Dictionary Learning Based Applications in Image Processing using Convex Optimisation

Abhay Kumar (12011)
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Email: abhayk@iitk.ac.in

Saurabh Kataria (12807637)
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Email: saurabhk@iitk.ac.in

***Abstract*—In this term paper, sparse based representation of images has been exploited for various applications. Sparse and redundant representation is based on the assumptions that the signals can be described as a linear combination of a few atoms from the pre-defined dictionary. Dictionary atoms is either selected using k-SVD algorithm or taken as standard DCT atoms. Dictionary atoms can lso be set by randomly sampling patches from the image. Different applications of sparse based representation have been presented. Image Inpainting, classification and Image denoising applications based on sparse representation are presented. Sparse representation based applications have very similar performances with the state of the art methods.**

Keywords—Sparse, OMP, K-SVD, DCT.

I. Introduction

Natural signals such as images admit a sparse representation i.e. they can be represented as a linear combination of few vectors (atoms of a learned dictionary). It has been established that there is much redundancy in natural signals. It can be represented in terms of few basis vectors. Since the sparse representation consists of mostly zeros, we only have to store only the non-zero elements. Generally, Sparse based approaches are computationally more efficient in terms of computation and memory requirements. Sparse representation are also discriminative in nature and are extensively used for classification purpose as well.

Sparse representations of signals have received a lot of interests in recent years. Sparse representation is exploited to give the most compact representation of a signal. Any signal can be represented as a linear combination of atoms in an overcomplete dictionary. Recent developments in multi-scale and multi-orientation representation of signals are an important incentive for the research on the sparse representation. Different dictionary learning algorithms such as matching pursuit, orthogonal matching pursuit, method of optimal direction (MOD) or k-singular value decomposition (k-SVD) are extensively popular in the literature. Sparse representation have been exploited for various applications such as signal separation, denoising, classification, image inpainting and reconstruction. Overcomplete dictionary has Using an overcomplete dictionary contains prototype of signal-atoms or combined multiple standard transforms, like curvelet transform, ridgelet transform and discrete cosine transform(DCT).

Sparse representation is highly dependent on the basis chosen. The number of atoms that represent a given signal i.e the sparsity varies with the basis chosen. The task to find the smallest set of atoms that represent a given signal is NP-hard. Hence, several approximation algorithms have been suggested, such as the matching pursuit, the basis pursuit, FOCUSS, and their variants.

A. Sparse Representation of Signals and Images

Given an overcomplete dictionary ϕ^{mxk} that contains K dictionary atoms, a signal $\mathbf{y} \in \mathbf{R}^m$ can be represented in terms of a vector $\mathbf{x} \in \mathbf{R}^k$ that contains the representation coefficients of the signal $\mathbf{y}$. The sparse representation problem can be formulated as:

$$\min_{\mathbf{x}} ||\mathbf{x}||_{\mathbf{0}} \quad \mathbf{s.t.} \quad \mathbf{Dx} = \mathbf{y} \tag{1}$$

where $||\mathbf{x}||_{\mathbf{0}}$ is the l_0 norm, that counts the total number of non-zero elements in $\mathbf{x}$. Also, the above optimisation problem is not convex and the solution for the l_0 norm has been shown to be NP-hard The l_0 is relaxed to l_1 norm. In order for the signal reconstruction to be robust to noise, above optimisation problem is formulated as:

$$\min_{\mathbf{x}} ||\mathbf{x}||_{\mathbf{1}} \quad \mathbf{s.t.} \quad ||\mathbf{y} - \mathbf{Dx}||_{\mathbf{2}} < \epsilon \tag{2}$$

where ϵ is the permitted error in the reconstruction Least Absolute Shrinkage and Selection Operator (LASSO) is a method proposed by Tibshirani [1] to solve the l_1 -minimization problem more efficiently. LASSO finds an estimate of $\mathbf{x}$ by minimizing the least square error subject to a l_1 norm constraint in the solution vector, formulated as:

$$\min_{\mathbf{x}} \frac{1}{2}||\mathbf{y} - \mathbf{Dx}||_2 + \beta||\mathbf{x}||_{\mathbf{1}} \tag{3}$$

where $\beta > 0$ is the parameter that controls the trade-off between the sparsity and the reconstruction error. This convex optimisation problem will give the globally optimal solution. On the contrary, various greedy algorithms like Matching Pursuit (MP), Orthogonal Matching Pursuit (OMP), and Compressive Sampling Pursuit (CoSaMP) try to find the best solution to the problem iteratively. Greedy algorithms may not find the optimal solution but find a local solution that approximates to the global solution.

The rest of the paper is organized as follows. Section II presents the state of the art dictionary learning algorithms. Three different applications of sparse representation in image processing with performance evaluation and the results is presented in section III. A brief conclusion is presented in Section IV and Section V gives the scope of future work in this respect.

II. Dictionary Learning and Sparse Representation

While pre-constructed or adapted dictionaries typically lead to fast transforms (of complexity O(n log n) instead of nm in a direct use), they are typically limited in their ability to sparsify the signals they are designed to handle. Furthermore, most of those dictionaries are restricted to signals/images of a certain type, and cannot be used for a new and arbitrary family of signals of interest. This leads us to yet another approach for obtaining dictionaries that overcomes these limitations by adopting a learning point-of-view. This learning option starts by building a training database of signal instances, similar to those anticipated in the application, and constructing an empirically learned dictionary, in which the generating atoms come from the underlying empirical data, rather than from some theoretical model [2]. Such a dictionary can then be used in the application as a fixed and redundant dictionary. We wish to estimate D given the model deviation. Consider the following optimization problem:

$$\min_{D,\{\mathbf{x_i}\}_1^{\mathbf{M}}} \sum_{i=1}^{M} ||\mathbf{x_i}||_\mathbf{0} \quad \text{subject to} ||\mathbf{y_i} - \mathbf{Dx_i}||_2 \leq \epsilon, \quad \mathbf{1 \leq i \leq M} \tag{4}$$

Alternatively, above problem can also be formulated as:

$$\min_{D,\{\mathbf{x_i}\}_1^{\mathbf{M}}} \sum_{i=1}^{M} ||\mathbf{y_i} - \mathbf{Dx_i}||_2^2 \quad \text{subject to} \quad ||\mathbf{x_i}||_\mathbf{0} \leq \mathbf{k_0}, \quad \mathbf{1 \leq i \leq M} \tag{5}$$

A. *Method of Optimal Direction (MOD))*

Above problems can be posed as a nested minimization problem: an inner minimization of the number of nonzeros in the representation vectors $\mathbf{x_i}$, for a given fixed D and an outer minimization over D [3]. A strategy of alternating minimization thus seems very natural. At the k-th iteration, we use the dictionary $D_{(k-1)}$ from the $k-1$th step and solve M instances of entry $\mathbf{y_i}$, and each using the dictionary $D_{(k-1)}$. This gives us the matrix $X_{(k)}$, and we then solve for $D_{(k)}$ by Least-Squares.

$$D_{(k)} = \underset{D}{\operatorname{argmin}} ||Y - DX_{(k)}||_F^2 \tag{6}$$
$$= YX_{(k)}^T(X_{(k)}X_{(k)}^T)^{-1} \tag{7}$$
$$= YX_{(k)}^+ \tag{8}$$

where $X_{(k)}^+$ is the pseudo-inverse.

B. *k-singular value decomposition (k-SVD)*

The sparse representation problem can be viewed as a generalization of the Vector Quantisation objective function, in which we allow each input signal to be represented by a linear combination of codewords. In dictionary learning, dictionary atoms are the analogical to the codewords. Now, coefficients vector can have more than one nonzero entry and arbitrary values. The objective problem for k-SVD dictionary learning is:

$$\min_{\mathbf{D,X}} ||\mathbf{Y} - \mathbf{DX}||_\mathbf{F}^\mathbf{2} \quad \text{subject to} \quad ||\mathbf{x_i}||_\mathbf{0} \leq \mathbf{k_0} \quad \forall \mathbf{i} \tag{9}$$

where $\mathbf{Y} = \{\mathbf{y_i}| \text{ i in } [\mathbf{1}, \mathbf{K}], \mathbf{y_i} \in \mathbf{R^n}\}$ and $\mathbf{X}$ is formed by column stacking all vectors $\mathbf{x_i}$ and $||||_F^2$ denotes the Frobenius norm square.

General views of the different steps in k-SVD algorithm are mentioned here:

- The K-SVD algorithm [4] tries to minimize the cost function iteratively. It first find a sparse representation for the given signals using the standard OMP algorithm, using an initial estimate of the dictionary. This coding is sought such that it minimizes the error in representation, and at the same time maintain the sparsity constraint
- After the sparse coding stage, the algorithm proceeds to update the atoms of the dictionary. One atom of the dictionary is updated at a time such that the error term is further reduced. Proceeding in similar manner, the algorithm reduces error of representation at each iteration.
  The representation error term can thus be modified and written as

$$||\mathbf{Y} - \mathbf{DX}||_\mathbf{F}^\mathbf{2} = ||\mathbf{Y} - \textstyle\sum_{\mathbf{j=1}}^{\mathbf{K}} \mathbf{d_j x_j^r}||_\mathbf{F}^\mathbf{2} \tag{10}$$
$$= ||\mathbf{Y} - \textstyle\sum_{\mathbf{j \neq k}} \mathbf{d_j x_j^r} - \mathbf{d_k x_k^r}||_\mathbf{F}^\mathbf{2} \tag{11}$$
$$= ||\mathbf{E_k} - \mathbf{d_k x_k^r}||_\mathbf{F}^\mathbf{2}. \tag{12}$$

- Error term has two parts depending on whether when the atom $\mathbf{d_k}$ is not taken into account or nor. Also, we have achieved the decomposition of the multiplication of matrices into a summation of K rank-1 matrices. Of these, the first $K-1$ are assumed to fixed. The problem of minimizing the total error thus boils down to finding a rank-1 matrix which best approximates the error matrix E_k. We can't perform singular value decomposition on Ek and use the largest singular value and its corresponding vector for estimating the matrix as this doesn't take into account the sparsity constraint of the resulting $\mathbf{X}$ matrix.
- We first need to identify all the signals that use the k-th atom of the dictionary. Then split the error into two terms, one term defining the error of representation of those signals with the $\mathbf{d_k}$ atom removed and the rest for all other atoms

$$||\mathbf{Y} - \mathbf{DX}||_\mathbf{F}^\mathbf{2} = ||\mathbf{E_k^R} - \mathbf{d_k x_k^R}||_\mathbf{F}^\mathbf{2} \tag{13}$$

  where $\mathbf{E_k^R}$ takes into account the error for just those signals that are supported by the atom $\mathbf{d_k}$. Now the The error function minimization can be carried out by a rank-1 approximation of the $\mathbf{E_k^R}$ matrix using a singular value decomposition.

C. *Orthogonal Matching Pursuit (OMP)*

The Orthogonal Matching Pursuit (OMP) algorithm [5], [6] is a greedy algorithm with attempts to find a sparse representation of a signal given a specific dictionary. The algorithm attempts to find the best basis vectors (atoms) iteratively such that in each iteration the error in representation is reduced. This achieved by selection of that atom from the dictionary which has the largest absolute projection on the error vector. This essentially implies that we select that atom, which adds the maximum information and hence maximally reduces the error in reconstruction. Given a signal vector y and a dictionary D the algorithm attempts to find the code vector x in three steps

- Select the atom which has maximal projection on the residual
- Update $\mathbf{x^k} = \text{argmin}_{\mathbf{x}}^{\mathbf{k}} \ ||\mathbf{y} - \mathbf{Dx^k}||_2$
- Update the residual $\mathbf{r^k} = \mathbf{y} - \mathbf{y^k}$

III. Three Applications of Dictionary Learning and sparse representation in Image Processing

In this section, three different applications based on sparse representation , namely Image Inpainting , Image Denoising and Image classification have been presented.

A. Image Impainting

Image Inpainting is a method of filling up the missing pixels in an image with the help of the existing pixels. Inpainting is often referred to as disocclusion, meaning removing a obstruction or unmask a masked image. The success of inpainting lies on how well it infers the missing pixels from the observed pixels[7], [8]. Broadly there are there are two main appoaches for image inpainting: PDE based approaches and exemplar based approaches. The formers is based on structure propagation while the later approaches adopt texture synthesis method to synthesize the pixels in the user specified region. Given the dictionary $\mathbf{D} = [\mathbf{d^1}, \mathbf{d^2}, ..., \mathbf{d^N}]$ and the input signal $\mathbf{y} = [\mathbf{y^1}, \mathbf{y^2}, ..., \mathbf{y^P}]^\mathbf{T}$, the coefficients can be estimated using the Lasso algorithm:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\text{argmin}} \ ||\mathbf{y} - \mathbf{Dx}||_2^2 + \beta||\mathbf{x}||_\mathbf{1} \tag{14}$$

The term $||\mathbf{x}||_\mathbf{1}$ encourages the sparsity of the coefficient vector and β controls the tradeoff between reconstruction error and sparsity. We can generalize the above formulation accounting the corrupted components as well.

$$\mathbf{y} = \mathbf{Dx} + \mathbf{e} \tag{15}$$

where $\mathbf{e}$ is the error and e_i is non-zero for the corrupted component. Target region to be inpainted is known beforehand i.e the indices of the corrupted components of y are known. Lets denote the corrupted components index set by I where $I = \{i|e_i \neq 0\}$. $\mathbf{y_{\backslash I}}$ denotes the denotes the vector obtained by removing the corrupted components i.e whose indexes are in I from $\mathbf{y}$. $\mathbf{D_{\backslash I}}$is the corresponding reduced dictionary matrix, obtained after removing all the columns whose indexes are in I from $\mathbf{D}$.Now the sparse coefficient $\mathbf{x}$ can be estimated as follows.

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\text{argmin}} \ ||\mathbf{y}_{\backslash I} - \mathbf{D}_{\backslash I}\mathbf{x}||_2^2 + \beta||\mathbf{x}||_\mathbf{1} \tag{16}$$

Now the corrupted image is recovered by the following rule:

$$\hat{y}_i = \begin{cases} y_i, & \text{if } i \notin I \\ (\mathbf{D}\hat{\mathbf{x}})_i, & \text{if } i \in I \end{cases} \tag{17}$$

The user selects the region to be removed or filled. Lets denote this 'target region' as Ω and the remaining region is called the 'source region' and it is denoted as Φ. The boundary of target region is denoted by $\delta\Omega$. Now, priority $P(p)$ is computed for all pixels on the target boundary. Pixel with maximum priority is filled or removed first.

Now consider the k-dimensional patch centred at p_m. Denote the patch as Ψ_{p_m}. Some pixels of the patch belongs to the source region and other belongs to the target regions. Recover the corrupted pixels of the patch Ψ_{p_m} using equations (16) and (17).

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\text{argmin}} \ ||\mathbf{\Psi}_{p_m \backslash I} - \mathbf{D}_{\backslash I}\mathbf{x}||_2^2 + \beta||\mathbf{x}||_\mathbf{1} \tag{18}$$

$$\hat{y}_i = \begin{cases} \mathbf{\Psi}^i_{p_m}, & \text{if } i \notin I \\ (\mathbf{D}\hat{\mathbf{x}})_i, & \text{if } i \in I \end{cases} \tag{19}$$

Algorithm 1 : Image Impainting via sparse representation

```
Input :Image, source region $\Phi$,target region $\Omega$ (user specified)
Dictionary Construction : Either learn a dictionary by k-SVD, MOD methods or take some standard overcomplete dictionary. Dictionary can also be formed by sampling random patches from the source region of the given image.
Recovery of corrupted pixels in target region :
    • compute priority for all pixels on the target boundary.
    • Select the patch with maximum priority.
    • Use reconstruction equations (18) and (19) to recover the corrupted pixels.
    • set the pixel of target region of the selcected patch to the recovered values. Update the target boundary $\delta\Omega$.
output : Inpainted image.
```

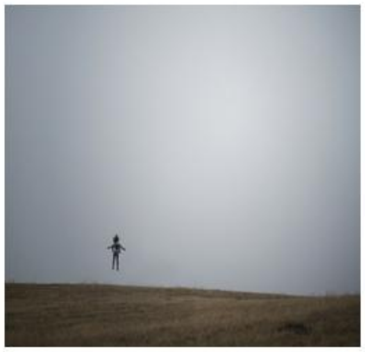

Fig. 1. Input Image [9]

Comment on image inpainting for various values of β:

- In this case, inpainting is better for lower value of sparsity i.e higher value of β . This is mainly because of distinct dictionary atoms, which are randomly choosen. The dictionary atoms seems to be sparse so that lesser number of distinct atoms is sufficient enough to get better reconstruction.
- Since we have not taken into account any edge information or structure of image explicitly, reconstruction

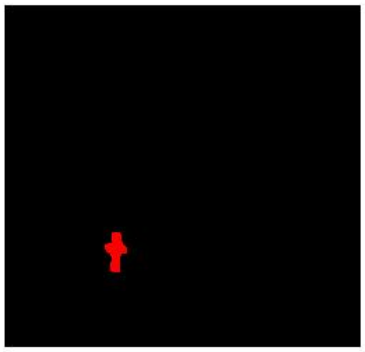

Fig. 2. Binary image illustrating the target region

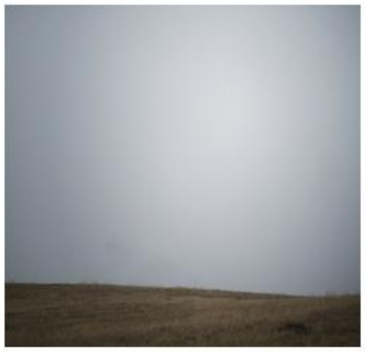

Fig. 3. Inpainted image after removing the target region $\beta = 10$

may not be good enough sometimes (generally for complex images).

B. Image Denoising

From the theory it is obvious that if the signal vectors have a high dimensionality, the size of the dictionary needs to be large for a stable acceptable reconstruction. This may not be the most suitable thing to do as the computational complexity scales up with the size of the vectors and the number of atoms in the dictionary. To address this issue the authors propose to break up the noisy image into patches and treat the vectorized version of each patch as signals, thereby restricting the dimensionality of each atom in the dictionary. However, the size of the patch has to be chosen such that it encodes enough details of the underlying signal. It is also natural to select these patches to be overlapping in nature in order to reduce blocking artefacts that might result at the boundaries. Dealing with patches as signals, the K-SVD algorithm can be effectively scaled to denoise large images.

Application of the method to denoising necessitates posing the problem of denoising first. For a given image, which can now be thought of as a set of signals Y, the denoising problem can be stated as the ill-posed problem of finding a set of

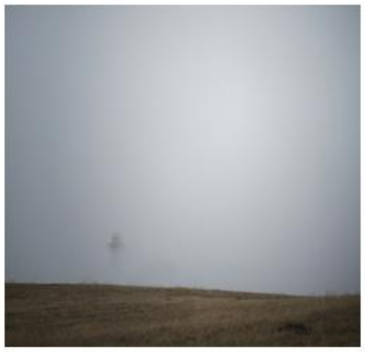

Fig. 4. Inpainted image after removing the target region with $\beta = 1$

patches Z which are related by

$$\mathbf{Y} = \mathbf{Z} + \eta \tag{20}$$

[10], [11] where η is assumed to be the noise which corrupts the patches. The noise over the entire image is assumed to be zero mean Gaussian noise. In order to find the denoised image patches Z, we define an optimization problem which involves minimization of the cost function:

$$\hat{\mathbf{X}}, \hat{\mathbf{Z}} = \underset{\mathbf{X},\mathbf{Z}}{\text{argmin}} \ ||\mathbf{Z} - \mathbf{Y}||_2^2 + \beta||\mathbf{D}\mathbf{X} - \mathbf{Z}||_F^2 + \textstyle\sum_i \mu_i ||\mathbf{x_i}||_\mathbf{0} \tag{21}$$

This can be viewed as solving a set of smaller optimization problems which is defined by:

$$\hat{\mathbf{x}}_\mathbf{i}, \hat{\mathbf{Z}} = \underset{\mathbf{x_i},\mathbf{Z}}{\text{argmin}} \ ||\mathbf{Z} - \mathbf{Y}||_2^2 + \beta||\mathbf{D}\mathbf{x}_i - \mathbf{R}_i\mathbf{Z}||_F^2 + \textstyle\sum_i \mu_i ||\mathbf{x_i}||_\mathbf{0} \tag{22}$$

where $\mathbf{R}_i$ is defined as the matrix which selects the ith patch from $\mathbf{Z}$ i.e. $\mathbf{z_i} = \mathbf{R_i}\mathbf{Z}$. This cost function allows us to minimize the error between the restored image and the input noisy one, under the assumption that each patch in the input image can be represented as a sparse linear combination of patches in the dictionary $\mathbf{D}$. Ideally, for denoising the first term should be rewritten as $||\mathbf{ZY}||_\mathbf{2}^\mathbf{2} < \mathbf{C}\sigma^\mathbf{2}$ where C is a constant and σ^2 is the variance of the noise. However, this term is implicitly incorporated into the cost function in the selection of the β parameter which will depend of the noise variance. The closed form solution to this cost function is given by

$$\hat{\mathbf{Z}} = \frac{\beta\mathbf{Y} + \sum_\mathbf{i} \mathbf{R_i}\mathbf{D}\mathbf{x_i}}{\beta\mathbf{I} + \sum_\mathbf{i} \mathbf{R_i^T}\mathbf{R_i}} \tag{23}$$

The solution to this problem thus involves averaging of overlapping patches after each patch has been sparse coded along with a weighted sum of the original noisy patch. Each pixel in a patch is hence a weighted linear combination of different pixels, the weights being derived from the sparse coding. Since the patches are overlapping, the final value of each pixel is thus an average of all representations obtained from sparse coding stage.

1) Comment on result:: The denoised image from the sparse based approach has similar SNR as some of standard denoising algorithms.

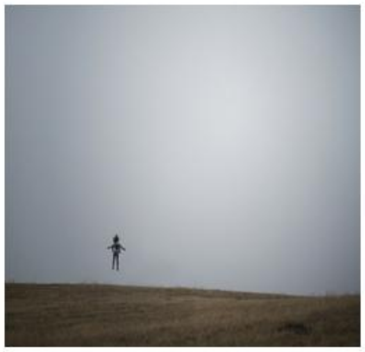

Fig. 5. Origianl Image [9]

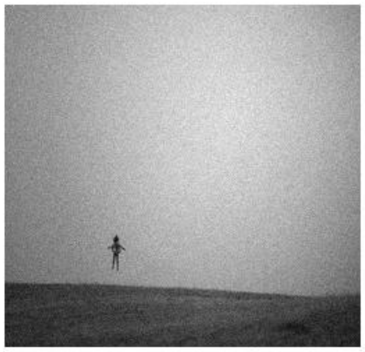

Fig. 6. Noisy image $\sigma = 10$

C. Image classification

If the natural signal has sparse representation with respect to a learned dictionary, then the dictionary atoms are distinctive in nature. We can use those coefficient vectors of the dictionary atoms as a feature for classification. We have applied image classification technique for binary classification problem. [12]–[14].

TABLE I. Accuracy

Differnt Method for MNIST dataset	Accuracy
Sparse based Image classification	81.3
Conventional method like KNN	96.65

1) Comments:: Although the accuracy of sparse based approach is not as high as some of the conventional classification methods, but sparse based approach may prove to be computationally more feasible if we are able to get a good dictionary. Classification accuracy is highly dependent on how the dictionary atoms are discriminative. Sparse based approach also seems to give close to 80% accuracy.

IV. Conclusion

In this term paper, we have briefly described various dictionary learning algorithms like k-SVD, MOD etc. Dictionary learning methods varies depending on the applications. we have briefly reviewed sparse and redundant representations as a new model that harnesses the local low-dimensional structure of natural images. Generally, standard DCT or random patches from the images act as good dictionary atoms. Once the dictionary is learnt, we can have the sparse representation of the signal in terms of the dictionary atoms. Sparse representation have various image processing based applications like image denoising, reconstruction, image inpainting, image classifica-

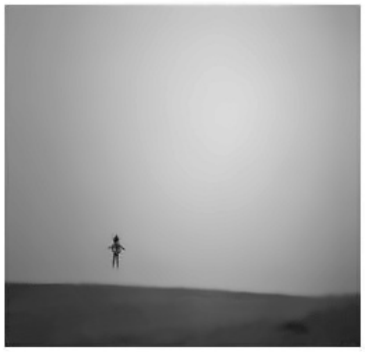

Fig. 7. Denoised Image

Algorithm 2 : Image classification via sparse representation

```
dataset :Images of '0' digit and '1' digit from MNIST database. Label y for 0 digit is -1 and for 1 digit it is +1. Image size : 64 x 64
Dictionary Construction : Either learn a dictionary by k-SVD, MOD methods or take some standard overcomplete dictionary. Dictionary can also be formed by sampling random patches from the source region of the given image. Randomly Initialise dictionary D of size (784 x 1000) and weight vector w of size (1000 x 1).
Solve Optimisation problem 1 : Solve the following optimisation problem to calculate the sparse representation $\hat{\alpha}$ for each training vector $\mathbf{x}$ w.r.t. the dictionary $\mathbf{D}$:
    $$\hat{\alpha}(\mathbf{x},\mathbf{D}) = \underset{\alpha\in\mathbf{R^p}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{x}-\mathbf{D}\alpha||_2^2 + \beta_1||\alpha||_1 + \frac{\beta_2}{2}||\alpha||_2^2 \quad (24)$$
feature vector : $\hat{\alpha}$ can be treated as a feature vector of $x$ with respect to the Dictionary $\mathbf{D}$
Solve Optimisation problem 2 : Now update $\mathbf{w}$ and $\mathbf{D}$ after solving the following optimisation problem.
    $$\min_{\mathbf{w}\in\mathbf{R^p},\mathbf{D}} \mathrm{E}[\log(1+exp(-\mathbf{y}\mathbf{w}^T\hat{\alpha}(\mathbf{x},\mathbf{D}))] + \frac{\beta}{2}||\mathbf{w}||_2^2 \quad (25)$$
Solve Optimisation problem 2 : Now update $\mathbf{w}$ and $\mathbf{D}$ after solving the following optimisation problem.
Iteration : Go to optimisation problem 1 and do this for some iterations.
Testing :Now the training is complete and we have $\mathbf{D}$ and $\mathbf{w}$. For a given test vector x, find its sparse representation $\hat{\alpha}$ and calculate $t = w^T\alpha$. If $t < 0$, predicted label is $-1$. If $t > 0$, predicted label is $+1$.
output : Labels for test images
```

tion. Three applications of the sparse based representation of images are presented in the paper with the simulation and results.

V. Fututre work

Within this framework of sparse based representation, more efficient and effective solutions to many conventional image processing tasks, including but not limited to image compression, denoising, deblurring, inpainting, super resolution, segmentation, can be explored. Despite its success so far, many difficult and open problems remain regarding why these algorithms work so well and under what conditions. We hope to investigate these problems in future.

VI. Codes Submitted

1) image inpainting using cvx
2) image inpainting using omp.m
3) Image classification for MNIST dataset.
4) knn classifier for MNIST dataset so as to compare the result with sparse based approach. (python code with MNIST read file available from internet)
5) Image denoising with k-svd dictionary
6) image denoising with DCT dictionary

References

[1] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[2] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM review*, vol. 51, no. 1, pp. 34–81, 2009.

[3] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Discriminative learned dictionaries for local image analysis," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[4] Q. Zhang and B. Li, "Discriminative k-svd for dictionary learning in face recognition," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2691–2698.

[5] Y. C. Pati, R. Rezaiifar, and P. Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*. IEEE, 1993, pp. 40–44.

[6] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *Information Theory, IEEE Transactions on*, vol. 53, no. 12, pp. 4655–4666, 2007.

[7] B. Shen, W. Hu, Y. Zhang, and Y.-J. Zhang, "Image inpainting via sparse representation," in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*. IEEE, 2009, pp. 697–700.

[8] S. K. Sahoo and W. Lu, "Image denoising using sparse approximation with adaptive window selection," in *Information, Communications and Signal Processing (ICICS) 2011 8th International Conference on*. IEEE, 2011, pp. 1–5.

[9] M. LLC. (1999) MS Windows NT kernel description. [Online]. Available: http://wallpaperswa.com/Nature/Fields/landscapes$_{n}ature_{f}ields_{g}low_{e}vening_{s}ky_{1}$12768

[10] K. Su, H. Fu, B. Du, H. Cheng, H. Wang, and D. Zhang, "Image denoising based on learning over-complete dictionary," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*, May 2012, pp. 395–398.

[11] M. Elad, M. A. Figueiredo, and Y. Ma, "On the role of sparse and redundant representations in image processing."

[12] M. Yang, D. Dai, L. Shen, and L. V. Gool, "Latent dictionary learning for sparse representation based classification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 4138–4145.

[13] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 4, pp. 791–804, 2012.

[14] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 2, pp. 210–227, 2009.

1) image inpainting using cvx

```
clear all; clc; close all;
source_img=double(imread('pic12.png'));   figure;imshow(uint8(source_img));
target_img=double(imread('mask12.png'));figure;imshow(uint8(target_img));
target_img= ~(rgb2gray(target_img));
[row_ind,col_ind] = find(~target_img);
[left_most_row_index,~] = min(row_ind);
[right_most_row_index,~] = max(row_ind);
[bottom_most_col_index,~] = max(col_ind);
[top_most_col_index,~] = min(col_ind);

patch_cell_r = mat2cell(source_img(:,:,1), 8*ones(1,size(source_img,1)/8),
8*ones(1,size(source_img,2)/8));
patch_cell_g = mat2cell(source_img(:,:,2), 8*ones(1,size(source_img,1)/8),
8*ones(1,size(source_img,2)/8));
patch_cell_b = mat2cell(source_img(:,:,3), 8*ones(1,size(source_img,1)/8),
8*ones(1,size(source_img,2)/8));



%%%% target image %%%%%%
patch_cell_target_r = mat2cell(target_img(:,:,1), 8*ones(1,size(target_img,1)/8),
8*ones(1,size(target_img,2)/8));
Num_NonZero_pixels = zeros(32,32);
temp = [];
temp1 = [];
temp2 = [];
for i =1:32
   for j = 1:32
       x = reshape(patch_cell_target_r{i,j},64,1);
       Num_NonZero_pixels(i,j) = nnz(x);
       if Num_NonZero_pixels(i,j) == 64
           temp = [temp;Num_NonZero_pixels(i,j)];
           temp1 = [temp1;i];
           temp2 = [temp2;j];
       end
   end
end


%%%%%%%%%%% Dictionary by sampling images %%%%%%%%%%%%%%%%%%%%%
Dict_size = 128;
rng('shuffle');
r = randi([1 size(temp1,1)],1,Dict_size);

Dict_R_value = zeros(64, Dict_size);
Dict_G_value = zeros(64, Dict_size);
Dict_B_value = zeros(64, Dict_size);
```

```
for i=1:Dict_size
p = temp1(i);
q = temp2(i);
x = reshape(patch_cell_r{p,q},64,1);
Dict_R_value(:,i) = x;
x = reshape(patch_cell_g{p,q},64,1);
Dict_G_value(:,i) = x;
x = reshape(patch_cell_b{p,q},64,1);
Dict_B_value(:,i) = x;
end

% figure;
% for i = 1:Dict_size/2
%     subplot(8,8,i);
%     imshow(uint8(patch_cell_r{temp1(i),temp2(i)}));
% end
% figure;
% for i = 1:Dict_size/2
%     subplot(8,8,i);
%     imshow(uint8(patch_cell_r{temp1(i+Dict_size/2),temp2(i+Dict_size/2)}));
% end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% sliding
window and reconstructin using OMP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k = 1;         %%% sparsity = k %%%
p = 3;
q = 4;
lambda1=10;
% figure;
% imshow(uint8(source_img(left_most_row_index-
p:left_most_row_index+q,top_most_col_index+3-p:top_most_col_index+3+q)));
for i = left_most_row_index:right_most_row_index
    for j = top_most_col_index:bottom_most_col_index
        data_vect_R = reshape(source_img(i-p:i+q,j-p:j+q,1),64,1);
        data_vect_G = reshape(source_img(i-p:i+q,j-p:j+q,2),64,1);
        data_vect_B = reshape(source_img(i-p:i+q,j-p:j+q,3),64,1);
        d=128;
         cvx_begin
            variable y(d)
            minimize( 0.5*norm(Dict_R_value*y-data_vect_R)+lambda1*norm(y,1))
        cvx_end
        alpha_R = y;
         cvx_begin
            variable y(d)
            minimize( 0.5*norm(Dict_G_value*y-data_vect_G)+lambda1*norm(y,1))
        cvx_end
        alpha_G = y;
         cvx_begin
            variable y(d)
            minimize( 0.5*norm(Dict_B_value*y-data_vect_B)+lambda1*norm(y,1))
        cvx_end
        alpha_B = y;
```

```
%        alpha_G = omp(data_vect_G,Dict_G_value,k);
%        alpha_B = omp(data_vect_B,Dict_B_value,k);
%



        recontructed_patch_R = reshape(Dict_R_value * alpha_R, 8, 8);
        recontructed_patch_G = reshape(Dict_G_value * alpha_G, 8, 8);
        recontructed_patch_B = reshape(Dict_B_value * alpha_B, 8, 8);
%        alpha_R = OMP_1(k,data_vect_R',Dict_R_value);
%        alpha_G = OMP_1(k,data_vect_G',Dict_G_value);
%        alpha_B = OMP_1(k,data_vect_B',Dict_B_value);
%        recontructed_patch_R = reshape(Dict_R_value * alpha_R', 8, 8);
%        recontructed_patch_G = reshape(Dict_G_value * alpha_G', 8, 8);
%        recontructed_patch_B = reshape(Dict_B_value * alpha_B', 8, 8);
        window_patch = target_img(i-p:i+q,j-p:j+q);

        source_img(i-p:i+q,j-p:j+q,1) = source_img(i-p:i+q,j-p:j+q,1).*window_patch +
recontructed_patch_R.*(~window_patch);
        source_img(i-p:i+q,j-p:j+q,2) = source_img(i-p:i+q,j-p:j+q,2).*window_patch +
recontructed_patch_G.*(~window_patch);
        source_img(i-p:i+q,j-p:j+q,3) = source_img(i-p:i+q,j-p:j+q,3).*window_patch +
recontructed_patch_B.*(~window_patch);

%        target_img(i-p:i+q,j-p:j+q) = ones(8,8);

%        source_img(i-p:i+q,j-p:j+q,1) = recontructed_patch_R.*(~window_patch);
%        source_img(i-p:i+q,j-p:j+q,2) = recontructed_patch_G.*(~window_patch) ;
%        source_img(i-p:i+q,j-p:j+q,3) = recontructed_patch_B.*(~window_patch) ;

%        if i ==left_most_row_index && j == top_most_col_index+3
%            figure;
%            imshow(uint8(source_img(i-p:i+q,j-p:j+q)));
%        end

%        source_img(i-7:i,j-7:j,1) = source_img(i-7:i,j-7:j,1).*window_patch ;
%        source_img(i-7:i,j-7:j,2) = source_img(i-7:i,j-7:j,2).*window_patch ;
%        source_img(i-7:i,j-7:j,3) = source_img(i-7:i,j-7:j,3).*window_patch ;

%        source_img(i-7:i,j-7:j,1) = recontructed_patch_R.*(~window_patch) ;
%        source_img(i-7:i,j-7:j,2) = recontructed_patch_G.*(~window_patch) ;
%        source_img(i-7:i,j-7:j,3) = recontructed_patch_B.*(~window_patch) ;

    end
end

figure;imshow(uint8(source_img));
```

2) Omp.m

```
function x = omp(data_vector,Dict_matrix,sparsity)
N = size(Dict_matrix,2);                        % compute k-sparse approximation to b with matrix A
using Matching pursuit
x = zeros(N,1);
res = data_vector;
support = [];                           % empty support
for i = 1:sparsity
   corr = Dict_matrix'*res;                         % compute correlation between residual and columns
of A
   [~,n] = max(abs(corr));                    % find position n (and value c) of the maximally correlated
column
   support(end+1) = n;                       % extend the support
   x(support) = pinv(Dict_matrix(:,support))*data_vector;          % update the representation
   %res  = res - Dict_matrix(:,support)*x(support);       % update the residual
   res  = data_vector - Dict_matrix(:,support)*x(support);
end
end
```

3)Image classification for MNIST dataset

```
load('mnist01.mat');
N = size(trainX, 2); % dimension of data i.e. 784
n = size(trainX, 1); % number of traininng examples i.e. 87
d = 50; % number of atoms in dictionary, D, is now N x d
D = randn(N,d); % random initialisation of dictionary D
w = randn(d,1); % weight vector
alpha = zeros(d,n);
numIter = 25; % number of iterations set beforehand
lambda1 = 10;
lambda2=10;
sum = 0;
param.L = 3;   % number of elements in each linear combination.
param.K = 50; % number of dictionary elements
param.numIteration = 50; % number of iteration to execute the K-SVD algorithm.

param.errorFlag = 0; % decompose signals until a certain error is reached. do not use fix number of
coefficients.
%param.errorGoal = sigma;
param.preserveDCAtom = 0;


param.InitializationMethod =  'DataElements';
param.displayProgress = 1;

   [D,o] = KSVD(trainX',param);
for iter = 1:1
   for i = 1:n
      cvx_begin
         variable y(d)
         minimize( 0.5*norm(D*y-trainX(i,:)')+lambda1*norm(y,1)+lambda2*0.5*norm(y,2) )
      cvx_end
      alpha(:,i) = y; % update all alphas using CVX
   end

      cvx_begin
         variable w(d)
         for i=1:n
            sum = sum + log(1+exp(-testY(i)*w'*alpha(:,i)));
         end
            minimize( sum + lambda2*norm(w,2) )
      cvx_end


end
ntest = size(testX, 1);
accuracy = 0;
temp = zeros(d,1);
temp2 = 0;
n_i=50;
for i = 1:n_i
   cvx_begin
         variable y(d)
         minimize( 0.5*norm(D*y-testX(i,:)')+lambda1*norm(y,1)+lambda2*0.5*norm(y,2) )
      cvx_end
   temp = y;
   temp2 = dot(w,temp);
   if((temp2>0 && testY(i) == 1)||(temp2<0 && testY(i) == -1))
      accuracy = accuracy + 1;
   end
end
accuracy = accuracy/n_i*100;
disp(accuracy);
```

4) Image classification (knn_python code) to compare the result with the sparse based approach:

```
import os, struct
import numpy as np
from array import array as pyarray
from numpy import append, array, int8, uint8, zeros
from sklearn.neighbors import KNeighborsClassifier
path_name = "/home/abhay/CS771A/"

def load_mnist(dataset="training", digits=np.arange(10), path="."):
    """
    Loads MNIST files into 3D numpy arrays

    Adapted from: http://abel.ee.ucla.edu/cvxopt/_downloads/mnist.py
    """

    if dataset == "training":
        fname_img = os.path.join(path, 'train-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels.idx1-ubyte')
    elif dataset == "testing":
        fname_img = os.path.join(path, 't10k-images.idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels.idx1-ubyte')
    else:
        raise ValueError("dataset must be 'testing' or 'training'")

    flbl = open(fname_lbl, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_img, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [ k for k in range(size) if lbl[k] in digits ]
    N = len(ind)

    images = zeros((N, rows*cols), dtype=uint8)
    labels = zeros((N, 1), dtype=int8)
    for i in range(len(ind)):
        images[i] = array(img[ ind[i]*rows*cols : (ind[i]+1)*rows*cols ])
        labels[i] = lbl[ind[i]]

    return images, labels

# load_mnist(dataset="training", digits=np.arange(10), path="/home/abhay/CS771A/")
from pylab import *
Training_images, Training_labels = load_mnist(dataset="training", digits=np.arange(10),
path=path_name)
# imshow(images.mean(axis=0), cmap=cm.gray)
# show()
# print(len(images))
# print(images.shape)
metrics = ['minkowski','euclidean','chebyshev','manhattan']
Test_images, Test_labels = load_mnist(dataset="testing", digits=np.arange(10), path=path_name)
for metric in metrics:
	print(metric)
	for i in range(4):
		error= 0.0
		accuracy = 0.0
		knn = KNeighborsClassifier(metric= metric, n_jobs=4, n_neighbors=i+1)
```

```
            knn.fit(Training_images, Training_labels)
            Predicted_label = knn.predict(Test_images)
            for k in range(len(Test_labels)):
                if Predicted_label[k] != Test_labels[k]:
                    error += 1.0
            accuracy = ((len(Test_labels)-error)*100.0)/(len(Test_labels))
            print("n_neighbors is " + str(i))
            print(len(Test_labels))
            print(error)
            print(accuracy)
```

5)k-SVD (got from internet)

```
function [Dictionary,output] = KSVD(...
    Data,... % an nXN matrix that contins N signals (Y), each of dimension n.
    param)
if (~isfield(param,'displayProgress'))
    param.displayProgress = 0;
end
totalerr(1) = 99999;
if (isfield(param,'errorFlag')==0)
    param.errorFlag = 0;
end

if (isfield(param,'TrueDictionary'))
    displayErrorWithTrueDictionary = 1;
    ErrorBetweenDictionaries = zeros(param.numIteration+1,1);
    ratio = zeros(param.numIteration+1,1);
else
    displayErrorWithTrueDictionary = 0;
        ratio = 0;
end
if (param.preserveDCAtom>0)
    FixedDictionaryElement(1:size(Data,1),1) = 1/sqrt(size(Data,1));
else
    FixedDictionaryElement = [];
end
% coefficient calculation method is OMP with fixed number of coefficients

if (size(Data,2) < param.K)
    disp('Size of data is smaller than the dictionary size. Trivial solution...');
    Dictionary = Data(:,1:size(Data,2));
    return;
elseif (strcmp(param.InitializationMethod,'DataElements'))
    Dictionary(:,1:param.K-param.preserveDCAtom) = Data(:,1:param.K-param.preserveDCAtom);
elseif (strcmp(param.InitializationMethod,'GivenMatrix'))
    Dictionary(:,1:param.K-param.preserveDCAtom) = param.initialDictionary(:,1:param.K-
param.preserveDCAtom);
end
% reduce the components in Dictionary that are spanned by the fixed
% elements
if (param.preserveDCAtom)
```

```
    tmpMat = FixedDictionaryElement \ Dictionary;
    Dictionary = Dictionary - FixedDictionaryElement*tmpMat;
end
%normalize the dictionary.
Dictionary = Dictionary*diag(1./sqrt(sum(Dictionary.*Dictionary)));
Dictionary = Dictionary.*repmat(sign(Dictionary(1,:)),size(Dictionary,1),1); % multiply in the sign
of the first element.
totalErr = zeros(1,param.numIteration);

% the K-SVD algorithm starts here.

for iterNum = 1:param.numIteration
    % find the coefficients
    if (param.errorFlag==0)
        %CoefMatrix = mexOMPIterative2(Data, [FixedDictionaryElement,Dictionary],param.L);
        CoefMatrix = OMP([FixedDictionaryElement,Dictionary],Data, param.L);
    else
        %CoefMatrix = mexOMPerrIterative(Data,
[FixedDictionaryElement,Dictionary],param.errorGoal);
        CoefMatrix = OMPerr([FixedDictionaryElement,Dictionary],Data, param.errorGoal);
        param.L = 1;
    end

    replacedVectorCounter = 0;
            rPerm = randperm(size(Dictionary,2));
    for j = rPerm
        [betterDictionaryElement,CoefMatrix,addedNewVector] =
I_findBetterDictionaryElement(Data,...
            [FixedDictionaryElement,Dictionary],j+size(FixedDictionaryElement,2),...
            CoefMatrix ,param.L);
        Dictionary(:,j) = betterDictionaryElement;
        if (param.preserveDCAtom)
            tmpCoef = FixedDictionaryElement\betterDictionaryElement;
            Dictionary(:,j) = betterDictionaryElement - FixedDictionaryElement*tmpCoef;
            Dictionary(:,j) = Dictionary(:,j)./sqrt(Dictionary(:,j)'*Dictionary(:,j));
        end
        replacedVectorCounter = replacedVectorCounter+addedNewVector;
    end

    if (iterNum>1 & param.displayProgress)
        if (param.errorFlag==0)
            output.totalerr(iterNum-1) = sqrt(sum(sum((Data-
[FixedDictionaryElement,Dictionary]*CoefMatrix).^2))/prod(size(Data)));
            disp(['Iteration   ',num2str(iterNum),'   Total error is: ',num2str(output.totalerr(iterNum-1))]);
        else
            output.numCoef(iterNum-1) = length(find(CoefMatrix))/size(Data,2);
            disp(['Iteration   ',num2str(iterNum),'   Average number of coefficients:
',num2str(output.numCoef(iterNum-1))]);
        end
    end
    if (displayErrorWithTrueDictionary )
        [ratio(iterNum+1),ErrorBetweenDictionaries(iterNum+1)] =
I_findDistanseBetweenDictionaries(param.TrueDictionary,Dictionary);
        disp(strcat(['Iteration  ', num2str(iterNum),' ratio of restored elements: 
',num2str(ratio(iterNum+1))]));
        output.ratio = ratio;
    end
    Dictionary = 
I_clearDictionary(Dictionary,CoefMatrix(size(FixedDictionaryElement,2)+1:end,:),Data);

    if (isfield(param,'waitBarHandle'))
        waitbar(iterNum/param.counterForWaitBar);
    end
end

output.CoefMatrix = CoefMatrix;
Dictionary = [FixedDictionaryElement,Dictionary];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  findBetterDictionaryElement
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [betterDictionaryElement,CoefMatrix,NewVectorAdded] = 
I_findBetterDictionaryElement(Data,Dictionary,j,CoefMatrix,numCoefUsed)
if (length(who('numCoefUsed'))==0)
    numCoefUsed = 1;
end
relevantDataIndices = find(CoefMatrix(j,:)); % the data indices that uses the j'th dictionary element.
if (length(relevantDataIndices)<1) %(length(relevantDataIndices)==0)
    ErrorMat = Data-Dictionary*CoefMatrix;
    ErrorNormVec = sum(ErrorMat.^2);
    [d,i] = max(ErrorNormVec);
    betterDictionaryElement = Data(:,i);%ErrorMat(:,i); %
    betterDictionaryElement = 
betterDictionaryElement./sqrt(betterDictionaryElement'*betterDictionaryElement);
    betterDictionaryElement = betterDictionaryElement.*sign(betterDictionaryElement(1));
    CoefMatrix(j,:) = 0;
    NewVectorAdded = 1;
    return;
end

NewVectorAdded = 0;
tmpCoefMatrix = CoefMatrix(:,relevantDataIndices); 
tmpCoefMatrix(j,:) = 0;% the coeffitients of the element we now improve are not relevant.
errors =(Data(:,relevantDataIndices) - Dictionary*tmpCoefMatrix); % vector of errors that we want 
to minimize with the new element
% % the better dictionary element and the values of beta are found using svd.
% % This is because we would like to minimize || errors - beta*element ||_F^2. 
% % that is, to approximate the matrix 'errors' with a one-rank matrix. This
% % is done using the largest singular value.
[betterDictionaryElement,singularValue,betaVector] = svds(errors,1);
CoefMatrix(j,relevantDataIndices) = singularValue*betaVector';% *signOfFirstElem

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  findDistanseBetweenDictionaries
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ratio,totalDistances] = I_findDistanseBetweenDictionaries(original,new)
% first, all the column in oiginal starts with positive values.
catchCounter = 0;
totalDistances = 0;
for i = 1:size(new,2)
   new(:,i) = sign(new(1,i))*new(:,i);
end
for i = 1:size(original,2)
   d = sign(original(1,i))*original(:,i);
   distances =sum ( (new-repmat(d,1,size(new,2))).^2);
   [minValue,index] = min(distances);
   errorOfElement = 1-abs(new(:,index)'*d);
   totalDistances = totalDistances+errorOfElement;
   catchCounter = catchCounter+(errorOfElement<0.01);
end
ratio = 100*catchCounter/size(original,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  I_clearDictionary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Dictionary = I_clearDictionary(Dictionary,CoefMatrix,Data)
T2 = 0.99;
T1 = 3;
K=size(Dictionary,2);
Er=sum((Data-Dictionary*CoefMatrix).^2,1); % remove identical atoms
G=Dictionary'*Dictionary; G = G-diag(diag(G));
for jj=1:1:K,
   if max(G(jj,:))>T2 | length(find(abs(CoefMatrix(jj,:))>1e-7))<=T1 ,
      [val,pos]=max(Er);
      Er(pos(1))=0;
      Dictionary(:,jj)=Data(:,pos(1))/norm(Data(:,pos(1)));
      G=Dictionary'*Dictionary; G = G-diag(diag(G));
   end;
end;
```

6) Image denoising

```
clear all; clc; close all;
source_img=double(rgb2gray(imread('pic12.png')));   figure;imshow(uint8(source_img));
noisy_img = double(imnoise(uint8(source_img), 'gaussian',0,0.001));
figure; imshow(uint8(noisy_img));
d = size(noisy_img,1);
R = zeros(d,d,d*d/256);
sz = d*d/256;
% d=16;
% sz=4;
% R = zeros(d,d,d*d/64);
% for i=1:sz
%     for j = 1:(d/8)
%         for k = 1:(d/8)
%             R((j-1)*8+1:j*8 ,(k-1)*8+1:k*8, i  ) = ones(8,8);
%         end
%     end
% end
% d=32;
% sz=16;
% R = zeros(d,d,d*d/64);

    for j = 1:(d/16)
        for k = 1:(d/16)
            R((j-1)*16+1:j*16 ,(k-1)*16+1:k*16, (j-1)*sqrt(sz)+k  ) = ones(16);
        end
    end
Pn=ceil(sqrt(2*d));
bb=ceil(sqrt(d));
DCT=zeros(bb,Pn);
for k=0:1:Pn-1,
    V=cos([0:1:bb-1]'*k*pi/Pn);
    if k>0, V=V-mean(V); end;
    DCT(:,k+1)=V/norm(V);
end;
DCT=kron(DCT,DCT);
mu = ones(sz,1);
lambda=1
sumc=0;
sumb=0;
Rf = zeros(16,16,sz);
RE = zeros(d,d);
    cvx_begin
            variables y(d,d) z(size(DCT,2),sz)

            for j = 1:(d/16)
                for k = 1:(d/16)
                    RE=R( :,:, (j-1)*sqrt(sz)+k  )*y;
                    %Rf(:,:,(j-1)*sqrt(sz)+k) = RE((j-1)*16+1:j*16 ,(k-1)*16+1:k*16);
                    sumb = sumb +norm(DCT*z(:,(j-1)*sqrt(sz)+k)-vec(RE((j-1)*16+1:j*16 ,(k-
1)*16+1:k*16)));
                    k
                end
               j

            end
%            for i=1:sz
%
%                sumb = sumb + norm(DCT*z(:,i)-vec(Rf(:,:,i)));
%                i
%            end
            for i=1:sz
                sumc = sumc + mu(i)*norm(z(:,i),1);
                i
            end
            minimize( lambda*norm(noisy_img-y,'fro')+ sumc+sumb )
        cvx_end
```