随堂测验4

SLIC 超像素聚类方法的实验

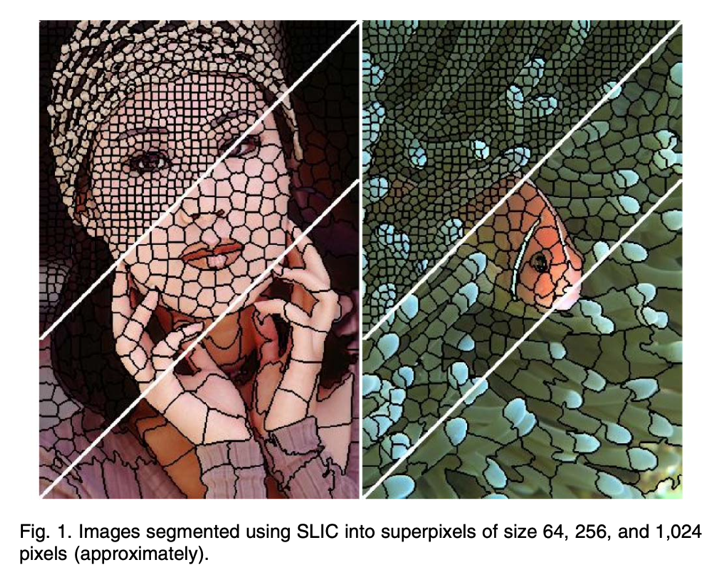
李春良，计算机学院，1120193226，07111908

本文旨在复现SLIC方法的经典论文：

Achanta R, Shaji A, Smith K, Lucchi A, Fua P, Süsstrunk S. SLIC superpixels compared to state-of-the-art superpixel methods. IEEE Trans Pattern Anal Mach Intell. 2012 Nov;34(11):2274-82. doi: 10.1109/TPAMI.2012.120. PMID: 22641706.

# 实验原理

超像素聚类将像素组合成感知（perceptually）有意义的原子区域，这些区域可被用于替换图像网格的刚性结构（Fig1）。



实现超像素聚类的方式主要有基于图论（Graph-Based）和基于梯度上升（Gradient-Ascent-Based）两种方式。基于图论的超像素生成方法将每个像素视为图的节点，两个节点之间的边权重与相邻像素之间的相似性成正比。通过最小化在图上定义的目标函数，可以获得超像素聚类。而基于梯度上升的方法则从粗略的像素初始聚类开始，迭代细化超像素聚类，直到满足收敛条件，从而获得超像素聚类。

超像素聚类的SLIC方法（Simple linear iterative clustering，简单线性迭代聚类）将-均值聚类的类似思想应用于超像素块的生成。

均值聚类的主要思想是，选择个聚类中心，将所有数据分配到最近的距离中心，将所有的数据分配到聚类中心，使得每个点与所属聚类中心的距离平方和最小。在完成数据到聚类中心的分配后，重新计算数据的聚类中心，并以新的聚类中心对所有数据重新进行聚类中心的分配，直到聚类中心的变化小于指定阈值。

SLIC方法的主要特点有：

1. **距离计算被大幅减少**，SLIC将聚类中心的搜索空间限制到了聚类中心的周围而不是整幅图像，充分考虑到了图像超像素内部像素可能具备的空间相似性。这使得算法复杂度仅仅和像素数量线性相关，与超像素数量线性无关。
2. **SLIC衡量像素间距离考虑到了色彩和空间相似性**，将二者加权来控制超像素的大小和紧凑程度。

SLIC算法的撰写者指出，衡量超像素聚类算法好坏的标准，主要包括：1) 超像素是否良好地依附于图像边界；2) 在应用于预处理步骤以减少计算复杂度时，超像素应当可以被快速计算、高效利用内存、并且易于使用；3) 用于图像分割的目的时，超像素应当又快又好地得到结果。

# 实验步骤

SLIC算法在CIELAB色彩空间下对图像进行处理。本文采用OpenCV提供的Mat矩阵和色彩空间转换函数复现SLIC，同时便于观察实验结果。在Lab色彩空间中，像素的颜色由三个分量描述，描述一个像素即为五个分量。

这里先给出SLIC的算法描述，如Algorithm1所示，读者可以跳过这一部分，在实验步骤的最后再来阅读。

**Algorithm 1.** SLIC superpixel segmentation

*/\* Initialization \*/*

Initialize cluster centers by sampling pixels at regular grid steps S.

Move cluster centers to the lowest gradient position in a neighborhood.

Set label for each pixel .

Set distance for each pixel .

**repeat**

*/\*Assignment\*/*

**for** each cluster center **do**

**for** each pixel in a region around **do**

Compute the distance between and .

**if** **then**

set

set

**end if**

**end for**

**end for**

/\*Update\*/

Compute new cluster centers.

Compute residual error E.

**until**

SLIC的主要实现步骤分为三步。

## Initialization Step 初始化

SLIC方法希望获得个大小近似相等的超像素块，因此需要先行选择个初始聚类中心：

网格间距（Grid Interval）, 这是因为，其中N是像素总数。

种子点（Seed location）是网格点邻域内的最低梯度点，这是为了防止1) 在边缘创建中心点；2) 在噪声像素上创建中心点。

实现初始化步骤时，假定给定了原始图像矩阵src和网格间距S，计算相应网格的行列数gridRows和gridCols以便确定聚类中心下标，使用vector<LABXY>存储聚类中心，LABXY是定义的五元组结构体。

计算梯度需要对图像进行求导，这是一个重要的卷积运算。本文利用OpenCV提供的Sobel算子作用在图像上，得到图像的梯度。然后在网格点作为聚类中心点的基础上，在邻域内寻找梯度最低点，将相应聚类中心点移动到该点作为种子点。

上述操作完成在SuperPixelSLIC类(区别于OpenCV的SuperpixelSLIC，后者的p是小写) 的构造函数中，如Fig2所示。

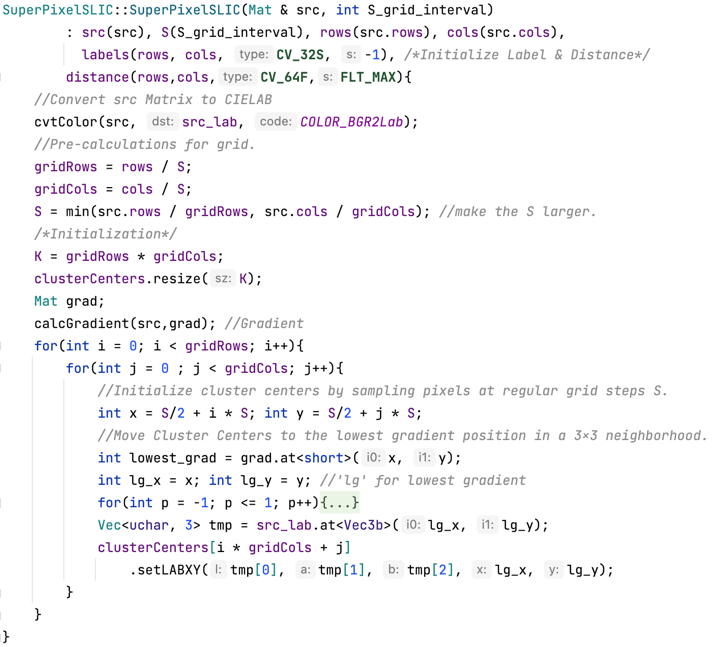


Fig.2 SuperPixelSLIC::SuperPixelSLIC(Mat &, int);

## Assignment Step 赋值与迭代

迭代计算的思路类似于K均值聚类，遍历聚类中心，以每个聚类中心为原点搜索2S邻域内的像素点，计算距离，并修改像素点的label归属。然后更新聚类中心，重复搜索。更新聚类中心时，对于属于某个聚类中心的像素点，分别计算五个分量的均值。

SLIC的作者指出，“10 iterations suffices for most images”, 所以本文选择直接控制迭代次数，而不是计算聚类中心的误差E来停止迭代。

实现距离计算，不能直接对LABXY五元组计算欧氏距离，而应当分别对色彩距离和空间距离加权考虑。

简化后的距离计算公式为

（compactness）默认值取10，可以在大多数情形下获得比较理想的效果。

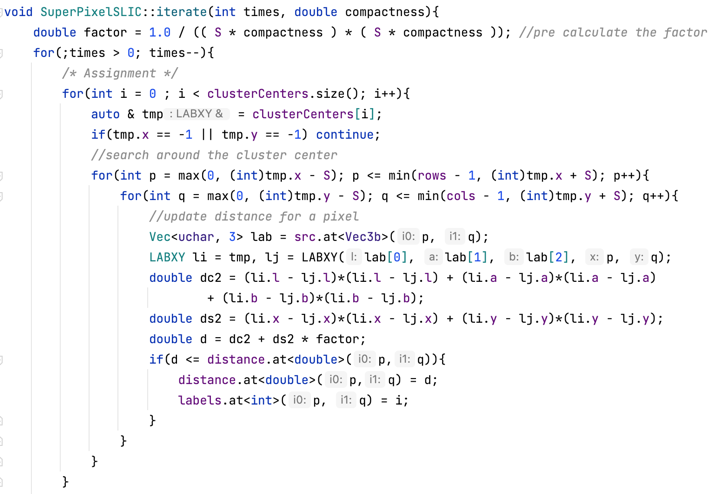


Fig3. Iterate: Assignment

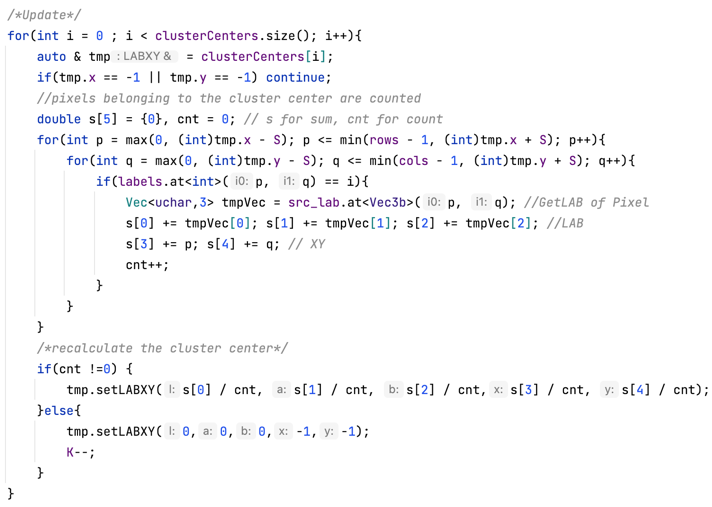
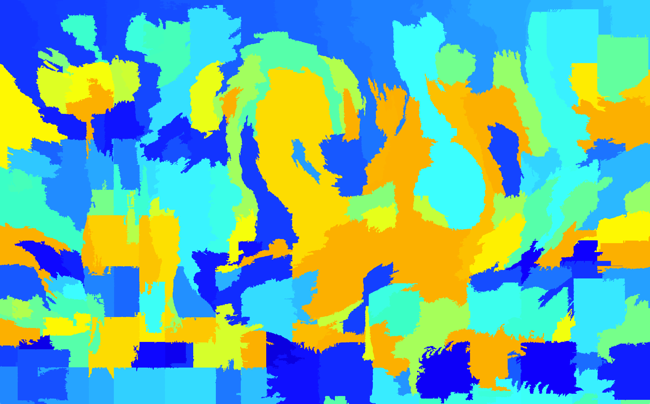
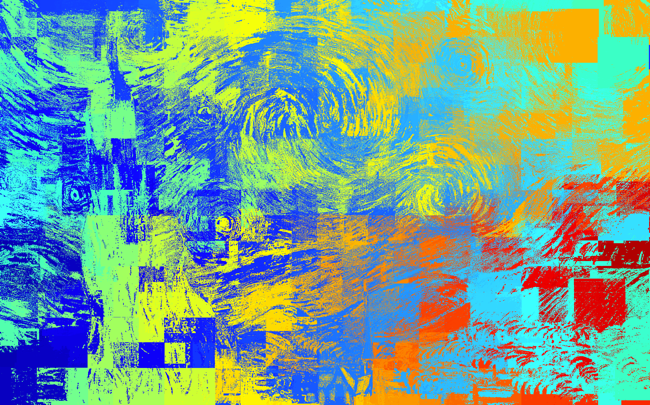


Fig4.Iterate:Update

## Postprocessing Step 后处理

SLIC算法和其他超像素算法一样，并不严格保证连通性。在后处理阶段，使用连通区域分析算法(Connected Components Algorithm)，把属于某超像素块但不连通的孤立像素修正到最近的其他聚类中。

常用的Connected Components Algorithm方法包括递归跟踪、并行生长、逐行跟踪，本文使用经典的逐行跟踪算法来实现增强label的连接性。主要方法是1) 在开始时为每个新的区域寻找一个邻接区域的标签 2) 在某个区域过小的情况下，将之前的邻接区域的标签赋给该区域，即将两个区域合并，并且此时不增长标签值。如Fig5所示。



|  |  |  |
| --- | --- | --- |
| Fig11. The Starry Night – Original | Fig12. Our Superpixel SLIC | Fig13. Our With Postprocessing |

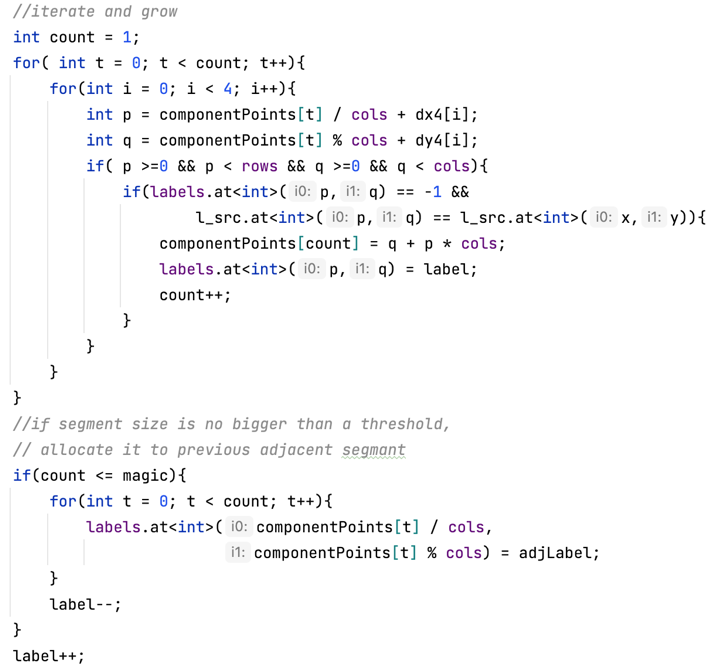


Fig5. Part of SuperPixelSLIC::enforceConnectivity

# 实验结果

首先编写测试代码。如Fig6，Fig7所示。这让本文的实验代码用起来非常接近OpenCV实现的SLIC。



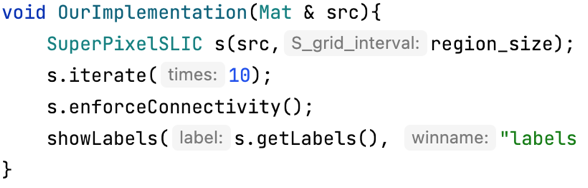
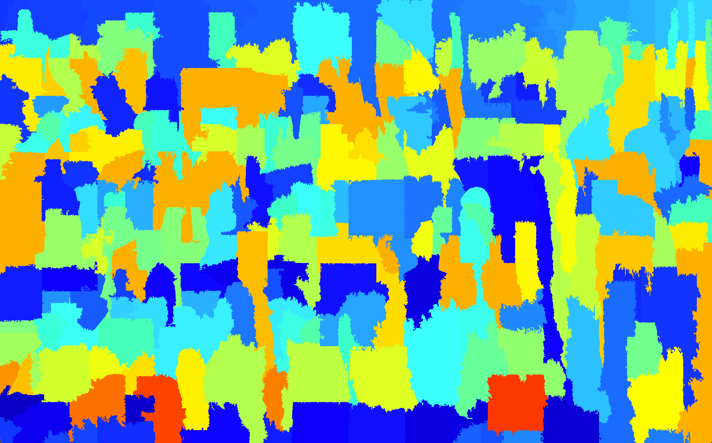
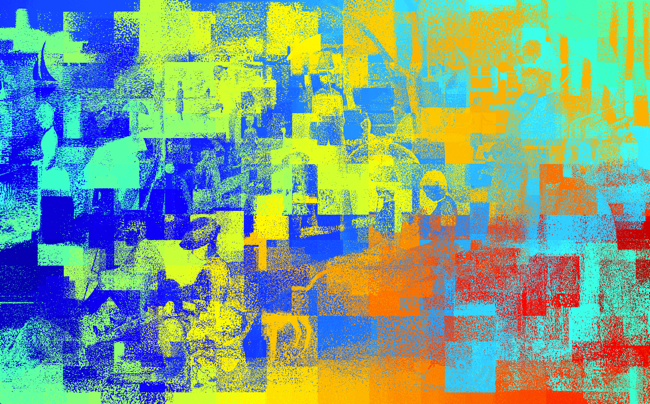


Fig6, Fig7. Test Code

为了便于观察超像素聚类SLIC的迭代结果，本文为label结果添加伪彩色。伪彩色应当能够使得各聚类之间明显可分辨，直接按照label数值赋灰度或者直接添加伪彩色，这样的label展示方式容易受到数值大小的影响，如Fig8和Fig9所示，各聚类之间难以分辨，不利于观察实验结果。



|  |  |  |
| --- | --- | --- |
| Fig14. Sunday Afternoon - Original | Fig15 Our Superpixel SLIC | Fig16. Our With Postprocessing |

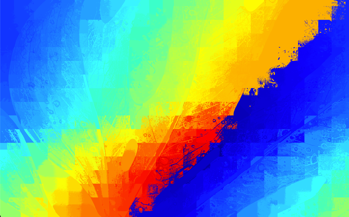


Fig8, Fig9. Bad Label Illustration

经过反复试验，本文的伪彩色添加代码如Fig10所示，它可以清晰展示聚类结果。 首先将label转换成彩色图像，采用的方式是将label值的特征有周期地映射到CIELAB色彩空间的亮度中，将大范围的label值映射为另外两个色彩分量，最后将其转换回BGR，再利用OpenCV为彩色图像应用热力图伪彩色。



Fig10. ShowLabels

对于Fig11和Fig14所示的彩色图像，应用本文实现的代码进行超像素聚类计算（参数：region\_size=100, num\_iterations=10）。分别得到如Fig12和Fig15所示的超像素聚类伪彩色图，可以清晰地看出各个像素的聚类结果。在进行后处理增强label的连接性后，得到的超像素聚类图如Fig13和Fig16所示，这符合一般情况下对实验结果的预期。

可以看到，本文实现的SLIC算法良好地完成了超像素聚类的查找，查找的超像素聚类同时兼顾了空间和色彩特征，基本上正确实现了原作者的SLIC算法。

# 实验分析

为了更好地说明本实验的正确性，在这一部分，将本文实现的算法与OpenCV实现的算法进行比较。在OpenCV中实现SLIC算法的方式如Fig17所示。



Fig17. OpenCV Implementation

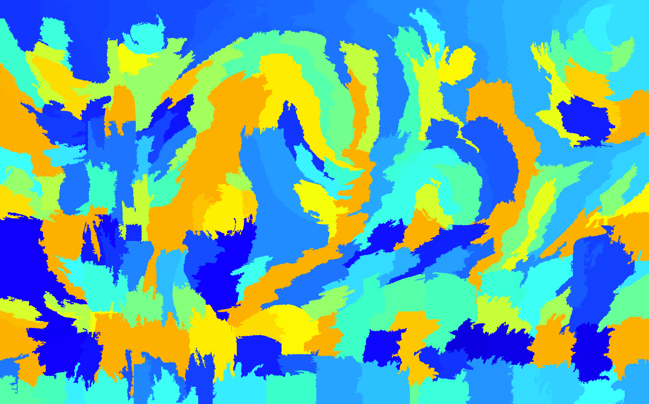


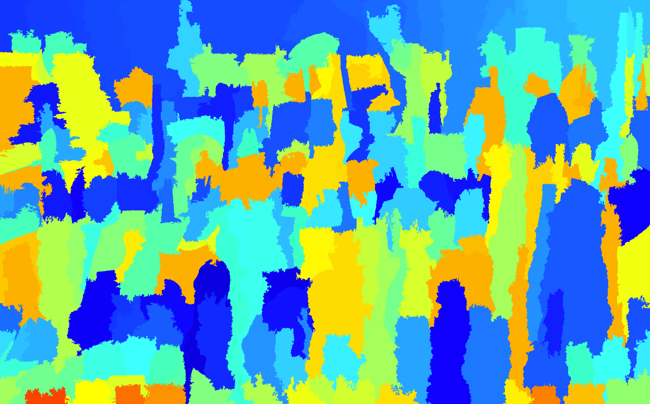
Fig18. The Starry Night - OpenCV 

Fig19. Sunday Afternoon - OpenCV

Fig18和Fig19分别是Fig11和Fig14在SLIC的OpenCV实现下执行的结果。可以看到，Fig18和Fig13基本一致，Fig19与Fig16也基本一致。推测产生差异的原因是OpenCV的ximgproc使用了并行运算来执行聚类中心搜索的循环。

值得注意的是，本实验仅对超像素聚类的SLIC算法本身进行了实现，没有进行相关算法的比较，这影响了实验结果的说服力。此外，虽然本实验的SLIC算法是正确和可用的，但实验对完整的SLIC模型进行了一定的简化处理，忽略了剩余误差和收敛性的判断。这可能对模型的执行速度和执行精度产生影响。

# 附录 实验源码

文件说明：main.cpp(测试代码); SuperPixelSLIC.cpp(本实验实现); SuprPixelSLIC.h(头文件).

实验环境：OpenCV 4.5.4-2, CMake 3.21.1, g++ version: Apple clang version 13.0.0 (clang-1300.0.29.3)

## main.cpp

1. #include <opencv2/opencv.hpp>
2. #include <opencv2/ximgproc.hpp>
3. #include "SuperPixelSLIC.h"
4. using namespace std;
5. using namespace cv;
6. using namespace ximgproc;
7. int region\_size = 100;
8. int num\_iterations = 10;
9. char img\_str[] = "Sunday Afternoon.jpg";
10. void showLabels(Mat label, const char \* winname);
11. void OurImplementation(Mat & src);
12. void OpenCVImplementation(Mat & src);
13. int main() {
14. Mat src = imread(img\_str,IMREAD\_COLOR), dst;
15. OurImplementation(src);
16. *//    OpenCVImplementation(src);*
17. waitKey(0);
18. return 0;
19. }
20. void showLabels(Mat label, const char \* winname){
21. *//为标签添加伪彩色以便观察*
22. Mat dst(label.rows, label.cols, CV\_8UC3);
23. for(int i = 0; i < label.rows; i++){
24. auto data = dst.ptr<Vec3b>(i);
25. for(int j = 0; j < label.cols; j++){
26. int c = label.at<int>(i,j);
27. data[j] = cv::Vec3b(5+c%50\*5, 5+(c/25)%25\*10,
28. 5+( (c+127)/25)%25\*10);
29. }
30. }
31. cvtColor( dst, label , COLOR\_Lab2BGR);*//转BGR*
32. applyColorMap(label,dst,COLORMAP\_JET);*//热力图方式*
33. imshow(winname, dst);
34. }
35. void OurImplementation(Mat & src){
36. SuperPixelSLIC s(src,region\_size);
37. s.iterate(num\_iterations);
38. s.enforceConnectivity();
39. showLabels(s.getLabels(), "labels - OurImplementation");
40. }
41. void OpenCVImplementation(Mat & src){
42. Ptr<SuperpixelSLIC> slic = createSuperpixelSLIC(
43. src,SLIC, region\_size);
44. slic->iterate(num\_iterations);
45. slic->enforceLabelConnectivity();
46. */\*labels\*/*
47. Mat labels;
48. slic->getLabels(labels);*//获取labels*
49. showLabels(labels, "labels - OpenCVImplementation");
50. }

## SuperPixelSLIC.h

1. #include <opencv2/ximgproc.hpp>
2. #include <opencv2/opencv.hpp>
3. #include <cmath>
4. #include <iostream>
5. using namespace std;
6. using namespace cv;
7. struct LABXY{
8. double l,a,b,x,y;
9. LABXY();
10. LABXY(double l, double a, double b, double x, double y);
11. void setLABXY(double \_l, double \_a, double \_b, double \_x, double \_y);
12. void print() const;
13. };
14. class SuperPixelSLIC{
15. private:
16. *//private params*
17. int rows, cols;
18. int gridRows, gridCols;
19. int S; *// clusterCenters interval*
20. int K;
21. Mat src\_lab, &src;
22. Mat labels,distance;
23. vector<LABXY> clusterCenters; *//聚类中心*
24. public:
25. */\*Initialization Step For SLIC\*/*
26. explicit SuperPixelSLIC(Mat & src, int S\_grid\_interval = 10);
27. void iterate(int times = 10, double compactness = 10.0);
28. Mat getLabels();
29. void enforceConnectivity(int minSegSize = 25);
30. private:
31. *//计算梯度*
32. static void calcGradient(Mat & src, Mat & grad);
33. };

## SuperPixelSLIC.cpp

1. *//*
2. *// Created by Li Chunliang on 2022/1/14.*
3. *//*
4. #include "SuperPixelSLIC.h"
5. #include <stack>
6. using namespace std;
7. LABXY::LABXY(): l(0), a(0), b(0), x(-1), y(-1){}
8. LABXY::LABXY(double l, double a, double b, double x, double y) : l(l), a(a), b(b), x(x), y(y) {}
9. void LABXY::setLABXY(double \_l, double \_a, double \_b, double \_x, double \_y) {
10. LABXY::l = \_l; LABXY::a = \_a; LABXY::b = \_b; LABXY::x = \_x; LABXY::y = \_y;
11. }
12. void LABXY::print() const {
13. printf("LABXY: %lf,%lf,%lf,%lf,%lf\n", l,a,b,x,y);
14. }
15. SuperPixelSLIC::SuperPixelSLIC(Mat & src, int S\_grid\_interval)
16. : src(src), S(S\_grid\_interval), rows(src.rows), cols(src.cols),
17. labels(rows, cols, CV\_32S, -1), */\*Initialize Label & Distance\*/*
18. distance(rows,cols,CV\_64F,FLT\_MAX){
19. *//Convert src Matrix to CIELAB*
20. cvtColor(src, src\_lab, COLOR\_BGR2Lab);
21. *//Pre-calculations for grid.*
22. gridRows = rows / S;
23. gridCols = cols / S;
24. S = min(src.rows / gridRows, src.cols / gridCols); *//make the S larger.*
25. */\*Initialization\*/*
26. K = gridRows \* gridCols;
27. clusterCenters.resize(K);
28. Mat grad;
29. calcGradient(src,grad); *//Gradient*
30. for(int i = 0; i < gridRows; i++){
31. for(int j = 0 ; j < gridCols; j++){
32. *//Initialize cluster centers by sampling pixels at regular grid steps S.*
33. int x = S/2 + i \* S; int y = S/2 + j \* S;
34. *//Move Cluster Centers to the lowest gradient position in a 3×3 neighborhood.*
35. int lowest\_grad = grad.at<short>(x, y);
36. int lg\_x = x; int lg\_y = y; *//'lg' for lowest gradient*
37. for(int p = -1; p <= 1; p++){
38. for(int q = -1; q <= 1; q++){
39. *//如果不超出图像边界*
40. if(x+p >=0 && x+p < rows && y+q >=0 && y+q < cols){
41. *//获得梯度值并记录最小梯度值*
42. int cgrad = grad.at<short>(x+p,y+q);
43. if(cgrad < lowest\_grad){
44. lowest\_grad = cgrad; lg\_x = x + p; lg\_y = y + q;
45. }
46. }
47. }
48. }
49. Vec<uchar, 3> tmp = src\_lab.at<Vec3b>(lg\_x, lg\_y);
50. clusterCenters[i \* gridCols + j]
51. .setLABXY(tmp[0], tmp[1], tmp[2], lg\_x, lg\_y);
52. }
53. }
54. }
55. void SuperPixelSLIC::iterate(int times, double compactness){
56. double factor = 1.0 / (( S \* compactness ) \* ( S \* compactness )); *//pre calculate the factor*
57. for(;times > 0; times--){
58. */\* Assignment \*/*
59. for(int i = 0 ; i < clusterCenters.size(); i++){
60. auto & tmp = clusterCenters[i];
61. if(tmp.x == -1 || tmp.y == -1) continue;
62. *//search around the cluster center*
63. for(int p = max(0, (int)tmp.x - S); p <= min(rows - 1, (int)tmp.x + S); p++){
64. for(int q = max(0, (int)tmp.y - S); q <= min(cols - 1, (int)tmp.y + S); q++){
65. *//update distance for a pixel*
66. Vec<uchar, 3> lab = src.at<Vec3b>(p, q);
67. LABXY li = tmp, lj = LABXY(lab[0], lab[1], lab[2], p, q);
68. double dc2 = (li.l - lj.l)\*(li.l - lj.l) + (li.a - lj.a)\*(li.a - lj.a)
69. + (li.b - lj.b)\*(li.b - lj.b);
70. double ds2 = (li.x - lj.x)\*(li.x - lj.x) + (li.y - lj.y)\*(li.y - lj.y);
71. double d = dc2 + ds2 \* factor;
72. if(d <= distance.at<double>(p,q)){
73. distance.at<double>(p,q) = d;
74. labels.at<int>(p, q) = i;
75. }
76. }
77. }
78. }
79. */\*Update\*/*
80. for(int i = 0 ; i < clusterCenters.size(); i++){
81. auto & tmp = clusterCenters[i];
82. if(tmp.x == -1 || tmp.y == -1) continue;
83. *//pixels belonging to the cluster center are counted*
84. double s[5] = {0}, cnt = 0; *// s for sum, cnt for count*
85. for(int p = max(0, (int)tmp.x - S); p <= min(rows - 1, (int)tmp.x + S); p++){
86. for(int q = max(0, (int)tmp.y - S); q <= min(cols - 1, (int)tmp.y + S); q++){
87. if(labels.at<int>(p, q) == i){
88. Vec<uchar,3> tmpVec = src\_lab.at<Vec3b>(p, q); *//GetLAB of Pixel*
89. s[0] += tmpVec[0]; s[1] += tmpVec[1]; s[2] += tmpVec[2]; *//LAB*
90. s[3] += p; s[4] += q; *// XY*
91. cnt++;
92. }
93. }
94. }
95. */\*recalculate the cluster center\*/*
96. if(cnt !=0) {
97. tmp.setLABXY(s[0] / cnt, s[1] / cnt, s[2] / cnt,s[3] / cnt, s[4] / cnt);
98. }else{
99. tmp.setLABXY(0,0,0,-1,-1);
100. K--;
101. }
102. }
103. }
104. }
105. Mat SuperPixelSLIC::getLabels(){
106. return labels;
107. }
108. void SuperPixelSLIC::calcGradient(Mat & src, Mat & grad){
109. Mat src\_gray; *//假定输入是彩色图像*
110. cvtColor(src, src\_gray, COLOR\_RGB2GRAY);
111. *// 创建 grad\_x 和 grad\_y 矩阵*
112. Mat grad\_x, grad\_y;
113. Mat abs\_grad\_x, abs\_grad\_y;
114. *// 求 X方向梯度*
115. Sobel( src\_gray, grad\_x, CV\_16S, 1, 0, 3, 1, 0, BORDER\_DEFAULT );
116. convertScaleAbs( grad\_x, abs\_grad\_x );
117. *// 求Y方向梯度*
118. Sobel( src\_gray, grad\_y, CV\_16S, 0, 1, 3, 1, 0, BORDER\_DEFAULT );
119. convertScaleAbs( grad\_y, abs\_grad\_y );
120. *// 合并梯度(近似)*
121. addWeighted( abs\_grad\_x, 0.5, abs\_grad\_y, 0.5, 0, grad );
122. }
123. void SuperPixelSLIC::enforceConnectivity(int minSegSize) {
124. if (minSegSize == 0 || minSegSize > 100) return;
125. *//四邻域连接*
126. const int dx4[4] = { -1,  0,  1,  0 };
127. const int dy4[4] = {  0, -1,  0,  1 };
128. const int magic = max(3, cols \* rows / K / int(100.0/(float)minSegSize + 0.5));
129. Mat l\_src = labels;
130. labels = Mat(rows, cols, CV\_32S, -1);
131. int label = 0;
132. int adjLabel = 0; *//adjacent label*
133. vector<int> componentPoints(rows\*cols);
134. for (int x = 0; x < rows; x++){
135. for (int y = 0; y < cols; y++){
136. if (labels.at<int>(x, y) == -1 ){*// Seed found*
137. labels.at<int>(x,y) = label;
138. componentPoints[0] = y + x \* cols;
139. *//Find an adjacent label*
140. for(int i = 0; i < 4; i++){
141. int p = componentPoints[0] / cols + dx4[i];
142. int q = componentPoints[0] % cols + dy4[i];
143. if( p >=0 && p < rows && q >=0 && q < cols){
144. if(labels.at<int>(p,q) != -1){
145. adjLabel = labels.at<int>(p, q);
146. break;
147. }
148. }
149. }
150. *//iterate and grow*
151. int count = 1;
152. for( int t = 0; t < count; t++){
153. for(int i = 0; i < 4; i++){
154. int p = componentPoints[t] / cols + dx4[i];
155. int q = componentPoints[t] % cols + dy4[i];
156. if( p >=0 && p < rows && q >=0 && q < cols){
157. if(labels.at<int>(p,q) == -1 &&
158. l\_src.at<int>(p,q) == l\_src.at<int>(x,y)){
159. componentPoints[count] = q + p \* cols;
160. labels.at<int>(p,q) = label;
161. count++;
162. }
163. }
164. }
165. }
166. *//if segment size is no bigger than a threshold,*
167. *// allocate it to previous adjacent segmant*
168. if(count <= magic){
169. for(int t = 0; t < count; t++){
170. labels.at<int>(componentPoints[t] / cols,
171. componentPoints[t] % cols) = adjLabel;
172. }
173. label--;
174. }
175. label++;
176. }
177. }
178. }
179. *//    return labels;*
180. }