# Audio Source Separation Using Deep Neural Networks

Low Latency Online Monoaural Source Separation For Drums, Bass and Vocals

Pritish Chandna

TESI DOCTORAL UPF / ANY 2016

DIRECTOR DE LA TESI
Director: Jordi Janer And Marius Miron Departament Music Technology Group

dedication

## Acknowledgments.

I would like to thank my supervisors Jordi Janer and Marius Miron, who, at times went out of his way to help me with this work and the project would have been impossible without his contribution. Special thanks also to Jordi Pons, who, although not involved directly in the project, was enthusiastically involved in discussions and provided meaningful insights at times. I would also like to thank the Music Technology Group (MTG) at the Universitat Pompeu Fabra Barcelona for providing me the oppurtunity, knowledge, guidance and tools required to complete this thesis.

# Abstract

This thesis presents a low latency online source separation algorithm based on convolutional neural networks. Building on ideas from previous research on source separation, we propose an algorithm using a deep neural network with convolutional layers. This type of neural network has resulted in state-of-the-art techniques for several image processing problems. We try to adapt these ideas to the audio domain, focusing on low-latency extraction of 4 tracks (vocals, bass, drums and other instruments) from a single-channel (monaural) musical recording. We try to minimize processing time for the algorithm without compromising on performance through data compression.

The Mixing Secrets Dataset 100 (MSD100) and the Demixing Secrets Dataset 100 (DSD100) are used for evaluation of the methodology . The results achieved by the algorithm show a 8.4 dB gain in SDR and a 9 dB gain in SAR for vocals over the state-of-the-art deep learning source separation approach using recurrent neural networks.The system's performance is comparable with other state-of-the-art algorithms like non-negative matrix factorization, in terms of separation performance, while improving significantly on processing time. This thesis is a stepping block for further research in this area, particularly for implementation of source separation algorithms for medical purposes like speech enhancement for cochlear implants, a task that requires low-latency.

# Preface

# Contents

# List of Figures

# List of Tables

# Introduction

Source separation as a research topic has occupied many researchers over the last few years, fueled by advancements in computer technology, signal processing and machine learning. While source separation has application across various strata including medicine, neuroscience, finance, communications and chemistry, this thesis will focus on source separation in the context of an audio mixture. Audio source separation is useful as an interediatory step for several applications such as automatic speech recognition, [Kokkinakis and Loizou, 2008] fundamental frequency estimation [Gómez et al., 2012] and beat tracking [Zapata and Gomez, 2013], or applications such as remixing or upmixing. Some applications of audio source separation require low-latency processing, such as the ones to be computed in real-time. One such application is speech enhancement for cochlear implant patients, where the speech is separated from the input mixture and remixed and enhanced before being processed in the implant. [Hidalgo, 2012]

The thesis focuses on separating four sources, vocals, bass, drums and others from a musical recordings of music from various genres. Deep neural networks, particularly convolutional neural networks are explored, building on work done by other researchers in the field and by advancements in image processing, where such networks have achieved remarkable success.

Formally, the problem at hand can be defined by considering a polyphonic audio signal to be a mixture of component signals. Source separation algorithms aim to extract the component signals from the polyphonic mixture. This is often referred to as a cocktail party problem taking a party with several people as an analogy. With a multitude of simultaneous voices, the task faced by an individual is to separate out the voice of interest from the others, which may be referred to as background noise. This problem has been well-studied for decades, since its introduction in the field of cognitive science by Colin Cherry.

A brief description of some of the state-of-the-art algorithms which attempt to solve the problem is provided herewith, along with links to original papers from which they have been cited. For a complete description of the algorithms, please refer to the cited papers. Following the state-of-the-art, a system for source separation is presented in the Methodology section. Finally, the the system is

evaluated on the MSD100 and DSD100 dataets and the performance is compared with that of state-of-the-art algortihms.

# Chapter 1

# STATE OF THE ART

## 1.1 Source Separation

Source separation in an audio context can be defined as the extraction or isolation of a component signal from a mixture of component signals. Audio, specially musical recordings can be done live, with several microphones capturing different instruments, but with limited isolation between the different instruments or in a studio, with a microphone or audio line dedicated to each instrument. These different sources are then down-mixed to form the final mixture. At this point, the final mix can be considered as a linear sum of the individual sources, with mixing filters applied to each source. This can mathematically be represented as:

$$x_i(t) = \sum_{j=1}^{J} \sum_{\tau=-\infty}^{\infty} a_{ij}(t-\tau,\tau) s_j(t-\tau) \tag{1.1}$$

Where $x_i(t)$ is the final mixture, $s_j$ is a source, $J$ is the number of sources and $a_{ij}$ is a filter which may be applied to the source while mixing. As long as the applied filter is linear, the mix is a linear combination of the sources. If the mixture is linear in time domain, the corresponding frequency domain representation (after STFT) is also linear:

$$X_i(f,t) = \sum_{j=1}^{J} \sum_{\tau=-\infty}^{\infty} A_{ij}(f,t-\tau,\tau) S_j(f,t-\tau) \tag{1.2}$$

However, other mixing effects can be applied to the final mix including compression, panning, reverb etc. At this point, the mixture is no longer a linear sum of the individual source channels. Depending on the recording technique, the mixture, in the context of source separation, can be classified under the following categories:

1. Under-determined or Over-determined based on the knowledge of the number of sources that were mixed together. For example, a live recording with several microphones recording different instruments is considered to be an over-determined mixture whereas a recording with just one microphone recording several instruments would be an under-determined mixture.

2. Instantaneous or convoluted: This categorization is determined by the knowledge of post-production effects applied to the final mixture.

3. Time-varying vs time-invariant: A mixture might not be static over time, for example, a recording done with a moving microphone will be considered as time-varying.

This thesis will focus on single-channel over-determined separation, with the assumption that the mixture is instantaneous and time-invariant. Source separation algorithms can broadly be divided into two main categories:

1. Blind source separation: This branch of source separation algorithms tries to extract component signals from the mixture with little or no prior knowledge about the sources. The most popular of these algorithms exploit the (assumed) statistical independence of the source signals. Some examples include: Independent Component Analysis (ICA), [Lopez P. et al., 2011] [Dadula and Dadios, 2014] Independent Subspace Analysis (ISA), [Wellhausen, 2006] and Non-Negative Matrix Factorization (NMF). [Duong et al., 2014, El Badawy et al., 2015]. Work has also been done using robust principal component analysis, [Huang et al., 2012]. With advancements in deep learning theories and technologies, many researchers in recent years have been exploring deep neural network architectures. Recent research in this field has focused on using machine learning techniques like Hidden Markov Models (HMMs), [Higuchi and Kameoka, 2014] factorial HMMs, [Ozerov et al., 2009] and Deep Neural Networks (DNNs). [Huang et al., 2014][Grais et al., 2014] [Nugraha et al., 2016]

2. Informed Source Separation: These methods assume prior knowledge of the source signals, in the form of MIDI or score or other representations of music. [Miron et al., 2015] However, since this thesis does not focus on informed source separation, a detailed explanation of the same will not be provided herewith.

Adaptive Weiner filtering in the time-frequency domain is usually used for source separation. After all sources $S_{imgij}$ have been estimated, they can be used to compute masks with gains between 0 and 1, which are applied to the mixture signal:

$$\hat{S}_{imgij}(n,f) = \frac{|S_{imgij}(n,f)|^2}{\sum_{j=1}^{J} |S_{imgij}(n,f)|^2} X_i(n,f) \tag{1.3}$$

Note that this operation is done only on the magnitude spectrogram while the phase of the original mixture is used for synthesis of the individual sources. Some of the common methods used for source separation are described in brief below.

### 1.1.1 Principal Component Analysis (PCA)

Principal Component Analysis and Independent Component Analysis are statistical techniques that have been used for Blind Source Separation by many researchers. [Lopez P. et al., 2011] [Dadula and Dadios, 2014] The basic idea behind these techniques is to project the data from a time series like an audio recording into a new set of axes that are based on some statistical criterion. These axes are set to be statistically independent in contrast to the Fourier Transform where the time domain data is projected onto a axes of frequencies, which might overlap. The frequency axes in a Fourier transform are fixed in terms of the frequency bins and will remain the same regardless of the piece analyzed, whereas in the PCA and ICA, the axes are dynamic and are different for each piece being analyzed. The axes, once discovered, can then be separated and then inverted to find the different sources present in the input signal. Variance is the measure used to separate axes in PCA. The axes are recursively chosen as directions in which variance in the signal is maximized, leading to decorrelation in the second order among the axes. Thus, the main components of the energy of the signal are contained in the first few axes. This can be used both for data compression and source separation.

In ICA, the fourth moment, known as kurtosis, is generally the criteria for defining the new axes. Kurtosis is basically a measure of the non-Gaussianity of a signal, a negative kurtosis indicates a signal that has a probability distribution function that is broader than a Gaussian distribution whereas a positive value would represent a distribution that is narrower. Therefore, ICA separates a signal into non-Gaussian independent sources while PCA separates a signal into Gaussian independent sources.

These techniques are often implemented as a series of matrix multiplications representing filters. In general, a signal $X$ with $N$ dimensions and $M$ samples (a matrix of dimensions $MxN$) can can be mapped into a signal $Y$ using a transform matrix $W$ of dimensions $NxN$ as $YT = WXT$. This transformation projects the signal into a different axis based on the transform matrix. If the length of the transformed signal is the same as the input signal, then the transformation is termed as an orthogonal transformation and the axes are perpendicular.

This is a lossless transformation as the original signal can be reconstructed without any loss of information. If one or more of the columns of the transform matrix is set to zero, it is termed as a lossy transformation and is used for filtering or compression of data. A biorthogonal transformation is one where the axes are not perpendicular. However, the transformation is still lossless. PCA and ICA are orthogonal and bi-orthogonal transformations respectively. The main challenge in the two cases is discovering the transformation matrix, $W$. The sources can then be separated and then reconstructed using the inverse of the $W$, $A$.

### 1.1.2 Non-Negative Matrix Factorization (NMF)

NMFs have been widely used for supervised source separation in the past. The basic idea is to represent a matrix $Y$ as a combination of basis ($B$) and activation gains ($G$) as $Y = BG$. The basis vector represent the frequency response of the source at a given time and can be thought of as a vertical vector, whereas $G$ represents the gain of the frequency response at any point along the time axis. $G$ is thus a horizontal vector along time. Thus, for source separation, if the mixture $Y$ is known to be composed of two sources,$S_1$ and $S_2$, such that $Y = S_1 + S_2$.And the basis vectors for these two source are computed as $B_1$ and $B_2$, then the mixture can be represented as $Y = B_1G_1 + B_2G_2$, where $G_1$ and $G_2$ are the respective activation gains for the two sources present are different moments of time in the mixture. However, it must be noted that the NMF approach assumes that mixture can be represented as a linear combination of the basis dictionaries. For $K$ sources, the mixture can be represented as:

$$X_{i,j} = \sum_{k=1}^{k=K} B_{i,k}G_{k,j} \tag{1.4}$$

For source separation, the divergence between $X$ and $BG$ is to be minimized to ensure that the sources found by the algorithm represent the mixture signal:

$$\{B, G\} = argmin_{B,G\geq 0}\mathcal{D}(X, BG) \tag{1.5}$$

Where $\mathcal{D}$ represents a divergence function, which may be:

1. Euclidean distance:

$$\mathcal{D}(A, B) = \|A - B\|^2 = \sum_{ij}(A_{ij} - B_{ij})^2 \tag{1.6}$$

2. Kullback-Leibler (KL) Divergence:

$$\mathcal{D}(A, B) = \sum_{ij}\left(A_{ij}log\frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij}\right) \tag{1.7}$$

The multiplicative update algorithm is one of the most common algorithms used for this divergence [Lee and Seung, 2001] and can be described as follows:

1. Vectors $B$ and $G$ are initialized with random values
2. For Euclidean divergence, $B$ is updated as:

$$B :\Rightarrow B \bigotimes \frac{XG^T}{(BG)G^T} \tag{1.8}$$

   For KL divergence, $B$ is updated as:

$$B :\Rightarrow B \bigotimes \frac{(\frac{X}{(BG)})G^T}{1G^T} \tag{1.9}$$

   where $1$ represents a matrix of all ones of the same size as $X$

3. For Euclidean divergence, $G$ is updated as:

$$G :\Rightarrow G \bigotimes \frac{B^T X}{B^T(BG)} \tag{1.10}$$

   While for KL divergence, it is updated as:

$$G :\Rightarrow G \bigotimes \frac{B^T(\frac{X}{BG})}{B^T 1} \tag{1.11}$$

This process is repeated for a pre-decided number of epochs or until the the divergence falls below a pre-decided threshold. Once the magnitude spectrograms of the sources are estimated, the sources can be synthesized using the phase of the original mixture signal. Note that the NMF algorithm needs to be adapted for different types of source separation, depending on the number of sources and the mixing process applied to generate the mixture.

### 1.1.3 Flexible Audio Source Separation Toolbox (FASST)

FASST is a variation of the NMF algorithm described in the previous section, proposed by [Ozerov et al., 2012], based on a generalized expectation-maximization (GEM) algorithm. [A. P. Dempster, ] The Toolbox is meant to be a generalized implementation, that is flexible for different use cases of source separation. In the implementation of the toolbox, each source to be separated from a mixture is represented by an excitation-filter model, as shown in eq: 1.12.

$$V_j = V_j^{ex} \odot V_j^{ft} \tag{1.12}$$

where $V_j^{ex}$ represents the excitation spectral power and $V_j^{ft}$ represents the filter spectral power. $V_j^{ex}$ is further represented as a modulation of spectral patterns, denoted by $E_j^{ex}$ by time activation coefficients $P_j^{ex}$. These two terms can be seen as analogous to the basis vectors and activation gains used in NMF. The spectral patterns are further represented as linear combinations of narrow band spectral patterns $W_j^{ex}$ and weights $U_j^{ex}$. Similarly, the time activation patterns are reprsented as linear combinations of time localized patterns, $H_j^{ex}$ and $G_J^{ex}$. A similar representation is carried out for $V_j^{ft}$, resulting in the following equation:

$$V_j = (W_j^{ex} U_j^{ex} G_j^{ex} H_j^{ex}) \odot (W_j^{ft} U_j^{ft} G_j^{ft} H_j^{ft}) \tag{1.13}$$

These parameters are estimated for each source using the GEM algorithm, with increments along each epoch following the Multiplicative Update algorithm. [Lee and Seung, 2001] Following this, the estimated sources are used to calculate weiner masks which are applied to the mixture spectrogram to determine the final source estimates.

## 1.2 Source Separation Evaluation

Evaluation of source separation algorithms is a tricky subject as each person might have his own views on the quality of an audio recording. Fortunately, the valuation metric has been well-studied and objectified by Emmanuel Vincent et al. [Vincent et al., 2006] [Vincent et al., 2007]

The paper suggests that source separation evaluation should be done in the context of the application of the separation. A mixture can be considered to be a linear sum of sources, $(s_j)_{j \in J}$ and some noise which might be mixed into the signal, $n$. The simplest perceivable measure is to use the L2 normalized difference between the estimated source $\widehat{s}_j$ and the target source $s_j$:

$$D = min_{\epsilon = \pm 1} \left\| \frac{\widehat{s}_j}{\|\widehat{s}_j\|} - \epsilon \frac{s_j}{\|s_j\|} \right\|^2 \tag{1.14}$$

The difference is always positive and will be zero if and only if $\widehat{s}_j = s_j$. However, even in case of the worst case scenario, where $\widehat{s}_j$ is orthogonal to $s_j$, the difference would still be limited to a maximum of two, as the sources have been normalized. Also, in case of a noise or distortion which is correlated to the desired source, this measure gives a low value, which may or may not be desirable, depending on the application. For example, in a high quality audio recording, a correlated noise may be considered undesirable, but in the case where the extracted source is to be remixed or processed further (as in the case of speech recognition) some amount of distortion may be allowed. To take these cases into account the paper proposes some measures for evaluation of blind source separation:

To calculate these measures, the estimated source is first decomposed into four constituents as:

$$\widehat{s}_j = s_{target} + e_{interf} + e_{spat} + e_{artif} \tag{1.15}$$

$s_{target}$ is a modified version of the target source, which may contain certain allowed distortions, $\mathcal{F}$. $e_{interf}$ represents interference coming from unwanted sources $(s_{j'})_{j' \neq j}$ which might be mixed along with $\widehat{s}_j$. $e_{spat}$ represents spatial distortions and $e_{artif}$ represents other noises such as forbidden distortions, not in the set $\mathcal{F}$ and burbling artifacts.

For analysis, orthogonal projections are considered. These can be defined as follows: For a subspace with vectors $y_1, y_2......y_n$, $\Pi\{y_1, y_2......y_n\}$ is a matrix which projects a vector onto this subspace. Hence, the following matrices are defined:

$$P_{s,j} := \Pi(s_j) \tag{1.16}$$

$$P_S := \Pi\{(s_{j'})_{1 \leq j' \leq n}\} \tag{1.17}$$

$$P_{S,n} := \Pi\{(s_{j'})_{1 \leq j' \leq n}, (n_i)_{1 \leq i \leq m}\} \tag{1.18}$$

These projectors are used to find the constituents of $\widehat{s}_j$ as:

$$s_{target} := P_{s,j}\widehat{s}_j \tag{1.19}$$

$$e_{interf} := P_S\widehat{s}_j - P_{s,j}\widehat{s}_j \tag{1.20}$$

$$e_{spat} := P_{S,n}\widehat{s}_j - P_{s,j}\widehat{s}_j \tag{1.21}$$

$$e_{artif} := \widehat{s}_j - P_{S,n}\widehat{s}_j \tag{1.22}$$

The computation of $s_{target}$ is straightforward, using an inner product:

$$s_{target} := \langle \widehat{s}_j, s_j \rangle \frac{s_j}{\|s_j\|^2} \tag{1.23}$$

To compute $P_S$ and $P_{S,n}$, a vector $c$ of coefficients is defined as:

$$c = R_{SS}^{-1}[\langle \widehat{s}_j, s_1 \rangle ... \langle \widehat{s}_j, s_n \rangle]^H \tag{1.24}$$

where $R_{SS}$ denotes the Gram matrix. Then $P_S\widehat{s}_j$ can be written as:

$$P_S\widehat{s}_j = \sum_{j'=1}^{n} \bar{c}_{j'} s_{j'} = c^H s \tag{1.25}$$

where $(.)^H$ denotes the Hermitian transposition. However, if the sources and the noise signal are assumed to be orthogonal to each other, the above equations can be re-written as:

$$e_{interf} = \sum_{j' \neq j} \langle \widehat{s}_j, s_{j'} \rangle \frac{s_{j'}}{\|s_{j'}\|^2} \tag{1.26}$$

$$P_{S,n}\widehat{s}_j = P_S\widehat{s}_j + \sum_{i=1}^{m} \langle \widehat{s}_j, n_i \rangle \frac{n_i}{\|n_i\|^2} \tag{1.27}$$

Using these values, the following evaluation measures can be computed:

1. Source to Distortion Ratio:

$$SDR := 10log_{10}\frac{||s_{target}||^2}{||e_{interf} + e_{spat} + e_{artif}||^2} \tag{1.28}$$

   This measure represents the overall performance of the source separation algorithm.

2. Source to Interferences Ratio:

$$SIR := 10log_{10}\frac{||s_{target}||^2}{||e_{interf}||^2} \tag{1.29}$$

   The suppression of interfering sources in the separation is objectified by this measure.

3. Image to Spatial distortion Ratio:

$$ISR := 10log_{10}\frac{||s_{target} + e_{interf}||^2}{||e_{spat}||^2} \tag{1.30}$$

4. Sources to Artifacts Ratio:

$$SAR := 10log_{10}\frac{||s_{target} + e_{interf} + e_{spat}||^2}{||e_{artif}||^2} \tag{1.31}$$

   This measure estimates the artifacts introduced by the source separation process.

The values of the four components, $s_{target}$, $e_{interf}$, $e_{spat}$ and $e_{artif}$ vary over time and the paper suggests windowing the signal in question to calculate local values for these features. The values can then be summarized using statistical measures.

## 1.3 Neural Networks

Following the success of machine learning techniques in other fields, particularly image processing,[Krizhevsky et al., 2012a] several researchers have adopted advanced algorithms to the Source Separation paradigm. The most widely discussed of these include Deep Neural Networks, which use neural networks with more than one layer to learn information about audio sources. [Huang et al., 2014, Huang et al., 2014, Grais et al., 2013, Grais et al., 2014, Simpson, 2015, Nugraha et al., 2016] A brief introduction to neural networks is provided below, followed by some specific examples of the application of the same to the source separation problem.

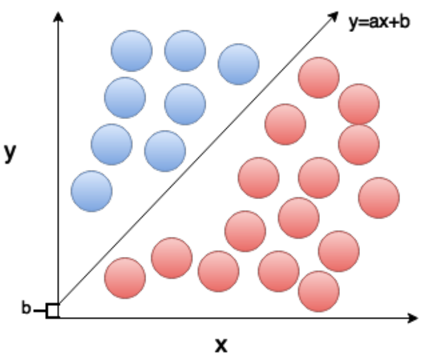


Figure 1.1: Linear Representation: The blue dots represent class A and the red dots represent class B. The line $y = ax + b$ can be used to represent the entire data as Class A: $y > ax + b$ and Class B: $y <= ax + b$

### 1.3.1 Linear Representation Of Data

It can be seen from the sections above that conventional source separation techniques like PCA and NMF try to find different representations of the mixture data from which the individual components can be extracted. Here, a brief description of the techniques for representing data, leading upto neural networks is provided. To make sense of data, it sometimes needs to be represented in another form, form example, classified as belonging to Class A or B. Some data, such as the one shown in fig 1.1, is linearly separable, i.e. it can be represented entirely with a linear equation. For the example shown, the blue dots represent data belonging to class A whereas the red dots represent data belonging to class B. This data can thus be represented as a linear equation as:

$$y = ax + b \tag{1.32}$$

where

$$ClassA : y > ax + b \tag{1.33}$$

and

$$ClassB : y <= ax + b \tag{1.34}$$

However, it is not always possible to find linear representations of data as for the one shown in fig 1.2. Neural networks provide an elegant way to find representations of data which cannot be done with simple models as the one shown in eq 1.32

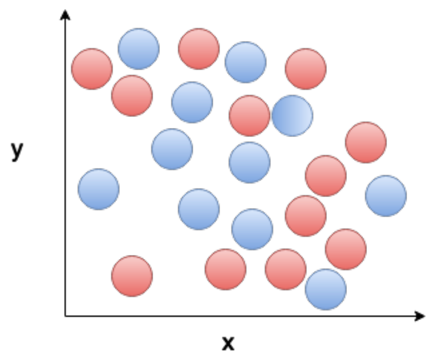


Figure 1.2: Data that is not linearly separable

### 1.3.2 Neural Networks For Non-Linear Representation Of Data

A neural network is an information processing system inspired by the human nervous system.[Grossberg, 1988] Like the nervous system, artificial neural networks comprise of connections of nodes called neurons. Each neuron receives an input, processes the information in the input and gives an output. These neurons contain parameters, which must be optimized using a training set, which has a labeled ground truth. Once trained, the network can be used to input data to produce outputs for testing and for future use. While many possible arrangements of neurons are possible, the most commonly used consists of three vertically arranged layers, as shown in figure 1.3

1. Input Layer
2. Hidden Layer
3. Output Layer

Each layer consists of neurons, which can mathematically be described in terms of its parameters. These parameters are optimized during the training phase of the network and are described below:

1. Inputs: $X$, representing the input vector:: $x_1 ... x_n$
2. A bias unit: $x_0$, which is a constant value added to the input vector.
3. Weights for each of the inputs and the bias: $W$ representing the vector $w_0 ... w_n$.
4. A non-linear activation function, $\sigma$, which could be:

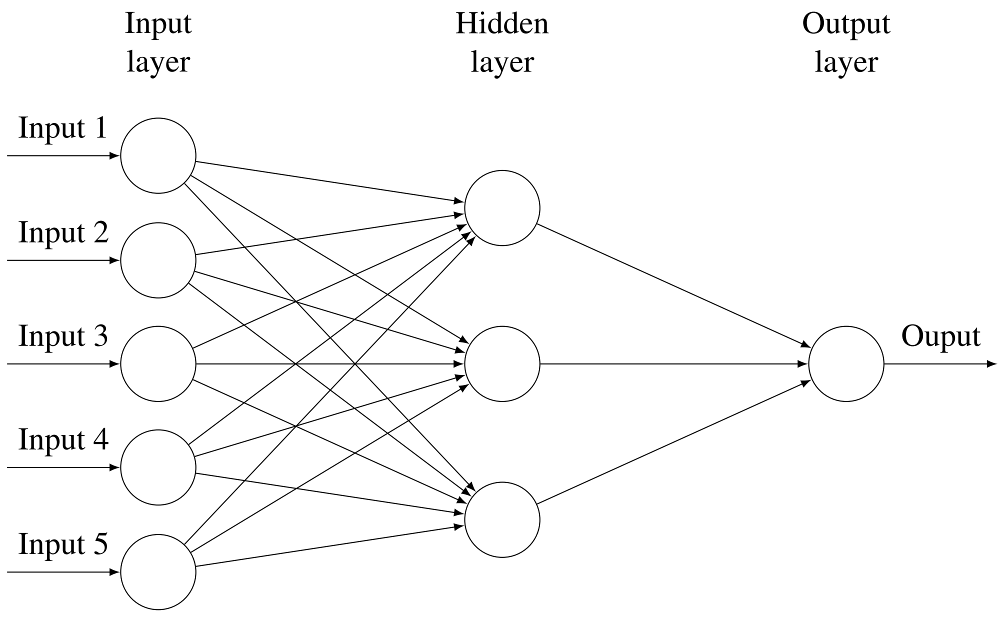


Figure 1.3: Basic Neural Network: connections are shown between neurons for the input layer, the hidden layer and the output layer. Note that each node in the input layer is connected to each node in the hidden layer and each node in the hidden layer is connected to each node in the output layer.

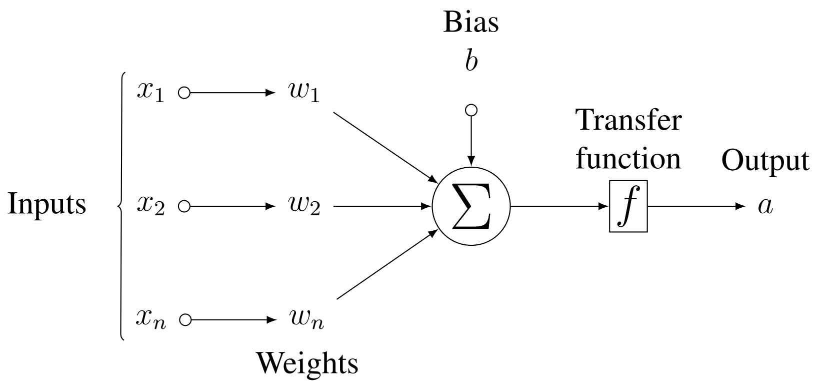
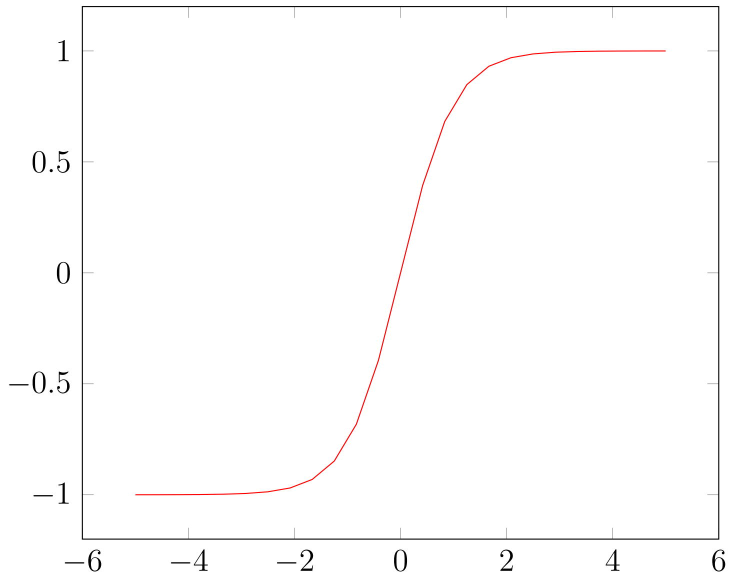


Figure 1.4: A node or neuron of the neural network with a bias unit: $x_0$, inputs: $x_1 ... x_n$, weights: $w_0 ... w_n$ and the output: a

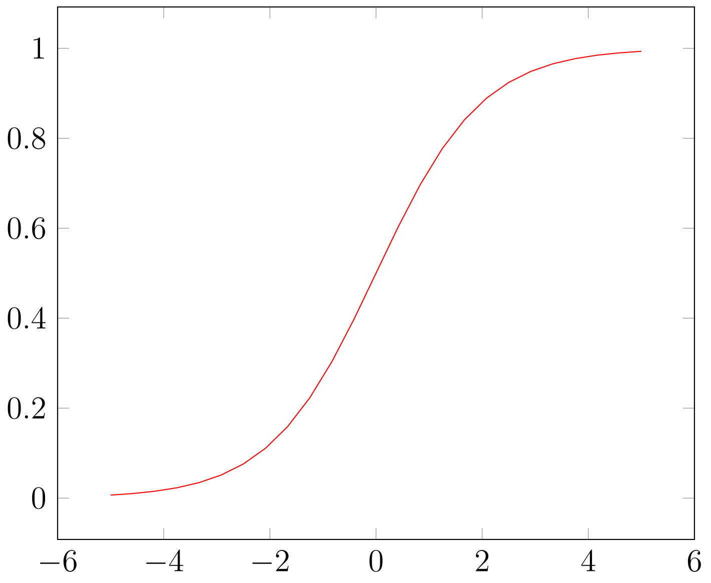
(a) Tanh

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.35}$$



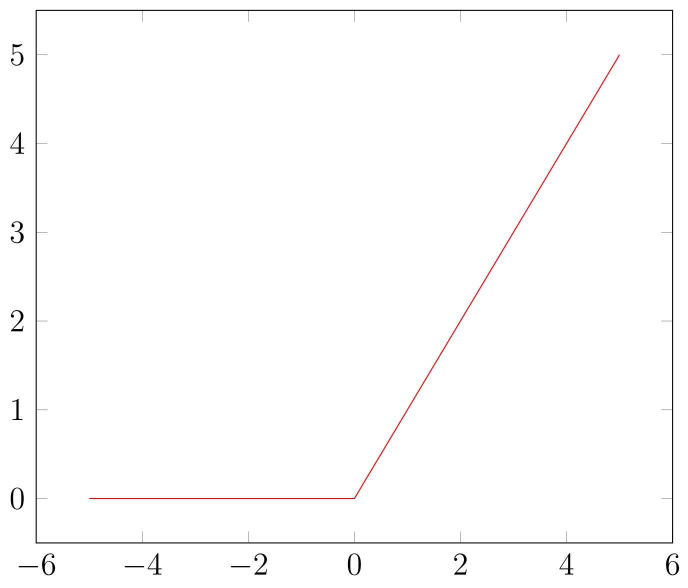
(b) Sigmoid

$$S(x) = \frac{1}{1 + e^{-x}} \tag{1.36}$$



(c) Rectified Linear Unit (ReLU)

$$f(x) = max(0, x) \tag{1.37}$$



5. The output, a, which can be represented as:

$$a = \sigma(\sum_{i=0}^{i=N} w_i x_i) \tag{1.38}$$

Which can be re-written as:

$$a = \sigma\left(w_0 x_0 + \sum_{i=1}^{i=N} w_i x_i\right) = \sigma\left(b + \sum_{i=1}^{i=N} w_i x_i\right) \tag{1.39}$$

where $b$ represents the bias added to the layer. Here, $b$ and $W$ are the parameters to be optimized using the training data set.

As each node contains a non-linear function, the network is capable of learning various non-linear representations of the input data through various combinations of input nodes and activation functions.

### 1.3.3 Deep Neural Networks

Deep neural networks are neural networks with more than one hidden layer, as shown in fig 1.5 . As data passes through more than one layer, more abstract representations of the data can be discovered, which might help in better classification of the data. Each layer of the deep network has inputs and outputs and the number of inputs of each layer is dependent on the number of outputs of its predecessor. If the input of the $k_{th}$ layer is defined as $x_k$, then its output can be written as:

$$a_k = \sigma_k\left(b_k + \sum_{i=1}^{i=N} w_{k,i} x_{k,i}\right) \tag{1.40}$$

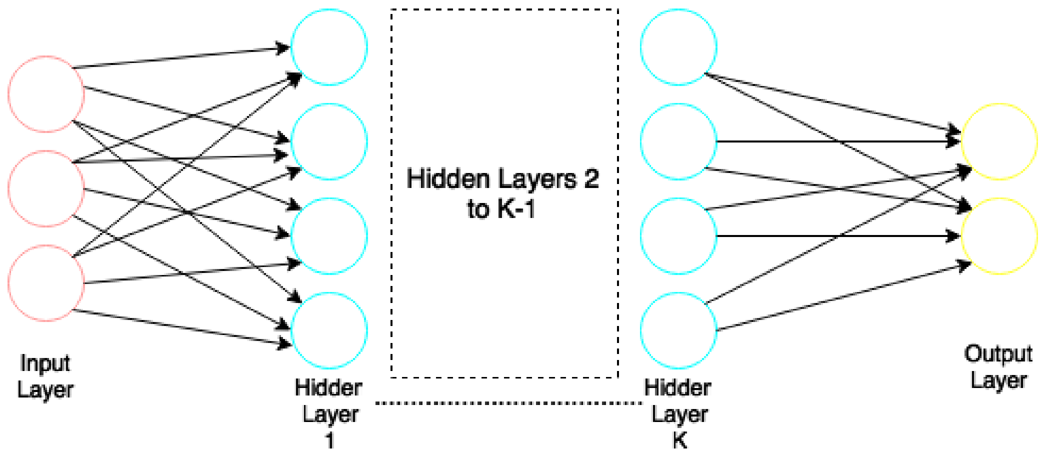


Figure 1.5: Deep Neural Networks with K hidden layers. Only the first and last hidden layers are shown in the figure, but as many as needed can be added to the architecture. Each node in each hidden layer is connected to each node in the following hidden layer.

Note that different activation functions may be used for different layers, hence $\sigma_k$ is given for each layer.

### 1.3.4 Autoencoders

Autoencoders are a special class of neural networks designed to find an efficient representation of data by learning correlations between the data points. This type of network is often used for denoising data. [Vincent et al., 2010] The inputs and outputs of an autoencoder are the same. The autoencoder has two parts:

1. Encoder or the hidden layer which finds the desired representation for the data.
$$y = \sigma_1 \left( b_1 + \sum_{i=1}^{i=N} w_{1,i} x_i \right) \tag{1.41}$$
This allows the network to find a representation of the input data based on the correlations between input-data.

2. Decoder or the output layer, which reconstructs the input from the representation found in the encoding stage.
$$\hat{x} = \sigma_2 \left( b_2 + \sum_{i=1}^{i=N} w_{2,i} y_i \right) \tag{1.42}$$

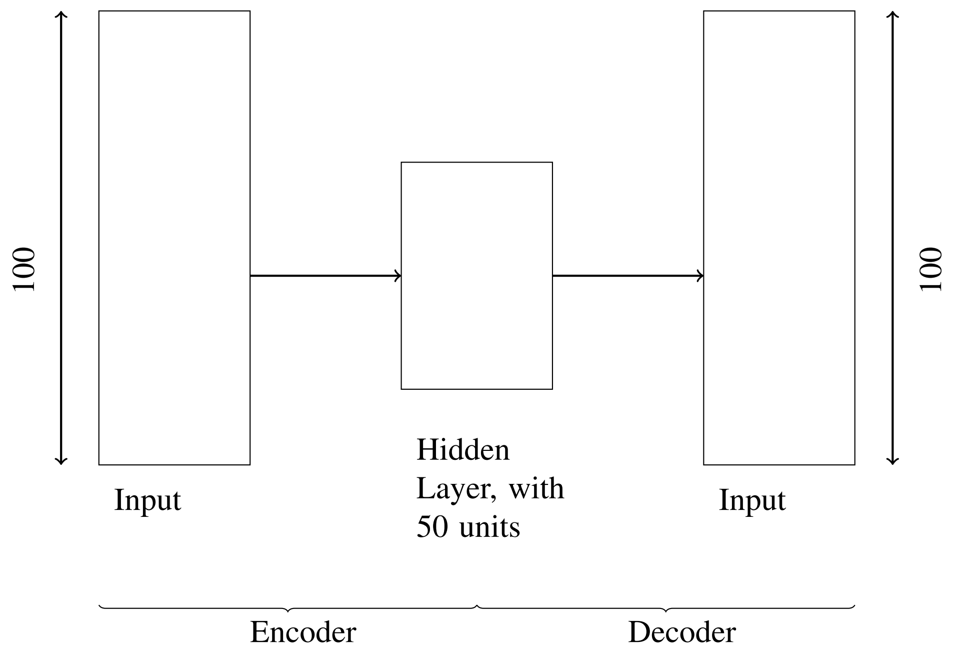


Figure 1.6: Autoencoder with bottleneck

This stage of the network uses the activations of the encoder stage to regenerate the input, thereby learning a function $h_{W,b}(x) \approx x$, where $W$ and $b$ are parameters of the network.

Autoencoders are particularly useful for data compression. For example, if the input vector has 100 values and the hidden layer has 50 nodes, then the autoencoder tries to learn a 50-point representation of the input based on the correlations between the 100 input values. This is also referred to as a bottle-neck. [Sainath et al., 2012] Figure 1.6 shows the structure of such an autoencoder. Note that if each point in the input comes from an an independent Gaussian, i.e. the input data is completely random, then the autoencoder will not be able to learn any compressed representation. The autoencoder described above relies on the bottleneck for data compression, other types of autoencoders have also been desgined, including:

1. Sparse Autoencoder: This type of autoencoder has a higher number of units in the hidden encoding layer, but with a a sparsity constraint imposed on the activations of the hidden layer. This means that for a given input, only a small number of units will be activated in the hidden layer. The sparsity cost is imposed as follows:

$$\sum_{j=1}^{s2} \rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}_j} \tag{1.43}$$

Where

$$\hat{\rho}_j = \frac{1}{m} \sum_{1=1}^{m} [a_j^{(2)} x^{(1)}] \tag{1.44}$$

Where $a_j^{(2)}$ represents the $j^{th}$ activation of the hidden layer.

2. Contractive Autoencoder: For this autoencoder, learning is enforced by adding to the network loss the square root of the Frobenius norm of Jacobian of the hidden layer with respect to input values, as shown in equation 1.45. This value represents the change in the hidden layer representation with respect to a change in the input. A low value means that a change in the input value leads to a small change in the hidden representation. Adding this term to the loss forces the network to learn a useful representation from the input data.

$$\|\nabla x^{(t)} h(x^{(t)})\|_F^2 = \sum_j \sum_k \left( \frac{\partial h(x_j^{(t)}}{\partial x_k^{(t)}} \right)^2 \tag{1.45}$$

### 1.3.5 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a variant of neural networks inspired by the human visual cortex [Hubel and Wiesel, 1968] and have been widely used in image processing. [Fleet et al., 2014] CNNs exploit the local spatial correlation among input neurons from an image by using local receptive fields. For illustrative purposes, the input layer, instead of being a one-dimensional layer of neurons can be thought of as a two-dimensional matrix, as is the case with an image file. For images, the values of this two-dimensional matrix represent the pixel intensities at the coordinate index of the matrix. Now, in a normal neural network, each of these input neurons would be connected to each of the neurons of the first hidden layer, thus forming a fully connected layer. However, in CNNs, each neuron of the first hidden layer is only connected to a small, local region of the input layer, known as the local receptive field, as shown in figure 1.7.

This local receptive field is moved across the input array to form the hidden layer. The movement can be of different stride lengths. In figure 1.7, a stride of $(1, 1)$. The number of neurons in the hidden layer is dependent on the number of units in the input layer, the stride and the number of units in the local receptive field. If the input is a matrix of dimensions $MxN$, the stride is $(1, 1)$ and the local receptive field is an array of $AxB$ then the hidden layer will have dimensions $(M - A + 1)(N - B + 1)$ since the local receptive field can only move $M - A$ units across and $N - B$ units downwards, if a stride length of $1$ is used. The key feature of CNNs is that the neurons of the hidden convolutional layer share

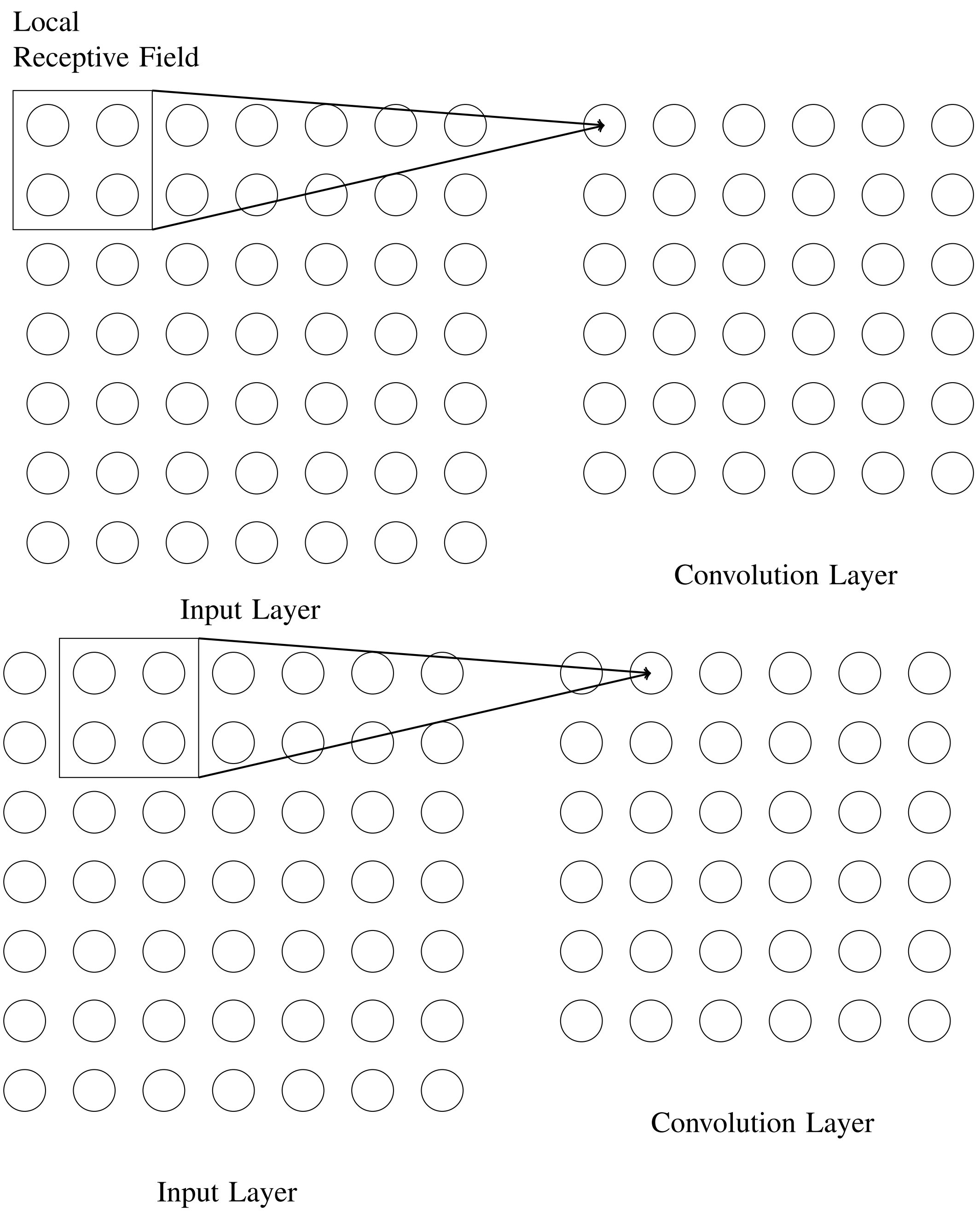


Figure 1.7: Convolutional Neural Networks: Local Receptive Field

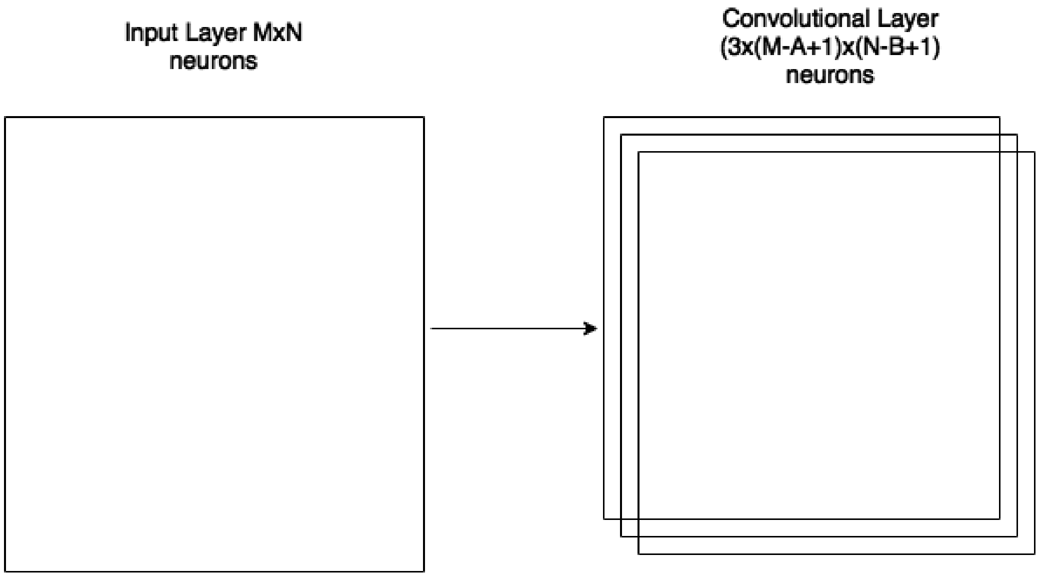


Figure 1.8: Convolutional Neural Networks: 3 $AxB$ filters are convolved across an input of $MxN$ resulting in a layer with 3 kernels of shape $(M - A + 1)x(N - B + 1)$

weights and biases, so that the entire process is akin to a convolution of a matrix of size $AxB$ over a matrix of dimensions $MxN$. This is called weight sharing and is an important concept in convolutional neural networks. In mathematical terms, the output of the $j, k_{th}$ neuron of the convolutional layer can be expressed as:

$$a_{j,k} = \sigma \left( b + \sum_{l=0}^{l=A-1} \sum_{m=0}^{m=B-1} w_{l,m} x_{j+l,k+m} \right) \tag{1.46}$$

In other words, the convolutional layer computes the activation of a feature of dimensions $AxB$ across different regions of the input layer. The mapping from the input layer to the convolutional layer is often called a feature map and the shared weights and the bias unit are termed as the kernel. Since each of these kernels is detecting one feature over the input layer, the convolutional layer usually includes more than one kernel or feature map, as shown in fig 1.8. Convolutional neural networks have extensively been used for image classification including the MNIST handwritten digit set. [Krizhevsky et al., 2012a] One of the biggest advantages of using CNNs is that memory and resources required are lower than those used by a regular fully connected neural network, as the weights and biases across hidden layers are shared.

To implement more than one convolutional layer and find a more abstract representation of the features, pooling layers are often included in CNNs, generally after the convolutional layer. The pooling layer compresses information stored in the convolutional layer, by taking a statistical feature of a local region of neurons in the convolutional layer. The most popular pooling layer is the Max-Pooling layer, which just takes the maximum of a local region of neurons in the convolutional layer.

Apart from Max-Pooling, L2 pooling is also quite frequently used. In this type of pooling, the square root of the sum of the squares of a region of neurons is taken, thereby compressing the information stored within those neurons.

### 1.3.6 Recurrent Neural Networks

Since audio signals have temporal context, the neural network must be given some memory to add contextual information from the past. One solution to do this is to use a recurrent neural network. In this case, the hidden layer is connected to itself, with a weight applied to the output of time $t-1$ added to the function at time $t$. The input of the layer at time t thus becomes:

$$J(j,t) = \sum X(i,t)W_1(i,j) + b_1 W_1(i+1,j) + W_2 H(J(j,t-1)) \tag{1.47}$$

Where $H(J(j,t-1))$ represents the hypothesis of the network at time $t-1$. This function has infinite memory since each the output at each time step is given an equal weight. The gradient in this case has often been observed to explode (reach a high value) or vanish (reach 0). [Pascanu et al., 2012] Therefore, despite the theoretical usefulness of RNNs, practical implementation can be quite difficult. This problem particularly affects the lower layers of a deep neural network; while the higher layers are learning information through successive iterations, it has been observed that the gradient for the lower layers often goes down to zero and therefore further information on these layers is often not learned. One simple solution to this problem is to restrict the memory of a node to a few inputs, thereby reducing the time context that the node uses for learning. RNNs therefore are useful for modeling short-term temporal dependencies but fall short on modeling long-term dependencies. It has been found that proper initialization of parameters helps in alleviating these problems. [Bengio, 2009]

LSTMs or Long Short Term Memory Networks offer an elegant solution to the memory predicament by using gates to decide when the information learned from an input is useful to save in the context of the learning and when it is not. [Stollenga et al., 2015] The gate parameters are also learned during the training phase.

The gates are illustrated in the figure above. In this particular case, the hypothesis from the previous time frame is $h(t-1)$ and the input vector is $x(t)$. The LSTM, at any point of time has a cell state denoted by $C_{t-1}$ which decides whether an input is to be remembered or not.

A closer look at the flow of $x(t)$ and $h(t-1)$ reveals that the input and prior hypothesis passes through an activation function, $\sigma$ in this case and is multiplied with $C_{t-1}$. This activation function determines the contribution of the previous state to the current hypothesis, a value of zero would result in null contribution whereas a value of one would lead to a full contribution. For understanding, this term can be represented as

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{1.48}$$

The second phase of the LSTM decides whether the new information from the input needs to be saved for future use by updating the cell state. To do this, $x(t)$ and $h(t-1)$, are passed through two activation functions, $\sigma$ and $tanh$, leading to two values:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{1.49}$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C) \tag{1.50}$$

These values are thus used to update the cell state as:

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \tag{1.51}$$

Finally, the output of the current time state is calculated by using another activation function, hence determining the amount of the decided state that should influence the output at the time:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{1.52}$$

$$h_t = o_t tanh(C_t) \tag{1.53}$$

A detailed explanation of LSTMs is beyond the scope of this thesis, but an interesting blog explaining the same can be found at http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

### 1.3.7 Training Neural Networks

Neural networks need to be trained before they can be useful. To do this, a training set is usually prepared with inputs $x$ and labeled outputs $y$. The set of inputs can be represented as a vector $X$ and the outputs as $Y$. For training, the inputs are passed through the network to obtain the output of the network, $\hat{y}$. Note that the dimensions of the output layer of the network must match the dimension of the desired output, $y$. Following this, an objective or loss function is computed, representing the difference between $y$ and $\hat{y}$. Although various types of difference functions maybe used, the most commonly used are the Euclidean distance :

$$J(\theta) = ||y - \hat{y}||^2 \tag{1.54}$$

and the KL divergence:

$$J(\theta) = \sum \left( ylog\frac{y}{\hat{y}} - y + \hat{y} \right) \tag{1.55}$$

Where $J(\theta)$ is the objective function parameterized by the parameters $\theta$ of the network, the biases of the layers, $B$ and the weight matrix, $W$. For each input, $x$ in $X$, the parameters are updated in the opposite direction of the gradient of the objective function, $\nabla_\theta J(\theta)$. For any curve, the gradient of the curve represents the direction of the curve going forward. Hence, a step opposite to this direction would ideally lead to a global or local minima. [Hassoun, 1995] This procedure is called gradient descent and has three variants: (http://sebastianruder.com/optimizing-gradient-descent/)

1. Batch gradient descent, which computes the gradient of the objective function with respect to the parameters for the entire set of inputs before updating. The update is done as:

   $$\theta :\Rightarrow \theta - \eta.\nabla_\theta J(\theta) \tag{1.56}$$

   Where $\eta$ represents the learning rate or the amount by which a step should be taken opposite to the direction specified by the gradient. This update is repeated over the entire training set for a pre-defined number of epochs. The batch gradient descent algorithm is guaranteed to find a global minima for convex error surfaces, but will stick to a local minima for non-convex problems.

2. Stochastic gradient descent (SGD) is similar to the batch gradient descent algorithm, but performs a parameter update after each training example $x_i$ of training set $X$.

   $$\theta :\Rightarrow \theta - \eta.\nabla_\theta J(\theta; x_i; y_i) \tag{1.57}$$

   SGD is faster than the batch descent algorithm and also avoids redundant calculations that might be involved in the later. However, it too converges to a local minima for non-convex error surfaces and is prone to overshooting as updates are performed frequently with high variance.

3. Mini-batch gradient descent combines the best of both the SGD and the batch gradient algorithms by performing an update after each mini batch of size $n$ of the input set $X$.

   $$\theta :\Rightarrow \theta - \eta.\nabla_\theta J(\theta; x_{i:i+n}; y_{i+n}) \tag{1.58}$$

   This leads to a lower variance of updates and a more stable convergence. It also allows the use of optimized matrix operations making the mini-batch algorithm faster than the other two algorithms. Note that the mini-batch algorithm is also often referred to as SGD.

### Optimizing Gradient Descent Algorithms

While the aforementioned algorithms are good for optimizing networks for convex objective functions, they often get stuck at local minima while handling non-convex problems. [Dauphin et al., 2014] Using momentum [Qian, 1999] provides an elegant solution to this problem. This method adds a fraction of the update of the last time step to the current upgrade:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{1.59}$$

$$\theta :\Rightarrow \theta - v_t \tag{1.60}$$

The momentum term, $\gamma$ increases the update when the two updates are in the same direction and decreases the update when the two terms are in opposite direction. This reduces oscillation in the descent leading to faster convergence. $\gamma$ is usually set to 0.9 Other similar ideas for optimization of gradient descent algorithms are presented below:

1. Nesterov accelerated gradient (NAG) is a method which approximates the values of the parameters after an update before performing an update. In other words, it performs a look ahead to see if the update is going in the right direction before making the update. mathematically, this can be represented as:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{1.61}$$

$$\theta :\Rightarrow \theta - v_t \tag{1.62}$$

2. Adagrad [Duchi et al., 2011] adapts the learning rate for different parameters in the parameter set. Parameters which are updated frequently use a smaller learning rate, while infrequently updated parameters are updated using a higher learning rate. Defining $g_{t,i}$ to be the gradient of the loss function with respect to parameter $\theta_i$ of the parameter set $\theta$ at time t, the update process can be written as:

$$g_{t,i} = \nabla_\theta J(\theta_i) \tag{1.63}$$

$$\theta_{t+1,i} :\Rightarrow \theta_{t,i} - \eta . g_{t,i} \tag{1.64}$$

The learning rate is updated each time for each parameter $\theta_i$ using the past gradients used to modify the parameter:

$$\theta_{t+1,i} :\Rightarrow \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} . g_{t,i} \tag{1.65}$$

$G$ is a diagonal matrix where each diagonal element $G_{i,i}$ is the sum of the squares of the gradients with respect to $\theta_i$ up to the $t_t h$ time step. $\epsilon$ is a small term used to smoothen the descent and avoid division by zero. Adagrad does not require the setting of the learning rate manually and has provided good results in various image and video processing problems. [Coates et al., 2011] However, since the gradients are accumulated in the denominator, the learning rate for parameters which are frequently updated goes down steeply and might at some time be too small for the network to gain further knowledge.

3. Adadelta [Zeiler, 2012] is an algorithm similar to adagrad which seeks to resolve the problem of diminishing learning rate. This is done by restricting the number of accumulated gradients to a window $w$. The algorithm uses the running average $E[g^2]_t$ of past $w$ squared gradients:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{1.66}$$

The update is done using the root mean squares (RMS) of parameter updates and the gradient

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \tag{1.67}$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} \tag{1.68}$$

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon} \tag{1.69}$$

$$\theta_{t+1} = \theta_t - \frac{RMS[\Delta\theta]_t}{RMS[g]_t} g_t \tag{1.70}$$

**Backpropogation**

The learning techniques described in the previous section use the gradient of the objective function to optimize the parameters of the network. However, finding the gradient of the network is not a trivial task. Backpropogation is one of the techniques used to find the gradient of an objective function of a neural network. Each input data point is first fed forward through the network to obtain activations for each of the layers. Then partial derivatives are estimated for each layer using the activation function as:

$$\frac{\partial J(\theta)}{\partial \theta_{ij}^l} = \frac{\partial J(\theta)}{\partial s_j^l} \frac{\partial s_j^l}{\partial \theta_{ij}^l} \tag{1.71}$$

The derivative of the activation function with respect to the weights, $\frac{\partial s_j^l}{\partial \theta_{ij}^l}$ can easily be computed as $h_i^{l-1}$. but the derivative of the cost, $\frac{\partial J(\theta)}{\partial s_j^l}$ as $\delta_j^l$ requires some mathematical manipulation.

The intuition here is that the term $\delta_j^l$ represents, to some degree, the contribution of a node $j$ of layer $l$ to the overall loss. Thus to calculate this, loss contribution is calculated for the last layer as:

$$\delta_j^{L-1} = \frac{\partial J(\theta)}{\partial s_j^{L-1}} = \frac{\partial J(\sigma^{L-1} s_j^{L-1}, y_j)}{\partial s_j^{L-1}} \tag{1.72}$$

Once calculated, $\delta_j^{L-1}$ can be backpropogated through the network to find the loss contribution of the other layers:

$$\delta_i^{l-1} = \sigma'^{(l-1)}(s_i^{l-1}) \sum_{j=1}^{d(l)} \delta_j^l \theta_{ij}^l \tag{1.73}$$

This is in turn used to find the derivatives of the activation function as given by equation 1.71

### 1.3.8 Initialization Of Neural Networks

For non-convex error surfaces, initialization of weights of the various layers in a neural networks is very important to ensure that the optimization process does not land on a local minima. [Xavier Glorot, ] Furthermore, for deep neural networks with more than one hidden layer, if the weights are initialized to very small values, the variance of the input diminishes rapidly as it is propagated through the layers. As a result, the network does not efficiently learn. Similarly, if the the variance increases rapidly if the weights are initialized to high values also leading to inefficient learning. To prevent these problems, it is important to ensure that the variance remains similar across the layers. Or, in other words, the variance of the input of a layer and its output should be the same. Xavier Golot et al. [Xavier Glorot, ] proposed an efficient method for initialization of layers, described below.

The variance of the output $y$ of a layer $l$ of the network can be given as:

$$var(y) = var(x_1 w_1 + x_2 w_2 ......x_n w_n) \tag{1.74}$$

Note that the variance of the bias node, $b$ is 0 since it is a constant. The individual terms of this equation can be written as:

$$var(x_i w_i) = E(x_i)^2 var(w_i) + E(w_i)^2 var(x_i) + var(w_i) var(x_i) \tag{1.75}$$

Where $E(x)$ represents the expectation value of $x$ or the mean value. If the layer is initalized with a Gaussian with zero mean, then equation 1.75 can be rewritten as:

$$var(x_i w_i) = var(w_i)var(x_i) \tag{1.76}$$

Therefore, equation 1.74 can be written as:

$$var(y) = var(x_1)var(w_1)+var(x_2)var(w_2)......var(x_n)var(w_n) = Nvar(x_1)var(w_1) \tag{1.77}$$

For the layers to have equal variance, $Nvar(x_1)$ should be equal to 1. Therefore, the paper suggests initializing the weights with a Gaussian with zero mean and variance given by:

$$var(w) = \frac{1}{N_{avg}} \tag{1.78}$$

where

$$N_{avg} = \frac{N_{in} + N_{out}}{2} \tag{1.79}$$

Where $N_{in}$ is the number of input nodes of a layer and $N_{out}$ is the number of output weights of the layer.

## 1.4 Neural Networks For Source Separation

This section provides a description of some of the techniques which have been proposed in the recent years applying neural networks for source separation:

### 1.4.1 Using Deep Neural Networks

[Uhlich et al., 2015] and [Nugraha et al., 2016] have proposed systems using Deep Neural Networks for source separation. While the former focuses on instrument based separation, the later uses neural networks for multi-channel sound source separation.

#### Instrument Based Separation

[Uhlich et al., 2015] designed a deep neural network system for source separation with five fully connected layers, with ReLU nodes. The input vector, $x$, for the network consists of the fast Fourier transform of recordings of two sets of three instruments: a horn, a piano and a violin and a bassoon, a piano and a trumpet. These recording were done with a sample rate of 32kHz. Each input frame of length $L$ was concatenated with $C$ succeeding and preceding frames, with no overlap, leading to a total vector length of $(2C + 1)L$, so that the network could model temporal context. With $C = 3$ and $L = 513$, the input vector had 3591 elements, covering 224 milliseconds of the audio input. The input vector, $x$, was normalized by $\gamma$, a scalar representing the Euclidean norm of the $2C + 1$ frames. This ensures that the input vector is independent of amplitude variations which may occur in different recordings.

The network was trained layer-wise for 600 iterations using the Limited-memory Broyden Fletcher Goldfarb Shanno (L-BFGS) algorithm and for an additional 3000 iterations for fine tuning. Thus for each layer $k$ of the network, the following optimization was performed.

$$\{W_k^{Init}, b_k^{Init}\} = argmin_{Wk,bk} \sum_{p=1}^{P} \|s^{(p)} - (W_k x_k^{(p)} + b_k)\|^2 \tag{1.80}$$

Where $s^{(p)}$ represents the $p^{th}$ source to be separated.

### Multichannel Source Separation

[Nugraha et al., 2016] trained a DNN for source separation using multichannel audio input. The researchers used a single time-frame for training, thus focusing entirely on spectral characteristics of a single frame, without taking temporal evolution into account. Training was done using the ADADELTA algorithm and five different cost measures were tested for optimization of the network.

1. Itakura-Saito divergence, which is widely used in the speech processing community and focuses on perceptual quality.

$$D_{IS} = \frac{1}{JFN} \sum_{j,f,n} \left( \frac{|\tilde{c}_j(f,n)|^2}{v_j(f,n)} - log \frac{|\tilde{c}_j(f,n)|^2}{v_j(f,n)} - 1 \right) \tag{1.81}$$

   Where $J$ represents the number of sources, $c$ the short-time Fourier transform of the sources, $N$ is the number of time frames, $F$ is the number of frequency coefficients and $v_j$ denotes the power spectral density of the $j_{th}$ source.

2. The Kullbak-Leibler (KL) divergence:

$$D_{KL} = \frac{1}{JFN} \sum_{j,f,n} \left( |\tilde{c}_j(f,n)| \frac{|\tilde{c}_j(f,n)|}{\sqrt{v_j(f,n)}} - |\tilde{c}_j(f,n)| + \sqrt{v_j(f,n)} \right) \tag{1.82}$$

3. The Cauchy cost function

$$D_{Cau} = \frac{1}{JFN} \sum_{j,f,n} \left( \frac{3}{2} log(|\tilde{c}_j(f,n)| + v_j(f,n)) - log\sqrt{v_j(f,n)} \right) \tag{1.83}$$

4. The phase sensitive cost function

$$D_{PS} = \frac{1}{2JFN} \sum_{j,f,n} |m_j(f,n)\tilde{x}(f,n) - \tilde{c}_j(f,n)|^2 \tag{1.84}$$

   where $m_j(f,n) = \frac{v_j(f,n)}{\sum_{j'} v_{j'}(f,n)}$ is the single channel Wiener filter for each source.

5. The mean square error (MSE)

$$D_{MSE} = \frac{1}{2JFN} \sum_{j,f,n} (|\tilde{c}_j(f,n)| - \sqrt{v_j(f,n)})^2 \tag{1.85}$$

The researchers concluded that the Kullbak-Leibler (KL) divergence had the best performance for the source separation task, followed closely by the mean-square divergence.

### 1.4.2 Using Recurrent Neural Networks

Po-Sen Huang et al. [Huang et al., 2014] recently proposed a methodology using a deep recurrent neural network for separating singing voice from single channel musical recordings. The proposed model, shown in figure 1.9, takes as an input a vector of 513 values, corresponding to the magnitude spectrum of a 1024-point FFT of the mixture of voice and accompaniment.

This input $x_t$ is fed through three hidden layers, shown in figure 1.9 as $h_t^1$, $h_t^2$ and $h_t^3$. The second hidden layer, $h_t^2$ is a recurrent layer, with activation at time $t$ given by:

$$h_t^2 = f_h(x_t, h_{t-1}^2) = \sigma_2(U^2 h_{t-1}^2 + W^2(W^1 \sigma_1 x_t)) \tag{1.86}$$

Where $W^l$ is the weight matrix associated with a layer $l$ and $U^l$ is the weight matrix for the recurrent connection at the $l-th$ layer. The output of the third hidden layer is used to generate two vectors $\hat{y}_{1t}$ and $\hat{y}_{2t}$, of the same size as the input vector. The output predictions are used to calculate a soft time-frequency mask as:

$$m_t(f) = \frac{|\hat{y}_{1t}(f)|}{|\hat{y}_{1t}(f) + \hat{y}_{2t}(f)|} \tag{1.87}$$

The estimated mask is then applied to the input mixture signal to estimate the voice, $\tilde{y}_1$ and the accompaniment, $\tilde{y}_2$ using the equations:

$$\tilde{y}_1(f) = m_t(f) x_t(f) \tag{1.88}$$

$$\tilde{y}_2(f) = (1 - m_t(f)) x_t(f) \tag{1.89}$$

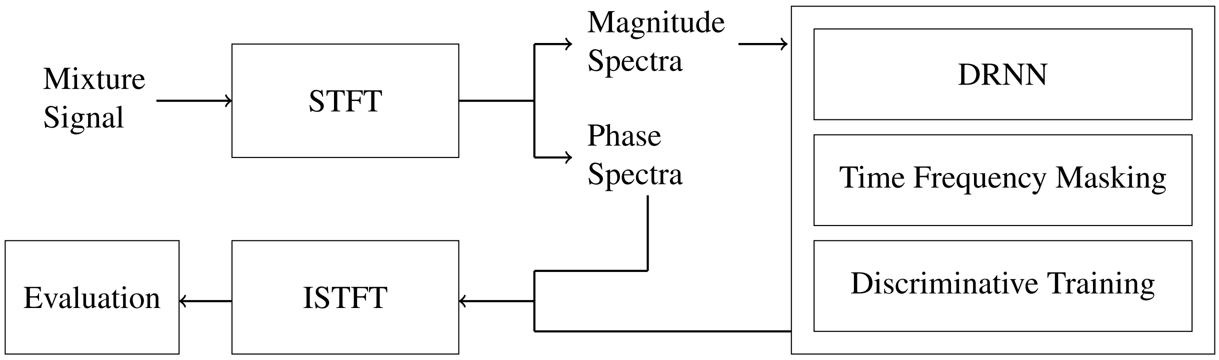


Figure 1.9: Framework proposed by [Huang et al., 2014]

To train the network, the researchers used a an objective function taking into account the similarity between target sources and the corresponding estimated target sources as well as dissimilarity between the estimated sources and the other target source. This objective function is shown in equation 1.90

$$||\hat{y}_{1t} - y_{1t}|| - \gamma||\hat{y}_{1t} - y_{2t}|| + ||\hat{y}_{2t} - y_{2t}|| - \gamma||\hat{y}_{2t} - y_{1t}|| \tag{1.90}$$

The model was tested on the MIR-1K [Jang, 2010] dataset, which contains a thousand song clips with durations ranging from 4 to 13 seconds, encoded with a sample rate of 16KHz. These song clips were extracted from Chinese karaoke recordings, performed by two amateur singers, male and female. The model was trained for a maximum of 400 epochs, from random initialization. To increase the training samples, the researchers did a circular shift of the singing voice and mixed the voice with the accompaniment. The researchers noted that the short time Fourier transform provided better results than using log-mel filterbank features or log power spectrum.

## 1.5 Convolutional Neural Networks In Image Processing

Convolutional neural networks have been very effective in image processing fields like object identification, etc One of the most interesting applications of convolutional neural networks in the field is for automatic image colorization of grayscale images. [Satoshi Iizuka et al., 2016] The researchers implemented a three level convolutional network, with a grayscale image as input, as shown in fig 1.11. The network can be scaled to work with images of any size, represented here by height $H$ and Width $W$. For illustrative purposes, the input image is assumed to be of size 224x224. The layers of the network are

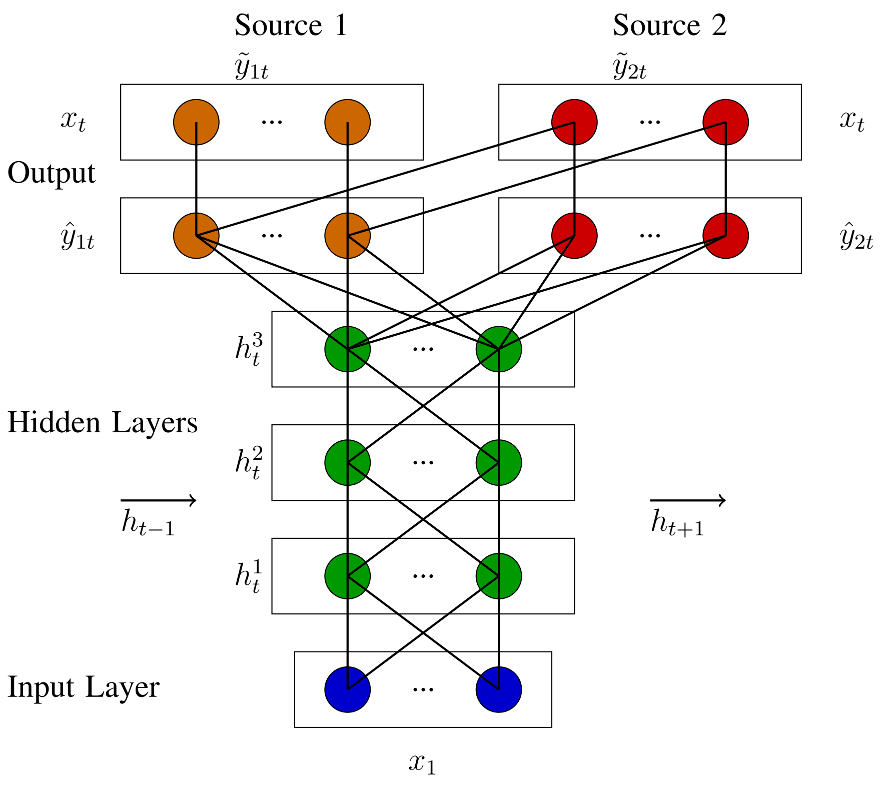


Figure 1.10: Neural Network Architecture proposed by [Huang et al., 2014]. The hidden layer $h_t^2$ is a recurrent layer

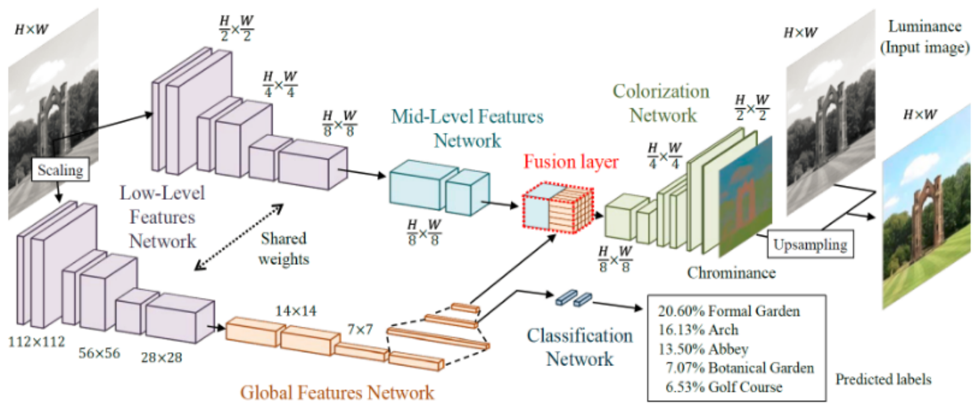


Figure 1.11: Network Architecture for Image Colorization, extracted from [Satoshi Iizuka et al., 2016]

1. Low level features: this is a sub-network with 6 convolutional layers, which computes low level features within an image, i.e.: performs convolution over small areas of the network ($3x3$ filters) to detect features within a small area of the input image. This layer feeds to both the layer for Mid Level features and Global Level Features. The six layers are listed in the table below:

| Type | Kernel | Stride | Number Of Filters |
|---|---|---|---|
| Conv | $3x3$ | $2x2$ | 64 |
| Conv | $3x3$ | $1x1$ | 128 |
| Conv | $3x3$ | $2x2$ | 128 |
| Conv | $3x3$ | $1x1$ | 256 |
| Conv | $3x3$ | $2x2$ | 256 |
| Conv | $3x3$ | $1x1$ | 512 |

The final output of the low level layer consists of 512 filters, with a size of $\frac{H}{8}x\frac{W}{8}$. For an input of $224x224$, this gives a layer of size $28x28$

2. Mid Level Features: These features are computed using two convolutional layers on the low level features. This is done by bottlenecking the 512 features from the output of the low level features 256 mid level features, forcing the network to learn a compact representation of the image. Since only convolutional layers are used up to this point, the image can be regenerated entirely from this representation.

| Type | Kernel | Stride | Number Of Filters |
|---|---|---|---|
| Conv | $3x3$ | $1x1$ | 512 |
| Conv | $3x3$ | $1x1$ | 256 |

3. Global Image Features: The low level features are processed by four convolutional layers and three fully-connected layers, as shown in the table below

| Type | Kernel | Stride | Number Of Filters |
|---|---|---|---|
| Conv | $3x3$ | $2x2$ | 512 |
| Conv | $3x3$ | $1x1$ | 512 |
| Conv | $3x3$ | $2x2$ | 512 |
| Conv | $3x3$ | $1x1$ | 512 |
| Fully Connected | - | - | 1024 |
| Fully Connected | - | - | 512 |
| Fully Connected | - | - | 256 |

This leads to a 256 point compact representation of the input data.

4. Fusion Layer: The mid level layer and the global layer are fused together in the fusion layer, leading to a layer which incorporates information from both these layers. The output of the layer can be represented as:

$$y_{u,v}^{fusion} = \sigma \left( b + W \left| \begin{array}{c} y^{global} \\ y_{u,v}^{mid} \end{array} \right| \right) \tag{1.91}$$

5. Colorization Network: From the fusion layer, a colored image is reconstructed using convolutions and upsampling:

| Type | Kernel | Stride | Number Of Filters |
|---|---|---|---|
| Fusion | - | - | 256 |
| Conv | $3x3$ | $1x1$ | 128 |
| Upsample | - | - | 128 |
| Conv | $3x3$ | $1x1$ | 64 |
| Conv | $3x3$ | $1x1$ | 64 |
| Upsample | - | - | 64 |
| Conv | $3x3$ | $1x1$ | 32 |
| Output | $3x3$ | $1x1$ | 2 |

This leads to an output, representing the chrominance, half the size of the input. This chrominance is combined with the input image to produce the color image.

For training, the Mean Square Error function is used. Color images are first converted to greyscale and input through the network to produce normalized CIR L*a*b components, which are then compared with the original chrominance components to calculate the loss to optimize the network. Further, the model also classifies the objects seen in the image, through the Global Features Layer. This allows the model to learn context from the input image.

# Chapter 2

# METHODOLOGY

The model presented in this thesis tries to incorporate the underlying idea of NMF, presented in section 1.1.2. The basis vector $B$ learned in the NMF technique represents the timbrel structure of the instrument in question across one time frame, while the activation gains $G$ represent the evolution of this structure across time. The NMF model tries to learn the basis vector for an instrument across the whole spectrum, therefore for each note of an instrument, a separate basis vector has to be learned. However, using convolutional neural networks, smaller, robust timbrel structures can be learned across smaller subsections of the spectrogram.

Convolutional networks have the powerful ability to map features from a given input. For Images, square shaped filters are usually used as a certain symmetry is expected in a image across both the X and Y axis. However, for audio data, the information contained across frequency bins is not the same as that across the time axis. Therefore, we decided to have two convolutional layers in the model. The first to model timbrel features along just the frequency axis and the second to model the evolution of these features across time. The general model architecture is presented in figure 2.2 and is similar to the one presented by suggested by [Pons et al., 2016]. The aim of the network is to learn robust timbrel models for generalized classes of instruments, in this case, voice, bass and drums. Further details are presented the following section.

## 2.1 Proposed framework

The diagram for proposed source separation framework is presented in Figure 2.1. The Short-Time Fourier Transform (STFT) is computed on a segment of time context $T$ of the mixture audio. The resulting magnitude spectrogram is then passed through the CNN, which outputs an estimate for each of the separated sources. The estimate is used to compute time-frequency soft masks, which are

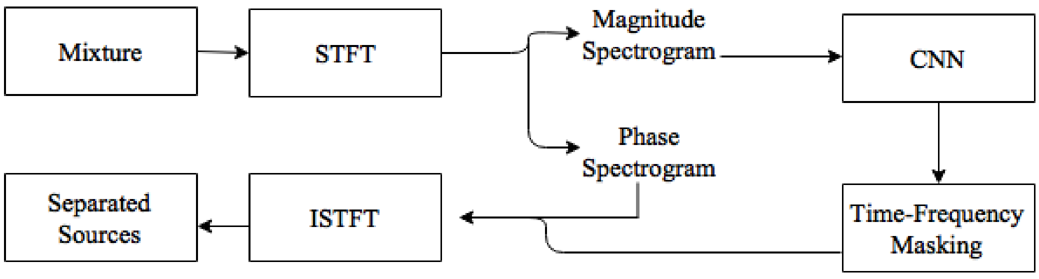


Figure 2.1: Data Flow

applied to the magnitude spectrogram of the mixture to compute final magnitude estimates for the sources. These estimates, along with the phase of the mixture, are used to obtain the audio signals corresponding to the sources.

### 2.1.1 Convolution stage

This part of the network consists of two convolution layers and a max-pool layer.

1. Timbre Layer: This convolution layer has the shape $(t_1, f_1)$, spanning across $t$ time frame and taking into account $f_1$ frequency bins. This layer tries to capture local timbre information, allowing the model to learn timbre features. These features are shared among the sources to be separated, contrary to the NMF approach, where specific basis and activation gains are derived for each source. Therefore, the timber features learned by this layer need to be robust enough to separate the required source across songs of different genres where the type of instruemnts and singers might vary. $N_1$ filters were used in this layer.

2. Max Pool Layer: This layer compresses information determined in the first layer across frequency and time to attain a compact data representation. The models are tested without pooling, with pooling across frequency and with pooling across both frequency and time dimensions. The dimensions of the pooling layer are represented as $p_t, p_f$.

3. Temporal layer: This layer learns various types of temporal evolution for different instruments from the features learned in the Timbre Features layer. This is particularly useful for modeling time-frequency characteristics of the different instruments present in the sources to be separated. Again, the aim is to learn a robust representation for a general class of instruments. The filter shape of this layer is $(t_2, f_2)$ and $N_2$ filters were used.

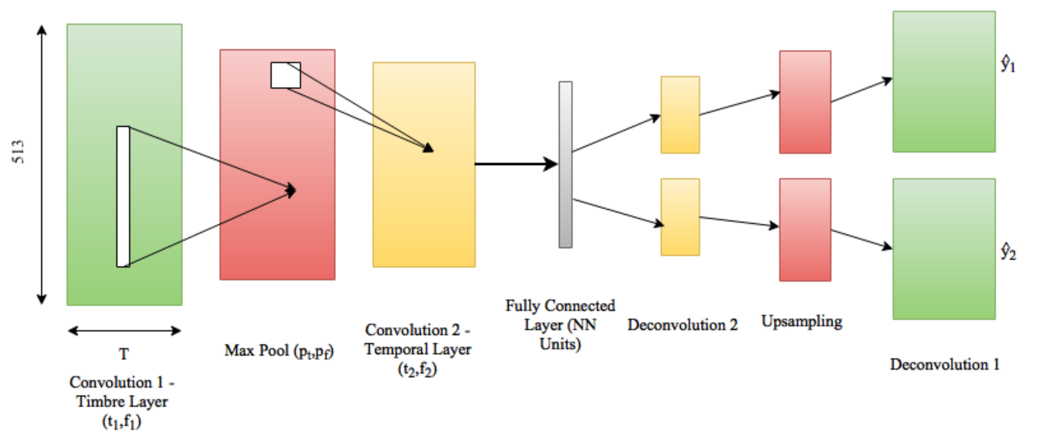

513
T
Convolution 1 - Timbre Layer (t1,f1)
Max Pool (pt,pf)
Convolution 2 - Temporal Layer (t2,f2)
Fully Connected Layer (NN Units)
Deconvolution 2
Upsampling
Deconvolution 1
ŷ1
ŷ2

Figure 2.2: Network Architecture for Source Separation, Convolution Stage

Note that none of these layers have non-linearities and only convolutional layers are used up to this point. This leads to a compact representation of the input mixture, which can be fed to a fully connected layer for learning.

### 2.1.2 Fully Connected Layer

This is a fully connected ReLU layer which acts as a bottleneck, achieving dimensionality reduction [Sainath et al., 2012]. This layer consists of a non-linear combination of the features learned from the previous layers, with a ReLU non-linearity. The layer is chosen to have fewer elements to reduce the total parameters of the network and to ensure that the network does not over-fit the data and is able to produce a robust representation of the data. The number of nodes in the model is represented as $NN$

### 2.1.3 Deconvolution network

The output of the first fully connected layer to another fully connected layer, with a ReLU non-linearity and the same size as the output of the second convolution layer. Thereafter, this layer is reshaped and passed through successive deconvolution and up-sampling layers, the inverse operations in the convolution stage. This process is repeated to compute estimates, $\hat{y}_{nt}$, for each of the sources, $y_{nt}$.

### 2.1.4 Time-frequency masking

As advocated in [Huang et al., 2014], it is desirable to integrate into the network the computation of a soft mask for each of the sources. From the output of the network $\hat{y}_{nt}(f)$, we can compute a soft mask as follows:

$$m_at(f) = \frac{|\hat{y}_{at}(f)|}{\sum_{n=1}^{N} |\hat{y}_{nt}(f)|} \tag{2.1}$$

where $\hat{y}_{nt}(f)$ represents the output of the network for the $n^{th}$ source at time $t$ and $N$ is the total number of sources to be estimated.

The estimated mask is then applied to the input mixture signal to estimate the sources $\tilde{y}_n$.

$$\tilde{y}_n(f) = m_nt(f)x_t(f) \tag{2.2}$$

Where $x_t(f)$ is the spectrogram of the input mixture signal.

### 2.1.5 Parameter learning

The neural network was trained to optimize parameters using a Stochastic Gradient Descent with AdaDelta algorithm, as proposed in [Zeiler, 2012], in order to minimize the squared error between the estimate and the original source.

$$L_{sq} = \sum_{i=1}^{N} \|\tilde{y}_{nt} - y_{nt}\|^2 \tag{2.3}$$

# Chapter 3

# EVALUATION

## 3.1 Dataset

For training and testing this architecture, the Demixing Secrets Dataset 100 (DSD100) and Mixing Secrets Dataset 100 (MSD100) datasets were used. Both these datasets consist of 100 professionally produced full track songs from the The Mixing Secrets Free Multitrack Download Library and are designed to evaluate signal source separation from music recordings. The datasets contain separate tracks for drums, bass, vocals and other instruments for each song in the set, present as stereo WAV files with a sample rate of 44.1 kHz. The four source tracks are summed together and normalized to create the mixture track for each song. The average duration of a song in the dataset is 4 minutes and 10 seconds.

## 3.2 Training

Given the input mixture spectrogram and the spectrogram of the constituent sources, optimization of the network is done using an Ada-delta algorithm [Zeiler, 2012] in order to minimize the squared error between the estimate and the original source. Lasagne, [Dieleman et al., 2015] a framework for neural networks built on top of [Al-Rfou et al., 2016], was used for data flow and network training on a computer with GeForce GTX TITAN X GPU, Intel Core i7-5820K 3.3GHz 6-Core Processor, X99 gaming 5 x99 ATX DDR44 motherboard.

## 3.3 Testing

Once the network was trained, the song examples from the Test set were passed through it, to separate the mixture into four components: vocals, bass, drums and other. To do this, the STFT of the mixture was computed, using FrameSize of 1024 samples with a $75\%$ overlap. The resulting magnitude spectrogram was split into batches of 30 frames with $50\%$ overlap. The batches were fedforward through the network and the resulting source estimates were overlap-added to produce the magnitude spectrogram for the estimate source. The audio for the respective sources was computed using an ISTFT, with the phase spectrum of the original mixture. For evaluation, the audio files was split into segments of 30 seconds with $50\%$ overlap. The evaluation measures, SDR, SIR, SAR and ISR were computed for these segments.

### 3.3.1 Adjustments To Learning Objective

After some initial experimentation, we observed that we needed to add an additional loss term representing the difference between the estimated sources, as used by [Huang et al., 2014]. In addition, we observed that, across songs from different genres, the instruments other than voice, bass and drums varied a lot and the network was not able to efficiently learn a representation for this category as it tries to learn a general timbre class instead of particularities of the different instruments. To overcome this problem, we used the estimated 'other' source for computing a difference, instead of considering it into the $L_{sq}$. This difference encouraged between sources such as 'vocals' and 'other', 'bass' and 'other' and 'drums' and 'other'. Also, we noted that the 'other' source comprised of harmonic instruments such as guitars and synths, which were similar to the 'vocals' source. To encourage difference between these two sources, a $Ł_{othervocals}$ loss element, which represents the difference between the estimated vocals and the other stem, was introduced.

$$L_{diff} = \sum_{i=1}^{N-1} \|\tilde{y}_{nt} - \tilde{y}_{\hat{n} \neq nt}\|^2 \tag{3.1}$$

$$L_{othervocals} = \|\tilde{y}1t - y_{Nt}\|^2 \tag{3.2}$$

$$L_{other} = \sum_{i=2}^{n-1} \|\tilde{y}_{nt} - y_{Nt}\|^2 \tag{3.3}$$

The total cost is then written as:

$$L_{total} = L_{sq} - \alpha L_{diff} - \beta L_{other} - \beta_{vocals} L_{othervocals} \tag{3.4}$$

$y_{1t}$ represents the source corresponding to vocals and $y_{Nt}$ represents that corresponding to other instruments. The parameters $\alpha$, $\beta$ and $\beta_{vocals}$ were experimentally determined to be $0.001$, $0.01$ and $0.03$ respectively.

| Model | E | T | f1 | N1 | S1 | MP | f2,t | N2 | S2 | Alpha | Beta | Betavoice |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model 1 | 30 | 30 | 450 | 50 | 1,1 | 1,2 | 10,1 | 30 | 1,1 | 0.001 | 0.01 | 0.03 |
| Model 2 | 30 | 30 | 450 | 50 | 1,1 | 1,1 | 10,1 | 30 | 1,1 | 0.001 | 0.01 | 0.03 |
| Model 3 | 30 | 30 | 50 | 30 | 1,5 | 1,2 | 20,5 | 30 | 1,1 | 0.001 | 0.01 | 0.03 |
| Model 4 | 30 | 30 | 50 | 50 | 1,5 | 1,2 | 5,5 | 50 | 1,1 | 0.001 | 0.01 | 0.03 |
| Model 5 | 30 | 30 | 50 | 30 | 1,5 | 1,2 | 5,5 | 50 | 1,1 | 0.001 | 0.01 | 0.03 |
| Model 6 | 30 | 30 | 30 | 50 | 1,5 | 1,2 | 5,5 | 30 | 1,1 | 0.001 | 0.01 | 0.03 |
| Model 7 | 25 | 30 | 30 | 30 | 1,5 | 1,2 | 5,5 | 30 | 1,1 | 0.001 | 0.001 | 0.1 |
| Model 8 | 50 | 30 | 30 | 30 | 1,5 | 1,2 | 5,5 | 30 | 1,1 | 0.001 | 0.01 | 0.01 |
| Model 9 | 50 | 30 | 30 | 30 | 1,5 | 1,2 | 5,5 | 30 | 1,1 | 0.001 | 0.01 | 0.01 |

Table 3.1: Model Parameters

### 3.3.2 Experiments With MSD100 Dataset

Initial experiments were carried out with the MSD100 dataset to test the following parameters:

1. Number of epochs used for training: $E$
2. Time context: $T$
3. Timbrel features layer shape: $f_1$ and $t_2$
4. Number of filters in timbrel features layer: $N_1$
5. Stride for filter: $S_1$
6. MaxPool: $p_t, p_f$.
7. Evolution layer shape: $f_2$ and $t_2$
8. Number of filters in evolution layer: $N_2$
9. Stride for filter: $S_2$
10. $\alpha$
11. $\beta$
12. $\beta_{vocals}$

To this end, experiments were carried out with the parameters shown in Table 3.1

| Model | Measure | Bass | Drums | Vocals | Others |
|---|---|---|---|---|---|
| Model 1 | SDR | 2.4±2.7 | -0.5±1.7 | -1.1±5.2 | 1.0±1.2 |
| | SIR | 7.3±4.4 | 0.8±3.5 | 2.4±6.4 | 3.4±3.2 |
| | SAR | 11.4±3.0 | 0.1±1.6 | 2.8±2.4 | 4.2±2.0 |
| | ISR | 10.5±2.6 | 4.8±2.2 | 7.3±2.7 | 6.0±2.6 |
| Model 2 | SDR | 2.4±2.6 | -0.5±1.7 | -1.3±5.3 | 1.2±1.1 |
| | SIR | 7.3±4.3 | 1.1±3.5 | 2.2±6.6 | 3.5±3.2 |
| | SAR | 11.3±2.9 | 0.2±1.7 | 2.2±6.6 | 3.7±1.9 |
| | ISR | 10.6±2.6 | 4.8±2.2 | 7.6±2.8 | 5.6±2.4 |
| Model 3 | SDR | 2.1±2.8 | -0.7±1.4 | -1.3±5.4 | 1.2±1.1 |
| | SIR | 6.8±4.3 | -0.6±3.3 | 1.8±6.1 | 3.5±3.2 |
| | SAR | 11.9±3.0 | 0.2±1.4 | 3.9±2.2 | 3.9±2.1 |
| | ISR | 10.0±2.2 | 4.8±2.2 | 8.0±2.8 | 5.0±2.3 |
| Model 4 | SDR | 2.2±3.3 | -4.7±4.9 | -0.7±5.7 | -1.0±1.7 |
| | SIR | 5.7±4.2 | -8.4±3.2 | -1.2±5.6 | 1.6±3.9 |
| | SAR | 15.2±2.8 | 10.8±1.4 | 8.7±1.3 | 8.3±1.4 |
| | ISR | 10.7±1.4 | 4.8±2.2 | 6.6±2.5 | 2.3±2.6 |
| Model 5 | SDR | 2.2±2.7 | -0.6±1.6 | -1.2±5.4 | 1.2±1.2 |
| | SIR | 7.0±4.3 | 0.3±3.4 | 2.2±6.5 | 3.6±3.2 |
| | SAR | 11.4±3.0 | 0.1±1.4 | 3.5±2.5 | 3.7±2.0 |
| | ISR | 10.4±2.3 | 4.8±2.2 | 7.7±2.6 | 5.3±2.3 |
| Model 6 | SDR | 2.5±2.7 | -0.7±1.7 | -1.0±5.5 | 1.2±1.3 |
| | SIR | 7.2±4.4 | 0.0±3.3 | 2.4±6.3 | 3.4±3.3 |
| | SAR | 11.2±3.0 | 0.5±1.5 | 3.6±2.5 | 4.2±2.1 |
| | ISR | 10.6±2.3 | 4.8±2.2 | 7.4±2.7 | 5.6±2.6 |
| Model 7 | SDR | 2.3±2.8 | -0.6±1.6 | -1.1±5.2 | 1.1±1.2 |
| | SIR | 7.1±4.4 | -0.3±3.5 | 2.8±6.6 | 3.6±3.2 |
| | SAR | 11.8±3.1 | 0.9±1.3 | 2.9±2.7 | 4.3±2.2 |
| | ISR | 10.4±2.2 | 4.8±2.2 | 7.0±2.7 | 5.5±2.5 |
| Model 8 | SDR | 2.2±2.8 | -0.7±1.6 | -1.3±5.3 | 1.2±1.2 |
| | SIR | 7.0±4.3 | 0.1±3.5 | 1.9±6.2 | 3.6±3.2 |
| | SAR | 11.4±3.1 | 0.2±1.5 | 3.4±2.5 | 3.8±2.1 |
| | ISR | 10.3±2.1 | 4.8±2.2 | 7.9±2.8 | 5.2±2.3 |
| Model 9 | SDR | 2.2±2.8 | -0.6±1.6 | -1.3±5.3 | 1.3±1.2 |
| | SIR | 7.0±4.3 | 0.6±3.5 | 1.5±6.2 | 3.4±3.3 |
| | SAR | 11.5±2.9 | -0.1±1.6 | 3.9±2.3 | 4.1±2.0 |
| | ISR | 10.2±2.2 | 4.8±2.2 | 7.9±2.8 | 5.2±2.3 |

Table 3.2: Comparison between models, values are presented in Decibels: Mean±Standard Deviation

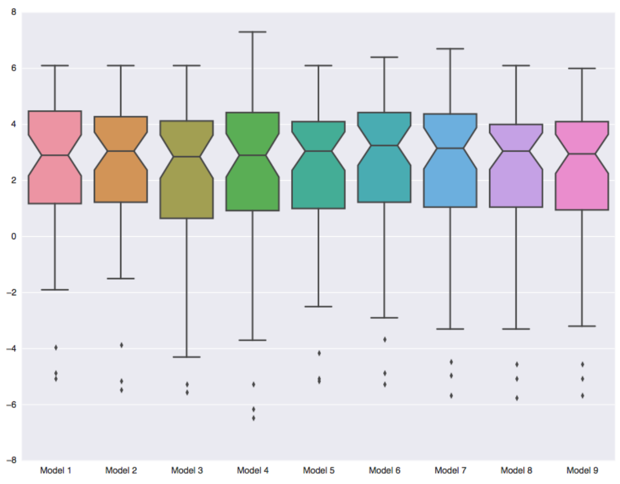


Figure 3.1: Box Plot comparing SDR for bass between models

Table 3.2 presents the mean SDR, SIR, SAR and ISR computed for the Test set of the MSD100 dataset.

Figures 3.1, 3.2, 3.3 and 3.4 show box plot analysis for the SDR computed for the different models proposed in the thesis. It can be seen that model 4, which had 50 filters in both the first and second layer performed significantly poorly on drums and other instrument separation, while the other models were more or less level in terms of performance.

It can be seen from Table 3.2 that the optimal choice for $N_1$ and $N_2$ was $50$ and $30$ respectively as these are the lowest values of these parameters which does not adversely effect the performance of the system. Similarly, the optimal choice for the parameters $\alpha$, $\beta$ and $\beta_{vocals}$ was determined to be $0.001$, $0.01$ and $0.03$ respectively. Strides of $1, 1$ in both layers were found to be efficient as no loss of information was observed with these stride lengths.

### 3.3.3 Experiments With DSD100 Dataset, Low Latency Audio Source Separation

After initial experiments with the MSD100 dataset, we decided to focus on optimizing the network for performance in a low-latency scenario. To do this, we needed to minimize the total number of parameters that needed to be learned and also experiment with the input time context. To this end, the following parameters were tested: Time context $T$ frames, Filter shapes, $(t_1, f_1)$ and $(t_2, f_2)$, Pooling $p_t, p_f$, Number of nodes in the bottleneck, $NN$. The following shows the evaluation of select models for each of the four aforementioned sources plus the ac-

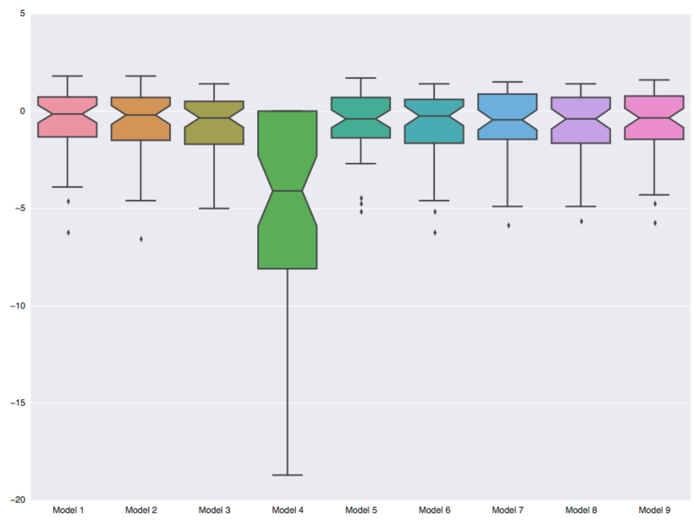


Figure 3.2: Box Plot comparing SDR for drums between models

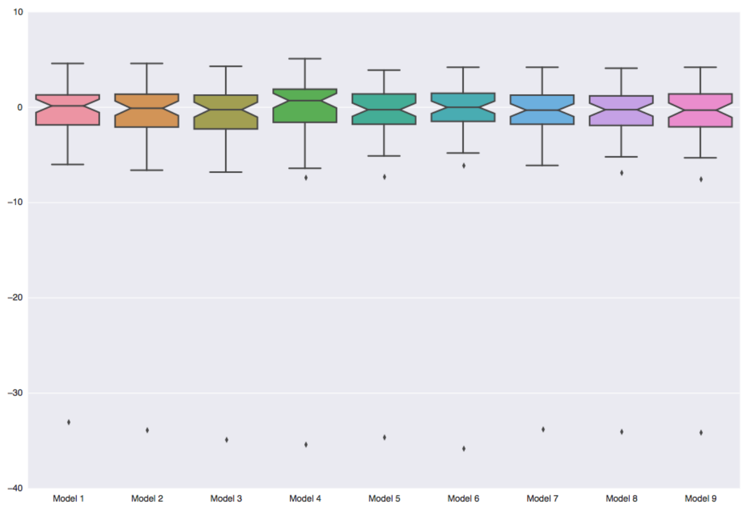


Figure 3.3: Box Plot comparing SDR for vocals between models

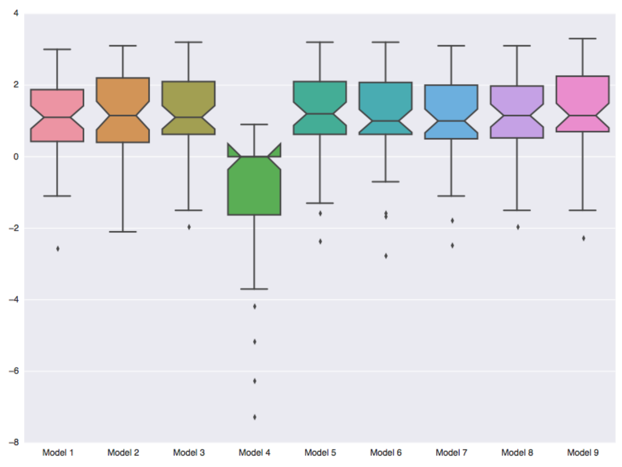


Figure 3.4: Box Plot comparing SDR for other instruments between models

companiment, which refers to the entire mix minus the vocals. The total number of parameters to be optimized and the processing time (PT) for a batch of $T$ time frames for each model are also reported. The PT reported was calculated without the use of the GPU, on the CPU. Only significant results for the Test set are shown herewith, while a complete description of the experiments and results are to be published soon in a paper for the LVA ICA conference.

### Time Context

For this set of experiments,The time context was varied from $T = 10, 15, 20, 25, 30, 40$. No max pooling was used and the timbre layer was chosen to have $f_1 = 513$, while the temporal layer had $t_2 = T/2$. The number of units in the fully connected layer was fixed to be $NN = 512$.

Results in Table 3.3 show that the performance of the system did not reduce greatly while changing the input time context from $290$ milliseconds to $116$ milliseconds. The processing time for the two cases shown is quite low, on the same scale as the input time context.

### Filter shapes and pooling

From the results of the time context experiments, it was seen that both time contexts $T = 10$ and $T = 25$ deserved futher experimentation, in the form of the shape of the filters of the two convolutional layers . Square shaped filters, which are commonly used in image processing, with $f_1 = t_1$ and $f_2 = t_2$ were tried out

| Model Number | Measure | Bass | Drums | Vocals | Others | Accompanime |
|---|---|---|---|---|---|---|
| TC1 | SDR | 1.0+_2.9 | 1.8+_2.5 | 0.6+_2.9 | 1.3+_1.2 | 4.2+_0.8 |
| $T = 10$, $116$ milliseconds | SIR | 4.2+_4.4 | 7.5+_4.9 | 5.0+_4.1 | 3.8+_3.7 | 15.2+_3.6 |
| 125982+$N$x92341 parameters | SAR | 6.4+_2.2 | 7.2+_2.7 | 6.9+_2.4 | 2.5+_2.1 | 13.5+_3.2 |
| PT = 190.4 milliseconds | ISR | 11.2+_3.6 | 8.3+_2.4 | 7.7+_3.1 | 3.6+_1.4 | 6.4+_1.4 |
| TC2 | SDR | 1.1+_2.6 | 2.0+_2.2 | 0.8+_2.5 | 1.2+_1.1 | 4.1+_0.8 |
| $T = 25$,290 milliseconds | SIR | 4.4+_4.4 | 8.0+_4.5 | 5.8+_3.7 | 4.2+_3.8 | 15.5+_3.6 |
| 259362+$N$x215461 parameters | SAR | 6.8+_2.1 | 7.2+_2.8 | 7.0+_2.8 | 1.9+_2.6 | 13.9+_3.2 |
| PT = 290.3 milliseconds | ISR | 11.5+_3.5 | 8.3+_2.4 | 7.3+_2.8 | 3.6+_1.6 | 6.2+_1.3 |

Table 3.3: Experiments with Time Context, values are presented in Decibels: Mean±Standard Deviation

| Model Number | Measure | Bass | Drums | Vocals | Others | Accompan |
|---|---|---|---|---|---|---|
| SP2 $T = 10$ | SDR | 1.3±2.7 | 2.0±2.5 | 1.1±2.8 | 1.7±1.3 | 4.3±0.8 |
| Square Filters, $3, 3$ No Pooling | SIR | 4.6±4.0 | 8.0±4.9 | 8.2±3.8 | 7.0±4.0 | 17.5±3.6 |
| PT = 1026.3 milliseconds | SAR | 6.2±2.1 | 7.4±2.8 | 7.2±2.6 | 2.6±2.3 | 13.8±3.1 |
| 46924062+$N$x47001061 parameters | ISR | 11.8±3.7 | 8.5±2.3 | 7.2±2.8 | 4.5±1.7 | 6.2±1.3 |
| SP5 $T = 25$ | SDR | 1.3±2.5 | 2.9±1.9 | 1.9±2.4 | 1.8±1.3 | 4.3±0.7 |
| Square Filters, $5, 5$ Max Pool $2, 2$ | SIR | 5.1±4.2 | 9.7±4.2 | 9.7±3.3 | 7.4±4.0 | 17.4±3.4 |
| PT = 1524.3.4 milliseconds | SAR | 7.5±2.5 | 7.5±2.7 | 7.3±2.6 | 3.7±2.2 | 14.8±3.1 |
| 23079422+$N$x23085001 parameters | ISR | 11.3±3.3 | 8.5±2.3 | 7.6±3.0 | 4.7±1.7 | 6.0±1.2 |

Table 3.4: Experiments with Filter Shapes And Pooling, values are presented in Decibels: Mean±Standard Deviation

in these experiments. Pooling operations were done in both frequency and time. The number of units in the fully connected layer was $NN = 512$.

Table 3.4 shows that while the total number of parameters to be learned and the processing time required went up, square filters did not greatly improve performance of the system. Given that we are aiming for a low-latency system, it was therefore decided to avoid this shape of filters.

It should be noted that using max-pooling in both frequency and time had a positive impact on results, whereas max-pooling in just one of the dimensions did not. This also reduced the number of parameters to be learned, but not to the extent that using horizontal and vertical filters does.

### Bottleneck

The final set of experiments focuses on the number of hidden units in the fully connected layer. The bottleneck is reduced down to $64$ units and also increased up to $100$ times the input dimension.

| Model Number | Measure | Bass | Drums | Vocals | Others | Accompaniment |
|---|---|---|---|---|---|---|
| BN1 $T = 25$ | SDR | 0.9±2.7 | 2.4±2.0 | 1.3±2.4 | 0.8±1.5 | 3.7±0.8 |
| $NN = 128$ | SIR | 4.6±4.4 | 9.1±4.3 | 7.2±3.6 | 3.8±4.0 | 14.7±3.5 |
| PT = $160.8$ milliseconds | SAR | 6.9±2.3 | 7.0±2.8 | 5.3±2.9 | 2.8±2.4 | 14.0±3.4 |
| 97698+$N$x54181 parameters | ISR | 11.5±3.4 | 8.5±2.2 | 7.3±3.0 | 4.4±1.7 | 6.1±1.3 |
| BN3 $T = 10$ | SDR | 1.3±2.9 | 2.5±2.0 | 0.5±2.8 | 1.2±1.5 | 4.3±0.8 |
| $NN = 100xTx513$ | SIR | 4.4±4.4 | 9.0±4.5 | 5.1±3.8 | 3.6±4.1 | 15.7±3.4 |
| PT = $1469.7$ milliseconds | SAR | 7.0±2.1 | 7.0±2.8 | 7.3±2.4 | 2.7±2.2 | 13.6±3.0 |
| 92886310+$N$x92340181 parameters | ISR | 11.2±3.5 | 8.5±2.2 | 7.4±3.0 | 4.0±1.6 | 6.3±1.3 |

Table 3.5: Experiments with Bottleneck, values are presented in Decibels: Mean±Standard Deviation

It can be observed from Table 3.5 that reducing the bottleneck down to $NN = 128$ slightly improved the results over a bottleneck of $NN = 512$ and also considerably reduced the number of parameters to be learned and the processing time. On the other hand, increasing the bottleneck up to 100 times the input size did not greatly improve the performance of the system while adding greatly to the number of parameters to be learned. Also, it was noted that combining the smaller bottleneck with max pooling did not greatly influence the results.

# Chapter 4

# CONCLUSIONS, COMMENTS AND FUTURE WORK

This chapter presents the findings of the experiments carried out and also discusses some of interesting takeaways from these findings. Future work to improve the system is also proposed here.

## 4.1 Comparison with state of the art

Comparison with state of the art algorithms is shown in Table 4.1, for experiments on the MSD100 dataset. It can be seen that the proposed methodology improves greatly on the results of [Huang et al., 2014], with a 8.4 dB gain in SDR and a 9 dB gain in SAR for vocals. The separation performance for the accompaniment was similar for both models. Comparison with [Ozerov et al., 2012] and an ideal mask is also shown. These figures were extracted from MUS 2015, a part of SiSEC 2015.

Figure 4.1, shows the first and second layer filters learned by model 7. It can be observed that the first layer learns some local frequency domain features for a single time frame, which contain some peak structures. The second layer learns evolution of these features along time. Further investigation on these features will be undertaken in the coming months, but they can be seen analogous to the filters learned by convolutional neural networks used for image classification, [Krizhevsky et al., 2012b] shown in fig 4.2, which are observed to learn edges and other image features.

| Model | Measure | Bass | Drums | Vocals | Others | Accompar |
|---|---|---|---|---|---|---|
| CNN | SDR | 2.6±2.5 | -0.6±1.6 | -0.6±4.9 | 1.3±1.4 | 4.5±1.2 |
| | SIR | 7.5±4.2 | -0.3±3.6 | 1.9±6.2 | 3.4±3.3 | 14.9±5.6 |
| | SAR | 11.7±3.2 | 1.3±1.3 | 3.6±2.1 | 5.0±1.9 | 16.4±2.9 |
| | ISR | 10.3±2.5 | 8.2±2.6 | 7.3±2.7 | 5.6±2.2 | 6.9±1.0 |
| RNN [Huang et al., 2014] | SDR | - | - | -8.9 ± 4.4 | - | 4.1 ± 2.6 |
| | SIR | - | - | 10.8 ± 3.1 | - | 10.7 ± 3.8 |
| | SAR | - | - | -6.6 ± 4.6 | - | 12.1 ± 3.9 |
| | ISR | - | - | 4.8 ± 2.8 | - | 5.0 ± 3.0 |
| FASST [Ozerov et al., 2012] | SDR | 4.4 ± 3.2 | -1.6 ± 1.8 | -1.8 ± 5.8 | 1.1 ± 0.7 | 8.9 ± 3.2 |
| | SIR | 5.3 ± 4.3 | -5.8 ± 3.0 | -3.7 ± 7.5 | 2.8 ± 4.3 | 13.7 ± 3.0 |
| | SAR | 10.0 ± 2.2 | 1.2 ± 1.3 | 4.3 ± 1.4 | 1.8 ± 1.6 | 13.1 ± 3.2 |
| | ISR | 11.7 ± 3.2 | 2.5 ± 1.0 | 5.6 ± 2.5 | 1.9 ± 0.9 | 13.5 ± 2.5 |
| Ideal Mask | SDR | 9.6 ± 2.9 | 4.5 ± 1.2 | 8.4 ± 2.0 | 7.4 ± 2.0 | 19.2± 5.5 |
| | SIR | 13.1 ± 4.0 | 8.7 ± 2.7 | 16.5 ± 3.0 | 10.6 ± 3.1 | 38.8± 18. |
| | SAR | 13.0 ± 3.0 | 5.4 ± 1.6 | 9.6 ± 2.5 | 10.2 ± 1.9 | 22.5 ± 5.6 |
| | ISR | 13.8 ± 3.5 | 7.6 ± 1.7 | 12.2 ± 2.6 | 12.8 ± 7.0 | 22.5 ± 5.5 |

Table 4.1: Comparison with FASST and RNN models, extracted from `http://www.onn.nii.ac.jp/sisec15/evaluation_result/MUS/MUS2015.html`. Values are presented in Decibels: Mean±Standard Deviation

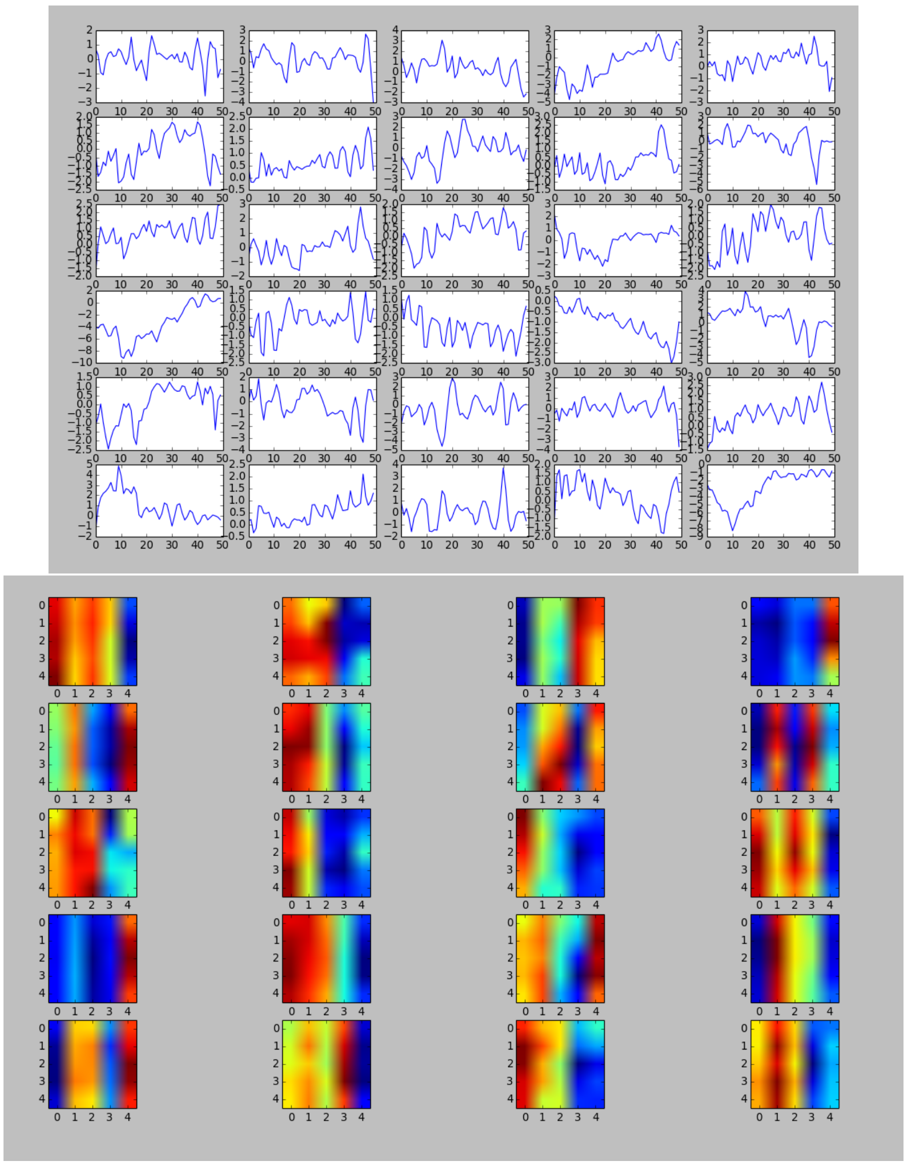


Figure 4.1: First and second convolutional filters for Model 1

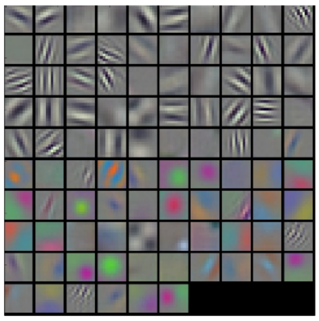


Figure 4.2: Filters learned by convolutional neural networks used in image classification

## 4.2 Conclusions

A low latency online source separation algorithm using convolutional neural networks has been proposed and tested on the MSD100 and DSD100 datasets in this thesis. The algorithm shows a significant improvement in results over [Huang et al., 2014], while providing results on par with [Ozerov et al., 2012], with a significant improvement in processing time. Some of the main discoveries of the research are the use of single dimension filters along the frequency domain, which find local features and another convolution layer which follows the evolution of these features. These convolution filters are more suited to the audio domain as opposed to standard square filters used in image processing, improving greatly on processing time which is a crucial factor for low-latency applications. Contrary to other approaches, which try to model both the target instrument and other background instruments, the presented algorithm just models the target source. The other instruments stem is used primarily as a cost function, to encourage difference between the target source and other instruments. Finally, the cost for each of the stems was monitored individually to ensure that the performance for each stem improved over epochs and not just the contribution was not just from a reduction in cost for one stream.

## 4.3 Future Work

The ground work has been laid for audio source separation using convolutional neural networks, but there is room for improvement. The filters presented in figure 4.1 need to be analyzed to gain insight into what the system is learning. Also, trying to feed context like predominant pitch for vocals might help performance, as it does for [Satoshi Iizuka et al., 2016]. Source separation in itself is not an end goal but rather a stepping stone for other applications and this system can be modified to suit specific applications. We also plan to explore the applicability of the presented algorithm to low-latency applications such as speech enhancement for cochlear implants.

# Bibliography

[A. P. Dempster, ] A. P. Dempster, N. M. Laird, D. B. R. Maximum likelihood from incomplete data via the EM algorithm.

[Al-Rfou et al., 2016] Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., Bengio, Y., Bergeron, A., Bergstra, J., Bisson, V., Bleecher Snyder, J., Bouchard, N., Boulanger-Lewandowski, N., Bouthillier, X., de Brébisson, A., Breuleux, O., Carrier, P.-L., Cho, K., Chorowski, J., Christiano, P., Cooijmans, T., Côté, M.-A., Côté, M., Courville, A., Dauphin, Y. N., Delalleau, O., Demouth, J., Desjardins, G., Dieleman, S., Dinh, L., Ducoffe, M., Dumoulin, V., Ebrahimi Kahou, S., Erhan, D., Fan, Z., Firat, O., Germain, M., Glorot, X., Goodfellow, I., Graham, M., Gulcehre, C., Hamel, P., Harlouchet, I., Heng, J.-P., Hidasi, B., Honari, S., Jain, A., Jean, S., Jia, K., Korobov, M., Kulkarni, V., Lamb, A., Lamblin, P., Larsen, E., Laurent, C., Lee, S., Lefrancois, S., Lemieux, S., Léonard, N., Lin, Z., Livezey, J. A., Lorenz, C., Lowin, J., Ma, Q., Manzagol, P.-A., Mastropietro, O., McGibbon, R. T., Memisevic, R., van Merriënboer, B., Michalski, V., Mirza, M., Orlandi, A., Pal, C., Pascanu, R., Pezeshki, M., Raffel, C., Renshaw, D., Rocklin, M., Romero, A., Roth, M., Sadowski, P., Salvatier, J., Savard, F., Schlüter, J., Schulman, J., Schwartz, G., Serban, I. V., Serdyuk, D., Shabanian, S., Simon, E., Spieckermann, S., Subramanyam, S. R., Sygnowski, J., Tanguay, J., van Tulder, G., Turian, J., Urban, S., Vincent, P., Visin, F., de Vries, H., Warde-Farley, D., Webb, D. J., Willson, M., Xu, K., Xue, L., Yao, L., Zhang, S., and Zhang, Y. (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

[Bengio, 2009] Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and TrendsÂ® in Machine Learning*, 2(1):1–127.

[Coates et al., 2011] Coates, A., Carpenter, B., Case, C., Satheesh, S., Suresh, B., Wang, T., Wu, D. J., and Ng, A. Y. (2011). Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning. In *2011*

*International Conference on Document Analysis and Recognition*, pages 440–445. IEEE.

[Dadula and Dadios, 2014] Dadula, C. P. and Dadios, E. P. (2014). A genetic algorithm for blind source separation based on independent component analysis. In *2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pages 1–6. IEEE.

[Dauphin et al., 2014] Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.

[Dieleman et al., 2015] Dieleman, S., SchlÃ$\frac{1}{4}$ter, J., Raffel, C., Olson, E., SÃ¸nderby, S. K., Nouri, D., Maturana, D., Thoma, M., Battenberg, E., Kelly, J., Fauw, J. D., Heilman, M., de Almeida, D. M., McFee, B., Weideman, H., TakÃ¡cs, G., de Rivaz, P., Crall, J., Sanders, G., Rasul, K., Liu, C., French, G., and Degrave, J. (2015). Lasagne: First release.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Duong et al., 2014] Duong, N. Q. K., Ozerov, A., Chevallier, L., and Sirot, J. (2014). An interactive audio source separation framework based on non-negative matrix factorization. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1567–1571. IEEE.

[El Badawy et al., 2015] El Badawy, D., Ozerov, A., and Duong, N. Q. K. (2015). Relative group sparsity for non-negative matrix factorization with application to on-the-fly audio source separation. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 256–260. IEEE.

[Fleet et al., 2014] Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors (2014). *Computer Vision ECCV 2014*, volume 8689 of *Lecture Notes in Computer Science*. Springer International Publishing, Cham.

[Gómez et al., 2012] Gómez, E., Cañadas, F., Salamon, J., Bonada, J., Vera, P., and Cabañas, P. (2012). Predominant Fundamental Frequency Estimation vs Singing Voice Separation for the Automatic Transcription of Accompanied Flamenco Singing. *13th International Society for Music Information Retrieval Conference (ISMIR 2012)*.

[Grais et al., 2013] Grais, E. M., Sen, M. U., and Erdogan, H. (2013). Deep neural networks for single channel source separation. page 5.

[Grais et al., 2014] Grais, E. M., Sen, M. U., and Erdogan, H. (2014). Deep neural networks for single channel source separation. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3734–3738. IEEE.

[Grossberg, 1988] Grossberg, S. (1988). Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1(1):17–61.

[Hassoun, 1995] Hassoun, M. H. (1995). *Fundamentals of Artificial Neural Networks*. MIT Press.

[Hidalgo, 2012] Hidalgo, J. (2012). Low Latency Audio Source Separation for Speech Enhancement in Cochlear Implants.

[Higuchi and Kameoka, 2014] Higuchi, T. and Kameoka, H. (2014). Joint audio source separation and dereverberation based on multichannel factorial hidden Markov model. In *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

[Huang et al., 2014] Huang, P., Kim, M., Hasegawa-Johnson, M., and Smaragdis, P. (2014). Singing-Voice Separation From Monaural Recordings Using Deep Recurrent Neural Networks. *Ismir 2014*, (Ismir):477–482.

[Huang et al., 2012] Huang, P.-S., Chen, S. D., Smaragdis, P., and Hasegawa-Johnson, M. (2012). Singing-voice separation from monaural recordings using robust principal component analysis. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 57–60. IEEE.

[Hubel and Wiesel, 1968] Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–43.

[Jang, 2010] Jang, J.-S. (2010). On the Improvement of Singing Voice Separation for Monaural Recordings Using the MIR-1K Dataset. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(2):310–319.

[Kokkinakis and Loizou, 2008] Kokkinakis, K. and Loizou, P. C. (2008). Using blind source separation techniques to improve speech recognition in bilateral cochlear implant patients. *The Journal of the Acoustical Society of America*, 123(4):2379–90.

[Krizhevsky et al., 2012a] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012a). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

[Krizhevsky et al., 2012b] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012b). ImageNet Classification with Deep Convolutional Neural Networks.

[Lee and Seung, 2001] Lee, D. D. and Seung, H. S. (2001). Algorithms for Non-negative Matrix Factorization. In *Advances in Neural Information Processing Systems*, pages 556–562.

[Lopez P. et al., 2011] Lopez P., M. G., Molina Lozano, H., Sanchez F., L. P., and Oliva Moreno, L. N. (2011). Blind Source Separation of audio signals using independent component analysis and wavelets. In *CONIELECOMP 2011, 21st International Conference on Electrical Communications and Computers*, pages 152–157. IEEE.

[Miron et al., 2015] Miron, M., Carabias, J. J., and Janer, J. (2015). Improving score-informed source separation for classical music through note refinement. *16th International Society for Music Information Retrieval (ISMIR) Conference*.

[Nugraha et al., 2016] Nugraha, A. A., Liutkus, A., and Vincent, E. (2016). Multichannel audio source separation with deep neural networks. Technical report.

[Ozerov et al., 2009] Ozerov, A., Fevotte, C., and Charbit, M. (2009). Factorial Scaled Hidden Markov Model for polyphonic audio representation and source separation. In *2009 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 121–124. IEEE.

[Ozerov et al., 2012] Ozerov, A., Vincent, E., and Bimbot, F. (2012). A General Flexible Framework for the Handling of Prior Information in Audio Source Separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(4):1118–1133.

[Pascanu et al., 2012] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training Recurrent Neural Networks.

[Pons et al., 2016] Pons, J., Lidy, T., and Serra, X. (2016). Experimenting with Musically Motivated Convolutional Neural Networks. *14th International Workshop on Content-based Multimedia Indexing (CBMI 2016).*

[Qian, 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

[Sainath et al., 2012] Sainath, T. N., Kingsbury, B., and Ramabhadran, B. (2012). Auto-encoder bottleneck features using deep belief networks. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4153–4156. IEEE.

[Satoshi Iizuka et al., 2016] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa (2016). Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)*, 35(4).

[Simpson, 2015] Simpson, A. J. R. (2015). Probabilistic Binary-Mask Cocktail-Party Source Separation in a Convolutional Deep Neural Network.

[Stollenga et al., 2015] Stollenga, M. F., Byeon, W., Liwicki, M., and Schmidhuber, J. (2015). Parallel Multi-Dimensional LSTM, With Application to Fast Biomedical Volumetric Image Segmentation.

[Uhlich et al., 2015] Uhlich, S., Giron, F., and Mitsufuji, Y. (2015). Deep neural network based instrument extraction from music. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2135–2139. IEEE.

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

[Vincent et al., 2007] Vincent, E., Sawada, H., Bofill, P., Makino, S., and Rosca, J. P. (2007). First Stereo Audio Source Separation Evaluation Campaign: Data, Algorithms and Results. In *Independent Component Analysis and Signal Separation*, pages 552–559. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *The Journal of Machine Learning Research*, 11:3371–3408.

[Wellhausen, 2006] Wellhausen, J. (2006). Audio Signal Separation Using Independent Subspace Analysis and Improved Subspace Grouping. In *Proceedings of the 7th Nordic Signal Processing Symposium - NORSIG 2006*, pages 310–313. IEEE.

[Xavier Glorot, ] Xavier Glorot, Y. B. Understanding the difficulty of training deep feedforward neural networks.

[Zapata and Gomez, 2013] Zapata, J. R. and Gomez, E. (2013). Using voice suppression algorithms to improve beat tracking in the presence of highly predominant vocals. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 51–55. IEEE.

[Zeiler, 2012] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. page 6.