图像处理实验报告

学号：1120162362

作者：高祥君

时间：2018/12/13 11:16

空域实现拉普拉斯滤波和高斯滤波

一、实验原理

1. 拉普拉斯锐化滤波

拉普拉斯算子是二阶微分线性算子，与一阶微分相比，二阶微分的边缘定位能力更强，锐化效果更好，使用二阶微分算子的基本方法是定义一种二阶微分的离散形式，然后根据这个形式生成一个滤波模板，与图像卷积。

拉普拉斯算子是各向同性滤波器，即图像旋转后响应不变，这就要求滤波模板自身是对称的，如果不对称，结果就是，当原图旋转90°时，原图某一点能检测出细节（突变）的，现在却检测不出来，这就是各向异性的原因。实验中我们更关心的是各向同性滤波模板，对图像的旋转不敏感。

由于拉普拉斯算子是最简单的各向同性微分算子，它具有旋转不变性。一个二维图像函数的拉普拉斯变换是各向同性的二阶导数，定义为：

$$\mathrm{Laplace(f)} = \frac{\partial^2 \mathrm{f}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{f}}{\partial \mathrm{y}^2}$$

对于二维图像f(x,y)，二阶微分最简单的定义--拉普拉斯算子定义为：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

对于任意阶微分算子都是线性算子，所以二阶微分算子和后面的一阶微分算子都可以用生成模板然后卷积的方式得出结果。

根据前面对二阶微分的定义有：

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

根据上面的定义，与拉普拉斯算子的定义相结合，得到：

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

也就是一个点的拉普拉斯的算子计算结果是上下左右的灰度的和减去本身灰度的四倍。同样，可以根据二阶微分的不同定义，所有符号相反，也就是上式所有灰度值全加上负号，就是-1，-1，-1，-1，4。但要注意，符号改变，锐化的时候与原图的加或减应当相对变化。
最后的锐化公式：

$$g(x,y) = f(x,y) + c\left[\nabla^2 f(x,y)\right]$$

g是输出，f为原始图像，c是系数，中心值>0时，c=1，中心值<0，c=-1。

拉普拉斯算子还可以表示成模板的形式，以便更好编程需要。

0	1	0
1	-4	1
0	1	0

（a）拉普拉斯运算模板

1	1	1
1	-8	1
1	1	1

（b）拉普拉斯运算扩展模板

0	-1	0
-1	4	-1
0	-1	0

-1	1	-1
1	8	-1
-1	1	-1

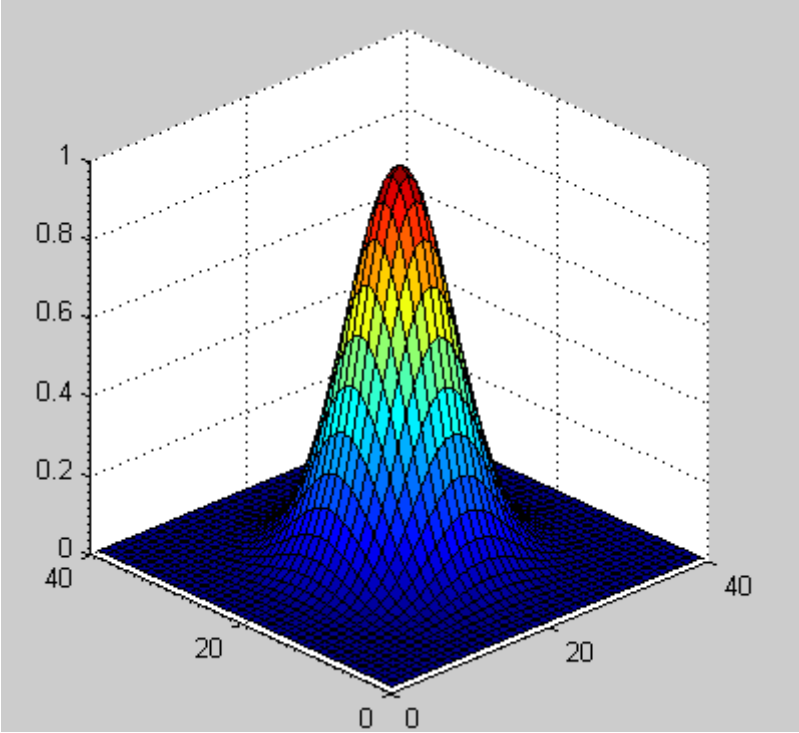
（c）拉普拉斯其他两种模板

2.高斯平滑滤波

1. 空域卷积定义：模板在图像中逐像素移动，将卷积核的每个元素分别和图像矩阵对应位置元素相乘并将结果累加，累加和作为模板中心对应像素点的卷积结果。通俗的讲，卷积就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到。
2. 高斯函数公式：

$$h(x,y) = \frac{1}{2\pi\sigma^2} exp(-\frac{x^2+y^2}{2\sigma^2})$$

形状：



3. 高斯模板的生成

$$G(x,y) = \frac{1}{2\pi\sigma^2}exp(-\frac{(x-m/2)^2+(y-n/2)^2}{2\sigma^2})$$

因为是离散取样，不能使取样和为1，最后还要做归一化操作。

二、实验步骤

1. 编码拉普拉斯算子

```
# 获得一个拉普拉斯锐化算子，此处均采用3*3的算子
# 参数： center：锐化算子中心的权重（4 或 8）
def get_laplas_kernel(center=4):
    laplas_kernel = np.zeros((3, 3), dtype=np.int32)
    d = {4: 0, 8: 1}
    laplas_kernel = np.array([
        [[0, -1, 0],
        [-1, 4, -1],
        [0, -1, 0]],
        [[-1, -1, -1],
        [-1, 8, -1],
        [-1, -1, -1]]
    ])
    return laplas_kernel[d[center]]
```

2. 编码高斯滤波算子

```python
# 获得一个高斯滤波算子，可以选择算子的大小以及方差sigma的取值
# 参数：     sigma：方差
#            kernel_size:算子大小
def get_gaussian_kernel(sigma=3, kernel_size=3):
    # 初始化高斯核全0
    guassian_kernel = np.zeros((kernel_size, kernel_size))

    # 计算像素点的平方和距离
    distance = lambda x, y: (x - int(kernel_size/2))**2 + \
        (y - int(kernel_size/2))**2

    # 计算高斯函数值
    gaussian_func = lambda x, y: np.exp(-distance(x, y) /
                                        (2*sigma*sigma)) / (2*3.1416*sigma*sigma)
    # i，j位置使用高斯核进行卷积
    total = 0.
    for i in range(kernel_size):
        for j in range(kernel_size):
            guassian_kernel[i][j] = gaussian_func(i, j)
            total += guassian_kernel[i][j]

    # 高斯核归一化
    guassian_kernel = guassian_kernel/total

    return guassian_kernel
```

3. 编写滤波函数

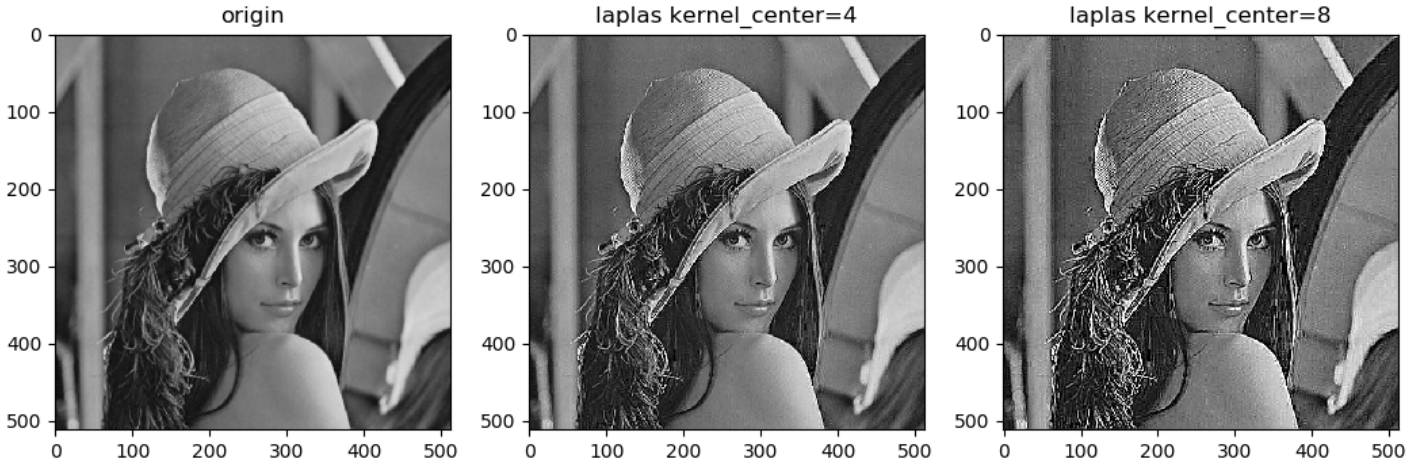
```python
# 利用滤波算子在图像上滑动进行滤波
# 参数：     img：图片数组
#            kernel：滤波算子
def filter(img, kernel):
    kernel_size = kernel.shape[0]
    img_height = img.shape[0]
    img_width = img.shape[1]
    # 初始化滤波后的图片全0
    filtered_img = np.zeros(img.shape, dtype=np.int32)

    # offset偏移量
    offset = int(kernel_size/2)
    # 滑动算子，计算卷积后的取值
    for i in range(offset, img_height-offset):
        for j in range(offset, img_width-offset):
            total = 0
            for m in range(kernel_size):
                for n in range(kernel_size):
                    total += kernel[m][n]*img[i-offset+m][j-offset+n]
            filtered_img[i][j] = total

    return filtered_img
```

4. 对lenna.jpg 分别进行滤波，输出原图像和滤波后的图像。
    1. 分别使用中心值为4 或者8 的3*3 拉普拉斯滤波器，滤波后和原图像进行对比。
    2. 分别使用sigma=1 size=3 ， sigma=1 size=5 , sigma=3 size=3 , sigma=3 size=5 的拉普拉斯滤波器，滤波后和原图像进行对比。

三、实验结果

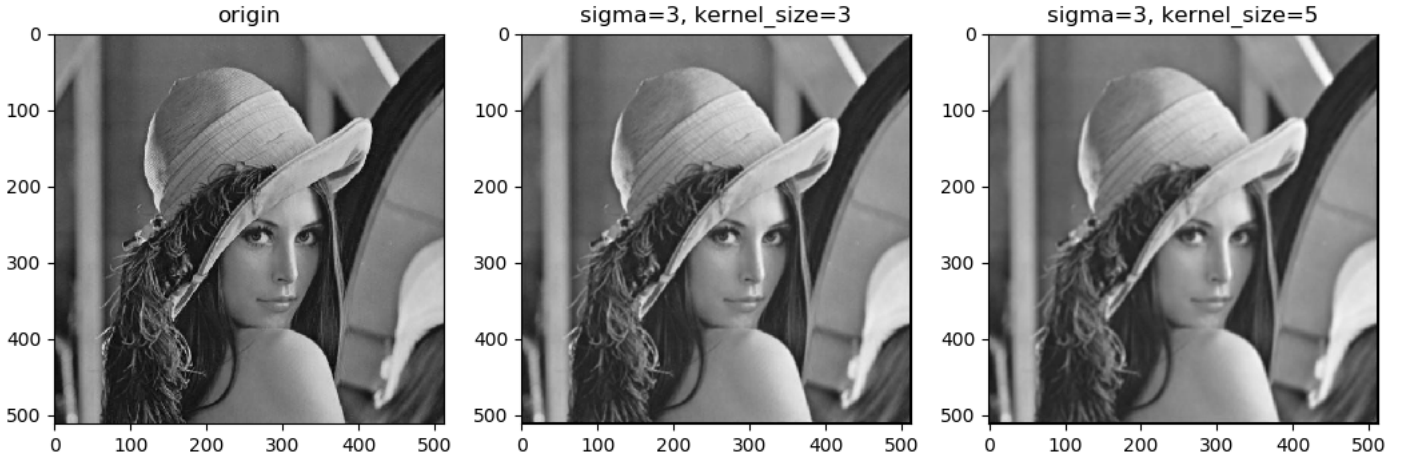
1. 拉普拉斯滤波

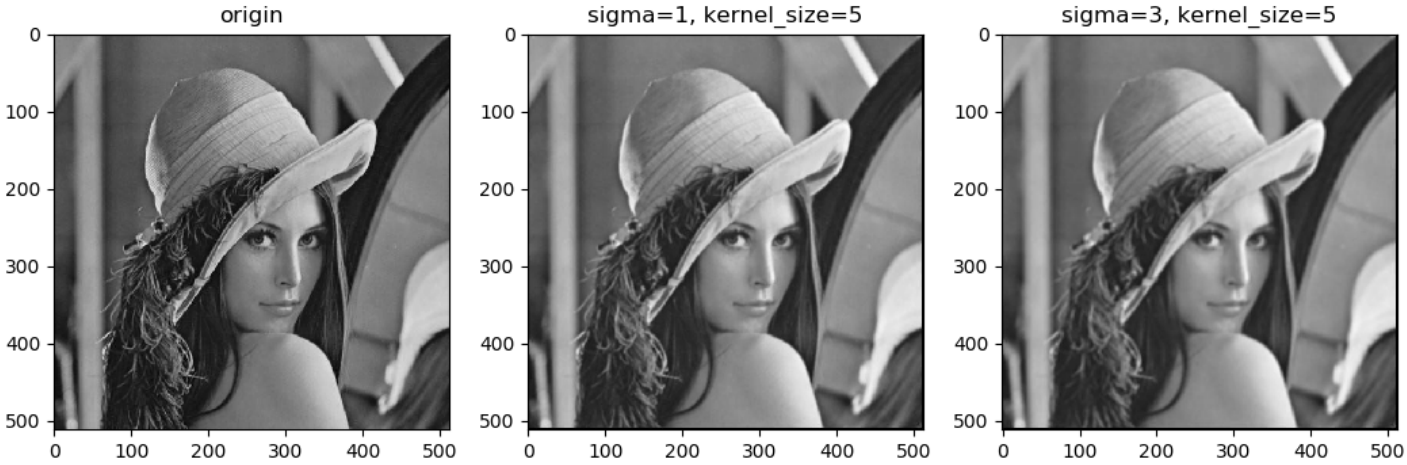
分别使用center=4 ， center=8 的3*3滤波器对lenna.jpg 进行滤波，实验对比图象如下：



2. 高斯滤波

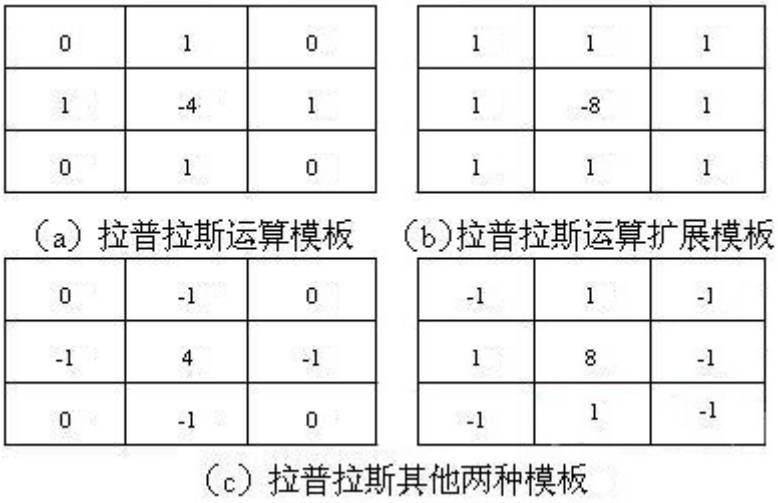
分别使用sigma=1 size=3， sigma=1 size=5，sigma=3 size=3，sigma=3 size=5 对lenna.jpg 进行滤波，实验对比图象如下：





四、实验结果分析

1. 拉普拉斯滤波



0	1	0
1	-4	1
0	1	0

（a）拉普拉斯运算模板

1	1	1
1	-8	1
1	1	1

(b)拉普拉斯运算扩展模板

0	-1	0
-1	4	-1
0	-1	0

-1	1	-1
1	8	-1
-1	1	-1

（c）拉普拉斯其他两种模板

图（a）表示离散拉普拉斯算子的模板，图（b）表示其扩展模板，图（c）则分别表示其他两种拉普拉斯的实现模板。从模板形式容易看出，如果在图像中一个较暗的区域中出现了一个亮点，那么用拉普拉斯运算就会使这个亮点变得更亮。因为图像中的边缘就是那些灰度发生跳变的区域，所以拉普拉斯锐化模板在边缘检测中很有用。

一般增强技术对于陡峭的边缘和缓慢变化的边缘很难确定其边缘线的位置。同梯度算子一样，拉普拉斯算子也会增强图像中的噪声，有时用拉普拉斯算子进行边缘检测时，可将图像先进行平滑处理。

由于拉普拉斯是一种微分算子，它的应用可增强图像中灰度突变的区域，减弱灰度的缓慢变化区域。因此，锐化处理可选择拉普拉斯算子对原图像进行处理，产生描述灰度突变的图像，再将拉普拉斯图像与原始图像叠加而产生锐化图像。

这种简单的锐化方法既可以产生拉普拉斯锐化处理的效果，同时又能保留背景信息，最终结果是在保留图像背景的前提下，突现出图像中小的细节信息。

根据对比可以发现， 图（b）**center = 8拉普拉斯算子产生了边缘更加清晰的锐化图像**，这也不难理解，灰度值差值由原来和上下左右四个点的差值和变为和周围八个点差值和，差值变化更大，边缘更加清晰。

实验中，滤波后与原始图像相加，可能会导致某一点的灰度值大于255，这时我想到归一化操作，将所有像素点的灰度值进行归一化，按照相应比例缩小到（0-255）范围内，但是这样会导致平滑部分的灰度值与原始图像相差过大，所以放弃归一化，直接对超过255的设置成255，低于0的设置成0。

2. 高斯滤波

根据实验结果的对比，得到以下结论：

1、空域高斯函数的方差越大，高斯函数越宽，处理的图像越模糊；

2、模板尺寸越大，处理的图像越模糊

3、计算量：空域高斯滤波的计算花费随着模板的规模的增大而增大

4、在所选择的高斯滤波器里面， `sigma=3 size=5` **获得最好的平滑效果**