Signal Processing 131 (2017) 181–189

Contents lists available at ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

Convolutional Deep Stacking Networks for distributed compressive sensing ☆

Hamid Palangi [a,*], Rabab Ward [a], Li Deng [b]

[a] Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada V6T 1Z4
[b] Microsoft Research, Redmond, WA 98052, USA

ARTICLE INFO

Article history:
Received 14 December 2015
Received in revised form
18 May 2016
Accepted 5 July 2016
Available online 12 July 2016

Keywords:
Distributed Compressive Sensing
Deep learning
Deep Stacking Networks
Convolutional neural networks

ABSTRACT

This paper addresses the reconstruction of sparse vectors in the Multiple Measurement Vectors (MMV) problem in compressive sensing, where the sparse vectors are correlated. This problem has so far been studied using model based and Bayesian methods. In this paper, we propose a deep learning approach that relies on a Convolutional Deep Stacking Network (CDSN) to capture the dependency among the different channels. To reconstruct the sparse vectors, we propose a greedy method that exploits the information captured by CDSN. The proposed method encodes the sparse vectors using random measurements (as done usually in compressive sensing). Experiments using a real world image dataset show that the proposed method outperforms the traditional MMV solver, i.e., Simultaneous Orthogonal Matching Pursuit (SOMP), as well as three of the Bayesian methods proposed for solving the MMV compressive sensing problem. We also show that the proposed method is almost as fast as greedy methods. The good performance of the proposed method depends on the availability of training data (as is the case in all deep learning methods). The training data, e.g., different images of the same class or signals with similar sparsity patterns are usually available for many applications.

1. Introduction

Compressive Sensing (CS) has the advantage of acquiring signals in a compressed form [6–8]. Unlike classic sensing and compression schemes where part of the transformed coefficients are thrown away after spending a computational cost to calculate them, in compressive sensing, only the necessary information is "*measured*" at the encoder. The only requirements are the sparsity of the signal in a certain basis and the incoherency between the sparsifying basis and the measurement matrix. Since many natural signals are sparse in time or space or a transform domain, CS has found many applications in medical imaging, remote sensing, geophysical data analysis, health telemonitoring, communications and others.

In CS, instead of acquiring N samples of a signal $\mathbf{x} \in \Re^{N\times 1}$, M random measurements are acquired where $M < N$. This is expressed by:

$$\mathbf{y} = \mathbf{\Phi x} \tag{1}$$

where $\mathbf{y} \in \Re^{M\times 1}$ is the known measured vector and $\mathbf{\Phi} \in \Re^{M\times N}$ is a random measurement matrix. To uniquely recover $\mathbf{x}$ given $\mathbf{y}$ and $\mathbf{\Phi}$, $\mathbf{x}$ must be sparse in a given basis $\mathbf{\Psi}$. This means that

$$\mathbf{x} = \mathbf{\Psi s} \tag{2}$$

where $\mathbf{s}$ is $K-sparse$, i.e., $\mathbf{s}$ has at most K non-zero elements. From (1) and (2):

$$\mathbf{y} = \mathbf{As} \tag{3}$$

where $\mathbf{A} = \mathbf{\Phi\Psi}$. Since the above problem has only one measurement vector, it is usually called the Single Measurement Vector (SMV) problem.

For compressive sensing with Multiple Measurement Vectors (MMV), also known as distributed compressive sensing, the aim is to reconstruct a set of L unknown sparse vectors $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_L\}$ from a set of L measurement vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_L\}$. Assuming that $\mathbf{S}$ is a matrix whose columns are the sparse vectors and $\mathbf{Y}$ is a matrix whose columns are the corresponding measurement vectors, the MMV problem can be stated as:

$$\mathbf{Y} = \mathbf{AS} \tag{4}$$

To reconstruct $\mathbf{S}$ using $\mathbf{Y}$ and $\mathbf{A}$ in (4), one can either reconstruct each of the sparse vectors in $\mathbf{S}$ independently or

☆This work was made possible by NPRP grant # 7 - 684 - 1 - 127 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

* Corresponding author.
E-mail addresses: hamidp@ece.ubc.ca (H. Palangi), rababw@ece.ubc.ca (R. Ward), deng@microsoft.com (L. Deng).

http://dx.doi.org/10.1016/j.sigpro.2016.07.006
0165-1684/

reconstruct **S** jointly. Using an average case analysis, it has been shown that solving the MMV problem jointly can lead to better uniqueness guarantees than those obtained by solving the SMV problem for each sparse vector independently [9]. More formally, assume that **S** is jointly sparse, i.e., the non-zero entries of each sparse vector occur at the same indices as those of other vectors. In other words, the sparse vectors have the same support. Then, the necessary and sufficient condition to obtain a unique **S** given **Y** is:

$$|supp(\mathbf{S})|<\frac{spark(\mathbf{A})-1+rank(\mathbf{S})}{2} \tag{5}$$

where $|supp(\mathbf{S})|$ is the number of rows in **S** with non-zero energy and *spark* of a given matrix is the smallest possible number of linearly dependent columns of that matrix [10]. *spark* gives a measure of linear dependency in the system modelled by a given matrix. In the SMV problem, no rank information exists. In the MMV problem, the rank information exists and affects the uniqueness bounds.

In the current literature, **S** in (4) is reconstructed using one of the following types of methods: (1) greedy methods [11] like Simultaneous Orthogonal Matching Pursuit (SOMP) which performs non-optimal subset selection, (2) relaxed mixed norm minimization methods [12] where the ℓ_0 norm is relaxed to ℓ_1 norm to formulate a convex optimization problem, or (3) Bayesian methods like [13–15] where a posterior density function for the values of **S** is created, assuming a prior belief, e.g., **Y** is observed and **S** should be sparse in a basis $\mathbf{\Psi}$. As shown in [13–15], in the MMV problem, the model based methods like the Bayesian methods usually perform better than the first two groups of methods because they exploit the statistical dependencies among the different sparse vectors in **S**.

In many applications, a huge amount of data similar to the data to be reconstructed, i.e., **S**, is available. Examples are the case of a security camera recording the same environment, recordings of the different channels of electroencephalogram (EEG) of the same patient over time, different patches of images of the same class, e.g., buildings or flowers, etc. In these applications, the sparse vectors in **S** usually have some form of correlation (dependency) with each other. The question is: (i) how can we use the available data to capture the dependencies among the channels and (ii) how to exploit the captured information to improve the performance of the reconstruction algorithm in the MMV problem? To address the first part of this question, we propose a Convolutional version of Deep Stacking Networks (DSNs) [16] which we refer to as CDSN. To capture the dependencies among different channels we propose the use of a sliding convolution window over the columns of the matrix **S** (where each convolution window contains w consecutive columns of **S** where w is the size of convolution window). To address the second part of above question, we propose a two step greedy algorithm to exploit this information at the decoder during the reconstruction. By performing experiments on an image dataset, we show that the proposed method outperforms the well known MMV solver SOMP and the model based Bayesian methods proposed in [17,14,15]. This is shown using the popular Wavelet and DCT transforms as the sparsifying basis. We emphasize that the proposed method does not add any complexity to the encoder, i.e., the encoder is a random matrix. The complexity is added at the decoder.

The contributions of this paper are as follows: a convolutional version of the Deep Stacking Networks (DSNs), which we refer to as CDSN, is proposed. We then propose the use of CDSN, which is a data driven method, to capture the dependencies among different channels in the MMV problem. We then use a two step greedy reconstruction algorithm to exploit the information captured by CDSN at the decoder to reconstruct **S**.

Note that the great success of deep learning [18–21], motivated new applications reported in this paper. The rest of the paper is organized as follows: in the next section, related work is discussed. In Section 3 the proposed method is presented. Experimental evaluation and discussion are presented in Section 4. In Section 5, conclusions and future work are presented.

2. Related work

Model based methods that exploit the information in the structure of sparse vector(s) have been studied extensively in the compressive sensing literature [13–15,17,22–27]. In [22], it has been theoretically shown that using signal models that exploit these structures results in a decrease in the number of measurements. In [23], a thorough review on CS methods that exploit the structure present in sparse signal is presented. In [17], a Bayesian framework for CS is presented. This framework uses a prior information about the sparsity of the vector **s** to provide a posterior density function for the entries of **s** (assuming **y** is observed). It then uses a Relevance Vector Machine (RVM) [28] to estimate the entries of the sparse vector. This method is called Bayesian Compressive Sensing (BCS). In [14], a Bayesian framework is presented for the MMV problem. It assumes that the L channels or "*tasks*" in the MMV problem (4), are not statistically independent. By imposing a shared prior on the L channels, an empirical method that estimates the hyperparameters is presented and extensions of RVM used for the inference step. This method is known as Multitask Compressive Sensing (MT-BCS). In [14], it is experimentally shown that the MT-BCS outperforms three methods, the method that applies Orthogonal Matching Pursuit (OMP) on each channel, the Simultaneous Orthogonal Matching Pursuit (SOMP) method which is a straightforward extension of OMP to the MMV problem, and the method that applies BCS on each channel independently. In [13], the Sparse Bayesian Learning (SBL) [28,29] is used to solve the MMV problem. It is shown that the global minimum of the proposed method is always the sparsest one. The authors in [15] address the MMV problem when the entries in each row of **S** are correlated. An algorithm based on SBL is proposed and it is shown that the proposed algorithm outperforms the mixed norm ($\ell_{1,2}$) optimization as well as the method proposed in [13]. The proposed method is called T-SBL. In [24], a greedy algorithm aided by a neural network is proposed to address the SMV problem in (3). The neural network parameters are calculated by solving a regression problem and are used to select the appropriate column of **A** at each iteration of OMP. The main modification to OMP is that of replacing the correlation step with a neural network. This method was experimentally shown to outperform OMP and ℓ_1 optimization. This method is called Neural Network OMP (NNOMP). An extension of [24] with a hierarchical Deep Stacking Network (DSN) [16] is proposed in [25] for the MMV problem. DSN architecture has different applications, for example see [1–5]. The joint sparsity of **S** is an important assumption in the proposed method. To train the DSN model, the Restricted Boltzmann Machine (RBM) [30] is used to pre-train DSN and then fine tuning is performed. It has been experimentally shown that this method outperforms SOMP and $\ell_{1,2}$ in the MMV problem. The proposed methods are called Nonlinear Weighted SOMP (NWSOMP) for the one layer model and DSN-WSOMP for the multilayer model. In [26], a feedforward neural network was used to solve the SMV problem as a regression task. A pre-training phase followed by fine tuning was used. For pre-training, the Stacked Denoising Auto-encoder (SDA) proposed in [31] had been used. Note that an RBM with Gaussian visible units and binary hidden units (i.e., the one used in [25]) has the same energy function as an auto-encoder with sigmoid hidden units and real valued observations [32]. Therefore the extension of [26] to the MMV problem is expected to give similar performance to that of [25]. In [27], a different MMV problem is solved where the sparsity patterns of different sparse vectors in **S** are NOT

similar. A method based on Long Short Term Memory (LSTM) was proposed to address this problem. In this paper, we assume that the sparse vectors in $\mathbf{S}$ have similar sparsity patterns. For example, the sparse vectors are the DCT or Wavelet transforms of images. In the sparse representation literature, the dictionary learning method [33] uses the available training data to learn the sparsifying basis ($\mathbf{\Psi}$ in (2)) that can represent the signal as compactly as possible. The main difference between dictionary learning and our work here is that we assume the sparsifying basis as given and there is no need to learn it. In other words, the sparse vectors in $\mathbf{S}$ are not necessarily very sparse. Although we expect the performance to improve by combining dictionary learning with our proposed method, in this paper we focus on the performance improvement obtained by using the proposed approach only.

3. Proposed method

To give a high level picture of how the proposed method reconstructs the sparse vectors represented as columns of $\mathbf{S}$, given the measured vectors represented as columns of $\mathbf{Y}$ and the measurement matrix $\mathbf{A}$, we can think of it as a greedy method with two steps. At the first step, the "*location*" of a non-zero entry in a sparse vector in $\mathbf{S}$ is predicted using CDSN, and at the second step, the updated estimate of the corresponding sparse vector is calculated.

The above two steps are performed at each iteration, for a number of iterations. At each iteration, the location of one of the non-zero entries in a column in $\mathbf{S}$ is predicted using CDSN. The stopping criteria for iterations depend on whether the residual between $\mathbf{Y}$ and the estimate of it at that iteration is less than a threshold or the number of iterations equals to the total number of entries in $\mathbf{S}$.

We continue our explanation of the proposed method using the block diagrams presented in Fig. 1. In Fig. 1(a), the dashed lines show that the process of reconstructing the sparse vectors repeats for a number of iterations. At each iteration, each column of $\mathbf{S}$ is estimated by a separate process. The inputs to this process are the residuals at each iteration and the outputs are the estimated columns of $\mathbf{S}$.

More formally, in the proposed method, before the i-th iteration of reconstructing the j-th column of $\mathbf{S}$, i non-zero entries of that column are predicted so far. We represent the j-th column of $\mathbf{S}$ by $\mathbf{s}_j$. At the i-th iteration, the first step predicts the location of the $(i+1)$-th non-zero entry of $\mathbf{s}_j$, using the residuals of columns contained in a sliding convolution window. In Fig. 1(a), an example with convolution window of size 3 is represented. This sliding convolution window helps in capturing the dependencies among channels. The predicted location of the $(i+1)$-th non-zero entry is then added to the support of $\mathbf{s}_j$. This support is represented by Ω_j in Fig. 1(a). The second step finds the updated estimate of $\mathbf{s}_j$ by solving a linear least squares problem that finds $\mathbf{s}_j$ given $\mathbf{y}_j$ (the j-th column of $\mathbf{Y}$) and $\mathbf{A}^{\Omega_j}$:

$$\hat{\mathbf{s}}_j = \underset{\mathbf{s}_j}{\operatorname{argmin}} \left\| \mathbf{y}_j - \mathbf{A}^{\Omega_j}\mathbf{s}_j \right\|_2^2 \tag{6}$$

where $\mathbf{A}^{\Omega_j}$ is a matrix that includes only those columns of $\mathbf{A}$ that correspond to the support of $\mathbf{s}_j$. The definition of the residual matrix at the i-th iteration is $\mathbf{R}_i = \mathbf{Y} - \mathbf{A}\mathbf{S}_i$ where $\mathbf{S}_i$ is the estimate of the sparse matrix $\mathbf{S}$ at the i-th iteration. Columns of $\mathbf{R}$ are represented by $\mathbf{r}_j$, $j = 1, 2, \ldots, L$ in Fig. 1(a).

Now the remaining important questions are:

(i) how can we find the parameters of CDSN represented in Fig. 1 (b), i.e., $\mathbf{W}_1^{(1)}$, $\mathbf{W}_2^{(1)}$, $\mathbf{W}_1^{(2)}$, $\mathbf{W}_2^{(2)}$, $\mathbf{W}_1^{(3)}$, $\mathbf{W}_2^{(3)}$? Note that in Fig. 1(b), $\mathbf{W}_1^{(l)}$ is the matrix of weights from input layer to hidden layer for the l-th layer of CDSN and $\mathbf{W}_2^{(l)}$ is the matrix of weights from hidden layer to output layer for the l-th layer of CDSN. Residual vectors of channel one, two and three are represented by $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ respectively in Fig. 1(b).

(ii) how should the training data be represented to find the parameters of CDSN given that at each iteration of the proposed method the location of one of the non-zero entries is determined? This means that the CDSN should observe the non-zero entries in the training data one by one. In the other words, we cannot simply use the given training data (e.g., images), and an appropriate representation of it is necessary.

We first answer question (ii) and then explain the method for question (i).

To address question (ii), we use the following procedure on the given training data. Consider a sparse vector of the j-th channel $\mathbf{s}_j$ in the training data. Assume that it has k non-zero entries. We first calculate $\mathbf{y}_j$ using $\mathbf{y}_j = \mathbf{A}\mathbf{s}_j$. Then we find the entry in $\mathbf{s}_j$ that has the maximum value. Assume that the index of this entry is k_0. Then we calculate the residual vector from:

$$\mathbf{r}_j = \mathbf{y}_j - \mathbf{A}^{\Omega_j} s_j(k_0) \tag{7}$$

where $\mathbf{A}^{\Omega_j}$ only includes the k_0-th column of $\mathbf{A}$ and $s_j(k_0)$ is the k_0-th entry of $\mathbf{s}_j$. It is obvious that this residual value results from not including the remaining $k-1$ non-zero entries of $\mathbf{s}_j$. Now we set the k_0-th entry of $\mathbf{s}_j$ to zero. From the remaining $k-1$ non-zero entries, the second largest value of $\mathbf{s}_j$ offers the main contribution to $\mathbf{r}_j$ in (7). Therefore, we use $\mathbf{r}_j$ to predict the location of the second largest value of $\mathbf{s}_j$. Assume that the index of the second largest value of $\mathbf{s}_j$ is k_1. We normalize all entries in $\mathbf{s}_j$ with respect to the value in the k_1-th entry. Therefore the training pair is $\mathbf{r}_j$ in (7) as input and the normalized $\mathbf{s}_j$ with k_0-th entry set to zero as output target. Now we set the k_1-th entry of $\mathbf{s}_j$ to zero. This gives us a new sparse vector with $k-2$ non-zero entries. Then we calculate the new residual vector from:

$$\mathbf{r}_j = \mathbf{y}_j - \mathbf{A}^{\Omega_j}\left[s_j(k_0), s_j(k_1) \right]^T \tag{8}$$

where $\mathbf{A}^{\Omega_j}$ includes the k_0-th and k_1-th columns of $\mathbf{A}$. We use the residual in (8) to predict the location of the third largest value in $\mathbf{s}_j$. Assume that the index of the third largest value of $\mathbf{s}_j$ is k_2. We normalize all entries in $\mathbf{s}_j$ with respect to the value in the k_2-th entry. Therefore the new training pair is $\mathbf{r}_j$ in (8) as input and the normalized $\mathbf{s}_j$ with k_0-th and k_1-th entries set to zero as output target. We continue this procedure up to the point where $\mathbf{s}_j$ does not have any non-zero entry. Note that all above procedure is done for one training sample of the j-th column of $\mathbf{S}$. The result is the training pair ($\mathbf{r}_j$, $\mathbf{s}_j$) where $\mathbf{r}_j$ is the input and $\mathbf{s}_j$ is the output target. Then we continue with the next training sample. We do the same procedure for each channel.

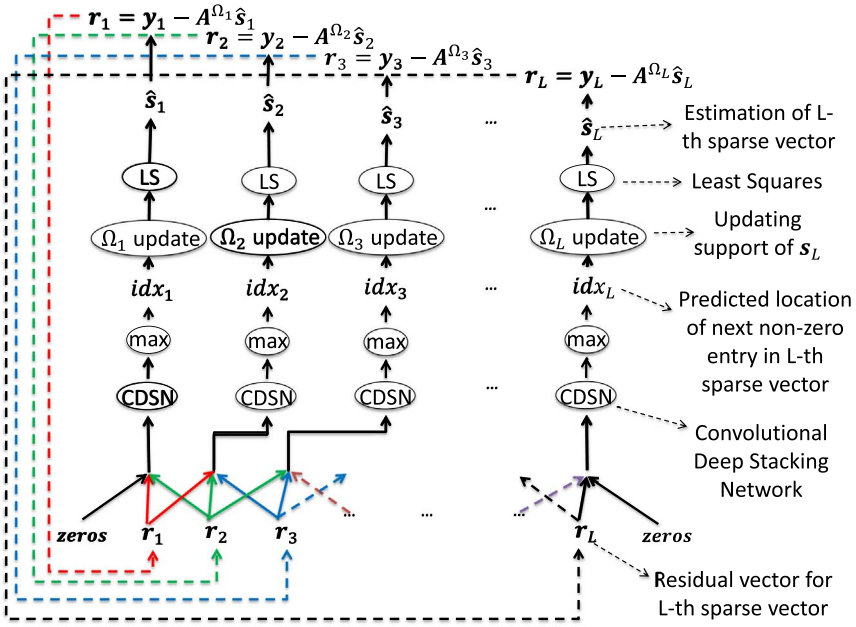
We address question (i) by describing the CDSN formally and explaining the training method. The forward pass for l-th layer of CDSN represented in Fig. 1(b) is:

$$\begin{aligned} \mathbf{h}^{(l)} &= \frac{1}{1 + e^{-\mathbf{W}_1^{(l)}\mathbf{z}^{(l)}}} \\ \mathbf{v}^{(l)} &= \left[\mathbf{W}_2^{(l)}\right]^T \mathbf{h}^{(l)} \end{aligned} \tag{9}$$

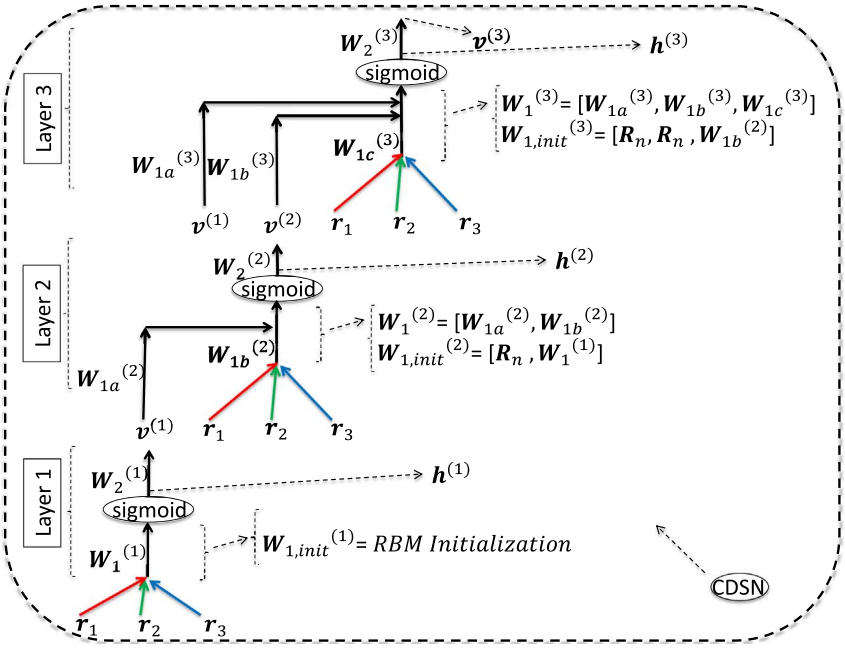
In (9), $\mathbf{v}^{(l)}$ is the output vector and $\mathbf{z}^{(l)}$ is the input vector of l-th layer and is defined as follows:

$$\mathbf{z}^{(l)} = [\mathbf{v}^{(1)}, \ \mathbf{v}^{(2)}, \ \ldots, \ \mathbf{v}^{(l-1)}, \mathbf{r}] \tag{10}$$

In (10), $\mathbf{r}$ is the vector formed by the concatenation of all



(a) Block diagram of the proposed method with convolution window size 3.



(b) Block diagram of the convoltional deep stacking network with 3 layers.

Fig. 1. Proposed method. This diagram shows CDSN for channel 2 in Fig. 1(a), and $\mathbf{R}_n$ denotes a random matrix.

residual vectors in each convolution window. To find the CDSN unknown parameters $\mathbf{W}_1^{(l)}$ and $\mathbf{W}_2^{(l)}$ for each layer, l, a mean squared error cost function is minimized:

$$\{\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}\} = \underset{\{\mathbf{W}_1^{(l)}, \mathbf{W}_2^{(l)}\}}{\operatorname{argmin}} \frac{1}{2} \left\| \mathbf{V}^{(l)} - \mathbf{T} \right\|_2^2 \tag{11}$$

where $\mathbf{T}$ is a matrix whose columns are the target vectors in the training set and $\mathbf{V}^{(l)}$ is a matrix whose columns are the corresponding output vectors from the l-th layer. Each layer of CDSN is a one layer neural network with a non-linear hidden layer and a linear output layer. In a CDSN, similar to a DSN [16], the optimization problem in (11) is solved for each layer separately. The linearity of the output layer for each layer of CDSN makes it possible to find a closed form solution for $\mathbf{W}_2^{(l)}$ given $\mathbf{W}_1^{(l)}$ and $\mathbf{T}$:

$$\mathbf{W}_2^{(l)} = \left[\mathbf{H}^{(l)} \left[\mathbf{H}^{(l)} \right]^T \right]^{-1} \mathbf{H}^{(l)} \mathbf{T}^T \tag{12}$$

where $\mathbf{H}^{(l)}$ is a matrix whose columns are $\mathbf{h}^{(l)}$ in (9) corresponding to different training samples in the training set. To prevent overfitting and to have a reliable solution for $\mathbf{W}_2^{(l)}$ when $\mathbf{H}^{(l)}$ is ill conditioned, usually an ℓ_2 regularization term is added to (11). In other words, to calculate $\mathbf{W}_2^{(l)}$ the following optimization problem is solved:

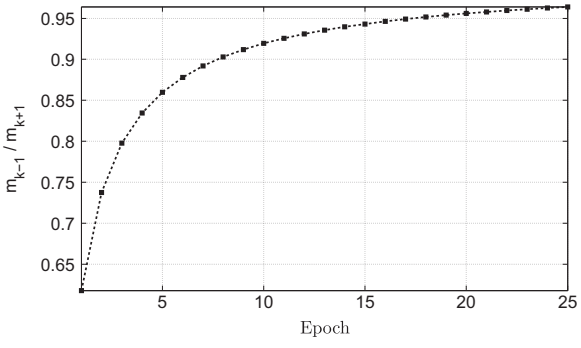


Fig. 2. The curve of FISTA coefficients $\frac{m_{k-1}}{m_{k+1}}$ in (16) with respect to the epoch number.

$$\mathbf{W}_2^{(l)} = \underset{\mathbf{W}_2^{(l)}}{\operatorname{argmin}} \frac{1}{2} \left\| \left[\mathbf{W}_2^{(l)}\right]^T \mathbf{H}^{(l)} - \mathbf{T} \right\|_2^2 + \mu \left\| \mathbf{W}_2^{(l)} \right\|_2^2 \tag{13}$$

which results in:

$$\mathbf{W}_2^{(l)} = \left[\mu \mathbf{I} + \mathbf{H}^{(l)} [\mathbf{H}^{(l)}]^T \right]^{-1} \mathbf{H}^{(l)} \mathbf{T}^T \tag{14}$$

where $\mathbf{I}$ is the identity matrix.

To find $\mathbf{W}_1^{(l)}$, for each layer of CDSN we use the stochastic gradient descent method. To calculate the gradient of the cost function with respect to $\mathbf{W}_1^{(l)}$ given the fact that $\mathbf{W}_2^{(l)}$ and $\mathbf{H}^{(l)}$ depend on $\mathbf{W}_1^{(l)}$, it can be shown [34] that the gradient of the cost function in (11) with respect to $\mathbf{W}_1^{(l)}$ is:

$$\frac{\partial \left\| \mathbf{V}^{(l)} - \mathbf{T} \right\|_2^2}{\partial \mathbf{W}_1^{(l)}} = \mathbf{Z}^{(l)} \left(\left[\mathbf{H}^{(l)} \right]^T \circ \left[1 - \mathbf{H}^{(l)} \right]^T \circ \left[\left[\mathbf{H}^{(l)} \right]^\dagger \left[\mathbf{H}^{(l)} \mathbf{T}^T \right] \left[\mathbf{T} [\mathbf{H}^{(l)}]^\dagger \right] - \mathbf{T}^T \left[\mathbf{T} [\mathbf{H}^{(l)}]^\dagger \right] \right] \right) \tag{15}$$

where $\mathbf{Z}^{(l)}$ is a matrix whose columns are $\mathbf{z}^{(l)}$ in (10) corresponding to different training samples in the training set and $\circ$ is the Hadamard product operator. Using the gradient information from past iterations can help to improve the convergence speed in convex optimization problems [35]. Although the problem in (11) is not necessarily convex because of the stack of non-linear hidden layers, but we found out experimentally that the gradient information from the past iterations can be helpful here as well. As in [34], we use the FISTA algorithm to accelerate the fine tuning. Therefore, the update equations for $\mathbf{W}_1^{(l)}$ at the k-th iteration are as follows:

$$\begin{aligned} \mathbf{W}_{1,k}^{(l)} &= \hat{\mathbf{W}}_{1,k}^{(l)} - \rho \frac{\partial \left\| \mathbf{V}^{(l)} - \mathbf{T} \right\|_2^2}{\partial \hat{\mathbf{W}}_{1,k}^{(l)}} \\ m_{k+1} &= \frac{1}{2} \left(1 + \sqrt{1 + 4 m_k^2} \right) \\ \hat{\mathbf{W}}_{1,k+1}^{(l)} &= \hat{\mathbf{W}}_{1,k}^{(l)} + \frac{m_{k-1}}{m_{k+1}} \left(\mathbf{W}_{1,k}^{(l)} - \mathbf{W}_{1,k-1}^{(l)} \right) \end{aligned} \tag{16}$$

The curve of the FISTA coefficients $\frac{m_{k-1}}{m_{k+1}}$ with respect to the epoch number is represented in Fig. 2. After computing $\mathbf{W}_1^{(l)}$ from (16), we use the closed form formulation in (14) to find $\mathbf{W}_2^{(l)}$.

Another important consideration for training the CDSN is that the cost function in (11) is not necessarily convex, therefore the initialization of $\mathbf{W}_1^{(l)}$ before fine tuning plays an important role. For initialization of the first layer of CDSN, we train a Restricted Boltzmann Machine (RBM) [36,30] with Gaussian visible units and binary hidden units. This results in the following energy function between visible units, i.e., entries of $\mathbf{z}^{(1)}$, and hidden units, i.e., entries of $\mathbf{h}^{(1)}$:

$$E\left(\mathbf{z}^{(1)}, \mathbf{h}^{(1)}\right) = \frac{1}{2} \left(\mathbf{z}^{(1)} - \mathbf{b}_1 \right)^T \left(\mathbf{z}^{(1)} - \mathbf{b}_1 \right) - \mathbf{b}_2^T \mathbf{h}^{(1)} - \left[\mathbf{z}^{(1)} \right]^T \mathbf{W}_{1,init}^{(1)} \mathbf{h}^{(1)} \tag{17}$$

where $\mathbf{b}_1$ and $\mathbf{b}_2$ are vectors of bias values for the visible and hidden units respectively. The goal is to find $\mathbf{W}_{1,init}^{(1)}$ from the training input data, i.e., the residual vectors, $\mathbf{z}^{(1)}$, in the training data generated as explained earlier. Then we use $\mathbf{W}_{1,init}^{(1)}$ to initialize $\mathbf{W}_1^{(1)}$ as shown in Fig. 1(b). This approach has been shown to be also helpful in training neural networks and specifically

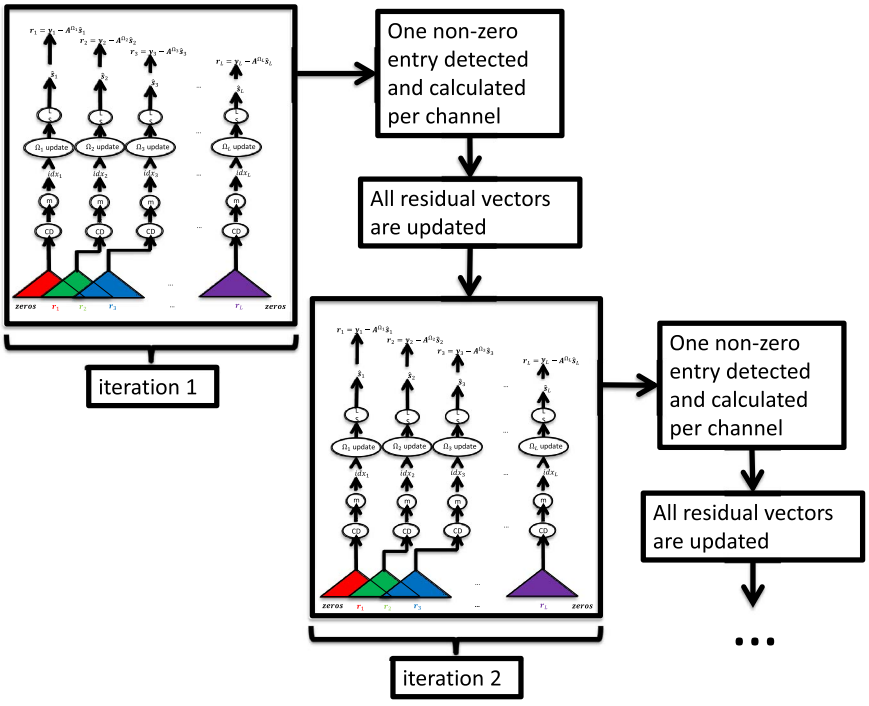


Fig. 3. High level block diagram of the proposed method.

autoencoders [37]. The parameters of the RBM can be found by maximizing the log probability that the RBM assigns to the input data, which is a function of the energy function in (17), using the contrastive divergence method [30]. The details on the general RBM training method used in this work can be found at [38]. As shown in the block diagram of CDSN in Fig. 1(a), to initialize the parameters of the upper layers of CDSN, $\mathbf{W}_1^{(l+1)}$, we use the learned parameters of the lower layer, $\mathbf{W}_1^{(l)}$, as initialization. This approach has been shown to be helpful in training DSNs [16] and it was helpful in our task as well. This completes the description of the training method for CDSN and the answer for question (ii).

Given the trained CDSN, a summary of the proposed reconstruction algorithm that finds the sparsest solution $\mathbf{S}$ given $\mathbf{Y}$ and $\mathbf{A}$ in (4) is presented in Algorithm 1. We refer to this algorithm as CDSN-CS since we have used a convolutional DSN for distributed compressive sensing. A more high level architecture of the proposed method is also presented in Fig. 3.

$$MSE = \frac{\| \hat{\mathbf{S}} - \mathbf{S} \|}{\| \mathbf{S} \|} \tag{18}$$

where $\mathbf{S}$ is the actual sparse matrix and $\hat{\mathbf{S}}$ is the sparse matrix recovered by the reconstruction algorithm. We encoded 8 blocks ($L=8$) of each image simultaneously using a random measurement matrix and reconstructed them at the decoder. Therefore, $\mathbf{S}$ in (4) had 8 columns and each column had $N=64$ entries. We used 40% measurements, i.e., $\mathbf{Y}$ in (4) had 8 columns and each column had $M=25$ entries. The encoder was a typical compressive sensing encoder, i.e., $\mathbf{A}$ was a randomly generated matrix. Each column of $\mathbf{A}$ was normalized to have unity norm. To simulate the measurement noise, Gaussian noise with standard deviation 0.005 was added to the measurement matrix $\mathbf{Y}$ in (4). We used two popular transforms, DCT and Wavelet, as the sparsifying basis $\mathbf{\Psi}$ in (2). For the wavelet transform we used the Haar wavelet transform with 3 levels of decomposition. We used 55 images for the training set,

Algorithm 1. Distributed Compressive Sensing using Convolutional Deep Stacking Network (CDSN-CS).

Inputs: CS measurement matrix $\mathbf{A} \in \Re^{M \times N}$; matrix of measurements $\mathbf{Y} \in \Re^{M \times L}$; minimum ℓ_2 norm of residual matrix "*resMin*" as stopping criterion; Trained "*cdsn*" model; Convolution window size " *w*"

Output: Matrix of sparse vectors $\hat{\mathbf{S}} \in \Re^{N \times L}$

Initialization: $\hat{\mathbf{S}} = 0$; $j=1$; $i=1$; $\Omega = \varnothing$; $\mathbf{R} = \mathbf{Y}$.

procedure CDSN-CS $\mathbf{A}$, $\mathbf{Y}$, *cdsn*
  while $i \le N$ and $\| \mathbf{R} \|_2 \le resMin$ **do**
    $i \leftarrow i + 1$
    for $j = 1 \rightarrow L$ **do**
      $\mathbf{R}(:, j)_i \leftarrow \frac{\mathbf{R}(:, j)_{i-1}}{max(|\mathbf{R}(:, j)_{i-1}|)}$
      $\mathbf{v}_j \leftarrow cdsn\left(\left[\mathbf{R}\left(:, j - \frac{w}{2}\right)_i, \mathbf{R}\left(:, j - \frac{w}{2} + 1\right]_i, \ldots, \mathbf{R}\left(:, j + \frac{w}{2} - 1\right)_i, \mathbf{R}\left(:, j + \frac{w}{2}\right)_i\right)\right)$
      $idx \leftarrow Support(\max(\mathbf{v}_j))$
      $\Omega_i \leftarrow \Omega_{i-1} \cup idx$
      $\hat{\mathbf{S}}^{\Omega_i}(:, j) \leftarrow (\mathbf{A}^{\Omega_i})^{\dagger}\mathbf{Y}(:, j)$ ▷Least Squares
      $\hat{\mathbf{S}}^{\Omega_i^C}(:, j) \leftarrow 0$
      $\mathbf{R}(:, j)_i \leftarrow \mathbf{Y}(:, j) - \mathbf{A}^{\Omega_i}\hat{\mathbf{S}}^{\Omega_i}(:, j)$
    end for
  end while
end procedure

4. Experimental evaluation and discussion

In this section we experimentally demonstrate: (i) How is the performance of the proposed method compared to other reconstruction algorithms discussed in this paper? (ii) How fast is the proposed method? (iii) What are the effects of the convolution window size in CDSN-CS? (iv) What are the effects of the RBM initialization?

To address the above issues, we performed experiments on three different classes of images belonging to the natural image dataset of Microsoft Research in Cambridge [39]. Ten randomly selected test images from each class of this dataset were used for experiments. The images are shown in Fig. 4. The size of each of the used images was 64×64. Each image was divided into 8×8 non-overlapping blocks. After reconstructing all the blocks of an image, the reconstruction error for the reconstructed image was calculated. The reconstruction error is defined as:

5 images for the validation set and 10 images for the test set. The PC used to perform the experiments had an Intel(R) Core(TM) i7 CPU with clock 2.93 GHz and with 16 GB RAM.

The performance of the proposed reconstruction algorithm (CDSN-CS) was compared with 5 reconstruction methods for the MMV problem. These methods are: (1) Simultaneous Orthogonal Matching Pursuit (SOMP) which is a well known baseline for the MMV problem, (2) Bayesian Compressive Sensing (BCS) [17] applied independently on each channel, (3) Multitask Compressive Sensing (MT-BCS) [14] which takes into account the statistical dependency among the different channels, (4) Sparse Bayesian Learning for Temporally correlated sources (T-SBL) [15] which exploits the correlation among different sources in the MMV problem and (5) Nonlinear Weighted SOMP (NWSOMP) [25]. For the BCS method we set the initial noise variance of i-th channel to the value suggested by the authors, i.e., $std(\mathbf{y}_i)^2/100$ where $i \in \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $std(\cdot)$ calculates the standard

Fig. 4. Natural images randomly selected from three different classes used for test. The first row is "buildings", the second row is "cows" and the third row is "flowers".

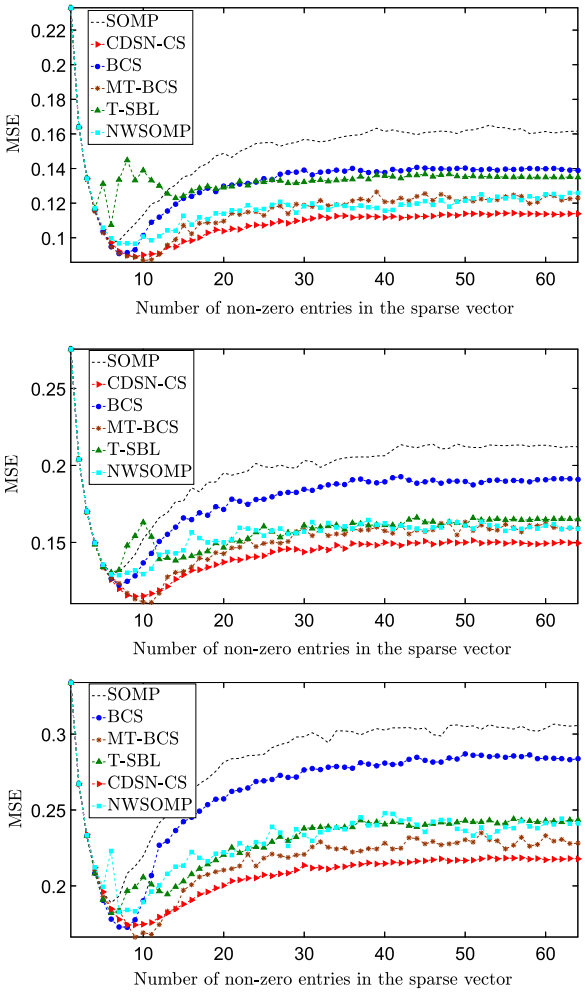


Fig. 5. Comparison of different MMV reconstruction algorithms performance for the natural image dataset using DCT transform. Image classes from top to bottom are buildings, cows and flowers respectively.

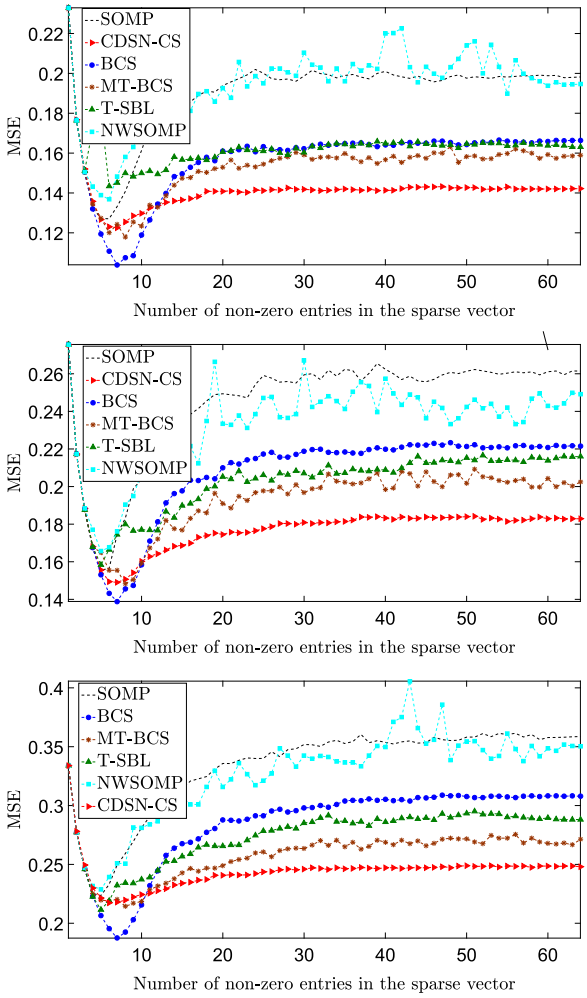


Fig. 6. Comparison of different MMV reconstruction algorithms performance for the natural image dataset using Wavelet transform. Image classes from top to bottom are buildings, cows and flowers respectively.

deviation. The threshold for stopping the algorithm was set to 10^{-8}. For MT-BCS we set the parameters of the Gamma prior on noise variance to $a = 100/0.1$ and $b=1$ which are the values suggested by the authors. We set the stopping threshold to 10^{-8} as well. For T-SBL, we used the default values recommended by the authors. For NWSOMP, during training, we used three layers, each layer having 512 neurons and 25 epochs of parameters update. For CDSN-CS, during training, we used three layers, 64 neurons per layer with different window sizes and 25 epochs of parameter updates. For RBM initialization, we ran 200 epochs of RBM parameter update with step size 0.01. To monitor overfitting of the RBM, we used free energy as explained in [38]. For fine tuning CDSN-CS after RBM initialization we used step size 0.002. The regularization parameter μ in (14) was set to 0.01. To monitor and prevent overfitting, we used 5 images per channel as the validation set and we used early stopping if necessary. Note that the images used for validation were different from those used in the training set or in the test set.

The results for the different classes of images are presented in Fig. 5 for the DCT transform and in Fig. 6 for the Wavelet transform. In these figures, the vertical axis is the *MSE* defined in (18) and the horizontal axis is the number of non-zero entries in each sparse vector. The number of measurements, M, is fixed to 25. Each point

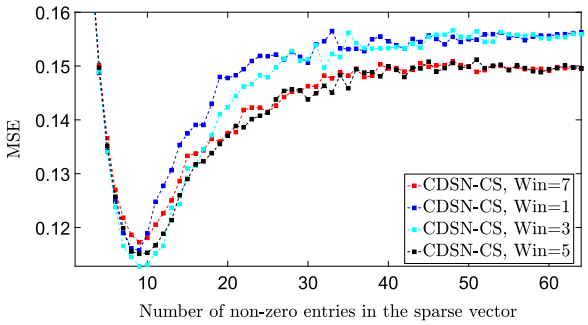


Fig. 7. The performance of CDSN-CS with different window sizes. This experiment was conducted for image class of cows with DCT transform as sparsifying basis.

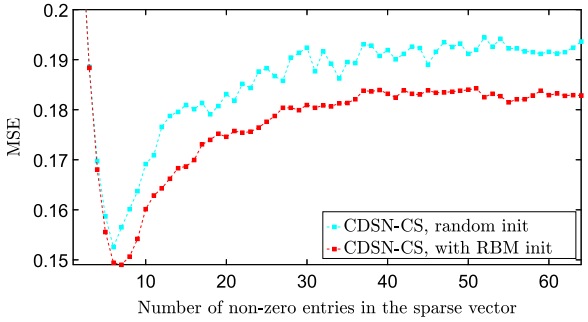


Fig. 8. The performance of CDSN-CS with and without RBM initialization. This experiment was conducted for image class of cows with Wavelet transform as sparsifying basis.

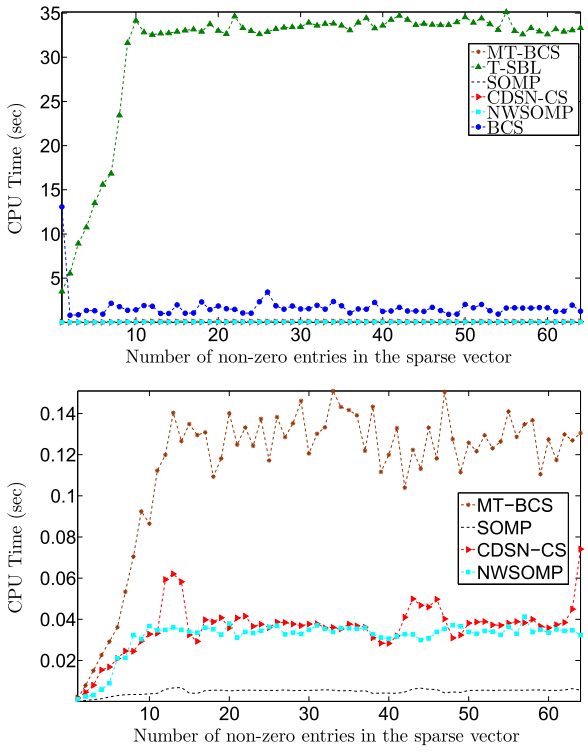


Fig. 9. CPU time of different MMV reconstruction algorithms discussed in this paper. Up: all methods. Bottom: removing T-SBL and BCS to make the difference among remaining methods more visible.

on the curves is the average of the *MSE*s over 10 reconstructed test images at the decoder. For all results, we used a convolution window of size 5 because it gave the best performance compared to other window sizes. For image class of flowers (the bottom part of Figs. 5 and 6), a convolution window of size 7 gave better performance. As observed in these figures, CDSN-CS outperforms the five reconstruction methods SOMP, BCS applied to each channel independently, MT-BCS, T-SBL and NWSOMP. We believe that this improvement in the performance is due to exploiting the dependencies among the different channels by CDSN network.

To study the effect of the convolution window size, a comparison among the different convolution window sizes in CDSN-CS for the image class "*cows*" with the DCT transform is presented in Fig. 7. As observed from this figure, increasing the window size improves the results up to a point after which the results do not improve any more. Since we use distinct patches from each image, we might assign this behavior to the fact that the residuals of image patches that are far from each other might be less correlated than the residuals of image patches that are close to each other.

To show that RBM initialization is helpful for our task, we conducted two experiments. In the first experiment the CDSN is trained using random initialization. In the second experiment it is trained using RBM initialization. The results are presented in Fig. 8. As observed in this figure, RBM initialization improves the reconstruction performance.

To conclude this section we present the CPU time for the different reconstruction algorithms discussed in this section in Fig. 9. Since all methods are run in MATLAB and on the same machine, Fig. 9 gives a rough idea about how fast the different methods discussed in this paper are. As observed in this figure, the proposed method is faster than the Bayesian methods discussed and is almost as fast as the greedy methods.

5. Conclusions and future work

This work proposes a method based on Convolutional Deep Stacking Networks (CDSN) to reconstruct sparse vectors in the MMV problem (from random measurements). The proposed method improves the performance of sparse reconstruction methods by exploiting the dependency among different channels. We showed that the performance of the proposed method was better than SOMP and a number of model based Bayesian methods. The proposed method is a data driven method that learns the inter-channel information among the sparse vectors in the MMV problem (during training) and uses this information during reconstruction. We have experimentally shown that the proposed method is almost as fast as greedy methods. Note that we did not use a huge training set, or advanced deep learning methods like drop out or many layers of representation all of which are expected to improve upon our results presented here. This paper constitutes a proof of concept that convolutional neural networks, and specifically CDSN, can improve the performance of distributed compressive sensing reconstruction algorithms by capturing the dependency information among the sparse vectors. In our future work we plan to apply the proposed method to applications where a significant amount of dependency among the sparse vectors exists. Examples are video compressive sensing and health telemonitoring that use compressive sensing of different electroencephalogram (EEG) channel signals.

References

[1] Li Deng, Gokhan Tur, Xiaodong He, Dilek Hakkani-Tür, Use of Kernel Deep Convex Networks and End-To-End Learning for Spoken Language Understanding, IEEE Workshop on Spoken Language Technologies, December 1, 2012.

[2] Li Deng, Dong Yu, Deep Convex Network: A Scalable Architecture for Speech Pattern Classification, in Interspeech, International Speech Communication

Association, August 1, 2011.
[3] Brian Hutchinson, Dong Yu, Li Deng, Tensor Deep Stacking Networks, in IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE, August 1, 2013.
[4] Li Deng, Xiaodong He, Jianfeng Gao, Deep Stacking Networks for Information Retrieval, IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), May 1, 2013.
[5] Li Deng, A Tutorial Survey of Architectures, Algorithms, and Applications for Deep Learning, in APSIPA Transactions on Signal and Information Processing, Cambridge University Press, January 1, 2014.
[6] D. Donoho, Compressed sensing, IEEE Trans. Inf. Theory 52 (4) (2006) 1289–1306.
[7] E. Candes, J. Romberg, T. Tao, Stable signal recovery from incomplete and inaccurate measurements, Commun. Pure Appl. Math. 59 (8) (2006) 1207–1223.
[8] R. Baraniuk, Compressive sensing (lecture notes), IEEE Signal Process. Mag. 24 (4) (2007) 118–121.
[9] Y. Eldar, H. Rauhut, Average case analysis of multichannel sparse recovery using convex relaxation, IEEE Trans. Inf. Theory 56 (1) (2010) 505–519.
[10] M. Davies, Y. Eldar, Rank awareness in joint sparse recovery, IEEE Trans. Inf. Theory 58 (2) (2012) 1135–1146.
[11] J. Tropp, A. Gilbert, M. Strauss, Algorithms for simultaneous sparse approximation. Part I: greedy pursuit, Signal Process. 86 (3) (2006) 572–588.
[12] J. Tropp, Algorithms for simultaneous sparse approximation. Part II: convex relaxation, Signal Process. 86 (3) (2006) 589–602.
[13] D.P. Wipf, B.D. Rao, An empirical Bayesian strategy for solving the simultaneous sparse approximation problem, IEEE Trans. Signal Process. 55 (7) (2007) 3704–3716.
[14] S. Ji, D. Dunson, L. Carin, Multitask compressive sensing, IEEE Trans. Signal Process. 57 (1) (2009) 92–106.
[15] Z. Zhang, B.D. Rao, Sparse signal recovery with temporally correlated source vectors using sparse Bayesian learning, IEEE J. Sel. Top. Signal Process. 5 (5) (2011) 912–926.
[16] L. Deng, D. Yu, J. Platt, Scalable stacking and learning for building deep architectures, in: Proceedings of ICASSP, 2012, pp. 2133–2136.
[17] S. Ji, Y. Xue, L. Carin, Bayesian compressive sensing, IEEE Trans. Signal Process. 56 (6) (2008) 2346–2356.
[18] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[19] G. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[20] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97.
[21] G. Dahl, D. Yu, L. Deng, A. Acero, Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition, IEEE Trans. Audio Speech Lang. Process. 20 (1) (2012) 30–42.
[22] R. Baraniuk, V. Cevher, M. Duarte, C. Hegde, Model-based compressive sensing, IEEE Trans. Inf. Theory 56 (4) (2010) 1982–2001.
[23] M. Duarte, Y. Eldar, Structured compressed sensing: from theory to applications, IEEE Trans. Signal Process. 59 (9) (2011) 4053–4085.
[24] D. Merhej, C. Diab, M. Khalil, R. Prost, Embedding prior knowledge within compressed sensing by neural networks, IEEE Trans. Neural Netw. 22 (10) (2011) 1638–1649.
[25] H. Palangi, R. Ward, L. Deng, Using deep stacking network to improve structured compressed sensing with multiple measurement vectors, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.
[26] A. Mousavi, A.B. Patel, R.G. Baraniuk, A deep learning approach to structured signal recovery, arXiv:abs/1508.04065.
[27] H. Palangi, R. Ward, L. Deng, Distributed compressive sensing: a deep learning approach, arXiv:abs/1508.04924.
[28] M.E. Tipping, Sparse Bayesian learning and the relevance vector machine, J. Mach. Learn. Res. 1 (2001) 211–244.
[29] A.C. Faul, M.E. Tipping, Analysis of sparse Bayesian learning, in: Advances in Neural Information Processing Systems (NIPS), vol. 14, MIT Press, Vancouver, BC, Canada, 2001, pp. 383–389.
[30] G.E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Comput. 14 (8) (2002) 1771–1800.
[31] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: ICML, 2008, pp. 1096–1103.
[32] P. Vincent, A connection between score matching and denoising autoencoders, Neural Comput. 23 (7) (2011) 1661–1674.
[33] M. Aharon, M. Elad, A. Bruckstein, k -svd: an algorithm for designing overcomplete dictionaries for sparse representation, IEEE Trans. Signal Process. 54 (11) (2006) 4311–4322.
[34] D. Yu, L. Deng, Efficient and effective algorithms for training single-hidden-layer neural networks, Pattern Recognit. Lett. (2012) 554–5580.
[35] A. Beck, M. Teboulle, Gradient-based algorithms with applications to signal-recovery problems, in: Yonina Eldar, Daniel Palomar (Eds.), Convex Optimization in Signal Processing and Communications, Cambridge University Press, Cambridge university press, 2009.
[36] P. Smolensky, Information processing in dynamical systems: foundations of harmony theory in: D.E. Rumelhart, J.L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, MIT Press, Cambridge, MA, USA, 1986, pp. 194–281.
[37] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[38] G. Hinton, A Practical Guide to Training Restricted Boltzmann Machines: Version 1, Technical Report, University of Toronto, 2010.
[39] ⟨http://research.microsoft.com/en-us/projects/objectclassrecognition⟩.