Musical source separation

Master Thesis

August 2020

Logitech

Student:
Alexandru Mocanu
IC - School of Computer Science and Communication Systems

Supervisors:
Dr. Milos Cernak (Logitech)
Prof. Pierre Vandergheynst (EPFL, Signal Processing Lab 2)
Dr. Benjamin Ricaud (EPFL, Signal Processing Lab 2)

Abstract

Musical source separation is a complex topic that has been extensively explored in the signal processing community and has benefited greatly from recent machine learning research. Many deep learning models with impressive source separation quality have been released in the last couple of years, all of them dealing with studio recorded music split into four instrument categories, *vocals*, *drums*, *bass* and *other*. We study how we can extend the number of instrument categories and conclude that *electric guitar* is also feasible to separate. We then turn our attention towards learning relevant signal encodings using parameterized filterbanks and we observe that filterbanks can not improve over simple convolutions on their own, but can help if the encoder is composed of both convolutions and filterbanks. Finally, we try to adapt models trained on studio music to live music separation and conclude that models trained on clean data also provide the best performance on live music as well.

Contents

Chapter 1

Introduction

1.1 Motivation and objectives

The topic of musical source separation has been extensively researched in the audio signal processing community, many advances being made especially in the last decade, after deep learning techniques have started being used. Musical source separation is a particular case of blind source separation, in which given a mixture signal of several sources, we wish to recover the individual source signals.

The applications of musical source separation are numerous: surround sound generation, music transcription, karaoke track generation, individual instruments manipulation etc. Imagine for example that you are a DJ and you want to mute the vocals and make the bass sound louder. In order to make this kind of manipulations you need access to the individual instrument tracks. Here is where the musical source separation models come into play.

The three objectives of the project are the following:

- analyze musical source separation with multiple instruments
- explore ways of learning useful encodings of the music signals that would facilitate the separation
- adapt the source separation models to perform good separation on live music

Most of the literature concerning musical source separation in the last years makes use of the MUSDB18 dataset [1]. This dataset consists of music tracks separated into four source classes: *bass*, *drums*, *vocals* and *other*. Therefore, all of the models developed using this dataset were used for four-class separation only. We aim to perform separation in more classes using our own proprietary dataset.

The first part of performing source separation consists of encoding the mixture signal in a relevant way. This is either done through some handcrafted transformation, such as STFT, or through some convolutional layers whose parameters are learned. While handcrafted transformations offer the advantage of interpretability and use of domain knowledge, they can not adapt to different data and tasks. The convolutional layers on the other hand are flexible, but less interpretable and do not allow the use of prior knowledge. Using parameterized filterbanks, we hope to find a balance between the two encoding methods, as we build convolutional filters that respect some functional form that uses fewer learnable parameters than the convolutional filter size.

Both the MUSDB18 dataset and our proprietary dataset used for multi-instrument separation consist of music tracks that were recorded in a studio setting. The source signals therefore contain no noise and no interference with other instruments. In live concerts however, we can not perfectly isolate an instrument and recording it leads to also leaking crowd noise and other instruments. We explore how we can take advantage of the existing musical source separation models by using a dataset comprised of music from the Montreux Jazz 2018 festival.

1.2 Report structure

The thesis is organized as follows:

- **Audio source separation problem description** - We describe the problem of source separation and the evaluation metrics generally used.
- **Literature review** - We summarize the research on musical source separation, with a focus on the methods that are of interest for the current project.
- **Multi-instrument source separation** - We discuss source separation of songs in more than four instrument categories.
- **Learnable signal encodings** - We explore the use of parameterized filterbanks to encode signals.
- **Live music source separation** - We present experiments with musical source separation models on live music.
- **Conclusion** - We summarize our study, draw the main conclusions and indicate possible future work.

Chapter 2

Audio source separation problem description

2.1 Problem statement

Let $x \in \mathbb{R}^{2\times T}$, with $T \in \mathbb{N}^*$ the number of samples, be a stereo signal composed of the source signals $s_i \in \mathbb{R}^{2\times T}$, $i \in \{1, 2, ..., K\}$, with $K \in \mathbb{N}^*$ the number of sources:

$$x = \sum_{i=1}^{K} s_i \tag{2.1}$$

The goal of audio source separation is to determine the source signals s_i given only x.

2.2 Evaluation metrics

Evaluation itself is an open-question in the audio source separation community. Many times, when evaluating an audio source separation model both objective and subjective measures are used.
In our work, we use the objective evaluation metrics described in [2]. According to this evaluation procedure, a source signal estimation $\hat{s}$ can be decomposed as follows:

$$\hat{s} = s + e_{spat} + e_{interf} + e_{artif} \tag{2.2}$$

where s is the clean source signal, e_{spat} is the error due to spatial distortions, e_{interf} is the error due to interference with other sources and e_{artif} is the error due to artifacts.
The errors e_{spat}, e_{interf} and e_{artif} are computed based on the following projection matrices

$$P_{s_j} = \Pi\{s_j\} \tag{2.3}$$

$$P_{\mathbf{s}} = \Pi\{(s_{j'})_{1\leq j'\leq n}\} \tag{2.4}$$

as follows:

$$e_{spat} = P_{s_j}\hat{s}_j - s_j \tag{2.5}$$

$$e_{interf} = P_{\mathbf{s}}\hat{s}_j - s_j - e_{spat} \tag{2.6}$$

$$e_{artif} = \hat{s}_j - s_j - e_{spat} - e_{interf} \tag{2.7}$$

From this decomposition we get the following metrics:

- Source to Distortion Ratio

$$\text{SDR} = 10\log_{10}\frac{\|s\|^2}{\|e_{spat} + e_{interf} + e_{artif}\|^2} \tag{2.8}$$

- Image to Spatial Distortion Ratio

$$\text{ISR} = 10\log_{10}\frac{\|s\|^2}{\|e_{spat}\|^2} \tag{2.9}$$

- Source to Image Ratio

$$\text{SIR} = 10\log_{10}\frac{\|s + e_{spat}\|^2}{\|e_{interf}\|^2} \tag{2.10}$$

- Source to Artifacts Ratio

$$\text{SAR} = 10\log_{10}\frac{\|s + e_{spat} + e_{interf}\|^2}{\|e_{artif}\|^2} \tag{2.11}$$

The usual way of evaluating these metrics on a dataset, that we also use, is by computing the median of these metrics over all the songs in a dataset. For one song, we split it in windows and take the median of these metrics over the windows.

We also use the Scale-Invariant SDR [3] defined as:

$$\text{SI-SDR}(s, \hat{s}) = 10\log_{10}\frac{||\alpha s||^2}{||\alpha s - \hat{s}||^2}, \alpha = \frac{\hat{s}^T s}{||s||^2} \tag{2.12}$$

The Scale-Invariant SDR, as opposed to SDR, can not be tweaked artificially by just changing the scale of the estimation. For SI-SDR, we compute the median over songs, and for one song, we compute the SI-SDR over the entire signal, no splitting in windows. The SI-SDR is also used in the loss functions of our models.

Chapter 3

Literature review

Over the past two decades, there has been a lot of research efforts in the topic of blind source separation. The early models made use of signal processing techniques which leveraged prior knowledge, such as the shapes of the spectrograms for specific musical instruments. However, in the last decade, as machine learning became a viable option due to the increase of computing power and adaptation of neural networks for training on GPUs, most of the researchers started concentrating their efforts onto this path. Therefore, the direction changed from prior knowledge-based source separation to data-driven source separation, where the models usually perform better the more data they have for training.

Nowadays, all the best performing audio source separation models make use of deep learning techniques, as it can be seen in the last Signal Separation Evaluation Campaign (SiSEC) [4]. We take a look at classic signal processing techniques and then put the emphasis on the relevant deep learning models that we used in our project.

3.1 Signal processing models

All the initial research in audio source separation and even past decade research has focused on developing solutions based on prior knowledge about the music signals. [5] presents an overview of the most prominent such methods. They can mainly be split in two broad categories: parametric and non-parametric audio source separation algorithms.

A first popular choice for parametric source separation consists of a range of non-negative matrix factorization (NMF) algorithms, which take advantage of several properties of the music structure to perform the factorization. Generically, given a matrix of non-negative values $X \in \mathbb{R}_{+}^{N \times M}$, a NMF algorithm aims to find non-negative matrices $B \in \mathbb{R}_{+}^{N \times K}$, called the dictionary, and $H \in \mathbb{R}_{+}^{K \times M}$, called the activation matrix, such that $X = BH$ and $K \ll N, M$. The rows of B can be interpreted as embeddings for each of the N entries of X, while the columns of H contain the scaling factors of the elements of these embeddings. In our case, X would be the magnitude of the spectrogram. [6] were the first ones to use NMF for music in the context of polyphonic music transcription. NMF models were used to either model the parameters of the musical instruments or rely on

musical signal properties.
In the case of the instrument properties modeling, [7] exploits harmonicity and inharmonicity in the form of constraints on the partial frequencies in order to transcribe piano music. [8] make use of the time evolution by applying an Auto-Regressive Moving Average (ARMA) model for constraining the activation matrix. Yet another choice is to represent an instrument as a source-filter model (such as vocal cords as a source and vocal tract as a filter), an approach found in [9]. There, a distinction is made between vocals and background by writing the magnitude spectrogram as $|X|^2 = \underbrace{W^F}_{\text{filter}} \underbrace{W^{F_0}}_{\text{source}} + \underbrace{B^M H^M}_{\text{background}}$. The filter matrix is then decomposed into $W^F = B^F H^F H^\Phi$, with dictionary B^F providing the atomic filter elements, H^F generating the filter shapes and H^Φ weighting these shapes. The source matrix is also factorized as $W^{F_0} = B^{F_0} H^{F_0}$, with B^{F_0} gathering the source spectra for a predefined range of frequency patterns and H^{F_0} weighting these spectra.
For the NMF models handling sound properties, we have two types of models. First, there are models which exploit the distinction between harmonic and percussive sound structure, such as in [10] where the magnitude spectrogram is decomposed into a harmonic component and a percussive one by structured projective NMF: $|X|^2 = \underbrace{B^h H^h}_{\text{harmonic}} + \underbrace{B^p H^p}_{\text{percussive}}$, with the harmonic component being obtained by projective NMF, while the percussive one is obtained by regular NMF. Second, other models apply musical constraints, such as [11] where they rely on the beat structure.

Other parametric approaches concentrate only on harmonic instruments, taking advantage of the structure given by the pitch and overtones. A first set of such methods focuses on pitch estimation, which mainly reduces to estimating the fundamental frequency F_0 and applying a mask on the spectrogram according to the harmonic structure. There are such methods that work in the time domain [12], frequency domain [13], or even make use of information from both time and frequency domains [14]. The multi-pitch estimation allows us to perform the separation according to pitches, but it does not offer instrument identification as well. For this, a second set of methods deals with timbre modeling. To perform instrument recognition in polyphonic music mixtures, [15] use the uniform discrete cepstrum (UDC), a timbre representation that can be computed from isolated spectral points, as opposed to the MFCC that needs the full spectrum, which makes it infeasible for representing the spectral envelope of a single source that has not been separated. Other techniques perform clustering on different representations of harmonic features, such as in [16] where they use Gammatone Frequency Cepstral Coefficients (GFCC) to cluster TF units for speech separation. Last but not the least, when we have access to the score, this can especially aid the separation, as seen in [17]. Of particular interest is using score information to perform the separation in real-time, like in [18].

Instead of modeling the signals through a parametric model, we can also apply non-parametric methods to obtain usually more efficient separation algorithms that do not impose some functional form to the signals. Harmonic-Percussive Source Separation

(HPSS) [19] applies median filters on the spectrogram, one along the time axis to get the harmonic contribution and another one along the frequency axis to obtain the percussive component. REPET [20] exploits the repetitive nature of accompaniment in order to separate the singing voice, while REPET-sim [21] does not assume a perfect periodicity, but rather only leverages the similarity between time frames. Finally, Kernel Additive Modeling (KAM) [22] provides a bridge between HPSS and REPET. Figure 3.1 shows different kernels used to detect percussive, harmonic, repetitive and smooth sound patterns.

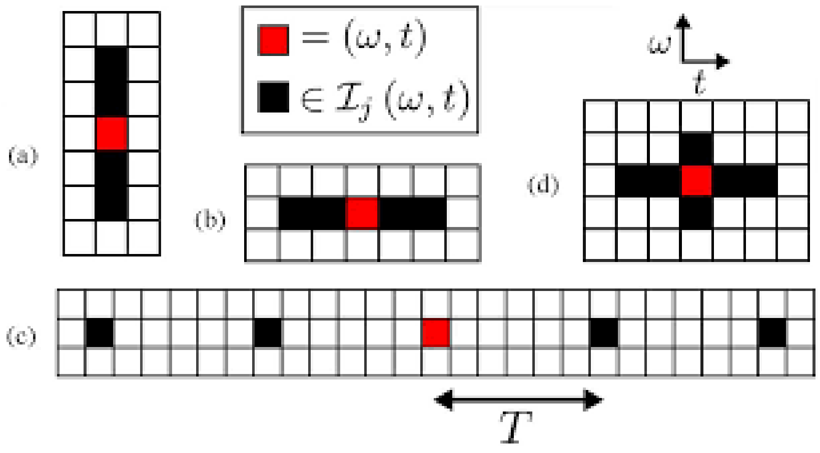


Figure 3.1: Kernels used to model (a) percussive, (b) harmonic, (c) repetitive and (d) smooth sounds, as presented in [22]

All the methods described perform separation without the knowledge of what the resulting instruments are. A more exotic approach that also lets the user separate one specific instrument is provided in [23]. There, the user provides a sound that mimics the instrument they want to separate (by humming, for example) and this information is used to point the separation algorithm towards the desired instrument's signal.

3.2 Deep learning models

Audio source separation models based on signal processing techniques have the advantage that they are interpretable. However, most of them do not provide a clear instrument identification, as we saw in the previous section, but rather perform an unsupervised separation based on features of the signal. Also, they mostly rely on our prior knowledge about music data, which may not be optimal. As an alternative, deep learning techniques have started to be used, which proved to bring a clear improvement.

The usual deep learning model consists of three parts:

- an encoder, which takes the mixture signal and generates an appropriate representation of it

- a separation module, also called masking module, which computes a mask to be applied to the mixture's encoding in order to obtain the encoding corresponding to the source we want to separate
- a decoder, which reconstructs the source signal from its encoding

Based on how the encoding and decoding are performed, we distinguish two types of models:

- spectrogram-based models, which use a Short-Time Fourier Transform (STFT), a mel-transform or a similar fixed time-frequency transformation for encoding
- waveform-based models, which learn the encoding and decoding modules jointly with the separation module

In what follows, we present a few spectrogram-based and waveform-based models which we used in our work.

3.2.1 Spectrogram-based models

Until recently, the most successful deep learning models to hold state of the art performance in audio source separation have been spectrogram-based. The best performing spectrogram-based model is MMDenseLSTM [24] where they combine DenseNet [25] and LSTM, and which outperforms the Ideal Binary Mask oracle [4]. However, they do not provide an open-source implementation. The best model which provides code is therefore Open-Unmix [26], which is intended to serve as an easily extensible baseline for the source separation community. Figure 3.2 presents the architecture of Open-Unmix.

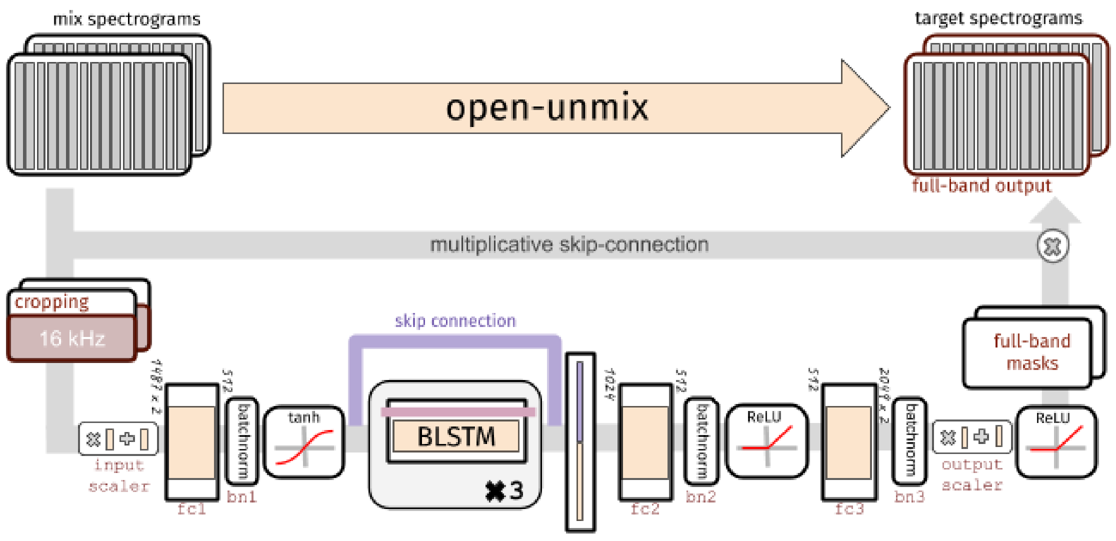


Figure 3.2: The architecture of the Open-Unmix model [26]

The input to Open-Unmix separation module is a magnitude spectrogram, so phase information is not used and the phase of the mixture is applied at the end when reconstructing the signal. The model starts with a fully-connected layer, which applies a linear transformation to the normalized input magnitude spectrogram. The output is passed through a batch-normalization and a tanh activation before being fed to a three-layer bidirectional LSTM, which learns from the sequence data. The output and input of the bidirectional LSTM are concatenated and passed through a sequence of two fully-connected layers with batch-normalization and ReLU activations, which yields the mask that is applied to the input spectrogram to obtain the spectrogram of the source that we want to separate. The sources are ultimately separated using a Multi-channel Wiener Filter (MWF) initialized with the estimated sources. The optimization criterion used is the L2 loss between the estimated and the actual magnitude spectrograms of the individual instrument signals.
Note that the network only separates one source, so we need one network for each of the sources that we want to separate. Note also that due to the bidirectional LSTM, the model is non-causal. Replacing the bidirectional LSTM with a simple LSTM leads to a decrease in performance. We use Open-Unmix in the first and third parts of our project.

3.2.2 Waveform-based models

The usual source separation pipeline would normally include first computing a spectrogram of the signal and taking the magnitude for further processing, thus losing phase information. Due to this loss of information, researchers have tried to build end-to-end systems, which would hopefully also exploit the phase. The first attempts to circumvent the use of time-frequency representations for signal encoding and to train an end-to-end system did not lead to state of the art performance. Among these, we remind [27] who reuse the Wavenet architecture [28], used initially for audio generation, and [29] who adapt the U-Net architecture for one-dimensional audio data.

Nevertheless, recent efforts have proven that end-to-end systems which operate directly in the waveform domain can successfully outperform their spectrogram-based counterparts. These models take inspiration from Conv-TasNet [30], a model that successfully surpassed the performance of the Ideal Ratio Mask oracle [4] in the speech separation task. This model came as an improvement over the TasNet model [31], which they argued was inflexible due to the LSTM module that it incorporated. Figure 3.3 depicts the architecture of Conv-TasNet.
The encoder consists of just one convolution and the decoder, symmetrically, of a transposed convolution. The separation module is a temporal convolutional network, composed of stacks of convolutional blocks with exponentially increasing dilation factors, which have both skip connections and residual connections, and ending with a 1x1 convolution followed by a sigmoid that produces the masks for each of the sources. A convolutional block uses 1x1 convolutions and a depthwise convolution, to reduce the number of parameters, as well as group layer normalization and PReLU activations. The model is trained

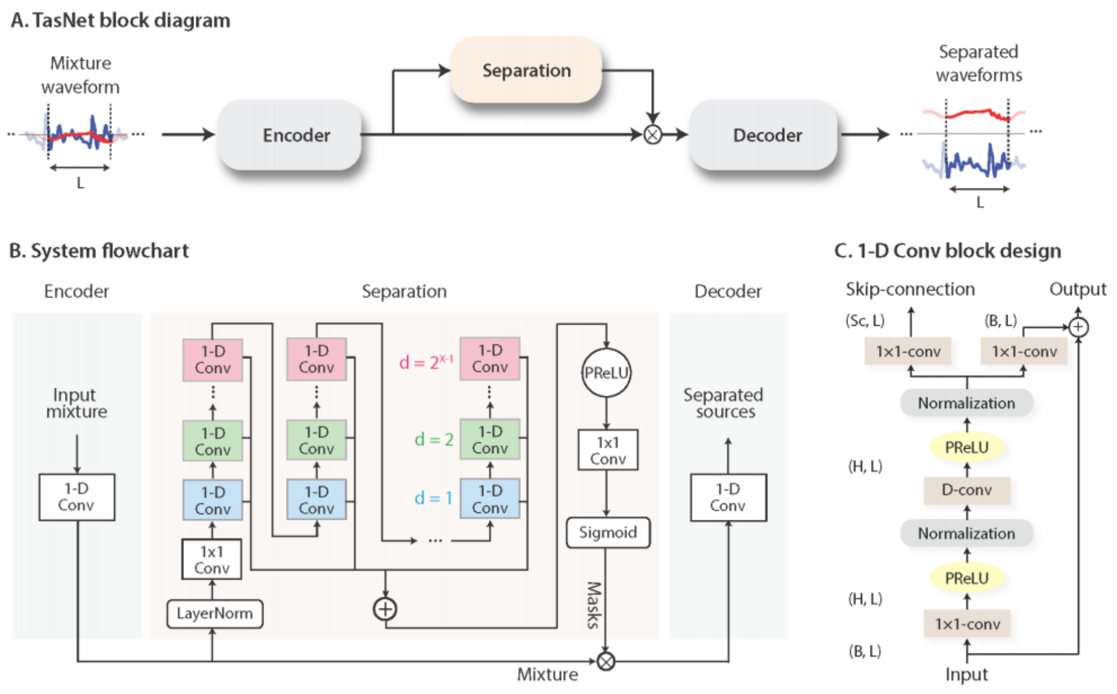
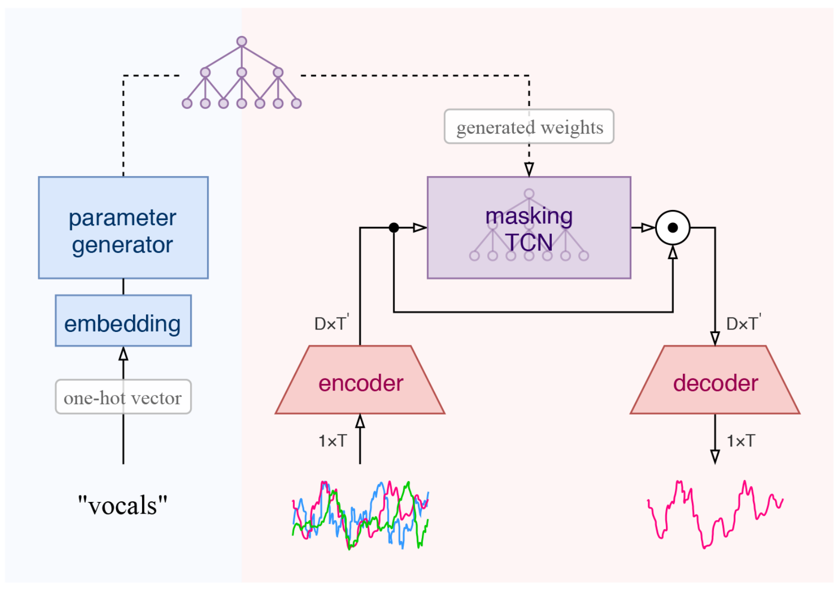


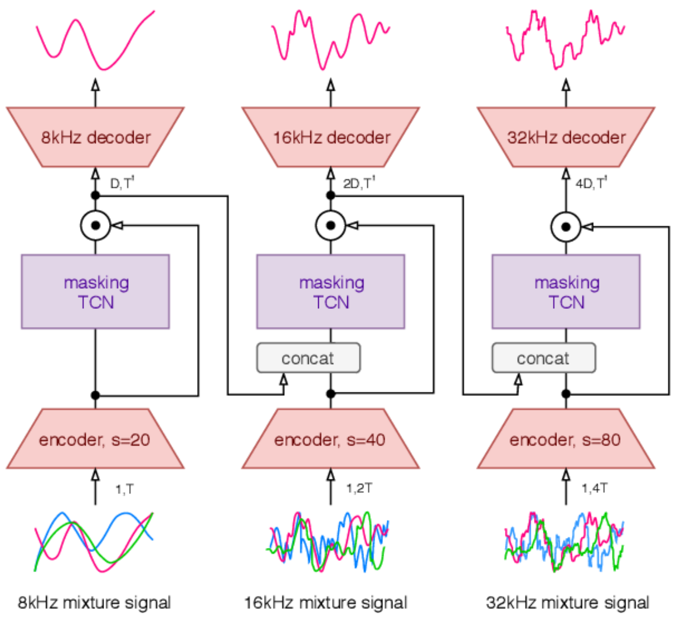
Figure 3.3: The architecture of the Conv-TasNet model [30]

using SI-SDR as the cost function.
In case the network needs to be used in a causal fashion, the group layer normalization can be replaced by a causal layer normalization, however this impacts the performance of the model. The use of many stacked convolutional blocks, along with increasing dilation factors, provides the receptive field size needed to have enough context when computing the separation masks. We insist on these details as we use this architecture extensively in the second part of the project. Note that unlike Open-Unmix, where we needed to use one network to separate each of the sources, Conv-TasNet produces all the source signals using one network only.

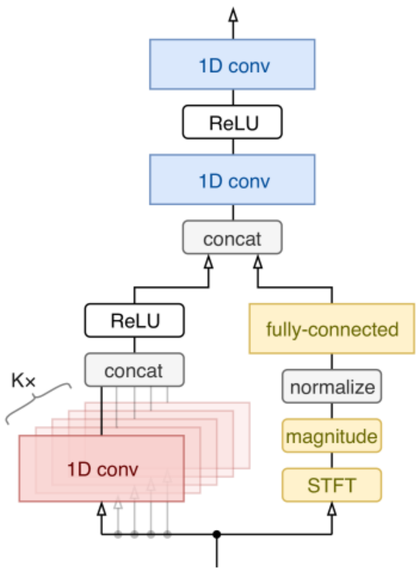
In [32] they adapt the Conv-TasNet architecture for musical source separation, by significantly increasing the model size, and they claim to achieve state of the art by surpassing Open-Unmix. They also propose their own model, called Demucs, which when trained on MUSDB18HQ only has a bit lower scores than the adapted Conv-TasNet model, but higher subjective evaluation scores.
Another model based on Conv-TasNet, that we use in the first and third parts and take inspiration from in the second part of the project, is Meta-TasNet [33], depicted in figure 3.4.
Here, only the encoder and decoder are learnt directly, while the separation module's weights are generated for each of the instruments. The parameter generating module in figure 3.4a produces the parameters for each of the layers in the separation module by using an embedding of the instrument we want to separate that is passed through two fully-connected layers. To be able to work with different signal resolutions, the mixture is



(a) One stage



(b) Resampled stages



(c) Encoder architecture

Figure 3.4: The architecture of the Meta-TasNet model [33]

downsampled from 44.1 kHz to 8, 16 and 32 kHz and the separation is done for each of these signals in stages, as shown in figure 3.4b. The encoder also has increased capacity compated to the one in Conv-TasNet, as it consists of both convolutions and an STFT transform, as shown in figure 3.4c.
When learning, apart from optimizing the SI-SDR, the loss also includes three additional terms: a *dissimilarity* loss that encourages the encodings of different sources to be different, a *similarity* loss that ensures that the encodings of the same source are similar, and a *reconstruction* loss that makes the decoding of an encoded source signal to be the same as the original source signal. This model surpasses Open-Unmix and Demucs and provides a new approach to source separation, as we can argue that using embeddings of the instruments to generate the network parameters makes the model have a better understanding of the various instruments' characteristics.

Chapter 4

Multi-instrument source separation

The goal of musical source separation is to separate different instrument categories from a song, usually with the aim of isolating the instrument categories as much as possible from one another and without adding any noise. Ideally, we would want to separate each instrument from a song, but this is difficult for both unsupervised algorithms, which may face difficulties when trying to differentiate instruments with similar frequency content, and supervised algorithms, which would need large amounts of data for each of the possible instruments that we would like to separate. Because of such limitations, most of the latest literature on the subject has concentrated on performing four-class separation, consisting namely of *vocals*, *drums*, *bass* and *other*, using primarily a standard dataset called MUSDB18 [1].
In this part of the project, we try to increase the number of instruments to separate by using a new dataset which includes the individual tracks for each of the instruments of a song. To our knowledge, there is no study in the literature regarding the separation of more than four classes in songs that include both vocals and different types of instruments.

4.1 Dataset

As the MUSDB18 dataset only provides the four classes mentioned, we needed to collect some other data in order to extend the separation. Therefore, we decided to collect data from a website called Karaoke Version [34]. They provide studio recorded tracks for each instrument of a multitude of songs from various genres. The tracks are in mp3 format and are sampled at 44.1 kHz with a bitrate of 16 bits.
The music tracks are not provided for free, so we first downloaded metadata for all the songs available on the website and analysed distributions of genres, release years, number of instruments per song etc. This way, we were able to decide which songs to buy in order to keep nearly uniform distributions of release years and main genres, so that we would have enough diversity in the data. The entire dataset that we downloaded consisted of 1500 songs, 10 times more than in the MUSDB18 dataset.
The next problem was to prepare the dataset for training our model and here we

encountered the questions of how to mix the instruments together into instrument categories and, more importantly, what instrument classes we should use.

4.1.1 Initial choice of instrument classes

In order to answer the first question, of how to mix instruments, we first needed an initial set of instrument classes. After downloading the songs and examining the different instruments, we came up with 11 instrument categories: *vocal*, *drum*, *bass*, *guitar*, *piano*, *synth*, *strings*, *brass*, *woodwind*, *percussion*, *other*.
The idea behind this categorization was to start from the initial classes, that we could already separate well, and to add other instrument classes whose instruments would provide perceptually similar sounds. We also took into account to have a reasonable number of songs in our dataset (above 200) for each of the instrument categories.
The effort of separating such a large number of classes would however prove too ambitious and possibly infeasible, as we have concluded after some objective analysis. Nevertheless, this initial categorization served our exploration for a suitable mixing procedure of the instrument signals.

4.1.2 Instrument tracks mixing

Having established an initial set of instruments, we then needed to decide how to mix instrument tracks. There are a few points to keep in mind before deciding how to mix the signals:

- The most naive way of mixing would be to just add the signals together. However, this procedure could result in integer overflow, which would impose clipping the mixture signal, an effect that we do not want to see.
- Songs may include an unbalanced number of instruments from different instrument categories. For example, many songs contain multiple guitars, but only one set of drums.
- Intuitively, the safest way of ensuring that a model would learn to separate every instrument category would be to make them equally loud.
- Naturally, the mixture signal should be the sum of the instrument category signals.

These observations and intuitions led to four different mixing procedures:

- **summing** - We simply add together instrument signals into instrument category signals. Then, we add the instrument category signals into the mixture signal.
- **averaging** - We take the average of the instrument signals to form instrument category signals. Then, we take the average of the instrument category signals to form the mixture signal.

- **scaling** - We add instrument signals into instrument categories and rescale the results to avoid overflowing. We repeat the same procedure when mixing instrument category signals into the mixture signal.
- **adaptive rescaling** - We add instrument signals into instrument categories and rescale the results to avoid overflowing. We then add instrument category signals together and compute the rescaling factor α needed to avoid overflowing. Lastly, we apply the rescaling factor α to the mixture signal and to the instrument category signals as well.

To evaluate how well we could separate each of the instrument categories, we use the Ideal Ratio Mask (IRM) oracle [4]. Given single channel spectrograms $y_j(f,t)$ of the sources $1 \leq j \leq J$, with J the number of sources, IRM computes the masks of each of the sources as:

$$M_j(f,t) = \frac{|y_j|^\alpha}{\sum_{j'} |y_{j'}|^\alpha} \tag{4.1}$$

In our case, we choose $\alpha = 2$ to obtain a Wiener filter. We chose this type of oracle as it is the best performing one used in the literature for comparison with source separation models.
These masks are applied to the mixture spectrogram to determine estimates of the spectrograms of the sources. These objective evaluation scores of the estimation act as an upper bound for magnitude spectrogram-based masking methods.
For each of the instrument classes, we extracted 50 songs that contained them and we computed both the SDR and the SI-SDR using IRM. The SDR for a song was computed as the median over 6 second chunks and, for a set of songs, we took the median of these values. On the other hand, we computed the SI-SDR on whole songs and took the median over all songs. This way, for SI-SDR, if an instrument was not very active in a song, it could still have a good score if IRM separated it well. Table 4.1 illustrates the scores that we obtained for the four mixing procedures.
We see that the scaling and adaptive rescaling procedures yield the same SI-SDR, as expected, as the only difference between the two is the scale of the signals. However, SDR is sensitive to scale, so the average and scaling procedures, for which the mixture is not the sum of the instrument categories, have low SDR scores.
Overall, as the adaptive rescaling procedure yields the best SI-SDR scores for six out of eleven classes and also computes the mixture as the sum of the instrument categories, it is the most suitable for mixing. Compared to summing or averaging, it also preserves the property of having the mixture signal equal to the sum of the instrument categories signals.

4.1.3 Refining the choice of instrument classes

After initial experiments with guitar separation (the toughest class to separate according to SI-SDR), we realised that it may be infeasible to try to separate that many classes. We therefore tried to group instruments in fewer categories, so that separation would become

	summing		averaging		scaling		adaptive rescaling	
	SI-SDR	SDR	SI-SDR	SDR	SI-SDR	SDR	SI-SDR	SDR
vocal	**10.961**	**9.930**	10.516	0.684	9.027	3.573	9.027	8.077
drum	6.864	**7.460**	**8.081**	0.691	5.659	3.115	5.659	6.486
bass	6.594	7.173	7.997	5.802	**8.855**	3.799	**8.855**	**9.276**
guitar	**4.272**	**5.122**	0.631	0.340	3.353	2.434	3.353	4.806
piano	2.724	3.576	**4.527**	0.491	3.706	2.059	3.706	**3.934**
synth	3.940	3.138	2.803	0.337	**5.650**	2.389	**5.650**	**4.358**
strings	2.574	2.624	3.948	0.397	**4.695**	1.781	**4.695**	**3.401**
brass	6.141	1.491	**7.613**	0.240	5.621	1.224	5.621	**2.041**
woodwind	5.010	0.250	6.518	0.053	**8.574**	0.333	**8.574**	**0.546**
percussion	4.206	3.754	4.880	0.386	**7.142**	2.497	**7.142**	**4.923**
other	2.127	2.814	3.480	0.338	**4.567**	1.762	**4.567**	**3.474**

Table 4.1: IRM performance for different mixing procedures on Karaoke Version dataset

	SI-SDR	SDR
vocals	10.328	9.274
drums	8.531	8.704
bass	6.338	6.924
other	7.582	7.844

Table 4.2: IRM performance on MUSDB18HQ dataset

more feasible. The scores obtained with IRM on the MUSDB18HQ dataset would be good indicators of the scores we should expect for the instrument categories we would want to separate. These results are presented in table 4.2.
We see that we should achieve SI-SDR scores of around 6 to expect a separation comparable to that obtained for the classes from the MUSDB18HQ dataset.
As guitar proved hard to separate and it was one of the classes that we wanted to separate the most, we also decided to split the guitar class into electric guitar and acoustic guitar. This proved that the acoustic guitar was the one that was dragging down the scores for the guitar class.
After grouping the initial instrument classes in several ways, we ended up with the configuration presented in table 4.3.
The *wind* class is composed of the *brass* and *woodwind* classes. The *other* class is composed of the remaining instrument classes, including *acoustic guitar*.

vocal	drum	bass	wind	electric guitar	other
9.438	5.597	10.997	7.362	5.427	5.548

Table 4.3: SI-SDR for IRM on the final list of instrument categories

4.2 Experiments

4.2.1 Open-Unmix guitar separation

Before diving into separating all the instrument categories that we identified (and before definitively establishing which instrument categories to separate), we tried training the Open-Unmix model to separate guitar. We used the same architecture configuration as the one that was previously used successfully for *vocals*, *drums* and *bass* and we expected to see comparable results for guitar. We initially trained with 100 samples and then with 200 samples, but each time the SDR score for guitar was below 0 dB when testing on a test set of 50 samples.
This was an indicator that we should look into the problem of how well we could expect to separate each instrument. Also, note that IRM, the oracle model used to indicate the classes to separate, provided an upper bound of the performance we could expect for magnitude spectrogram-based models. Therefore, we also redirected our attention to using a model operating directly in the waveform domain, that may have a chance of surpassing this upper-bound. This is how we shifted towards Meta-TasNet [33], a model that learns the encoding to be applied to the signal and also learns how to generate parameters for the separation module based on instrument embeddings.

4.2.2 Meta-TasNet multi-class separation

Opposed to Open-Unmix, Meta-TasNet separates all the instrument categories simultaneously. As we would expect, it needs more parameters than each Open-Unmix module dedicated to some instrument. Therefore, given the limited computational resources, we reduced our experiments to only using 8 kHz signals as input. Originally, Meta-TasNet was using three stages of separation of signals sampled at 8, 16 and 32 kHz. We trained using only the first stage, at 8 kHz, and expected that reasonable performance at this sampling rate would be an indicator that we could perform good separation when scaling up.
We initially built a training set of 900 samples and a validation set of 100 samples by randomly picking songs from the entire 1500 dataset. We then reduced the size of the one-stage model and trained with several such configurations. Every time, the original classes, *vocal*, *drum* and *bass*, were decreasing their validation losses, while for the new ones the validation losses increased. Figure 4.1 shows this behaviour for a full-sized one-stage Meta-TasNet model, model that we call **all_data**. As the losses were not evolving well, we stopped the training after 62 epochs.
Due to this behaviour we decided to filter from the training and validation datasets the songs that did not contain all the instrument classes. We therefore ended up with 287 samples for training and 38 samples for validation. Training again a full-sized one-stage model, this time for 250 epochs, we now obtained the validation loss curves in figure 4.2. We call this model **all_instruments**.

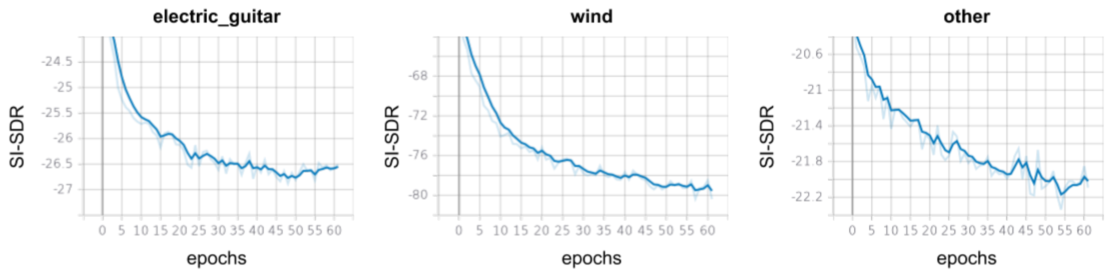


Figure 4.1: Validation loss for **all_data** model

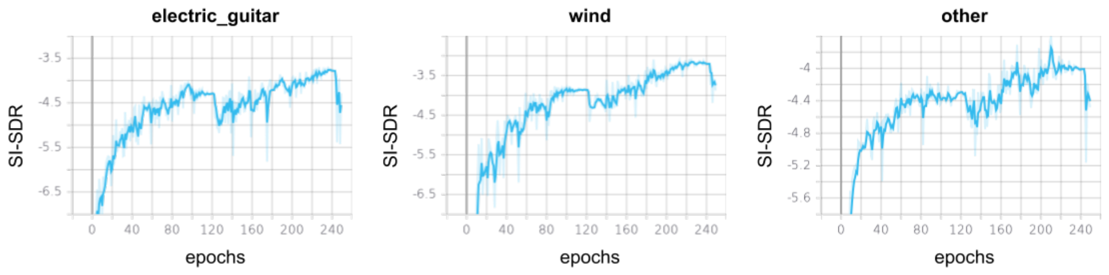


Figure 4.2: Validation loss for **all_instruments** model

We notice that the validation losses for the *electric guitar*, *wind* and *other* classes now evolve normally, even though their values are very low. We need to keep in mind that these classes should be much harder to separate than the ones from the MUSDB18HQ dataset and that we are using a model reduced in size and signal resolution.
For objective evaluation of the two models, we used a test set of 48 songs containing all the instrument classes. Table 4.4 provides the results.

	all_data				all_instruments			
	SDR	SIR	ISR	SAR	SDR	SIR	ISR	SAR
vocal	3.059	8.176	5.717	2.948	**3.507**	**9.757**	**5.870**	**3.271**
drum	8.908	14.568	6.553	3.945	**4.173**	**15.593**	**7.052**	**4.589**
bass	10.689	15.882	13.927	10.494	**11.206**	**16.957**	**14.134**	**10.796**
electric guitar	0.982	-0.767	3.247	**3.068**	**1.351**	**1.171**	**3.532**	2.284
wind	-6.660	-13.256	0.977	-0.231	**-5.326**	**-12.828**	**1.635**	**0.242**
other	1.509	-0.255	4.098	**3.627**	**1.877**	**0.915**	**4.511**	2.979

Table 4.4: Meta-TasNet scores for six class separation

To verify how these scores translate into perceptive impression, we also listened to some separation samples. The *wind* class, like the scores indicate, does indeed contain a lot of interference with other classes, especially vocals. The *electric guitar* and *other* classes yield scores that look promising for an eventual scaling up of the model and also sound clean enough when listened to.

4.3 Conclusion

The four instrument categories from the MUSDB18 dataset present quite distinct time-frequency features. Bass occupies the lower frequency spectrum, drums have specific impulses well-localized in time and spread in frequency, while vocals present distinctive, irregular harmonic patters, due to the source-filter structure (the vocal cords acting as the source and the vocal tract playing the role of a filter). This may be one reason why researchers try to improve deep learning models in this context.
Moving to the next step, that of separating other instrument classes, first requires finding a good dataset and taking care of how to prepare it. Then, deciding what instruments to separate is another challenge that is not well-defined. Last but not the least, tailoring a model for the new set of instruments could start from already existing architectures, but may need further improvements targeted at the new classes in order to achieve a performance comparable to the separation of the initial classes.
In our experiments, we managed to successfully separate electric guitar with a decent SDR score of 1.35, even with a model trained on 8 kHz data only. This is an indication that with enough computing resources, using a full-sized Meta-TasNet model should provide competitive separation on electric guitar. To our knowledge, our study is the first one to try extending the number of classes used for separation. Of course, this was just an initial exploration using one type of deep learning model that was originally developed for four class separation. Nevertheless, Meta-TasNet seems to be a good option for heading towards multi-instrument separation due to its procedure of representing instruments through embeddings and using those to dictate the separation. However, further adjustments may be brought to the system, and more musicology expertise may be beneficial for defining the changes required.

Chapter 5

Learnable signal encodings

When a signal is processed by a neural network to be separated in instrument signals, it is typically first encoded either using a fixed transformation, such as STFT, or by applying some transformation learned by some convolutional filters. While the STFT is easily interpretable, it is also rigid, while the convolutional filters are hardly interpretable, although they give a lot of flexibility. Our objective is to see if we can find a middle-ground, namely use some convolutional filter that respects a well-known functional form with a reduced number of parameters to learn. The idea of using filterbanks for learning encodings was inspired from [35].

5.1 Dataset

For this part of the project, we used the MUSDB18HQ dataset [1], which consists of 150 studio-recorded songs, 86 for training, 14 for validation and 50 for testing, with a total of 9.82 hours of music. Each song comes with five files corresponding to the mixture and the instruments: *mixture.wav*, *bass.wav*, *drums.wav* and *vocals.wav*. The tracks are stereo and sampled at 44.1 kHz with bitrate of 16 bits. We downsampled the songs at 8 kHz and used them to allow faster training and evaluation.

The *musdb* Python library is generally used to load the MUSDB dataset. However, we wrote our own dataset class to have more flexibility in terms of data handling and to allow loading other datasets as well. There are three types of augmentations that we apply to the data:

- randomly choose one of the two channels for an instrument signal
- randomly apply a scaling factor between [0.75, 1.25] to each instrument
- shuffle instrument channels in half of the batch

5.2 Models

For this task, we started from the architecture of Conv-TasNet [30], which performs the encoding and decoding using one convolution layer each, as described in subsection 3.2.2. As the encoding and decoding modules are well delineated, we decided to replace these with our own encoder and decoder setups and keep the masking module's structure.
Apart from Conv-TasNet, in order to faster evaluate the quality of an encoder, we also used a separation module based on the Ideal Ratio Mask [4]. We call this family of models FB-IRM. Figure 5.1 shows a high-level diagram of how this model operates.

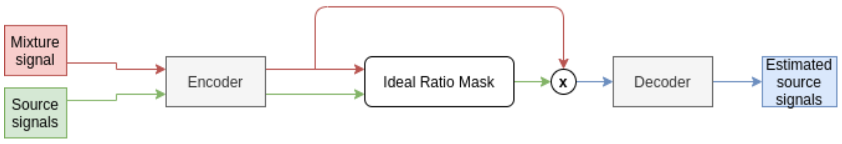


Figure 5.1: High-level diagram of a FB-IRM model

Given J encoded source signals $y_j(n,t)$, $1 \leq j \leq J$, with n the encoding dimension and t the time dimension, the Ideal Ratio Mask separator computes the mask to be applied to the mixture encoding as:

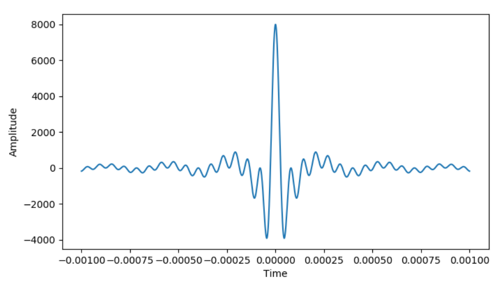
$$M_j(n,t) = \frac{|y_j(n,t)|}{\sum_{j'} |y_{j'}(n,t)|} \tag{5.1}$$

We decided to use two types of filterbanks:

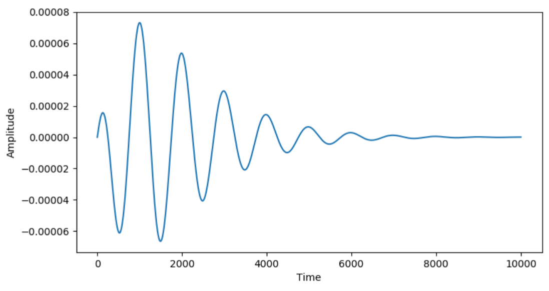
- the sinc filters from SincNet [36], which were successfully used for speaker recognition. Given two frequencies f_1 and f_2, the n^{th} value of the filter is $g[n, f_1, f_2] = 2f_2 sinc(2\pi f_2 n) - 2f_1 sinc(2\pi f_1 n)$. f_1 and f_2 will be the learnable parameters of the filterbank.

- a parameterized gammatone filter. Gammatone filters are used to model the auditory system. Given filter order k, center frequency f, phase ϕ and bandwidth b, the n^{th} value of the filter is $g[n, k, f, b, \phi] = n^{k-1} e^{-2\pi bn} cos(2\pi fn + \phi)$. We set $k = 2$ and use f, b and ϕ as learnable parameters of the filterbank.

The impulse responses of the two types of filters are displayed in figure 5.2.



(a) Impulse response of a SincNet filter



(b) Impulse response of a gammatone filter

Figure 5.2: Impulse responses of the filters used

The first encoder-decoder architecture that we tried consisted only of one filterbank in the encoder and another one in the decoder, as depicted in figure 5.3. We simply called this type of encoding *filterbank*.

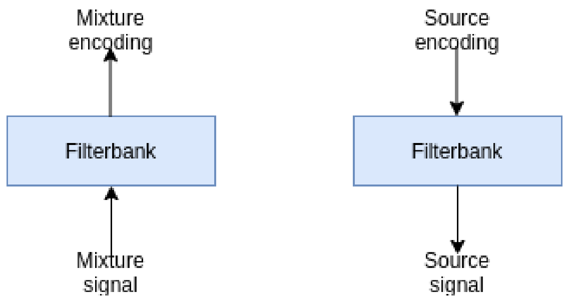


Figure 5.3: Encoder-decoder pair for the *filterbank* type of encoding

This type of encoding proved to be less successful than the simple convolution encoder used in the default implementation of Conv-TasNet. Therefore, we tried using filterbanks in new encoding setups. The first type of setup was inspired from the Meta-TasNet encoder and is called *hybrid*. It is represented in figure 5.4.

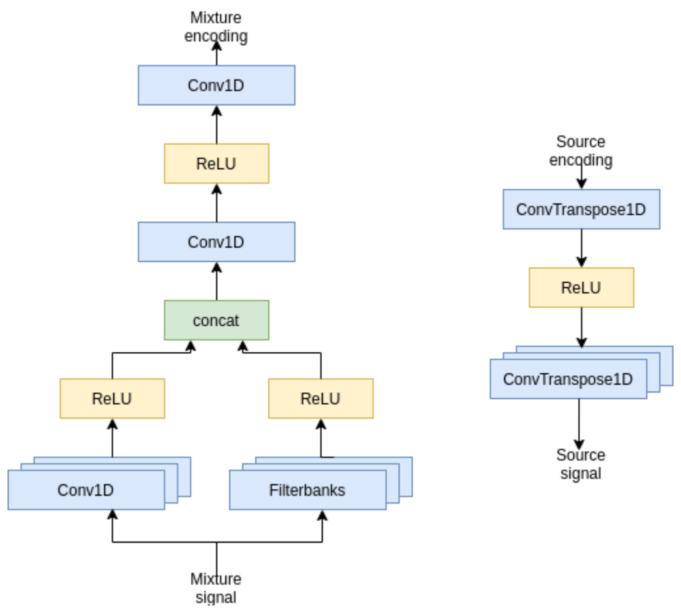


Figure 5.4: Encoder-decoder pair for the *hybrid* type of encoding

We took the encoder from Meta-TasNet and replaced the branch where STFT was applied with a set of filterbanks. The number of filter channels for the filterbanks is set by default to be equal to that of convolutions in the other encoding branch. The convolutions and the filterbanks may have multiple filters, each new filter having double the kernel size of the previous one, but also including padding to account for this increase in kernel size.

Last but not the least, we also tried stacking up filterbanks, to obtain the encoding type called *deep*. It is depicted in figure 5.5.

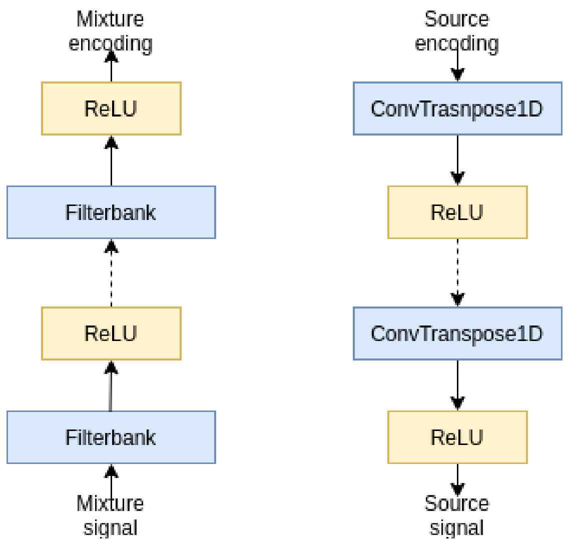


Figure 5.5: Encoder-decoder pair for the *deep* type of encoding

The kernel size of all the layers in the *deep* encoder are kept the same.

When referring to the original one convolution encoder used in Conv-TasNet, we will simply call it the *convolutional* encoder.

5.3 Results

To be able to perform a fast enough exploration of the several encoding configurations, we trained our models on 8 kHz data. We judged the relative performance of the different encoders by looking at the validation losses. After deciding on the most promising configuration that used filterbanks, we trained that one on 44.1 kHz data and with a large separation module. We present both the exploration and the results of the larger model. We also include a list of all the configurations tried and brief observations about them in the appendix.

5.3.1 Encoder architecture exploration

For the exploration phase, we tried the various types of encoders described in the previous section on 8 kHz signals with both the FB-IRM model and the Conv-TasNet model with a separation module consisting of 3 stacks of 6 conv blocks each. The FB-IRM was used to get an initial impression of what encoder would work best, while the Conv-TasNet model was used for confirmation for the best configurations obtained with FB-IRM. When not explicitly mentioning FB-IRM, the Conv-TasNet model was used.

Filterbank encoders

For the *filterbank* encoder, apart from the type of filterbank to use, we also had to choose whether we would compute the decoder as the pseudoinverse of the encoder or let the two adapt independently. Figure 5.6 shows the FB-IRM model for these two cases for the sinc filter, along with the *convolutional* encoder.

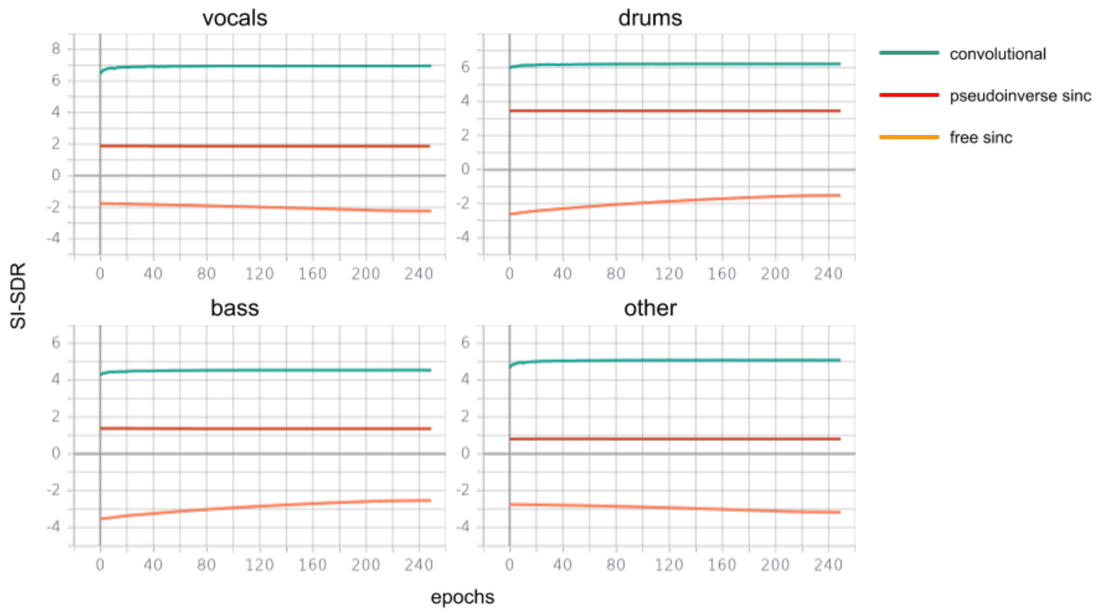


Figure 5.6: FB-IRM validation loss for *convolutional* encoder (in green), *filterbank* sinc encoding with decoder as the pseudoinverse of the encoder (in red) and *filterbank* sinc encoding with independent encoder and decoder (in orange)

We see that the setup with the pseudoinverse filter performs better in case of sinc, a fact which is confirmed for the gammatone filter as well. However, both of these filterbanks have performance clearly below that of the *convolutional* encoder, as it can be seen in figure 5.7, where both filterbanks are used in the pseudoinverse setup. Still, the gammatone filter performs better than the sinc one, obtaining around 2 dB more for every instrument class.
For completion, we also ran the *convolutional* encoder and the *filterbank* encoders, in the pseudoinverse setup, with the Conv-TasNet model and obtained the validation losses in figure 5.8. We notice that the gammatone filter performs better than the sinc filter in the real setting, just as previously predicted with FB-IRM. In addition to that, the *filterbank* encoder with gammatone filter provides similiar performance to the *convolutional* encoder, even surpassing it for *vocals*. Morevover, the *filterbank* encoder with sinc filter also obtains similar performance to the *convolutional* encoder on *vocals*, which is consistent with the successful application of the sinc filters for speech recognition in [20].

The conclusion of these experiments is that filterbanks alone would not be enough to surpass the *convolutional* encoder when scaling up the model size (at least not for all the instrument classes). Therefore, we should look for more complex encoder architectures to bring improvements over the original Conv-TasNet model. We also conclude that gammatone filterbanks are preferable to the sinc filterbanks, so they will become the main focus for the next explorations.

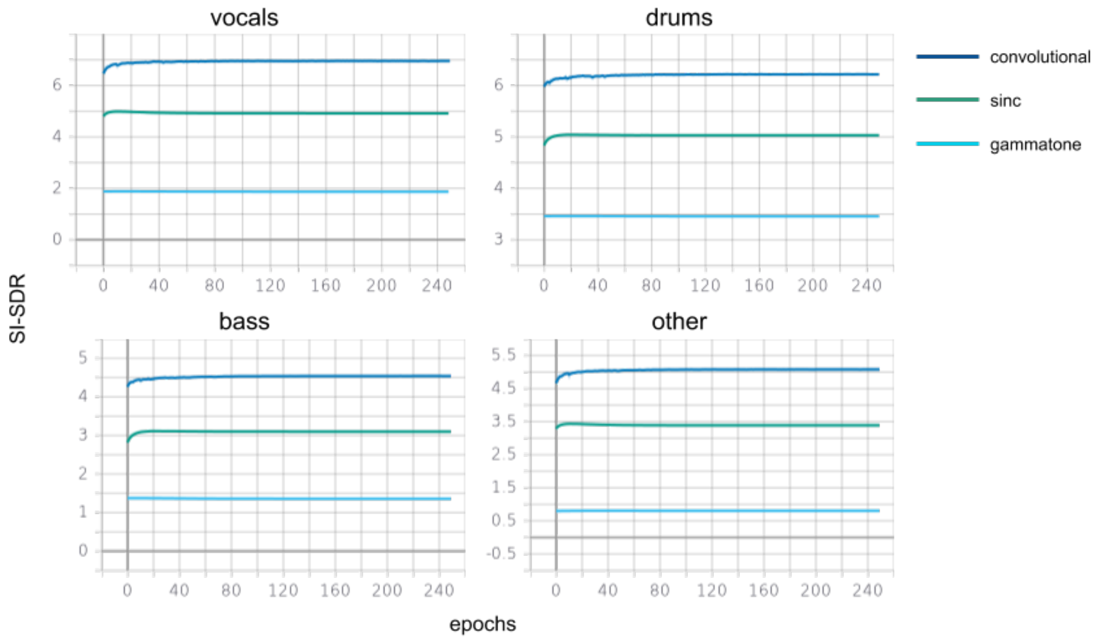


Figure 5.7: FB-IRM validation loss for *convolutional* encoder and *filterbank* encoders

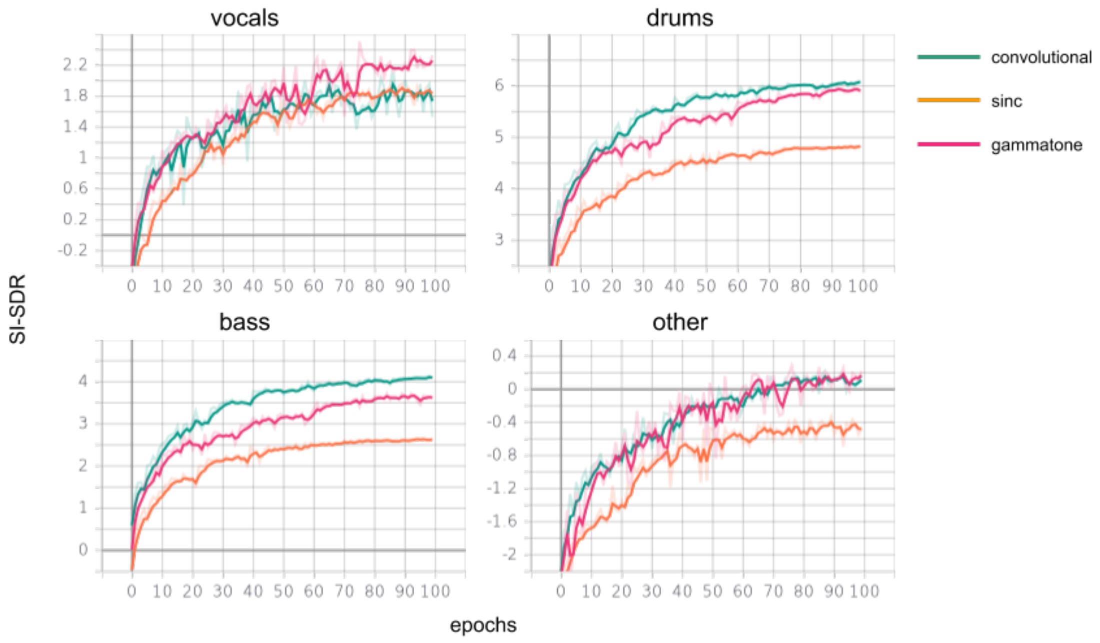


Figure 5.8: Conv-TasNet trained with *convolutional* encoder and *filterbank* encoders

Hybrid encoders

The first thing to check about *hybrid* encoders was to see if they could improve over the *convolutional* encoder. For this, we ran the FB-IRM model with both types of filterbanks for the *hybrid* encoder and we obtained the results in figure 5.9. In both sinc and gammatone configurations, we used one layer of convolutions and one layer of filterbanks for encoding.

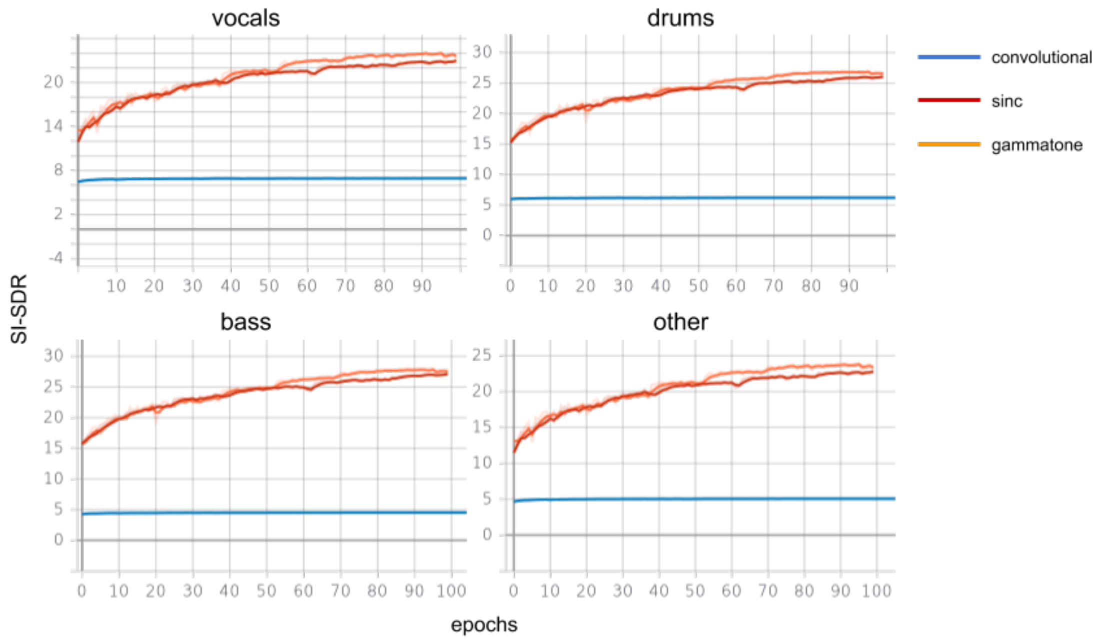


Figure 5.9: FB-IRM validation losses for *convolutional* encoder (in blue), *hybrid* encoder with sinc filterbank (in red) and *hybrid* encoder with gammatone filterbank (in orange)

We can see that the *hybrid* encoders clearly outperform the *convolutional* encoder due to their increased capacity. This benefit of increasing the capacity of the encoder was already previously tested in [33].
The next step was to see how different setups of *hybrid* encoders would perform when training a Conv-TasNet model. For this we prepared four different setups: an encoder where the filterbanks were actually set as simple convolutions (called *conv*), an encoder with sinc filters (called *sinc*), an encoder with gammatone filters (called *gammatone*) and an encoder like the one in Meta-TasNet, with an STFT instead of the filterbanks (called *stft*). The validation loss for these models can be seen in figure 5.10.

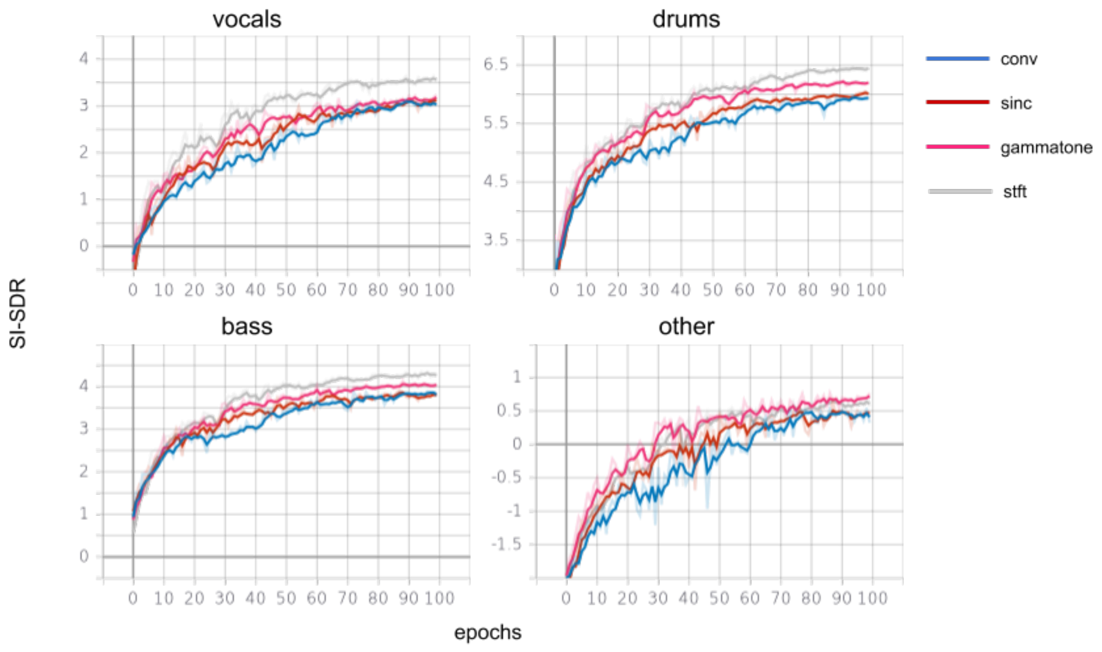


Figure 5.10: Validation losses for *hybrid* encoders: *conv* (in blue), *sinc* (in red), *gammatone* (in pink), *stft* (in grey)

We see that the best configuration is that of the Meta-TasNet encoder, which uses STFT. However, both filterbank-based encoders perform better than the one consisting only of convolutions and the gammatone-based encoder comes close enough to the STFT-based one. This means that there is benefit in mixing the flexible representations of the convolutions with the more rigid and robust ones of the filterbanks as opposed to using only convolutions. Regarding the better performance of the Meta-TasNet encoder, we argued that the STFT branch, which consisted not only of the STFT but also a convolution of kernel size 1 applied on top of the spectrogram, may have given the edge to this encoder. However, we also tweaked the *hybrid* gammatone encoder by adding one convolution of kernel size 1 on top of the filterbank layer, but still did not manage to outperform the STFT-based encoder.

Next, we tried to see if we could improve the gammatone-based encoder by using two layers instead of one or by increasing the kernel size from 20 to 100 (also increasing the stride from 20 to 100), which would give a better-defined shape of the gammatone filter. The results in figure 5.11 show that increasing the kernel size actually diminished the performance of the system, while increasing the number of layers did not bring any improvement either, as was the case when using STFT instead of gammatone filterbank.

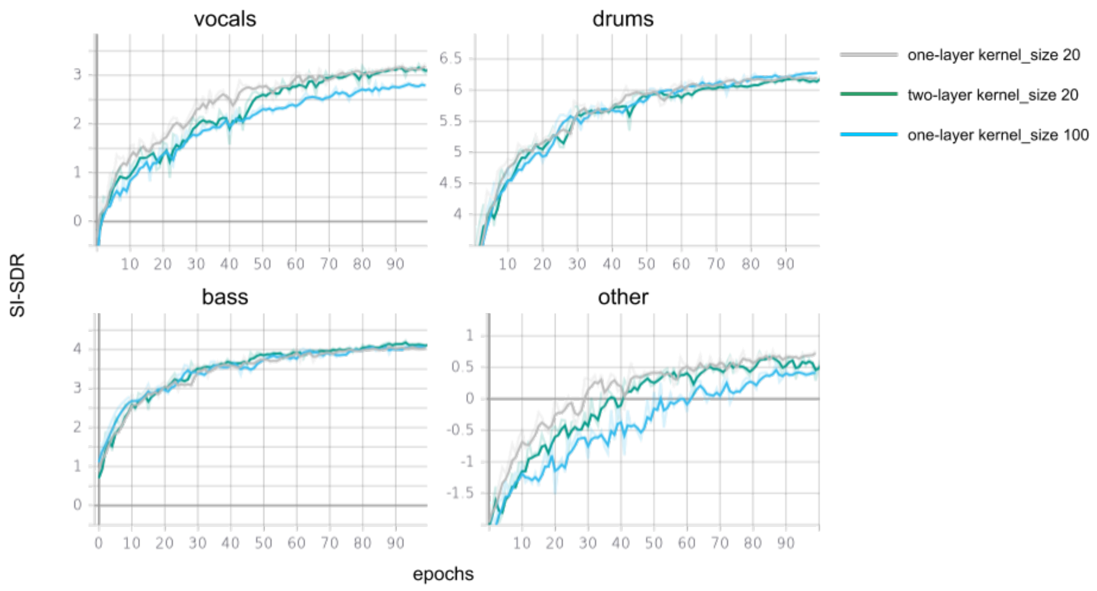


Figure 5.11: Validation losses of *hybrid* gammatone encoders: with default configuration (in grey), with two layers (in green) and with kernel size 100 (in light blue)

We next explored how the number of filter channels allocated to the filterbank branch of the encoder would influence the performance. By default, the number of filter channels of the convolution and filterbank branches are split equally. We chose configurations where the ratio of the filterbank filter channels in the encoder were 0 (only convolutions), 0.5, 0.75 and 1 (only filterbanks). We performed these tests with both gammatone (in figure 5.12) and sinc filters (figure 5.13).
Notice that changing the number of filterbank filter channels does not make too big of a difference in case of the average performance of the gammatone filterbank. However, for the sinc filter, we see that using convolutions clearly helps and using convolutions only is not the best option. If we want to understand why this happens, we need to take a look at the Conv-ReLU-Conv bottleneck applied at the end of the encoder. The outputs of the convolutional and the filterbank branches are concatenated and then the first Conv layer in the bottleneck is applied to that. In case of the gammatone-based encoder, the weights applied to the convolution branch output are one or two orders of magnitude higher than those applied to the filterbank branch output. This means that more emphasis is put on the convolutions, so the filterbank acts more as a regularizer in this case. In case of the sinc-based encoder, the weights applied to the convolution branch output and those applied to the filterbank branch output are of the same order of magnitude, so both encoder branches contribute equally.

Last but not the least, we pretrained the default *hybrid* gammatone encoder-decoder on FB-IRM and used it to train the separation module of Conv-TasNet. This decision was inspired by the fact that spectrograms, which are fixed transformations, were used successfully for source separation before. Therefore we thought that if we could use a

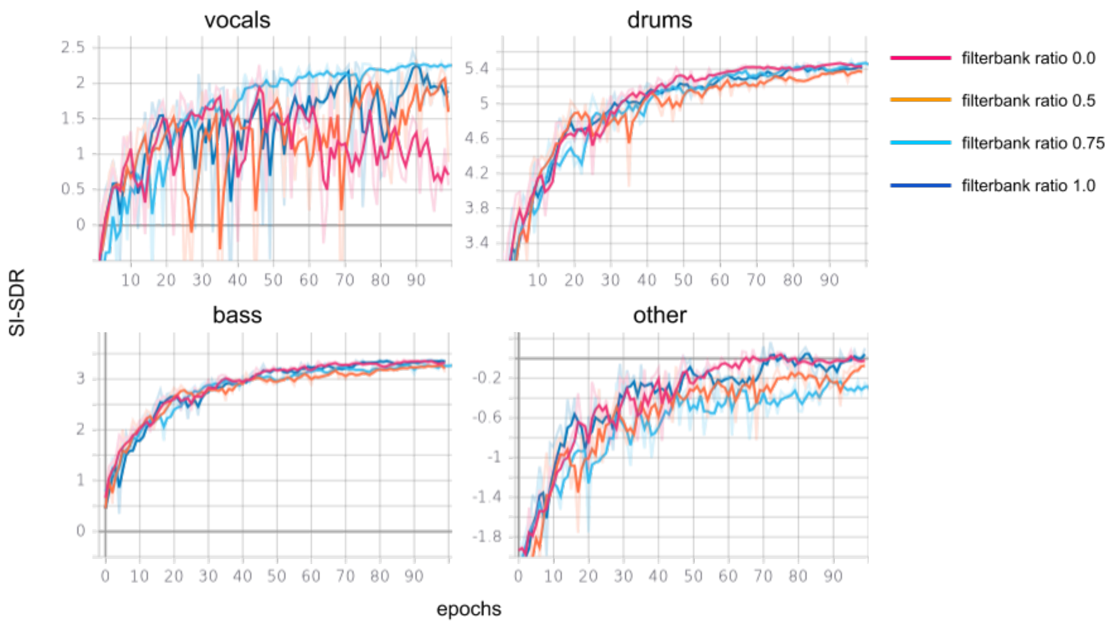


Figure 5.12: Validation loss for different **gammatone** filterbank filter channel ratios in the encoder: 0 (in pink), 0.5 (in orange), 0.75 (in light blue) and 1 (in dark blue)

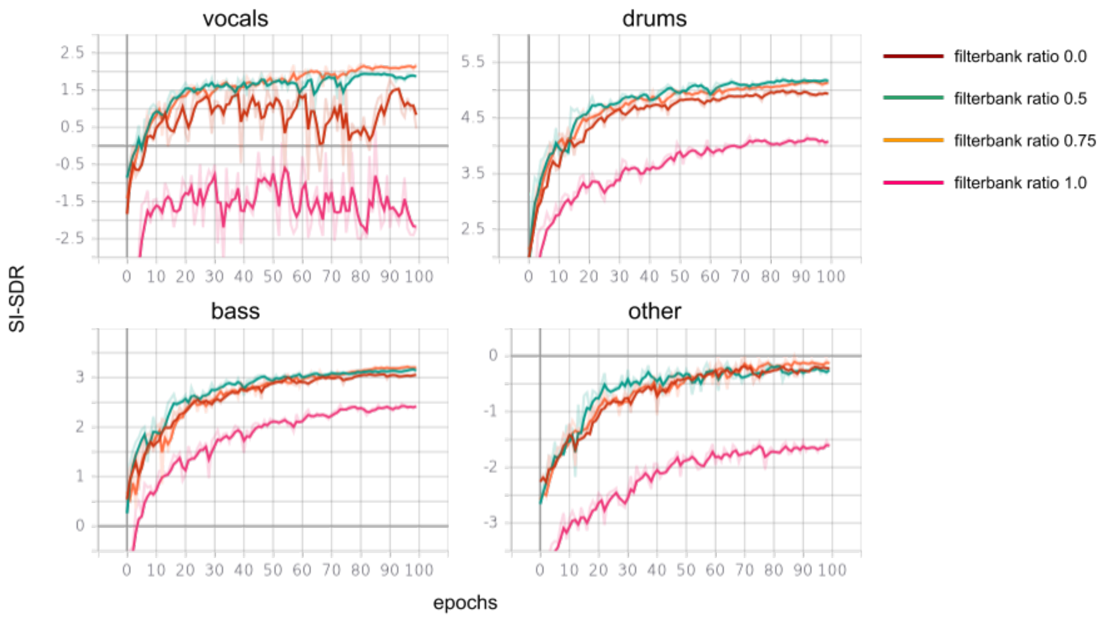


Figure 5.13: Validation loss for different **sinc** filterbank filter channel ratios in the encoder: 0 (in red), 0.5 (in green), 0.75 (in orange) and 1 (in pink)

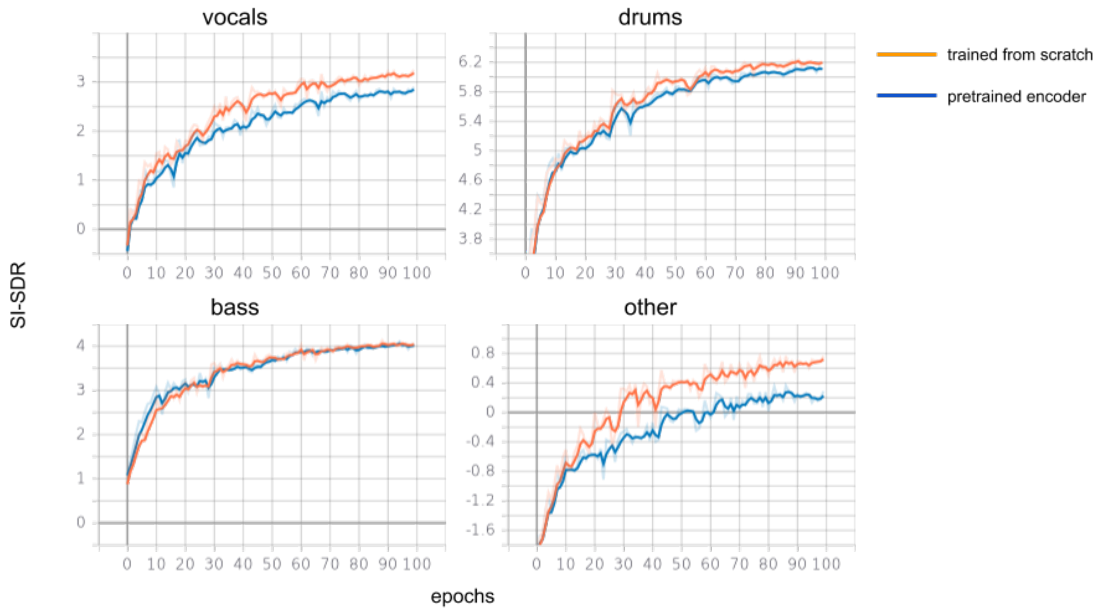


Figure 5.14: Validation losses for default *hybrid* gammatone encoder setup (in orange) and the setup with fixed pretrained encoder-decoder (in blue)

fixed encoding that was finetuned for musical source separation, it could provide more stable learning than jointly optimizing the encoder and decoder along with the separation module. The results in figure 5.14 however show that learning the encoding jointly with the separation module is still better than having a fixed encoder.
Overall, we could say that the best encoding architecture that we found is the one from Meta-TasNet, consisting of convolutional and STFT representations processed together. Nevertheless, the gammatone filters also come close to STFT and are beneficial when used alonsgide convolutions.

Deep encoders

We briefly inspected the *deep* type of encoder by stacking two layers of gammatone filterbanks together. We trained a Conv-TasNet model and we used kernel sizes 20, 16 and 10 for both layers in the encoder. The decoder consisted of two transposed convolutions with the same kernel sizes as in the encoder. Figure 5.15 presents the validation losses for the *deep* encoders as compared to a *filterbank* encoder using gammatone filters. The *filterbank* encoder had a kernel size of 20.

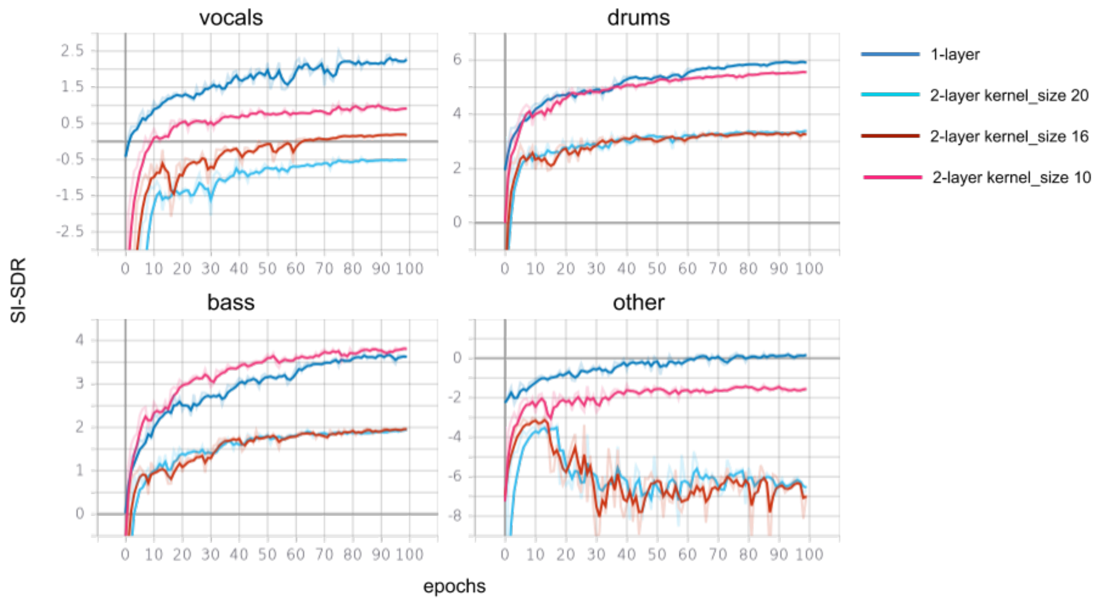


Figure 5.15: Validation losses for a gammatone-based encoders: *filterbank* (in dark blue), *deep* with 2 layers and kernel size 20 (in light blue), *deep* with 2 layers and kernel size 16 (in red) and *deep* with 2 layers and kernel size 10 (in pink)

We see that for most of the classes, the *deep* encoders actually perform worse than the *filterbank* encoder. A reduced kernel size of 10 for the *deep* encoder seems to bring a clear improvement over the larger kernel size. Decreasing the kernel size even further will however completely render the gammatone structure of the filter irrelevant, as the filter will consist of too few taps.
As the results of the *deep* encoders were not promising, we chose not to pursue them and decided that the *hybrid* encoders are the best option to use when scaling up the Conv-TasNet model size.

5.3.2 Large model results

We decided that the most promising type of encoding using filterbanks is the hybrid encoder with gammatone filterbanks, with an equal number of filter channels for both filterbanks and convolutions. Therefore, we trained a model with this type of encoding on 44.1 kHz data and with a separation module consisting of 4 stacks of 10 conv blocks each. The full configuration of the model is provided in the appendix. This separation module configuration was taken from [32], however with a smaller number of parameters for the separation module, as the full-size model could not fit our GPUs.
The SDR scores of this model, named Conv-TasNet-filterbanks, are provided in table 5.1. We also provide the results of Open-Unmix, Meta-TasNet and the Conv-TasNet model from [32] for comparison.
We see that the performance of our model is far from that of the state-of-the-art models, but that is expected given the smaller scale of our model. We would have expected to see

	vocals	drums	bass	other	avg
Open-Unmix [26]	6.32	5.73	5.23	4.02	5.36
Meta-TasNet [33]	6.40	5.91	5.58	4.19	5.52
Conv-TasNet [32]	6.81	6.08	5.66	4.37	5.73
Conv-TasNet-filterbanks	4.07	4.63	4.54	2.89	4.03

Table 5.1: SDR scores of Conv-TasNet with filterbanks compared to baselines

our model perform on par with the other ones if we could train it at the same size as the Conv-TasNet model from [32].

5.4 Conclusion

Signal encoding for musical source separation has been done primarily using either fixed transformations, that were known to extract relevant features, or flexible learnable transformations using convolutions. We explored the possibility of mixing the flexibility of a learnable encoding with the benefit of including prior knowledge by imposing some functional form on the convolutional filters used in the encoding, exposing only a few learnable parameters. Our experiments yielded the following main observations:

- Filterbanks used only by themselves to perform the encoding perform worse than simple convolutions.
- Combining the outputs of convolutions and filterbanks when performing the encoding brings an improvement over using convolutions only.
- Combining the outputs of convolutions and a spectrogram on which we apply a convolution with kernel size 1 achieves the best performance.

Trying our best encoding configuration that uses filterbanks on the MUSDB18 dataset, we were able to achieve reasonable separation (despite the 2.7 dB deficit below the best performing model) given that our model size could not be increased to a level at which we would have expected it to be competitive. We believe that with more computing resources, our model could be scaled up to compete with the current state-of-the-art models. At the same time, we think that in order to make better use of filterbanks, other encoding architectures and parameterized filterbank types should be explored.

Chapter 6

Live music source separation

As opposed to clean music, live music also contains noise from the crowd. When recording instruments on stage, we can not isolate them properly from each other and from the crowd, so the recording of each instrument will also contain crowd noise and interference from other instruments. Therefore, if training some model using live music data, we can not hope to cleanly separate the instruments, like we could aim for in case of studio recorded music. We can nevertheless try to emphasize each of the instruments, which would lead to a good enough separation in terms of subjective evaluation.
Using objective evaluation metrics, like the ones introduced in section 2.2, is not as useful as in the case of studio recorded music. That is because live music training data itself contains noise in the individual instrument tracks, noise that we can not aim to perfectly replicate. However, objective scores still provide a hint of how well the separation models perform, so we will use these metrics for live music as well.

6.1 Dataset

The dataset used in this part of the project is proprietary, belonging to the EPFL Cultural Heritage & Innovation Center. It consists of concerts from the 2018 Montreux Jazz Festival. The raw dataset contains recordings of the instruments in each concert and from different parts of the concert stages. As some of the instruments contained mono recordings while others contained 2 or 3 channels, we averaged the channel signals for each of the instruments, producing mono signals for each instrument. Our models operate on stereo recordings by default, so we replicated the channels of each of the instruments to obtain stereo instrument tracks.
To make the dataset usable for training musical source separation models, we grouped the instrument recordings into the four instrument categories from the MUSDB dataset: vocals, drums, bass and other. To obtain these categories, along with the mixture, for each of the concerts, we took the following steps:

- we manually distributed the instrument tracks into the four categories
- we added all the signals together into a signal s_{total}

- we computed the scaling factor α that applied to s_{total} gives mixture signal $s_{mix} = \alpha s_{total}$ which avoids overflow
- for each category, we added the instrument tracks belonging to it and scaled the result by α to obtain instrument category tracks s_{vocals}, s_{drums}, s_{bass}, s_{other}

As 4 out of 16 concerts did not contain tracks for all of the instrument categories, we decided to take these out of the dataset.
Songs in a concert are separated by breaks when only the crowd can be heard. We did not want to keep this information, so we manually selected the time intervals corresponding to the actual songs. The final version of the dataset, after all the processing, thus consists of 12 concerts with a total of 156 songs and 12.53 hours of music. The amount of data is thus comparable in size to that of the MUSDB dataset.
The last step was to do the split of the data into train, validation and test. We listened to each concert and decided which had more and which had less crowd noise. Then, we distributed the songs into the sets such that each set would have a similar distribution of crowd noise and the ratio of songs per set would be similar to that in MUSDB. The train set consists of 88 songs, the validation set of 16 songs and the test set of 52 songs.

6.2 Models

For this task, we used the original configurations of the Open-Unmix and Meta-TasNet models in the following setups:

- **unmix**: Open-Unmix pretrained on MUSDB
- **meta**: Meta-TasNet pretrained on MUSDB
- **tl**: Open-Unmix initialized with pretrained weights for which we train the last two fully-connected layers on Montreux Jazz data
- **retrain**: Open-Unmix initialized randomly and trained from scratch on Montreux Jazz data
- **finetune**: Open-Unmix initialized with pretrained weights and trained from scratch on Montreux Jazz data

We also wanted to experiment with transfer learning, as for setup **tl**, on Meta-TasNet, but we could not perform the training on our GPUs due to memory limitations.

6.3 Results

After objective evaluation on the test set of the five setups, we obtained the following SDR scores:

	vocals	**drums**	**bass**	**other**	**avg**
unmix	4.51	3.24	2.53	2.96	3.31
meta	**5.14**	**3.37**	**3.47**	**3.30**	**3.81**
tl	2.32	1.35	1.38	1.27	1.58
retrain	3.92	2.10	1.68	1.90	2.40
finetune	3.92	2.90	2.16	2.56	2.88

We see that the models trained on clean data obtain scores of more than 0.4 dB higher than the ones trained on live music data. When listening to the separations performed by the different models, we notice that the ones that were trained with live music add more noise and distortion than the ones trained with clean data from MUSDB. This should be due to the fact that if we train with noisy data, the models will try to replicate that noise. However, crowd noise and random interference between instruments have no clear structure which the neural networks could identify, as opposed to clean instrument signals. Therefore models **tl**, **retrain** and **finetune** struggle to replicate the noise that they find in the Montreux Jazz dataset instead of perfecting instrument isolation and in doing so they manage to distort the separation.

We conclude from these experiments that the best way to separate live music is to train models on clean music first and then apply them to the live music data. Trying to finetune the models with live music only confuses the models and decreases their performance as they can not find any structure in the noise from the live music data.

Chapter 7

Conclusion

We brought the following contributions through our study:

- We pushed towards increasing the number of instrument classes to separate from a song. In the process, we proposed a way of predicting how well we could separate one instrument class by using an oracle model.
- We explored the possibility of using parameterized filterbanks, as a trade-off between the flexibility of convolutions and the relevance of handcrafted fixed transformations, for finding a good learnable encoding of the music signal that would enhance source separation.
- We investigated techniques for extending musical source separation from studio recorded music to live music.

In terms of extending the number of instrument categories, we concluded that this should be feasible using existing techniques, given that sufficient data is available, careful data preprocessing is performed and the instruments that we try to separate are not too much alike or have little activity. Deciding what instruments we want to separate put us in front of a complicated choice, but we managed to find an indicator for this task using the Ideal Ratio Mask oracle [4]. Overall, we were able to provide a decent separation for *electric guitar* along with the initial *vocals*, *drums* and *bass* classes found in most of the latest research.

Concerning learnable encoding representations, we discovered that it is unlikely that an encoder consisting of parameterized filterbanks only will perform better than simple convolutions. Using filterbanks along with convolutions does improve over using convolutions only. However, we can obtain better performance by replacing the filterbanks in a *hybrid* encoder with a module consisting of a spectrogram passed thorugh a convolution with kernel size 1. Thus, other filterbank types and encoder architectures using them should be explored to make filterbank use relevant.

Finally, regarding live music separation, we found that training a model on clean studio recorded tracks provides the most robust models. Trying to adapt models to replicate the noise found in the live music dataset when performing the separation only affected the quality of the separation. As noise is of random nature, finetuning models pretrained on clean data using live music data corrupted with noise led to overfitting on the noise, which in turn translated into a distorted and noisy source separation.

In our opinion, future work could focus on developing models for different genres, as these present different sets of instruments and distinct musical features (harmonics, timbre etc.). Given enough data, it may be easier to initially train models specialized per genre. Another key task, that would enable sophisticated source separation models to run in real-time on mobile devices, is to develop techniques of distilling the knowledge from large network architectures, like Meta-TasNet [33] into more lightweight architectures. For models like Open-Unmix [26], which make use of recurrent neural networks, training a small student network with a large network used for language modelling [37], could represent a starting point.

Acknowledgements

I would first of all like to thank my supervisors, Dr. Milos Cernak and Dr. Benjamin Ricaud, for all the guidance that they have provided and all the time that they have taken to meet with me almost weekly. Your advice was always spot on and it helped me move forward with my project.
Next, I would like to thank my colleague interns from Logitech for the nice moments we spent together, be it small aperos or even simply going out for lunch. The time spent at the office was so much better with you there and while working from home I always wondered how much nicer it would have been to be in the office instead.
I would also like to thank all the other people at Logitech, especially Jean-Michel and Melanie, who have taken good care of us interns and were very supportive in these complicated times, and Pablo, with whom I had the pleasure to have my hiring interview and who has also shown interest in my project.
Last but not the least, I want thank my fiance, my family and my friends for always being there for me and supporting me along these two years at EPFL. The distance was not an easy obstacle to overcome, but that made the times when we saw each other even sweeter. I love you guys.

Bibliography

[1] Zafar Rafii et al. *MUSDB18-HQ - an uncompressed version of MUSDB18*. Aug. 2019. DOI: 10.5281/zenodo.3338373. URL: https://doi.org/10.5281/zenodo.3338373.

[2] C Févotte, R Gribonval, and E Vincent. "BSS EVAL toolbox user guide. IRISA, Rennes". In: *Tech. Rep., France, Tech. Rep. 1706* (2005).

[3] Jonathan Le Roux et al. "SDR–half-baked or well done?" In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 626–630.

[4] Fabian-Robert Stöter, Antoine Liutkus, and Nobutaka Ito. "The 2018 signal separation evaluation campaign". In: *International Conference on Latent Variable Analysis and Signal Separation*. Springer. 2018, pp. 293–305.

[5] Bryan Pardo et al. *Applying source separation to music*. 2018.

[6] Paris Smaragdis and Judith C Brown. "Non-negative matrix factorization for polyphonic music transcription". In: *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No. 03TH8684)*. IEEE. 2003, pp. 177–180.

[7] François Rigaud et al. "Does inharmonicity improve an NMF-based piano transcription model?" In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 11–15.

[8] Romain Hennequin, Roland Badeau, and Bertrand David. "NMF with time–frequency activations to model nonstationary audio events". In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.4 (2010), pp. 744–753.

[9] Jean-Louis Durrieu, Bertrand David, and Gaël Richard. "A musically motivated mid-level representation for pitch estimation and musical audio source separation". In: *IEEE Journal of Selected Topics in Signal Processing* 5.6 (2011), pp. 1180–1191.

[10] Clément Laroche et al. "A structured nonnegative matrix factorization for source separation". In: *2015 23rd European Signal Processing Conference (EUSIPCO)*. IEEE. 2015, pp. 2033–2037.

[11] Hirokazu Kameoka et al. “Constrained and regularized variants of non-negative matrix factorization incorporating music-specific constraints”. In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2012, pp. 5365–5368.

[12] Manuel Davy, Simon Godsill, and Jerome Idier. “Bayesian analysis of polyphonic western tonal music”. In: *The Journal of the Acoustical Society of America* 119.4 (2006), pp. 2498–2517.

[13] İsmail An et al. “Large scale polyphonic music transcription using randomized matrix decompositions”. In: *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*. IEEE. 2012, pp. 2020–2024.

[14] Li Su and Yi-Hsuan Yang. “Combining spectral and temporal representations for multipitch estimation of polyphonic music”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.10 (2015), pp. 1600–1612.

[15] Zhiyao Duan, Bryan Pardo, and Laurent Daudet. “A novel cepstral representation for timbre modeling of sound sources in polyphonic mixtures”. In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 7495–7499.

[16] Ke Hu and DeLiang Wang. “An unsupervised approach to cochannel speech separation”. In: *IEEE Transactions on audio, speech, and language processing* 21.1 (2012), pp. 122–131.

[17] Sebastian Ewert et al. “Score-informed source separation for musical audio recordings: An overview”. In: *IEEE Signal Processing Magazine* 31.3 (2014), pp. 116–124.

[18] Zhiyao Duan and Bryan Pardo. “Soundprism: An online system for score-informed source separation of music audio”. In: *IEEE Journal of Selected Topics in Signal Processing* 5.6 (2011), pp. 1205–1215.

[19] Derry Fitzgerald. “Harmonic/percussive separation using median filtering”. In: *Proceedings of the International Conference on Digital Audio Effects (DAFx)*. Vol. 13. 2010.

[20] Zafar Rafii and Bryan Pardo. “A simple music/voice separation method based on the extraction of the repeating musical structure”. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2011, pp. 221–224.

[21] Derry FitzGerald. “Vocal separation using nearest neighbours and median filtering”. In: (2012).

[22] Antoine Liutkus et al. “Kernel additive models for source separation”. In: *IEEE Transactions on Signal Processing* 62.16 (2014), pp. 4298–4310.

[23] Paris Smaragdis and Gautham J Mysore. “Separation by “humming”: User-guided sound extraction from monophonic mixtures”. In: *2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. IEEE. 2009, pp. 69–72.

[24] Naoya Takahashi, Nabarun Goswami, and Yuki Mitsufuji. “Mmdenselstm: An efficient combination of convolutional and recurrent neural networks for audio source separation”. In: *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*. IEEE. 2018, pp. 106–110.

[25] Gao Huang et al. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[26] Fabian-Robert Stöter et al. “Open-unmix-a reference implementation for music source separation”. In: (2019).

[27] Francesc Lluís, Jordi Pons, and Xavier Serra. “End-to-end music source separation: is it possible in the waveform domain?” In: *arXiv preprint arXiv:1810.12187* (2018).

[28] Aaron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

[29] Daniel Stoller, Sebastian Ewert, and Simon Dixon. “Wave-u-net: A multi-scale neural network for end-to-end audio source separation”. In: *arXiv preprint arXiv:1806.03185* (2018).

[30] Yi Luo and Nima Mesgarani. “Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation”. In: *IEEE/ACM transactions on audio, speech, and language processing* 27.8 (2019), pp. 1256–1266.

[31] Yi Luo and Nima Mesgarani. “Tasnet: time-domain audio separation network for real-time, single-channel speech separation”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 696–700.

[32] Alexandre Défossez et al. “Music source separation in the waveform domain”. In: *arXiv preprint arXiv:1911.13254* (2019).

[33] David Samuel, Aditya Ganeshan, and Jason Naradowsky. “Meta-learning Extractors for Music Source Separation”. In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 816–820.

[34] *Karaoke Version*. URL: https://www.karaoke-version.com/ (visited on 08/07/2020).

[35] Laurent Colbois. “Trainable filterbanks for end-to-end speech enhancement via time-frequency masking”. MA thesis. EPFL, 2020.

[36] Mirco Ravanelli and Yoshua Bengio. “Speaker recognition from raw waveform with sincnet”. In: *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2018, pp. 1021–1028.

[37] Yangyang Shi et al. “Knowledge distillation for recurrent neural network language modeling with trust regularization”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 7230–7234.

Appendix A

Learnable signal encodings experiments

In our filterbank experiments, we have tested several setups, which we include here. The ones marked in **bold** are mentioned in the thesis and are the most representative. Experiments are grouped in categories based on the type of encoder or type of experiment performed.
The parameters of the Conv-TasNet network are presented in table A.1. For every configuration, we mention only the parameters that were changed from the default value. The sample rate is 8 kHz for all models except the ones in the last section, "Large models".

Name	**Description**	**Default**
B	input dimension of the conv blocks in the separation module	160
bs	batch size to use in training	4
causal	whether the system is causal	False
encoding	type of encoding to use from ['simple', 'filterbanks', 'hybrid', 'deep']	'hybrid'
fb_ratio	proportion of the channels in the encoder which are used for filterbanks	0.5
fb_type	type of filterbank to use chosen from ['conv', 'param_sinc', 'pmpgtf', 'stft']; for filterbanks encoder only 'param_sinc and 'pmpgtf' (parameterized gammatone filter) are allowed	'pmpgtf'
filters	number of layers of convolutions and filterbanks to use	1
gammatone_init	initialize the encoder and decoder convolutions with gammatone filter parameters	False
H	hidden size in the conv blocks of the separation module	160
ideal_mask	use the ideal ratio mask for separation	False
init_type	what type of initialization to use for the filterbank from ['mel' ,'uniform']	'mel'
kernel	kernel size in the conv blocks in the separation module	3
L	kernel size in the encoder and decoder	20
large_tcn	use filters for each source in every layer of the separation module	False
layers	number of conv blocks in a stack of the separation module	10
lr	learning rate to use for training	0.001
lr_decay_gamma	learning rate decay factor used when validation loss does not improve after some epochs	0.5
lr_decay_patience	number of epochs with no improvement in validation loss after which to decay the learning rate	3
N	number of channels of the signal encoding	440
sample_rate	sample rate of the signal in kHz	44.1
seed	random seed used for reproducibility	42
sources	the sources which to separate	['drums, 'bass', 'other', 'vocals']
stack	number of stacks of conv blocks in the separation module	4
threads	number of workers used for loading data from disk	10
time_length	number of seconds of the chunks of signals that are fed to the network	8
track_samples	number of chunks to sample for each song in the train dataset in an epoch	64
W	stride in the encoder	20
weight_decay	weight decay used in training	0.0005
who_is_pinv	'enc' if the encoder is the pseudoinverse of the decoder, 'dec' if the decoder is the pseudoinverse of the encoder and None if the encoder and decoder are independent	None

Table A.1: Parameters of the Conv-TasNet model with filterbanks

A.1 One filterbank/convolution encoder

Here the encoder and decoder consist of only one layer, be it convolution or filterbank.

A.1.1 Ideal Ratio Mask experiments

The experiments in this section are performed with the Ideal Ratio Mask oracle, so the *ideal_mask* parameter is True.

- **irm_fb_sinc_free**

bs	encoding	fb_type	L	W
12	'filterbanks'	'param_sinc'	25	25

- irm_fb_sinc_free_uniform

bs	encoding	fb_type	init_type	L	W
12	'filterbanks'	'param_sinc'	'uniform'	25	25

- irm_fb_sinc_free_uniform_lr0.003

bs	encoding	fb_type	init_type	L	lr	W
12	'filterbanks'	'param_sinc'	'uniform'	25	0.003	25

- irm_fb_sinc_free_uniform_lr0.005

bs	encoding	fb_type	init_type	L	lr	W
12	'filterbanks'	'param_sinc'	'uniform'	25	0.005	25

- irm_fb_sinc_free_uniform_lr0.007

bs	encoding	fb_type	init_type	L	lr	W
12	'filterbanks'	'param_sinc'	'uniform'	25	0.007	25

- irm_fb_sinc_free_uniform_lr0.01

bs	encoding	fb_type	init_type	L	lr	W
12	'filterbanks'	'param_sinc'	'uniform'	25	0.01	25

- irm_fb_sinc_free_uniform_lr0.03

bs	encoding	fb_type	init_type	L	lr	W
12	'filterbanks'	'param_sinc'	'uniform'	25	0.03	25

- irm_fb_sinc_free_uniform_lr0.05

bs	encoding	fb_type	init_type	L	lr	W
12	'filterbanks'	'param_sinc'	'uniform'	25	0.05	25

- **irm_fb_sinc_pseudodec**

bs	encoding	fb_type	L	W	who_is_pinv
12	'filterbanks'	'param_sinc'	25	25	'dec'

- irm_fb_sinc_pseudodec_uniform

bs	encoding	fb_type	init_type	L	W	who_is_pinv
12	'filterbanks'	'param_sinc'	'uniform'	25	25	'dec'

- irm_fb_sinc_pseudodec_uniform_lr0.003

bs	encoding	fb_type	init_type	L	lr	W	who_is_pinv
12	'filterbanks'	'param_sinc'	'uniform'	25	0.003	25	'dec'

- irm_fb_sinc_pseudodec_uniform_lr0.005

bs	encoding	fb_type	init_type	L	lr	W	who_is_pinv
12	'filterbanks'	'param_sinc'	'uniform'	25	0.005	25	'dec'

- irm_fb_sinc_pseudodec_uniform_lr0.007

bs	encoding	fb_type	init_type	L	lr	W	who_is_pinv
12	'filterbanks'	'param_sinc'	'uniform'	25	0.007	25	'dec'

- irm_fb_sinc_pseudodec_uniform_lr0.01

bs	encoding	fb_type	init_type	L	lr	W	who_is_pinv
12	'filterbanks'	'param_sinc'	'uniform'	25	0.01	25	'dec'

- irm_fb_sinc_pseudodec_uniform_lr0.03

bs	encoding	fb_type	init_type	L	lr	W	who_is_pinv
12	'filterbanks'	'param_sinc'	'uniform'	25	0.03	25	'dec'

- irm_fb_sinc_pseudodec_uniform_lr0.05

bs	encoding	fb_type	init_type	L	lr	W	who_is_pinv
12	'filterbanks'	'param_sinc'	'uniform'	25	0.05	25	'dec'

- irm_fb_sinc_enc_conv_dec
  The encoder consists of a sinc filterbank ant the decoder of a transpose convolution.

bs	encoding	fb_type	init_type	L	W
12	'filterbanks'	'param_sinc'	'uniform'	25	25

- **irm_simple**

bs	encoding	L	W
12	'simple'	25	25

- irm_fb_pmpgtf_free

bs	encoding
12	'filterbanks'

- **irm_fb_pmpgtf_pseudodec**

bs	encoding	who_is_pinv
12	'filterbanks'	'dec'

- irm_fb_pmpgtf_L100_W50

bs	encoding	L	W
12	'filterbanks'	100	50

- irm_fb_pmpgtf_L100_W25

bs	encoding	L	W
12	'filterbanks'	100	25

- irm_fb_pmpgtf_L100_W10

bs	encoding	L	W
12	'filterbanks'	100	10

The initial experiments involved sinc filterbanks with a default kernel size of 25. They were used either with mel initialization or uniform initialization. After these experiments we concluded that mel initialization works better, pseudo-inverse decoders are better than decoders trained independently of the encoder, and that increasing the learning rate from the default 0.001 helps. We also noticed that using free convolutions instead of sinc filterbanks helps, either if we use convolutions in both encoder and decoder or we mix one filterbank in the encoder with a convolution in the decoder.
As an intermediary step, we tried initializing simple conv filters with the values from a gammatone filterbank. The results were worse then when doing a Xavier initialization of the convolutions.
The next step was trying gammatone filterbanks as well, with a default kernel size of 20. Using a pseudo-inverse decoder proved helpful here as well, but not as much as before. We therefore increased the kernel size to 100 for an independent decoder and tried several strides. The performance for a kernel size of 100 proved better than that for 20 and decreasing the stride also translated in a performance increase. Using gammatone filterbanks clearly improved over the sinc filterbanks and when using a kernel size of 100 and strides above 25, we even surpassed the performance of simple convolutions in the encoder and decoder.
The conclusion is that gammatone filterbanks with a large kernel size and much overlap (small stride) are promising to use instead of simple convolutions, but take longer to train then when using less overlap.

A.1.2 Conv-TasNet experiments

The experiments in this section are performed with the a Conv-TasNet architecture with the separator consisting of 3 stacks with 6 conv blocks each. Therefore stacks=3 and layers=6.

- large_fb_sinc_free

bs	encoding	fb_type	H	L	W
8	'filterbanks'	'param_sinc'	640	25	25

- large_fb_sinc_free_uniform

bs	encoding	fb_type	init_type	H	L	W
8	'filterbanks'	'param_sinc'	'uniform'	640	25	25

- **large_fb_sinc_pseudodec**

bs	encoding	fb_type	H	L	W	who_is_pinv
8	'filterbanks'	'param_sinc'	640	25	25	'dec'

- large_fb_sinc_pseudodec_uniform

bs	**encoding**	**fb_type**	**init_type**	**H**	**L**	**W**	**who_is_pinv**
8	'filterbanks'	'param_sinc'	'uniform'	640	25	25	'dec'

- **large_simple**

bs	**encoding**	**H**	**L**	**W**
8	'simple'	640	25	25

- large_simple_L20_W20

bs	**encoding**	**H**
8	'simple'	640

- **large_fb_pmpgtf_free**

bs	**encoding**	**H**
7	'filterbanks'	640

- **large_fb_pmpgtf_pseudodec**

bs	**encoding**	**H**	**who_is_pinv**
7	'filterbanks'	640	'dec'

We confirm again for sinc filterbanks that mel initialization and pseudoinverse decoders work better. Also, convolutions are again usually better than sinc filterbanks and convolutions with kernel size 25 are better than those with kernel size 20. The demucs model provides even worse performance than the simple convolutional model with kernel size 20. For gammatone filterbanks we also get better performance when using pseudoinverse decoders. They also work better than sinc filterbanks and get comparable performance to the convolutions.
The conclusion is that we generally confirm the results of the ideal ration mask model, especially that gammatone filterbanks are comparable to simple convolutions.

We conclude that gammatone filterbanks could prove to be a good replacement for simple convolutions, but they do not improve over them. They are also preferable to sinc filterbanks.

A.2 Hybrid encoder

Here the encoder consists of a combination of convolutions and filterbanks, while the decoder is composed of convolutions only (except for the first two experiments for Ideal Ration Mask and the first for Real separator).

A.2.1 Ideal Ratio Mask experiments

The experiments in this section are performed with the Ideal Ratio Mask oracle, so the ideal_mask parameter is True.

- irm_hybrid_sinc_free
  The encoder and decoder consist of one conv layer and one sinc filterbank each. The filterbanks are independent.

bs	fb_type	L	W
12	'param_sinc'	25	25

- irm_hybrid_sinc_pseudodec
  The encoder and decoder consist of one conv layer and one sinc filterbank each. The filterbank in the decoder is the pseudoinverse of the one in the encoder.

bs	fb_type	L	W	who_is_pinv
12	'param_sinc'	25	25	'dec'

- irm_hybrid_conv

bs	fb_type	L	W
12	'conv'	25	25

- **irm_hybrid_sinc**

bs	fb_type	L	W
12	'param_sinc'	25	25

- **irm_hybrid_pmpgtf**

bs
12

- irm_hybrid_stft

bs	fb_type	L	W
12	'stft'	25	25

- irm_hybrid_pmpgtf_stft_capacity
  The encoder uses a gammatone filterbank on top of which we add a convolution with kernel size 1, like we had for the spectrogram in irm_hybrid_stft.

bs
12

In the first two experiments we used one conv and one sinc filterbank both in the encoder and the decoder. We noticed that there is no difference in the performance if we restrict the filterbank in the decoder to be the pseudo-inverse of the one in the encoder or not. If we use a decoder consisting only of convolutions instead, we get a bit of improvement, so we decided to stick to the configuration with convolutions only in the decoder.
When comparing hybrid encoders with different filterbanks, we saw that the order in which they perform is stft, gammatone, sinc, conv. Therefore, the gammatone filterbank proves to be a better choice than convolutions in this scenario. By increasing the capacity of the encoder using gammatone filterbanks by adding a conv layer with kernel size 1 on top of the filterbank, just like we had for stft, we don't improve much over the initial hybrid gammatone encoder. This means that the best encoder setup remains the one of Meta-Tasnet.

A.2.2 Conv-TasNet experiments

The experiments in this section are performed with the a Conv-TasNet architecture with the separator consisting of 3 stacks with 6 conv blocks each. Therefore stacks=3 and layers=6.

- large_hybrid_sinc_free
  The encoder and decoder consist of one conv layer and one sinc filterbank each. The filterbanks are independent.

bs	fb_type	H	L	W
8	'param_sinc'	640	25	25

- **large_hybrid_conv**

bs	fb_type	H	L	W
8	'conv'	640	25	25

- **large_hybrid_sinc**

bs	fb_type	H	L	W
8	'param_sinc'	640	25	25

- **large_hybrid_pmpgtf**

bs	H
7	640

- **large_hybrid_stft**

bs	fb_type	H	L	W
8	'stft'	640	25	25

- large_hybrid_pmpgtf_stft_capacity
  The encoder uses a gammatone filterbank on top of which we add a convolution with kernel size 1, like we had for the spectrogram in large_hybrid_stft.

bs	H
8	640

- large_hybrid_pmpgtf_L100_W20

bs	H	L	W
7	640	100	20

- large_hybrid_pmpgtf_filters2

bs	filters	H
7	2	640

- large_hybrid_stft_filters2

bs	fb_type	filters	H
8	'stft'	2	640

- large_hybrid_pmpgtf_N256

bs	H	N
7	640	256

- large_hybrid_stft_N256

bs	fb_type	H	N
8	'stft'	640	256

- **large_hybrid_pmpgtf_pretrained_encoder**
  The encoder and decoder are fixed to the ones obtained after training irm_hybrid_pmpgtf and only the separation module is changed.

bs	H
7	640

- large_hybrid_pmpgtf_large_tcn
  The encoder consists of one conv layer and one gammatone filterbank layer. The encoding kernel size and the stride are 20. The separation module has 4 times more channels in every layer than before, as we try to allocate channels for each of the source classes.

bs	H	large_tcn
8	640	True

Again, a decoder composed only of convolutions performs slightly better. Almost the same order applies in terms of the performance of the different filterbanks in the encoder, with the exception that the conv filterbank is the worst performing one now.
Using a hybrid gammatone encoder again brings no significant improvement, which leaves the Meta-Tasnet encoder with the best performance.
Increasing the kernel size for gammatone actually diminishes the performance. Using two conv filters and two gammatone filterbanks doesn't improve either.
If we instead use two conv filters and an stft, we do improve over the one conv filter + stft setup.
Using a smaller encoding for the hybrid gammatone (256 instead of 440), decreases the overall performance only very slightly.
Using a pretrained encoder and decoder for the hybrid gammatone model leads to a drop in performance.
If for a hybrid gammatone encoder we increase the capacity of the separation module, we get better performance then with the hybrid stft encoder with a normal capacity separator.
Increasing the number of filterbanks in the encoder reduces the performance, so the conv layer is still the one that keeps a good performance through its flexibility.

We conclude that a hybrid encoder consisting of convolutions and stft is probably the best choice.

A.3 Deep encoder

Here the encoder and decoder consist of stacked layers, instead of layers ran in parallel for which we concatenate the outputs, as we had in the "Hybrid encoder" section.

A.3.1 Ideal Ratio Mask experiments

The experiments in this section are performed with the Ideal Ratio Mask oracle, so the ideal_mask parameter is True.

- irm_stacked2_fb_pmpgtf_conv
  The encoder consists of a stack of one gammatone filterbank and one convolutional layer. This is a custom implementation not reproducible with the last version of the code.

- irm_stacked3_fb_pmpgtf_conv
  The encoder consists of a stack of one gammatone filterbank and two convolutional layers. This is a custom implementation not reproducible with the last version of the code.

When stacking a gammatone filterbank and a conv layer and using a deep encoder, we get better performance than with an encoder and a decoder consisting of one gammatone filterbank each. However, if we stack a gammatone filterbank and two conv layers, the model immediately gets stuck on small weights, leading to a score of zero.

A.3.2 Conv-TasNet experiments

The experiments in this section are performed with the a Conv-TasNet architecture with the separator consisting of 3 stacks with 6 conv blocks each. Therefore stacks=3 and layers=6.

- large_deep2_pmpgtf_L20_bs8

bs	**encoding**	**filters**	**H**
8	'deep'	2	640

- large_deep2_pmpgtf_L20_bs16

bs	**encoding**	**filters**	**H**
16	'deep'	2	640

- large_deep2_pmpgtf_L16_bs8

bs	**encoding**	**filters**	**H**	**L**	**W**
8	'deep'	2	640	16	16

- large_deep2_pmpgtf_L16_bs16

bs	**encoding**	**filters**	**H**	**L**	**W**
16	'deep'	2	640	16	16

- large_deep2_pmpgtf_L10_bs8

bs	**encoding**	**filters**	**H**	**L**	**W**
8	'deep'	2	640	10	10

The only viable option is to use a small kernel size if we make deep encoders. The best performance of a deep encoder, for a kernel size of 10, is only slightly better than if we were to use only one filterbank in the encoder, but also a filterbank in the decoder, instead of a deep convolutional structure.

We conclude that using deep encoders does not look promising, especially if we use too large kernel sizes.

A.4 Change number of channels of gammatone filterbanks in encoder

These experiments were performed on a Conv-TasNet architecture with a separation module consisting of 3 stacks of 6 conv blocks each.

- **large_hybrid_pmpgtf_fratio0.0_H160**

bs	**fb_ratio**
16	0.0

- **large_hybrid_pmpgtf_fratio0.5_H160**

bs
16

- **large_hybrid_pmpgtf_fratio0.75_H160**

bs	**fb_ratio**
16	0.75

- **large_hybrid_pmpgtf_fratio1.0_H160**

bs	**fb_ratio**
16	1.0

- large_hybrid_pmpgtf_fratio0.0_H640

bs	**fb_ratio**	**H**
7	0.0	640

- large_hybrid_pmpgtf_fratio0.5_H640

bs	**H**
7	640

- large_hybrid_pmpgtf_fratio0.75_H640

bs	**fb_ratio**	**H**
7	0.75	640

- large_hybrid_pmpgtf_fratio1.0_H640

bs	**fb_ratio**	**H**
7	1.0	640

For a hidden size of 160, all models seem to perform similarly overall. The models have very noisy validation scores for vocals, except for large_hybrid_pmpgtf_fratio0.75_H160. The performance on the other class is very poor in all situations.
When moving to a hidden size of 640, there is not much difference in the evolution of the validation scores. Still, having no filterbanks at all seems to be a bit better in all cases except for other, where having 3/4 filterbanks is the best.
Increasing the hidden size from 160 to 640 did not translate into an increase of performance.

A.5 Change number of channels of sinc filterbanks in encoder

- **large_hybrid_sinc_fratio0.0_H160**

bs	**fb_type**	**fb_ratio**
16	'param_sinc'	0.0

- **large_hybrid_sinc_fratio0.5_H160**

bs	**fb_type**
16	'param_sinc'

- **large_hybrid_sinc_fratio0.75_H160**

bs	**fb_type**	**fb_ratio**
16	'param_sinc'	0.75

- **large_hybrid_sinc_fratio1.0_H160**

bs	**fb_type**	**fb_ratio**
16	'param_sinc'	1.0

When using only filterbanks, we clearly have the worst performance. The other three setups are close to each other, but the learning does seem stabler when including filterbanks. Overall, large_hybrid_sinc_fratio0.75_H160 has a bit of an edge over large_hybrid_sinc_fratio0.5_H160.

A.6 Large models

Here we use models trained on 44.1 kHz signals.

- full_data_large_hybrid_pmpgtf_large_tcn

t

L	**large_tcn**	**layers**	**stacks**	**W**
100	True	6	3	100

- full_data_huge_hybrid_pmpgtf_large_tcn

L	**large_tcn**	**W**
100	True	100

- **full_data_huge_hybrid_pmpgtf_L100_W100**

L	**W**
100	100

- full_data_huge_hybrid_pmpgtf_L50_W50

L	**W**
50	50

full_data_huge_hybrid_pmpgtf_large_tcn has better performance than full_data_large_hybrid_pmpgtf_large_tcn, as we increase the separation module size, and for the same reason it performs better than full_data_huge_hybrid_pmpgtf_L100_W100. However, full_data_huge_hybrid_pmpgtf_L50_W50 provides better scores at test time than full_data_huge_hybrid_pmpgtf_large_tcn, which uses a separator module 4 times the size.