EE 583
PROJECT REPORT

Sparse code shrinkage applied to image data

Ceyhun B. Akgül
ceyburak@ixir.com

Abstract

In this work, we investigate the use of sparse coding in conjunction with maximum likelihood principle for denoising image data. We discuss the links between sparse coding and independent component analysis and explore maximum likelihood denoising of non-Gaussian random variables. We have also performed practical implementation of a relatively recent method, sparse code shrinkage(SCS) and tested it on an image data set. We have seen that SCS outperforms basic denoising methods such as Gaussian smoothing and median filtering.

1 Introduction

Image denoising is an important task for especially improving perceptual quality of noisy digital image data. Corrupting noise may be due to amplifier distortions, encountered more or less in all of the processes that sense image data. In particular, the model for corrupting noise is additive Gaussian which lends itself to well-founded techniques such as Wiener filtering and wavelet shrinkage methods. A recently developed technique is sparse code shrinkage[1] which exploits the statistical properties of data to be denoised in a more realistic way than Wiener filter that is known to be optimal for Gaussian data and than wavelet shrinkage method[2] which does not deal with data statistics. The reason for Wiener filtering falls short in most of the image denoising aplications is that image data are rather sparsely distributed, i.e. supergaussian. On the other hand, wavelet shrinkage method adopts a fixed basis to linearly transform image data into another domain where denoising is more tractable. A wise idea which constitutes the rationale behind sparse code shrinkage(SCS) is to use a basis that is more suitable for data at hand. Once such a transformation is found, denoising can be achieved by maximum likelihood principles. The latter requires the parametrization of random variables in the sparse domain, in [1] several sparsity models are proposed. Furthermore, generic independent component analysis(ICA) methods can be used for the estimation of the sparsifying transformation.

In this work, we explore SCS technique by presenting the corresponding theory and practical issues concerning its implementation. The report is organized as follows. In the following section, a brief overview of ICA, which is very closely related to sparse coding, is provided from the perspective of its application to image data. In section 3, we discuss the motivation for using SCS, namely maximum likelihood denoising of non-Gaussian data as well as we summarize the SCS algorithm. In section 4, the results of several experiments are presented. The last section is devoted to the discussion of the algorithm and further directions of research.

2 Independent Component Analysis(ICA) and Image Data

2.1 Overview of ICA

Independent Component Analysis is motivated by the fact that an observed set of linear mixtures $x_1,....,x_m$ comes from m or less sources s_i. The term "independent" is associated with statistical properties of the components s_i. The model can be represented as follows:

$$\mathbf{x} = \mathbf{A}.\mathbf{s} \qquad (1)$$

where **x** and **s** are column vectors with entries x_i and s_i respectively, **A** is the mixing matrix. Accordingly j^{th} entry for the vector **x** can be expressed as:

$$x_j = a_{j1}s_1 + a_{j2}s_2 + ... + a_{jm}s_m \quad \text{for all } j \quad (2)$$

In the model, s_i are latent variables and the mixing process(or matrix) is assumed to be unknown. From this perspective, ICA is closely related to the method so called *Blind Source Separation*(BSS)[3]. "Blind" refers to

that we do not have any information on neither the mixing matrix nor the sources. In the most basic setup, it is well established that in case the sources are statistically independent, at mot one of them being Gaussian distributed, and the mixing is linear, BSS problem can be solved using ICA.

Once statistical independence of s_i is asserted, we shall refer to the so-called *Central Limit Theorem*(CLT) to explain the rationale behind ICA. CLT states that the distribution of a sum of independent random variables tends toward a gaussian distribution under certain conditions. Thus, a sum of two or more independent random variables has a distribution closer to gaussian than any of the original random variables. From this viewpoint, ICA reduces to maximizing the non-gaussianity of the observed mixtures **x** assuming **x** has been exposed to some preprocessing, namely centering and whitening. This heuristic approach to ICA had been justified in several articles and many algorithms had been proposed in the form of numerical optimization. Among those, FastICA algorithm**[4]** developed at CIS, Helsinki University of Technology, is used throughout this project due to its fast convergence properties.

2.2 Sparse Coding and ICA Model for image data

Sparse coding is a method for finding a suitable representation of data in which each of the components is only rarely active, i.e. the components are zero most of the time as suggested in [1]. Olshausen and Field had shown that this representation has some neurophysiological plausibility in [5]. Additionally sparse coding is associated with ICA in that the focus of maximizing non-gaussianity in ICA may be formulated as extracting the maximally sparse code out of image data in sparse coding.

In this project, sparse coding, or ICA equivalently, is used to extract features from image data. This extraction procedure provides a more data-dependent representation of given images than any other transformation like principal component analysis(PCA), discrete cosine or wavelet transforms would do since ICA makes use of statistical properties of the data. The idea behind extracting such sparse codes, is that the latter lends itself to maximum likelihood denoising which is the main issue of section 3. To illustrate how ICA features are more suitable for representing image data, observe Fig. 1. where ICA features and PCA features are shown side by side.

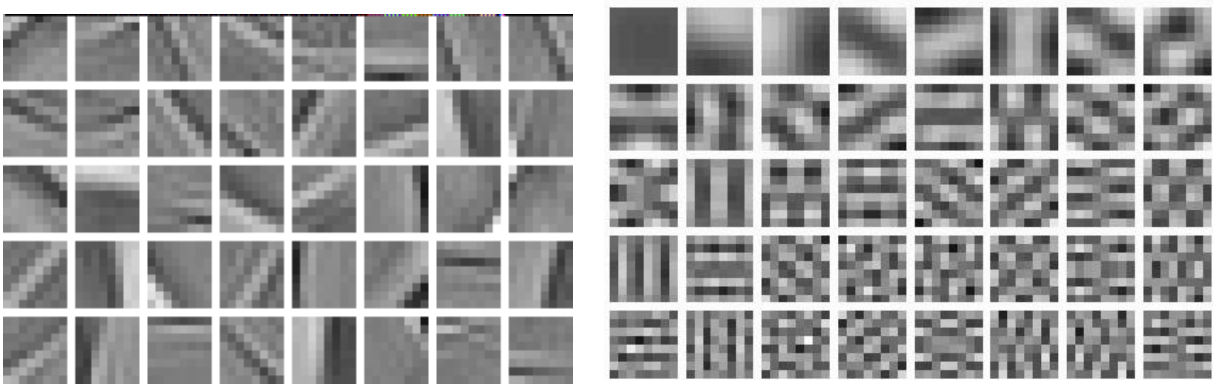

Fig. 1. ICA features(left) and PCA features(right).

As it is clear from Fig. 1 ICA features look much more natural than the PCA features do. The quality of ICA features are described as being oriented, localized and bandpass.

2.2.1 ICA Model for image data

In modeling image data for ICA, subimages of size NxN , taken from an arbitrary sized image, is considered as the observations of the nx1, letting $n=N^2$, random vector **x** of **(**1**)**. In this view, each **x** can be considered as a linear combination of the columns of the mixing matrix **A**, each of which is weighted by the independent components s_i. For instance, such a subimage would be obtained by superposing the patches of Fig. 1., each patch being weighted by the corresponding independent component of the sparse code. Hence the patches of Fig. 1. correspond to the columns of the mixing matrix and the independent components to sparse codes. In the ideal setup, the mixing matrix **A** is square**,** i.e. $n=m$ or there as many observed mixtures as the independent components**,** hence the relation (1) can be inverted to **s=W.x** where the matrix **W** is called the separating or demixing matrix. Hereafter, we will assume that the last assertion holds.

What ICA achieves is the estimation of the matrices **A** and **W** by just using the observed mixtures **x** that are indeed subimages written in the vector form. Once such a transformation pair is found, extraction of sparse codes is a simple matter of matrix-vector multiplication through **s**=**W**.**x** .

3 Maximum likelihood denoising and sparse code shrinkage

3.1 Maximum likelihood denoising of non-Gaussian random variables

In order to derive maximum-likelihood denoising of a non-Gaussian data corrupted with zero mean Gaussian noise of variance σ^2, the following additive model is considered

$$y = s + n \qquad (3)$$

where y the corrupted version of non-Gaussian random variable s and n is the random variable that stands for Gaussian noise. We stress that we only observe y and we only assume the noise model is additive and Gaussian. The maximum likelihood estimation of s can be formulated by first writing;

$$p(s \mid y) = \frac{p(y \mid s)p(s)}{p(y)} \qquad (4)$$

following Bayes rule. Maximizing $p(s|y)$ being equivalent to maximizing $\log p(s|y)$, we can write

$$\log p(s \mid y) = \log p(y \mid s) + \log p(s) - \log p(y) \qquad (5)$$

Now since $p(y|s)$ is the distribution of the new information added to s and the only new information is n=y-s, we can deduce that $p(y|s)$=$p(n)$=$p(y\text{-}s)$ then

$$\log p(s \mid y) = -\frac{1}{2\sigma^2}(y-s)^2 + \log p(s) - \log p(y) \qquad (6)$$

Dropping the evidence term $p(y)$ in (4) since it does not depend on s, maximizing (4)-(6) is equivalent to maximizing

$$J(s) = -\frac{1}{2\sigma^2}(y-s)^2 + \log p(s) \qquad (7)$$

Letting $f(s) = -\log p(s)$, formulating the problem as a minimization, we have the following estimator of s

$$\hat{s} = \arg\min_u \left\{ \frac{1}{2\sigma^2}(y-u)^2 + f(u) \right\} \qquad (8)$$

Whenever f is twice differentiable and its second derivative is strictly positive, we should solve

$$\frac{1}{\sigma^2}(\hat{s}-y) + f'(\hat{s}) = 0 \qquad (9)$$

or

$$\hat{s} = g(y) \qquad (10) \quad \text{with} \quad g^{-1}(u) = u + \sigma^2 f'(u) \qquad (11)$$

Accordingly, the ML estimator of s can be obtained by inverting a relation $g(u)$ that is a function of noise variance and the derivative of the -log $p(s)$ which is in turn a function of the distribution of s. In the literature [1], two approximations to (11) are proposed in case (11) cannot be inverted as follows:

$$\hat{s} = y - \sigma^2 f'(y) \qquad (12) \quad \text{and} \quad \hat{s} = sign(y)\max(0, |y| - \sigma^2 |f'(y)|) \qquad (13)$$

Accordingly, the exact and approximated ML estimators of s happen to look as (10) and (12) or (13) respectively. Since all of these equations depend on the distribution of s, one should offer some models for them based on the fact that such densities are sparsely distributed.

3.2 Sparse density models

Sparse data are characterized as being mildly or strongly sparse. Below general expressions for such models are given [1]. Note that Laplacian density is somewhat between them as shown in Fig. 2.

Mildly sparse density;

$$p(s) = C\exp(-as^2/2 - b|s|) \qquad (14)$$

- Strongly sparse density;

$$p(s) = \frac{1}{2d}\frac{(\alpha+2)\left[\alpha(\alpha+1)/2\right]^{(\alpha/2+1)}}{\left[\sqrt{\alpha(\alpha+1)/2} + |s/d|\right]^{(\alpha+3)}} \qquad (15)$$

- Laplacian density;

$$p(s) = \frac{1}{2d}\exp(-\frac{\sqrt{2}}{d}|s|) \qquad (16)$$

where a, b, α and d are the parameters that determine the nature of the distributions and should be estimated from observations.

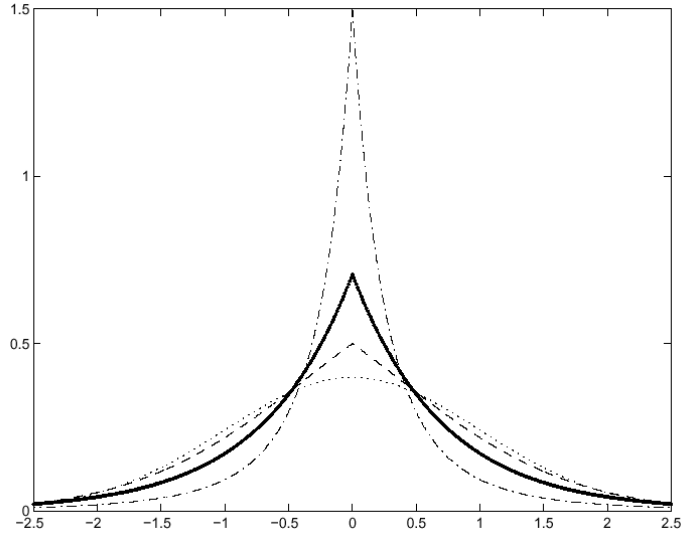

Fig. 2 Plots of densities corresponding to different models of sparse components. Solid line: Laplace density. Dashed line: a typical moderately sparse(super-Gaussian) density. Dash-dotted line: a typical strongly sparse(super-Gaussian) density. Dotted line: Gaussian density for reference.[due to Hyvaarinen et al. 1999]

The form of the so called shrinkage function $g(u)$ corresponding to the above distributions happens to be

- For mildly sparse densities;

$$g(u) = \frac{1}{1+\sigma^2 a}sign(u)\max(0, |u| - b\sigma^2) \quad (17)$$

$$b = \frac{2p_s(0)E\{s^2\} - E\{|s|\}}{E\{s^2\} - [E\{|s|\}]^2} \qquad (18)$$

$$a = \frac{1}{E\{s^2\}}[1 - E\{|s|\}b] \qquad (19)$$

- For strongly sparse densities;

$$g(u) = \frac{1}{2d} sign(u) \max(0, \frac{|u| - ad}{2} + \frac{1}{2}\sqrt{(|u| + ad)^2 - 4\sigma^2(\alpha + 3)}) \qquad (20)$$

$$d = \sqrt{E\{s^2\}} \qquad (21)$$

$$\alpha = \frac{2 - k + \sqrt{k(k+4)}}{2k - 1} \qquad (22) \quad \text{with } k = d^2 p_s(0)^2$$

$$a = \sqrt{\alpha(a+1)/2} \qquad (23)$$

- For Laplacian densities;

$$g(u) = sign(u) \max(0, |u| - 2\sigma^2 / d) \qquad (24)$$

where σ^2 is the noise variance in all cases.

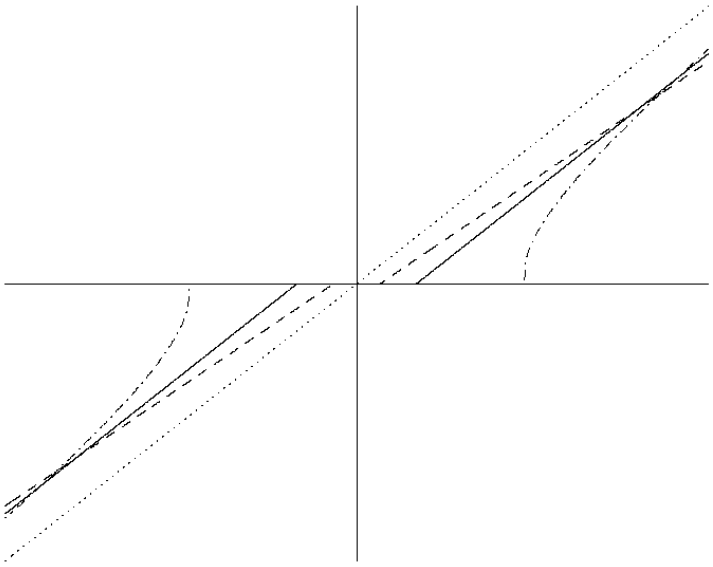

Fig. 3 Plots of the shrinkage functions. Solid line: shrinkage corresponding to Laplace density. Dashed line: shrinkage corresponding to moderately sparse(super-gaussian) density. Dash-dotted line: shrinkage corresponding to strongly sparse(super-gaussian) density. Dotted line: $y=x$ for reference. [due to Hyvaarinen et al. 1999]

The effects of the functions shown in Fig. 3 is to reduce the absolute value of its argument by an amount determined by the noise level. Small arguments are set to zero. The overall effect is a reduction in Gaussian noise by restoring sparseness back.

To determine which model to use, it is suggested in the limiting case of Laplacian density that the value $d.p_s(0)$ happens to be $1/\sqrt{2}$, below it the model is mildly sparse and above styrongly sparse. However, experience shows that sparse code extracted from real image data happens to follow a strong sparsity.

3.3 Sparse code shrinkage technique

In this subsection, we describe the SCS technique with its practical aspects. Before doing so, let us summarize thre rationale behind it. In previous sections, we have seen how ML principle can be used to remove additive Gaussian noise from non-Gaussian random variables, specificly super-Gaussian random variables. It is simply based on applying a nonlinearity on the observation $y=s+n$ **,** the type of the nonlinearity being dictated by the distribution of sparse codes s. However, in case of real image data we observe pixel values x_i or written in vector form subimages $\mathbf{x}$ and the additive noise model is $\mathbf{x} = \mathbf{A}\mathbf{s}+\mathbf{v}$ where $\mathbf{v}$ is uncorrelated Gaussian noise. Accordingly if we had acces to the the true transform pair $\mathbf{A}$ and $\mathbf{W}$, we could write $\mathbf{W}\mathbf{x} = \mathbf{s} + \mathbf{W}\mathbf{v}$ or $\mathbf{y} = \mathbf{s} + \mathbf{W}\mathbf{v}$. Furthermore to be able to use the ML principle, the noise term $\mathbf{W}\mathbf{v}$ should be uncorrelated. This is true whenever $\mathbf{W}$ is a similarity transform, that is an orthogonal matrix such that $\mathbf{W}^T\mathbf{W}=\mathbf{I}$ hence $\mathbf{W}^T=\mathbf{A}$. Two problems arise:

(i) How to estimate the transform pair $\mathbf{A}$ and $\mathbf{W}$ by just observing noisy image data $\mathbf{x}$?
(ii) How to achieve the orthogonality of $\mathbf{W}$ and hence $\mathbf{A}$?

The second task is a minor one and can be accomplished in an approximative manner by letting

$$\mathbf{W} \leftarrow \mathbf{W}(\mathbf{W}^T\mathbf{W})^{-1/2} \qquad (25)$$

The first task is a tough problem and computationally very demanding. One shortcut to this problem is to obtain a noise-free version of $\mathbf{x}$ and to estimate the mixing and separating matrices out of it by using generic ICA methods, such as FastICA. This idea is not so unrealistic since as far as image data are of concern, one can always find a suitable noise-free dataset that have the same characteristics as the noisy mixtures $\mathbf{x}$**.**

Now we can summarize the SCS technique:

(i) Obtain a representative noise-free dataset $\mathbf{z}$ with the same statistical properties as the noisy n-dimensional $\mathbf{x}$. Using one of the generic ICA methods, estimate the separating matrix $\mathbf{W}$ and orthogonalize it following (25).
(ii) For every $i=1,\ldots,n$, determine the density model and the parameters of the corresponding shrinkage function following (17)-(24).
(iii) For every noisy data sample $\mathbf{x}(t)$, $t=1,..,T$ compute the projections on th sparsifying basis by $\mathbf{y}(t)=\mathbf{W}\mathbf{x}(t)$
(iv) Apply the corresponding shrinkage nonlinearity for every y_i to obtain estimates of s_i by $\hat{s}_i = g(y_i)$
(v) Transform the estimated sparse code back by $\mathbf{x}(t) = \mathbf{W}^T\hat{\mathbf{s}}(t)$.

3.3.1 Practical considerations

- Given a noisy image I, the whole image is first normalized to zero-mean and unit variance in order the method is not affected by the global image statistics. Without this normalization, one gets erronous results. After denoising these statistics can be recovered.
- Every possible NxN blocks in the image should be denoised in order to avoid blocking artifacts since neigboring pixels in an image strongly correlated. This is unfortunately very burdensome in terms of computation and memory requirements. A batch computation of the vectors $\mathbf{y}(t)$ requires the extraction of the vectors $\mathbf{x}(t)$ priorly and a considerable amount of storage. On the other hand if we want to compute one by one, the computational complexity increases.
- Each image block is again normalized to zero-mean and unit variance in order to compensate for possible non-stationarity of data.
- Any known ICA method can be used for the estimation of the mixing and separating matrices as long as we orthogonalize them in order to get the sparsifying transformation.

4 Experiments

In this section, we present the results of the implemented SCS denoising scheme. In all the experiments, a sparsifying basis extracted from 8x8 image blocks is used. The method we used for the estimation of the mixing and separating matrices is FastICA. Five images are reserved for both training and testing. The images in both the training and test sets were natural and man-made scenes. The sparsifying basis is shown below. Again we observe the localized, oriented and bandpass features.

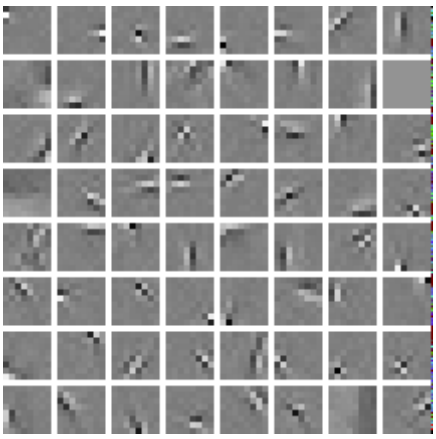

Fig. 4. The orthogonal sparsifying basis used in the experiments.

The experiments are carried for additive Gaussian noise levels 0.3 and 0.5 (the unity was the standard deviation of a given image). Results are compared with basic denoising methods such as Gaussian smoothing with 3x3 masks and median filtering in a 3x3 neighborhood. We have considered the peak signal-to-noise ratio (PSNR) as a performance index that can be computed as

$$PSNR = 10\log_{10}\frac{255^2}{\sum\sum\left[J(m,n)-I(m,n)\right]^2} \quad (26)$$

Note that as the noise level increases, PSNR value decreases.

The results are tabulated below.

Image	PSNR values(dB)			
	Corrupted	Gaussian smoothed	Median filtered	SCS applied
1	-25.68	-18.26	-19.43	-18.89
2	-23.27	-16.60	-17.83	-16.94
3	-19.22	-16.98	-18.39	-18.24
4	-19.77	-18.39	-18.88	-17.45
5	-23.85	-19.60	-20.23	-20.14

Table 1. PSNR values on the test set when noise level is 0.3.

Image	PSNR values(dB)			
	Corrupted	Gaussian smoothed	Median filtered	SCS applied
1	-29.87	-21.99	-23.18	-20.55
2	-27.60	-20.01	-21.18	-18.55
3	-23.69	-18.56	-19.91	-20.43
4	-24.21	-19.70	-20.60	-19.81
5	-28.14	-21.78	-22.75	-22.25

Table 2. PSNR values on the test set when noise level is 0.5.

In terms of the PSNR values, the SCS technique yields better results than median filtering and coparable results with Gaussian smoothing. However the PSNR is not always a good measure which reflects the perceptual quality of the output. Observing the following figures where the original image is corrupted with 0.3 and 0.5 levels respectively, the perceptual quality of SCS output is significantly better than any other technique. This is appearnt from Figs. 5 and 6.

Fig. 5. Result of denoising a test image with different methods when the noise level is 0.3; top: original, second row left: corrupted, second row right: Gaussian smoothed, third row left: median filtered, third row right: SCS applied. The perceptual quality of SCS is much better than the remaining ones.

Fig. 6. Result of denoising another test image with different methods when the noise level is 0.5; top: original, second row left: corrupted, second row right: Gaussian smoothed, third row left: median filtered, third row right: SCS applied. The perceptual quality of SCS is again significantly better than the remaining ones.

5 Conclusion

In this work, we have investigated the capabilities of sparse coding in the field of image denoising. The approach is based on ML estimation of super-Gaussian random variables corrupted with additive, uncorrelated Gaussian noise. Fortunately, ICA offers means of extracting sparse codes out of image data. Once such a code is found, the ML principle can be applied to denoise image data.

Experiments show that the perceptual quality of SCS outputs are significantly better than the ones of basic methods such as Gaussian smoothing and median filtering. True performance of SCS shall be evaluated in comparison with more sophisticated methods like Wiener filtering and Wavelet shrinkage methods.

One of the disadvantages of SCS is that one must always find a noise-free data set for estimating the sparsifying orthogonal transformation. Faithfully, as far as image data are of concern, such a noise-free data set can always be found. Another point is that the technique is computationally heavy since it involves sliding window approach. In Matlab, a 256x256 image is denoised in 4-5 minutes. An implementation in a lower-level language is necessary.

Further work can concentrate on applying the ML principle in case the additive noise is no more Gaussian. In this situation, more complex expressions would be obtained and the process would get much more sophisticated. Another issue may be considering of other sparsifying bases, such as independent subspace analysis bases that are known to model data dependendencies through the concept of feature invariants. This way, more essential features of image data can be caught.

References

[1] A. Hyvärinen, "Sparse Code Shrinkage: Denoising of Nongaussian Data by Maximum Likelihood Estimation", *Neural Computation*, 11(7):1739-1768, 1999.

[2] D.L. Donoho, I.M. Johnstone, G. Kerkyacharian, D. Picard, "Wavelet shrinkage: asymptopia? ", *Journal of the Royal Statistical Society ser. B.*, 57:301-337, 1995.

[3] A. Hyvaarinen, E. Oja, "Independent Component Analysis: A Tutorial", CIS, Helsinki University of Technology, Finland, April 1999.

[4] A. Hyvaarinen, E. Oja, "A fast fixed point algorithm for Independent Component Analysis", *Neural Computation*,vol. 9, pp. 1483-1492, 1997.

[5] B.A. Olshausen, D.J. Field, "Emergence of simple-cell receptive field properties bu learning a sparse code for natural images", *Nature*, no. 381, pp. 607-609, 1996.

APPENDIX

```
function [I_est,noise] = SCS2(img,I_noisy,W,whitenMat,dewhitenMat,p,noiseVar)
% ------------------------------------------------------------------------
% Sparse Code Shrinkage
% ------------------------------------------------------------------------
% [I_est,noise] = SCS2(img,I_noisy,W,whitenMat,dewhitenMat,p)
% ------------------------------------------------------------------------
% I/O variables
%         img --> original image
%     I_noisy --> corrupted version
%           W --> Orthogonal ICA transform
%   whitenMat --> Whitening Matrix
% dewhitenMat --> deWhitening Matrix
%           p --> struct that contains shrinkage parameters
% ------------------------------------------------------------------------
% Usage:
% -load prms_SCS.mat into the workspace
% -obtain a noisy version of an image, use gaussian noise
% ------------------------------------------------------------------------
% Ceyhun B. Akgül, Dec 2002
% ------------------------------------------------------------------------

tic,

% --------------------------------------
% fixed window size,
winSize = 8;
% --------------------------------------

numOfComps = size(W,1);

I = double(I_noisy);
% ----------------------------
% normalize the image
meanI = mean(mean(I));
I = I - meanI;
stdI = sqrt(mean(mean(I.^2)));
I = I/stdI;
% ----------------------------

[R,C] = size(I);
winArea = winSize^2;
Itemp = zeros(R+2*winSize-2,C+2*winSize-2);
Itemp(winSize:R+winSize-1,winSize:C+winSize-1) = I;
Itemp_out = zeros(R+2*winSize-2,C+2*winSize-2);

a = p.a;
b = p.b;
d = p.d;
alpha = p.alpha;
sparseness = p.sparseness;


% ----------------------------------------------------
% apply the shrinkage to the sparse components
% of each image windows
k = 1;
for r = 1:R+winSize-1
    for c = 1:C+winSize-1
        temp = Itemp(r:r+winSize-1,c:c+winSize-1);
        x = temp(:);

        % ----------------------------
        % preprocessing...
        % normalize and whiten each column vector
        meanx = mean(x);
        x = x-meanx;
        x = whitenMat*x;
        stdx = std(x);
        if stdx~=0
            x = x./stdx;
        end
        % ---------------------------

        % compute noisy sparse components
```

```
        y = W*x;
        % shrinkage phase
        for i = 1:numOfComps
            if sparseness(i)<1/sqrt(2)
                s_est(i) = (1/(1+noiseVar*a(i)))*sign(y(i))*max(0,abs(y(i))-b(i)*noiseVar);
            else temp = (abs(y(i))+a(i)*d(i))^2-4*noiseVar*(alpha(i)+3);
                if temp<0
                    s_est(i) = 0;
                else s_est(i) = sign(y(i))*max(0,0.5*(abs(y(i))-a(i)*d(i))+0.5*sqrt(temp));
                end
            end
        end

        s_est = s_est(:);
        % estimate denoised image blocks
        x_est = W'*s_est;

        % ----------------------------
        % reverse of the preprocessing
        if stdx~=0
            x_est = x_est.*stdx;
        end
        x_est = dewhitenMat*x_est;
        x_est = x_est+meanx;
        % ----------------------------

        Itemp_out(r:r+winSize-1,c:c+winSize-1) = ...
            Itemp_out(r:r+winSize-1,c:c+winSize-1)+reshape(x_est,winSize,winSize);

        k = k+1;
        if(~rem(k,2500))
            fprintf([num2str(k) 'th step completed!\n']);
        end
    end
end

% average the average of each estimated pixel
Itemp_out = Itemp_out/(winSize*winSize);
I_est = Itemp_out(winSize:R+winSize-1,winSize:C+winSize-1);

% restore the mean the variance of the image
I_est = I_est*stdI;
I_est = I_est + meanI;

noise = double(I_est)-double(I_noisy);

toc,

return;
```

```
function [S,prms] = estShrinkPrms(X,W)
%-----------------------------------------
% X: data
% W: orthogonal sparsifying transformation
%-----------------------------------------

[numOfComps,numOfSamples] = size(X);


S = W*X;

for i = 1:numOfComps
    s = S(i,:)-mean(S(i,:));
    m2 = moment(s,2);
    p0 = kernelEst(s,0,0.1);
    sparseness(i) = sqrt(m2)*p0;
end



for i = 1:numOfComps
    s = S(i,:)-mean(S(i,:));
    m2 = moment(s,2);
    p0 = kernelEst(s,0,0.1);
    sparseness(i) = sqrt(m2)*p0;
```

```
    if sparseness(i)<1/sqrt(2)
        mabs = mean(abs(s));
        b(i) = (2*p0*m2-mabs)/(m2-mabs^2);
        a(i) = (1/m2)*(1-mabs*b(i));
        alpha(i) = 0;
        d(i) = 0;
    else
        d(i) = sqrt(moment(s,2));
        k = m2*(p0^2);
        alpha(i) = (2-k+sqrt(k*(k+4)))/(2*k-1);
        a(i) = sqrt(alpha(i)*(alpha(i)+1)/2);
        b(i) = 0;
    end

end
prms.a=a;
prms.b=b;
prms.d=d;
prms.alpha = alpha;
prms.sparseness = sparseness;

return;
```

```
function p = kernelEst(X,x,h)

N = length(X);
p = 0;

for i = 1:N
    temp = (1/sqrt(2*pi))*exp(-((X(i)-x)/h)^2 / 2);
    p = p+temp;
end

p = p/(N*h);

return;
```

```
function [I_noisy,noisevar] = addGauss(I,mu,sigma_n);

[R,C] = size(I);
I = double(I);
stdI = std2(I);

gaussianNoise = wgn(R,C,20*log10(sigma_n*stdI));
noisevar = var(gaussianNoise(:));
I_noisy = uint8(I+gaussianNoise);

return;
```
