School of Data and Computer Science, Sun Yat-sen University

基于 CUDA 加速的改进 Mean Shift 图像分割算法

组长: 16337180 麦显忠, 16313018 李沐晗, 16337242 韦博耀, 16337259 谢江钊, 16337179 麦金杰

2019 年 6 月 6 日

目录

摘要

Here is where I would say what is in this document.

1 Introduction

1.1 Mean Shift 算法概述

Mean Shift 算法用于图像分割最早是由 (**comaniciu2002mean**) 提出的. 算法的主要步骤大概可以分为三步:

1. Mean Shift 滤波
2. 将像素点聚类为区域
3. 进行标签分配

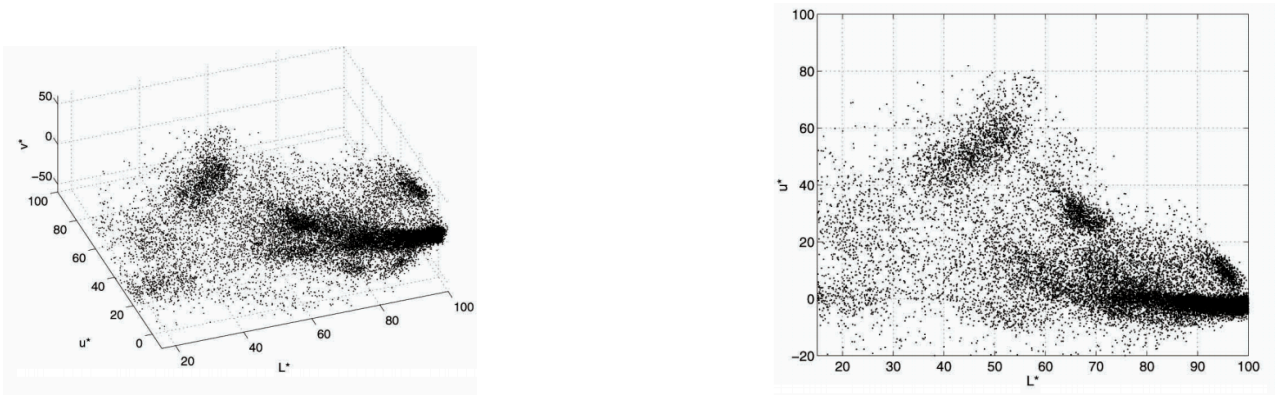
下面先以介绍这篇 PAMI 论文中提出的 Mean Shift 用于图像分割的实现.

1.1.1 Mean Shift 滤波

Mean Shift 滤波更直观的阐述是 mode 搜索过程. Mean Shift 算法将图像分割视为一个聚类问题. 要进行聚类, 本文中采用的方法是先找到聚类中心, 也就是所称的 mode, 再由此出发进行聚类的过程. 在原文中, 作者先将原图映射到特征空间, 再从这个映射得到的特征空间中搜索 mode 点.

图 1: 原图

特征空间的映射比较简单. 以原文中的颜色空间的例子来说. 如图2, 原始的输入图像图 a 先经过颜色转换后, 就映射到了 LUV 颜色特征空间图. 而具体到 mode 点搜索的过程, 为方便讨论, 如图2, 我们将三维的 LUV 特征空间简化为二维进行讨论, 如果我们将空间内点的密度作为搜索聚类中心 mode 的标准, 那么图 4 中所示的红色点无疑就是我们需要搜索的聚类中心.



(a) LUV (b) LU

图 2: LUV and LU

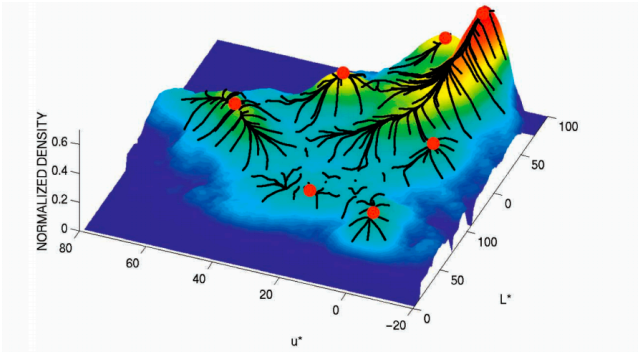


图 3: Mode

在原来 Mean shift 算法的实现中, 作者采用了基于核密度估计的策略来寻找 mode 点. 首先, 基于经典而传统的核函数公式:

$$\hat{f}(x) = \frac{1}{nh^d} K(\frac{x - x_i}{h}) \tag{1}$$

其中 x 代表输入数据, n 是数据点数, h 是带宽, d 是数据的维度. 作者提出了新的核密度估计方法. 首先有:

$$\hat{f}(x) = \frac{c_{k,d}}{nh^d} k(||\frac{x - x_i}{h}||^2) \tag{2}$$

$$\hat{\bigtriangledown} f_{h,K}(x) = \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (x - x_i) k'(||\frac{x - x_i}{h}||) \tag{3}$$

$$g(x) = -k'(x) \tag{4}$$

$$G(x) = c_{g,d} g(||x||^2) \tag{5}$$

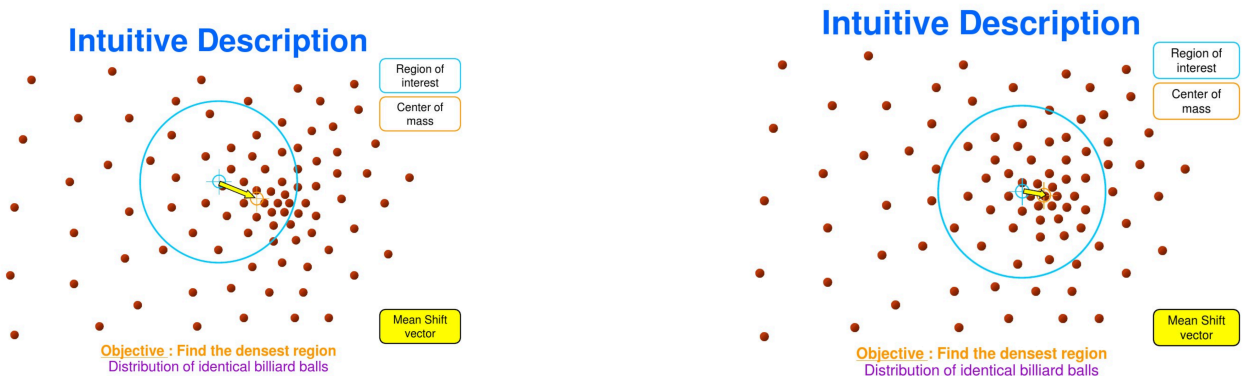


图 4: 搜索

这里我们用一步步导出的 G(x) 作为新的核函数. 而根据上面的符号我们可以将重写 2的导数为

$$\begin{aligned}
&\hat{\bigtriangledown} f_{h,K}(x) \\
&= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (x_i - x) g(||\frac{x - x_i}{h}||^2) \\
&= \frac{2c_{k,d}}{nh^{d+2}} [\sum_{i=1}^{n} g(||\frac{x - x_i}{h}||^2)][\frac{\sum_{i=1}^{n} x_i g(||\frac{x-x_i}{h})||^2}{g(||\frac{x-x_i}{h})||^2} - x]
\end{aligned} \tag{6}$$

因此, 算法中的第二项就是我们算法中所称的 Mean shift $m_{h,G}(x)$, 由上面的推导可以得到:

$$\begin{aligned}
m_{h,G}(x) &= \frac{\sum_{i=1}^{n} x_i g(||\frac{x-x_i}{h})||^2}{g(||\frac{x-x_i}{h})||^2} - x \\
&= \frac{1}{2} h^2 c \frac{\hat{\bigtriangledown} f_{h,K}(x)}{\hat{f}_{h,G}(x)}
\end{aligned} \tag{7}$$

从上面的式子 7我们就可以看到, Mean shift 与数据密度的梯度方向相同, 这就证明了从任意点开始, 只要朝 mean shift 的方向移动, 就会慢慢的走向密度估计最大的地方, 而这也正是我们需要找的聚类中心, 也就是之前提到的 mode 点. 一个更加形式化的说明可以参照下面两幅图4, 可以看到, 不断朝着 mean shift 的方向进行移动, 最终到达了密度最大处.

1.1.2 聚类和分割

经过上面的步骤后, 对于输入数据中的每个点 (像素点), 我们都能找到它的 mode 点, 即聚类中心. 接下来我们就可以开始围绕着聚类中心进行聚类. 原文中的具体方法是根据空间距离和颜色距离这两个度量指标, 采取类似于种子生成的策略一步步的将 mode 点附近的像素加入到各个 mode 当中, 最后对这些以 mode 为中心的, 零碎的小区域进行平滑和合并, 就得到了最终的分割图像. 比较直观的可视过程可以参见下图5.

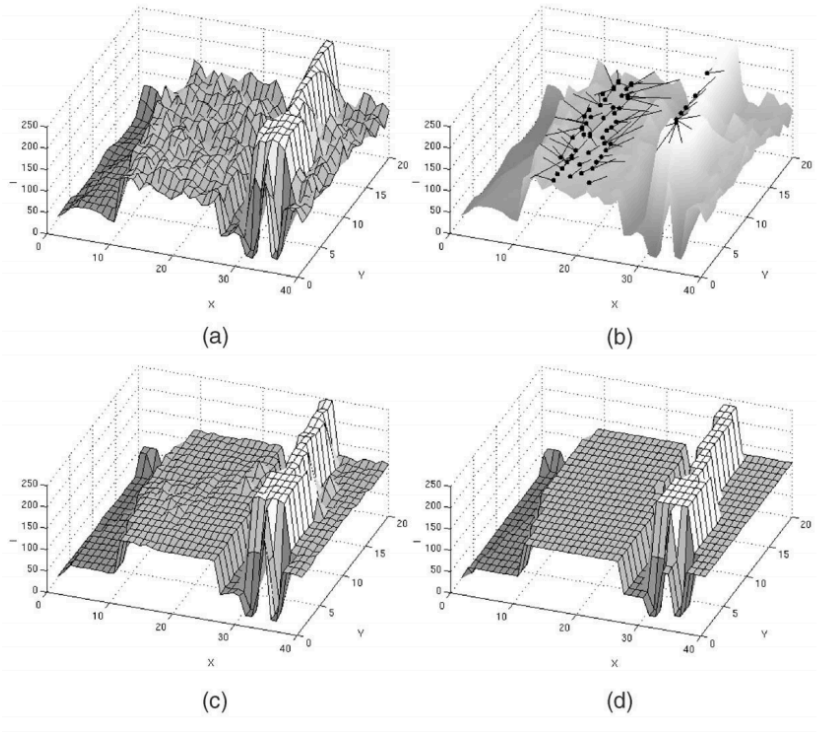


图 5: 流程

其中 a 图即是原图像经映射后得到的特征空间, b 图表示了从图像的各个点出发, 同时寻找对应的 mode 点的过程. c 图表示了根据距离相似性和色彩空间相似性得到的聚类结果, d 图即是经过了平滑, 极小区域擦除后的最终分割图像. 一个完整的例子可以参见下图6.

图 6: 例子

1.2 课程实验

因此, 我们组决定参照此篇论文提出的基于 Mean shift 的图像分割算法, 进行一定的复现和改进. 我们计划实现的算法大体框架如下:

1.2.1 Mean shift 滤波

和原算法一致, 这一步同样是为了找到聚类中心. 在本次实验中我们采取了稍有不同的实现. 首先, 我们只采用 LUV 颜色空间作为特征, 据此先给出算法的伪代码1如下.

Algorithm 1: Meanshift code

Input: 图片 I

Output: Meanshift 分割后的图片 I'

for I 中的每个像素点 *(x,y)* **do**
  while 达到迭代次数上限 **do**
    像素点 (x,y) 当前的 LUV 值为 $LUV_{x,y}$
    设置 color_sum = [0,0,0], index_sum = [0, 0], summed_pixels = 0
    for 对以带宽 dh 为半径的圆或正方形邻域内的像素点 *(x',y')* **do**
      计算 $\Delta LUV = |LUV_{x,y} - LUV_(x', y')|$
      if $\Delta LUV > color\_max\_diff\_range$ **then**
        $color\_sum+ = LUV_{x',y'}$
        $index\_sum+ = (x', y')$
        $summed\_pixels++$
      $cs\_avg = \frac{color\_sum}{summed\_pixels}$
      $is\_avg = \frac{index\_sum}{summed\_pixels}$
      $shift = |cs\_avg|^2 + |is\_avg|^2$
      更新 $LUV_{x,y} = cs\_avg$
      if $shift < shift\_range$ **then**
        break
return 输出图像 *I'*

可以看到, 在我们的实现伪代码**??**中, 每个点的像素值不断地朝区域平均值的方向移动, 也即 mean shift. 待到算法终止时, 每个像素点的像素值都 mean shift 到了某一簇像素.

1.2.2 聚类 (Flooding)

完成初步的滤波后, 我们还需要对每一个像素都分配一个特定的标签, 以此完成图像分割的任务要求. 而由于 Mean shift 方法本质上是无监督的算法, 因此我们只能给每个像素分配没有实际意义的标签. 这一步骤的具体过程如伪代码2所示.

Algorithm 2: Flooding 洪泛搜索

Input: 经过 Mean shift 后的图片 I

Output: 图片 I 上每个像素点对应的 label

```
for I 中的每个像素点 (x,y) do
    分配一个唯一的 label_{x,y} = y * width + x
    while 达到迭代次数上限 do
        for 对以带宽 dh 为半径的圆或正方形邻域内的像素点 (x',y') do
            if ΔLUV > color_max_diff_range then
                更新 label_{x',y'} = label_{x,y}
return 图像 I 对应的 label
```

在以上步骤中, 不难发现, 如果对上述过程进行并行化, 可能会有多个像素点抢一个像素点的情况 (赋给自己的 label), 这会由线程间的竞争决定最后谁赋值成功. 也就是说, 对于相同的输入, 标签分配算法并不能保证得到完全相同的输出. 但需要说明的是, 这里线程的竞争导致的标签不一致, 可以在后续步骤的区块合并中得到解决.

但在实际实验中, 我们发现如果对每个像素都进行并行化, 会造成比较密集的冲突和重复赋值, 降低并行化的效率. 为了解决这一情况, 我们在实际中进行了优化. 在某一部分像素点钟, 只让一个像素点线程进行洪泛搜索. 这些搜索部分的划分可以由行和列确定, 例如每个 $n * n$ 的子图像只让一个像素点进行标签洪泛. 在循环中采取类似卷积移动滑动窗口的方式, 对下一个部分进行标签分配.

1.2.3 合并小区块

最后一步, 根据上一步得到的聚类结果, 我们再进行最后一步后处理得到最后的分割. 对于过小的区块和相似的区块, 我们将它和邻近的进行合并, 并对分割图像进行一定的平滑和滤波处理. 即得到了分割结果.

但这一部分中, 因为需要记录所有区块目前的标签来动态的合并, 因此这一部分不太适合采用 CUDA 进行并行化. 而且考虑到在这一步中, 区块已经比较少, 因此在这一步中我们采用了 pthread 进行并行. 考虑上面计划实现的算法, 我们计划综合采用指令级的 CPU AVX 并行, 线程级的多线程并行和 CUDA 并行对上面的算法进行优化, 主要进行并行的步骤有:

1. 在聚类中心的搜索过程中, 由于对于每个数据点都需要视为起点, 进行一次 mean shift 的聚类中心搜索. 因此我们可以对这一部分进行并行处理, 每个线程负责对一个单独的数据点起始的 mode 搜索.

2. 在种子生成过程中, 对于多个类别的种子, 我们都需要执行生长过程. 因此我们这里也可以采用并行技术, 把每个种子点进行生成的任务分配给不同的线程进行.

2 Implementation

2.1 Mean Shift 滤波的并行化

需要实验结果, 暂略.

2.2 聚类 flooding 的并行化

需要实验结果, 暂略.

2.3 Union Find 的并行化

需要实验结果, 暂略.

2.4 并行加速效果评估

串行实现速度, 对比等, 需要实验结果, 暂略.

3 Conclusion

本次实验中我们实现了并行化的 Mean Shift 算法, 但是还是有一些不足之处, 如不能自适应, 需要一些超参数, 不能很好适应不同大小和纹理的图片, 敏感度高分割会分出很多噪点, 敏感度低又会损失细节等.