C/C++:一维熵阈值分割

2020-10-17 | 图像处理 | 23 |

一、什么是一维熵阈值分割

信息熵表示从信号中可能获得的信息的多少。
就比如投硬币，出席那正反两面的概率都是0.5，那么根据信息熵的公式得出，“硬币是正面”这个预测所携带的信息熵是：

$$I_1 = -p_1 \ln(p_1) = -0.5 \times \ln(0.5) = 0.35$$

如果投出正面的概率是0.9，那么“硬币是正面”这个预测所携带的信息熵是：

$$I_2 = -p_2 \ln(p_2) = -0.9 \times \ln(0.9) = 0.095$$

如果投出正面的概率是0.1，那么“硬币是正面”这个预测所携带的信息熵是：

$$I_3 = -p_3 \ln(p_3) = -0.1 \times \ln(0.1) = 0.23$$

可以看到，当硬币出现正反面的概率一样的时候，“硬币是正面”这个预测所携带的信息熵最大，也就是信息量最大。同理可以推广至更多种预测结果的情况，比如投正二十面体，当“出现任意一个面朝上的概率相同”时，“第13号面朝上”这个预测所携带的信息量最大。

回到图像上，图像有256个灰度级，若“一个像素是0到255中的任意一个的概率相同”，那么这张图象所携带的信息量最大。但不用多想，这种图就像黑白电视机没信号时候的图像一样，没有可解读的含义。至于为什么如此，还需要更深刻的理解，我暂时解释不清。

上面讲到了整幅图像的信息，现在更进一步，将图像分成两部分，其一是目标区域，其二是背景区域。如果二者都大致符合正态分布，那么图像的熵可以表示为目标区域的熵和背景区域的熵之和。基于此，一维熵阈值分割的目标就是，寻找一个分割灰度t，使得目标区域和背景区域的熵之和最大。

二、一维熵阈值分割公式

基本变量

像素分成两类，一类是目标区域，一类是背景区域，每一类像素各自有两个基本变量：

- w 像素总个数
- e 区域的一维熵（用像素个数代替像素概率密度，并且不加负号）
  由于不知道小于阈值t的像素属于目标还是背景，所以将小于阈值t的像素集设定为0。下面以小于阈值t的像素为例：

$$w_0 = \sum_{i=1}^{t} n_i$$

$$e_0 = \sum_{i=0}^{t} n_i \times \ln(n_i)$$

指标变量

指标变量有两个，一个是目标区域的一维熵，另一个是背景区域的一维熵，同样由于不知道背景和目标谁比较亮，所以Ho和HB都是相对来说的，一个为目标，一个为背景，二者等价。假设目标的灰度级小于背景，则定义如下：

$$H_o = -\sum_{i=1}^{t}(p_i / p_t)\times \ln(p_i / p_t)$$
$$H_B = -\sum_{i=t+1}^{L-1}(p_i / (1-p_t))\times \ln(p_i / (1-p_t))$$

判断条件

总体的一维熵表示为目标区域一维熵和背景一维熵之和，化简之后结果如下（使用数学上的定义，pi代表第i各灰度级的概率密度）：

$$\varphi(t) = H_O + H_B = \lg p_t(1-p_t) + \frac{H_t}{p_t} + \frac{H_L - H_t}{1-p_t}$$
$$H_t = -\sum_i p_i \lg p_i, \qquad i = 1,2,\cdots t$$
$$H_L = -\sum_i p_i \lg p_i, \qquad i = 1,2,\cdots L$$

为了提高程序效率，我们进一步将该公式变换成利于编程的形式，采用我们定义的基本变量表示，将该公式的三个部分分别化简成如下公式：

$$a = \ln(w_0) + \ln(M\times N - w_0) - 2\ln(M\times N)$$
$$b = \frac{-e_0}{w_0} + \ln(M\times N)$$
$$c = \frac{-e_1}{w_1} + \ln(M\times N)$$

三、程序思路

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算一个一维熵，如果这个一维熵比最大一维熵大，那么更新最大一维熵，并将这次的t作为待定阈值。

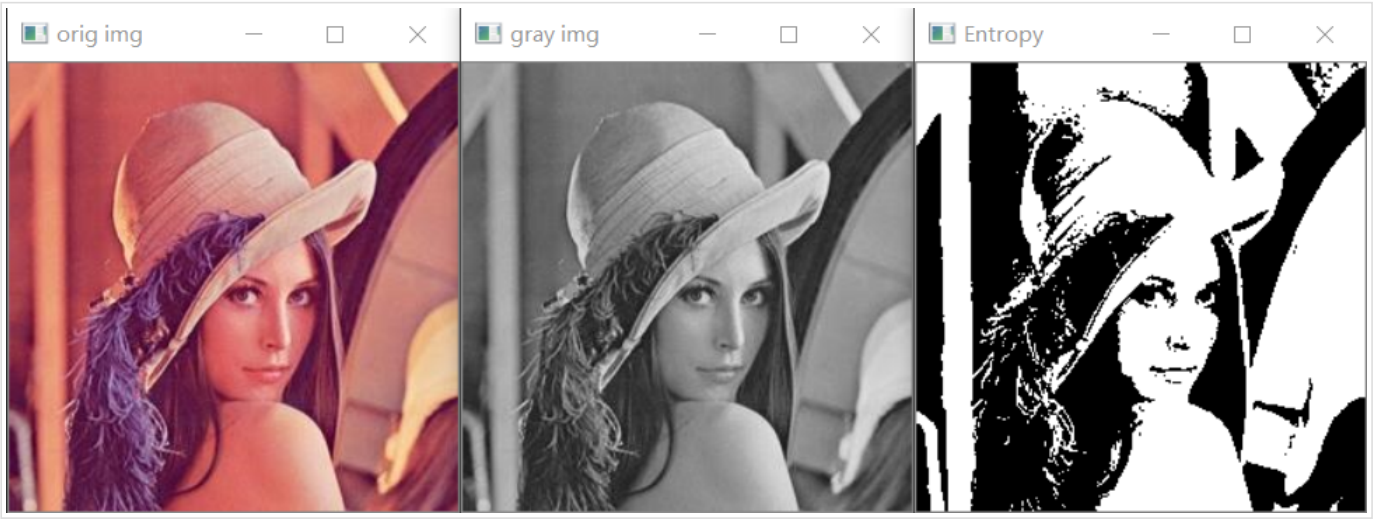
四、代码实现

```
int entropy(cv::Mat &input_img)
{
        int M = input_img.rows;
        int N = input_img.cols;
        int size = M * N;
        double log_size = std::log(M*N);

        double a, b, c; //公式里相加的三项
        double phi, phiMax = 0.0; //phi公式最大值
        int w0;    //小于阈值t的像素的个数
        double e0; //小于阈值t的像素一维熵（用像素个数代替概率）
        int w1;    //大于阈值t的像素的个数
        double e1; //大于阈值t的像素一维熵（用像素个数代替概率）

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

        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;
                }
        }

        for (t = 1; t < 254; ++t)
        {
                w0 = w1 = 0;
                e0 = e1 = 0.0;
```

```
32
33                  //小于阈值t的像素的参数
34                  for (cnt1 = 0; cnt1 < t; ++cnt1)
35                  {
36                          gray_num = gray_arr[cnt1];
37                          w0 += gray_num;
38                          if (gray_num != 0)
39                          {
40                                  e0 += gray_num * std::log(gray_num);
41                          }
42                  }
43
44                  //大于阈值t像素的参数
45                  for (cnt2 = t; cnt2 < 256; ++cnt2)
46                  {
47                          gray_num = gray_arr[cnt2];
48                          //w1 += gray_arr[cnt2]; // (不用累加)
49                          if (gray_num != 0)
50                          {
51                                  e1 += gray_num * std::log(gray_num);
52                          }
53                  }
54                  w1 = size - w0;
55
56                  a = std::log(w0) + std::log(size - w0) - 2 * log_size;
57                  b = log_size - e0 / w0;
58                  c = (log_size*w1 - e1) / w1;
59
60                  phi = a + b + c;
61                  if (phi > phiMax)
62                  {
63                          t_optm = t;
64                          phiMax = phi;
65                  }
66          }
67
68          return t_optm;
69  }
```

五、实验结果

阈值120



六、失误和总结

std::log()函数输入范围

由于没考虑到一个灰度级的像素个数可能为0的情况，导致输入std::log()函数的值为0，输出为nan，后续程序无法运行，后来增加了对某个灰度级像素个数的判断，只有当其不为0时才继续输入std::log()函数，进行基本变量的累加。

```
for (cnt1 = 0; cnt1 < t; ++cnt1)
{
    w0 += gray_arr[cnt1];
    e0 += std::log(gray_arr[cnt1])*gray_arr[cnt1];
}
```

错误代码

```
for (cnt1 = 0; cnt1 < t; ++cnt1)
{
    gray_num = gray_arr[cnt1];
    w0 += gray_num;
    if (gray_num != 0)
    {
        e0 += gray_num * std::log(gray_num);
    }
}
```

考虑0输入代码

C/C++ # 图像处理

C/C++:OTSU阈值分割 Matlab:直方图均衡化vs同态滤波