Remixing musical audio on the web using source separation

Gerard Roma
Centre for Vision, Speech and Signal Processing
University of Surrey
g.roma@surrey.ac.uk

Andrew J.R. Simpson
Centre for Vision, Speech and Signal Processing
University of Surrey
andrew.simpson@surrey.ac.uk

Emad M. Grais
Centre for Vision, Speech and Signal Processing
University of Surrey
grais@surrey.ac.uk

Mark D. Plumbley
Centre for Vision, Speech and Signal Processing
University of Surrey
m.plumbley@surrey.ac.uk

ABSTRACT

Research in audio source separation has progressed a long way, producing systems that are able to approximate the component signals of sound mixtures. In recent years, many efforts have focused on learning time-frequency masks that can be used to filter a monophonic signal in the frequency domain. Using current web audio technologies, time-frequency masking can be implemented in a web browser in real time. This allows applying source separation techniques to arbitrary audio streams, such as internet radios, depending on cross-domain security configurations. While producing good quality separated audio from monophonic music mixtures is still challenging, current methods can be applied to remixing scenarios, where part of the signal is emphasized or deemphasized. This paper describes a system for remixing musical audio on the web by applying time-frequency masks estimated using deep neural networks. Our example prototype, implemented in client-side Javascript, provides reasonable quality results for small modifications.

1. INTRODUCTION

Audio source separation refers to a set of signal processing and machine learning techniques that allow estimating the different component signals in a mixture. The difficulty of this task is strongly related to the number of recordings (i.e. from different microphones) that are available for a given mixture. Thus, for several years research focused on linear techniques such as Independent Component Analysis (ICA) [1] algorithms that could exploit the availability of multiple channels. However, for applications to musical audio, it is desirable to obtain algorithms that can operate on single-channel (monophonic) or stereo signals, which is much more challenging. In recent years several efforts have allowed to improve results on this front, mainly by transitioning from ”blind” to ”guided” source separation [7], for example by using score information and Non-negative Matrix Factorization (NMF) algorithms [2]. Deep Neural Networks (DNN) are increasingly being used in a supervised setting to learn how to separate different sources using a set of training examples where the reference sources are available. A good summary of the state of the art is provided by the SiSEC source separation evaluation campaign [5]. In the last edition, the evaluation for the music task shifted from allowing algorithms and parameter sets tailored for specific songs, to a larger scale evaluation where the same algorithms and set of parameters are tested on 50 songs, with 50 songs available for training. DNN-based approaches tended to perform better than traditional NMF-based approaches.

While the results of current research in source separation of musical audio (as shown in the SiSEC results) are still far from being able to isolate different instruments of an arbitrary recording with good quality, it is easy to think of applications that could use some degree of separation. One example is being able to remix audio to the listener’s taste, not unlike using an equalizer. Another example is creative remixing for DJing and music creation. In this paper we explore the application of audio separation based on DNNs for remixing musical audio in a web browser. Such application is possible with current generally available technologies, although some additions to the Web Audio API (such as generic FFT) would be desirable. We describe an implementation that allows using state-of-the-art source separation techniques in a browser that supports this API.

In the next section we summarize the framework for training DNNs to perform audio source separation. We then describe an implementation using web audio and discuss our initial results.

Web Audio Conference WAC-2016, April 4–6, 2016, Atlanta, USA

2. AUDIO SOURCE SEPARATION USING TIME-FREQUENCY MASKS

2.1 Time-frequency masks

Many supervised source separation algorithms work by estimating a time-frequency mask that is used to filter the mixture in the frequency domain. Such masks can be either binary or real-valued. An ideal mask is computed for training data where the sources are available, and the model is trained to learn the mapping from a mixture to the ideal mask. The most common time-frequency representation is the Short-Time Fourier Transform (STFT). The model is then used to produce mask estimates for unseen data. We represent signals as time-frequency matrices with indices t, f. Let X be a mixture of two signals: a target signal S and an unwanted signal U. An Ideal Binary Mask (IBM) can be simply defined as

$$B_{t,f} = \begin{cases} 1 & \text{if } |S_{t,f}|^2 > |U_{t,f}|^2 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $|S_{t,f}|$ denotes the magnitude of the complex spectrogram S. These masks rely on the assumption that each time-frequency bin is perceived to belong to either S or U but not to both at the same time. The element-wise multiplication $B \odot X$ usually renders a good approximation of S. Such masks are commonly used as the objective in machine learning algorithms for audio source separation [4]. The result is an estimate $\hat{B}$ of the mask that allows obtaining an estimate of the target and unwanted signals as

$$\hat{S} = \hat{B} \odot X \tag{2}$$

$$\hat{U} = |1 - \hat{B}| \odot X \tag{3}$$

While they usually produce good results in the ideal case, approximations of binary masks can quickly degrade audio quality. For applications requiring reasonable quality, ratio masks are usually preferred. Ideal Ratio Masks (IRM) for both the desired source and the background can be computed as

$$R_{S\,t,f} = \frac{|S_{t,f}|^p}{|S_{t,f}|^p + |U_{t,f}|^p} \tag{4}$$

$$R_{U\,t,f} = \frac{|U_{t,f}|^p}{|S_{t,f}|^p + |U_{t,f}|^p} \tag{5}$$

where the exponent p can take different values [3]. For $p = 2$, R can be seen as a Wiener filter [8]. Since the mask is real-valued, the element-wise multiplication $R \odot X$ usually results in better sound quality than binary masks, at the expense of separation strength.

2.2 Estimation with deep neural networks

The system described in this paper is based on our previous work on separation of vocals [6]. In this previous work, a binary mask for separating vocals from background music was estimated by training a DNN with 20 consecutive stacked spectral frames per frame. Here, we use a similar approach downscaled to one frame for real-time operation on a browser. However, in this case we train a different model for each instrument, which allows combining the outputs from all models in the prediction of the mask. The training data comes from the dataset used in the SiSEC evaluation campaign [5]. In this collection, 100 songs are provided with separate stems and consistently labelled as “vocals”, “bass”, “drums”, and “other”. The proposed approach is to train one DNN for the first three of these categories, where the mask is computed by comparing the spectrogram of the target source to that of the sum of the remaining signals. This has the advantage that arbitrary instrument models could be added, at the cost of one additional network prediction in real-time operation. Audio files are analyzed using 20ms windows and 10ms overlap, which results in frames of 512 spectral bins. For each instrument, a DNN with two fully-connected hidden layers is trained using back-propagation. All layers are the size of one spectral frame and use sigmoid activation. The final layer can be described as a *regression layer* that tries to minimize the difference between the output an the mask. While our framework supports both binary and soft masks, the later are generally preferable for good quality remixes. Unlike the experiments in [6], the models are trained using the *adadelta* algorithm [9], which produces a sharper decrease of the cost function than plain Stochastic Gradient Descent (SGD).

2.3 Combining models

In order to remix monophonic audio, it is analyzed using the same FFT parameters and fed into each of the models. Spectral frames are processed by a forward pass of the network to predict a ratio mask. The value of the mask can be interpreted as a probability that the time-frequency bin belongs to the instrument learned by the model. A user-settable threshold parameter θ is used to control the minimum required probability. In addition, a set of user-settable parameters, $g_{1..n}$, is used to change the relative volume of each instrument. This value is used for the mask for the bins that are estimated to belong to that instrument. Thus, each time-frequency bin of the incoming audio is multiplied by g_i where i is the index of the max with the highest prediction for that bin, but only if this prediction is above θ.

This could be seen as a sort of ”mixture of experts” model. In order to save cpu cycles, some of the models can be deactivated. In that case their prediction is not used. Figure 2 shows block diagrams for the training and remix stages.

3. IMPLEMENTATION

We have implemented the approach described above in a web prototype (Figure 3). Models are trained using Convnetjs[1]. This framework provides the basics for the most common types of DNNs, plus a set of convenient training algorithms, to the extent that it can be treated almost like a black box. It also provides serialization of the network parameters to a JSON file, so that they can be later loaded in the browser. Our STFT module uses the FFT implementation in dsp.js[2]. A simple windowing procedure is used for analysis during the training stage, while a more complicated buffering strategy is required for real-time overlap-add.

In order to test the proposed approach, we built a simple prototype with a mixer style interface. It can be seen as a ”two-way mixer”. The interface allows loading several example songs which are then processed using the Web Audio

[1]http://cs.stanford.edu/people/karpathy/convnetjs
[2]https://github.com/corbanbrook/dsp.js

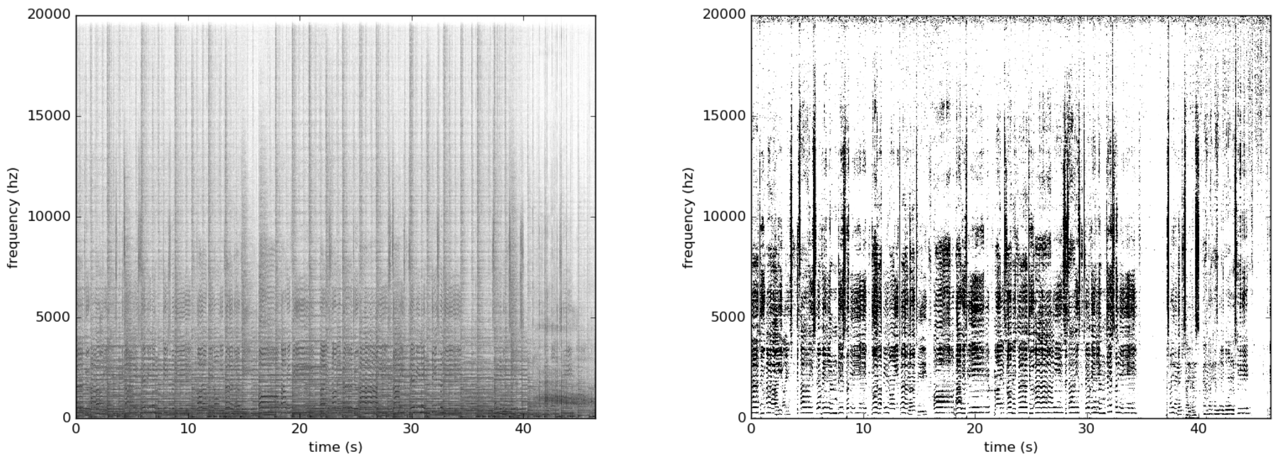
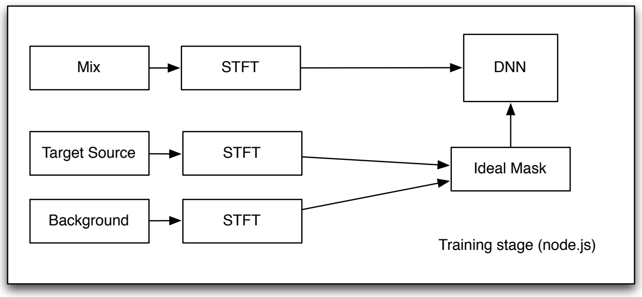


Figure 1: Example of spectrogram and estimated ratio mask



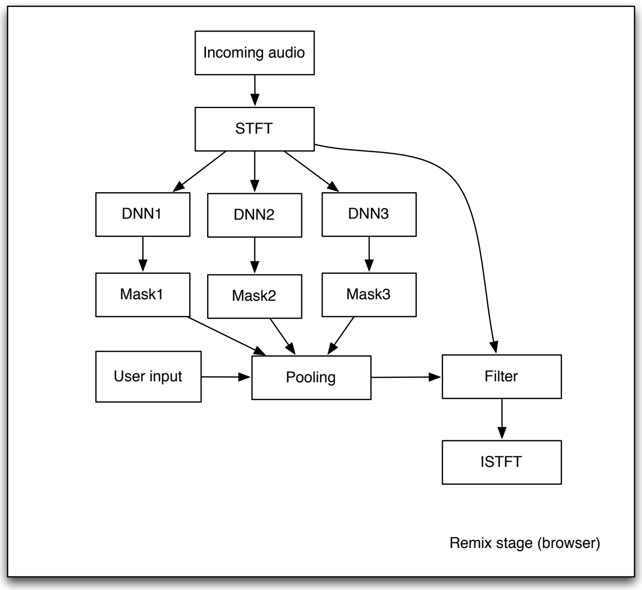


Figure 2: Block diagram of the training and testing phases.

API[3]. Clearly, one advantage of implementing this functionality in a web browser is that it facilitates the use of arbitrary audio available on the internet. In practice, though, this possibility is limited by security restrictions. In order for audio processing to work, the provider of audio content must support CORS headers[4]. We hope that in a foreseeable future significant amounts of music released under permissive licenses (e.g. Creative Commons) will be served as to allow creative modification in web applications.

For the moment, our prototype uses examples hosted at the same server that hosts the code. The audio is loaded into an *AudioBufferSourceNode*. This node is connected to a *ScriptProcessorNode* that computes the STFT of incoming audio and feeds it through the DNN models previously loaded at start-up. The models predict the mask for each instrument, which is scaled for each frame. Models can be switched on or off in order to save cpu cycles. The pooling process can be seen as a classification step, where each time-frequency bin is classified into one of the active models, if the mask prediction is higher than the value of the hreshold slider. Each bin is then multiplied in complex domain by the corresponding slider value in order to change the relative loudness of that instrument. This is done in an overlap-add setting, where at least two windows are processed for each *ScriptProcessorNode* buffer, and their inverse transforms are added. A larger buffer can be used to process more windows each time to reduce the cpu load at the expense of latency.

4. DISCUSSION

In the current implementation, three models can be handled in the browser with a current desktop CPU. When *AudioWorkers* are implemented in major browsers, it should be possible to make better use of multiple cores. Also, our project would benefit from a native FFT (including phase) or a frequency-domain *ScriptProcessorNode* . However, the main computational cost is due to the DNN prediction. Informal experiments with the prototype showed

[3]http://webaudio.github.io/web-audio-api/
[4]https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

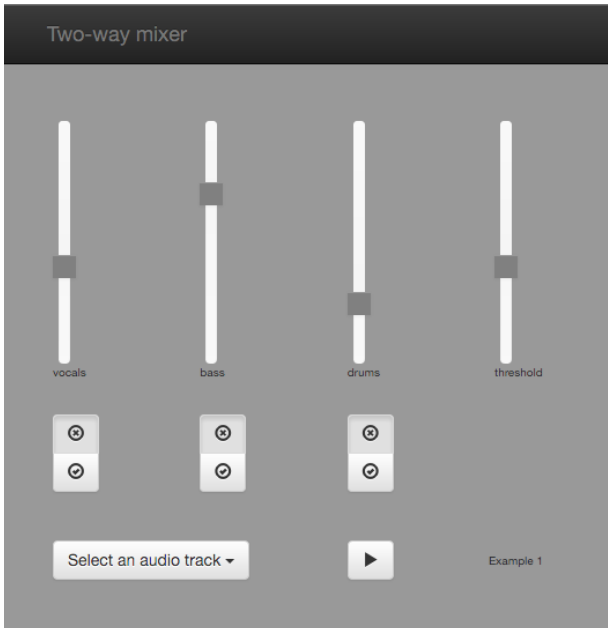


Figure 3: Screenshot of the prototype

that it is easy to introduce audible artifacts by emphasizing or de-emphasizing specific time-frequency bins. This can be controlled by the threshold slider. For smaller values of this slider, more artifacts may be introduced due to the fast switching of the mask. For higher values, separation is more noticeable and artifacts are reduced, although unwanted spectral filtering may be introduced when the models prediction is wrong (i.e. confusions between instruments). At very high values, the effect of separation vanishes because all predictions fall below the threshold. The artifacts depend also on the amount of imbalance between the sources defined by the mixing sliders. The system works reasonably well for small modifications, although it is not yet possible to do complete isolation or muting of the component signals. Small variations in the relative loudness of each instrument can be used as a sort of smart equalizer.

5. CONCLUSIONS

In this paper we have described a system for audio remixing based on source separation that can be run from end to end in Javascript, thanks to the availability of libraries like Convnetjs and dsp.js, and the web audio API. This allows making current methods used in audio signal processing research available to a wider audience. While separation is not perfect, application to remix and creative applications is already possible. In addition, our framework provides an interactive playground that allows testing different configurations and parameters, facilitating qualitative evaluation of different models, although limited by the possibilities of real-time processing on a web browser. While our models are trained offline, the prototype does not require a custom server to run. As shown in the Convnetjs site, the DNNs can even be trained in the browser, although this would require loading a significant amount of training data. We plan to improve on this work by adding more kinds of sounds, which will make it useful for a wider range of music. The prototype currently hosted at http://cvssp.org/projects/maruss/twowaymixer/. The code is available on github: https://github.com/g-roma/twowaymixer.

6. ACKNOWLEDGEMENT

This work is supported by grant EP/L027119/1 from the UK Engineering and Physical Sciences Research Council (EPSRC).

7. REFERENCES

[1] P. Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

[2] S. Ewert and M. Müller. Using score-informed constraints for nmf-based source separation. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 129–132. IEEE, 2012.

[3] E. M. Grais and H. Erdogan. Single channel speech music separation using nonnegative matrix factorization and spectral masks. In *Digital Signal Processing (DSP), 2011 17th International Conference on*, pages 1–6. IEEE, 2011.

[4] Y. Li and D. Wang. On the optimality of ideal binary time–frequency masks. *Speech Communication*, 51(3):230–239, 2009.

[5] A. Liutkus. The 2015 signal separation evaluation campaign. In *Latent Variable Analysis and Signal Separation: 12th International Conference, LVA/ICA 2015, Liberec, Czech Republic, August 25-28, 2015, Proceedings*, volume 9237, page 387. Springer, 2015.

[6] A. J. Simpson, G. Roma, and M. D. Plumbley. Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network. In *Proceedings of the International Conference on Latent Variable Analysis and Signal Separation (LVA/ICA)*, pages 429–436, Liberec, Czech Republic, 2015.

[7] E. Vincent, N. Bertin, R. Gribonval, and F. Bimbot. From blind to guided audio source separation: How models and side information can improve the separation of sound. *IEEE Signal Processing Magazine*, 31(3):107–115, 2014.

[8] N. Wiener. *Extrapolation, interpolation, and smoothing of stationary time series.* MIT Press Cambridge, MA, 1949.

[9] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.