Spectral graph theory diffusion-based approach to image smoothing

Artur Semião
(Dated: 10th January 2023)

Based on the paper from F. Zhang and E.R. Hancock [1], it was studied an image smoothing technique, based on spectral graph theory diffusion. The method was applied to a grey-scale 256×256 image corrupted with additive Gaussian white noise through the convolution of the corrupted image with the heat kernel. The heat kernel is a function of the graph's Laplacian, which controls the rate of the diffusion process through a carefully defined weight function.

I. INTRODUCTION

The goal of image smoothing methods is to recover the original image from a noisy measured signal,

$$v(i) = u(i) + n(i), \tag{1}$$

where $u(i) \equiv u(\mathbf{x}_i)$ is the underlying latent signal of interest at a position $\mathbf{x}_i = (x_{1i}, x_{2i})$, $v(i)$ is the noisy measured signal and $n(i)$ is the noise perturbation at a given pixel i. The smoothing of an image aims at reducing the distortions provoked by the noise while preserving the important features of the original image.

The present work will focus on a diffusion-based partial differential equation (PDE) filter for image smoothing presented by F. Zhang and E.R. Hancock [1]. Most diffusion-based PDEs methods for image smoothing assume that the image is a continuous two-dimensional function on $\mathbb{R}^2$ which must be discretised for numerical implementation. However, images with high levels of noise may not be sufficiently smooth, presenting a big obstacle in the implementation of such methods. To tackle this problem, we present a model whose diffusion process acts not on the image function itself, but the on a graph representation of the image. This representation will be discussed in the Section II. Hence, instead of using diffusion-based PDEs in a continuous domain, we do it on a graph and no such discretisation of the image is required, since the graph of the pixel lattice will be already discrete.

The anisotropic diffusion throughout the graph with time is captured by the heat equation, whose fundamental solution, the heat kernel, is found by exponentiating the graph Laplacian with time. Finally, in order to smooth the image, the noise corrupted image is convolved with the heat kernel. The convolution is numerically implemented using the Krylov subspace technique, explained in Section III.

The present work will finish with some experiments done on an image corrupted with additive Gaussian white noise in Section IV.

II. IMAGE SMOOTHING WITH SPECTRAL GRAPH THEORY

This section will first focus on how we represent an image as a weighted undirected local connected graph and then how, using spectral graph theory, we find the heat kernel for such graph.

A. Graph representation of a grey-scale image

We represent a grey-scale image as an undirected weighted graph Γ. The graph Γ is represented by a node set V and an edge set $E \subseteq V \times V$. The nodes V of the graph are the pixels of the image with intensities $v(i)$ ranging from 0 (black) to 255 (white). The number of connections that each pixel has, i.e. the number of edges $e_{ij} \in E$, must satisfy a connectivity requirement on the pixel lattice, imposed by a threshold distance parameter r.

The graph Γ must preserve the image structure without losing any information. This is done by defining a function that maps changes in the image data to edge weights $w(i,j)$. The weight function, $w(i,j) : V \times V \to (0,1]$, by determining the weight of an edge e_{ij}, is crucial in our model since it will control the flow of heat across the nodes of the graph, i.e. the pixels. If $w(i,j)$ is large, heat can flow easily between pixel i and j, whereas if $w(i,j)$ is small, the flow of heat is difficult. The model uses a Gaussian weighting function:

$$w(i,j) = \begin{cases} \exp\left\{-\frac{d_\sigma^2(i,j)}{\kappa^2}\right\} & \text{if} \quad \|\mathbf{x}_i - \mathbf{x}_j\|_2 \leq r, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where r is the distance threshold between two neighbouring pixels, assuring local connectivity of the graph, and κ is a parameter used to regularise the exponential decay. The factor $d_\sigma(i,j)$ is a function that measures the similarity between two pixels i and j.

B. Pixel similarity: neighbouring windows

To simplest way to measure the similarity between two neighbour pixels, say i and j, is by computing the norm between the pixel's intensities $v(i), v(j)$:

$$d(i,j) = |v(i) - v(j)|. \tag{3}$$

If two pixels have the same intensity, the similarity is $d(i,j) = 0$, the weight will therefore be maximal $w(i,j) = 1$ and the heat will flow freely between these neighbouring pixels. If instead, the difference of intensities of both pixels is very large, i.e. $d(i,j)$ is large, the weight function goes to 0 and the heat diffusion between those two pixels is very hard.

However, a more reliable approach to study pixel similarity, discussed by Buades et al.[3], consists on representing each pixel not only using the individual intensity of the pixel itself, but by using the intensities of all the neighbouring pixels within a patch $\mathcal{N}_k$ of size $n \times n$ centred at a pixel k. Therefore we must now measure the similarity between two patches $\mathcal{N}_i$ and $\mathcal{N}_j$ whose intensities are encoded as vectors $v(\mathcal{N}_i)$ and $v(\mathcal{N}_j)$. The generalisation of (3), is by computing the Euclidean distance between these two vectors:

$$d(i,j) = \|v(\mathcal{N}_i) - v(\mathcal{N}_j)\|_2. \tag{4}$$

However, we want to give more importance to the centre of the respective patches, therefore we apply a Gaussian filtering to (4). The similarity between two pixels i and j is now measured using the Gaussian weighted Euclidean distance:

$$\begin{aligned} d^2_\sigma(i,j) &= \|v(\mathcal{N}_i) - v(\mathcal{N}_j)\|^2_{2,\sigma} \\ &\equiv (v(\mathcal{N}_i) - v(\mathcal{N}_j))^T G_\sigma (v(\mathcal{N}_i) - v(\mathcal{N}_j)), \end{aligned} \tag{5}$$

where G_σ is a fixed diagonal matrix containing Gaussian weights. For images with low level of noise, no such weight would be needed. However, for images corrupted with mild and high levels of noise, one may face big oscillations of intensity values for nearby pixels and therefore this pre-processing is widely used as a regularisation operator to improve the performance of nonlinear diffusion-based smoothing models.

C. Heat kernel of an undirected weighted graph

The heat kernel is the fundamental solution for the heat equation. For the classic case, a function $H_t(\vec{x}) \equiv H(\vec{x}, t) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$, the fundamental solution is the solution of the homogeneous heat equation

$$\left(\frac{\partial}{\partial t} + \Delta_x\right) H_t(\vec{x}) = 0, \tag{6}$$

where Δ is the d-dimensional Laplacian, with initial conditions

$$H_0(\vec{x}) = \delta(\vec{x}) \tag{7}$$

Then, the general solution for a a general function $g(x,t)$ with initial condition

$$g(x,0) = g_0(x), \tag{8}$$

where $g_0(x)$ is any bounded continuous function, is given by the convulution

$$g(x,t) = \int_{\mathbb{R}^d} H_t(x-y)\, g_0(y)\, dy. \tag{9}$$

In spectral graph theory, the Laplacian operator of a graph Γ is $L = T - W$, where T is the diagonal degree matrix $T(i,i) = deg(i) \equiv \sum_{j \in V} w(i,j)$, and W is the weights matrix, with elements $W(i,j) = w(i,j)$. The Laplacian entries are given by:

$$L(i,j) = \begin{cases} deg(i) - w(i,i) & \text{if} \quad i = j \\ -w(i,j) & \text{if} \quad e_{ij} \in E \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

In matrix form, the generalised heat equation is

$$\frac{\partial H_t}{\partial t} = -LH_t, \tag{11}$$

where the fundamental solution $H_t(i,j)$ is now a $|V| \times |V|$ matrix with initial condition $H_0 = I_{|V|}$ (the $|V| \times |V|$ identity matrix) and the Laplacian L is given by the expression (10). The solution for (11) is found by exponentiating the Laplacian Matrix with time t,

$$H_t = e^{-tL}. \tag{12}$$

The heat kernel represents the evolution of thermal energy on an initial configuration. The Laplacian is hence a very important matrix since its elements, the weights, dictate the rate of the flow across the graph. If all edge weights would be the same, then the diffusion would be isotropic. However, this is not the case as $w(i,j)$ are in general different. The edge weights will therefore act as thermal conductivity and the diffusion will be anisotropic.

The diffusion process starts from an initial configuration, the noisy image in our case, where the initial amount of thermal energy "injected" at each node is determined by the corresponding pixel intensity of the noisy image. The initial thermal energies will diffuse through the graph along the edges of each node as the time t progresses. The thermal evolution of the image is implemented through the convolution of the vectorised

noisy image $\vec{\mathcal{I}}$ with the heat kernel H_t, in matrix notation:

$$v_t(j) = \sum_{i=1}^{|V|} \mathcal{I}(i) \times H_t(i,j), \tag{13}$$

where $v_t(j) \equiv v(j,t)$ is a function that gives the intensity (thermal energy) of the pixel j at a given time t, i.e. $v_t(j) : V \times R \to [0, 255]$.

As we noted, the heat kernel H_t is a function of the graph Laplacian L, and it could be represented through the eigenvectors and eigenvalues of the very same Laplacian. However, as I will note in the next section, the Laplacian for such graph will be too big to work with, and therefore, I introduce the "compressed Laplacian". This new Laplacian, together with the Krylov subspace technique, allows the computation of the convolution (13) of the heat kernel with the initial noise corrupted vector image $\vec{\mathcal{I}}$, without recurring to the Laplacian's eigenspectrum.

III. NUMERICAL IMPLEMENTATION OF DIFFUSION-BASED SMOOTHING TECHNIQUE

The following section is dedicated to the main difficulties I came across upon the numerical implementation of the diffusion-based PDE smoothing technique presented in the last section, applied to grey-scale images. The programming language used to develop the algorithm is *Python*, with the help of useful packages for image processing such as *Numpy* or *Scipy*.

The main challenge I faced is the size of the Laplacian Matrix, fundamental for the whole model. The size of the Laplacian matrix for a $|V| \times |V|$ pixels image would be $|V|^2 \times |V|^2$. Considering an image with $256 \times 256 = 65536$ pixels, the Laplacian matrix of a is an extremely sparse 65536×65536 matrix, where the non zero entries are floating point numbers. Hence, in terms of memory, any normal computer could not build such big matrix, let alone compute its exponential form to build the heat kernel. In the first part of this Section I will explain how I handled this problem and could build a "compressed" form of the Laplacian matrix by just considering the non zero entries.

The consequences of building this compressed Laplacian is that its exponential form is not by any chance the same of the original Laplacian, neither its eigenspectrum. In the second part I will explain how one may use the Krylov subspace projection technique to compute the action of a matrix exponential operator on a vector without having to compute explicitly the matrix exponential.

A. Compressed Laplacian

As mentioned in the introduction to this section, the Laplacian matrix for a 256×256 pixel image would be too heavy for any normal computer to build and manipulate. Since we are dealing with local connected graphs, each row of the Laplacian matrix has $(2r+1)^2$ non zero entries with, where r is the distance threshold from the expression (2). Consider for $r = 1$, each row would have only 9 non zero entries given by the negative 8 non zero weights of the pixel with its surroundings plus the diagonal entry, given by the degree of the pixel minus the weight with itself. This would mean that for each row with 65535 elements, only 9 would be different from 0, and in total $65535 \times (65535 - 9)$ zeros. The only pixels with which a given pixel i will have non zero weights is given by the neighbour of i, $U(i)$. The size of this neighbour $U(i)$ will be $(2r+1)^2$, centred at i. Consider, for example, the pixel $i = 260$, for $r = 1$, the neighbour will be:

$$U(260) = \{3, 4, 5, 259, 260, 261, 515, 516, 517\}. \tag{14}$$

The respective row 260 of the Laplacian Matrix will be

$$L(260, j) = \begin{bmatrix} 0 \\ ... \\ 0 \\ -w(260,3) \\ -w(260,4) \\ -w(260,5) \\ 0 \\ ... \\ 0 \\ -w(260,259) \\ deg(260) - w(260,260) \\ -w(260,261) \\ 0 \\ ... \\ 0 \\ -w(260,515) \\ -w(260,516) \\ -w(260,517) \\ 0 \\ ... \\ 0 \end{bmatrix}^T$$

The main idea for condensing all the (useful) information, is by creating a $|V|^2 \times (2r+1)^2$ matrix, the compressed Laplacian L_c. This new matrix is the same as L but without all the 0's. The row 260 of L_c is therefore

$$L_c(260,j) = \begin{bmatrix} -w(260,3) \\ -w(260,4) \\ -w(260,5) \\ -w(260,259) \\ deg(260) - w(260,260) \\ -w(260,261) \\ -w(260,515) \\ -w(260,516) \\ -w(260,517) \end{bmatrix}^T$$

Generally, we have

$$L_c(i,j) = \{L(i,j) : j \in U(i)\}. \tag{15}$$

The arising issue of this hypercondensation of L is that the matrix exponential of the Laplacian e^{-tL} and the matrix exponential of the compressed Laplacian e^{-tL_c} will have no such similar mapping. Nevertheless, the Krylov subspace projection technique will make life easier since we don't have to compute the matrix exponential explicitly.

B. Krylov Subspace technique

Consider a $n \times n$ sparse matrix A, a n-vector v and a scalar $t \in \mathbb{R}^+$. One may approximate

$$w(t) = e^{tA}v = v + \frac{(tA)}{1!}v + \frac{(tA)^2}{2!}v + ... \tag{16}$$

with an element of the Krylov subspace,

$$\mathcal{K}_m(tA, v) = \text{Span}\{v, (tA)v, ..., (tA)^{m-1}v\}, \tag{17}$$

where m is the dimension of the Krylov subspace and is much smaller than n. The approximation being used is

$$w(t) \approx \beta V_{m+1} \exp\big(t\bar{H}_{m+1}\big)e_1, \tag{18}$$

where e_1 is the first column of the identity matrix I_m and $\beta = \|v\|_2$. $V_{m+1} = [v_1, ..., v_{m+1}]$ is the orthonormal basis and $\bar{H}_{m+1}$ the upper Hessenberg matrix resulting from the Arnoldi process.

The known Arnoldi process does not require one to compute $e^{-tL_c}\vec{\mathcal{I}}$ explicitly, but just $(-tL_c)\vec{\mathcal{I}}$ in order to obtain the basis of the Krylov subspace. Note, however, that this multiplication is not so straightforward since now we are multiplying a $|V|^2 \times (2r+1)^2$ matrix by a $|V|^2$ vector. In order to do so, one must select the $(2r+1)^2$ indexes from $\vec{\mathcal{I}}$. Taking again the example of the pixel $i = 260$, the multiplication between the Laplacian L and $\vec{\mathcal{I}}$ for the row 260 is:

$$\sum_{j=1}^{|V|^2} L(260,j) \times I(j) = \sum_{j \in U(260)} L_c(260,j) \times I(j)$$

Therefore, by just selecting the indexes of the pixels that belong to the neighbourhood of i for $\vec{\mathcal{I}}$, we are multiplying a $|V|^2 \times (2r+1)^2$ matrix by a $(2r+1)^2$-vector, giving the same result as the full L and $\vec{\mathcal{I}}$ but now without all the useless multiplications. This compressed multiplication and its optimisation is one of the fundamental aspects to be taken care of in the algorithm.

The implementation of the Krylov subspace method for solving (13) is inspired on the algorithm of the Expokit package [5], a MATLAB subroutine, with help of lecture notes from the ACME programme of Brigham Young University [6].

C. Handling graph's borders

The borders of the graph were a recurring issue throughout the algorithm implementation. I had to handle the borders in two occasions, in the pixels neighbourhoods $U(i)$ and for the pixels patches $\mathcal{N}_i$.

In general, to have the neighbours of a certain pixel, I created a matrix where the entries where simply from 0 to |rows|×|columns| - 1, where rows and columns are the number of vertical and horizontal pixels, respectively. Then I used the $Scikit - Learn$ function to get all the neighbours of the pixels. To handle the pixels in the borders, I padded that matrix with negative ones, where the deepness of the padding would be equal to the threshold distance. Consider a 3×3 image with $r = 1$:

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & 1 & 2 & -1 \\ -1 & 3 & 4 & 5 & -1 \\ -1 & 6 & 7 & 8 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{19}$$

The neighbour of 1 is

$$U(1) = \{-1, -1, -1, 0, 1, 2, 3, 4, 5\}.$$

And by just implementing a condition in the code of jumping negative pixels, the weight with the borders is not computed.

For the patches case, I padded the image with zeros, with the deepness of the pad this time was given by the patch radius which would is obtained by $(n-1)/2$, with n the patch size. Like this, when computing the similarity between a pixel i that lies on a border, all the intensity of the pixel's patch that lie outside of the image are 0.

D. Algorithm résumé

The algorithm, deconstructed into steps, is:

1. Corrupt a ground truth image with noise to create the noise corrupted image.

2. Build the compressed Laplacian matrix L_c of the graph with which we represent the noise corrupted image using the weight function (2).

3. Compute $\vec{v}_t$ from (13) at time t using the Krylov subspace technique.

IV. EXPERIMENTS AND RESULTS

This section is dedicated to the experimental results obtained from the heat diffusion algorithm using spectral graph theory, described in the last section. The algorithm was applied to noisy 256×256 pixels grey-scale images. I chose the threshold distance to weights computation $r = 1$ and so the image is represented as an 8-connected undirected weighted graphs.

A. Noise addition

The addition noise started by first scaling the intensities of the ground truth *Lenna* image, with pixel intensities v laying in the interval $0 - 255$ to the range $0 - 1$. Then, I generated additive white Gaussian noise (AWGN) with mean 0 and standard deviation 0.05 and added to the ground truth scaled image. Finally, I rescaled the image back to the range 0-255, creating the *Lenna* noisy image with intensities u. The comparison between the ground truth and the noisy image can be seen in Fig. 1. For $\sigma = 0.05$, the root mean square error (RMSE) of the ground truth image and the noise corrupted image, is about 12.77. From now on, the RMSE will be the main tool to analyse the performance of the algorithm.

B. Parameters optimisation

In order to get the best results, which means smoothing the corrupted image while preserving the image's fine details, or simply getting a low RMSE, I studied how the algorithm behaves under the variation of parameters present in the weight function (2), $\{\kappa, n, \sigma\}$, and the diffusing stopping time t_s.

The weight function (2) depends on three parameters. In order to compute $d_\sigma(i, j)$ we must fix the size n of a general patch $\mathcal{N}_k$ surrounding a pixel k and the σ for the Gauss kernel whose values will constitute the diagonal entries of the matrix G_σ from (5). The size of the patch for a given pixel could vary from $n = 1$ (only the pixel itself) until a large n. However, one must be aware that as n grows, the surrounding pixels may start to belong to far away regions of the image and hence differ to much from the central pixel. Hence, we choose a patch size $n = 5$ which is a far size and does not take to much CPU time. For $n = 5$, a good standard deviation σ to compute the Gaussian weighted euclidean distances (5), is half the standard deviation of the additive white noise, $\sigma = 0.025$. In Fig.2 we can see the differences for $n = 1$ and $n = 5$, keeping all the the other parameters equal. For $n = 5$, fine details such as *Lenna*'s hat or lips are better preserved.

After choosing both n and σ, we are able to compute the pixel similarity $d_\sigma(i, j)$. However, we still have to choose the parameter κ. To adapt κ for images with different contrasts, we normalise d_σ from 0 to 1, rescaling the intermediate values accordingly. Under this normalisation, the value of κ is normally set within the interval $0.05 - 0.15$. The choice of the diffusion stopping time t_s is also important. An optimal smoothed image is achieved with a small diffusion time (order of the unity). To investigate the effect of these parameters on the smoothing process, I computed the RMSE for varying values of κ and t_s, displayed in the following table:

		κ				
		0.06	0.08	0.10	0.12	0.14
t_s	1	9.99	7.84	6.87	6.70	6.93
	2	9.01	7.12	6.82	7.24	8.02
	3	8.48	6.96	7.10	7.90	9.00
	4	8.16	6.97	7.43	8.51	9.85
	5	7.94	7.04	7.75	9.07	10.58

Table I. RMSE values resulting of smoothing process for different values of κ and t_s.

We see that to get good, i.e. low, values of RMSE, for small values of κ, the diffusion time t_s must increase, while as we increase κ, the best values of RMSE are for low values of t_s. The optimal RMSE is 6.70 for $\kappa = 0.12, t_s = 1$, while the highest value is for $\kappa = 0.14, t_s = 5$. Both images for the best and worst combinations of κ and t_s are displayed in Fig. 3. We see that for a big diffusion type, the heat equation starts to blur the image, similar to isotropic diffusion (Gaussian smoothing) on images.

V. CONCLUSIONS

The main objective of the project I proposed myself to the Complex Networks course was to smooth a noise corrupted 256×256 grey-scale image, using a diffusion-based smoothing technique through a graph spectral approach. The smoothed images were obtained by convolving the heat kernel with noise corrupted image over

time. The heat kernel, a function of the graph's Laplacian, is the fundamental solution for the matrix form of the heat equation.

We first saw that computing the pixel's similarity using all the intensities of the patches centred at the pixels itself instead of just the pixel intensity helps with preserving the image's fine details. We then proceed to find the optimal values of κ, t_s for a fixed patch size $n = 5$.

From the didactic point of view, this project gave me important insights in three main fronts. The first one was on spectral graph theory. The second one was on matrix manipulation with *Python*, more specifically *Numpy*. I learned about optimisation techniques when trying to reduce the time that my program would take to perform its tasks, such as creating the compressed Laplacian or when performing the compressed multiplication. The third was on digital image processing, me as a student who is orientating its path in physics toward a more theoretical one, I would have never came across digital image processing if not for this project. I really enjoyed since I learned about noise the pixel structure of an image, adding noise, different methods of image smoothing, even if in the end I just focused in the diffusion-based ones.

Although I think the compressed Laplacian technique was a good idea to avoid some memory problems, it may still be largely optimised.

[1] F. Zhang, E. R. Hancock, "Graph spectral image smoothing using the heat kernel", Pattern Recognition vol.41, (2008) 3328 - 3342.

[2] Daniel Remondini, Complex Networks 2021-2022 course notes, University of Bologna.

[3] A. Buades, B. Coll, J. Morel, "A review of image denoising algorithms, with a new one", Multiscale Model. Simul. vol.4, no.2, (2005) 490-530.

[4]

[5] R. Sidje, "Expokit: a software package for computing matrix exponentials", ACM Trans. Math. Software vol.24, no.1, (1998) 130-156.

[6] ACME programme lecture notes, "Arnoldi Methods", Lab 16, Brigham Young University.

[7] F. Catte, P. L. Lions, J.M Morel, T. Coll, "Image Selective Smoothing and Edge Detection by Nonlinear Diffusion", SIAM Journal on Numerical Analysis Vol. 29, No. 1, (1992), 182-193.

[8] L. Alvarez, P.L. Lions and J.M. Morel, "Image Selective Smoothing and Edge Detection by Nonlinear Diffusion II", SIAM Journal on Numerical Analysis Vol. 29, No. 3, (1992), 845-866.

[9] P.Milanfar, "A Tour of Modern Image Filtering: New insights and methods, both practical and theoretical", IEEE Signal Processing Magazine Vol. 30, No. 1, (2013), 106 - 128.

[10] J. Darbon; A. Cunha; T. F. Chan; S. Osher; G. J. Jensen, "Fast nonlocal filtering applied to electron cryomicroscopy", 2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 14-17 May 2008.

Figure 1. Comparison of the ground truth *Lenna* image (left) and noise corrupted *Lenna* image (right) comparison. The second image was obtained by adding a zero-mean AWGN with $\sigma = 0.05$ to the ground truth image.

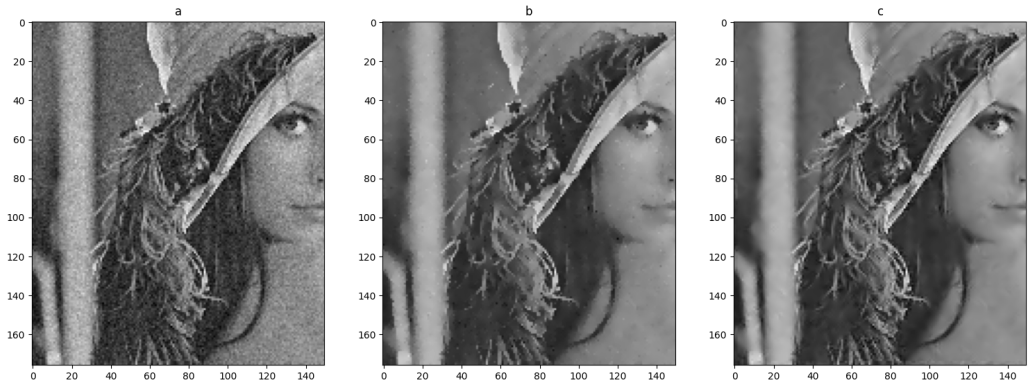

Figure 2. Comparison of algorithm effect on noisy image for different patch sizes for the same κ and t_s. (a) noisy image, RMSE = 12.77, (b) smoothed image with $n = 1$, RMSE = 7.26, (c) smooth image with $n = 5$, RMSE = 6.8. The smoothed result in (c) better preserves the fine details than in (b).

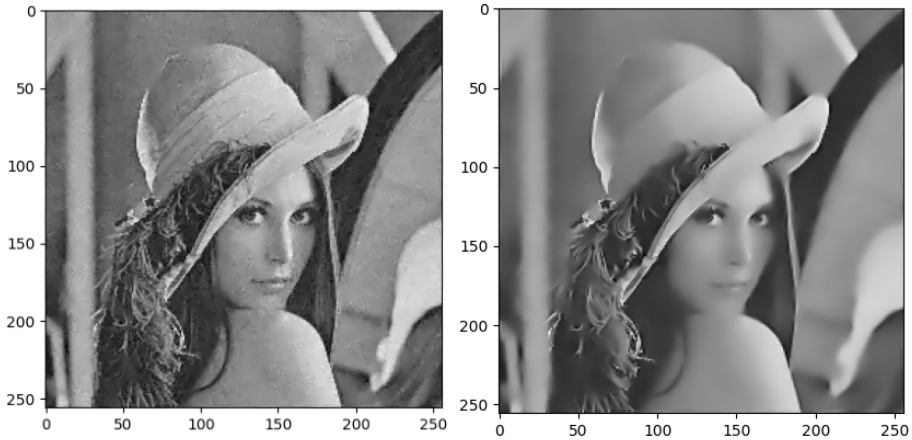

Figure 3. Comparison of two smoothing results. The optimal (left) and the least optimal (right) for a given range of parameters κ and t_s.