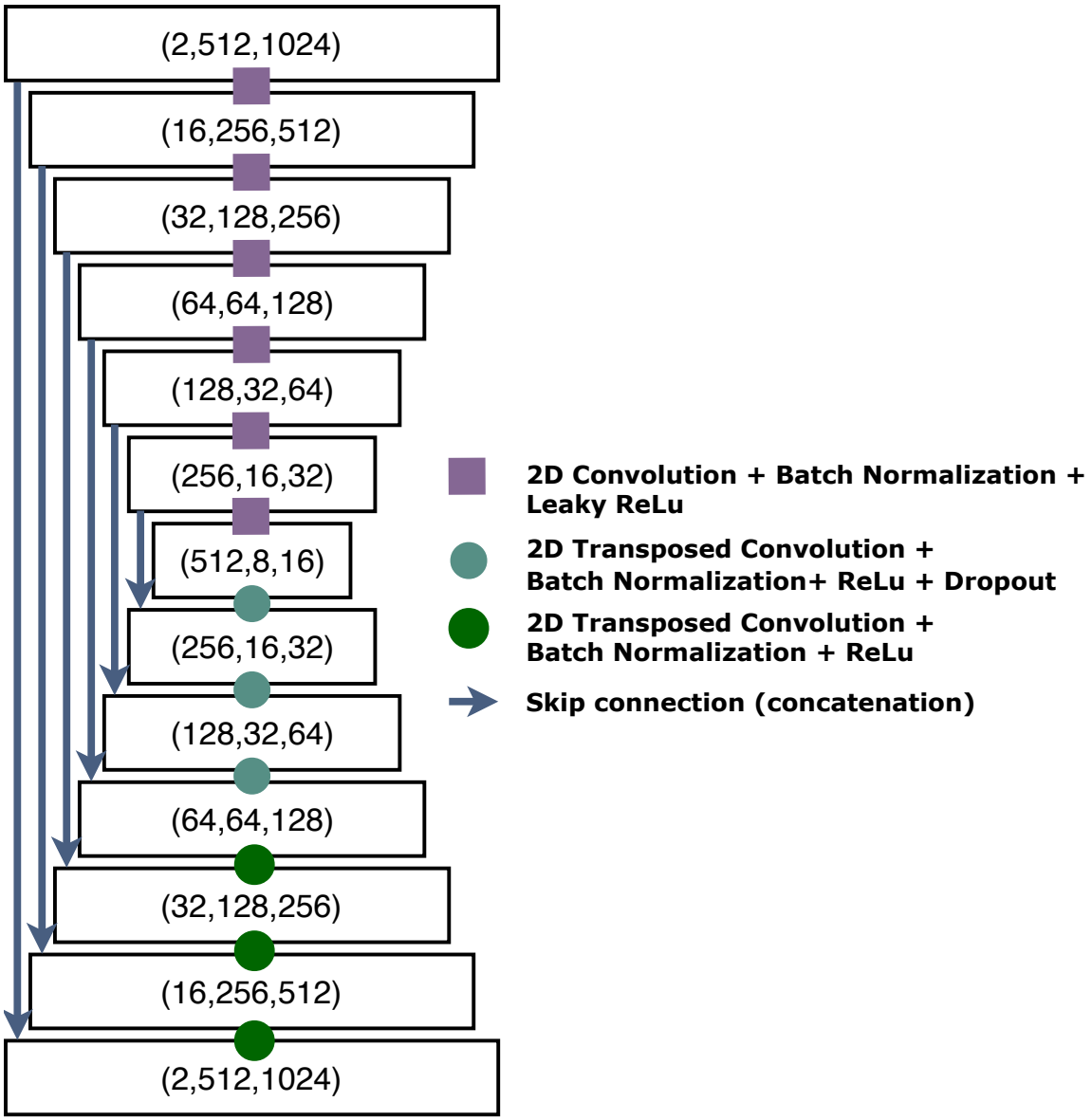


FIGURE 4.2: The U-net architecture.

The loss is, as presented in the paper, the L_1 norm of the difference between the target spectrogram Y and the input spectrogram X multiplied by the output $f(X, \Theta)$. This way, the network learns directly a mask:

$$Loss(X, Y, \Theta) = ||f(X, \Theta) \odot X - Y||_1. \tag{4.4}$$

Given our feature size, the learning rate is set to 0.0001. The batch size is 1. This network requires 3.4GB of GPU RAM for training, and has 9,822,725 parameters. The reconstruction

is done with the masking technique presented in 1.3.

In this architecture, all filters have a shape of (5,5). This way, at the lowest resolution, each filter covers a patch of 7.425 seconds in time and 320 of the initial frequency bins. We notice hence that this network can model very large time and frequency contexts.

DenseNet and MMDenseNet

This second network architecture was inspired from Takahashi and Mitsufuji, 2017 (Figure 4.3). The Densenet is basically a U-net, except that convolutional layers are replaced with **dense blocks**, a series of convolutional layers in which the input of each layer is the concatenation of the outputs of all preceding layers. Each dense block is described by a number of layers L and a growth rate k (number of new feature maps produced by each layer within the block). Typically, in our architecture, $L = 4$ and $k = 12$. This time, the size of the feature maps is not halved because of the stride in the convolution layers, but because of intermediary average pooling operations.

Compared to the original paper, we had to make some adaptations in order to use the same features as for the U-Net:

- Our audio segments are downsampled to 22050 Hz, while there is no mention of downsampling in the paper.
- The authors perform the STFT with a window size of 2048 and 50% overlap: we also have a window size of 2048, but with 75% overlap.
- The temporal length of the segments is not explicitly specified in the paper, but we estimated that it was around 8 seconds, versus 11.88 seconds for our features.
- The training is performed using the RMSProp optimizer. Instead of setting the learning rate to 0.001 and then to 0.0001, we set it directly to 0.0001 to avoid instability issues that we had when using a larger learning rate.
- The batch size is kept to 1, in order to the activation tensors to fit in memory.

For this system, some architecture information was missing in the original paper and had to be inferred from different sources, namely Huang et al., 2017 and Jégou et al., 2017:

- The number of filters in the initial convolution is equal to the growth rate k or twice the growth rate: that's why we set it to 24.
- During the downsampling step, (1,1) convolution filters halve the number of feature maps.
- During the upsampling step, the input of each dense block has $k * L = 48$ feature maps.

In total, the DenseNet requires 5.5G of GPU RAM and has 509,031 parameters. It occupies more memory space because it has a larger number of feature maps, but the smaller filter shape (3,3) reduces considerably the number of trainable parameters. Indeed, at the lowest resolution, each filter covers a patch of 0.557 seconds in time and 24 of the initial frequency bins. We notice that this network can model only smaller time and frequency contexts. The network fits directly the target spectrograms with a L_1 loss. The reconstruction, once again, is done with the masking technique presented in 1.3.

We also implemented the MMDenseNet, which is a multi-band variant of the DenseNet (Figure 4.4). The idea is to jointly train a full-band DenseNet and several other DenseNets

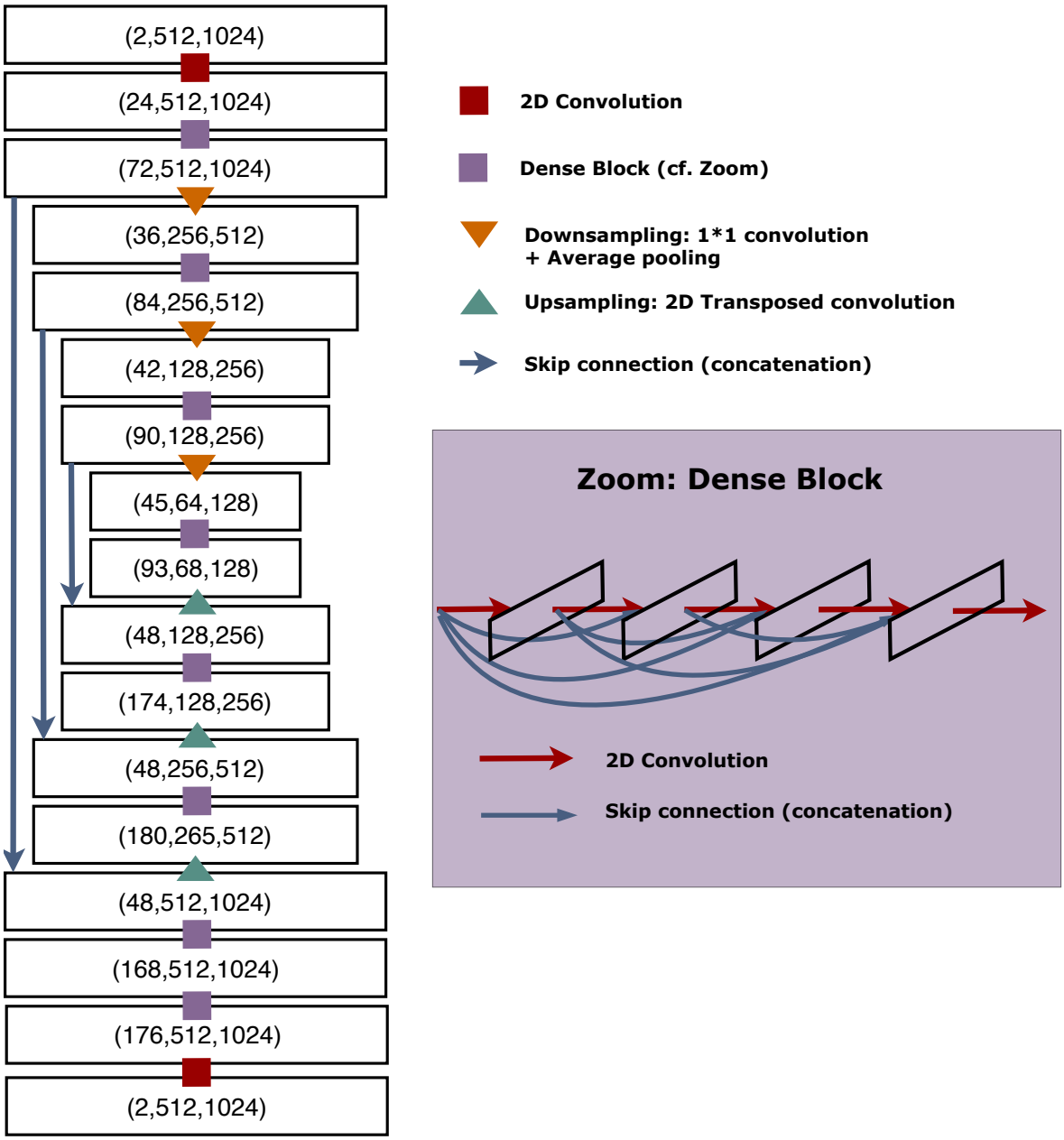


FIGURE 4.3: The DenseNet architecture.

which are specialized in a given frequency band. According to the original paper, this is supposed to provide a significant performance gain, since the spectrogram patterns are very different in high and low frequencies. We used the suggested configuration with two frequency bands. We end up with 672,115 parameters for this model, with a GPU RAM requirement of 7.1G.

4.4 Experiments

In this section, we test our network architectures in various training settings. We studied the impact of the training dataset, the training loss, the post-processing method and the type of data augmentation. The goal is to help the intuition in guessing good hyper-parameters for each model, even if we cannot perform an exhaustive search. As already mentioned, the

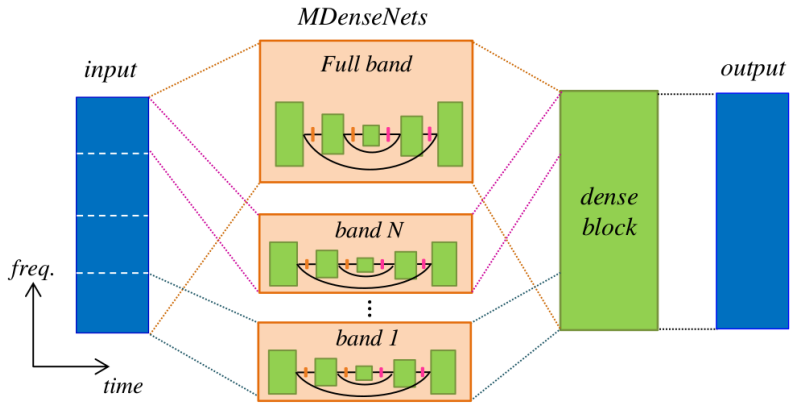


FIGURE 4.4: The MMDenseNet architecture.

Dataset name	Number of training samples	Number of validation samples
MUSDB Dev	1437	415
Catalog	37799	2000
Catalog + MUSDB Dev	43835	2414
Bean	17502	5000
Bean + MUSDB Dev	18939	5414

TABLE 4.1: Partition in training and validation for each dataset.

feature size of both targets and inputs of the networks is (1, 2, 512, 1024). Audio demonstrations can be listened to at:
https://drive.google.com/drive/folders/1Ey_RNAbul3sP-qJ2IfT_VH8N4EL63qfe?usp=sharing

4.4.1 Experiment 1: different training datasets

In this part, we trained the three network on each of the following datasets: MUSDB Dev, Catalog, Catalog added to MUSDB Dev, Bean and Bean added to MUSDB Dev. Each dataset was split into a training and a validation part, following table 4.1. For MUSDB Dev, we used the 80 first songs as the training dataset and the 20 remaining ones as the validation dataset. For Catalog and Bean, we selected the size of the dataset in such a way that the validation step was not too long compared to the training time for one epoch. Another table was made to provide insights on the training time required in such configurations (4.2).

The systems were tested on MUSDB Test, which contains 50 songs for a total of 1073 samples. The reconstruction was made using the masking technique, as described in 1.3. The training loss is the L_1 loss (`mean_absolute_error` in Keras).

On the following figures, we displayed the objective metrics for each network, trained on each dataset and for each source. The low baseline (MIX) and high baseline (IRM1) were represented as well. The low baseline (MIX) achieves optimal SAR scores, since the mixture contains no separation artifacts at all.

The first thing we notice on figures 4.5 to 4.10 is that the DenseNet and MMDenseNet have lower performances than the U-Net. One possible explanation for this is that they are less deep and thus have shorter temporal and frequential context for the filters (see 4.3). They also have less parameters, which possibly explains why they have low performance

Network	Dataset	Training time
U-Net	MUSDB	10h
U-Net	Catalog	24h
U-Net	Bean	36h
DenseNet / MMDenseNet	MUSDB	24h
DenseNet / MMDenseNet	Catalog	3 days
DenseNet / MMDenseNet	Bean	1 week

TABLE 4.2: Approximate training time for each model.

FIGURE 4.5: The U-Net results on different datasets (instrumentals).

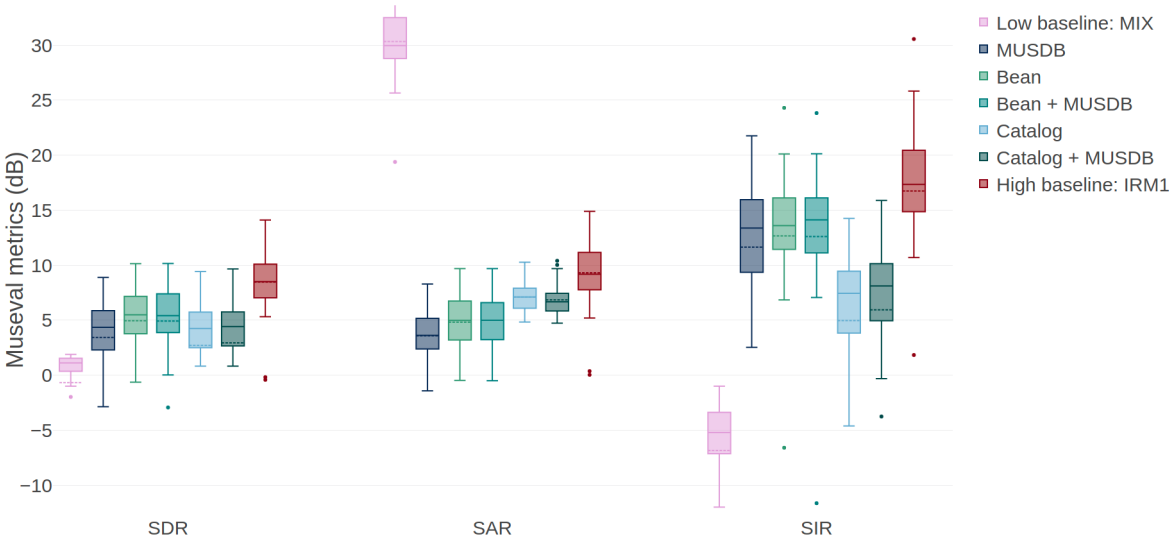


FIGURE 4.6: The U-Net results on different datasets (vocals).

gain when augmenting the size of the dataset. Only their SAR seems to be impacted. Moreover, the multi-band extension does not seem to bring performance gains compared to the single-band implementation.

The impact of the different datasets is more visible for the U-Net (Figures 4.5, 4.6). First, we notice that the Catalog dataset does not improve all the metrics. Compared to MUSDB

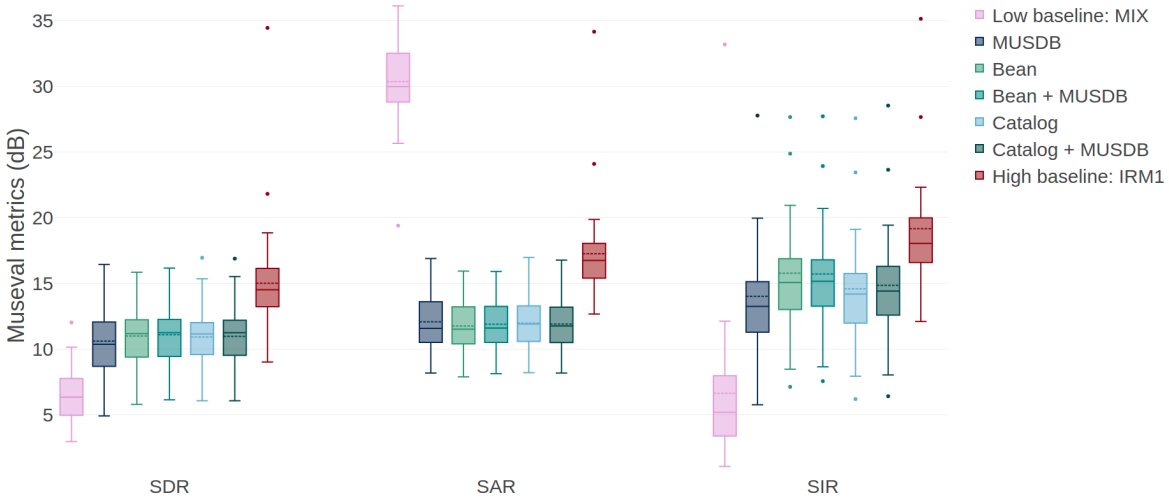


FIGURE 4.7: The DenseNet results on different datasets (instrumentals).

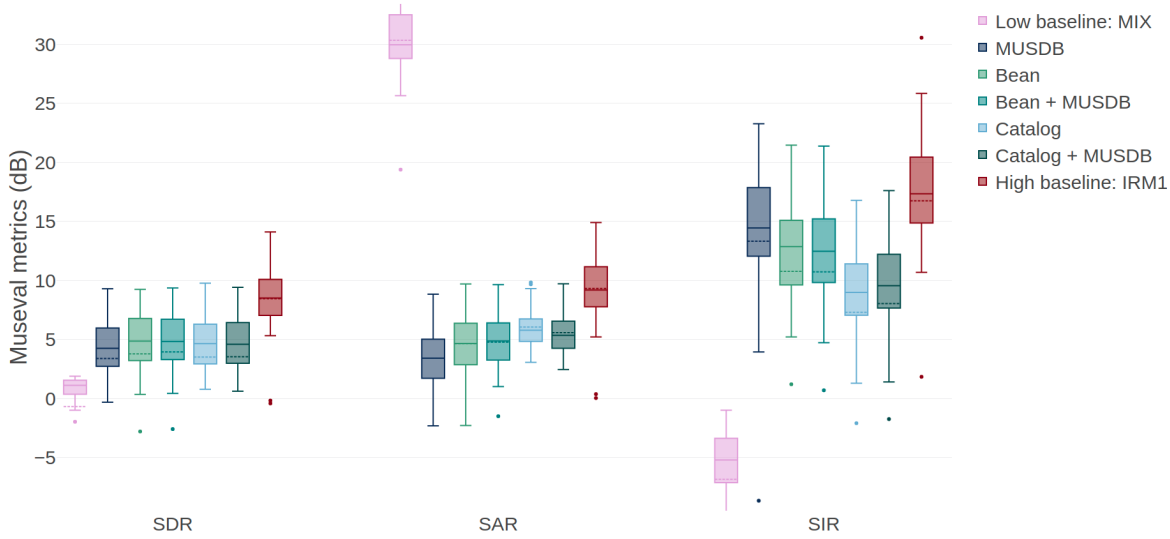


FIGURE 4.8: The DenseNet results on different datasets (vocals).

alone, it yields in higher SAR, but lower SIRs, resulting in a similar SDR. The effect is particularly visible on the vocals. It makes sense with the way the Catalog training dataset was built (see 3.2): the recordings are professionally produced, so the mixture quality is good, but significant leaks remain in the vocal target. In this case, where we train with weakly labeled data, it is relevant to keep MUSDB in the dataset. Even if MUSDB is much smaller than the Catalog dataset, it seems to "guide" the training and to have a positive effect on the metrics.

However, we cannot see significant differences between Bean and Bean + MUSDB. This could be because the Bean dataset in itself is representative enough to be used alone for training. It makes a significant increase in performance for the U-Net, and a small performance gain for the others.

4.4.2 Experiment 2: different losses

In the recent literature, the tendency to train singing voice separation systems is to use a rather simple loss, such as the mean squared error (MSE), or the L_1 norm of the difference

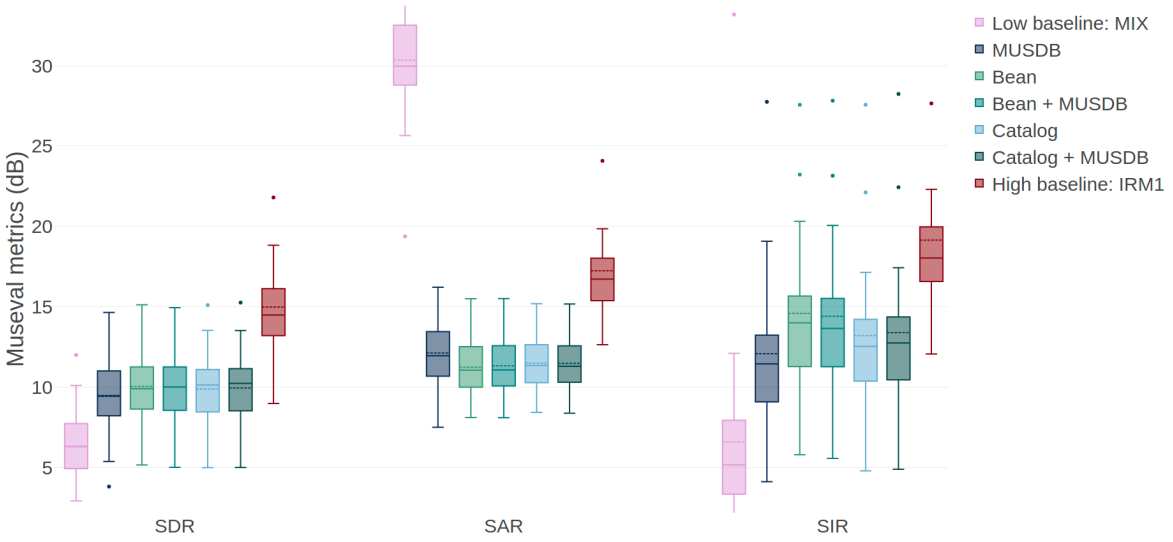


FIGURE 4.9: The MMDenseNet results on different datasets (instrumentals).

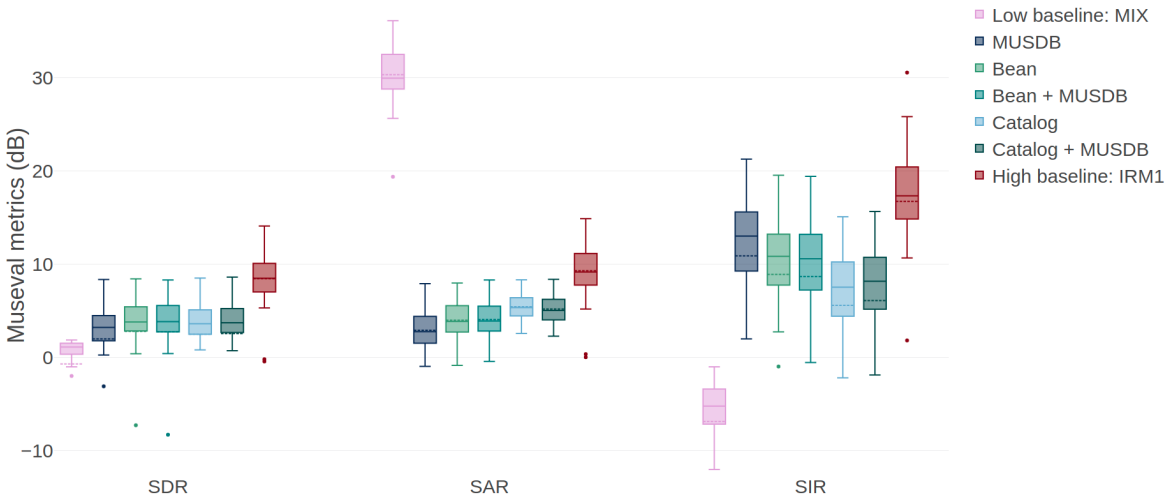


FIGURE 4.10: The MMDenseNet results on different datasets (vocals).

between the estimated and target spectrograms. Some papers mention empirically-added penalty terms to their losses. For example, in Chandna et al., 2017, the authors build a complex training loss based on their knowledge of the difficulties of the dataset (in particular, a $L_{othervocals}$ term enforces the difference between the vocals and other harmonical instruments, such as guitars). But no study was found mentioning the most adequate loss to train neural networks for separation.

When using a non-negative matrix factorization for separation, most authors use either the Kullback-Leibler divergence (Arberet et al., 2010) or Itakura-Saito divergence (Ozerov and Févotte, 2010). Some systems also work on spectrograms where the coefficients are logarithmically scaled (Schmidt and Olsson, 2006). Following this intuition, we conducted pilot experiments to see if these types of distances could be adapted as training loss for neural networks. We implemented a Kullback-Leibler divergence as:

$$d_{KL}(y, \hat{y}) = x \log\left(\frac{x}{y}\right) + y - x \tag{4.5}$$

and a Itakura-saito divergence as:

$$d_{IS}(y, \hat{y}) = \frac{x}{y} - \log\left(\frac{x}{y}\right) - 1. \tag{4.6}$$

We also implemented a logarithmic mean squared error, to model a behavior close to a training on logarithmically-scaled spectrograms:

$$d_{log}(y, \hat{y}) = ||\log(y) - \log(\hat{y})||_2^2. \tag{4.7}$$

Unfortunately, these three error functions did not bring any improvement compared to the L_1 loss and MSE. Both divergences produced estimates that were approximately the mix, with scores close to our lower baseline, while the logarithmic MSE gave approximatively the same results as a standard mean squared error. Consequently, we restricted ourselves to simpler loss functions in the next experiments. On figures ,4.11 and 4.12, we displayed the metrics for U-Nets trained with a L_1 loss and a MSE on the MUSDB and Bean datasets.

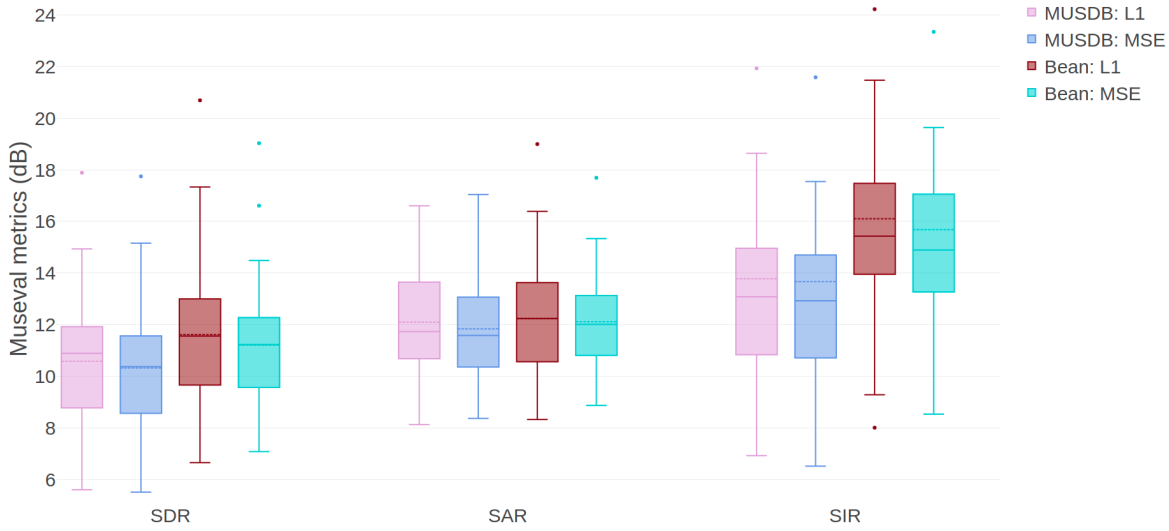


FIGURE 4.11: The U-Net trained with different losses on two datasets (instrumentals).

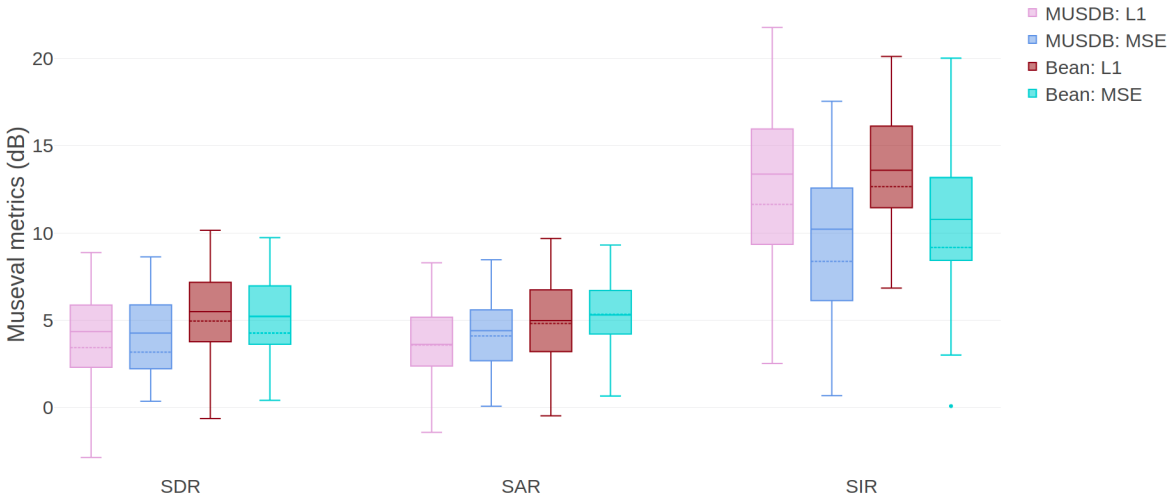


FIGURE 4.12: The U-Net trained with different losses on two datasets (vocals).

These results are surprising for us: since the metrics are computed in terms of power, we expected them to be better optimized by the MSE. Still, the results are consistent across both datasets.

Perceptively, we can hear some differences between the estimates of both configurations. (This is only an informal listening test, that should be completed by standard listening tests for confirmation.) On average, the sources sound better separated using the L_1 loss, and less interferences from other sources remain. But small artifacts are audible, especially in the residuals, producing distortion effects. Using the MSE, more residuals are audible, but they can be identified as music more easily, which makes these estimates sound more natural in our sense. Overall, there are less residuals in the instrumental tracks than in the vocal tracks. So if we were to build a karaoke application, we would suggest using the MSE, since small leaks of voice can be better tolerated than non-musical distortion effects.

4.4.3 Experiment 3: different post-processing methods

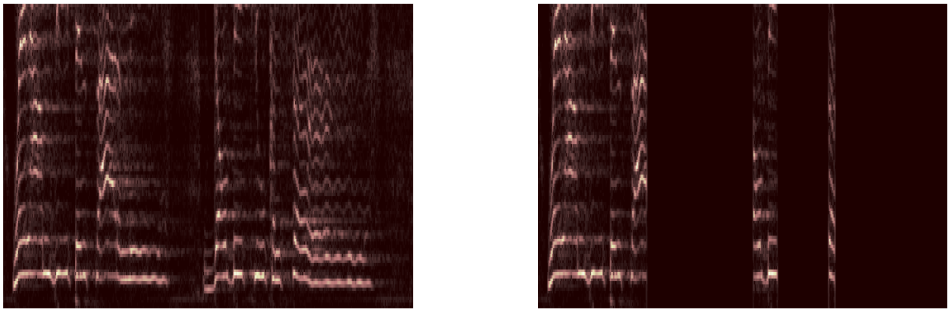
In this experiment, we tried to improve the results after training, during the reconstruction process. We trained a U-Net on the Bean dataset and prepared three post-processing methods: a multi-channel Wiener filtering, a sigmoid-like function to have a binarization effect on the masks and and a simple energy thresholding algorithm. We will detail the three processes below. This time, we computed the metrics directly on the **validation** dataset (5000 samples from the Bean dataset), where the performances of the network are optimal. This way, we eliminate the generalization problem that is inherent to our evaluation protocol. This also prevents us from overfitting on the test dataset by making a grid search on the best post-processing parameters for this set. Note that the results in this section are de facto not comparable to those in other sections.

Energy threshold

The simplest idea we started with is to apply an energy threshold on the predicted vocal signal. Indeed, the vocal part in the original tracks is not always active, and when it is not, we would like to hear silence, which is not always the case in the predictions: leaks from the instrumental part can remain.

This post-processing is applied after the reconstruction, directly on the waveform of the **vocal** part. Let b be the minimum energy ratio, compared to the maximum energy of the signal, for the estimated signal to be considered as containing voice and not only low-energy residuals. The energy of the signal is computed for each time sample of the song using a moving average window of duration 0.23s. Let a be the minimum duration of a silence, in seconds. If we detect a level of energy lower than b during a period longer than a, it is considered as a remaining of instrument, which should be a silence, and it sets the signal to 0 (with a very basic fade in - fade out effect). An example of a vocal part processed with $a = 0.5s$ and $b = 0.2$ can be seen on Figure 4.13

Pilot experiments showed that the effect was audible and perceptively interesting for $a = 1s$ and $b = 0.5$. We experimented further with shorter silences ($a = 0.5s$) and a less strict energy threshold ($b = 0.2$). The figures 4.14 show the results for all combinations of these parameters. We notice that for all our settings, the SIR improved. In return, artifacts were created and the SAR decreased slightly. The settings where $b = 0.5$ show the biggest impact on the metrics: the SIR raises of 1.5 dB. Unfortunately, this does not compensate for the drop in SAR. The best compromise seems to be $a = 1, b = 0.2$, which slightly improves the SDR.



(A) The original spectrogram. (B) The spectrogram after post-processing.

FIGURE 4.13: Illustration of the effect of the energy threshold post-processing.

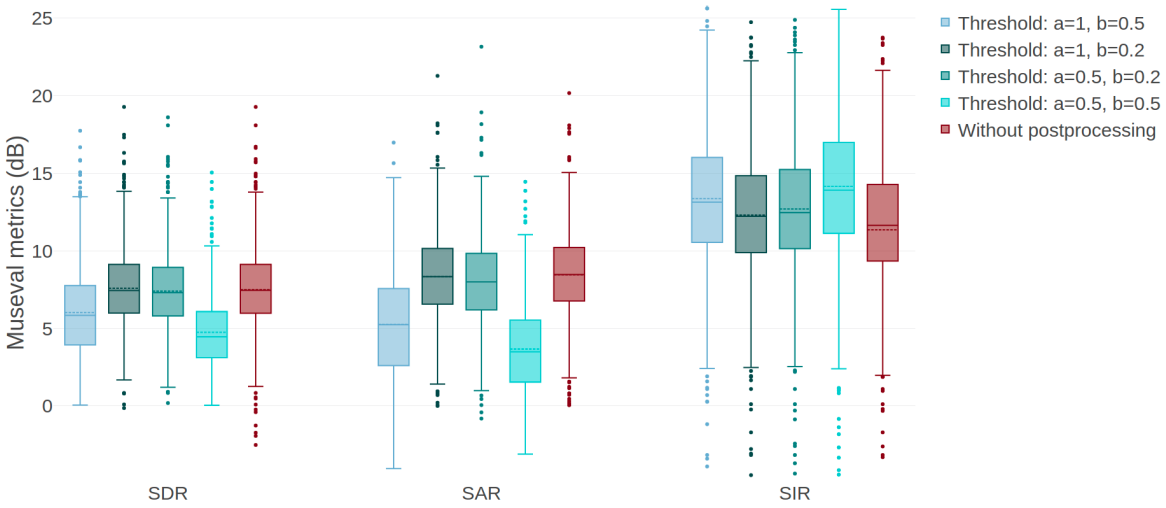


FIGURE 4.14: The U-Net trained on Bean with different energy ratio thresholds (vocals).

Mask binarization

The second post-processing method was thought to reduce interferences as well. It is applied on the estimated masks of the vocal and instrumental part: M_{vocals} and $M_{instrumentals}$. The idea is to force large values of the mask to 1, and small values to 0. This way, the separation algorithm has to distribute the spectrogram's energy in a more radical way, hopefully eliminating residuals of the other source.

To achieve this effect, we used a sigmoid-based, parametric function:

$$f_\alpha(x) = \frac{1}{1 + \exp(-(x - 0.5) * \alpha)}. \tag{4.8}$$

The parameter α tunes the slope of the function. The higher it is, the closer to a step function f_α will be, as we can see on Figure 4.15

Figures 4.17 and 4.16 show the effect of this post-processing for different values of α. Once again, the SIR clearly increases, but the other metrics tend to decrease. The higher α is, the more visible this effect is. We will thus keep the setting where $\alpha = 10$ as our best compromise.

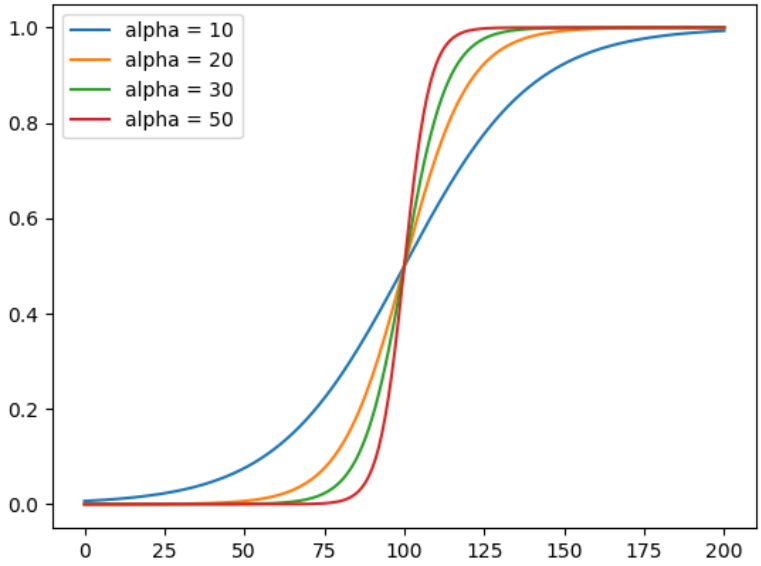


FIGURE 4.15: The binarization function used on the masks for post-processing.

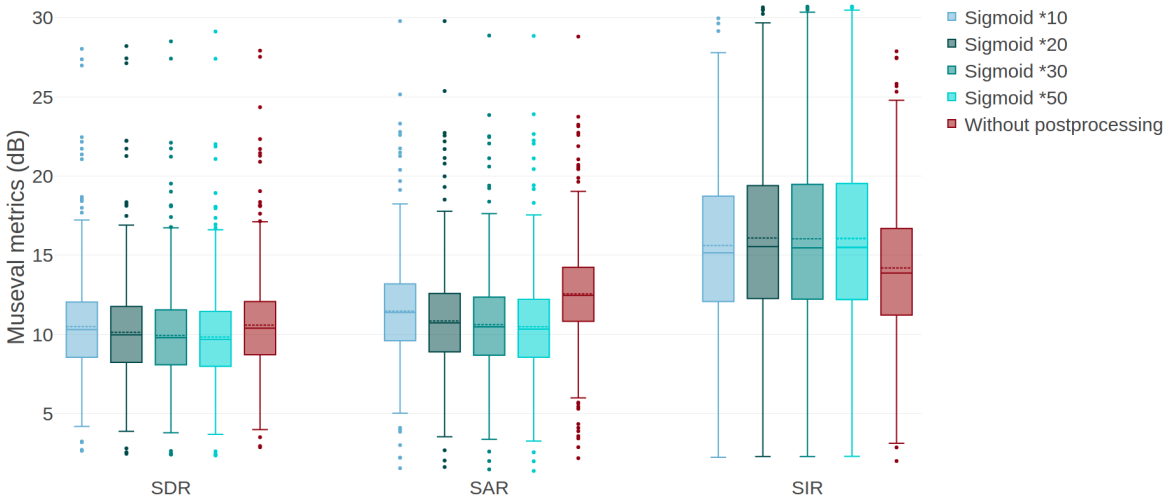


FIGURE 4.16: The U-Net trained on Bean with different parameterizations of the binarization function (instrumentals).

Multi-Channel Wiener filtering

This last post-processing method was first presented in Sivasankaran et al., 2015 and Nugraha, Liutkus, and Vincent, 2016. It was used in all DenseNet-based systems presented at SiSec 2018 (Takahashi and Mitsufuji, 2017). This post-processing step happens after masking and before reversing the STFT. It is applied independently to the complex spectrogram of the voice $\hat{S}_{vocals}$ and to the one of the instruments $\hat{S}_{instrumentals}$.

The idea is to re-estimate the coefficients of the complex spectrogram $\hat{S}$ by computing a spatial correlation matrix between both channels. First, a power spectral density $v(m, f)$ is computed for each time-frequency bin of the spectrogram. It is in fact the average of the

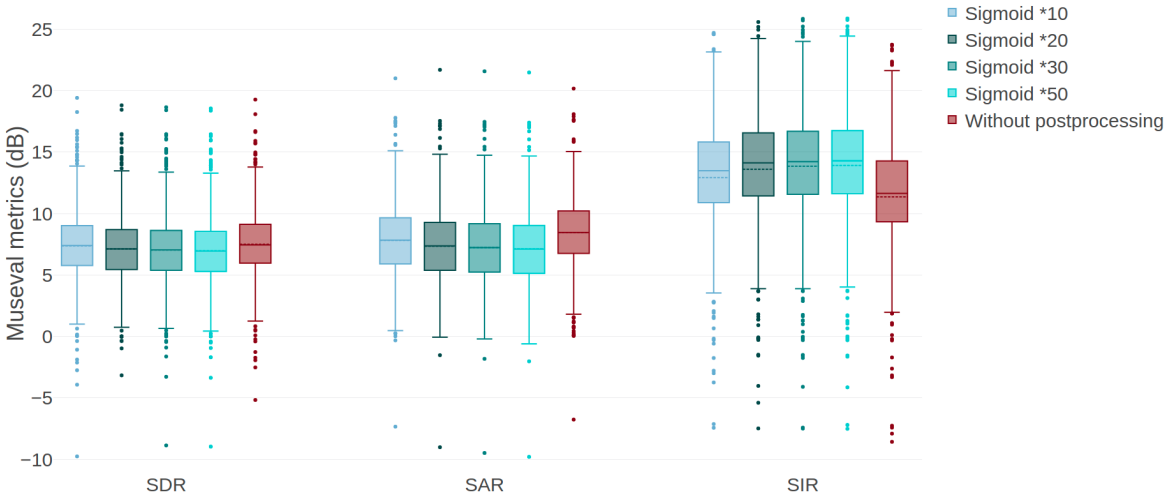


FIGURE 4.17: The U-Net trained on Bean with different parameterizations of the binarization function (vocals).

power of both channels: $v(m,n) = \frac{1}{2}||\hat{S}(m,n)||_2^2$. It is a scalar, while $\hat{S}(m,n)$ was a vector of length 2 (each time-frequency bin has two channels). Then, the spatial covariance matrix for this particular bin is estimated as:

$$R(n) = \frac{\sum_{m=1}^{M} \hat{S}(m,n)\hat{S}(m,n)^H}{\sum_{m=1}^{M} v(m,n)}. \tag{4.9}$$

where ^{H} denotes the conjugate transpose. $R(n)$ is thus a matrix of size (2,2). In this model, the spatial balance does not depend on the time m: the sources are assumed to be fixed in space. This hypothesis is reasonable for most recordings. Finally, the new estimate for $\hat{S}(m,f)$ is computed using the spectral power density and the spectral covariance matrix of both sources:

$$\hat{S}_{vocals}(m,f) = \frac{v_{vocals}(m,n)R_{vocals}(n)}{v_{vocals}(m,n)R_{vocals}(n) + v_{instrumentals}(m,n)R_{instrumentals}(n)} * S(m,f). \tag{4.10}$$

This operation resembles masking, except that it is performed directly on complex spectrograms. The implementation was done using SiSec's **MWF** oracle[4].

Figures 4.18 and 4.19 show the effect of the MWF post-processing, along with the best compromise found for the last two post-processings. Compared to the baseline (without post-processing), the MWF seems to have only a negative effect, or no effect at all. This is surprising given its success in the recent literature (Uhlich et al., 2017 ,Takahashi, Goswami, and Mitsufuji, 2018).

4.4.4 Experiment 4: different data augmentation methods

When training on a small dataset like MUSDB, it can be interesting to perform data augmentation (Uhlich et al., 2017). We inspired ourselves from Schlüter, 2017, in which the author uses a series of transformation on the spectrograms and tests the effect on a singing voice detection task. We set up a similar series of experiments to find if data augmentation was significant, and if yes, which one to use in priority. For singing voice separation, we adapted the transforms proposed by Schülter (pitch shifting, time stretching, loudness modification),

[4]https://github.com/sigsep/sigsep-mus-oracle/blob/master/MWF.py

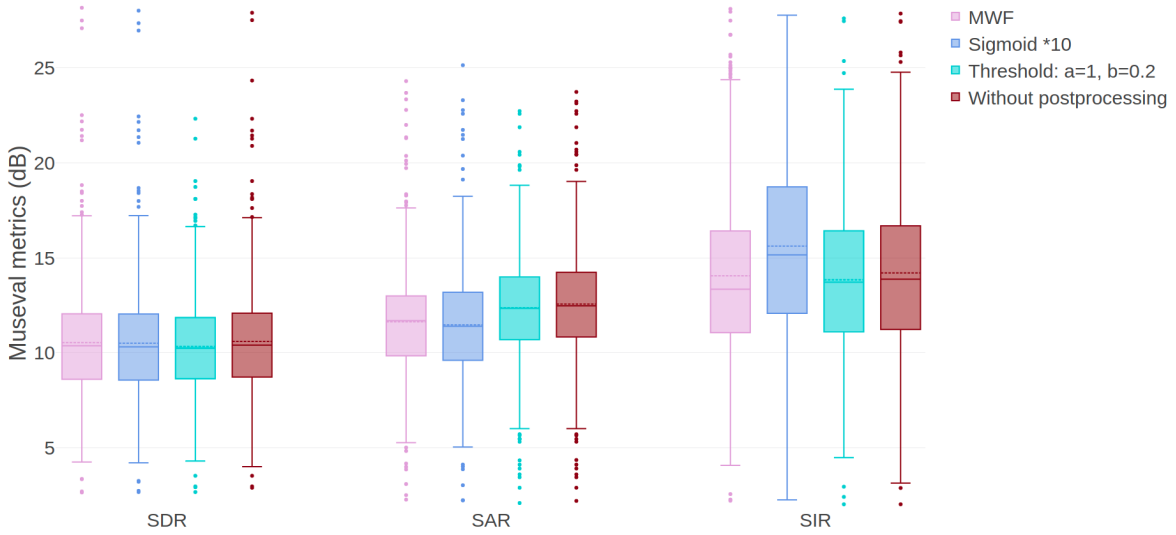


FIGURE 4.18: The U-Net trained on Bean with different post-processing methods (instrumentals).

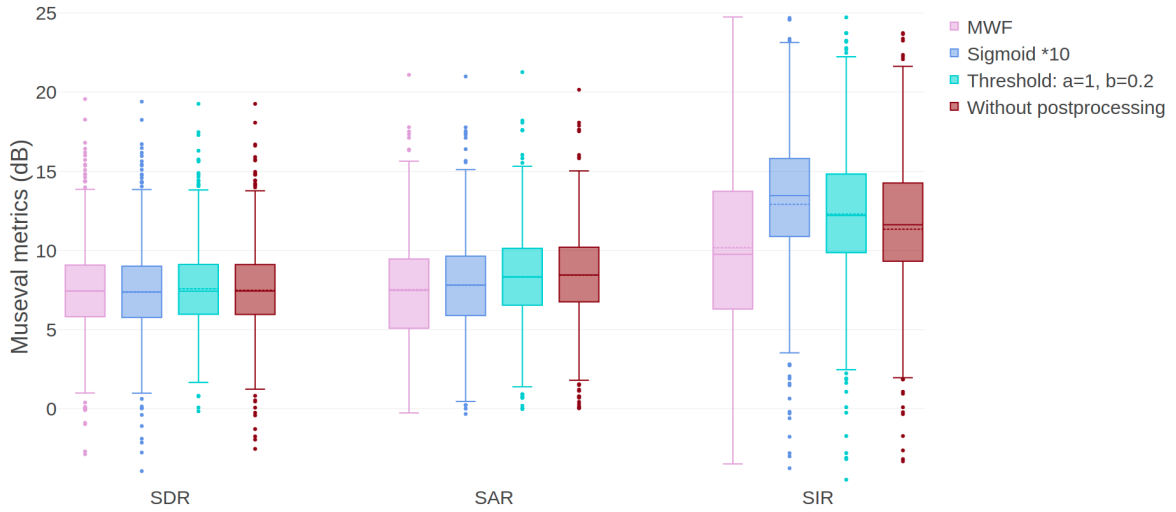


FIGURE 4.19: The U-Net trained on Bean with different post-processing methods (vocals).

and added one from Uhlich et al., 2017 (channels swapping). The specificity in separation is that both the target and the inputs must go through the exact same transformation for training. We did not implement the additive gaussian noise, neither the frequency filters, because we assumed they were not relevant in a musical context. Each transform, except the channel swapping, has a parameter β:

- **Channel swapping:** we exchange the left and right channels.
- **Time stretching:** we scale the spectrograms horizontally by a factor $\beta_{stretch}$ and keep the central part. Note that this is an approximation compared to an actual time stretching of the audio.
- **Pitch Shifting:** we scale the spectrograms vertically by a factor β_{shift} and keep the bottom part. Note that this is an approximation compared to an actual pitch shifting of the audio.

- **Loudness scaling:** we multiply all the coefficients of the spectrograms by a factor β_{scale}.

For each transform, we trained a U-Net on MUSDB Dev with a probability of 0.5 that the input and output spectrograms are transformed. Then, we randomly draw the parameter β from a certain range with a uniform probability. We tested two ranges for β: $[0.85, 1.15]$ ($\pm$ 15%) and $[0.7, 1.3]$ ($\pm$ 30%). We also implemented a "combined" transform, in which all the transforms have a probability of 0.3 to be applied, and the range for β is $[0.7, 1.3]$.

Figures 4.21 and 4.20 show the results of this experiment. No significant difference could be put forward by these new training modalities. The order of magnitude of the difference in the metrics is $\pm$ 0.2 dB, which is the same as the one we obtained in pilot experiments by changing the initializing seed of the convolutional layers. However, this is consistent with the values obtained in the literature by Uhlich et al., 2017. We are also reassured, because this shows that our initial system was not overfitting on the tempo and pitch of MUSDB.

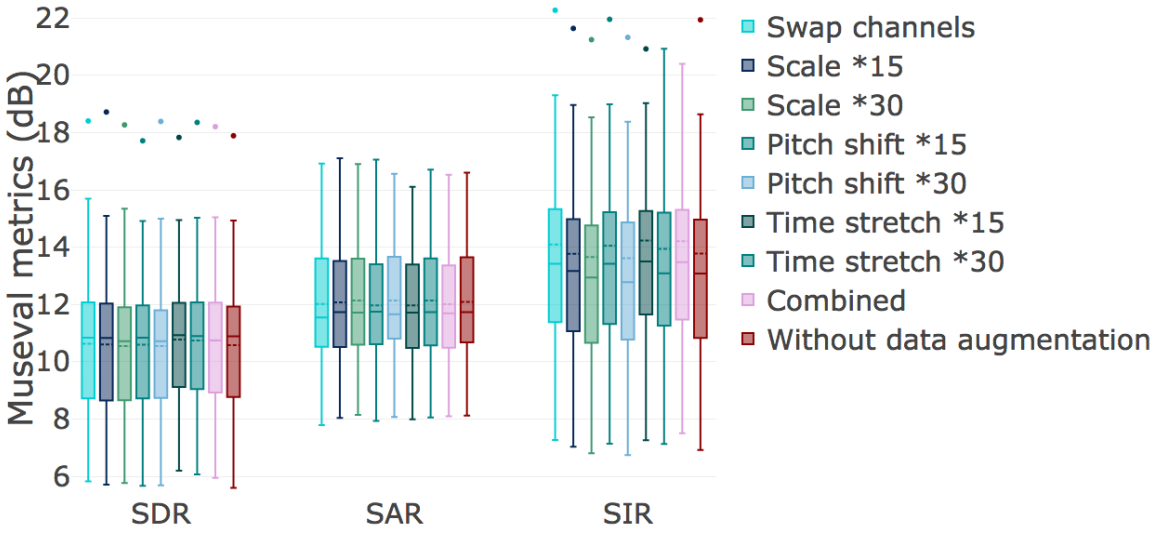


FIGURE 4.20: The U-Net trained on MUSDB with different augmentation methods (instrumentals).

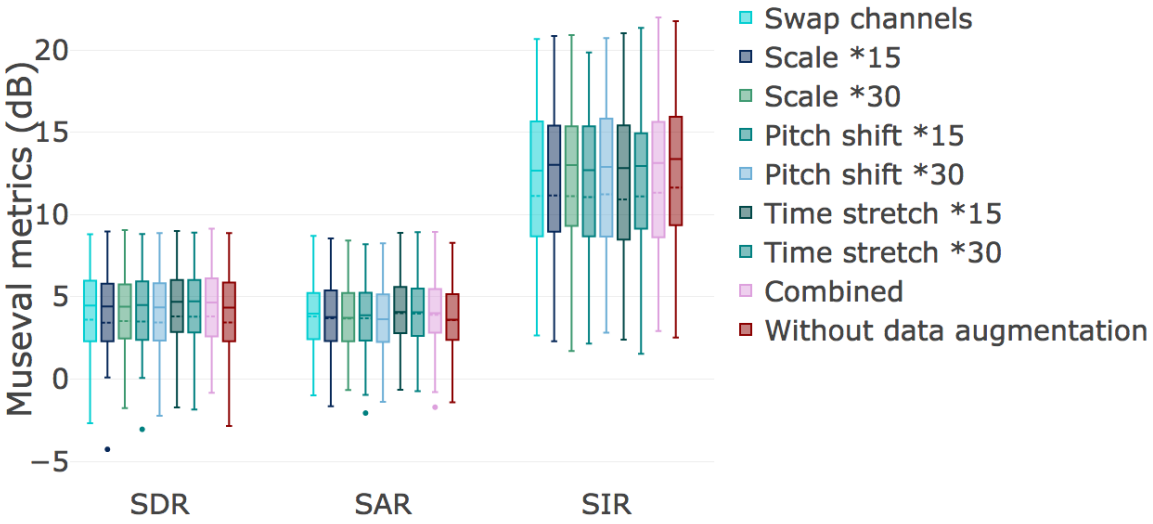


FIGURE 4.21: The U-Net trained on MUSDB with different augmentation methods (vocals).

Chapter 5

Discussions and future work

5.1 Discussion

In this internship, we studied different modalities on supervised singing voice separation systems.

We first re-implemented two state-of-the-art systems and trained them on datasets with various size and quality characteristics. We obtained good results with the U-Net, and more limited performances on the DenseNet and MMDenseNet. We believe that this is due to the significant memory requirement of the DenseNets, which made it impossible in our setup to build a very deep network and to train with large batches. But we could observe significant performance variations depending on the training dataset for each system.

We studied the impact of the training loss and concluded that depending on the final application of the separation, we would either recommend the L_1 loss or MSE for training. We tested different data augmentation methods on the spectrograms and found that the impact was probably not much larger than an ensembling of the same system with different seeds. The post-processing method, however, has a significant impact both on objective and perceptive measures. It allows one to re-balance the compromise between the interferences and the artifacts.

In conclusion, we have built a variety of singing voice separation algorithms, associated to a performance benchmark which provides directions to choose some parameters according to the desired application. We also created a demonstration script which separates any track of Deezer's catalog, using the U-Net trained on Bean. Anis Khlif ported it to a mobile application for demonstrating the viability of such a feature to Deezer's C-level.

5.2 Future work

We showed that it was possible to produce good estimates of the vocal and instrumental tracks of a song, provided it follows the scheme of standard western songs. While the instrumental estimates are often very convincing, the vocals, even if well isolated, do not sound always realistic. We believe that there is still room for improvement on this part of the problem. An idea would be to train the vocals-dedicated network to auto-encode a capella songs during the training. We could alternate training samples of a capella songs to be auto-encoded with training samples of mixtures to be separated. For this task, a complementary dataset of a capella songs would be easy to create using a music streaming service catalog. The use of Generative Adversarial Networks (GANs) is another way to enforce the realistic aspect of the vocals (Goodfellow et al., 2014). GANs have proven efficiency to produce realistic images in the field of computer vision (Isola et al., 2017). In this architecture, two neural network compete, one to produce realistic voice spectrograms from the mixture, and the other to discriminate the estimate from real voice spectrograms (Fan, Lai, and Jang, 2017).

During our experiments, we faced some dilemma about the design of the training and validation datasets. Should the dataset represent very wide variety of genres, with the risk of being less consistent (should black metal lyrics be considered as singing voice or not)? Or should we design the dataset as close as possible to the music that it will be evaluated on? Of course, the answer to this question depends on the final application of the separation algorithm. Further studies could be conducted by filtering the Catalog dataset by genre, for example.

Finally, the way to assess the quality of the separation must equally take into account the final application. While the objective metrics already provide good insights on the general quality of the algorithm, we noticed that they could not cover every aspect of our perception of the separation (Cano, FitzGerald, and Brandenburg, 2016). In particular, the fact the computation of the metrics do not depend on the frequency makes it likely to have audible artifacts in high frequencies, even with a system judged "good" by the objective evaluation. Concerning the interpretation of the metrics, a comprehensive study on the statistical relevance of the results could be conducted, as in Simpson et al., 2016.

Appendix A

Code snippets

A.1 An example of a score file produced by the Museval package for a short song.


```
{
  "targets": [
    {
      "frames": [
        {
          "duration": 1.0,
          "metrics": {
            "SIR": 17.08709,
            "ISR": 13.08037,
            "SDR": 8.72669,
            "SAR": 9.54819
          },
          "time": 0.0
        },
        {
          "duration": 1.0,
          "metrics": {
            "SIR": 9.29407,
            "ISR": 10.96202,
            "SDR": 6.68774,
            "SAR": 6.42729
          },
          "time": 1.0
        }
      ],
      "name": "vocals"
    },
    {
      "frames": [
        {
          "duration": 1.0,
          "metrics": {
            "SIR": 15.74727,
            "ISR": 19.81654,
            "SDR": 10.12923,
            "SAR": 11.34651
```

```
          },
          "time": 0.0
        },
        {
          "duration": 1.0,
          "metrics": {
            "SIR": 14.67816,
            "ISR": 16.85895,
            "SDR": 10.43156,
            "SAR": 12.52901
          },
          "time": 1.0
        }
      ],
      "name": "accompaniment"
    }
  ]
}
```

Appendix B

Details of network architectures

B.1 The transposed convolution used in the upsampling path of the networks.

On Figure B.1, we can see an illustration of the transpose convolutional layer used in the upsampling path of the U-Net, the DenseNet and the MMDenseNet. The idea is to apply a standard convolution to a feature maps that was first padded with zeros between each original value. In our case, the stride is 2, so the number of zeros between two successive coefficients is $stride - 1 = 1$.

An alternative to the padding is to linearly interpolate the feature map to double its size, then apply the convolution.

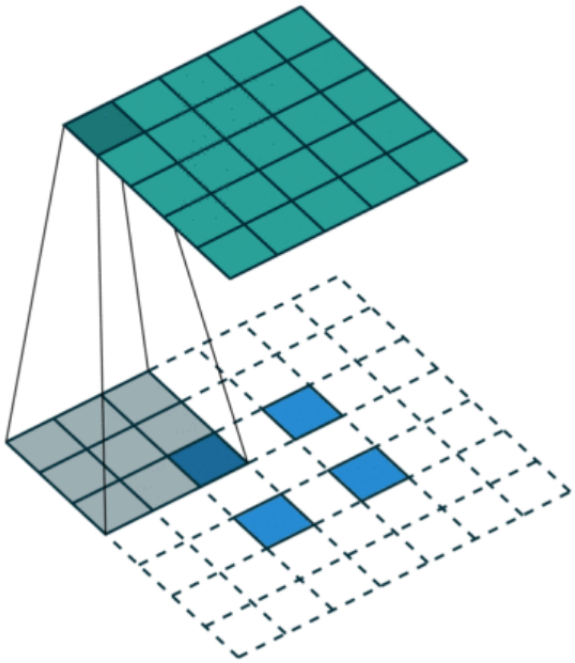


FIGURE B.1: Image from:
towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Bibliography

Arberet, Simon et al. (2010). "Nonnegative matrix factorization and spatial covariance model for under-determined reverberant audio source separation". In: *Information Sciences Signal Processing and their Applications (ISSPA), 2010 10th International Conference on*. IEEE, pp. 1–4.

Bittner, Rachel M et al. (2014). "MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research." In: *ISMIR*. Vol. 14, pp. 155–160.

Cano, Estefanía, Derry FitzGerald, and Karlheinz Brandenburg (2016). "Evaluation of quality of sound source separation algorithms: Human perception vs quantitative metrics". In: *Signal Processing Conference (EUSIPCO), 2016 24th European*. IEEE, pp. 1758–1762.

Chan, Tak-Shing T and Yi-Hsuan Yang (2017). "Informed Group-Sparse Representation for Singing Voice Separation". In: *IEEE Signal Processing Letters* 24.2, pp. 156–160.

Chandna, Pritish et al. (2017). "Monoaural audio source separation using deep convolutional neural networks". In: *International Conference on Latent Variable Analysis and Signal Separation*. Springer, pp. 258–266.

Dieleman, Sander and Benjamin Schrauwen (2014). "End-to-end learning for music audio". In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, pp. 6964–6968.

Dupin, Maxime et al. (2011). "Effects of the training dataset characteristics on the performance of nine species distribution models: application to Diabrotica virgifera virgifera". In: *PLoS One* 6.6, e20957.

Durrieu, Jean-Louis et al. (2009). "Main instrument separation from stereophonic audio signals using a source/filter model". In: *Signal Processing Conference, 2009 17th European*. IEEE, pp. 15–19.

Fan, Zhe-Cheng, Yen-Lin Lai, and Jyh-Shing Roger Jang (2017). "SVSGAN: Singing Voice Separation via Generative Adversarial Network". In: *arXiv preprint arXiv:1710.11428*.

Févotte, Cédric, Rémi Gribonval, and Emmanuel Vincent (2005). "BSS_EVAL toolbox user guide–Revision 2.0". In:

Gómez, Emilia et al. (2018). "Deep Learning for Singing Processing: Achievements, Challenges and Impact on Singers and Listeners". In: *arXiv preprint arXiv:1807.03046*.

Goodfellow, Ian et al. (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pp. 2672–2680.

Grais, Emad M and Mark D Plumbley (2017). "Single channel audio source separation using convolutional denoising autoencoders". In: *arXiv preprint arXiv:1703.08019*.

Hansen, Michael S et al. (2004). "On the influence of training data quality in k-t BLAST reconstruction". In: *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine* 52.5, pp. 1175–1183.

He, Kaiming et al. (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky (2012). "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on*, p. 14.

Huang, Gao et al. (2017). "Densely Connected Convolutional Networks." In: *CVPR*. Vol. 1. 2, p. 3.

Huang, Po-Sen et al. (2014a). "Deep learning for monaural speech separation". In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, pp. 1562–1566.

— (2014b). "Singing-Voice Separation from Monaural Recordings using Deep Recurrent Neural Networks." In: *ISMIR*, pp. 477–482.

Humphrey, Eric et al. (2017). "Mining labeled data from web-scale collections for vocal activity detection in music". In: *Proceedings of the 18th ISMIR Conference*.

Humphrey, Eric J, Juan Pablo Bello, and Yann LeCun (2012). "Moving Beyond Feature Design: Deep Architectures and Automatic Feature Learning in Music Informatics." In: *ISMIR*. Citeseer, pp. 403–408.

Isola, Phillip et al. (2017). "Image-to-image translation with conditional adversarial networks". In: *arXiv preprint*.

Jansson, Andreas et al. (2017). "Singing voice separation with deep U-Net convolutional networks". In: *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pp. 323–332.

Japkowicz, Nathalie and Shaju Stephen (2002). "The class imbalance problem: A systematic study". In: *Intelligent data analysis* 6.5, pp. 429–449.

Jégou, Simon et al. (2017). "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE, pp. 1175–1183.

Klatt, Dennis H and Laura C Klatt (1990). "Analysis, synthesis, and perception of voice quality variations among female and male talkers". In: *the Journal of the Acoustical Society of America* 87.2, pp. 820–857.

Kruspe, Anna M and IDMT Fraunhofer (2014). "Keyword Spotting in A-capella Singing." In: *ISMIR*. Vol. 14, pp. 271–276.

Li, Yipeng and DeLiang Wang (2005). *Separation of singing voice from music accompaniment for monaural recordings*. Tech. rep. Ohio State University Columbus United States.

Liutkus, Antoine et al. (2017). "The 2016 Signal Separation Evaluation Campaign". In: *Latent Variable Analysis and Signal Separation - 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*. Ed. by Petr Tichavský et al. Cham: Springer International Publishing, pp. 323–332.

McFee, Brian et al. (2015). "librosa: Audio and music signal analysis in python". In: *Proceedings of the 14th python in science conference*, pp. 18–25.

Mesaros, Annamaria (2012). "Singing voice recognition for music information retrieval". In: *Tampereen teknillinen yliopisto. Julkaisu-Tampere University of Technology. Publication; 1064*.

Mesaros, Annamaria and Tuomas Virtanen (2010). "Automatic recognition of lyrics in singing". In: *EURASIP Journal on Audio, Speech, and Music Processing* 2010, p. 4.

Mesaros, Annamaria, Tuomas Virtanen, and Anssi Klapuri (2007). "Singer Identification in Polyphonic Music Using Vocal Separation and Pattern Recognition Methods." In: *ISMIR*, pp. 375–378.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Nugraha, Aditya Arie, Antoine Liutkus, and Emmanuel Vincent (2016). "Multichannel music separation with deep neural networks". In: *Signal Processing Conference (EUSIPCO), 2016 24th European*. IEEE, pp. 1748–1752.

Orio, Nicola et al. (2006). "Music retrieval: A tutorial and review". In: *Foundations and Trends® in Information Retrieval* 1.1, pp. 1–90.

Ozerov, Alexey and Cédric Févotte (2010). "Multichannel nonnegative matrix factorization in convolutive mixtures for audio source separation". In: *IEEE Transactions on Audio, Speech, and Language Processing* 18.3, pp. 550–563.

Ozerov, Alexey et al. (2007). "Adaptation of Bayesian models for single-channel source separation and its application to voice/music separation in popular songs". In: *IEEE Transactions on Audio, Speech, and Language Processing* 15.5, pp. 1564–1578.

Pollastri, Emanuele (2002). "A pitch tracking system dedicated to process singing voice for music retrieval". In: *Multimedia and Expo, 2002. ICME'02. Proceedings. 2002 IEEE International Conference on*. Vol. 1. IEEE, pp. 341–344.

Rafii, Zafar and Bryan Pardo (2013). "Repeating pattern extraction technique (REPET): A simple method for music/voice separation". In: *IEEE transactions on audio, speech, and language processing* 21.1, pp. 73–84.

Rafii, Zafar et al. (2017). *The MUSDB18 corpus for music separation*. DOI: 10.5281/zenodo.1117372. URL: https://doi.org/10.5281/zenodo.1117372.

Raj, Bhiksha et al. (2007). "Separating a foreground singer from background music". In: *Proc. Int. Symp. Frontiers Res. Speech Music*, pp. 8–9.

Royo-Letelier (2015). *Detection and characterization of singing voice using deep neural networks*.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Ryynanen, Matti et al. (2008). "Accompaniment separation and karaoke application based on automatic melody transcription". In: *Multimedia and Expo, 2008 IEEE International Conference on*. IEEE, pp. 1417–1420.

Schlüter, Jan (2017). "Deep Learning for Event Detection, Sequence Labelling and Similarity Estimation in Music Signals". Chapter 9. PhD thesis. Austria: Johannes Kepler University Linz.

Schmidt, Mikkel N and Rasmus K Olsson (2006). "Single-channel speech separation using sparse non-negative matrix factorization". In: *Ninth International Conference on Spoken Language Processing*.

Simpson, Andrew JR et al. (2016). "Evaluation of audio source separation models using hypothesis-driven non-parametric statistical methods". In: *Signal Processing Conference (EUSIPCO), 2016 24th European*. IEEE, pp. 1763–1767.

Sivasankaran, Sunit et al. (2015). "Robust ASR using neural network based speech enhancement and feature simulation". In: *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*. IEEE, pp. 482–489.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Stoller, Daniel, Sebastian Ewert, and Simon Dixon (2018). "Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation". In: *arXiv preprint arXiv:1806.03185*.

Stöter, Fabian-Robert, Antoine Liutkus, and Nobutaka Ito (2018). "The 2018 Signal Separation Evaluation Campaign". In: *International Conference on Latent Variable Analysis and Signal Separation*. Springer, pp. 293–305.

Takahashi, Naoya, Nabarun Goswami, and Yuki Mitsufuji (2018). "MMDenseLSTM: An efficient combination of convolutional and recurrent neural networks for audio source separation". In: *arXiv preprint arXiv:1805.02410*.

Takahashi, Naoya and Yuki Mitsufuji (2017). "Multi-Scale multi-band densenets for audio source separation". In: *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2017 IEEE Workshop on*. IEEE, pp. 21–25.

Torralba, Antonio and Alexei A Efros (2011). "Unbiased look at dataset bias". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, pp. 1521–1528.

Uhlich, Stefan et al. (2017). "Improving music source separation based on deep neural networks through data augmentation and network blending". In: *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, pp. 261–265.

Vembu, Shankar and Stephan Baumann (2005). "Separation of Vocals from Polyphonic Audio Recordings." In: *ISMIR*. Citeseer, pp. 337–344.

Virtanen, Tuomas, Annamaria Mesaros, and Matti Ryynänen (2008). "Combining pitch-based inference and non-negative spectrogram factorization in separating vocals from polyphonic music." In: *SAPA@ INTERSPEECH*, pp. 17–22.

Wang, Avery et al. (2003). "An Industrial Strength Audio Search Algorithm." In: *Ismir*. Vol. 2003. Washington, DC, pp. 7–13.

Weninger, Felix et al. (2014). "Discriminative NMF and its application to single-channel source separation". In: *Fifteenth Annual Conference of the International Speech Communication Association*.