图像去噪上机实验报告

蒋颜丞，自动化(电气)1903，3190102563

1 实验内容与要求

任意选取1张图像，添加高斯噪声和椒盐噪声。实现均值滤波、中值滤波和双边滤波等去噪方法，比较各方法去噪结果，并分析各去噪方法的特点。

本次实验将以下图为例，进行加噪和去噪。

2 实验原理

2.1 加噪

- 高斯噪声的概率密度函数服从高斯分布，在加噪时，只需按照高斯分布生成噪声即可。
- 椒盐噪声指在一幅图像里随机将一个像素点变为椒噪声或盐噪声，其中椒噪声像素值为“0”，盐噪声像素值为“255”，在加噪时，只需利用随机数函数随机加噪即可。

2.2 均值滤波

均值滤波的思想是通过一点和邻域内像素点求平均来去除突变的像素点，从而滤掉一定的噪声，其主要优点是算法简单，计算速度快， 但其代价是会造成图像一定程度上的模糊。

均值滤波可用下式表示:

$$g(x,y)=\frac{1}{M}\sum_{(i,j)\in S}f(i,j)$$

式中：$x,y=0,1,...,N\text{-}1$；S是以(x,y)为中心的邻域的集合，M是S内的点数。

2.3 中值滤波

中值滤波就是用一个奇数点的移动窗口，将窗口中心点的值用窗口内各点的中值代替。

二维中值滤波可由下式表示：

$$y_{ij} = \underset{A}{Med}\{f_{ij}\}$$

式中：A为窗口；$\{f_{ij}\}$为二维数据序列。

二维中值滤波的窗口形状和尺寸对滤波效果影响较大，不同的图像内容和不同的应用要求，往往采用不同的窗口形状和尺寸。常用的二维中值滤波窗口有线状、方形、圆形、十字形以及圆环形等。窗口尺寸一般先用3×3，再取5×5逐渐增大，直到滤波效果满意为止。

2.4 双边滤波

双边滤波是在高斯滤波的基础上加入了像素值权重项，即既要考虑距离因素，也要考虑像素值差异的影响，像素值越相近，权重越大。将像素值权重表示为G_r，空间距离权重表示为G_s，有：

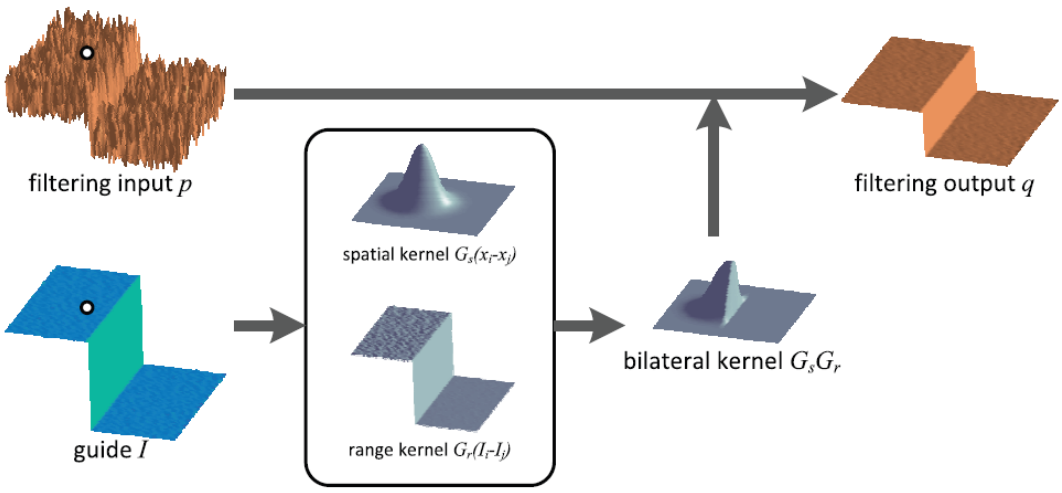
$$G_s = exp(-\frac{\|p-q\|^2}{2\sigma_s^2}), G_r = exp(-\frac{\|I_p - I_q\|^2}{2\sigma_r^2})$$

将滤波器表示为BF，则滤波结果为：

$$BF = \frac{1}{W_q}\sum_{p\in S} G_s(p)G_r(p) * I_p$$

$$= \frac{1}{W_q}\sum_{p\in S} exp(-\frac{\|p-q\|^2}{2\sigma_s^2})exp(-\frac{\|I_p - I_q\|^2}{2\sigma_r^2}) * I_p$$

在平坦区域，滤波器中每个像素点的G_r值相近，空间距离权重G_s主导滤波效果。在边缘区域，边缘同侧的G_r值相近，且远大于边缘另一侧的G_r值，此时另一侧的像素点的权重对滤波结果几乎不影响，边缘信息得到保护。表现出了一定的自适应性。



3 源代码

```
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt

def add_noise_Guass(img, mean=0, var=30**2):
    '''
    添加高斯噪声
    输入：灰度图像，高斯噪声均值，高斯噪声方差
    输出：添加高斯噪声后的图像
    '''
    noise = np.random.normal(mean, var ** 0.5, img.shape) # 生成噪声
    img = img + noise # 加入噪声
    img_guass = np.clip(img, 0, 255) # 防止值超限
    img_guass = np.uint8(img_guass)
    return img_guass

def add_noise_SP(img, SNR=0.6):
    '''
    添加椒盐噪声
    输入：灰度图像，信噪比（0~1之间）
    输出：添加椒盐噪声后的图像
    '''
    SP = int((img.shape[0]*img.shape[1])*(1-SNR)) # 计算椒盐噪声个数
    for i in range(SP):
        randx=np.random.randint(1,img.shape[0]-1)   # 生成一个 1 至 hight-1
之间的随机整数
        randy=np.random.randint(1,img.shape[1]-1)   # 生成一个 1 至 width-1
之间的随机整数
        if np.random.random()<=0.5:   # np.random.random()生成一个 0 至 1 之
间的浮点数
            img[randx,randy]=0
        else:
            img[randx,randy]=255
    return img

def mean_Filter(img, size=(5,5)):
    '''
    均值滤波
    输入：灰度图像，模板尺寸
    输出：均值滤波后的图像
    '''
    kernal = np.ones(size, np.float32)/size[0]/size[1] # 模板
    img_results = img.copy()
    h = img.shape[0]
    w = img.shape[1]
```

```
    for x in range(int((size[0]-1)/2),int(h-(size[0]-1)/2)):
        for y in range(int((size[1]-1)/2),int(w-(size[1]-1)/2)):
            window = img[int(x-(size[0]-1)/2):int(x+(size[0]-1)/2+1),int(y-(size[1]-1)/2):int(y+(size[1]-1)/2+1)] # 当前操作窗口
            img_results[x][y] = np.sum(window*kernal) # 对各点进行模板操作
    return img_results

def mid_Filter(img, size=(5,5)):
    '''
    中值滤波
    输入：灰度图像，模板尺寸
    输出：中值滤波后的图像
    '''
    img_results = img.copy()
    h = img.shape[0]
    w = img.shape[1]
    for x in range(int((size[0]-1)/2),int(h-(size[0]-1)/2)):
        for y in range(int((size[1]-1)/2),int(w-(size[1]-1)/2)):
            window = img[int(x-(size[0]-1)/2):int(x+(size[0]-1)/2+1),int(y-(size[1]-1)/2):int(y+(size[1]-1)/2+1)] # 当前操作窗口
            img_results[x][y] = np.median(window) # 对各点进行模板操作
    return img_results

def bf_Filter(img, size=(5,5), sigma_s = 10, sigma_r = 10):
    '''
    双边滤波
    输入：灰度图像，模板尺寸，空间域方差，像素值域方差
    输出：双边滤波后的图像
    '''
    img_results = img.copy()
    kernal = np.ones(size, np.float32) # 模板初始化
    h = img.shape[0]
    w = img.shape[1]
    for x in range(int((size[0]-1)/2),int(h-(size[0]-1)/2)):
        for y in range(int((size[1]-1)/2),int(w-(size[1]-1)/2)):
            window = img[int(x-(size[0]-1)/2):int(x+(size[0]-1)/2+1),int(y-(size[1]-1)/2):int(y+(size[1]-1)/2+1)] # 当前操作窗口
            for i in range(window.shape[0]):
                for j in range(window.shape[1]):
                    dist = ((i-(window.shape[0]-1)/2)**2+(j-(window.shape[0]-1)/2)**2)/2/sigma_s
                    dValue = ((float(window[i][j]) - float(window[int((window.shape[0]-1)/2)][int((window.shape[1]-1)/2)]))**2)/2/sigma_r
                    kernal[i][j] = math.exp(-dist)*math.exp(-dValue) # 计算模板权重
            kernal /= np.sum(kernal) # 权重归一化
            img_results[x][y] = np.sum(window*kernal) # 对各点进行模板操作
    return img_results
```

```python

if __name__ == '__main__':

    # 读取原始图像
    img = cv2.imread('GrayPhoto.jpg', cv2.IMREAD_GRAYSCALE)
    # cv2.imshow("img", img)

    # 对原图像添加高斯噪声
    img_guass = add_noise_Guass(img, 0, 30**2)
    cv2.imshow("img_guass", img_guass)
    cv2.imwrite('img_guass.jpg', img_guass)

    # 对原图像添加椒盐噪声
    img_SP = add_noise_SP(img, 0.92)
    cv2.imshow("img_SP", img_SP)
    cv2.imwrite('img_SP.jpg', img_SP)

    # 均值滤波
    img_mean_guass_cv = cv2.blur(img_guass, (5,5))
    cv2.imshow("img_mean_guass_cv", img_mean_guass_cv)
    cv2.imwrite("img_mean_guass_cv.jpg", img_mean_guass_cv)
    img_mean_guass = mean_Filter(img_guass, (5,5))
    cv2.imshow("img_mean_guass", img_mean_guass)
    cv2.imwrite("img_mean_guass.jpg", img_mean_guass)

    img_mean_SP_cv = cv2.blur(img_SP, (5,5))
    cv2.imshow("img_mean_SP_cv", img_mean_SP_cv)
    cv2.imwrite("img_mean_SP_cv.jpg", img_mean_SP_cv)
    img_mean_SP = mean_Filter(img_SP, (5,5))
    cv2.imshow("img_mean_SP", img_mean_SP)
    cv2.imwrite("img_mean_SP.jpg", img_mean_SP)


    # 中值滤波
    img_mid_guass_cv=cv2.medianBlur(img_guass,5)
    cv2.imshow("img_mid_guass_cv", img_mid_guass_cv)
    cv2.imwrite("img_mid_guass_cv.jpg", img_mid_guass_cv)
    img_mid_guass=mid_Filter(img_guass,(5,5) )
    cv2.imshow("img_mid_guass", img_mid_guass)
    cv2.imwrite("img_mid_guass.jpg", img_mid_guass)

    img_mid_SP_cv=cv2.medianBlur(img_SP,5)
    cv2.imshow("img_mid_SP_cv", img_mid_SP_cv)
    cv2.imwrite("img_mid_SP_cv.jpg", img_mid_SP_cv)
    img_mid_SP=mid_Filter(img_SP,(5,5) )
    cv2.imshow("img_mid_SP", img_mid_SP)
    cv2.imwrite("img_mid_SP.jpg", img_mid_SP)


```

```
    # 双边滤波
    img_bf_guass=bf_Filter(img_guass,(5,5),5000**2,150**2)
    cv2.imshow("img_bf_guass", img_bf_guass)
    cv2.imwrite("img_bf_guass.jpg", img_bf_guass)
    img_bf_guass_cv=cv2.bilateralFilter(img_guass,5,100,15)
    cv2.imshow("img_bf_guass_cv", img_bf_guass_cv)
    cv2.imwrite("img_bf_guass_cv.jpg", img_bf_guass_cv)

    img_bf_SP=bf_Filter(img_SP,(5,5),5000**2,150**2)
    cv2.imshow("img_bf_SP", img_bf_SP)
    cv2.imwrite("img_bf_SP.jpg", img_bf_SP)
    img_bf_SP_cv=cv2.bilateralFilter(img_SP,5,100,15)
    cv2.imshow("img_bf_SP_cv", img_bf_SP_cv)
    cv2.imwrite("img_bf_SP_cv.jpg", img_bf_SP_cv)

    K = cv2.waitKey(0)
```

4 实验结果与分析

4.1 实验结果

滤波方法	高斯噪声	椒盐噪声
未滤波		
5×5均值滤波(自实现)		

滤波方法	高斯噪声	椒盐噪声
5×5均值滤波(openCV)		
5×5中值滤波(自实现)		
5×5中值滤波(openCV)		
5×5双边滤波(自实现)		

滤波方法	高斯噪声	椒盐噪声
5×5双边滤波 (openCV)		

4.2 比较分析

4.2.1 均值滤波

从图中可以看到，均值滤波对高斯噪声和椒盐噪声均有一定程度的抑制，但不能完全去除，且对椒盐噪声的去除程度要略小于高斯噪声。与此同时，均值滤波不能很好地保护图像细节，在给图像去噪的同时也破坏了图像细节部分，丢失图像特征信息，使图像变得模糊。

因此，均值滤波多用于具有大面积色块的图像去噪。

4.2.2 中值滤波

从图中可以看到，中值滤波对高斯噪声和椒盐噪声均有较大程度的抑制，但不能完全去除，且对椒盐噪声的去除程度要远大于高斯噪声。处理后图像的模糊程度相比均值滤波有所改善，但仍有涂抹感（油画感）。

二维中值滤波的窗口形状和尺寸对滤波效果影响较大，不同的图像内容和不同的应用要求，往往采用不同的窗口形状和尺寸。就一般经验来讲，对于有缓变的较长轮廓线物体的图像，采用方形或圆形窗口为宜。对于包含有尖顶物体的图像， 用十字形窗口， 而窗口大小则以不超过图像中最小有效物体的尺寸为宜。如果图像中点、线、尖角细节较多，则不宜采用中值滤波。

4.2.3 双边滤波

从图中可以看到，双边滤波对高斯噪声和椒盐噪声均有较大程度的抑制，但不能完全去除，虽然图像轮廓及边缘得到很好的保留，但涂抹感（油画感）仍然存在，丢失了一些细节。当然，对双边滤波而言，滤波的效果与参数的选择有很大关系。

对于两个σ值而言，其值越大，说明权重差别越小，表示不强调这一因素的影响，反之，则表示更强调这一因素导致的权重的不均衡。σ_s表示的是空域的平滑，因此，对于没有边缘的、变化慢的部分更适合；σ_r表示的是值域的差别，因此，减小σ_r可以突出边缘，但同时也会减弱去噪效果。

σ_s较小时，表示更多采用近邻的值作平滑，说明图像的空间信息更重要，即越相近的越相似。σ_s较大时，表示图像每个区域的权重基本都源于值域滤波的权重，对于空间邻域信息不是很敏感。

σ_r较小时，表示更强调值域的相似性，对于值域差距很大的点，其平滑权重会很小。σ_r较大时，则不太考虑值域的相似性，权重多来自于空间距离，近似于普通的高斯滤波，图像的保边性能下降。

综上所述，如果想更多的去除平滑区域的噪声，应该提高σ_s，如果像保持边缘，则应该减小σ_r。