

程序报告

学号：

姓名：

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)

本问题主要分成两个部分：其一是对图像生成噪音，对原图的噪声遮罩的可以每行用 0.8/0.4/0.6 的噪声比例进行混淆，即噪声遮罩每个通道每行以 80%/40%/60% 的像素是 1，其他为 1；其二，使用一种算法模型进行图像恢复。

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)

对于生成噪音，笔者使用的方法主要是调用 numpy 的函数 random.choice 一行一行生成噪声，然后将生成的噪声遮罩和原图进行逐项相乘。对于恢复图像，笔者尝试了诸多方法，一开始主要是使用 size 大小的区域线性回归，发现其恢复并不是很理想；后来发现直接使用自动调节范围的均值滤波器效果更好；笔者再尝试了自动调节的线性回归发现效果又优化了一些；笔者最后尝试结合均值滤波器和线性回归，发现不是很理想。最后使用了自动调节范围的线性回归。

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)

首先是噪声生成的算法：

```
# 受损图片初始化
noise_img = None
# -----实现受损图像答题区域-----
height = img.shape[0]
width = img.shape[1]
channel = img.shape[2]
unnoise_ratio = [1 - ratio for ratio in noise_ratio]
noise_ratioRGB = list(zip(noise_ratio, unnoise_ratio))
noise = np.empty((height, width, channel))
for h in range(height):
    noise_img_rowRGB = np.zeros(shape = (channel, width))
    for c in range(channel):
        noise_img_rowRGB[c] = np.random.choice(a = np.array([0, 1]), size = width, p = list(noise_ratioRGB[c]))
    noise[h] = noise_img_rowRGB.T
noise_img = img * noise

# -----
return noise_img
```

笔者主要是创建了一个 $H \times W \times C$ 的噪声遮罩数组 noise 和一个 $C \times W$ 的辅助数组 noise_img_rowRGB。接下来调用 np.random.choice 函数来生成长度为 W 的噪声，生成 C 个填入 noise_img_rowRGB 之后，然后进行转置最后放进 noise 中，这样填满一层。重复 H 次即可填满整个 noise 数组。最后将 img 和 noise 逐项相乘。注意上面 choice 函数的三个参数，

分别是可能值、生成长度、概率（选中与非选中的概率列表），因此笔者使用 zip 函数来创建最后一个概率参数。

笔者首先使用下面的线性回归函数进行恢复，笔者尝试了 LinearRegression、Ridge、Lasso 三种线性回归函数，发现最后的相似度 SSIM 仅在 30%左右，Cosine 相似度约 90%，笔者发现使用线性回归使得周围有很多像素被遮盖的像素恢复困难，因为常常会出现某一个像素周围边长为 4 的范围内都被遮盖。

```
model = LinearRegression()
for c in range(channel):
    noise_maskC = noise_mask[:, :, c]
    noise_imgC = noise_img[:, :, c]
    for x in range(height):
        for y in range(width):
            if noise_maskC[x][y] != 0:
                continue
            minRow = max(0, x - size // 2)
            maxRow = min(width, x + size // 2)
            minCol = max(0, y - size // 2)
            maxCol = min(height, y + size // 2)
            input = []
            output = []
            for i in range(minRow, maxRow):
                for j in range(minCol, maxCol):
                    if i != x and j != y and noise_maskC[i][j] != 0:
                        input.append([i - minRow, j - minCol])
                        output.append([noise_imgC[i - minRow][j - minCol]])
            if len(output) == 0:
                continue
            model.fit(input, output)
            predictResult = model.predict([[x - minRow, y - minCol]])[0]
            res_img[x][y][c] = max(min(predictResult, 1), 0)
```

笔者接下来尝试其他的图像恢复方法，为了弥补上面大量点的值不能被恢复的弊端，笔者决定尝试使用均值滤波器，并且当某个区间没有足够多的样本点的时候，就自动扩大 size 来寻找更多的样本点，发现效果很好，SSIM 相似约 89%，Cosine 相似度达到 99%。代码如下：

```

for c in range(channel):
    noise_maskC = noise_mask[:, :, c]
    for x in range(height):
        for y in range(width):
            if noise_maskC[x][y] != 0:
                continue
            expandSize = size
            num = 0
            while num == 0:
                sum = 0.0
                num = 0
                minRow = max(0, x - expandSize // 2)
                maxRow = min(width, x + expandSize // 2)
                minCol = max(0, y - expandSize // 2)
                maxCol = min(height, y + expandSize // 2)
                targetMatrix = noise_maskC[minRow : maxRow, minCol : maxCol]
                for i in range(maxRow - minRow):
                    for j in range(maxCol - minCol):
                        if targetMatrix[i][j] != 0:
                            num += 1
                            sum += res_img[minRow + i][minCol + j][c]
            if num != 0:
                res_img[x][y][c] = sum / num
            expandSize += 1

```

接下来笔者考虑使用能自动调节范围的线性回归算法，实现方法主要是上面两种的结合，最后 SSIM 相似度达到 90 左右，Cosine 相似度超过 99%，效果已经很好。

```

model = LinearRegression()
for c in range(channel):
    noise_maskC = noise_mask[:, :, c]
    for x in range(height):
        for y in range(width):
            if noise_maskC[x][y] != 0:
                continue
            input = []
            output = []
            expandSize = size
            num = 0
            while num == 0:
                num = 0
                minRow = max(0, x - expandSize // 2)
                maxRow = min(width, x + expandSize // 2)
                minCol = max(0, y - expandSize // 2)
                maxCol = min(height, y + expandSize // 2)
                targetMatrix = noise_maskC[minRow : maxRow, minCol : maxCol]
                for i in range(maxRow - minRow):
                    for j in range(maxCol - minCol):
                        if targetMatrix[i][j] != 0:
                            num += 1
                            input.append([i, j])
                            output.append([res_img[minRow + i][minCol + j][c]])
            if num != 0:
                model.fit(input, output)
                res_img[x][y][c] = max(0, min(model.predict([[x - minRow, y - minCol]])[0], 1))
            expandSize += 1

```

最后笔者考虑将两者结合起来，想法是首先使用均值滤波器，给所有点都计算其像素值，然后再针对原来被遮盖的点使用固定 size 的线性回归恢复，因为已经给所有点都计算出了像素值，所以尽管固定 size，样本也已经够多了 (size*size-1)，但是发现效果并不是很好，SSIM 相似度也在 30% 左右。最终放弃了这种优化，均值滤波器的代码就如上，线性回归部分代码如下：

```
model = LinearRegression()
for c in range(channel):
    noise_maskC = noise_mask[:, :, c]
    noise_imgC = noise_img[:, :, c]
    for x in range(height):
        for y in range(width):
            if noise_maskC[x][y] != 0:
                continue
            minRow = max(0, x - size // 2)
            maxRow = min(width, x + size // 2)
            minCol = max(0, y - size // 2)
            maxCol = min(height, y + size // 2)
            input = []
            output = []
            for i in range(minRow, maxRow):
                for j in range(minCol, maxCol):
                    if i != x and j != y:
                        input.append([i - minRow, j - minCol])
                        output.append([noise_imgC[i][j]])
            model.fit(input, output)
            predictResult = model.predict([[x - minRow, y - minCol]])[0]
            res_img[x][y][c] = max(min(predictResult, 1), 0)
```




四、实验结果

(实验结果, 必填)

1. 使用自动调节大小的均值滤波器得到的结果

测试详情

×

测试点	状态	时长	结果
测试噪声图片的恢复（噪声种类 1）	✓	1s	恢复成功，在 150 x 150 的测试图片，得到的误差为 19.701, SSIM 相似度为 0.85, Cosine 相似度为 0.99 
测试噪声图像的生成	✓	0s	生成的噪声图片的噪声比例无误
测试噪声图片的恢复（噪声种类 2）	✓	1s	恢复成功，在 150 x 150 的测试图片，得到的误差为 15.982, SSIM 相似度为 0.894, Cosine 相似度为 0.994 
测试噪声图片的恢复（噪声种类 3）	✓	0s	恢复成功，在 150 x 150 的测试图片，得到的误差为 16.452, SSIM 相似度为 0.89, Cosine 相似度为 0.993 

2. 使用自动调节窗格大小的线性回归算法

测试点	状态	时长	结果
测试噪声图像的生成	✓	0s	生成的噪声图片的噪声比例无误
测试噪声图片的恢复（噪声种类 1）	✓	15s	恢复成功，在 150 x 150 的测试图片，得到的误差为 18.735, SSIM 相似度为 0.87, Cosine 相似度为 0.991 
测试噪声图片的恢复（噪声种类 3）	✓	12s	恢复成功，在 150 x 150 的测试图片，得到的误差为 15.098, SSIM 相似度为 0.906, Cosine 相似度为 0.994 
测试噪声图片的恢复（噪声种类 2）	✓	12s	恢复成功，在 150 x 150 的测试图片，得到的误差为 14.746, SSIM 相似度为 0.91, Cosine 相似度为 0.995 

提交结果

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**）

本次实验基本达到了预期目标，在做这个问题的时候常常出现 BUG，主要是自己的代码技术相对弱。目前的恢复方法还是相对比较粗糙的，因为只使用单一的图像恢复方法，之后可以更好地尝试和结合多种方法来比较效果。

