

数字图像处理综合作业 3——交互式图片分割

陈昭熹 2017011552

2020 年 1 月 13 日

目录

1 运行说明	3
1.1 文件结构	3
1.1.1 基础算法类	3
1.1.2 功能函数	3
1.1.3 GUI	3
1.2 使用方法	3
2 算法流程与实现	4
2.1 后端算法	4
2.1.1 SLIC 超像素分割	4
2.1.2 超像素个数自适应计算	5
2.1.3 Gabor 特征融合	7
2.1.4 超像素分水岭	8
2.1.5 超像素连通性增强	10
2.2 GUI 设计	10
2.2.1 曲线标注前背景	10
2.2.2 交互逻辑	11
3 中间流程与结果	12
3.1 效果图	12
3.2 部分中间过程	14

目录	2
----	---

3.2.1 超像素数的影响	14
3.2.2 自适应梯度平滑效果	15
3.2.3 Gabor 特征效果	17
3.2.4 分水岭的实用性	17

1 运行说明

1.1 文件结构

1.1.1 基础算法类

SLIC_Proc.m SLIC 超像素分割算法

useWaterShed.m 基于超像素分割的分水岭算法

adaptive_ks.m 自适应计算所需超像素个数

1.1.2 功能函数

freehanddraw.m 曲线标注前背景交互接口

Enforce...ty.m 超像素分割后处理接口，增强连通性

1.1.3 GUI

segmenter.mlappinstall GUI 安装包

1.2 使用方法

按照 GUI 的按钮标签提示便可以轻松上手。处理一张图皮的整体流程是，首先加载图片，此时会弹出后续进行前背景标注的窗口（标号 99），此时选择超像素分割性能参数后，点击开始计算超像素，其过程会显示在主窗口右侧的图片框内。计算完成后标注模块会被点亮，可以点击标注自动切换到刚刚弹出的窗口内进行标注。标注时的操作注意事项请见2.2.1。一旦前景背景均有标注后，会在主窗口实时更新当前的分割结果，用户若对当前结果满意，则可以点击确认蒙版完成分割。

2 算法流程与实现

2.1 后端算法

后端算法整体流程如下所示：

Algorithm 1: 后端算法

Input: 待分割图片

Output: 分割结果

- 1 自适应计算超像素个数
 - 2 融合 Gabor 特征的 SLIC 超像素分割
 - 3 求取超像素梯度图
 - 4 自适应阈值滤波
 - 5 超像素分水岭
 - 6 用户交互式分割
-

下面几节将分模块介绍算法的实现细节。

2.1.1 SLIC 超像素分割

超像素分割算法受到了前人工作 [1] 的启发，在此基础上针对本文所面对的任务做了简化与更合适的处理，这部分的改进会在后面章节提到，下面给出基于原文框架实现伪代码，改进部分后文会详细阐述。值得注意的是，算法的聚类分割部分执行次数设置为经验值 10 个循环，考虑到计算速度和效率的原因，并没有采用原文中的残差度量。实践中，10 个循环对于绝大部分图像来说就已经严格收敛了。

值得注意的是，由于本文在实现算法过程中进行了计算上的优化，并充分利用了 MATLAB 的向量化特点，使得算法能够使用双精度浮点，在 LAB 色彩空间中进行求解，且时间效率在可接受的范围内。

另一点是，在实现过程中，两个像素点之间的距离度量使用了原文中实际应用的距离定义，表达式如下所示：

$$D = \sqrt{d_c^2 + \left(\frac{d_s}{S}\right)^2 m^2} \quad (1)$$

Algorithm 2: SLIC 超像素分割算法

Input: I-Image, ks-Superpixels' number, M-hyper param

Output: Superpixels label

```

1 /*Initialization*/
2  $S \leftarrow \lfloor height \times width / ks \rfloor$ 
3 Initialize cluster centers  $C_k$  by sampling pixels at regular grid
   steps  $S$ 
4 label  $l(i) \leftarrow -1$  for each pixel  $i$ 
5 distance  $d(i) \leftarrow \infty$  for each pixel  $i$ 
6 /*Assignment*/
7 while not 10 times do
8   for each cluster center  $C_k$  do
9     for each pixel  $i$  in  $2S \times 2S$  region surrounding  $C_k$  do
10       Computer distance  $D(C_k, i)$ 
11       if  $D < d(i)$  then
12          $d(i) \leftarrow D$ 
13          $l(i) \leftarrow k$ 
14       end
15     end
16   end
17   /*Update*/
18   Compute new cluster centers, update  $C_k$ 
19 end

```

2.1.2 超像素个数自适应计算

使用了原始的 SLIC 算法后发现，作为输入参数之一的超像素个数 ks 对于分割效果影响很大。ks 太多会导致过分割，产生很多杂乱的超像素或无意义的分割，同时计算量过大；ks 太少会导致欠分割，会出现有些需要分割的边界没有被超像素分隔开，这会导致后面的懒人分割法效果较差。同时注意到，不同的图像、不同的场景下，效果较好的 ks 是完全不一样的，不存在一个通用的经验值。因此根据图像的自身特性和特点自适应的计算所需超像素个数以求最好的分割效果便是一个迫切的需求。

事实上，从经验来看，图像越复杂代表着其纹理越复杂，需要分割的部

分也就越多，反之图像越简单需要分割的部分也就越少。从这一点出发，可以建立起超像素分割个数与图像复杂度的一个关系。如果能够量化图像复杂度，那么就可以针对每一幅图像自适应的计算出所需的最佳超像素个数。这部分的内容受到前人工作的 [2] 一点启发，从两个切入点考量图像复杂度：信息熵和灰度共生矩阵。

信息熵表征了某种特定信息在图像中出现的概率，作为一个求和式其值越大，代表着图像越复杂，反之越简单，使用 $E_{entropy}$ 表达信息熵，则定义如下：

$$E_{entropy} = - \sum_{i=1}^k \frac{n_i}{N \log(n_i/N)} \quad (2)$$

其中 n_i 表示灰度值为 i 的像素点个数， N 为图像总像素点个数，应当有 $N = height \times width$ 。信息熵将被作为图像复杂度的一部分度量。

仅仅用信息熵还不能准确表征图像复杂度，从形式上可以看出，信息熵仅仅包含着全局信息，但在分割任务中更关心的是局部纹理特征，关注邻域内的像素分布特点，关注像素的空间相关特性。基于这一点，选择使用灰度共生矩阵来描述图像的空间相关特性，进而能够表征局部纹理，纳入复杂度量内。

得到灰度共生矩阵 $M([h \times w]$ 维) 后，利用一些典型度量特征参数来纳入到图像复杂度的计算中，本文所使用的有 E(能量)、Contrast(对比度)、Covariance(协方差)，下面逐一介绍其定义。能量 E 实际上是一个二范数意义下的度量，将灰度共生矩阵内的元素平方后求和，本质是求出了所有可能的相邻像素对分布的强度之和，定义式：

$$E = \sum_{i=1}^h \sum_{j=1}^w M^2(i, j) \quad (3)$$

对比度 Con 强调找到的像素对之间的差距，本质上是沿着生长方向，在单位生长步长上的灰度梯度，这可以很好的表示图像纹理特征，较比直接用梯度算子求梯度效果更好。定义式：

$$Con = \sum_{i=1}^h \sum_{j=1}^w [(i - j)^2 M(i, j)] \quad (4)$$

协方差 Cov 客观的反映了像素对之间的相关性，其值越大表明相关性越强，在复杂度中应当起到负向作用。定义式与一般意义上的矩阵协方差相同，在此不赘述。

最终由以上多种考量综合在一起，通过加权的方式得到图像复杂度度量 T :

$$T = E_{entropy} + Con - E - Cov \quad (5)$$

实现过程中使用生长方向为 0 度的灰度共生矩阵，灰度范围为 8，生成之前需对图像进行归一化处理。

得到图像复杂度后，根据图片尺寸即可确定最佳超像素个数 ks :

$$ks = \frac{height + width}{T} \quad (6)$$

当然，实现过程中 ks 需要进行取整，下面给出本节流程的伪代码:

Algorithm 3: 自适应 ks 计算

Input: I-Image

Output: ks-Adaptive Number of Superpixel

- 1 Compute $entropy$
 - 2 Generate Gray Level Co-occurrence Matrix
 - 3 Compute energy E
 - 4 Compute Contrast Con
 - 5 Compute Covirace Cov
 - 6 $T \leftarrow entropy + Con - E - Cov$
 - 7 $ks \leftarrow \lfloor \frac{height \times width}{T} \rfloor$
-

2.1.3 Gabor 特征融合

为了改进超像素分割结果，让超像素在局部能够更加贴合图像的自然边缘，或符合语义上的分割倾向，本文在原 SLIC 基础上融入了 Gabor 特征作为辅助特征，作为一部分度量增加到超像素分割的像素点之间距离的计算过程中。

使用四个方向 0, 45, 90, 135 的 Gabor 滤波器，针对不同图像的纹理特征设定不同的步长和步数，对图像的灰度图进行滤波，将得到的特征图附加在原图的第三个维度上，作为其若干个新的通道输入到 SLIC 求解器中进行聚类求解，完成超像素分割。诚然，加入新的特征应该能够起到正面的辅助作用，但是如何将新的特征融入到原特征度量中，不破坏 SLIC 中原有度量的出色作用，又能在一些极端纹理情况下发挥出辅助特征的作用，是一个难题。本文最终的做法是加权，在计算像素点之间的距离过程中，让 (1) 式定

义的距离作为主要因素，加入一定权重的各通道 Gabor 特征 D_G 从而产生最后的距离度量作为像素距离的判据。具体如下：

$$D' = \sqrt{D^2 + k\Sigma D_G^2} \quad (7)$$

需要注意的是，由于波长会随着步数增加而改变，不同的 Gabor 特征通道的绝对数值是没有可加性的，因此需要进行归一化操作之后，再进行上式的特征计算。否则会使得波长大的特征成为度量中的主要成分，反客为主，导致传统 SLIC 的 LAB 三通道特征无法发挥作用，反而降低分割效果。

2.1.4 超像素分水岭

为了进一步提高图像的分割性能，减少用户的交互次数，在较好的超像素分割结果上，直接使用分水岭算法。充分利用分水岭算法速度快、轮廓提取效果好的特点，进一步缩小图像分割的范围，为最后一部用户交互扫清障碍。但直接使用传统分水岭算法往往会由于过分割导致图像分割结果无法使用，而如果使用 SLIC 算法的聚类结果，则通过聚类的方法首先利用了图像的冗余信息，为分水岭算法过滤掉了不必要的噪声，可以抑制分水岭算法的过分割的产生。因此对 SLIC 生成的超像素再进行分水岭算法处理，几乎就可以得到所需的分割结果。

利用超像素分割结果的标注图像 L ，将原图中每一个超像素区域进行平滑处理，即把每一个超像素内的像素设置为超像素的均值，从而将超像素内部梯度抹平，图像中仅仅在每个超像素的边界存在梯度。利用 Sobel 梯度算子计算平滑后的图像的梯度图，将该梯度图作为输入，使用分水岭算法得到粗分割结果。粗分割结果一般能够较好的贴合感兴趣的分割区域，只会在一些细节点出现错误的分割结果，但这些细节点只需要最后通过用户局部的交互标注就可以快速解决。因此使用分水岭算法处理超像素分割结果的梯度，相比于直接使用超像素分割结果让用户交互分割，极大地减少了用户的标注次数，从而提升了整体的算法效率和性能。

值得注意的是，即使使用超像素分割的梯度作为分水岭算法的输入，该梯度图中仍会存在一些噪声或伪影，或不连续的梯度线。这种影响的来源是超像素分割时，某些区域的连通性较差或特征变化剧烈。而这会让分水岭算法在执行过程中“修建错误的大坝”或者不同区域之间“漏水”，导致意想不到的分割结果。为解决这一问题，本文使用两个解决方案，一是对梯度图进行自适应的阈值滤波，去除不需要的梯度；二是对 SLIC 超像素分割结果

进行连通性增强，将面积小于设定阈值的超像素同化到周围区域内，同时修建并平滑边缘，这一部分将会在下一节介绍，下面阐述对梯度图的自适应阈值滤波。

由于 Sobel 算子处理后的梯度图是图像中原始的梯度表现，即使梯度很小的地方也会有微小的梯度响应，这会让梯度图出现一些不必要的“盆地”。起初笔者想借鉴 canny 算子的思路，进行局部极值抑制，但效果不佳。注意到由超像素得到的梯度图具有稀疏的特点，因此想到用简单的阈值抑制方法应该就可以滤掉一些弱响应，保留响应较强的边缘。因此定义阈值如下：

$$\text{threshold} = m_G + n\sigma_G \quad (8)$$

其中 threshold 即所求的梯度图像阈值， G 为通过 sobel 算子得到的原始梯度图， m_G 为原始梯度图均值， σ_G 为原始梯度图方差， n 为可调节的权重。

利用 (8) 对原始梯度图进行滤波，仅保留大于阈值的梯度，就可以很好的屏蔽掉弱梯度响应，增强分水岭的效果。下面给出这部分的流程伪代码：

Algorithm 4: 超像素分水岭算法

Input: I-Smooth Image based on Superpixel

Output: wt- Watershed result

```

1 /*Compute Gradient*/
2 filter ← Sobel(3, 3)
3 /*Adaptive Filter*/
4 G ← filter(I)
5 mG ← avg(G)
6 σG ← var(G)
7 threshold ← mG + nσG
8 for each pixel i in G do
9   | if i < threshold then
10    |   | i ← 0
11   | end
12 end
13 /*Watershed*/
14 wt ← watershed(G)

```

2.1.5 超像素连通性增强

事实上，使用前文所述改进后的 SLIC 算法，对于某些分割情形，会出现超像素边缘贴合不紧密，产生若干面积较小的超像素，边缘杂乱等等问题。这些问题的产生是由于在预先设置超像素种子时采取均匀播撒的策略，即使采取了基于局部 3×3 邻域的梯度优化算法，保证种子不落入局部最优点，但这并不妨碍复杂纹理背景下超像素的边缘杂乱，局部产生孤岛。例如在处理斑马图片过程中，由于背景是杂草，在光照不均匀的条件下，存在复杂的梯度变化，这会导致不理想的超像素分割效果（详见后文效果图）。为了解决这一问题，通过形态学连通性处理和一些局部判据对原始超像素进行连通性增强，保证不存在孤岛超像素、不存在尺寸过小超像素，同时平滑边缘。

算法主体是维护两个标签，分别存放邻接标签和当前标签，通过从上到下从左到右的顺序遍历来重新定义原始超像素图像的标签，分别处理区域内的标签，存在连通性则同化标签，尺寸过小则与相邻区域合并。值得注意的是，如上的遍历顺序自然地将连通性带入到标签判断中，而不需要额外维护一个连通性状态，并减少了不必要的邻域内迭代，这样精妙的思路受到了 SLIC 论文的工程实现的启发。

2.2 GUI 设计

2.2.1 曲线标注前背景

为了保证良好的用户体验，更加逼真的模仿 matlab 工具箱中的 Image Segmenter 的使用体验，在用户标注前背景的交互逻辑上，本文实现了可连续标注的曲线标注接口，能够让用户对分割进行精细控制，以达到良好的分割效果。

通过使用 matlab 的 figure 窗体环境，利用 gca、gcf 等函数获得鼠标位置，首先将图像所在平面视为连续的二维平面，通过鼠标位置的反复采样得到连续的浮点型坐标值，并绘制到图像上，避免了像素坐标的离散特性导致难以处理平滑曲线绘制的问题。

曲线标注模块的交互逻辑如下所示，该逻辑会重复执行直至主窗口中用户切换模式：

鼠标左键 设置标注起点

移动鼠标 松开左键拖动鼠标绘制曲线

鼠标右键 设置标注终点

2.2.2 交互逻辑

GUI 整体交互逻辑可以从界面内简明的按钮标签得知，此处需要阐明整体的操作流程以及几点注意事项。需要注意的是，为了保证连续多次标注，并实时更新 mask 的体验，交互式分割窗口选择通过弹窗的方式独立于 GUI 外，以便于控制和交互。几点注意事项：

- 1 在标注过程中，请注意不要将笔迹移动到图像边界以外，在回到主窗口选择功能时，请将当前笔迹通过鼠标右键终止。
- 2 由于 matlab 的 GUI 不提供高级中断功能，导致在切换前景标注与背景标注模式时，会滞后“一拍”，即切换标注模式后，回到标注窗口当前的标注模式仍是切换前，需要完成一条笔迹后才能切换到需要的模式，可以通过直接点击鼠标右键来完成这一切换。
- 3 暂不提供擦除功能，如果进行了错误的标注，请重新加载图片。

3 中间流程与结果

3.1 效果图



图 1: 图片 1 结果

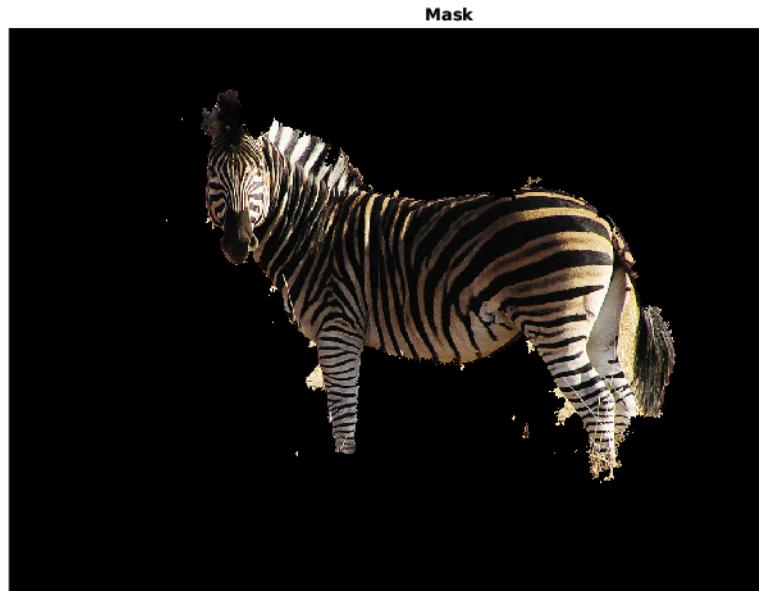


图 2: 图片 2 结果

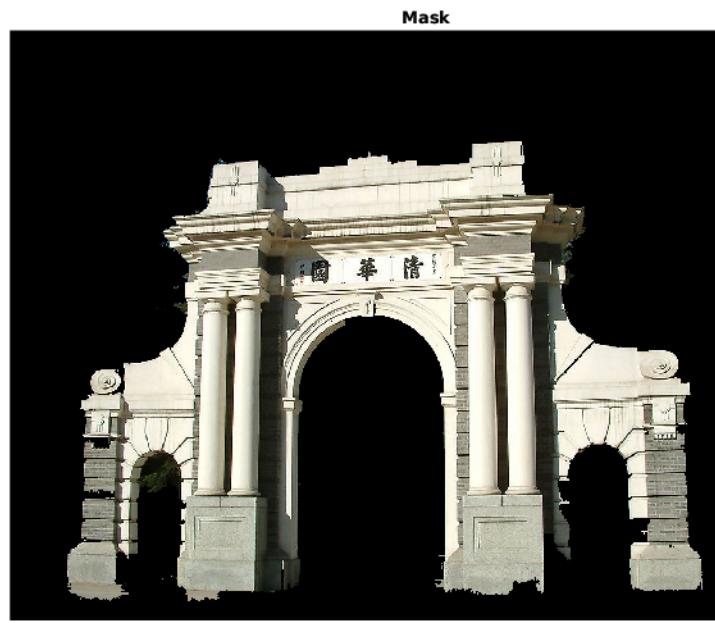


图 3: 图片 3 结果

3.2 部分中间过程

3.2.1 超像素数的影响

超像素过少会产生欠分割，需要的分割边缘不在超像素边界上；超像素过多容易产生过分割，边缘杂乱，且计算时间过长。

下图可以看出三种 ks 取值的不同结果，小的超像素数任务边缘（左胳膊）并没有在超像素边界上，这会导致分割并不能如意。而超像素过多会导致分割的边缘较为杂乱（图中许多超像素边缘已经出现了打卷、成环的现象。这也体现了自适应计算 ks 的必要性。



图 4: $ks=100$ vs 自适应计算 $ks=222$ vs $ks=1000$

可以看到以斑马图为例，在超像素较小时，不能较好的将斑马边缘分割到超像素边界上（马臀部就未被分割到边界上），而超像素较多时已经出现了肉眼可见的不合理边缘，且在马肚子位置也出现了不合理的聚类，导致其边缘与背景被分到同一区域。而自适应计算的结果再次较好的完成了任务，平衡了效果和计算量。



图 5: $ks=100$ vs 自适应计算 $ks=196$ vs $ks=1000$

3.2.2 自适应梯度平滑效果

为了让分水岭算法获得更好的效果，采用自适应平滑超像素梯度图，效果如下所示，每张图像的滤波力度不同恰恰印证了自适应的阈值计算对于不同图片产生的不同效果。

由于原图中伪影较少且噪声不明显，因此计算出的阈值也较小，被滤除的梯度也较少。

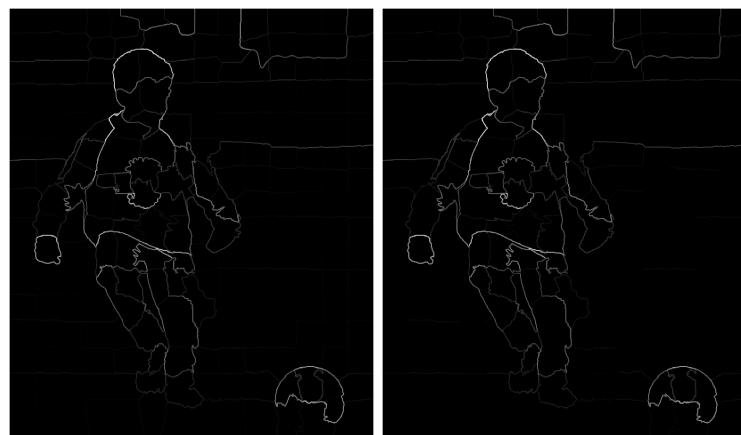


图 6: 平滑前 vs 平滑后

而斑马图的超像素梯度则噪声较多，自适应滤波后可以看出效果较好。

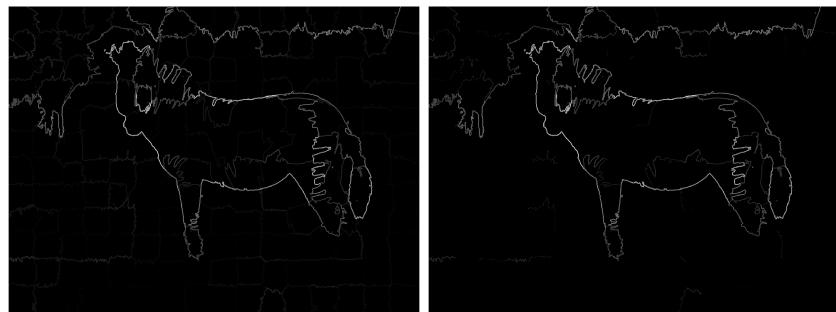


图 7: 平滑前 vs 平滑后

相同的情况也出现在二校门图片中，可以看到自适应滤波将门中的伪影和背景中树叶产生的不需要的梯度进行了滤除。

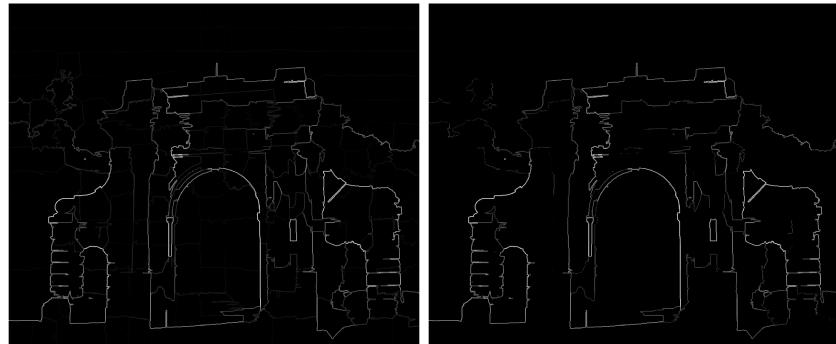


图 8: 平滑前 vs 平滑后

在实现过程中使用三通道梯度计算方法进行了一次可视化，为的是清晰地了解彩色图片各通道的梯度变化，如果能够将这些因素分别单独的考虑，可能会获得更好的效果（例如能够较好的区分红色衣服和背景的跑道）。



图 9: 彩色三通道梯度图

3.2.3 Gabor 特征效果

融入 Gabor 特征后，对于某些图片能够提高其分割效果，此次任务中的斑马图就是特别适合增加 Gabor 特征辅助分割的情景，原因是斑马的斑纹局部呈现周期性。而在实验过程中，增加了 Gabor 特征后，会使得原本不能分割出的边缘被超像素轻易地分割出来，下图中可以看到效果之明显。增加 Gabor 特征后，聚类器能够更好的分辨斑马颈部的条纹，即使因为光度变化导致其颜色与背景相似，但是超像素在此处还是很好的在边缘闭合，将原本没分开的前背景分开了。

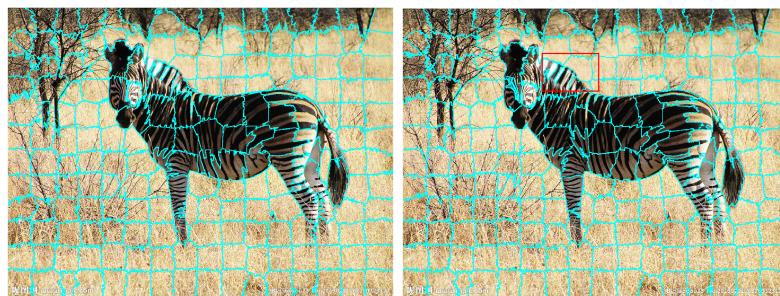


图 10: 未融入 Gabor vs 融入 Gabor 特征

3.2.4 分水岭的应用性

在增加超像素分水岭处理后，在获得相同的分割效果前提下，能够显著地减少用户标注的笔迹数量，这证明了超像素分水岭的应用性，能够加快分割处理。

首先以斑马图为例，获得超像素分水岭粗分割效果如下：



图 11: 超像素分水岭结果

利用上面的结果直接交付用户标注，可以获得如下效果的分割以及标注数量：

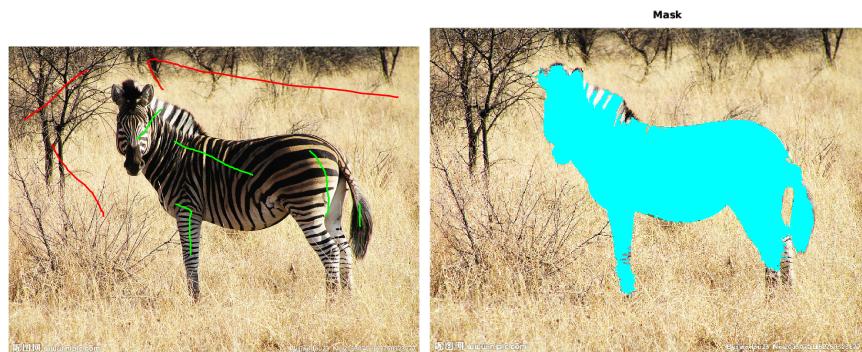


图 12: 使用 Watershed-标注数量与分割结果

而不使用超像素分水岭处理，则需要更多的用户标注来获得相同的效果：



图 13: 无 Watershed-标注数量与分割结果

而以第一张小冯图片为例，通过超像素分水岭处理后，只需要前景背景各一个笔迹，就可以完成较好的粗分割。

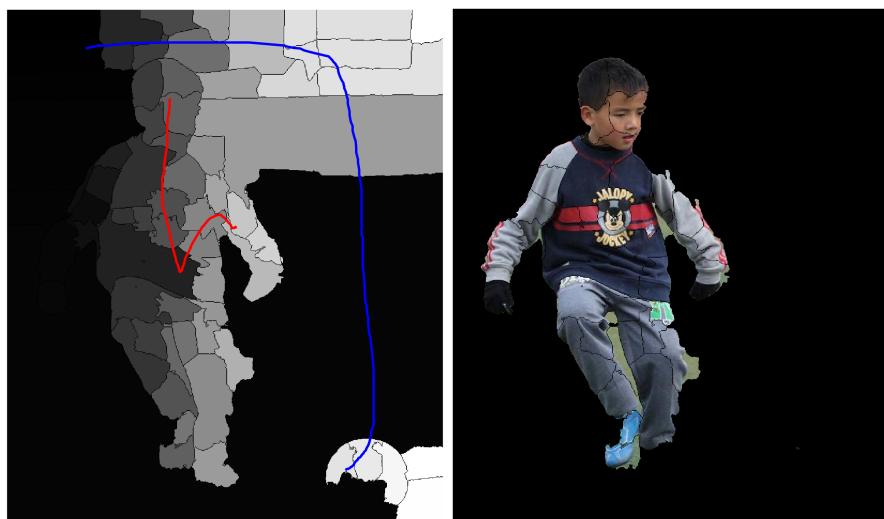


图 14: 分水岭结果和分割结果

值得一提的是，如果想要获得更精细的分割效果，还是需要回到原超像素分割的基础上进行细节标注。

参考文献

- [1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2274-2282, Nov. 2012.
- [2] Zhi-qiang, H., Meng-qi, Z., Wang-sheng, Y., You-mou, L., & Sugang, M. (2019). Color image segmentation based on SLIC and watershed algorithm.