自然图像 matting 的闭式解-代码实现

Dezeming Family

2023 年 2 月 11 日

DezemingFamily 系列文章和电子书**全部都有免费公开的电子版**，可以很方便地进行修改和重新发布。如果您获得了 DezemingFamily 的系列电子书，可以从我们的网站 [https://dezeming.top/] 找到最新的版本。对文章的内容建议和出现的错误也欢迎在网站留言。

目录

一 用户接口 **1**
1 1 函数 (1) 解释 1
1 2 函数 (2) 解释 1

二 计算 matting **2**
2 1 函数 (4) 的主体部分 2
2 2 求 Laplacian 2
2 2.1 坐标 win_inds 的生成 3
2 2.2 生成 matting Laplacian 4
2 3 求解背景和前景 4

参考文献 **4**

一　用户接口

我们根据论文 [1]（发布在 TPAMI 上的版本）来介绍代码。

在 [3] 中提供的源码中，user_interface_dezeming 是我们添加的测试代码，用于根据 trimap 来计算 alpha 值，生成 alpha 的 jpg 图像。

closed_form_matting.py 可以根据涂鸦图 scribbles 或者 trimap 图来提取 alpha 图：

```
// 使用涂鸦图，标记为函数(1)
alpha = closed_form_matting.closed_form_matting_with_scribbles(image,
    scribbles)
// 使用trimap，标记为函数(2)
alpha = closed_form_matting.closed_form_matting_with_trimap(image, trimap)
// 获得Matting Laplacian，标记为函数(3)
laplacian = compute_laplacian(image, optional_const_mask)
```

上面的计算 alpha 的两个函数都会调用下面的函数（后面再介绍它的含义）：

```
// 标记为函数(4)
alpha = closed_form_matting.closed_form_matting_with_prior(image, prior,
    prior_confidence, optional_const_mask)
```

solve_foreground_background.py 是根据提供的 alpha 图来求解前景和背景的。

```
// 标记为函数(5)
foreground, background = solve_foregound_background(image, alpha)
```

1 1　函数 (1) 解释

如果我们没有 trimap，只有原图和原图上绘制的涂鸦，我们就需要从涂鸦中抽取 trimap。

```
assert image.shape == scribbles.shape, 'scribbles must have exactly same
    shape as image.'
prior = np.sign(np.sum(scribbles - image, axis=2)) / 2 + 0.5
consts_map = prior != 0.5
```

np.sign(x) 的功能如下：

$$np.sign(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{一.1}$$

因此，[np.sign(x)/2+0.5] 就可以把输入 x 的值变为要么是 1（涂白的前景），要么是 0（涂黑的背景），当然也支持其他颜色（TPAMI 的版本支持对过渡区域使用不同的颜色，但其实并没有增加什么难度，因此我们不再加以区分介绍）。

np.sum 中设置相加轴为 2，意味着把 RGB 通道颜色值相加，得到一张单通道图。因此，prior（其实就是 trimap）是单通道的。consts_map 也是单通道图，其值表示该像素是否被约束，对应 prior 不为 0.5 的像素值为 1（表示被约束的像素），其他值为 0（表示没有被约束的像素）。传入到函数 (4) 时，会把原图、prior 和 consts_map 都传入。

1 2　函数 (2) 解释

trimap 本身就是单通道的，consts_map 中的像素值表示是否被约束，因此，如果某像素小于 0.1 或者大于 0.9，就认为它被约束了，否则就认为没有被约束。

```
consts_map = (trimap < 0.1) | (trimap > 0.9)
```

二 计算 matting

本节介绍函数 (4) 的实现过程。

2 1 函数 (4) 的主体部分

先放上公式：

$$\alpha = \operatorname{argmin}\ \alpha^T L\, \alpha + \lambda(\alpha^T - b_S^T)D_S\, (\alpha - b_S), \quad (13) \qquad (L + \lambda D_S)\alpha = \lambda b_S. \quad (14)$$

主体部分首先计算 matting laplacian（我们放在后面讲该函数）：

```
laplacian = compute_laplacian(image, ~consts_map if consts_map is not None
    else None)
```

然后先将 prior_confidence 图像拉成一维（使用 ravel() 函数），然后赋值到对角矩阵中（注意在调用函数 (4) 时，传入的参数 prior_confidence 默认是 consts_map 中每个像素值乘以 100，这里的 100 就是公式 (13) 的 λ 值，论文中也建议取一个稍微大一点的值）：

```
confidence = scipy.sparse.diags(prior_confidence.ravel())
```

然后调用 scipy.sparse.linalg.spsolve 函数，该函数求解公式 (14) 所述的线性系统：

```
solution = scipy.sparse.linalg.spsolve(
    laplacian + confidence,
    prior.ravel() * prior_confidence.ravel()
)
```

spsolve() 函数输入的两个参数，第一个参数就是 $L + \lambda D_S$，第二个参数就是 λb_S。

2 2 求 Laplacian

compute_laplacian() 函数的几个参数中，img 表示原始图像；mask 表示是否有区域不要被计算到，比如用户想屏蔽一些区域，不纳入考虑；eps 表示论文里的 ϵ，默认为 0.1^7；win_rad 即窗口半径 (window radius)，默认为 1，表示 3×3 的窗口。

```
win_size = (win_rad * 2 + 1) ** 2
```

然后计算图像横轴和纵轴有多少中心像素，每个轴上去掉两边边缘的半径大小的像素数即可：

```
c_h, c_w = h - 2 * win_rad, w - 2 * win_rad
```

之后的操作与公式 (12) 是完全对应的，但是有些实现细节我再详细描述一下，我觉得这个代码的作者对 python 的 numpy 的熟练程度还是很厉害的。

$$\sum_{k|(i,j)\in w_k} \left(\delta_{ij} - \frac{1}{|w_k|}(1 + (I_i - \mu_k)(\Sigma_k + \frac{\epsilon}{|w_k|} I_3)^{-1}(I_j - \mu_k)\right), \quad (12)$$

为了描述方便，我们假设原图尺寸是 200×100 的。

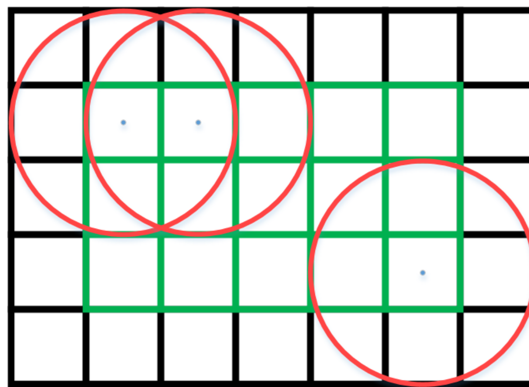
2 2.1 坐标 win_inds 的生成

_rolling_block() 函数中,计算得到的 shape(希望生成的矩阵的 shape)是 (100-3+1,200-3+1)+(3,3)=(98,198,3,3); 计算得到的 stride（希望生成的新矩阵的 stride）是 (800,4,800,4)。

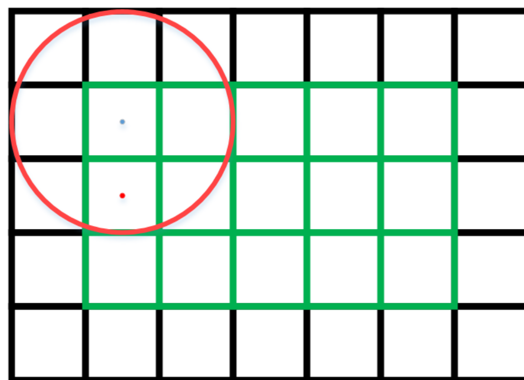
```
def _rolling_block(A, block=(3, 3)):
    """Applies sliding window to given matrix."""
    shape = (A.shape[0] - block[0] + 1, A.shape[1] - block[1] + 1) + block
    strides = (A.strides[0], A.strides[1]) + A.strides
```

解释一下上面的代码得到的结果。800 意味着图像每行有 200 乘以 4 字节（输入到该函数的 A 是单通道 4 字节）；4 意为着第二个维度下，每个维度有单通道 4 字节的元素（需要明确 shape 和 stride 的区别与联系）。

(98,198,3,3) 具体是什么意思，需要这么理解（调用 as_strided 意味着拆分矩阵）：下图中，假设窗半径是 1，绿色方格像素表示需要考虑的窗的像素中心，每个像素中心所在的窗都包含了邻域所在的 3×3 大小的区域。



再说得更通俗一点，通过 stride 访问元素位置，比如访问下图中 (0,0,2,1) 位置的元素，(0,0) 的位置在下图就是蓝点的位置，(2,1) 就是蓝点所在窗口的第 3 排（注意序号 2 就是第 3 个）的第 2 个像素，也就是下图中红点的位置，该位置的所在原图中的字节位置的计算就是 $2 \times 800 + 1 \times 4$（示意图长宽没有那么假设的那么大，但是不妨碍理解）：



调用 _rolling_block() 函数意味着把每个窗都提取出来，构成一个新矩阵。不过由于 as_strided() 的默认参数 subok 是 false，意味着只是生成矩阵的新视角（用 shape 为 (98,198,3,3) 的视角来观察这个矩阵），而不是生成新矩阵。strides 还是原始图像的 strides。

然后 win_inds（表示窗口数据在原图中的索引）会被 reshape 为 (98,198,9)：

```
win_inds = _rolling_block(indsM, block=(win_diam, win_diam))
// win_size即默认窗口半径为1时，3X3=9
win_inds = win_inds.reshape(c_h, c_w, win_size)
```

在调用 compute_laplacian() 时，参数没有 mask 的情况下，win_inds 的 shape 是 (19404,9)，也就是 (98*198,9)。

之后，将 RGB 三通道的原图也拉到对应维度（winI 的 shape 是 (98*198,9,3)）。

```
ravelImg = img.reshape(h * w, d)
winI = ravelImg[win_inds]
```

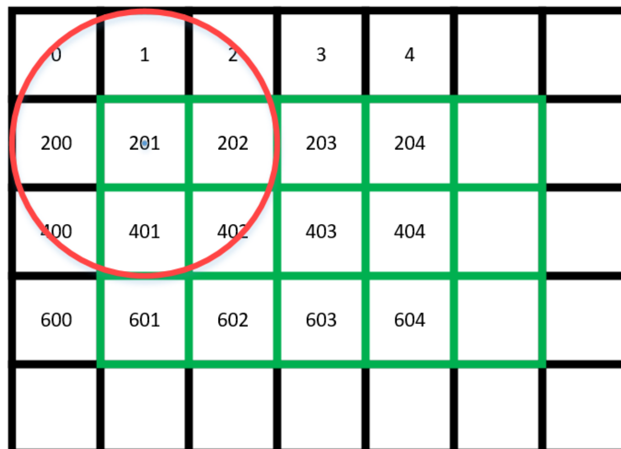
下面说一下有 mask 时的情况(即我们给出涂鸦的情况,注意即使没有涂鸦,也可以计算全图的 matting Laplacian 矩阵，只是用法不同)，注意 mask 就是 consts_map 取反的结果。consts_map 中的像素值表示是否被约束，被约束了则为 1。consts_map 当做 mask 时，我们不需要计算被约束的项，所以取反：

```
laplacian = compute_laplacian(image, ~consts_map if consts_map is not None
    else None)
```

当提供了 mask 时，先调用膨胀操作，cv2.dilate() 函数中，先把 mask 转换为 uint8 类型，然后再用原始大小的窗口膨胀。然后再使用 ravel() 拉成和 win_inds 同样的表示维度（即 (98,198,9)），然后认为 mask 中只要一个窗内存在大于 0 的值，这个窗的中心像素就是没有被约束的像素。

```
if mask is not None:
    mask = cv2.dilate(
        mask.astype(np.uint8),
        np.ones((win_diam, win_diam), np.uint8)
    ).astype(np.bool)
    win_mask = np.sum(mask.ravel()[win_inds], axis=2)
    # 统计被约束的像素
    win_inds = win_inds[win_mask > 0, :]
```

最后，取 win_inds 中的随便一组数来看一下索引，比如就取图中最左上角像素（绿框表示对应于 win_inds 所示的区域，而黑框是原图的分辨率，数字对应于原图的索引）：



之后把 win_inds 中索引的原图的坐标都提取出来，得到 winI：

```
ravelImg = img.reshape(h * w, d)
winI = ravelImg[win_inds]
```

winI 就是图中所有没有被约束的像素对应的窗。

2 2.2 生成 matting Laplacian

之后使用了相对复杂的 np.einsum 函数来做矩阵运算。

2 3 求解背景和前景

该求解过程比较复杂，对应于论文源码给出的 solveFB.m（Matlab 版本），等以后有精力再回来写详细过程。

参考文献

[1] Levin, A. , D. Lischinski , and Y. Weiss . "A Closed-Form Solution to Natural Image Matting." IEEE Transactions on Pattern Analysis and Machine Intelligence 30.2(2007):228-242.

[2] https://github.com/MarcoForte/closed-form-matting

[3] https://github.com/feimos32/ComputerVision-Code-Implementation-and-Collection