

Image Processing Homework2

3017207553 黃千慧

2019 年 11 月 6 日

图像复原的几种滤波器实现和对比以及常见的噪声模型实现。实现的滤波器主要有均值滤波器，分为算术均值滤波器、几何均值滤波器、谐波滤波器、逆谐波滤波器，统计滤波器分为中值滤波器、最大值和最小值滤波器、中点滤波器、修正后的阿尔法均值滤波器和自适应中值滤波器。噪声模型实现的有高斯噪声、瑞利噪声、均匀分布噪声、胡椒和盐噪声。下面是各种滤波器实现的代码：

```
1 class Filters:
2     def __init__(self, img):
3         self.img = img
4
5     def core(self, mode=None, **kwargs):
6         allowedtypes = {
7             'arithmetic_mean': 0,
8             'geometric_mean': 1,
9             'harmonic_wave_mean': 2,
10            'reverse_harmonic_wave_mean': 3,
11            'median_filter': 4,
12            'max_filter': 5,
13            'min_filter': 6,
14            'middle_filter': 7,
15            'revision_alpha': 8,
16            'adaptive_mean': 9}
17
18     kwdefaults = {
19         'm': 2,
20         'n': 2,
21         'q': -1.5,
22         'd': 2,
23         'max_size': 7,
24         'step': 2}
25
26     allowedkwargs = {
27         0: [ 'm', 'n'],
28         1: [ 'm', 'n'],
```

```

29     2: [ 'm' , 'n' ] ,
30     3: [ 'm' , 'n' , 'q' ] ,
31     4: [ 'm' , 'n' ] ,
32     5: [ 'm' , 'n' ] ,
33     6: [ 'm' , 'n' ] ,
34     7: [ 'm' , 'n' ] ,
35     8: [ 'm' , 'n' , 'd' ] ,
36     9: [ 'm' , 'n' , 'max_size' , 'step' ] }

37
38     for key in kwargs:
39         if key not in allowedkwargs[allowedtypes[mode]]:
40             raise ValueError( "%s keyword not in allowed
41                               keywords %s" %
42                               (key, allowedkwargs[allowedtypes[
43                               mode]]))
44
45     # Set kwarg defaults
46     for kw in allowedkwargs[allowedtypes[mode]]:
47         kwargs.setdefault(kw, kwdefaults[kw])
48
49     image = self.img.copy()
50     h, w, c = image.shape
51     for i in range(h):
52         for j in range(w):
53             for k in range(c):
54                 m, n = kwargs['m'], kwargs['n']
55                 temp = image[i:i + m, j:j + n, k]
56                 if allowedtypes[mode] == 0:
57                     image[i][j][k] = np.mean(temp)
58                 elif allowedtypes[mode] == 1:
59                     image[i][j][k] = np.prod(np.power(temp, 1
60                                         / (m * n)))
61                 elif allowedtypes[mode] == 2:
62                     image[i][j][k] = (m * n) / (np.sum(1 /
63                                         temp))
64                 elif allowedtypes[mode] == 3:
65                     image[i][j][k] = np.sum(temp ** (kwargs[ 'q' ]
66                                         + 1)) / np.sum(temp ** kwargs[ 'q' ]
67                                         ))
68                 elif allowedtypes[mode] == 4:
69                     image[i][j][k] = np.median(temp)

```

```

65         elif allowedtypes[mode] == 5:
66             image[i][j][k] = np.max(temp)
67         elif allowedtypes[mode] == 6:
68             image[i][j][k] = np.min(temp)
69         elif allowedtypes[mode] == 7:
70             image[i][j][k] = (np.min(temp) + np.max(
71                 temp)) / 2
72         elif allowedtypes[mode] == 8:
73             if kwargs['d'] < 0 or kwargs['d'] > m * n -
74                 - 1:
75                 raise ValueError('%d < 0 or > m * n - '
76                 '1',
77                         kwargs['d'])
78             temp = np.sort(temp.reshape(temp.shape[0]
79                 * temp.shape[1], 1), axis=0)
80             min_num = math.ceil(kwargs['d'] / 2)
81             max_num = kwargs['d'] // 2
82             temp = temp[min_num: temp.shape[0]*temp.
83                         shape[1] - max_num]
84             image[i][j][k] = np.mean(temp)
85         elif allowedtypes[mode] == 9:
86             image[i][j][k] = adaptive_mean(image, i,
87                 j, k, m=kwargs['m'], n=kwargs['n'],
88                 max_size=kwargs['max_size'])
89
90     return image
91
92
93
94
95
96
97
98

```

```

99     # judge B
100    if min_val < zxy < max_val:
101        return zxy
102    else:
103        return med_val
104    else:
105        # enlarge the kernel size
106        min_size += step
107        if min_size <= max_size:
108            return adaptive_mean(image, i, j, k, min_size=min_size,
109                                  max_size=max_size)
110        else:
111            return med_val

```

滤波器的比较

1. 算术均值滤波与几何滤波对叠加了高斯噪声的图像的复原效果比较

可以看到如图 1 所示，对噪声衰减都有作用，但几何均值滤波比算术均值滤波减少了对图像的模糊

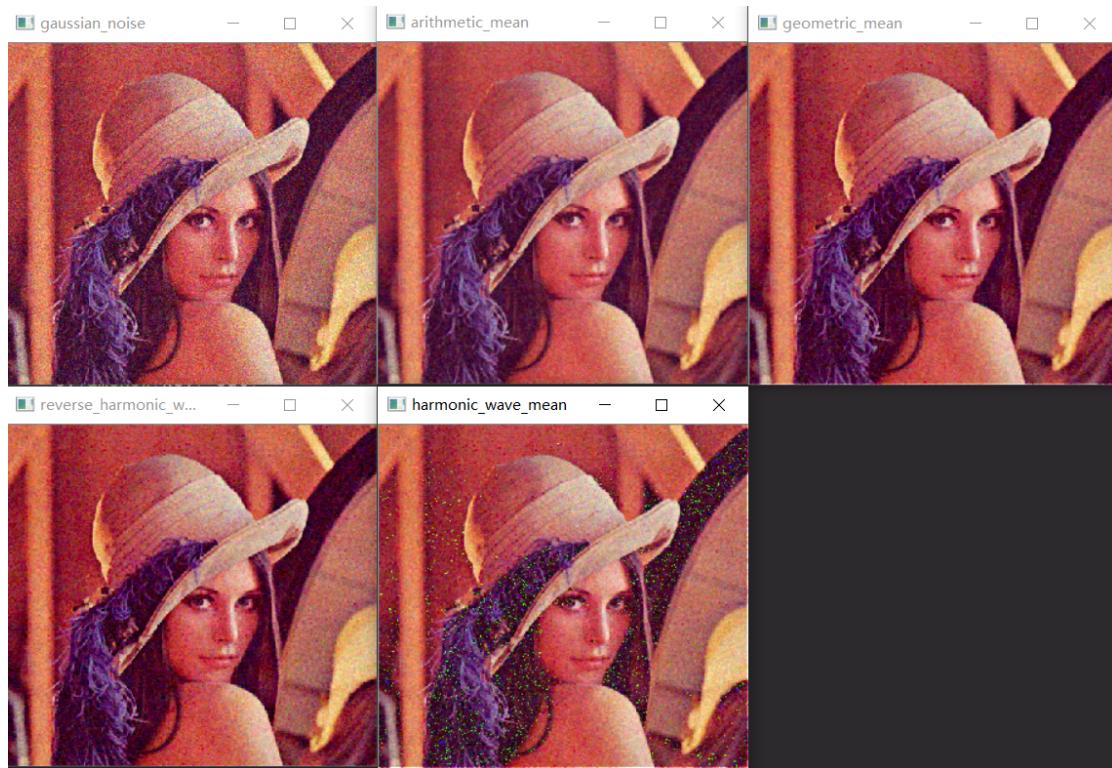


图 1: 运行结果

2. 分别用 $Q=1.5$ 对胡椒噪声和 $Q=-1.5$ 对盐噪声进行处理

可以看到如图 2 3 所示，正阶滤波器在使暗区模糊的损失下，使背景较为清晰。负阶则相反。

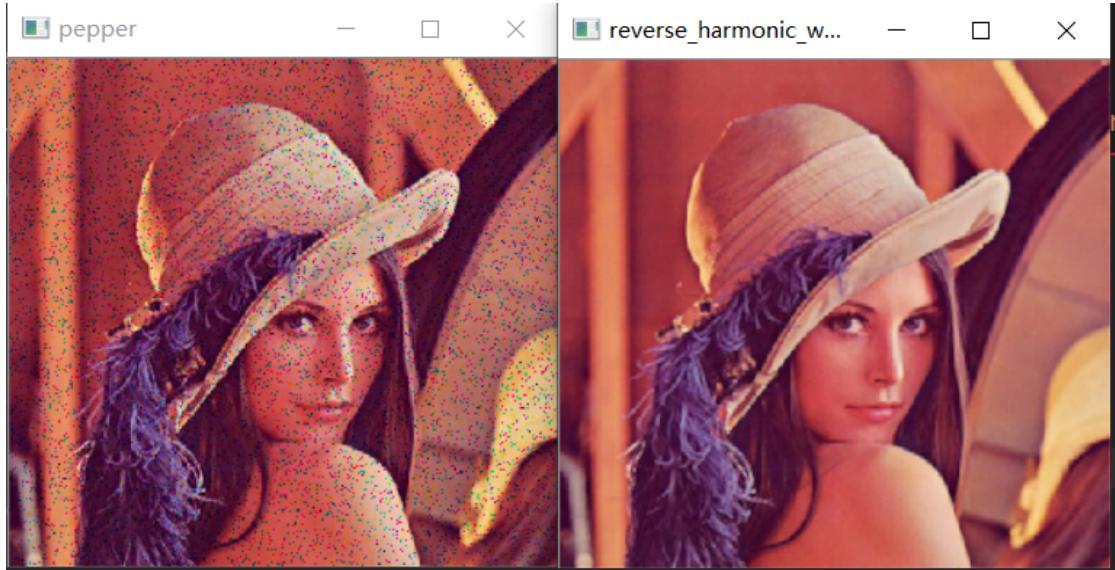


图 2: 运行结果

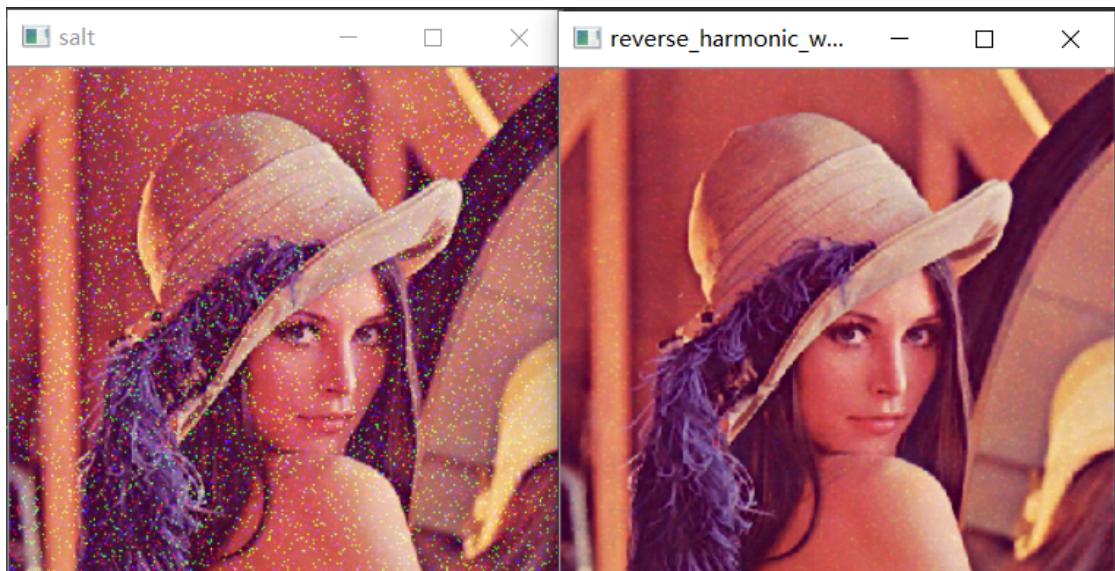


图 3: 运行结果

3. 对椒盐噪声反复用中值滤波处理

可以看到如图 4 所示，过度重复使用中值滤波可能会对图像造成模糊

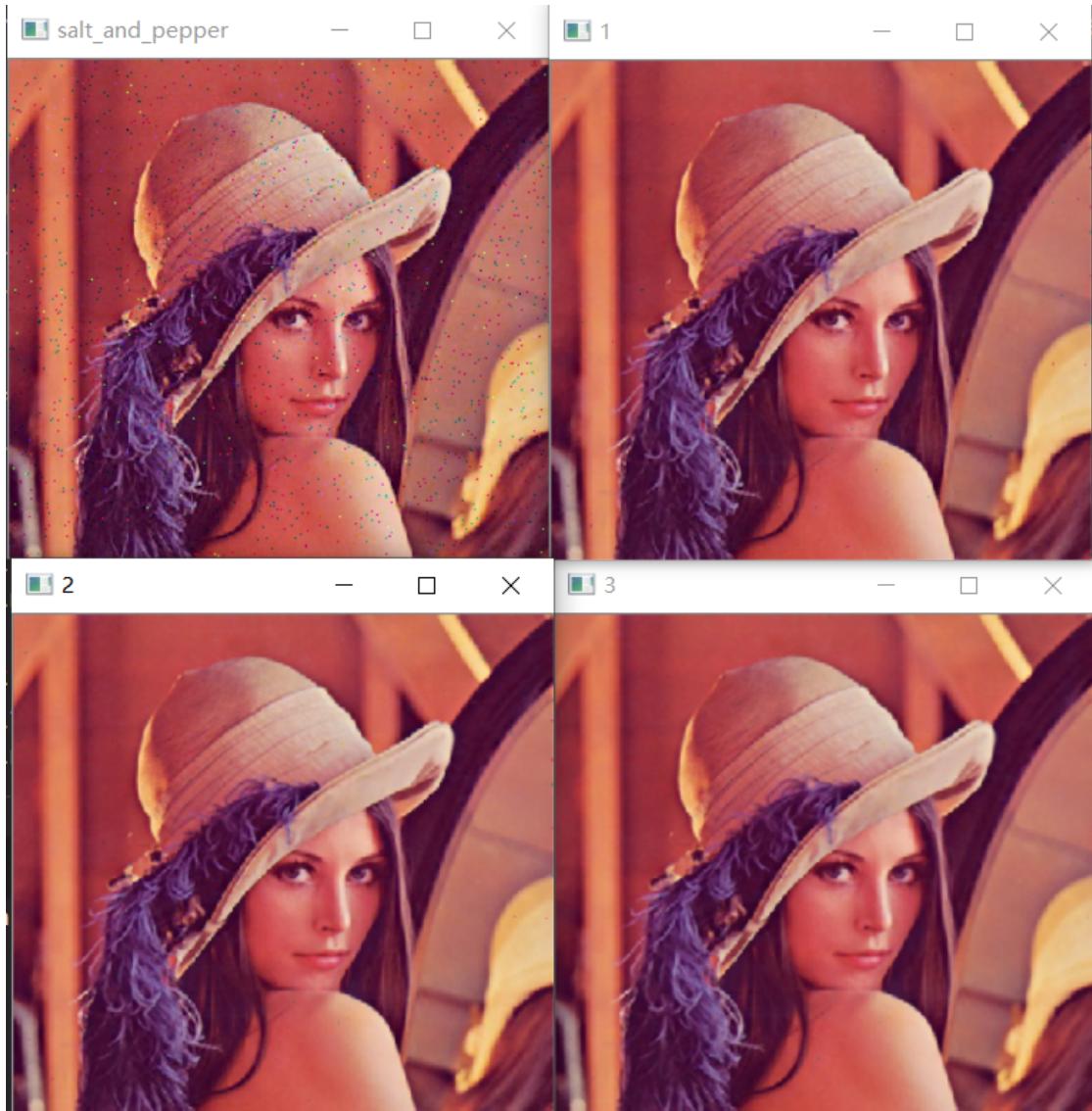


图 4: 运行结果

4. 对椒盐噪声使用最大值滤波和对盐噪声使用最小值滤波

可以看到如图 5 所示，最大值滤波器移除了一些暗像素，最小值滤波器则移除一些亮像素。

5. 对高斯噪声加上椒盐噪声污染的图像分别用算术均值、几何均值、中值滤波和 $d=5$ 修正后后的阿尔法均值滤波器比较

可以看到如图 6 所示，由于脉冲噪声的存在，算术均值滤波器和几何均值滤波器没有起到良好作用。中值滤波器和阿尔法滤波器效果更好，阿尔法最好。

6. 对椒盐噪声污染的图像分别用中值滤波和 $S_{max}=7$ 的自适应中值滤波处理

可以看到如图 7 所示，噪声去除水平与中值滤波效果相近，但图像保持了点的尖锐性及其细节。

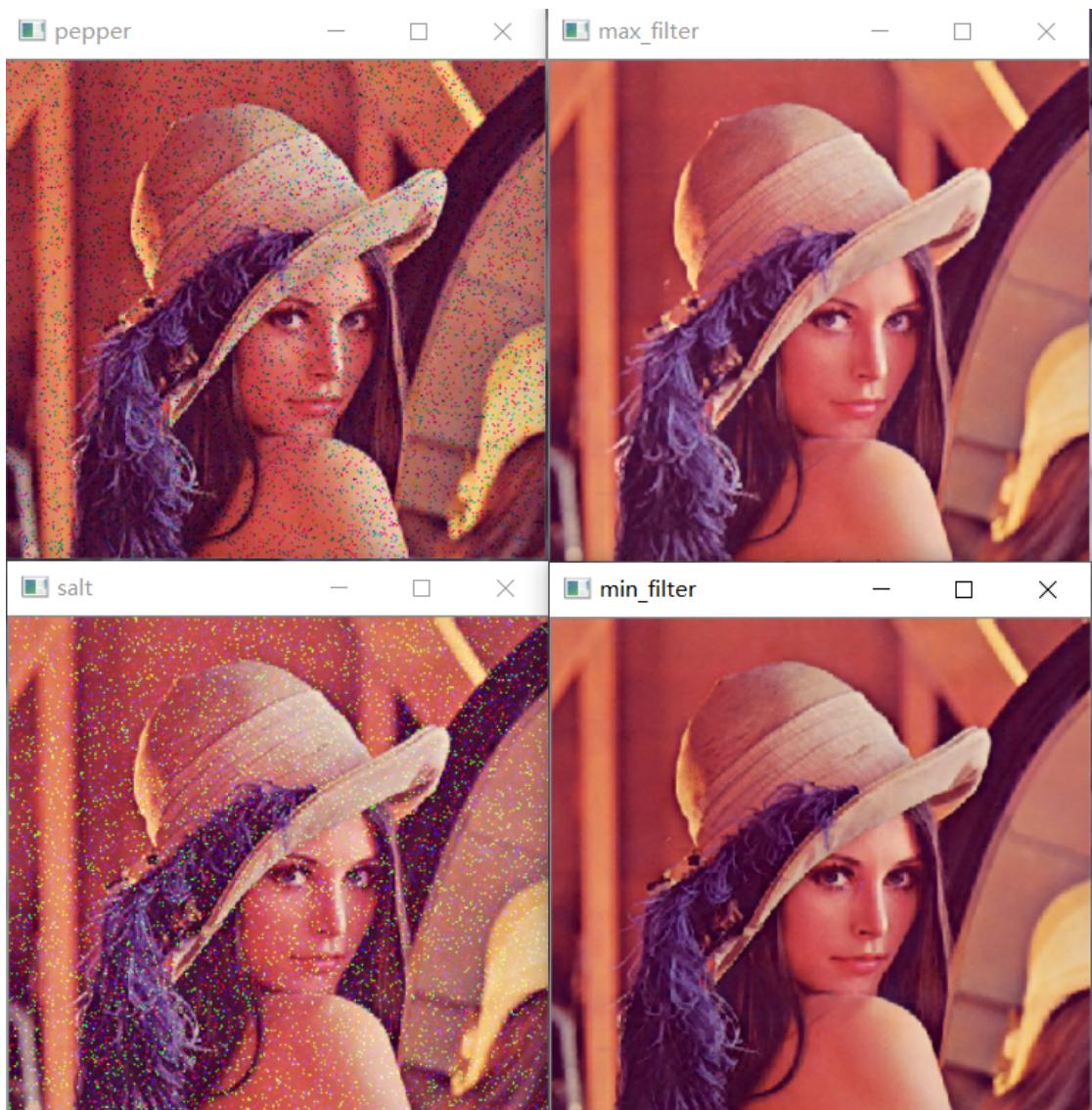


图 5: 运行结果

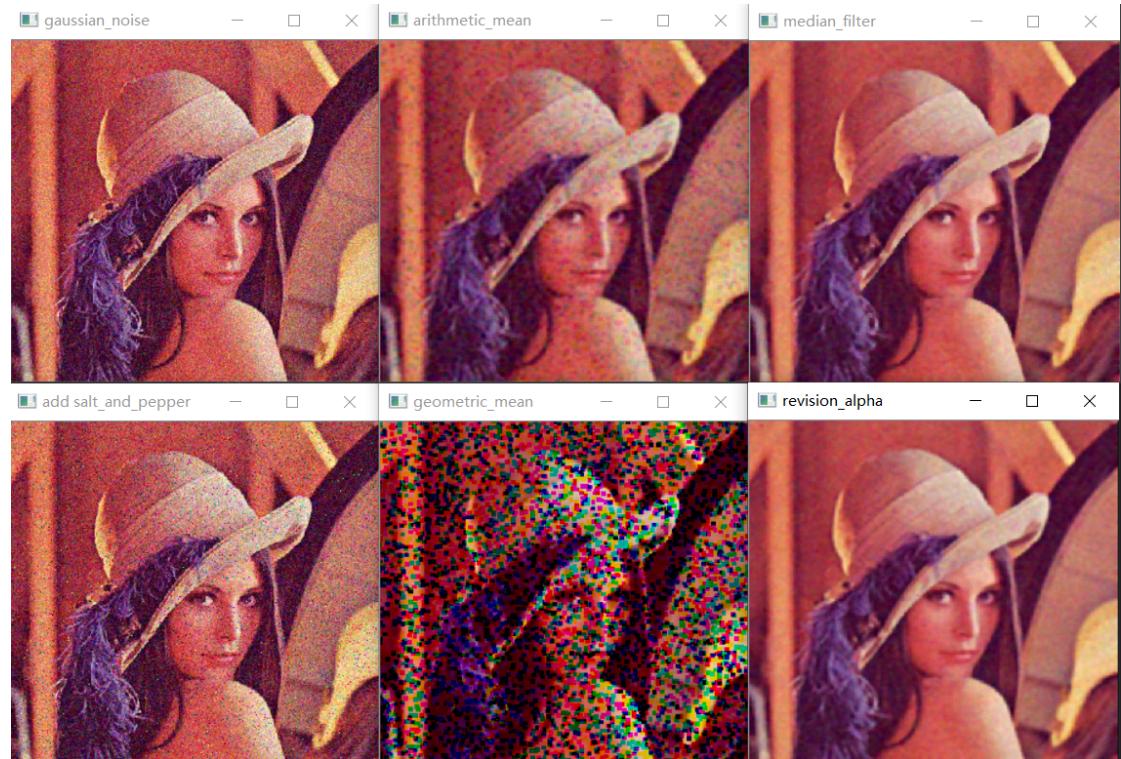


图 6: 运行结果

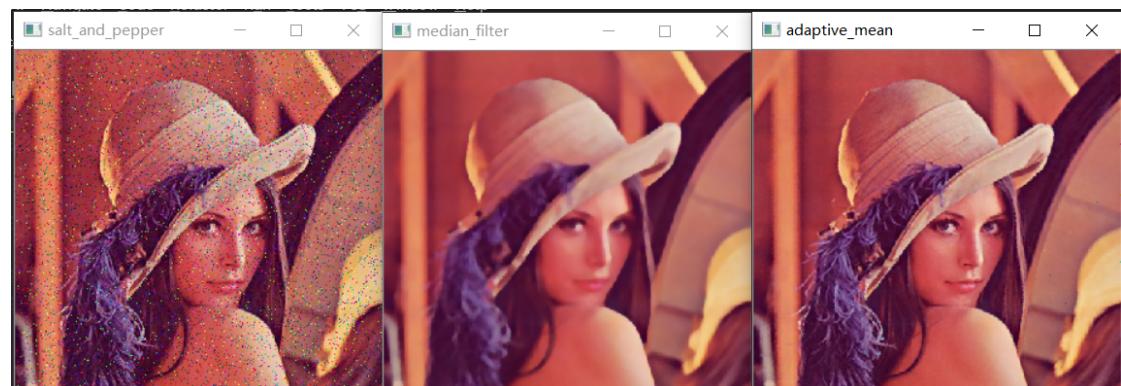


图 7: 运行结果

运行结果在文件目录的 report/image 目录下