CS230

DeepDeMix: Music Source Separation via Contrastive Pretraining

Ivan Villa-Renteria
Department of Computer Science
Stanford University
ivillar@cs.stanford.edu

1 Introduction

We propose the use of supervised contrastive pretraining for the task of music source separation. Music source separation is a particular case of audio source separation, where given a mixture of audio signals from different sources, we wish to recover the individual source signals [1]. Here, the mixed audio signals sources are musical instruments. Music source separation is of particular interest in artificial intelligence, music signal processing, and music information retrieval (MIR) because it allows music artists to recover individual music instrument recordings from musical tracks and manipulate them to enhance the tracks or create novel music. Moreover, the results from music source separation can be used for other downstream tasks such as automatic music transcription and sorround sound generation. We hypothesize that the use of supervised contrastive learning, [2] which is often used to learn useful representations for downstream tasks, can be applied to music source separation.

2 Related Work

Usual deep learning models for music source separation consist of an encoder, a separation module, and a decoder [1]. The encoder takes as input the signal mixture and generates an appropriate encoding for it. The separation module then computes a mask to apply to the mixture signal's encoding; this allows us to obtain the encoding corresponding to the source we'd like to separate from the mixture. The decoder then attempts to reconstruct the source signal from the encoding obtained from the separation module after applying the mask. Spectrogram-based models apply a Fourier Transform to obtain a fixed time-frequency encoding, whereas waveform-based models learn the encoding and decoding alongside the separation module in an end-to-end manner.

MMDenseLSTM [3] is a spectrogram-based model which combines DenseNet and an LSTM module. This model manages to outpeform previous attempts at music source separation using DenseNet and LSTMs separately, while reducing the total number of parameters used. Waveform-based models in the field take inspiration from Conv-TasNet [4], speech separation model which consists of a linear layer to learn encodings from for the encoder, Temporal Convolutional Networks (TCN) for the separation module, and final linear for the decoder. This model achieved an SDRi score of 13.4 dB, outperforming previous waveform-based speech separation models. Other attempts have been made at deep learning models combining both spectrogram and waveforms. [5] [6].

Supervised Contrastive Pretraining is a procedure in which some sort of model that outputs representations for datapoints is trained on a dataset in order to strengthen the quality of said representations. [2] This is done with labelled data, which is leveraged to push representations of the same class together and representations of other classes apart. This procedure is done in order to warm-start

CS230: Deep Learning, Spring 2022, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

the parameters of another model used for downstream tasks, and has seen success in the fields of computer vision and NLP[7].

3 Dataset

3.1 Raw Data

The dataset we are using is the MUSDB18 [8] dataset. It consists of 150 full-length music tracks with an average track length of 236 seconds with a standard deviation of 95 seconds, totalling to around 10 hours of full/stereo music data. Moreover, the datasets are each split into 84 songs in the training set, 16 songs in the dev set, and 50 songs in the test set, and all songs are accompanied with their isolated "drums", "bass", "vocals", and "others" tracks. The .stem.mp4 files are encoded at 44.1Hz. For convenience purposes, this baseline model is currently trained on a version of MUSDB18 where each track is a 7-second preview.

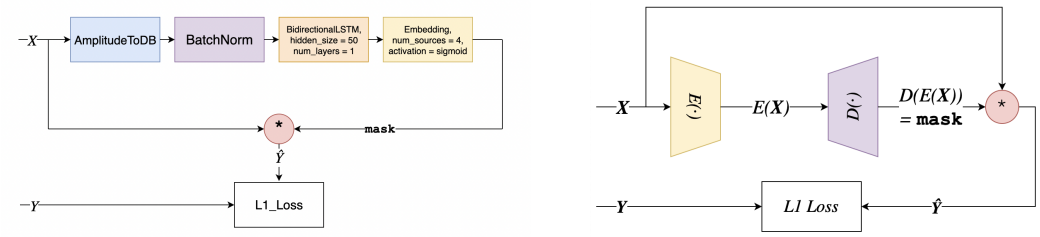
We used the audio source separation library `nussl`[9] and code from a music source separation tutorial[10] to convert each .stem.mp4 file to spectrograms for the mixture signals and the source signals. These signals were converted into spectrograms using Short-time Fourier Transform.

3.2 Data Augmentation

Since the 7-second version of the MUSSB18 dataset we were using didn't have an adequate amount of training data, we used data augmentation to generate new samples of data by engaging creating new coherent mixes of the original training data. These coherent mixes were generated by perturbing the instrumental source histograms of each piece slighty; the equivalent of 6,000 new training and 300 new validation examples were generated via this method. Each new mixture was 5 seconds in length.

4 Methods

We trained a baseline model, and a DeepDeMix model (described below) on spectrograms. We then performed supervised contrastive learning on a DeepDeMix encoder, which was then used to train another DeepDeMix model.



(a) Baseline model

(b) DeepDeMix autoencoder model

Figure 1: Model architectures explored

4.1 Baseline Model

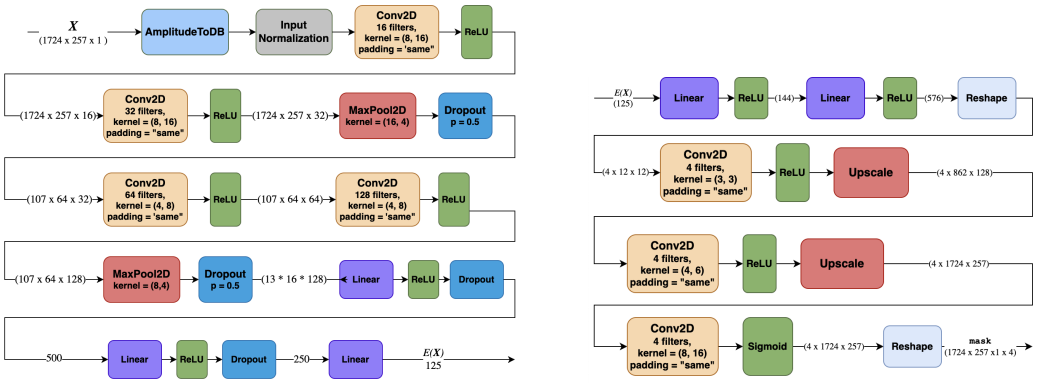
The attempt at a baseline model takes as input data audio spectrogram data in the shape of $(m, n_t, n_f, n_{ac}, s_{in})$, and outputs predictions in the shape of $(m, n_t, n_f, n_{ac}, s_{out})$, where m is the batch size, n_f is the number of frequency bins in the spectrogram, n_t is the number of time hops (time duration bins) in the histogram, n_{ac} is the number of audio channels, s_{in} is the number of sources that the model accepts for each training example, and s_{out} is the number of sources that the model outputs per training example. In this case, $s_{in} = 1$ because our model takes as input training examples where all of the instruments are merged into a single mixture (i.e. one source). Moreover, $s_out = 4$ because our model is attempting to reconstruct the signal sources from four different instruments, (i.e. four sources).

Our model implementation makes use of `nussl` modules (which are based in PyTorch) for source separation. The data goes through an `nussl` AmplitudeToDB module, which converts magnitude spectrograms to log amplitude spectrograms in decibels. Next, the data undergoes a batchnorm layer. After this, the data is then fed to a bidirecitonal LSTM with a hidden state size of 50 and 2 layers (num_layers set to one results in two LSTM layers when implementing a bidirectional LSTM). After this, the output of the bidirecitonal LSTM is fed to an `nussl` embedding layer, which in practice acts as a linear layer with sigmoid activations at the end, resulting in an output of size $(m, n_t, n_f, n_{ac}, s_{out}$. This output is a mask, which is then multiplied element-wise with the input to produce the output $\hat{Y}$, the model's attempted reconstructed the input. This is a broadcast matrix multiplication, producing a final model estimate of size $(m, n_t, n_f, n_{ac}, s_{out})$.

4.2 DeepDeMix Model

DeepDeMix (Figure 1b) follows an encoder-decoder arthicecture. The encoder takes data in the shape of (m, n_t, n_f, s_{in}), and outputs predictions in the shape of $(m, n_f, n_h, n_{ac}, s_{out})$. Data of shape $m, n_t, n_f, n_{ac}, n_c)$ is reshaped to (m, n_t, n_f, n_c) before being fed to DeepDeMix. This can be done because $s_{in} = 1$. This time, however, instead of using a bidirectional LSTM, the encoder uses 2 (Conv2D, ReLU, Conv2D, ReLU) blocks, followed by (MaxPool2D, Dropout) blocks. After this, the resulting representation is then reshaped into a vector which is then fed to two (Linear, ReLU, Dropout) blocks, and then finally fed to a single linear block to obtain a latent representation. See Figure 2 for more information.

Once this intermediate representation is obtained, it is then fed to the decoder (Figure 2b) to two (Linear, ReLU) blocks to get a 576-dimensional representation. This representation is then reshaped into a (4, 12, 12) shape, and fed to two (Conv2D, ReLU, Upscale) blocks. After this, the data is fed into a (Conv2D, Sigmoid) block to obtain a shape of (m, s_{out}, n_t, n_f), and then reshaped to $(m, n_t, n_f, n_c, s_{out})$. Mask multiplication takes place in the same way as the baseline model.



(a) Encoder model

(b) Decoder model

Figure 2: DeepDeMix encoder and decoder archictures

4.3 DeepDeMix Encoder Contrastive Pretraining

We also used a DeepDeMix model where the encoder portion was separately trained on a self-supervised contrastive learning task. We train the encoder on batches of (anchor, positive, negative) samples, where the anchor and positive samples are spectrograms of the same instrument in different songs, and the negative sample is the spectrogram of a different instrument in any of the songs. For this, we fed the encoder batches of data of size $(m, n_t, n_f, n_{ac}, 3)$. The last integer in the tuple denotes the number of sources that correspond to an anchor, positive, and negative example in a given batch. The outputs are in the shape $(m, 3, 125)$. The middle 3 in this tuple also corresponds to an anchor, positive, and negative example. The goal of this model is to get the vector representations of the anchor an positive example close to each other and the representations of the anchor and negative examples away from each other relative to some distance metric.

4.4 Loss

We used L1 Loss evaluated on the computed reconstructions of the instrument histrograms and ground truth instrument source spectograms in order to train the baseline model and DeepDeMix models. L1 loss has been found to outperform other loss metrics for magnitude spectrogram matching [11] and is a common loss function in the music source separation literature. For the pretraining task, we evaluated the encoder by using Triplet Margin Loss, defined for a single (anchor, positive, negative) pair as:

$$L(\mathbf{a}, \mathbf{p}, \mathbf{n}) = \max(||\mathbf{a} - \mathbf{p}||_l - ||\mathbf{a} - \mathbf{n}||_l + \alpha, 0),$$

where **a** is the vector representation of the anchor, **p** is the vector representation of a positive example, **n** is the representation of a negative example, l is a norm, and α is a margin hyperparameter.

5 Results

5.1 Evaluation Metrics

We evaluated the results of our baseline, DeepDeMix, and pretrained DeepDeMix using both Scale-to-Distortion Ratio (SDR) and Scale- Invariant Signal-to-Distortion ratio, or SI-SDR. SI-SDR is a modification of SDR (Signal-to-Distortion ratio), a measure of the log ratio of the volume of the source projection onto the ground truth, and the volume of what is orthogonal to this projection. SDR is calculated by:

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

where s_{target} is a version of the clean source signal modified by an allowed distortion, and e_{interf}, e_{noise}, and e_{artif} are errors due to interferences, noise, and artifacts, resepectively.[12]

SI-SDR is given by the following:

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

where $\hat{s}$ is the estimate signal and s is the target. This metric overcomes some critical failures of SDR; it cannot be tweaked artificially by just changing the scale of the estimation, making it robust to scaling.[13] For both SI-SDR and SDR, the higher the values, the better.

5.2 Hyperparameters

When converting the audio waveforms into spectrograms, we performed a Short-time Fourier Transform, with the number of time hops set to $n_t = 1724$, and number of frequency bins set to $n_f = 257$. We trained the baseline model for for 30 epochs with a batch size of 16. For the pretraining task, we trained the encoder for 40 epochs with a batch size of 16. The two DeepDeMix model variations were trained for 45 epochs each with batch sizes of 16. We used an Adam optimizer with a learning rate of 0.001 and β_1 of 0.9 and β_2 of 0.999. for all models. For the pretraining task, we used a norm of $l = 2$ and a margin $\alpha = 1.0$ for the triplet margin loss function.

5.3 Model Results

Experiments show that the baseline model achieves poor performance in music source separation on both the training dataset and testing dataset. However, there are improvements in the DeepDeMix model without pretraining. Moreover, there are improvements in the DeepDeMix model with pretraining in both SI-SDR and SDR metrics. We can see that pretrained DeepDeMix achieves a 0.44

		SDR					**SI-SDR**				
		Overall	*Bass*	*Drums*	*Other*	*Vocal*	*Overall*	*Bass*	*Drums*	*Other*	*Vocals*
Baseline	***Train***	-1.47	1.63	-2.12	-3.28	1.04	-1.88	-1.60	-3.05	-3.66	0.78
	Test	-2.01	-2.92	-1.24	-3.42	-0.41	-2.14	-3.00	-1.66	-3.87	-0.03
DDM	***Train***	1.23	1.53	1.48	1.72	2.93	1.03	1.34	1.38	1.58	-0.2
	Test	0.98	0.94	1.37	1.25	0.29	0.71	0.90	1.28	1.05	-0.37
DDM (pt)	***Train***	1.52	2.12	1.90	1.91	1.48	1.41	1.96	1.23	1.79	0.67
	Test	1.42	2.01	1.24	1.59	1.24	1.20	1.86	0.92	1.37	0.65

Table 1: SDR and SI-SDR metrics

SDR and 0.49 SI-SDR point increase when it is pretrained on MUSDB18 training data. However, these models are still a cry away from the state of the art results presented by models such as Hybrid Demucs [5], which achieves an average SDR of 7.68. Conv-TasNet achieves an average SDR of 5.73 [4].

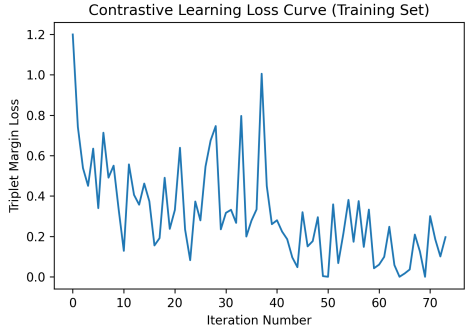


Figure 3: Loss curve for encoder pretraining task

Figure 3 shows the triplet margin loss curve for the pretraining task on the encoder on the training dataset. This, along with the score increase on the pretrained DeepDeMix network, serve as evidence that the encoder learns useful and rich representations for the spectrograms that serve as inputs.

6 Discussion

All of the models do worse on MUSDB than current state-of-the-art models. Looking at the results presented in the previous section, it is clear that the weakest model was the baseline, since it suffered from both bias and variance issues. It's highly likely that the model did not have the representational capacity to capture the data distribution of MUSDB18.

Although the results achieved by the experiments are underwhelming compared to the state of the art, these experiments validate the hypothesis that contrastive learning is a useful technique in the task of music source separation. This is evidenced by the loss curve that the encoder achieved in the pretraining task, along with the improved performance that DeepDeMix achieved after pretraining the encoder.

7 Next Steps

From the experiments it has become evident that contrastive pretraining improves model performance on music source separation. Moving forward, our goal becomes to increase the scores that were achieved in the experiments. We propose the use of the pretraining task on more sophisticated model architectures such as Hybrid Demucs or Conv-TasNet. We also suggest pretraining on additional music source separations datasets in order to obtain higher quality embeddings. Additional model architecture experimentation is also highly recommended. Finally, we suggest the use of different contrastive pretraining regimes which make use of multiple postive and negative exmaples per anchor, such as those explored in [2].

8 Project Code

The code for this project can be found in https://github.com/ivillar/DeepDeMix.

9 Contributions

Ivan Villa-Renteria is the sole person working on this project. We would like to acknowledge Elaine Sui, our assigned mentor, for giving us guidance throughout the duration of the project, especially for suggesting strategies to incorporate contrastive pretraining.

References

[1] Alexandru Mocanu. Musical source separation. Technical report, 2020.

[2] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2020.

[3] 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)*, pages 106–110, 2018.

[4] Yi Luo and Nima Mesgarani. Conv-TasNet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(8):1256–1266, aug 2019.

[5] Alexandre Défossez. Hybrid spectrogram and waveform source separation, 2021.

[6] Minseok Kim, Woosung Choi, Jaehwa Chung, Daewon Lee, and Soonyoung Jung. Kuielab-mdx-net: A two-stream neural network for music demixing, 2021.

[7] Nils Rethmeier and Isabelle Augenstein. A primer on contrastive pretraining in language processing: Methods, lessons learned and perspectives, 2021.

[8] Zafar Rafii, Antoine Liutkus, Fabian-Robert Stöter, Stylianos Ioannis Mimilakis, and Rachel Bittner. The MUSDB18 corpus for music separation, December 2017.

[9] Ethan Manilow, Prem Seetharaman, and Bryan Pardo. The northwestern university source separation library. Proceedings of the 19th International Society of Music Information Retrieval Conference (ISMIR 2018), Paris, France, September 23-27, 2018.

[10] Ethan Manilow, Prem Seetharman, and Justin Salamon. *Open Source Tools & Data for Music Source Separation*. https://source-separation.github.io/tutorial, 2020.

[11] Enric Gusó. *On Loss Functions for Music Source Separation*. PhD thesis, Universitat Pompeu Fabra, August 2020.

[12] E. Vincent, R. Gribonval, and C. Fevotte. Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1462–1469, 2006.

[13] Jonathan Le Roux, Scott Wisdom, Hakan Erdogan, and John R. Hershey. SDR - half-baked or well done? *CoRR*, abs/1811.02508, 2018.