

报错时有时无

0是黑色，255是白色。

一、联通组件标记算法原理

  很长时间没更新了，今天开始继续更新OpenCV学习笔记相关内容。今天说的是图像处理中最常用的分析方法：连通组件标记算法。这里只介绍两步法，也就是OpenCV库中自带的那种。

  两步法的连通组件标记算法的实质是分两步扫描二值图像。第一步扫描图像的每个像素点，对于像素值相同的而且相互连通分为相同的组(group)，标记为Label 1，持续进行扫描，最终得到图像中所有的像素连通组件。扫描的方式可以是从上到下，从左到右，对于一幅有N个像素的图像来说，如果以4邻域计算，最大连通组件个数为N/2。第一步完成以后会获得多个Label，有的Label与Label之间是有像素点连接在一起的，这些Label可以合并为一个Label，这就是第二步需要做的工作。

在正式讲解两步法前先回顾下邻域的概念：

如果一个像素P(x,y)，它和它左边的像素、上边的像素、右边的像素、下边的像素相邻，那么我们称为4邻域，就像下面的图中黄色部分：

如果一个像素P(x,y)，它和它左边的像素、左上的像素、左下的像素、上边的像素、右上的像素、右边的像素、右下的像素、下边的像素相邻，那么我们称为8邻域，就像下面的图中黄色部分：

  之所以再次介绍邻域是因为OpenCV中联通组件标记算法的API函数中默认是8邻域统计的。接下来以图片的形式展示两步法：

第一步：按照从左到右、从上到下的顺序扫描图像中所有像素点，给每个不同的连通域赋予不同的labels，如下图所示，在第一步执行完以后会出现总共10个labels。假如以左上顶点作为坐标原点(0、0)，需要注意的是第三行，第六列的这个点P(2、5)，该点被赋予label 4而不是label 6的原因是我们这里使用的是8邻域的查找方法，同时这里还会标记label 6 -> label 4，标签6隶属于标签4。这样在第二步的时候就会将具有从属关系的标签进行合并。

第二步：将第一步中统计出来的具有从属关系的label进行合并，合并完成以后再将标签重新排序，得到如下图所示结果：

这样就统计出来总共有8个不同的连通域存在。注：如果是4邻域统计的话，就会有12个不同的连通域。

二、OpenCV中的API函数

  下面是OpenCV中关于联通标记组件算法的API函数，总共有两种，一种是不带状态标记的，一种是带状态标记的。

//带状态标记的API函数

CV\_EXPORTS\_W int connectedComponentsWithStats(InputArray image, OutputArray labels,

OutputArray stats, OutputArray centroids,

int connectivity = 8, int ltype = CV\_32S);

//不带状态标记的API函数

CV\_EXPORTS\_W int connectedComponents(InputArray image, OutputArray labels,

int connectivity = 8, int ltype = CV\_32S);

这里只介绍带状态标记的API函数

image：输入图像，格式必须是8位单通道图像，可以是灰度图，也可以是二值化图

labels：输出的标签图像，格式由最后一个参数ltype决定，一般是默认的CV\_32S

stats：输出的统计信息，里面包含每个连通域的面积，最小外接矩形的x、y、width、height等；

centroids：每个连通域的质心坐标；

connectivity：决定连通域寻找方法，有4邻域和8邻域两种，默认是8邻域；

ltype：决定labels的图像格式，有CV\_32S和CV\_16U两种

三、代码实现

先将效果图展示出来，如下图所示：

分别统计出了连通域个数、每个连通域的面积、质心坐标等参数。

// ConnectComponent.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。

//

#include "pch.h"

#include <iostream>

#include "opencv2/opencv.hpp"

using namespace std;

using namespace cv;

RNG rng(12345);

//将RGB图片拆分成3个通道图像,转化为单通道灰度图

void SplitRGB2SingleGray(Mat Image, int i, Mat &Gray\_Single\_Image)

{

vector<Mat>RGB\_Channels(3);

split(Image, RGB\_Channels);

Gray\_Single\_Image.create(Image.rows, Image.cols, CV\_8UC1);

for (int row = 0; row < Gray\_Single\_Image.rows; row++)

{

for (int col = 0; col < Gray\_Single\_Image.cols; col++)

Gray\_Single\_Image.at<uchar>(row, col) = RGB\_Channels[i].at<uchar>(row, col);

};

}

//过滤掉红色细胞中绿色部分干扰

void Filter\_G\_Channel(Mat MainImage, Mat AssistantImage, Mat &OutImage)

{

for (int row = 0; row < MainImage.rows; row++)

{

for (int col = 0; col < MainImage.cols; col++)

{

if (MainImage.at<uchar>(row, col) - AssistantImage.at<uchar>(row, col) > 0)

OutImage.at<uchar>(row, col) = MainImage.at<uchar>(row, col);

else

OutImage.at<uchar>(row, col) = 0;

}

}

}

void ConnectComponent\_demo(Mat src)

{

Mat Gray\_R\_Channel, Gray\_G\_Channel;

SplitRGB2SingleGray(src, 2, Gray\_R\_Channel); //生成R通道灰度图

SplitRGB2SingleGray(src, 1, Gray\_G\_Channel); //生成G通道灰度图

Mat FilterGrayImage;

FilterGrayImage = Mat::zeros(Gray\_R\_Channel.rows, Gray\_R\_Channel.cols, CV\_8UC1);

Filter\_G\_Channel(Gray\_R\_Channel, Gray\_G\_Channel, FilterGrayImage);

imshow("FilterImage", FilterGrayImage);

Mat BinImage;

threshold(FilterGrayImage, BinImage, 120, 255, THRESH\_BINARY);

imshow("BinImage", BinImage);

Mat LabelImage;

int Num\_Label = 0;

Num\_Label = connectedComponents(BinImage, LabelImage, 8, 4);

printf("Total\_Label:%d\n", Num\_Label - 1);

//imshow("LabelImage", LabelImage);

vector<Vec3b>colors(Num\_Label);

colors[0] = Vec3b(0, 0, 0); //背景颜色设置为黑色

for (int i = 1; i < Num\_Label; i++)

colors[i] = Vec3b(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));

//输出标记后的图像

Mat LabelImage\_Color;

LabelImage\_Color = Mat::zeros(src.size(), src.type());

for (int row = 0; row < LabelImage\_Color.rows; row++)

{

for (int col = 0; col < LabelImage\_Color.cols; col++)

{

int Label = 0;

Label = LabelImage.at<int>(row, col);

if (Label != 0)

LabelImage\_Color.at<Vec3b>(row, col) = colors[Label];

}

}

imshow("LabelImage", LabelImage\_Color);

}

void ConnectComponent\_stats\_demo(Mat src)

{

Mat Gray\_R\_Channel, Gray\_G\_Channel;

SplitRGB2SingleGray(src, 2, Gray\_R\_Channel); //生成R通道灰度图

SplitRGB2SingleGray(src, 1, Gray\_G\_Channel); //生成G通道灰度图

Mat FilterGrayImage;

FilterGrayImage = Mat::zeros(Gray\_R\_Channel.rows, Gray\_R\_Channel.cols, CV\_8UC1);

Filter\_G\_Channel(Gray\_R\_Channel, Gray\_G\_Channel, FilterGrayImage);

imshow("FilterImage", FilterGrayImage);

Mat BinImage;

threshold(FilterGrayImage, BinImage, 120, 255, THRESH\_BINARY);

imshow("BinImage", BinImage);

Mat LabelImage, stats, centroids;

int Num\_Label = 0;

Num\_Label = connectedComponentsWithStats(BinImage, LabelImage, stats, centroids, 8, 4);

printf("Total\_Label:%d\n", Num\_Label - 1);

//imshow("LabelImage", LabelImage);

vector<Vec3b>colors(Num\_Label);

colors[0] = Vec3b(0, 0, 0); //背景颜色设置为黑色

for (int i = 1; i < Num\_Label; i++)

colors[i] = Vec3b(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));

//输出标记后的图像

Mat LabelImage\_Color;

LabelImage\_Color = Mat::zeros(src.size(), src.type());

for (int row = 0; row < LabelImage\_Color.rows; row++)

{

for (int col = 0; col < LabelImage\_Color.cols; col++)

{

int Label = 0;

Label = LabelImage.at<int>(row, col);

if (Label != 0)

LabelImage\_Color.at<Vec3b>(row, col) = colors[Label];

}

}

for (int j = 1; j < Num\_Label; j++)

{

Vec2d point;

point = centroids.at<Vec2d>(j, 0);

int x, y, height, width, area;

x = stats.at<int>(j, CC\_STAT\_LEFT);

y = stats.at<int>(j, CC\_STAT\_TOP);

width = stats.at<int>(j, CC\_STAT\_WIDTH);

height = stats.at<int>(j, CC\_STAT\_HEIGHT);

area = stats.at<int>(j, CC\_STAT\_AREA);

printf("Area:%d,center point:%.2f,%.2f\n", area, point[0], point[1]);

circle(LabelImage\_Color, Point(point[0], point[1]), 1, Scalar(0, 0, 255), 2, 8, 0);

rectangle(LabelImage\_Color, Rect(x, y, width, height), Scalar(0, 255, 0), 1, 8, 0);

putText(LabelImage\_Color, format("Num:%d", j), Point(point[0], point[1]), FONT\_HERSHEY\_SIMPLEX, 0.4, Scalar(0, 0, 255), 1, 8, false);

}

imshow("LabelImage", LabelImage\_Color);

}

int main()

{

Mat InImage;

InImage = imread("cell.jpg", IMREAD\_COLOR);

if (InImage.data == nullptr)

{

cout << "Image load error!" << endl;

return 0;

}

imshow("SrcImage", InImage);

//ConnectComponent\_demo(InImage);

ConnectComponent\_stats\_demo(InImage);

while (true)

{

if (waitKey(10) == 27)

break;

}

}