数字图像处理第四次作业

浮焕然 PB22061345

问题：

1. 为什么一般情况下对离散图象的直方图均衡并不能产生完全平坦的直方图？
   灰度级是离散的，同一灰度级的像素不能映射到不同的灰度级上

2. 设已用直方图均衡化技术对一幅图象进行了增强，试证明再用这个方法对所得结果增强并不会改变其结果
   直方图均衡化所用的变换函数为原始图像的累积直方图，均衡化后得到的增强图像的累积直方图除有些项合并外，其余项与原始图像的累积直方图相同。再次进行均衡化，所用的是均衡化后的增强图像的累积直方图(并且不会有新的合并项)，所以不会改变其结果

3. 讨论用于空间滤波的平滑滤波器和锐化滤波器的相同点、不同点以及联系。
   相同点：都是通过减弱或消除信号傅里叶空间的某些分量来达到增强图像的效果；实现方法类似。
   不同点：平滑滤波器是减弱或消除高频分量，增强低频，平滑图像；锐化滤波器减弱或消除低频分量，增强高频，锐化图像。
   联系：两者效果相反，从原始图像中减去平滑滤波器的结果得到锐化滤波器的效果，从原始图像中减去锐化滤波器的结果得到平滑滤波器的效果。

4. 有一种常用的图象增强技术是将高频增强和直方图均衡化结合起来以达到使边缘锐化的反差增强效果，以上两个操作的先后次序对增强结果有影响吗？为什么？
   调换操作的先后次序会有影响。
   原因如下：
   高频增强是一种线性操作，直方图均衡是一种非线性操作，线性操作与非线性操作不可互换次序。
   先做直方图均衡，会使得像素点的灰度值尽可能达到均匀分布，这样再做高频增强就更容易使边缘得到突出。（因为在均衡化过程中，像素值间差别在统计意义上增大了）。
   对于一幅几乎没有高频分量的图像，先做高频增强，基本上还是原图，再进行直方图均衡，则会增强对比度；若先进行直方图均衡，则会人为的增加高频分量，再进行高频增强，会使得高频分量进一步增强。

5. 编程实现对 lena.bmp 分别加入高斯噪声和椒盐噪声，再进行局域平均和中值滤波。

图 1.高斯噪声添加结果-局域平均-中值滤波

图 2.椒盐噪声添加结果-局域平均-中值滤波

第五题代码如下（省略了图片读取、显示、保存部分）：

```
# 添加高斯噪声
def add_gaussian_noise(image, mean=0, std=5):
    gaussian_noise = np.random.normal(mean, std,
image.shape).astype(np.uint8)
    output = np.zeros(image.shape, np.uint8)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            temp = int(gaussian_noise[i][j])+image[i][j]
            if temp<0:
                output[i][j] = 0
            elif temp>255:
                output[i][j] = 255
            else:
                output[i][j] = temp
    return output

# 添加椒盐噪声
def add_salt_and_pepper_noise(image, ratio):
    output = np.zeros(image.shape, np.uint8)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rand = random.random()
            if rand < ratio:  # salt pepper noise
                if random.random() > 0.5:  # change the pixel to 255
                    output[i][j] = 255
                else:
                    output[i][j] = 0
            else:
                output[i][j] = image[i][j]
    return output

# 局域平均滤波
def LocalAverageFilter(imarray, k=3):
    height, width = imarray.shape[:2]
    edge = int((k-1)/2)
    output = np.zeros((height, width), dtype="uint8")

    for i in range(height):
        for j in range(width):
            if i <= edge - 1 or i >= height - 1 - edge or j <= edge - 1
or j >= width - edge - 1:
                output[i, j] = imarray[i, j]
            else:
```

```
                window = imarray[i - edge:i + edge + 1, j - edge:j +
edge + 1]
                output[i, j] = np.mean(window)

    return output
# 中值滤波
def MedianFilter(imarray, k=3):
    height, width = imarray.shape[:2]
    edge = int((k-1)/2)
    output = np.zeros((height, width), dtype="uint8")
    for i in range(height):
        for j in range(width):
            if i <= edge - 1 or i >= height - 1 - edge or j <= edge - 1
or j >= width - edge - 1:
                output[i, j] = imarray[i, j]
            else:
                output[i, j] = np.median(imarray[i - edge:i + edge + 1,
j - edge:j + edge + 1])
    return output

# 添加高斯噪声
gaussian_noisy_image = add_gaussian_noise(image,0,1)
# 添加椒盐噪声
salt_pepper_noisy_image = add_salt_and_pepper_noise(image,0.1)
# 对高斯噪声图像进行局域平均滤波和中值滤波
gaussian_mean_filtered = LocalAverageFilter(gaussian_noisy_image)
gaussian_median_filtered = MedianFilter(gaussian_noisy_image)
# 对椒盐噪声图像进行局域平均滤波和中值滤波
salt_pepper_mean_filtered = LocalAverageFilter(salt_pepper_noisy_image)
salt_pepper_median_filtered = MedianFilter(salt_pepper_noisy_image)
```