实验背景

图像是一种非常常见的信息载体，但是在图像的获取、传输、存储的过程中可能由于各种原因使得图像受到噪声的影响。
如何去除噪声的影响，恢复图像原本的信息是计算机视觉中的重要研究问题。

常见的图像恢复算法有基于空间域的中值滤波、基于小波域的小波去噪、基于偏微分方程的非线性扩散滤波等，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。

实验要求

A. 生成受损图像。

- 受损图像（X）是由原始图像（$I\in R^{HWC}$）添加了不同噪声遮罩（noise masks）（$M\in R^{HWC}$）得到的（$X=I \bigodot M$），其中$\bigodot$是逐元素相乘。
- 噪声遮罩仅包含 {0,1} 值。对原图的噪声遮罩的可以每行分别用 0.8/0.4/0.6 的噪声比率产生的，即噪声遮罩每个通道每行 80%/40%/60% 的像素值为 0，其他为 1。

B. 使用你最擅长的算法模型，进行图像恢复。

C. 评估误差为所有恢复图像（R）与原始图像（I）的 2-范数之和，此误差越小越好。$ \rm error=\sum_{i=1}^{3}norm(R_i(😃)-I_i(😃),2)$，其中(:)是向量化操作，其他评估方式包括 Cosine 相似度以及 SSIM 相似度。

生成受损图像（椒盐噪声）

生成受损图像的实验要求：

- 受损图像（X）是由原始图像（$I\in R^{HWC}$）添加了不同噪声遮罩（noise masks）（$M\in R^{HWC}$）得到的（$X=I \bigodot M$），其中 $\bigodot$ 是逐元素相乘。
- 噪声遮罩仅包含 {0,1} 值。对原图的噪声遮罩的可以每行分别用 0.8/0.4/0.6 的噪声比率产生的，即噪声遮罩每个通道每行 80%/40%/60% 的像素值为0，其他为1。

生成受损图像相关代码：

算法思想：对每行通过sample随机取部分元素为0

```
def noise_mask_image(img, noise_ratio):
    """
    根据题目要求生成受损图片
    :param img: 图像矩阵，一般为 np.ndarray
    :param noise_ratio: 噪声比率，可能值是0.4/0.6/0.8
    :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
    print(img.shape)
    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))
            for k in range(img.shape[1]):
                if k in mask:
                    noise_img[j,k,i] = 0
    # ----------------------------------------------

    return noise_img
```

获取噪声点：

```
def get_noise_mask(noise_img):
    """
    获取噪声图像，一般为 np.array
    :param noise_img: 带有噪声的图片
    :return: 噪声图像矩阵
    """
    # 将图片数据矩阵只包含 0和1,如果不能等于 0 则就是 1。
    return np.array(noise_img != 0, dtype='double')
```

评估误差

评估误差为所有恢复图像（R）与原始图像（I）的2-范数之和，此误差越小越好。$ \rm error=\sum_{i=1}^{3}norm(R_i(😃-I_i(😃,2)$，其中(:)是向量化操作。

理解2-范数参考资料：

- https://blog.csdn.net/SusanZhang1231/article/details/52127011
- https://baike.baidu.com/item/%E4%BA%8C%E8%8C%83%E6%95%B0

误差评估函数：

```
def compute_error(res_img, img):
    """
    计算恢复图像 res_img 与原始图像 img 的 2-范数
    :param res_img:恢复图像
    :param img:原始图像
    :return: 恢复图像 res_img 与原始图像 img 的2-范数
```

```
    """
    # 初始化
    error = 0.0

    # 将图像矩阵转换成为np.narray
    res_img = np.array(res_img)
    img = np.array(img)

    # 如果2个图像的形状不一致，则打印出错误结果，返回值为 None
    if res_img.shape != img.shape:
        print("shape error res_img.shape and img.shape %s != %s" % (res_img.shape,
img.shape))
        return None

    # 计算图像矩阵之间的评估误差
    error = np.sqrt(np.sum(np.power(res_img - img, 2)))

    return round(error,3)
```

图像相似度计算：

```
from skimage.measure import compare_ssim as ssim
from scipy import spatial

def calc_ssim(img, img_noise):
    """
    计算图片的结构相似度
    :param img: 原始图片， 数据类型为 ndarray, shape 为[长，宽，3]
    :param img_noise: 噪声图片或恢复后的图片，
                      数据类型为 ndarray, shape 为[长，宽，3]
    :return:
    """
    return ssim(img, img_noise,
                multichannel=True,
                data_range=img_noise.max() - img_noise.min())

def calc_csim(img, img_noise):
    """
    计算图片的 cos 相似度
    :param img: 原始图片， 数据类型为 ndarray, shape 为[长，宽，3]
    :param img_noise: 噪声图片或恢复后的图片，
                      数据类型为 ndarray, shape 为[长，宽，3]
    :return:
    """
    img = img.reshape(-1)
    img_noise = img_noise.reshape(-1)
    return 1 - spatial.distance.cosine(img, img_noise)
```

图像恢复

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。

此处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；

```
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

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
```