数字图像处理第四次作业：图像去噪与边缘检测

自 64　　赵文亮　　2016011452

1　问题重述

图 1 为一张有噪图像，选择去噪方法去除其噪声，并对去噪后的图像进行边缘检测，分析结果的好坏。

图 1: 原始图像

2　图像去噪

2.1　原理介绍

2.1.1　简单低通滤波的局限

由于图 1 中的噪点过多，使用简单的低通滤波并不能得到很好的效果。例如，使用不同参数的高斯滤波器进行低通滤波的结果如图 2 所示。从中可见，当滤波窗口较小时，很多噪声没有被滤除；而当滤波窗口过大时，虽然噪声被滤除，但是图像也变得模糊。

使用滤波器滤波的局限在于，每一个像素点的滤波结果只和原始图像中该像素点的一个邻域（取决于滤波器大小）中的点有关，所以只考虑到了局部的信息。为了解决这个问题，我使用 Non-local means 算法 [1] 来实现去噪。

(a) $5 \times 5, \sigma = 1$ (b) $10 \times 10, \sigma = 2$ (c) $20 \times 20, \sigma = 4$

图 2: 不同参数高斯滤波器的滤波效果

2.1.2 Non-local means 算法

Non-local means 算法的基本思想是，每一个像素点去噪后的值与原始图像中的许多点的值有关，即

$$NL[v](i) = \sum_{j \in I} W(i,j)v(j)$$

其中 $v = \{v(x)|x \in I\}$ 表示待去噪图像，对于像素点 x，对应的去噪后图像的像素值为 $NL[v](x)$。而 $W(x,y)$ 则表示像素点 y 的权重，这个权重可以可以用像素点 ix 和 y 的相似度来衡量。记像素点 k 的一个方形邻域为 $\mathcal{N}_k$，则 $v(\mathcal{N}_x)$ 和 $v(\mathcal{N}_y)$ 可以

$$W(x,y) = \frac{1}{SW(x)} e^{-\frac{\|v(\mathcal{N}_x) - v(\mathcal{N}_y)\|^2}{h^2}}$$

其中 $SW(x)$ 表示点 x 与其他像素之间的权重和，用来归一化；h 为滤波参数。在实际应用中，往往不会再整个图片里搜索 y，而是在一个较大的搜索框内。

表 1: 符号说明

符号	含义
d	比较邻域 $\mathcal{N}$ 的边长
ds	$ds = (d-1)/2$
D	搜索框的边长
Ds	$Ds = (D-1)/2$
V	输入图像
Vd	输出图像
$Vsym$	边界填补后的图像
$N1, N2$	图像的行列数
W	权重
SW	权重和（用作归一化）

本节采用表 1 所示的符号说明，一种 Non-Loal means（NLM）算法如算法 1 所示。

算法 1: NLM 算法

input : V, ds, Ds, h

output: Vd

(N1, N2)← image size;

Vsym ← symmetrized noisy image V with border Ds+ds;

for $x = (x1, x2) = (1, 1)$ **to** $(N1, N2)$ **do**

  $\mathsf{Vd}(x) \leftarrow 0$;

  $SW(x) \leftarrow 0$;

  for $y = (y1, y2) = (x1 - Ds, x2 - Ds)$ **to** $(x1 + Ds, x2 + Ds)$ **do**

    $\mathsf{Dist2} \leftarrow 0$;

    for $z = (z1, z2) = (-ds, -ds)$ **to** (ds, ds) **do**

      $\mathsf{Dist2} \leftarrow \mathsf{Dist2} + (\mathsf{Vsym}(x + z) - \mathsf{Vsym}(y + z))^2$

    end

    $W(x, y) = e^{-\mathsf{Dist2}/h^2}$;

    $\mathsf{Vd}(x) \leftarrow \mathsf{Vd}(x) + W(x, y) * \mathsf{Vsym}(y)$;

  end

  $\mathsf{Vd}(x) \leftarrow \min(\max(\mathsf{Vd}(x)/SW(x), 0))$;

end

算法 2: 快速 NLM 算法

input : V, ds, Ds, h

output: Vd

Function *IntegralImage(Vsym, Ds, t);*

(N1, N2)← image size;

Vsym ← symmetrized noisy image V with border Ds+ds;

for $t = (t1, t2) = (-Ds, -Ds)$ **to** (Ds, Ds) **do**

  $\mathsf{Vd}(x) \leftarrow 0$;

  $SW(x) \leftarrow 0$;

  $II \leftarrow IntegralImage(\mathsf{Vsym}, Ds, t)$;

  for $x = (x1, x2) = (1, 1)$ **to** $(N1, N2)$ **do**

    $y \leftarrow x + t$;

    $\mathsf{Dist2} \leftarrow II(x + (ds, ds)) + II(x - (ds, ds)) - II(x + (ds, -ds)) - II(x + (-ds, ds))$;

    $\mathsf{Dist2} \leftarrow \mathsf{Dist2}/d^2$;

    $W(x, y) = e^{-\mathsf{Dist2}/h^2}$;

    $SW(x) \leftarrow SW(x) + W(x, y)$;

    $\mathsf{Vd}(x) \leftarrow \mathsf{Vd}(x) + W(x, y) * \mathsf{Vsym}(y)$;

  end

  $\mathsf{Vd}(x) \leftarrow \min(\max(\mathsf{Vd}(x)/SW(x), 0))$;

end

2.1.3 快速 NLM 算法

虽然算法 1 能够实现去噪，但是对每一个像素的遍历十分消耗时间。注意到每两个像素点灰度插值的平方会在多次循环中用到并求和，由此可以使用积分图像来简化求和 [2]，如算法 2 所示。该算法在具体实现时，利用 MaTLAB 的矩阵运算还可以进一步简化（见附录 A.1 ）。

2.2 去噪结果

使用快速 NLM 算法进行去噪，结果如图 3 所示。

(a) 去噪前　　(b) 去噪后

图 3: 图像去噪前后对比

可见去噪的效果非常好，既去除噪声又保证了图像的清晰。

3 边缘提取

边缘提取的基本原理是高通滤波器，但为了获得更好的效果，需要一些其他处理。于是我使用了 Canny[3] 的算法。

3.1 原理介绍

Canny 边缘检测算法可以分为以下几步：

1. 平滑滤波
2. 计算梯度
3. 非最大值抑制
4. 双阈值判断
5. 边缘跟踪

3.1.1 平滑滤波

使用高斯滤波器滤波，可以对图像做一定的平滑，从而减弱噪声的影响。

3.1.2 计算梯度

可以使用微分滤波器来计算图像的灰度梯度，此处我选择了 Sobel 算子，假设原始图像为 $\boldsymbol{I}$，则有

$$\boldsymbol{G}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \boldsymbol{I}, \quad \boldsymbol{G}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \boldsymbol{I}$$

则可以由此计算出梯度的大小和方向：

$$\begin{cases} \boldsymbol{G} = \sqrt{\boldsymbol{G}_x^2 + \boldsymbol{G}_y^2} \\ \boldsymbol{\Theta} = \arctan2(\boldsymbol{G}_y^2, \boldsymbol{G}_x^2) \end{cases}$$

之后将计算得到的角度离散到 $45°$ 的倍数的角度上。

3.1.3 非最大值抑制

由计算得到的离散化的梯度角度，比较某一个像素 p 沿着梯度方向的两个相邻像素点 p_1, p_2 处的梯度大小 G_1, G_2 与这个像素处的梯度大小 G，如果 $G < G_1$ 或 $G < G_2$，则将此处的梯度置为 0。

3.1.4 双阈值判断

双阈值判断需要事先设定高低阈值，分别记作 t_h, t_l。所有梯度大小低于 t_l 的点被认为是非边缘点，高于 t_h 的被认为是强边缘点，介于之间的为弱边缘点。

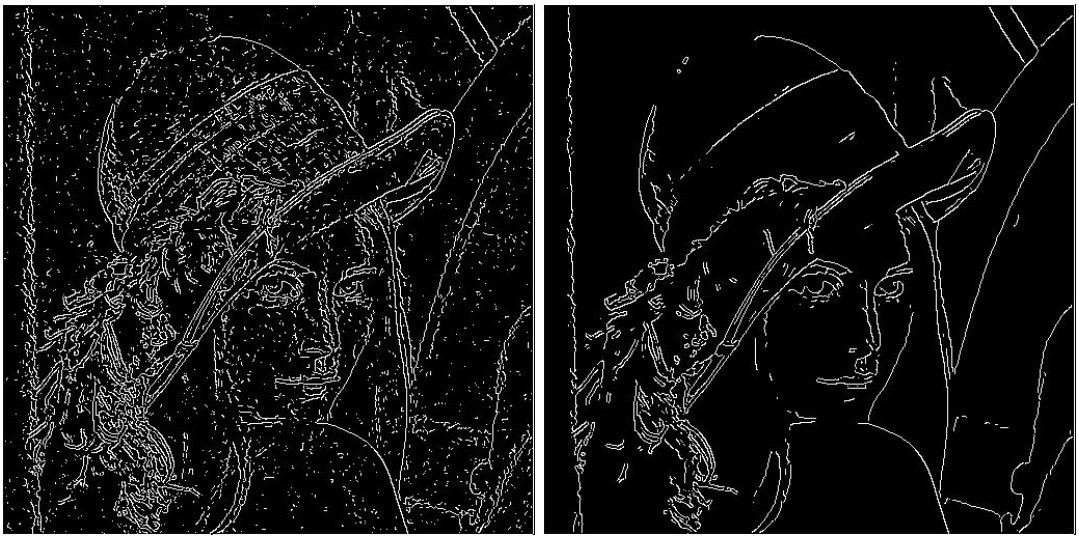
3.1.5 边缘跟踪

寻找所有的弱边缘点，如果该点的八个相邻像素中有强边缘点，则认为这个点是边缘点；否则则认为不是边缘点。至此，我们已经获得了所有的边缘点。

3.2 检测效果

Canny 边缘检测的代码见附录 A.2 。设定高斯滤波器的大小为 5×5，$\sigma = 1$。令双阈值为 $[0.15, 0.3]$，分别对原始图像和去噪后的图像进行边缘检测，结果如图 4 所示。从中可见，原始图像的检测结果中存在大量的噪点，而去噪图像的检测结果中几乎没有噪点，这也从另一个角度检验了去噪效果。

另一方面，Canny 算法可以通过控制双阈值大小有效滤除噪点。将阈值参数改为 $[0.2, 0.4]$，结果如图 5 所示。可见增大阈值后，即便是去噪前的图像，也能得到较好的边缘检测效果。



(a) 原始图像 (b) 去噪图像

图 4: 边缘检测结果 ($[t_l, t_h] = [0.1, 0.2]$)

(a) 原始图像 (b) 去噪图像

图 5: 边缘检测结果 ($[t_l, t_h] = [0.2, 0.4]$)

参考文献

[1] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 60–65, IEEE, 2005.

[2] J. Froment, "Parameter-free fast pixelwise non-local means denoising," *Image Processing On Line*, vol. 4, pp. 300–326, 2014.

[3] J. F. Canny, "Finding edges and lines in images," 1983.

A 代码

A.1 去噪

```
function Vd = fastNLMp(V, ds, Ds, h)
    % ds : half length of patch to compare
    % Ds : half size of search window
    % h : deviation of the gaussian
    if size(V, 3) == 3
        Vd = V;
        Vd(:, :, 1) = fastNLMp(V(:, :, 1), ds, Ds, h);
        Vd(:, :, 2) = fastNLMp(V(:, :, 2), ds, Ds, h);
        Vd(:, :, 3) = fastNLMp(V(:, :, 3), ds, Ds, h);
        return;
    end
    V = double(V);
    % padding
    Vsym = padarray(V, [Ds + ds, Ds + ds], 'symmetric');
    d = 2 * ds + 1; d2 = d * d;
    h2 = h * h;
    % sum weight, output image
    SW = 0; Vd = 0;
    % traverse the offset
    for t1 = -Ds : Ds
        for t2 = -Ds : Ds
            Vsym_y = Vsym(1 + Ds + ds + t1 : end - Ds - ds + t1, 1 + Ds + ds + t2 : end - Ds - ds + t2);
            II = IntegralImage(Vsym, Ds, t1, t2);
            Dist2 = II(1 : end - 2 * ds, 1 : end - 2 * ds) + II(1 + 2 * ds: end, 1 + 2 * ds : end) - ...
                II(1 : end - 2 * ds, 1 + 2 * ds : end) - II(1 + 2 * ds: end, 1 : end - 2 * ds);
            Dist2 = Dist2 / d2;
            W = exp(- Dist2 / h2);
            SW = SW + W;
            Vd = Vd + W .* Vsym_y;
        end
    end
    Vd = min(max(Vd ./ SW, 0), 255);
    Vd = uint8(Vd);
end

% get integral image
function II = IntegralImage(Vsym, Ds, t1, t2)
    Dist2 = (Vsym(1 + Ds : end - Ds, 1 + Ds : end - Ds) - ...
        Vsym(1 + Ds + t1 : end - Ds + t1, 1 + Ds + t2 : end - Ds + t2)) .^ 2;
    II = cumsum(Dist2, 1);
    II = cumsum(II, 2);
end
```

A.2 边缘检测

```
function output = edgeDetection(inputImg, s, sigma, threshold)
    % canny edge detection
    % Gaussian filter
    inputImg = double(inputImg) / 255;
    H = fspecial('gaussian', s, sigma);
    B = imfilter(inputImg, H);
    % finding the intensity gradient of the image
    Gx_H = fspecial('sobel')';
    Gy_H = fspecial('sobel');
    Gx = imfilter(B, Gx_H);
    Gy = imfilter(B, Gy_H);
    G = sqrt(Gx .^ 2 + Gy .^ 2);
    Theta = atan2(Gy, Gx);
    dir = angleDiscrete(Theta);

    dir = padarray(dir, [1, 1], nan);
    G = padarray(G, [1, 1], nan);
    tmp_G = zeros(size(G));
    % Non-maximum suppression
    for i = 2 : size(G, 1) - 1
        for j = 2 : size(G, 2) - 1
            switch dir(i, j)
                case 0
                    if G(i, j) < G(i, j + 1) || G(i, j) < G(i, j - 1)
                        tmp_G(i, j) = 0;
                    else
                        tmp_G(i, j) = G(i, j);
                    end
                case 1
                    if G(i, j) < G(i + 1, j + 1) || G(i, j) < G(i - 1, j - 1)
                        tmp_G(i, j) = 0;
                    else
                        tmp_G(i, j) = G(i, j);
                    end
                case 2
                    if G(i, j) < G(i + 1, j) || G(i, j) < G(i - 1, j)
                        tmp_G(i, j) = 0;
                    else
                        tmp_G(i, j) = G(i, j);
                    end
                case 3
                    if G(i, j) < G(i + 1, j - 1) || G(i, j) < G(i - 1, j + 1)
                        tmp_G(i, j) = 0;
                    else
                        tmp_G(i, j) = G(i, j);
                    end
            end
        end
    end
    G = tmp_G;
    % double threshold
    week = (G > threshold(1)) .* (G < threshold(2));
    strong = G > threshold(2);

    % edge tracking
    for i = 1 : size(G, 1)
```

```
57          for j = 1 : size(G, 2)
58              if week(i, j)
59                  if strong(i, j + 1) || strong(i, j - 1) || strong(i - 1, j) || strong(i + 1, j) || ...
60                          strong(i + 1, j + 1) || strong(i + 1, j - 1) || strong(i - 1, j - 1) || strong(i - 1, j + 1)
61                      G(i, j) = threshold(2);
62                  end
63              end
64          end
65      end
66      G = G(2 : size(G, 1) - 1, 2 : size(G, 2));
67      output = G > threshold(2);
68  end
69
70  % get discrete gradient direction
71  function direction = angleDiscrete(theta)
72      if theta < 0, theta = theta + 2 * pi; end
73      direction = mod(floor(4 * theta / pi), 4);
74  end
```

A.3 主函数

```
%% Read image
I = imread('lena.jpg');
%% non-local means denoise
J = fastNLM(I, 2, 10, 14);
figure();
imwrite(J, 'lena_denoise.jpg');
imshowpair(I, J, 'montage');
%% edge detection
edge1 = edgeDetection(I, 5, 1, [0.1, 0.2]);
edge2 = edgeDetection(J, 5, 1, [0.1, 0.2]);
figure();
imshowpair(edge1, edge2, 'montage');
```