CS230

TimbreNet: A Convolutional Network for Blind Audio Source Separation

Scott Reid
Department of Electrical Engineering
Stanford University
shrcor@stanford.edu

Nathaniel Okun
Department of Computer Science
Stanford University
nokun@stanford.edu

Abstract

We aimed to separate the instruments in multi-instrument audio tracks, a process commonly known as audio source separation. Mixed musical sources are difficult to analyze, transcribe and sample. Background noise, distortion and the tendency for harmonics to mix together complicate the task for even professional musicians. Improvements made in this domain can be applied more generally to dozens of important problems from dialogue transcription to seismic monitoring[3]. We developed a novel convolutional architecture that embeds musical information into a "timbral" dimension–thereby leveraging information about the different sound qualities present in the mixture audio to separate individual instruments. One key challenge in data-driven musical source separation is the scarcity of publicly available mixture audio with separated instrument audio tracks. We demonstrate that this challenge can be overcome with data augmentation. With less than 5 minutes of training data, we were able to generate effectively infinite synthetic training data by permuting one second frames of different instruments. Our model extracted instruments from test audio with a source to distortion ratio of 8.16dB and excellent subjective audio quality (as evaluated by both authors, each with over 10 years of musical experience).

Our code is hosted at https://github.com/natoparkway/AudioSourceSeparation.

1 Introduction and Related Work

Deep learning has only recently been applied to source separation. Previous techniques relied on assumptions such as the tendency of vocal parts to be harmonic and have a fundamental frequency[5]. By tracking the pitch of the vocal track, algorithms could estimate the harmonics and reconstruct the entire separated audio. Matrix factorization methods such as Non-Negative Matrix Factorization and Independent Component Analysis have both been applied to the audio source separation problem, as well as the related problem of music transcription[9], with limited success on audio tracks with multiple instruments.

Other source extraction methods rely on common recording techniques, such as the fact that vocals frequently have approximately the same energy in the left and right channels of stereophonic (commonly referred to as “stereo”) recordings while instruments are mostly mixed on one channel[6]. We focused on the monoaural audio source separation problem because it is the most challenging manifestation of the audio source separation problem.

Huang et al. presented an early application of deep learning to source separation in 2014. They utilized recurrent neural networks to achieve a source to distortion ratio of around 7dB when extracting

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

a single voice from noisy audio. Jannson et al. utilized a series of convolutional and deconvolutional layers on the same task. Both converted audio to the time-frequency dimension, a process we detail in section 3.1, before inputting them to their networks.

A key challenge in data-driven source-separation methods is the scarcity of large datasets of recorded audio and its component instruments. One of the largest datasets, MUSDB150, consists of only 150 songs[2]. Uhlich et al. tackled the problem of scarce data by generating training data through random combinations of solo instrument performances. We utilize a similar data generation method in our work to great effect.

2 Dataset and Features

We performed most of our experiments on the Bach10 dataset prepared by Duan et al. The dataset consists of 10 J.S Bach chorales performed by four different instruments - a saxophone, bassoon, clarinet and violin. Each chorale is roughly 30 seconds and the dataset has a cumulative length of 5 minutes and 34 seconds. Each instrument was recorded in isolation while the musician listened to the recordings of others through a headphone. The recordings are monoaural (commonly referred to as "mono") and have a sampling rate of 44100 kHz. Our main focus for this project was to develop an architecture and cost function for source-separation. This small dataset was well-suited for fast iteration through different ideas.

We found that 5 minutes of audio was insufficient to perform well on the task. By splitting data into frames and permuting all possible combinations of instruments, we were able to increase our effective training size by many orders of magnitude. This significantly improved performance and solves the problem of scarce data that hinders many data-driven models[5]. Notably, we found our model can perform well on harmonically consistent input even after being trained on cacophonous data.

3 Methods

3.1 Short-Time Fourier Transform

We used the discrete Short-Time Fourier Transform (STFT) to convert audio files into spectrograms, thus representing musical information in the time-frequency domain. The STFT of a signal, $\mathbf{S} \in \mathbb{C}^{T \times F}$, is a matrix with complex entries. Here, T is the number of time-windows in the STFT and F, the number of frequencies in the spectrogram, is a parameter that can be specified. $\mathbf{S}$ can be decomposed into real-valued magnitude and phase spectrograms as $\mathbf{S} = \bar{\mathbf{S}} \star e^{i\mathbf{\Phi}}$, where $\star$ represents element-wise multiplication and the exponential is performed element-wise. STFTs can be inverted with the Inverse STFT transform (ISTFT). The complex phase of the STFT entries is crucial for the inverse transformation, but are not essential for audio source separation. Spectrogram representations of audio are useful for source separation because individual sources are easier to distinguish in the time-frequency domain than in the original waveform.

3.2 Network Input and Output

The input to our network is the magnitude spectrogram $\bar{\mathbf{S}}$ for a given audio sample. The phase spectrogram is not used as an input to the network, although in future work the network may be generalized to leverage complex phase information. We reshape the magnitude spectrogram into a 3D array of shape $1 \times T \times F$. The first dimension of this array indexes the *timbral space* or instrument space, and is expanded through upsampling to separate the single input spectrogram into spectrograms for each instrument in the mixture audio. The second and third dimensions index time and frequency, respectively.

The output of the network is a set of N spectral filters, contained in an array of shape $N \times T \times F$. Each of the N filters is an array of shape $T \times F$, the same shape as the mixture spectrogram. The spectral filters have entries that are bounded by 0 and 1 (Fig. 2). The spectral filters are applied to the mixture spectrogram through element-wise multiplication, yielding the network's prediction of instrument spectrograms for each instrument in the mixture. For example, letting $\mathbf{\Gamma}^{\langle i \rangle}$ be the spectral filter for the i^{th} instrument, the predicted complex instrument spectrogram for instrument i is given by $\hat{\mathbf{I}^{\langle i \rangle}} = \mathbf{\Gamma}^{\langle i \rangle} \star \mathbf{S}$. Our cost function is evaluated on the predicted instrument magnitude spectrograms

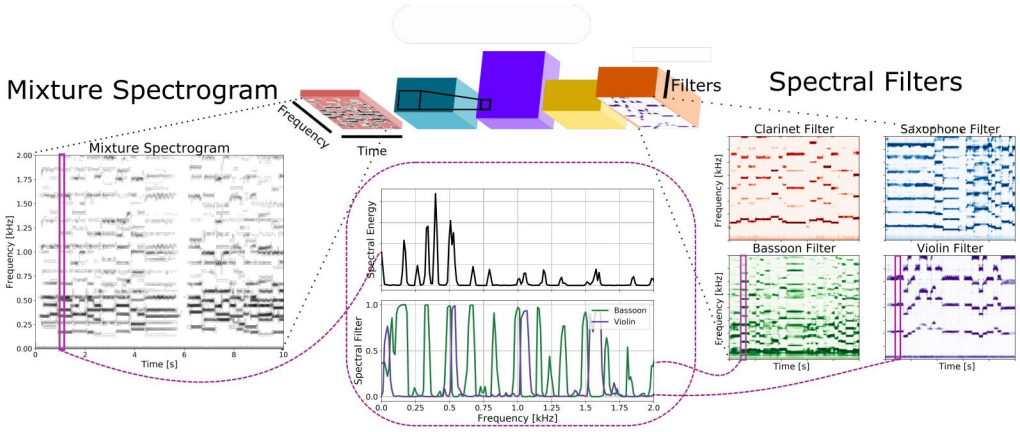

Figure 1: **Network Input and Output**
The mixture magnitude spectrogram (*right*) is fed into TimbreNet and, through upsampling and 2D convolutions (Fig. 2), the network produces spectral filters (*right*) for each instrument in the mixture. At each time-step, the spectral filters assign each spectral peak (*middle, top*) to an instrument (*middle, bottom*). To accurately recover the timbre of an instrument, all of the harmonics must captured by the spectral filters–thus, the filters resemble frequency combs.

$\hat{\bar{\mathbf{I}}}^{\langle i \rangle}$; thus, when evaluating the cost function, the spectral filters are applied to the mixture magnitude spectrogram $\bar{\mathbf{S}}$. However, in order to use the ISTFT, we need complex instrument spectrograms. Thus, when recovering instrument audio, the spectral filters are applied to the complex mixture spectrogram $\mathbf{S}$.

3.3 Basic Loss Function

In cases where a consistent ordering of instrument can be chosen, it is sufficient to use a basic squared loss function on instrument magnitude spectrograms. In training examples, the audio from each individual source is known – thus, we can compute the magnitude spectrograms of each instrument using the STFT. Training examples are represented in arrays $\mathbf{Y} \in \mathbb{R}^{N \times T \times F}$, where $\mathbf{Y}_i = \bar{\mathbf{I}}^{\langle i \rangle}$ is the magnitude spectrogram of the i^{th} instrument in the training example. The spectral filters $\mathbf{\Gamma}^{\langle 1 \rangle} \dots \mathbf{\Gamma}^{\langle N \rangle}$ are used to compute predicted instrument magnitude spectrograms $\hat{\mathbf{Y}} \in \mathbb{R}^{N \times T \times F}$ through element-wise multiplication with the mixture magnitude spectrogram $\bar{\mathbf{S}}$. The basic loss is given by the squared frobenius norm of the difference:

$$\mathcal{L} = ||\mathbf{Y} - \hat{\mathbf{Y}}||_{\mathrm{F}}^2 \tag{1}$$

We note that the loss function is defined on predicted magnitude spectrograms, and thus does not explicitly depend on the spectral filters $\mathbf{\Gamma}^{\langle i \rangle}$. The predicted magnitude spectrograms depend on the filters through the element-wise product. Thus, we consider our model to be end-to-end insofar as it learns the best spectral filters such that the predicted instrument spectrograms are as close to ground truth as possible.

3.4 Unordered Loss Function

When the network is trained, the ordering of instrument filters at the network output may not match the ordering of instruments in training examples. This problem arises in the case of music that combines any of hundreds of possible instruments, and where a consistent ordering of instruments can not be chosen. To solve this problem, we developed an *unordered* loss function that returns the squared error from the best pairing of output and training example instrument spectrograms.

We first define the pairwise loss vector $\mathbf{p} \in \mathbb{R}^{N^2}$, where N is the number of instruments. Each component of $\mathbf{p}$ is the squared difference between a predicted instrument magnitude spectrogram

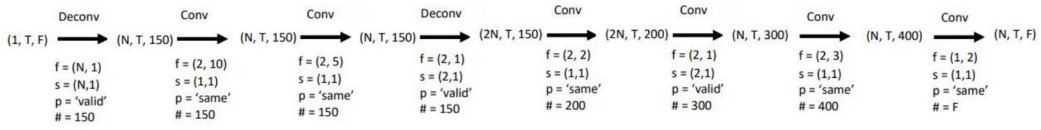

Figure 2: **TimbreNet Architecture**
Deconvolutional layers are used to upsample from one mixture spectrogram to N instrument spectral filters. 2D convolutions are taken in the timbral-temporal domain, while frequency remains a channel dimension. All layers use ReLu activations except for the output layer, which uses sigmoid activations to ensure that the entries of the spectral filters are bounded between 0 and 1. f is the kernel shape, s is the stride, p is the padding, and # is the number of channels for each convolutional layer.

$\hat{\bar{\mathbf{I}}}^{\langle j \rangle}$ and a ground truth instrument magnitude spectrogram $\bar{\mathbf{I}}^{\langle k \rangle}$.

$$\mathbf{p} = \left[||\hat{\bar{\mathbf{I}}}^{\langle 1 \rangle} - \bar{\mathbf{I}}^{\langle 1 \rangle}||_{\mathrm{F}}^2 \quad ||\hat{\bar{\mathbf{I}}}^{\langle 1 \rangle} - \bar{\mathbf{I}}^{\langle 2 \rangle}||_{\mathrm{F}}^2 \quad \dots \quad ||\hat{\bar{\mathbf{I}}}^{\langle 1 \rangle} - \bar{\mathbf{I}}^{\langle N \rangle}||_{\mathrm{F}}^2 \quad \dots \quad ||\hat{\bar{\mathbf{I}}}^{\langle N \rangle} - \bar{\mathbf{I}}^{\langle N \rangle}||_{\mathrm{F}}^2 \right]^\top \tag{2}$$

Next, we define matrix $\mathbf{A} \in \mathbb{R}^{N! \times N^2}$ such that the product $\mathbf{Ap}$ yields a vector of all of the possible squared losses for each of the $N!$ unique pairings of the output spectrograms with the target spectrograms. Each of $\mathbf{A}$'s rows is a transposed stacked vector of a distinct permutation of the standard basis vectors for $\mathbb{R}^N$. For example, in the case of $N = 2$,

$$\mathbf{A} = \begin{bmatrix} \mathbf{e}_1^\top & \mathbf{e}_2^\top \\ \mathbf{e}_2^\top & \mathbf{e}_1^\top \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{3}$$

Our loss is the minimal element of $\mathbf{Ap}$. We use `LogSumExp`, a smooth and differentiable approximation to `max`, to evaluate the minimum. Thus, our loss function is:

$$\mathcal{L} = -\text{LogSumExp}(-\mathbf{Ap}) \tag{4}$$

4 Experiments, Results, and Discussion

We evaluated our model by calculating the extracted sources' similarity to ground truth as represented by the source to distortion ratio. The evaluation, as implemented by Stöter et al. and designed by Vincent et al., estimates the predicted source as the sum of the target source, noise, interference and other artifacts using orthogonal projections of the predicted source.

$$s_{\text{predicted}} \approx s_{\text{target}} + e_{\text{interference}} + e_{\text{noise}} + e_{\text{artifact}} \tag{5}$$

The source to distortion ration (SDR) is given by:

$$\text{SDR} = 10 \log_{10} \left(\frac{||s_{\text{target}}||^2}{||e_{\text{interference}} + e_{\text{noise}} + e_{\text{artifact}}||^2} \right) \tag{6}$$

A source distortion ratio of 3dB implies that the extracted audio is 10^3 times more similar to the target audio than to noise. We achieved an average source to distortion ratio over all instruments in our test set of 8.16dB after tuning hyperparameters. This results in professional sounding extracted audio quality. See *Table C* for a comparison of subjective audio quality and SDR.

We focused on tuning three hyperparameters - length of the frame used for each training example, the number of frequency bins in the spectrogram used as input and the probability that we omitted an instrument when creating a training example. We kept the number of training epochs (60), batch size (16), number of training examples (6000) and instrument order ('clarinet', 'bassoon', 'saxophone', 'violin') constant. We kept epochs low to avoid overfitting. We believe we succeeded in this regard – the model performs consistently well on all the songs in our test set and has a validation loss similar to training loss.

We found that increasing frame size and the number of frequency bins up to a certain point improves performance. We found that omitting instruments with some probability when creating training

A

	0.22	0.9	1.8
501	$4.96dB$	0.30	5.44
1001	6.47	7.36	7.41
2001	5.86	7.85	**8.16**
4001	5.86	7.66	7.80

B

	0%	10%
501	$0.30dB$	5.21
1001	7.36	7.10
2001	**7.85**	7.19
4001	7.66	7.54

C

	ReidEval	*OkunEval*
0.30	1	1
5.20	2	3
7.10	4	4
8.16	**5**	**5**

Figure 3: Tables **A** and **B** show the test SDR for different hyperparameter values. In Table **A**, the vertical axis is the number of STFT frequencies and the horizontal axis is the training example duration (in seconds). In Table **B**, the vertical axis is again the number of STFT frequences and the horizontal axis is the instrument dropout probability. Table **C** gives the authors' subjective measure of audio quality for different SDR values. Audio quality is rated on a scale of 1 - 5, with 1 meaning "bad," 3 meaning "fair," and 5 meaning "excellent."

examples (e.g creating a training example with a clarinet, saxophone and bassoon but no violin) worsens performance. There was one notable exception to both these rules – increasing frame size to 0.9 seconds from 0.22 seconds decreases SDR by over 4dB when we encode audio in the time-frequency domain with 501 frequency bins. We hypothesize this is because our learning function became stuck at a local minimum for this exact combination of hyperparameters. Tellingly, changing the probability of dropping an instrument while creating a training example resulted in 4 decibels better performance. Since increasing this probability generally worsened performance elsewhere, we hypothesize that it provided enough change to break out of a local minimum.

We performed a preliminary experiment with the unsorted loss function, and at this time only have qualitative results to report. In our experiment, we reduced the number of output instruments N to 3. We produced 1 second long training examples consisting of random combinations of 3 of the 4 instruments. After training, we found that the model could separate audio from 30 second-long test examples with 3 instruments with good subjective audio quality. However, we observed a *channel-switching* phenomenon between the separated clarinet and saxophone audio. Every few seconds, the clarinet audio would switch to saxophone. At exactly the same time, the saxophone audio would switch to clarinet. Bassoon and violin audio did not exhibit this channel-switching phenonenon, likely because the timbre of the bassoon and violin are much more distinct than saxophone and clarinet. Training on longer examples will almost certainly reduce the frequency of channel-switching.

5 Conclusion and Future Work

To conclude, our model extracts clarinet, saxophone, bassoon, and violin audio from monophonic mixture audio with exceptionally high fidelity. We ascribe this performance to our convolutional architecture with timbral embedding, and to our cacaphonous data augmentation techniques. Our novel unordered cost function will be crucuial when we expand to more instruments, as a consistent ordering of instruments can not be chosen. Preliminary results show that this cost function is effective, but that it will be neccessary to train on longer duration training examples.

More work needs to be done to interpret the network and learn how it uses upsampled timbral dimension. An interpretation of how this network works will yield important intuition about how to develop improved convolutional architectures for audio source separation. In the future, we hope to use a 3D convolutional network which performs convolutions over the whole timbral-temporal-frequency domain to increase parameter-sharing. Incorporating a recurrent layer, such as an LSTM, in our network may help to leverage musical structure and repeated elements to aid in source separation.

Performance may be further improved by using more input infomration. The mixture phase spectrogram may yield useful information about how to separate audio, particularly when harmonics from two instruments lie on top of each other. As previously noted, instruments are frequently mixed across left and right channels in stereophonic music in ways that make source extraction much easier[6]. Rafii et al. detail dozens of models which rely heavily on the existence of two channels to extract different sources. We are confident that using this information will improve extraction quality.

6 Contributions

Both team members worked equally on this project. Scott Reid spent relatively more time on model and loss function design; Nathaniel Okun spent relatively more time on infrastructure and hyperparamter tuning.

References

[1] Zhiyao Duan and Bryan Pardo, “Soundprism: an online system for score-informed source separation of music audio,” IEEE Journal of Selected Topics in Signal Process., vol. 5, no. 6, pp. 1205-1215, 2011.

[2] F.R Stöter, A. Liutkus and N. Ito, "The 2018 signal separation evaluation campaign," International Conference on Latent Variable Analysis and Signal Separation., 2018

[3] J.J Sallas, D. Corrigan and K.P Allen (1998). United States Patent no. 5721710

[4]. Emmanuel Vincent, Rémi Gribonval, Cédric Févotte. Performance measurement in blind audio source separation. IEEE Transactions on Audio, Speech and Language Processing, Institute of Electrical and Electronics Engineers, 2006, 14 (4), pp.1462–1469. <inria-00544230>

[5] Z. Rafii, A. Liutkus, F. R. Stöter, S. I. Mimilakis, D. FitzGerald and B. Pardo (2018). "An Overview of Lead and Accompaniment Separation in Music." IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2018, 26, (8), pp.1307-1335 <10.1109/TASLP.2018.2825440>

[6] C. Avendano, “Frequency-domain source identification and manipulation in stereo mixes for enhancement, suppression and re-panning applications,” in IEEE workshop on applications of signal processing to audio and acoustics, 2003.

[7] Chollet, François and others (2015). Keras [Computer software]. Available from http://www.keras.io

[8] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, TensorFlow: a system for large-scale machine learning, Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation, November 02-04, 2016, Savannah, GA, USA

[9] Smaragdis, Paris, and Judith C. Brown. "Non-negative matrix factorization for polyphonic music transcription." IEEE workshop on applications of signal processing to audio and acoustics. Vol. 3. No. 3. 2003.

[10] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, “Deep learning for monaural speech separation,” in IEEE international conference on acoustics, speech and signal processing, 2014.

[11] S. Uhlich, F. Giron, and Y. Mitsufuji, “Deep neural network based instrument extraction from music,” in IEEE international conference on acoustics, speech and signal processing, 2015.

[12] A. Jansson, E. Humphrey, N. Montecchio, R. Bittner, A. Kumar, and T. Weyde, “Singing voice separation with deep U-Net convolutional networks,” in 18th international society for music information retrieval conferenceng, 2017.