Software Methodologies COMP2231 2019/2020 Image processing assignment

clvp22

January 19, 2020

1

1.1 Introduction

The non-local means (NL-means) denoising algorithm is an effective algorithm for removing noise from a desired image. The purpose of denoising algorithms is to recover the original image from a noisy image. The wide class of denoising algorithms share the same basic principle, that denoising an image can be acheived by averaging; some algorithms perform averaging locally, by minimizing variation in the pixel neighbourhood, using calculus, or by frequency analysis and wavelet thresholding methods. The NL-means algorithm that will be described in Section 1.2 [1] also uses averaging. The NL-means algorithm works by estimating a pixels's value as a weighted average of the values of all other pixels in the image. The weights are based on the similarity between the local neighbourhood of the pixel being processed and the local neighbourhoods of the other pixels.

1.2 NL-means algorithm

We are given some noisy image $v = \{v(p)|p \in P\}$ where P represents the image domain. For each pixel p we calculate some value $NL[v](p)$ and we construct the denoised image using these values as we go along. $NL[v](p)$, for a pixel p is computed as a wieghted average of all the pixels in the image v, it is given by,

$$NL[v](p) = \sum_{q \in P} w(p,q)v(p),$$

where $w(p,q)$ is a list of weights denoting the similarity between the neighbourhoods of pixel p and q, and $0 \leq w(p,q) \leq 1$ and $\sum_q w(p,q) = 1$ hold. The similarity between the neighbourhoods of pixel p and q is calculated using the vectors $v(N_p)$ and $v(N_q)$, these vectors represent a square neighbourhood of fixed size N x N (typically 7 x 7) centered around pixel p and q respectively. The similarity between the two neighbourhoods is then calculated using the formula $\| v(N_p) - v(N_q) \|^2_{2,\alpha}$, where $\alpha > 0$ is the standard deviation of the gaussian kernel, and the formula itself is effectively the squared euclidean distance between the two vectors. Pixels with similar gaussian neighbourhoods to p have larger weights than those that don't, the weights are precisely defined as follows,

$$w(p,q) = \frac{1}{Z(p)} e^{-\frac{\|v(N_p) - v(N_q)\|^2_{2,\alpha}}{h^2}},$$

where $Z(p)$ is a normalizing agent, and h is a filtering parameter (controls the amount of smoothing). The NL-means algorithm is special because it compares the geometry of the neighbourhoods at p and q not just the values at p and q. This means if the two pixels p and q that have very similar values, but strucurally different neighbourhoods, we get a small weight $w(p,q)$.

2 Implementations

2.1 Pixelwise implementation

The pixelwise implementation that can be found in [2] is the non-local means [1,2] filter, which for a colour image $v = (v_1, v_2, v_3)$ and pixel p is given by,

$$NL[v_i](p) = \frac{1}{Z(p)} \sum_{q \in B_p} v_i(q) w(p,q),$$

$$Z(p) = \sum_{q \in B_p} w(p,q),$$

where v_i represents some colour channel $i = 1, 2, 3$, Z(p) is our normalizing agent again, and B_p is a neighbourhood of fixed size B x B (typically 21 x 21) centered around p. Notice how this implementation of the NL-means algorithm differs from the original description; our pixel p is not compared to every other pixel in the image, it is instead compared to every other pixel in B_p. This algorithm limits our research

window dramatically in the interests of computation time. Typically our research window has a size of 21 x 21, but for larger σ values (more guassian noise) we can use a 35 x 35 window. The weights $w(p,q)$ are calculated as before,

$$w(p,q) = e^{-\frac{\|v(N_p)-v(N_q)\|^2_{2,\alpha}}{h^2}},$$

notice that the $Z(p)$ normalizing function is omitted for simplicity and has been moved outside the summation. After the procedure each pixel p and its colour channels are fixed with their new values, giving us a denoised image.

2.2 Patchwise implementation

The patchwise implementation differs from the pixelwise implementation, in that it denoises patches of pixels as opposed to a single pixel at at time. The algorithm denoises some colour image $v = (v_1, v_2, v_3)$ and a particular fixed size square region $B = B_p$ (centered at pixel p) as follows,

$$NL[v_i](B) = \frac{1}{Z} \sum_{N=N_q \in B_p} v_i(N) w(B,N),$$

$$Z = \sum_{N=N_q \in B_p} w(B,N),$$

where again v_i represents some colour channel $i = 1,2,3$, Z is our normalizing agent, B_p is the reserach window, and N_p is the N x N comparison vector. Our weights function $w(N_p, N_q)$ is formulated the same as before. However, with this implementation we dispose of $|N|^2$ possible estimates for each pixel. These estimates can be averaged at the end for each pixel location, to construct the resulting denoised image,

$$v_i(p) = \frac{1}{|N|^2} \sum_{N=N_q | q \in N_p} N_{i,p}.$$

The patchwise implemenation improves on the pixelwise implementation, giving larger noise reduction. However, as a result we fail to preserve the fine details and the quality of the picture. Both implementations improve the running time of the NL-means algorithm, at the cost of visual quality and accurate noise removal. Improvements on these implementations, and another intuitive implementaion that uses convolutions to express the NL-means algorithm [5] will be discussed in Section 5. .

3 Parameters

There are three parameters that can be discussed about from the above implementations of the NL-means algorithm; the size of the N x N comparison vectors , the size of the B x B research windows, and the filtering parameter h. The interesting thing is, that all of these parameters need to be adjusted for the value of σ (the standard deviation of the gaussian noise). As σ increases, the level of noise increases. Therefore, we need larger research windows - this increases the noise removal by finding more similar pixels. We also need larger comparison vectors for more accurate weights. And we also need to modify $h = k\omega$ for greater noise removal. The ideal values can be found in [2]. Figure 1 illustrates the effect and importance of modifying these parameters.

Figure 1: From top left to bottom right: noisy image, h = 3 N = 7 x 7 B = 21 x 21, h = 5 " ", h = 10 " ", very noisy image, h = 10 N = 11 x 11 B = 35 x 35, h = 13 " ", h = 15 " ", extremely noisy image, h = 25 N = 11 x 11 B = 35 x 35, h = 35 " ", h = 45 " "

4 Strengths and limitations

The Conditional expectation theorem outlined in [1] tells us that the NL-means algorithm corrects a noisy image v instead of seperating noise from the true image. Paper [1] uses this theorem and goes on to propose that the NL-means algorithm minimizes the mean square error (euclidean distance between the original and restored image) and is theoretically optimal in this respect. This suggests the NL-means algorithm performs optimally in restoring the original image, experimental mean square values can be found in [2].

For the rest of Section 4 we will compare the NL-means algorithm to the guassian filter. We will use a 7 x 7 comparison vector and a 21 x 21 research window. Giving us a complexity of 49 x 441 x N^2 for N^2 pixels.

Method noise works by evaluating how denoising algoritms affect a non noisy image; Let v be an image and D_h be a denoising operator that depends on some parameter h, then the method noise is defined as follows,

$$v - D_h v.$$

A denoising algorithm should not change a non noisy image, so the method noise should be small. If we visualise the method noise, a good performing algorithm should produce an image that just looks like noise with little to no structure. Figure 2 visualises method noise results and illustrates how the NL-means algorithm adapts well to the geometry of the image, and how the more naive denoising algorithm doesn't. Figure 3

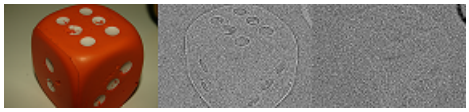

Figure 2: From left to right: original image, guassian smoothing filter, NL-means algorithm.

presents us with some some visual experiments. The original natural image has added guassian noise with *s.d.* σ. The NL means algorithm performs well in restoring and maintaining sharp lines or edges, but fails to preserve details and texture in flat areas. As a result the NL means struggles with objective and visual image quality, compared with some more advanced algorithms dicussed in [5] (that don't necessarily work in the spatial domain). Improvements for this will be discussed in Section 5. The guassian filter does struggle with maintaining lines and edges however.

Figure 3: From left to right: original image, noisy image, guassian smoothing filter, NL-means algorithm.

The other main issue with the NL-means algorithm is its computational complexity. The original description of the NL-means algorithm in Section 1 has quadratic running time which is bad in practice. The pixelwise and patchwise implementations in Section 2 do improve the computational complexity of the algorithm but fail to realise the original description. This means the previous statements about optimality do not necessarily hold for these implementations. In section 5 we will discuss other improvements that can speed up the NL-means algorithm.

Another less obvious limitation mentioned in [4], is that the weights are computed directly from the noisy image, and the noisy properties of the pixel neighbourhoods are not considered. The weights themselves are very sensitive to noise. Two pixels with similar neighbourhoods in the original image could have very different neighbourhoods in the noisy image. This issue will be discussed again in Section 5.

5 Modifications and extensions

5.1 Improving the NL-means algorithm

In this section we will consider extending the weights function for more robust pixel comparisons and improved visual quality of the restored image. Paper [3] briefly outlines some better performing weights functions that preserve sharp edges better, such as: the Turkey bi-weight function and bilateral filtering (which depends on euclidean distance and radial difference).

Paper [4] outlines many ways to improve the visual quality of the restored image. One solution is to add a local filtering post-proessing step; this method works by tracking the noise variance at every location and removing the remaining noise accordingly in the post-processing step, this helps to remove noise and preserve texture in flat areas of the image, which the original algorithm does poorly.

To deal with the noisy properties of the pixel neighbourhoods we could apply a rough denoising preprocessing step. However, we could fail to preserve details and texture with this method. In paper [4] we are given an alternative approach - applying the NL-means algorithm iteratively. This works because every iteration of the NL-means algorithm reduces the average noise variance, giving more accurate weights each iteration, and better denoising overall.

Paper [4] also goes on to explore the affect of using faster decaying weights functions, and hard or soft thresholds. The hard threshold weighting functions excludes pixels that are too dissimilar (giving them a weight of 0) whereas the soft threshold includes them. Experimentally it is seen that the modified Bisquare function in [4] (that uses a fast decaying weights function, and soft thresholding) improves the NL-means algorithm *w.r.t* noise reduction and image quality.

5.2 Improving the running time

The NL-means algorithm has a quadratic running time, which in practice is a problem. We can speed

up the NL-means algorithm by exploiting weight symmetry, that is, $w(i,j) = w(j,i)$; clearly we need to maintain a matrix of weights, but only for $j > i$, this speeds up our computation by a factor of 2.

We can also implement fast computation of euclidean distances; our weights function is a function of euclidean disatnces which can quickly be computed using a moving average filter applied to the squared difference between the signals, that is, $j = i + \Delta i$, so, $w(i,j) = \| y_i - y_{i+\Delta i} \|^2$. This brings our complexity down again to $O(N^2)$.

The final improvement is done by restricting our research window, in the same way we did in Section 2. We reduce the search window to a local neighbourhood of size B x B, and we set the weight of any pixels outside this window to 0. This signifigantly reduces our computation time and still gives good results if we apply it iteratively.

Other methods mentioned in [4] include neighbourhood pre-classification techniques, and FFT-based computation of the neighbourhood similarities. The implementation in [5] that uses convolutions also helps speed up the algorithm.

6 Applications

Digital imaging devices such as cameras produce noise because of the analogue nature of their circuitury, noise which we want to remove using some digital post processing technique. One soution is to take the same picture multiple times and average the observed values. This solution is not always practical if we are taking a picture of something that is moving. This gives rise to the class denoising algorithms.

The denoising of images its is very important for the robustness and consistency of other image processing procedures, such as image registration, image segmentation. These procedures have real world applications, such as face recognition, object identification, digital entertainment, remote sensing imaging and AI image processing. As a result denoising algorithms have become a critical part of image processing and enhancing. Paper [7] outlines the importance of denosing algorithms in the medical field. MRI images contain valuable information than can become corrupted by noise during the aquisition and transmission of the image. Noise removal methods are essential for recovering the fine details that may be lost. Image denoising methods can improve the visual quality of an image and are vital for further medical imaging processes.

Image desnoising a processing is also widely used police departments and other similar agencies for enhancing images from thermal imaging cameras or security footage. [8] explains how important image denoising is for detecting and tracking objects at sea, and how it is used in the automation of thermal imaging systems.

References

[1] Antoni Buades, Bartomeu Coll, and J.M. Morel. "A non-local algorithm for image denoising", *IEEE Computer Vision and Pattern Recognition 2005, Vol 2*, `http://dx.doi.org/10.1109/CVPR.2005.38`, 2005.

[2] Antoni Buades, Bartomeu Coll, and J.M. Morel. "Non-Local Means Denoising", *Image Processing On Line*, `http://dx.doi.org/10.5201/ipol.2011.bcm_nlm`, 2011. *hal-00512801v1*, 2010.

[3] Lingli Huang. "Improved Non-Local Means Algorithm for Image Denoising", *Journal of Computer and Communications, 2015, 3, 23-29*, `http://dx.doi.org/10.4236/jcc.2015.34003`, 2015.

[4] Bart Goosens, Hiep Luong, Aleksandra Pizurica, and Wilfred Philips "AN IMPROVED NON-LOCAL DENOISING ALGORITHM".

[5] Laurent Condat. "A Simple Trick to Speed Up the Non-Local Means", *hal-00512801v1*, 2010, 2011 (revisited).

[6] Ling Shao, Ruomei Yan, Xuelong Li and Yan Liu. "From Heuristic Optimization to Dictionary Learning: A Review and Comprehensive Comparison of Image Denoising Algorithms", *IEEE TRANSACTIONS ON CYBERNETICS, VOL 44, NO. 7, JULY 2014.*

[7] S. Satheesh, DR. KVSVR Prasad. "Medical Image Denoising using Adaptive Threshold Based on Contourlet Transform", *Advanced Computing: An International Journal (ACIJ), Vol.2, No.2, March 2011.*

[8] Dr Duncan Hickman, Tom Riley. "Thermal Imager Technical Guidance Document", *WS/10/CPNI/TI/002, 14th April 2010.*

Appendices

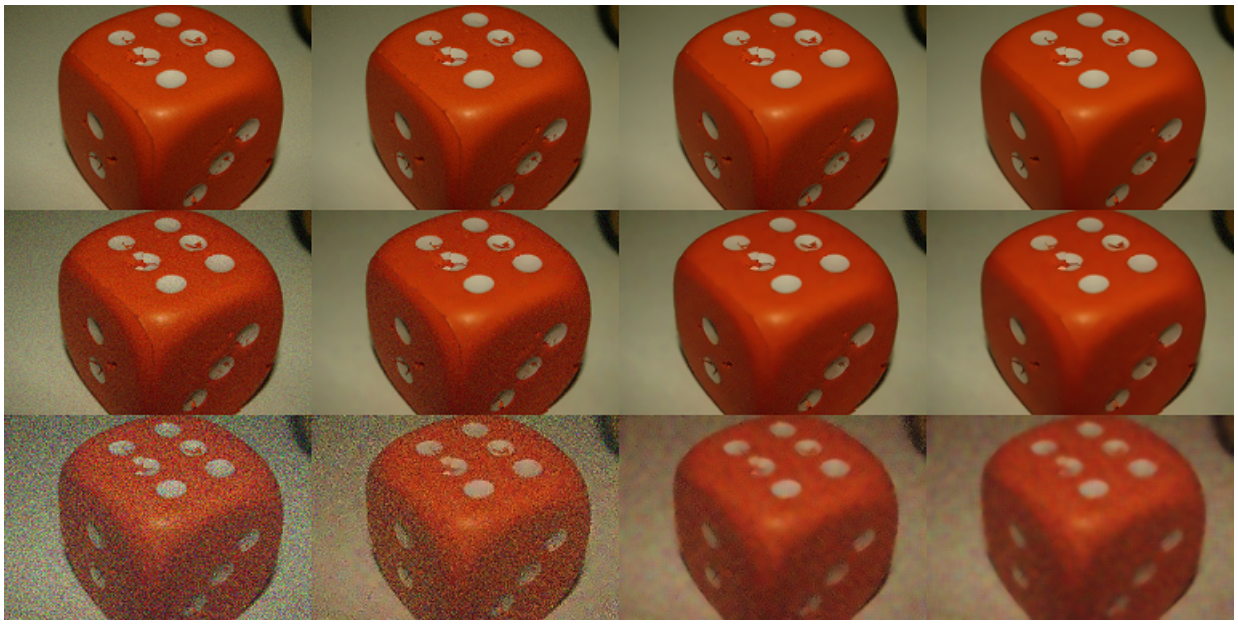

Figure 4: From top left to bottom right: noisy image, h = 3 N = 7 x 7 B = 21 x 21, h = 5, h = 10, very noisy image, h = 10 N = 11 x 11 B = 35 x 35, h = 13, h = 15, extremely noisy image, h = 25 N = 11 x 11 B = 35 x 35, h = 35, h = 45

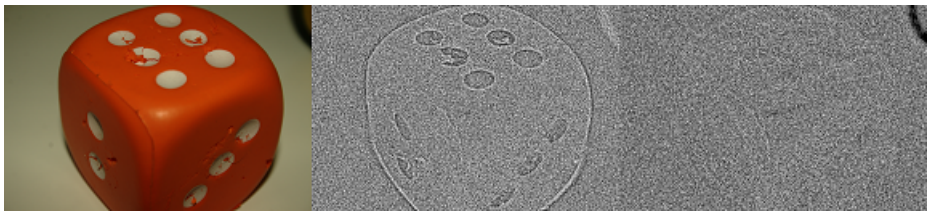

Figure 5: From left to right: original image, guassian smoothing filter, NL-means algorithm.

Figure 6: From left to right: original image, noisy image, guassian smoothing filter, NL-means algorithm.