Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. Hermann Ney

Seminar Titel im SS 2018

Deep Clustering for ANN Supported Source Separation

Patrick von Platen

Matrikelnummer 331 430

Datum des Vortrages

Betreuer: Tobias Menne

Contents

List of Tables

List of Figures

1 Introduction

When perceiving sound created by multiple acoustic sources, the human auditory system is extremely good at focussing on parts of the sound coming from only one acoustic source.

Imagine yourself at a cocktail party. The sound that you perceive is made up of all kinds of acoustic sources: The voice of the person you are listening to, the sound of music, the background chatter of other people talking, the sound of clinking glasses, etc. Still you can clearly understand the speaker in your group, even though its acoustic signal overlaps at every moment with the acoustic signal of other people speaking and background noise.

The difficulty of understanding speech in a multiple speaker environment is often called "The cocktail party problem" [4], first mentioned by Colin Cherry in 1953. It was also Colin Cherry, who first brought up the idea of automatic speech separation in his famous paper on "experiments of the recognition of speech" [7]. The first method to yield some promising results was introduced by A.S. Bregman in 1990 using *Auditory scene analysis* [3]. Since then, multiple methods have been explored. One of them is *Spectral Clustering* , a method based on partitioning data points into clusters according to the eigenstructure of a similarity matrix.

With the receent rise of deep learning in a variety of applications, the first method to be applied to source separation was presented by Chao Weng and co. in 2015 [24]. Quickly, different deep learning methods based on a technique called *Deep Clustering* emerged, forming the state-of-art in the source separation problem. This report, will focus on the deep clustering method as it was introduced by John R. Hershey and co. in 2015 [10] and will study *TasNet*, a deep clustering system performing audio separation in the time domain yielding state-of-the art results [14].

The applications of automated source separation are wide-ranging. To name a few:

- *Automatic meeting transcription*: In business meetings, multiple speakers are alteranating in a short time or even speaking at the same time. Making it possible to automatically separate speakers from each other and transcribe would be of great use for companies.

- *Virtual assistant*: Virtual assistants, like Alexa, Siri and Google Home are playing a critical part in smart home systems. Filtering out noise and separating the owner's voice from other voices are challenges that can be taken on by automated source separation techniques.

- *Automatic subtitling of music/video*: According to the World Health Organization (WHO) over 5% of the world population suffers from disabling hearing loss [1]. Subtitling of music and video content, which are often made up of sound coming from multiple acoustic sources, is therefore essential for them.

To begin with, the source separation problem is mathematically defined and basic mathematical operations and basic deep learning methods used in deep clustering are explained 2. In the following conventional methods and first attempts of using deep learning techniques for automated source separation are presented 3. The essence of this paper being deep clustering and *TasNet* will be presented in-detail in section 4. Finally, a conclusion is drawn 5.

2 Basics

This section firstly defines the source separation problem and then gives an overview of the basic operations and methods used in the later presented systems.

2.1 Definition source separation problem

A generel definition of the source separation problem was given by Jean-Francois Cardoso, defininig the problem as "recovering unobserved signals or sources from several observed mixtures" [5]. He also added two important restrictions that we will apply to our definition as well:

1. The source signals are not observed
2. No information is available about the mixture

The condition of not knowing how many sources the mixture is composed of became known as *blind* source separation [5]. In this paper, we add another restriction: The signal is observed over a single channel (single microphone). To sum it up, the *source separation problem* is defined in this paper as "recovering the original signals of an unknown amount of sources from a single-channel mixture of the source signals".

Putting the above definiton in a mathematical form:

- the unknown number of acoustic sources: N.
- the original sound of the i-th source at time t: $x_i(t)$.
- the single received sound signal at time t: $x(t) = \sum_{i=1}^{N} x_i(t)$.

The goal of source separation is to recover $x_1(t), ..., x_N(t)$ from $x(t)$. An algorithm performing this task is presented below.

Input: $x(t)$
Output: $(N, \{x_1(t), ..., x_N(t)\})$

2.2 Mathematical operations

This section aims to shine light on essentiel mathematical operations used in the systems that will be explained in sections 3, 4.

2.2.1 Short term fourier transform

In order to understand the short term fourier transform, one firstly has to get an understanding of the fourier transform itself. Having a continous signal $s(t)$, the fourier transform is defined as

$$S(f) = \int_{\mathbb{R}} s(t)e^{-2j\pi ft}dt$$

The fourier transform $S(f)$ gives the magnitude of every frequency in the signal. A signal can be decomposed as the sum of signals consisting only of one frequency (so called sinusoids). Thus, one can say that the higher the magnitude of a frequency of the fourier transform, the higher is the percentage of the signal being composed of the sinusoid of this frequency. An important aspect to notice is that the fourier transform integrates over the whole time domain of the signal in order to get the exact magnitude of every freuency present in the signal. Thus integrating only over a part of the time domain leads to approximated values of the frequencies magnitudes. On the other hand, a shorter time intervall to integrate over leads to a better time localization of the frequency. This famous trade-off is well-known as the uncertainty principle in signal analysis [16].

The short term fourier transform divides the signal into local sections using a window function $w(\tau) = 0 : \tau \notin [-a, a]$ over which the fourier transform is applied.

$$S(f,t) = \int_{\mathbb{R}} s(\tau)w(\tau - t)e^{-2j\pi f\tau}d\tau$$

Appling the short term fourier transform converts a signal $X(t)$ to multiple (F, T) bins that can be plotted in a *spectrogram* as can be seen in Figure 1. It therefore is used to describe the change in frequency over time. Using an appropiate window funciton leads to a good trade-off between time localization and frequency approximation.

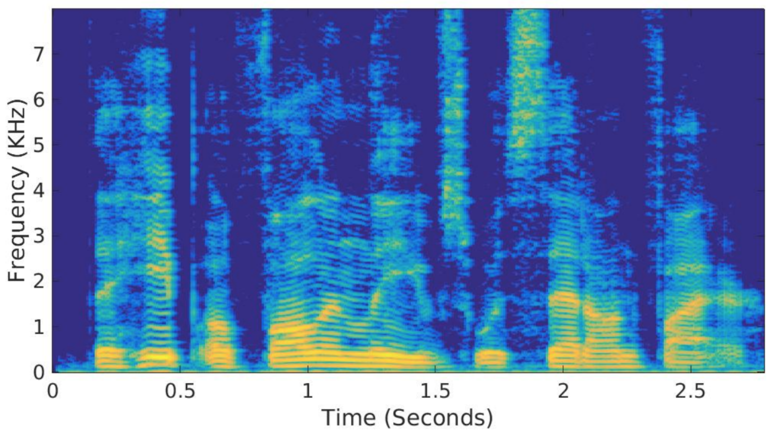


Figure 1: Spectrogram of a speech signal

2.2.2 Embedding space

An embedding is a structure-preserving, injective mapping $f : X \rightarrow Y$. Therefore, every object of domain X is mapped to a distinct object in domain Y. In this report, we are interested in embeddings that map to normed spaces meaning that it is possible to measure the similarity of two objects $x_1, x_2 \in X$ by measuring its distance $d(y_1, y_2) \in \mathbb{R} : y_1, y_2 \in Y$. It is essentially used to convert data to a feature representation where certain properties can be preserved by distance, so that the data is easily comparable by computers.

One famous example are word embedding models, such as *word2vec* that maps words to a vector representation. Assuming that words have multiple degrees of similarity, word embeddings are used to quantisize the similarity of words on multiple axis [15]. Later, we will see that the function mapping objects from one domain to an embedding space can be trained on artificial neural networks and are very powerful tools for the source separation problem 4.1.

2.3 Deep neural network basics

This section aims to clarify basic deep learning methods and architectures used in the sections 3, 4.

2.3.1 Convolutional neural network

Convolutional neural networks are neural networks that are specialized for processing data that has grid-like topology, for example time series data for a 1D convolution [9]. The idea is based on the convolution operation $*$ being:

$$s(t) * g(t) = \int_{\mathbb{R}} s(\tau)g(t-\tau)d\tau$$

which corresponds to

$$s(n) * g(n) = \sum_{i=-\infty}^{\infty} s(i)g(n-i)$$

in the discrete case. The fourier transform of a signal to its spectrum can essentially be seen as a convolution operation [25].

This principle can now be applied to artificial neural networks. Let x be the output layer and w the weights of a convolutional layer. Such a layer is defined by its filter size k and other parameters, such as stride, padding, etc. For more information please refer to [9]. The weight array is also called "Kernel" [21] and normally a convolutional layer is decomposed of multiple kernels, such that there are multiple parallel outputs. The convolution operation in neural networks is defined as follows:

$$y_{i,j} = w_i * x = f(\sum_{n=j}^{j+k}(w_{i,n-j}x_n) + b_i)$$

$y_{i,j}$ presents the j-th neuron of the output layer of the i-th kernel. Similarly $w_{i,l}$ presents the l-th weight of the i-th kernel. x_n is the output of the n-th neuron of the previous layer and b_i is the bias corresponding to the i-th kernel. Finally $f()$ is the activation function applied to each convolution operation.

Convolutional neural networks can very well be applied to act as filterbanks for the signal as was shown in [8] and as we will later see in section 4.1.

2.3.2 Recurrent neural network

Recurrent neural networks are neural networks that are "specialized in processing a sequence of values $x_1, ..., x_t$" [9]. For that reason, they are very well applicable when dealing with acoustic signals having high autocorrelation (high correlation between $s(t)$ and $s(t-k), s(t+k)$ for $k > 0$). Recurrent neural networks are extremely effective when dealing with sequential data. First, they process data of different time instances in one computation step, thus taking full advantage of autocorrelation. Second, they can handle data sequences of variable length in contrast to feed forward neural networks where the input length is determined by the size of the input layer. This is achieved by loops between recurrent units inside the neural network [9]. Unfolding a recurrent unit z of a recurrent neural network at different time instances leads to so-called hidden copies of the network being connected to each other: $z^0, z^1, ..., z^{t-1}, z^t$ As a result, such a recurrent unit z^t at time t takes both the output of the node leading to it (x^t) and the output of the same recurrent unit of one time instance earlier (z^{t-1}) as input resulting in:

$$z^t = f(Vx^t + Wz^{t-1} + b)$$

with $f()$ being the activation function, V the weights between recurrent unit z and input x, W the weights between two hidden recurrent units z^{t-1} and z^t and b the constant bias term. This basic setup is known to have the vanishing/exploding gradient problem due to an exponential decrease/increase in value of long-term components ($\frac{\partial z^t}{\partial z^{t-k}} \rightarrow$ *const.* $\times W^k \rightarrow 0$ or ∞)[17]. To overcome this problem, Hochreiter and co. introduced a "better" recurrent unit, called long shot term memory.

2.3.3 Long short term memory

To overcome the exploding/vanishing gradients problem, the long short term memory unit introduces an internal state which can be seen as the unit's "memory" [11]. At every time instance t, the unit updates its memory c^t using the memory c^{t-1} and the unit's output h^{t-1}, as well as the new input x^t. c^t in combination with h^{t-1} and x^t is then used to produce the new output. The structure of the long short term memory unit is shown in Figure 2. It is important to notice that the long short term memory uses x^t and h^{t-1} to compute four different gates: f,i,g,o which are used to control the flow of information in the unit. Also, we can see that when doing backpropagation, the gradient easily "flows" to earlier hidden units without the weight matrix being applied to it (shown by the red arrow) in 2. This "highway" for the gradient strongly reduces the vanishing/exploding gradient effect and allows the unit to take into account long term dependencies of the input data.

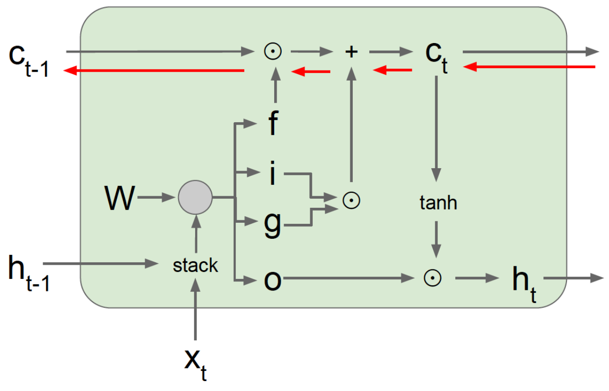


Figure 2: Long short term memory

To furter improve this ability, bi-directional long short term memory units were proposed to exploit correlations between data in both time directions and are used in many state-of-the-art systems nowadays. Going more into detail is out of scope for this report, but more informationcan be found in [11].

2.3.4 Autoencoder

An autoencoder is a neural network that is "trained to attempt to copy its input to its output" [11]. It always consists of two parts, the encoder and the decoder. In its simplest form, the encoder maps the input x to one hidden layer $f(x) = z$ and the decoder tries to reconstruct the original data $g(z) = x'$ from this layer. Even though autoencoders are trained with loss functions trying to minimize the difference between x and x', perfect recontruction of the data input is not the goal. Instead, autoencoders should be trained in a way preventing them to just copy the input to the output, but to extract useful information about data distribution instead [9]. Thereby, the hidden layer should represent the most useful information.

Two approaches are considered in this report. First, a undercomplete autoencoder is defined by the hidden layer having much lower dimension than the data, thus forcing the network to "store all data in a compressed form". A backdraw of this approach is that the hidden layer might be two small to learn enough useful information. Second, a sparse autoencoder forces the hidden layer to be sparse, by introducing a sparsity penalty on the hidden layer $P(h)$, which will then be added to the loss function $\mathbb{L}(x, x') + P(h)$. Especially in the field of machine translation, autoencoders in combination with recurrent neural networks produce state-of-the-art results [20]. For more detailed information, please refer to [9].

3 Conventional methods

The first theoretical framework for automated source separation was introduced by A.S. Bregman in 1990 [3]. Systems using this approach are called *computational auditory scene analysis* and produced first results which are acceptable [18]. Section 3.1 will cover this method.

Spectral Clustering, a method well-know for data clustering, was then applied to the source separation problem and showed good results (see 3.2). It was the first method used for source separation to train some of its parameters on actual data [2].

3.1 Computational auditory scene analysis

Computational auditory scene analysis for source separation is an automation of the auditory scene analysis which is inspired by the principles of processing sound in human auditory system [18]. In computational auditory scene anlysis the algorithm consists of two parts: *segmentation* and *grouping.* A more detailed description of the algorithm can be seen in the following and it based on the paper of *Wang* and *Brown* [18].

Input: (N, $X(t)$)
Algorithm:

1. Segmentation:
   - X(t) is transformed into time-frequency space using short term fourier transform
   - Segmenation rules are used to define different time-frequency (T,F) regions
2. Grouping:
   - (F, T) bins are combined to streams corresponding to sound sources
   - Auditory masks are created using different streams

Output: $(\{x_1(t), ..., x_N(t)\})$

The segmentation rules used in the segmentation part are hand-crafted similarity features upon which the different (T,F) regions are created. For every speaker, an audiotry masks is created as a last step of the grouping part. These auditory masks are boolean matrices that can be applied to the spectogram to retrieve the part of the spectogram corresponding to a speaker. Important to notice is that the amount of different speakers N has to be an input of the algorithm and cannot be computed by the system itself. Since computational auditory scene analysis is a rule based system, it is very difficult to generelize to broader conditions and needs very precise tuning in order to work.

Computational auditory scene analysis was the first version to produce some meaningful results, but is not powerful enough for the general source seperation problem as we defined it in section 2.1.

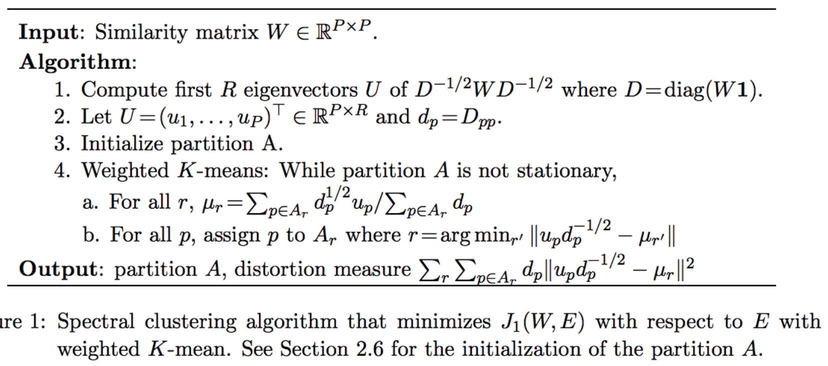
3.2 Spectral Clustering

Spectral Clustering is a well-known method in multivariante statistic for the clustering of data. It makes use of the eigenvalues (also called spectrum, thus the name *Spectral clustering*) of a similarity matrix W of the data to reduce the dimensionality of the data. Since the idea of spectral clustering will apply to methods in latter chapters 4.1 it will be described here in detail. The similarity matrix W can be defined in multiple ways depending on the task, but it should have the following properties.

1. W should be a hermitian matrix: $W^T = W$.
2. $(W)_{i,j} \geq 0, \forall i, j$
3. W should be non-negative definite.

Considering a dataset of P entries $D \in \mathbb{R}^P$, W describes the similarity between each of these datapoints, thus $W \in \mathbb{R}^{P \times P}$. Since W can be considered as a distance measure, with $W(d_1, d_2) = W(d_2, d_1)$ and only positive distances, the first two of the above defined properties seem reasonable. The third property ensures that all eigenvalues of W are ≥ 0 being a characteristic of non-negative definite matrices.

Having defined a similarity matrix, an example of an algorithm for spectral clustering used in [2] is presented in Figure 3.



```
Input: Similarity matrix $W \in \mathbb{R}^{P \times P}$.
Algorithm:
    1. Compute first $R$ eigenvectors $U$ of $D^{-1/2}WD^{-1/2}$ where $D = \mathrm{diag}(W\mathbf{1})$.
    2. Let $U = (u_1, \dots, u_P)^\top \in \mathbb{R}^{P \times R}$ and $d_p = D_{pp}$.
    3. Initialize partition A.
    4. Weighted $K$-means: While partition $A$ is not stationary,
        a. For all $r$, $\mu_r = \sum_{p \in A_r} d_p^{1/2} u_p / \sum_{p \in A_r} d_p$
        b. For all $p$, assign $p$ to $A_r$ where $r = \arg\min_{r'} \|u_p d_p^{-1/2} - \mu_{r'}\|$
Output: partition $A$, distortion measure $\sum_r \sum_{p \in A_r} d_p \|u_p d_p^{-1/2} - \mu_r\|^2$
```

Figure 1: Spectral clustering algorithm that minimizes $J_1(W, E)$ with respect to E with weighted K-mean. See Section 2.6 for the initialization of the partition A.

Figure 3: Spectral clustering algorithm

First, all eigenvalues and their corresponding eigenvectors of the similarity matrix are calculated. The eigenvectors $v_i \in \mathbb{R}^P$ can be sorted according to their eigenvalues in descending order, such that $v_0, v_1, ..., v_r, ..., v_p$ with $\lambda(v_0) = max_{i \in \{0,...,P\}} \lambda_i$. Now we can choose the first R eigenvectors to have R different clusters. Our set of eigenvectors is defined as $V = (v_0, v_1, ..., v_r) \in \mathbb{R}^{P \times R}$. We can now define a set $U = (u_0, u_1, ..., u_p) = V^T \in \mathbb{R}^{R \times P}$, whereas u_i corresponds to exactly one data point and is the vector containing the i-th elements of all eigenvectors $v_1, ..., v_r$. As a last step, we apply a weighted version of the *k-means algorithm* [19] by first randomely selecting R vectors $m_1, ..., m_r$ as mean vectors and then successively assigning the vectors $u_1, ..., u_p$ and updating $m_1, ..., m_r$. After conversion, all vectors assigned to m_j present the j-th cluster.

Bach and Jordan applied spectral clustering for source separation of two-speaker mixtures. The method described in the following is based on their paper "Learning spectral clustering, with application to speech separation" [2].

First, the definition of a similarity matrix W corresponding to the mixture signal $X(t)$ will be derived. As a first step the mixture signal can be converted to (F, T) bins using the short term fourier transform as explained in 2.2.1. The (F, T) are then used to create multiple features, which are based on: *Energy, continuity, common fate cues, pitch estimation, timbre, etc...* In this way, we create multiple features $s_i(f, t)$ for every data point (f, t) whereas the calculation of $s_i(f, t)$ can make use of neighoring data points.

The features are essentially embeddings making the data points more comparable, so that we can define k similarity matrices corresponding to k features $W_k(i, j) = f_k(s_i, s_j)$ choosing appropriate functions f_k. The final similarity matrix consists of a weighed combination of each similarity matrix $W = W_1^{\alpha_1} \odot W_2^{\alpha_2} \odot ... \odot W_k^{\alpha_k}$ with $\odot$ being the hadamard product. The similarity matrix can then be inserted into our previously explained spectral clustering algorithm 3 and the two clusters can be defined resulting in a neat spectogram for every speaker. The conversion of the data points to a combined similarity matrix works a like front-end processor.

The only thing left to explain is how to define the weigth vector $\alpha \in \mathbb{R}^K$. Having N labeled data sets with the (F, T) bins correspondng to one of two speakers so that every data set has a reference partition E'_n, a partition $E_n(\alpha) = \text{K-Means}(W_n(\alpha))$ can be derived. The *k-means* algorithm is applied to the similarity matrix depending on α and the previously defined features as well as the *k-means* algorithm. Finally, a cost function $\mathbb{L} = \sum_i^N H(E'_n, E_n(\alpha))$ is used to find the α minimizing $\mathbb{L}$.

Even though spectral clustering can produce very good results on two-speaker separation [2] it has its limitations. Already four seconds of speech sampled at 5.5 kHz leads to 22000 data points and thus, a similarity matrix having 22000×22000 entries not even applying the short term fourier transform. It is obvious that eigenvalue and eigenvector calculations become very expensive. A remedy is applying low rank matrix approximations such as the Nystroem approximation to perform eigenvalue and eigenvector on a matrix that is reduced from $N \times N$ to $N \times D$. This allows for significant performance improvement and is especially useful on sparse matrices [2].

Additionally, the algorithms relies on "handcrafted features" and uses only shallow learning for the similarity matrices by essentially applying linear regression to optimize the scaling vector α. More speakers means more overlapping speakers per (F, T) bin so that features taking into account multiple data points for calculation contain multiple speakers, thus making it pointless to cluster them. A speaker segmentation would be needed beforehand to create useful features, which would miss the point of creating features to perform the segmentation. Deep clustering as explained in 4.1 tries to solve this problem.

4 Artificial neural network supported source separation methods

In this section, two state-of-the-art approaches using artificial neural networks for the source separation problem will be discussed. The first approach, called "Deep clustering" 4.1 is very similar to *spectral clustering* as explained in the section before and works in the frequency domain of the signal. The second approach, called "TasNet" works in the time domain and will be discussed in detail in 4.2. Finally, the subsection introduces a metric for performance measurement and compares the two methods.

When applying artificial neural networks to the source separation problem there are two common problems that occur.

- The permutation problem
- The dimension output problem

To best understand the permutation problem of artificial neural networks, let us go through an example. An artificial network for two-speaker source separation that takes n (F, T) bins corresponding to the mixed signal $X(t)$ as an input in the first layer would need an output layer of size $2 \times n$ with the first half corresponding to the first speaker and the second half to the second speaker. Having three two-speaker datasets, for example (A,B), (A,C) and (B,C) to train such a network the following problem occurs. First the network will learn to activate the (F, T) corresponding to speaker A on the left half of the output layer and the output for speaker B on the right half. Second, the network learns to assign speaker A again to the left half and speaker C to the right half. Third, when training on the dataset of speaker B and C, the network will try to assign the right half to both speaker B and C [6]. These constant "mis-learnings" will make it most probably impossible for the learning process to converge.

The dimension output problem refers to the static input and output dimension of normal feed forward neural networks. Having defined an output dimension corresponding to two speakers for example the network will learn to separate only two-speaker signal mixtures making it impossible to separate signal mixtures containing more than two speakers.

The explanation of the two methods will be structured by answering the following questions.

1. What is the idea of the method?
2. How is the system trained?
3. How is the trained system applied to new data?
4. How does the method tackle the two problems described above?

4.1 Deep clustering method - frequency domain audio separation based methods

Deep clustering was introduced by John R.Hershey and co. [10]. The method extends the application of spectral clustering by applying artificial neural networks to model an embedding V replacing the similarity matrix W as described in 3.2.

4.1.1 Structure

The artificial neural network is represented by $f_\theta(S) = V$ with

- θ being the parameter space of the artificial neural network
- S with $s = s_{t,f} = s_n \in S \in \mathbb{R}^N, N = T \times F$
- V with $v = v_n \in \mathbb{R}^K$ and $v_n \in V \in \mathbb{R}^{N \times K}$ and $|v_n|^2 = 1$

S is the spectrogram of the mixed signal $x(t)$. V is a K-dimensional embedding, with each v_n representing a normalized embedding for s_n (one bin of the (T,F) spectrogram). Clustering of v_n using the k-means algorithm then results in binary masks that can be applied to the (T,F) spectrogram to retrieve the separated signals for every speaker. It is obvious that much more complex structures and relations can be learned in *deep clustering* than in *spectral clustering.*

4.1.2 Training

To train the network, the following cost function is applied to θ. For practical reasons we will write $V(\theta)$ simply as V, but it should be kept in mind that V depends on θ.

$$C(\theta) = |VV^T - YY^T|_F^2 \tag{1}$$

with $C(\theta)$ the cost function and

- $VV^T \in \mathbb{R}^{N \times N}$ being the similarity matrix
- $Y \in \mathbb{R}^{N \times C}$ with C being the number of different speakers. $y_{n,i} = 1$ if speaker i is dominating in the (T,F) bin n and $y_{n,i} = 0$ otherwise. Thus y_n presents a one-hot encoded vector. Thus $YY^T(n_1, n_2) = 1$ only if $y_{n_1,i} = y_{n_2,j}, \forall i \in 1, ..., c$ and 0 otherwise.
- F being the Frobenius norm with $|A|_F^2 = \sum_{i,j} A(i,j)^2$

Having the optimal source separation Y^* we can derive Y using a one-hot encoding replacing dominante values with 1 and the rest with 0. Equation 1 can be rewritten.

$$\begin{aligned} C(\theta) = |VV^T - YY^T|_F^2 &= \sum_{i,j} (v_i^T v_j - y_i^T y_j)^2 \\ &= \sum_{i,j;y_i=y_j} (v_i^T v_j - 1)^2 + \sum_{i,j;y_i \neq y_j} (v_i^T v_j)^2 \\ = \sum_{i,j;y_i=y_j} (v_i^T v_j)^2 - 2(v_i^T v_j) + 1 + \sum_{i,j} (v_i^T v_j)^2 &- \sum_{i,j;y_i=y_j} (v_i^T v_j)^2 \\ &= \sum_{i,j;y_i=y_j} (-2v_i^T v_j + 1) + \sum_{i,j} (v_i^T v_j)^2 \end{aligned} \tag{2}$$

Using the "parallelogram law": $u^T v = \frac{1}{2}(|u|^2 + |v|^2 - |u-v|^2)$ with u, v being vectors, we get

$$\begin{aligned} C(\theta) = \sum_{i,j;y_i=y_j} (-(|v_j|^2 + |v_i|^2 - |v_i - v_j|^2) + 1) &+ \sum_{i,j} (v_i^T v_j)^2 \\ = \sum_{i,j;y_i=y_j} (|v_i - v_j|^2 - 1) &+ \sum_{i,j} (v_i^T v_j)^2 \\ = \sum_{i,j;y_i=y_j} |v_i - v_j|^2 &+ \sum_{i,j} (v_i^T v_j)^2 - N \end{aligned} \tag{3}$$

We derived an equation that allows for a more intuitive representation: For $y_i = y_j$ the respective embeddings v_i and v_j should be as close together as possible when trying to minimize the first addend. The second addend and third addend can be seen as a regulizer preventing trivial solutions. As a conclusion, the embedding V is very similar to the low rank approximation matrix used in spectral clustering.

To optimize computation performance the expensize $N \times N$ matrix calculation in equation 3 can be replaced by the less expensive $K \times K$ with K being the embedding dimension of V which is normally magnitudes smaller than N. We arrive at:

$$C(\theta) = |V^T V|_F^2 - 2|V^T Y| + |Y^T Y| \tag{4}$$

Finally, the derivations $\frac{\partial C(\theta)}{\partial \theta}$ can be split into $\frac{\partial C(\theta)}{\partial V} \frac{\partial V}{\partial \theta}$. While the calculation of $\frac{\partial V}{\partial \theta}$ depends heavily on the network architecture and can be quite expensive using, for example long short term memory units, as in [6], it is easy to calculate:

$$\frac{\partial C(\theta)}{\partial V} = 4V(V^T V) - 4Y(Y^T V)$$

The parentheses in the above equation make sure that the computational less expensive matrix operations are executed.

4.1.3 Testing

Having succesfully trained the parameters θ of the neural network, we can now compute V of unseen data X using the short term fourier transform and applying the neural network $V = f_\theta(\text{ShortTermFourierTransform}(X)$. Analog to *spectral clustering*, we use the *k-means* algorithm by minimizing its inference cost with the following equation:

$$\hat{Y} = \text{argmin}_Y V - Y(Y^TY)^{-1}Y^TV \tag{5}$$

$\hat{Y}$ are used as binary masks to separate the sources. So we can lastly apply the masks $\hat{Y}(c)$ dot-wise to the spectogram of the mixture signal:

$$S_c = \hat{Y}(c) \odot X$$

Since $(Y^TY)^{-1}Y^T \in \mathbb{R}^{C\times D}$ presents the means of all clusters, minimizing the cost function 5 is essentially the same as performing k-means clustering on $v_n \in V$ with sucessive creating of the binary masks from the assignments [6]. In the case that $VV^T = Y^*Y^T$ both cost functions 5, 1 coincide in their global minimum.

4.1.4 Conclusion

Since training essentially relies on YY^T both the permutation and the dimension output problem are solved. $YY^T(n_1, n_2) = YY^T(n_2, n_1) = 1$ if y_{n_1} corresponds to the same class as y_{n_2} and 0 otherwise. So the order of the classes is cleary disregarded since only "same" or "different" classes matters. Also YY^T always has the same dimension $N \times N$ no matter how many speaker are mixed in the signal.

As we will later see in section 4.3, results achieved by *clustering* are impressive, there is room of improvement considering that the (F, T) are hard assigned to only one class. There is one disadvantage using the deep clustering approach which is its high latency time. When encoding test data on a trained network because of its format to take the whole signal as input before giving an output.

4.2 TasNet - time domain audio separation based methods

Tasnet was first introduced by Luo and Mesgarani in [14]. Instead of performing spectral clustering on the mixed signal x, the mixed signal is divided into K chunks of equal length L, normalized and fed into an autoencoder which is made up of three parts

- Encoding
- Separation
- Decoding

After decoding, the K chunks of separated signals can be retrieved. The following Figure 4 shows *TasNet* in detail.

4.2.1 Encoding

The signal chunks x_k can be described by N weighted basis functions B so that $x = wB$ with $w \in \mathbb{R}^N, X \in \mathbb{R}^L, B \in \mathbb{R}^{N\times L}$. The convolutional network implicitly applies the inverse matrix B^{-1} to the input signal so that the output will be the weights of the basis functions $w = xB^{-1}$. The convolution takes place over the time dimension of the mixed signal x. This is analog to the short term fourier transform 2.2.1, by which the signal is decomposed to sinusoid of different frequencies only that the sinusoid are replaced by basic functions b_i that will be learned by the neural network as we will later see in 4.2.3. As can be seen in Figure 4, a gated convolutional layer is applied to the input X_k:

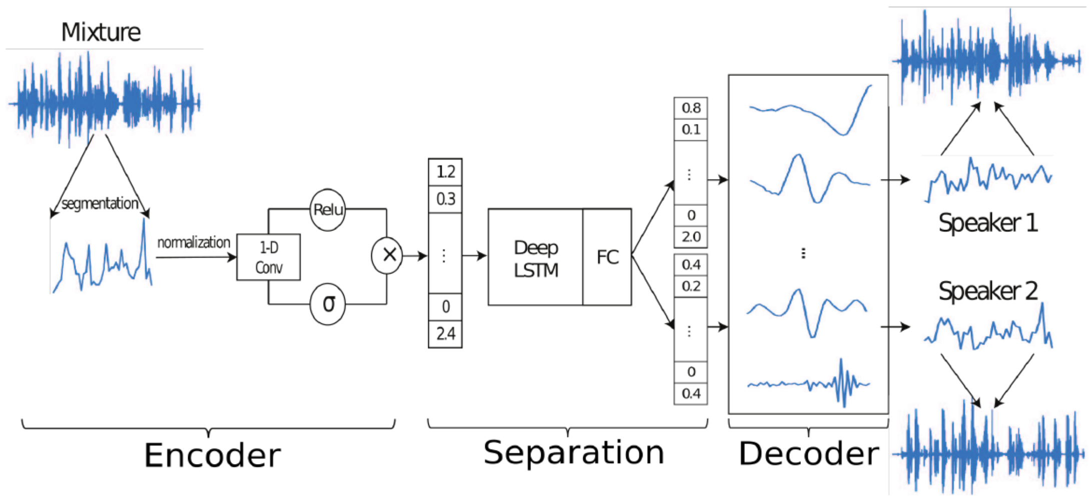


Figure 4: TasNet

$$w_k = \text{ReLu}(x_k * U) \odot \text{Sigmoid}(x_k * V) \tag{6}$$

It is called a gated convolution layer because $\text{Sigmoid}(x_k * V) \rightarrow 0$ if $x_k * V << 0$ and $\rightarrow 1$ if $x_k * V >> 0$. Thus, the term acts like a gate - in most cases it is either very close to 0 or to 1 - and therefore decides which basis functions b_i should be fully considered or disregarded. Since $U, V \in \mathbb{R}^{N \times L}$, the convolutional layer consists of $2 \times N$ different *kernels* each being applied to the whole signal x_k of length L and thus acting like a filterbank.

Therefore V is the matrix deciding whether a signal is composed of a basis function b_i and U is deciding the percentage of the signal being composed of the basis function. The output of the encoder is therefore the vector w_k describing the percentage of each basis function of which the original signal chunk x_k is made of.

4.2.2 Separation

The separation part of the network consist of a deep long short term memory network that takes w_k as input and gives different masks vectors $M_k = [m_{k,1}, m_{k,2}, ..., m_{k,c}]$ as output with c being the number of different speakers.

First, to speed up the training process, the weights w_k are normalized by the following equations

$$\hat{w} = \frac{g}{\sigma} \odot (w_k - \mu) + b \tag{7}$$

σ and μ present the variance and mean respectively of the vector w_k and $g, b \in \mathbb{R}^{1 \times N}$ are gain and bias vectors that are jointly optimized with the network. This is a standard normalization and makes $\hat{w}$ scale invariant. Empirically, it was seen to be also more efficient for training [14]. The normalized weights are then put into a deep recurrent neural network using long short term memory units to output the different masks vectors:

$$M_k = f_\theta(w_k)$$

f_θ describes herewith the recurrent neural network. More details about how the network works and is trained will be covered in 4.2.4.

4.2.3 Decoding

Each mask vector $m_{k,i}$ for the signal of the i-th source in the k-th chunk can then be applied to the "mixed" weight *non-normalized* vector w_k to get the weight vector for source i:

$$d_{k,i} = m_{k,i} \odot w_k$$

Having the weight vector for source i we can finally use our basis vectors to recover the signal $x_{k,i} = d_{k,i}B$ of length L, by means of a *deconvolutional* layer:

$$x_{k,i} = \text{deconv}_{\hat{\theta}}(d_{k,i})$$

This is essential the inverse convolution operation of 4.2.1 with $\hat{\theta}$ being the parameters to train so that $\text{deconv}_{\hat{\theta}}$ is the same as appling the basis matrix B to $d_{k,i}$. Now it is clear that the matrix B is never actually given as an output but more or less used as a theoretical framework and only fully described by the weights of the encoder and decoder of the network. Nevertheless, the weights of the kernels of both the convolutional and deconvolutional layers of the encoder and decoder would show signs of learned filterbanks. More in-detail information about *deconvolutional* layer can be read in [26]. Having recovered all of the K decoded signal chunks for one source, makes it easy to concatenate them to get the complete separated signal for source i:

$$x_i = [x_{1,i}, ..., x_{K,i}] .$$

4.2.4 Training

Since the output of the network are the separated source signals x_i, the network can directly be trained on a signal to noise ratio which is the common metric when evaluation source separation systems 4.3. For *TasNet* a scale-invariant source-to-noise ratio SNR is used as the training criterion:

$$\text{SNR}_i = 10 \log_{10} \frac{|s_i|^2}{|e_i|^2} \tag{8}$$

Furthermore, we define:

- $s_i = \frac{x_i^T x_i^* x_i^*}{|x_i^*|^2}$
- $e_i = x_i - s_i$

s_i presents our normalized target value and e_i is the error of the estimated signal. It is scale invariant meaning that a in itself correct signal that has the wrong scale will be penalized much less. Also, it has been shown that a high SNR or *Signal Distortion Ratio* objectively corresponds much more to clean source separation than only the error e_i [22]. Therefore our loss function is defined as to minimize the inverse SNR of all sources:

$$L(\theta_{all}) = \sum_{i=1}^{C} \mathrm{SNR}_i \tag{9}$$

All parameters of the network presented above as θ_{all} can jointly be trained together. The kernel weights convolutional layer 4.2.1 and the deconvolutional layer 4.2.3 respectively will be learned to act as “filterbanks”, thus defining the best basic functions b_i the signal can be described with.

The separation part of the network has to learn the parameters of its long short term memory unit. Here we encounter the typical permutation problem while training. The input w_k gets mapped to the different source masks $m_{1,k}, ..., m_{c,k}$. These are then used to create $x_{1,k}, ..., x_{c,k}$. We have the target source vectors $x^*_{1,k}, ..., x^*_{c,k}$, but we do not know for sure which target source vectors corresponds to the produced output, since the permutation can be arbitary. Therefore permutation invariant training has to be applied, as can be seen in Figure 5.

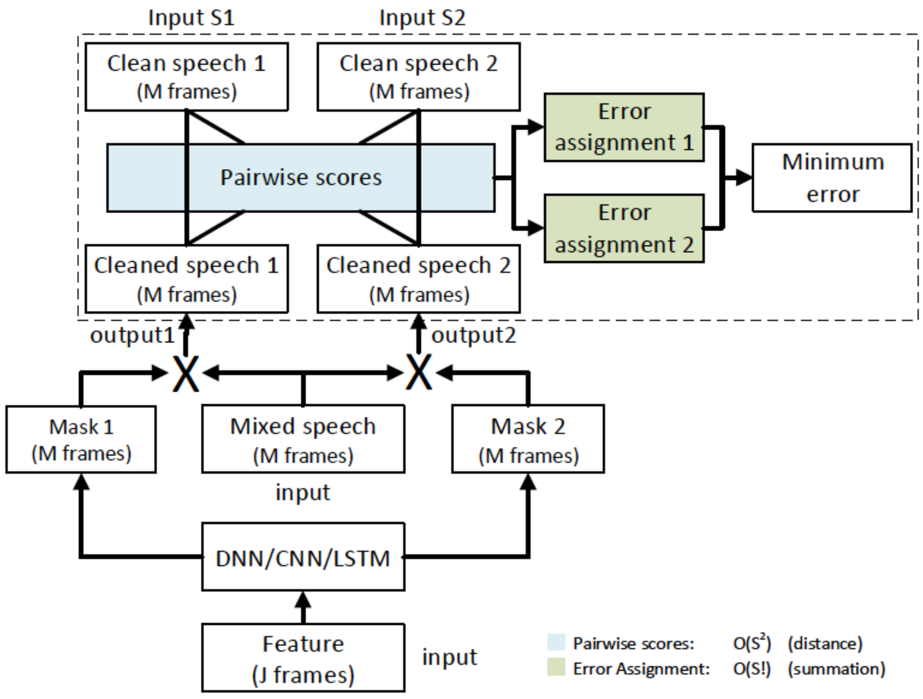


Figure 5: Permutation Invariant Training

As we can see in the diagramm, the masks are applied to “Mixed Speech”, which would be the mixed weight vector w_k in our case and the single source signals “Clean Speech” are each assigned to corret label using permutation invariant training. Thus, the network always assigns the correct label to the output for each source and there will not be any convergence issues.

In order for that to happen [13] proposes an approach called *utterance-level permutation invariant training*. The following idea is used here. For every mixed signal chunk x_k, there is the correct single source signal $x^*_{k,i}$. We can use 6 to derive the “correct” weight vectors for every single source $w^*_{k,i}$. The output of the separation layer gives the source masks $m_{1,k}, ..., m_{c,k}$ used to get $w_{1,k}, ..., w_{c,k}$, which will now be assigned to the "correct" weight vectors $w^*_{k,i}$ to make sure that $x_i = w_i * B$ with its correct label x^*_i are used in the loss function 9. To get the correct assignment, the following equation is applied at every iteration step of training [13]:

$$\theta^* = \text{argmin}_{\theta \in \mathcal{P}} \sum_{s=1}^{c} |m_{s,k} \odot w_k - w^*_{s,k}(\theta(s))|^2 \tag{10}$$

$\mathcal{P}$ is the space of all possible permutations of $w_{k,1}, ..., w_{k,c}$ consisting of $c!$ possible solutions. Thus θ^* minimizes the mean squared error between $w_{s,k} = m_{s,k} \odot wk$ and the "correct" label $w^*_{s,k}$. This calculation might seem computationally expensive because of the factor $c!$ but we only need to calculate c^2 possible combinations. Then, there are still $c!$ possible combinations to choose from. The remedy is that in most cases c is very small, rarely > 4, so that all combinations can quickly be computed and the minimum can be chosen. There are two other reasons that were empirically shown [13].

- Using long short term memory units, the network learns which masks of a signal chunk $m_{s,k}$ was put out on which range of the output nodes and thus the network shows a tendency to output the same masks for all k at the same "spot".
- Minimizing equation 10 reinforces the minimization of the global loss function 9 in that a correct assignment leads to an overall better cost than a wrong assignment. Therefore, there will be no convergence problems when trying to minimize both functions.

4.2.5 Testing

Having succesfully trained the network, applying it to unseen test data is straight forward. x will be separated into chunks x_k each of length L. These will be encoded, separated and decoded as explained in 4.2.1, 4.2.2 and 4.2.3 to give the output $x_{1,k}, ..., x_{c,k}$. These chunks will be concatenated back together and we have our single source signals $x_1, ..., x_c$.

4.2.6 Conclusion

TasNet uses a very "natural" approach to solve the source separation problem by using the raw signal as an input and thus letting the network learn the best transformation via a convolutional layer in the encoding part 4.2.1. Also, it has a great latency time in that it can directly separate single chunks of the mixed signal to the single source signals instead of having to analyse the whole mixed signal. There is a huge drawback though. As one might have noticed, the output of the separation part in the network is static. That means that the network can only be applied to a mixed signal with a predefined number of speakers. While it succesfully solves the permutation problem, it does not solve the dimension output problem.

4.3 Comparison of the two methods

In order to compare the two methods, we first have to define a metric for comparison. A common metric is the so-called *Signal-to-distortion ratio* $:=$ *SDR* as defined in [22].

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

For more in-detail information, please refer to [22]. The following table shows the performance of *TasNet* and the deep clustering method as stated in [12] for a two-speaker single channel source separation. Both networks were trained on the WSJ0-2mix dataset.

Table 1: SDR (dB) for different methods on two speaker WSJ0-2mix dataset

TasNet-BLSTM	**10.8**
DPCL++	**10.8**

As we can see both networks yield the exact same result. Deep clustering has the advantage that it can be trained on mixed signals of an undefined number of sources and thus can also be applied to "real" source separation as we defined it in 2.1. The advantage of *TasNet* is that is has a very low latency time and thus could be applied to real-time applications, whereas deep clustering methods cannot. Finally it can be said, that *TasNet* can be regarded to be more of an *end-to-end* model, where all components belong to one autoencoder and are trained together.

5 Conclusion

As a general conclusion it can be said that conventional methods made too many "hand-crafted" assumptions about the underlying nature of a mixed signal and thus were often not able to generalize to slighly changing conditions. In this report, we saw an overview of some of the state-of-the-art methods that are used for the source separation problem and can say with certainty that artificial neural networks are very useful methods to use when it comes to source separation. There is a lot of research about artificial neural network supported source separation and pretty much every year new systems and methods are presented that show better results than the previous state-of-the-art systems. While writing this report a new *end-to-end* system using so-called *Unfolded Iterative Phase Reconstruction* was presented [23] that yielded a new state-of-the-art *SDR* of **12.8** on the dataset WSJ0-2mix.

There surely will be more improvement in the near future so that systems will be able to cleanly separate mixture speech and solve the long-lasting "Cocktail party problem".

References

[1] Millionslivewithhearingloss. `http://www.who.int/pbd/deafness/news/Millionslivewithhearingloss.pdf`. Accessed: 2018-05-05.

[2] Francis R. Bach and Michael I. Jordan. Learning spectral clustering, with application to speech separation. *J. Mach. Learn. Res.*, 7:1963–2001, December 2006.

[3] A.S. Bregman. *Auditory Scene Analysis: The Perceptual Organization of Sound*. A Bradford book. Bradford Books, 1994.

[4] Adelbert Bronkhorst. The cocktail party phenomenon: A review of research on speech intelligibility in multiple-talker conditions. 86:117–128, 01 2000.

[5] Jean-Francois Cardoso. Blind signal separation: Statistical principles, 2003.

[6] Z. Chen. *Single Channel auditory source separation with neural network*. Dissertation, Columbia University, 2017.

[7] E. Colin Cherry. Some experiments on the recognition of speech, with one and with two ears. *The Journal of the Acoustical Society of America*, 25(5):975–979, 1953.

[8] Pavel Golik, Zoltán Tüske, Ralf Schlüter, and Hermann Ney. Convolutional neural networks for acoustic modeling of raw time signal in lvcsr. In *INTERSPEECH*, 2015.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe. Deep clustering: Discriminative embeddings for segmentation and separation. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 31–35, March 2016.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[12] Yusuf Isik, Jonathan Le Roux, Zhuo Chen, Shinji Watanabe, and John R. Hershey. Single-channel multi-speaker separation using deep clustering. *CoRR*, abs/1607.02173, 2016.

[13] M. Kolbk, D. Yu, Z. H. Tan, and J. Jensen. Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(10):1901–1913, Oct 2017.

[14] Yi Luo and Nima Mesgarani. Tasnet: time-domain audio separation network for real-time, single-channel speech separation. *CoRR*, abs/1711.00541, 2017.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[16] Sangnam Nam. An uncertainty principle for discrete signals. *CoRR*, abs/1307.6321, 2013.

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org, 2013.

[18] J. Rouat. Computational auditory scene analysis: Principles, algorithms, and applications (wang, d. and brown, g.j., eds.; 2006) [book review]. *IEEE Transactions on Neural Networks*, 19(1):199–199, Jan 2008.

[19] H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1:801–804, 1956.

[20] Jinsong Su, Shan Wu, Deyi Xiong, Yaojie Lu, Xianpei Han, and Biao Zhang. Variational recurrent neural machine translation. *CoRR*, abs/1801.05119, 2018.

[21] Zhun Sun, Mete Özay, and Takayuki Okatani. Design of kernels in convolutional neural networks for image classification. *CoRR*, abs/1511.09231, 2015.

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

[23] Z.-Q. Wang, J. Le Roux, D. Wang, and J. R. Hershey. End-to-End Speech Separation with Unfolded Iterative Phase Reconstruction. *ArXiv e-prints*, April 2018.

[24] C. Weng, D. Yu, M. L. Seltzer, and J. Droppo. Deep neural networks for single-channel multi-talker speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(10):1670–1679, Oct 2015.

[25] A. I. Zayed. A convolution and product theorem for the fractional fourier transform. *IEEE Signal Processing Letters*, 5(4):101–103, April 1998.

[26] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.