Demucs

Neural Networks project

Simone Bartolini 1752197

1 Introduction

For this project, I've reimplemented in a simplified form the first version of the Demucs neural network, a model for music source separation. It has been proposed in the paper [2] and the source code is available on GitHub [1].
I've coded my version using python and the PyTorch library and trained it using Google Colab to have access to more powerful hardware than the one I have in my computer. My source code is also available on GitHub.
In the rest of this document:

- I describe the structure of the network and the changes I made with respect to the model presented in the paper in section 2;
- I detail the dataset generation and data augmentation in section 3;
- I define the training procedure in section 4;
- I discuss the evaluation technique and compare my results to the one in the paper in section 5;
- I provide my closing thoughts in section 6.

2 Model

Demucs is an

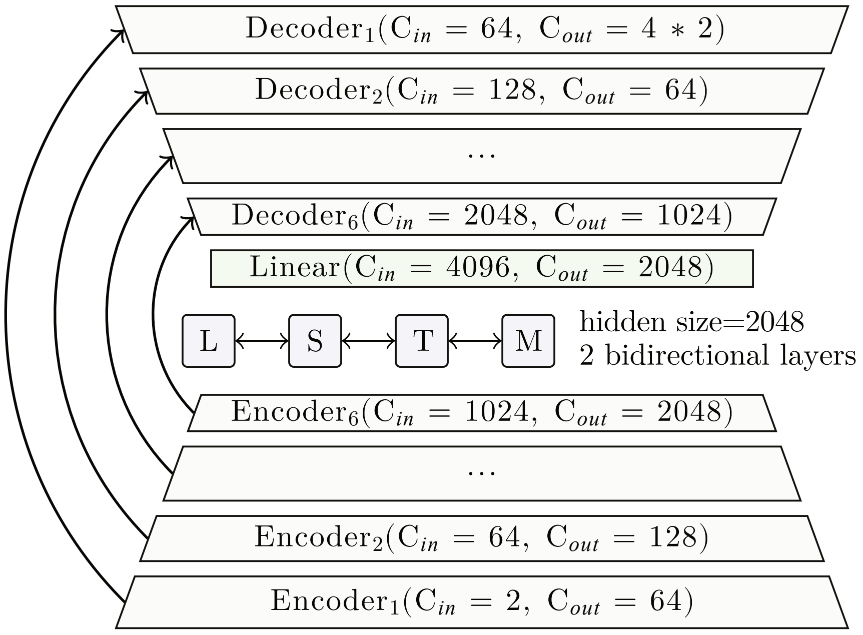
> *encoder/decoder architecture composed of a convolutional encoder, a bidirectional LSTM, and a convolutional decoder, with the encoder and decoder linked with skip U-Net connections [2].*

It takes in input a stereo mixture and produces in output four stereo tracks, one for each source (*vocals*, *drums*, *bass* and *others*).

The network is structured as follows (figure 1):

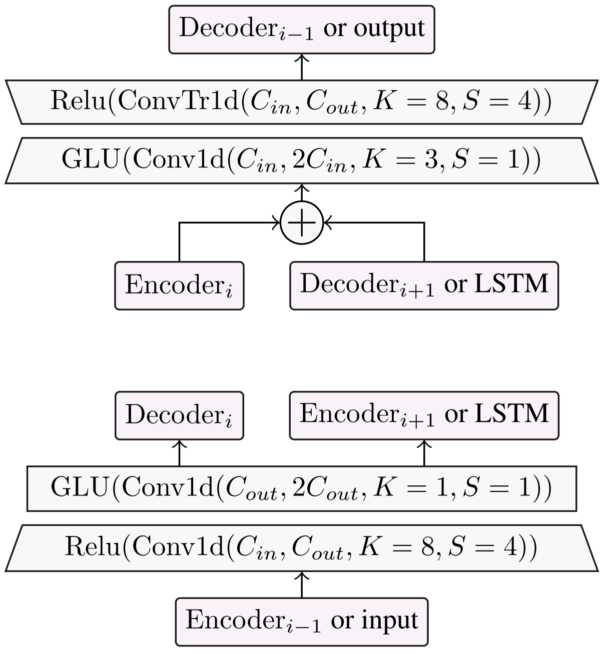
- The encoder is made of six blocks, numbered from 1 to 6, each composed of a convolution with a kernel size of 8, a stride of 4 and ReLU activation and another convolution with a kernel size of 1, a stride of 1 and GLU activation. The first convolution produces in output $2^{i-1} * C$ channels, where i is the block number and $C = 64$, while the second convolution at first doubles the output channels but then the GLU activation layer halves it again, thus the final output of the block is still $2^i * C$ channels. The first block takes in input the two channels of the input mixture, while the remaining blocks take in input the output channels of the preceding block.
- The bidirectional LSTM has two layers and a hidden size equal to the number of output channels of the encoder, i.e., 2048. The 4096 output channels of the LSTM are then passed through a linear layer that halves them to 2048.
- The decoder is also made of six blocks, this time numbered from 6 to 1, each one composed of a convolution with a kernel size of 3, a stride of 1 and GLU activation and a transposed convolution with a kernel size of 8, a stride of 4 and ReLU activation. The first block takes in input the 2048 channels outputted by the linear layer, while the remaining blocks take in input the output channels of the preceding block. Additionally, the input of each decoder block is summed with the output of the relative encoder block with the same number (skip U-Net connections). Inside each block, the first convolution produces in output $2^i * C$ channels, where i is the block number and $C = 64$, that get halved by the GLU activation and then halved again by the second convolution, thus the whole block produces in output $2^{i-2} * C$ channels, except for the last block, where the transposed convolution gives in output 8 channels, i.e., two channels for each of the four output tracks.

Note that the original model had C=100, but to reduce the size of the model I've changed it to 64, as suggested in the source to make it possible to train the model on a system with a limited amount of RAM and VRAM.



Decoder$_1$(C$_{in}$ = 64, C$_{out}$ = 4 ∗ 2)
Decoder$_2$(C$_{in}$ = 128, C$_{out}$ = 64)
...
Decoder$_6$(C$_{in}$ = 2048, C$_{out}$ = 1024)
Linear(C$_{in}$ = 4096, C$_{out}$ = 2048)
L
S
T
M
hidden size=2048
2 bidirectional layers
Encoder$_6$(C$_{in}$ = 1024, C$_{out}$ = 2048)
...
Encoder$_2$(C$_{in}$ = 64, C$_{out}$ = 128)
Encoder$_1$(C$_{in}$ = 2, C$_{out}$ = 64)

(a) Demucs architecture (modified version of the image in [2]).



Decoder$_{i-1}$ or output
Relu(ConvTr1d($C_{in}, C_{out}, K = 8, S = 4$))
GLU(Conv1d($C_{in}, 2C_{in}, K = 3, S = 1$))
Encoder$_i$
Decoder$_{i+1}$ or LSTM
Decoder$_i$
Encoder$_{i+1}$ or LSTM
GLU(Conv1d($C_{out}, 2C_{out}, K = 1, S = 1$))
Relu(Conv1d($C_{in}, C_{out}, K = 8, S = 4$))
Encoder$_{i-1}$ or input

(b) Detailed view of a decoder (top) and an encoder (bottom) block [2].

Figure 1: Model

3 Dataset

The model has been trained on the *musdb18* dataset [3]. *musdb18* is a dataset of 150 stereo songs, purposely developed for training and evaluation of source separation models. For each song it contains the full mixture plus the separate tracks for the sources (*vocals*, *drums*, *bass* and *others*), that summed together form the mixture. The dataset is divided into 84 songs for training, 16 songs for validation and 50 songs for testing.
In the original code, the songs were passed in input to the network as 11-second extracts with a stride of 1 second. This resulted in extremely long training time (multiple days) on the hardware I've access to, i.e. an Nvidia Tesla K4 with 16GB of VRAM obtained through Google Colab (note that the original network has been trained on 16 Nvidia V100 GPUs, each with 32GB of VRAM). For this reason, I've decided to extract at random only a limited number of 11-seconds chunks from each song, specifically 1.25 chunks per minute of song, effectively reducing the size of the dataset and thus the training time to around 6 hours (of course at the cost of the performance of the trained model).
The data is normalized using z-score normalization, i.e., the mean of each song is subtracted from every 11-second chunk extracted from it, and the result is divided by the standard deviation of the song. After that, each chunk is shifted in time by a random number between 0 and 1 and only 10 seconds are kept.
Then the following data augmentation is performed:

- randomly flipping the audio channels;
- randomly flipping the sing;
- shuffling the sources within a batch.

4 Training

4.1 Weight initialization

Correctly initializing the weight of the network can have a huge impact on performance. The authors of the paper take a peculiar approach: the weight of the convolutions are initialized at random and then rescaled in order to get their standard deviation closer to a target value of $a = 0.1$. This is done as follows: let's denote with w the initial value of a weight. We then compute $\alpha = \mathrm{std}(w)/a$ and replace w with $w' = w/\sqrt{\alpha}$.

4.2 Loss function

The paper suggests two possible loss functions to use:

1. the average absolute error $L_1(\hat{x}_s, x_s) = \frac{1}{T}\sum_{t=1}^{T} |\hat{x}_{s,t} - x_{s,t}|$;
2. the average mean square error $L_2(\hat{x}_s, x_s) = \frac{1}{T}\sum_{t=1}^{T} (\hat{x}_{s,t} - x_{s,t})^2$;

where x_s is a waveform containing T samples corresponding to the source s, $\hat{x}_s$ is a predicted waveform and the subscript t denotes the t-th sample of a waveform.
I've trained and evaluated the model using both loss functions.

4.3 Training cycle

After the initialization, the model is trained using the Adam trainer for 240 epochs with a learning rate of $3e-4$ and passing the data examples in input at batches of 8 (note that this is reduced from the original batch size of 64, again due to memory limitation of the hardware at my disposal).
The images below (figures 2 and 3) describe the evolution of the loss over the course of the training for the two variants (note that I've computed the validation loss only every 10 epochs).

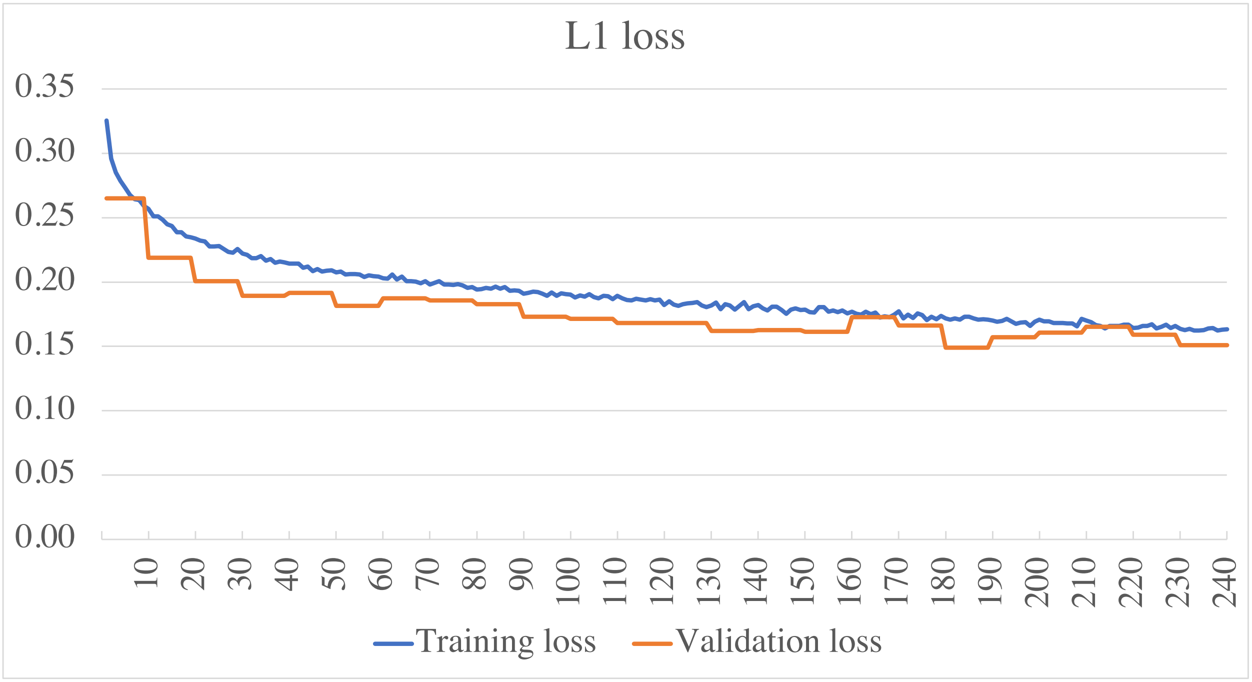


Figure 2: L1 loss during training

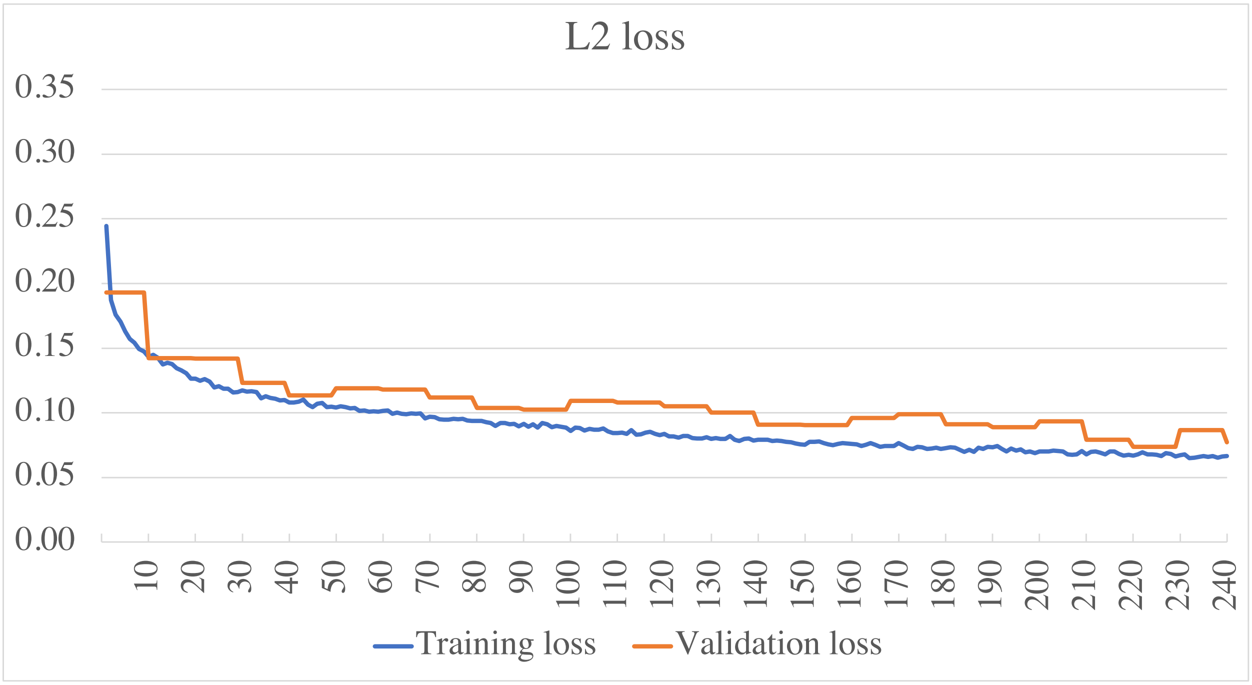


Figure 3: L2 loss during training

5 Results

5.1 The shift trick

As is, the Demucs model is not time equivariant, i.e., shifting the input mixture by X samples will not shift the output by the same amount. This reduces the accuracy of the model. To mitigate the problem, the paper suggests a simple workaround called *randomized equivariant stabilization* (or also the *shift trick*) that consist of sampling $S = 10$ random shifts of an input mixture and averaging the prediction of the model for each, after having applied the opposite shift, to compute the final output. Note that this is only done during evaluation and not training as it would substantially increase the time required.

5.2 Evaluation procedure

The evaluation is done using the 50 testing songs in the dataset. First, the average loss (either L1 or L2, depending on which has been used for training) is computed, then the predictions are analyzed using the `museval` python package [5], which implements the SiSec Mus 2018 [4] evaluation that is used in literature for evaluating source separation models trained over the *musdb18* dataset. The package computes four metrics for each source:

- Source to Distortion ratio (SDR);
- Source to Artefact ratio (SAR);
- Source to Interference ratio (SIR);
- Image to Spatial distortion ratio (ISR).

These metrics are computed for each sample of each song and the final result is the median over all songs. Out of the four metrics, the primary one used to judge the quality of the source separation is the SDR. The tables below (tables 1 and 2) show the results for each of the four output tracks plus the average over all tracks.

	All	**Vocals**	**Drums**	**Bass**	**Other**
SDR	***3.961***	3.137	5.302	5.291	2.113
SAR	4.872	3.387	6.105	5.418	4.577
SIR	7.274	8.636	8.783	9.229	2.446
ISR	7.427	5.086	8.611	9.211	6.798
L1 loss	0.194				

Table 1: Results of training using L1 loss

	All	**Vocals**	**Drums**	**Bass**	**Other**
SDR	***3.928***	3.663	5.077	4.479	2.494
SAR	5.534	4.876	5.300	6.896	5.064
SIR	5.936	7.438	6.177	7.467	2.662
ISR	7.722	6.375	9.519	8.705	6.287
L2 loss	0.103				

Table 2: Results of training using L2 loss

5.3 Results analysis

As the results highlight, the model trained using the L1 loss performs slightly better on average, although the L2 variant separated better the *vocals* and *other* tracks. Also the authors of the paper found a small improvement when using the L1 loss, with an average SDR of 5.58 compared to 5.55 when using L2.
The table 3 shows the SDR, SAR, SIR and L1 loss of the original model (note that the ISR is missing because it was not reported in the paper). The changes I made to make it possible to train the model on less powerful hardware definitely had an impact on the quality of the separation, especially with the *vocals* and *other* tracks, which is not surprising as they are the most difficult to separate. This is consistent with what I can hear listening to some of the output tracks of my model compared to the original: while the *drums* and *bass* tracks are mostly ok, there is a lot of cross bleeding in the *vocals* and *other* tracks, definitely more than in the output of the original model (although even that struggles a bit with separating the two).

	All	**Vocals**	**Drums**	**Bass**	**Other**
SDR	***5.58***	6.29	6.08	5.83	4.12
SAR	6.08	6.54	6.18	6.41	5.18
SIR	10.39	13.31	11.81	10.55	5.90
L1 loss	0.164				

Table 3: Results of training of the original model

6 Conclusion

Demucs can achieve really impressive results in the separation of music sources. My reimplementation falls shorts of reaching the same quality due to the aforementioned reduction of the size of the model and of the dataset, although it requires significantly less powerful hardware and time to train. With more time at my disposal, it could be interesting to test if it is possible to achieve better results with my model by changing parameters like the learning rate or the rescale factor a at initialization, or by increasing the training epochs since the validation loss did not show signs of overfitting.

References

[1] Défossez, A., Usunier, N., Bottou, L., and Bach, F. Demucs. Available from: https://github.com/facebookresearch/demucs/tree/v1.

[2] Défossez, A., Usunier, N., Bottou, L., and Bach, F. Music source separation in the waveform domain (2019). Available from: https://arxiv.org/pdf/1911.13254v1.pdf.

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

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

[5] Stöter, F.-R., Liutkus, A., Samuel, D., Miner, L., Voituret, F., pyup.io bot, Bot, S., and tobeperson. sigsep/sigsep-mus-eval: museval 0.4.0 (2021). Available from: https://doi.org/10.5281/zenodo.4486535, doi:10.5281/zenodo.4486535.