Computer Vision: Representation and Recognition
Assignment 2

211220028, 党任飞, dolphinrenfei@gmail.com

2024 年 5 月 19 日

1 Edge Detection

All codes are provided with this PDF file in the uploaded .zip file.
Available path: *EdgeDetection/prob_seg/solutions*.

1.1 edgeGradient

按照要求，对 RGB 通道分别进行高斯滤波（模仿作业一，使用 $ksize = 4 * sigma + 1$）；然后分别计算 x、y 方向的梯度，并用两个方向的 L2 范数作为该通道的梯度强度；最后用所有通道的梯度 L2 范数作为总的强度。

用分别计算出的 RGB 梯度强度，取其中最大的作为总的梯度的方向。

在 *edgeGradient* 函数中，调用 *canny* 核来获得 edge 的位置，然后在对应的位置上保留之前计算出来的梯度及其方向，作为简化版的 NMS 操作。

总体结果如下：

1. overall F-score = 0.570
2. average F-score = 0.622

部分示例如下：

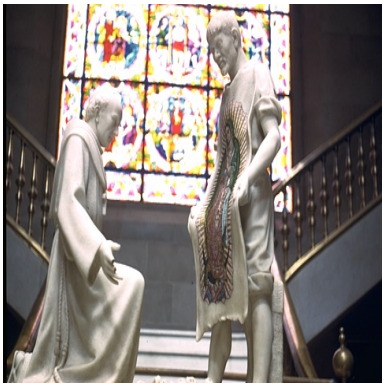

图 1: Original Image 24077.

图 2: Gradient Edge of 24077.

图 3: Original Image 62096.

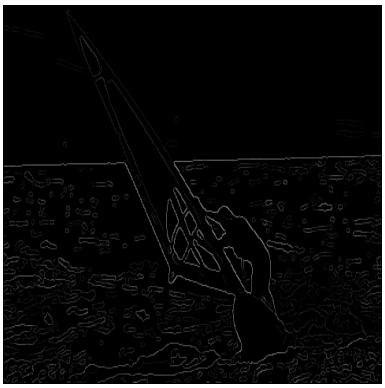

图 4: Gradient Edge of 62096.

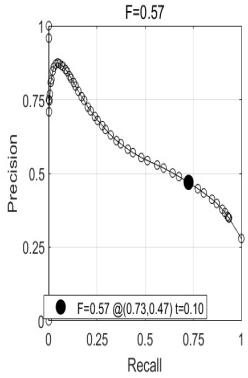

图 5: PR

1.2 edgeOrientedFilters

高斯核的设置和之前一样。根据课上讲过的知识，高斯核分别对 x、y 求偏导之后可以用三角函数对其进行复合，这样等价于不同角度的高斯核。选取角度为 $[0, 45, -45, 90, -90, 30, -30, 60, -60]$。

总体结果如下：

1. overall F-score = 0.603
2. average F-score = 0.635

部分示例如下（原图同上）（因为只采用了一个最大值方向作为返回值，所以总体的强度看起来很小，没做 rescaling）：

图 6: Oriented Edge of 24077.

图 7: Oriented Edge of 62096.

1.3 Ideas for improvement

以结果较高的 *edgeOrientedFilters* 为基础进一步改进。所以提交的代码就是改进之后的代码了。注释里仍有之前的代码，因此之前的结论可复现。

一开始尝试了使用 double 提升计算精确度，但是对实验结果的 F-score 等毫无影响。

发现手册里说我可能会用到 $rgb2hsv$ 函数，但是我写完整个作业别的地方都没用到，我估计这里要用。用了之后不是灰度图了，channel 数增加为 3，因此将强度设为三个 channel 各自强度的 L2-norm，得到如下最好结果：

1. overall F-score = 0.634
2. average F-score = 0.657

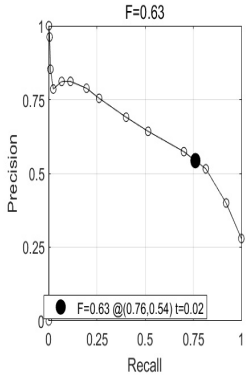

图 8: PR2

2 Imagesegmentation with k-means

All codes are provided with this PDF file in the uploaded .zip file.
Available path: $ImageSegmentation/$.

2.1 BASICS

所有的函数按照手册要求完成。值得一提的是计算 textons 的时候随机选取不超过 1k 个 pixel。并且完成了一个 *segmentMain.m* 脚本用于自动化测试。

2.2 Experiments-超参数选择

基于随机选取的超参数：

1. k = 10;
2. winSize = 5;
3. numColorRegions = 5;
4. numTextureRegions = 5;

有如下结果：

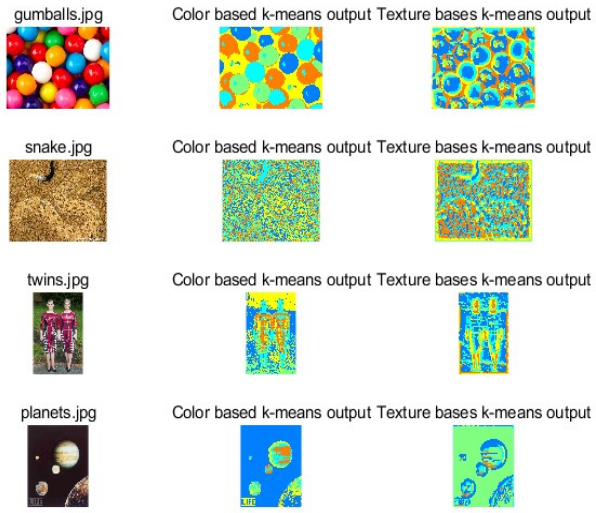

图 9: output1

当我设置 $numColorRegions = 8$ 时，报出了警告称 kmeans 未能在 100 次迭代内收敛，结合如下结果可以看出是颜色太多了，尤其是图三的任务上已经全是噪点了。同时发现 texture 可能设置过多，并且也可以适当提升 winSize。

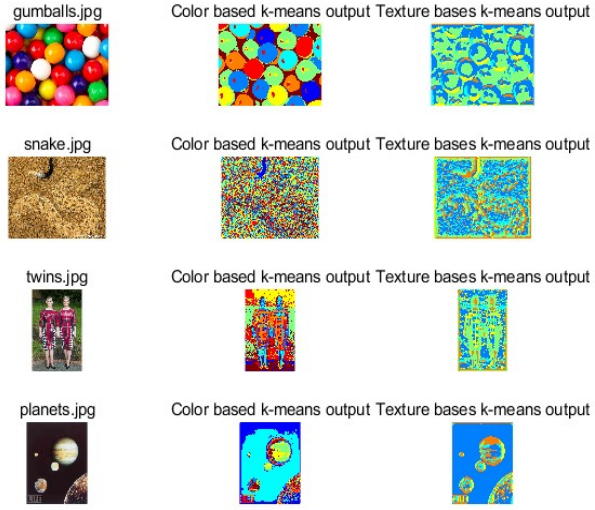

图 10: output2

最后在如下超参数：

1. k = 10;
2. winSize = 12;
3. numColorRegions = 6;
4. numTextureRegions = 4;

有如下较好的结果：

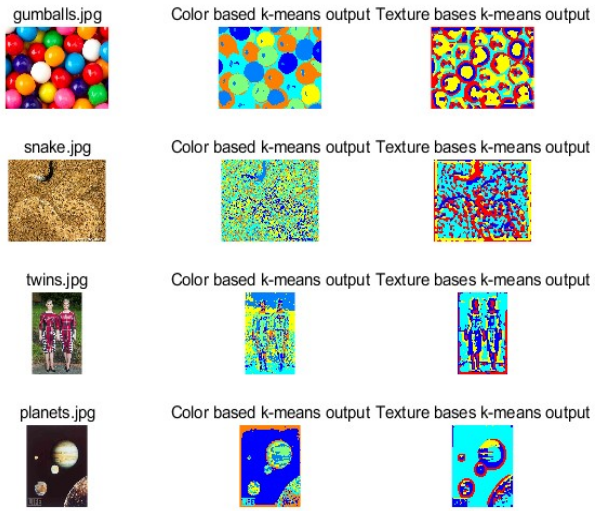

图 11: output3

观察图片，各有不同。例如 snake 需要更小的 winSize；gumballs 需要更少的 texture；planets 需要更少的 color。但是如果针对不同的图像设置不同的超参数，那就失去了计算机自动处理图像的意义了。

2.3 Experiments-winSize

在如下超参数下做实验：

1. k = 10;
2. winSize = 12;
3. numColorRegions = 5;
4. numTextureRegions = 5;

得到了 output4，与 output1 的唯一区别是 winSize。

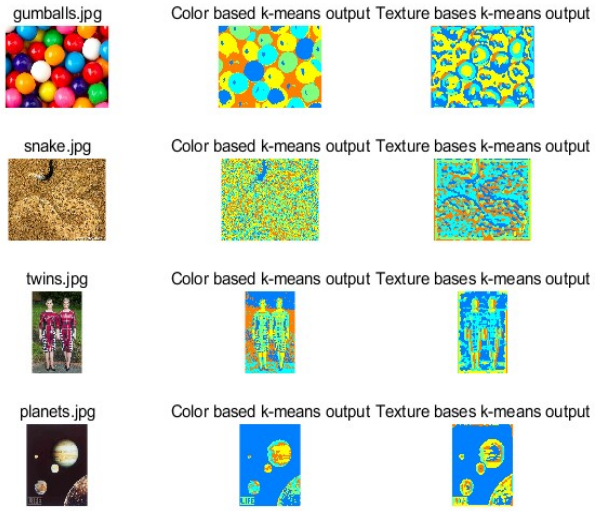

图 12: output4

1. 对于图片 snake，这是一个非常“细碎”的图片，output1 中可以看出将其分为了非常多的部分，而 output4 中相对小。这一点在其 texture-based 特征中尤其明显。
2. 另外三张图片都是包含大面积色块的，差异不明显。
3. 由此可知，进一步提升 winSize 只会使得图片中的分割块相对变大。

2.4 Experiments-filterBanks

调用 *displayFilterBanks* 函数可以得到下图：

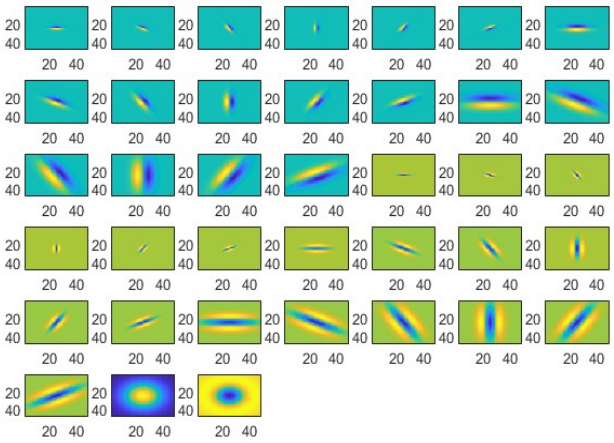

图 13: FilterBanks

根据上图，选取了所有接近于横着的 filter，[1,2,6,7,8,12,13,14,18,19,20,24,25,26,30,31,32,36]。用这些 filter 得到如下结果：

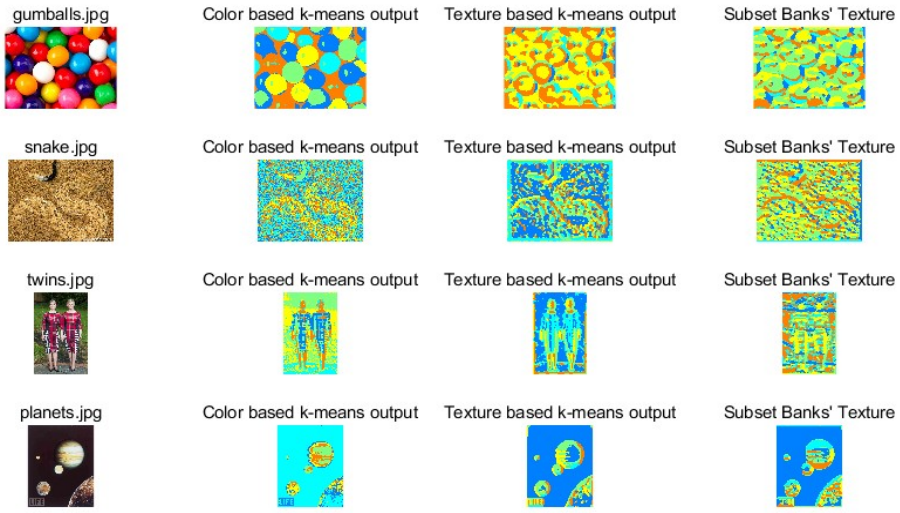

图 14: subset

仔细研究后两列，是全体 banks 和横着的 banks 子集的比较。可以明显看出，后者得到的 textures 也是趋向于横着，尤其以 twins 图片最为明显，人的腿都被横着切开了。