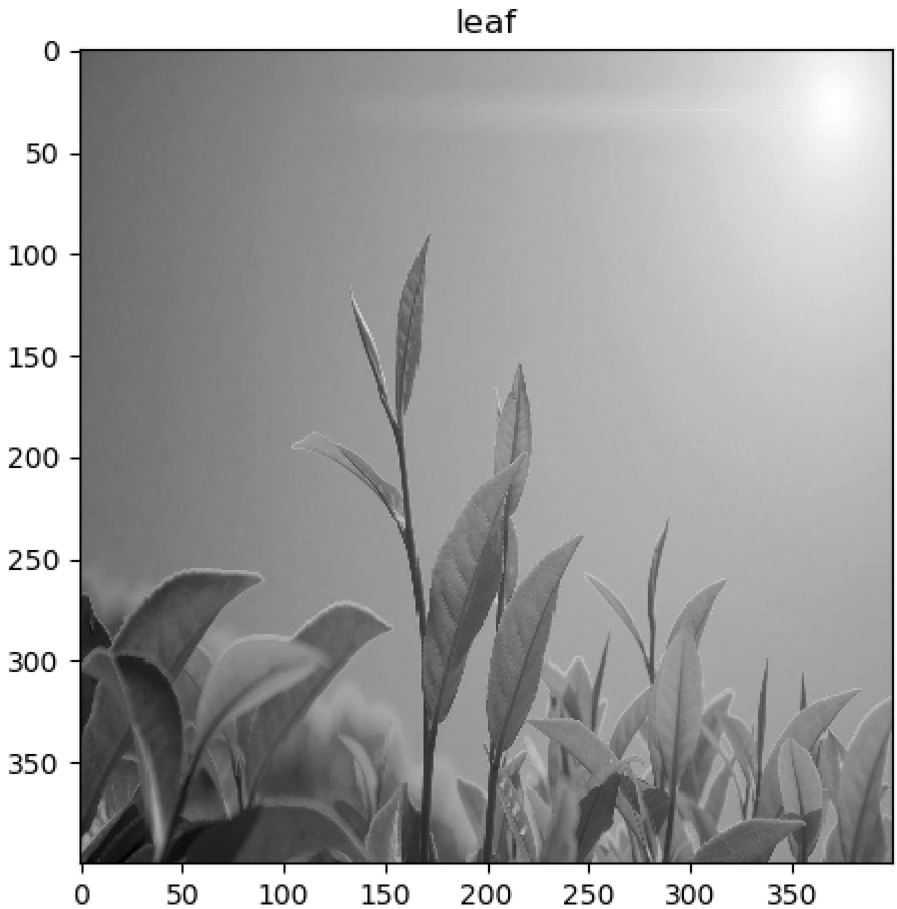
数字图像处理作业报告三

学号：71194506019 姓名：姜志刚 专业：计算机技术

题目

对一副图像加噪声，进行平滑，锐化作用。

待处理图像：



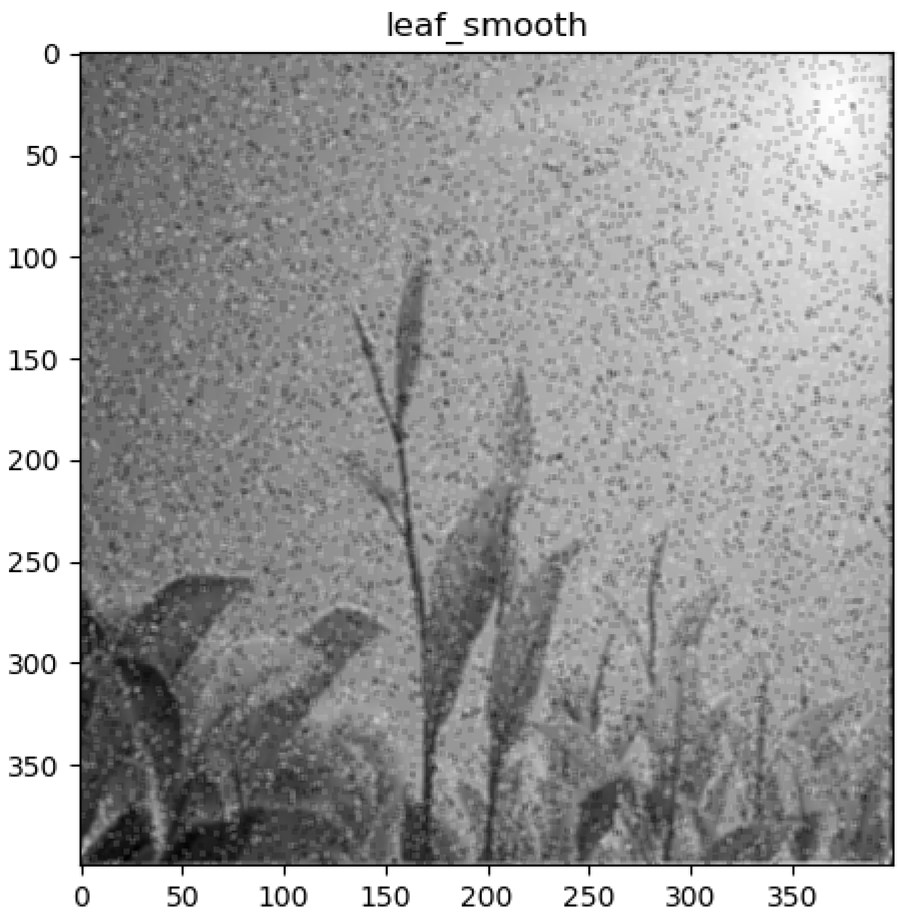
加噪

生成椒盐噪声：

```
def sp_noisy(image, s_vs_p=0.5, amount=0.08):
    out = np.copy(image)
    num_salt = np.ceil(amount * image.size * s_vs_p)
    coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image.shape]
    out[tuple(coords)] = 255
    num_pepper = np.ceil(amount * image.size * (1. - s_vs_p))
    coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in image.shape]
    out[tuple(coords)] = 0
    return out
```

结果

胡椒和盐各占0.5，总密度0.08的椒盐噪声：



平滑空间滤波（线性）

均值滤波

均值滤波过程：

$$g(x,y)=\frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b}w(s,t)f(x+s,y+t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b}w(s,t)}$$

$a=(m-1)/2$
$b=(n-1)/2$

m=n=3方形卷积模板：

```
kernel = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [1, 1, 1]], np.float32)/9
```

外围补0的线性滤波器：

```
def linear_filter(image, x, y, kernel, out):
    sum_wf = 0
    m = kernel.shape[0]
    n = kernel.shape[1]
    a = int((m - 1) / 2)
    b = int((n - 1) / 2)
    for s in range(-a, a + 1):
        for t in range(-b, b + 1):
            # convolution rotation 180
            x_s = (x - s) if (x - s) in range(0, image.shape[0] - 1) else 0
            y_t = (y - t) if (y - t) in range(0, image.shape[1] - 1) else 0
            sum_wf += kernel[a + s][b + t] * image[x_s][y_t]
    out[x][y] = sum_wf
```

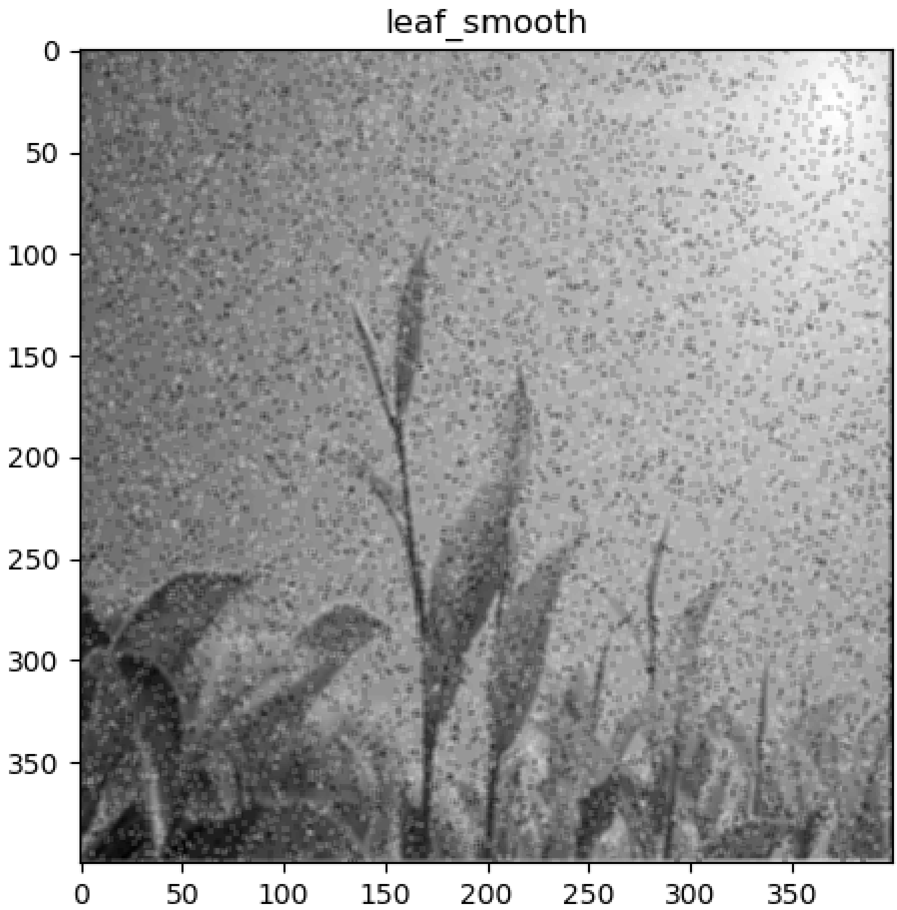
空间滤波函数实现：

```
def spatial_filtering(image, kernel, filter_):
    out = np.copy(image)
    h = image.shape[0]
    w = image.shape[1]
    for x in range(h):
        print(str(int(x/h * 100)) + "%")
        for y in range(w):
            filter_(image, x, y, kernel, out)
    return out
```

调用

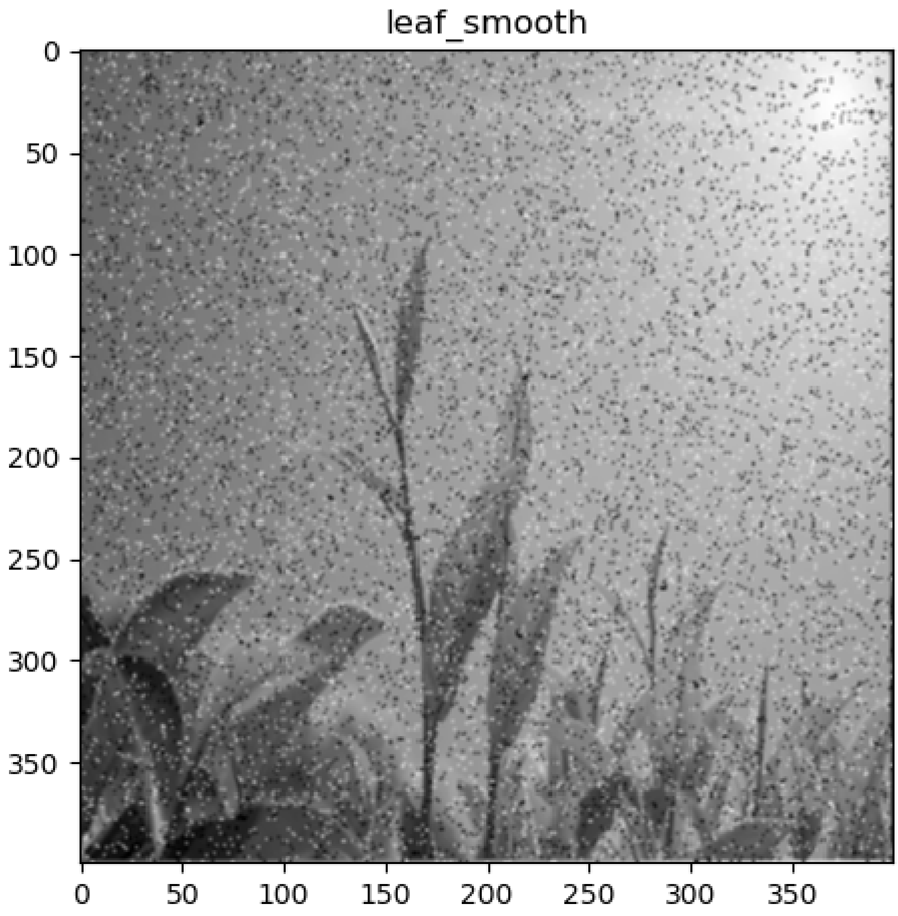
```
leaf_smooth = sp_convolution(leaf_sp_nose, k, linear_filter)
```

3 * 3均值滤波后：

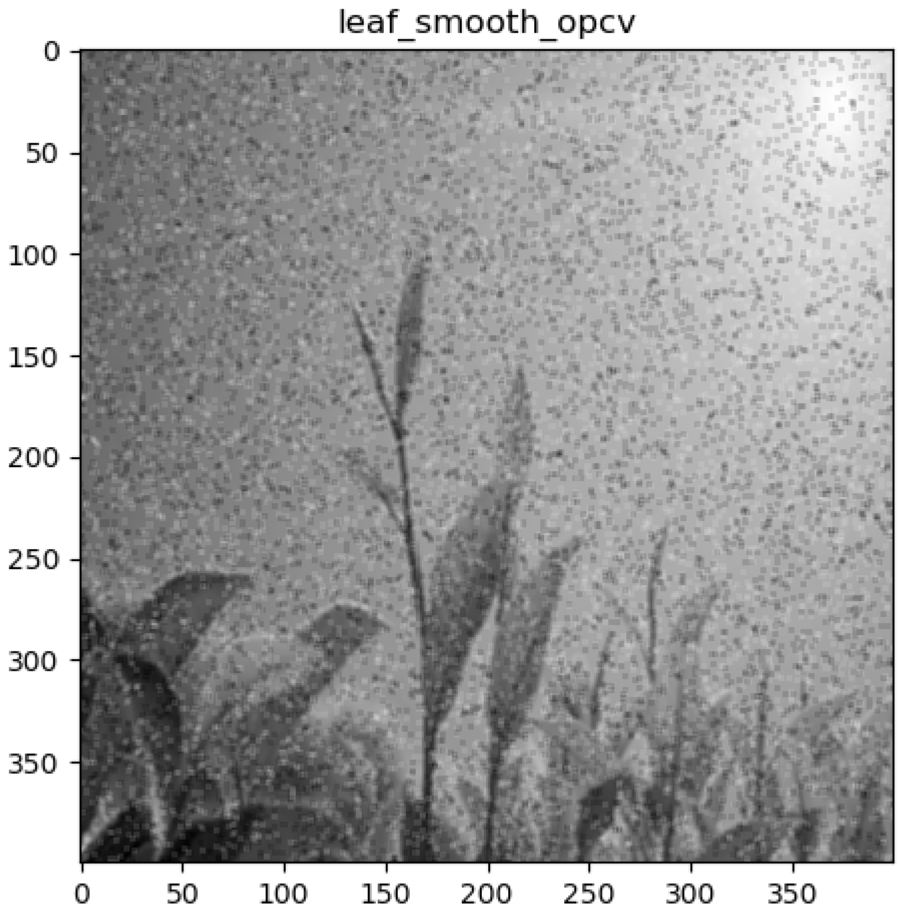


另一个3 * 3 的均值滤波模板结果：

```
kernel = np.array([[1, 2, 1],
                   [2, 4, 2],
                   [1, 2, 1]], np.float32)/16
```


leaf_smooth
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350

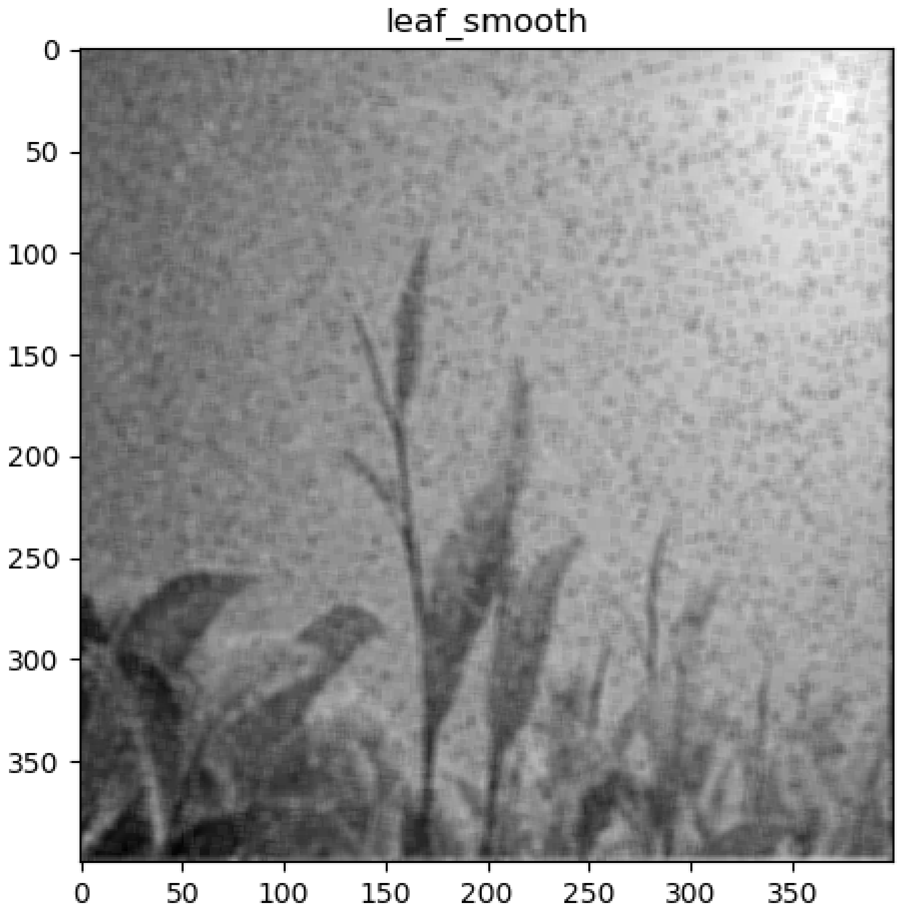
opencv 实现：


leaf_smooth_opcv
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350

opencv速度要快很多，最后的效果是一样

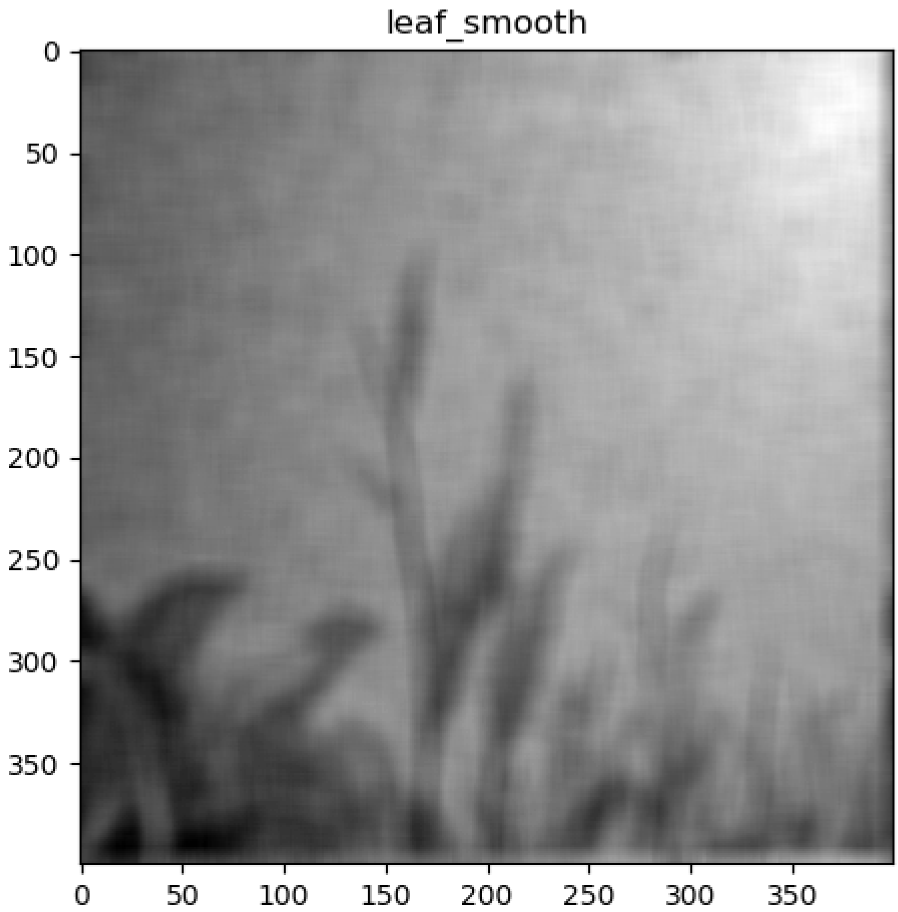
5 * 5均值滤波：

```
kernel = np.ones((5, 5), np.float32)/(5**2)
```



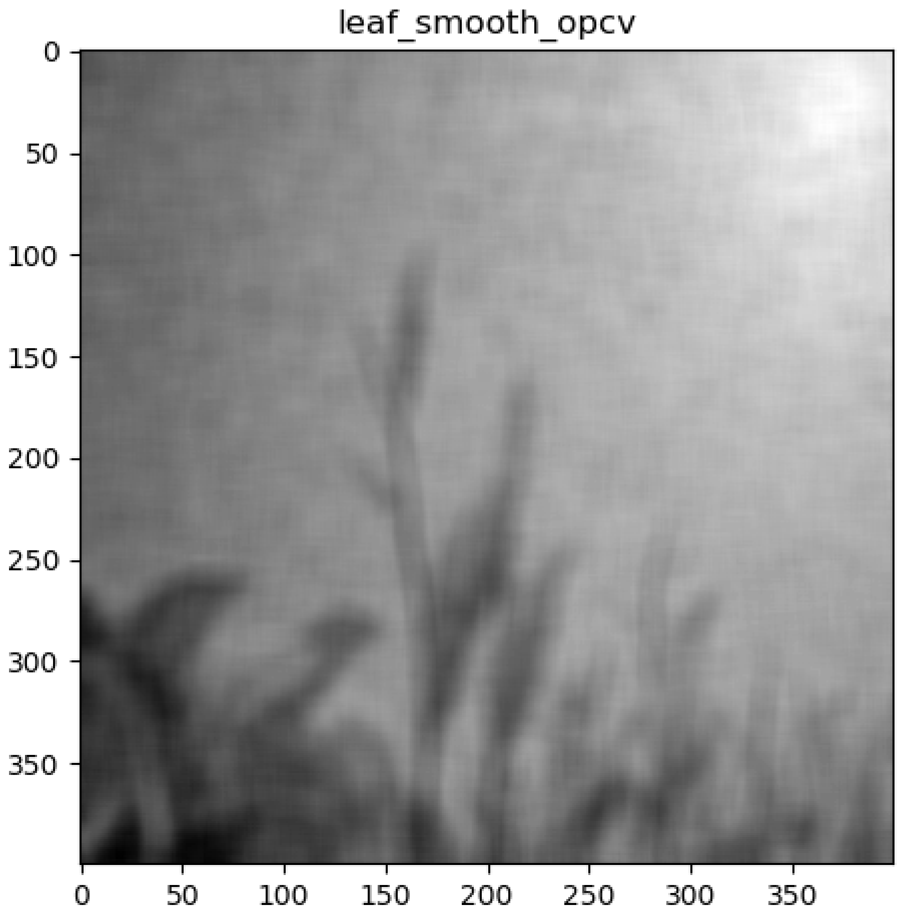
15 * 15均值滤波：

```
kernel = np.ones((15, 15), np.float32)/(15**2)
```



图像太过模糊，因为对外围取了0，可以明显看到周围有暗边

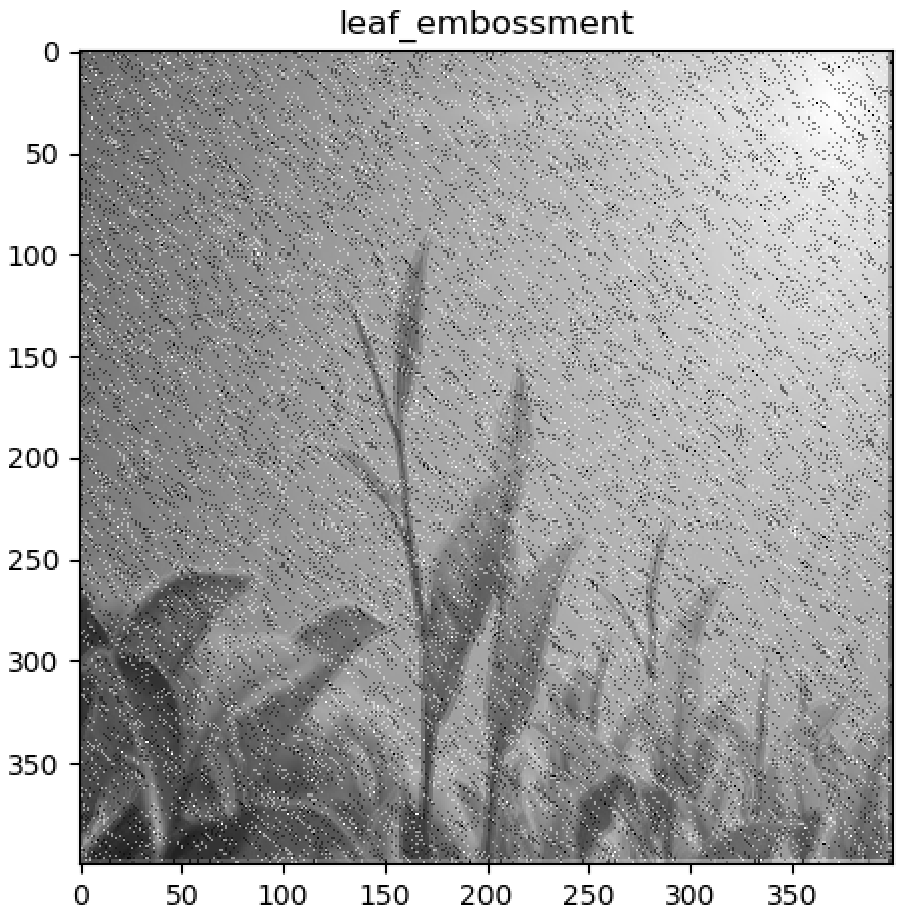
opencv：



opencv外围不是补0

Embossment算子

```
kernel = np.array([[2, 0, 0],
                   [0, 0, 0],
                   [0, 0, 2]], np.float32)/4
```



对去椒盐燥没什么效果

统计排序滤波（非线性）

中值滤波

过程为求领域内像素值的中值，窗口由kernel给出，置1为需要统计的像素
中值滤波器：

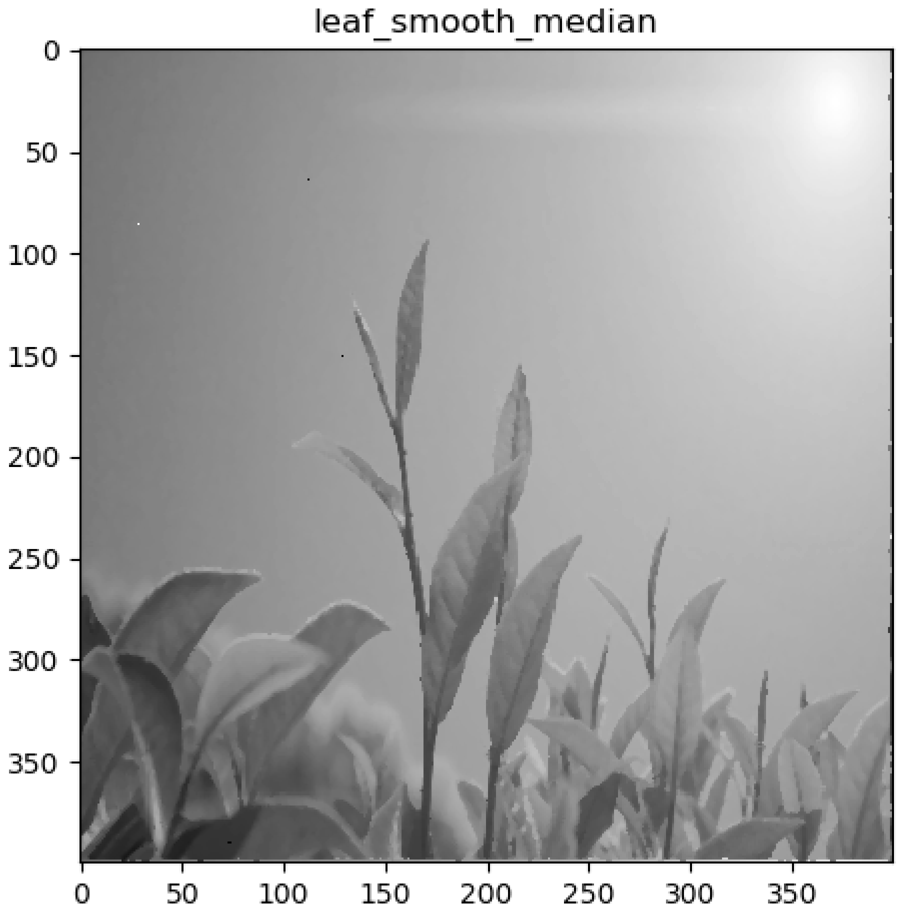
```
def nonlinear_median_filter(image, x, y, kernel, out):
    sp = []
    m = kernel.shape[0]
    n = kernel.shape[1]
    a = int((m - 1) / 2)
    b = int((n - 1) / 2)
    for s in range(-a, a + 1):
        for t in range(-b, b + 1):
            x_s = (x + s) if (x + s) in range(0, image.shape[0] - 1) else 0
            y_t = (y + t) if (y + t) in range(0, image.shape[1] - 1) else 0
            if kernel[a + s][b + t]:
```

```
                sp.append(image[x_s][y_t])
        out[x][y] = np.median(sp)
```

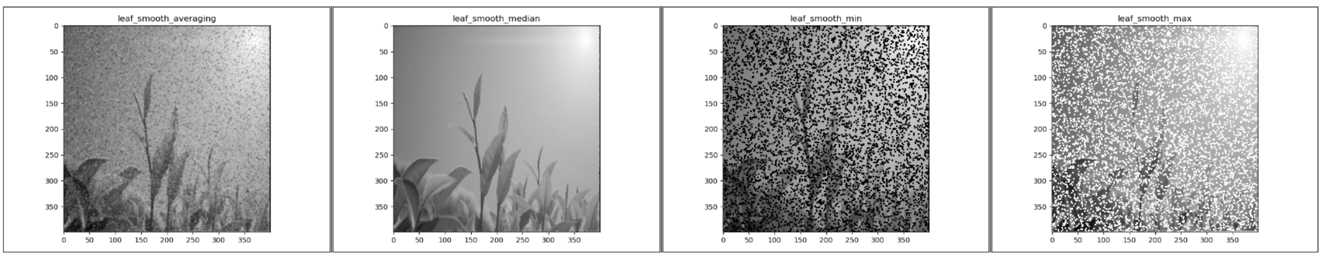
3*3中值滤波结果

模板：

```
k = np.ones((3, 3), np.float32)/(3**2)
```



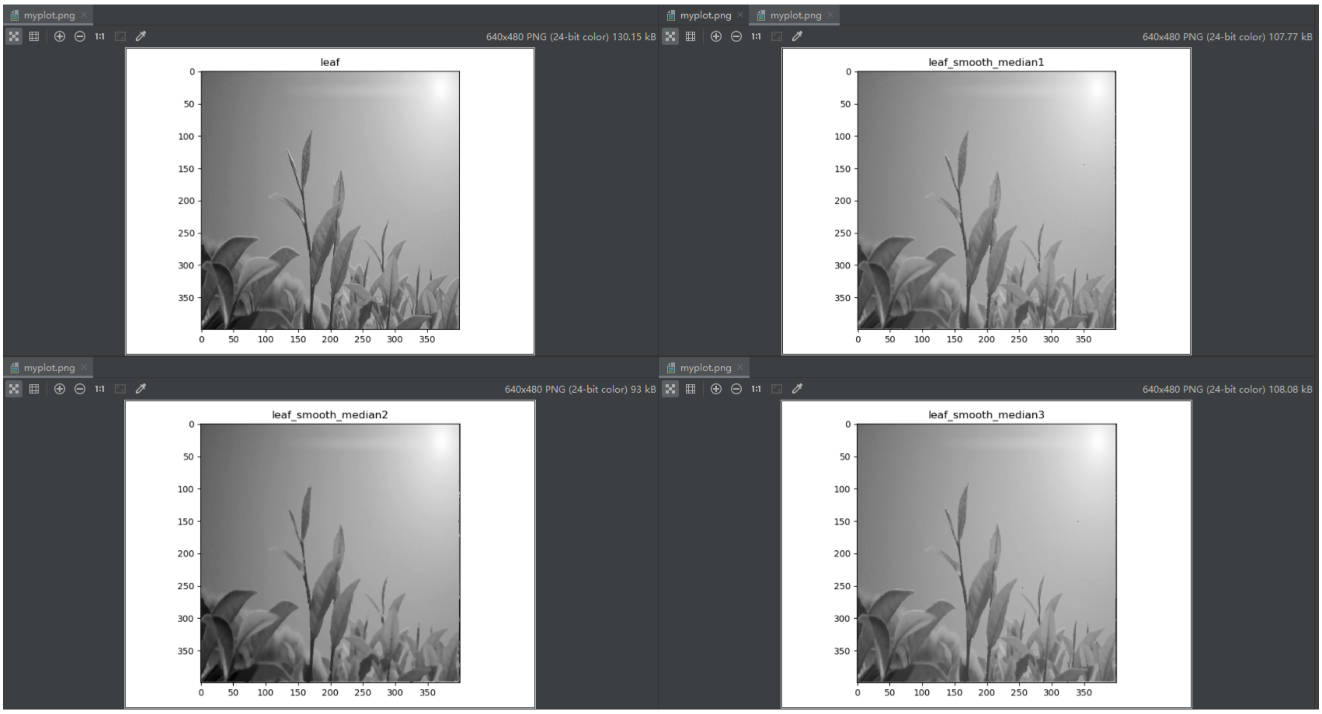
最大值最小值同理，下面是去取椒盐噪声对比图（均值，中值，最大值，最小值）：



可以看到中值效果最好，最大值和最小值不适用于去除椒盐噪声

不同模板对比

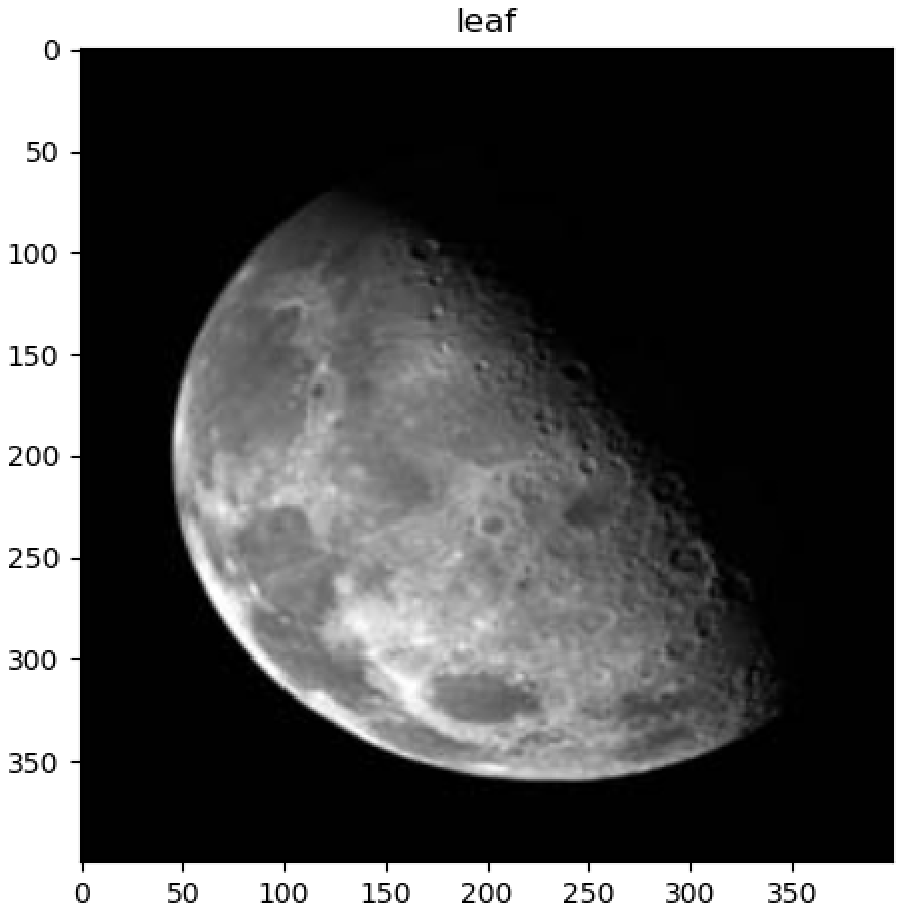
```
k1 = np.array([
    [1, 1, 1],
    [1, 1, 1],
    [1, 1, 1]
], np.float32)

k2 = np.ones((5, 5))

k3 = np.array([
    [0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0],
    [1, 1, 1, 1, 1],
    [0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0],
])
```



使用k2图像整体颜色偏暗，个人感觉k3效果最好

空间锐化滤波器

待处理图：



二阶微分-拉普拉斯算子（线性）

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

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

当$c=1$时

$$g(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 3f(x,y)$$

线性滤波实现函数基本和平滑的一样，但是锐化运算时会出现小于0或大于255的情况，所以需要对其处理
也是进行卷积运算：

```
def spatial_filtering(image, kernel, filter_):
    out = np.copy(image)
    h = image.shape[0]
    w = image.shape[1]
```

```python
    for x in range(h):
        # print(str(int(x/h * 100)) + "%")
        for y in range(w):
            filter_(image, x, y, kernel, out)
    return out


def linear_filter(image, x, y, kernel, out):
    sum_wf = 0
    m = kernel.shape[0]
    n = kernel.shape[1]
    a = int((m - 1) / 2)
    b = int((n - 1) / 2)
    for s in range(-a, a + 1):
        for t in range(-b, b + 1):
            # convolution rotation 180
            x_s = (x - s) if (x - s) in range(0, image.shape[0] - 1) else 0
            y_t = (y - t) if (y - t) in range(0, image.shape[1] - 1) else 0
            sum_wf += kernel[a + s][b + t] * image[x_s][y_t]
    if sum_wf < 0:
        sum_wf = 0
    if sum_wf > 255:
        sum_wf = 255
    out[x][y] = int(sum_wf)
```

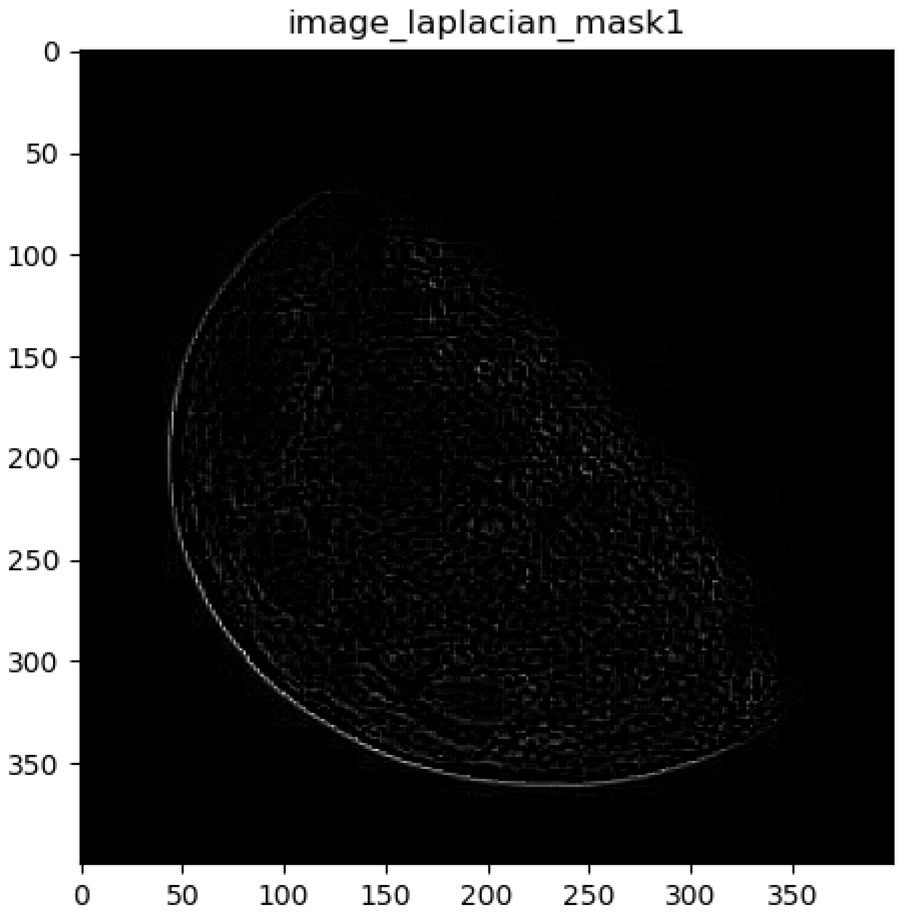
模板:

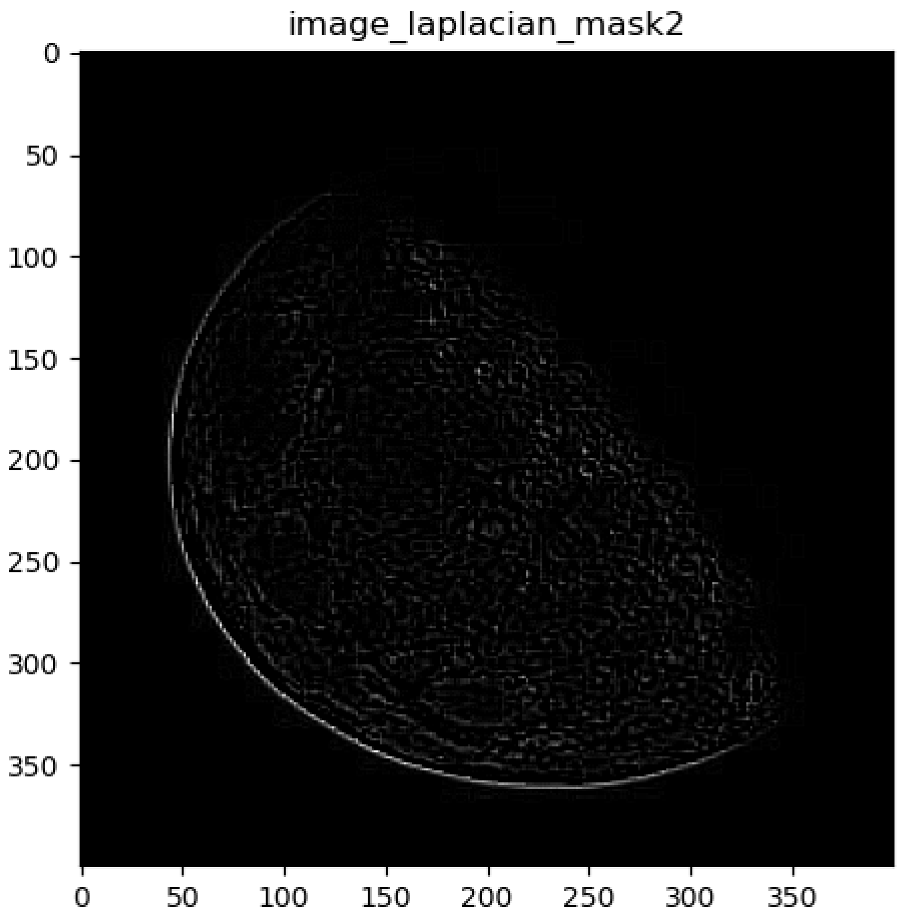
```python
laplacian_mask1 = np.array([
    [0,  1, 0],
    [1, -4, 1],
    [0,  1, 0],
])

laplacian_mask2 = np.array([
    [1,  1, 1],
    [1, -8, 1],
    [1,  1, 1],
])

laplacian_mask3 = np.array([
    [-1, -1, -1],
    [-1,  9, -1],
    [-1, -1, -1],
])
```

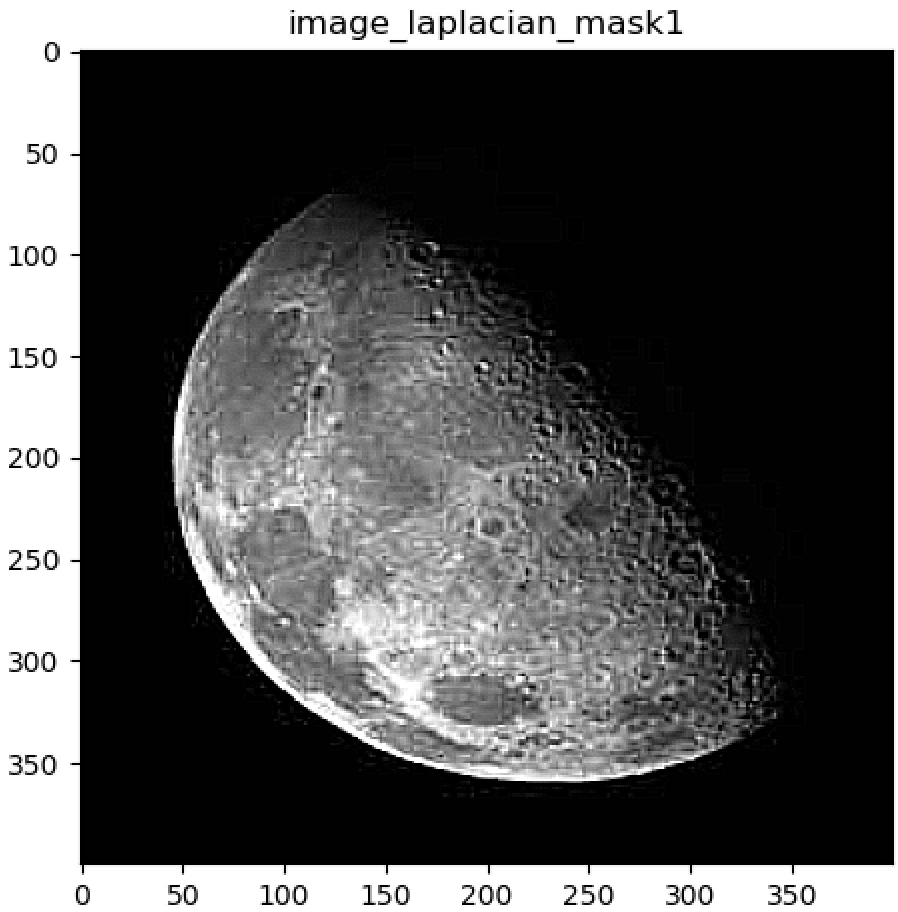
模板2考虑了对角项，模板3对原图像进锐化，由于是线性操作，直接调用线性滤波调用：

```
image_laplacian_mask_ = spatial_filtering(leaf, laplacian_mask_ , linear_filter)
```

结果


image_laplacian_mask1
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350


image_laplacian_mask2
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350



可以看到模板2的滤波效果要好与模板1，模板3实现了对原图像的锐化

一阶微分-梯度（非线性）

虽然是非线性的操作，但是求g_x, g_y是线性操作，因此可以分开求解，最后做非线性的操作，如求开方和绝对值：
简单起见，直接修改原来的线性滤波函数，改成求绝对值：

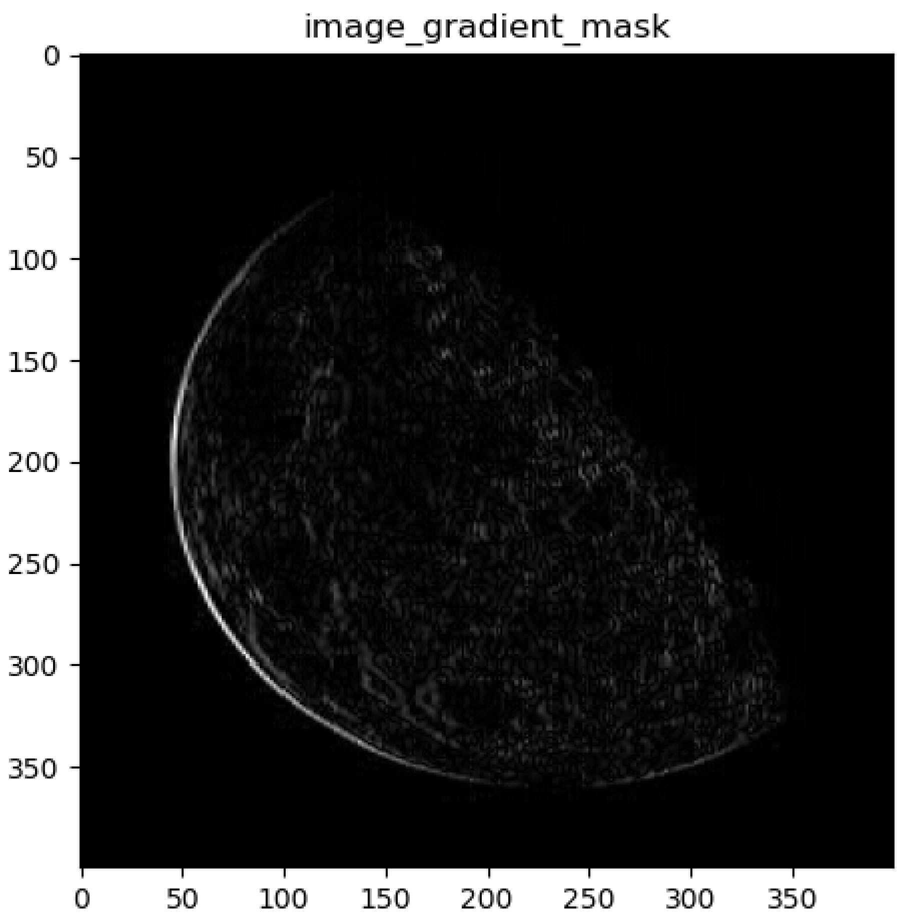
```
def linear_filter(image, x, y, kernel, out):
    sum_wf = 0
    m = kernel.shape[0]
    n = kernel.shape[1]
    a = int((m - 1) / 2)
    b = int((n - 1) / 2)
    for s in range(-a, a + 1):
        for t in range(-b, b + 1):
            # convolution rotation 180
            x_s = (x - s) if (x - s) in range(0, image.shape[0] - 1) else 0
            y_t = (y - t) if (y - t) in range(0, image.shape[1] - 1) else 0
            sum_wf += kernel[a + s][b + t] * image[x_s][y_t]

    sum_wf = abs(sum_wf)
    if sum_wf > 255:
```

```
        sum_wf = 255
    out[x][y] = int(sum_wf)
```

然后分别求$|g_x|, |g_y|$:

```
gradient_mask_1 = np.array([
    [0,  0,  0],
    [0, -1,  1],
    [0,  0,  0],
])
gradient_mask_2 = np.array([
    [0,  0,  0],
    [0, -1,  0],
    [0,  1,  0],
])
image_gradient_mask_1 = spatial_filtering(image, gradient_mask_1, linear_filter)
image_gradient_mask_2 = spatial_filtering(image, gradient_mask_1, linear_filter)
image_gradient_mask = image_gradient_mask_1 + image_gradient_mask_2
```

结果:

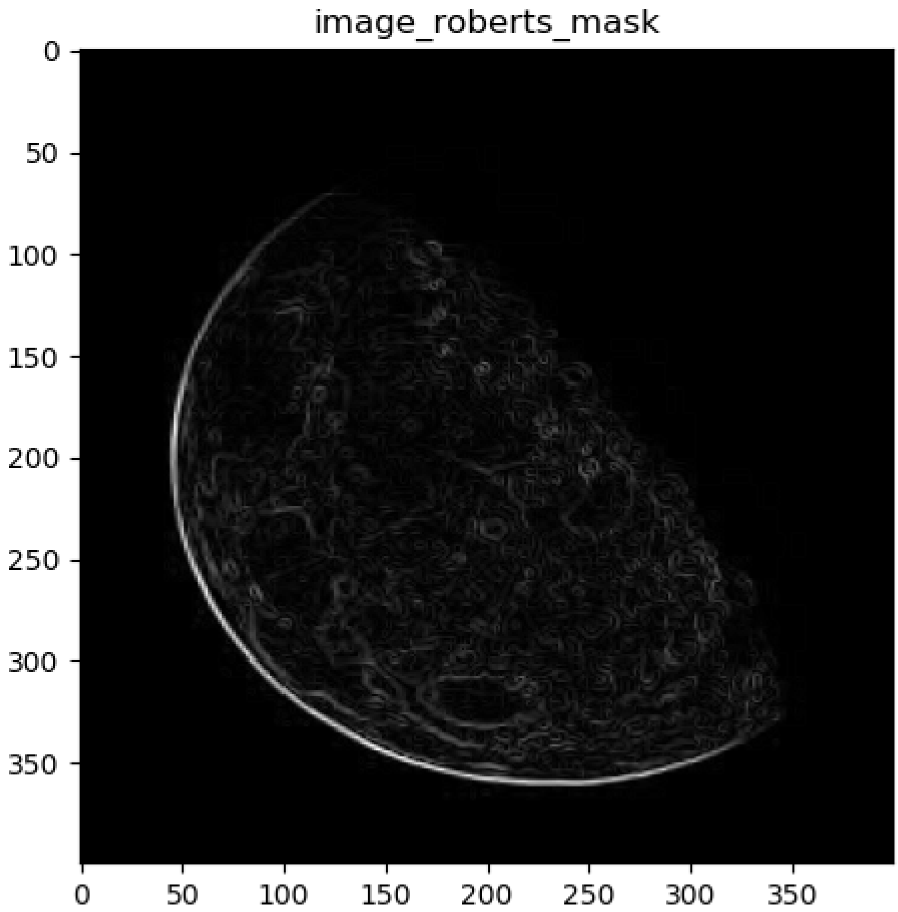


Roberts 算法 交叉差分

调用

```
roberts_mask_1 = np.array([
    [0,  0,  0],
    [0, -1,  0],
    [0,  0,  1],
])
roberts_mask_2 = np.array([
    [0,  0,  0],
    [0, 0,  -1],
    [0,  1,  0],
])

image_soble_mask_1 = spatial_filtering(image, gradient_mask_1, linear_filter)
image_soble_mask_2 = spatial_filtering(image, gradient_mask_1, linear_filter)
image_soble_mask = image_soble_mask_1 + image_soble_mask_2
```

结果

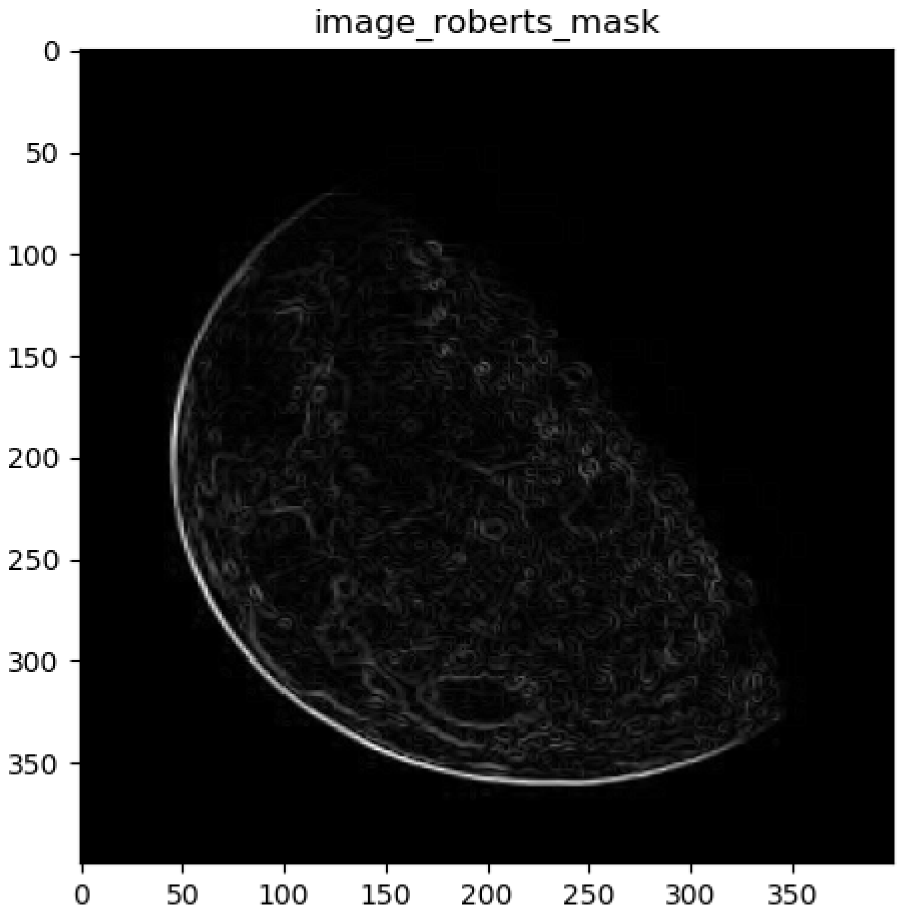


Soble算子

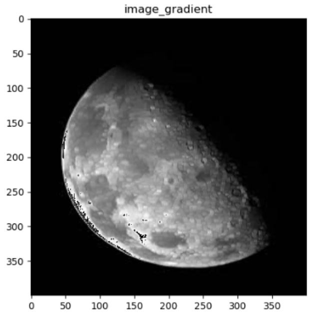
调用：

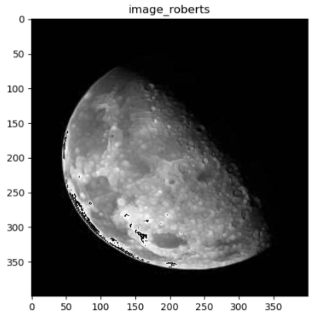
```
soble_mask_1 = np.array([
    [-1,  -2,  -1],
    [0,    0,   0],
    [1,    2,   1],
])
soble_mask_2 = np.array([
    [-1,   0,   1],
    [-2,   0,   2],
    [-1,   0,   1],
])

image_soble_mask_1 = spatial_filtering(image, soble_mask_1, linear_filter)
image_soble_mask_2 = spatial_filtering(image, soble_mask_2, linear_filter)
image_soble_mask = image_soble_mask_1 + image_soble_mask_2
```

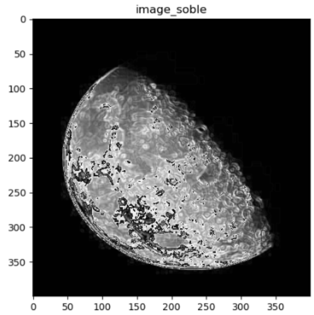
结果



最后都增加到原图中的效果：







可以看到：从梯度算子、Roberts 算子、Soble算子，效果依次增强