程序报告

学号：　　上海交通大学 518143910019　　姓名：周涵

一、问题重述

（简单描述对问题的理解，从问题中抓住主干，必填）

==========

鉴于图像是一种十分常见的信息载体，但在图像的获取、传输和存储过程中经常会受到噪声影响，因此本次实验重点要研究的就是如何对受损图像进行恢复，去除噪声的影响。而在此之前，我们也要探究如何为图像增加噪声遮罩，生成受损图像，并探究如何基于区域二元线性回归模型，去除噪声，进行图像恢复。

二、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**）

==========

1. 生成受损图像：

由于受损图像就是在原始图像的每个通道上添加噪声遮罩，而原始图像有三个通道，即 r、g、b，由于噪声即意味着该点的像素值为 0，因此可以首先构建一个包含三个通道的 rgb 列表，rgb=[None,None,None]，再遍历每个通道，构建对应的噪声图，由于每个通道的噪声比率已经给定，即 noise_ratio=[0.8,0.4,0.6]，因此可以利用调用 random 包中的 choice 函数，以对应比率生成 0 和 1 的序列。

又因为噪声遮罩应为一个大小维度都与原图一致的 0-1 矩阵，因此需要在获得行数 row 和列数 col 之后，在 rgb 的每个通道中，遍历每行，利用 random.choice 生成采样数为 col size 为（1，col）的每行列表，如：rgb[i]=np.random.choice(2,(1,col),p=[noise_ratio[i],1-noise_ratio[i]])

最后还需扩展列表 shape 至三维，rgb[i]=rgb[i][:,:,np.newaxis]，最终即可得到噪声遮罩，将噪声遮罩与原图像素值进行逐点相乘，得到受损图像

2. 图像恢复：

首先将图像分割成 10*10 的区域，考虑到图像可能无法被正好完整分隔，存在边缘区域，因此先定义 row_cnt 为 rows（行数）//10，col_cnt 为 cols（列数）//10，在每个通道中，分别遍历 row_cnt, col_cnt，实现区域的移动，到边界即返回，在同一个循环下，遍历当前区域的每个点，获取训练集，若为噪声点即加入测试集，训练集 x 为每个点的坐标位置，而 y 则为对应像素值，在获得所需数据后，即可进行线性回归的拟合和预测，最终获得恢复后的图像。

三、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）

==========

1. 生成受损图像：

```
row,col=img.shape[:2]
rgb=[None,None,None] #rgb
for i in range(3):
    #构建其中一个通道的噪声图
    for j in range(row):
        if rgb[i] is None:
            rgb[i]=np.random.choice(2,(1,col),p=[noise_ratio[i],1-noise_ratio[i]])
            #以对应比率生成噪声，值为0的概率即为噪声比率o，值为1的概率对应（1-噪声比率），采样值为1*col
        else:
            a = np.random.choice(2,(1,col),p=[noise_ratio[i],1-noise_ratio[i]])

            rgb[i]=np.concatenate((rgb[i],a),axis=0) #数组拼接
#扩展 shape
for i in range(3):
    rgb[i]=rgb[i][:,:,np.newaxis]

#合并生成噪声遮罩
rst = np.concatenate((rgb[0],rgb[1],rgb[2]),axis=2)
noise_img = rst*img#将噪声遮罩覆盖在原图上，即为受损图像
```

==========

2. 图像恢复：

首先进行区域分割

```
rows,cols, channel = res_img.shape
print(rows,cols,channel)
region=10 #10*10
row_cnt=rows//region
col_cnt=cols//region
#分割区域
for chan in range(channel):
    for rn in range(row_cnt+1):
        ibase = rn * region
        if rn == row_cnt:#到边界返回
            ibase = rows - region
        for cn in range(col_cnt+1):
            jbase = cn*region
            if cn == col_cnt:#到边界返回
                jbase = cols-region
```

再对每个区域上的点采样获取训练集和测试集，进行线性回归拟合，获得拟合后的像素值，恢复图像

```
                x_train=[]
                y_train=[]
                x_test=[]
                for i in range(ibase,ibase+region): #遍历每个10*10的区域
                    for j in range(jbase,jbase+region):
                        if noise_mask[i,j,chan]==0: #噪声点
                            x_test.append([i,j])#将噪声点加入测试集
                            continue
                        x_train.append([i,j]) #x训练集为坐标点
                        y_train.append([res_img[i,j,chan]]) #y训练集为对应坐标点像素值
                if x_train ==[]:
                    print("x_train is None")
                    continue
                reg = LinearRegression()
                reg.fit(x_train,y_train) #符合线性模型进行拟合
                pred = reg.predict(x_test)#对测试集进行预测

                for i in range(len(x_test)):
                    res_img[x_test[i][0],x_test[i][1],chan] = pred[i][0]
    print(res_img)
    res_img[res_img > 1.0]=1.0
    res_img[res_img<0.0]=0.0
```

四、实验结果

（实验结果，必填）

==========

1. **生成受损图像：**

受损图像的对应的像素值矩阵大致为如下所示：

```
array([[[0.        , 0.72941176, 0.72156863],
        [0.        , 0.75294118, 0.        ],
        [0.79215686, 0.        , 0.        ],
        ...,
        [0.        , 0.76078431, 0.        ],
        [0.78039216, 0.76470588, 0.        ],
        [0.        , 0.        , 0.        ]],

       [[0.        , 0.78431373, 0.        ],
        [0.78823529, 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        ...,
        [0.78039216, 0.76470588, 0.        ],
        [0.77254902, 0.        , 0.        ],
        [0.        , 0.75294118, 0.        ]],

       [[0.        , 0.        , 0.        ],
        [0.79607843, 0.79607843, 0.        ],
        [0.78039216, 0.78039216, 0.        ],
        ...,
        [0.        , 0.76862745, 0.75686275],
        [0.        , 0.74901961, 0.74509804],
        [0.        , 0.7372549 , 0.        ]],

       ...,
```

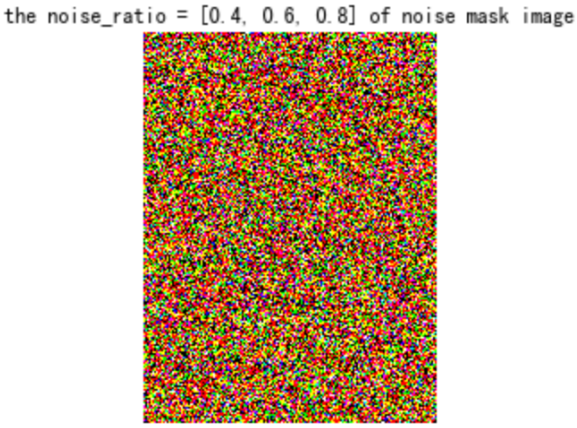
```
[[0.91764706, 0.90980392, 0.        ],
 [0.92156863, 0.91372549, 0.9254902 ],
 [0.9372549 , 0.92941176, 0.        ],
 ...,
 [0.        , 0.        , 0.94117647],
 [0.        , 0.91764706, 0.        ],
 [0.        , 0.        , 0.93333333]],

[[0.        , 0.94509804, 0.        ],
 [0.        , 0.        , 0.95686275],
 [0.        , 0.9372549 , 0.        ],
 ...,
 [0.        , 0.91764706, 0.94509804],
 [0.        , 0.        , 0.94509804],
 [0.93333333, 0.        , 0.94901961]],

[[0.        , 0.94901961, 0.        ],
 [0.        , 0.        , 0.95686275],
 [0.        , 0.92941176, 0.94509804],
 ...,
 [0.        , 0.        , 0.        ],
 [0.        , 0.9254902 , 0.        ],
 [0.        , 0.        , 0.95686275]]])
```

从而获得的受损图像为：

噪声遮罩为：



2. 图像恢复：

以下分别是原图，受损图像，以及恢复后的图像，其中，答案得出：

恢复图片与原始图片的评估误差： 64.057

恢复图片与原始图片的 SSIM 相似度： 0.8173807458685088

恢复图片与原始图片的 Cosine 相似度： 0.9985701658764836

original image

the noise_ratio = [0.4, 0.6, 0.8] of original image

restore image

五、总结

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

==========

本次实验达到了预期的目标，在实现过程中遇到过不知如何扩展 rgb 维度大小、恢复图像中不知如何进行区域划分等问题，可能区域划分的大小还能加以改进，以及滤波器的选择可以有不同尝试。