Published in Image Processing On Line on YYYY–MM–DD.
ISSN 2105–1232
This article is available online with supplementary materials, software, datasets and online demo at
http://dx.doi.org/10.5201/ipol.YYYY.XXXXXXXX

PREPRINT September 10, 2013

LSSC

Yohann Salaun[1] & Marc Lebrun[2]

[1] Polytechnique, France (yohann.salaun@polytechnique.org)
[2] CMLA, ENS Cachan, France (marc.lebrun@cmla.ens-cachan.fr)

Abstract

1 Overview

2 Theoretical Description

2.1 Non Local Means

Self similarities inside pictures have been studied for a long time. Efros and Leung used it for texture synthesis [1] and were followed by Buades, Coll and Morel for denoising purpose [2].
Considering a noisy picture $\mathbf{y}$ of n pixels seen as a column vector in $\mathbb{R}^n$, $\mathbf{y}$ is divided into n overlapping patches of equal size m. The i-th pixel of $\mathbf{y}$ is noted $\mathbf{y}[i]$ and the patch centered in $\mathbf{y}[i]$ and of size m is noted $\mathbf{y}_i$. Then, when two different patches $\mathbf{y}_i$ and $\mathbf{y}_j$ have similar values, their denoised version should also be similar. Moreover, assuming that the noise follows a Gaussian distribution, averaging similar patches should destroy the noise information and thus denoise the patch. Thus the denoised pixel $\mathbf{x}[i]$ is obtained with a linear combination of the others pixel in the noisy picture $\mathbf{y}$ weighted by the similarity of their corresponding patches:

$$\mathbf{x}[i] = \frac{1}{N}\sum_{j=1}^{n} G(\mathbf{y}_i - \mathbf{y}_j)\mathbf{y}[j] \tag{1}$$

where N is the normalization factor and G a Gaussian that weights the patch similarities.
Following this idea, Mairal et al. [5] divided the pictures into n overlapping patches in order to denoise through self-similarity methods.

2.2 Learned Sparse Coding

The second idea is to assume that the denoised patches can be approximated by a sparse linear combinations of elements from a basis set.
This basis set, that contains the denoised picture information, is called a dictionary $\mathbf{D} \in \mathbb{R}^{m\times k}$. The linear combination is represented by a vector $\boldsymbol{\alpha} \in \mathbb{R}^k$ called a code. The aim is to have a dictionary that is not redundant (independant columns), not too large ($k \ll n$) but that contains the whole picture information.

The denoising problem consists then of finding one dictionary for the picture and one sparse code for each patch. Thus it can be reformulated into a minimization problem where we look for the optimal dictionary and codes that are the most similar to each noisy patches:

$$\begin{aligned} &\min_{\mathbf{D},\boldsymbol{\alpha}_i} \sum_{i=1}^{n} ||\boldsymbol{\alpha}_i||_p \text{ s.t. } \forall i \in [1;n], \quad ||\mathbf{y}_i - \mathbf{D}\boldsymbol{\alpha}_i||_2^2 \leqslant \epsilon \\ &\text{with } \mathbf{D} \in \mathbb{R}^{m \times k} \text{ and } \forall i \in [1;n], \; \boldsymbol{\alpha}_i \in \mathbb{R}^k \end{aligned} \tag{2}$$

$\mathbf{D}\boldsymbol{\alpha}_i$ is the estimate of the i^{th} denoised patch which should be similar to the noisy patch and ϵ can be chosen according to the value of the estimated standard deviation of the noise.
Once the dictionary and the codes are learned, for each pixel, we have m estimations (from the m overlapping patches that contain it) and their averaging give us the denoised information of this pixel:

$$\forall i \in [1, n], \quad \mathbf{x}[i] = \frac{1}{m} \sum_{j=1}^{m} \mathbf{D}\boldsymbol{\alpha}_{\sigma(i,j)} \tag{3}$$

Where $\sigma(i,j)$ is the number of the patch that put pixel $\mathbf{x}[i]$ in j^{th} position.

Equation (2) is usualy minimized with $p = 0$ or 1. It becomes NP-hard to solve when $p = 0$ but a greedy algorithm such as Orthogonal Matching Pursuit (ORMP) [7] can quickly give a good approximation. The problem is convex with $p = 1$ and is efficiently solved with the Least Angle Regression (LARS) algorithm [3]. Experimental observations [8] have shown that the learning part is better with $p = 1$ and the recomposition part with $p = 0$.

2.2.1 Learning Dictionary

In this part, we consider that $p = 1$ in equation (2). The problems becomes convex and the equation is only solved in order to find the optimal dictionary $\mathbf{D}$ that could represents the denoised information of picture $\mathbf{y}$.
The method consists in updating the dictionary iteratively with only a small distribution of T patches. However they are chosen so that they are independently and identically distributed in the picture.
First, the initial dictionnary is learned offline on the 10 000 images of the PASCAL VOC'07 database using the online dictionary learning procedure of [4]. This procedure is then used on the noisy picture $\mathbf{y}$ in order to improve the dictionary efficiency.
For each patch $\mathbf{y}_i$, the corresponding code is computed with the current dictionary kept constant and then the dictionary is updated with all the previous codes.
Thus, we minimize iteratively the two equations below:

- For $\mathbf{D} \in \mathbb{R}^{m \times k}$ and patch t, $\boldsymbol{\alpha}^t = \text{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^k} ||\boldsymbol{\alpha}||_1$ s.t. $||\mathbf{y}_t - \mathbf{D}^{t-1}\boldsymbol{\alpha}||_2^2 \leq \lambda$
- For patches 1 to t and their codes $\boldsymbol{\alpha} \in \mathbb{R}^k$, $\mathbf{D}^t = \text{argmin}_{\mathbf{D} \in \mathbb{R}^{m \times k}} \frac{1}{t} \sum_{i=1}^{t} (\frac{1}{2} |\mathbf{y}_t^i - \mathbf{D}\boldsymbol{\alpha}^i||_2^2 + \lambda ||\boldsymbol{\alpha}^i||_1)$

The first equation is solved with the LARS algorithm.
TODO
The second equation can be re-written:

$$\mathbf{D}^t = \text{argmin}_{\mathbf{D} \in \mathbb{R}^{m \times k}} \frac{1}{t} (\frac{1}{2} \text{Tr}(\mathbf{D}^T \mathbf{D} B^t) - \text{Tr}(D^T C^t) \tag{4}$$

where B and C are incrementaly updated with:

- $B^t \leftarrow B^{t-1} + \boldsymbol{\alpha}^t \boldsymbol{\alpha}^{tT}$

- $C^t \leftarrow C^{t-1} + \mathbf{y}^t \boldsymbol{\alpha}^{tT}$

The dictionary update is thus computed through a block-coordinate descent with warmrestarts [9]. This is an iterative method that updates the columns of the dictionary needing only matrices B and C and no matrix inversion in opposite to other approaches as Newton method. Moreover, since the convex optimization problem (Equation (4)) admits separable constraints in the updated blocks (columns), convergence to a global optimum is guaranteed. However Mairal et al empirically found that a single iteration of the dictionary update was sufficient to achieve convergence.

2.2.2 Denoising

TODO: MARC :)

2.3 Simultaneous Sparse Coding

The idea that developped Mairal et al. in this section is that similar patches should have similar decomposition upon the dictionary. Thus, they created a partition of the picture into n_C groups of similar patches called clusters. They then added a simultaneous condition which impose similar decompositions for patches of the same cluster and equation (2) became:

$$\begin{aligned} &\min_{\mathbf{D},\boldsymbol{\alpha}_{i,j}} \sum_{j=1}^{n_C} \sum_{i=1}^{n_j} \frac{||\boldsymbol{\alpha}_{i,j}||_q^p}{n_j} \text{ s.t. } \forall j \in [1; n_C], \quad \sum_{i=1}^{n_j} ||\mathbf{y}_i - \mathbf{D}\boldsymbol{\alpha}_{i,j}||_2^2 \leqslant \epsilon_j \\ &\text{with } \mathbf{D} \in \mathbb{R}^{m \times k} \text{ and } \forall i \in [1; n_j], \forall j \in [1; n_C], \ \boldsymbol{\alpha}_{i,j} \in \mathbb{R}^k \end{aligned} \tag{5}$$

where n_j is the size of the j^{th} cluster and ϵ_j is chosen with respect to the noise inside it.

Equation (5) is usualy minimized with $(p, q) = (0, \infty)$ or $(1, 2)$. As for Equation (2), it becomes NP-hard to solve when $(p, q) = (0, \infty)$ and Simultaneous Orthogonal Matching Pursuit (SORMP) [10] gives a good approximation. The problem is convex when $(p, q) = (1, 2)$ and a slight modification of the LARS algorithm gives a solution. Once again, the convex problem is used for the learning part while the SORMP is used for the denoising part.
As for the non-simultaneous sparse coding, the denoised pixel is approximated by an average of the denoised versions of all the overlapping patches.

2.3.1 Simultanous Learning

Not seen in details. The algorithm used is the Simultaneous LARS which is similar to LARS but a "simultaneous" condition that uses cluster information.

2.3.2 Simultanous Denoising

Not seen in details. The algorithm used is the Simultaneous ORMP which is similar to ORMP but a "simultaneous" condition that uses cluster information.

3 Notations

4 Algorithm Description

4.1 Algorithm Overview

The previous parts are used in order to perform a global denoising. First, a rough denoising is realized upon the whole picture. This way, a coherent dictionary is learned and similar patches can be easily regrouped. Then, the clustering is performed and the previous process is applied once more on each cluster indepently. However, the similarity between patches is also used, adding a simultaneous condition upon patches. The whole algorithm is summed up in Algorithm 1.

Input : noisy picture $\mathbf{y}$, pre-learned dictionary $\mathbf{D}$
Output : denoised picture $\mathbf{x}$
Update $\mathbf{D}$ with LARS
Denoise $\mathbf{y}$ with ORMP
Regroup patches into clusters
for t = *1..N* **do**
    Update $\mathbf{D}$ with Simultaneous LARS for each cluster independently
    Denoise each cluster with SORMP
end

Algorithm 1: Online Dictionary Learning

Note that in practice $N = 1$ works well and the patchwise update for the dictionary is very costy and do not seem to increase the quality of the denoised picture.
The pseudo code is presented below with the same subsections as the theoretical part above. We implemented it in C++ and the link between the pseudo code and the C++ functions is precised each time.

4.2 Learned Sparse Coding

4.2.1 Learning Dictionary

- Main function of the dictionary learning part - *trainL1*:

It computes a distribution of iid patches with the function *getRandList*. It then iteratively minimizes equation (2) with either the dictionary $\mathbf{D}$ fixed or the codes $\boldsymbol{\alpha}$ fixed. When the dictionary is fixed, the LARS is used in C++ function *computeLars*. When the codes are fixed, the C++ function called is *updateDictionary*.

Input : initial dictionary $\mathbf{D}^0 \in \mathbb{R}^{m \times k}$, number of iterations $\mathbf{T}$, regularization parameter λ
Output : learned dictionary $\mathbf{D}$
Initialization : $\mathbf{A}^0 \in \mathbb{R}^{k \times k} \leftarrow 0$, $\mathbf{B}^0 \in \mathbb{R}^{m \times k} \leftarrow 0$
Compute i.i.d. sampling of $\mathbf{T}$ patches of the noisy picture $\mathbf{Y}_T$ **for** $t = 1..\boldsymbol{T}$ **do**

$\mathbf{y}_t = \mathbf{Y}_T$[t]
Sparse coding: compute with LARS algorithm:

$$\boldsymbol{\alpha}^t = \text{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^k} ||\boldsymbol{\alpha}||_1 \text{ s.t. } ||\mathbf{y}_t - \mathbf{D}^{t-1}\boldsymbol{\alpha}||_2^2 \leq \lambda$$

$\mathbf{A}^t \leftarrow \mathbf{A}^{t-1} + \boldsymbol{\alpha}^t \boldsymbol{\alpha}^{tT}$
$\mathbf{B}^t \leftarrow \mathbf{B}^{t-1} + \mathbf{y}_t^t \boldsymbol{\alpha}^{tT}$
Update dictionary from $\mathbf{D}^{t-1}$ to $\mathbf{D}^t$ so that:

$$\mathbf{D}^t = \text{argmin}_{\mathbf{D} \in \mathbb{R}^{m \times k}} \frac{1}{t} \sum_{i=1}^{t} (\frac{1}{2} |\mathbf{y}_t^i - \mathbf{D}\boldsymbol{\alpha}^i||_2^2 + \lambda ||\boldsymbol{\alpha}^i||_1)$$

end
return $\mathbf{D}^T$

Algorithm 2: *trainL1*

- Dictionary update - *updateDictionary*:

Block-coordinate descent with warmrestarts algorithm that minimizes Equation (4).
TODO: se renseigner sur la methode ? pk si A(j,j) = 0 ...

Input : input dictionary $\mathbf{D}= [\mathbf{d}^1, \ldots, \mathbf{d}^k] \in \mathbb{R}^{m \times k}$,
$\mathbf{A} = [\mathbf{a}^1, \ldots, \mathbf{a}^k] \in \mathbb{R}^{k \times k}$, $\mathbf{B} = [\mathbf{b}^1, \ldots, \mathbf{b}^k] \in \mathbb{R}^{m \times k}$
Output : updated dictionary $\mathbf{D}$
for $j = 1..k$ **do**

Update the j^{th} column:
if $\boldsymbol{A}(j,j) = 0$ **then**

$$\mathbf{d}^j \leftarrow 0$$

end
else

$$\mathbf{u}^j \leftarrow \frac{1}{\mathbf{A}(j,j)} (\mathbf{b}^j - \mathbf{D}\mathbf{a}^j) + \mathbf{d}^j$$

$$\mathbf{d}^j \leftarrow \frac{1}{\max(||\mathbf{u}^j||_2, 1)} \mathbf{u}^j$$

end

end
return $\mathbf{D}$

Algorithm 3: *updateDictionary*

- Least Angle Regression (LARS) - *computeLars*:

The LARS algorithm is very long and has been split up for better readibility.
($\min^+$ is the minimum between strictly positive values only)
TODO: prciser la version + pk inversion bizarre des matrices

Input : Input dictionary $\mathbf{D} \in \mathbb{R}^{m \times k}$, noisy patch $\mathbf{y} \in \mathbb{R}^m$, constraint $\lambda \in \mathbb{R}$
Output : code $\boldsymbol{\alpha} \in \mathbb{R}^k$
Initialization:
Initialize variables

for $i = 1..k$ **do**
  New atom:
  Add a new atom in $\boldsymbol{\alpha}$ if there is no downdating

  Compute steps:
  Compute the initial step (γ)
  Compute the critical step that impose downdating (stepDownDate)
  Compute the critical step that breaks the loop and ends the algorithm (stepMax)

  Update code:
  Compute the new code $\boldsymbol{\alpha}$ and downdate or break if needed

end
return $\boldsymbol{\alpha}$

Algorithm 4: *computeLars* - Main

Compute Gram matrix:
$G \in \mathbb{R}^{k \times k} \leftarrow \mathbf{D}^T\mathbf{D}$

Initialize patch norm and check if there is no trivial solution:
normPatch $\in \mathbb{R}^+ \leftarrow ||\mathbf{y}||_2^2$
if *normPatch* $< \lambda$ **then**
  return 0
end

Initialize code and active index:
$\boldsymbol{\alpha} \in \mathbb{R}^k \leftarrow \mathbf{0}$
$\mathcal{A} \in [0; k]^k \leftarrow \mathbf{0}$

Compute most correlated element:
$\hat{\mathbf{c}} \in \mathbb{R}^k \leftarrow \mathbf{D}^T\mathbf{y}$
$C \in \mathbb{R}^+ \leftarrow \max_{j=1..k}(|\hat{\mathbf{c}}[j]|)$
currentInd $\leftarrow j$ s.t. $\hat{\mathbf{c}}[j] = C$

Add a new atom at first iteration:
newAtom $\leftarrow$ **True**

Algorithm 5: *computeLars* - Initialization

if *newAtom* **then**
  $\mathcal{A}[i] \leftarrow$ currentInd
  $G_A \in \mathbb{R}^{k\times i}, G_S \in \mathbb{R}^{i\times i}$
  $G_A[i^{th}$ column$] \leftarrow G[$currentIndth column$]$
  $G_S[i^{th}$ line$] \leftarrow G_A[$currentIndth line$]$
  symmetrize G_S: copy the lower part of G_S into its upper part
  UPDATE G_S^{-1}
end

Algorithm 6: *computeLars* - New atom

Initial step:
u $\in \mathbb{R}^i \leftarrow G_S^{-1}(\mathrm{sgn}(\hat{\mathbf{c}}[\mathcal{A}[j]]))_{j\in[1;i]}$
$C \leftarrow |\hat{\mathbf{c}}[\mathcal{A}[1]]| = \max_{j=1..k}(|\hat{\mathbf{c}}[j]|)$
$\gamma \in \mathbb{R}_*^+ \leftarrow \min^+\left(\frac{C+\hat{\mathbf{c}}[j]}{1+(G_Au)[j]}, \frac{C-\hat{\mathbf{c}}[j]}{1-(G_Au)[j]}\right)_{j \text{ s.t. } \mathcal{A}[j]=0}$
currentInd $= j$ s.t. $\gamma = \frac{C\pm\hat{\mathbf{c}}[j]}{1\pm(G_Au)[j]}$
Downdate step:
ratio $\in \mathbb{R}^i \leftarrow \left(-\frac{\boldsymbol{\alpha}[\mathcal{A}[j]]}{u_j}\right)_{j\in[1;i]}$
stepDownDate $\in \mathbb{R}_*^+ \leftarrow \min^+$(ratio)
downDateInd $\in [1;k] \leftarrow \mathcal{A}[j]$ s.t. ratio[j] = stepDownDate
Breaking step:
$a \in \mathbb{R} \leftarrow \sum_{j\in[1;i]} \mathrm{sgn}(\hat{\mathbf{c}}[\mathcal{A}[j]])u[j]$
$b \in \mathbb{R} \leftarrow \sum_{j\in[1;i]} \hat{\mathbf{c}}[\mathcal{A}[j]]u[j]$
$c \in \mathbb{R} \leftarrow$ normPatch - λ
$\Delta \in \mathbb{R} \leftarrow b^2 - ac$
stepMax $\in \mathbb{R}^+ \leftarrow \min(\frac{b-\sqrt{\Delta}}{a}, C)$

Algorithm 7: *computeLars* - Compute steps

$\gamma \leftarrow \min(\gamma$, stepDownDate, stepMax$)$
for $j = 1..i$ **do**
  $\boldsymbol{\alpha}[\mathcal{A}[j]] \leftarrow \boldsymbol{\alpha}[\mathcal{A}[j]] + \gamma$u[j]
end
$\hat{\mathbf{c}} \leftarrow \hat{\mathbf{c}} - \gamma G_A u$
normPatch $\leftarrow$ normPatch + $a\gamma^2 - 2b\gamma$
if $|\gamma| < 10^{-6}$ **or** $\gamma = stepMax$ **or** $normPatch < 10^{-6}$ **or** $normPatch - \lambda < 10^{-6}$ **then**
  break
end
if $\gamma = stepDownDate$ **then**
  DOWNDATE G_S^{-1} w.r.t downDateInd
  $\mathcal{A}$[downDateInd] $\leftarrow$ 0
  $\boldsymbol{\alpha}$[downDateInd] $\leftarrow 0$
  newAtom $\leftarrow$ **False**
  i $\leftarrow$ i - 1
end
else
  newAtom $\leftarrow$ **True**
  i $\leftarrow$ i + 1
end

Algorithm 8: *computeLars* - Update code

Input : Gram matrix $G_S \in \mathbb{R}^{i \times i}$, and its former inverse to update $G_S^{-1} \in \mathbb{R}^{i-1 \times i-1}$
Output : updated $G_S^{-1} \in \mathbb{R}^{i \times i}$
if i = *1* **then**
  return $\frac{1}{G_S}$
end
$u \leftarrow G_S^{-1} G_S[i^{th}$ line]
$\sigma \leftarrow \frac{1}{G_s(i,i) - u.G_S[i^{th} \text{ line}]}$
$G_s^{-1}(i,i) \leftarrow \sigma$
$G_s^{-1}[i^{th}$ line] $\leftarrow$ -σu
return $G_s^{-1} \leftarrow G_s^{-1} + \sigma u u^T$

Algorithm 9: *updateGram* Marc: Algo correspondant, mais à valider.

Input : pseudo-Gram matrix $G_A \in \mathbb{R}^{k \times i}$ Gram matrix $G_S \in \mathbb{R}^{i \times i}$, and its inverse $G_S^{-1} \in \mathbb{R}^{i \times i}$, criticalInd $\in [1; k]$, current iteration i
Output : downdated matrices $G_A \in \mathbb{R}^{k \times i-1}, G_S, G_S^{-1} \in \mathbb{R}^{i-1 \times i-1}$
$\sigma \leftarrow \frac{1}{G_S^{-1}(\text{criticalInd},\text{criticalInd})}$
$u \leftarrow G_S^{-1}$[criticalIndth line] without its criticalIndth coefficient
for *j= criticalInd:i-1* **do**
  $G_A[j^{th}$ column] $\leftarrow G_A[(j+1)^{th}$ column]
  for *k= 1:criticalInd-1* **do**
    $G_S(j,k) \leftarrow G_S(j+1,k)$
    $G_S^{-1}(j,k) \leftarrow G_S^{-1}(j+1,k)$
  end
  for *k= criticalInd:i* **do**
    $G_S(j,k) \leftarrow G_S(j+1,k+1)$
    $G_S^{-1}(j,k) \leftarrow G_S^{-1}(j+1,k+1)$
  end
end
$G_s^{-1} \leftarrow G_s^{-1} - \sigma u u^T$

Algorithm 10: *downdateGram* Marc: Algo correspondant, mais à valider.

Glossary

Image Credits

© IPOL (there's no need to credit this image, here is used as an example.)

References

[1] A. Efros, and T. Leung *Texture synthesis by non-parametric sampling.* ICCV, 1999.

[2] A. Buades, B. Coll, and J. Morel *A non-local algorithm for image denoising.* CVPR, 2005.

[3] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani *Least angle regression.* Ann. Statist., 32(2):407499, 2004.

[4] J.Mairal, F. Bach, J. Ponce, and G. Sapiro *Online dictionary learning for sparse coding.* ICML, 2009.

[5] J. Mairal, F. Bach, J. Ponce, G. Sapiro and A. Zisserman *Non-local Sparse Models for Image Restoration*. International Conference on Computer Vision, 2009.

[6] J. Mairal *Representations parcimonieuses en apprentissage statistique, traitement dimage et vision par ordinateur*. PhD thesis, 2010.

[7] S. Mallat and Z. Zhang *Matching pursuit in a timefrequency dictionary*. IEEE T. SP, 41(12):33973415, 1993.

[8] M. Elad and M. Aharon *Image denoising via sparse and redundant representations over learned dictionaries*. IEEE T. IP, 54(12):37363745, 2006.

[9] D. P. Bertsekas *Nonlinear programming*. Athena Scientific Belmont, 1999.

[10] J. A. Tropp *Algorithms for simultaneous sparse approximation*. Sig. Proc., 86:572602, 2006.