数图第四次作业

标签： 数图作业

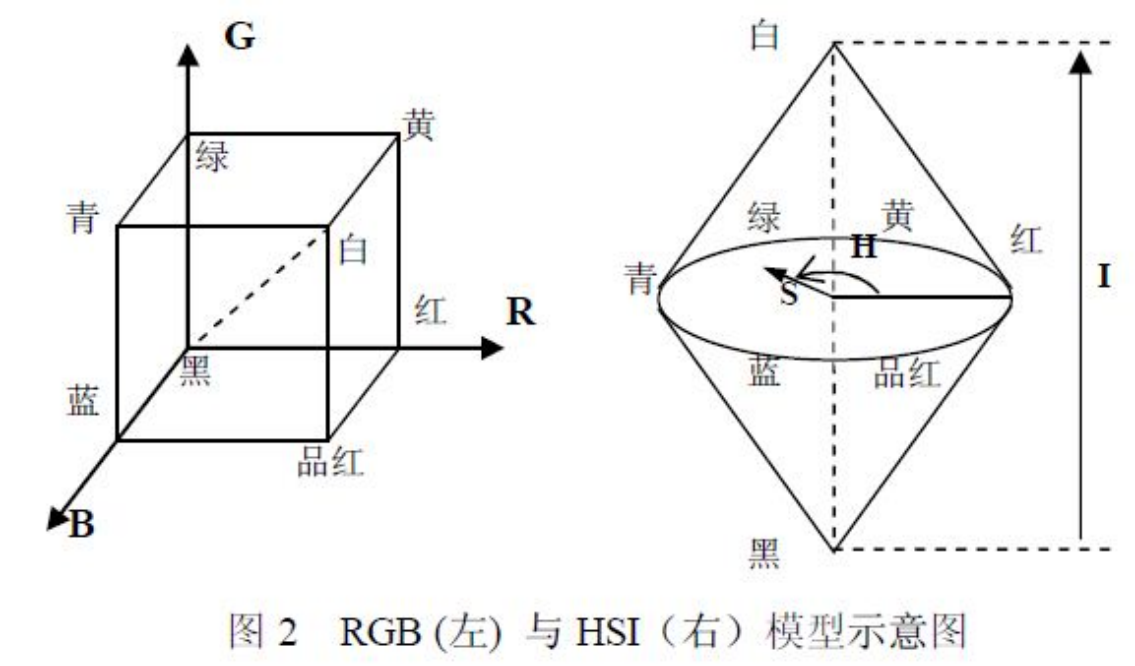
郭柱明 15331094

1.1

1

> 答：在灰度图中0-255，越接近0越黑，越接近255越白。原图领带是红色的，则r图的领带的灰度值是接近255，就是很白的，四张图中图d的领带是最白的，为R图；原图眼睛是蓝色的，图a眼睛最白，为B图；原图脸上的斑点为绿色，图b上的斑点最白，为G图。那么剩下的图c是灰度图。

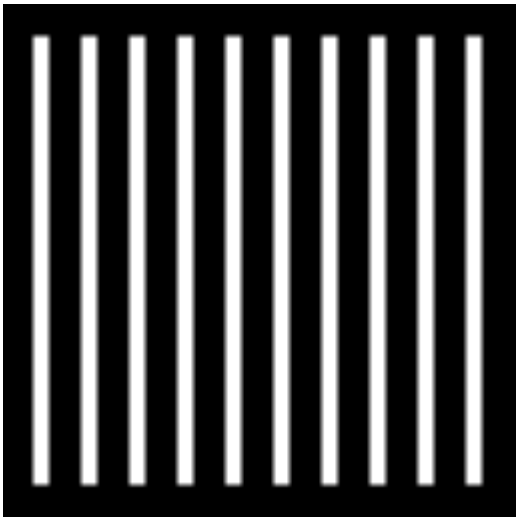
2

> 答：由RGB和HSI示意图可知：
>
> 
>
> 图 2　RGB (左) 与 HSI（右）模型示意图
>
> H增大60°在之后，原图的黄色转为绿色，而黑色和白色不变，因此是图b

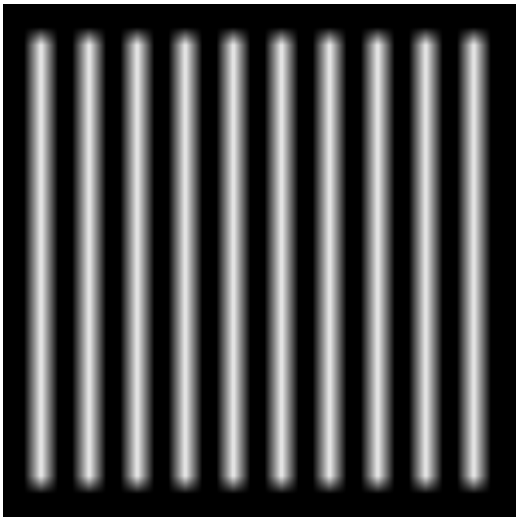
2

1.

3 * 3



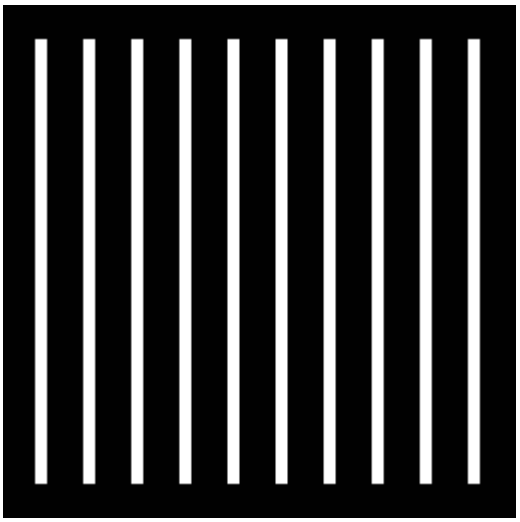
9 * 9



> 3*3滤波器滤波后的图像白色竖条的宽更小，黑色的竖条更宽
> 9*9滤波器滤波后的图像白色竖条的宽更大，黑色的竖条更窄

2.

3 * 3

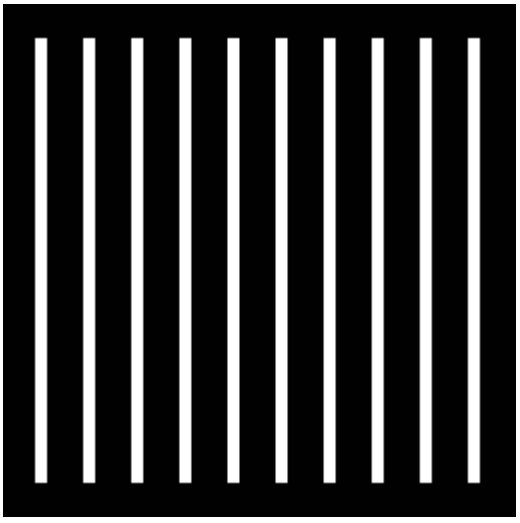


9 * 9

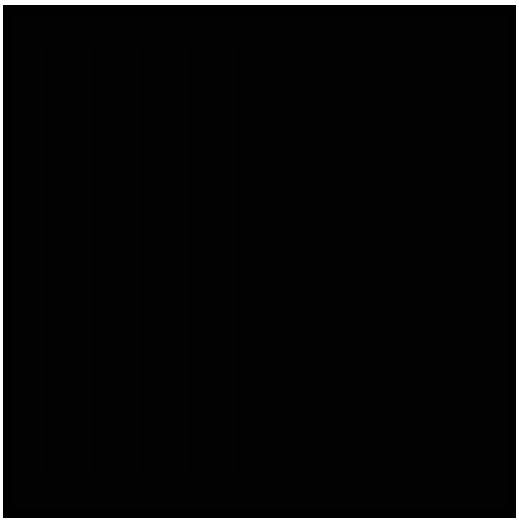
> 3*3的白色竖条宽度更大，黑色相对较宽
> 9*9白色竖条很细，黑色很宽。

3.

3 * 3



9 * 9



> 3 * 3中的白色竖条比原图细一点，但是9 * 9中完全找不到白色竖条了

2.3

1.
高斯噪声生成器：

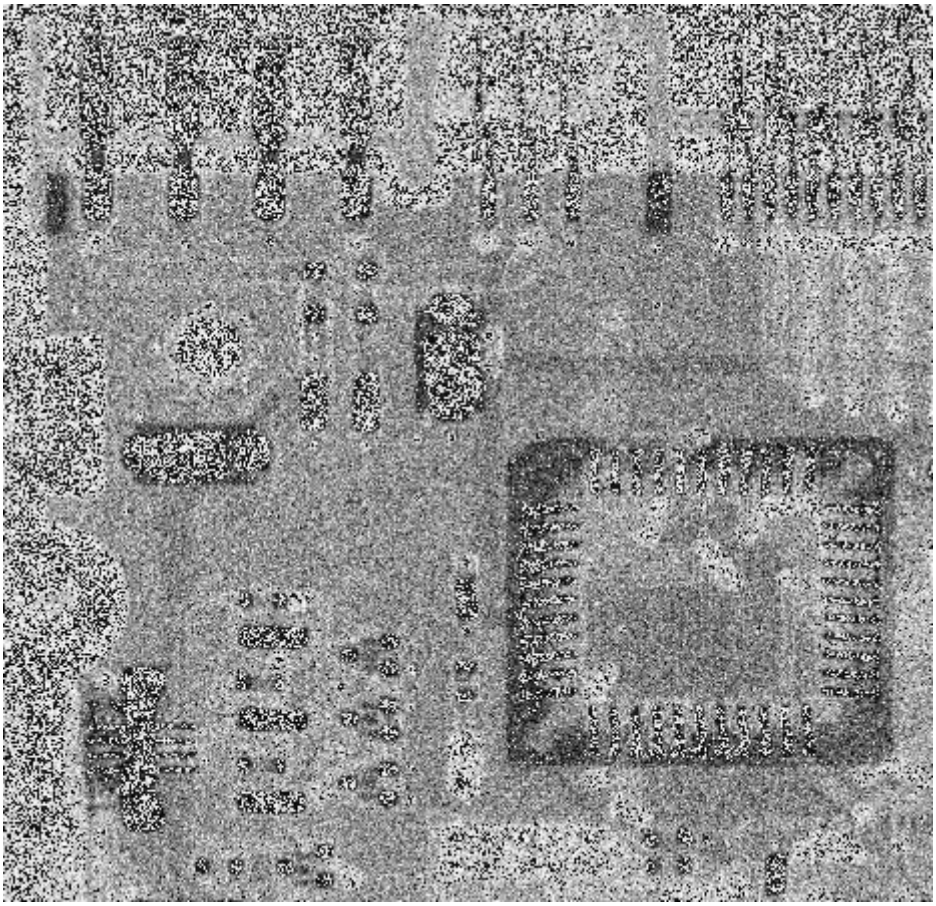
```
#mean为均值，sigma为标准差，img为输入图像数组
def gaussian_noise(mean, sigma, img):
    #获取图像高和宽
    heigh, width = img.shape
    size = heigh * width
    #按照均值，标准差和图像尺寸生成高斯分布的随机数序列
    s = numpy.random.normal(mean, sigma, size)
    index = 0
    #给图像加上高斯噪声
    for i in range(heigh):
        for j in range(width):
            img[i][j] = img[i][j] + s[index]
            index = index + 1
    #返回结果图像数组
    return img
```

椒盐噪声生成器：

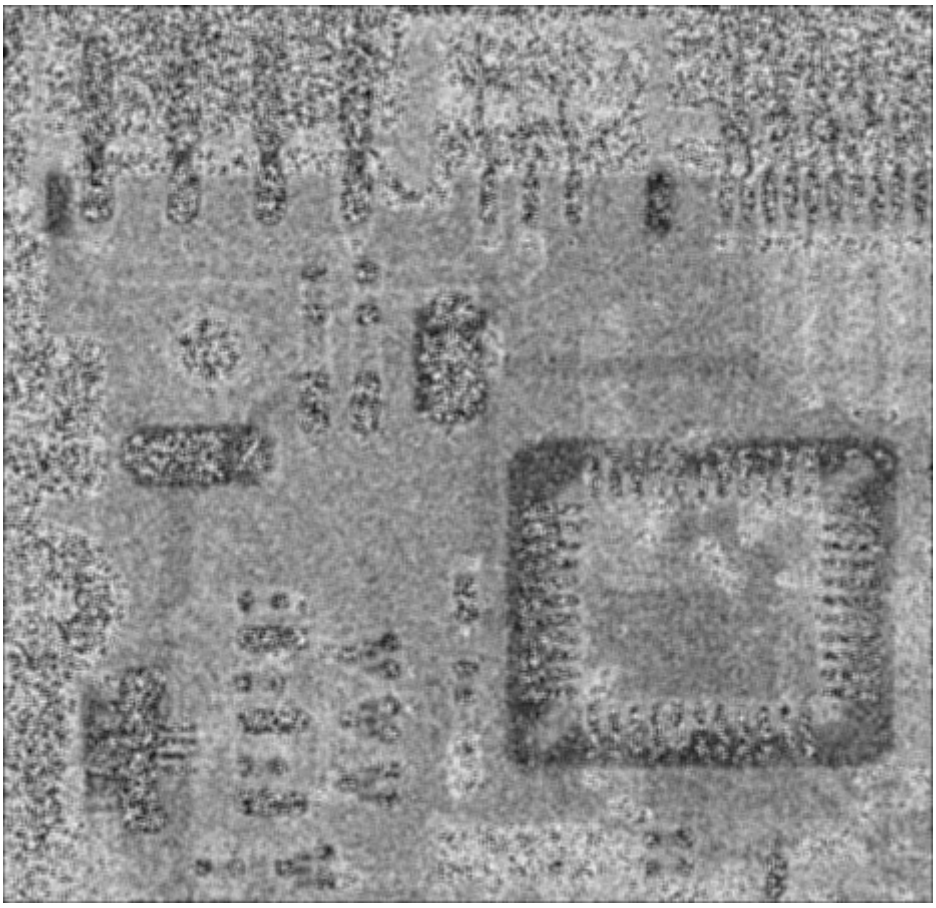
```python
#pa为椒的分布概率，pb为盐的分布概率
def salt_and_pepper_noise(pa, pb, img):
    pb = 1 - pb
    heigh, width = img.shape
    #给图像加上椒盐噪声
    for i in range(heigh):
        for j in range(width):
            p = numpy.random()
            if p <= pa:
                img[i][j] = 0
            elif p >= pb:
                img[i][j] = 255
    #返回结果图像数组
    return img
```

2.

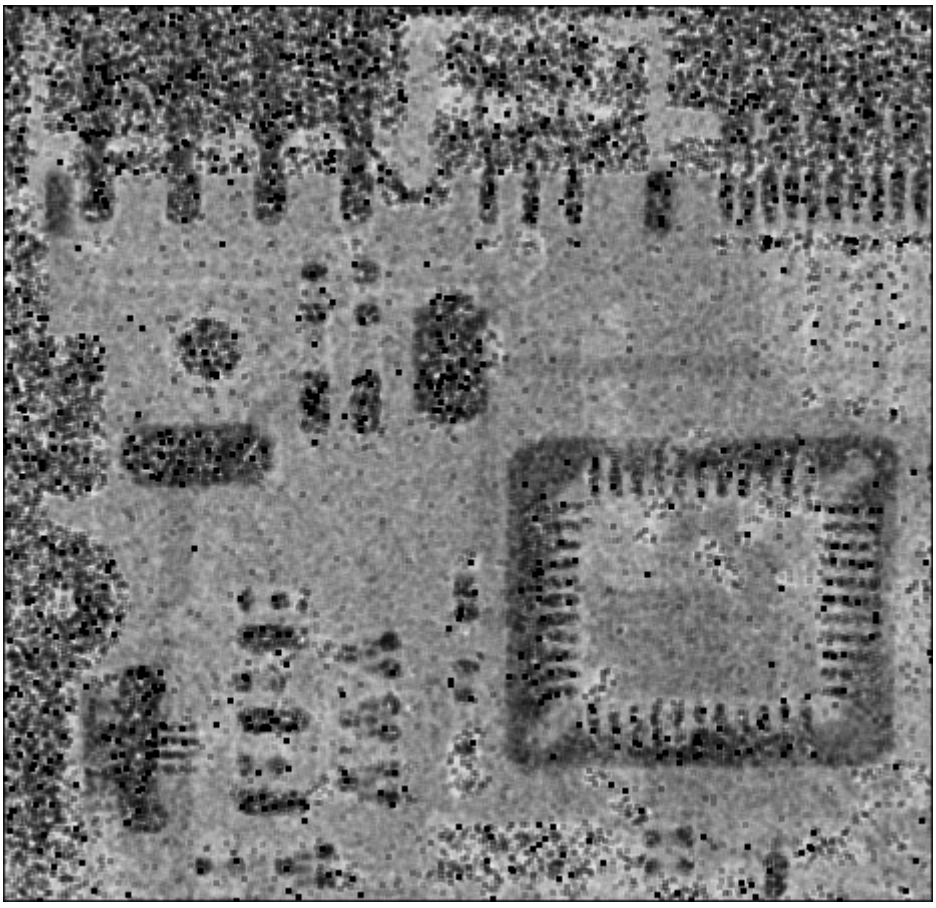
图片添加均值为0、标准差为40的高斯噪声，结果：



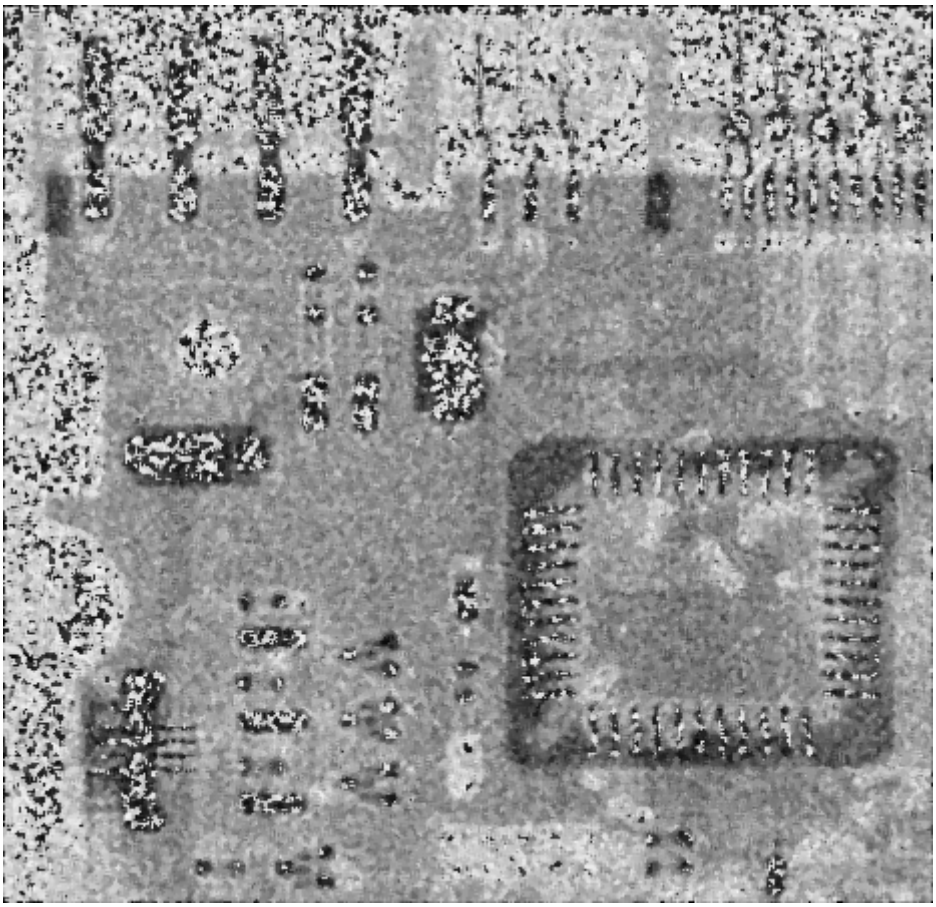
算术均值滤波处理：



几何均值滤波器处理：

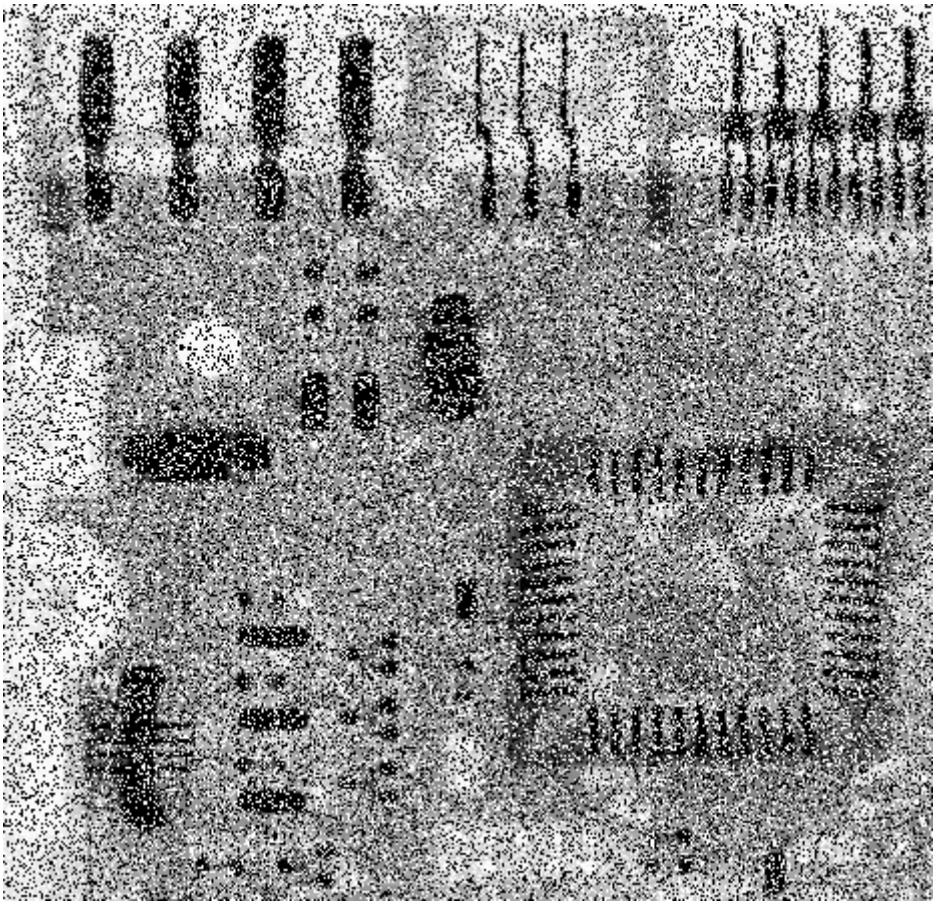


中值滤波器处理：

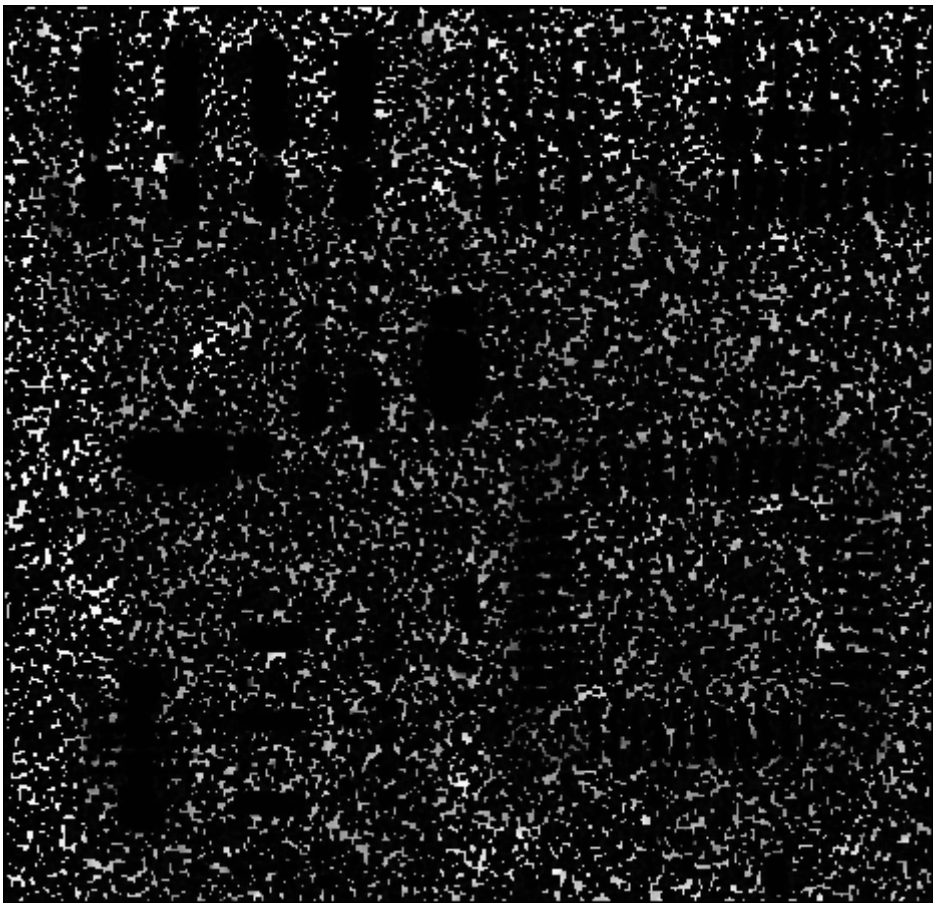


3.

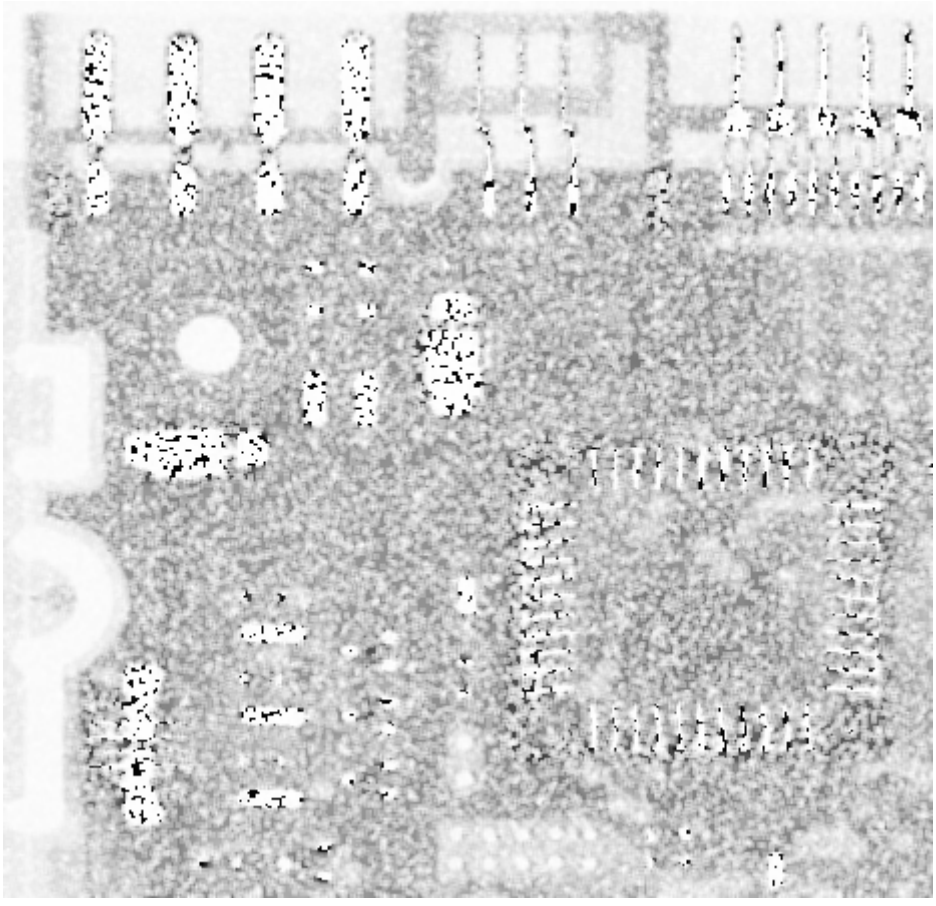
加上椒盐噪声



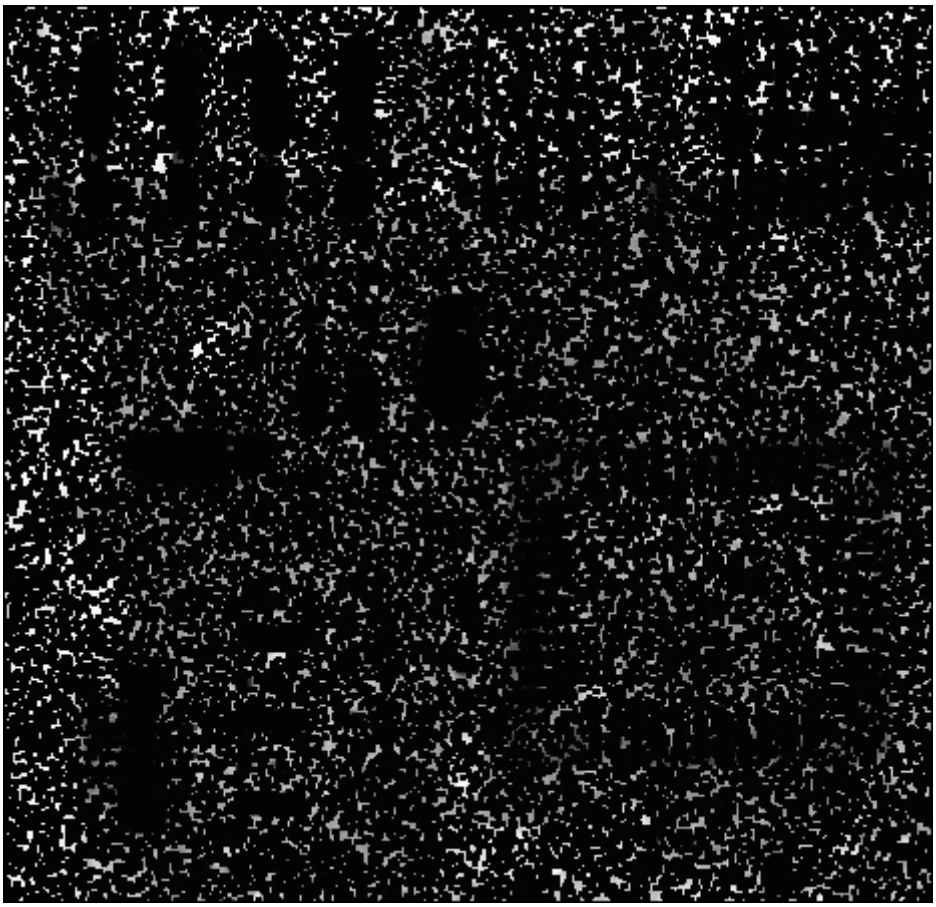
调和均值滤波



反谐波均值滤波，Q = 1.5



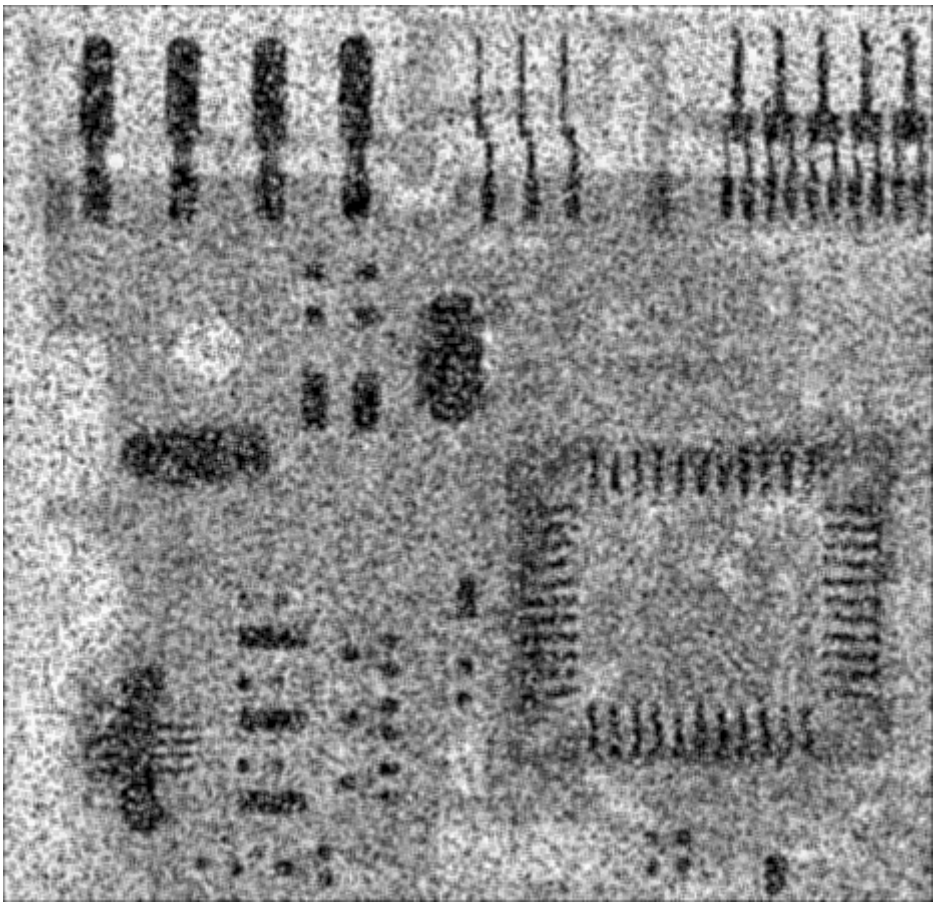
反谐波均值滤波，Q = -1.5



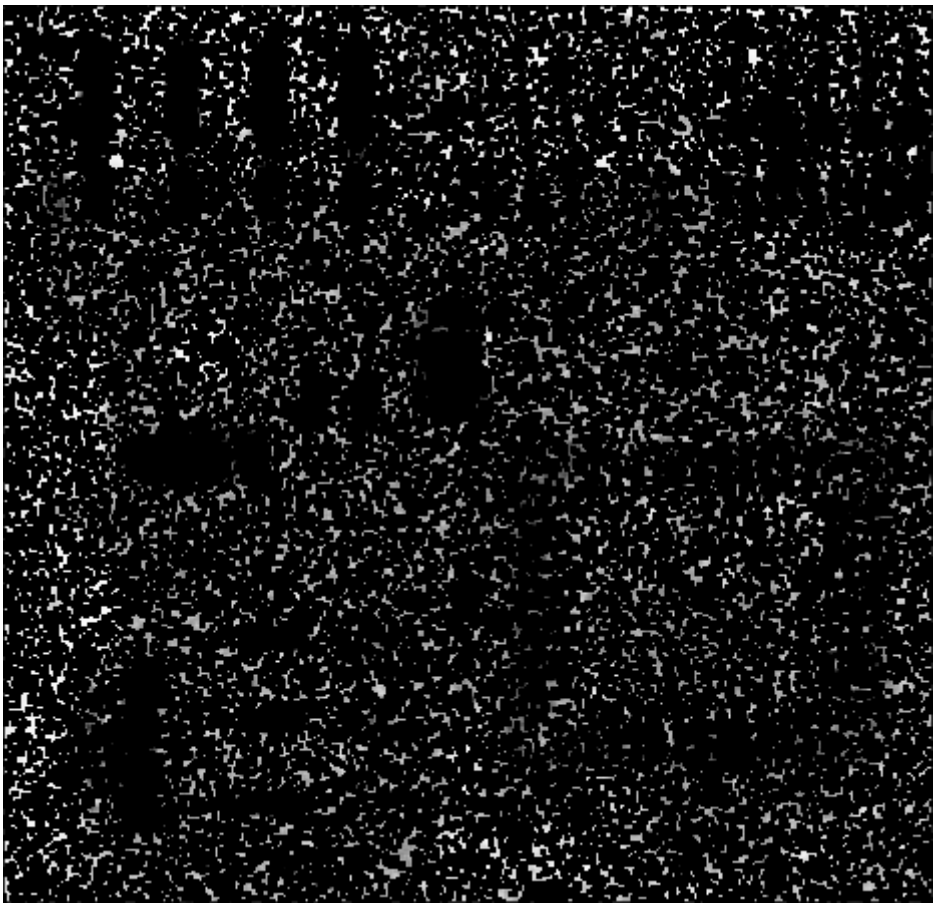
谐波滤波器适合处理椒盐噪声，但是必须实事先知道是椒噪声还是盐噪声，根据这个选择Q，Q>0适合处理椒噪声，Q<0适合处理盐噪声

4.

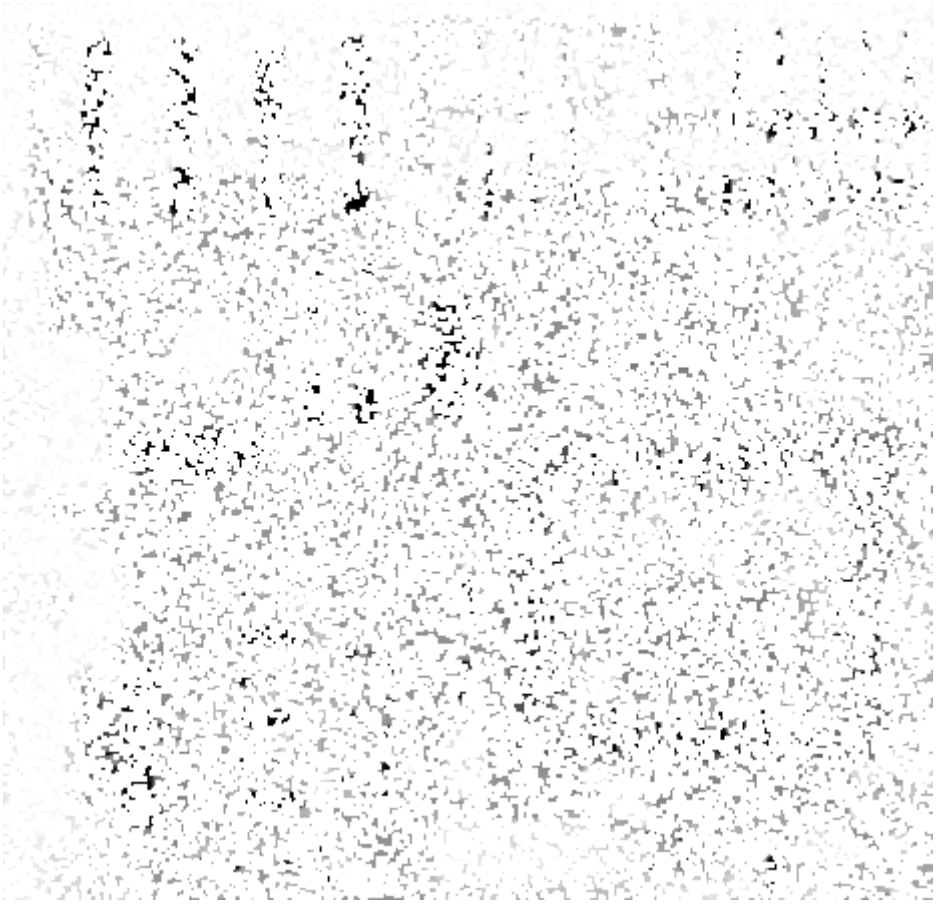
算术均值滤波



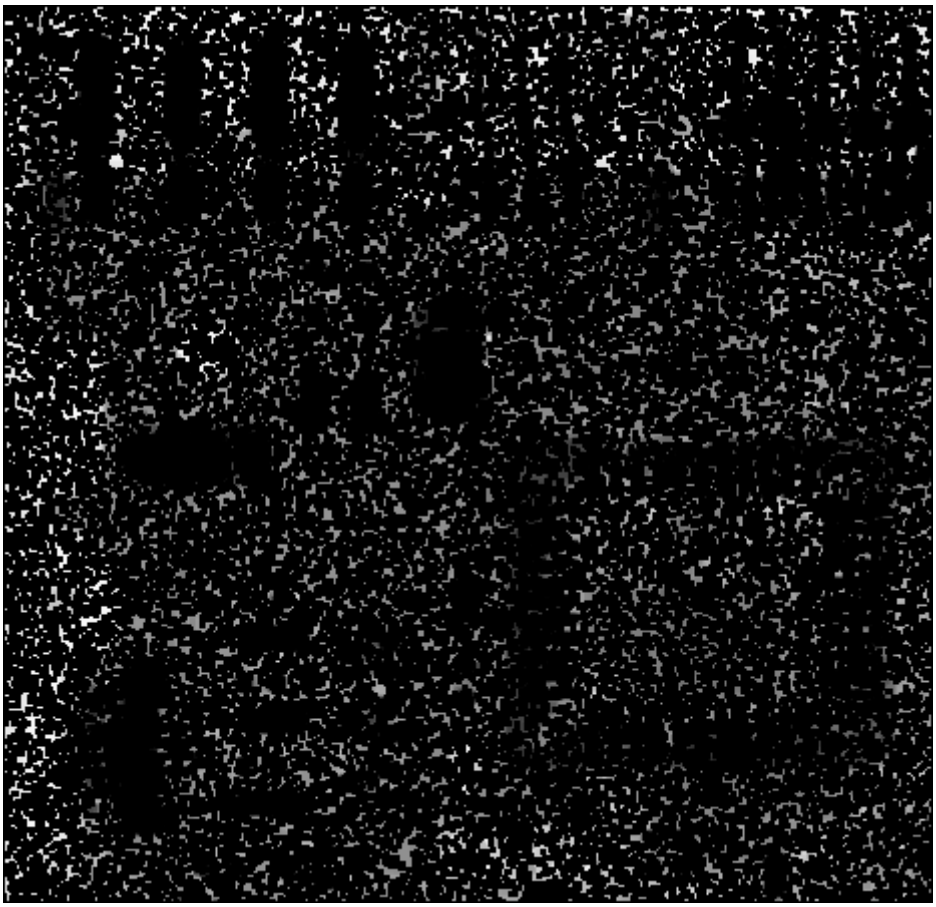
几何均值滤波



最大值滤波



最小值滤波



效果较好：算术均值滤波器；效果较差：几何均值滤波，最大最小值滤波

5.

> 算术均值滤波：算数均值滤波器就是一般全为1的再除以尺寸的矩阵，比如作业中我一直使用的算数均值滤波器如下：

```
arithmetic_mean_filter = numpy.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]]) /
 9
```

> 进行滤波操作，就是使用上几次作业自己写的卷积函数把图像的数组和这个算数均值滤波器进行卷积即可。
>
> 几何均值滤波：这个滤波器我是根据我之前写的卷积函数进行修改获得的，就是在之前卷积函数生成每点的卷积结果那里修改为获取那一个区域的几何均值

```
#几何均值滤波，边界补1
def geometric_mean_filters(arr, mask_h, mask_w):
    heigh, width = arr.shape
    result = numpy.zeros(shape = (heigh, width))
    max_y = int((mask_h - 1) / 2)
    max_x = int((mask_w - 1) / 2)
    #产生每一点的被滤波器覆盖区域的几何均值
    for i in range(heigh):
        for j in range(width):
            t_val = 1
            for m in range(mask_h):
                for n in range(mask_w):
                    val = 0
                    y_dis = int(m - max_y)
                    x_dis = int(n - max_x)
                    #判断是否越界
                    if i + y_dis < 0 or i + y_dis > heigh - 1 or j + x_di
s < 0 or j + x_dis > width - 1:
                        #越界则赋值此点值为1
                        val = 1
                    else:
                        #不越界则取那点值为原图像像素值的1/mask_h * mask_w方
                        val = math.pow(arr[i + y_dis][j + x_dis], 1 / (ma
sk_h * mask_w))
                    t_val = t_val * val
            result[i][j] = t_val
    return result.astype('int')
```

调和均值滤波：跟几何均值滤波器都是一样，都是修改我之前写的卷积函数而来，把卷积操作改为取那一点被滤波器覆盖区域的调和均值

反谐波均值滤波：跟几何均值滤波器都是一样，都是修改我之前写的卷积函数而来，把卷积操作改为取那一点被滤波器覆盖区域的反谐波均值

最大值滤波器：同上，把卷积操作改为取那一点被滤波器覆盖区域的最大值

最小值滤波器：同上，把卷积操作改为取那一点被滤波器覆盖区域的最小值

2.4

1.

2.

3.

4.

第一张是在每个色菜通道上做直方图均衡化，第二张是根据平均直方图均衡化前后的像素值映射关系应用到R、G、B三个通道，再重构一张RGB图，第三章是把图像转为hsi三通道，并对i通道进行直方图均衡化，所以不一样。