数字图像处理第二次作业

一、 加噪处理：

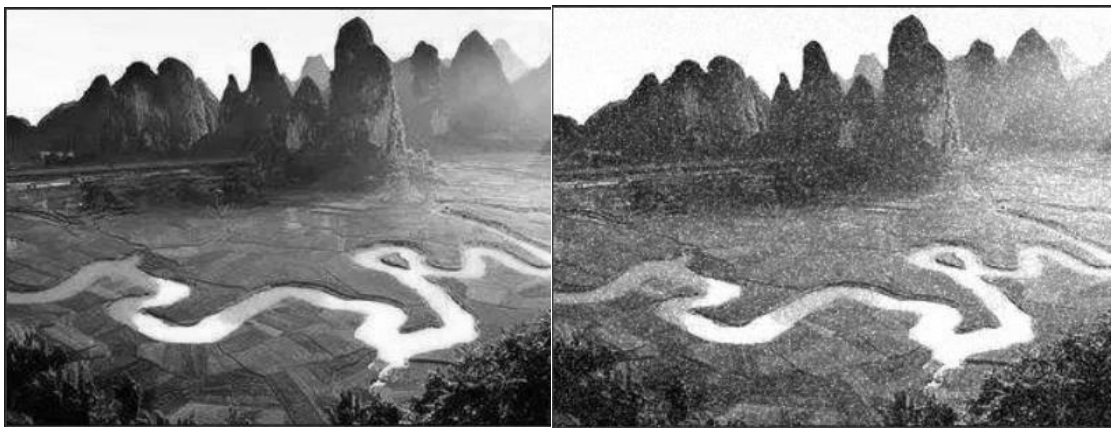
1．1 高斯噪声

①原理：使用 Numpy 库的 normal 函数，生成特定均值（means）和标准差（sigma）的、服从正态分布的随机噪声数组，最后将噪声随机加到图像矩阵中。

②关键代码：

```
'''
    高斯噪声
    means：均值
    sigma：标准差
    percetage：比例
'''
def addGaussNoise(img, means, sigma, percetage):
    noiseImg = img.copy().astype(np.int16)
    rows, cols = img.shape
    num = int(percetage*rows*cols)#添加噪声的数量
    noise = np.random.normal(means, sigma, num)
    for i in range(num):
        randX=random.randint(0, rows-1)
        randY=random.randint(0, cols-1)
        noiseImg[randX, randY]= noiseImg[randX,randY] + int(noise[i])#添加高斯噪声
        np.clip(noiseImg,0,255) #限制灰度值范围
    return noiseImg
```

③图像展示（原图与高斯噪声图）：



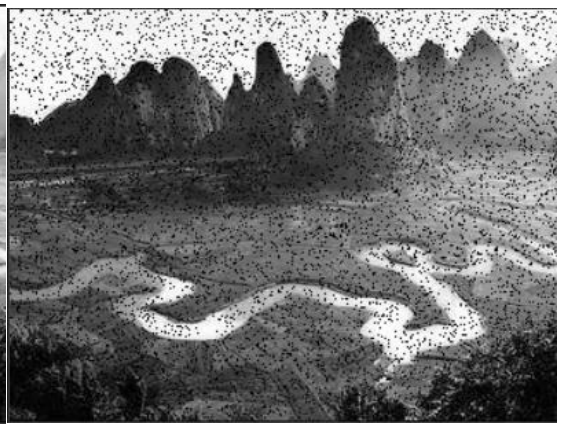
1．2 椒盐噪声

①原理：图像中部分点的灰度值改为 0，呈现为黑点，即椒噪声；图像中部分点的灰度值改为 255，呈现为白点，即盐噪声。

②关键代码：

```
'''
    添加椒噪声或盐噪声
    img -- 需要添加噪声的图像
    percetage -- 添加噪声的比例
    type -- 0 为盐噪声
            1 为椒噪声
            2 为椒盐噪声
'''
def addPepperOrSalt(img, percetage, type=0):
    rows = img.shape[0]
    cols = img.shape[1]
    noiseImg = img.copy()
    num = int(percetage * rows * cols) #添加噪声的数量
    #盐噪声
    if(type == 0):
        for i in range(num):
            randX = random.randint(0, rows-1)
            randY = random.randint(0, cols-1)
            noiseImg[randX, randY] = 255
    #椒噪声
    elif (type == 1) :
        for i in range(num):
            randX = random.randint(0, rows-1)
            randY = random.randint(0, cols-1)
            noiseImg[randX, randY] = 0
    #椒盐噪声
    elif (type == 2):
        for i in range(num):
            randX = random.randint(0, rows-1)
            randY = random.randint(0, cols-1)
            temp = random.random()
            if(temp>0.5):
                noiseImg[randX, randY] = 0
            else:
                noiseImg[randX,randY] = 255
    return noiseImg
```

③图像展示（原图像，椒噪声，盐噪声，椒盐噪声）：

1．3 均匀噪声

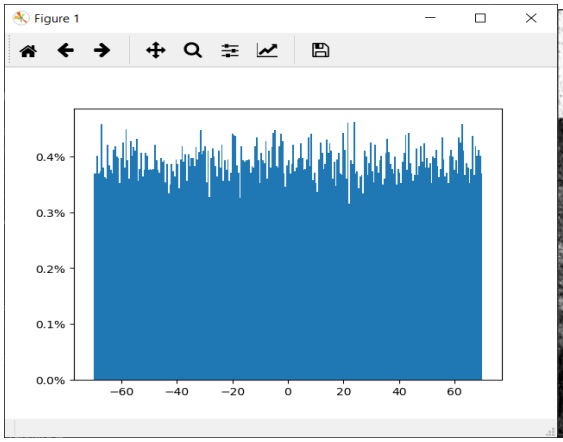
①原理：通过 Numpy 库的 uniform 函数生成下界为 low，上界为 high，服从均匀分布的随机数组。再将噪声数组加入到原图像中。

②关键代码：

```
'''
    添加均匀噪声
    low：下界
    high：上界
    percetage：比例
'''
def addAverageNoise(img, low, high, percetage=1):
    row,col = img.shape
    num = int(percetage * row * col)
    s = np.random.uniform(low,high,num) #均匀生成 num 个上界为 high，下界为 low 的随机数
    '''
        显示均匀分布的柱状图
    '''
    plt.hist(s,bins=256 ,weights= [1./ len(s)] * len(s))
    formatter = FuncFormatter(to_percent)
    plt.gca().yaxis.set_major_formatter(formatter)
    plt.show()

    '''
        将噪声添加到图像中
    '''
    noiseImg = img.copy().astype(np.int16)
    for i in range(num):
        randX = random.randint(0, row-1)
        randY = random.randint(0, col-1)
        noiseImg[randX, randY] = noiseImg[randX, randY] + int(s[i])
        np.clip(noiseImg,0,255)
    return noiseImg
```

③图像展示（噪声频率分布直方图，均匀加噪图像）：



二、 滤波器

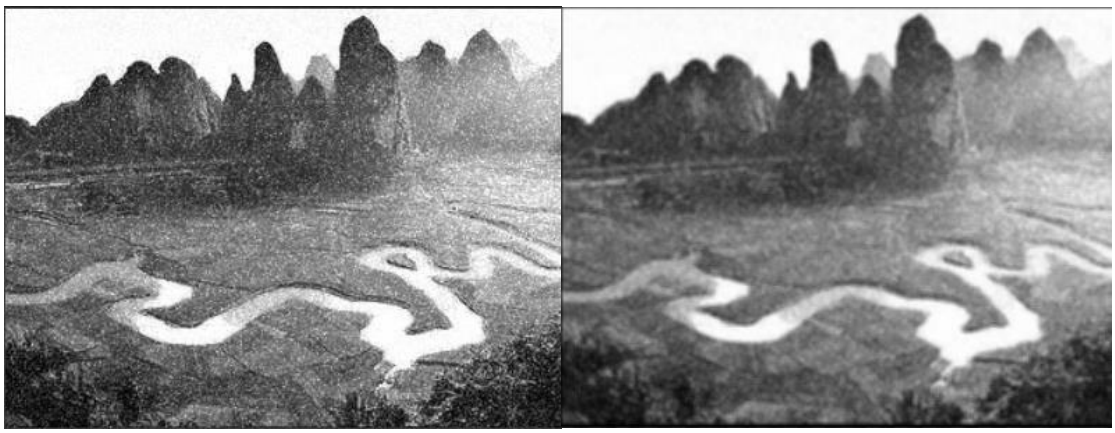
1. 1 均值滤波器

①原理：通过构造一个大小为 filter_size * filter_size 大小的滤波矩阵，扫描图像，将矩阵中的所有值相加，然后除以矩阵的大小，得到矩阵元素的算术平均值，然后使矩阵中心点的值等于算术平均值。

②关键代码：

```
'''
    均值滤波器
    filter_size -- 滤波器大小
'''
def MeansFilter(img, filter_size):
    tempImg1 = img.copy()
    pad_num = int((filter_size-1)/2)#中点距离边界的长度
    tempImg1 = np.pad(tempImg1, (pad_num, pad_num), mode="constant", co
nstant_values=0)
    tempImg2 = tempImg1.copy()
    m, n = tempImg1.shape
    for i in range(pad_num, m-pad_num):
        for j in range(pad_num, n-pad_num):
            #滤波矩阵中心点 = 滤波矩阵的算术平均值
            tempImg1[i,j] = np.sum(tempImg2[i-pad_num:i+1+pad_num,j-
pad_num:j+1+pad_num]) / (filter_size **2)
    return tempImg1
```

③图像展示（高斯噪声图，均值滤波去噪）：



1. 2 最小值滤波器

① **原理：**通过构造一个大小为 filter_size * filter_size 大小的滤波矩阵，扫描图像，使矩阵中心点的值等于滤波矩阵中的最小值，便于去除盐噪声（255）。

② **关键代码：**

```
'''
    最小值滤波器 -- 针对盐噪声
    filter_size -- 滤波矩阵大小
'''
def minFilter(img, filter_size):
    tempImg1 = img.copy().astype(np.int16)#避免溢出
    pad_num = int((filter_size-1)/2)#中点距离边界的长度
    tempImg1 = np.pad(tempImg1, (pad_num, pad_num), mode="constant", co
nstant_values=0)
    tempImg2 = tempImg1.copy()
    m, n = tempImg1.shape
    for i in range(pad_num+1, m-pad_num-1): #此处加 1 减 1 是为了防止 0 对结
果的影响
        for j in range(pad_num+1, n-pad_num-1):
            data_matrix = tempImg2[i-pad_num:i+pad_num+1, j-
pad_num:j+pad_num+1]
            tempImg1[i,j] = np.min(data_matrix)#中心值 = 滤波矩阵最小值
    return tempImg1
```

③ **图像展示（盐噪声图像，最小值滤波去噪）：**

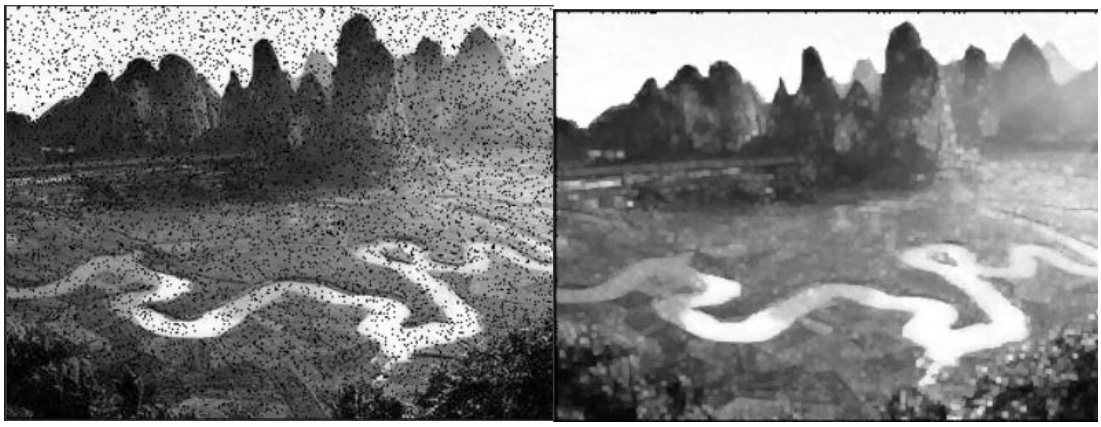
1. 3 最大值滤波器

① **原理：**通过构造一个大小为 filter_size * filter_size 大小的滤波矩阵，扫描图像，使矩阵中心点的值等于滤波矩阵中的最大值，便于去除椒噪声（0）。

② **关键代码：**

```
'''
    最大值滤波器 -- 针对椒噪声
    filter_size -- 滤波器大小
'''
def maxFilter(img, filter_size):
    tempImg1 = img.copy().astype(np.int16)
    filterMat = np.ones((filter_size, filter_size)) #滤波器模板
    pad_num = int((filter_size-1)/2)#中点距离边界的长度
    tempImg1 = np.pad(tempImg1, (pad_num, pad_num), mode="constant", co
nstant_values=0)
    tempImg2 = tempImg1.copy()
    m, n = tempImg1.shape
    for i in range(pad_num+1, m-pad_num-1): #此处加 1 减 1 是为了防止 0 对结
果的影响
        for j in range(pad_num+1, n-pad_num-1):
            data_matrix = tempImg2[i-pad_num:i+pad_num+1, j-
pad_num:j+pad_num+1]
            tempImg1[i,j] = np.max(data_matrix)
    return tempImg1
```

③ **图像展示(椒噪声，最大值滤波器去噪)：**



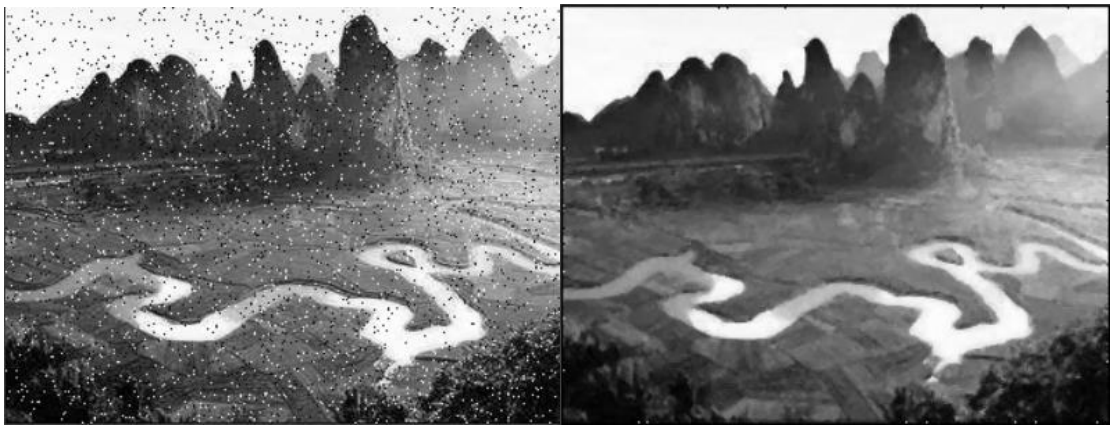
1. 4 中值滤波器

①原理：通过构造一个大小为 filter_size * filter_size 大小的滤波矩阵，扫描图像，使矩阵中心点的值等于滤波矩阵中的中值，便于去除椒盐噪声。

②关键代码：

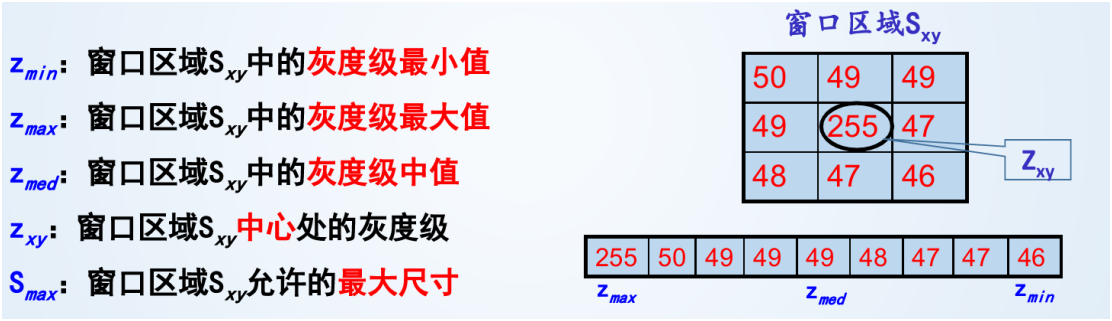
```
'''
    中值滤波器 -- 针对椒盐噪声
    filter_size -- 滤波器大小
'''
def medianFilter(img, filter_size):
    tempImg1 = tempImg2 = img.copy().astype(np.int16)
    pad_num = int((filter_size-1)/2)#中点距离边界的长度
    tempImg1 = np.pad(tempImg1, (pad_num, pad_num), mode="constant", co
nstant_values=0)
    m, n = tempImg1.shape
    for i in range(pad_num+1, m-pad_num-1): #此处加 1 减 1 是为了防止 0 对结
果的影响
        for j in range(pad_num+1, n-pad_num-1):
            data_matrix = tempImg2[i-pad_num:i+pad_num+1, j-
pad_num:j+pad_num+1]
            tempImg1[i,j] = np.median(data_matrix) #取中值
    return tempImg1
```

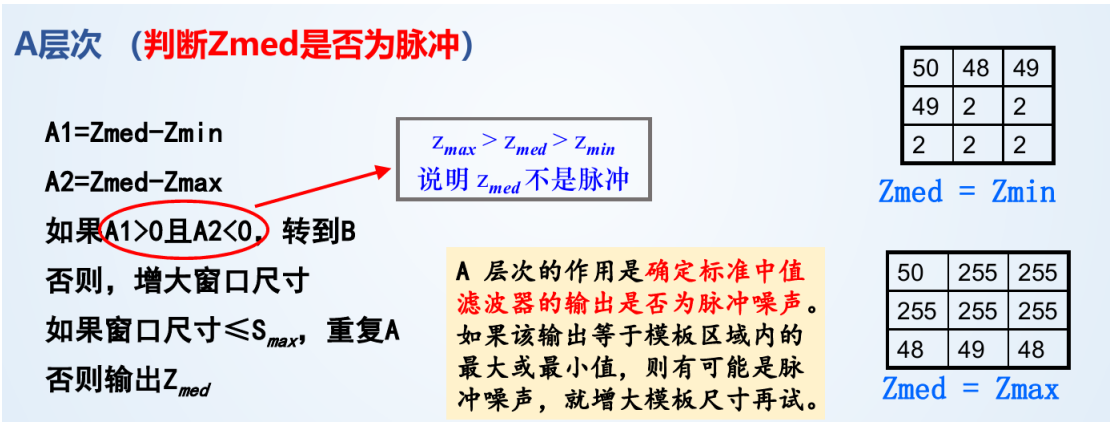
③图像展示（椒盐噪声，中值滤波器去噪）：

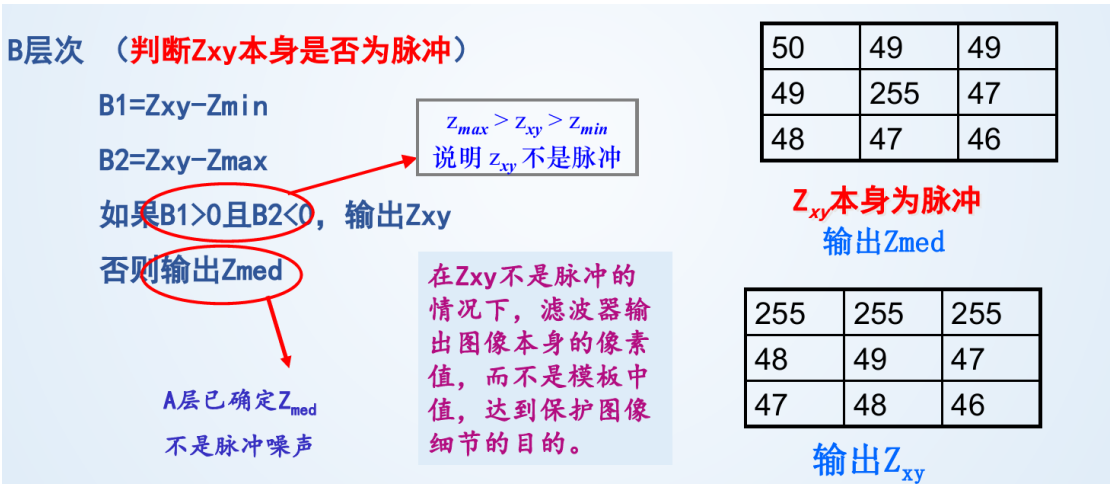


1. 5 自适应中值滤波器

①**原理：**与中值滤波器不同的是，自适应中值滤波器工作在 A,B 两个层次，A 层次判断 Zmed 是否为噪声，B 层次在确定了 Zmed 不是噪声的前提下，判断 Zxy 是否为脉冲噪声。







②关键代码：

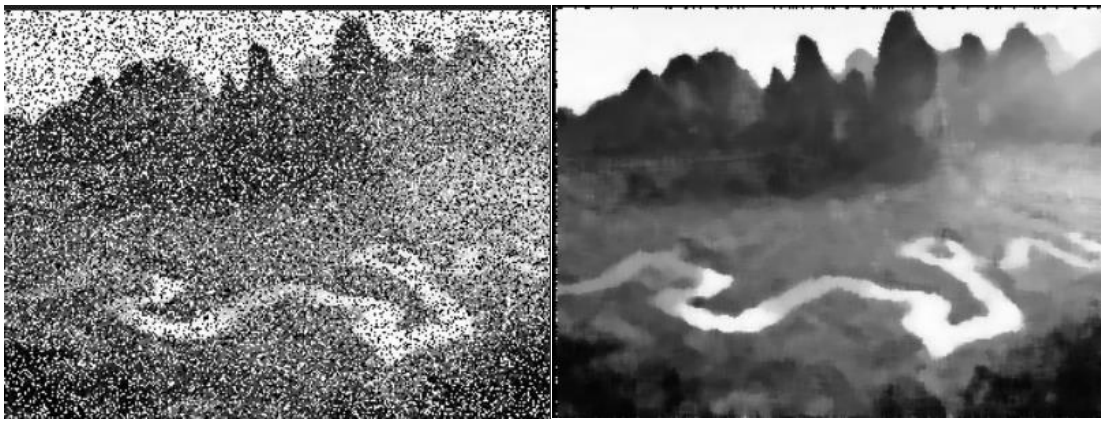
```
'''
    自适应中值滤波器 -- 针对椒盐噪声
    filter_size     -- 初始滤波尺寸
    Smax            -- 窗口允许的最大尺寸
'''
def autoMedianFilter(img, filter_size, Smax):
    tempImg1 = tempImg2 = img.copy().astype(np.int16)
    pad_num = int((filter_size-1)/2)#中点距离边界的长度
    pad_max = int((Smax - 1) /2)
    tempImg1 = np.pad(tempImg1, (pad_num, pad_num), mode="constant", co
nstant_values=0)
    m, n = tempImg1.shape
    for i in range(pad_max+1, m-pad_max-1): #此处加 Smax 减 Smax 是为了防止
扩大窗口尺寸造成滤波器溢出图像
        for j in range(pad_max+1, n-pad_max-1):
            tempImg1[i,j] = A(filter_size,Smax,tempImg2, i, j)
    return tempImg1

# A 过程
def A(filter_size,Smax,tempImg2,x,y):
    ans = 0
    while(filter_size <= Smax):
        pad_num = int((filter_size-1)/2)
        data_matrix = tempImg2[x-pad_num:x+pad_num+1, y-
pad_num:y+pad_num+1]
        A1 = np.median(data_matrix)-np.min(data_matrix)
        A2 = np.median(data_matrix)-np.max(data_matrix)
        if(A1 > 0 and A2 < 0):
            return B(data_matrix, pad_num, pad_num)
        else:
            filter_size = filter_size + 2
            ans = np.median(data_matrix)
    return ans




# B 过程
def B(data_matrix,x,y):
    B1 = data_matrix[x,y] - np.min(data_matrix) # B1 = Zxy-min
    B2 = data_matrix[x,y] - np.max(data_matrix) # B2 = Zxy-max
    if(B1>0 and B2<0):
        return data_matrix[x,y]
    else:
        return np.median(data_matrix)
```

③图像展示（重度椒盐噪声 Pa=Pb=0.25， 中值滤波处理 7*7）：



可以看出，中值滤波器对于高密度的椒盐噪声，虽然能够去噪，但是图像失真率也提高了，所以采用自适应中值滤波（Smax = 7），保留图像细节。