C/C++:OTSU阈值分割

2020-10-16 | 图像处理 | 9 |

一、什么是OTSU阈值分割

OTSU分割的基本思想是：选择一个阈值分割图像后，两部分的类间方差最大（两个类差异最大），两部分各自的类内方差最小（每个部分比较集中）。

二、OTSU公式

注意，为了程序的运行速度，下面的公式和纯数学公式的变量含义稍有差别。

基本变量

像素分成两类，每一类像素各自有三个基本变量：

- w 像素总个数
- u 像素加权和（权重为像素灰度）
- v 像素加平方权和（权重为像素灰度的平方）

  现在给出三个参数的定义，以小于待定阈值t的参数为例，大于阈值t的参数只是修改求和符号的累加范围：

$$w_0 = \sum_{i=0}^{t-1} n_i$$
$$u_0 = \sum_{i=0}^{t-1} i \times n_i$$
$$v_0 = \sum_{i=0}^{t-1} i^2 \times n_i$$

指标变量

两类像素各自有衡量自身离散程度的指标：

- delta 方差
- deltaB 类间方差
- deltaW 类内方差
- deltaT 总方差

  这里给出这些指标变量的表达式：

$$\sigma_0 = \frac{v_0}{w_0} - (\frac{u_0}{w_0})^2$$
$$\sigma_b = w_0 \times w_1 \times (\frac{u_0}{w_0} - \frac{u_1}{w_1})^2 \times \frac{1}{(M \times N)^2}$$
$$\sigma_w = (v_0 + v_1 - \frac{{u_0}^2}{w_0} - \frac{{u_1}^2}{w_1}) \times \frac{1}{M \times N}$$

判决准则

数学上一共有三个判别准则，这三个准则等价：

$$\lambda(t)=\frac{\sigma_B^2}{\sigma_w^2}$$
$$\eta(t)=\frac{\sigma_B^2}{\sigma_T^2}$$
$$\kappa(t)=\frac{\sigma_T^2}{\sigma_w^2}$$

由于总方差和待定阈值无关，所以我们选择第二个判别准则，使第二个公式值最大的阈值t作为最终结果。

三、算法思路

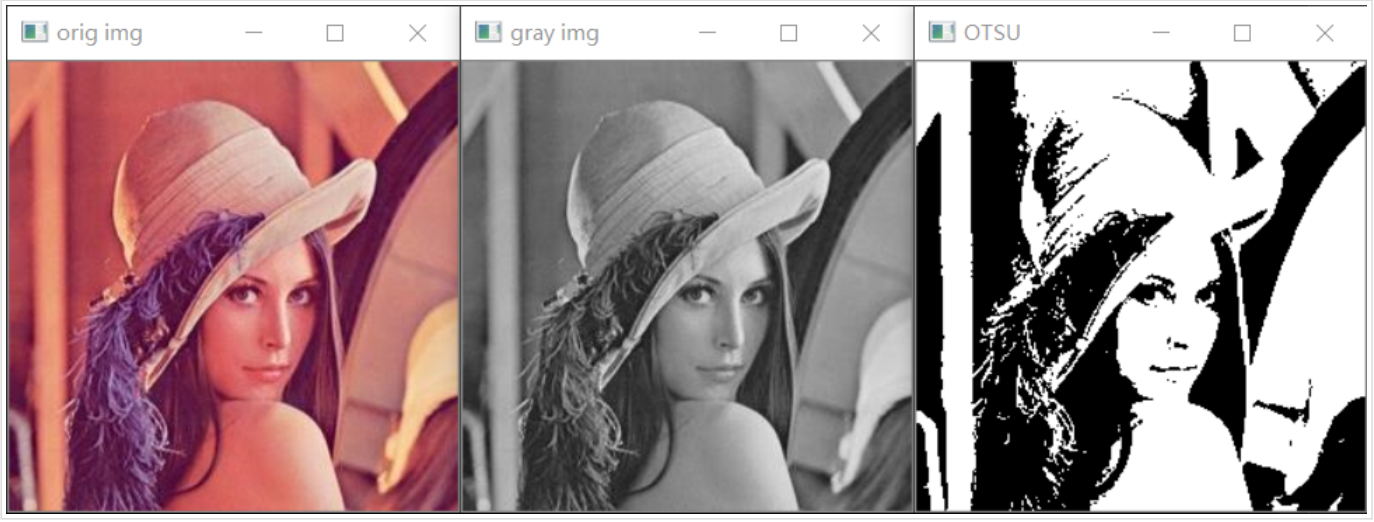
设置一个变量t作为待定的阈值，然后t遍历整个灰度级（实际上1-254即可），每一个t算一个类间方差值，如果这个类间方差值比最大类间方差值大，那么更新最大类间方差值，并将这次的t作为待定阈值。

四、代码实现

```
int otsu(cv::Mat &input_img)
{
        int M = input_img.rows;
        int N = input_img.cols;
        int size = M * N;

        double deltaB, deltaW, etaMax = 0.0;
        long int w0; //灰度小于t的像素个数
        long int w1; //灰度大于t的像素个数
        double u0; //灰度小于t的像素的加权和，权重为灰度值（不是灰度值/灰度范围，方便后续计算）
        double u1; //灰度大于t的像素的加权和，权重为灰度值（不是灰度值/灰度范围，方便后续计算）
        double v0; //灰度大于t的像素的加权和，权重为灰度值的平方（不是灰度值/灰度范围，方便后续计算）
        double v1; //灰度大于t的像素的加权和，权重为灰度值的平方（不是灰度值/灰度范围，方便后续计算）
        double u0_2;
        double u1_2;

        int gray_level, gray_arr[256] = { 0 };
        int t, t_optm = 127;
        long int cnt1, cnt2;

        //统计灰度
        for (cnt1 = 0; cnt1 < M; ++cnt1)
        {
                for (cnt2 = 0; cnt2 < N; ++cnt2)
                {
                        gray_level = input_img.at<uchar>(cnt1, cnt2);
                        gray_arr[gray_level] += 1;
                }
        }

        //计算最佳阈值t
        for (t = 1; t < 254; ++t)
        {
                w0 = w1 = 0;
                u0 = u1 = v0 = v1 = 0.0;
                for (cnt1 = 0; cnt1 < t; ++cnt1)
                {
                        w0 += gray_arr[cnt1];
                        u0 += cnt1 * gray_arr[cnt1];
                        v0 += cnt1^2 * gray_arr[cnt1];
                }
                for (cnt2 = t; cnt2 < 256; ++cnt2)
                {
                        //w1 += gray_arr[cnt2]; // (不用累加)
                        u1 += cnt2 * gray_arr[cnt2];
                        v1 += cnt2^2 * gray_arr[cnt2];
```

```
                }
                w1 = size - w0;


                u0_2 = pow(u0, 2);
                u1_2 = pow(u1, 2);

                //deltaB计算公式
                //                     u0   u1            1
                //deltaB = w0 * w1 * ( —— - —— )^2 * ——————————
                //                     w0   w1       (M * N) ^2

                deltaB = w0 * w1 * pow( (u0 / w0 - u1 / w1), 2 ) / size / size;

                //delta0计算公式（不用计算）
                //         v0     u0
                //delta0 = —— - ( —— )^2
                //         w0     w0

                //deltaW计算公式（不用计算）
                //            1                 u0^2   u1^2
                //deltaW  = ———— * ( v0 + v1 - ——— - ——— )
                //          M * N                w0     w1
                //deltaW = (v0 + v1 - u0_2 / w0 - u1_2 / w1) / size;

                if (deltaB > etaMax)
                {
                        t_optm = t;
                        etaMax = deltaB;
                }
        }

        return t_optm;
}
```

五、实验结果



六、失误和总结

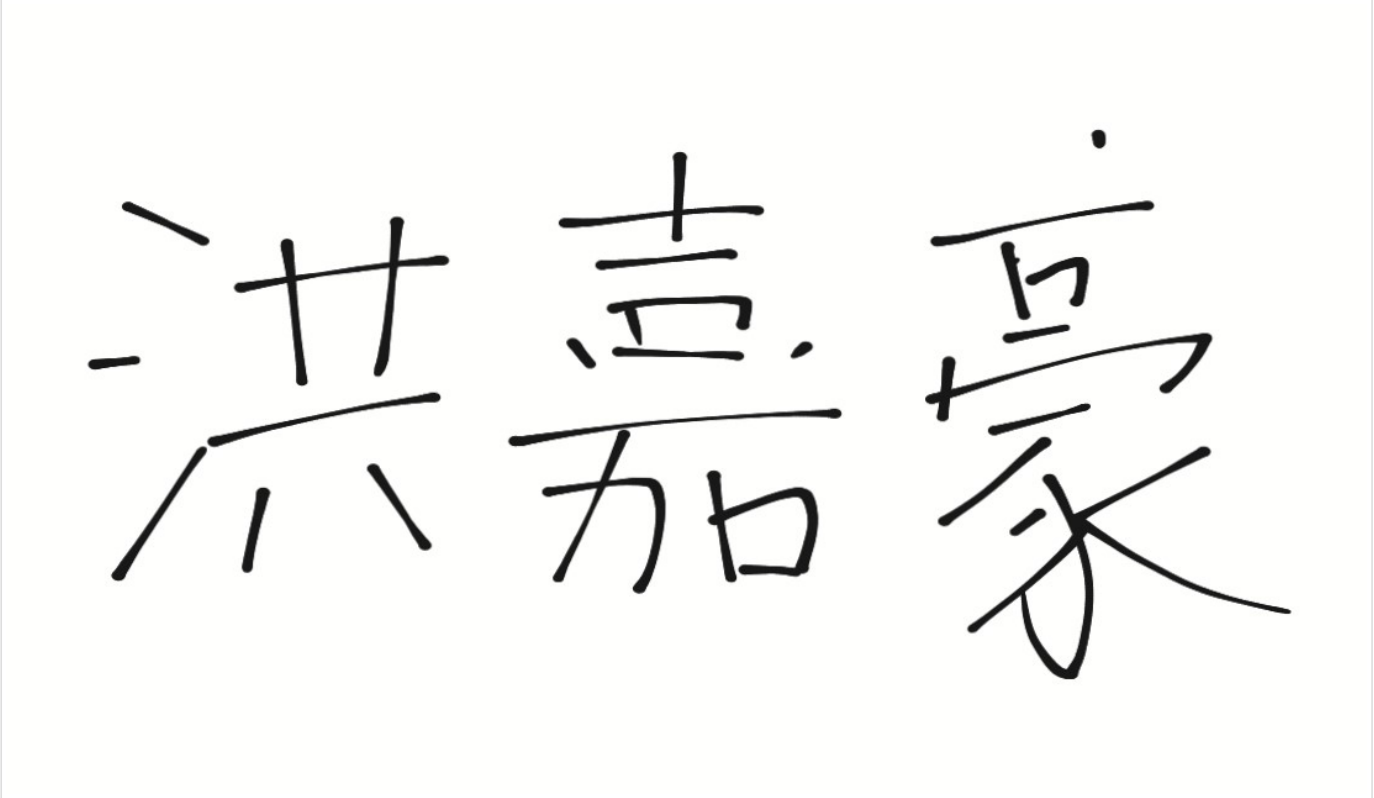
强制类型转换

写完程序后，测试结果总是1，即deltaB运算结果一直为0，调试过程中查看了很多变量也没发现问题，最后鼠标一个个查声明的变量时发现问题：因为**数字1**和**变量size**都是**int类型**的，在进行除法时结果是0，所以之后的计算结果一直为0，达不到预期效果。

```
int otsu(cv::Mat &input_img)
{
    int M = input_img.rows;
    int N = input_img.cols;
    int size = M * N;
    double size_ = 1 / size;
    double siz  size_ 0.00000000000000000
```

double类型的表示范围

最初采用的判别标准是namda公式，在计算deltaW的时候由于u0和u1变量中有一个会很大，导致其平方会超出double表示范围，体现的结果就是deltaW出现负值，而在数学上这个值应该恒为正（两个平方数相加）。之后考虑到计算量，采用了eta公式，并且只用计算deltaB，简化了计算。



C/C++ # 图像处理

C/C++:中值滤波和均值滤波

© 2020 Darcy
Powered by Hexo | Theme — NexT.Mist v5.1.4