Graph-cut 代码实现报告

杨训迪 1120172169

1 方法原理

Graph-Cut 是由 Yuri Y.Boykov 等人于 2001 年提出的交互式的图像分割算法，是一种十分有用和流行的能量优化算法，主要用于前背景分割、抠图等。

该方法的主要思想是和图的最大流最小割问题相关联，首先构造一个无向图 G=<V, E>表示要分割的图，该无向图的点由图片全部的像素点和两个端点 s，t 构成，边由两种边构成，一种是 n-links 即每两个邻域顶点（对应于图像中每两个邻域像素）的连接，另一种是 t-links 即每个普通顶点和这 2 个终端顶点之间的连接。在前后景分割中，s 一般表示前景目标，t 一般表示背景。

其中每一条边都有一个非负的权值 w，也可以称之为 cost。一个割就是边集合 E 的一个子集 C，该割的 cost 就是子集 C 所有边的权值之和。

Graph Cut 算法的目的就是求一个最小割，这个最小割把图的顶点划分为两个不相交的子集 S 和 T，其中其中 s ∈S，t∈ T 和 S∪T=V。这两个子集就对应于图像的前景像素集和背景像素集，那就相当于完成了图像分割。

图割可以看作为像素标记问题，这个过程可以通过最小化图割来最小化能量函数得到。假设整幅图的像素集为 $A = (A_1, \dots A_p)$，其中A_i是"obj"或者"bkg"，向量 A 定义了一次分割，设定每次分割的损失函数如下

$$E(A) = \lambda R(A) + B(A)$$

其中

$$R(A) = \sum_{p \in \mathcal{P}} R_p(A_p)$$

$$B(A) = \sum_{\{p,q\} \in \mathcal{N}} B_{\{p,q\}} \cdot \delta(A_p, A_q)$$

且

$$\delta(A_p, A_q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{otherwise.} \end{cases}$$

其中，R(A)为区域项，B(A)为边界项，λ是区域项和边界项之间的重要因子，

决定它们对能量的影响大小。

在区域项计算公式中，$R_p(A_p)$表示为像素 p 分配标签A_p的惩罚，$R_p(A_p)$能量项的权值可以通过比较像素 p 的灰度和给定的目标和前景的灰度直方图或者通过 RGB 三通道的混合高斯模型 GMM 来获得（本次作业采用的是单通道高斯模型获得的）。为了使相似度高的损失小，一般对概率取负对数，公式如下：

$$\begin{aligned} R_p(\text{“obj”}) &= -\ln \Pr(I_p|\mathcal{O}) \\ R_p(\text{“bkg”}) &= -\ln \Pr(I_p|\mathcal{B}). \end{aligned}$$

在边界项计算公式中，p 和 q 为邻域像素，边界项主要体现分割 A 的边界属性，$B_{\{p,q\}}$可以解析为像素 p 和 q 之间不连续的惩罚，如果 p 和 q 越相似，那么$B_{\{p,q\}}$越大，如果他们非常不同，那么$B_{\{p,q\}}$就接近于 0, $B_{\{p,q\}}$的计算公式如下：

$$B_{\{p,q\}} \propto exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right) \cdot \frac{1}{dist(p,q)}.$$

使用上述公式建立好图之后，便能通过最大流最小割的各种算法找到最小割了，而最小割就是对应着损失函数最小。割完之后所有和 s 处于同一子集的像素点便是前景。

2 运行环境

2.1 硬件及系统环境

处理器	Intel(R) Core™ i5-8265U@1.60GHz
RAM	16.00GB
系统类型	Windows10.0.18363 64 位
内核	4
逻辑处理器	8
L1 缓存	256 KB
L2 缓存	1.0MB
L3 缓存	6.0MB

2.2 软件即库版本

- python 3.7.0
- numpy 1.17.4
- opencv-python 4.1.2.30
- tqdm 4.40.2
- pymaxflow 1.2.12

3 模块及功能描述

3.1 交互模块

功能：该模块用于与用户进行交互，记录用户的各种事件，并把用户设置的前景背景信息传递给分割模块。

具体实现：为图片显示窗口绑定处理鼠标事件的函数，记录用户的EVENT_LBUTTONDOWN，EVENT_LBUTTONUP，EVENT_MOUSEMOVE 事件，把用户点击拖动时的点全部记录下来，并传给分割模块。同时不断循环，监听用户的键盘输入，当输入’t’时进行前景背景模式切换，当输入’g’时进行图像分割，当输入’s’时保存图片。

3.2 分割模块

功能：该模块为一个名为 GraphCut 的类，用来根据用户设置的前景背景 seeds 以及参数进行图像切割。

具体实现：

1. 根据用户传入的图片路径以及参数进行初始化，包括加载图片，设置参数，设置变量。
2. 生成 n-links 边权，即每个像素与邻域像素间的边。并把所有边存入字典中。
3. 根据设定的前景以及背景像素点得到前景背景的高斯分布。
4. 利用上一步得到的高斯分布生成 t-links 边权，即每个像素点和 2 个人为添加的终端顶点之间的边权。
5. 利用得到的 n-links 和 t-links 建立图，使用最大流最小割的思想进行分割。

并输出分割后的像素。

3.3 最大流最小割分割模块

该模块进行最大流最小割分割，由于时间精力有限，直接使用的 pymaxflow 库，而且该库是由 C++实现，因此效率要比自己使用 python 实现高很多。

4 遇到的问题

1. 开始实现的只把用户设定的前景分离出来。

   原因：因为用户在图片上进行交互划线的时候实际上会改变原图片的像素值，即在本次作业中，在图片上划一道红线，意味着把原图片的这块像素值全部变为（0， 0， 255），所以在分割的时候这部分的 boundary cost 会非常的高，导致程序只会把这一块分割出来。

   解决：在 GraphCut 初始化的时候对原图片进行一次拷贝，交互均在拷贝后的图片进行，不影响原图片。

2. 像素很相似的地方却被分割开来。

   原因：因为在计算两个像素点的差值的时候是直接使用 img[p] – img[q]实现的，但是 opencv 默认读取的数据类型是 np.uin8,即(0, 0, 0) – (1, 1, 1)会发现负溢，得到(255, 255, 255)而不是(-1, -1, -1)。

   解决：在相减的时候进行一次类型转化，转化为 np.int32。

3. 分割区域与预计不符。

   原因：从 opencv 交互得到的坐标与 numpy 中表示图像的坐标有所不同。

   解决：在添加从 opencv 得到的交互坐标时进行相关处理，即(x, y) -> (y – 1, x – 1)。

4. 分割的前景包含的很多零散的背景像素。

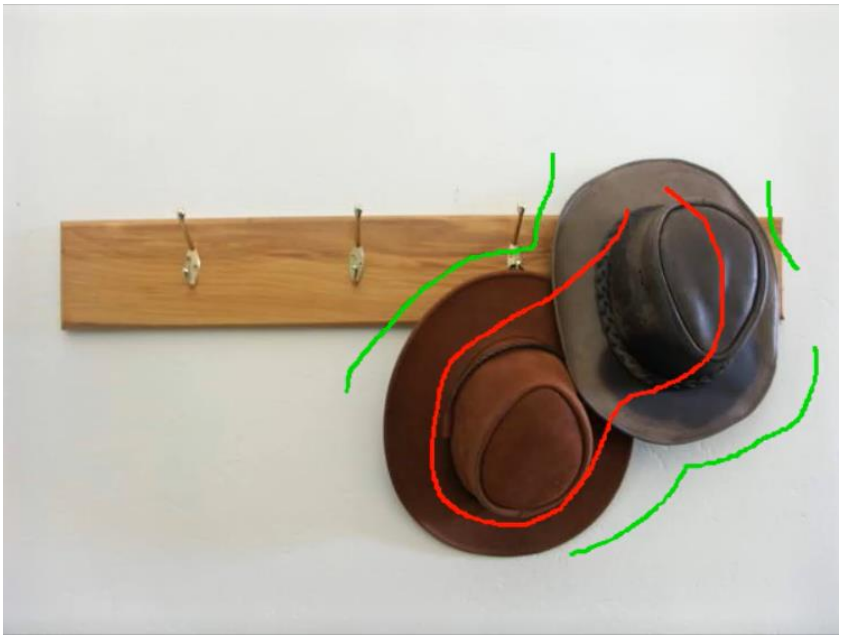
   原因：lambda 参数过大，即 regional cost 影响过大，这样部分背景中颜色比较接近前景的像素会得到较大的 t-links 边权，因此可能被错误的划分到前景。

   解决：使用多个 lambda 值。

5 实验结果

5.1 输入

5.2 用户操作



5.3 输出

6 算法优点：

1. 算法多项式时间内可解
2. 只需要用户给出较少的前景和背景信息，即可进行精细的分割
3. 使用高斯模型 GMM，所以与传统的 GraphCut 不同，本算法能彩色图像进行分割
4. 能量函数在连续的状态下实现，所以最终能得到较高精度的边缘

7 算法不足之处：

1. 一次只能分割出前景和背景，即分割类别(标签)只有两个。若分割类型多于两个，则只能近似处理。而多区域分割算法或是聚类分割算法，可以分割出多个不同类型的对象(>2)
2. 求取能量函数最值时，容易陷入局部极值。
3. 只能做一次用户操作，而 GrabCut 可以在得到输出结果后，再次将输出结果作为输入进行迭代，同时用户能对图片进行再次的标记