媒体计算第二次大作业

计71 张程远 2017011429

0 实验内容

通过Seam Carving算法实现图像的智能缩放，将N x M的图片剪裁成N x M'的图片，使得缩放时图像的主要内容不变形。

1 基本算法

以细缝为竖直的情况为例，首先计算出输入图像需要删除的列数，作为整个算法的迭代轮数。对于每一次迭代，首先要计算出每个像素点新的能量，能量值我实现了下面三种计算公式：

$$E(i,j) = |I(i,j) - I(i,j-1)| + |I(i,j) - I(i,j+1)| + |I(i,j) - I(i-1,j)| + |I(i,j) - I(i+1,j)|$$

上式为第一种，计算四个方向上的梯度，并求和。

$$E_x = |I(i-1,j-1) - I(i-1,j+1)| + 2|I(i,j-1) - I(i,j+1)| + |I(i+1,j-1) - I(i+1,j+1)|$$

$$E_y = |I(i-1,j-1) - I(i+1,j-1)| + 2|I(i-1,j) - I(i+1,j)| + |I(i-1,j+1) - I(i+1,j+1)|$$

$$E(i,j) = E_x + E_y$$

上式为第二种，用sobel算子求梯度，取周围8个格子计算能量，其中上下左右四个方向的权值更高。以上两种都属于以梯度的方式计算能量。

$$E(i,j) = \sum_{i \in z} -e_i log(p_i)$$

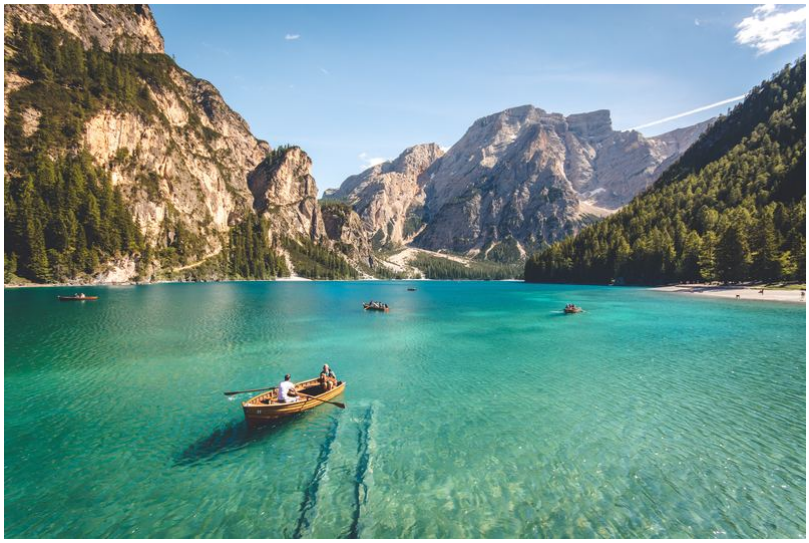
上式为第三种，首先把图像转化为灰度图，这样灰度值为0-255的整数；然后计算以i，j为中心的9*9正方形z内的熵，其中ei是灰度值，pi是每个灰度值出现的概率。

在计算完能量之后，接下来就是按照论文中的递推公式求出能量最小的细缝。这一部分实现在min_seam_vertical中，其中path代表每个像素点的能量来自于上一行的哪个像素。计算到最后一行之后，找到最小能量的位置，沿path矩阵向上回溯，就找到每一行要删除的像素的坐标。然后在seam_carving_vertical中去除细缝，得到新的图像，重复上述步骤。

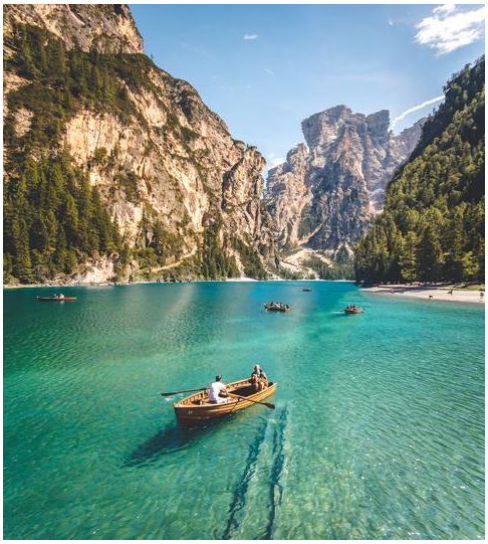
如果细缝是横向的，那么在进入整个迭代过程前，将图像旋转90度，转化成删除图像的列，算法处理完之后再把图像旋转回去。

2 实验结果

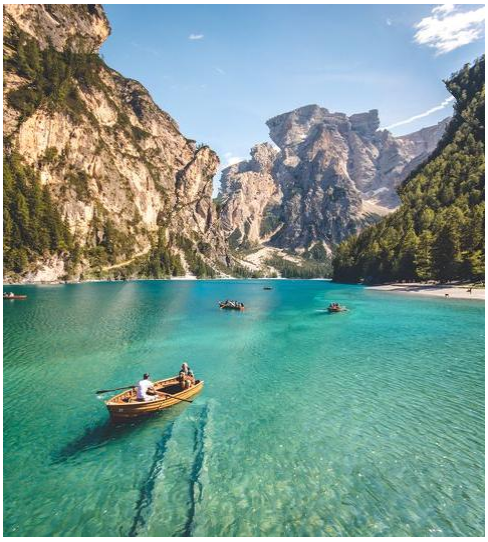
纵向细缝 原图：



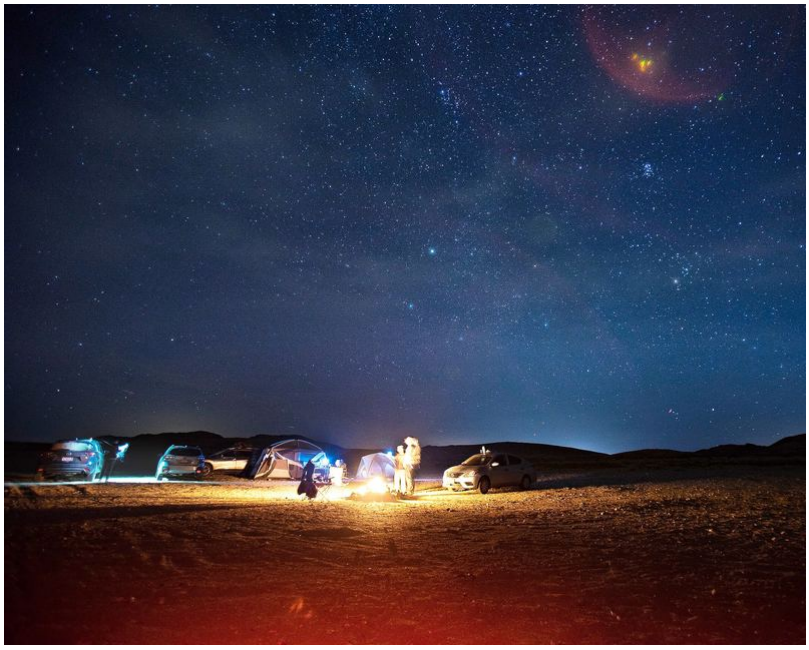
通过梯度能量函数让宽度缩减40%：



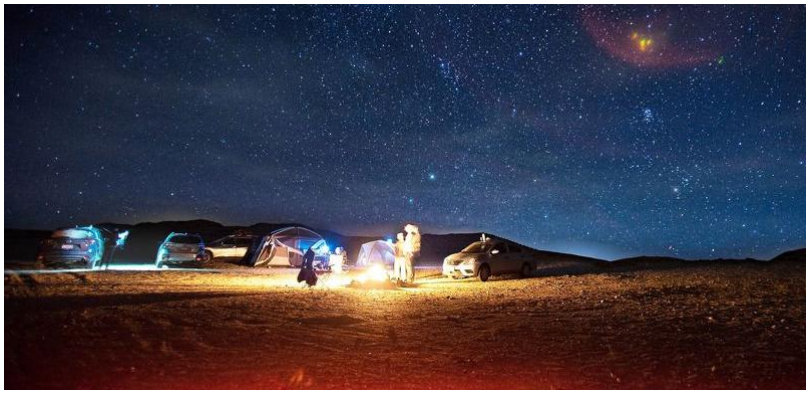
通过entropy能量函数让宽度缩减40%：



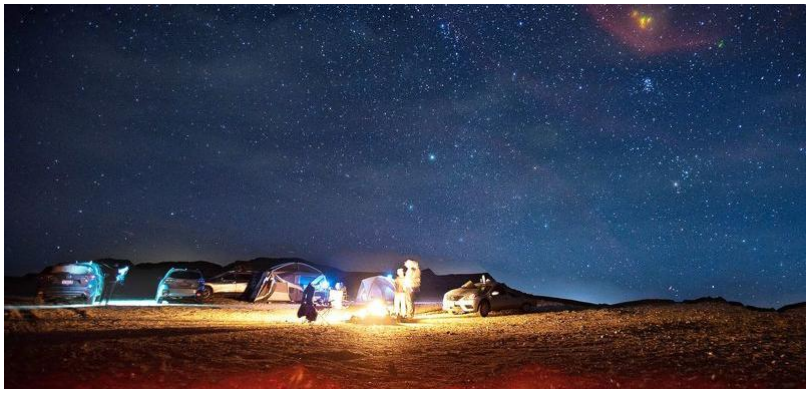
横向细缝 原图：



通过梯度能量函数让高度缩减40%：



通过entropy函数让高度缩减40%：

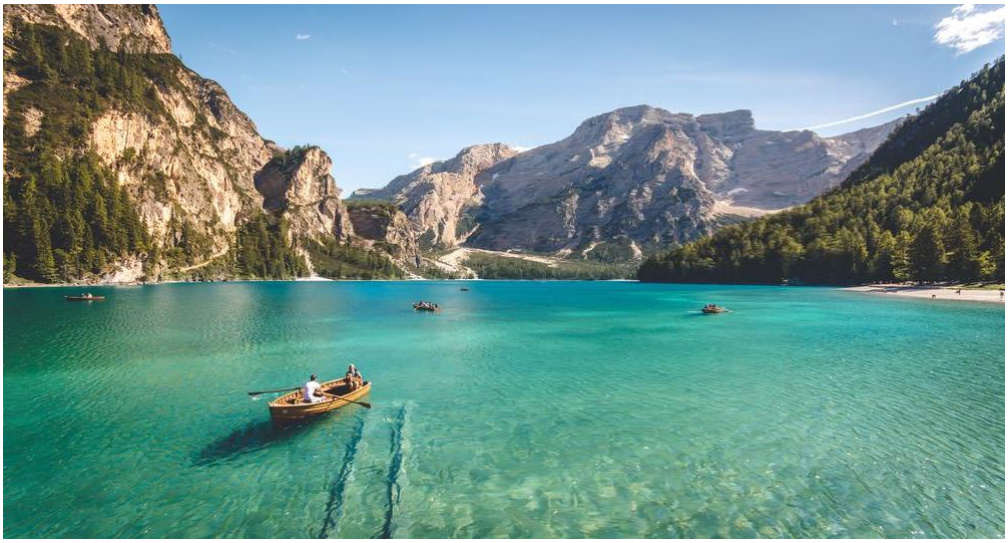


3 Bonus

3.1 图像扩展

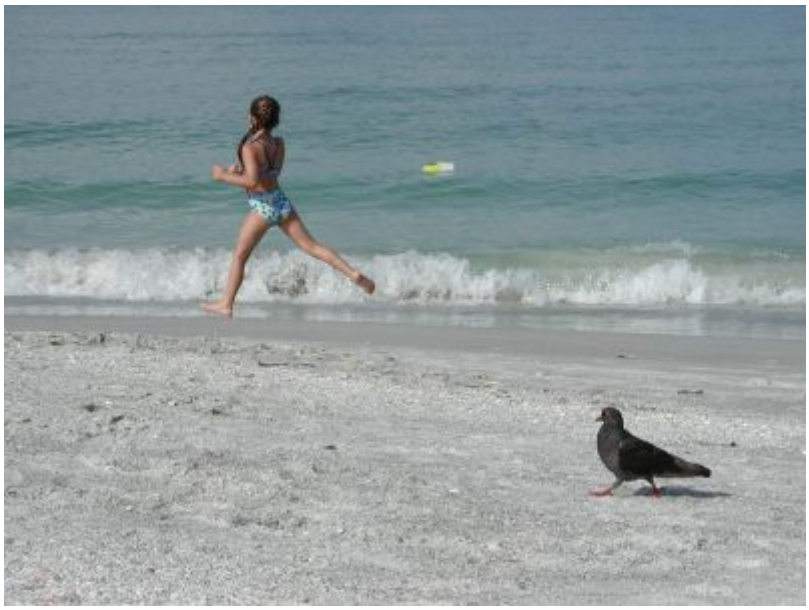
我实现了纵向和横向的统一扩展。首先我把图像拷贝一份，然后对这个拷贝的图像做seam carving，把每一轮的seam都记录在一个list中。一个seam是指在当前图像中，每一行要切掉的像素位置，因此每个seam应该都有src_img.shape[0]个元素。由于切割以后，像素相比于原图像的位置可能会变化，因此对于这个list中的元素i中第k行的索引值idx，需要判断元素0 - i-1中第k行的索引值有多少小于等于idx的，有几个则让idx加几。这样就给每个像素恢复了它在原图中的位置。然后接下来再遍历原图，在所有细缝经过的位置插入新的像素，值为该位置原像素和左面的像素的平均值。

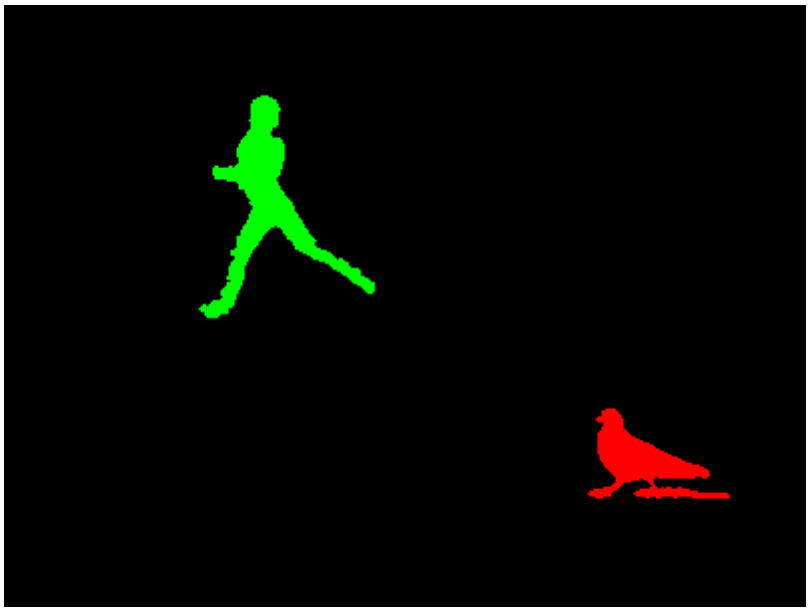
以下是我把纵向细缝结果中的原图宽度扩展到1000像素后的结果。



3.2 目标保护/移除

在mask中将需要保护的像素标为红色，需要移除的像素标为绿色，然后计算能量时红色位置的能量增加一个inf值，同时绿色位置的能量减去一个inf值，这样经过红色位置的细缝能量值为inf，一定不会被移除；经过绿色位置的细缝能量值为-inf，会被优先移除。注意移除时需要同步移除mask张对应位置的像素，以保证每次迭代时mask的位置正确。如下是我使用的图像和mask，以及保护/移除结果。





3.3 改进能量公式

3.3.1 前向能量

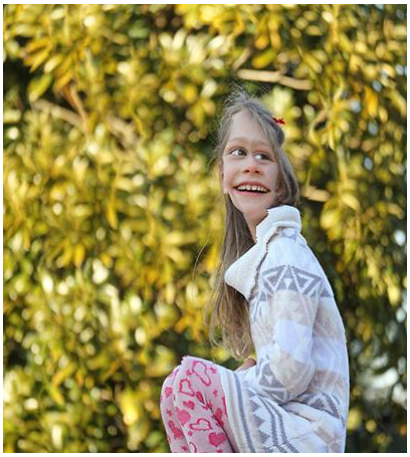
前向能量的目的是让留在图像中的能量最小，因此跟原本的能量公式计算没有关系，把P(i, j)置为0即可，然后按论文中的公式计算能量。我找到了论文中的图片样例并进行了复现，其中先对其做了原本的seam_carving算法，然后使用前向公式重新计算了一次，效果差距明显。

3.3.2 人脸判别

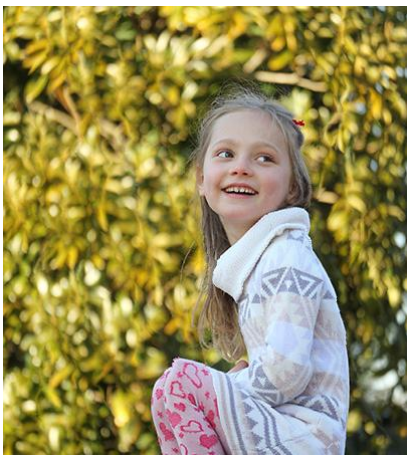
我在代码中使用了openCV的检测库。算法主要检测人脸的Haar特征，然后利用分类器把人脸提取出来。最后提取出的区域是一个代表人像的矩形，然后把矩形区域的能量增加inf值，从而对人脸做很好的保护。

算法实现在face_detect_cut函数中，使用了OpenCV提供的分类器XML文件haarcascade_frontalface_default.xml。效果相比于没有人脸检测的朴素版算法提升巨大。下面是我使用的测例。

使用朴素算法将图像的长度缩减40%，效果如下：



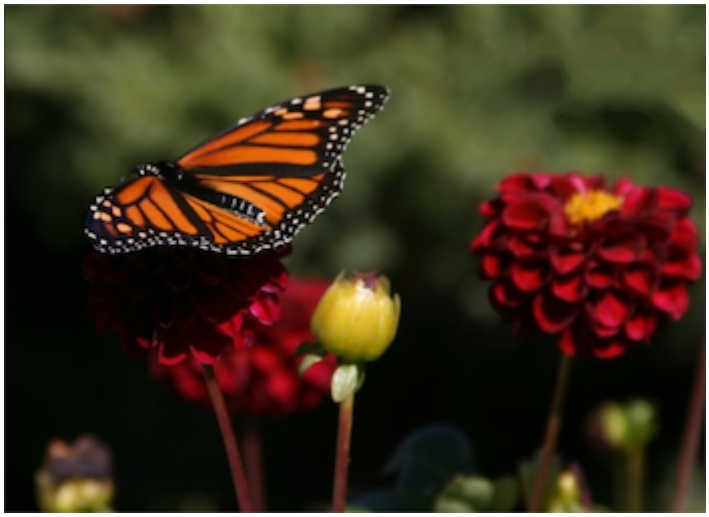
然后加上人脸保护以后，效果如下：

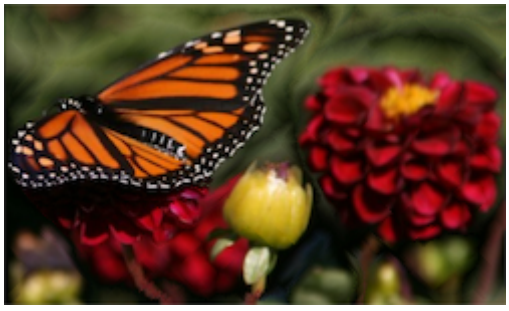


3.4 细缝顺序

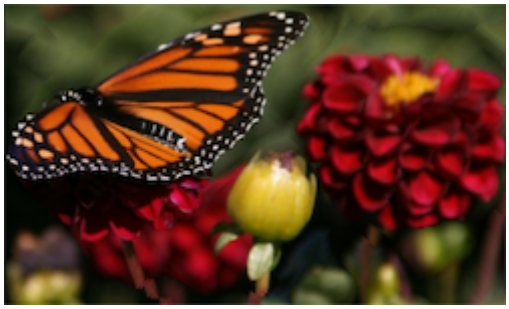
当图片的长宽都需要被剪裁的时候，我们需要计算出一个最佳的细缝删除顺序。由于删除r行c列到达的状态一定是由删除r-1行c列或者r行c-1列达到的，因此可以用递推的动规去计算中间状态。论文中的算法被实现在optimal_carve函数中，其中img_history记录每做一次删除操作之后图片的中间状态，这样每次计算时可以很方便的查询上一个状态的图片什么样子。对于每个状态，分别尝试从两个状态裁剪细缝到达现在的状态，然后保存能量图中最后一行的累计能量E，选择$T(i-1,j)+E_y$ 和 $T(i,j-1)+E_x$中能量较小的作为到达当前状态的能量值。

我找到了论文中介绍这一部分所用的原始图片，并把它放缩到150行 250列的形状，和论文结果几乎别无二致。



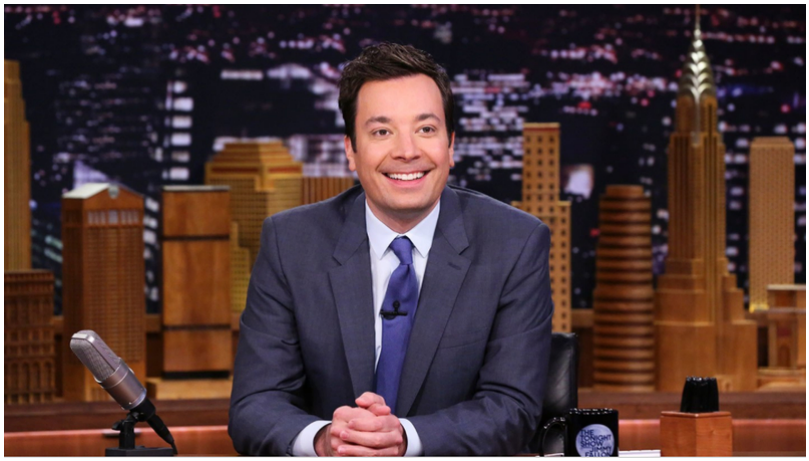


作为比较，先去除横向细缝然后再去除纵向细缝的结果如下，与论文结果基本相同。



4 不太好的结果

举一个下面的例子，即便是加上人脸检测结果也不够好。原因是虽然人像部分能量很高，但通过肩膀部分的细缝可以不经过人脸的部分。就算是整个人体受到保护，背景中的大楼也会发生扭曲。因此这里如果想取得好的剪裁效果是比较困难的，需要保护的物体太多，修改任意一个都会导致图片的不自然。



5 运行方法

在源代码最后有各个测试的结果，用引号注释符分别隔开。如果需要复现报告中的结果，去除测试上方下方的引号即可。