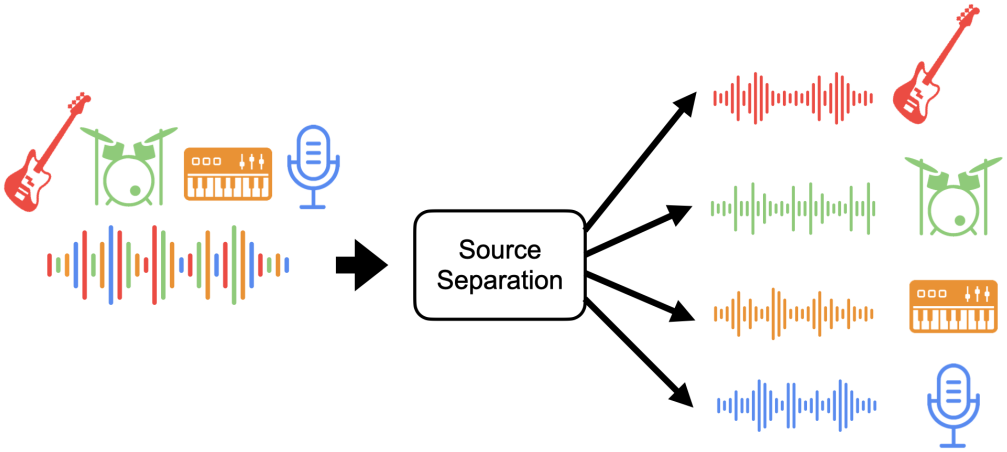
Master Thesis : Music Source Separation with Neural Networks

Auteur : Laurent, Martin
Promoteur(s) : Louppe, Gilles
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/18349

LIÈGE université
Sciences Appliquées

Music Source Separation with Neural Networks



Master thesis submitted in partial fulfillment of the requirements for the degree of Master of Data Science and Engineering

Supervisor:
Professor Gilles Louppe

Author:
Martin Laurent

Academic year 2022-2023

Abstract

Music source separation (MSS) involves extracting the individual instrument's audio sources from a mixed music signal and is known to be a very challenging problem with a long history of scientific activity. In the last decade, the field has known significant progress thanks to the emergence of Deep Learning, with deep neural network architectures achieving state-of-the-art performances. In this master thesis we explore how Deep Learning is applied to the problem of MSS and examine some of the best models to date. We then attempted to reproduce one of those models which applies the traditional Short Time Fourier Transform to the input signal. We finally explore a potential of an alternative transform, the Short Time Discrete Cosine Transform, for this architecture which brings the possibility of achieving the same results while reducing drastically the complexity and training time of the model.

Acknowledgement

First and foremost I would like to thank my supervisor, Professor Gilles Louppe, for his valuable guidance throughout this work. I am grateful for his availability and the monthly meetings without which I could not have finished this work.

I would also want to express my gratitude to Arnaud Delaunoy for answering my numerous questions and helping me solve many problems. Thank you to my friends Lucas Backes and Loïc Ongena for your advises and feedback regarding my report.

Finally, I would like to thank my family, my friends as well as my partner Aline for supporting and comforting me during this year.

Contents

Chapter 1

Introduction

1.1 Brief History

Music Source Separation (MSS) is a sub-field of the broader Sound Source Separation (SSS) problem. SSS algorithms try to simulate the ability of humans to differentiate individual sound sources within a mixture. Similarly MSS algorithms seek to deconstruct a music audio clip into its constituent sources (stems), unveiling the distinct contributions of each instrument or vocal track. MSS enables us to extract, modify, and analyze specific musical sources by untangling this complex web of sound, which has important applications in audio signal processing, music production, and audio analysis. Obtaining such separated stems can also be useful to re-mix, suppress or up-mix sources from a final recording mix as most often the original stems are not publicly available.

Typically MSS is more challenging than the general source separation problem due to several factors [1]. A 2-channel recording typically includes many musical instruments and vocals, resulting in a complicated blend of sources. Furthermore, these sources have often gone through several levels of processing during recording and mixing, including the use of filters, reverberation, and possibly nonlinear effects. This complicates the separating process even further. In some cases the mixture is time-varying with moving sources or changes in the production parameters.

Over the years, research in Music Source Separation has primarily focused around the application of model-based estimation methods. However, there has lately been a major increase in the use of Deep Neural Networks (DNNs) for MSS, reflecting their growing popularity in the field [1].

1.2 Problem Statement

Let $\mathbf{x}(n) \in \mathbb{R}^2$ denote the stereo mixture in the time domain which in the context of this thesis is known to be composed of the four sources *bass* $\mathbf{s}_B(n)$, *drums* $\mathbf{s}_D(n)$, *other* $\mathbf{s}_O(n)$ and *vocals* $\mathbf{s}_V(n)$ such that

$$\mathbf{x}(n) = \mathbf{s}_B(n) + \mathbf{s}_D(n) + \mathbf{s}_O(n) + \mathbf{s}_V(n) = \sum_{i \in \mathcal{I}} \mathbf{s}_i(n).$$

with $\mathcal{I} := \{B, D, O, V\}$ and $\mathbf{s}_i(n) \in \mathbb{R}^2$, $\forall i \in \mathcal{I}$.

The goal of MSS is then to retrieve stereo source estimates $\hat{\mathbf{s}}_i(n) \in \mathbb{R}^2$ that resemble as closely as possible the true sources $\mathbf{s}_i(n)$.

1.3 Applications of Music Source Separation

One of the prominent applications of MSS lies in music production and remixing. With this technology, producers and DJs can isolate individual instruments or vocals from a mixed audio track, opening up the possibility for rearranging, remixing, or enhancing specific elements of the music.

Moreover, music transcription and analysis can greatly benefit from MSS techniques. By isolating individual instruments or vocals, it is possible to distinguish more clearly the notes and characteristics from a specific instrument which could be harder to grasp from the original mixture. Thus MSS can facilitate the process of producing the instrument sheets for a piece of music. Furthermore MSS can also be coupled to automatic music transcription algorithms to obtain the music sheets of any instrument directly from an input mixture.

In the realm of entertainment, MSS can have a significant impact on karaoke systems and singing applications. By removing or lowering the original vocal track, users can sing along to their favorite songs or perform karaoke without the lead vocals, leading to an enhanced karaoke experience and enabling vocal training applications.

Additionally, music restoration and remastering may find use in MSS. By isolating specific instruments or vocals, these techniques help in restoring and improving the quality of old or degraded recordings.

Finally, platforms providing music streaming services could make use of MSS to dynamically respond to the users' preferences. If a user wants to specifically reduce the volume of the drums for example, MSS techniques could allow these kind of features. By separating and manipulating different sources, these systems provide personalized and immersive music experiences.

1.4 Objectives and Organization

The first objective of this Master's Thesis is to reproduce as closely as possible one of the currently best performing models for MSS, namely Band-Split RNN[1] (Luo et al., 2022) [2], as the official implementation has not been made public. Their innovative idea to split the spectrogram into frequency subbands to enhance performances has proved to be very beneficial and it seems very insightful to do so, indeed all the instruments do not exhibit energy in the same range of frequencies or even in the same manner. Allocating resources to perform finer separation where needed intuitively should increase performances.

The secondary objective is to adapt BSRNN to a change in pre-processing method and evaluate the impact. In 2022, Sgouros et al. proposed an alternative pre-processing method to the traditional Short-Time Fourier Transform (STFT) for MSS called Short-Time Discrete Cosine Transform (STDCT) [3]. This alternative avoids phase recovery problem faced by models

[1]The details of the model are explained in a later section.

operating on STFT magnitude spectrograms and proved to bring enhanced performances on their model compared to when it has been trained on the former method.

1.5 Thesis Outline

This thesis composed of seven chapters:

The first chapter being this introduction, it is followed by the second chapter introduces the fundamentals of MSS. Providing an overview of audio signal processing, the different characteristics expected in music signals and techniques, DNN and non DNN based, used to perform MSS.

After that, the third chapter covers some of the most influential deep learning architectures used to separate audio sources in music signals.

The fourth chapter introduces the main dataset used commonly across every MSS models as well as the pre-processing and a data simulation techniques employed in this work.

In the next chapter, the experimental set up and methodology are detailed. Including the hyperparameters configuration of the various models, the evaluation metrics and finally, the evaluation process.

Lastly, chapter six and seven respectively contain a thorough study and analysis of the results and the conclusions for this work along with the fields of improvement of the experiments.

Chapter 2

Fundamentals of Music Source Separation

2.1 Audio Signal Processing

2.1.1 Waveform Signal

A waveform refers to a graphical representation depicting the shape and characteristics of a signal as it propagates through a medium, be it gaseous, liquid, or solid. Specifically for sound, a waveform represents the pattern of variation in sound pressure (or amplitude) over time. In informal contexts, the term ”waveform audio” is often used to refer to the recorded sound itself [4].

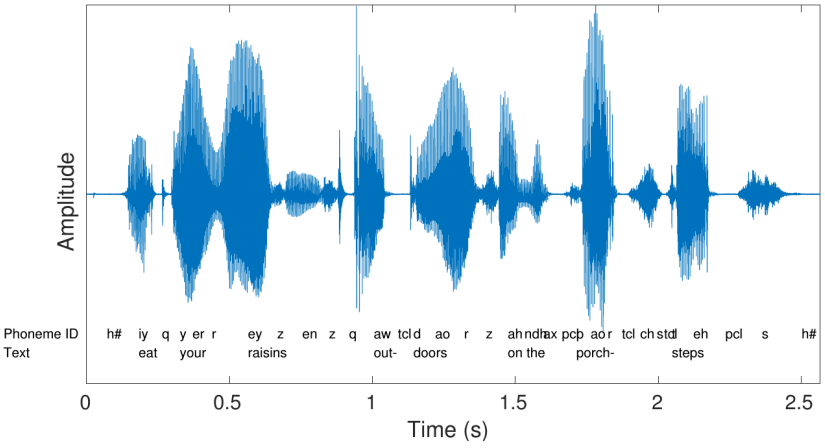


Figure 2.1: Waveform of an audio signal. Image from [5].

It involves converting analog audio signals, which are continuous in nature, into discrete digital representations that can be stored, transmitted, and processed by computers or other digital systems.

2.1.2 Analog to Digital Conversion (ADC)

The process of ADC consists of two parts, the Sampling and the Quantization.

Sampling

Sampling is the process of turning a continuous-time signal to a discrete-time signal by breaking it down into a sequence of individual samples. Each sample indicates the signal's amplitude at a certain point in time.

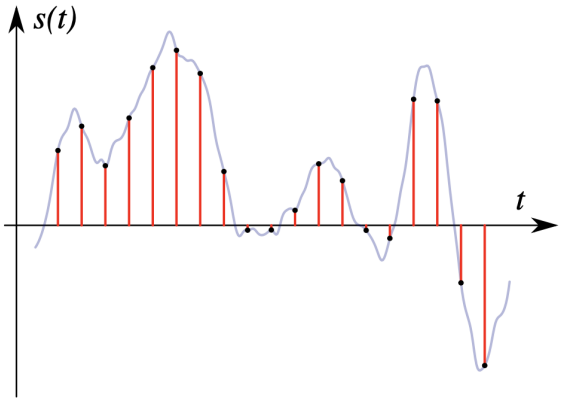


Figure 2.2: Visualization of the sampling process. Image from [6].

Let $s(t)$ be a continuous signal to be sampled, and let sampling occur by measuring the value of the continuous signal every T seconds, which is referred to as the sampling period. Then the sampled function is given by the sequence: $s(nT)$ with integer values of n.

From this sampling period we derive the very important concept of sampling rate (or sampling frequency), $f_s = \frac{1}{T}$, which is the number of samples obtained in one second. In general the sampling rate for an audio signal is 44.1kHz, this is due to the Nyquist-Shannon sampling theorem [7]:

Theorem 2.1.1 (Nyquist-Shannon sampling theorem)**.** *If a function $s(t)$ contains no frequencies higher than B hertz, then it can be completely determined from its ordinates at a sequence of points spaced less than $\frac{1}{2B}$ seconds apart.*

The Nyquist frequency, denoted f_N, is the maximum frequency that can be accurately represented or reconstructed from a discrete-time signal [8]. For a signal sampled at $f_s = 44.1$kHz, the Nyquist frequency can be determined as $f_N = \frac{f_s}{2} = 22.05$kHz. Given that the human ear perceives audible frequencies within a range of approximately 20Hz to 20kHz, it follows that a sample rate of 44.1kHz is suitable for faithful representation of signals falling within the audible frequency spectrum.

Quantization

Digital signal representation necessitates the use of a finite number of discrete levels to describe the signal's amplitude. Consequently, the process of quantization becomes imperative [5]. It consists in discretizing the amplitudes of the samples (obtained through the sampling process) by mapping their continuous values onto a finite set of available levels. The process is depicted in Figure 2.3.

By applying quantization one inevitably creates what is called "quantization error" which decreases when the resolution increases. The resolution of quantization refers to the smallest distinguishable increment or step size between adjacent quantization levels. It is typically determined by the number of bits allocated for representing each level.
For example, a CD has a 16 bits resolution, indicating its capacity to represent $2^{16} = 65536$ distinct amplitude values.

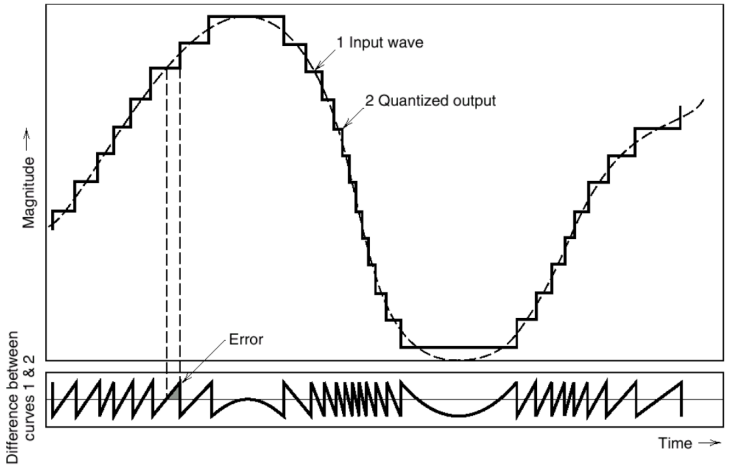


Figure 2.3: Visualization of the quantization process. Image from [9].

2.1.3 Transforms

The time domain representation of an audio signal does not fully capture all the information inherent to the signal. This is because the characteristics of sound are not solely defined by time, but also by frequency, which is a crucial descriptor of sound. This frequency information is not encompassed within a time-domain representation of the signal. Therefore, the need arises to convert the time-domain representation of the signal into a frequency-domain representation.

Fourier Transform

The Fourier Transform (FT) is a mathematical technique used to analyze and transform a function or signal from the time domain to the frequency domain. It relies on a a fundamental fact [10]: "*All waveforms, no matter what you scribble or observe in the universe, are actually just the sum of simple sinusoids of different frequencies.*".

The FT expresses a signal as a sum of complex sinusoidal functions also known as pure tones. Each of them represents a specific frequency present in the signal and is characterized by its amplitude and phase. These pure tones are depicted in Figure 2.4.
Given a continuous-time signal $s(t)$, $s : \mathbb{R} \to \mathbb{R}$, its (continuous) Fourier Transform $\hat{s}(f)$, $\hat{s} : \mathbb{R} \to \mathbb{C}$, is defined as:

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t) e^{-i2\pi ft} dt$$

$\hat{s}(f)$ is the frequency-domain representation of the signal $s(t)$, providing information about the amplitude and phase of each frequency component present.
The inverse Fourier Transform is defined as:

$$s(t) = \int_{-\infty}^{\infty} \hat{s}(f) e^{i2\pi ft} df$$

which gives back the original signal $s(t)$.
However when working with a signal on a computer, this signal is not continuous but discrete due to the ADC, more specifically the sampling process, explained in section 3.1.2. We then define the discrete Fourier Transform (DFT).

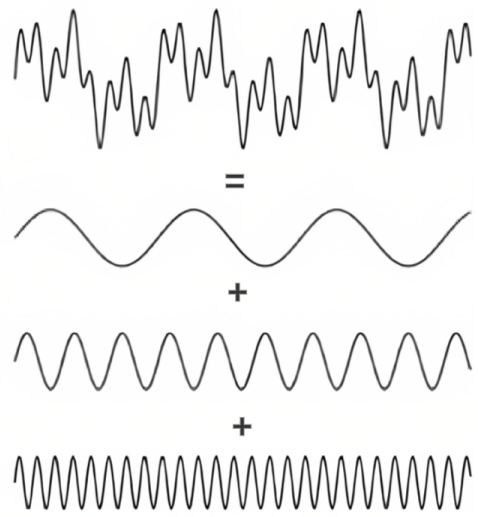


Figure 2.4: Pure tones component of a signal. Image from [11].

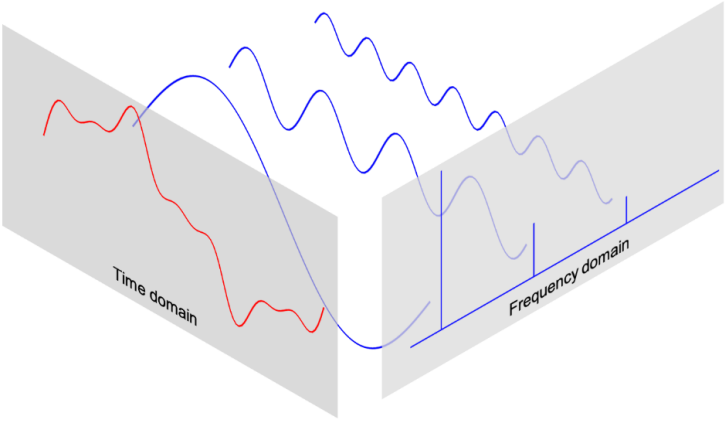


Figure 2.5: Visualization of the Fourier Transform. Image from [12].

Given a sequence of N samples $(s[0], s[1], ..., s[N-1])$, the DFT provides a sequence of N complex numbers $(\hat{s}[0], \hat{s}[1], ..., \hat{s}[N-1])$ defined as [13]:

$$\hat{s}[k] = \sum_{n=0}^{N-1} s[n] e^{-\frac{i2\pi}{N} kn}$$

The frequency $F(k)$ corresponding to a specific value of k is $F(k) = \frac{k}{NT_r} = \frac{kf_r}{N}$ where T_r is the sampling period and f_r is the sampling rate. Hence the frequencies evaluated always belong to the range $[0\ ,\ f_r]$.
Finally the inverse discrete Fourier Transform is defined as:

$$s[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{s}[k] e^{\frac{i2\pi}{N} kn}$$

The FT is an effective technique for analyzing the frequency components present in a signal. However, it has a limitation in that it provides a static representation of the frequency content for the entire duration of the signal. In many cases, signals exhibit temporal variations, such as in music where the nature of the sound implies multiple changes over time. Therefore,

there is a need to represent the frequency components and their time-varying behavior in a more comprehensive manner by utilizing a time-frequency domain representation.

Short-Time Fourier Transform & Time-Frequency Spectrogram

The Short-Time Fourier Transform (STFT) is a mathematical technique that aims to represent a signal by dividing it into overlapping segments, performing the Fourier Transform on each segment, and combining the resulting frequency components into a unified representation.

To accomplish this, the STFT divides the signal into smaller segments or frames by using a sliding window function of fixed duration N_{fr}, called the frame size or window length. The Hann window, defined below, is commonly employed for this purpose.

$$w[n] = \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{L}\right)\right), \quad 0 \leq n \leq N_{fr}$$

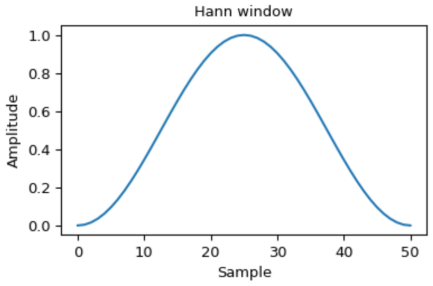
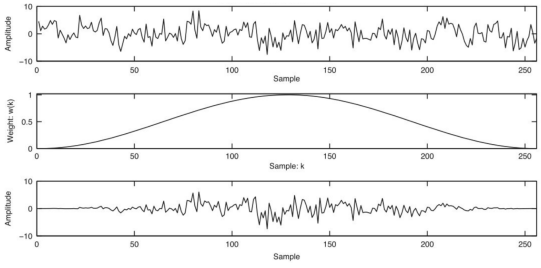
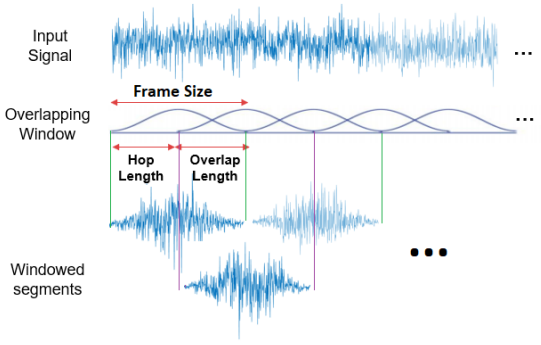


Figure 2.6: Hann window graph. Image from [14].

This shape is specifically designed to flatten and attenuate the amplitude of the signal segment near the edges, as can be observed in Figure 2.7 (a), thereby minimizing spectral leakage effects and preserving the integrity of the analyzed signal. This so-called spectral leakage is a phenomenon occurring when the discontinuities at the edges of the unwindowed segment create unwanted frequency artifacts in the FT [15]. To prevent the loss of information associated with the flattened sections of signal segments, it is common to employ overlap in the segmentation. Rather than sliding the window over the signal by an amount equal to its length, a smaller displacement referred to as the **hop length** or hop size is utilized, this can be observed in Figure 2.7 (b).



(a) Windowing operation. Image from [16].



(b) Overlapping windows. Image from [17].

Figure 2.7: Segmenting the signal.

Each frame is then subjected to the DFT individually. By applying the DFT to these shorter segments, we can capture the frequency content of the signal over time and observe any

variations or changes that occur. These frame-specific frequency content are stacked to create the STFT of the signal.

Let m designate the m^{th} frame of the signal and k the k^{th} discrete frequency evaluated by the transform. The Fourier coefficient for a tuple (m, k) is given by:

$$S[m,k] = \sum_{n=0}^{N_{fr}-1} x[n+mH]w[n]e^{-i2\pi n\frac{k}{N_{fr}}}$$

where N_{fr} refers to the frame size and H to the hop size.

The result of STFT is a complex matrix $\mathbf{S} \in \mathbb{C}^{T\times F}$ where $T = \frac{N_{fr}}{2}+1$ is the number of frames and $F = \frac{N_x - N_{fr}}{H} + 1$, N_x being the number of samples in the entire signal, is the number of frequency bins.

Finally, the magnitude squared of the STFT yields the spectrogram[1] representation of the power spectral density of the signal:

$$\text{spectrogram}\{x(t)\}[m,k] = |X[m,k]|^2$$

The overall process of STFT is depicted in Figure 2.8.

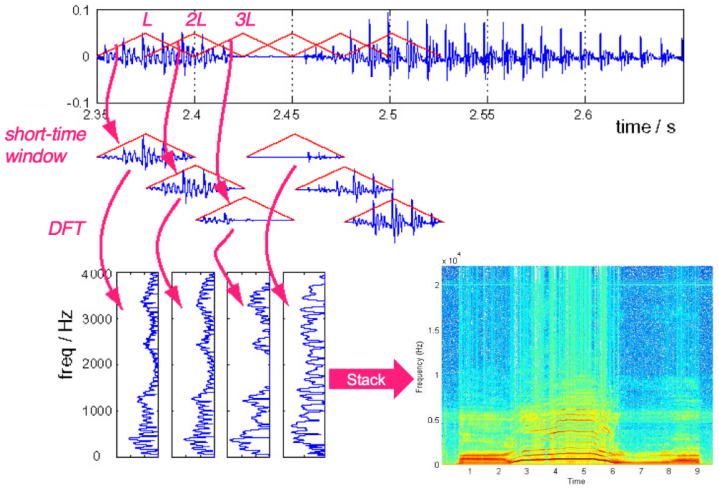


Figure 2.8: Illustration of the STFT process. Image from [18].

The inverse STFT is given by:

$$x[n+mH] = \frac{1}{N_{fr}w[n]} \sum_{k=0}^{N_{fr}-1} S[m,k]e^{i2\pi n\frac{k}{N_{fr}}}$$

Short-Time Discrete Cosine Transform

The spectrogram resulting from the STFT consists of complex-valued coefficients, which can be computationally expensive for neural networks to handle. Hence, it is common practice

[1]When working with the complex matrix $\mathbf{S}$, it is common to refer to it as the spectrogram despite it not being the magnitude squared of the STFT coefficients but the STFT coefficients themselves.

to utilize the magnitude of the spectrogram as a representation for processing and separation tasks[2]. After separating the sources, the inverse STFT is applied to transform the separated outputs back to the time domain. However, this process requires knowledge of the phase information for each separated source, which is typically unknown since only the magnitude was estimated during the separation.

To estimate the phase of each individual source, many approaches have been used, allowing signal inversion back to the time domain. These strategies, however, are complex and computationally demanding, and they may not consistently produce considerable improvements in outcomes.

To address these challenges, Sgouros et al. (2022)[3] proposed an alternative to the STFT known as the Short-Time Discrete Cosine Transform (STDCT). The STDCT follows a similar mechanism as the STFT but employs the Discrete Cosine Transform (DCT) instead of the Fourier Transform. Each frame of the signal is subjected to windowing, and a 1D-DCT is applied to each window. By utilizing the STDCT, an alternative time-frequency representation can be obtained for further analysis and processing of the signal. Assuming a signal frame $s[n]$ of length N_{fr}, it can be expressed as follows:

$$S[k] = \sqrt{\frac{2}{N_{fr}}}\beta[k] \sum_{n=0}^{N_{fr}-1} s[n] \cos\Big[\frac{\pi k(2n+1)}{2N_{fr}}\Big], \ \forall k \in [0, N_{fr}-1]$$

$$\beta[k] = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & k = 1, ..., N_{fr}-1 \end{cases}$$

This feature simplifies the separation operation by removing the need for additional processing steps. As a result, the real-valued DCT "spectrogram," with no further alterations or information loss, can be used immediately for training the separation network.

Furthermore, because no phase recovery post-processing is required, the real-valued DCT "spectrogram" source estimate can be inverted directly to the time domain. Using the 1D-iDCT, each column is translated to the time domain. DCT type-III is used for the inverse transformation, which is defined as follows:

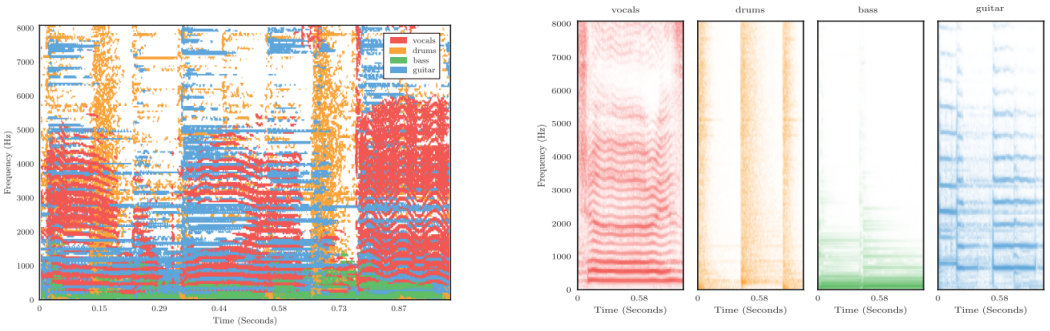
$$s[n] = \sqrt{\frac{2}{N_{fr}}} \sum_{k=0}^{N_{fr}-1} \beta[k]S[k] \cos\Big[\frac{\pi k(2n+1)}{2N_{fr}}\Big], \ \forall n \in [0, N_{fr}-1]$$

where $\beta[k]$ is evaluated as before.

2.2 Characteristics of Music Signals

Music signals have particular features that separate them from other audio signals such as voice or environmental noises, making them an important consideration in the design of MSS techniques. As a result, a thorough comprehension of these distinguishing characteristics is required.

[2]Note that this is not the case for the main model of this project, namely BSRNN. It will be explained in a later section.



(a) Music mixture (b) Separated sources

Figure 2.9: Representations in the time-frequency domain. The dominant musical source in each time-frequency bin is displayed with a different color. Images from [1].

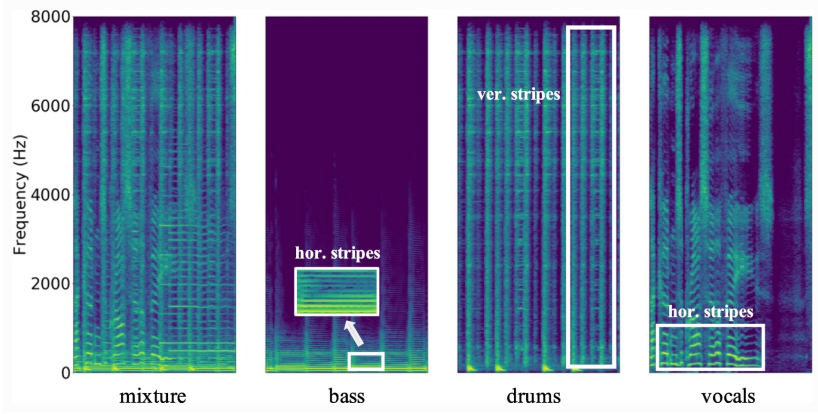


Figure 2.10: Spectrograms of an example music piece and its source signals. Image from [19].

Musical sources can be classified as harmonic, percussive, or vocals [1]. The spectrogram of harmonic sources, such as the one for the bass source in Figure 2.10, exhibit a structured pattern with a fundamental frequency (F0) and its harmonics. The fundamental frequency is the lowest frequency component of a sound that defines its perceived pitch. Harmonics, on the other hand, are integer multiples of the fundamental frequency. These sources are consistent over time and often appear in spectrograms as horizontal components. However, harmonious interplay among sources can cause frequency overlap, making separation difficult.

Percussive signals, on the contrary, have energy distributed across a wide frequency range. They have a relatively flat frequency spectrum and are distinguished by brief, high-energy events. This is evident in Figure 2.10 and Figure 2.9(b), where the spectrogram of the drums signal reveals distinct vertical structures. Percussive instruments are important in communicating rhythmic information in music, as well as adding a sense of speed or tempo to a piece of music.

In reality, music signals frequently contain a mix of harmonic and percussive elements. This is noticeable in a variety of musical sources, such as singing voice, which consists of an intricate blend of harmonic components caused by vocal cord vibrations along with percussive-like elements such as consonant sounds, like 'k' or 'p'. This mixture can be observed in Figure 2.10.

Finally, one distinguishing feature of musical sources is their tendency to be sparse, which means that they have little energy across a significant number of time and frequency points. The presence of repeated components across various time scales, such as a recurring drumming pattern lasting a few seconds, is another distinguishing feature of music signals. These repetitions provide valuable leverage when performing MSS.

2.3 Non DNN-based Techniques for Separation of Sources

2.3.1 Non-Negative Matrix Factorization

One of the most popular signal processing algorithms for solving MSS problems is Non-Negative Matrix Factorization (NMF) [1], a method popularized by Daniel D. Lee & H. Sebastian Seung papers [20].

The NMF algorithm is designed to factorize a given data matrix M into two non-negative unknown matrices W and H so that their product imitates the original matrix.

$$WH = \hat{M} \sim M$$

In the case of MSS, M is the mixture's non-negative magnitude spectrogram, W is a dictionary of spectral templates describing the spectral properties of the sources, and H is a matrix of time activations. More specifically, W is known as the dictionary matrix, and H is known as the activation matrix. Figure 2.11 shows an example of a series of spectral templates and their corresponding time activations. Peaks in the time activations indicate the times when a specific spectral feature was identified in the mix. Notice, for example, that the peaks in the time activation H_1 correspond to the spectrogram's percussive hits.

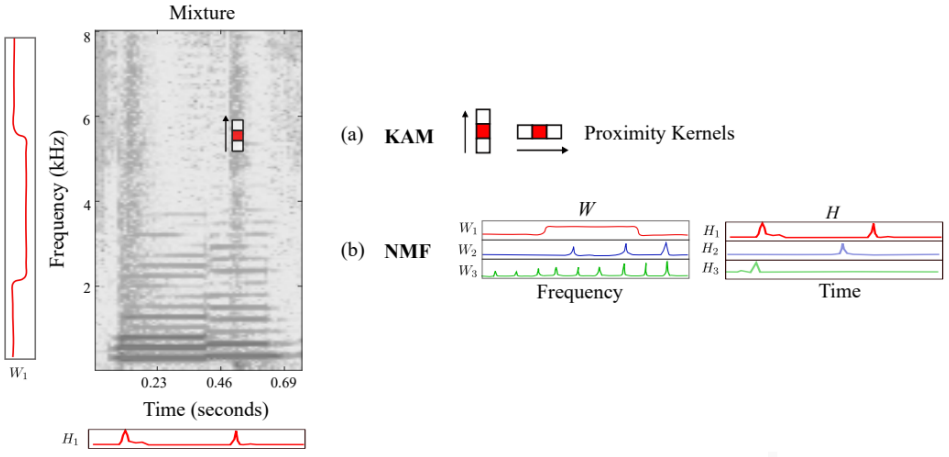


Figure 2.11: Example of different models used in MSS: (a) Proximity kernels used for harmonic-percussive separation within a Kernel Additive Modeling (KAM) approach, (b) Spectral templates W and time activations H within a Non-Negative Matrix Factorization approach. Image from [1].

Given M of dimension $K \times N$. W and H, are of dimensions $K \times S$ and $S \times N$, respectively, where S is the number of sources to be extracted [21].

The factorization process is solved as an optimization problem by minimizing the discrepancy

between M and WH using popular divergence measures such as Kullback Leibler or Itakura-Saito.

$$\arg\min_{W,H\geq 0} D(M|WH) = D(M|\hat{M}) = \sum_{i}^{K}\sum_{j}^{N} d(V_{i,j}|[WH]_{i,j})$$

Once the matrices W and H are obtained, the spectrogram $\hat{S}_k$ of the estimated source k is obtained with

$$\hat{S}_k = W_k H_k$$

with W_k being the k^{th} column of W and H_k being k^{th} line of H.

2.3.2 Kernel Methods

Kernel Additive Models (KAM) are music separation models that take advantage of local features visible in music spectrograms such as continuity and repetition [1]. KAM involves picking a set of time-frequency bins that, given the nature of the target source, ought to be similar in value in order to estimate an audio source at a specific time-frequency point. This collection of time-frequency bins is known as a proximity kernel. Examples of proximity kernels are depicted in Figure 2.11 as well as in Figure 2.12.

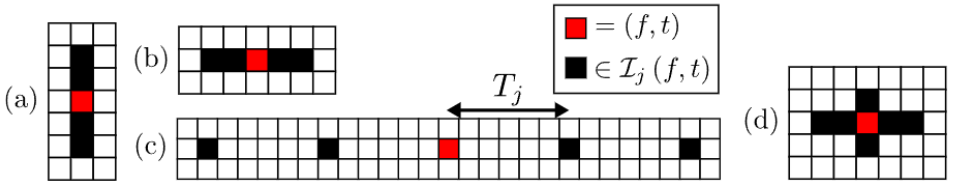
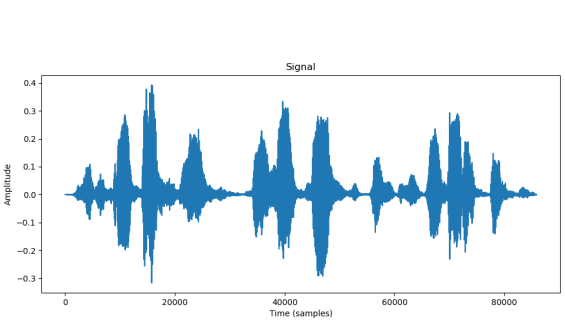


Figure 2.12: Examples of proximity kernels to account for prior knowledge about sources. (a) vertical, for percussive elements,(b) horizontal, for stable harmonic elements, (c) periodic, for repetitive elements, (d) cross-like, for smoothly varying spectrograms such as vocals. Image from [22].

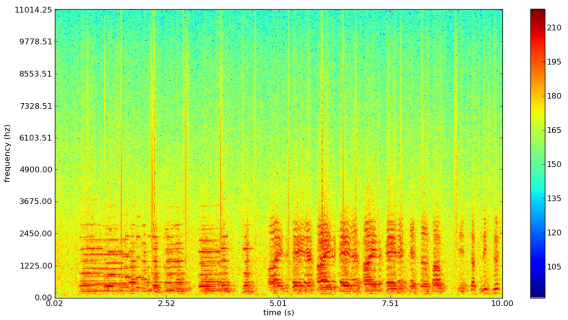
2.4 Performing Separation with Neural Networks

As explained earlier, the use of Deep Neural Networks (DNNs) in MSS has rapidly increased in recent years. MSS models can be divided into two main categories: waveform and pre-processed waveform (e.g., spectrogram) algorithms [23].
Within the waveform category, the model accepts the raw waveform signal as its input and conducts source separation operations directly in the temporal domain. This approach involves the direct prediction of individual source waveforms based on the provided mixture waveform.

Pre-processed waveform models on the other hand, take a slightly different approach by incorporating pre-processing steps into the source separation pipeline. These models include a preliminary pre-processing stage in which the input audio waveform is subjected to some form of feature extraction or transformation. The pre-processed portrayal is then fed into the



(a) Waveform signal. Image from [5].



(b) Spectrogram. Image from [24].

Figure 2.13: Visualization of both categories of inputs.

separation model as input. The most common pre-processing technique used in pre-processed waveform models is the STFT explained in section 2.1.3.

2.4.1 Spectrogram-based Models and Masking

Separation can be accomplished in two ways when a DNN-based method that takes magnitude spectrograms as inputs. Targets can be defined as either the magnitude spectrograms of the desired sources or their separating masks. A mask is a matrix that shares the dimensions of the spectrogram. Each mask value in the range $[0.0, 1.0]$ determines the percentage of energy from the initial mix that a source accounts for. In other words, a value of 1.0 allows the allocation of all sound components from the mixture to the considered source in a given time-frequency bin, whereas a value of 0.0 blocks the transmission of any sound component from the mixture. [25].

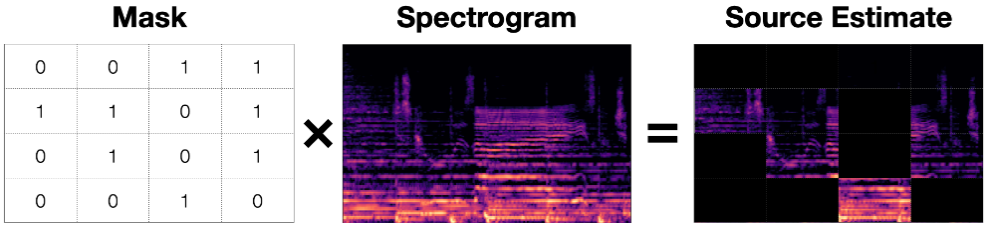


Figure 2.14: Simplified visualization of the masking operation with a binary mask. Image from [25].

As expressed in the Figure 2.14, the source is estimated by element-wise multiplication of the mask obtained from the model with the spectrogram of the initial mixture. Let $\hat{M}_i \in [0.0, 1.0]^{T \times F}$ be the mask of the i^{th} source S_i in a mixture represented by a magnitude spectrogram $|Y| \in \mathbb{R}^{T \times F}$, the estimate of the source magnitude spectrogram $\hat{S}_i$ is obtained with:

$$\hat{S}_i = \hat{M}_i \odot |Y|$$

Three types of masks can be considered:

- Binary masks: masks in which the entries are limited to only two values, namely 0.0 or 1.0. Such masks make the assumption that a single source dominates each time-frequency bin in the mixture. It is worth noting that binary masks are no longer commonly used.

- Soft masks or ratio masks: Soft masks can accept values from the inclusive interval [0.0, 1.0]. Soft masks, as opposed to binary masks, assign a certain percentage of the energy in a mixture to a source. As a result, the energy in a mixture's time-frequency bin can be distributed among several sources. Soft masks typically produce better sounding results due to their flexibility. It is rare for the entirety of the energy in a mixture to be exclusively allocated to a single source. This is the type of masks in use in current spectrogram-based models.

- Ideal masks: The term "Ideal Mask" implies the best possible performance in a mask-based source separation approach. Even with an ideal mask, the estimated source spectrogram may not exactly match the true source spectrogram due to factors such as noise, imperfect modeling, or separation algorithm limitations. As a result, while an ideal mask strives to deliver the best possible performance in source separation, it cannot guarantee that the estimated source $hatS_i$ is the same as the true source S_i.

2.4.2 Waveform-based Models and Hybrid Approach

While the magnitude spectrogram is used as an input by the majority of source separation techniques [26], an important aspect of the signal, namely the phase, is frequently left out because most spectrogram approaches use the magnitude spectrogram. Excluding possibly helpful data from the phase presents the risk of acquiring a sub-optimal solution. The goal of waveform-based models is to directly approach the problem of music source separation through end-to-end learning. Several models such as the Wave-U-Net (Stoller et al., 2018) [27] model or the DEMUCS (Défossez et al., 2021) [28] model provided results confirming that waveform-based deep learning models can outperform spectrogram-based models.

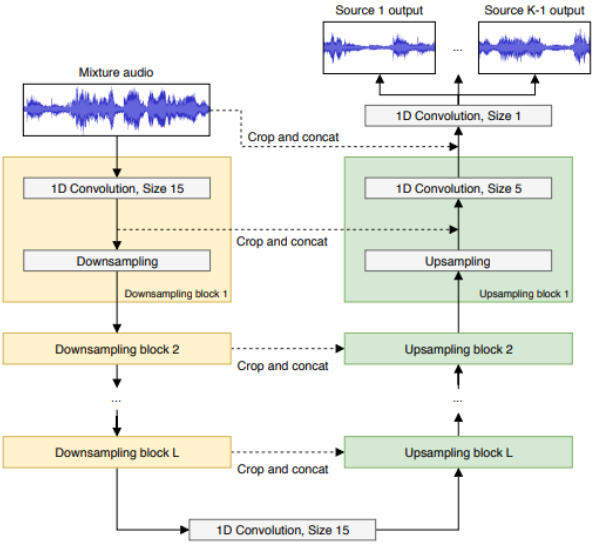


Figure 2.15: Wave-U-Net model. Image from [27].

However a new approach has recently been proposed with `Hybrid Demucs` (Défossez et al., 2022) [29] to bridge the gap between spectrogram-based and waveform-based models. The model performs separation in both domains and combines the results. The latest version of their model, `Hybrid Transformer Demucs` (Défossez et al., 2022) [30], currently holding the best results on the MUSDB18 dataset (see description of the dataset in Section 4.1).

2.4.3 Typical Approach

As already mentioned, most of the DNN models perform the separation in the time-frequency domain with magnitude spectrograms. On the "typical" approach to MSS with Neural Networks, depicted in Figure 2.16, one can observe that a model needs to be trained for each instrument specifically. This is due to the very different nature of each source and its representation on a spectrogram, as seen in section 2.2. Although some rare models, such as `Demucs`, separate all the sources directly with one model.

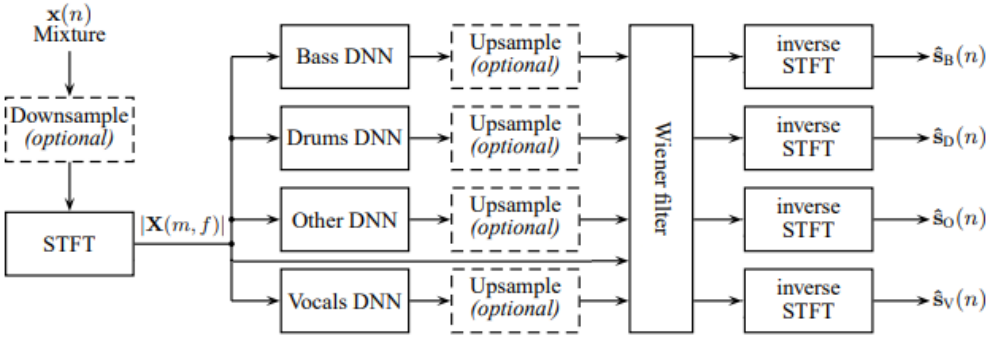


Figure 2.16: Typical approach to MSS with DNNs. Image from [31].

First the STFT of the mixture that requires separation is computed following an optional downsampling step[3]. Subsequently, the STFT magnitude spectrogram is passed through the previously trained models, each corresponding to one specific source, to acquire estimations of the corresponding STFT magnitudes. A post-processing step is executed using either a single channel Wiener filter or a multi-channel Wiener filter, aiming to enhance the separation performance [31]. The use of the Single-Channel Wiener Filter (SWF) or Multi-Channel Wiener Filter (MWF)[4] can be an important post-processing step that ensures the aggregation of the four estimates yields the original mixture. This step helps in reducing interference from other sources as well as separation artifacts. Lastly, the inverse STFT is applied to these filtered spectrograms to recover the final estimates of the sources.

Finally, one can notice that the amount and nature of the sources to separate are fixed beforehand, having MSS systems that can dynamically identify and isolate an arbitrary amount of sources is an open challenge, and deep clustering methods represent one potential approach to developing such systems. A further limitation of most DNN-based architectures is that their cost functions frequently use Mean Squared Error (MSE), which has a poor correlation with perceived audio performance. This is why the development of more suitable cost functions is an active research subject in MSS as well [1].

2.5 Multi-channel Wiener-filter

The multi-channel Wiener filter assumes the signal model [31]

$$\mathbf{X}(m, f) = \mathbf{S}_i(m, f) + \mathbf{Z}_i(m, f)$$

[3]generally from 44.1kHz to 32kHz.

[4]In this piece of work the audio data is a stereo signal, hence the MWF is in use.

with $i \in \mathcal{I}$ and $\mathbf{Z}_i(m,f) = \sum_{j \in \mathcal{I} \setminus i} \mathbf{S}_i(m,f)$ where each time-frequency bin $\mathbf{S}_i(m,f)$ is a complex Gaussian with zero-mean and covariance matrix $v_i(m,f)\mathbf{R}_i(f)$. $\mathbf{R}_i(f)$ being the time-invariant spatial covariance matrix and $v_i(m,f)$ being the power-spectral density (PSD).

To apply the MWF, we first need to approximate the PSDs $v_i(m,f)$ and spatial covariance matrices $\mathbf{R}_i(f)$ which is done from the output of each instrument's specific model. In particular, we estimate them from M consecutive frames by

$$\hat{v}_i(m,f) = \frac{1}{2}||\hat{\mathbf{S}}_i(m,f)||^2, \;\; \hat{\mathbf{R}}_i(f) = \frac{\sum_{m=1}^{M} \hat{\mathbf{S}}_i(m,f)\hat{\mathbf{S}}_i(m,f)^H}{\sum_{m=1}^{M} \hat{v}_i(m,f)}$$

The minimum mean squared error estimator for $\mathbf{S}_i(m,f)$ from $\mathbf{X}(m,f)$ is then given by

$$\hat{\mathbf{S}}_i(m,f) = \hat{v}_i(m,f)\hat{\mathbf{R}}_i(f)\left(\sum_{j \in \mathcal{I}} \hat{v}_i(m,f)\hat{\mathbf{R}}_i(f)\right)^{-1} \mathbf{X}(m,f)$$

Chapter 3

DNN-based MSS Architectures

This chapter introduces three significant models from the MUSDB18 benchmark [32], which holds much importance in the field. The first model, which has been surpassed in performance as of now, provided a reference implementation based on deep neural networks as until then, no open-source implementation that achieves state-of-the-art results was available [33]. Consequently, the source code for this model was employed as a base throughout this study. The remaining two models represent the most advanced and effective approaches currently available for tackling the MSS problem.

3.1 OpenUnmix

The primary motivation for OpenUnmix was to provide an open-source implementation and pre-trained music source separation models. As previously stated, there was a scarcity of easily accessible and state-of-the-art open-source techniques for source separation prior to OpenUnmix. The developers hoped to promote collaboration, reproducibility, and further innovations in the field by making OpenUnmix freely accessible and easily comprehensible. Scientists could start with OpenUnmix, fine-tune the models on their own datasets, and then contribute back to the community [33].

To perform separation into multiple sources, Open-unmix employs a collection of models specifically trained for each individual target. The models are trained to estimate the magnitude spectrogram[1] of a desired source, such as vocals, given the magnitude spectrogram of a mixed input. The prediction is internally generated by applying a mask to the input spectrogram.

First, the input STFT magnitude spectrogram is cropped at 16kHz, then the global mean and standard deviation across all time frames are calculated to standardize this input [34]. Then, the network compresses the frequency and channel dimensions of the model in order to reduce redundancy within the data and facilitate faster convergence of the model. After that, at the heart of open-unmix lies a three-layer bidirectional LSTM network as can be observed in Figure 3.1. Following the three-layer BLSTM, its output is decompressed back to the dimensions of the original cropped input. Finally, the signal is rescaled and passed through a ReLu activation function to generate the mask which is multiplied with the input magnitude spectrogram to generate the estimated target spectrogram.
During optimization, the model is trained using the mean squared error criterion.

[1]Note that this implies working with real values instead of the initial complex STFT coefficients.

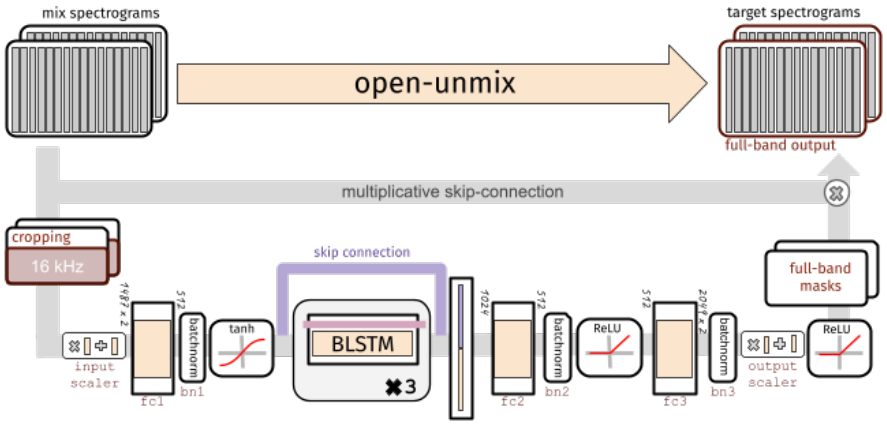


Figure 3.1: OpenUnmix architecture. Image from [34].

OpenUnmix can be trained and evaluated using audio signals of any length. However, because it relies on both future and past knowledge at the same time, the model is unsuitable for applications that operate in real time [34].

3.2 Band-Split RNN

Band-Split RNN (BSRNN) is a frequency-domain MSS model that splits a spectrogram into a series of subband spectrograms with a set of predefined bandwidths. The bandwidths are adjusted for different instrument types based on a priori knowledge on the characteristics of the target source in order to optimize the performance on this specific target instrument. For example, if the target source mainly lies in lower frequency parts with a relatively low fundamental frequency, such as the bass, we can perform fine-grained band-splitting scheme at lower frequency parts to increase the frequency resolution and coarse-grained band splitting scheme at higher frequency parts to save the computational complexity [2].
They show by experiments that band-splitting schemes do play an important role in the separation performance and that BSRNN can surpass the performance of existing state of-the-art music source separation systems.

The overall architecture[2] is divided into three modules, namely: the "Band Split Module", the "Band and Sequence Modeling Module" and finally the "Mask Estimation Module". Figure 3.2 shows the overall pipeline of the BSRNN model.

Band Split Module

This first module, which is depicted in Figure 3.2(B), takes as input $\mathbf{X} \in \mathbb{C}^{F \times T}$, a complex-valued spectrogram[3] generated by STFT, F and T being the number of frequency bins and the number of time frames respectively. It then splits this input into K subband spectrograms $\mathbf{B}_i \in \mathbb{C}^{G_i \times T}, i = 1, ..., K$ with instrument-specific predefined bandwidth $\{G_i\}_{i=1}^{K}$ such that $\sum_{i=1}^{K} G_i = F$. The real and imaginary part of each subband spectrogram $\mathbf{B}_i$ are then concatenated before passing through a layer normalization module and a fully-connected (FC) layer to derive a real-valued subband feature $\mathbf{Z}_i \in \mathbb{R}^{N \times T}$ where N is the number of

[2]Being the core of this project, the architecture and functioning of BSRNN are explained in details.
[3]It is important to note that this model does not operate on the magnitude spectrogram as OpenUnmix does but on the complex values directly.

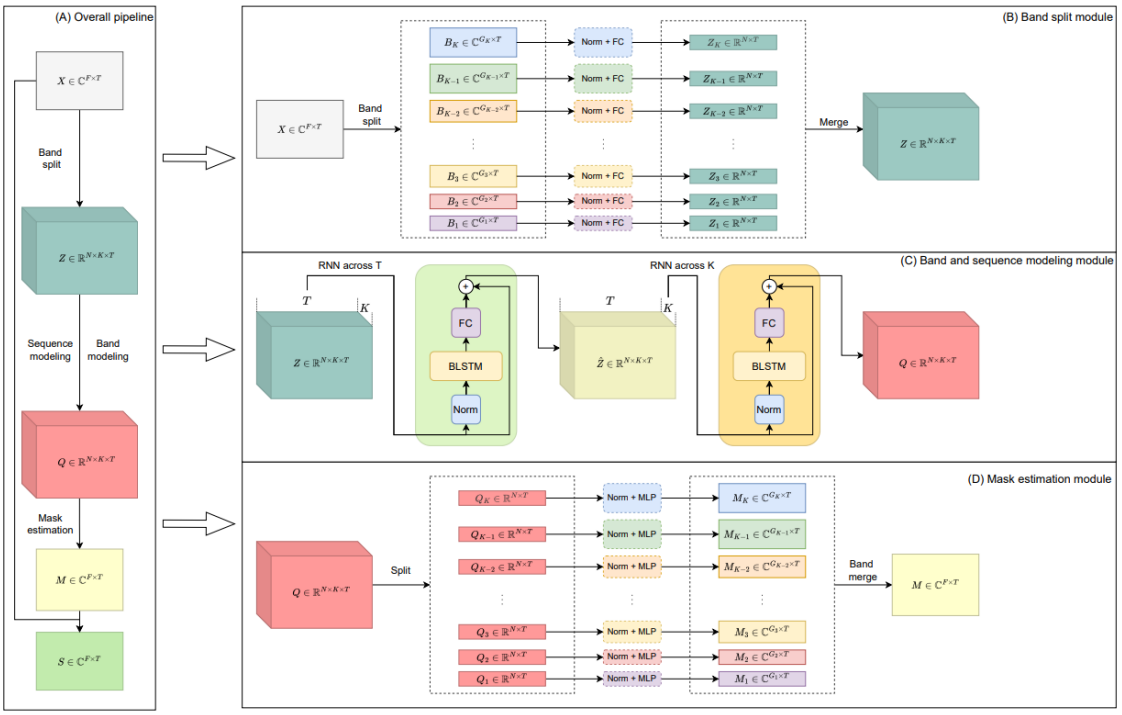


Figure 3.2: Band-Split RNN architecture. Image from [2].

features. The K subband features are then stacked to obtain a transformed fullband feature representation $\mathbf{Z} \in \mathbb{R}^{N \times K \times T}$ of the input.

Band and Sequence Modelling Module

In the second module, observable in Figure 3.2(C), BSRNN performs interleaved sequence-level and band-level modeling with two distinct bidirectional LSTMs. Firstly, $\mathbf{Z}$ is fed as input to the sequence-level BLSTM which performs modelling across the time dimension T. The band-level BLSTM is then applied to $\hat{\mathbf{Z}}$, the output of the first BLSTM, across the band dimension K, where for each temporal frame, the BLSTM is expected to identify the intra-band feature relationships across the K subbands. In both cases a group normalization module is first applied to the module's input followed by the BLSTM itself and finally, the actual modelling is performed by a FC layer. Lastly, a residual connection is added between the input and the output of the sub-module. The output of the last layer is denoted by $\mathbf{Q} \in \mathbb{R}^{N \times K \times T}$. A deeper architecture can be created by stacking multiple instances of these sub-modules.

Mask Estimation Module

The third and final module, the design of which is detailed in Figure 3.2(D), is the mask estimation module. It aims to derive a complex-valued[4] time-frequency mask from the output $\mathbf{Q}$ of the previous module in order to extract the target source. Similarly to the operation of the Band Split module, $\mathbf{Q}$ is segmented into K features $\{\mathbf{Q}_i\}_{i=1}^{K} \in \mathbb{R}^{N \times T}$. Each of these subband features first passes through a layer normalization module, then through a multilayer

[4]Note that the masking method explained in section 3.3.1 made use of real-valued masks for the magnitude spectrogram. The process here is the same one but we derive complex-valued masks for the complex-valued spectrogram.

perceptron (MLP) with one hidden layer to produce the real and imaginary parts of the T-F masks $\mathbf{M}_i \in \mathbb{C}^{G_i \times T}, i = 1, ..., K$, where G_i are the same predefined bandwidths described in the Band Split module. The hyperbolic tangent function and a gated linear unit (GLU) are used as the nonlinear activation function and as the output layer of the MLP respectively. Each subband feature has its own normalization module and MLP. Finally, and still similarly to the first module, all subband masks $\mathbf{M}_i$ are then merged to form the full T-F mask $\mathbf{M} \in \mathbb{C}^{F \times T}$. The element-wise multiplication between this mask and the original mixture spectrogram $\mathbf{X}$ is finally performed to generate the target spectrogram $\mathbf{S} \in \mathbb{C}^{F \times T}$.

3.3 Family of Demucs models

3.3.1 Demucs

Until the publication of the paper "Music Source Separation in the Waveform Domain" by Défossez et al. in 2019 [28], all state-of-the-art approaches in MSS operated on the spectrograms derived by the STFT. Although some end-to-end architectures for directly producing waveforms were proposed, their performance were outmatched by that of state-of-the-art spectrogram-based methods.

Défossez et al. introduced Demucs, a waveform-to-waveform model comprising of a U-Net architecture with a convolutional encoder, a convolutional decoder and skip U-Net connections linking the encoder and decoder. Notably, a bidirectional LSTM is incorporated between the encoder and decoder modules. The complete architecture can be observed in Figure 3.3.

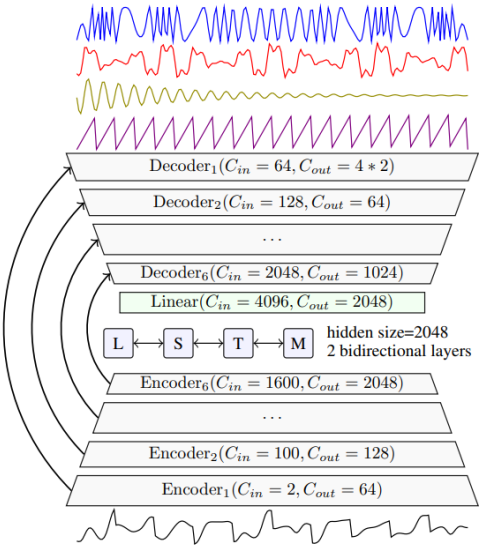


Figure 3.3: Demucs complete architecture. Image from [28].

During training, a straightforward L1 loss is employed to calculate the discrepancy between the estimated and ground truth sources.

The authors demonstrated that by applying appropriate data augmentation techniques, Demucs surpassed all existing state-of-the-art architectures at the time of publication.

3.3.2 Hybrid Demucs

Published by Alexandre Défossez in 2021, the paper "Hybrid Spectrogram and Waveform Source Separation" [29] proposed for the first time a true end-to-end hybrid source separation making use of both time-domain and frequency-domain. Although other prior methods made use of network blending model blending[5] from different domains as a simpler post-training alternative.

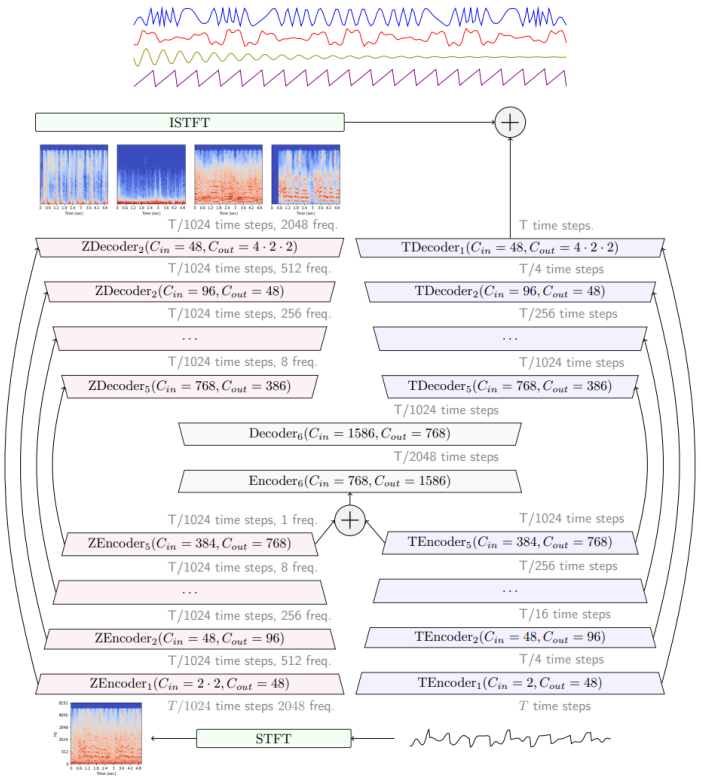


Figure 3.4: Hybrid Demucs complete architecture. Image from [29].

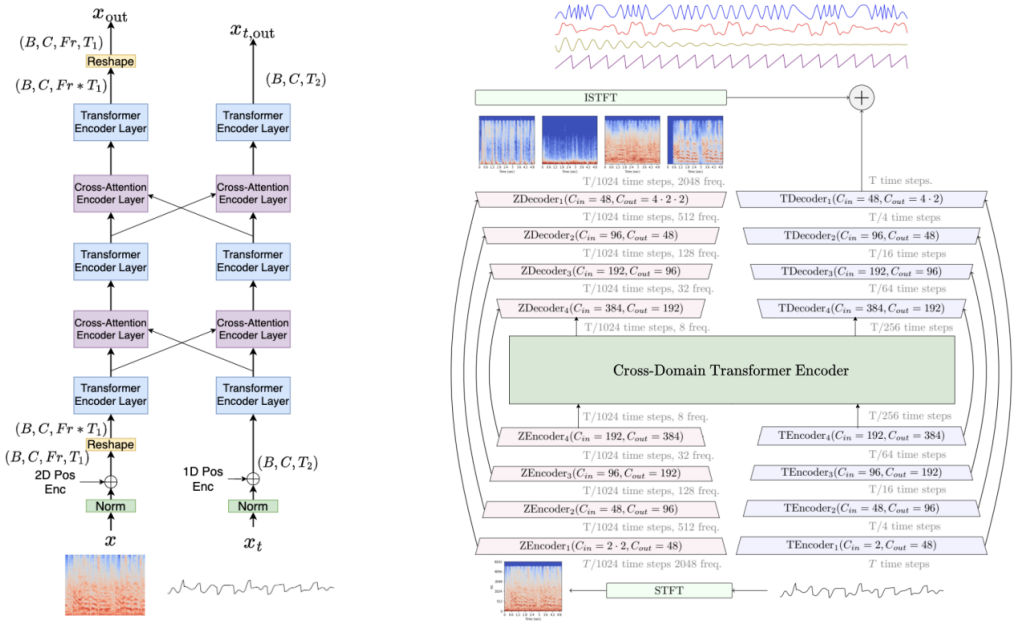
Hybrid Demucs is a Demucs architecture extension that performs hybrid waveform/spectrogram domain source separation. The original U-Net architecture is enhanced with two parallel branches: one in the time domain and one in the frequency domain. At the core of its architecture, Hybrid Demucs, like the original Demucs, relies on BLSTMs. The spectral branch output is transformed back into temporal domain using the iSTFT and summed with the temporal branch output to create the final estimation of the model. Because of this overall design, the model is able to use whichever domain is most suitable for different parts of the signal, even if they are from the same source, and is able to exchange information between the two representations.

When it was published, Hybrid Demucs outperformed all the other State-of-the-art methods bringing strong improvements of the overall quality and absence of bleeding between sources as measured by human evaluations. It has later been outperformed by BSRNN.

[5]Network or model blending consists in a weighted sum of the outputs of two different networks to produce a new prediction which potentially improves performances.

3.3.3 Hybrid Transformer Demucs

In the paper "Hybrid Transformers for Music Source Separation" published in 2022 [30], Défossez et al. introduced Hybrid Transformer Demucs (HT Demucs), an hybrid temporal/spectral bi-U-Net based on Hybrid Demucs, where the central BLSTM-based encoder is replaced by a cross-domain Transformer Encoder, using self attention within one domain, and cross-attention across domains as can be observed in Figure 3.5(b). However, it only outperformed Hybrid Demucs (trained on the same data) by 0.45 dB of SDR when using the original dataset of 100 songs, MUSDB18 which is described in the next section, with 800 additional training songs, and displayed poor performances on the original data alone. It also outperformed BSRNN and is currently the best performing on the MUSDB18 benchmark.



(a) Cross-domain Transformer Encoder of.

(b) Hybrid Transformer Demucs.

Figure 3.5: Hybrid Transformer Demucs architecture. Images from [30].

Chapter 4

Dataset and Data Processing

4.1 MUSDB18 Dataset

The base[1] dataset that is in use in almost every model performing MSS is the MUSDB18 dataset. It was created specifically to be utilized as a central reference for the purpose of developing and evaluating source separation methods, and it was selected as the official dataset for the Signal Separation Evaluation Campaign (SiSEC) in 2018, an internationally recognized initiative with the goal of analyzing the effectiveness of source separation algorithms [35].

The MUSDB18 dataset consists of 150 full songs representing different styles of music, including both mixed stereo records and their original isolated sources. The dataset is divided into two folders: a training folder of 100 songs and a test folder with the remaining 50 songs. [36].

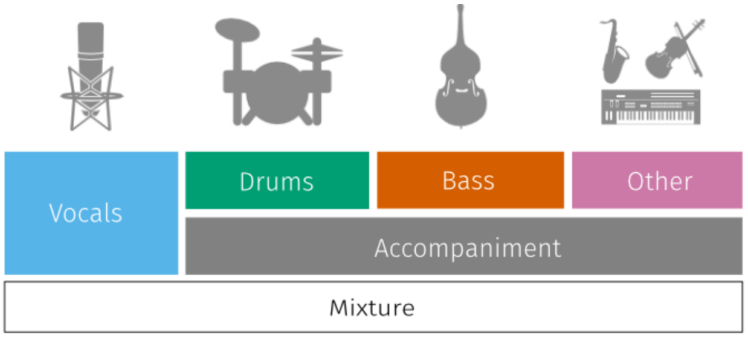


Figure 4.1: Image from [37].

This collection of songs is composed of various sources, including the DSD100 dataset comprising 100 tracks [38], the MedleyDB dataset encompassing 46 tracks [39], and an additional four tracks from diverse origins. Finally, this substantial dataset exists in two forms, the "standard" compressed MUSDB18 dataset and the uncompressed MUSDB18-HQ dataset.

MUSDB18 - Compressed STEMS

All files within the dataset are stored with the Native Instruments stems format (.mp4). This multitrack format includes five stereo channels, each of which is encoded using Advanced

[1]The term "base" implies that it is not necessarily the only dataset used. BSRNN and Demucs for example both curated private datasets of more than a thousand additional songs.

Audio Coding (AAC). AAC is a popular audio compression format known for its ability to provide high-quality sound at relatively lower bitrates compared to other codecs [40].
These streams correspond to the following audio signals:

0 - The mixture,

1 - The drums,

2 - The bass,

3 - The rest of the accompaniment, briefly called ”other”,

4 - The vocals.

Summing all of the individual signals yields the mixture signal. All signals are stereophonic, which means they use two audio channels to create the sensation of multi-directional sound perception, and are encoded at a standard sample rate of 44.1kHz.
The size of this compressed dataset is 4.4 Gb.

MUSDB18-HQ - Uncompressed WAV

Alternatively, SigSep proposes the uncompressed WAV files for models that aim to predict high bandwidth of up to 22 kHz resulting in a dataset size of 22.7 Gb. Other than that, MUSDB18-HQ is identical to MUSDB18 [41].

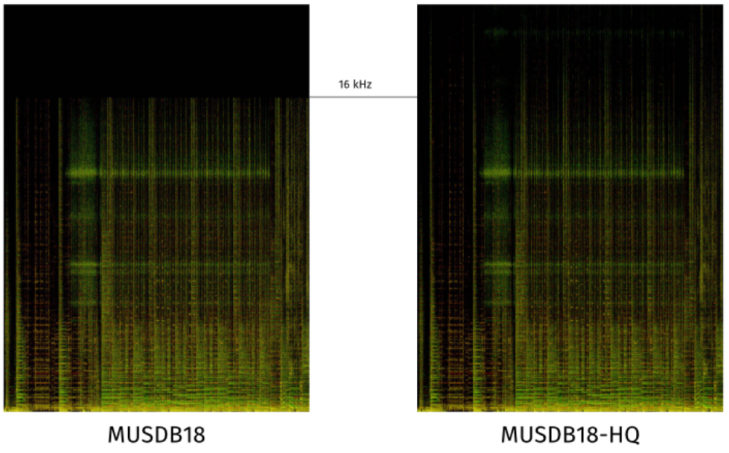


Figure 4.2: Comparison of the spectrograms of the same sample from MUSDB18 and MUSDB18-HQ. Image from [37].

4.2 Pre-processing

The models explored in the previous section are not directly trained on the full length of a track at once because it could be computationally very expensive. The tracks thus need to be segmented into smaller training samples. In this work the primary objective is to reproduce the BSRNN model, hence the pre-processing method proposed by Luo et al [2] for BSRNN, described below, is reproduced.

In order to properly prepare the training data, a simple energy based thresholding method serving as a source activity detector (SAD), is applied to remove the silent regions in the sound tracks and only retain the salient ones.

Each sound track is first split into overlapped segments of length L, measured in seconds, with an overlap of duration $L/2$. Each segment is further split it into 10 chunks of length $L/10$ and the respective energy of each of these chunks, considering the mixture and not specific targets, is calculated. In this training configuration, L is set to 6 seconds as in the BSRNN paper.
To evaluate this energy the following formulas are used[2]:
• for MUSDB18-HQ: $10 \log_{10} \sigma^2$ (in decibels)
• for MUSDB18: $\sum_n x[n]^2$

Due to the difference in encoding methods for MUSDB18 and MUSDB18-HQ the order of magnitude of the amplitude values describing the tracks differ drastically between the two datasets. While the uncompressed MUSDB18-HQ records the amplitude as potentially large integers, often in the order of 10^3, the compressed version records them as small float values, generally in the order of 10^{-2}. The implication of this difference is that the formula for calculating the energy in decibels (dB) [42], namely $10 \log_{10} \sigma^2$, can not be used for the compressed dataset as it would yield negative values of energy. Hence another evaluation of the energy of an audio signal is used for it [43], $\sum_n x[n]^2$.

For silent chunks, their energy is set to a small value $\epsilon = 1e - 5$. After that an energy threshold is derived for the full sound track by calculating the 15% quantile of the energy of all chunks. If more than half of the chunks in a segment have their energy higher than the threshold, then the segment is considered a salient segment and is retained as a valid training data segment. Note that for each segment, the corresponding mixture and all of the isolated targets are saved.

This pre-processing method led to the creation of 6565 valid samples for the compressed data and 6567 samples for the HQ dataset so the two distinct evaluation methods of energy lead to almost exactly the same results with about $\sim 88\%$ of the segments retained.

4.3 On-the-fly Data Simulation

In order to avoid being limited by the amount of data available, additional segments of data are simulated by taking inspiration of both the data simulation performed by BSRNN and the methods of data augmentation for MSS proposed in the paper "*Improving Music Source Separation Based on Deep Neural Networks through Data Augmentation and Network Blending*" by Uhlich et al. (2017) [31].

Sound tracks from different songs are randomly mixed during training to allow for batch-level on-the-fly data simulation. First, four SAD pre-processed salient segments are randomly selected and a different target is isolated in each of them. All these isolated target segments are scaled by a random target-specific amplitude value drawn from a uniform distribution in the range $[0.25, 1.25]$. After that all the target segments are added together to form a

[2]The paper does not specify how the energy is calculated, however other mentions of energy in the paper use decibels as a unit. Hence the choice of the first formula.

simulated valid sample of training data. A visual representation of the process is depicted in Figure 4.3

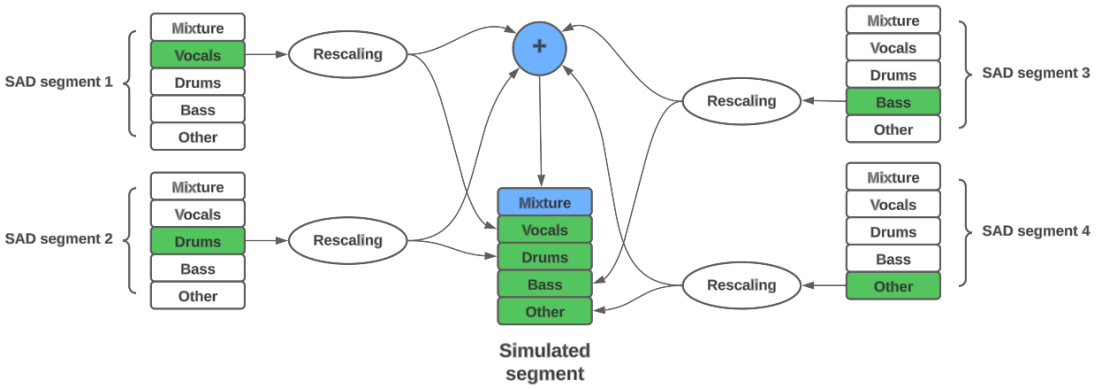


Figure 4.3: Representation of the on-the-fly data simulation process. Source: Martin Laurent.

Chapter 5

Experiment Configuration

In this work, several models have been trained and evaluated on the samples obtained from the compressed MUSDB18 dataset. First the official OpenUnmix model implementation [34] in order to serve as a baseline for comparison purposes. After that a reproduction of the BSRNN model which has been described in section 3.2 and the implementation of which constituted the core of this project. And finally, a modified version of BSRNN that performs separation on real-valued spectrograms obtained with the Short-Time Discrete Cosine Transform (see section 2.1.3). The first and third modules of the model had to be modified to accommodate for real-valued spectrograms instead of complex-valued ones.

For every model the set of SAD pre-processed samples are split as follows, the first 85% of them are used for training and the remaining 15% serve as validation set. As the order of the segments follows the order of the songs in the original dataset, there can only be at most 3 seconds (the duration of the overlap in the pre-processing) of shared data between the train set and the validation set. In other words, with 6565 SAD samples of 6 seconds, at most $\frac{3}{6565.6} \sim 0.0076\%$ of the data is shared, which is negligible.

For every considered architecture one model has to be trained for each target. Each model is trained with the Adam optimizer with an initial learning rate of $1e-03$ and epoch processes 10000 samples of either SAD pre-processed samples or simulated samples. More specifically, among these 10000 samples, $\lfloor 6565 \times 0.85 \rfloor = 5580$ samples are SAD pre-processed and 4420 samples are simulated. The data simulation allowed to almost double the amount of data for training.

5.1 OpenUnmix

As mentioned previously, OpenUnmix serves as a baseline in this project, for comparison purposes as well as the foundation from which the unofficial implementation of BSRNN has been built. Even though the official git repository of the model provides pre-trained models for each target, they have not been trained with the same pre-processing and simulation techniques as those in use in this work. For this reason, it has been fully retrained on the configuration detailed above.

Naturally, the hyperparameter configuration of the original OpenUnmix has remained untouched as well. As already cited in Section 3.1, OpenUnmix takes magnitude spectrograms as inputs, those are obtained via STFT with frame size $N_{fr} = 4096$ and hop size $H = 1024$

applied to the 6 seconds data samples. The 'hidden size' of the model, which corresponds to the input dimension of the BLSTM as depicted in the diagram in Figure 3.1, is set to 512 while the number of layers in the BLSTM is set to 3. Finally, the model trains on a maximum of 1000 epochs with a batch size of 16 and the learning rate is decayed on plateau by 0.3 if no improvement is observed for 80 epochs. Early stopping is applied when a better validation is not found in 140 consecutive epochs. Lastly, the training objective is the MSE loss between the estimated target spectrogram and the true target spectrogram.

5.2 Reproducing and Adapting BSRNN

5.2.1 Band Splitting Schemes

First and foremost, as the model is based on the frequency-wise segmentation of the input spectrogram, the manner in which these inputs are split must be established. The sources displaying their own set of characteristics, each of them require its own band splitting scheme. In the original paper, they empirically determined the best separations for each source among a set of 7 different options. The objective being to reproduce their experiment, their optimal splits for each instrument are reproduced as well.

First, *vocals* and *other* share the same partition, resulting in 41 subbands, detailed in Table 5.1

Frequencies f	$f \leq$1kHz	1kHz< $f \leq$4kHz	4kHz< $f \leq$8kHz	8kHz< $f \leq$16kHz	16kHz< $f \leq$20kHz	20kHz< $f \leq$22.5kHz
Bandwidths	100Hz	250Hz	500Hz	1kHz	2kHz	1 subband

Table 5.1: Band splitting scheme for *vocals* and *other*.

The Table 5.2 shows the splitting for the *drums*, resulting in 55 subbands.

Frequencies f	$f \leq$1kHz	1kHz< $f \leq$2kHz	2kHz< $f \leq$4kHz	4kHz< $f \leq$8kHz	8kHz< $f \leq$16kHz	16kHz< $f \leq$22.5kHz
Bandwidths	50Hz	100Hz	250Hz	500Hz	1kHz	1 subband

Table 5.2: Band splitting scheme for *drums*.

Finally, the *bass* possess 30 subbands with the scheme depicted in Table 5.3.

Frequencies f	$f \leq$500Hz	500Hz< $f \leq$1kHz	1kHz< $f \leq$4kHz	4kHz< $f \leq$8kHz	8kHz< $f \leq$16kHz	16kHz< $f \leq$22.5kHz
Bandwidths	50Hz	100Hz	500Hz	1kHz	2kHz	1 subband

Table 5.3: Band splitting scheme for *bass*.

5.2.2 BSRNN with STFT

Reproducing the hyperparameter configuration described in the original paper, the input spectrograms are obtained via the pytorch implementation of the STFT with frame size $N_{fr} = 2048$ and hop size $H = 512$. The feature dimension N, i.e. the number features used to represent the concatenated real and imaginary parts of the subbands in the *Band Split* module, is set to 128. The BLSTMs in the *Band and Sequence Modelling* module are repeated 12 times while the hidden unit of BLSTM layers is set to be $2N = 256$ and the hidden size in the mask estimation MLP is fixed at $4N = 512$. The target-specific models are trained for a maximum of 100 epochs with a batch size of 2. The learning rate is decayed by 0.98 for every two epochs, and gradient clipping by a maximum gradient norm of 5 is applied. Early stopping occurs when a better validation is not found in 10 consecutive epochs.

Finally, the training objective is defined as the sum of a frequency-domain mean-absolute error (MAE) loss and a time-domain MAE loss:

$$\mathcal{L}_{obj} = |\mathbf{S}_r - \bar{\mathbf{S}}_r| + |\mathbf{S}_i - \bar{\mathbf{S}}_i| + |\text{iSTFT}(\mathbf{S}) - \text{iSTFT}(\bar{\mathbf{S}})|$$

where $\bar{\mathbf{S}} \in \mathbb{C}^{F \times T}$ and $\mathbf{S} \in \mathbb{C}^{F \times T}$ denote the complex valued spectrograms of the estimated and true targets respectively. The subscript r and i denote the real and imaginary parts and iSTFT denotes the inverse STFT operator.

5.2.3 BSRNN with STDCT

The input spectrograms are obtained with Jonas Haag's python implementation of the STDCT `pydct.sdct_torch` [44]. The hop length, or the parameter `frame_step` of the function, is set to 512 as in the STFT. However the parameter `frame_length` sets the size of the frequency dimension of the spectrogram. Hence, in order to produce a spectrogram of the same dimensions as in the former version of BSRNN, this parameter is set to 1024.

The frequency band segmentation defined in section 5.2.1 are maintained for this transformation. Indeed, Figures A.1, A.2, A.3 and A.4 clearly show that the spectrograms obtained with the two different transforms display very similar characteristics in the same areas of the frequency band thus allowing for the same splits of the input.

The original BSRNN *Band Split* module encodes the concatenated vectors of real and imaginary parts of the STFT spectrogram as a vector of $N = 128$ features. However the STDCT only producing real values and having no imaginary part to be concatenated, one could supposedly encode the information on only $N = 64$ features. This would lead to the hidden unit of the BLSTM layers to be $2N = 128$ and the hidden size of the mask estimation MLP to be $4N = 256$. For comparison purposes, both the configuration where $N = 64$ and the configuration where $N = 128$ will be explored. The BLSTMs in the *Band and Sequence Modelling* module are still repeated 12 times and the rest of the training configuration is the same as for the BSRNN with STFT spectrograms.

Finally, the training objective of the original BSRNN can not be used here due to the absence of an imaginary part in the spectrograms. Thus the training objective is defined as:

$$\mathcal{L}_{obj} = |\mathbf{S} - \bar{\mathbf{S}}| + |\text{iSTDCT}(\mathbf{S}) - \text{iSTDCT}(\bar{\mathbf{S}})|$$

where $\bar{\mathbf{S}} \in \mathbb{R}^{F \times T}$ and $\mathbf{S} \in \mathbb{R}^{F \times T}$ denote the real valued spectrograms of the estimated and true targets respectively and iSTDCT denotes the inverse STDCT operator.

5.3 Evaluation Process & Metrics

5.3.1 Producing Full Tracks Estimates

The evaluations of the models are effectuated by assessing the efficiency of the separations on the 50 full-length tracks from the test folder of the MUSDB18 dataset. OpenUnmix performs separation directly on the magnitude spectrogram of the entire track. Then it passes the real valued estimates magnitude spectrograms derived from the trained models and the complex valued mixture spectrogram of the track to the Wiener filter to derive the final complex estimate spectrograms that can be inverted back to temporal domain to produce the final waveform estimates.

While OpenUnmix produce the estimates of the entire track directly, it is not as straightforward for BSRNN. Indeed the track is first be split into smaller overlapping chunks that are all processed one at the time to produce all the consecutive target estimations which are then reassembled to produce the entire estimates for the full track.

In further details, the processed track is segmented into 6 seconds long chunks, with an overlap of 1 second between each consecutive chunk, to be consistent with the duration used for training. If needed, padding is applied to the last chunk to obtain a 6 seconds segment. Each chunk is then transformed with either STFT or STDCT and the resulting spectrogram is fed to the four models, each pre-trained to extract the spectrogram of a specific target. Once the target spectrogram estimations are obtained, they are inverted back to the time domain with either iSTFT or iSTDCT. Finally, the estimation of the entire track is produced with linear transition between the estimations of all the consecutive chunks. In other words, the values corresponding to the last second (overlap duration) of each chunk estimations are linearly weighted down from 1 to 0 except for the last chunk, and the values corresponding to the first second of each chunk are gradually weighted up from 0 to 1 except for the first chunk. After that overlap-add is applied to obtain estimates of the entire song, these estimates are cropped if padding was applied earlier to have estimates of the same duration as the original track.

Finally, the authors of BSRNN did not specify whether or not a Wiener filter was used before the inverse transforms of the estimated spectrograms. For this reason, a version of the evaluation described in the precedent paragraph is performed with the addition of a Wiener filter prior to the inverse STFT for comparison purposes.

5.3.2 The Metrics

The main evaluation metric used to evaluate the performances of MSS models is the *Sources to Distortion Ratio* (SDR), which served as the default evaluation metric in the SiSEC 2018 campaign [35]. However, the validity of this measurement has been recently questioned, as the results do not appear to necessarily correlate with perceptual evaluations gained through listening tests [1].

SiSEC used the Blind Audio Source Separation (BASS) algorithms proposed by Vincent et al. in the work "Performance Measurement in Blind Audio Source Separation" [45] in 2010. For these algorithms, the estimated sources $\hat{s}_i$, $\forall i \in \mathcal{I} := \{B, D, O, V\}$ (refer to section 1.2),

are each decomposed as:

$$\hat{s_i} = s_{target} + e_{interf} + e_{noise} + e_{artif}$$

Where s_{target} is the original source, e_{interf} is the interference coming from undesired sources, e_{noise} is the sensor noise and e_{artif} refers to the musical noise self-generated by the separation algorithm. The decomposition they propose is based on orthogonal projections and its mathematical details can be found in Appendix B.

They define several numerical performance criteria which are the SDR, the *Sources to Interferences Ratio* (SIR) and the *Sources to Artifacts Ratio* (SAR). These metrics are defined as:

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

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

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

The SDR measures the ratio of the power of the desired source signal to the power of the distortion introduced during separation. The SIR measures the ratio of the power of the desired source signal to the power of the interfering sources in the separated signal. And finally, the SAR measures the ratio of the power of the desired source signal to the power of any residual artifacts present in the separated signal. Higher values of these metrics indicate better separation as it implies that fewer distortion, interferences or artifacts are present in the estimated signal.

The official implementation of these metrics, `museval` [46] is used in this work. Provided all the true sources and corresponding estimates for a given track, `museval.evaluate` splits all these signals into 1 seconds chunks without overlap and evaluates all the metrics described above on these chunks. After that, for each metric and each source, the median across all chunks is calculated to derive an evaluation for this specific track. This process is repeated for each song in the test folder of MUSDB18 and the median of each metric and each source is computed across all songs. However, the evaluation of these performance indicators is a very time-consuming process.

In 2021, the Music Demixing (MDX) challenge [47] introduced a novel SDR measure defined as:

$$SDR = 10 * \log_{10} \frac{\sum_n ||s[n]||^2 + \epsilon}{\sum_n ||s[n] - \hat{s}[n]||^2 + \epsilon}$$

with ϵ being a very small value, typically $1e-07$, to avoid division by 0 or the evaluation of the log of 0. It evaluates directly the entire signal instead of 1 second chunks and its advantages are its fast evaluation and simplicity. In order to evaluate the performance of a model, the mean across the SDR scores of all songs is reported. It is worth noting that the performances reported on the MUSDB18 benchmark [32] are the ones obtained with the first definition of SDR on the compressed MUSDB18 dataset test folder.

Luo et al. used both of these metrics for evaluating BSRNN, hence all models in this project are evaluated with both SDR calculations as well. For the rest of this work the first definition of SDR will be referred to as **museval SDR** while the second definition will be **mdx SDR**.

5.4 Non DNN Baseline

Additionally, a python implementation of the Non-Negative Matrix Factorization algorithm [48] has been adapted to perform MSS. Its purpose is not to serve as a robust baseline for comparison but rather to provide an idea of the general quality of this technique compared to the DNN-based MSS methods. For this reason separation is performed only on ten randomly selected 6 seconds long samples for each song in the MUSDB18 test folder, the separation obtained is then evaluated with the same methods described previously.

A difficulty with this algorithm is that the user can't specify which line/column of the factorized matrices correspond to which target. To decide on the attribution of the obtained spectrograms to the targets the following process has been applied:

- Mark all targets as being unassigned.
- Repeat for all k:
    - Consider a spectrogram $\hat{S}_k = W_k H_k$.
    - Apply iSTFT to $\hat{S}_k$ to obtain the estimated waveform $\hat{s}_k$.
    - Compute the mdx SDR value for $\hat{s}_k$ with all the true targets that have not been assigned yet.
    - Assign $\hat{s}_k$ to the target producing the highest mdx SDR and remove it from the pool of available targets.

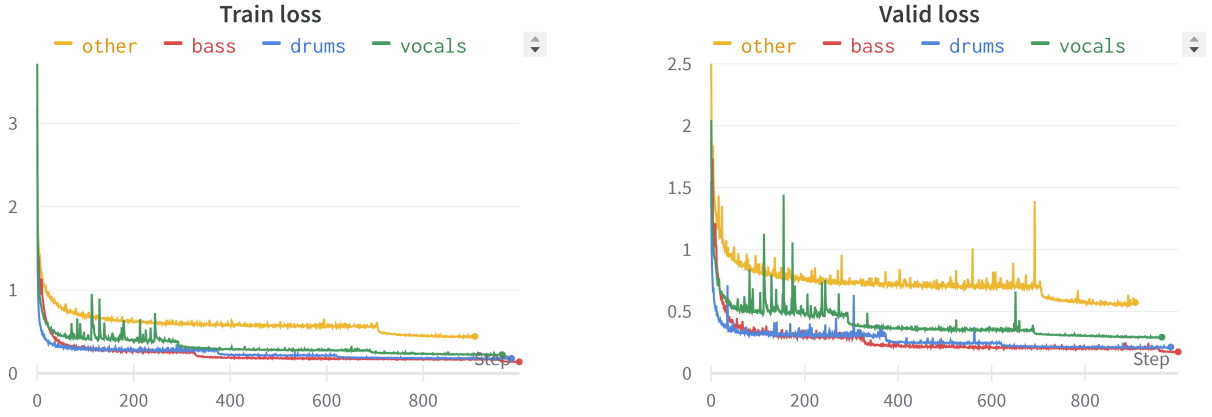
Once all the estimated waveforms have been assigned to a target, the museval metrics can be calculated and the medians across all segments are reported. However, the target assignment process described above is highly unreliable. Indeed, it was often observed that several spectrograms estimated from the same segment would be assigned to the same target if this target remained available.

Chapter 6

Results

6.1 OpenUnmix

Figure 6.1 presents the evolution of the MSE losses for every target, on both the training set and the validation set, during the training of OpenUnmix. We can observe a rapid convergence of all the losses in the first fifty epochs, however the improvement continues as demonstrated by the sudden drops in losses caused by the strong decaying of the learning rate by a factor of 0.3 in these iterations. Overall the training is quite stable except for *vocals* for which the losses present brief but significant spikes in their values.



(a) Train loss.

(b) Valid loss.

Figure 6.1: Evolution of the losses during the training of OpenUnmix.

As the inverse STFT requires a complex spectrogram to invert it back to a waveform and OpenUnmix generates real magnitude spectrogram, it was impossible to plot the evolution of the mdx SDR during training. Indeed, OpenUnmix relies on the Wiener filter to derive simultaneously for each target's estimated magnitude spectrogram, obtained out of the same input, corresponding complex filtered spectrograms that can be inverted back to the time domain.

The Table 6.1 displays the evaluation results for all targets, with the museval metrics and mdx SDR, of the newly trained OpenUnmix compared to their official equivalence on the MUSDB18 benchmark [32]. It is important to note that the nature and characteristics of the targets which are translated in the spectrograms make the losses evolve in different ranges

of value. It is not because the curve of one target lies below the curve of another that this target is better separated. Indeed, we can see in Figure 6.1 that the lowest curve is the one for *bass*, however, when looking at the Table 6.1, one can see that *bass* is only the third best separated target.

Benchmark	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
museval SDR	6.32	5.73	5.23	4.02	5.33
Experiments results	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
museval SDR	**6.37**	5.38	4.95	4.07	5.19
mdx SDR	**5.15**	4.85	4.43	4.35	4.69
SIR	**12.49**	12.15	8.58	7.61	10.21
SAR	6.12	5.73	**6.17**	4.38	5.6

Table 6.1: Metrics evaluation for the trained OpenUnmix models.

We can see that in terms of museval SDR performances, the retrained model is a very satisfying copy of the official one even if it is slightly less efficient on average. It shows that the SAD preprocessing coupled with the data simulation detailed in sections 4.2 and 4.3 allow to successfully train the OpenUnmix model to achieve very similar results to the ones found on the benchmark.

Furthermore, the SIR results translate few interferences from other sources in the final estimates, especially for *vocals* and *drums*. For the last metric, namely the SAR, the results are very decent but they imply however that the models introduce a small amount of unwanted sounds that were not present in the original sources but not enough for their power to rival that of the desired sources.

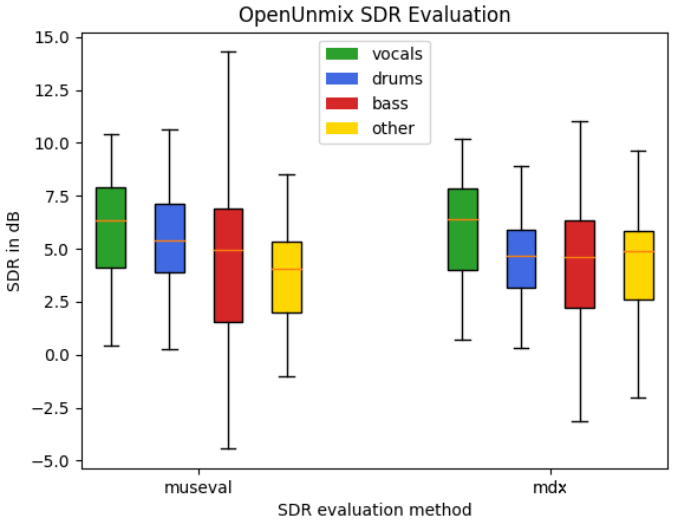


Figure 6.2: SDR evaluation boxplot for OpenUnmix.

Finally, when examining Figure 6.2, one can see that whether it be for museval SDR or mdx SDR, the performances are quite unstable and may vary a lot between songs, particularly the source *bass* which spans a range of museval SDR values between -4.44 and 13.56.

6.2 Band-Split RNN

6.2.1 STFT spectrograms

In Figure 6.3 we can see the evolution of the losses during the training of BSRNN with STFT spectrograms. It may seem like the valid losses do not evolve very much compared to the losses of OpenUnmix but each of them evolves in its own range and the improvements in the curve of a MAE loss is less steep than the improvement in the curve of a MSE loss. The improvements might be easier to perceive on Figures C.1, C.2, C.3 and C.4 in Appendix C. We can see in these plots a rapid decrease of these losses in the first 20 epochs before the validation loss stabilizes around the 50^{th} training iteration for *drums, bass* and *other* while the one for *vocals* converged later around the 70^{th} epoch. Further training could lead to overfitting and indeed the early stopping conditions were met shortly after.

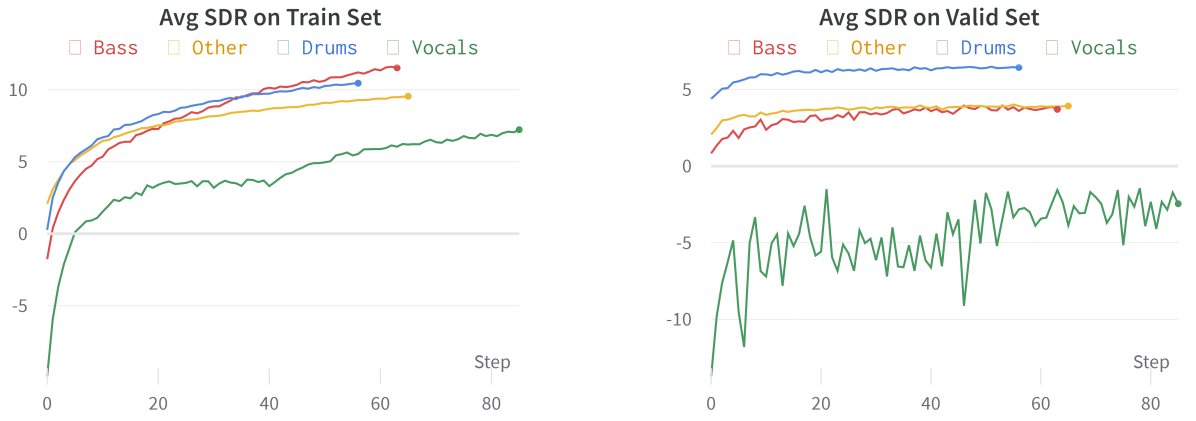
(a) Train loss.

(b) Valid loss.

Figure 6.3: Evolution of the losses during the training of BSRNN with STFT.

We can observe in Figure 6.4 the evolution of the average mdx SDR metric on both the valid set and the training set during training. The BSRNN producing directly complex spectrograms, the inverse STFT can be applied to derive the estimated waveforms and evaluate the fast mdx SDR. Even if the metric seem to increase properly with each epoch, the curve for the SDR on the valid set for *vocals* is surprisingly low, especially when comparing it to the evaluation results in Table 6.3. However there is a potential explanation to this. Indeed, the vocals are typically an instrument that can be silent in numerous parts of a track, like intros, solos or endings, when the rest of the instruments still emits high levels of energy. There are even tracks in MUSDB18, such as "PR - Happy Daze", in which the vocals are almost silent for their entire duration. Because the SAD preprocessing of the tracks retains segments based on their energy, which is evaluated on the entire mixture, it is highly probable that the retained segments include a non negligible amount of elements that are silent in voice. Let us consider a SAD preprocessed segment that is perfectly silent on vocals, not on the other instruments, and BSRNN tries to reproduce the vocals from this segment. Let's now imagine that the separation is perfect despite a slight interference from another instrument in only one time frame bringing an small error of 0.002. The mdx SDR for this configuration would be $10 * log_{10} \frac{\epsilon}{(0.002)^2+\epsilon} = -16.13$, with $\epsilon = 1e-07$, even though the estimation is almost perfect. Let us now imagine a similar scenario but the true vocals segment now contains only one non zero value of 0.02 which is perfectly reproduced by BSRNN but the interference is still

present. The mdx SDR would now be $10 * log_{10} \frac{(0.02)^2+\epsilon}{(0.002)^2+\epsilon} = 19.89$. This shows that the mdx SDR is highly unstable when it comes to silent or almost silent segments which would explain these low values for *vocals* during training. It could also explain the relative instability of the evolution of the valid SDR for *vocals* compared to the other sources. The curve is then helpful to depict the improvement of the separation but does not reflect the true quality of it.



(a) Train SDR.

(b) Valid SDR.

Figure 6.4: Evolution of the Music Demixing Challenge SDR during the training of BSRNN with STFT.

Table 6.2 helps to answer the question of the use of a Wiener filter during the evaluation. Indeed, we see that applying a Wiener filter seems to have a negative impact on the performances. The most important impacts are on *drums* and *bass* while *vocals* is very lightly affected and *other* has received a small improvement. This negative impact is further confirmed when looking at the full comparison table, Table D.1, and Figure D.1 in Appendix D where the SIR results show that the filter even introduce more interferences from other sources than when it is not applied, defeating its purpose. However, the SAR have very slightly increased but not enough to be of any significance.

It can be safely assumed that the authors did not apply Wiener filtering in their evaluation of the model. Hence the remaining of the results of BSRNN with both STFT and STDCT will only treat the case without the filtering.

museval SDR	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
no Wiener filter	7.03	6.56	6.1	4.41	6.02
Wiener filter	6.98	6.02	5.04	4.5	5.63

Table 6.2: Comparison of the evaluation of museval SDR with and without Wiener filtering.

The Table 6.3 contains the full results from the evaluation of BSRNN with STFT and no Wiener filter compared to the results present in the original paper. It is obvious from these results that the reproduction of the model failed to achieve performances as remarkable as the true model. However, this was expected, partly due to the fact that the authors trained their model on 10000 batches with a batch size of 2. In other words, they trained their model on twice as much data as this model. Processing this amount of data for training was not considered as the full training of the models already lasted around nine days and a half.

Paper results	Vocals	Drums	Bass	Other	Average
museval SDR	10.21	8.58	7.51	6.62	8.23
mdx SDR	9.92	8.68	6.77	5.97	7.84
Experiments results	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
museval SDR	**7.03**	6.56	6.1	4.41	6.02
mdx SDR	**7.31**	6.96	5.57	4.25	6.02
SIR	**17.64**	16.57	13.3	9.03	14.13
SAR	**6.9**	6.6	5.49	4.27	5.82

Table 6.3: Metrics evaluation for the trained BSRNN models with STFT.

Looking at the results for the SIR metric, we can notice that they are remarkably high, particularly so for *vocals* and *drums*. One can expect very little interference from other sources when separating a track with this model. Now considering the SAR metric, we observe satisfying results but it is expected to have introduced artifacts in the separations.

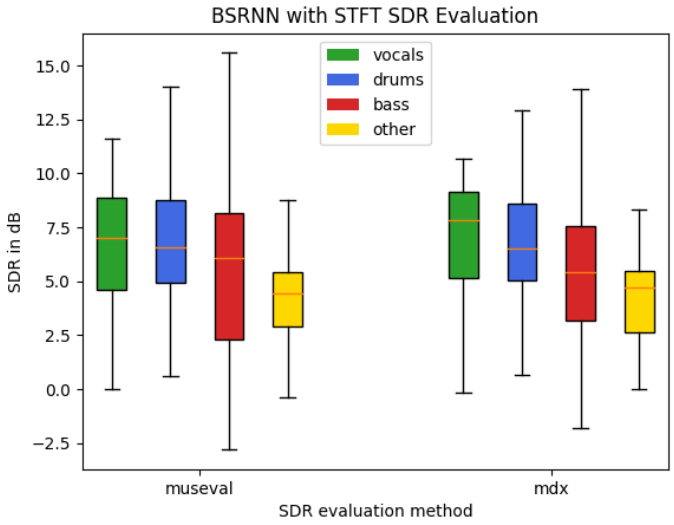


Figure 6.5: SDR evaluation boxplot for BSRNN with STFT.

Lastly, the boxplot in Figure 6.5 shows that results may vary extensively between songs but the obtained results all lie above the 0 threshold, except for *bass*. This means that when separating sources in a song, the power of the true signal in the resulting estimates should always be stronger than that of the noise.

6.2.2 STDCT spectrograms

In Figure 6.6 we can see the train and valid losses decrease during training of BSRNN with STDCT. The plain lines correspond to the models trained with the number of features N set to 64 while the dashed lines correspond to N set at 128. For a clearer appreciation of each source's curve we can refer to Figures E.1, E.3, E.5 and E.7. In these curves we can observe that the training losses seem to converge faster with 64 features and that they even converge at a higher loss than their 128 features counterparts. On the other hand the valid losses for $N = 64$ generally lie lower than the ones for $N = 128$. The fact that the configuration with fewer features seems to perform worse on the train loss, while simultaneously performing

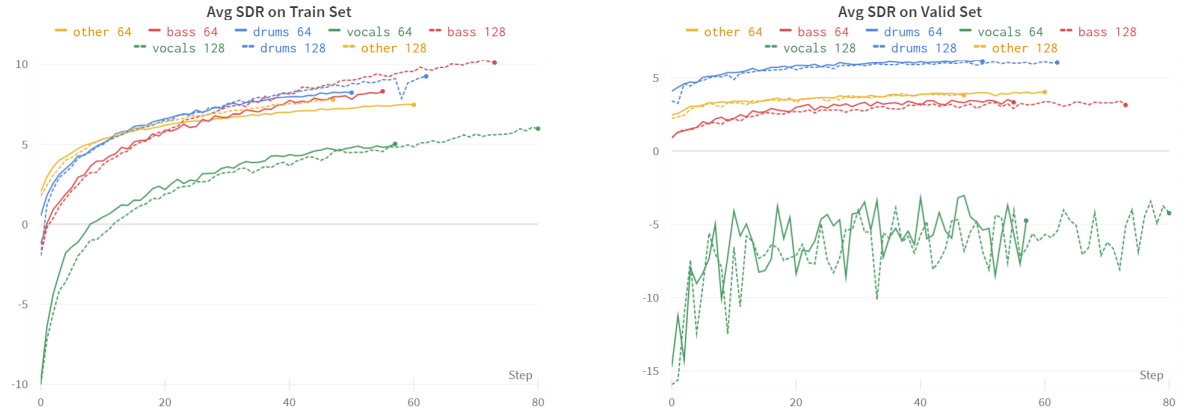
similarly or even better on the valid loss than the other configuration, indicates that it is less prone to overfitting. Indeed, the model with more features gets unnecessarily specialized on the training data while the other model generalizes better as proven by the valid losses. This indicates that that additional 64 features lead to a higher model complexity that is potentially superfluous.

(a) Train loss.

(b) Valid loss.

Figure 6.6: Evolution of the losses during the training of BSRNN with STDCT. The plain lines depict the models trained with $N = 64$ while the dashed lines depict the models trained with $N = 128$.

We can observe in Figure 6.7 the same problem that occured during the training of BSRNN with STFT where the average SDR on the valid set is surprisingly low, especially for *vocals*. The explanation to this event is the same as well. Similarly to the evolution of the losses described above, the curves for the mdx SDR seem to be worse on the training data when $N = 64$ while performing as well or better on the validation data. This is easier to observe on Figures E.2, E.4, E.6 andE.8. This observation brings additional weight to the overfitting when $N = 128$ hypothesis.



(a) Train SDR.

(b) Valid SDR.

Figure 6.7: Evolution of the Music Demixing Challenge SDR during the training of BSRNN with STDCT. The plain lines depict the models trained with $N = 64$ while the dashed lines depict the models trained with $N = 128$.

The Table 6.4 contains the entirety of the results for both $N = 64$ and $N = 128$. In this table one can see that the SDR results for *drums* and *other* have been improved when using fewer features while those for *bass* have remained almost identical. However it seems at first that the *vocals* result have deteriorated. But when looking at Figure E.1, it appears that the training for *vocals* with $N = 64$ has been stopped prematurely by the stopping conditions, and that the model could have improved further. Indeed, from the beginning of training since the point where the stopping conditions were met, the curve for the 64 features configuration lied below its more complex counterpart. It is very likely that it would have remained as such should the training have continued. On average, both the mdx SDR and museval SDR have been slightly improved by decreasing the number of features.

$N = 128$	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
museval SDR	**6.74**	6.12	5.9	4.01	5.69
mdx SDR	**7.18**	6.65	5.18	3.96	5.74
SIR	**16.11**	15.45	12.14	7.76	12.87
SAR	**6.61**	5.78	5.03	4.33	5.44
$N = 64$	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
museval SDR	**6.56**	6.51	5.84	4.33	5.81
mdx SDR	6.87	**6.88**	5.28	4.19	5.81
SIR	**15.84**	13.67	11.61	8.15	12.32
SAR	**6.38**	6.08	5.38	4.23	5.52

Table 6.4: Metrics evaluation for the trained BSRNN models with STDCT with the number of features $N = 128$ and $N = 64$.

However, the SIR scores for $N = 64$ have decreased except for *other*. It is then expected to find more interferences with this amount of features. On the other hand, the SAR, have been slightly improved.

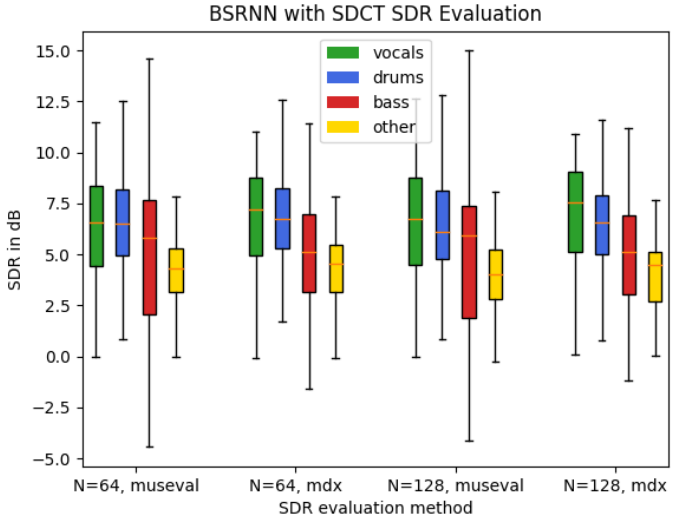


Figure 6.8: SDR evaluation boxplot for BSRNN with STDCT.

The boxplot in Figure 6.8 shows the same behaviour of the SDR metrics as was observed in Figure 6.5. Finally, it is quite an important fact to be pointed out that the processing time of the models is also greatly reduced by decreasing the amount of features used. Indeed, as said

in section 5.2.3, the number of features directly impacts other modules complexities within the entire model. Thus reducing the number features not only provides potentially better results but it also provides them in almost half the time.

6.3 Comparison of the models

In Table 6.5 the results for the different models are grouped by metric for comparison purposes. In this table we can notice, as expected, that the non DNN-based method, namely the NMF algorithm, is far less efficient for performing MSS in all metrics than the DNN models. Although the technique exhibits decent results in terms of SAR, which can be explained by the fact that the factorization of the mixture spectrogram into a frequency dictionary matrix W and a time activation matrix H is such that their product recreates the original spectrogram. Hence this technique, although very inefficient, should not introduce a significant amount of artifacts.

museval SDR	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
NMF algorithm	**0.09**	**0.75**	**-0.1**	**0.14**	**0.22**
OpenUnmix	6.37	5.38	4.95	4.07	5.19
BSRNN with STFT	**7.03**	**6.56**	**6.1**	**4.41**	**6.02**
BSRNN with STDCT, $N = 128$	6.74	6.12	5.9	4.01	5.69
BSRNN with STDCT, $N = 64$	6.56	6.51	5.84	4.33	5.81
mdx SDR	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
NMF algorithm	**-3.87**	**-1.76**	**-3.21**	**-1.36**	**-2.55**
OpenUnmix	5.15	4.85	4.43	4.35	4.69
BSRNN with STFT	**7.31**	**6.96**	**5.57**	**4.25**	**6.02**
BSRNN with STDCT, $N = 128$	7.18	6.65	5.18	3.96	5.74
BSRNN with STDCT, $N = 64$	6.87	6.88	5.28	4.19	5.81
SIR	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
NMF algorithm	**-2.96**	**0.99**	**-4.42**	**-3.15**	**-2.38**
OpenUnmix	12.49	12.15	8.58	7.61	10.21
BSRNN with STFT	**17.64**	**16.57**	**13.3**	**9.03**	**14.13**
BSRNN with STDCT, $N = 128$	16.11	15.45	12.14	7.76	12.87
BSRNN with STDCT, $N = 64$	15.84	13.67	11.61	8.15	12.32
SAR	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
NMF algorithm	**2.64**	**4.31**	**4.39**	**3.49**	**3.71**
OpenUnmix	6.12	5.73	6.17	4.38	5.6
BSRNN with STFT	**6.9**	**6.6**	**5.49**	**4.27**	**5.82**
BSRNN with STDCT, $N = 128$	6.61	5.78	5.03	4.33	5.44
BSRNN with STDCT, $N = 64$	6.38	6.08	5.38	4.23	5.52

Table 6.5: Full comparative table of the results obtained with each model. The results are separated by metric. The model achieving the best value of the metrics is highlighted in green for each source and the worse in red.

We can observe that every version of the BSRNN model surpasses the performances of the OpenUnmix model in all metrics, except for SAR with the versions of BSRNN working with STDCT spectrograms. Hence, although the reproduction of the original BSRNN is far from

perfect, it still outperforms the OpenUnmix baseline. The SIR results for example, indicate that one should expect less interferences from the separation perform by BSRNN than the one performed by OpenUnmix. This is confirmed by listening separated tracks with both OpenUnmix and BSRNN with STFT. The former model separated the song "*Side Effects Project - Sing With Me*"[1] while the latter separated the song "*BKS - Too Much*"[2]. Focusing on the vocals extraction of both songs, which both achieved a similar museval SDR score of 10.37 and 10.03 respectively, one can clearly hear constant remnants of other sources in the extracted vocals with OpenUnmix, especially interferences with the drums, while the vocals extracted form the second song with BSRNN is significantly cleaner with very few interferences. This higher score obtained by OpenUnmix compared with the lower perceived quality, though subjective, demonstrates the problem cited in Section 5.3.2 that the SDR does not necessarily correlates with the perceived performance of an audio evaluation.

The version of BSRNN with STFT spectrograms outperforms every other model in every metric. Using STDCT spectrograms instead of STFT ones as input seems to have a negative impact on the performances. However these models have been trained on exactly the same configuration as the version with STFT inputs, and they might require their own specific configuration and band-splitting schemes to train in an optimal manner. Furthermore, as said in the previous section, the training of the model extracting *vocals* with 64 features may have been stopped prematurely, thus making the obtained results for this source potentially not representative of the expected quality.

In all DNN models, the ordering of the separation quality seems to be the same, being: *vocals*, *drums*, *bass*, *other*. The source *other* being systematically the least well separated is probably due to its nature which is simply anything that is not *vocals*, *drums* or *bass*. This means that the instrumental content of this source can vary, sometimes containing guitars, sometimes including piano or both for example. Hence the characteristics in the corresponding spectrogram may change as well and make it harder for the model to extract this source.

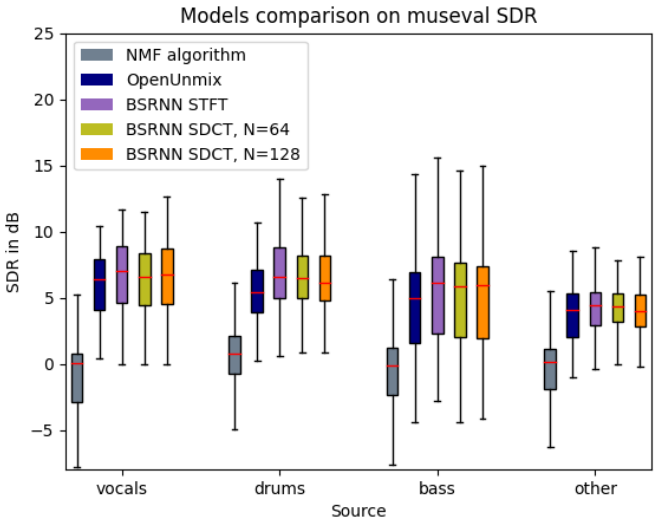


Figure 6.9: Comparison of the models on museval SDR.

[1]The audio samples for the 'good' separation with OpenUnmix can be found on bit.ly/3OHQFvg
[2]The audio samples for the 'good' sepration with BSRNN can be found on bit.ly/3E3nlKz

When observing Figure 6.9, we can see that all the models produce very varying results. For example, when listening to the separation of the song "Pr - Oh No"[3] produced by BSRNN with STFT, in particular the vocals extraction, we can directly notice the significant sparsity of the vocal interactions within the song. This majority of silent sections can cause problems when evaluating museval SDR as it reports the median across 1 second long segments. Any interference or artifact during a silent segment will produce a strongly negative SDR value for this segment. As there is very few active parts in the source to be accurately reproduced and bring positive contributions to the SDR evaluation, any imperfection will have a significant impact on the evaluation compared to very active vocal source. In this case, considering the mdx SDR evaluating directly the entire song, which for this one was a score of 3.31, seems more relevant as the separation, while being arguably unsatisfactory, produces a sample with the power of the true signal being perceptually stronger than the power of the noise.

[3]The audio samples for the 'bad' separation with BSRNN can be found on bit.ly/45G9F49

Chapter 7

Conclusion and Future Work

7.1 Future work

The first amelioration to bring in this project for BSRNN, both with STFT and STDCT, would be to thoroughly review the architecture and identify potential discrepancies with the official architecture. The band-splitting schemes should be revised as well, indeed the authors said themselves, referring to the *bass* source, that the band-split scheme needs further investigation to better capture low and mid-frequency range information.

Concerning the attempt at using real valued spectrogram from the STDCT, various hyper-parameter configurations must be envisaged and tried out to fully explore the potential of this alternative which presents the possibility to achieve similar results in far shorter time. Furthermore, the results seemed to show that BSRNN with STDCT trained with 64 features achieved better results with less overfitting in almost half the training time. But to be directly compared to the original version of BSRNN, we need to explore the impacts on training this version with a reduced amount of features as well.

Finally, taking inspiration from the improvements made to Hybrid Demucs to create the superior Hybrid Transformer Demucs, in which the BLSTM-based central encoder was replaced by transformer encoders, it has been envisaged to replace the BLSTMs within the *Band and Sequence Modelling* module of BSRNN with transformer encoders to improve the results. The concept of this new module is represented in Figure F.1. However the authors of Hybrid Transformer Demucs remarked that their new architecture performed poorly when trained on MUSDB18 alone and only outperformed its predecessor, Hybrid Demucs, trained on the same data, when using 800 extra training songs. Unable to curate such an amount of additional training tracks and the models training already being time-consuming, this idea has been abandoned for this piece of work but it represents a particularly interesting field of exploration.

7.2 Conclusion

During this project, we attempted to reproduce the state-of-the-art DNN model for Music Source Separation, namely Band-Split RNN, with as much fidelity as was possible. Even though the obtained reproduction did not achieve the exceptional results of its official counterpart, these results still outperformed the DNN baseline, OpenUnmix both on the official results and on the one that was retrained for this work.

We then explored an alternative transform of the input signal by using the Short Time Discrete Cosine Transform instead of the Short Time Fourier Transform. The derived real valued spectrograms hinted at the possibility of decreasing the model complexity compared to the original which used complex valued spectrograms and achieve similar results. We showed that decreasing the number of features when using this alternative is less prone to overfitting and potentially lead to increased performance compared to keeping the original complexity for the same transform. Finally, although the performance were slightly decreased, they remained similar to those of the more complex model using the Fourier transform. This alternative shows promising results that opens the possibility to achieve the same results in only half the processing time.

Bibliography

[1] E. Cano, D. FitzGerald, A. Liutkus, M. D. Plumbley, and F.-R. Stöter, "Musical source separation: An introduction," *IEEE Signal Processing Magazine*, vol. 36, no. 1, pp. 31–40, 2019.

[2] Y. Luo and J. Yu, "Music source separation with band-split rnn," 2022. [Online]. Available: https://arxiv.org/abs/2209.15174

[3] T. Sgouros, A. Bousis, and N. Mitianoudis, "An efficient short-time discrete cosine transform and attentive multiresunet framework for music source separation," *IEEE Access*, vol. 10, pp. 119 448–119 459, 2022.

[4] G. Federal Agencies Digital Guidelines Initiative. Term: Waveform (sound). [Online]. Available: https://www.digitizationguidelines.gov/term.php?term=waveformsound

[5] I. to Speech Processing. Waveform. [Online]. Available: https://speechprocessingbook.aalto.fi/Representations/Waveform.html

[6] P. L. a Pro. (2023) Choosing the sample rate for your project. [Online]. Available: https://producelikeapro.com/blog/sample-rate/

[7] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[8] Wikipedia. (2023) Nyquist frequency. [Online]. Available: https://en.wikipedia.org/wiki/Nyquist_frequency

[9] SlidePlayer. (2019) Pulse code modulation. [Online]. Available: https://slideplayer.com/slide/17073211/

[10] TheFourierTransform.com. (2010-2022) Fourier transforms. [Online]. Available: https://www.thefouriertransform.com/

[11] M. News. (2012) The faster-than-fast fourier transform. [Online]. Available: https://news.mit.edu/2012/faster-fourier-transforms-0118

[12] P. N. Methods. (2020) Discrete fourier transform. [Online]. Available: https://pythonnumericalmethods.berkeley.edu/notebooks/chapter24.02-Discrete-Fourier-Transform.html

[13] O. University. Discrete fourier transform. [Online]. Available: https://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf

[14] Numpy. (2022) numpy.hanning. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.hanning.html

[15] MathWorld. (2023) Leakage. [Online]. Available: https://mathworld.wolfram.com/Leakage.html

[16] ResearchGate. (2017) Real-time control of mobile robot using hmm-based speech recognition system. [Online]. Available: https://www.researchgate.net/publication/322157431_Real-Time_Control_of_Mobile_Robot_Using_HMM-based_Speech_Recognition_System/figures?lo=1

[17] MathWorks. (2023) Short-time fft. [Online]. Available: https://nl.mathworks.com/help/dsp/ref/dsp.stft.html

[18] "Representing audio," https://source-separation.github.io/tutorial/basics/representations.html.

[19] J. Qian, X. Liu, Y. Yu, and W. li, "Stripe-transformer: deep stripe feature learning for music source separation," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2023, 01 2023.

[20] D. Lee and H. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, pp. 788–91, 11 1999.

[21] Z. Benslimane. (2023) Audio source separation using non-negative matrix factorization (nmf). [Online]. Available: https://medium.com/@zahrahafida.benslimane/audio-source-separation-using-non-negative-matrix-factorization-nmf-a8b204490c7d

[22] A. Liutkus, Z. Rafii, B. Pardo, D. Fitzgerald, and L. Daudet, "Kernel spectrogram models for source separation," 05 2014.

[23] F. L. Soler, "Music source separation using deep neural networks," Master Thesis, Universitat Politècnica de Catalunya, 2020.

[24] F. Zalkow. (2013) Create audio spectrograms with python. [Online]. Available: https://www.frank-zalkow.de/en/create-audio-spectrograms-with-python.html

[25] E. Manilow, P. Seetharman, and J. Salamon, *Open Source Tools & Data for Music Source Separation*. https://source-separation.github.io/tutorial, 2020. [Online]. Available: https://source-separation.github.io/tutorial

[26] F. Lluís, J. Pons, and X. Serra, "End-to-end music source separation: is it possible in the waveform domain?" 2019.

[27] D. Stoller, S. Ewert, and S. Dixon, "Wave-u-net: A multi-scale neural network for end-to-end audio source separation," *arXiv preprint arXiv:1806.03185*, 2018.

[28] A. Défossez, N. Usunier, L. Bottou, and F. Bach, "Music Source Separation in the Waveform Domain," Apr. 2019, working paper or preprint. [Online]. Available: https://hal.science/hal-02379796

[29] A. Défossez, "Hybrid Spectrogram and Waveform Source Separation," Aug. 2021. [Online]. Available: https://arxiv.org/abs/2111.03600

[30] S. Rouard, F. Massa, and A. Défossez, "Hybrid transformers for music source separation," 2022. [Online]. Available: https://arxiv.org/abs/2211.08553

[31] S. Uhlich, M. Porcu, F. Giron, M. Enenkl, T. Kemp, N. Takahashi, and Y. Mitsufuji, "Improving music source separation based on deep neural networks through data augmentation and network blending," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 261–265.

[32] P. W. Code. (2018-2023) Music source separation on musdb18. [Online]. Available: https://paperswithcode.com/sota/music-source-separation-on-musdb18

[33] F.-R. St
"oter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, "Open-unmix - a reference implementation for music source separation," *Journal of Open Source Software*, 2019. [Online]. Available: https://doi.org/10.21105/joss.01667

[34] ——, "Open-unmix for pytorch," 2019-2021. [Online]. Available: https://github.com/sigsep/open-unmix-pytorch

[35] F.-R. Stöter, A. Liutkus, and N. Ito, "The 2018 signal separation evaluation campaign," in *Latent Variable Analysis and Signal Separation: 14th International Conference, LVA/ICA 2018, Surrey, UK*, 2018, pp. 293–305.

[36] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "The MUSDB18 corpus for music separation," Dec. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1117372

[37] S. datasets. (2022) Musdb18. [Online]. Available: https://sigsep.github.io/datasets/musdb.html#sisec-2018-evaluation-campaign

[38] ——. (2016) Dsd100. [Online]. Available: https://sigsep.github.io/datasets/dsd100.html

[39] R. M. Bittner, J. Wilkins, H. Yip, and J. P. Bello, "Medleydb 2.0: New data and a system for sustainable data collection," *ISMIR Late Breaking and Demo Papers*, p. 36, 2016.

[40] api.video. (2023) Aac (advanced audio coding). [Online]. Available: https://api.video/what-is/aac-advanced-audio-coding/

[41] Z. Rafii, A. Liutkus, F.-R. Stöter, S. I. Mimilakis, and R. Bittner, "MUSDB18-HQ - an uncompressed version of musdb18," Dec. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3338373

[42] I. to Speech Processing. Signal energy, loudness and decibel. [Online]. Available: https://speechprocessingbook.aalto.fi/Representations/Signal_energy_loudness_and_decibel.html

[43] M. I. Retrieval. Energy and rmse. [Online]. Available: https://musicinformationretrieval.com/energy.html

[44] J. Haag. (2021) pydct. [Online]. Available: https://github.com/jonashaag/pydct

[45] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1462–1469, 2010.

[46] F.-R. Stöter and A. Liutkus. (2019, Aug.) museval 0.3.0. [Online]. Available: https://doi.org/10.5281/zenodo.3376621

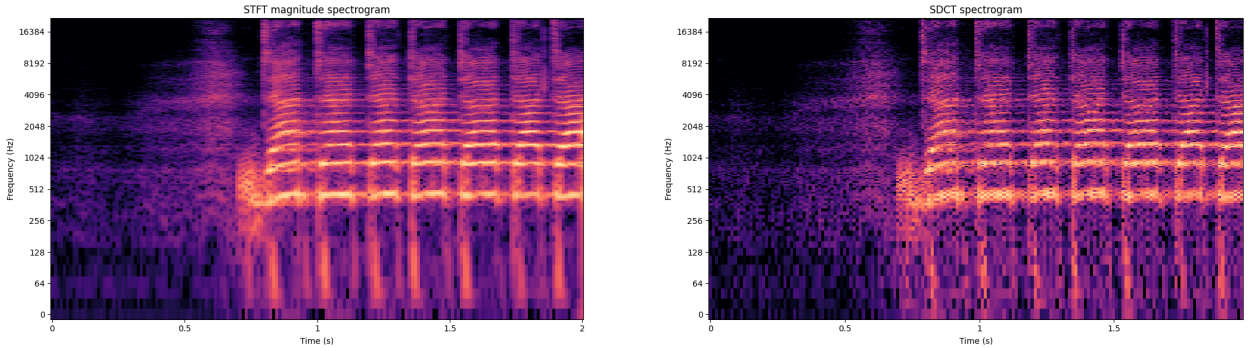
[47] Y. Mitsufuji, G. Fabbro, S. Uhlich, F.-R. Stöter, A. Defossez, M. Kim, W. Choi, C.-Y. Yu, and K. W. Cheuk, "Music demixing challenge 2021," *Frontiers in Signal Processing*, vol. 1, 01 2022.

[48] Z. Benslimane. (2017) Audio source separation using non-negative matrix factorization (nmf). [Online]. Available: https://github.com/ZahraBenslimane/SoundSourceSeparation_usingNMF

Appendix A

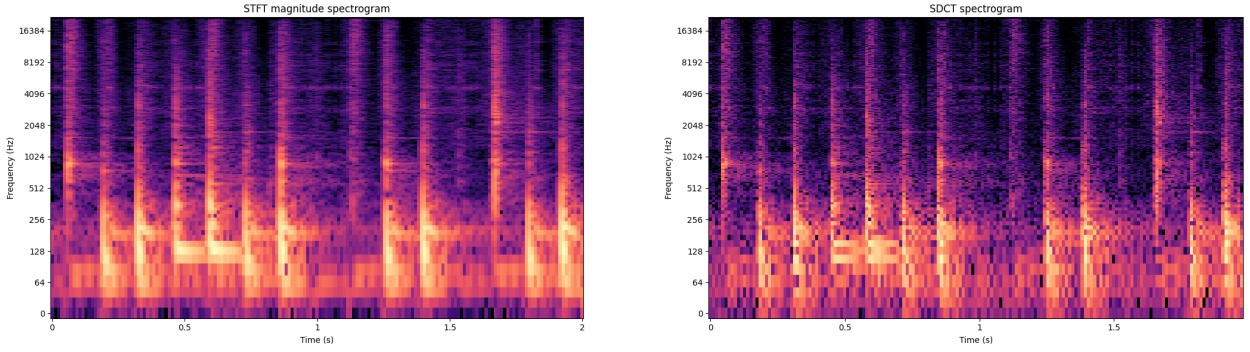
STFT and STDCT spectrograms comparison

The following spectrograms have all been created from the same 2 seconds long segment drawn arbitrarily from the song "*Clara Berry and Wooldog - Stella*" present in the test folder of MUSDB18.



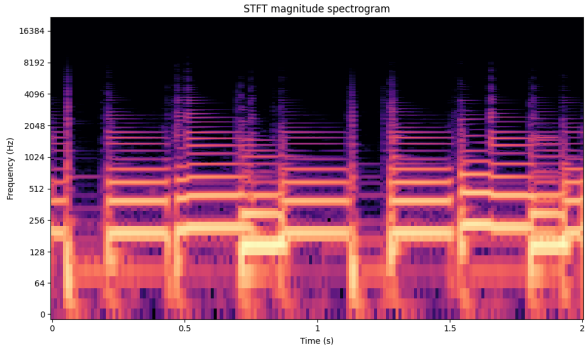
(a) STFT magnitude spectrogram. (b) STDCT spectrogram.

Figure A.1: Spectrograms of *vocals*.

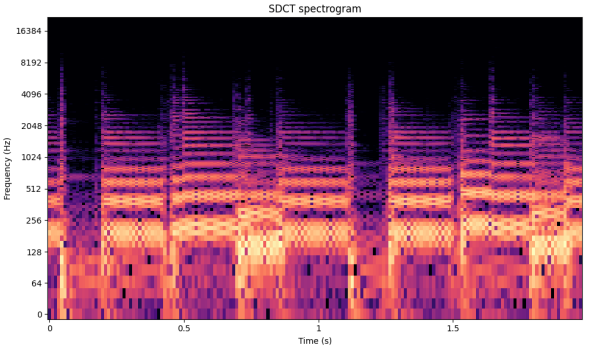


(a) STFT magnitude spectrogram. (b) STDCT spectrogram.

Figure A.2: Spectrograms of *drums*.

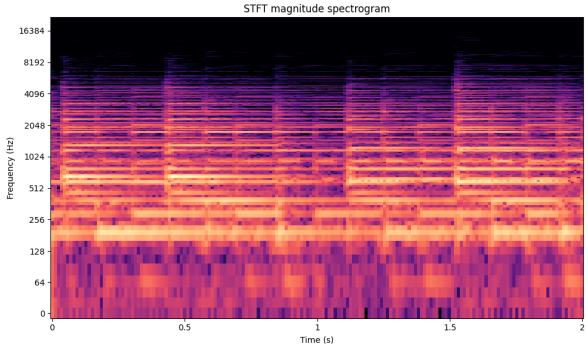


(a) STFT magnitude spectrogram.

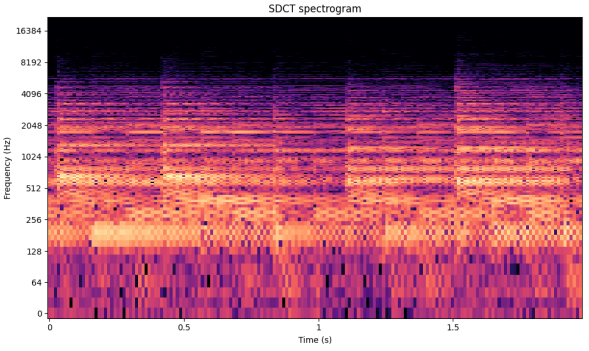


(b) STDCT spectrogram.

Figure A.3: Spectrograms of *bass*.



(a) STFT magnitude spectrogram.



(b) STDCT spectrogram.

Figure A.4: Spectrograms of *other*.

Appendix B

Estimated Sources Decomposition

This appendix is a description of the decomposition process of the estimated sources $\hat{s}_i$ described by Vincent et al. (2010) [45].

The estimated sources $\hat{s}_i$, $\forall i \in \mathcal{I} := \{B, D, O, V\}$, are each decomposed as:

$$\hat{s}_i = s_{target} + e_{interf} + e_{noise} + e_{artif}$$

Let's denote $\Pi\{y_1, ..., y_k\}$ the orthogonal projector onto the subspace spanned by the vectors $y_1, ..., y_k$. The projector is a $T \times T$ matrix, where T is the length of these vectors. We consider the three orthogonal projectors

$$\begin{aligned} P_{s_i} &:= \Pi\{s_i\}, \\ P_{\mathbf{s}} &:= \Pi\{(s_{i'})_{1 \leq i' \leq n}\}, \\ P_{\mathbf{s},\mathbf{n}} &:= \Pi\{(s_{i'})_{1 \leq i' \leq n}, (n_j)_{1 \leq j \leq m}\}. \end{aligned}$$

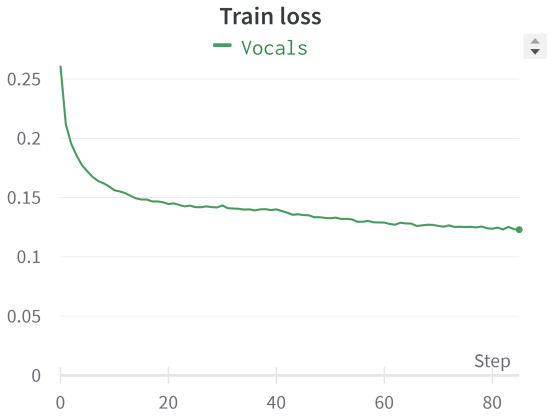
And we decompose $\hat{s}_i$ as the sum of the four terms

$$\begin{aligned} s_{target} &:= P_{s_i} \hat{s}_i \\ e_{interf} &:= P_{\mathbf{s}} \hat{s}_i - P_{s_i} \hat{s}_i \\ e_{noise} &:= P_{\mathbf{s},\mathbf{n}} \hat{s}_i - P_{\mathbf{s}} \hat{s}_i \\ e_{artif} &:= \hat{s}_i - P_{\mathbf{s},\mathbf{n}} \hat{s}_i \end{aligned}$$

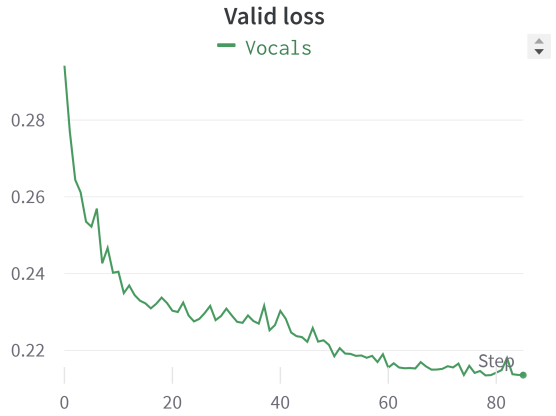
The calculation of s_{target} is simple since it only requires a basic inner product: $s_{target} = \langle \hat{s}_i, s_i \rangle s_i / ||s_i||^2$. The computation of e_{interf} is a little more complicated. If the sources are orthogonal to each other, then $e_{interf} = \sum_{i' \neq i} \langle \hat{s}_i, s'_i \rangle s'_i / ||s'_i||^2$. Otherwise, if we use a vector $\mathbf{c}$ of coefficients such that $P_{\mathbf{s}} \hat{s}_i = \sum_{i'=1}^{n} \bar{c}_{i'} s_{i'} = \mathbf{c}^H \mathbf{s}$, then $\mathbf{c} = {\mathbf{R}_{\mathbf{ss}}}^{-1} [\langle \hat{s}_i, s_1 \rangle, ..., \langle \hat{s}_i, s_n \rangle]^H$, where $\mathbf{R}_{\mathbf{ss}}$ is the Gram matrix of the sources defined by $(\mathbf{R}_{\mathbf{ss}})_{ii'} = \langle s_i, s'_i \rangle$. The computation of $P_{\mathbf{s},\mathbf{n}}$ is similar, however most of the time we may assume that the noise signals are mutually orthogonal and orthogonal to each other, so that $P_{\mathbf{s},\mathbf{n}} \hat{s}_j \approx P_{\mathbf{s}} \hat{s}_j + \sum_{i=1}^{m} \langle \hat{s}_i, n_i \rangle n_i / ||n_i||^2$.

Appendix C

BSRNN with STFT individual training losses

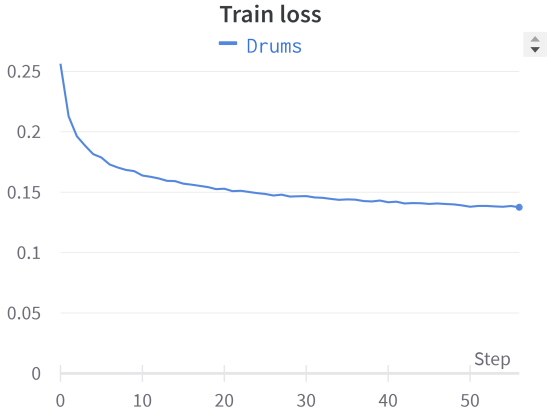


(a) Train loss.

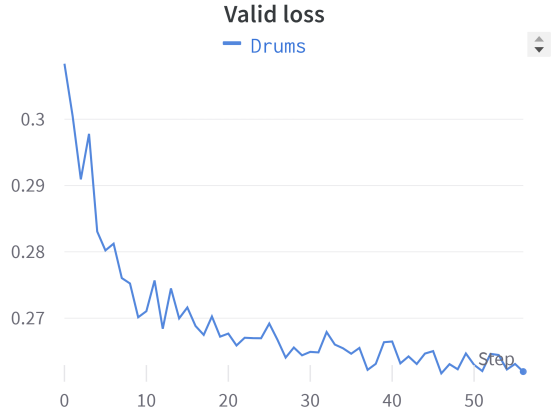


(b) Valid loss.

Figure C.1: Losses evolution of BSRNN with STFT for *vocals*. Source: Martin Laurent.



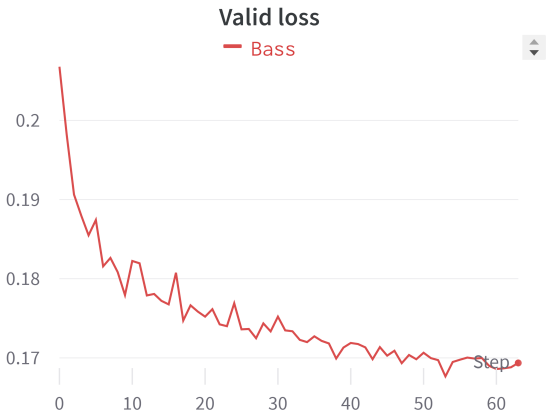
(a) Train loss.



(b) Valid loss.

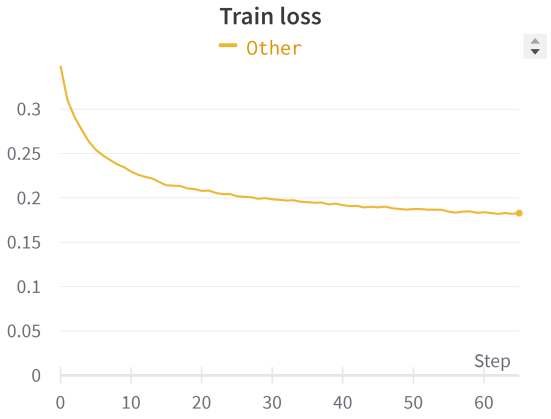
Figure C.2: Losses evolution of BSRNN with STFT for *drums*. Source: Martin Laurent.

(a) Train loss.

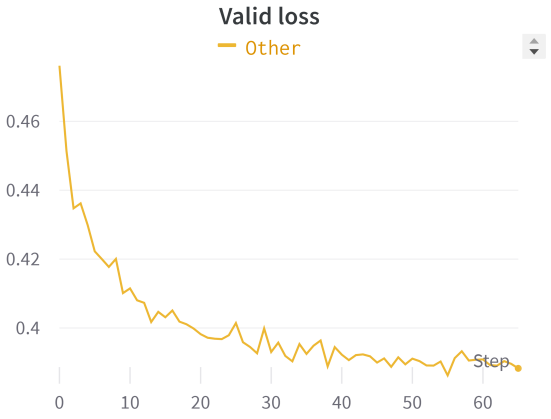


(b) Valid loss.

Figure C.3: Losses evolution of BSRNN with STFT for *bass*. Source: Martin Laurent.



(a) Train loss.

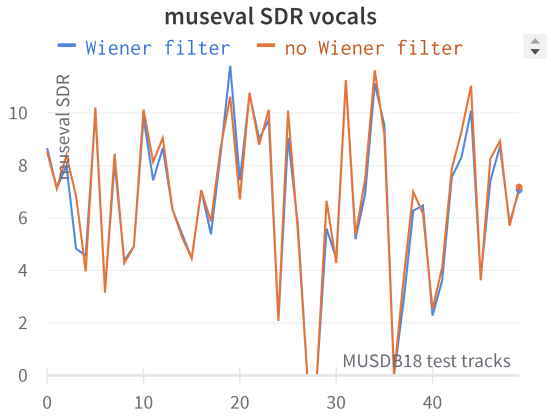


(b) Valid loss.

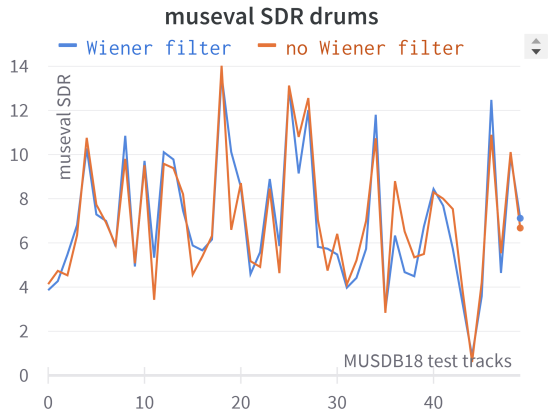
Figure C.4: Losses evolution of BSRNN with STFT for *other*. Source: Martin Laurent.

Appendix D

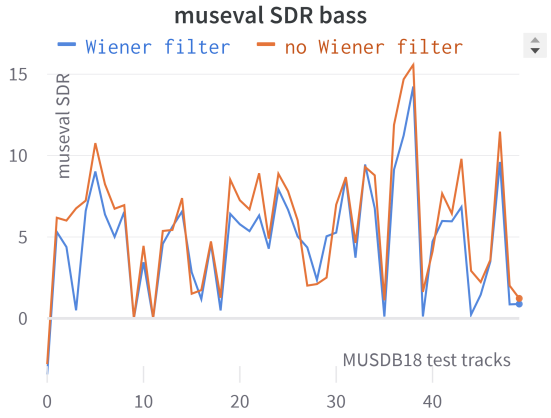
Evaluation of BSRNN with and without Wiener filter



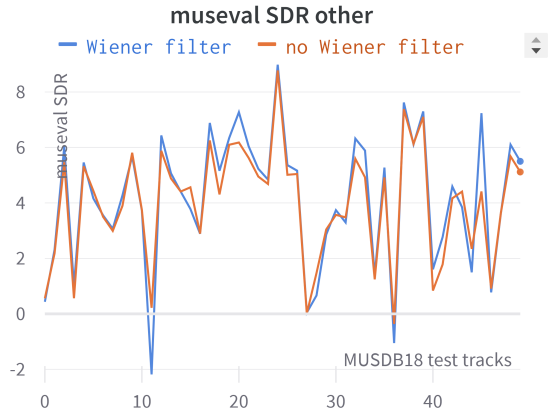
(a) Vocals.



(b) Drums.



(c) Bass.



(d) Other.

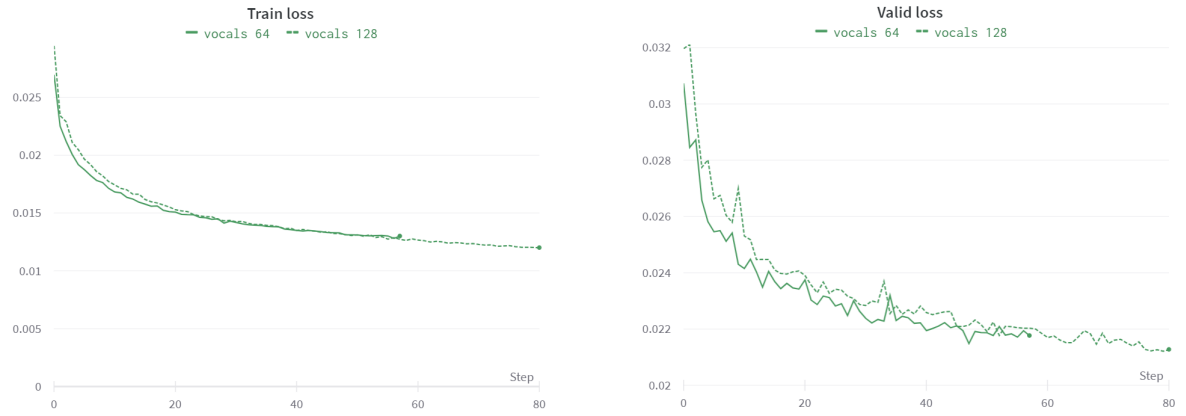
Figure D.1: Comparison of museval SDR metric on MUSDB18 test folder with and without Wiener filter. Source: Martin Laurent.

no Wiener filter	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
museval SDR	7.03	6.56	6.1	4.41	6.02
mdx SDR	7.31	6.96	5.57	4.25	6.02
SIR	17.64	16.57	13.3	9.03	14.13
SAR	6.9	6.6	5.49	4.27	5.82
Wiener filter	**Vocals**	**Drums**	**Bass**	**Other**	**Average**
museval SDR	6.98	6.02	5.04	4.5	5.63
mdx SDR	7.16	6.99	4.65	4.4	5.8
SIR	16.04	12.95	9.41	8.26	11.67
SAR	6.82	5.84	5.81	4.96	5.86

Table D.1: Comparison of the different metrics with and without Wiener filtering.

Appendix E

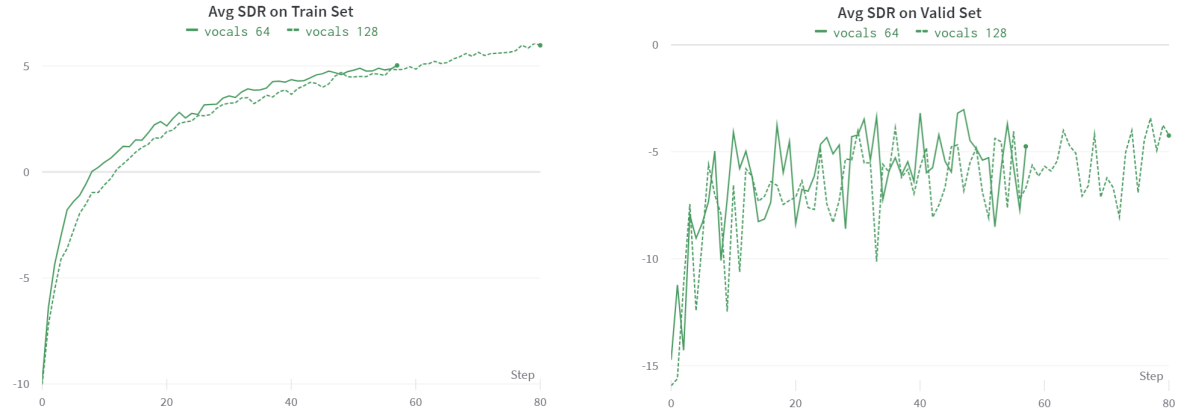
BSRNN with STDCT individual training losses and average SDR



(a) Train loss.

(b) Valid loss.

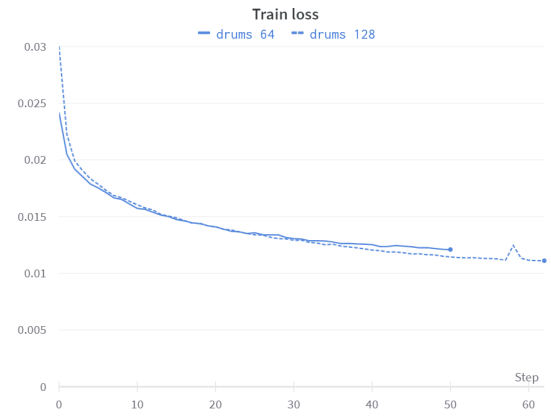
Figure E.1: Losses evolution of BSRNN with STDCT for *vocals*. Source: Martin Laurent.



(a) Train average SDR.

(b) Valid average SDR.

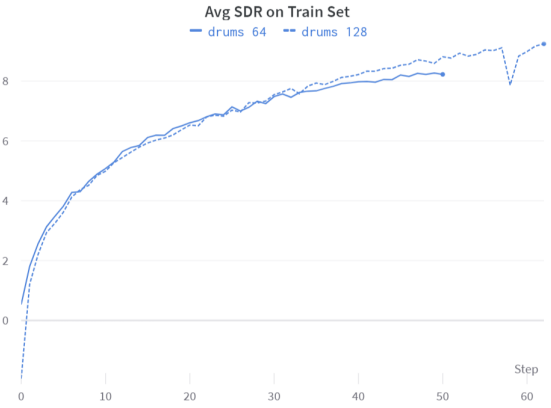
Figure E.2: Average SDR evolution of BSRNN with STDCT for *vocals*. Source: Martin Laurent.



(a) Train loss.

(b) Valid loss.

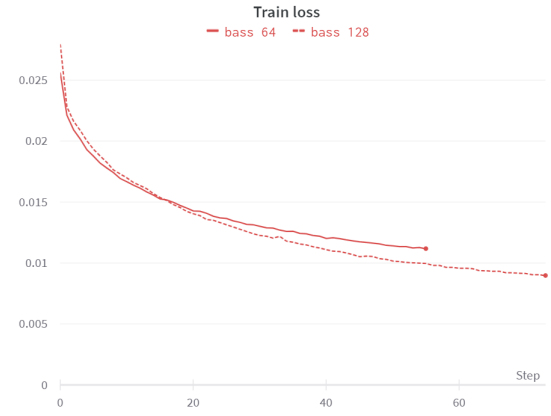
Figure E.3: Losses evolution of BSRNN with STDCT for *drums*. Source: Martin Laurent.



(a) Train average SDR.

(b) Valid average SDR.

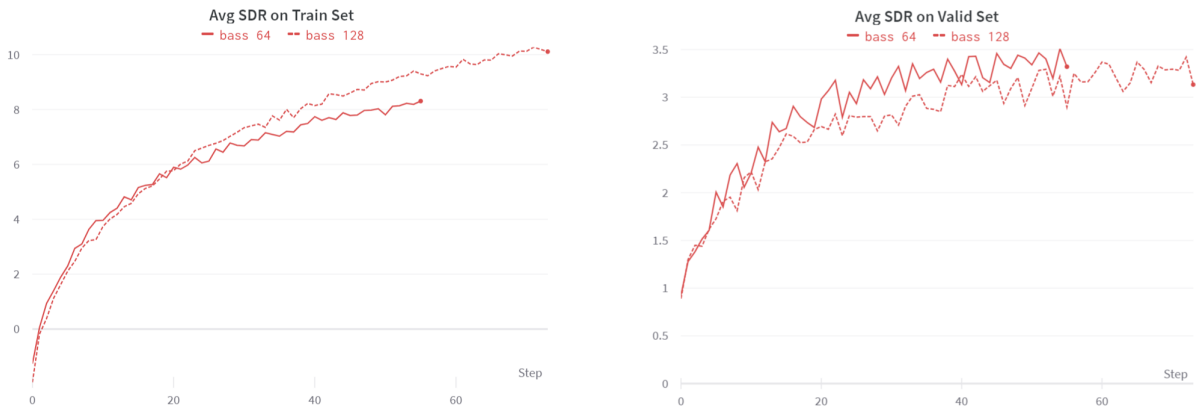
Figure E.4: Average SDR evolution of BSRNN with STDCT for *drums*. Source: Martin Laurent.



(a) Train loss.

(b) Valid loss.

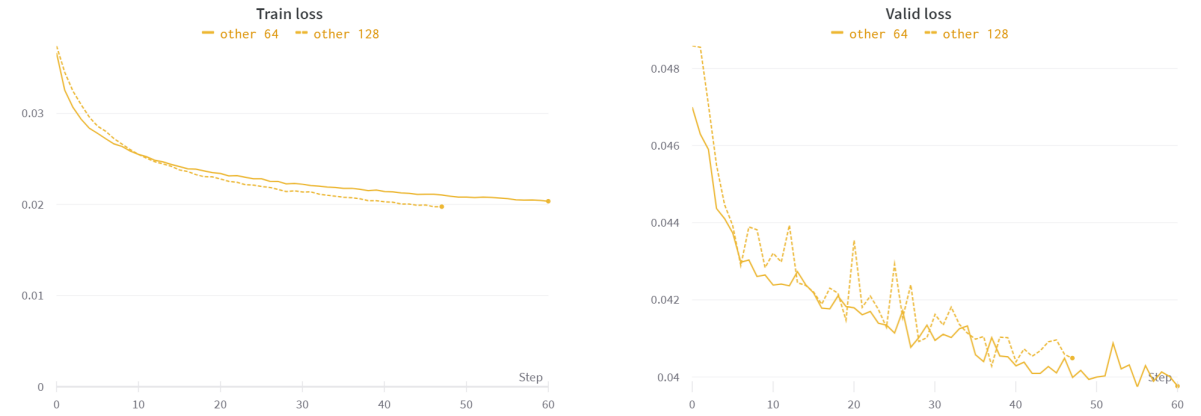
Figure E.5: Losses evolution of BSRNN with STDCT for *bass*. Source: Martin Laurent.



(a) Train average SDR.

(b) Valid average SDR.

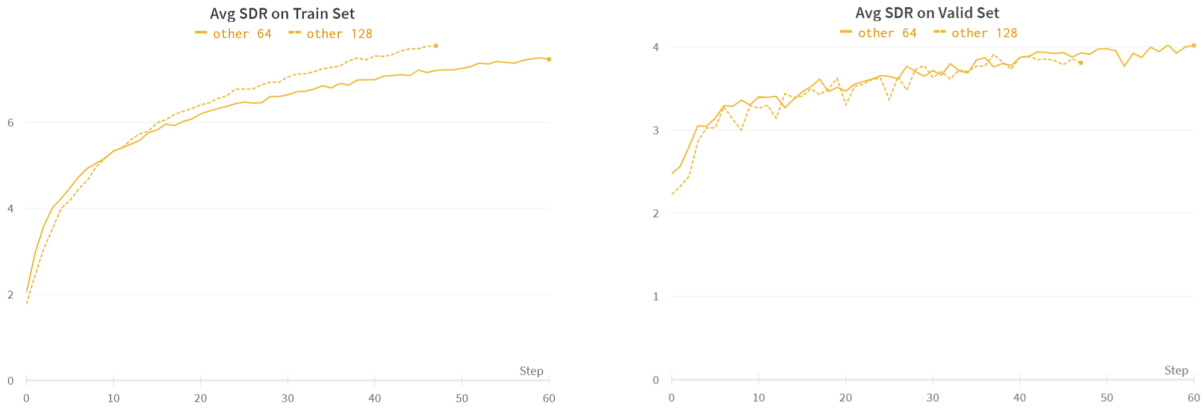
Figure E.6: Average SDR evolution of BSRNN with STDCT for *bass*. Source: Martin Laurent.



(a) Train loss.

(b) Valid loss.

Figure E.7: Losses evolution of BSRNN with STDCT for *other*. Source: Martin Laurent.



(a) Train average SDR.

(b) Valid average SDR.

Figure E.8: Average SDR evolution of BSRNN with STDCT for *other*. Source: Martin Laurent.

Appendix F

Attempt at BSTransformer

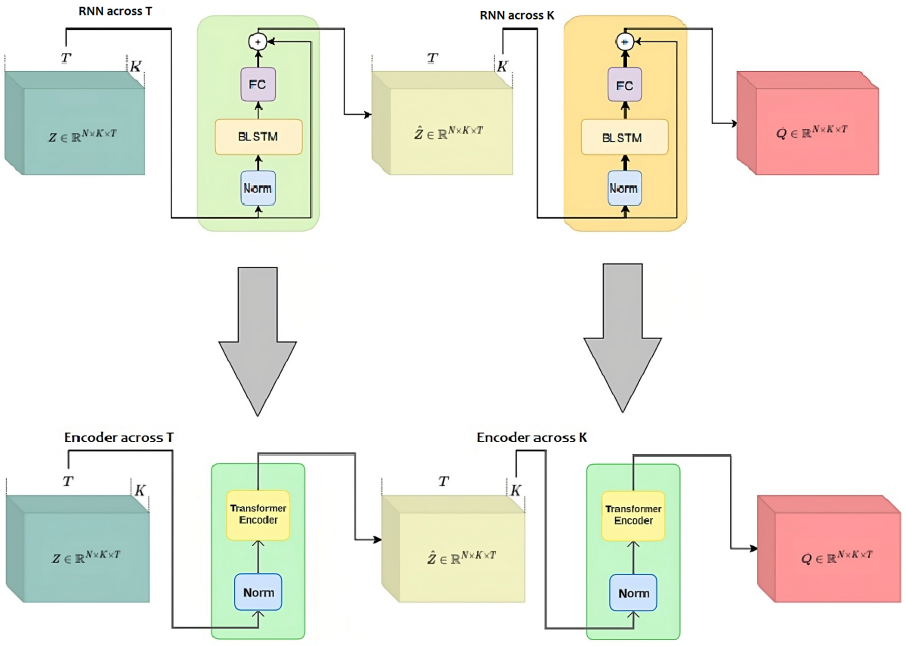


Figure F.1: Changes made to the Band an Sequence Modeling Module. Source: Martin Laurent.