Lab: Image Inpainting using Online Dictionary Learning

Xiaomin Zhang, Jinyu Xia, Jinman Zhao

December 17, 2015

1 Introduction

Image inpainting [2] refers to the process of restoring missing or damaged areas in an image. One efficient algorithm for image inpainting is to use the image signal sparse representation over a redundant dictionary. It is assumed that the natural signal for an image can admit a sparse decomposition over a redundant dictionary, and thus, finding out the dictionary would be critical. In this lab, we apply an online optimization algorithm for dictionary learning, based on stochastic approximation, which scales up gracefully to large datasets.

This lab is designed to help you use several machine learning tools to deal with image inpainting problem. The problem of inpainting is encountered in various image processing applications: image restoration, editing (e.g., object removal), disocclusion in image-based rendering, interpolation, loss concealment, texture synthesis or image resizing (e.g.,enlargement). There are several approach dealing with the problem of image inpainting. One method called patch sparse representation is related to the pattern recognition techniques we learn from the class. Additionally, the type of image inpainting we choose to explore in this lab is image restoration.

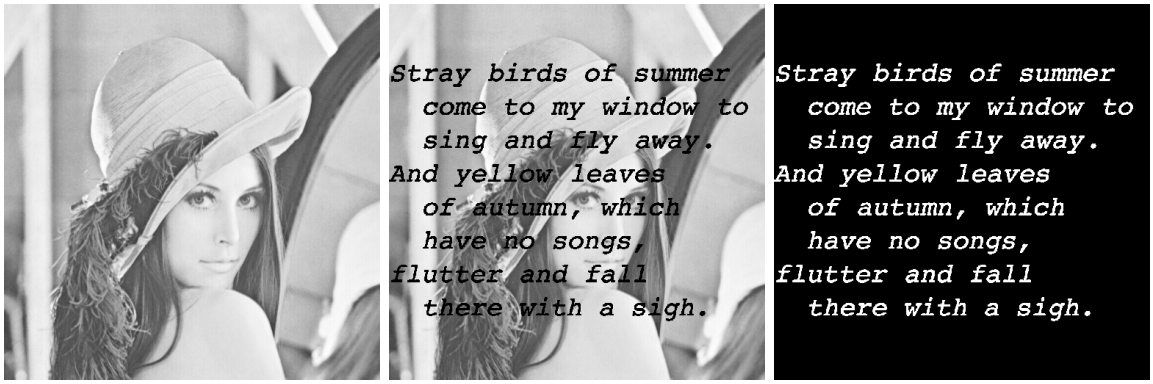
The image restoration problem is concerned with recovering an original image from various forms of degradations. In the restoration problem, the missing region is generally not too large, hence, local diffusion and patch-based or global methods give satisfactory results. Examples are shown in Fig. 1.

2 Overview

In this section, we will describe the basic concepts and introduces some of the Matlab tools that we will use in this lab. First, make sure that the data file `LennaDict.mat` is in your MATLAB current directory.

2.1 Lasso Regression

The sparse coding problem with fixed dictionary $\mathbf{D}$ is a Lasso linear regression. The main reason of applying Lasso regression on the weights $\boldsymbol{w}$ here is that it yields a sparse solution



(a) original (b) text-embedded (c) mask

(d) original (e) contaminated (f) mask

Figure 1: Some example input for image inpainting.

for $\boldsymbol{w}$, so that we can end up finding out a sparse representation over the learned dictionary $\mathbf{D}$.

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{x} - \mathbf{D}\boldsymbol{w}\|_2^2 + \lambda \|\boldsymbol{w}\|_1 \tag{1}$$

Note that l_1 regularizer do not have a closed-form solution. Iterative algorithms like the Landweber iteration can be applied to this optimization. In dictionary learning problem, for equation (1), we need to fix $\mathbf{D}$, given a signal $\mathbf{x}$, the optimize weights $\boldsymbol{w}_i$, which is the weight corresponding to the i^{th} observed entry from a data set $\mathbf{X}$.

2.2 Sparse Dictionary Learning

To conduct dictionary learning, we need to use equation (2). A natural approach to solving this problem is to alternate between the two variables, minimizing over one while keeping the other one fixed. More specifically, given the data set $\mathbf{X}$, the way of computing $\mathbf{D}$ is that within each iteration, first with $\mathbf{D}$ fixed, we optimize $\boldsymbol{w}$, and then with $\boldsymbol{w}$ fixed, we optimize $\mathbf{D}$. We need to keep doing these two regressions alternatively with an iteration time T.

$$\hat{\mathbf{D}}, \hat{\mathbf{W}} = \arg\min_{\mathbf{D}, \mathbf{W}=[\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k]} \frac{1}{2} \|\mathbf{X} - \mathbf{D}\mathbf{W}\|_F^2 + \lambda \sum_{i=1}^{k} \|\boldsymbol{w}_i\|_1 \tag{2}$$

Note that generally, the iteration time T (which is also the number of selected image patches n) should generally be much larger than the length of a weight vector $\boldsymbol{w}_i$ m and the number of atoms in dictionary $\mathbf{D}$.

2.3 Missing data

One important thing to note is that in most cases, there would be missing data in the data set $\mathbf{X}$. To deal with this situation, the learning process of our dictionary $\mathbf{D}$ should only base on the observed entries $\boldsymbol{x}_i$, where i is integer from 1 to n.

$$\arg\min_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{x} - \mathbf{D}\boldsymbol{w}\|_2^2, \text{ where } \mathbf{D} \text{ should only learn from observed image patches.} \quad (3)$$

2.4 Image Inpainting

In image restoration problems, we want to recover the corrupted part of an image, such as text, noise, scratches.

This seems like a totally different problem. But see it like this way: for an given image, we can extract huge amount of small areas from it (called *patches*) and view each of them as data point. Thus we can obtain a large redundant data set and can use them to learn a dictionary for the sparse representation of the local structure of the given image. The idea is also related to the so called wavelet method, where a carefully pre-designed dictionary (a set of "wavelet" functions) is used to analyze, compress or process the image. However, the advantage here is to learn a dictionary specialized for the given image itself and thus we may be able to achieve better performance or gain some idea of its local structure.

The idea of restoration here is simple: treat the parts that need to be inpainted in the image as missing data in some patches, impute them using the dictionary learned from other intact patches until the parts is completely filled.

3 Warm-up

With basic concepts and theories introduced, in this section, we would like you to get familiar with some building blocks for the lab.

3.1 Implement Lasso

Lasso is essentially a regularized least square question. See equation (1). Write your own version of `Lasso`. What is parameter λ doing here?

Generate your own noised data using the following script and try to recover `wtrue`. Show the resulting $\boldsymbol{w}$ and how it changes with different values of λ.

```
X = randn(100,5);
wtrue = [0;2;0;-3;0];
Y = X*wtrue + randn(100,1)*0.1; % small added noise
```

3.2 Image Operation

Try your hand at some basic operators on Image Processing using MATLAB. Table 1 refers to some handy MATLAB functions that is useful in this lab. We also write some helper functions specialized for our lab (Table 2).

Table 1: Some image related MATLAB functions

Name	Functionality
`imread`	load an image from file as a matrix
`imwrite`	write a matrix as an image into a file
`imshow`	show a matrix as an image

Table 2: Some helper functions we write for the lab

Name	Functionality
`imgData (filename)`	load an image from file as a matrix and automatically scale them into [0,1] scale
`getPatch (mat,sm,ipatch)`	take the i_{patch}-th patch (a $sm \times sm$ vector) from matrix mat
`getPatchMask (r,c,sm,ipatch)`	a $r \times c$ (0,1)-matrix indicates the appearance of the i_{patch}-th patch
`getPatchNum (r,c,sm)`	number of possible $sm \times sm$ patches in a $r \times c$ image
`getPatchPos (r,c,sm,ipatch)`	the starting position of the i_{patch}-th patch in a $r \times c$ image

Now try to load image `LennaWithText.png` (Fig. 2) as a matrix with each pixel ranges between 0 and 1. Could you find where the bad pixels is? Show your work. We will use it as a mask ([0,1]-matrix/image serves as an indicator that tells us where to be inpainted) for inpainting later.

Try to fetch some patches with, say, 10×10 pixels. Can you pick some ones containing eye, hair or something interesting within it? What is the 133295^{th} patch $\boldsymbol{p}_{133295}$ looks like?

3.3 Play with dictionary

In this section, we will provide you a calculated dictionary for Lenna. We visualize it in Fig. 3, so you can get some intuition as for what a dictionary looks like. Later you can visualize your own dictionary using `showPlot.m`.

Use the given dictionary and `Lasso` you just wrote. Try to learn sparse coding for each patch you extracted in the previous section and use the sparse coding to reconstruct the patches. What do they look like? Try to compare them with the original ones.

(extra) What if there are corrupted/unknown pixels in your patches? Can you propose some approaches to reconstruct them?

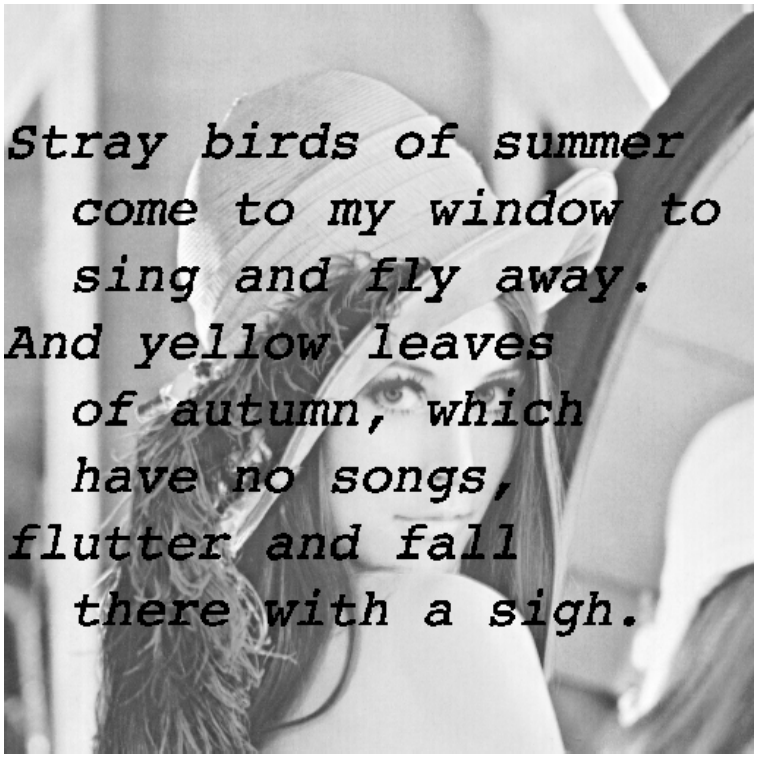


Figure 2: `LennaWithText.png`.

3.4 `lasso`?

There is a MATLAB function `lasso`. As its name suggests, it should do the Lasso regression for us. (optional) Try this MATLAB function `lasso`. Compare its result with your own version of `Lasso`. Does it agree with your expectation? What is different? (Hint: MATLAB `lasso` minimizes an objective function slightly different from what we learned from class. You should be able to convert its results to what we want not-so-difficultly. Check MATLAB documentation for details.)

In our case, the built-in `lasso` runs much faster than our own version of Lasso. If you are in the similar case, feel free to use `lasso` in the rest to save some running time. =) Or you can still use your own version for easy debug. (extra) What techniques does MATLAB use to make it so fast? Let us know if you have some clue!

4 Main Lab

In this section you will see how to learn a dictionary and how to use it to recover some images.

4.1 Dictionary Learning

Consider a signal $\boldsymbol{x}$ in $\mathbb{R}^m$. We say that it admits a sparse approximation over a dictionary $\mathbf{D}$ in $\mathbb{R}^{m\times k}$ with k columns referred to as atoms, if one can find a linear combination of a

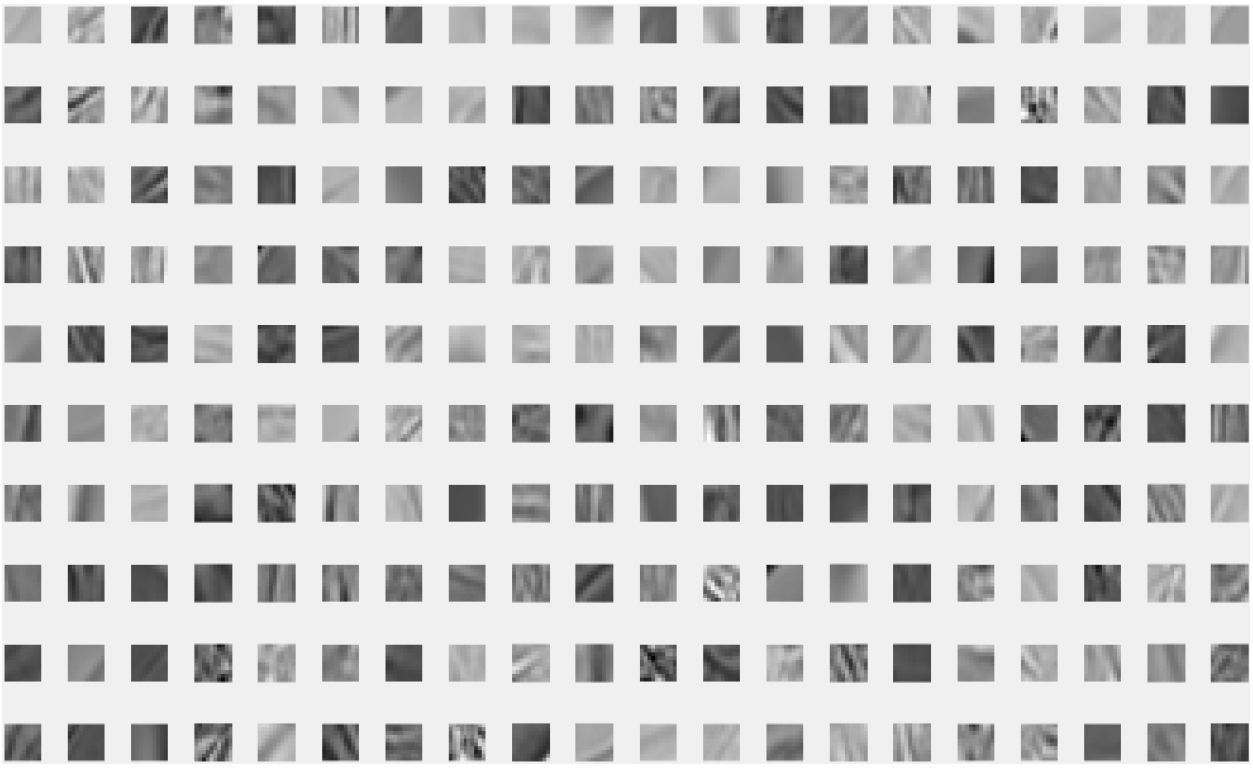


Figure 3: The dictionary of the first layer of `Lenna.png` (i.e., the red layer). This Dictionary has 100×200 atoms.

"few" atoms from $\mathbf{D}$ that is close to the signal $\mathbf{x}$. The number of samples n is usually larger, whereas the signal dimension m is relative small. In this lab, we choose $m = 100$ for 10×10 image patches, and $n \geq 100,000$ for typical image processing application. Besides, since we miss some data here, we extract T patches which only contain uncontaminated pixels. In general, we also have $k \ll T$ (e.g. $k = 100$ for $T = 20,000$), and each signal only uses a few elements of $\mathbf{D}$ in its representation. The setting gives us freedom in the choice of dictionary and at the same time helps capture some inherent structure of the data set..

As covered in the course lecture, the way of computing $\mathbf{D}$ is that within each iteration, fixing $\mathbf{D}$ first, optimizing $\boldsymbol{w}$, and then fixing $\boldsymbol{w}$, and optimizing $\mathbf{D}$. Both of the optimizations are just regularized LS problems, and we keep doing these two steps alternatively.

The problem is that when we have huge amount of data point, it would be tedious to optimize the equation involving the whole data matrix. So we turn to stochastic method which focus on one data point at a time, which saves computational power and can give handy approximate result at each update. Another similar example for the idea is gradient descent compared to stochastic gradient descent.

4.2 Online Dictionary Learning

4.2.1 Algorithm Outline

The fundamental two-step optimization structure of dictionary learning remain the same, but instead of optimizing for all the data points as a whole, we randomly pick a data point within the data set and focus on it within the iteration. This is to say we learn "sample" by "sample". Update the weight and the dictionary every time the new sample is drawed. To acquire a good online learning method is not quiet easy. What we adopt here is **Algorithm1** [4].

Algorithm 1 Online dictionary learning

Input: $x \in \mathbb{R}^m \backsim p(\boldsymbol{x})$, T (number of iterations), randomly initialized dictionary D, lambda λ, stepsize τ, convergence difference shreshold Δ.
Output: learned dictionary
$\mathbf{A_0} \leftarrow 0, \mathbf{B_0} \leftarrow \mathbf{0}$,
for $t = 1$ **to** T **do**
Randomly draw $\boldsymbol{x}_t$ from $p(\boldsymbol{x})$
Use Lasso regression to compute sparse coding:
$min \frac{1}{2}\|\mathbf{x_t} - \mathbf{D}\boldsymbol{w}_t\|^2 + \lambda\|\boldsymbol{w_t}\|_1$
$\mathbf{A}_t \leftarrow \mathbf{A}_{t-1} + \boldsymbol{w}_t\boldsymbol{w}_t^T$
$\mathbf{B}_t \leftarrow \mathbf{B}_{t-1} + \boldsymbol{x}_t\boldsymbol{w}_t^T$
Compute $\mathbf{D}_t$ using Dictionary update:
$\mathbf{D}_t \triangleq argmin \frac{1}{t}\sum \frac{1}{2}\|\mathbf{x}_i - \mathbf{D}\boldsymbol{w}_i\|^2 + \lambda\|\boldsymbol{w}_i\|$
return $\mathbf{D}_T$

To facilitate the understanding of this online dictionary learning, we provide several additional explanations regarding to it:

- Practically, we can see $p(\boldsymbol{x})$ as a matrix $\mathbf{P} \in \mathbb{R}^{m \times T}$ consists of n columns, each corresponding to the patched you randomly selected before. Every time during the iteration we just pick the t-th column of $\mathbf{P}$. Note the the image patches randomly selected are $\sqrt{m} \times \sqrt{m}$, so that you could transform each square patch into a column vector by applying MATLAB function reshape().

- In order to preserve the current and previous information of weight $\boldsymbol{w}$ and training data $\mathbf{x_i}$, we add current Lasso result and the current training data to matrices $\mathbf{A}$ and $\mathbf{B}$. Sometimes we probably want to give a trade off between current and previous information. Then we could calculate A and B as $\mathbf{A_t} = \beta\mathbf{A_{t-1}} + \sum_{i=1}^{\eta}\boldsymbol{w}_{t,i}\boldsymbol{w}_{t,i}{}^T$, $\mathbf{B_t} = \beta\mathbf{B_{t-1}} + \sum_{i=1}^{\eta}\mathbf{x}_{t,i}\boldsymbol{w}_{t,i}{}^T$, where β is the trade off parameter. If we do not want any previous effect, we could just set β as 0. Besides, we also need attention that some bad value of β could lead to not convergence.

4.2.2 Dictionary Update

As mentioned in the previous sections, we update $\boldsymbol{w}$ and dictionary D separately. This section is to present how to update the dictionary. The algorithm we adopt here is introduced in [4]. You can see $\mathbf{A}$ and $\mathbf{B}$ are created for the purpose of this algorithm.

Besides, using some simple algebra, we would yield the solution of the dictionary update with respect to the j-th column $\mathbf{d}_j$ and keep the constraint $\mathbf{d}_j^T\mathbf{d}_j \leqslant 1$. If we do not have this constraint to $\mathbf{d}_j$, $\mathbf{d}_j$ will goes to very large, and then $\boldsymbol{w}$ will goes to really small. The specific algorithm is shown as followed[4]:

Algorithm 2 Dictionary Update

Input: $\mathbf{D} = [\mathbf{d}_1, ..., \mathbf{d}_k] \in \mathbb{R}^{m\times k}$ (input dictionary),
$\mathbf{A} = [\boldsymbol{a}_1, ..., \boldsymbol{a}_k] \in \mathbb{R}^{k\times k} = \sum_{i=1}^{t} \boldsymbol{w}_i \boldsymbol{w}_i{}^T$,
$\mathbf{B} = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k] \in \mathbb{R}^{k\times k} = \sum_{i=1}^{t} \boldsymbol{x}_i \boldsymbol{w}_i{}^T$
Output: updated dictionary
  repeat
    for $j \leftarrow 1$to$k$ **do**
      Update the j-th column to optimize for
      $\mathbf{u}_j \leftarrow \frac{1}{\mathbf{A}_{jj}}(\mathbf{b}_j - \mathbf{D}\mathbf{a}_j) + \mathbf{d}_j$.
      $\mathbf{d}_j \leftarrow \frac{1}{\max(\|\boldsymbol{u}_j\|_2, 1)\boldsymbol{u}_j}$.
  until convergence
  return $\mathbf{D}_T$

Try implement **Algorithm1** and **Algorithm2** and use it to learn a dictionary for Lenna. Plot your learned dictionary to see if it appears good. (You can compare it with Fig. 3.).

4.3 Image Inpainting

Now that you have a dictionary, you can use it to do the inpainting. Here, we provide you with the `imgInpaint.m` (see Appendix A for its idea). So you do not need to worry about all the minor details of image processing. What you are expected to achieve is to write a `main.m` function. First, call Algorithm above you have written to obtain the Dictionary. Then, call the function `imgInpaint.m` to acquire the image whose mark should be removed.

Based on what we have introduced in previous sections, you should be able to recover Lenna as some thing like Fig. 4. How does your dictionary perform? For "Lenna" figure, does the dictionary work well?

In addition, perform the whole process on your own to inpaint `ThreePeopleWithM.png`. (We provide `MShapeMask.png` if you have some trouble extracting the mask.) Show your work. Does this dictionary work as well as the former one? And why?

Figure 4: `Restored Lenna`

References

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11):4311–4322, 2006.

[2] Christine Guillemot and Olivier Le Meur. Image inpainting: Overview and recent advances. *Signal Processing Magazine, IEEE*, 31(1):127–144, 2014.

[3] Boris Mailhé, Sylvain Lesage, Rémi Gribonval, Frédéric Bimbot, and Pierre Vandergheynst. Shift-invariant dictionary learning for sparse representations: extending k-svd. In *Signal Processing Conference, 2008 16th European*, pages 1–5. IEEE, 2008.

[4] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 689–696. ACM, 2009.

[5] Julien Mairal, Michael Elad, and Guillermo Sapiro. Sparse learned representations for image restoration. In *Proc. of the 4th World Conf. of the Int. Assoc. for Statistical Computing (IASC)*. Citeseer, 2008.

A `imgInpaint.m`

The idea of inpainting using a learned dictionary of the target image is to gradually fill in small amount of unknown pixels close to good areas. If we look into patches at the border of corrupted and intact area of the image, we can hopefully always find a certain number of patches that only contains very few pixels to be recovered. For such patches, we can treat the unknown pixels as missing data for a data point, use the rest of the entries to obtain a sparse coding and then impute the missing data. The entire process can be performed until all the area is filled. The psudocode is shown in Algorithm 3.

Algorithm 3 Inpainting with Dictionary

Input: $\mathbf{X}_0 \in [0,1]^{r\times c}$ (the image to be inpainted), $\mathbf{M}_0 \in \{0,1\}^{r\times c}$ (a mask indicates the pixels to be inpainted), $\mathbf{D} = [\boldsymbol{d}_1, ..., \boldsymbol{d}_k] \in \mathbb{R}^{m\times k}$ (dictionary learned using good patches), λ_{good} (how important is pixels from the original image compared to inpainted pixels), λ_{lasso} (how important is the sparsity in coding)

Output: inpainted image

$t \leftarrow 0$

repeat

$\mathbf{C}_{t+1} \leftarrow \lambda_{good}(\mathbf{1} - \mathbf{M}_t)$

$\mathbf{X}_{t+1} \leftarrow \mathbf{X}_t \hat{*} \mathbf{C}_{t+1}$

for $(i, j) \leftarrow$ enough positions of patches sampled in $\mathbf{X}_t$ **do**

$\boldsymbol{p} \leftarrow \mathbf{R}_{ij}\mathbf{X}_t$

$\boldsymbol{m} \leftarrow \mathbf{R}_{ij}\mathbf{M}_t$

if $0 < \|\boldsymbol{m}\|_1$ and is small **then**

$\hat{\boldsymbol{\alpha}} \leftarrow \arg\min_{\boldsymbol{\alpha}} \|\boldsymbol{m}\boldsymbol{p} - \boldsymbol{m}\mathbf{D}\boldsymbol{\alpha}\|^2 + \lambda_{lasso}\|\boldsymbol{\alpha}\|_1$

$\boldsymbol{p}' \leftarrow \mathbf{D}\hat{\boldsymbol{\alpha}}$

$\mathbf{C}_{t+1} \leftarrow \mathbf{C}_{t+1} + \mathbf{R}_{ij}^{-1}\mathbf{1}$

$\mathbf{X}_{t+1} \leftarrow \mathbf{X}_{t+1} + \mathbf{R}_{ij}^{-1}\boldsymbol{p}'$

$\mathbf{X}_{t+1} \leftarrow \mathbf{X}_{t+1} \hat{/} \mathbf{C}_{t+1}$

$t \leftarrow t + 1$

until All pixels are inpainted **return** $\mathbf{X}_t$

* $\hat{*}, \hat{/}$ are element-wise multiplication and division; binary matrix $\mathbf{R}_{ij}$ extracts the square $\sqrt{m} \times \sqrt{m}$ patch of coordinates [i, j]