**程序报告**

学校：复旦大学 学号：19307130303 姓名：于昕卉

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

在本次实验中，我们要对图像添加噪声，并对添加噪声的图像进行基于模型的去噪。

**1. 生成受损图像：**受损图像（X）是由原始图像添加了不同噪声遮罩得到的，噪声遮罩仅包含 {0,1} 值，对原图的噪声遮罩的可以用噪声比率产生的。

**2. 使用中值滤波算法模型，进行图像恢复**。中值滤波是一种非线性平滑技术，它将每一像素点的灰度值设置为该点某临域窗口内的所有像素点灰度值的中值。

**3. 评估误差**为所有恢复图像与原始图像的 2-范数之和，此误差越小越好，其他评估方式包括 Cosine 相似度以及 SSIM 相似度。

1. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

传统的中值滤波算法在椒盐噪声的去除领域有着比较广泛的应用，其具有较强的噪点鉴别和恢复能力，也有比较低的时间复杂度：其基本思想是采用像素点周围邻接的若干像素点的中值来代替被污染的像素点；但也存在一定的缺陷，随着图像被污染程度的加深，此方法恢复的图像细节模糊、边缘损失也会越严重。

中值滤波的思想就是比较一定领域内的像素值的大小，取出其中值作为这个领域的中心像素新的值。假设对一定领域内的所有像素从小到大进行排序，如果存在孤立的噪声点，比如椒盐噪声（椒噪声——较小的灰度值，呈现的效果是小黑点；盐噪声——较大的灰度值，呈现的效果是小白点），那么从小到大排序的这个数组中，那些孤立的噪声一定会分布在两边（要么很小，要么很大），这样子取出的中值点可以很好地保留像素信息，而滤除了噪声点的影响。

中值滤波器受滤波窗口大小影响较大，用于消除噪声和保护图像细节，两者会存在冲突。如果窗口较小，则能较好地保护图像中的一些细节信息，但对噪声的过滤效果就会打折扣；反之，如果窗口尺寸较大则会有较好的噪声过滤效果，但也会对图像造成一定的模糊效果，从而丢失一部分细节信息。

本代码采用改进的自适应中值滤波算法进行图像恢复：

**1.根据图像处理的空间相关性原则，采用自适应的方法选择不同的滑动窗口大小；**

**2.在算法中单滤波窗口大小达到最大值时，采用均值滤波；**

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

====================================================================

**def** noise\_mask\_image(img, noise\_ratio=[0.8,0.4,0.6]):  
 *"""  
 根据题目要求生成受损图片* **:param** *img: cv2 读取图片,而且通道数顺序为 RGB* **:param** *noise\_ratio: 噪声比率，类型是 List,，内容:[r 上的噪声比率,g 上的噪声比率,b 上的噪声比率]  
 默认值分别是 [0.8,0.4,0.6]* **:return***: noise\_img 受损图片, 图像矩阵值 0-1 之间，数据类型为 np.array,  
 数据类型对象 (dtype): np.double, 图像形状:(height,width,channel),通道(channel) 顺序为RGB  
 """  
 # 受损图片初始化* noise\_img = **None** *# -------------实现受损图像答题区域-----------------* **import** random  
 noise\_img = np.copy(img)  
 **for** i **in** range(3):  
 **for** j **in** range(img.shape[0]):  
 mask = list(range(img.shape[1]))  
 mask = random.sample(mask, int(img.shape[1]\*noise\_ratio[i]))  
 **for** k **in** range(img.shape[1]):  
 **if** k **in** mask:  
 noise\_img[j,k,i] = 0  
 *# -----------------------------------------------* **return** noise\_img

====================================================================

**def** restore\_image(noise\_img, size=4):  
 *"""  
 使用 你最擅长的算法模型 进行图像恢复。* **:param** *noise\_img: 一个受损的图像* **:param** *size: 输入区域半径，长宽是以 size\*size 方形区域获取区域, 默认是 4* **:return***: res\_img 恢复后的图片，图像矩阵值 0-1 之间，数据类型为 np.array,  
 数据类型对象 (dtype): np.double, 图像形状:(height,width,channel), 通道(channel) 顺序为RGB  
 """  
 # 恢复图片初始化，首先 copy 受损图片，然后预测噪声点的坐标后作为返回值。* res\_img = np.copy(noise\_img)  
  
 *# 获取噪声图像* noise\_mask = get\_noise\_mask(noise\_img)  
  
 *# -------------实现图像恢复代码答题区域----------------------------* **for** i **in** range(noise\_mask.shape[0]):  
 **for** j **in** range(noise\_mask.shape[1]):  
 **for** k **in** range(noise\_mask.shape[2]):  
 **if** noise\_mask[i,j,k] == 0:  
 sc = 1  
 listx = get\_window\_small(res\_img,noise\_mask,i,j,k)  
 **if** len(listx) != 0:  
 res\_img[i,j,k] = listx[len(listx)//2]  
 **else**:  
 **while**(len(listx) == 0):  
 listx = get\_window(res\_img,noise\_mask,sc,i,j,k)  
 sc = sc+1  
 **if** sc > 4:  
 res\_img[i,j,k] = np.mean(listx)  
 **else**:  
 res\_img[i,j,k] = listx[len(listx)//2]  
  
 *# ---------------------------------------------------------------* **return** res\_img  
**def** get\_window(res\_img,noise\_mask,sc,i,j,k):  
 listx = []  
  
 **if** i-sc >= 0:  
 starti = i-sc  
 **else**:  
 starti = 0  
 **if** j+1 <= res\_img.shape[1]-1 **and** noise\_mask[0,j+1,k] !=0:  
 listx.append(res\_img[0,j+1,k])  
 **if** j-1 >=0 **and** noise\_mask[0,j-1,k] !=0:  
 listx.append(res\_img[0,j-1,k])  
  
 **if** i+sc <= res\_img.shape[0]-1:  
 endi = i+sc  
 **else**:  
 endi = res\_img.shape[0]-1  
 **if** j+1 <= res\_img.shape[1]-1 **and** noise\_mask[endi,j+1,k] !=0:  
 listx.append(res\_img[endi,j+1,k])  
 **if** j-1 >=0 **and** noise\_mask[endi,j-1,k] !=0:  
 listx.append(res\_img[endi,j-1,k])  
  
 **if** j+sc <= res\_img.shape[1]-1:  
 endj = j+sc  
 **else**:  
 endj = res\_img.shape[1]-1  
 **if** i+1 <= res\_img.shape[0]-1 **and** noise\_mask[i+1,endj,k] !=0:  
 listx.append(res\_img[i+1,endj,k])  
 **if** i-1 >=0 **and** noise\_mask[i-1,endj,k] !=0:  
 listx.append(res\_img[i-1,endj,k])  
  
 **if** j-sc >= 0:  
 startj = j-sc  
 **else**:  
 startj = 0  
 **if** i+1 <= res\_img.shape[0]-1 **and** noise\_mask[i+1,0,k] !=0:  
 listx.append(res\_img[i+1,0,k])  
 **if** i-1 >=0 **and** noise\_mask[i-1,0,k] !=0:  
 listx.append(res\_img[i-1,0,k])  
  
 **for** m **in** range(starti,endi+1):  
 **for** n **in** range(startj,endj+1):  
 **if** noise\_mask[m,n,k] != 0:  
 listx.append(res\_img[m,n,k])  
 listx.sort()  
 **return** listx  
  
**def** get\_window\_small(res\_img,noise\_mask,i,j,k):  
 listx = []  
 sc = 1  
 **if** i-sc >= 0 **and** noise\_mask[i-1,j,k]!=0:  
 listx.append(res\_img[i-1,j,k])  
  
 **if** i+sc <= res\_img.shape[0]-1 **and** noise\_mask[i+1,j,k]!=0:  
 listx.append(res\_img[i+1,j,k])  
  
 **if** j+sc <= res\_img.shape[1]-1 **and** noise\_mask[i,j+1,k]!=0:  
 listx.append(res\_img[i,j+1,k])  
  
 **if** j-sc >= 0 **and** noise\_mask[i,j-1,k]!=0:  
 listx.append(res\_img[i,j-1,k])  
 listx.sort()  
 **return** listx

1. **实验结果**

（实验结果，必填）

====================================================================

图形用户界面, 应用程序, 网站

描述已自动生成