数字图像处理 project

Project title:	Edge Detection Combined with Smoothing and Thresholding	Project number:	Project 1001
姓　　名:	彭思达	学　　号:	3140103545
学　　院:	信电学院	专　　业:	信息工程
Date due:	2017.6.30	Date handed in:	June 1, 2017

Abstract

这篇报告实现了 sobel 算子和 otsu 算法。

首先文章讨论了 sobel 算子，给出了它的数学表示，并使用代码实现了 sobel 算子。

随后文章又讨论了直方图的数学表示，并用代码实现了直方图算法。

最后文章又讨论了 otsu 算法，给出了它的数学表示，并用代码实现了 otsu 算法。

文章以书上 fig 2.35(c) 为实验对象，得到了它平滑过后的图像、梯度图像、直方图以及经阈值处理后的图像，并在文章的最后附上了相应的实现代码。

June 1, 2017

1 Technical discussion

1.1 对 sobel 算子的讨论

要得到一幅图像的梯度，则要求在图像的每个像素为止处计算偏导数 $\frac{\partial f}{\partial x}$ 和 $\frac{\partial f}{\partial y}$。我们考虑中心点对端数据的性质，用大小为 3x3 的模板来近似偏导数的最简单的数字近似由下式给出：

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3) \tag{1}$$

和

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7) \tag{2}$$

sobel 算子对这两个公式的一个小小的变化是在中心系数上使用一个权值 2:

$$g_x = \frac{\partial f}{\partial x} = (z_7 + \mathbf{2}z_8 + z_9) - (z_1 + \mathbf{2}z_2 + z_3) \tag{3}$$

和

$$g_y = \frac{\partial f}{\partial y} = (z_3 + \mathbf{2}z_6 + z_9) - (z_1 + \mathbf{2}z_4 + z_7) \tag{4}$$

在中心位置处使用 2 可以平滑图像。

1.2 对直方图的讨论

灰度级范围为 [0, L-1] 的数字图像的直方图是离散函数 $h(r_k) = n_k$，其中 r_k 是第 k 级灰度值，n_k 是图像中灰度为 r_k 的像素个数。在实践中，经常用乘积 MN 表示的图像像素的总数除它的每个分量来归一化直方图，通常 M 和 N 是图像的行和列的位数。归一化的直方图公式如下：

$$p(r_k) = \frac{n_k}{MN} \tag{5}$$

归一化直方图的所有分量之和应等于 1。

1.3 对 otsu 算法的讨论

otsu 方法在类间方差最大的情况下是最佳的，即众所周知的统计鉴别分析中所用的度量。基本概念是好阈值分类就它的像素灰度值而论应该是截然不同的。

otsu 方法有一个重要的特性，就是它完全以在一幅图像的直方图上执行计算为基础，直方图是很容易得到的一维阵列。

现在，假设我们选择一个阈值 $T(k) = k$，并使用它把输入图像阈值化处理为两类

C_1 和 C_2。用该阈值，像素被分到类 C_1 中的概率 $P_1(k)$ 由如下的累积和给出：

$$P_1(k) = \sum_{i=0}^{k} p_i \tag{6}$$

而分配到类 C_1 的像素的平均灰度值为

$$m_1(k) = \frac{1}{P_1(k)} \sum_{i=0}^{k} i p_i \tag{7}$$

类似的，像素被分到类 C_2 中的概率 $P_2(k)$ 由如下的累积和给出：

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i \tag{8}$$

而分配到类 C_2 的像素的平均灰度值为

$$m_2(k) = \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} i p_i \tag{9}$$

类间方差 σ_B^2 定义为

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 \tag{10}$$

为了寻找最佳阈值 k，我们对 k 的所有整数值求 σ_B^2，并选取使得 σ_B^2 最大的 k 值。

2　Discussion of results

sobel 算子公式如下：

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{11}$$

和

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{12}$$

直方图公式如下：

$$p(r_k) = \frac{n_k}{MN} \tag{13}$$

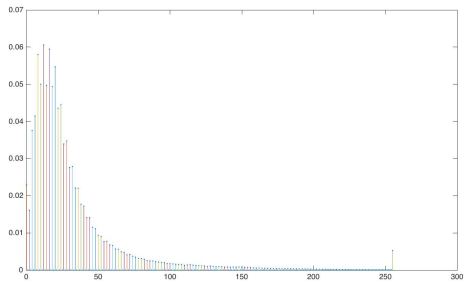
otsu 算法公式如下：

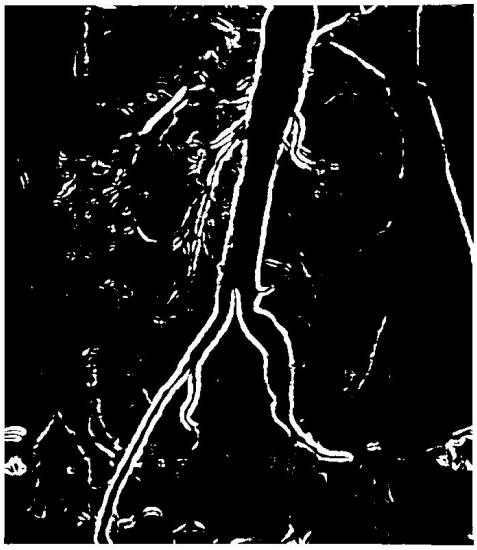
$$
\begin{aligned}
P_1(k) &= \sum_{i=0}^{k} p_i \\
m_1(k) &= \frac{1}{P_1(k)} \sum_{i=0}^{k} i p_i \\
P_2(k) &= \sum_{i=k+1}^{L-1} p_i \\
m_2(k) &= \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} i p_i \\
\sigma_B^2 &= P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2
\end{aligned}
\tag{14}
$$

求最大化 σ_B^2 的 k。

3　Results

下面四张图像，分别是平滑过后的图像、梯度图像、直方图以及经阈值处理后的图像。





4 Appendix

总的测试代码如下：

```
% test.m
img = imread('Fig0235(c)(kidney_original).tif');
% 平滑图像
img = smooth(3, img);
% 使用sobel算子计算梯度图像
result = sobel(img);
% 使用otsu算法计算最佳阈值
k = otsu(result)
% 阈值化图像
result = threshold(k, result);
imshow(result);
```

实现平滑操作的代码如下：

```
% smooth.m
function [ img ] = smooth( factor, img )

    img = double(img);
    % 填充图像
    img = fill((factor-1)/2, img);
    % 平滑图像
    img = mean(factor, img);
    img = uint8(img);

end

```

```
function [ result ] = fill( N, img )

    [m, n] = size(img);
    result = zeros(m+N*2, n+N*2);
    result(1+N:m+N, 1+N:n+N) = img;

end

function [ result ] = mean( factor, img )

    mask = ones(factor);
    mask = mask / (factor*factor);
    N = (factor-1) / 2;
    [m, n] = size(img);
    result = zeros(m-2, n-2);

    for i = 1+N:m-N
        for j = 1+N:n-N
            result(i-1, j-1) = spatial_filter(img(i-N:i+N, j-N:j+N), mask);
        end
    end

end

% 空间滤波器，输入相应的区域和掩模
function [ result ] = spatial_filter( region, mask )

    [m, n] = size(region);
    result = 0;

    for i = 1:m
        for j = 1:n
            result = result + region(i, j) * mask(i, j);
        end
    end

end
```

sobel 算子的实现代码如下：

```
% sobel.m
function [ result ] = sobel( img )

    img = double(img);
    % 填充图像
    img = fill(1, img);
    % 得到x方向的偏导数
    gx = get_gx(img);
    % 得到y方向的偏导数
    gy = get_gy(img);
    % 得到梯度图像
    result = abs(gx) + abs(gy);
    result = uint8(result);

end

```

```
17  function [ result ] = fill( N, img )
18
19      [m, n] = size(img);
20      result = zeros(m+N*2, n+N*2);
21      result(1+N:m+N, 1+N:n+N) = img;
22
23  end
24
25  function [ gx ] = get_gx( img )
26
27      mask = [-1, -2, -1;...
28              0,  0,  0;...
29              1,  2,  1];
30      [m, n] = size(img);
31      gx = zeros(m-2, n-2);
32
33      for i = 2:m-1
34          for j = 2:n-1
35              gx(i-1, j-1) = spatial_filter(img(i-1:i+1, j-1:j+1), mask);
36          end
37      end
38
39  end
40
41  function [ gy ] = get_gy( img )
42
43      mask = [-1,  0,  1;...
44              -2,  0,  2;...
45              -1,  0,  1];
46      [m, n] = size(img);
47      gy = zeros(m-2, n-2);
48
49      for i = 2:m-1
50          for j = 2:n-1
51              gy(i-1, j-1) = spatial_filter(img(i-1:i+1, j-1:j+1), mask);
52          end
53      end
54
55  end
56
57  function [ result ] = spatial_filter( region, mask )
58
59      [m, n] = size(region);
60      result = 0;
61
62      for i = 1:m
63          for j = 1:n
64              result = result + region(i, j) * mask(i, j);
65          end
66      end
67
68  end
```

otsu 算法实现代码如下：

```
% otsu.m
function [ threshold ] = otsu( img )

    img = double(img);
    % 得到直方图
    histProb = histogram(img);
    result = zeros(1, 256);

    for i = 1:256
        % 得到每个k对应的类间方差
        result(i) = getSigmaB(histProb, i-1);
    end

    [~, threshold] = max(result);
    % 得到使类间方差最大的阈值
    threshold = threshold - 1;

end

function [ histProb ] = histogram( img )

    [m, n] = size(img);
    numOfPixel = m * n * 1.0;
    histProb = zeros(1, 256);

    for i = 1:m
        for j = 1:n
            histProb(img(i, j) + 1) = histProb(img(i, j) + 1) + 1;
        end
    end

    histProb = histProb / numOfPixel;

end

function [ pK, m ] = getPM( histProb, from, to )

    if from == 256
        pK = 0;
        m = 0;
        return;
    end

    pK = sum(histProb(from+1:to+1));
    temp = 0.0;

    for i = from+1:to+1
        temp = temp + (i-1) * histProb(i);
    end

    m = temp / pK;

end

```

```
55 function [ sigma ] = getSigmaB( histProb, k )
56
57     [p1, m1] = getPM( histProb, 0, k);
58     [p2, m2] = getPM( histProb, k+1, 255);
59     sigma = p1 * p2 * ((m1 - m2)^2);
60
61 end
```

阈值化图像的代码如下：

```
1  % threshold
2  function [ result ] = threshold( k, img )
3
4      [m, n] = size(img);
5      result = zeros(m, n);
6      result = logical(result);
7      img = double(img);
8
9      for i = 1:m
10         for j = 1:n
11             % 大等于k的像素值为1，否则为0
12             result(i, j) = (img(i, j) > k) || (img(i, j) == k);
13         end
14     end
15
16 end
```