基于SLIC的超像素分割算法

目录

[基于SLIC的超像素分割算法 1](#_Toc23325168)

[摘要 3](#_Toc23325169)

[一． SLIC算法分割超像素原理 4](#_Toc23325170)

[二． SLIC的Python实现 6](#_Toc23325171)

[2.1 python代码及其解释 6](#_Toc23325172)

[2.2 程序的运行结果 12](#_Toc23325173)

[三． 对SLIC算法的认识 14](#_Toc23325174)

[3.1 SLIC算法与k-means聚类 14](#_Toc23325175)

[3.2 实际编码过程中得出的一个体会 14](#_Toc23325176)

[3.3 对参数m的理解 16](#_Toc23325177)

[3.4 对如何加速程序的思考 16](#_Toc23325178)

[四． 参考文献 17](#_Toc23325179)

# 摘要

超像素概念是2003年Xiaofeng Ren提出和发展起来的图像分割技术，是指具有相似纹理、颜色、亮度等特征的相邻像素构成的有一定视觉意义的不规则像素块。它利用像素之间特征的相似性将像素分组,用少量的超像素代替大量的像素来表达图片特征,很大程度上降低了图像后处理的复杂度，所以通常作为分割算法的预处理步骤。已经广泛用于图像分割、姿势估计、目标跟踪、目标识别等计算机视觉应用。

目前常见的超像素分割算法大致可以分类为基于图或梯度上升的方法。基于图的算法有GS04, NC05, SL08, GCa10, GCb10b等，基于梯度上升的算法有W91, MS02, TP09, QS09, SLIC等。【1】

本文要介绍的是SLIC(simple linear iterative clustering)，即简单线性迭代聚类算法。它在分割效果和分割速度两方面表现都比较优良，是一种比较实用的超像素分割算法。本文首先介绍SLIC算法的原理，其构造一个关于LAB空间色彩和空间距离的像素间距离的度量函数，选择一些初始点对图像像素进行局部聚类，之后根据聚类结果修改初始点位置，反复迭代，直到初始点的变化小于预期的阀值。

接下来本文通过python程序实现SILC,将展示每一部分python代码，并详细解释每一部分的功能。之后展示代码的实际运行效果，在安装二核的intel corei7-6500u处理器的笔记本电脑上处理311\*308像素的照片，迭代一次仅需约三秒，在分割速度方面令人满意。

在本文的最后一部分将讲述作者本人对SILC算法的认识，其与K-MEANS聚类算法的关系。

**关键词： 超像素， SLIC，K-means聚类，python**

# SLIC算法分割超像素原理

若图片原由N个像素，要将其分割成K个超像素，那么平均每个超像素包含像素的个数是N/K。若超像素等大且在图片中均匀分布，呈正方形，那么每个超像素正方形的边长是



首先，将K个聚类中心均匀分布到图片的像素点上，每个聚类中心与其相邻的四个聚类中心的距离均为S。

接下来，将中心移动到与3×3邻域中的最低梯度位置相对应的种子位置。这样做是为了避免将超像素定位在边缘上，并且减少用噪声像素接种超像素的机会。

由于超像素的预期空间范围是近似尺寸S×S的区域，因此在超像素中心周围的区域2S×2S中进行类似像素的搜索。每个像素可能会在4个聚类中心的搜索范围内，这里像素将与距离其最近的聚类中心相关联。

根据聚类算法的思想，我们需要构造一个距离函数D表示像素点到聚类中心的距离，这个距离函数D同时与像素与聚类中心的颜色差距DC以及空间距离DS有关。但是由于DC和Ds的单位不一样，直接采用的公式可能会导致空间距离和颜色差距中的某一个变量占主导地位，而另一个对聚类的影响微乎其微。因此需要对DC和Ds进行归一化处理。由于超像素在中心周围的区域2S×2S中进行类似像素的搜索，因此Ds的最大值为，去除常量后就是S。但是颜色差距的最大值在不同的像素簇和图片中并不是一个固定的值，因此这里采取一个经验值m, m的取值范围为。【1】

因此，距离度量函数为



由于在同一张图片中所有像素的m,S取值都相同，而我们只需要比较像素与不同聚类中心的D的大小关系，不需要绝对大小。因此将公式调整为



其中



表示当前像素的坐标，表示当前聚类中心的坐标。



表示当前像素在CIELAB空间中的值，表示当前聚类中心在CIELAB色彩空间中的值。

按照上述描述，可以将所有的像素与最近的聚类中心相关联。然后聚类中心调整为属于该聚类中心的所有像素的平均值。然后在新的聚类中心周围的区域2S×2S中进行类似像素的搜索。如此进行迭代。直到新的聚类中心的位置与先前的聚类中心的差值小于预期的阀值。

在本文并没有设置阀值来控制迭代的结束，而是通过人为的设置迭代次数来观察效果，最终确定一个合理的迭代次数。大多数时候迭代次数可以取10.【1】

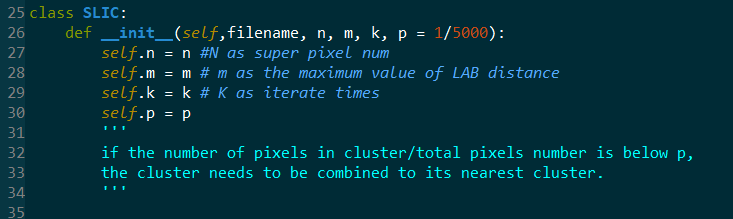
经过上述迭代优化可能出现以下瑕疵：出现多连通情况、超像素尺寸过小，单个超像素被切割成多个不连续超像素等，这些情况可以通过增强连通性解决。可以将过小的超像素合并到距离其最近的超像素，以消除孤立的结点。

# SLIC的Python实现

## 2.1 python代码及其解释

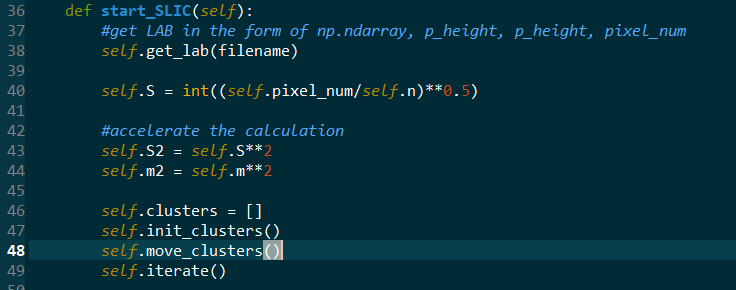
在本文的python代码中，SLIC的实现过程主要是自定义了一个名为SLIC的类，通过各种内置方法完成SLIC算法中初始化聚类中心、移动聚类中心到3\*3区域梯度最小的点，给像素分配聚类中心，生成超像素，迭代等过程。

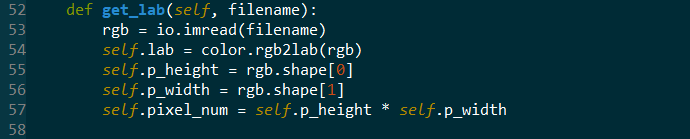
1. 定义SLIC类，并传入SLIC算法必须的参数。n代表要生成的超像素的个数，m代表上文提到的要归一化LAB距离所需要用到的LAB距离的最大值，k代表迭代次数。

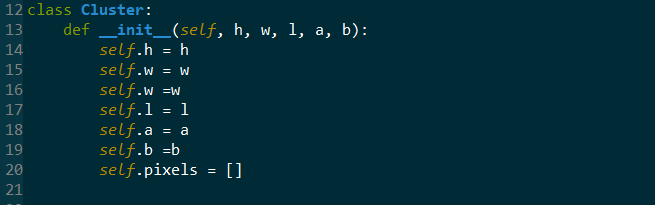


1. 调用方法start\_SLIC，start\_SLIC方法是一个总控制，包含实现SLIC算法中一系列过程的各个方法。对start\_SLIC中调用的各个方法的解释见后文。

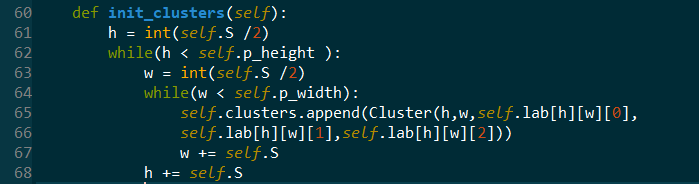
第40行代码计算了初始化聚类中心时两个相邻聚类中心的距离self.S,。第43,44行代码把self.S, self.m的平方计算出来并储存为类的属性，为的是避免在之后计算每个像素与聚类中心的距离时都需要重复计算这两个量，以缩短程序运行时间。46行的Self.clusters是一个存储所有的聚类中心的列表。



1. Get\_lab方法的作用是读入需要处理的image文件所在路径，通过调用skimage包中的color.rgb2lab方法，把图片的RGB数据转换为LAB数据存储在self.lab中。并获取了图片像素的高度和宽度等信息。
2. 为了方便存储聚类中心的信息，又定一个名为Cluster的类，其中的属性h,w,l,a,b分别代表一个聚类中心的像素坐标（h,w）和LAB空间中的值（l,a,b）。self.pixels代表属于此聚类中心的像素点的列表。



1. Init\_clusters的作用是初始化聚类中心。将聚类中心均匀分布在图片上，并将聚类中心cluster (type为（4）中定义的Cluster类)添加到self.clusters列表中。

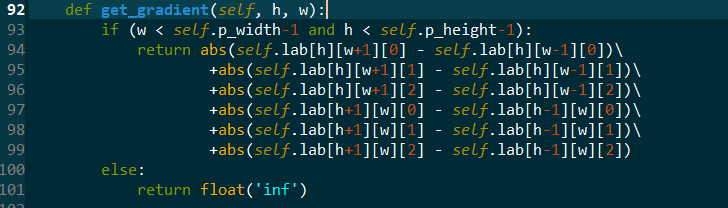


1. Move\_clusters 在每个聚类中心3\*3领域内选择出梯度最小的点作为新的聚类中心。这样做的目的是为了避免聚类中心落在梯度较大的轮廓边界，以影响后续聚类效果。代码78和80行的两个if语句是为了防止3\*3领域超出图片像素范围。76和81行调用的self.get\_gradient（h,w）方法会返回坐标为（h,w）点的梯度值。

图片包含 文字

描述已自动生成

1. self.get\_gradient（h,w）方法会返回坐标为（h,w）点的梯度值。当像素点落在图片边沿（w=p\_width-1或h=p\_height-1时），返回的梯度值为无穷大inf。

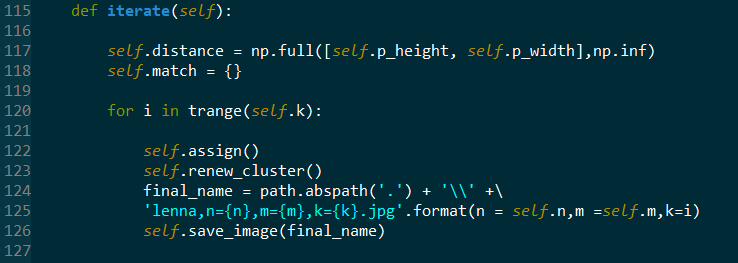


1. iterate方法对给像素分配聚类中心，更新聚类中心的值，存储照片这三个过程进行迭代。

其中代码117行初始化距离矩阵distance, distance[i][j]代表坐标为[I,j]的像素点到与所有2S\*2S包含该像素点的聚类中心的距离的最小值。初始化时distance中所有的值都为无穷大inf。

Distance中元素的值的计算过程如下：当坐标为[I,j]的像素点在第一个聚类中心的2S\*2S领域时，distance[i][j]的值由inf更新为距离D1。此后，当第二个2S\*2S领域包含该像素的聚类中心出现时，计算距离D2,如果D2<D1,则把distance[i][j]的值更新为距离D2。当整个遍历过程介绍，distance[i][j]存储着坐标为[I,j]的像素点到与所有2S\*2S临域包含该像素点的聚类中心的距离的最小值。

代码118行建立了一个字典match，字典的key为像素，value为聚类中心。字典存储所有已经配对的像素与聚类中心。



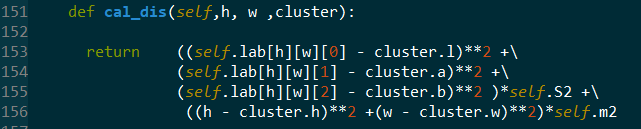
1. assign方法对所有聚类中心的2S\*2S领域内所有的像素点进行遍历。代码的134和136行是防止2S\*2S领域内的点超出像素范围。如果像素点[h,w]与聚类中心cluster[k]的距离小于distance[h][w], 则需要更新distance[h][w]的值。如果像素（h,w）在字典match的Key list中，说明（h,w）之前与其他的聚类中心配对过，需要除去之前与该像素配对的cluster的pixels里的该像素（代码143,144行）。之后将新的键值对添加到match字典中。最后把像素（h,w）加入到cluster的pixels列表中。

此处有字典match和列表cluster.pixels都记载了配对信息，这样做的好处是：当更新cluster时，只需要遍历cluster.pixels中的pixel，计算平均值，即可得到新的cluster。而match的作用是当需要更新与某个像素匹配的聚类中心时，可以通过match字典获得上一次与该像素匹配的聚类中心，由此去除该聚类中心的piexels列表中的该像素。

图片包含 文字, 笔记本电脑

描述已自动生成

1. dal\_dis方法，计算像素（h,w）与cluster的距离。



1. renew\_cluster方法，更新聚类中心。

图片包含 文字

描述已自动生成

1. 代码187行把self.lab的值拷贝到了lab\_copy,因为self.lab在下一次迭代时还需要使用，不能更改。所以更改lab\_copy里像素的lab值并输出为图片。

将每个聚类中心的cluster.pixels列表中的全部像素的lab值改为cluster的lab值，并把cluster的lab值设为（0,0,0），因此cluster在输出图片中将以黑点的形式显现。

最后调用color.lab2rgb和imsave方法输出图片到指定路径。

1. 主函数通过引用了相对路径（被执行的py文件所在目录），只需要把需要处理的图片放入py文件所在目录，即可以运行程序。即使文件改变了所在位置，也不需要对路径进行更改。

图片包含 室内

描述已自动生成

## 2.2 程序的运行结果

分别采用像素为311\*308，，设置超像素个数n=1000,LAB距离最大值m=30，迭代次数为10.

1. 像素为311\*308的图片迭代一次用时约3.7秒,迭代十次总共用时37秒。

原图 k=1 k=10

1. 像素为690\*458的图片迭代一次用时约13秒，迭代十次总共用时134秒。



原图

K=1 k=10

1. 像素为1600\*1143的图片迭代一次用时约87秒，迭代十次总共用时874秒。



原图



K=1 k=10

# 对SLIC算法的认识

## SLIC算法与k-means聚类

SLIC算法采用了k-means算法中初始化聚类中心，通过距离函数给每个点分配聚类中心，迭代，通过平均值产生新的聚类中心，迭代终值条件等思想和方法。两者最大的不同是k-means算法要求每一个像素必须与所有聚类中心比较，因此复杂度是O(K\*n\*N),其中k迭代次数，n是超像素个数，N是所有像素的个数。而SLIC算法将聚类中心的搜索范围限制为与超像素大小成正比的区域2S\*2S，因此每个像素只需要计算与四个聚类中心的距离，显著的降低了问题的复杂度，复杂度变为O(k\*N),复杂度与要生成的超像素的个数无关。

笔者认为将聚类中心的搜索范围限制为与超像素大小成正比的区域的方法不仅降低了算法的时间复杂度，而且不会对超像素分割的结果产生不好的影响，反而可能使分割效果更好。因为超像素分割本就要求属于同一个超像素的像素在空间上要有一定的连续性，如果距离过远的像素被分配到同一个超像素，则最后可能会产生一些“孤立的点”。限制搜索范围的方法相当于做了一种预处理，直接排除了距离大于两倍的超像素边长的像素与聚类中心相匹配。

直观上，这个搜索范围取2S\*2S比较合适，既大于超像素的理想面积S\*S，又不至于让距离过大的像素被匹配到聚类中心。

## 实际编码过程中得出的一个体会

在最开始的编码过程中，笔者认为由于在新一次的迭代前聚类中心的值发生了变化，因此需要把原来的距离矩阵distance重新初始化，距离设为无穷大inf，把每个聚类中心的cluster,pixels列表初始化为空列表,把字典match设为空字典。

但是最后出来的结果很不理想：除了第一次迭代（聚类中心均匀分布）产生的超像素很正常，之后产生的超像素都呈现方块状，如下图。



K=1 k=2



K=5

笔者将在每新一次的迭代前把原来的距离矩阵distance重新初始化，把每个聚类中心的cluster.pixels列表初始化，把match设置为空字典为空列表的代码去掉后，结果变得比较理想，见2.2.1图。

笔者思考原因如下：若在每次迭代前初始化距离矩阵，cluster.pixels, match，则丢失了上一次迭代过程中产生的聚类中心与像素的匹配信息。而在新一次的迭代中，cluster的取值是pixels的平均值，因此cluster在LAB距离上离像素更近，但是仍然是在2S\*2S的范围内扫描，因此空间距离不会改变。此时空间距离成为了主要因素，因此之后超像素呈现方块状。

而如果在每次迭代前不初始化（见2.1.8部分的代码），在新的迭代过程中计算出的距离需要与上一次迭代过程中时的距离进行比较，上一次迭代过程中时的距离记录了LAB距离的信息，因此呈现出正常的结果。

## 对参数m的理解

参数m在对LAB距离做归一化时用到作为LAB距离的最大值。观察变形后的距离函数（推导过程见一）：



在S取值固定的情况下，m实际上是作为Dc与Ds之间的一个比例系数，m越大，表示空间上的临近性更重要，所得到的超像素越紧凑；m越小，表示颜色上的相似性更重要，由于图像在边界的颜色差别较大，因此所得到的超像素有更好的边界粘附性。

## 对如何加速程序的思考

本算法最耗时的部分为assign方法（如下）对所有聚类中心的2S\*2S领域内所有的像素点进行遍历。

图片包含 文字, 笔记本电脑

描述已自动生成

因此要加速算法，就主要是对三重for循环里面的部分（138到147行）进行改进。

本程序首先对距离计算函数cal\_dis(138行进行了改进)，把self.S, self.m的平方计算出来并储存为类的属性，为的是避免在之后计算每个像素与聚类中心的距离时都需要重复计算这两个量，并且去掉了距离函数的开方部分，因为只需要比较距离的相对大小。

再就是有字典match和列表cluster.pixels都记载了配对信息，这样做的好处是：当更新cluster时，只需要遍历cluster.pixels中的pixel，计算平均值，即可得到新的cluster。而match的作用是当需要更新与某个像素匹配的聚类中心时，可以通过match字典获得上一次与该像素匹配的聚类中心，由此去除该聚类中心的piexels列表中的该像素。

因此当需要去除cluster.pixels里旧的pixel值时，不需要在多个cluster.pixels列表里查找，而是在字典match中查找到cluster值，再在对应的一个cluster.pixels中查找即可。因为字典的查找远远快于列表，而且此处只需要查找一次字典和一次列表，相比于不使用字典时可能需要最多n次查找列表，也加快了程序的运行。

# 参考文献

1. Radhakrishna Achanta, et al. "SLIC superpixels compared to state-of-the-artsuperpixel methods." Pattern Analysis and Machine Intelligence, IEEETransactions on 34.11 (2012): 2274-2282.