

Generating different noise and comparing different noise reduction methods

问题描述

In this problem, you are required to write a program to generate different types of random noises started from the Uniform noise and Gaussian noise. (one of the reference may be “Digital image processing using Matlab” PP.143-150. And then add some of these noises to the circuit image (Circuit.tif) and investigate the different mean filters and order statistics as the textbook did at pages 344-352.

算法思想

该问题需要我们先对图像添加噪声，在数字图像处理领域中，噪声主要来自图像的获取和传输的过程中。本题要求我们自己写出添加均匀噪声和高斯噪声的算法。

1. 均匀噪声

本题中提及的均匀噪声中包含有方差参数，故我认为实际上应该是高斯白噪声，即一个噪声的幅度分布服从高斯分布，功率谱密度又是均匀分布的。

在本题中我直接根据方差参数来调用高斯函数产生一个高斯分布矩阵，进而得到高斯噪声，并且保证其功率谱密度均匀分布，即得到了均匀噪声。

2. 高斯噪声

高斯噪声是指它的概率密度函数服从高斯分布（即正态分布）的一类噪声。常见的高斯噪声包括起伏噪声、宇宙噪声、热噪声和散粒噪声等等。

高斯分布，也称正态分布，又称常态分布，记为 $N(\mu, \sigma^2)$ ，其中 μ, σ^2 为分布的参数，分别为高斯分布的期望和方差。

在本题中我也是直接根据方差参数来调用高斯函数产生一个高斯分布矩阵，进而得到高斯噪声。

在图像加入噪声后，我们需要对其进行还原，本题中的图片仅有加性噪声时，故我们可以考虑空间滤波的方法来对图片进行还原。

以下是本题用到的几种均值滤波器。

1. 算术均值滤波器

算术均值滤波器的定义如下：

$$\hat{f}(x, y) = \frac{1}{mn} \sum_{(s,t) \in S_{xy}} g(s, t)$$

其中 S_{xy} 表示中心在 (x, y) 处，大小为 $m * n$ 的矩形子图像窗口的一组坐标。

2. 几何均值滤波器

几何均值滤波器的定义如下：

$$\hat{f}(x, y) = \left[\prod_{(s,t) \in S_{xy}} g(s, t) \right]^{\frac{1}{mn}}$$

几何均值滤波器的平滑效果跟算术均值滤波器相当，并且其丢失的图像细节更少。

3. Harmonic 均值滤波器

谐波均值滤波器的定义如下：

$$\hat{f}(x, y) = \frac{mn}{\sum_{(s,t) \in S_{xy}} \frac{1}{g(s, t)}}$$

谐波均值滤波器对于盐粒噪声的效果好，但是不适用于胡椒噪声。

4. Constrainedharmonic 均值滤波器

逆谐波均值滤波器的定义如下：

$$\hat{f}(x, y) = \frac{\sum_{(s,t) \in S_{xy}} g(s, t)^{Q+1}}{\sum_{(s,t) \in S_{xy}} g(s, t)^Q}$$

Q 为逆谐波滤波器的阶数，该滤波器适用于椒盐噪声的复原，当 Q 为正值时，适用于消除胡椒噪声，当 Q 为负值时，适用于消除盐粒噪声，当 $Q = 0$ 时，该滤波器简化为算术均值滤波器，当 $Q = -1$ 时，该滤波器简化为谐波均值滤波器。

5. 中值滤波器

中值滤波器的定义如下：

$$\hat{f}(x, y) = median_{(s,t) \in S_{xy}} \{g(s, t)\}$$

中值滤波器使用一个像素邻域中灰度级的中值来代替该像素的值，非常适用于某些类型的随机噪声，且比相同尺寸的线性平滑滤波器引起的模糊更少。

在存在单极或双极脉冲噪声的情况下，中值滤波器十分有效。

6. 最大值滤波器

最大值滤波器的定义如下：

$$\hat{f}(x, y) = max_{(s,t) \in S_{xy}} \{g(s, t)\}$$

该滤波器对于发现图像中的亮点十分有用，适用于消除胡椒噪声的影响。

7. 最小值滤波器

最小值滤波器的定义如下：

$$\hat{f}(x, y) = min_{(s,t) \in S_{xy}} \{g(s, t)\}$$

该滤波器对于发现图像中的暗点十分有用，适用于消除盐粒噪声的影响。

8. 修正的 alpha 均值滤波器

修正的 alpha 均值滤波器定义如下：

$$\hat{f}(x, y) = \frac{1}{mn - d} \sum_{(s,t) \in S_{xy}} g_r(s, t)$$

由公式可知，与算术均值滤波器相比，修正的 alpha 均值滤波器去掉了 $d / 2$ 个最低的灰度值和 $d / 2$ 个最高的灰度值， $gr(s,t)$ 代表剩下的 $mn - d$ 个像素。当 $d = 0$ 时该滤波器退化为算术均值滤波器，当 $d = mn - 1$ 时，该滤波器退化为中值滤波器，当 d 取其他值时该滤波器对多种噪声的消除很有效。

源码分析

- 预处理图像，对图像外围加上一层边，其灰度值等于其原本边界像素点的灰度值。

```
1 def imagePreprocessing(input):
2     row, col = input.shape
3     output = np.zeros((row + 2, col + 2))
4     for i in range(1, row + 1):
5         for j in range(1, col + 1):
6             output[i, j] = input[i - 1, j - 1]
7     for i in range(row + 2):
8         output[i, 0] = output[i, 1]
9         output[i, col + 1] = output[i, col]
10    for j in range(col + 2):
11        output[0, j] = output[1, j]
12        output[row + 1, j] = output[row, j]
13    return output
```

- 读取原始图像，对原始图像添加各种噪声，然后使用各种滤波器进行噪声的消除，最后展示图像

```
1 def main():
2     originalImage = np.array(Image.open('./resource/Circuit.tif'))
3     plt.subplot(2, 2, 1)
4     plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
5     plt.title('Original')
6
7     gaussianNoiseImage = AddNoise.addNoisy("Gaussian", originalImage)
8     plt.subplot(2, 2, 2)
9     plt.imshow(gaussianNoiseImage, cmap = plt.get_cmap('gray'))
10    plt.title('Gaussian Noise')
11
12    arithmeticMeanImage = MeanFilters.ArithmeticMeanFilter3(gaussianNoiseImage)
13    plt.subplot(2, 2, 3)
14    plt.imshow(arithmeticMeanImage, cmap = plt.get_cmap('gray'))
15    plt.title('Arithmetic Mean Filter')
16
17    geometricMeanImage = MeanFilters.GeometricMeanFilter3(gaussianNoiseImage)
18    plt.subplot(2, 2, 4)
19    plt.imshow(geometricMeanImage, cmap = plt.get_cmap('gray'))
20    plt.title('Geometric Mean Filter')
21
22    plt.show()
23
24    pepperNoiseImage = AddNoise.addNoisy("Pepper", originalImage)
25    plt.subplot(2, 2, 1)
26    plt.imshow(pepperNoiseImage, cmap = plt.get_cmap('gray'))
27    plt.title('Pepper Noise')
28
29    saltNoiseImage = AddNoise.addNoisy("Salt", originalImage)
30    plt.subplot(2, 2, 2)
31    plt.imshow(saltNoiseImage, cmap = plt.get_cmap('gray'))
32    plt.title('Salt Noise')
33
34    contraharmonicMeanImage1 = MeanFilters.ContraharmonicMeanFilter(pepperNoiseImage, 1.
```

```

5)
35 plt.subplot(2, 2, 3)
36 plt.imshow(contraharmonicMeanImage1, cmap = plt.get_cmap('gray'))
37 plt.title('Pepper Noise Use Contraharmonic Mean Filter Q = 1.5')
38
39
40 contraharmonicMeanImage2 = MeanFilters.ContraharmonicMeanFilter(saltNoiseImage, -1.5
)
41 plt.subplot(2, 2, 4)
42 plt.imshow(contraharmonicMeanImage2, cmap = plt.get_cmap('gray'))
43 plt.title('Salt Noise Use Contraharmonic Mean Filter Q = -1.5')
44
45 plt.show()
46
47 contraharmonicMeanImage3 = MeanFilters.ContraharmonicMeanFilter(pepperNoiseImage, -1
.5)
48 plt.subplot(1, 2, 1)
49 plt.imshow(contraharmonicMeanImage3, cmap = plt.get_cmap('gray'))
50 plt.title('Pepper Noise Use Contraharmonic Mean Filter Q = -1.5')
51
52 contraharmonicMeanImage4 = MeanFilters.ContraharmonicMeanFilter(saltNoiseImage, 1.5)
53 plt.subplot(1, 2, 2)
54 plt.imshow(contraharmonicMeanImage4, cmap = plt.get_cmap('gray'))
55 plt.title('Salt Noise Use Contraharmonic Mean Filter Q = 1.5')
56
57 plt.show()
58
59 NoiseImage = AddNoise.addNoisy("Salt", originalImage)
60 saltAndPepperNoiseImage = AddNoise.addNoisy("Pepper", NoiseImage)
61 plt.subplot(2, 2, 1)
62 plt.imshow(saltAndPepperNoiseImage, cmap = plt.get_cmap('gray'))
63 plt.title('Salt And Pepper Noise')
64
65 medianImage = MeanFilters.MedianFilter(saltAndPepperNoiseImage)
66 plt.subplot(2, 2, 2)
67 plt.imshow(medianImage, cmap = plt.get_cmap('gray'))
68 plt.title('1st Median Filter')
69
70 medianImage = MeanFilters.MedianFilter(medianImage)
71 plt.subplot(2, 2, 3)
72 plt.imshow(medianImage, cmap = plt.get_cmap('gray'))
73 plt.title('2nd Median Filter')
74
75 medianImage = MeanFilters.MedianFilter(medianImage)
76 plt.subplot(2, 2, 4)
77 plt.imshow(medianImage, cmap = plt.get_cmap('gray'))
78 plt.title('3rd Median Filter')
79
80 plt.show()
81
82 maxImage = MeanFilters.MaxFilter(pepperNoiseImage)
83 plt.subplot(1, 2, 1)
84 plt.imshow(maxImage, cmap = plt.get_cmap('gray'))
85 plt.title('Max Filter')
86
87 minImage = MeanFilters.MinFilter(saltNoiseImage)
88 plt.subplot(1, 2, 2)
89 plt.imshow(minImage, cmap = plt.get_cmap('gray'))
90 plt.title('Min Filter')
91
92 plt.show()

```

```

93
94 uniformNoiseImage = AddNoise.addNoisy('Uniform', originalImage, variance = 800)
95 plt.subplot(2, 3, 1)
96 plt.imshow(uniformNoiseImage, cmap = plt.get_cmap('gray'))
97 plt.title('Uniform Noise')
98
99 unifSaltNoiseImage = AddNoise.addNoisy('Salt', uniformNoiseImage)
100 unifSaltPepperNoiseImage = AddNoise.addNoisy('Pepper', unifSaltNoiseImage)
101 plt.subplot(2, 3, 2)
102 plt.imshow(unifSaltPepperNoiseImage, cmap = plt.get_cmap('gray'))
103 plt.title('Uniform Salt Pepper Noise')
104
105 arithmeticMeanImage5 = MeanFilters.ArithmeticMeanFilter5(unifSaltPepperNoiseImage)
106 plt.subplot(2, 3, 3)
107 plt.imshow(arithmeticMeanImage5, cmap = plt.get_cmap('gray'))
108 plt.title('Arithmetic Mean Filter')
109
110 geometricMeanImage5 = MeanFilters.GeometricMeanFilter5(unifSaltPepperNoiseImage)
111 plt.subplot(2, 3, 4)
112 plt.imshow(geometricMeanImage5, cmap = plt.get_cmap('gray'))
113 plt.title('Geometric Mean Filter')
114
115 medianImage5 = MeanFilters.MedianFilter5(unifSaltPepperNoiseImage)
116 plt.subplot(2, 3, 5)
117 plt.imshow(medianImage5, cmap = plt.get_cmap('gray'))
118 plt.title('Median Filter')
119
120 alphaTrimmedMeanFilter = MeanFilters.AlphaMeanFilter(unifSaltPepperNoiseImage, 5)
121 plt.subplot(2, 3, 6)
122 plt.imshow(alphaTrimmedMeanFilter, cmap = plt.get_cmap('gray'))
123 plt.title('Alpha Trimmed Mean Filter')
124
125 plt.show()

```

- 生成均匀噪声，高斯噪声，盐粒噪声，胡椒噪声

```

1 def addNoisy(noiseType, originalImage, mean = 0, variance = 400, prob = 0.1):
2     row, col= originalImage.shape
3     noisyImage = np.zeros(originalImage.shape)
4     if noiseType == "Uniform":
5         std = variance ** 0.5
6         uniformNoise = np.random.normal(mean, std, (row, col))
7         uniformNoise = uniformNoise.reshape(row, col)
8         noisyImage = originalImage + uniformNoise
9         for i in range(row):
10             for j in range(col):
11                 noisyImage[i, j] = round(noisyImage[i, j])
12                 if noisyImage[i, j] < 0:
13                     noisyImage[i, j] = 0
14                 if noisyImage[i, j] > 255:
15                     noisyImage[i, j] = 255
16
17     elif noiseType == "Gaussian":
18         std = variance ** 0.5
19         gaussianNoise = np.random.normal(mean, std, (row, col))
20         gaussianNoise = gaussianNoise.reshape(row, col)
21         noisyImage = originalImage + gaussianNoise
22         for i in range(row):
23             for j in range(col):
24                 noisyImage[i, j] = round(noisyImage[i, j])
25                 if noisyImage[i, j] < 0:

```

```

26             noisyImage[i, j] = 0
27         if noisyImage[i, j] > 255:
28             noisyImage[i, j] = 255
29
30     elif noiseType == "Salt":
31         for i in range(row):
32             for j in range(col):
33                 rdn = random.random()
34                 if rdn < prob:
35                     noisyImage[i, j] = 255
36                 else:
37                     noisyImage[i, j] = originalImage[i, j]
38
39     elif noiseType == "Pepper":
40         for i in range(row):
41             for j in range(col):
42                 rdn = random.random()
43                 if rdn > 1 - prob:
44                     noisyImage[i, j] = 0
45                 else:
46                     noisyImage[i, j] = originalImage[i, j]
47
48     return noisyImage

```

- 均值滤波函数，包括了算术均值滤波器，几何均值滤波器，谐波滤波器，逆谐波滤波器

```

1 def ArithmeticMeanFilter3(originalImage):
2     result = copy.copy(originalImage)
3     row, col = originalImage.shape
4     image = imagePreprocessing(originalImage)
5     for i in range(1, row + 1):
6         for j in range(1, col + 1):
7             result[i - 1, j - 1] = (image[i - 1, j - 1] + image[i - 1, j] + image[i - 1,
j + 1] + image[i, j - 1] + image[i, j] + \n
8             image[i, j + 1] + image[i + 1, j - 1] + image[i + 1, j] + image[i + 1, j
+ 1]) / 9
9             result[i - 1, j - 1] = round(result[i - 1, j - 1])
10
11    return result
12
12 def GeometricMeanFilter3(originalImage):
13     result = copy.copy(originalImage)
14     row, col = originalImage.shape
15     image = imagePreprocessing(originalImage)
16     e = 1 / 9
17     for i in range(1, row + 1):
18         for j in range(1, col + 1):
19             val = (image[i - 1, j - 1] * image[i - 1, j] * image[i - 1, j + 1] * \
20                   image[i, j - 1] * image[i, j] * image[i, j + 1] * \
21                   image[i + 1, j - 1] * image[i + 1, j] * image[i + 1, j + 1])
22             result[i - 1, j - 1] = round(np.power(val, e))
23             if result[i - 1, j - 1] > 255:
24                 result[i - 1, j - 1] = 255
25
26    return result
27
27 def HarmonicMeanFilter(originalImage):
28     result = copy.copy(originalImage)
29     row, col = originalImage.shape
30     image = imagePreprocessing(originalImage)
31     t = 9
32     for i in range(1, row + 1):
33         for j in range(1, col + 1):
34             val = (1 / image[i - 1, j - 1] + 1 / image[i - 1, j] + 1 / image[i - 1, j +

```

```

1] + \
35           1 / image[i, j - 1] + 1 / image[i, j] + 1 / image[i, j + 1] + \
36           1 / image[i + 1, j - 1] + 1 / image[i + 1, j] + 1 / image[i + 1, j + \
1])
37           result[i - 1, j - 1] = round(t / val)
38           if result[i - 1, j - 1] > 255:
39               result[i - 1, j - 1] = 255
40       return result
41
42 def ContraharmonicMeanFilter(originalImage, q):
43     result = copy.copy(originalImage)
44     row, col = originalImage.shape
45     image = imagePreprocessing(originalImage)
46     q1 = q + 1
47     q2 = q
48     for i in range(1, row + 1):
49         for j in range(1, col + 1):
50             val1 = (image[i - 1, j - 1] ** q1 + image[i - 1, j] ** q1 + image[i - 1, j + \
1] ** q1 + \
51                   image[i, j - 1] ** q1 + image[i, j] ** q1 + image[i, j + 1] ** q1 + \
52                   image[i + 1, j - 1] ** q1 + image[i + 1, j] ** q1 + image[i + 1, j + \
1] ** q1)
53             val2 = (image[i - 1, j - 1] ** q2 + image[i - 1, j] ** q2 + image[i - 1, j + \
1] ** q2 + \
54                   image[i, j - 1] ** q2 + image[i, j] ** q2 + image[i, j + 1] ** q2 + \
55                   image[i + 1, j - 1] ** q2 + image[i + 1, j] ** q2 + image[i + 1, j + \
1] ** q2)
56             if val2 == 0 and q > 0:
57                 result[i - 1, j - 1] = 0
58             elif val2 == 0 and q < 0:
59                 result[i - 1, j - 1] = 255
60             elif val1 == float('inf') and q > 0:
61                 result[i - 1, j - 1] = 255
62             elif val1 == float('inf') and q < 0:
63                 result[i - 1, j - 1] = 0
64             else:
65                 result[i - 1, j - 1] = round(val1 / val2)
66             if result[i - 1, j - 1] > 255:
67                 result[i - 1, j - 1] = 255
68     return result

```

- 统计滤波函数，包括了中值滤波器，最大值滤波器，最小值滤波器，修正的 alpha 均值滤波器

```

1 def MedianFilter(originalImage):
2     result = copy.copy(originalImage)
3     row, col = originalImage.shape
4     image = imagePreprocessing(originalImage)
5     for i in range(1, row + 1):
6         for j in range(1, col + 1):
7             result[i - 1, j - 1] = np.median((image[i - 1, j - 1], image[i - 1, j], imag
e[i - 1, j + 1], \
8                     image[i, j - 1], image[i, j], image[i, j + 1], \
9                     image[i + 1, j - 1], image[i + 1, j], image[i + 1, j + 1]))
10    return result
11
12
13 def MaxFilter(originalImage):
14     result = copy.copy(originalImage)
15     row, col = originalImage.shape

```

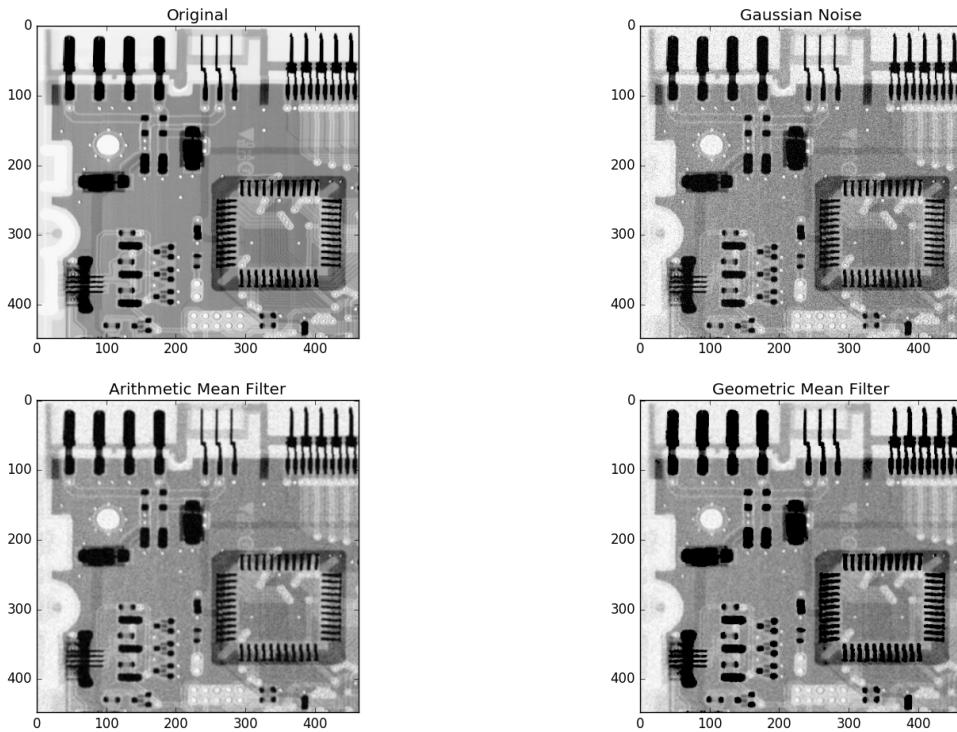
```

16     image = imagePreprocessing(originalImage)
17     for i in range(1, row + 1):
18         for j in range(1, col + 1):
19             result[i - 1, j - 1] = np.max((image[i - 1, j - 1], image[i - 1, j], image[i
- 1, j + 1], \
20                                         image[i, j - 1], image[i, j], image[i, j + 1], \
21                                         image[i + 1, j - 1], image[i + 1, j], image[i + 1, j + 1]))
22     return result
23
24
25 def MinFilter(originalImage):
26     result = copy.copy(originalImage)
27     row, col = originalImage.shape
28     image = imagePreprocessing(originalImage)
29     for i in range(1, row + 1):
30         for j in range(1, col + 1):
31             result[i - 1, j - 1] = np.min((image[i - 1, j - 1], image[i - 1, j], image[i
- 1, j + 1], \
32                                         image[i, j - 1], image[i, j], image[i, j + 1], \
33                                         image[i + 1, j - 1], image[i + 1, j], image[i + 1, j + 1]))
34     return result
35
36 def AlphaMeanFilter(originalImage, d = 5):
37     result = copy.copy(originalImage)
38     row, col = originalImage.shape
39     image = imagePreprocessing(originalImage)
40     image = imagePreprocessing(image)
41     for i in range(2, row + 2):
42         for j in range(2, col + 2):
43             listPix = (\
44                 image[i - 2, j - 2], image[i - 2, j - 1], image[i - 2, j], image[i - 2,
j + 1], image[i - 2, j + 2], \
45                 image[i - 1, j - 2], image[i - 1, j - 1], image[i - 1, j], image[i - 1,
j + 1], image[i - 1, j + 2], \
46                 image[i, j - 2], image[i, j - 1], image[i, j], image[i, j + 1], image[i,
j + 2], \
47                 image[i + 1, j - 2], image[i + 1, j - 1], image[i + 1, j], image[i + 1,
j + 1], image[i + 1, j + 2], \
48                 image[i + 2, j - 2], image[i + 2, j - 1], image[i + 2, j], image[i + 2,
j + 1], image[i + 2, j + 2])
49             listPix = sorted(listPix)
50             listPix = listPix[round(d / 2) : round(d / 2) - d]
51             result[i - 2, j - 2] = round(np.mean(listPix))
52     return result

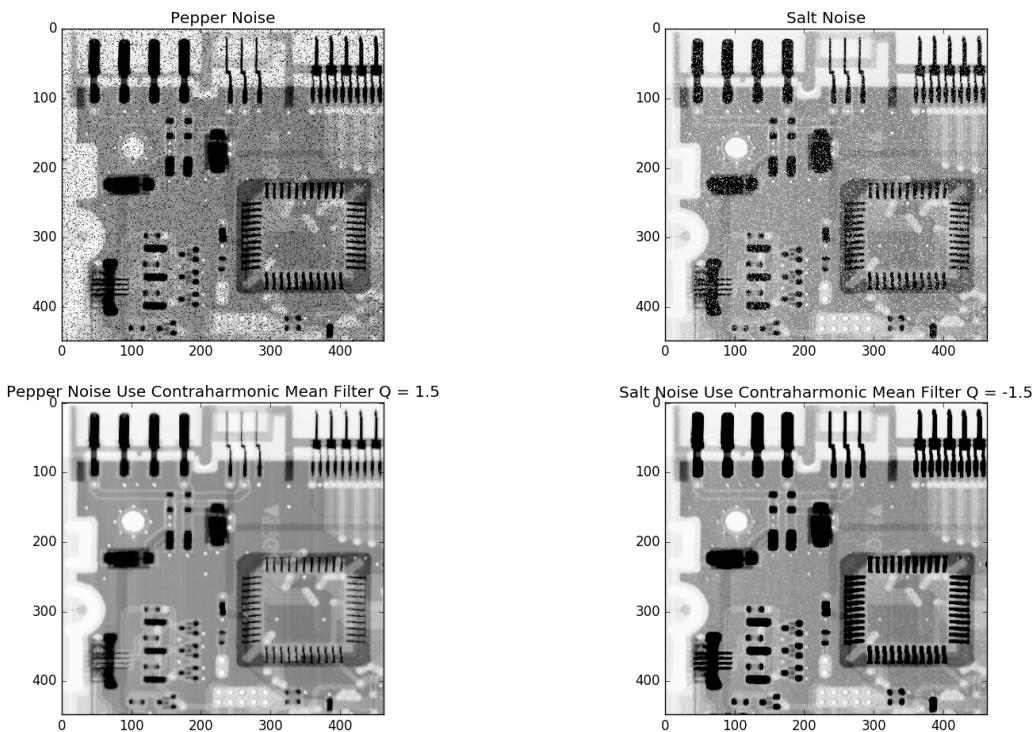
```

实验结果

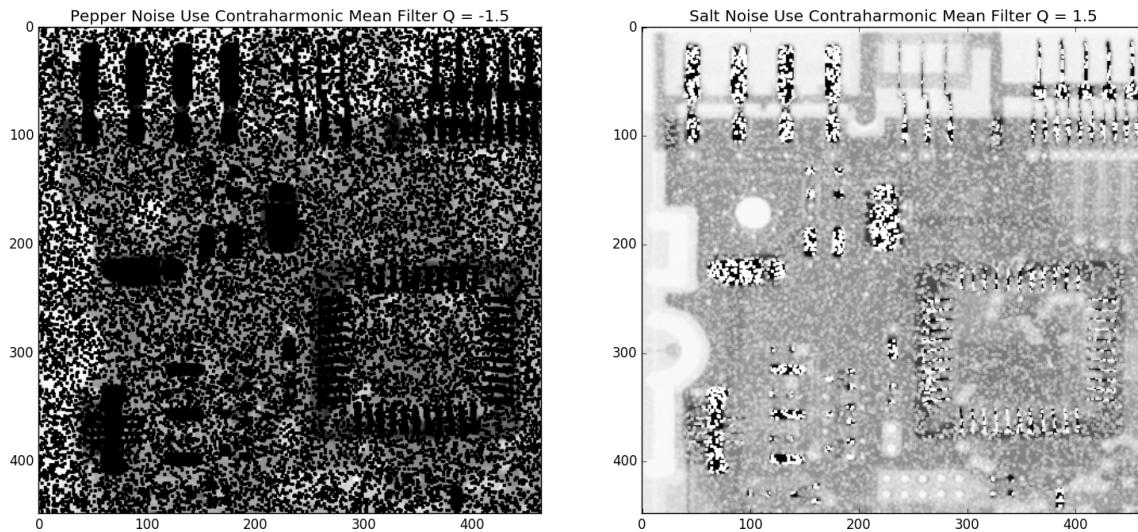
下图为添加高斯噪声后，使用算术均值滤波器和几何均值滤波器消除噪声的结果。



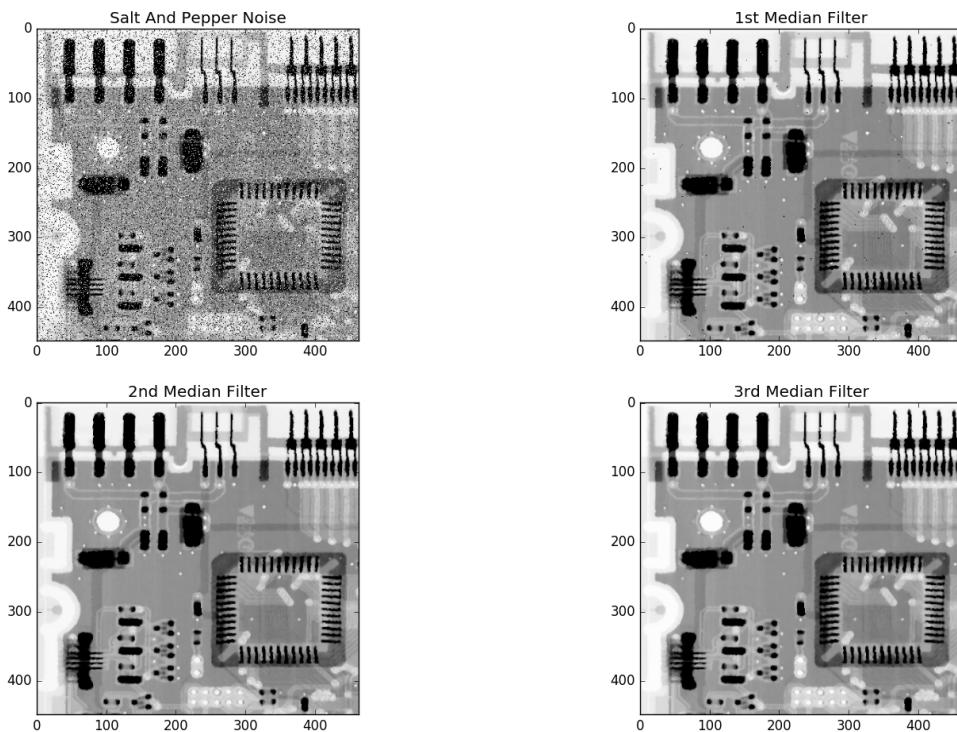
由于当 Q 为正值时，逆谐波滤波器适用于消除胡椒噪声，当 Q 为负值时，逆谐波滤波器适用于消除盐粒噪声，故对于两种噪声的消除效果都良好。



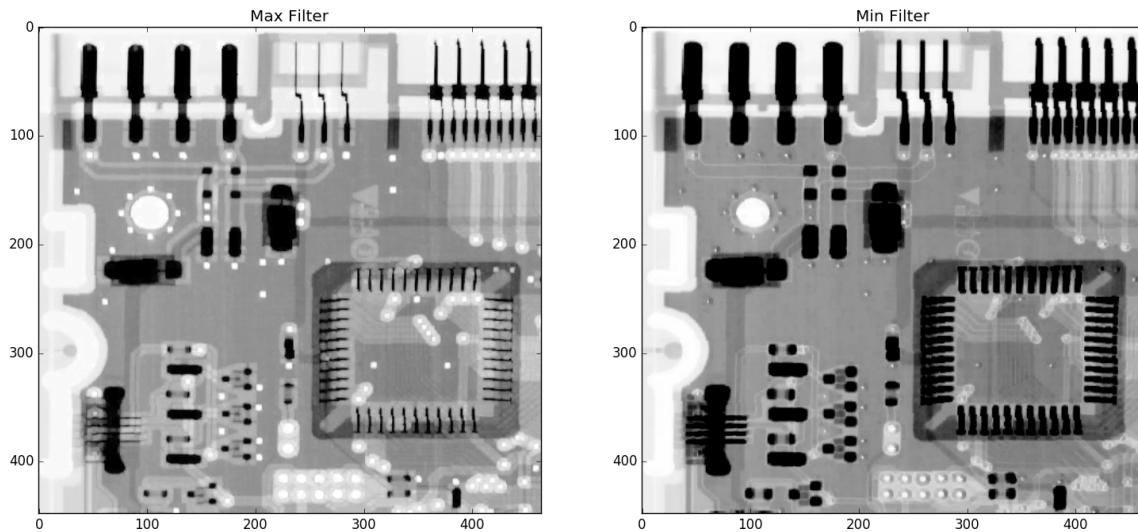
当 Q 的值使用不当时，逆谐波滤波器不仅不能消除两种噪声，还会使图片得到严重损坏。



如图下可知，使用多次中值滤波器后，噪声的消除效果会越来越好，这是因为噪声引起的扰动在多次取中值后，较易被抹消。



如下左图为最大值滤波器，整体图像偏白，白色边缘部分相比原图变大，因为白色的灰度值为 255，右图为最小值滤波器，整体图像偏黑，黑色边缘部分相比原图变大，因为黑色的灰度值为 0。



如下图可知，在添加多种噪声后，算数均值滤波器去噪效果不佳，几何均值滤波器效果最差，因为由定义可知，几何均值滤波器用到几个像素灰度值的连乘，在加入多种噪声后，很容易使得多个像素点的灰度值都为0，连乘后依旧为0，使得图片呈现出一片黑。

相反，统计滤波器中的中值滤波器不太受多种噪声叠加的影响，因为加上多种噪声后其整体中值的变动不会太大。修正的alpha中值滤波器表现也相对较为良好。

