

计算机视觉与模式识别作业十三

姓名:	李智骋	学号:	2196113526
班级:	计算机95	得分:	

1、如下是一个简单的拉普拉斯算子对应的模板

	-1	
-1	4	-1
	-1	

而拉普拉斯算子又对应着函数

$$\Delta I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

尝试回答

(1)、 $\frac{\partial^2 I}{\partial x^2}$ 对应的算子是什么，提示根据 $\frac{\partial^2 I}{\partial x^2} = \frac{\partial}{\partial x} \otimes \frac{\partial}{\partial x} \otimes I$ 进行求解，其中 $\frac{\partial}{\partial x}$ 对应的模板是 $[-1, 1]$;

(1)、 $\frac{\partial^2 I}{\partial y^2}$ 对应的算子是什么，提示根据 $\frac{\partial^2 I}{\partial y^2} = \frac{\partial}{\partial y} \otimes \frac{\partial}{\partial y} \otimes I$ 进行求解，其中 $\frac{\partial}{\partial y}$ 对应的模板是 $[-1; 1]$;

1) 根据模板运算的结合律:

$$\begin{aligned}\Delta I &= \frac{\partial}{\partial x} \otimes \frac{\partial}{\partial x} \otimes I = \left(\frac{\partial}{\partial x} \otimes \frac{\partial}{\partial x} \right) \otimes I \\ \frac{\partial}{\partial x} \otimes \frac{\partial}{\partial x} &= \begin{bmatrix} -1 & 2 & -1 \end{bmatrix}\end{aligned}$$

因而 $\frac{\partial^2 I}{\partial x^2}$ 算子为 $[-1 \quad 2 \quad -1]$

2)

$$\begin{aligned}\Delta I &= \frac{\partial}{\partial y} \otimes \frac{\partial}{\partial y} \otimes I = \left(\frac{\partial}{\partial y} \otimes \frac{\partial}{\partial y} \right) \otimes I \\ \frac{\partial}{\partial y} \otimes \frac{\partial}{\partial y} &= \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}\end{aligned}$$

因而 $\frac{\partial^2 I}{\partial y^2}$ 算子为 $\begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}$

2、给定图像如下所示：

0	0	0	0	0	0	0	0	0
0	0	0	127	255	127	0	0	0
0	0	0	127	255	127	0	0	0
0	127	127	127	255	127	127	127	0
0	255	255	255	255	255	255	255	0
0	127	127	127	255	127	127	127	0
0	0	0	127	255	127	0	0	0
0	0	0	127	255	127	0	0	0
0	0	0	0	0	0	0	0	0

源图像

100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	100

目标图像

(1)、计算源图像的拉普拉斯图像

(2)、将源图像的梯度插入到目标图像的相应位置，试着计算目标图像。

1)

$$Laplace = \frac{\partial^2 I}{\partial y^2} + \frac{\partial^2 I}{\partial x^2}:$$

Laplace =

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

采用拉普拉斯算子对原图像 I_{source} 进行卷积运算

$$I_{source} \otimes Laplace =$$

$$\begin{array}{ccccccccc} 0 & 0 & 0 & -127 & -255 & -127 & 0 & 0 & 0 \\ 0 & 0 & -127 & 126 & 511 & 126 & -127 & 0 & 0 \\ 0 & -127 & -254 & -1 & 256 & -1 & -254 & -127 & 0 \\ -127 & 126 & -1 & -256 & 256 & -256 & -1 & 126 & -127 \\ -255 & 511 & 256 & 256 & 0 & 256 & 256 & 511 & -255 \\ -127 & 126 & -1 & -256 & 256 & -256 & -1 & 126 & -127 \\ 0 & -127 & -254 & -1 & 256 & -1 & -254 & -127 & 0 \\ 0 & 0 & -127 & 126 & 511 & 126 & -127 & 0 & 0 \\ 0 & 0 & 0 & -127 & -255 & -127 & 0 & 0 & 0 \end{array}$$

2)

目标图像被视为全为 100 的图像，考虑到其大小为 9*10 的矩阵，将其扩充为 11*11 的矩阵，边缘用 100 填充；将原图像的拉普拉斯结果也扩充为 11*11 的矩阵，边缘用 0 填充，便于计算；

由于直接利用梯度建立方程组会导致方程部分为欠定、部分为超定的，无法求解。因而转化为原图像梯度矩阵和所求图像梯度矩阵间差值的二范数最小（最小二乘法），即原图像拉普拉斯矩阵和所求图像拉普拉斯矩阵相等，边缘条件是所求图像部分区域仍为目标图像。

因而，该最小二乘法可以转化为 $\Delta f = b$ 的方程，又因为拉普拉斯算子是个线性算子，进一步转化为 $Ax = b$ 的线性方程。这里将 11*11 整个图像中的 121 个像素都作为 x 。

需要将源图像梯度插入的像素（9*9=81 个）根据拉普拉斯算子建立方程：

$$4x_{central} - x_{above} - x_{below} - x_{left} - x_{right} = b_{Laplace}$$

其中 x 代表拉普拉斯算子对应位置中源图像的值， $b_{Laplace}$ 代表中心像素点对应的源图像的拉普拉斯图像的值。

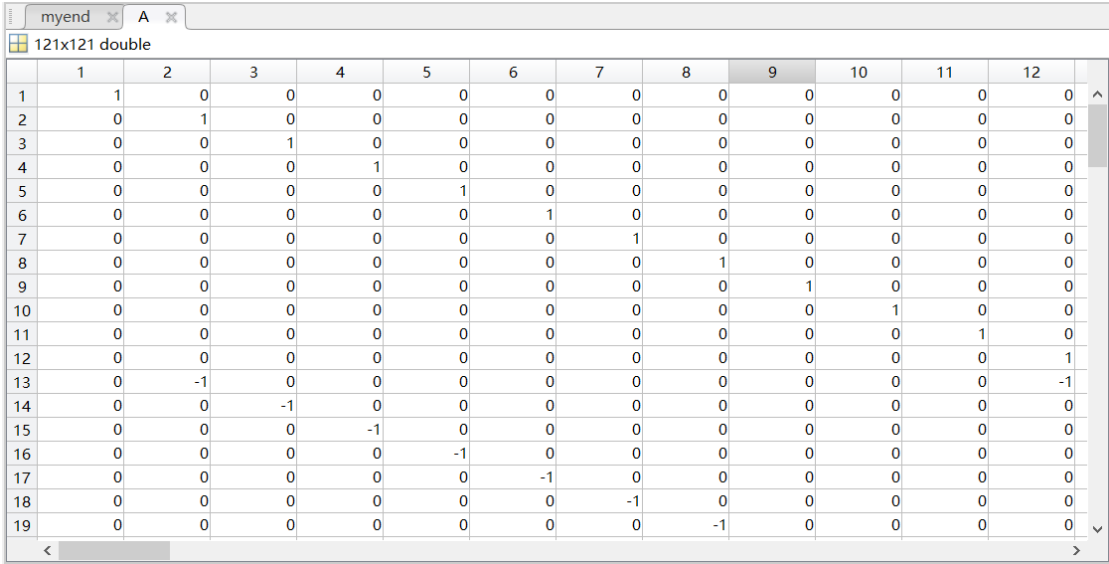
不需要将源图像梯度插入的像素为边缘条件：

$$x_{central} = b_{Target}$$

其中 b_{Target} 为目标图像相应位置的值。

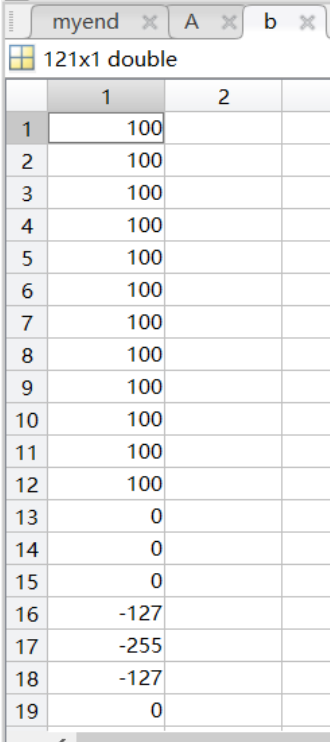
程序中采用遍历以上 121 个像素点而非 81 个像素点,对整个图像建立方程,这样减少了 b 的计算和特殊情况的考虑,将问题抽象出来。根据每个像素点上的上述两种方程记录 A 添加值的行号、列号、值,利用`sparse`函数建立稀疏矩阵 A ,减少了对边缘数据的考量。

由于图像的填充后为 11*11, A 矩阵因而大小为 121*121 的矩阵,部分如下:



	1	2	3	4	5	6	7	8	9	10	11	12	
1	1	0	0	0	0	0	0	0	0	0	0	0	
2	0	1	0	0	0	0	0	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	0	0	0	
4	0	0	0	1	0	0	0	0	0	0	0	0	
5	0	0	0	0	1	0	0	0	0	0	0	0	
6	0	0	0	0	0	1	0	0	0	0	0	0	
7	0	0	0	0	0	0	1	0	0	0	0	0	
8	0	0	0	0	0	0	0	1	0	0	0	0	
9	0	0	0	0	0	0	0	0	1	0	0	0	
10	0	0	0	0	0	0	0	0	0	1	0	0	
11	0	0	0	0	0	0	0	0	0	0	1	0	
12	0	0	0	0	0	0	0	0	0	0	0	1	
13	0	-1	0	0	0	0	0	0	0	0	0	0	-1
14	0	0	-1	0	0	0	0	0	0	0	0	0	0
15	0	0	0	-1	0	0	0	0	0	0	0	0	0
16	0	0	0	0	-1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	-1	0	0	0	0	0	0	0
18	0	0	0	0	0	0	-1	0	0	0	0	0	0
19	0	0	0	0	0	0	0	-1	0	0	0	0	0

b 向量因而大小为 121*1 的向量,部分如下:



	1	2
1	100	
2	100	
3	100	
4	100	
5	100	
6	100	
7	100	
8	100	
9	100	
10	100	
11	100	
12	100	
13	0	
14	0	
15	0	
16	-127	
17	-255	
18	-127	
19	0	

最后结果 x (121*1 的向量) 转化为大小为 9*10 的矩阵 (与目标图像大小一致) 如下:

100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000
100.0000	100.0000	100.0000	227.0000	355.0000	227.0000	100.0000	100.0000	100.0000	100.0000
100.0000	100.0000	100.0000	227.0000	355.0000	227.0000	100.0000	100.0000	100.0000	100.0000
100.0000	227.0000	227.0000	227.0000	355.0000	227.0000	227.0000	227.0000	100.0000	100.0000
100.0000	355.0000	355.0000	355.0000	355.0000	355.0000	355.0000	355.0000	100.0000	100.0000
100.0000	227.0000	227.0000	227.0000	355.0000	227.0000	227.0000	227.0000	100.0000	100.0000
100.0000	100.0000	100.0000	227.0000	355.0000	227.0000	100.0000	100.0000	100.0000	100.0000
100.0000	100.0000	100.0000	227.0000	355.0000	227.0000	100.0000	100.0000	100.0000	100.0000
100.0000	100.0000	100.0000	227.0000	355.0000	227.0000	100.0000	100.0000	100.0000	100.0000
100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000	100.0000

两小问的 Matlab 代码如下

第二小问中程序中采用遍历以上 121 个像素点而非 81 个像素点，这样减少了 b 的计算。根据每个像素点上的上述两种方程记录为 A 添加值的行号、列号、值，利用 *sparse* 函数建立稀疏矩阵 A ，减少了对边缘数据的考量：

```
clc;clear;
SP = zeros(9);
for i = 2 : 1 : 8
    SP(4, i) = 127;
    SP(i, 4) = 127;
    SP(6, i) = 127;
    SP(i, 6) = 127;
end
for i = 2 : 1 : 8
    SP(5, i) = 255;
    SP(i, 5) = 255;
end
Source = SP;
% disp("Source");disp(Source);

%% Laplace
Laplace = [0, -1, 0;
           -1, 4, -1;
           0, -1, 0];
conv = filter2(Laplace, Source, 'same');
% disp(conv);

%% Gradient Blending
% Target = 100 * ones(11,11);
% Source = padarray(Source, [1 1], 0); % 11 * 11
% conv = padarray(conv, [1 1], 0); % 11 * 11
% mask = zeros(11, 11);
% for i = 2 : 1 : 10
%     for j = 2 : 1 : 10
```

```

%         mask(i, j) = 1;
%     end
% end
Target = 100 * ones(9,9);
mask = zeros(9, 9);
for i = 1 : 1 : 9
    for j = 1 : 1 : 9
        mask(i, j) = 1;
    end
end
[result, A, b] = gradient_blend(Source, mask, Target);
A = full(A);
disp("A = ");disp(A);
disp("myend = ");disp(result);
% disp(myend(2:10,2:11,1))

function [result, A, b] = gradient_blend(source, mask,
target)
% padding
source = padarray(source, [1,1], 'symmetric');
target = padarray(target, [1,1], 'symmetric');
mask = padarray(mask, [1,1]);

% turning into a column vector
[t_rows, t_cols, ~] = size(target);
s = reshape(source, t_rows*t_cols, []);
t = reshape(target, t_rows*t_cols, []);

% Af = b
b = zeros(t_rows*t_cols, 1);
row_vec = zeros(t_rows*t_cols, 1); % row indexes of A
col_vec = zeros(t_rows*t_cols, 1); % column indexes of
A
value_vec = zeros(t_rows*t_cols, 1); % values of A
equation_num = 1; % the current row of the three above
% t_rows*t_cols <= equation_num <= 4*t_rows*t_cols

for index = 1:t_rows*t_cols
    if mask(index)
        % Laplace
        b(index,:) = 4*s(index,:) - s(index-1,:) -
s(index+1,:) - s(index+t_rows,:) - s(index-t_rows,:);

        % Insert a 4 into A at the index of the current

```

```

central pixel
    row_vec(equation_num) = index;
    col_vec(equation_num) = index;
    value_vec(equation_num) = 4;
    equation_num = equation_num + 1;

    % Insert a -1 for the pixel below
    row_vec(equation_num) = index;
    col_vec(equation_num) = index + 1;
    value_vec(equation_num) = -1;
    equation_num = equation_num + 1;

    % Insert a -1 for the pixel above
    row_vec(equation_num) = index;
    col_vec(equation_num) = index - 1;
    value_vec(equation_num) = -1;
    equation_num = equation_num + 1;

    % Insert a -1 for the pixel to the left
    row_vec(equation_num) = index;
    col_vec(equation_num) = index - t_rows;
    value_vec(equation_num) = -1;
    equation_num = equation_num + 1;

    % Insert a -1 for the pixel to the right
    row_vec(equation_num) = index;
    col_vec(equation_num) = index + t_rows;
    value_vec(equation_num) = -1;
    equation_num = equation_num + 1;
else
    % copy the target value
    row_vec(equation_num) = index;
    col_vec(equation_num) = index;
    value_vec(equation_num) = 1;
    equation_num = equation_num + 1;

    b(index,:) = t(index,:);
end
end
disp(equation_num)
% sparse matrix, the same index will add each other
A = sparse(row_vec, col_vec, value_vec, t_rows*t_cols,
t_rows*t_cols);

```

```
f = A \ b(:,1);  
f = reshape(f, [t_rows, t_cols]);  
result = f(2:t_rows-1, 2:t_cols-1, :);  
end
```