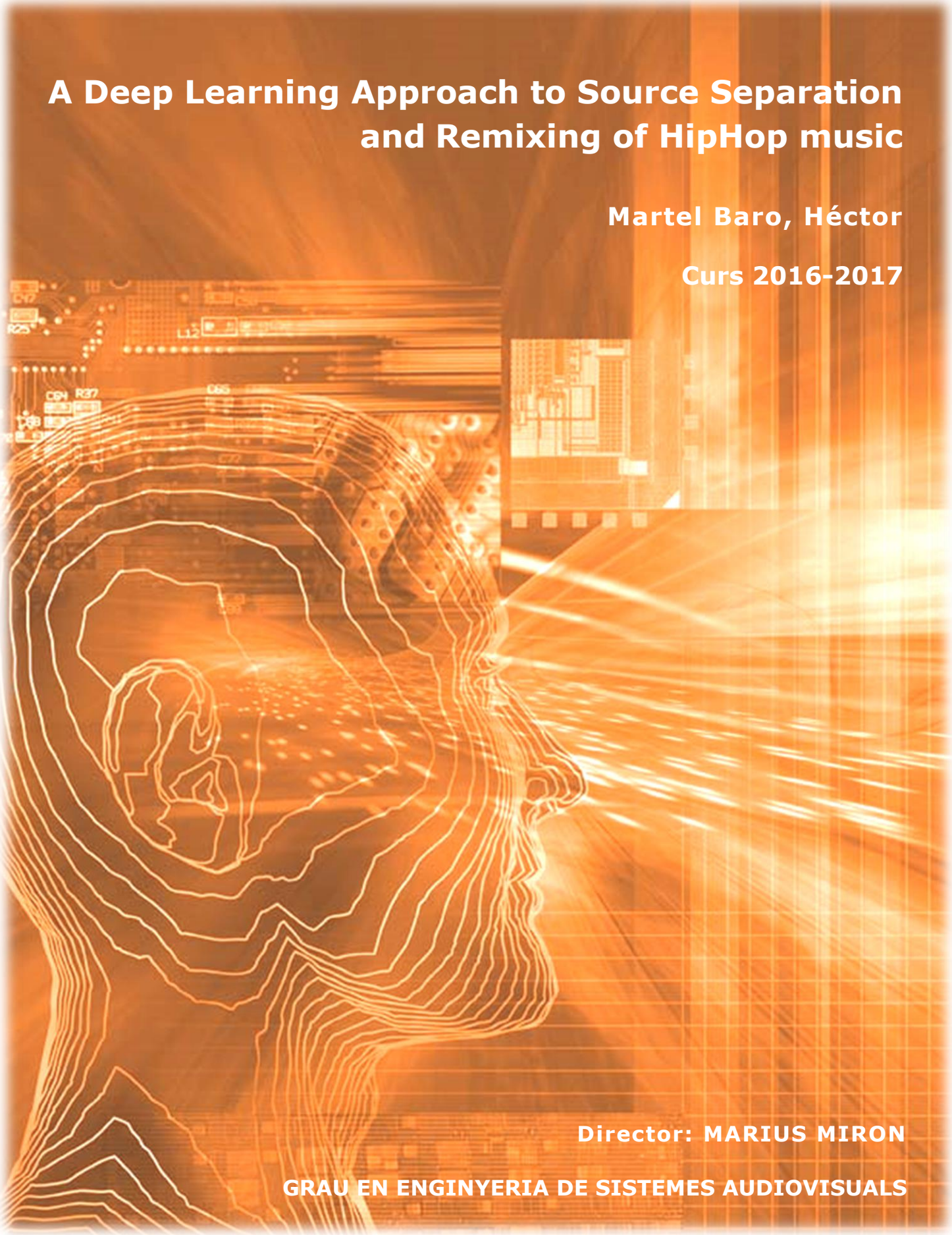
A Deep Learning Approach to Source Separation and Remixing of HipHop music

Martel Baro, Héctor

Curs 2016-2017



Director: MARIUS MIRON

GRAU EN ENGINYERIA DE SISTEMES AUDIOVISUALS

Universitat Pompeu Fabra Barcelona
Escola Superior Politècnica

Treball de Fi de Grau

A Deep Learning Approach to Source Separation and Remixing of HipHop music

Héctor Martel Baro

UNDERGRADUATE THESIS

Degree in Audiovisual Telecommunication Systems Engineering

ESCOLA SUPERIOR POLITÈCNICA UPF

Academic year: 2016-2017

THESIS SUPERVISOR

Marius Miron

A mis padres, por haber despertado mi curiosidad y apoyarme para que nunca dejara de soñar.

Acknowledgements

The idea of getting involved in the Audio Source Separation task described in this project comes from an anonymous person who one day asked me to change the instruments in an audio mixture during a live performance. At that time I simply answered that it was not possible, but this incident made me curious about the possibility of obtaining separated tracks from an audio mixture. Without that apparently innocent query, these pages would have never been written.

I would like to thank Marius Miron, my supervisor, for the time and dedication he has put into this project, especially during the last months of the execution phase. He has always been open to share ideas and to contribute with very helpful feedback. I also appreciate the effort made by the Music Technology Group (MTG) for providing me with the necessary knowledge, people, and resources to turn this idea into a reality. I felt integrated as one more member of the group, since they also gave me the opportunity to attend to some workshops and network with other researchers.

From inside and outside of the University, thanks to all my relatives, friends and foes that have either supported my work or made me learn the most valuable lessons along the way. I would like to thank each and every person who made me improve and who believed in me in one way or another, even in the moments I did not.

ABSTRACT

Audio source separation has been one of the major research fields in audio processing during the past years. The main purpose of this discipline is to decompose a mixture signal into simpler components called sources, which applied to professionally produced music means to recover the instrument tracks. This is achieved by undoing the mixing process, which might vary depending on the music genre. In this work, an existing framework based on Deep Neural Networks will be adapted to the particularities to HipHop music and tested with a proposed dataset. The aim is to determine whether or not it is a suitable approach to be implemented in a remixing application. For this reason, objective and subjective quality tests are performed to evaluate the resulting separations.

The isolated instrument tracks can be used for many different purposes, specially focusing on the following two aspects. First, each instrument can be placed in a 3D space so that the song can be reproduced not only in stereo, but in more complex configurations for which the song was not initially produced. Second, all the instruments can be controlled separately to edit the song, thus, enabling the possibility to remix it. Consequently, further work on this subject will make it possible for musicians and producers to manipulate mixed songs in new powerful ways to create their content.

Keywords: source separation, deep learning, music processing, 3D audio, upmixing, remixing

RESUMEN

La separación de fuentes de audio ha sido uno de los campos de investigación principales en procesamiento de audio durante los últimos años. El objetivo principal de esta disciplina es descomponer una señal mezclada en componentes más simples llamados fuentes, lo cual aplicado a música producida de forma profesional implica recuperar los instrumentos en pistas. Esto se consigue deshaciendo el proceso de mezcla, que puede variar dependiendo del género musical. En este estudio, un método existente basado en Deep Neural Networks se adaptará a las particularidades de la música HipHop y se evaluará con un conjunto de datos propuesto. La finalidad es determinar si es o no un método apropiado para ser implementado en una aplicación de remezcla. Por este motivo, se realizan pruebas objetivas y subjetivas de la calidad para evaluar los resultados de las separaciones.

Las pistas aisladas se pueden utilizar para diferentes propósitos, especialmente centrándose en los siguientes dos aspectos. Primero, cada instrumento puede ser colocado en un espacio 3D de manera que la canción pueda reproducirse no solo en stereo, sino en configuraciones más complejas para las que no fue producida inicialmente. Segundo, todos los instrumentos se pueden controlar independientemente para editar la canción y, por tanto, brinda la posibilidad de remezclarla. De esta manera, trabajo posterior en esta área posibilitará a músicos y productores la manipulación de canciones mezcladas de nuevas maneras para crear su contenido.

Palabras clave: Separación de fuentes, Deep learning, procesamiento de música, audio 3D, upmixing, remezcla

摘要

音频源分离是近年来音频处理领域的主要研究课题之一，主要目的在于将混合信号分解成其更简单的组成部分——
源信号。该项研究在专业音乐制作中的乐器轨道恢复方面具有重要的价值，其应用主要依据音乐的不同类型，通过撤消混合来实现。本文中，我们将采用一种基于“深层神经网络”的现有框架，根据嘻哈音乐的特性对其进行调整，并结合相应的数据集进行测试，以确定其是否适用于混音的制作，由此，客观和主观的质量测试在分离结果的测评中均得到使用。

独立的乐器轨道具有有多种功能，主要如下：1、每个乐器都可以放置在3D空间中，这样，歌曲不仅可以立体声重现，而且还可以在更加复杂的配置中以新的形式再生。2、所有的乐器都可以单独控制，以编辑歌曲，从而使重新混音成为可能。因此，该领域的进一步研究将为音乐人和制作人的混合音乐创作提供更强大的途径。

关键词：源分离　深度学习　音乐处理　3D音频　上混　混音

Table of contents

List of Figures

List of tables

1. INTRODUCTION

1.1. Motivation

Summing audio tracks to create a professional music production is apparently a trivial process from the technical point of view. It only involves some gain coefficients and linear algebraic operations. However, recovering the original sources that are present in a given mixture is not such an easy task to perform, at least for a computer. This task is commonly known as Source Separation [1]. Human listeners can do Source Separation with ease and in real-time, for example when having a conversation in a noisy environment, known as the cocktail party effect [2].

Source Separation is useful in cases of remixing in which it is required to suppress a particular instrument or place it in a different spatial position in the mix without altering the others [3]. Under these constraints, and without the application of a source separation method, making a remix of a song is currently limited to the artists or producers who have the project with all the separated tracks to work with. Otherwise only minor changes can be made to the original song. Moreover, the need of altering a song can happen in a live setting where latency plays a crucial role in the overall performance.

Another important consideration is the 3D audio part, also called spatial audio [4]. With the increasing popularity of 3D audio systems for professional and home users, there is also an increasing demand for upmixing solutions. Some proposals can be found in [3], [5], [6]. The upmixing process consists of obtaining a render for all the channels in the system from the original mixture, so the total number of channels is greater at the output. Standard commercial music is typically distributed in Stereo 2.0 format, which is composed of two audio channels, but reproduction systems can have a larger number of channels. Therefore, the rest must in inferred by a rendering process in order to reproduce the track.

Signal Source Separation is a research field that tries to tackle the above mentioned and many other problems, using different approaches to separate the individual components of a mixed signal. Then, the separated tracks can be used for other purposes. There are methods that rely on prior information, which are called informed, and others which outcome is entirely determined by the mixture itself, which are called blind [1]. It can be thought that implementing blind methods is a better solution in a real scenario, but the performance that can be achieved is not so optimal because the side information helps to obtain better results.

Most of the current advances in Source Separation for music applications have been developed based on specific genres, especially classical music. One remarkable example is the PHENICX project [7]. Even though urban and popular music are quite widespread among the potential users, there are only a few studies focused on HipHop source separation. The application proposed in this work consists of a system that can be easily trained with a few songs from a producer. Then, it can be used to separate and remix songs of a similar style in a different reproduction layout.

There are some particularities of HipHop that make it different from other genres. For instance, the voice is not sung, while carrying rhythmic patterns. Therefore, existing

singing voice source separation strategies consisting on following the pitch would not work well. The drums and the bass can be acoustic, synthesized or sampled from vinyl records of other pieces, which makes the variability of the sources very high. The song structure is based on loops except for the voice, so repetitions along the song can be exploited like in [8] to separate HipHop music.

In addition, most advances in Source Separation have been made on informed methods. Consequently, this work is an opportunity to apply some existing state of the art Blind Source Separation methods to this genre and discover whether more refinement is required or not to obtain an acceptable performance for a remixing application.

1.2. Objectives

This work is intended to cover the process of listening to a HipHop song in a 3D Audio reproduction system and allow the user to change parameters of the individual sources. In order to accomplish these goals, a source separation and a rendering stage are combined.

This work will be mostly focused on the source separation stage because it is considered to be the major challenge of the two. For the testing phase of this project, a proposed dataset has been elaborated containing a compilation of HipHop songs of diverse styles. The purpose of this is to verify the implementation and release the dataset to the public for further research.

From the usability point of view, the user of the final system should be able to take a HipHop song as input and obtain the separation within a reasonable waiting time and with a quality as close to the original mix as possible. The use case contemplated here is, for instance, that a HipHop producer can train a specialized model using a reduced set of songs from a well-defined production style. Then, this model should be able to separate similar songs with low latency performing better than a generic model.

The proposed system is intended to be based on a personal computer, but it would be interesting to embed it inside a spatial audio renderer.

To mention a few, the objectives of this study are the following:

- Explore source separation methods in use
- Develop a methodology for HipHop source separation
- Elaborate and test different models
- Apply spatial audio techniques to the sources
- Reduce the quality loss of the result as much as possible
- Reduce processing time by adding metadata and parallel computation
- Develop a graphical interface (service on the web)

1.3. Organization

The structure of this document consists of 6 content sections apart from the introductory pages.

A theoretical framework is introduced at the beginning under the name of State of the Art. In this part, the relevant advances in the fields of Source Separation and 3D Audio are covered in order to establish the relationship between the main concepts involved in both disciplines and their implications in this work.

Then, the Methodology used in this particular case is explained more in detail based on the methods mentioned in State of the Art. The description covers the explanation of the method in use and its actual implementation for both the source separation and the spatial audio stages.

Finally, the Evaluation of the system and the Conclusions are presented. The results are analyzed and discussed together with the problems found during the experiments, and possible solutions to those issues are commented. At the end, some ideas are exposed as a proposal for further development and work on this matter.

2. STATE OF THE ART

2.1. What is source separation?

Source separation is a digital signal processing problem that consists on isolating the individual components or sources from a mixed signal. The mixed signal is typically an additive combination of the different components, so that the objective is to recover the components separately. The signal can be an audio recording, an image or any other type of signal that implies a combination of sources.

When several sources are present simultaneously, the resulting signal in time domain is simply the superposition of all the sources. Let the sources be denoted by $s_i(n)$ where the subindex i goes from 1 to M, being M the total number of sources, and each source has an amplitude denoted by a_i. The total duration is N and n represents the time index. The expression for the mixture signal $x(n)$ would be as follows:

$$x(n) = \sum_{i=1}^{M} a_i \cdot s_i(n) \qquad n = 1, \dots, N \tag{2.1}$$

Therefore, the problem consists on recovering each $s_i(n)$ given $x(n)$. From a practical perspective it is not always required to separate all of them, but just some sources in particular which are the most relevant ones for the application at hand.

The definition given in Eq. (2.1) is the most simplified mathematical model for source separation, since linearity is assumed. In a real environment there are delays and reverberations, so a more general way to reformulate the problem is to add a delay term δ_{ij} which results from the convolution of the sources with the associated impulse response $h_j(n)$.

$$x(n) = \sum_{i=1}^{M} h_j(n) \ * \{a_i \cdot s_i(n)\} \qquad n = 1, \dots, N \tag{2.2}$$

From the properties of the convolution, the expression (2.2) can be rewritten as follows. Note that the amplitude coefficient has changed from a_i to a_{ij} denoting that it has been multiplied by the amplitude of the response.

$$x(n) = \sum_{j=1}^{K_i} \sum_{i=1}^{M} a_{ij} \cdot s_i\left(n - \delta_{ij}\right) \qquad n = 1, \dots, N \tag{2.3}$$

The human brain can perform this operation with outstanding quality and in real time. For instance, in the case of audio, it is manifested when dealing with speech and noise, two different speeches or a musical instrument with noise. Also, a trained listener can distinguish between various musical instruments playing together. However, according to the definition given above another problem arises. It can be argued that the definition of a sound source $s_i(n)$ is ambiguous since human perception is different from actual physics. According to physics, every object producing vibrations on its own should be considered as an independent source, but there are occasions in which human listeners

tend to perceive a section of instruments as a single source rather than as separated entities. This can happen when they are all the same instrument and play the same pitch.

Several approaches have been proposed in order to enable computers to perform the task of source separation, but the current results are still behind human perception capabilities [9]. This work is going to be focused on source separation applied to musical signals, in which a fairly acceptable separation can be achieved from a stereo recording. Monoaural recordings, however, present a greater challenge because the similarities between channels cannot be used as side information [1], [10]. Throughout the following pages the most relevant approaches will be classified and discussed.

2.1.1. The role of prior information

The design of a suitable source separation method greatly depends on the information about the sources that is available previously. This additional information means more data to be processed, but it also reduces the complexity of the problem significantly. Having prior information is an advantage because the algorithms can be optimized according to some constraints. Assumptions are generally based on the number of instruments, harmonic and rhythmic patterns of specific music genres, statistical resemblances between instruments, or even the partial or the entire score [1], [11].

When there is previous information the method is said to be informed *(Informed Source Separation, ISS),* whereas when there is very little or no information the method is called blind *(Blind Source Separation, BSS).* In this study, it is assumed that the music genre is always HipHop, in such a way that the tempo range, the song structure, and some instrument categories are implicitly determined. In the Methodology described in Section 3, the songs are treated using a supervised and timbre-informed method. The system not fully blind, although it is trained based on the separated tracks in the dataset with no other previous assumption.

2.1.2. Quality requirements

Source separation methods can be applied for different purposes depending on the specific task. There are cases in which the sound quality of the separated sources is crucial whereas in other situations like Foreground/Background Separation [8] it may not be necessary to fully recover all the individual sources. The quality may not be very relevant if the estimation of the sources is an intermediate result that will be further processed. According to this criterion, two categories can be clearly differentiated [9].

- Audio Quality Oriented (AQO)

  The objective is to recover the sources with the highest possible sound quality. As a consequence, the computing cost is also higher as filtering operations need to be performed in greater orders and artifacts must be minimized. Audio quality oriented approaches are mostly used for music applications such as post-production, remixing, upmixing, unmixing or hearing aids [3], [12].

- Significance Oriented (SO)

The important information is not strongly conditioned by the quality of the audio. In this case, the quality of the separated sources has just to be enough to facilitate the semantic analysis of a complex signal. Therefore, significance oriented approaches are not as demanding in terms of performance. The main applications are music information retrieval, polyphonic transcription and object-based audio coding [1], [9].

2.1.3. Evaluation of Source Separation

Given a set of separated tracks that have been computed using a specific source separation method, it is not an easy task to evaluate the performance in objective terms. Human listeners can agree on how well isolated the tracks are from each other or on whether the distortion introduced by the separation is acceptable according to their listening experience. However, the problem is to express it using a mathematical model.

By simplifying the mixing model to be a linear combination of the sources plus some added noise, it is possible to measure the separation, as proposed in [13]. Let $s_i(n)$ be the target source and $\hat{s}_i(n)$ be its estimation, from a set of M sources, and $\varepsilon(n)$ be the noise added to the mix. The target mix $x(n)$ and the estimated mix $\hat{x}(n)$ can both be expressed as a summation where the coefficient a_i represents the gain for the source s_i .

$$x(n) = \sum_{i=1}^{M} a_i \cdot s_i(n) + \varepsilon(n) \qquad n = 1, \dots, N \tag{2.4}$$

The similarity between the two sources can be expressed as a normalized subtraction of $\hat{s}_i$ and s_i. In this case, the simplest perceivable measure is to take the square of the normalized difference as follows:

$$D = \min_{\epsilon=\pm 1} \left\| \frac{\hat{s}_i}{\| \hat{s}_i \|} - \epsilon \frac{s_i}{\| s_i \|} \right\|^2 \tag{2.5}$$

Given that the difference is squared, the value of D will always be positive except for the perfect reconstruction of the source $\hat{s}_i = s_i$, case in which it will be zero. Even though it might seem that this measure of similarity performs well, it does only when the estimation is close enough to the target source. When the sources are orthogonal the expected value would be $\pm\infty$, but the expression above is limited to a maximum of 2. This happens because of the normalization of the sources and the condition of D being the minimum argument.

Another situation that causes the expression from (2.5) to be inaccurate is when there is noise or distortion in the estimation that is somehow correlated with the target source. Under this condition the value obtained would be low, but the amount of noise or distortion may be undesirable depending on the application at hand. For example, this situation would not be acceptable for the purpose of this work since the audio tracks are intended to be used in a remixing application, where quality is important. However, other applications such as speech recognition may allow some amount of distortion.

The approach in [13] proposes some measures for evaluation of blind source separation. To calculate them the estimate source must be decomposed into these four terms:

$$\hat{s}_i = s_{target} + e_{interf} + e_{spat} + e_{artif} \tag{2.6}$$

The target source s_{target} is a modified version of the original source s_i, in which some distortion denoted by $\mathcal{F}$ is allowed. The interference term e_{interf} represents the amount of other sources present in $\hat{s}_i$ due to the fact that the isolation is not perfect and, thus, the other sources might be mixed along with $\hat{s}_i$. e_{spat} represents the spatial distortions resulting from the comparison between the spectrogram images and e_{artif} refers to any other forbidden distortion or noise that is not contained in the set of $\mathcal{F}$.

It is important to notice that the values of s_{target}, e_{interf}, e_{spat} and e_{artif} may vary over time. It is suggested that their values are calculated locally by windowing the signal and then a global value is obtained from the local data using statistical measures. Once these four terms are computed, the following measures can be obtained:

- Source to Distortion Ratio

The overall performance of the source separation method can be expressed by taking the ratio between the target source and sum of all the unwanted distortions denoted by $e_{interf} + e_{spat} + e_{artif}$. This measure is directly related to how close the estimation is respect to the target.

$$\mathrm{SDR} = 10\log_{10}\frac{\left\|s_{target}\right\|^2}{\left\|e_{interf} + e_{spat} + e_{artif}\right\|^2} \tag{2.7}$$

- Source to Interferences Ratio

The amount of isolation achieved can be expressed as the ratio between the target source and the interferences present in the estimation.

$$\mathrm{SIR} = 10\log_{10}\frac{\left\|s_{target}\right\|^2}{\left\|e_{interf}\right\|^2} \tag{2.8}$$

If there are little interferences the value of the SIR will be high. However, notice that it does not imply that the estimation is well reconstructing the target source. In fact, there is a tradeoff between the isolation of a source and the distortions or artifacts introduced in the separation.

- Image to Spatial distortion Ratio

$$\mathrm{ISR} = 10\log_{10}\frac{\left\|s_{target} + e_{interf}\right\|^2}{\left\|e_{spat}\right\|^2} \tag{2.9}$$

- Source to Artifacts Ratio

The amount of artifacts present in the estimation respect to the other components can be expressed as:

$$\mathrm{SAR} = 10\log_{10}\frac{\left\|\mathrm{s_{target}} + \mathrm{e_{interf}} + \mathrm{e_{spat}}\right\|^2}{\|\,\mathrm{e_{artif}}\|^2} \tag{2.10}$$

The artifacts are the only components taken into account in this measure, so the situation is the opposite as in SIR. If the artifacts are low, then the interferences will be high.

2.2. Independent Component Analysis (ICA)

Independent Component Analysis is a statistical technique that is used within the framework of Blind Source Separation to separate a mixture signal into additive subcomponents [14], [15]. The idea is to project the data from its time representation into a new set of axes determined by some statistical properties of the signal.

Let X be the mixture signal and S a vector of N additive components of X. The weight matrix W is the full transformation from the original space to the space found by the algorithm. It can be expressed as the following multiplication:

$$\mathrm{S} = \mathrm{W} \cdot \mathrm{X} \tag{2.11}$$

There are some assumptions that must hold for this method to work. First, the subcomponents must be non-Gaussian, meaning that their value at any time t is not determined by a Normal distribution. Also, they must independent from each other since the method relies on the decorrelation property to perform the separation of the sources.

The axes should also be statistically independent from each other, even though some approaches are valid under non-orthogonality. Moreover, the axes are determined dynamically for each signal to be analyzed, and will be different for every piece. Once the axes have been found they are separated and inverted in order to extract the sources present in the input signal. This is achieved by performing an N-dimensional matrix rotation that minimizes the Gaussianity of the projection over all N dimensions.

2.3. Non-negative Matrix Factorization (NMF)

The basic idea behind NMF is to decompose the mixed signal X into matrices, two in most cases, which multiplication approximately reconstructs X [16]. These matrices have non-negative elements and are typically referred to as basis (B) and activation gains (G). The expression of the matrix factorization is simply:

$$\mathrm{X} \cong \mathrm{B} \cdot \mathrm{G} \tag{2.12}$$

NMF is commonly the method of choice when the data is presented as magnitude or power spectrograms, and it does not need the sources to be statistically independent [17]. It has been proven that non-negativity is sufficient for obtaining a separation that well approximates the sources.

Let X be the spectrogram of the audio signal. Then, the matrices B and G are the frequency and the time dimensions respectively. Thus, B represents the frequency response at a given time and is vertical because it evolves along the frequency axis. In contrast, G represents the gain of those frequencies at any instant of time, so that is horizontal and goes along the time axis. Considering M frequency bins and N time indices, the product can be expanded:

$$\begin{bmatrix} X_{11} & \cdots & X_{1N} \\ \vdots & \ddots & \vdots \\ X_{M1} & \cdots & X_{MN} \end{bmatrix} \cong \begin{bmatrix} B_1 \\ \cdots \\ B_M \end{bmatrix} \cdot [G_1 \quad \cdots \quad G_N] \tag{2.13}$$

The main assumption of NMF is that the sources that are present in the mixture can be expressed as a linear combination of the basis. The following expression is applied for K sources:

$$X_{i,j} = \sum_{k=1}^{K} B_{i,k} \cdot G_{k,j} \quad i = 1, \dots, M \;\; j = 1, \dots, N \tag{2.14}$$

In order to separate the sources from the mixture and guarantee that they reconstruct the mixed signal, the divergence between X and the product $B \cdot G$ must be minimized. This divergence, denoted by D can be any cost function such as the Euclidean Distance or the Kullback-Leibler Divergence, for example.

$$\{B, G\} = argmin_{B,G \geq 0} D(X, BG) \tag{2.15}$$

In practice, this expression is calculated iteratively to find the minimum value starting from randomly initialized values, and updating the matrices B and G according to the gradient of the cost function. The process continues either until a number of iterations has been reached or until the value of the divergence falls below a certain threshold.

2.4. Neural Networks

Neural Networks are used for many machine learning approaches in different fields such as Natural Language Processing [18], Image Processing [19] or Speech Recognition [20], and have recently gained popularity among researchers. The main idea behind Neural Networks is to build a computational model of artificial neurons emulating the principles of the human brain. Neural Networks adapt their output to the training data they are exposed to, and this process is called learning. The use of Neural Networks is considerably powerful in situations when some features of the data are not easy to represent in a conventional way.

Each neural unit behaves as an analogy of a biological one. This means that it performs the summation of all the inputs and its output is fed to other units. There are interconnections between units that can either enhance or inhibit the activation response, and every input is weighted. Learning is achieved by updating the value of the weights on every connection in order for the output to fit the training data as accurately as possible.

Mathematically the neural units can be described as having a vector X of inputs $\{x_1, \dots, x_n\}$, a bias value b which is added as an offset to the elements in X, an output a and a series of weights $W = \{w_0, \dots, w_n\}$ that are element wise multiplied with b and X.

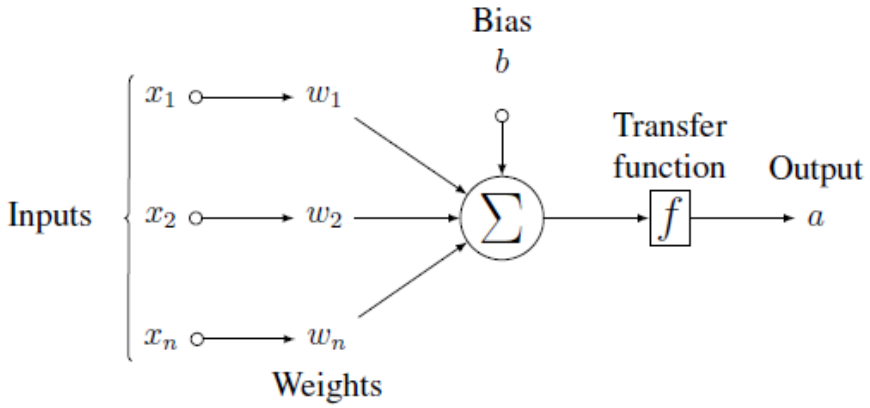
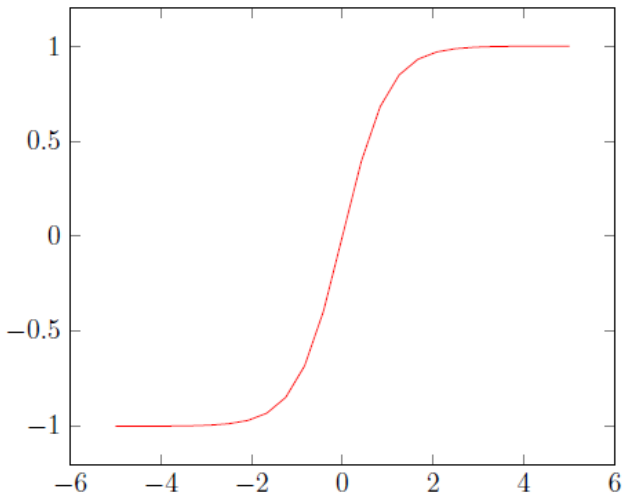


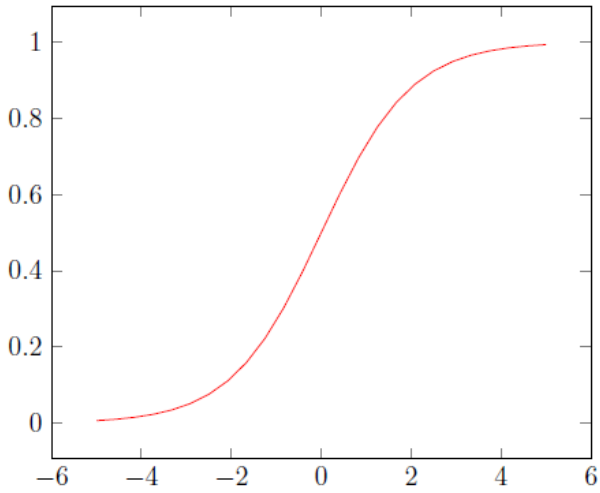
Figure 2.4.1: Mathematical model of a neural unit. Extracted from [21].

The representations can be linear or non-linear depending on the transfer function f used to model each neural unit. It must be mentioned that the data is linearly separable in very limited contexts only, so Neural Networks are typically designed based on a non-linear transfer function to achieve a more generic purpose. However, for some applications a combination of linear and non-linear layers can be used. Among the most common transfer functions there are the hyperbolic tangent, the sigmoid or the linear rectified.

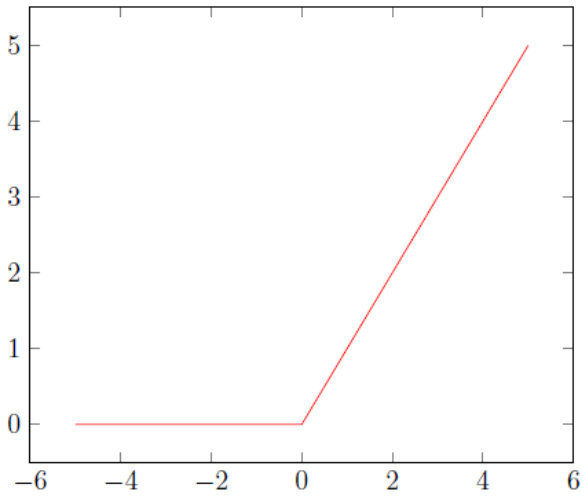
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.16}$$



$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$



$$relu(x) = \max(0, x) \tag{2.18}$$



The relation of the output a and the inputs x_i, the weights w_0, the bias $b = x_0 w_0$ and the transfer function σ can be written as:

$$a = \sigma\left(\sum_{i=0}^{N} x_i \cdot w_i\right) = \sigma\left(x_0 \cdot w_0 + \sum_{i=1}^{N} x_i \cdot w_i\right) = \sigma\left(b + \sum_{i=1}^{N} x_i \cdot w_i\right) \tag{2.19}$$

As research in neuroscience advances, some new patterns are introduced, but the most common arrangement consists of three basic layers.

- Input layer

- Hidden layer

- Output layer

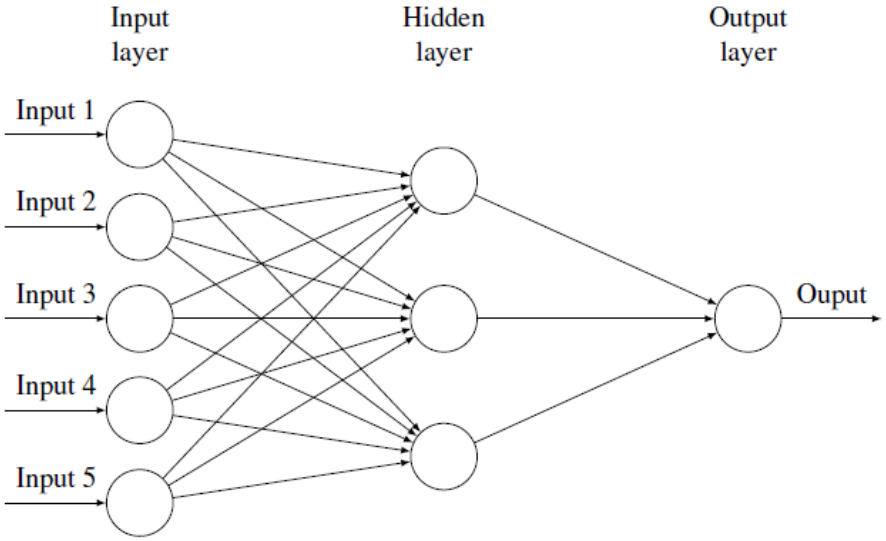


Figure 2.4.2: Representation of a Neural Network composed by an Input layer, one Hidden layer and an Output layer. Considering two consecutive layers, there are links between any pair of nodes from one layer to the other. Extracted from [21].

2.4.1. Deep Neural Networks (DNN)

This is a particular case of the Neural Network explained above, which has more than one hidden layer. The advantage of this arrangement is the possibility of obtaining more abstract representations of data since it has to go through several layers. Therefore, Deep Neural Networks can infer patterns of higher degrees of complexity such as Audio Source Separation.

The presence of more hidden layers allows for using a combination of transfer functions which may differ from one layer to another. Each layer can have an arbitrary number of inputs and outputs, which depend on their interconnection with the neighboring layers.

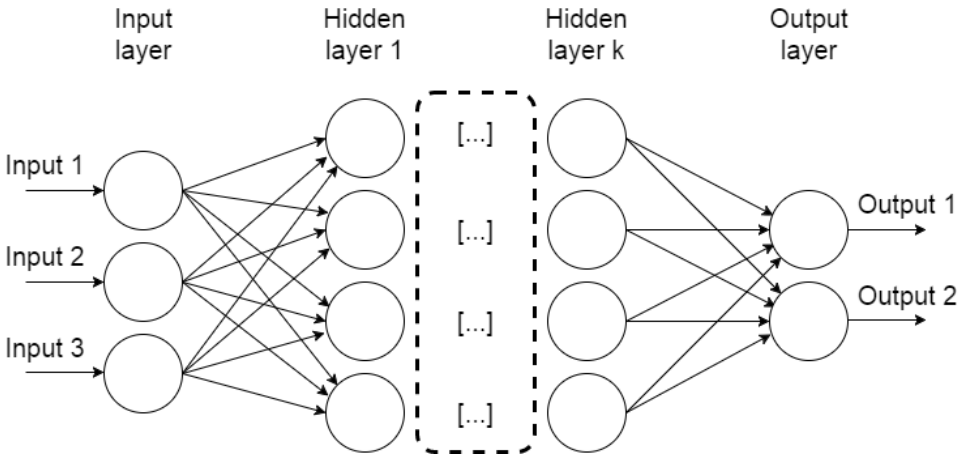


Figure 2.4.1.1: Representation of a Deep Neural Network composed by an Input layer, k Hidden layers and an Output layer. The number of layers k can be arbitrary and greater than 1. Note that this is a generalization of the model shown in Figure 2.4.0.2

and shares the same properties. There is a connection between any pair of nodes belonging to neighboring layers.

According to this arrangement, the output equation in (2.19) formulated for the non-deep neural network can be rewritten for the k_{th} layer as:

$$a_k = \sigma_k\left(b_k + \sum_{i=1}^{N} x_{k,i} \cdot w_{k,i}\right) \tag{2.20}$$

2.4.2. Recurrent Neural Networks (RNN)

Until now all the connections of the networks presented pass data from one layer to another, but always in the same direction. The connections going from the input layer to the output layer are called feed forward. A network is considered recurrent if it also has connections in the opposite direction, which is called feedback [22].

When there is feedback the inputs together with the order in which they are fed into the network is important, so sequential information can be processed. For this reason, RNNs are popular in Natural Language Processing and Speech Recognition [20], because words need to be analyzed in a sequence.

Incorporating feedback into a model exploits the temporal properties of the inputs by assuming there are correlations between the state at time t and the past $t - n$ states, and future states $t + n$ is if the system is non-causal. The recursion obeys to the fact that the same function is applied to each element in the sequence, but in slightly different ways according to other inputs from the past computations. This definition is related to the concept of memory, since the information about the states is saved in the network in order to be accessed and used later.

One intuitive way to look at RNNs is to represent a state diagram by unfolding the feedback connections over time, as illustrated in Figure 2.4.2.1.

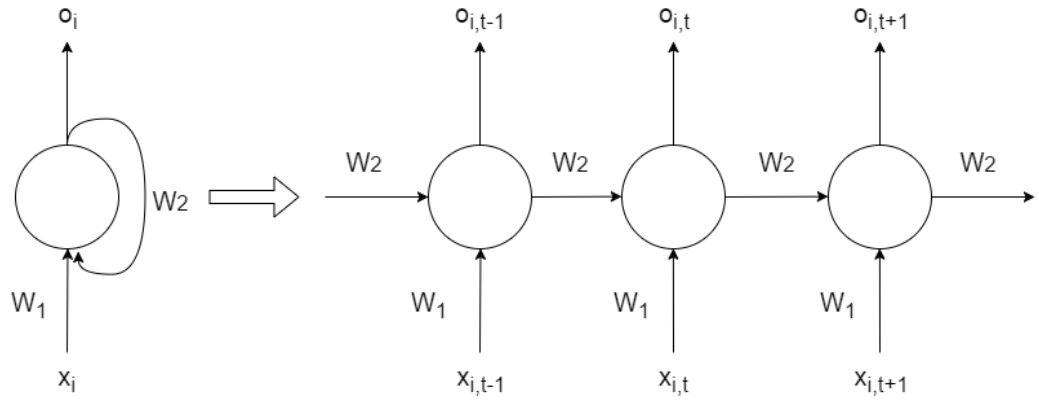


Figure 2.4.2.1: A recurrent neural network and the unfolding in time of the units involved in its forward computation.

Mathematically, the model of a recurrent neural network can be derived from the expression of a non-recurrent neural network by adding a time dependency. Assuming an RNN with one hidden layer, the behavior of the network can be expressed as follows, where $H(J(j,t-1))$ is the hypothesis of the network at time $t-1$.

$$J(j,t) = \sum X(i,t) \cdot W_1(i,j) + b_1 W_1(i+1,j) + W_2 H(J(j,t-1)) \qquad (2.21)$$

Since there is no limit in the past steps considered for the computation, this function has infinite memory, which is impractical to implement. Restricting the maximum number of past states makes the time context finite and defines the length of the short-term temporal dependencies that can be modeled.

2.4.3. Convolutional Neural Networks (CNN)

Convolutional Neural Networks are inspired from the visual cortex, where the input images are processed using image morphology filters. In this configuration, each neural unit is sensitive to a certain sub-region in the input space, which is called receptive field [22]. In order to cover the entire space of the input, the receptive fields are tiled, and may also be overlapped, along the whole image. Whereas RNNs are designed to exploit time correlations between inputs, CNNs perform well inferring local space correlations which are present in natural images. This has made the use of CNNs very popular for Image Processing tasks [23].

The locality properties of the receptive fields imply that each neural unit in only works together with their neighboring units, and it is connected to one unit of the next layer for each convolution occurring inside its receptive field. This reduces considerably the number of connections to be computed respect to a fully connected layer, where each unit in one layer is connected to each unit from the next layer. A diagram of a CNN is shown in Figure 2.4.3.1 to illustrate this property.

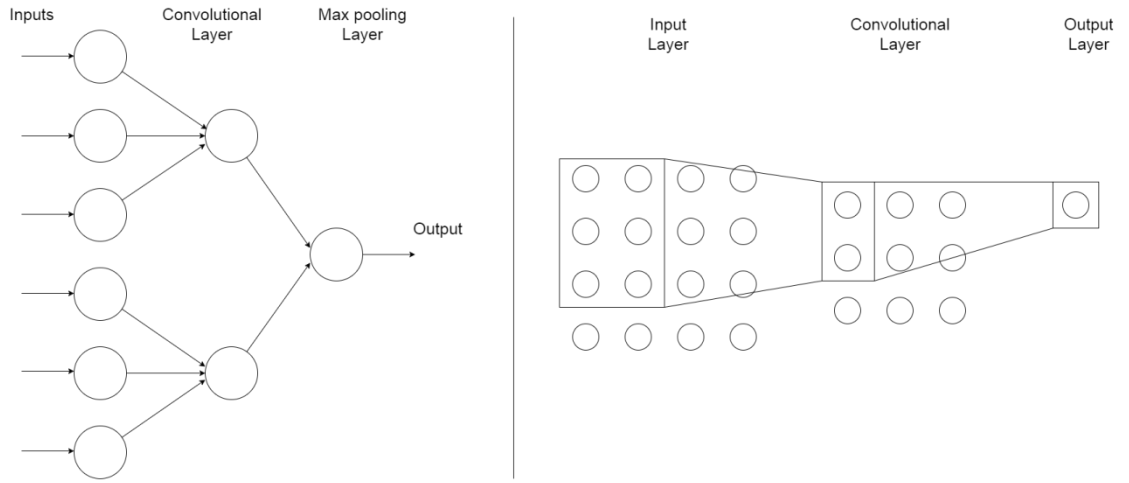


Figure 2.4.3.1: Structure of a CNN and its receptive fields. The structure is shown on the left, and the equivalent receptive fields for that structure are shown on the right.

Given that the convolution operation is the same over the receptive field of each unit, another important property is that the weight matrix W is shared [24]. Let the input image be a matrix of dimensions MxN, the receptive field has dimensions AxB and is moved along the input in a stride of $(1,1)$. Then, the next layer must have dimensions $(M-A+1)x(N-B+1)$ according to the convolution property.

The mathematical expression for a CNN can also be derived from the one of a non-convolutional case in (2.20), by adding the convolution operation expressed as follows:

$$a_{j,k} = \sigma\left(b + \sum_{l=0}^{A-1}\sum_{m=0}^{B-1} w_{l.m} \cdot x_{j+l,k+m}\right) \tag{2.22}$$

2.4.4. Autoencoders

An Autoencoder is an unsupervised learning algorithm which objective is to find a compact representation of the input data. They are useful to compression task, where the dimensionality of the data is typically reduced in the first layers [22]. This process is called Encoding and must be reverted afterwards in the Decoding stage to obtain the output. The output is a reconstruction $\tilde{x}$ that approximates x with some deviation, since the compression is lossy. The structure of an Autoencoder is shown in Figure 2.4.4.1.

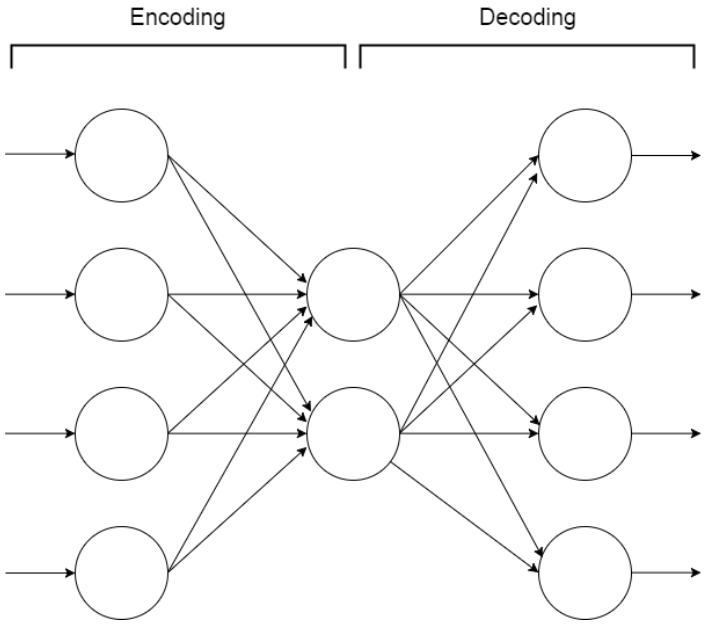


Figure 2.4.4.1: Network structure of an Autoencoder, showing the Encoding and Decoding stages. The compressed representation of the data is stored in the layer in the middle, which is a dense layer of smaller dimensions.

Notice that as more abstract representations are needed, more dense layers can be added to the structure presented above. Then, it is called a Deep Autoencoder [22].

Mathematically, an Autoencoder can be expressed as follows. Let W_1, b_1 be the weights and biases of the Encoding stage and W_2, b_2 be the weights and biases of the Decoding stage. The term σ represents the non-linearity applied by the neural units. The expression of the reconstructed value $\tilde{x}_i$ in terms of the input x_i is the following:

$$\tilde{x}_i = \sigma(W_2(\sigma(W_1 x_i + b_1)) + b_2) \tag{2.23}$$

Considering this, the cost function $J(\theta) = J(W_1, b_1, W_2, b_2)$ of an Autoencoder can then be formulated as:

$$J(W_1, b_1, W_2, b_2) = \sum_{i=1}^{N} (\tilde{x}_i - x_i)^2 \tag{2.24}$$

$$= \sum_{i=1}^{N} (\sigma(W_2(\sigma(W_1 x_i + b_1)) + b_2) - x_i)^2 \tag{2.25}$$

Another use of the Autoencoder is for initialization of neural networks. In some cases finding the initial values of the weights can be useful for faster convergence. This is called pretraining, and it is used to avoid overfitting and optimization problems [22].

2.4.5. How do neural networks learn?

As an analogy to the biological human brain, a Neural Network must go through a process called training in which the parameters are learnt before being ready to carry out a certain task.

For this purpose, the network has to be exposed to examples of the ideal inputs and outputs, which are represented by the vectors X and Y respectively. During the training phase the elements x contained in the set of inputs X are fed into the network. Then, the output given by the network is extracted, denoted by $\hat{y}$. The goal is that after several iterations the output given by the network $\hat{y}$ converges to the desired output y. The convergence is achieved by minimizing a loss function, also called error or cost function, which represents the difference between the two outputs. For example, the Euclidian Distance or the KL Divergence can be used.

Euclidian Distance

$$J(\theta) = \|y - \hat{y}\|^2 \tag{2.26}$$

Kullback-Leibler Divergence

$$J(\theta) = \sum \left(y \cdot log\left(\frac{y}{\hat{y}}\right) - y + \hat{y} \right) \tag{2.27}$$

The term $J(\theta)$ represents a parametrization of the cost function in terms of the variables θ that can be updated in the network, which are the weight matrix W and the bias matrix B.

- Gradient descent

In order to optimize these parameters to adapt the network for the desired task, it is important to recall the definition of the gradient. The gradient of any function is the direction in which the slope is maximum. Therefore, a step in towards the opposite

direction would ideally lead to a local or global minimum, decreasing the cost in the direction of the gradient [25]. For this intuition, this method is called gradient descent.

Let $\nabla_\theta J(\theta)$ be the gradient of the cost function. The update that is computed for each input element $x \in X$ can be expressed as follows:

$$\theta :\Rightarrow \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.28}$$

The parameter η is often called the learning rate and represents the amount of change allowed from one iteration to another. It is used as a scale factor and it is typically compressed within the range $\eta \in [0, 1]$. The purpose of this parameter is to regulate how fast the convergence happens, and also to avoid jumping between different regions of the curve or accidentally miss the minimum value. Notice that the expression works well only for convex cases in which the global minima can be reached. Otherwise, the convergence happens to local minima instead of finding the value that truly minimizes the cost function. Moreover, problems with higher complexity cannot be modeled with a convex cost function.

- Backpropagation

The gradient descent algorithm presented above assumes that the gradient of the network is known. Calculating the gradient of a neural network is not an easy task, especially when there are many hidden layers involved.

Backpropagation is a method that can be used to find a function for the network [26] and, strictly speaking, the process is divided in feedforward and backpropagation. First, one sample from the input data must be applied to the input layer and pass through the subsequent layers of the network to obtain the correspondent activations for each layer. The output given by the network $\hat{y}$ is then compared to the expected output y, from which an error signal is computed for each output node. Since all the nodes have contributed to the errors present in the output layer, the error is distributed among them according to their activations. Then, the output error signals are transmitted backwards recursively until they reach the input layer, thus, calculating the contribution to the output loss for all the nodes present in the network.

Let $J(\theta)$ be the cost function with parameters θ, s the activations and L the total number of layers. The partial derivative in a node j for a layer l can be expressed as:

$$\frac{\partial J(\theta)}{\theta_{ij}^l} = \frac{\partial J(\theta)}{\partial s_j^l} \cdot \frac{\partial s_j^l}{\theta_{ij}^l} = \delta_j^l \cdot h_i^{l-1} \tag{2.29}$$

This decomposition gives as first term the derivative of the loss respect to the activation function of the layer, denoted by δ_j^l. The second term is the derivative of the activation function respect to the weights, and can be calculated as h_i^{l-1}.

For a node j in the last hidden layer of the network $L-1$ the loss can be expressed in terms of the output of that node y_j and the cost function.

$$\delta_j^{L-1} = \frac{\partial J(\theta)}{\partial s_j^{L-1}} = \frac{\partial J(\sigma^{L-1} \cdot s_j^{L-1}, y_j)}{\partial s_j^{L-1}} \tag{2.30}$$

Once the value of δ_j^{L-1} has been calculated, it can be backpropagated to the preceding layers. The general expression for the contribution δ_j^{l-1} of the layer $l-1$ is the following:

$$\delta_j^{l-1} = \sigma'^{(l-1)} s_i^{l-1} \cdot \sum_{j=1}^{d(l)} \delta_j^l \theta_{ij}^l \tag{2.31}$$

2.5. What is 3D Audio?

According to the physical definition, the sound is a pressure wave that travels through a medium, and its propagation occurs in a 3-dimensional space. The name of 3D audio, also called spatial audio, refers to the listening experience rather than the physical reality. The main purpose of spatial audio is to emulate a real sonic environment. This is achieved by taking advantage of human perception and sound processing techniques to locate sound objects in the space around the listener. Many studies in the field of psychoacoustics [4], [27] have revealed that the human hearing system works as a filter bank and, thus, it plays an important role in determining the spatial properties of perceived sounds.

In order to obtain a 3D listening experience in a reproduction system (loudspeakers or headphones), it should be able to handle more than a single channel. Otherwise, it is not possible to recreate more than one dimension. Commercial music is typically distributed in stereo 2.0 format, which is the first and simplest sound spatialization technique consisting of 2 channels [28]. The position of the source can be selected as any point between the two loudspeakers by adjusting their respective levels or delays.

As research in spatial audio advanced more reproduction layouts appeared, and they were usually named after the number of loudspeakers involved [28]. Some examples are 4.0 (quadraphonic), 5.1 (surround) or 7.1 (surround). Nowadays there are multiple configurations of higher complexity that can handle up to 40 loudspeakers.

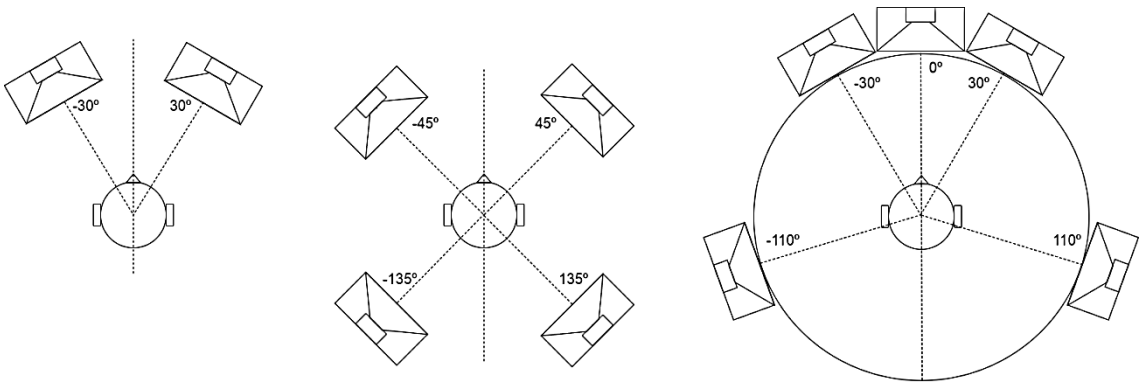


Figure 2.5.1: Some of the most common speaker configurations for audio spatialization. From left to right they are Stereo 2.0, Quadraphonic and Surround 5.1. Notice that the angle between a given pair of loudspeakers determines the available angular resolution to represent sources. As the angle is wider, the resolution is poorer.

2.5.1. Channels and Objects

The number of sound channels in a reproduction system does not necessarily have to match the number of loudspeakers. This is because the discipline of spatial audio deals with the following two approaches [29]:

- Channel-based: The production of the piece is done directly over the channels, and the audio format has as many tracks as loudspeakers. The advantage is that no further processing is needed because each track can be sent to its corresponding loudspeaker. The drawback is that this approach is dependent on the layout. A piece that has been produced for a particular layout must be reproduced under the exact same conditions. When reproduced in a different setup, it can lead to an incomplete representation of the sonic space if the number of loudspeakers is smaller. In the opposite case, no additional information will be added to the mix if the number of loudspeakers exceeds the number of channels required.

- Object-based: As opposed to the channel based approach, the information is transmitted as a set of Sound Objects, so that every object is composed by an audio track and some metadata associated to its spatial characteristics. The most important implication of this paradigm is the layout independence, but also the need of having a decoder (or renderer) that translates the information to the corresponding loudspeakers, which can cause some overhead in the system performance if a large number of Sound Objects needs to be computed.

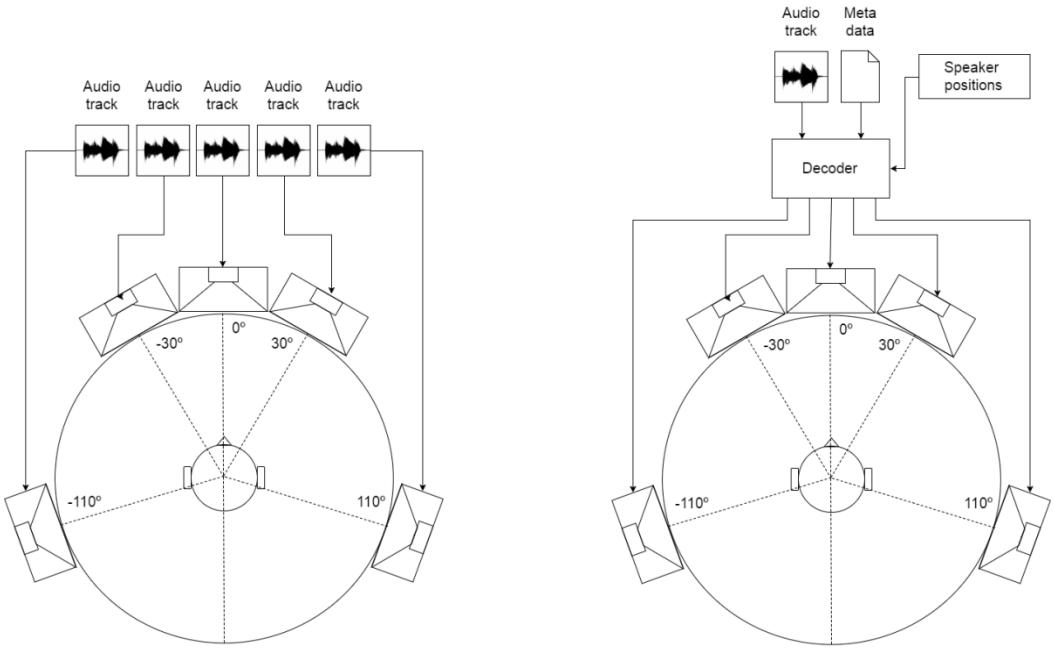


Figure 2.5.1.1: Comparison between the Channel-based approach (on the left) and the Object-based approach (on the right). Another consideration is that the one on the left depends on the layout in contrast to the one on the right, which is independent. It can be observed that, in the first case, an individual audio track is required to drive every single speaker of the setup. In the second case, there is only one audio track and some metadata, but the signals to drive each speaker must be computed by a decoder.

2.5.2. Phantom effect

The illusion of a source located in a position where no loudspeaker is present is called Phantom effect [28]. This effect is based on human perception and it can be achieved within the arch between two loudspeakers. The human brain can determine the location of an audio source by measuring either the Interaural Level Difference (ILD) or the Interaural Time Difference (ITD). These two measurements refer to the arrival of sound to one ear respect to the other in terms of intensity (ILD) or delay (ITD) [30].

For the phantom effect to be noticeable, the emissions of both loudspeakers must be coherent. This implies that they should be exactly equal in terms of content, with just a difference in level or some amount of delay. Under this condition, the listener's brain would integrate the arriving information and average it to the phantom location.

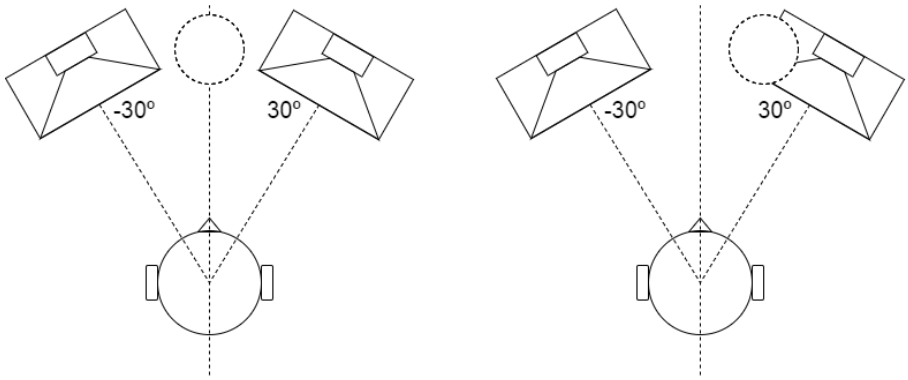


Figure 2.5.2.1: Illustration of the Phantom effect in a stereo 2.0 system. The phantom source is represented by the dashed circle and only the horizontal plane is considered. If the emission of both loudspeakers is the same as in the first case, the integration leads to locate the phantom source in the center. If one loudspeaker emits at a higher intensity or arrives earlier to the listener, the source tends to deviate towards that specific location as in the second case.

2.6. Amplitude and Time Panning

The manipulation of a sound so that it appears to come from a certain location is called sound balancing, most commonly known as panning. As it has been commented in the last section, the phantom effect can be used to create this impression based on Level (ILD) or Delay (ITD).

Human listeners can distinguish two sounds in space as far as the angle between them is larger than the Minimal Audible Angle (MAA). In case those two sources come from a closer angle, they would be perceived as if they were in the same location [4], [28]. Evolution has made it considerably easier for humans to locate sources in the horizontal plane than in any of the vertical planes, and it is also easier when the source is in front of the listener. The following table shows an approximation of the MMAs for some general cases:

HORIZONTAL			VERTICAL		
Front	Rear	Lateral	Front	Rear	Lateral
≈3°	≈5°	≈10°	≈10°	≈12°	≈20°- 30°

Table 2.6.1: Approximate values of Minimal Audible Angle (MAA) for horizontal and vertical sound localization around the listener. The best resolution is achieved in the horizontal plane and sources coming from the front.

When panning is intended to achieve the phantom effect, the spatial resolution of the source will depend on the angle of aperture between the pair of loudspeakers and the plane respect to the listener. The best resolution is achieved when the angle is small and when it happens in the horizontal plane, as reflected in Table 2.6.1.

According to psychoacoustic principles, the level differences are perceived much better than the time differences [4], [28]. Amplitude panning has experimentally proven to work in more diverse contexts. The performance of the human ear for ILD or ITD does indeed depend on the frequency, but this fact does not represent a problem in most cases since low frequencies tend to have poorer directionality.

- Level differences (ILD) work better from up to 1.5 kHz because the associated wavelength is the critical distance of ≈18 cm which is the average diameter of the human head. Therefore, the head produces an acoustic shadow for any smaller wavelength. This means that the side that is further away from the source presents attenuation, creating a difference in level.

- Time differences (ITD) perform better for low frequencies, starting at around 800 Hz and below. Since the wavelength associated to such frequencies is significantly larger than the dimensions of a normal human head, attenuation is negligible. The sound can be localized because the phase shift of the incoming wave corresponds to a time delay that is large enough to be discriminated by the human brain.

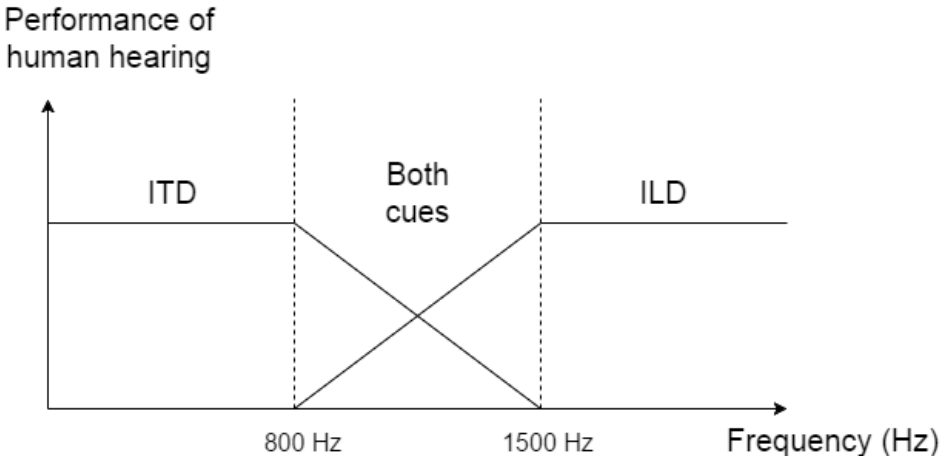


Figure 2.6.1: Graphic of how the Time differences (ITD) and Level differences (ILD) are considered for spatial localization with respect to the frequency of the source. This

is just intended to be a rough representation of the concept to illustrate their respective roles in hearing, so no specific units of performance are given.

2.6.1. Tangent law

The tangent law has been proposed as a mathematical tool to calculate amplitude based panning between a pair of loudspeakers [30]. Let θ be the angle at which the phantom source is located, θ_t half the angle of aperture between the loudspeaker pair and g_1, g_2 the respective gains of loudspeakers 1 and 2. The relationship between these variables can be expressed as follows:

$$\frac{\tan(\theta)}{\tan(\theta_t)} = \frac{g_1 - g_2}{g_1 + g_2} \tag{2.32}$$

Notice that there are two unknown values g_1, g_2, but only one equation. Therefore, some further assumption is required to find them. The simplest way to solve this issue is by imposing the condition of the sum of the gains equal to one. Then, the following expression can be derived:

$$2 \cdot g_2 = 1 - \frac{\tan(\theta)}{\tan(\theta_t)} \qquad g_1 + g_2 = 1 \tag{2.33}$$

The gains obtained sum to one, although it does not imply that the energy is constant. In fact, that would only hold when just one loudspeaker is active. In order to obtain the correct gains g'_1, g'_2 for which the energy is conserved, normalization must be applied. This type of panning is called Constant Power or Equal Power panning.

$$g'_i = \frac{g_i}{\sum_{j=1}^{2} g_j} \qquad i = 1, 2 \tag{2.34}$$

2.6.2. Vector-Based Amplitude Panning (VBAP)

A generalization of the Tangent Law for a 3-dimensional setup has been proposed by Pulkki [31] under the name of Vector-Based Amplitude Panning, also known as VBAP. The Tangent law can be reformulated by expressing the loudspeaker positions in a vector base of 2 or 3 dimensions depending on the needs [30], [31]. The 2-dimensional case is presented first, and then its generalization for 3 dimensions.

Let the listener position be the point at the origin of the coordinate system. The base is defined by the unitary vectors l_i, which point from the origin towards each loudspeaker i. The point p is the location of the virtual source, expressed according to the same principle. Therefore, as shown in the Tangent law, they can be considered as a linear combination where g_i are non-negative gain coefficients.

$$p = g_1 \cdot l_1 + g_2 \cdot l_2 \tag{2.35}$$

Notice that in a 2-dimensional configuration the maximum number of active loudspeakers at any instant of time is two. This is because the vector base requires two linearly independent vectors to express any point in space as a linear combination of

these base vectors. To obtain the best possible resolution, the selection of the loudspeaker pair must be made according to their proximity to the virtual source. It can also be thought as selecting the closest loudspeaker and one of its neighbors which must be the second closest loudspeaker. This is achieved by computing the Euclidean distance for all the loudspeakers in the configuration and selecting the two minimum values. Alternatively, the angles between the virtual source and each loudspeaker can be computed.

$$\{l_1, l_2\} = \min_i(\sqrt{(l_{i1} - p_1)^2 + (l_{i2} - p_2)^2}) \tag{2.36}$$

It is important to mention that in some cases there is only one active loudspeaker. This happens when the position of the virtual source is coincident with the position of one loudspeaker. Therefore, there would be one gain coefficient equal to zero.

The linear combination can be rewritten in matrix form as follows:

$$p^T = g \cdot L_{12} \;\rightarrow\; g = p^T \cdot L_{12}^{-1} \tag{2.37}$$

$$(p_1,\, p_2) = (g_1,\, g_2) \cdot \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{bmatrix} \rightarrow (g_1,\, g_2) = (p_1,\, p_2) \cdot \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{bmatrix}^{-1} \tag{2.38}$$

In order to obtain the gain coefficients the inverse matrix L_{12}^{-1} must exist, which implies that the loudspeaker pair must be a valid base. This is not true when the angle between the loudspeakers is 0° or 180°, since the resulting vectors would be linearly dependent. Assuming that the condition of linear independence is satisfied, the resulting coefficients must be normalized to preserve the energy of the source. The term C is a constant that can take an arbitrary value.

$$g'_i = \frac{\sqrt{C}\, g_i}{\sqrt{g_1^2 + g_2^2}} \quad i = 1, 2 \quad , \qquad g_1^2 + g_2^2 = C \tag{2.39}$$

A generalization of the 2D case can be made to adapt the expression for a 3D coordinate system. Ideally, the loudspeakers and the virtual sources should be positioned on the surface of a unit sphere, so that they are equidistant from the listener. Then, the vector $l_i = \{l_{i1}, l_{i2}, l_{i3}\}$ defines the position of the loudspeaker i respect to the origin, and the vector $p = \{p_1, p_2, p_3\}$ corresponds to the virtual source. Again, the relation can be expressed as a linear combination and in matrix form.

$$p = g_1 \cdot l_1 + g_2 \cdot l_2 + g_3 \cdot l_3 \tag{2.40}$$

$$p^T = g \cdot L_{123} \;\rightarrow\; (p_1,\, p_2,\, p_3) = (g_1,\, g_2,\, g_3) \cdot \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \tag{2.41}$$

$$g = p^T \cdot L_{123}^{-1} \;\rightarrow\; (g_1,\, g_2,\, g_3) = (p_1,\, p_2,\, p_3) \cdot \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix}^{-1} \tag{2.42}$$

The inverse matrix L_{123}^{-1} must exist under the assumption that the three vectors from the loudspeaker pair are linearly independent. The normalization is performed as in the 2 dimensional case, but now using the 3 gain coefficients instead.

$$g'_i = \frac{\sqrt{C}g_i}{\sqrt{g_1^2 + g_2^2 + g_3^2}} \quad i = 1,2,3 \quad , \qquad g_1^2 + g_2^2 + g_3^2 = C \tag{2.43}$$

3. METHODOLOGY

The system that combines the source separation and the remixing stage will be presented in the following pages**.** This section focuses on presenting the architecture of the Deep Learning model used for Source Separation, which constitutes the core of the proposal. Also, the 3D Audio stage is commented together with the implementation of the Web interface.

3.1. Proposed System

The general picture of the system architecture is depicted in Figure 3.1.1. It consists of two independent stages of Source Separation and 3D Audio that will be commented in further detail in dedicated sections later. The input the audio processing chain is a mono or stereo 2.0 audio stream, which is the song to be separated. It is first fed into the Source Separation stage to obtain an estimation of the instruments. Then, the instrument tracks are interpreted as the sound objects to be rendered in a 3D Audio environment according to the user specifications and the spatial distribution of the reproduction system.

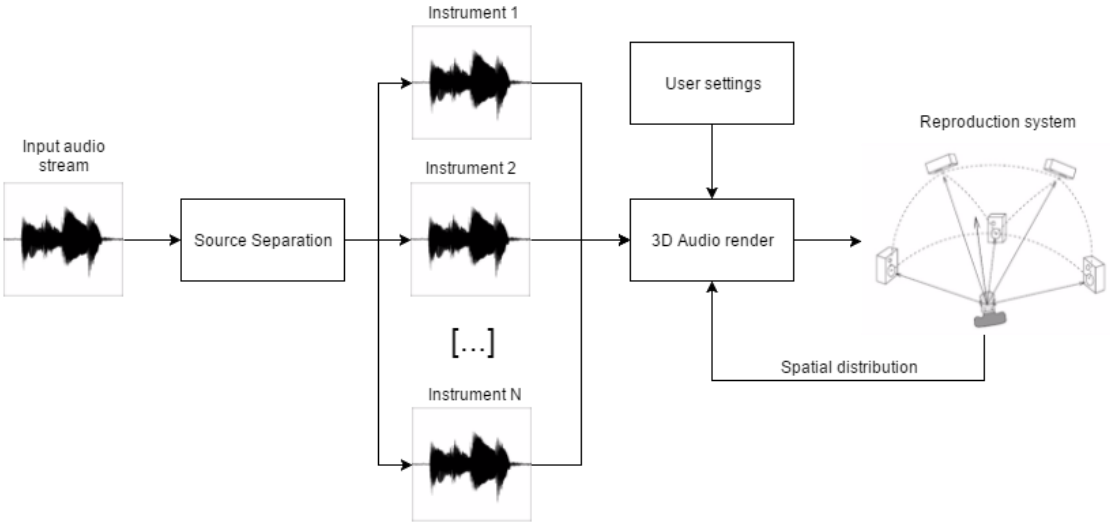


Figure 3.1.1: Diagram of the source separation and 3D upmixing stages combined.

As a result, the song can be manipulated by changing the spatial locations of the instruments in the mix, muting them, and applying effects to some of the tracks. In terms of runtime, the latency of the whole chain comes entirely from the source separation stage. This task implies the highest computational cost of the two. 3D Audio Rendering can be done faster than the minimum delay perceived by a human listener, so it will be considered as real-time.

Recently, a low latency framework based on Convolutional Neural Networks (CNN) has been proposed for source separation in [12], [21]. The authors claim that it can be applied to cochlear implants with some improvements given the fast processing time it has achieved.

3.2. Source Separation

The model used by [12], [32] is inspired by the principles of NMF, in which the timbre characteristics of a source are modeled in the basis vector B, and its temporal evolution is represented by the gain vector G, recalling Eq. (2.12). The parameters are learnt using a Convolutional Neural Network (CNN), which takes 2-dimensional inputs.

Therefore, it is required to represent the input audio signal according to the frequencies present in one time frame. The Short-Time Fourier Transform (STFT) is computed to obtain this time-frequency representation on a segment of duration T, which is the analysis time context of the audio mixture. Even though this operation returns the magnitude and phase of the spectrogram, only the magnitude is considered a relevant feature to learn, since phases may take random values.

The network is designed to output an estimate of the magnitude spectrogram for each source. These outputs are not used directly for the reconstruction, but are used to compute Time-Frequency soft masks. These masks are applied to the original mixture to recover the spectrogram of the sources [32]. Then, the resulting magnitude is combined with the original phase of the mixture to compute the Inverse Short-Time Fourier Transform (ISTFT). Finally, a reconstruction of the audio signal in the time domain is obtained for the separated sources.

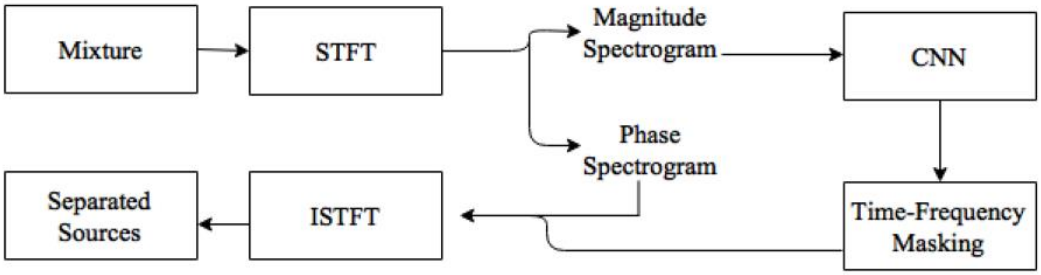


Figure 3.2.1: Data flow of the Source Separation stage. Extracted from [12].

3.2.1. Network architecture

Convolutional Neural Networks (CNN) have proven to be very effective finding efficient representations of data and also to be able to perform operations at different scales for Image Processing tasks [23].

In contrast to other architectures where the sources are modeled in full resolution, this approach relies on finding a compact model. This reduces the number of parameters to be learnt by the network based on the principles of an Autoencoder architecture, having an Encoding and a Decoding stage. The dimensionality reduction makes the computation faster because the optimization problem is simplified.

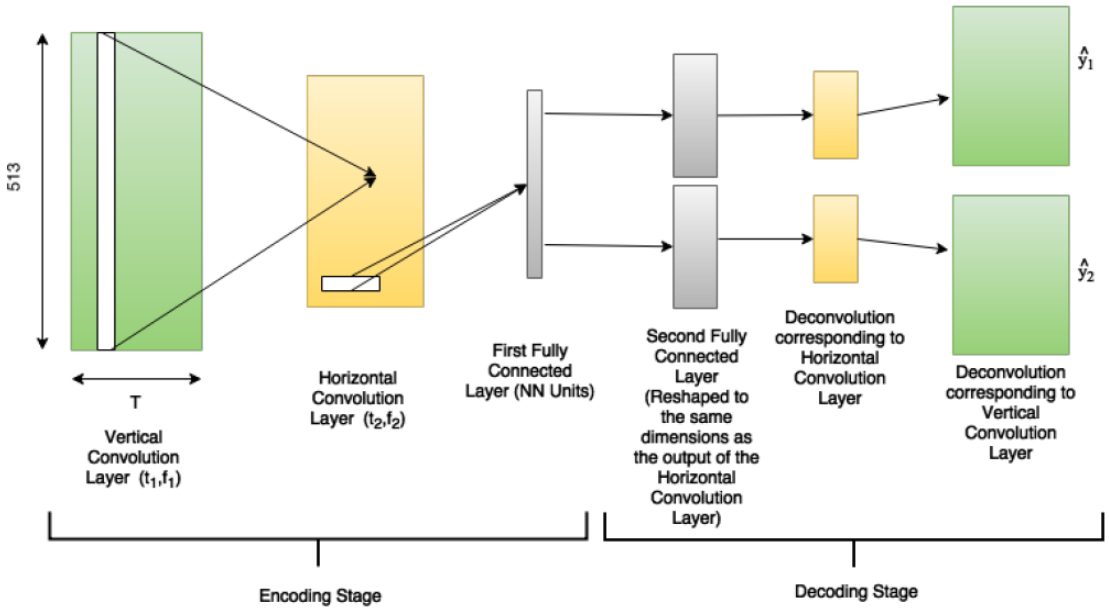


Figure 3.2.1.1: Network Architecture for Source Separation, using Vertical and Horizontal convolutions, showing the Encoding and Decoding stages. Extracted from [12].

3.1.2.1. Encoding stage

This part consists of two convolutional layers that learn time and frequency features both vertically and horizontally, and a fully connected dense layer. The last layer acts as a bottleneck and it is where the information is compressed. To understand how the convolutional layers extract the features, it is important to keep in mind that a spectrogram is a representation of frequency along the vertical axis and of time along the horizontal axis. Each layer is explained more in detail next.

- Vertical Convolution Layer

The first layer of the network is intended to model the timbre characteristics of the input at a local scale, since the convolution is taken along all the frequency bins at a fixed time index. In this sense, this approach is analogous to the NMF, but the main difference lies in the fact that here the timbre characteristics are shared among all the instruments, whereas in NMF the estimation of the basis and activations is different for each of the sources.

The dimensions of this layer match the input spectrograms, which have a time frame of length t_1 and f_1 frequency bins, being expressed as a shape of (t_1, f_1). The number of filters used, denoted by N_1 determine by how much the dimensionality is reduced to the next layer.

- Horizontal Convolution Layer

This layer models the temporal evolution of the local timbre characteristics extracted in the Vertical Convolution Layer. It has been designed to learn some

time-frequency discriminations between the instruments to be separated, since the temporal evolution from each source may be different despite sharing some timbre features.

The dimensions are now expressed as a shape of (t_2, f_2), being smaller than the first layer. The number of filters used, denoted by N_2 keep reducing the dimensionality of the features.

- Fully Connected Layer

This layer receives the output of the Horizontal Convolution Layer and consists of making non-linear combinations of the features learnt in the convolutional layers by applying a Rectified Linear Unit (ReLU) function. This characteristic makes the layer behave as a bottleneck and reduce the dimensionality up to a single column vector of length NN, which corresponds to the number of ReLU nodes present in the layer.

3.2.1.2. Decoding stage

The representation obtained in the first Fully Connected Layer is passed through another Fully Connected Layer that applies ReLU non-linearity. The number of outputs is dependent on the number of sources to be separated, and each output is dimensioned according to the output of the Horizontal Convolution Layer. Then, two deconvolutions are applied in the two successive layers, performing the inverse operations of the Horizontal Convolution Layer and the Vertical Convolution Layer. At the end, the estimations $\hat{y}_i$ are obtained for each source y_i.

3.2.2. Time-Frequency masking

According to the model diagram presented in Figure 3.2.1, the estimations $\hat{y}_i$ of the sources cannot be used directly. Instead they are used to compute a soft masking function that is applied to the spectrogram of the audio mixture, using Wiener Filtering as proposed in [32], [33]. These masks can be thought as the element wise normalization of the estimations over the frequency bins $\hat{y}_i(f)$. For N sources, the mask can be expressed mathematically as follows:

$$m_i(f) = \frac{|\hat{y}_i(f)|}{\sum_{i=1}^{N} |\widehat{y}_i(f)|} \qquad i = 1, .., N \tag{3.1}$$

The estimated mask can then be applied to the mixture spectrogram $x(f)$ to obtain an appropriate reconstruction of the sources, denoted by $\tilde{y}_i$.

$$\tilde{y}_i = m_i(f) \cdot x(f) \qquad i = 1, .., N \tag{3.2}$$

3.2.3. Parameter learning and adjustments

The optimization of the parameters is done using Stochastic Gradient Descent with AdaDelta algorithm [34]. The cost function is expressed as the squared error between the estimate $\tilde{y}_i$ and the original source y_i.

$$L_{sq} = \sum_{i=1}^{N} \|\tilde{y}_i - y_i\|^2 \tag{3.3}$$

This initial approach presented the problem that the bass, drums, and vocals had some characteristics in common among different songs, but the content in others was very diverse. Assuming that others correspond to the last estimation N, the cost function can be reformulated to contemplate only the first $N-1$ estimations as follows:

$$L_{sq}{}' = \sum_{i=1}^{N-1} \|\tilde{y}_i - y_i\|^2 \tag{3.4}$$

Under this condition, the *others* track will be the residual after having extracted 3 instruments from the mixture. Now the cost function performs better than before, but can be further improved by imposing some constraint about the similarities between the estimations, as proposed in [12]. The term L_{diff} can be added to the cost function in order to prevent the estimations from presenting high similarity.

$$L_{diff} = \sum_{i=1}^{N-1} \left\|\tilde{y}_i - \tilde{y}_j\right\|^2 \quad , \quad \forall j \neq i \tag{3.5}$$

By regulating the contribution of L_{diff} in the cost function L with a coefficient α, it can be rewritten as:

$$L = L'_{sq} - \alpha \cdot L_{diff} \tag{3.6}$$

$$L = \sum_{i=1}^{N-1} \|\tilde{y}_i - y_i\|^2 - \alpha \cdot \sum_{i=1}^{N-1} \left\|\tilde{y}_i - \tilde{y}_j\right\|^2 \quad , \quad \forall j \neq i \tag{3.7}$$

The difference from each source respect to the estimation of *others* is not considered in the above expression, but it has shown to improve the model's performance. It can be included in the cost function by adding the terms $L_{othervocals}$ and L_{other}, which correspond to the similarity between other respect to vocals and other respect to the rest of the instruments. This distinction is done to fine tune the separation of the vocals independently from the rest of the sources. Let the estimation of the vocals be $\tilde{y}_1$ and the estimation of others be $\tilde{y}_N$. The differences can be expressed as follows:

$$L_{othervocals} = \|\tilde{y}_1 - \tilde{y}_N\|^2 \tag{3.8}$$

$$L_{other} = \sum_{i=2}^{N-1} \|\tilde{y}_i - \tilde{y}_N\|^2 \tag{3.9}$$

The contribution of these terms is regulated by the coefficients β and β_{vocals}. The total cost function can be expressed as:

$$L_{Total} = L'_{sq} - \alpha \cdot L_{diff} - \beta \cdot L_{other} - \beta_{vocals} \cdot L_{othervocals} \quad (3.10)$$

3.3. 3D Audio

Once separated, the audio tracks corresponding to the sources can be treated as audio objects by a 3D Audio renderer. The relation is direct 1:1, meaning that 1 source has 1 associated audio object. Also recall from Channels and Objects, in Section 2.5.1, that an Object Based approach needs the audio waveforms and their associated metadata in order to place the object correctly in the space. The diagram of the system is presented in Figure 3.3.1 as follows:

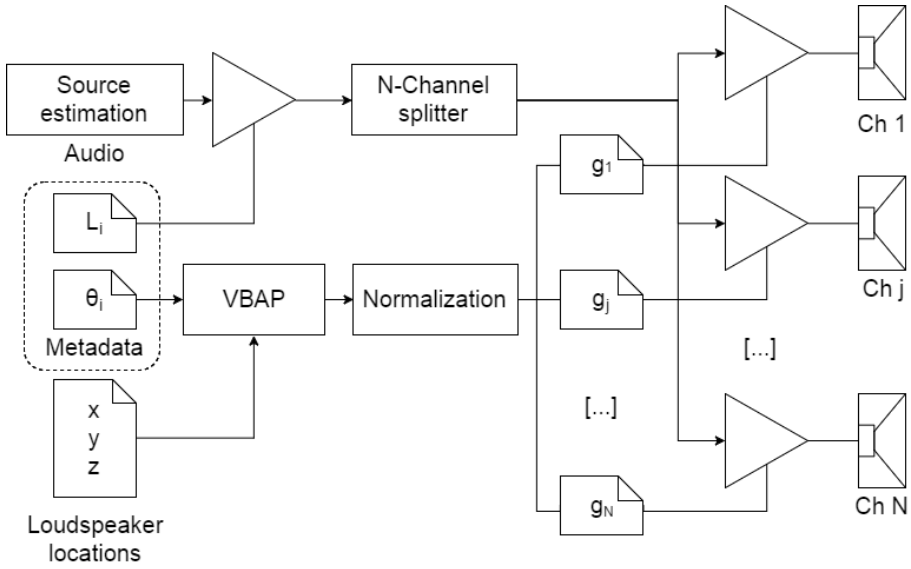


Figure 3.3.1: Data flow of the 3D Audio stage. For simplicity, this representation covers the spatialization of one source estimation only. The actual system consists of a repetition of these same stages for each source and a summation stage before being fed into the loudspeakers.

For the application considered in this work, the metadata consists of the angle in the horizontal plane (XY) and the level at which the audio object should be played. Notice that this simplification has been made under the assumption that typical home reproduction systems are stereo 2.0 and surround 5.1 or 7.1, where the loudspeakers have a planar configuration. However, the panning methods used for the implementation of this rendering chain allow the capabilities of the system to be easily extended to work in 3 dimensions (XYZ).

The angle and intensity level data is generated by the user input in a graphical interface, and the technique used to place the audio objects in the space is a modified version of VBAP that has been chosen because of its flexibility. The modification consists on imposing the condition of energy conservation, so that the panning only changes the orientation of the source while leaving its level untouched. Because of this, only the

angle θ_i is required. A second gain can be applied to the entire audio object in order to set the level of the source, which can be thought as the relative distance to the listener. For this reason, there are two gain stages in the diagram presented above. The gain g_j corresponds to the value calculated by the VBAP algorithm after applying equal power normalization, and L_i corresponds to the intensity level assigned to that audio object of index i.

3.3.1. Web Audio API

The Web Audio API, which has been used to build the user interface, provides the following abstraction:

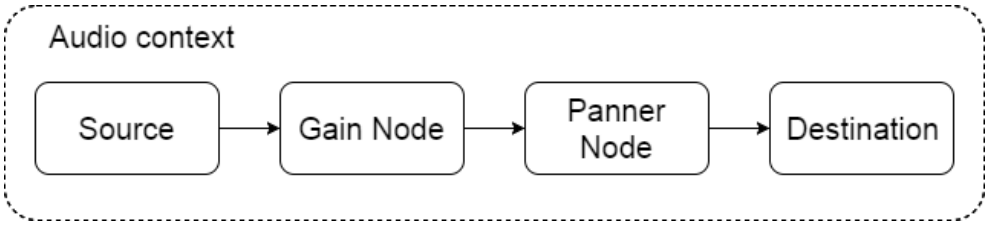


Figure 3.3.1.1: Data flow in Web Audio API. Notice that this model is equivalent to the one shown in Figure 3.3.1, and represents the processing of a single source.

Audio Nodes are the basic building blocks in WebAudio API, and can be thought as a generic audio processing unit that applies a function to an audio stream. Audio Nodes can be used for many different purposes such as compression, panning, gain stages or filtering, among others. The interface of a block provides inputs and outputs with a given number of channels. Different audio nodes can be interconnected or cascaded to create what is called the Processing Graph, and the whole chain is contained inside of the Audio Context. The most relevant cases of Audio Nodes for the application developed in this work are the following:

- Source Node

A source is an Audio Node with no inputs and one output, which dimensionality is determined by the number of channels. In the Web application, each audio track is associated with a source in order to be played. The technical name used in the APIs documentation is AudioBufferSourceNode, and it consists of storing some audio signal in its internal memory. The audio is streamed out when required by incoming playback events.

- Gain Node

The gain stages are implemented in Gain Nodes, which intuitively behave as a linear amplifier. The interface has exactly one input and one output, and the processing consists on multiplying the input by a scale factor, called gain, to obtain the output. Therefore, the function applied by this node is linear and the slope of the line represents the amplification factor.

$$y = Gx \tag{3.11}$$

As a consequence, a change in Volume is applied to the audio. Let the gain factor be denoted by G, the behavior of the Gain Node is the following:

$$0 < |G| < 1 \quad \rightarrow \quad Attenuation$$

$$|G| = 1 \quad \rightarrow \quad Identity\ function$$

$$|G| > 1 \quad \rightarrow \quad Amplification$$

Notice that a negative sign just implies a phase shift of π in the output. Also, if the value of G is greater than one it can cause undesired distortions. Therefore, the best practice is to set the node to attenuate the signal and assume that the input is at full volume.

- Panner Node

A Panner Node can be used in order to apply panning to an audio signal to position the source in space. The position of the source is represented in a right-hand Cartesian coordinate system in 3 dimensions and it is computed in relation to the Listener position, which is at the origin by default. In case one dimension is not needed it can be set to zero or to a constant value, giving a plane as a result. There are many other parameters such as the movement or the directionality of the source, but those go beyond the typical use case.

The interface has exactly one input and one output, which number of channels is variable and is interpreted as speakers. The number of input channels can be 1 for Mono, 2 for Stereo or more for other formats. The number of output channels must be at least 2. Notice that this is due to the fact that in the single channel case no panning effect can be achieved. As in Tangent Law and VBAP, individual gains are applied to each channel to generate a phantom source.

- Destination

Once the audio has been processed by the chain, the result can be sent to the sound card in fixed size blocks, which get converted to a continuous audio signal that can be listened.

For further details about the functionalities and the different classes included in the library, the reader may find useful to consult the Web Audio API reference documentation[1].

3.3.2. Web Application

The Web Application has been built as an interactive visualization and listening tool. The separated tracks are loaded into the interface and can then be manipulated by changing their volume, panning and setting individual tracks to mute or solo.

[1] Web Audio API Reference Documentation: https://www.w3.org/TR/webaudio/

In terms of design, it has been conceived as a minimalist application with two main screens. The purpose is to make the user experience as simplified as possible, given that manipulating audio like in a music production environment would be more complex from both the implementation and the usability point of view. The first screen is the Uploader, in which the user can select the tracks to be loaded into the system. The second screen is the Mixer, in which the user can manipulate the tracks. When the page is reloaded or when the user navigates back the Uploader screen is displayed again. The following state diagram shows the two states and the actions taking place during the state transitions.

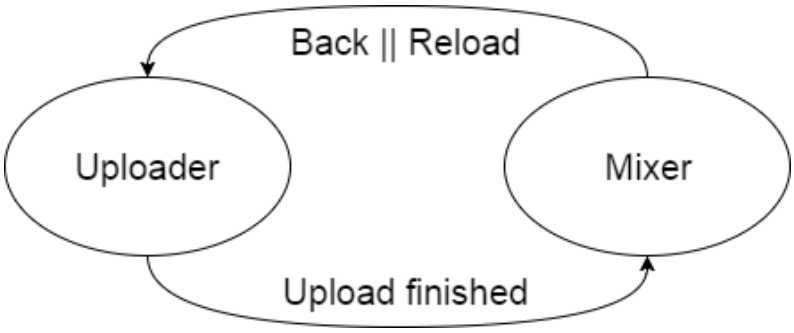
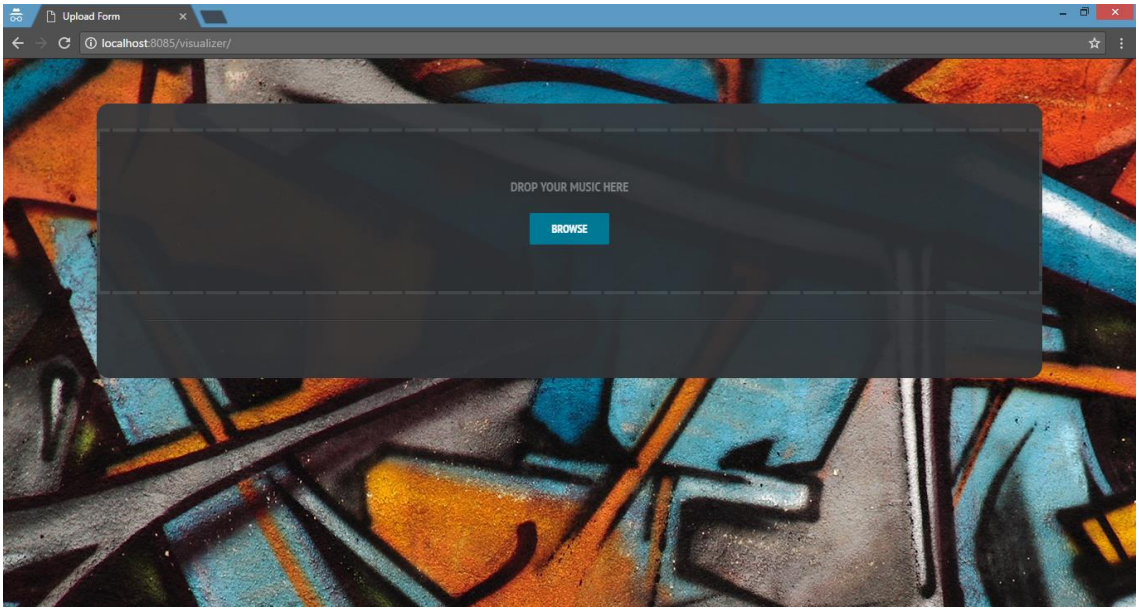


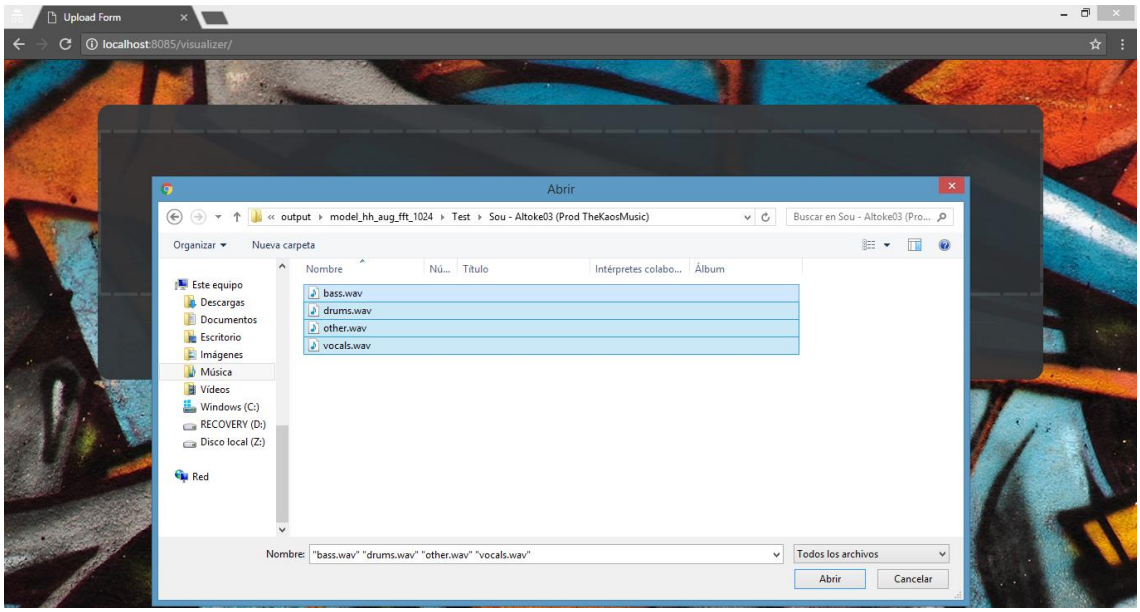
Figure 3.3.2.1: State diagram of the Web Application.

3.3.2.1. Uploader

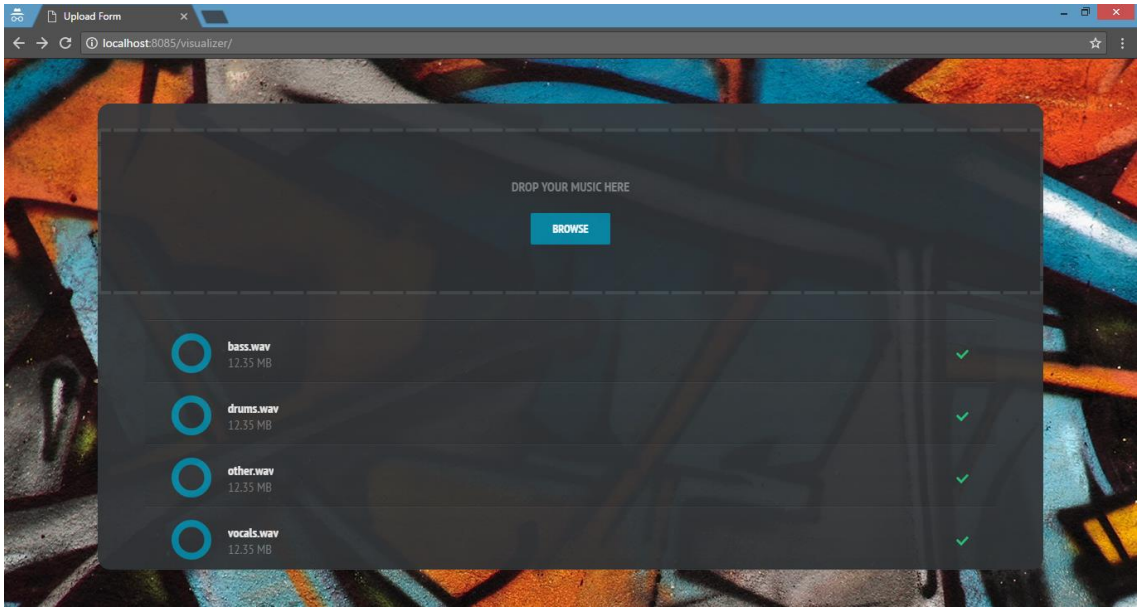
The Uploader area supports two types of inputs, since the tracks can be either selected with an Open File dialog box or by a Drag and Drop action directly from the file system into the browser. Once the tracks are selected the upload process starts automatically by sending a request to the server for each new file that is added to the queue. Then, a transition to the Mixer screen happens when all the files in the queue have finished uploading. The Uploader screen is shown in Screenshot 3.3.2.1.1, Screenshot 3.3.2.1.2 and Screenshot 3.3.2.1.3.



Screenshot 3.3.2.1.1: Uploader screen of the Web Application with an empty queue.



Screenshot 3.3.2.1.2: Uploader screen of the Web Application showing the Open File dialog.



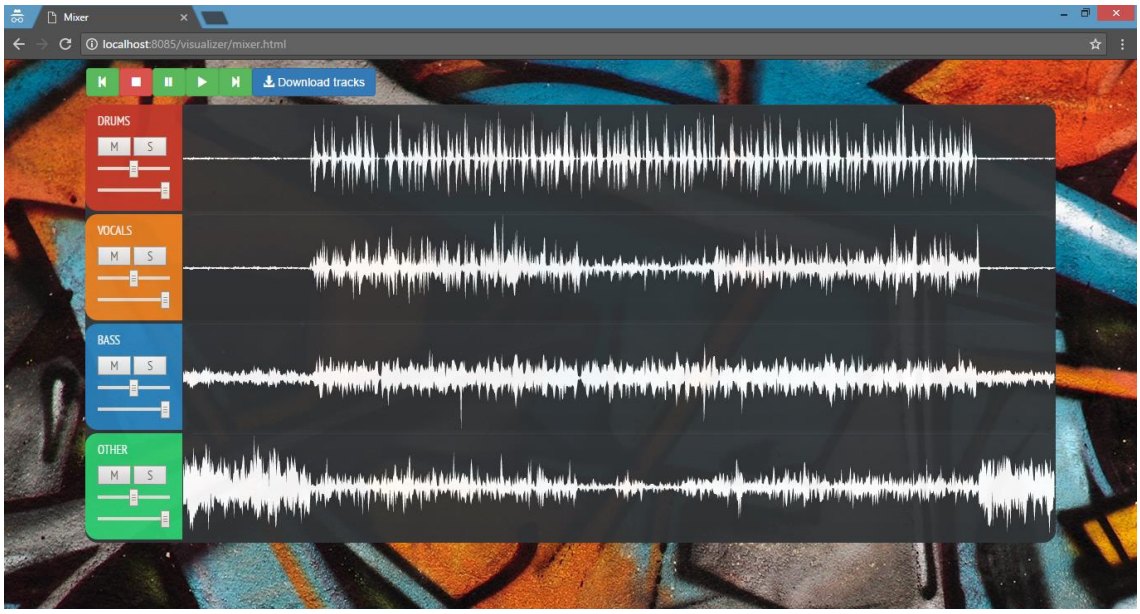
Screenshot 3.3.2.1.3: Uploader screen of the Web Application processing an upload queue of four tracks.

The frameworks used for this screen are the Uploader widget from JQuery[2] and the CSS from Bootstrap[3], for the functionality and the style respectively. The reader is encouraged to check the reference documentation in the links below.

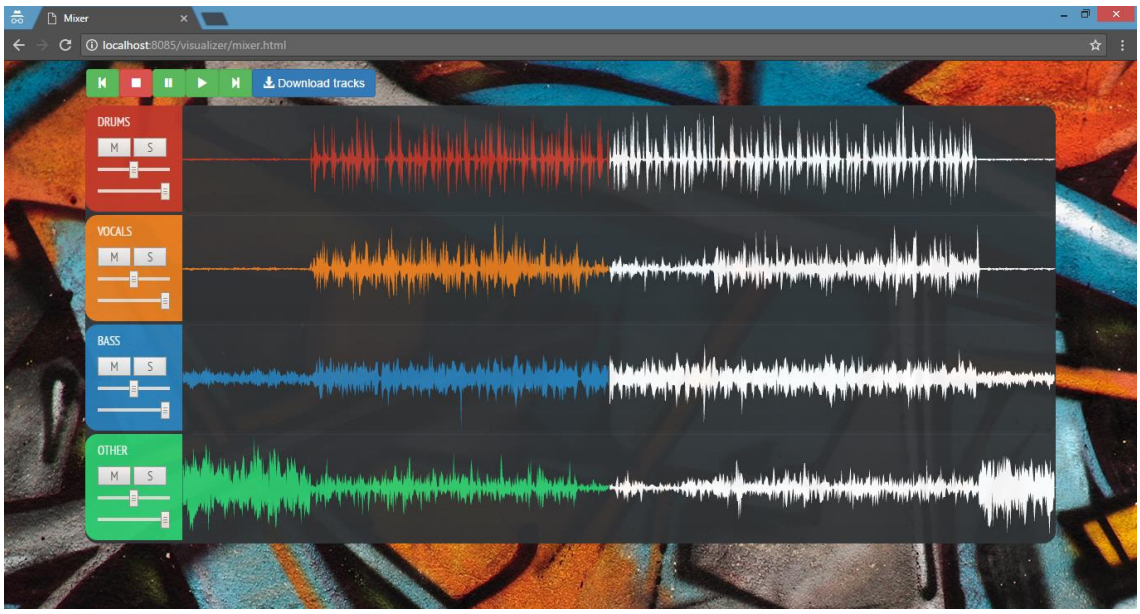
[2] JQuery reference Documentation: https://api.jquery.com/
[3] Bootstrap CSS: http://getbootstrap.com/css/

3.3.2.2. Mixer

The mixer shows the playback controls, the audio tracks and the mixing parameters for each track. The greatest area in the middle corresponds to the waveforms of the four instrument tracks of Bass, Drums, Vocals and Others, obtained at the Source Separation stage. Each track is presented with a different color and labeled according to the name of the instrument. The part of the track that has been played is highlighted with the color of that track, and the rest ahead is filled in white. These characteristics can be seen in Screenshot 3.3.2.2.1 and Screenshot 3.3.2.2.2.

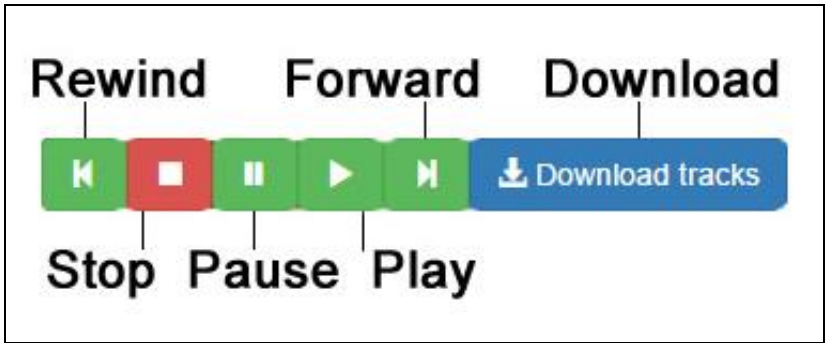


Screenshot 3.3.2.2.1: State of the audio waveforms with the cursor at the beginning of the song.



Screenshot 3.3.2.2.2: State of the audio waveforms with the cursor at the middle of the song.

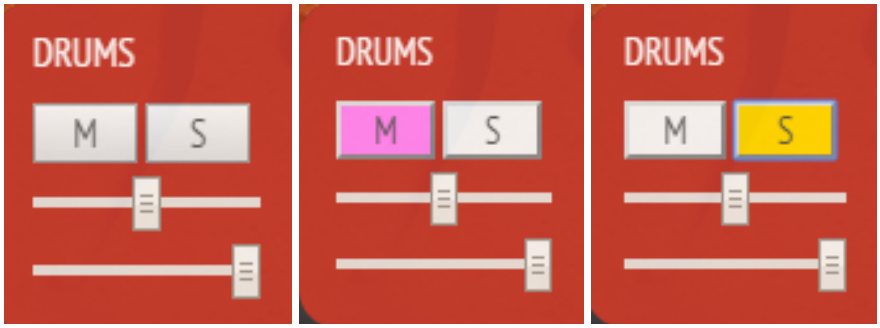
The bar on top of the waveforms contains the playback buttons, which perform actions to Play, Pause or Stop. Additionally, the tracks can be fast forwarded or rewind. The actions are labeled over Screenshot 3.3.2.2.3.



Screenshot 3.3.2.2.3: Playback controls with their actions labeled.

The library used to visualize the waveforms is called Wavesurfer[4] and it is an audio visualization framework implemented in JavaScript. Despite each audio track is treated independently, a synchronization event had to be created in order for the four tracks to respond to the playback events together and play in sync as in the original song arrangement. Doing this also enables the cursor to be placed at any point in the song by clicking over any track with the mouse.

The mixing controls consist of two buttons and two sliders that are located on the track next to its corresponding label. A detailed view is provided in Screenshot 3.3.2.2.4.



Screenshot 3.3.2.2.4: Mixing controls of an audio track.

- Buttons: The button indicated with the letter M stands for Mute and the button indicated with S stands for Solo. These two actions are mutually exclusive, meaning that if a track cannot have both states active simultaneously. Setting a track to Mute means that the track is in silence, while setting a track to Solo activates the current track and sets the rest to mute.

- Sliders: The top slider is the Panning, which controls the relative angle between the source and the listener. The second slider is the Volume of the track, compressed within the range [0,1].

[4] Wavesurfer JS Documentation: https://wavesurfer-js.org/docs/

4. DATASETS

4.1. DSD100

DSD100 (Demixing Secrets Dataset)[5] is a commonly used resource in the field of Source Separation. It has been designed to evaluate signal source separation from music recordings from a variety of genres. For this reason, DSD100 is the official Dataset of the MUS Challenge within the Signal Separation Evaluation Campaign (SiSEC)[6]. This challenge consists of separating professionally produced music.

The Dataset is composed of a total of 100 songs, from which 50 are used as training samples and the other 50 are used to test the model's performance. Each song has 4 sources (bass, drums, vocals and other), which are the ideal separation of the song instruments. The directory organization is structured distinguishing from Mixtures and Sources at the first level and between Dev and Test subsets inside both categories, as shown in Figure 4.1.1.

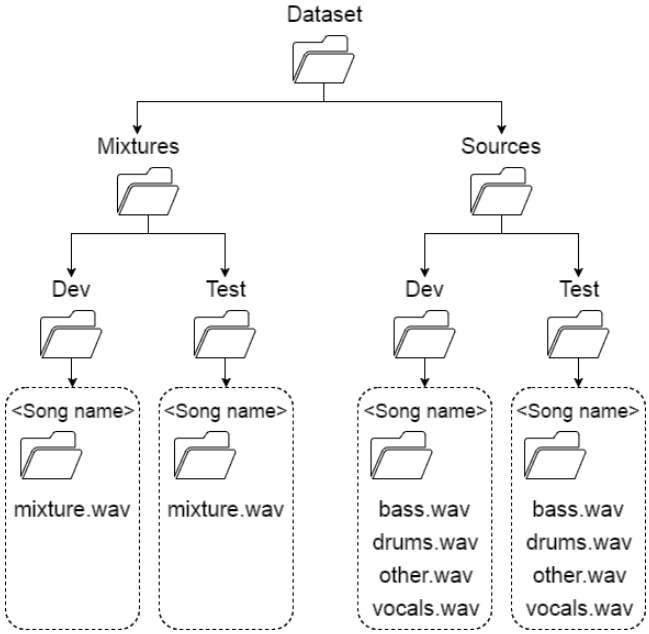


Figure 4.1.1: Hierarchic structure of the DSD100 dataset. The Dev and Test directories are the ones used for training and evaluation respectively. Notice that the songs inside Mixtures and Sources must have the same names in both branches and must be in the same directory, either Dev or Test.

4.2. Proposed Dataset (HHDS)

The system described in Source Separation Stage, in Section 3.2, has been trained using a compilation of Hip Hop songs as a dataset, which will be referred to as HHDS [7] from now. The structure of the dataset follows the convention of DSD100. The proposed dataset has been arranged to have the same structure as DSD100 to make it easier to

[5] Demixing Secrets Dataset (DSD100), SiSEC2016: http://liutkus.net/DSD100.zip
[6] Signal Separation Evaluation Campaign (SiSEC): https://sisec.inria.fr/
[7] HipHop Dataset (HHDS), Héctor Martel: goo.gl/5Hu51y

work with HHDS as well. The main difference of the dataset used for these experiments with respect to DSD100 is that in this case there are Hip Hop songs only, instead of many genres.

This dataset contains the separated tracks for the categories of bass, drums, vocals and others, which are monophonic WAV files with a sampling rate of 44100Hz. The mixture is calculated by summing the four tracks and normalizing the result without adding any effects or performing other nonlinear operations. This is not a common practice in a real mixing case, but the problem is simplified significantly under this assumption.

The total number of songs included in this dataset is 18, from which 13 are used for training and 5 are used for evaluation. They are representative examples of independent underground HipHop songs because they have been produced in a home studio, as the vast majority of nowadays releases in this genre. A total of 12 artists have recorded lyrics over the beats of 1 producer to make the 18 songs of HHDS. For all the songs the main language of the lyrics is Spanish, even though some little fragments in English may be found as well. The following table shows a list of the songs contained in the HHDS dataset, together with the artists, the duration and the subgenres they belong to within Hip Hop music. The first column refers to the global ID that is given to each song during the experiments, although they are listed in alphabetical order here. The tempo range is from 84 BPM to 94 BPM, and the average duration of a song is 2 minutes and 53 seconds.

ID	Song name	Artist(s)	Subgenre	Folder	Tempo	Duration
0	Thomarraso	Almablack	90s Boombap	Dev	92.0	1:28
1	Encontrandome	Dani	Commercial	Dev	90.0	4:16
16	MVP	Big-L	90s Boombap	Test	90.0	2:51
17	Intro	Dani	Commercial	Test	90.0	1:16
2	Yo aprendí	Portaboss; Sou	Melodic	Dev	90.0	2:34
3	No Change	Krash	Hard	Dev	92.0	2:28
4	One Player	Portaboss	Hard	Dev	91.0	1:47
5	Never die	El Extranjero; Raekine	90s Boombap	Dev	88.0	2:18
6	Alma	Sou	Reflexive	Dev	92.0	1:33
18	Altoke03	Sou	Reflexive	Test	90.0	2:20
7	Altoke04	Sou	Reflexive	Dev	84.0	3:27
8	Aquella foto vuela	Sou	Reflexive	Dev	90.0	2:52
9	Distinta sangre	Dosk; Retv; West; Wos	Melodic	Dev	93.0	4:29
19	Cadena Perpetua	Wos	Dark	Test	94.0	3:04
20	Es por mi	Wos	Dark	Test	90.0	3:47
10	Mi Generación	Wos	Melodic	Dev	92.0	3:34
11	Sonrisa de caries	Wos	Dark	Dev	90.0	2:31
12	Ausencia	Portaboss; Sou; Wos	Melancholic	Dev	88.0	5:30

Table 4.2.1: List of the songs contained in HHDS.

Moreover, some songs from DSD100 that are also labeled as HipHop were included in the Test subset. The purpose of this is to do cross-validation of the models with songs from another dataset to see how accurate is the generalization achieved. Some features of these songs are radically different from the ones in HHDS concerning the tempo range, the kind of instruments used and the languages, among others. These songs are shown in Table 4.2.2:

ID	Song name	Artist(s)	Subgenre	Folder	Tempo	Duration
13	Easy Tiger	Animal	UK HipHop	Test	126.0	3:25
14	My Own	Little Chicago's Finest	N/A	Test	108.0	4:42
15	Sing With Me	Side Effects Project	West Coast	Test	115.0	4:04

Table 4.2.2: List of the HipHop Test cases contained in DSD100.

It is also important to analyze how the songs are composed in terms of instruments and structure, since this is crucial to optimize the behavior of the network to separate the sources. All the songs have in common that they can be split into the 4 instrument categories of *bass*, *drums*, *vocals*, and *others*. However, the category *others* in this case is the one actually containing the main melody and harmony.

In terms of structure, the songs might vary depending on the number of artists and whether there are choruses or not. Typically there are 2 or 4 bars of intro where the only sources present are the melody and some percussive sounds from the drums. Then, there might be a chorus of 4 or 8 bars long or a verse from 8 to 16 bars long, and these elements can be repeated later in the song. To reinforce the chorus over the verses one more instrument layer is usually added. Finally, there is the outro to close the song for the last 2 or 4 bars.

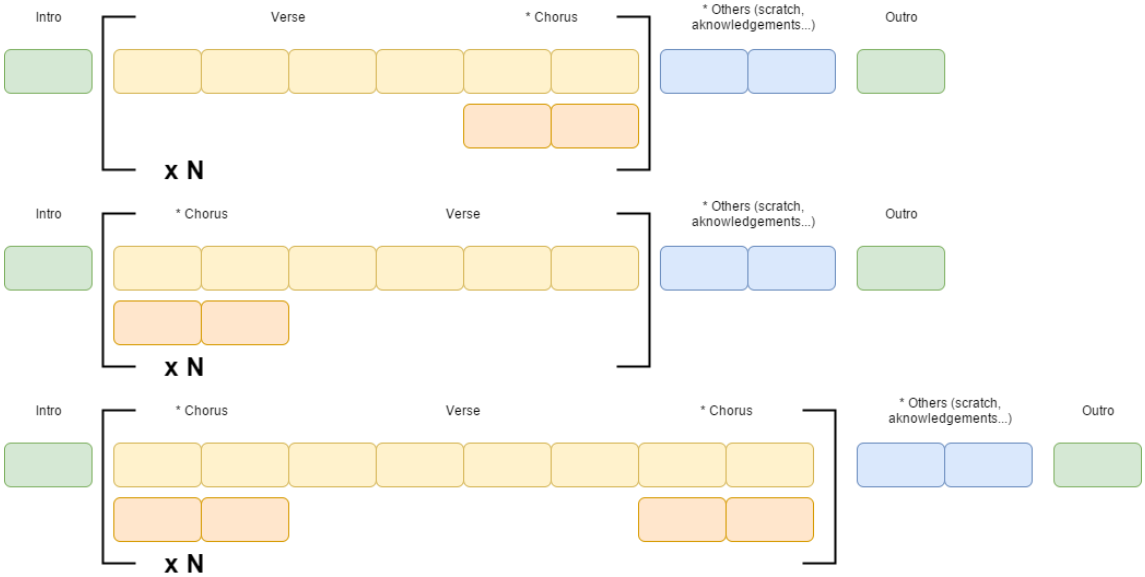


Figure 4.2.2: Generalization of the structure of a HipHop song, where N can be an arbitrary number of repetitions. Each cell represents multiple of 2 or 4 bars. Notice that the parts marked with an asterisk such as Chorus and Others are optional, so they might not be present in every song.

4.3. Data Augmentation Techniques

The performance and the level of abstraction that can be achieved during the training phase of a Neural Network depend on the variety and the amount of available training data. This section covers data augmentation techniques that can be useful when a dataset has a very reduced set of examples.

There are some techniques that can be applied to the dataset in order to extend the number features and, thus, maximize the results without the need of including any additional examples [35]. This augmentation must be carefully carried out because it can cause the opposite effect and alter the data by generating unrealistic examples, making the network prone to overfitting and error. Nevertheless, data augmentation serves as a way to obtain new representations that are still statistically relevant when done appropriately [36], [37].

The methods used and tested in this work are presented here, although the names are just intended to be intuitive without following any convention. The reader can find the data augmentation methods included in the DeepConvSep framework in [38]. One of them has achieved success, while the other two require more refinement to be effective with the current system.

4.3.1. Instrument Augmentation

All the songs have 4 sources, as described in DSD100, in Section 4.1. One possible way to obtain new mixes is to set some of them to zero instead of summing them with the rest. Let n be the number of sources and k the number of sources which are muted. The total amount of augmentations, denoted by N, can be expressed as:

$$N = \sum_{k=1}^{n-1} \binom{n}{k} = \sum_{k=1}^{n-1} \frac{n!}{k!\,(n-k)!} \tag{4.1}$$

Therefore, 1 song yields to obtain a maximum of $N + 1$ training examples, provided that the additional one is the original mix. Figure 4.3.1.1 illustrates how this augmentation can be done for one song.

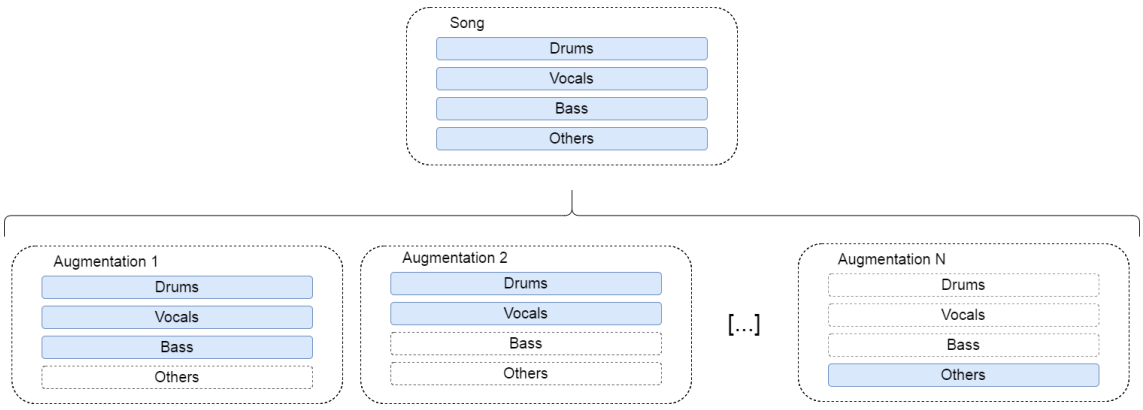


Figure 4.3.1.1: Example of instrument augmentation performed with 1 song. Tracks in blue are originally from Song 1, while tracks with no fill are set to mute.

This approach was introduced because some tracks in the Test cases of the dataset had long silences, which were not appropriately modeled by the network when trained with complete songs. The fact of muting one or many instruments results into silences to be modeled and null filters to be learnt, at least in theory. In fact, this is a binary simplification of the Dynamics augmentation approach described in [38]. During the experiments, this data augmentation method was discarded because the error committed on each epoch could not be minimized effectively. The main reason is that the expression of the training cost has not been designed to contemplate the scenario of a particular source not playing. This issue is commented in further detail in Problems found, in Section 6.2.

4.3.2. Mix Augmentation

Following the same principles as the last approach, this consists of combining the individual sources to form new mixes. This time, however, none of the instruments is set to mute. The strategy of mix augmentation consists of calculating the features for the original mixes, plus some permutations combining tracks from various songs.

The main concerns are the harmony, the length, and the tempo of the songs, since combining different songs may result in a piece that is not possible to mix or not representative. Notice that the impact of the first two depends on the time context of the analysis. Under the assumption of a small enough time frame, the tempo and the harmony can be considered locally irrelevant. The strategy of mix augmentation is illustrated in Figure 4.3.2.1, with an example showing only 2 songs for simplicity.

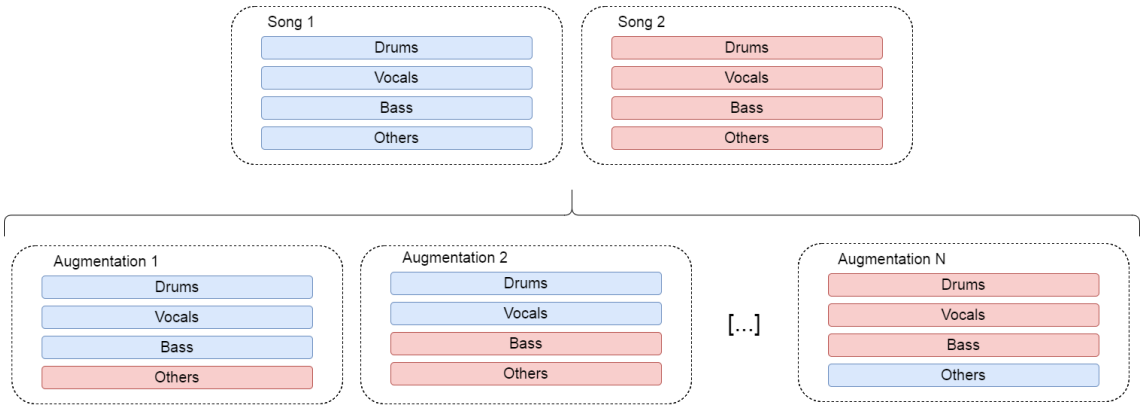


Figure 4.3.2.1: Example of mix augmentation performed with 2 songs. Tracks in blue are originally from Song 1, and tracks in red are from Song 2.

In the experiment, the songs were selected in sets of 4, so that one track is extracted from one song at once. Then they are mixed in all possible combinations without repetition, and these combinations are picked randomly with sampling factors of 1% or 10% to avoid generating a big amount of feature data. While the model obtained has shown to be robust providing stable results, its peak performance is not comparable to the other models commented in Training, in Section 5.1. This issue is also analyzed in further detail in Experiment Discussion, in Section 6.1.

4.3.3. Circular Shift

Another augmentation technique is to apply Circular Shift to the audio tracks to generate more samples, as proposed in [38], although the approach of Circular Shift is generally used in the field of Image Processing [36]. Under the context of audio Circular Shift consists of rearranging the spectrogram frames from a track so that the first or last frame is moved to the opposite extreme and the others are displaced one position. A circular shift is a special kind of permutation called cyclic permutation because the starting point is reached again after a certain number of iterations. An example of circular shift for one audio track is shown in Figure 4.3.3.1.

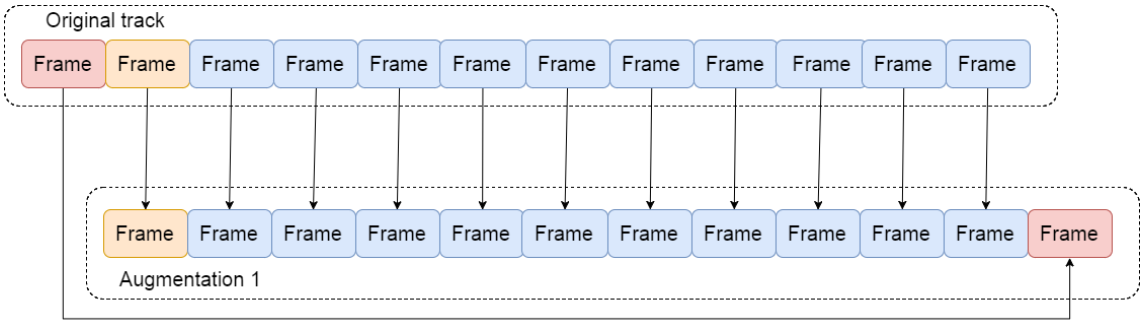


Figure 4.3.3.1: Example of circular shift for one audio track. The original audio on top is shifted by moving the first frame (in red) to the end. Each block corresponds to a frame, and the next iteration would be to move the orange frame to the end.

Notice that the number of permutations of an audio track is equal to the number of frames. The implication of this operation, when applied to a song, is that the continuity is preserved while note durations and rhythm patterns can be altered. Therefore, new songs are generated with a valid structure and harmony that can be used to train the model in more general cases.

A frame size of 200 ms was considered for performing the circular shift to all the tracks in the experiment. However, they were shifted by different amounts according to a combination of shift parameters previously computed. This way, the temporal alignment of the instruments is slightly modified, which helps to extract individual onsets. The model trained with this approach is theoretically more robust than the one trained with the non-augmented version of HHDS, and its peak performance arguably resulted in the best of the models tested. A more in-depth analysis can be found in Experiment Discussion, in Section 6.1.

5. EVALUATION

5.1. Training

As it has been commented in Proposed System, in Section 3.1, the first step is to get the input data from the dataset is to compute the STFT and get the spectrograms of the sources and the mixture. Then, the spectrograms are passed through the network iteratively for a certain number of epochs.

The frameworks Theano[8] and Lasagne[9] are used to build the network architecture on top of the Python programming language, and it has been trained in a computer at the Music Technology Group Department at UPF. The reader can find the implementation in the DeepConvSep[10] repository on GitHub maintained by Marius Miron, Pritish Chandna and other collaborators from the MTG. The code is executed on the GPU instead of on the CPU in order to allow for parallel computation on a machine with the following hardware specifications:

GeForce GTX TITAN X GPU
Intel Core i7-5820K 3.3GHz 6-Core Processor
X99 gaming 5 x99 ATX DDR44 motherboard

5.1.1. Training parameters

The training phase has been carried out using the HHDS Dataset and the parameters specified in Table 5.1.1.1. For each parameter, the name is shown together with the place in the code where it is assigned, the variable name (if applicable), and the value it has been given.

Place in the code	Parameter name	Variable name	Value
Initialization	Batch size	batch_size	32
Initialization	Batch memory	batch_memory	200
Initialization	Time context	time_context	30
Initialization	Overlap	overlap	25
Initialization	Number of processors	nprocs	7
Initialization	Number of epochs used for training	nepochs	40
Initialization	Scale factor	scale_factor	0.3
First conv. layer	Frequency bins (f1)	feat_size	513
First conv. layer	Time indices (t1)	time_context	30
First conv. layer	Number of filters (N1)	num_filters	50
First conv. layer	Filter size (t1,f1)	filter_size	(1,513)
First conv. layer	Stride of filters (S1)	stride	(1,1)

[8] Theano reference documentation: http://deeplearning.net/software/theano/
[9] Lasagne reference documentation: https://lasagne.readthedocs.io/en/latest/
[10] DeepConvSep repository on GitHub: https://github.com/MTG/DeepConvSep

Second conv. layer	Frequency bins (f2)	N/A	1
Second conv. layer	Time indices (t2)	time_context	30
Second conv. layer	Number of filters	num_filters	50
Second conv. layer	Filter size (t2,f2)	filter_size	(30,1)
Second conv. layer	Stride of filters (S2)	stride	(1,1)
Bottleneck layer	Number of hidden units (NN)	N/A	128
Cost function	Alpha component (α)	alpha	0.001
Cost function	Beta component (β)	beta	0.01
Cost function	Beta component for vocals (β_{vocals})	beta_voc	0.03

Table 5.1.1.1: Parameters used in the training phase, with their respective variable names and values.

5.1.2. Models

During the training phase, 5 models have been generated in order to compare the performance later in the experiments presented in Testing, in Section 5.2. The main idea behind these experiments is to evaluate a general purpose music source separation model trained with DSD100, three models trained exclusively for HipHop source separation using HHDS and one hybrid trained with both datasets. The objective is to determine which one gives the best performance for the application at hand. The models are generated using the Pickle library for Python, which serializes the network object and exports it in .pkl format so they can be used by another Python application without having to be recomputed. The following table shows a comparison of the 5 models.

model_dsd_fft_1024_Marius.pkl	Trained with DSD100 songs only, used as a reference for general purpose music source separation model.
model_dsd_hh_fft_1024.pkl	Trained from the raw DSD100 model adding the songs of HHDS in a new training phase. It is intended to work as a more specialized model respect to the reference but keeping its main generalization.
model_hh_fft_1024.pkl	Trained with HHDS songs only, used as a specialized case.
model_hh_aug_fft_1024.pkl	Trained with HHDS songs only with Circular Shift augmentation to extend the training samples, used as the most specialized case.
model_hh_mix_aug_fft_1024.pkl	Trained with HHDS songs only with mix augmentation to extend the training samples, used to test the effectiveness of this data augmentation strategy.

Table 5.1.2.1: Models generated during the training phase with their respective filenames, accompanied by a brief description. They have been trained with 50 songs in

case of DSD100, 13 songs in case of HHDS only and 50 (first stage) + 13 (second stage) songs in case of DSD100 + HHDS.

5.2. Testing

The Songs contained in both Test subsets were passed through the network to verify that the separation can be achieved in cases different from the training data. The objective evaluation measures SDR, SIR and SAR have been computed for all the songs in the dataset to compare the performance according to different criteria.

In addition to this, the output results have been analyzed by inspection and active listening to determine whether the possible distortions that may appear in the separated sources are acceptable for a human listener. In that case, they would be considered suitable for the proposed remixing application. The models reported in the following results and in Table 5.1.2.1 are only the ones which provided a reasonable compromise between separation and audio quality. For instance, one attempt of Instrument Augmentation was discarded because the error committed on each epoch could not be minimized effectively when instruments are muted. This issue is commented in Problems found, in Section 6.2.

As a starting point, it is important to clarify how the results are presented. Because of the variability between the training data and the test data, the results are always displayed as separated sets Dev or Test. Moreover, the Test results are split into TestDSD100 and TestHHDS, since the HipHop test cases from DSD100 were used in order to do cross-validation on the results from HHDS. It is always expected that the numerical values of the Test data are lower than Dev given that the test cases are unknown by the models, unlike the training data for which the models have been optimized. The following three plots show the comparison side by side of the three measures introduced above.

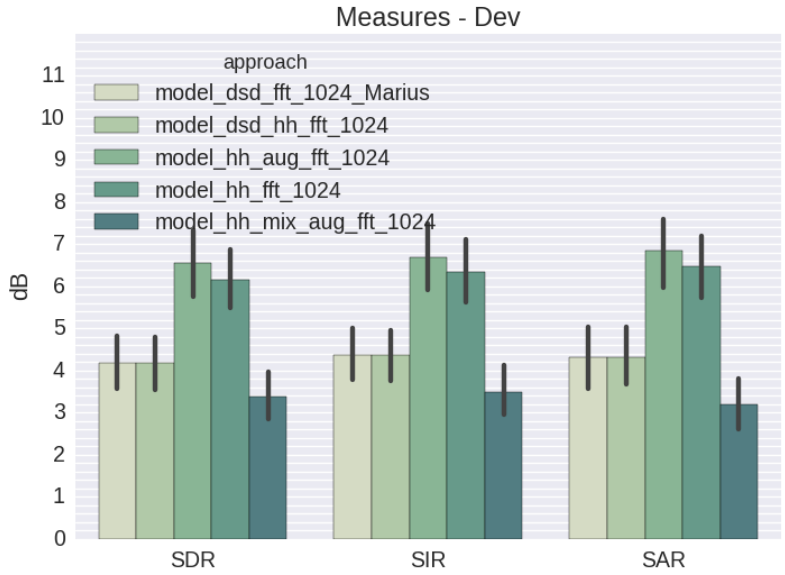


Figure 5.2.1: Bar plot of the evaluation measures SDR, SIR and SAR for the Dev subset. The hue represents the approach from the models described in Table 5.1.2.1.

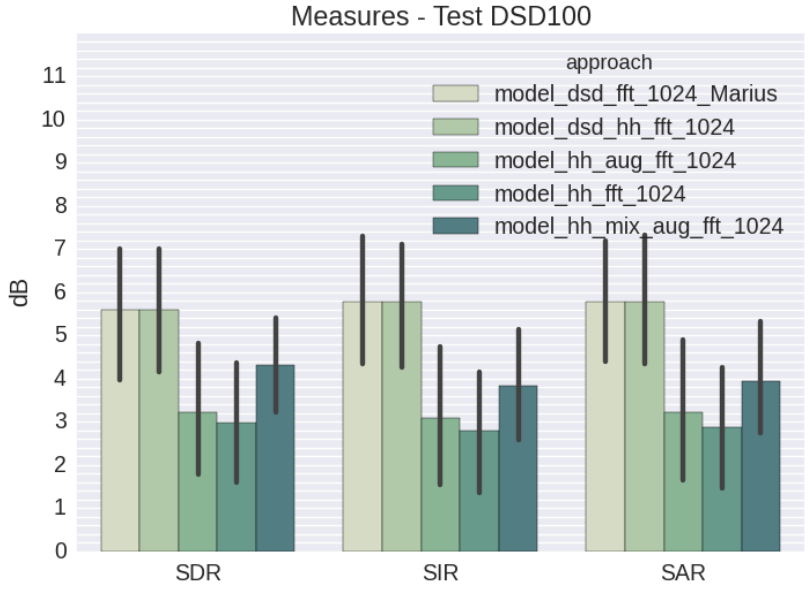


Figure 5.2.2: Bar plot of the evaluation measures SDR, SIR and SAR for the TestDSD100 subset. The hue represents the approach from the models described in Table 5.1.2.1.

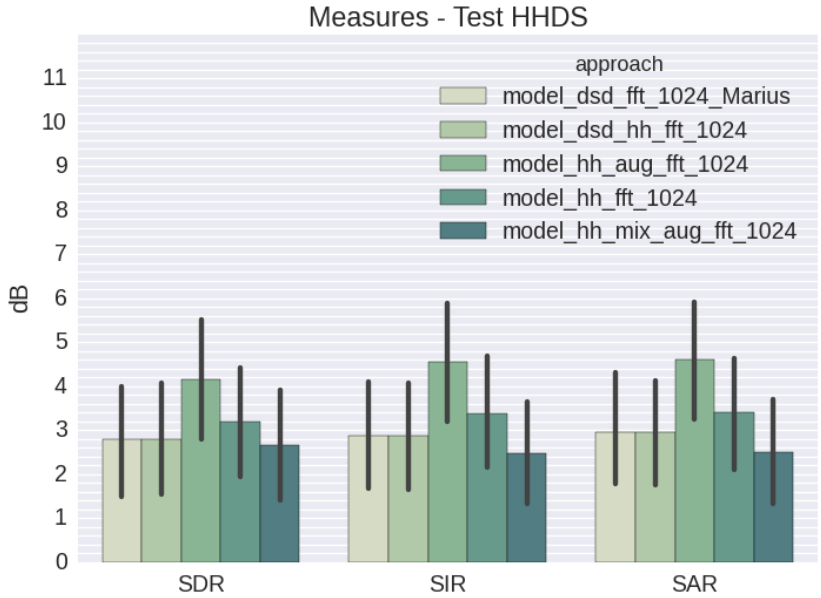


Figure 5.2.3: Bar plot of the evaluation measures SDR, SIR and SAR for the TestHHDS subset. The hue represents the approach from the models described in Table 5.1.2.1.

As mentioned in the Evaluation of Source Separation, in Section 2.1.3, the SDR is the metric that relates to the overall performance of the source separation approaches and, thus, it will be the main focus in the experiments presented here. The following figure shows the SDR for the subsets Dev and Test side by side to compare the values.

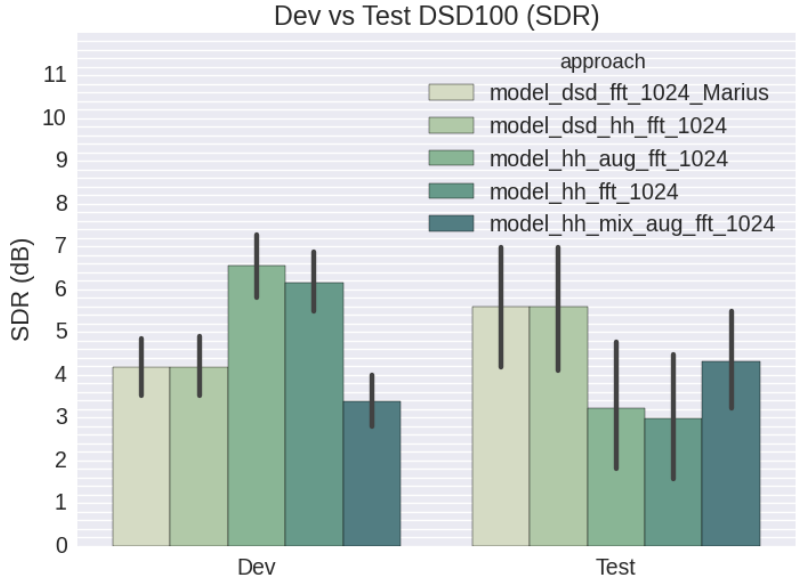


Figure 5.2.4: Bar plot of the evaluation SDR measure comparing the Dev and TestDSD100 subsets. The hue represents the approach from the models described in Table 5.1.2.1.

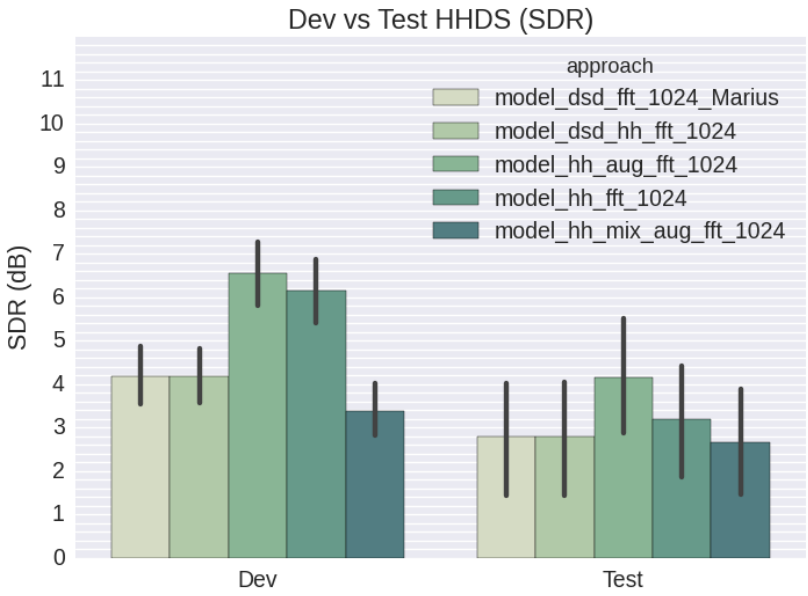


Figure 5.2.5: Bar plot of the evaluation SDR measure comparing the Dev and TestHHDS subsets. The hue represents the approach from the models described in Table 5.1.2.1.

It can be observed that the Dev values are higher than the ones in Test, as expected. Despite this condition, the results show that the performance is quite acceptable since the SDR difference is about 2.0 dB between the two subsets and holds for all five approaches. This means that there is stability among the models and, therefore, overfitting has not had a significant impact on the results.

To check that this observation is effectively the general case, another thing that can be done is to consider each song individually. This is also a good practice to see if one song in particular is causing an undesired behavior, and try to understand why it is happening. The following figures show the values of SDR for the songs in Dev and Test. The global ID of the song in the dataset corresponds to the alphabetical order inside their respective subsets and is represented along the horizontal axis.

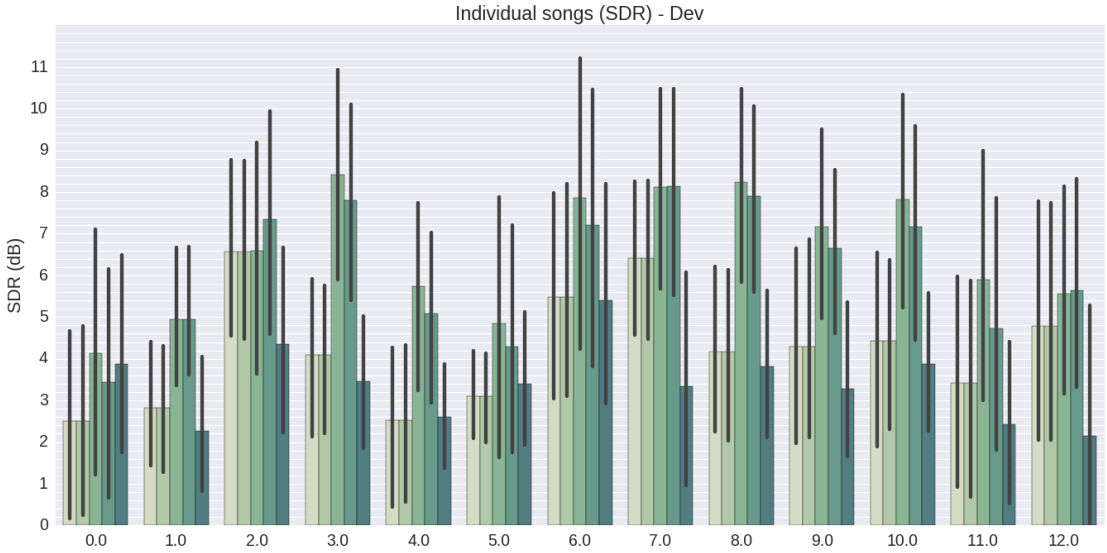


Figure 5.2.6: Bar plot of the evaluation SDR measure comparing the individual songs in the Dev subset. The hue represents the approach from the models described in Table 5.1.2.1.

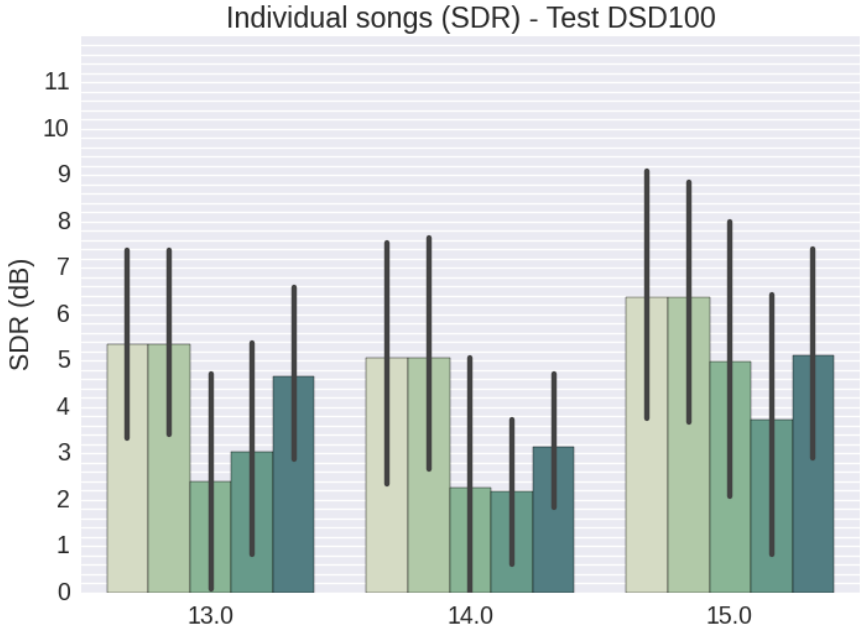


Figure 5.2.7: Bar plot of the evaluation SDR measure comparing the individual songs in the TestDSD100 subset. The hue represents the approach from the models described in Table 5.1.2.1.

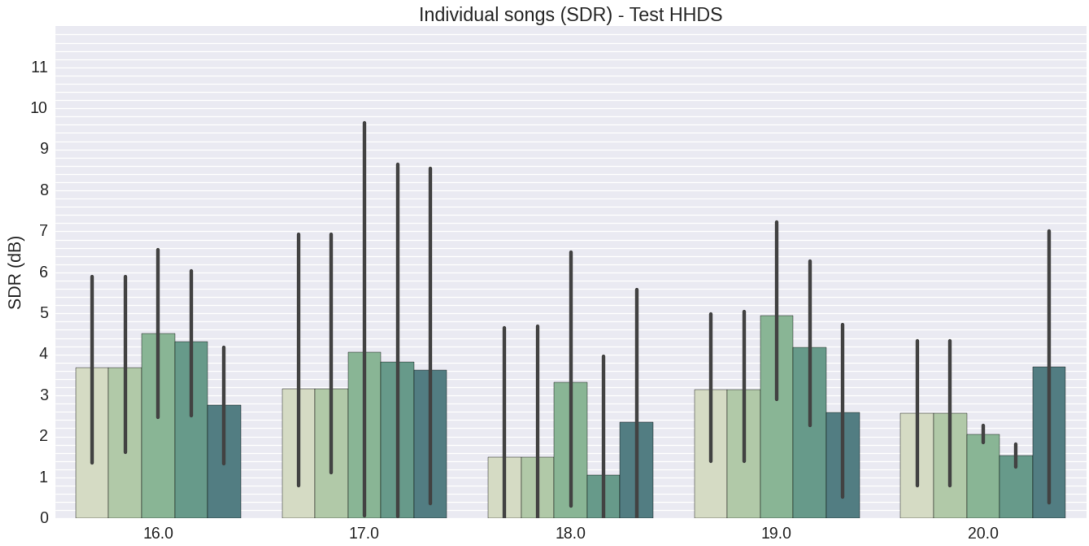


Figure 5.2.8: Bar plot of the evaluation SDR measure comparing the individual songs in the TestHHDS subset. The hue represents the approach from the models described in Table 5.1.2.1.

The approach using Circular Shift augmentation over HHDS seems to be the best of the five approaches tested, followed by the raw HHDS and then the ones trained with DSD100 for the majority of songs analyzed.

For the purpose of the experiment, in the HHDS Test subset, there are 4 songs that have vocals in Spanish and from the same artists used during the training phase. One additional song, with ID no. 16, has been included with vocals in English and from an artist that is not present in the training data. The results for this song are on par with the others from HHDS, but this can be caused by the particular timbre characteristics of the artist's voice or any other factor rather than the language. Although the numbers are an indicator of a good estimation, it cannot be affirmed given the limited number of testing data examples.

Another interesting insight is given by the analysis of the individual sources. Some applications may require better performance in particular instruments while tolerating some amount of error in others. This may be the case of extracting the vocals to obtain the instrumental of the song or, on the contrary, of obtaining the isolated vocals. In the first scenario, the distortions in the vocals and some instruments may be tolerated, whereas in the second it is more critical since any distortion can affect the intelligibility of the voice. The following figures show the SDR values for the bass, drums, vocals, and others in the Dev and Test subsets separately.

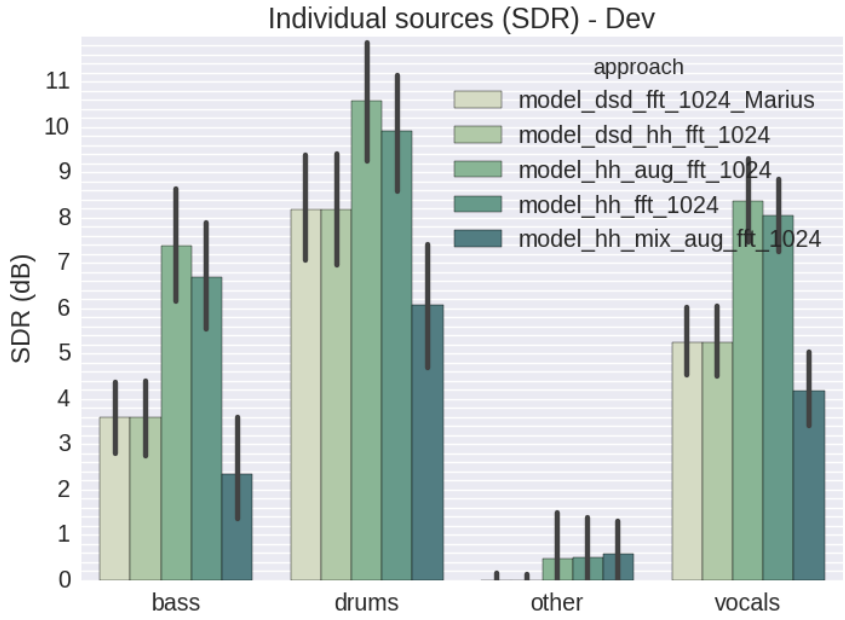


Figure 5.2.9: Bar plot of the evaluation SDR measure comparing the individual sources in the Dev subset. The hue represents the approach from the models described in Table 5.1.2.1.

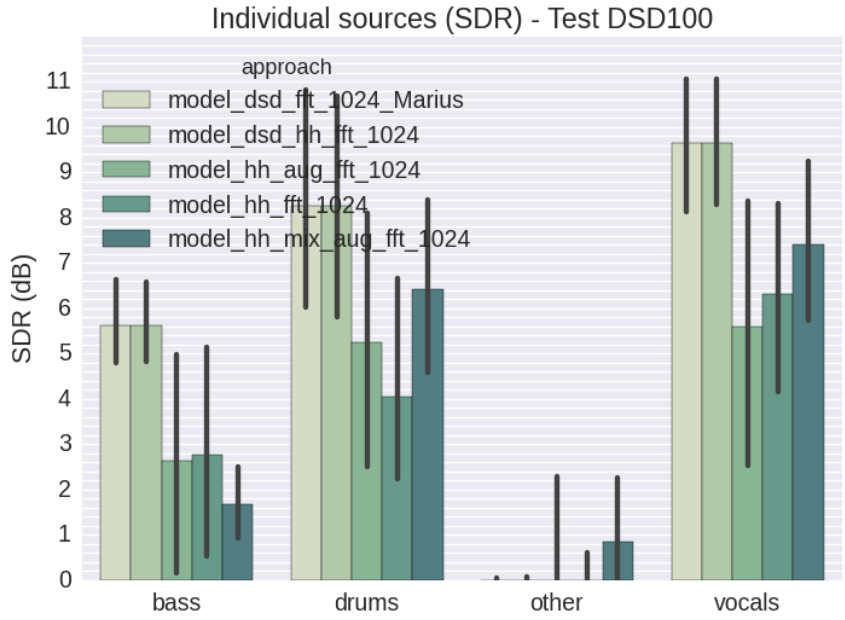


Figure 5.2.10: Bar plot of the evaluation SDR measure comparing the individual sources in the TestDSD100 subset. The hue represents the approach from the models described in Table 5.1.2.1.

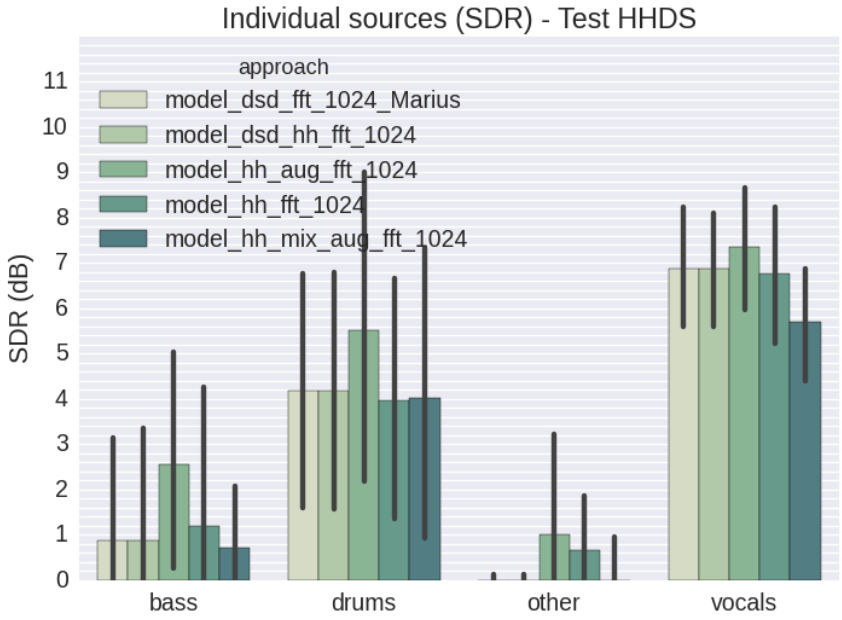


Figure 5.2.11: Bar plot of the evaluation SDR measure comparing the individual sources in the TestHHDS subset. The hue represents the approach from the models described in Table 5.1.2.1.

It can be seen that there are significant differences between the Dev and the Test results, reaching up to 5.5 dB in the worst case. The sources that are better estimated, and which estimation can be considered relevant here, are the vocals and the drums. The behavior of the bass estimation is not very reliable since it presents a variation that reaches negative values, as it happens with the track other.

6. CONCLUSIONS

In this section, the results will be analyzed and discussed in order to propose solutions to the issues found during this work. Also, some ideas for further work on the field will be commented, since there are always some aspects that can be improved in the future.

6.1. Experiment discussion

According to the results presented in the last section, it has been shown that the best performance among the experimental approaches is achieved using a model specifically trained for the task of separating HipHop music from HHDS. The general purpose models trained with the DSD100 dataset obtained an SDR of 2.0 dB less than HipHop only methods in this task. However, the fact of having more than double of training data examples in DSD100 than in HHDS makes the general purpose methods more stable against different kinds of Test songs regardless the genre. This can happen even though they are all HipHop songs, as it can be extracted from the DSD100 test cases used for cross-validation. This can be seen in Figure 5.2.4 and Figure 5.2.5, in Section 5.2, that shows the comparative results of SDR for the Dev and Test subsets. The difference between the Dev SDR and the Test SDR is reduced when the methods have achieved a more abstract representation of the data, so they can generalize it to obtain similar SDR results in both subsets. This means they can perform well under unknown test data.

A drastically lower SDR in the Test subset respect to the Dev is an indicator of overfitting. Data augmentation, as well as changing the learning parameters, can be used to avoid it. The model trained with Circular Shift augmentation is, nevertheless, the one that presents the larger decrease. As it has been mentioned before, this can be due to the fact that the training data is very limited and, thus, the augmentations do not lead the network to learn a generalization robust enough. It is also worth mentioning that even in this case the performance in terms of SDR is 0.75 dB higher than the non-augmented HHDS model, and 1.5 dB higher respect to the other approaches.

The trend is practically the same for all the songs, as shown in Figure 5.2.6, Figure 5.2.7 and Figure 5.2.8, in Section 5.2, except for songs 13, 14 and 15, for which the models trained only with HipHop give worse results than the ones trained with DSD100. The individual case of song 20 should be further analyzed to determine why the trend is inverted and if there are any similarities or differences with respect to the reference training data. Given the limited amount of training and test cases, at the moment it is not possible to establish the reason statistically and with enough confidence.

The differences in the evaluation results of TestDSD100 and TestHHDS may be partially ought to the type of production. In this sense, while the models trained with DSD100 tend to find a more general solution, the ones trained with HHDS are overfitted to one specific production scheme. On the one hand, DSD100 has various music genres as commented in DSD100, in Section 4.1. On the other hand, the genres in HHDS are very similar, so that the resulting models are too specialized and cannot capture the broad essence of HipHop music. Overfitting for HHDS with Circular Shift and the non-augmented model is manifested in the 0.75 dB decay between TestHHDS and TestDSD100, which can be seen in Figure 5.2.4 and Figure 5.2.5, in Section 5.2.

The size of both datasets is also a very important factor. Taking into account that DSD100 has 50 training songs while HHDS has only 13, it is expected that the former perform better than the latter. Machine learning algorithms typically need large datasets to be able to extract significant features from the data, even though some *One-Shot Learning* approaches have been recently proposed [39]. Having fewer songs means that the internal representation in the network has a lower degree of complexity and, thus, it is easier to fit in the training data. Eventually, this causes the model to deviate from the abstraction of the task rather than to perform it better.

From the results presented in Testing, in Section 5.2, it can be extracted that the number of songs present in HHDS should be increased together with the diversity of examples in order to obtain a more reliable performance. The deviations obtained so far make it difficult to establish statistically significant comparisons between the tested approaches. However, the effect of the augmentation done on the data of HHDS is very noticeable and has caused a positive impact on the results. The two augmentation techniques of Circular Shift and Mix Augmentation are discussed.

- The *Circular Shift* technique provides with more robust time-energy features by changing the alignment of the instruments. Because of this, the overall performance of the model trained using this technique and 13 songs is comparable with the performance of a model trained using 50 songs, which is a remarkable achievement. This is especially relevant for the separation of the drums and the bass, which carry the most prominent onsets. Circular Shift has resulted in the best performance of the Test over HHDS, showing an improvement respect to the first two models with enough significance in terms of all the evaluation metrics, SDR, SIR and SAR. In the test cases from DSD100, it is still 2.0 dB SDR below the first two models, since the shift can only affect the time characteristics of the training examples that are in HHDS, and these are remarkably different from DSD100 in terms of harmony and instruments.

- The *Mix Augmentation* technique has shown to provide a very stable result, despite its SDR values are always below the rest of the models except for the DSD100 test cases. The mixes generated using this technique are indeed unrealistic, because the harmony and the tempo of the different tracks do not match. Consequently, the time-frequency correlations are not very well defined in the augmentations, leading some of the learnt features to never be found in actual songs from the test cases. Also, active listening has determined that this model is the one that introduces more audible artifacts. Nevertheless, the fact of generating more songs with this strategy helps to prevent overfitting. To improve the quality of the separation results while maintaining the generalization, one possible solution would be to apply time stretching and pitch shifting to the tracks before mixing them together. This solution would find a compromise in the augmentations to generate only realistic mixes.

6.2. Problems found

Some issues have been found during and after the execution phase of this study, which are exposed in this section as a proposal for further development of Audio Source Separation related works.

First, there are problems in the estimation of sources when there are muted sections in the song. The evaluation metrics assume that the energy of the sources should never be equal to zero, because otherwise the solution calculated from the ratios is not mathematically defined. As a consequence, the computation of the evaluation measures caused NaN values to appear locally in the fields where a section is muted in the song. In this situation a singular matrix is generated, and the resulting linear system cannot be solved. This means that no decomposition for e_{interf}, e_{spat} and e_{artif} could be found. Recall the formula for decomposing the estimation based on the target source and distortion terms, from Evaluation of Source Separation in Eq. 2.6.

$$\hat{s}_i = s_{target} + e_{interf} + e_{spat} + e_{artif} \tag{2.6}$$

In order to avoid the singularity, some random perturbation can be added to the matrix. Since the determinant is no longer equal to zero an inverse matrix must exist, and the numerical results are approximately equal for non-zero regions if the amount of noise is small enough. However, this solution leads to creating more deviation in the data because outlier values appear in the zero energy regions. It might be interesting to propose a modified version of the Source Separation Evaluation metrics from [13] to contemplate this case.

Due to some irregularities of the proposed dataset, there was one song in which the track "other.wav" played for the introduction and then stayed in silence for the rest of the song, whereas the rest of the sources were introduced in the mix at the exact position where the fade-out happened. Given that this condition can rarely appear in a real case scenario, and that the model was trained with songs in which the energy distribution is more homogeneous, it produced estimations for all four tracks even though there was one in mute. In this case, any estimation different from zero would cause a considerably high error in the outcome of the muted track, affecting to the evaluation results.

The Data Augmentation technique of Instrument Augmentation has been proposed as a possible solution for this issue. It would be done by computing the features resulting from muting individual tracks in all the possible permutations, as described in Data Augmentation, in Section 4.3. However, none of the experiments performed during this work can confirm its effectiveness. The main problem with this approach is the difficulty of training the network to learn not only how to discriminate one instrument from another, but also whether one instrument is playing or not. There are two main changes that can be made to the current method to meet this new requirement.

- Side information, such as labels, can be computed along with the spectral features of the training data to indicate if any given instrument is playing or not. This information can be fed into the network as another input and be learnt together with the instrument spectra. This approach would allow for instrument detection and make the method suitable for more flexible source separation scenarios. At the same time, it would require implementing major changes in the organization of the neural network to accommodate more inputs and outputs and perform a more complex task, possibly compromising performance and adding some extra computing cost.

- The minimization of the training error can be done taking into account only the error committed in the instruments that are playing and discarding the rest, which are in silence and, thus, have no contribution to the mix. Since the

features are organized in batches and randomized, it is not a trivial task to assign zero cost to an instrument when it is not playing. Although it is possible to sort the features in some ways to get a more deterministic result, some changes should be made to the organization of the training data and the training phase.

6.3. Final remarks

The overall results of this work are adjusted to the objectives proposed at the beginning of this document. A Deep Learning approach to separate HipHop music has been successfully developed and tested using a proposed dataset. The results are acceptable in some of the use cases it has been conceived for, like live manipulation of the sources, while it can be improved to cover more demanding needs for professional audio applications.

Until now in this chapter, the results of the Source Separation stage have been covered. It must be mentioned that the runtime taken to separate an individual song from an existing model is less than the duration of the song. For the cases tested using the Hardware settings described in Training, in Section 5.1, the computation never took more than 20 seconds for songs longer than 4 minutes. This relatively low processing time will be reduced even more as computers get faster and computational power in increased. Also, some optimization of the existing method may help to take this approach to mobile devices and, eventually, it will be possible to embed this system inside an audio rendering standalone unit.

Regarding the 3D Remixing part, it allows for basic manipulation of the sources and, in general, it fulfills the objectives of live manipulation and user interaction with the music. It is also a valid upmixing solution in cases the when the songs need to be reproduced in different layouts. Of course, there are some limitations at the current state of the application. There are two main aspects that are worth considering.

- The instruments can be placed in different positions in the 3D space of the mix with reasonable quality. This is achieved by altering their individual panning and volume parameters. However, there might be occasions in which phase cancellations happen. If an instrument is placed in a position where interferences from the other tracks are noticeable, it will be attenuated. This is because all sources are present in all tracks with a greater or a smaller contribution, given that the separation is not perfect. Therefore, at some specific settings of panning angle and volume, the instruments may vanish. The effect would get less noticeable by reducing the Signal to Interference Ratio (SIR), but it must be noted that more artifacts are introduced as the isolation of the instruments is increased.

- It is not possible to fully suppress a source from the mix. Again, the compromise achieved between the naturalness of the sources and the separation implies that all the sources are present in all tracks. Therefore, even though a particular track is set to mute its corresponding source can still be heard, but at a considerably lower level. Because of this, it can be stated that the remixing capabilities are more oriented to emphasize some instruments with respect to others rather than to perform individual manipulations over the sources. This property cannot be exploited by professional producers to make remixes, but it

can be useful for enthusiasts to analyze the elements of a song and the way it has been composed.

Finally, the fact of making this service available on the web opens new possibilities for music artists to learn from other songs, to make some arrangements to them, to come up with new ideas, and to improve the way they create their own content.

6.4. Future work

There are some aspects about the work that has been carried out that may be changed or improved. The ultimate goal would be to obtain a separation with the minimum interference in which the introduced artifacts and distortions are close to non-audible. This would make the tracks suitable for the remixing application that has been proposed.

It may be interesting to apply some extra filtering stages, such as Wiener filters, to the inputs and outputs of the network in order to reduce the interferences as much as possible without compromising the audio quality. Done appropriately, this approach can increase the SDR up to 2.0 dB more.

Some attempts have been made to expand the capabilities of the current model. Initially, the track corresponding to the source "other" was not included in the cost of the training phase. In other words, it was conceived as the residual part resulting from having extracted the rest of the tracks from the mixture. However, the particularities of HipHop music and other genres like as Electronic or Rock music allow for modeling the melody as well apart from bass, drums, and vocals. Nevertheless, the inclusion of this track in the training cost has led to a worse separation so far in terms of interference among sources. The underlying reason might be that there are more parameters to learn, so the optimization of the cost converges to a solution where the error is evenly distributed among the tracks. Moreover, the aforementioned genres are based on loops of a fixed length that can be detected and exploited using other network architectures. Therefore, further work can be focused on improving the separation of new categories in order to obtain a more complete model of the instruments present in this kind of music.

In fact, the separation can be more flexible and accurate if the problem of instrument detection is integrated into the method, as proposed before in this chapter, in Section 6.2. A large set of instruments can be predefined in the model beforehand, and an instrument would only be extracted if its activation is found.

References

[1] A. Liutkus, “Source separation tutorial ICASSP 2014,” 2014.

[2] C. Cherry, “Some Experiments on the Recognition of Speech, with One and with Two Ears.” 1953.

[3] D. FitzGerald, “Upmixing from mono-a source separation approach,” *Digit. Signal Process. (DSP), 2011 17th …*, 2011.

[4] J. Blauert, *Spatial hearing : the psychophysics of human sound localization*, Rev. ed. Cambridge (Mass.) : MIT Press, 1997.

[5] S. Kraft and U. Zölzer, “Low-complexity stereo signal decomposition and source separation for application in stereo to 3D upmixing,” *Audio Eng. Soc. 140th Conv. 2016 June 4–7, Paris, Fr.*, 2010.

[6] C. Republic and S. Kraft, “Time-domain implementation of a stereo to surround sound upmix algorithm,” pp. 113–120, 2016.

[7] E. Gómez *et al.*, “PHENICX: Performances as Highly Enriched aNd Interactive Concert Experiences,” *SMAC Stock. Music Acoust. Conf. 2013 SMC Sound Music Comput. Conf. 2013*, pp. 681–688, 2013.

[8] Z. Rafii, A. Liutkus, and B. Pardo, “REPET for Background/Foreground Separation in Audio,” pp. 395–411, 2014.

[9] J. J. Burred, “Musical Source Separation : Principles and State of the Art,” no. June, 2008.

[10] A. A. Nugraha, A. Liutkus, and E. Vincent, “Multichannel audio source separation with deep neural networks,” vol. 24, no. 9, pp. 1652–1664, 2016.

[11] M. Miron, J. J. Carabias-Orti, J. J. Bosch, E. Gomez, and J. Janer, “Score-informed source separation for multichannel orchestral recordings,” *J. Electr. Comput. Eng.*, vol. 2016, 2016.

[12] P. Chandna, M. Miron, J. Janer, and E. Gomez, “Monoaural Audio Source Separation Using Deep Convolutional Neural Networks,” 2017.

[13] E. Vincent, R. Gribonval, and C. Fevotte, “Performance measurement in Blind Audio Source Separation,” *IEEE Trans. Audio, Speech Lang. Process.*, 2005.

[14] P. Comon, “Independent component analysis, A new concept?*,” *Organization*, vol. 36, pp. 287–314, 1994.

[15] L. De Lathauwer, B. De Moor, and J. Vandewalle, “An introduction to independent component analysis,” *J. Chemom.*, vol. 14, no. 3, pp. 123–149, 2000.

[16] D. D. Lee and H. S. Seung, “Learning the parts of objects by non-negative matrix factorization.,” *Nature*, vol. 401, no. 6755, pp. 788–91, 1999.

[17] D. Lee and H. Seung, “Algorithms for non-negative matrix factorization,” *Adv. Neural Inf. Process. Syst.*, no. 1, pp. 556–562, 2001.

[18] E. Malmi, P. Takala, H. Toivonen, T. Raiko, and A. Gionis, “DopeLearning: A Computational Approach to Rap Lyrics Generation,” 2015.

[19] Z. Cheng, Q. Yang, and B. Sheng, “Deep colorization,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2016, vol. 11–18–Dece, pp. 415–423.

[20] A. Graves, A. Mohamed, and G. Hinton, “Speech Recognition with Deep Recurrent Neural Networks,” no. 3, 2013.

[21] P. Chandna, “Audio Source Separation Using Deep Neural Networks: Low Latency Online Monoaural Source Separation For Drums, Bass and Vocals,” 2016.

[22] Q. V Le, “A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional

Neural Networks and Recurrent Neural Networks," *Tutorial*, pp. 1–20, 2015.
[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012.
[24] A. Ng *et al.*, "Deep Learning Tutorial," *Univ. Stanford*, 2015.
[25] S. Ruder, "An overview of gradient descent optimization algorithms," pp. 1–12, 2016.
[26] Q. V Le, "A Tutorial on Deep Learning Part 1: Nonlinear Classifiers and The Backpropagation Algorithm," *Tutorial*, pp. 1–18, 2014.
[27] K. Johnson, *Acoustic and auditory phonetics*, 2nd ed. Malden : Blackwell Pub, 2003.
[28] F. Rumsey, *Spatial audio*. Oxford [etc.] : Focal Press, 2001.
[29] A. Pérez-López, "Pérez-López, A. (2014). Real-Time 3D Audio Spatialization Tools for Interactive Performance. Retrieved from http://www.andresperezlopez.com/portfolio/Andres_Perez_Master_Thesis.pdfReal-Time 3D Audio Spatialization Tools for Interactive Performance," 2014.
[30] V. Pulkki, a E. S. Member, M. Karjalainen, and a E. S. Fellow, "Localization of Amplitude-Panned Virtual Sources I : Stereophonic Panning *," *J. Audio Eng. Soc.*, pp. 739–752, 2000.
[31] V. Pulkki, "Virtual Sound Source Positioning Using Vector Base Amplitude Panning," vol. 144, no. 5, pp. 357–360, 1997.
[32] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Deep learning for monaural speech separation," *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. Proc.*, 2014.
[33] Y. Wang, A. Narayanan, and D. L. Wang, "On training targets for supervised speech separation," *IEEE/ACM Trans. Speech Lang. Process.*, vol. 22, no. 12, pp. 1849–1858, 2014.
[34] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," 2012.
[35] J. Salamon and J. P. Bello, "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification," pp. 1–5, 2016.
[36] M. Bloice, "Data Augmentation," no. March. 2014.
[37] S. Uhlich *et al.*, "Improving music source separation based on deep neural networks through data augmentation and network blending," *Icassp*, pp. 261–265, 2017.
[38] M. Miron, J. Janer, and G. Emilia, "Generating data to train convolutional neural networks for classical music source separation," *Univ. Pompeu Fabra*, 2017.
[39] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching Networks for One Shot Learning," 2016.