SINGING VOICE SEPARATION USING DEEP RECCURRENT NEURAL NETWORKS: A COMPREHENSIVE GUIDE TO AUDIO DEEP LEARING IN KERAS

Ryan Bhular
Department of Electrical and Computer Engineering, University of Rochester
{rbhular@ur.rochester.edu}

ABSTRACT

Deep learning for audio has quickly taken the industry by storm. Starting out in this subject can be daunting due to the lack of comprehensive material available. In this paper we will explore the basics of deep learning and audio preprocessing to train a variation on a popular sound source separation model.

1. INTRODUCTION

Computer audition has quickly become a trendy subject. From automatic music generation to the voice processing algorithms in smartphones, it has taken consumer products and created a new standard never before achieved. Some of the most predominant algorithms in this field deal with deep learning, outperforming classical approaches.

While attempting to learn this subject, the amount of resources seem to pale in comparison to those on computer vision and image processing. Although at a conceptual level deep learning for both images and audio are very similar, it might be hard to make the correlation between them when one first begins the educational journey.

In this paper I hope to explore how to get started in deep learning for audio, as well as walk through a prevalent research paper on singing voice separation [2]. We will also cover some Python programming and will be using the Keras framework with a TensorFlow backend. Keras is a widely used deep learning framework that is intuitive and robust.

1.1. Python Dependencies

In order to follow this paper, there are a few Python packages and tools we will need to download.

1.1.1. Anaconda Python
This is a Python environment commonly used. Feel free to use the standard python environment as an alternative. While using this environment to install other packages you may use the terminal command `conda install package_name`

1.1.2. NumPy
This library is used for matrix operations in python.

1.1.3. SciPy
This is the scientific python library. One of the functions we will be using in here is the STFT function which we will cover later.

1.1.4. Librosa
This is an alternative to SciPy for STFT. This library is primarily focused on processing audio files.

1.1.5. CUDA/CUDNN
This is a tool for supported NVIDIA graphics cards. This will allow the backend to run using the graphics process rather than the CPU on your system.

1.1.6. TensorFlow
This is the deep learning backend for our program. TensorFlow has Keras integrated into its API. Please also make sure to use the GPU version of TensorFlow.

2. AUDIO PREPROCESSING

The audio that we perceive every day is a time-based signal composed of an amplitude (magnitude) at each point in time. While this is ideal for storing audio files, it provides very few features of the audio. Instead of this, we can convert the audio from the time domain to the frequency domain using Fourier Transforms. This will allow us to see more of the content in the audio. However, doing this to an entire song will make it hard to narrow down the frequency content at certain points of time.

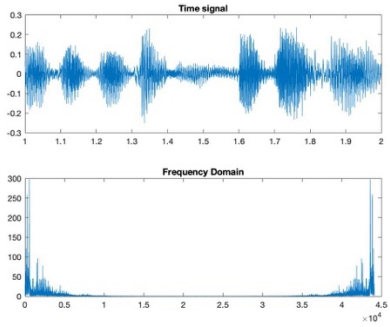

Figure 2.1: Frequency and time domain representation

2.1. Short-time Fourier Transform

By taking the Fourier Transform of small segments of the audio, we can find the frequency content at specific points in time. Usually these segments are determined in samples rather than time using lengths that are of the power of 2. These sections are called windows, and have common sizes of 256, 512, 1024, and 2048 samples. Then, a hop size is used in order to move over a predetermined number of samples to take the next window of the audio. This hop size is usually an integer division of the window length. The most common hop sizes make it so that the overlap between windowed signals is 25% or 50%. If the window length is 2048 samples with 50% overlap, the hop size will be 1024 samples. This means that the first 1024 samples in frame n will be the same as the last 1024 samples in frame n-1. In order to normalize the magnitudes of the overlapped sections, we multiply the signal by a normalization function. The most common functions used for audio are Gaussian, Hann, and Hamming. Below, we can see how this windowing process works with the blue plot being the time domain audio signal and the red plot being a Hamming window applied to be multiplied (sample by sample) to the audio signal.

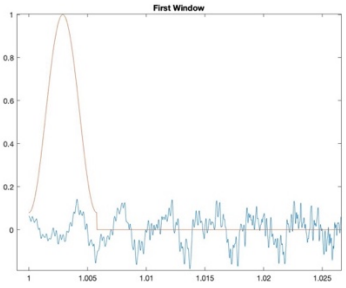

Figure 2.2: Window vs. Time domain audio

When we take the Fourier transform of each section, we get the Short-time Fourier Transform. In figure 2.3 we can see this transform. The x-axis marks the time segments of each window and the y-axis shows the frequency content sectioned off into frequency bins. With 50% overlap, the number of frequency bins is 1+hop size.

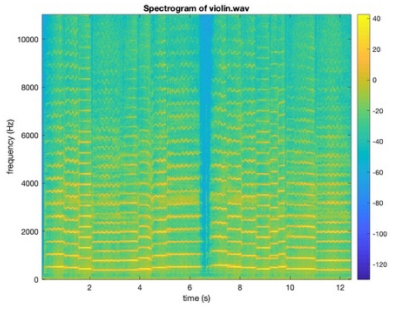

Figure 2.3: Short Time Fourier Transform

Both SciPy and Librosa contain STFT functions.

2.1.1. Magnitude and Phase Spectrums

When converting any time signal to the frequency domain using Fourier transforms, the transform contains real and imaginary data. In order to process audio, we typically only require the magnitude spectrogram. This is found by taking the absolute value of the spectrogram. However, when inversing only the magnitude spectrums, a user may notice a drastic change in the audio reproduced. This is because the audio lacks phase information. This phase information is useful to give the audio unique characteristics outside of the capacity of its magnitude. Because of this, when we take the STFT of any signal, we should make sure to save the phase information as well and add it back before inversing the STFT. Even if you have changed the magnitude spectrum of the signal, adding the phase information from the original signal is very important, and this should not have any diminishing effects on the returned audio. The magnitude and phase spectrograms of S can be found using `numpy.abs(S)` and `numpy.angle(S)` respectively.

2.2. Chunking the Audio (creating batches)

Taking the STFT 2-dimensional matrices with varying length in the time axis is returned. However, neural network layers need to accept matrices of the same length. To remedy this we can split the magnitude spectrum of each audio file into equal sized chunks and store them in order in 3-dimensional NumPy matrix.

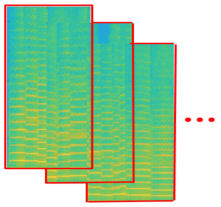

Figure 2.4: Creating chunks of the audio

2.3. Preprocessing Parameters

For this application, an STFT is taken with a window length of 258 samples with 50% overlap. The spectrogram is segmented every 100 timesteps. This creates matrices of 129 frequency bins by 100 windows.

3. DATASET

This paper deals with a supervised learning model. This means that while training, we have input data and a corresponding ground-truth output.

3.1. Generating Subsets

In order to successfully train, monitor and test a model the dataset is usually split into three different subsets. For supervised learning, these subsets are stored in separate lists or arrays in python after being preprocessed, for example, taking the STFT and chunking the audio. Each subset should

also have two separate lists each. One for the input data to the model and one for the ground truth data for reference.

3.1.1. Training Set

The training set is typically the largest subset. This set is used to train the model to update the weights through each epoch.

An epoch is when all the training data has been input to the model one time. To create stronger weights in the model, it goes through a large number of epochs in order to build strong enough weights.

3.1.2. Validation Set

This set is used to monitor the training process. It is one of the smaller subsets.

3.1.3. Test Set

The test set is used to test the model after the training process is complete. This data is new to the model, so that the performance of the model can be accessed accurately.

3.2. DSD 100 Dataset

For the model covered in this paper, we will use the DSD 100 Dataset provided by SigSep [3]. This dataset has a predetermined development and test set for training, validation, and testing. It also contains mixture audio of vocals and instrumentation to be the input of the model as well as separated ground truth audio of vocals, percussion, bass, and other. For our purposes we only used the mixture files to input into the model and the vocal files for ground truth.

4. INTORDUCTION TO NEURAL NETWORK

Neural networks are comprised of layers containing nodes or tensors. Each tensor contains a function of some sort and the output from these tensors are fed to the tensors of the next layer. These layers usually accept a 2-dimensional array input but are fed in, one feature at a time, for each frequency bin. For a magnitude spectrum, the first layer will usually have a number of tensors equal to the number of bins in the spectrum.

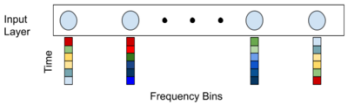

Figure 4.1: Typical input layer

Keras provides a framework in which we do not have to code the output function for each layer. However, we will go over a couple relevant and commonly used layers.

4.1. Dense/Fully Connected Layer

This first type of layer we will cover is the dense or fully connected layer. This layer consists of tensors that connect to each tensor of the next layer. The output of this layer is based on an activation function shown in Equation 4.1.

$$output = activation(input \odot weights + bias)$$

Equation 4.1

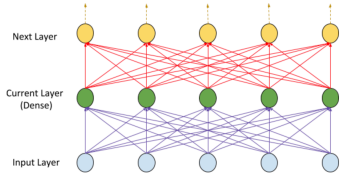

Figure 4.2: Fully connected Layer

4.2. Recurrent Layer

Recurrent layers are layers that not only have weighted connections between layers but also within the layer themselves. This makes recurrent neural networks very good with sequential data such as audio. These layers are usually used as hidden layers (between the input and output).

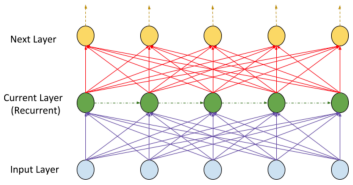

Figure 4.3: Recurrent Layer

4.1.1. Long-Short Term Memory (LSTM) layer

This recurrent layer architecture has gained popularity in audio and speech recognition models. Traditional recurrent layers only deal with short term memory from the tensors directly preceding it. LSTMs propose a way to integrate long term memory into the layer with the assumption that features earlier in the sequence do influence other features. These layers are very similar to fully connected layers between layers, but within the layer itself, there are separate activation functions to connect the tensors.

4.3. Activation Functions

Activation functions help to set how the output of each tenor behaves. Each activation has its benefits for different learning objectives. Figure 4.4 shows the graphs of these activation functions.

4.3.1. Sigmoid

This function is a smoothed binary step. This is helpful because it allows the output to have more variation than a binary step so that desired features do not get cut off as easily.

4.3.2. Rectified Linear (ReLu)

This is one of the most common activation functions in deep learning. Typically, these functions operate very fast and are good for reducing the training time.

4.3.3. Leaky ReLu

The problem with ReLu functions is that the model can often get stuck on the negative side because the output is 0 for any negative value. In order to remedy this, a leaky ReLu is used to set negative values to negative outputs. This allows the network to recover from negative inputs.

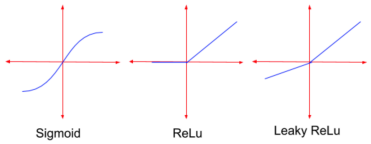

Figure 4.4: Activation Functions

4.4. Dropout

In order to avoid overfitting of the model on the training data, we include dropout at each layer. Dropout sets a random probability that the output from a tensor will be ignored for one epoch.

5. PROPOSED MODEL

5.1. Model Architecture

For our source separation we are using a simplified variation of the model presented in [2]. This model takes in a spectrogram of a mixture, in order to produce a mask to be reapplied to the mixture for separation. The model in this paper differs from the model proposed in [2] because we are only outputting one source for the vocals. The explored model can be seen in Figure 5.1.

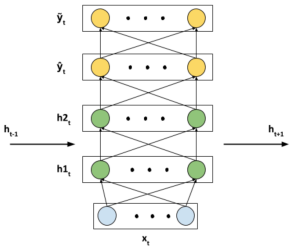

Figure 5.1: Model Architecture

In the model, x_t is the spectrogram of the input mixture. There are 2 hidden LSTM layers (h_1 and h_2) with leaky ReLu activations. These feed into a dense layer ($\hat{y}$) which has a sigmoid activation function. Finally, this dense layer feeds into a custom masking layer.

5.1.1. Time-Frequency Soft Mask

A mask is a spatial filter in image processing by which the image is multiplied pixel by pixel with a binary value (hard mask) or a value from 0 to 1 (soft mask). Applying a mask to spectrogram data will help to create a filter in the frequency domain. For our purposed model, we will use a Time-Frequency soft mask that can be found using equation 5.1.

$$mask = \frac{\text{separated } Spectrogram}{mixture\ Spectrogram}$$

Equation 5.1

5.2. Training Objective

Training of the model uses a loss function to compute the difference between the ground truth output and the output from the model at each epoch. In this example, binary cross entropy is used.

6. EXPERIMENTS

6.1. Running the Model

While running the model, a very slow change in loss at every epoch was observed, and at around 10 epochs the loss plateaus at around 5.5. At closer examination, this may be because of defective time frequency masks that the model is trained on.

6.2. Issues with The Model

In theory, the Time-Frequency Soft Masking approach presented is correct at a very nascent level. However, due to the separated source audio having higher frequency intensity than the mixture input into the model at certain instances, the mask may return values significantly greater than the expected upper limit, 1. This leads me to believe that the reason the architecture proposed in [1] and [2] have two separate outputs is because by adding these separate sources together, it becomes easier to obtain a normalized mask. Thus, if the mixture is made up of two sources, y_1 and y_2 (i.e. the singing voice and background instrumentation), we can express the Time-Frequency Soft Mask as defined in [4]:

$$m_1(f) = \frac{|\widehat{y_1}(f)|}{|\widehat{y_1}(f)| + |\widehat{y_2}(f)|}$$

Equation 6.1

This issue could also be due to the Librosa's and SciPy's STFT function and the input parameters. Using MATLAB, we are able to get a maximum value in the mask as 1.57. However, this value is still incorrect. This needs to be explored in later work in order to complete this comprehensive guide.

7. FUTURE WORK

With the momentum from this research, I hope to continue developing this model in order to obtain a model capable of performing source separation for multiple sources. I believe this is possible using the resources and DSD100 dataset covered in this paper. This is because the mixture is found by adding the remaining separated sources spectrograms in the DSD100 dataset. This will create a more interesting and complex model, while also allowing us to address the issue encountered with the time-frequency mask.

8. CONCLUSION

Moving forward, we need to determine the cause of the wrong mask value for training. While I do feel confident that this model should work give the correct masks as targets, I will need to verify this before moving to a more complex model with multiple sources. As we continue to develop this in future work, we should explore the use of data augmentation for audio files in order to create a richer dataset and produce a stronger deep neural network model.

9. ACKNOWLEDGMENT

I would like to thank Professor Zhiyao Duan for inspiring me to take on this interesting and complex topic. I would also like to thank Emre Eskimez, Yichi Zhang, and the teaching assistants for the Spring 2019 ECE 472 course: Mingqing Yun, Ge Zhu, and Christos Benetatos for their continued guidance through this work.

10. REFERENCES

[1] P. S. Huang, S. D. Chen, P. Smaragdis, and M. HasegawaJohnson, "Singing-voice separation from monaural recordings using robust principal component analysis," in IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), Mar. 2012, pp. 57– 60.

[2] 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 (ICASSP), 2014.

[3] Liutkus, A., St¨oter, F.R., Rafii, Z., Kitamura, D., Rivet, B., Ito, N., Ono, N., Fontecave, J.: The 2016 signal separation evaluation campaign. In: International Conference on Latent Variable Analysis and Signal Separation, Springer (2017) 323–332

[4] D. Wang, "Time-frequency masking for speech separation and its potential for hearing aid design," Trends in Amplification, vol. 12, no. 4, pp. 332–353, 2008.

[5] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735-1780.

[6] Wang, Y. and Wang, D. (2013). Towards Scaling Up Classification-Based Speech Separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(7), pp.1381-1390.

[7] M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks. In Advances in Neural Information Processing Systems, pages 190–198, 2013.

[8] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Processing Magazine, 29:82–97, Nov. 2012.

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