

人工智能作业报告

Project 2 —— 图像恢复

Jessie Peng

2019/05/16

1 实验内容

给定三张受损图像（一张黑白和两张彩色图像）和对应的噪声遮罩（noise mask），尝试恢复它们的原始图像。

受损图像是由原始图像按照一定比率（0.4, 0.6, 0.8）添加噪声（随机将某些像素变为0）产生的，具体来说，噪声遮罩的每个通道每行80%，40%，60%的像素为0，其它为1。

2 实验环境

编程语言：Python

开发环境：PyCharm 2018.3

操作系统：macOS 10.14.1 (18B75)

3 实验原理

3.1 线性回归

采用回归的方法来恢复受损的图像，就是把图像的每个通道看作像素关于位置的函数，我们希望通过已有的该函数上的数据点估计出该函数，从而可以预测受损位置的像素。

线性回归是假设该函数具有如下的线性表示形式：

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

其中 ϕ_j 是某种线性基函数。

我们可以使用最小二乘法来估计权值矩阵：

$$\mathbf{w}_{ML} = (\boldsymbol{\phi}_p^T \boldsymbol{\phi}_p)^{-1} \boldsymbol{\phi}_p^T \mathbf{y}_p$$

其中下标 p 表示已知的数据点。

然后就可以预测受损位置的像素了：

$$\mathbf{y}_n = \mathbf{w}_{ML}^T \boldsymbol{\phi}_n$$

其中下标 n 表示待求的数据点。

3.2 高斯基函数

线性回归中基函数可以有多种选择，本实验中我们使用高斯基函数：

$$\phi_j(x) = e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}}$$

此外，在本实验中，我们使用按行回归的方式，对每一行的数据点单独进行预测。

3.3 k-近邻回归

按行回归具有局限性，由于每一行是独立进行回归和预测的，就可能会出现相邻的但是不同行的点像素差距较大，恢复出的图像在垂直方向上出现断层。

使用 k-近邻回归可以改善这个问题。kNN (k-Nearest Neighbor) 是选取每个待预测点附近最近的 k 个已知点，求这 k 个邻近点的加权平均（权重由距离决定）作为预测值。

4 实现过程与代码分析

本次实验的步骤是首先写好按行回归和 kNN 两种图像恢复的模型，然后使用自己的图片对模型进行比较分析，选择出较优的模型用于恢复作业提供的受损图像。

4.1 按行回归

对于每个通道分别进行按行回归分析，代码如下：

```
# for each channel
for c in range(corrImg.shape[2]):
    inputImg = corrImg[:, :, c]
    mask = noiseMask[:, :, c]

    # analysis by line
    for i in range(h):
        line = inputImg[i]
        tp = np.array([line[j] for j in range(w) if mask[i][j] == 1])
        xp = np.array([np.linspace(0, 1, num=w)[j] for j in range(w) if mask[i][j] == 1])
        xn = np.array([np.linspace(0, 1, num=w)[j] for j in range(w) if mask[i][j] == 0])
```

```

# Gaussian basis function
phi_p = np.zeros((basisNum, xp.shape[0]))
phi_n = np.zeros((basisNum, xn.shape[0]))
for j in range(basisNum):
    phi_p[j] = np.exp(-(xp - Phi_mu[j])**2 / (Phi_sigma[j]**2 * 2))
for j in range(basisNum):
    phi_n[j] = np.exp(-(xn - Phi_mu[j])**2 / (Phi_sigma[j]**2 * 2))

W_ml = np.linalg.pinv(phi_p.dot(np.transpose(phi_p))).dot(phi_p).dot(tp)
tn = np.transpose(W_ml).dot(phi_n)

# restore image
cnt = 0
for j in range(w):
    if line[j] == 0:
        resImg[i][j][c] = tn[cnt]
    cnt = cnt + 1

```

4.2 kNN

同样地，对每个通道单独进行 k-近邻回归。这里使用到 `sklearn` 库中的 `neighbors.KDTree` 来方便求得 k 个最邻近的点。具体的 k 取多少要实验决定。代码如下：

```

# for each channel
for c in range(corrImg.shape[2]):
    inputImg = corrImg[:, :, c]
    mask = noiseMask[:, :, c]

    # prepare data
    train_x = []
    test_x = []
    for i, tmp in enumerate(mask):
        for j, val in enumerate(tmp):
            if val == 1:
                train_x.append([i, j])
            else:
                test_x.append([i, j])
    train_x = np.array(train_x)
    test_x = np.array(test_x)
    train_y = [inputImg[ind[0]][ind[1]] for ind in train_x]
    train_y = np.array(train_y)

    # build KD-Tree
    kd_tree = KDTree(train_x)
    dist, ind = kd_tree.query(test_x, k=3)

    # predict
    pred_y = np.sum(train_y[ind] / dist, axis=1) / np.sum(1 / dist, axis=1)

    # restore image
    for i in range(test_x.shape[0]):
        resImg[test_x[i][0]][test_x[i][1]][c] = pred_y[i]

```

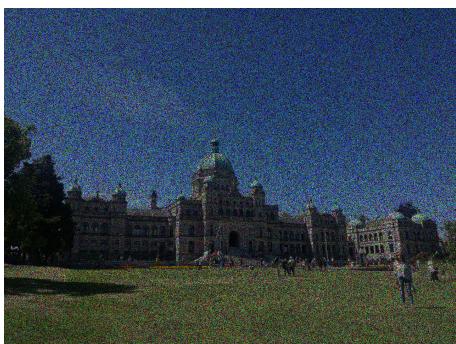
5 结果分析与实验总结

5.1 模型比较

使用如下一幅彩色图和一幅灰度图进行测试：



分别添加 0.5 和 0.7 的噪声：



采用高斯基函数按行进行回归的结果：



	distance (original, corrupted)	distance (original, reconstructed)
castle (0.5)	850.4066	97.4013
car (0.7)	268.4701	23.6799

采用 kNN (分别取 $k = 2, 3, 4, 5, 6$) 的结果：

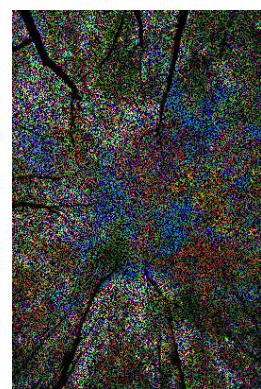
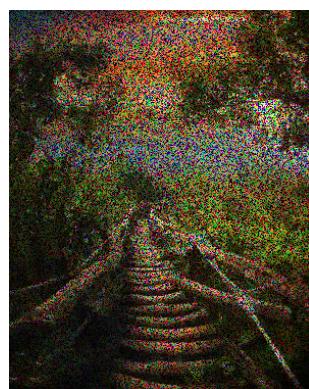
	k	distance (original, reconstructed)
castle (0.5)	2	53.5607
	3	51.3067
	4	51.6921
	5	52.3486
	6	53.0910
car (0.7)	2	14.5852
	3	14.0975
	4	14.1922
	5	14.4154
	6	14.6862

可见在 $k = 3$ 时效果最好，并且也明显优于按行回归的结果，恢复出的图像如下：

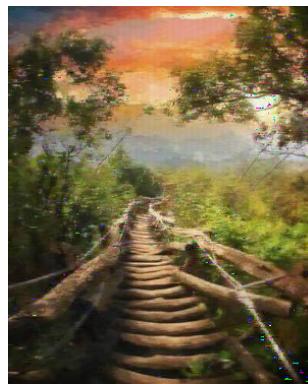


5.2 测试结果

作业中提供的受损图像如下：



按行回归恢复出的图像如下：



采用 kNN ($k = 3$) 恢复出的图像如下：



由于上一节模型比较的结果是 kNN 的效果更好（从这三幅测试图片中也能明显看出优劣），因此选用 $k = 3$ 的 kNN 模型恢复出的图像作为最终的作业结果提交。

5.3 问题与解决

本次作业一开始我按照作业指导中的按行回归进行图像恢复，但是发现效果不太好，恢复出的图像在行与行之间产生了很明显的断层，于是考虑到行与行之间的像素是相关联的，改用 k -邻近回归，恢复出的图像便更加连贯了。

6 参考文献

sklearn 文档——最近邻：<https://www.jianshu.com/p/84d745b85fd5>