

Seam Carving 实现图像智能缩放

Python implementation for Seam Carving.

本项目使用 Python 实现。图的读取和显示等功能用到了 `opencv-python`, 可以使用 `pip` 安装。

本报告会描述整个项目构建的过程，从构建最简单的例子开始，到我如何加入更多细节实现复杂的算法。每一节都是在前面小节基础上的拓展。

要运行代码，请直接修改源代码中的参数（如文件路径、选择算法等），它们通常位于代码文件的末尾 `main` 函数部分，然后用 `python` 运行。每个程序都不需要命令行参数。可能需要安装一些依赖库，包括 `numpy`, `scipy`, `opencv-python`, `skimage` 等。

1 基础的 Seam Carving 实现

1.1 介绍

在本节中，将展示我实现的 Seam Carving 的基础实现，它可以做到：

- 支持图片水平缩小和竖直缩小
- 使用以下能量函数：L1-norm gradient, L2-norm gradient, entropy
- 显示增删过程和能量分布

比如对于如下照片，可以分别水平缩小和竖直缩小。



(原图, 水平缩小, 竖直缩小)

本节的所有代码在 `seam_carving_simple.py` 中。

1.2 实现

能量函数

首先介绍代码中实现的三个能量函数。

`energy_e1` 对应论文中的 e_1 能量，即 L1-norm gradient。通过 Sobel 算子获得 y 和 x 方向上的梯度，然后取绝对值相加。

```
def energy_e1(image):
    grad_x = ndimage.convolve1d(image, np.array([1, 0, -1]), axis=1, mode='wrap')
    grad_y = ndimage.convolve1d(image, np.array([1, 0, -1]), axis=0, mode='wrap')

    grad_x = np.abs(grad_x).sum(axis=2)
    grad_y = np.abs(grad_y).sum(axis=2)
    gradient_map = (np.abs(grad_x) + np.abs(grad_y))

    return gradient_map
```

`energy_e2` 对应论文中的 L2-norm gradient。通过 Sobel 算子获得 y 和 x 方向上的梯度，然后平方相加。

```
def energy_e2(image):
    grad_x = ndimage.convolve1d(image, np.array([1, 0, -1]), axis=1, mode='wrap')
    grad_y = ndimage.convolve1d(image, np.array([1, 0, -1]), axis=0, mode='wrap')

    grad_x = np.abs(grad_x).sum(axis=2)
    grad_y = np.abs(grad_y).sum(axis=2)
    gradient_map = np.sqrt((grad_x * grad_x) + (grad_y * grad_y))

    return gradient_map
```

`energy_entropy` 对应论文中的 $entropy$ 能量。它首先计算 $e1$ ，然后对每一个像素，取周围的 9×9 的窗口，计算 81 个像素的信息熵，加到原来的 $e1$ 中。

```

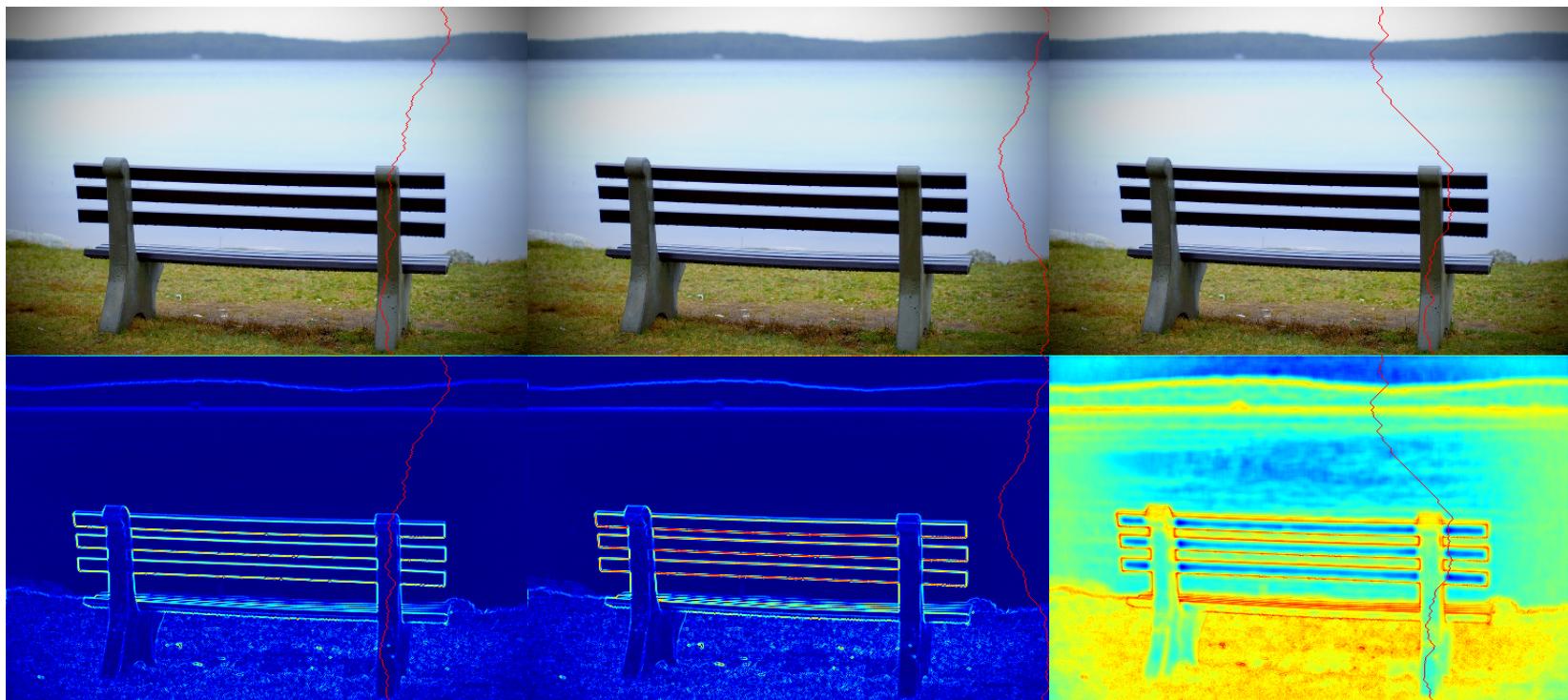
def energy_entropy(image):
    gradient_map = energy_e1(image)
    height, width, _ = image.shape
    for j in range(height):
        for i in range(width):
            # crop a 9 x 9 window around the pixel (j, i)
            y_start = max(j - 4, 0)
            y_end = min(j + 4 + 1, height)
            x_start = max(i - 4, 0)
            x_end = min(i + 4 + 1, width)
            sub_image = image[y_start:y_end, x_start:x_end]

            entropy = skimage.measure.shannon_entropy(sub_image)
            gradient_map[j, i] += entropy

    return gradient_map

```

对于同一张图，不同的能量函数如下，从左到右为 $e1$, $e2$, $entropy$, 看出 $entropy$ 的能量值要普遍大。另外计算 $entropy$ 使用时间也更长。



寻找能量最小细缝

在获得能量图之后，就可以计算代价最小的缝了，比如计算一条竖直方向的缝：

```

def find_min_energy_seam_vertical(image_energy, protect_mask=None, remove_mask=None):
    height, width = image_energy.shape
    image_energy = np.array(image_energy) # make a copy
    traceback = np.zeros_like(image_energy, dtype=np.int)

    if protect_mask is not None:
        image_energy[protect_mask] = 1000000

    if remove_mask is not None:
        image_energy[remove_mask] = -1000000

    for y in range(1, height):
        for x in range(0, width):
            if x == 0:
                idx = np.argmin(image_energy[y - 1, x:x + 2])
                traceback[y, x] = idx + x
                min_energy = image_energy[y - 1, idx + x]
            else:
                idx = np.argmin(image_energy[y - 1, x - 1:x + 2])
                traceback[y, x] = idx + x - 1
                min_energy = image_energy[y - 1, idx + x - 1]

            image_energy[y, x] += min_energy

    # backtrack to find path
    seam_idx = []
    seam_mask = np.ones((height, width), dtype=np.bool)
    j = np.argmax(image_energy[-1]) # last row
    for i in range(height - 1, -1, -1):
        seam_mask[i, j] = False
        j = traceback[i, j]

```

```

    seam_idx.append(j)
    j = traceback[i, j]

seam_idx.reverse()
return np.array(seam_idx), seam_mask

```

对于一个能量图 `image_energy`, 可以加上保护或者移除的 mask, 通过把对应能量值设为一个极大或者极小的值, 就可以控制细缝避开或者穿过此区域。之后使用动态规划计算, 最后标记缝的位置。

计算水平方向的缝也类似, 见 `find_min_energy_seam_horizontal`

缩小图片的主要逻辑 (水平或竖直)

水平缩小图片较为简单, 需要反复寻找最小代价的缝, 然后用 `seam_mask` 选择出需要的像素。由于数组中用行主序, 所以选择出像素后可以直接 `reshape` 成需要的形状, 而这在竖直缩小中是不行的。

水平缩小的代码:

```

for i in range(width_to_remove):
    energy_map = energy_func(input_image)
    heatmap = make_heatmap(energy_map)
    seam_index, seam_mask = find_min_energy_seam_vertical(energy_map, None, None)
    output_image = np.array(input_image)
    output_image[~seam_mask] = [0, 0, 1.]
    heatmap[~seam_mask] = [0, 0, 255.]
    concat = np.concatenate([(output_image * 255).astype(np.uint8), heatmap], axis=0)
    cv2.imshow("seam image + energy map", concat)
    cv2.waitKey(1)
    input_image = output_image[seam_mask].reshape((output_image.shape[0], output_image.shape[1] - 1, 3))

```

此处还动态展示了缩放的过程, 画出了删除的边。

竖直缩小需要把原来的图片旋转 90 度, 处理完后再旋转回来, 这是因为删除一条水平的缝之后, 直接 `reshape` 会像素错位。

竖直缩小的代码:

```

for i in range(height_to_remove):
    energy_map = energy_func(input_image)
    heatmap = make_heatmap(energy_map)
    seam_index, seam_mask = find_min_energy_seam_horizontal(energy_map, None, None)
    output_image = np.array(input_image)
    output_image[~seam_mask] = [0, 0, 1.]
    heatmap[~seam_mask] = [0, 0, 255.]
    concat = np.concatenate([(output_image * 255).astype(np.uint8), heatmap], axis=1)
    cv2.imshow("seam image + energy map", concat)
    cv2.waitKey(1)
    output_image = np.rot90(output_image, 1, axes=(0, 1))
    seam_mask = np.rot90(seam_mask, 1, axes=(0, 1))
    input_image = output_image[seam_mask].reshape((output_image.shape[0], output_image.shape[1] - 1, 3))
    input_image = np.rot90(input_image, -1, axes=(0, 1))

```

1.3 更多例子

文件夹中有一个视频文件 `simple_demo.mp4` 展示了动态的过程。

以下是一些其他例子的结果。





以下是不那么成功的例子，比如下面这辆车的图片。



水平缩小100、200、300个像素的结果如下，不能很好地保存原来的一些几何特征：



2 拓展功能

2.1 介绍

本部分介绍完成的所有拓展功能。为了更好地加入新的功能，我对1小节中的代码进行重构，把各个功能包装成一个对象。

本节的代码在 `seam_carving.py` 中。

我新加入了 `SeamCarvingImage` 类，负责管理状态，之前的函数作为它的成员函数，同时加入了如保存、显示等功能。`SeamCarvingImage` 类的用法可以在之后看到。

2.2 图像拓展

2.2.1 介绍

图像拓展实现了3种方式：迭代扩展、统一扩展、分阶段扩展。

迭代扩展每次选取cost最小的缝并扩展。扩展缝即将它一分为二，和左右邻居取平局。

统一扩展指要扩展n个像素的时候，找出cost最低的n条缝，然后对它们一起扩展。

分阶段扩展指要扩展n（如200）个像素的时候，每次扩展不超过m（如60）条缝。这样需要扩展 60+60+60+20条缝，共4次。

它们效果如下。

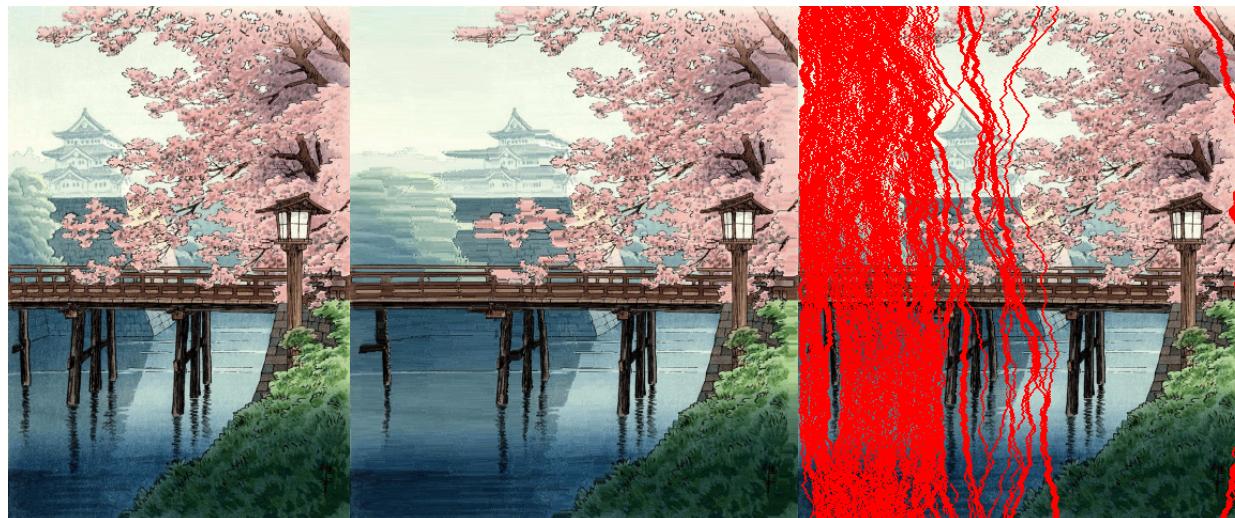


(迭代扩展、统一扩展、分阶段扩展的缝分布)



(迭代扩展、统一扩展、分阶段扩展的结果)

迭代扩展几乎每次都会拓展同一位置。在拓展的幅度不大的时候，统一扩展效果比较好。



运行本部分可使用 `demo7()`, `demo8_1()`, `demo8_2()`, `demo8_3()`。

2.2.2 实现

这部分代码查看 `expand_width(self, width_to_expand, min_seam_count)`。

`width_to_expand` 是总共扩展大小，`min_seam_count` 是每次最大拓展缝的数目。

当 `min_seam_count == 1` 时，为迭代扩展。

当 `width_to_expand == min_seam_count` 时，为同一扩展。

当 `width_to_expand > min_seam_count` 时，为分阶段扩展。

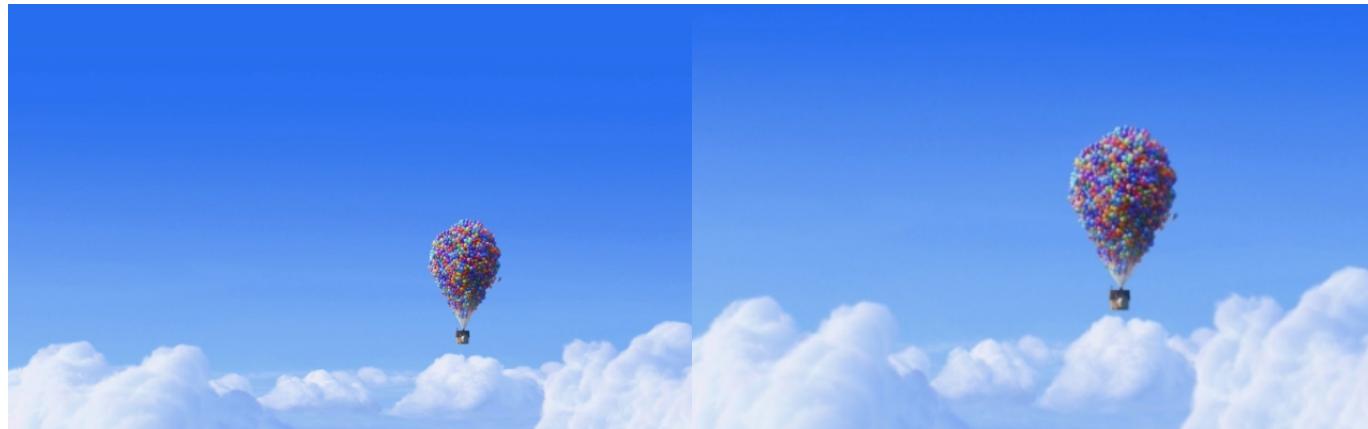
值得注意的是，在插入像素的时候，会使图像右侧的像素位置偏移1，需要小心处理。

2.3 内容放大 (Content Amplification)

2.3.1 介绍

内容放大的原理是先把图片用标准缩放放大一定尺寸，然后用 seam carving 缩小到原来的大小。

以下左为原图，右为放大的图。



运行本部分可使用 `demo1()`。

2.3.2 实现

本部分代码见 `amplify()` 方法：

```
def amplify(self, factor):  
    pass
```

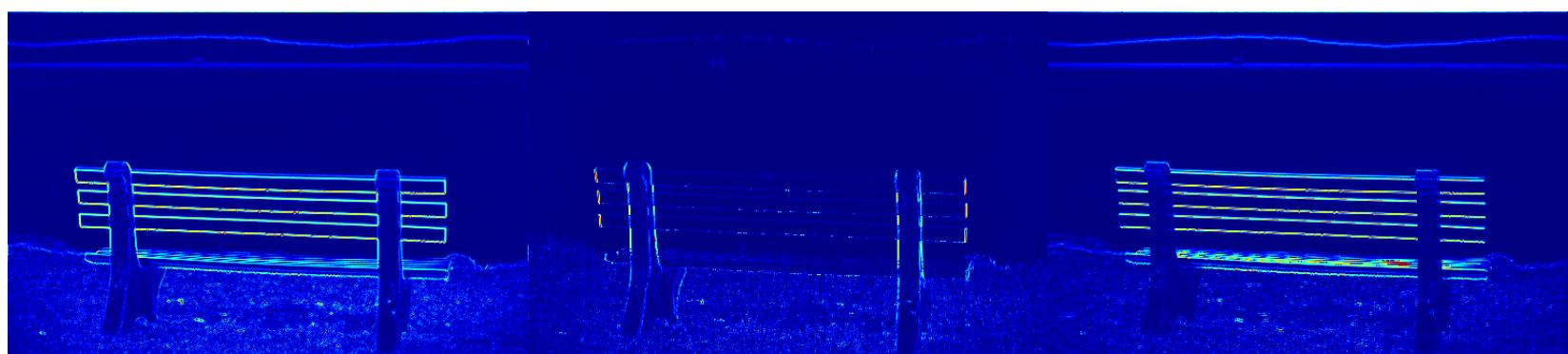
`factor` 定义了一个放大的系数，比如 1.2 倍，之后使用 `reduce_width()` 和 `reduce_height()` 缩减大小。

2.4 前向能量函数

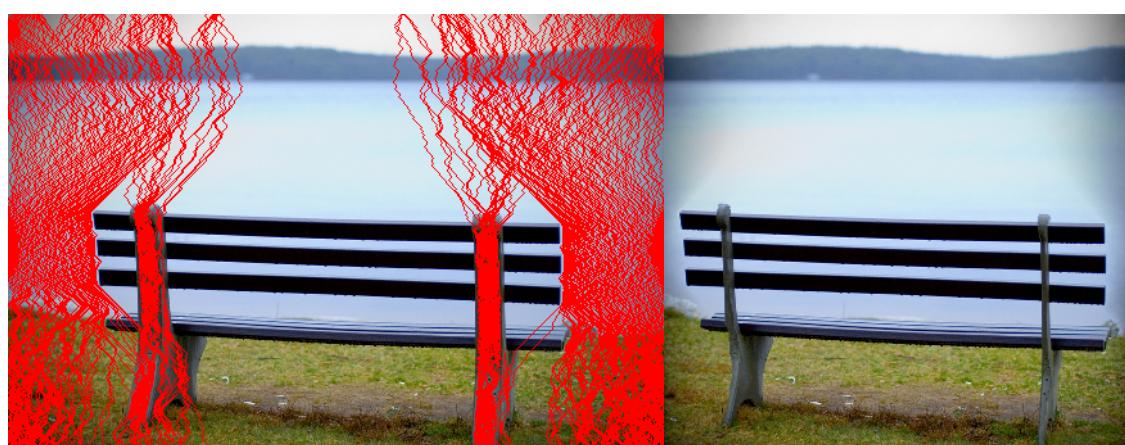
2.4.1 介绍

前向能量函数是更高级的能量函数，它区分水平和竖直方向，它可以使细缝从物体的内部穿过。

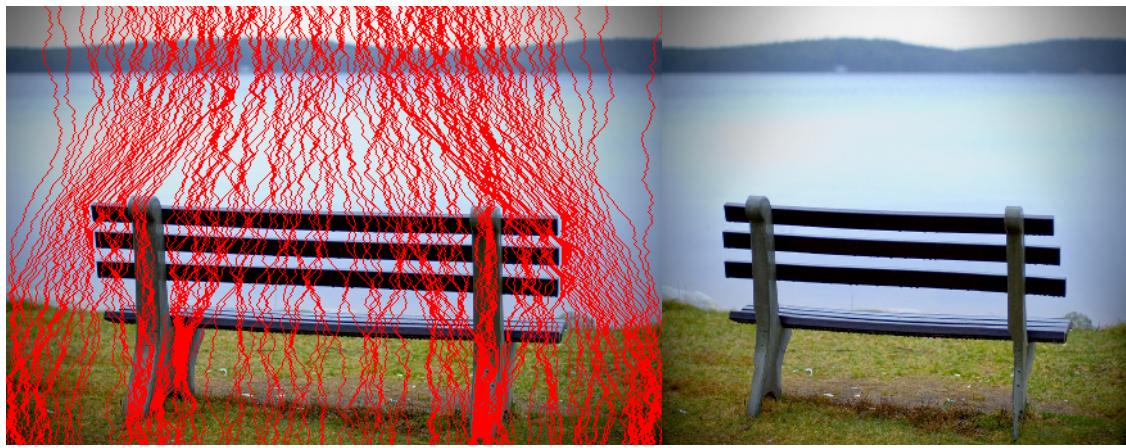
对于如下的长椅图，这是不同的能量函数对比：



(e1 能量, 竖直前向能量, 水平原向能量)



上图为 e1 能量的细缝，基本不从长椅内部穿过。



上图为 forward 能量的细缝，结果更加自然。

运行本部分可使用 `demo4()`。

2.4.2 实现

使用的方法是在创建对象的时候使用 `energy_function='forward'`。

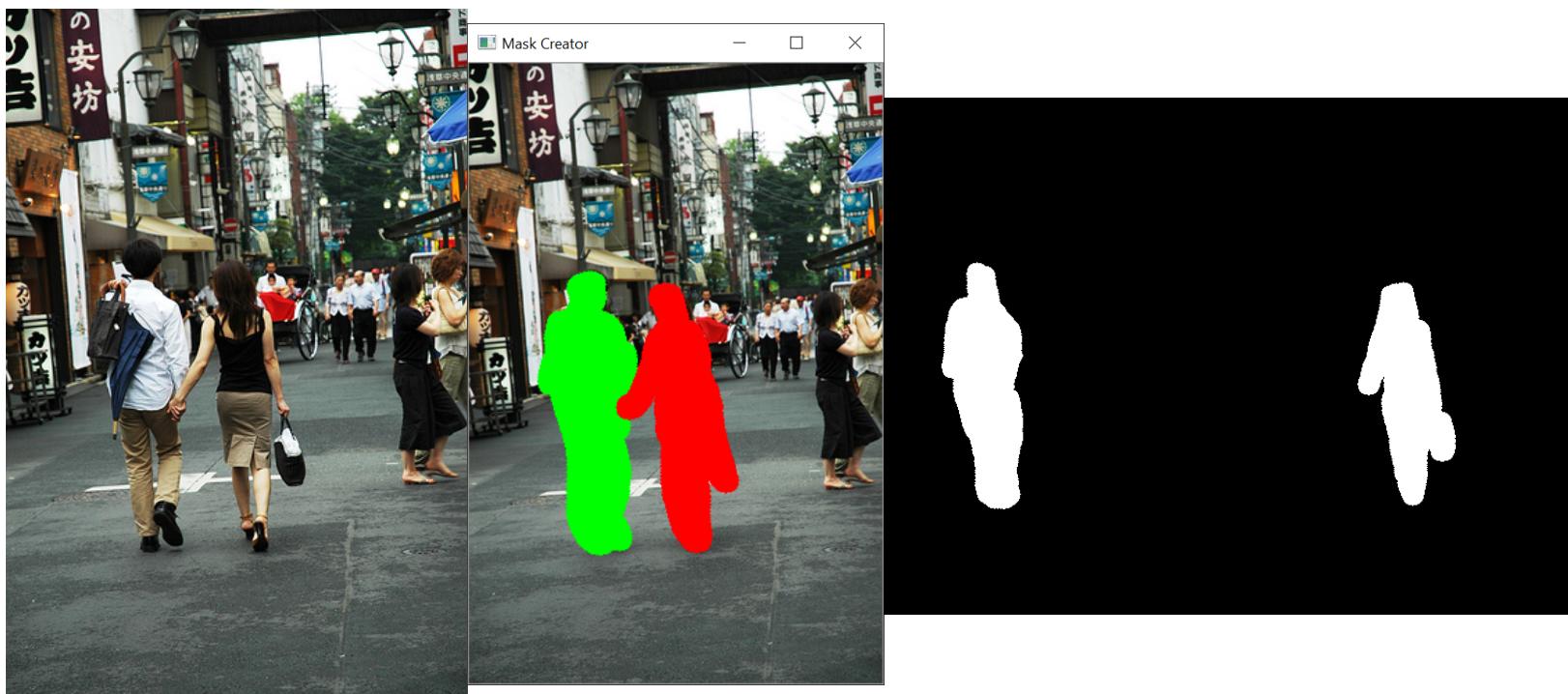
实现方法查看 `energy_forward()`。

2.5 目标保护和移除

2.5.1 介绍

目标保护和移除的原理是通过 mask，修改相应像素的能量值为一个极大或者极小的值，来控制细缝的走向。

我写了一个 GUI 工具来创建 mask，见 `cv_gui.py`。



物体移除的效果如下（移除女人，移除男人）：



运行本部分可使用 `demo2()` 和 `demo3()`。

可以打开 `remove.mp4` 视频查看运行过程。

2.5.2 实现

绘制mask的GUI代码 `cv_gui.py`，使用 opencv 实现窗口交互。

缩放的时候，在 `find_min_energy_seam_vertical` 函数中传入保护和移除 mask 即可。

2.5.3 GUI 的使用方法

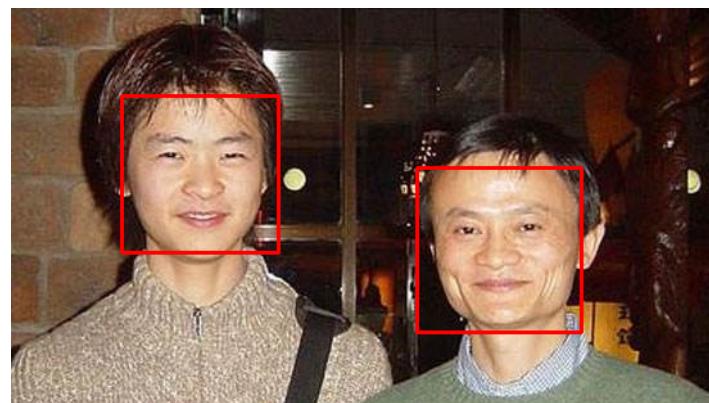
可以打开 `draw_mask.mp4` 视频查看GUI使用。

在代码中设置要打开的图像，运行。按 p 进入绘制保护mask模式，按 r 进入绘制保护移除模式，按s保存两个mask图片，按 ESC 退出。

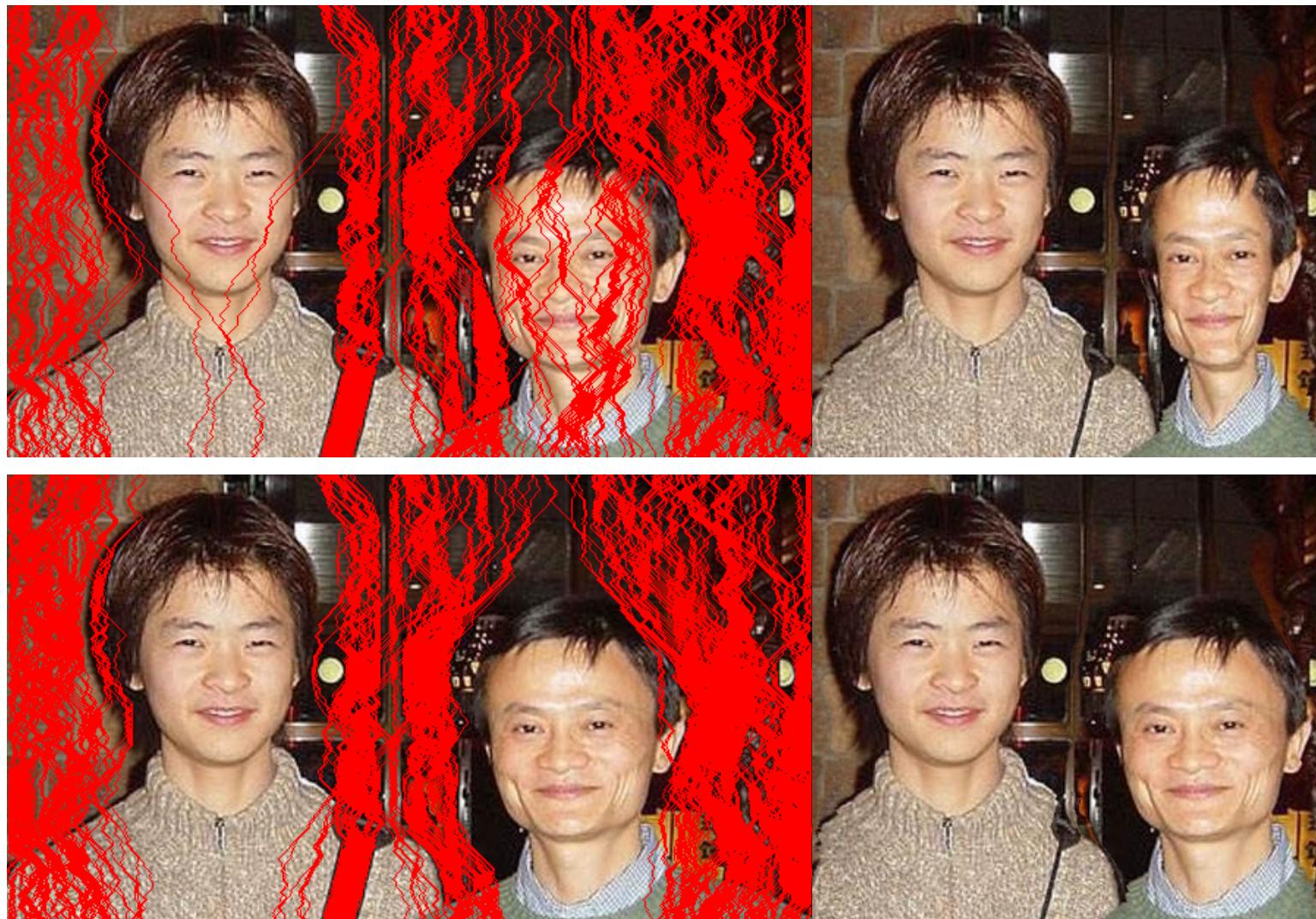
2.6 面部处理

2.6.1 介绍

面部处理的目的是让细缝不要经过面部造成扭曲变形。



如下是没有使用面部保护和使用面部保护的细缝分布：



人脸识别的检测使用了网上开源的配置文件，配合opencv实现，配置文件为 `haarcascade_frontalface_default.xml`。

运行本部分可使用 `demo6()`。

2.6.2 实现

`find_faces()` 使用 `cv2.CascadeClassifier` 进行人脸探测。

`make_protect_face_mask()` 生成保护 mask。

之后正常删减即可。

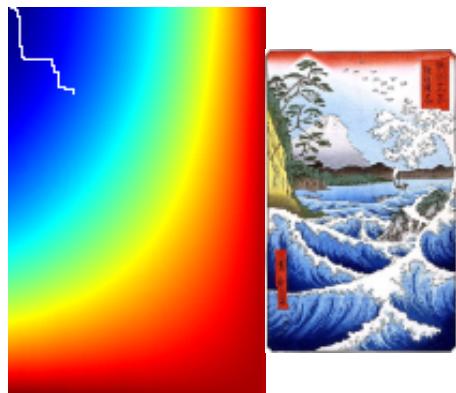
2.7 最优删缝顺序

2.7.1 介绍

在同时减小宽度和长度的时候，存在最优删缝顺序问题。解这个问题要使用动态规划的方法。

但是求解最优顺序是一个十分耗时的操作。

把动态规划的cost可视化出来如下所示：



白色是动态规划计算得到的路径。

运行本部分可使用 `demo5()`。

2.7.2 实现

本部分主要查看 `calculate_optimal_seam_order()` 方法。

`seam_order_cost` 是一个二维矩阵，`seam_order_cost[j][i]` 保存缩小高和宽 j 和 i 的最小代价。

`seam_order_back_trace` 保存路径信息。

3 实现功能总结

- 基础功能
 - 缩小
 - 水平缩小
 - 垂直缩小
 - 多种梯度
 - L1梯度
 - L2梯度
 - entropy
- 图像扩展
 - 迭代扩展（每次选1条）
 - 统一扩展
 - 分阶段扩展
- 目标保护和移除
 - 使用mask约束
 - 实现交互界面
- 改进能量公式
 - 前向能量
 - 处理面部
- 优化删缝顺序
 - 动态规划求解实现
 - 可视化顺序图
- 内容强调（Content Amplification）