CV 作业二报告

姓名：秦昇　学号：3120000060　课程：计算机视觉

软件开发说明：

1. 基于 OpenCV 库，对含有细胞的图像进行处理。
2. 统计图像中所含细胞数量。
3. 输出最大和最小细胞的面积，周长（以像素为单位），方向，和该细胞的中心位置。
4. 输出所有细胞的平均面积。
5. 主要的中间步骤处理结果以 cvShowImage 的形式输出。

算法具体步骤

1. **图像的读入和转化**

读入图像后转化为灰度图像

```
    // Load image to src and Show source image
Mat source = imread("cell1.bmp", CV_LOAD_IMAGE_COLOR);
    imshow("Source image", source);


    // Get gray graph of source image.
```

```
    Mat gray;
    cvtColor(source, gray, CV_RGB2GRAY);
 imshow("Gray image", gray);
```

2. 分离细胞与背景，并将图像二值化

```
int thresholdn = Otsu(gray);
Mat bin;
threshold(gray, bin, thresholdn, 255, CV_THRESH_BINARY);
```

在分离细胞与背景图像的时候，最重要的是要找到一个合适的阈值，来进行图像和背景的二值化分离。这里我阅读了一些参考文献，决定采用 otsu 算法即最大类间方差法，又称大津算法。它是按图像的灰度特性,将图像分成背景和目标 2 部分。背景和目标之间的类间方差越大,说明构成图像的 2 部分的差别越大,当部分目标错分为背景或部分背景错分为目标都会导致 2 部分差别变小。因此,使类间方差最大的分割意味着错分概率最小。

3. 获取细胞外围轮廓，及其面积、周长等。

这里我们使用 findContours()函数来实现对细胞轮廓的获取，在此基础上舍弃一些轮廓，留下最终我们需要的，由于图像中有些和细胞颜色接近而面积远小于细胞的杂质，因此我们还需要对面积进行判断并且舍弃部分杂质。

```
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
Mat cont = Mat::zeros(bin.size(), CV_8UC1);
Mat ellp = Mat::zeros(bin.size(), CV_8UC1);
findContours(bin, contours, hierarchy,
   CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
```

计算平均面积：

```
for (int i = 0; i < contours.size(); i++){
      if (hash[i]){
          area[i] = contourArea(contours[i]);
          totalarea += area[i];
```

```
        }
    }


    int average = totalarea / counter;
```

杂质的判断并记录最大最小细胞的编号

```
    int minarea = cont.rows*cont.cols, maxarea = 0, mini, maxi;
    for (int i = 0; i < contours.size(); i++){
        if (hash[i])
            if (area[i]<average * 0.3){
                hash[i] = false;
                counter--;
            }
            else{
                if (area[i]<minarea){
                    minarea = area[i];
                    mini = i;
                }
                if (area[i]>maxarea){
                    maxarea = area[i];
                    maxi = i;
                }
            }
    }
```

4. 对细胞进行椭圆拟合，输出最大最小细胞的信息

虽然只需输出最大最小细胞的信息，但为了便于观察分析我对所有细胞都进行了椭圆拟合。

```
    if (hash[i]){
        CvPoint center;
```

```
          CvSize size;
          CvBox2D32f* box;
          CvPoint* PointArray;
          CvPoint2D32f* PointArray2D32f;

          PointArray = (CvPoint*)malloc(count*sizeof(CvPoint));
          PointArray2D32f =
(CvPoint2D32f*)malloc(count*sizeof(CvPoint2D32f));

          // Alloc memory for ellipse data.
          box = (CvBox2D32f*)malloc(sizeof(CvBox2D32f));



          // Convert CvPoint set to CvBox2D32f set.
          for (int j = 0; j<count;j++){
             PointArray2D32f[j].x = (float)contours[i][j].x;
             PointArray2D32f[j].y = (float)contours[i][j].y;
          }

          // Fits ellipse to current contour.
          cvFitEllipse(PointArray2D32f, count, box);

          // Convert ellipse data from float to integer
representation.

          center.x = cvRound(box->center.x);
          center.y = cvRound(box->center.y);
          size.width = cvRound(box->size.width*0.5);
          size.height = cvRound(box->size.height*0.5);
```

```
            drawContours(cont, contours, i, 255);
            ellipse(ellp, center, size, box->angle, 0, 360, 255,
1, CV_AA, 0);



            if (i == mini)
            cout << "最小细胞面积为" << area[i] << ',' << "周长：" <<
count << ','
                << "短轴与X+轴成：" << 180 - box->angle << "度角," <<
"中心" << '(' << center.x << ',' << center.y << ");"<<endl;
            if (i == maxi)
                cout << "最大细胞面积为" << area[i] << ',' << "周长："
<< count << ','
                << "短轴与X+轴成：" << 180 - box->angle << "度角," <<
"中心" << '(' << center.x << ',' << center.y << ");" << endl;
            // Free memory.
            free(PointArray);
            free(PointArray2D32f);
            free(box);
        }
```

算法实现要点

1. Otsu 算法

对于图像 I(x,y),前景(即目标)和背景的分割阈值记作 T,属于前景的像素点数占整幅图像的比

例记为 ω0,其平均灰度 μ0;背景像素点数占整幅图像的比例为 ω1,其平均灰度为 μ1。图像的总平均灰度记为 μ,类间方差记为 g。

假设图像的背景较暗,并且图像的大小为 M×N,图像中像素的灰度值小于阈值 T 的像素个数记作 N0,像素灰度大于阈值 T 的像素个数记作 N1,则有:

ω0=N0/ M×N (1)

ω1=N1/ M×N (2)

N0+N1=M×N (3)

ω0+ω1=1 (4)

μ=ω0*μ0+ω1*μ1 (5)

g=ω0(μ0-μ)^2+ω1(μ1-μ)^2 (6)

将式(5)代入式(6),得到等价公式: g=ω0ω1(μ0-μ1)^2 (7)

采用遍历的方法得到使类间方差最大的阈值 T,即为所求。

Otsu 算法步骤如下：

设图象包含 L 个灰度级(0,1…,L-1)，灰度值为 i 的的象素点数为 Ni ，图象总的象素点数为 N=N0+N1+...+N(L-1)。灰度值为 i 的点的概为：

P(i) = N(i)/N.

门限 t 将整幅图象分为暗区 c1 和亮区 c2 两类，则类间方差 σ 是 t 的函数：

σ=a1*a2(u1-u2)^2 (2)式中，aj 为类 cj 的面积与图象总面积之比，a1 = sum(P(i)) i->t, a2 = 1-a1; uj 为类 cj 的均值，u1 = sum(i*P(i))/a1 0->t,

u2 = sum(i*P(i))/a2, t+1->L-1

该法选择最佳门限 t 使类间方差最大，即：令 Δu=u1-u2，σb = max{a1(t)*a2(t)Δu^2}

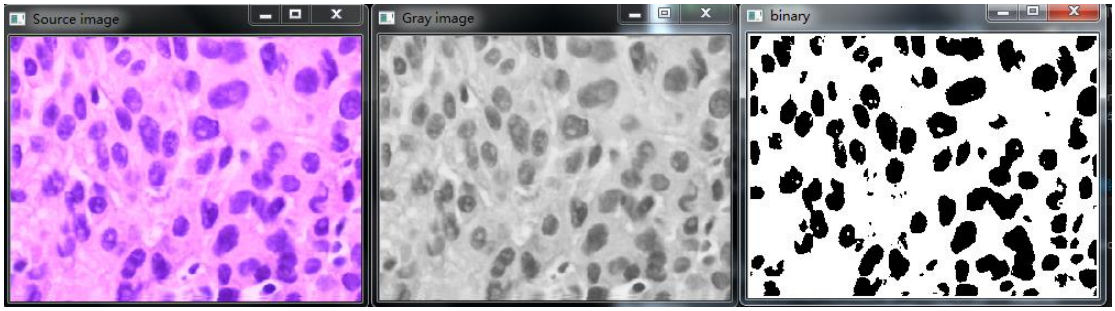
具体实现代码如下：

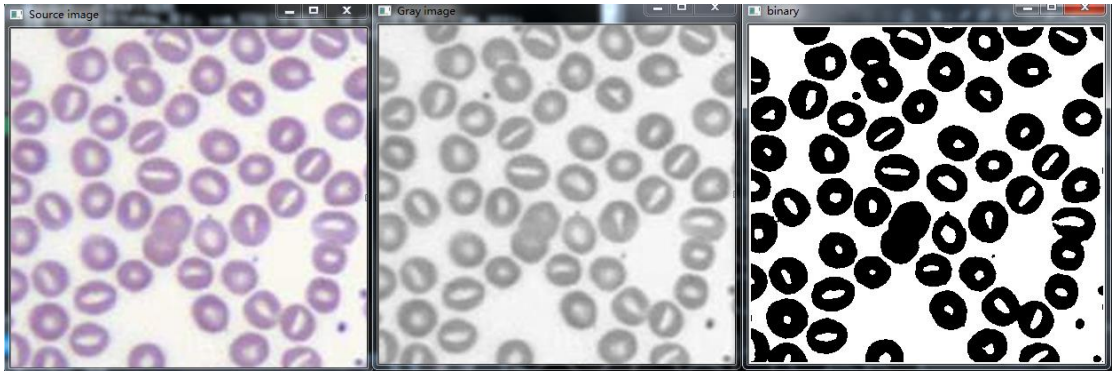
```
int Otsu(Mat& src)
{
   int height = src.rows;
   int width = src.cols;

   //histogram
   float histogram[256] = { 0 };
   for (int i = 0; i < src.rows; i++)
   for (int j = 0; j < src.cols; j++){
      histogram[src.at<uchar>(i, j)]++;
   }
   //normalize histogram
```

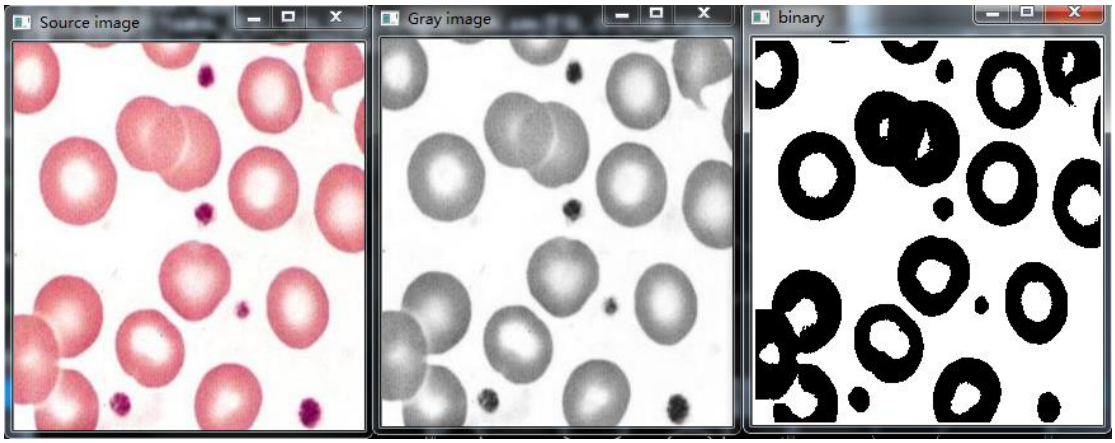
```
        int size = height * width;
        for (int i = 0; i < 256; i++)
        {
            histogram[i] = histogram[i] / size;
        }

        //average pixel value
        float avgValue = 0;
        for (int i = 0; i < 256; i++)
        {
            avgValue += i * histogram[i];  //整幅图像的平均灰度
        }

        int threshold;
        float maxVariance = 0;
        float w = 0, u = 0;
        for (int i = 0; i < 256; i++)
        {
            w += histogram[i];  //假设当前灰度 i 为阈值, 0~i 灰度的像素(假设像素值在此范围的像素叫做前景像素) 所占整幅图像的比例
            u += i * histogram[i];  // 灰度 i 之前的像素(0~i)的平均灰度值： 前景像素的平均灰度值

            float t = avgValue * w - u;
            float variance = t * t / (w * (1 - w));
            if (variance > maxVariance)
            {
                maxVariance = variance;
                threshold = i;
```

```
            }
        }

        return threshold;
    }
```

Otsu 算法二值化结果：

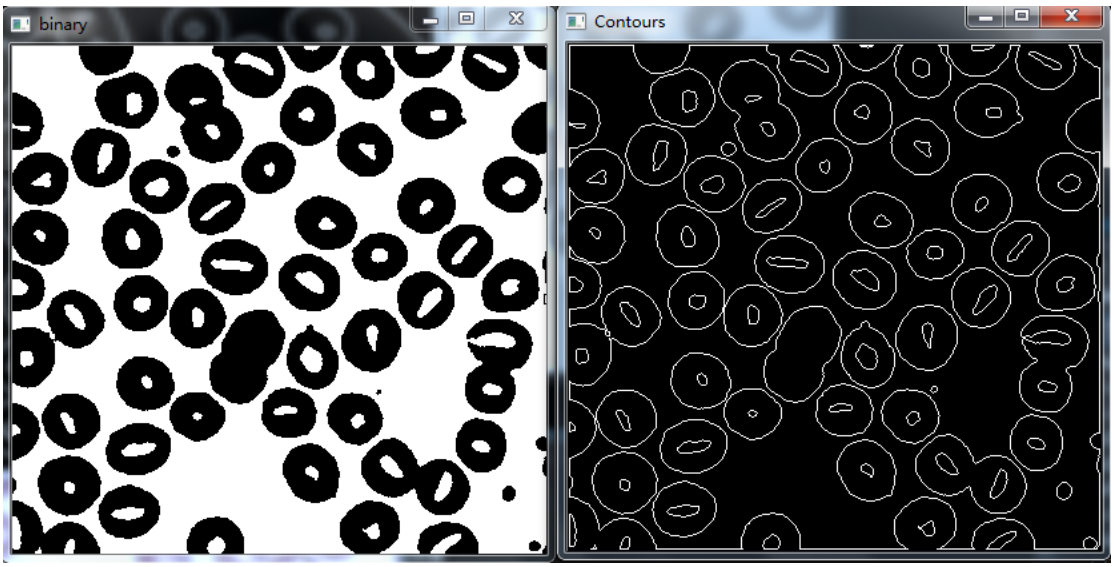






2. 细胞轮廓提取与分离

findContours()函数会提取图像中的所有轮廓，效果如下



此种情况下有两个问题：

1. 由于函数实现中将整幅图的边界像素值当做 0 来处理，因而会有一个包含了部分图像边界在内的极大轮廓，而且该轮廓并不是我们需要的。
2. 由于细胞内部相比于边界，颜色较浅且较为接近背景颜色，因而利用 Otsu 算法无法将整个细胞化为一个整体，所以在细胞内会存在“空洞”。

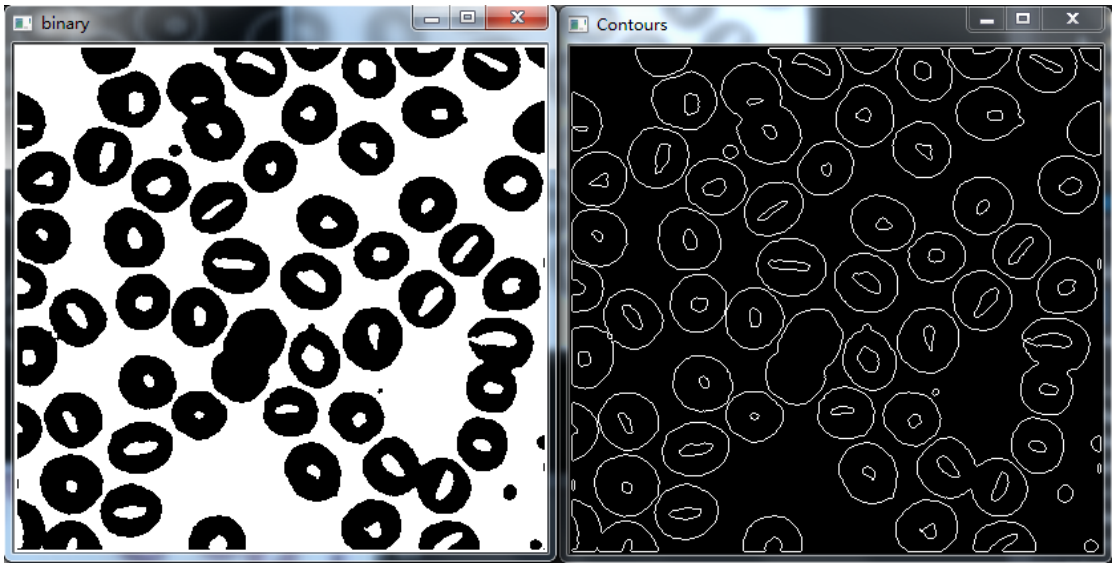
解决方案：

首先，在采用 findContours 函数的时候，制定 MODE 为 CV_RETR_TREE，这是将所有的轮廓根据包含关系，建成一个树状结构，若大的轮廓包含了小的，则小的轮廓为大的轮廓的儿子节点，且该树可以有多层。

而根据测试，当我们将整幅图像的边界以内宽度为 3 的边框的像素赋为 255 的时候，就能够实现一个覆盖整幅图的边框，并将穿越图像边界的细胞（可能为细胞的一部分）完全分离出来。

```
for (int i = 0; i < bin.rows; i++)
   {
      for (int j = 1; j < 3; j++){
         bin.at<uchar>(i, j - 1) = 255;
         bin.at<uchar>(i, bin.cols - j) = 255;
```

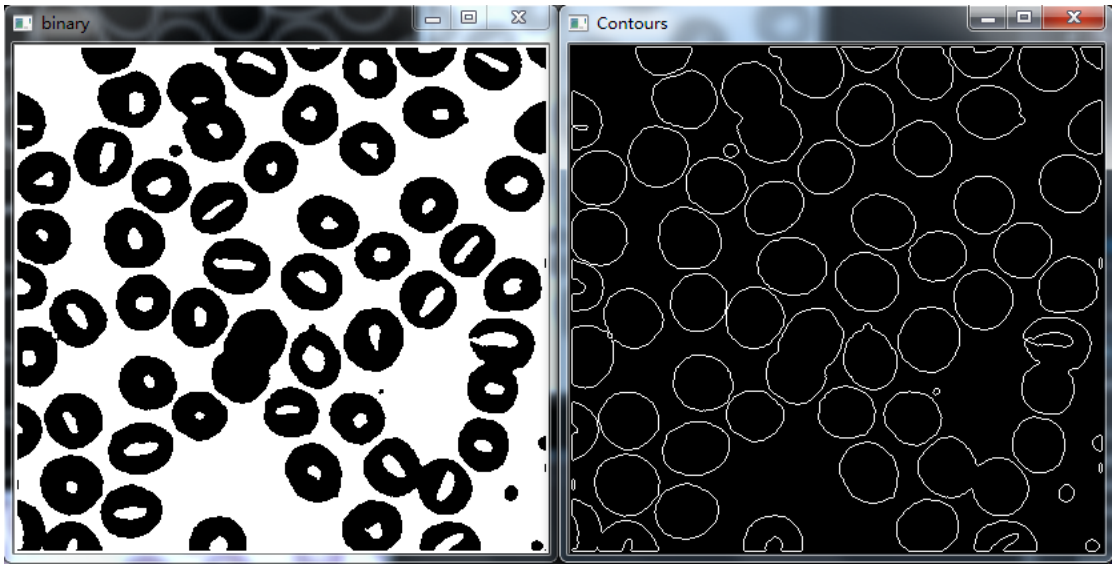
```
        }
        }

        for (int i = 0; i < bin.cols; i++)
        {
            for (int j = 1; j < 3; j++){
                bin.at<uchar>(j - 1, i) = 255;
                bin.at<uchar>(bin.rows - j, i) = 255;
            }

    }
```



于是得到如下算法①将外层轮廓删除②所有父节点是最外层轮廓的（即为细胞最外层）保留，其余轮廓删除。

```
    for (int i = 0; i < contours.size(); i++)
        if ((hierarchy[i][3] == -1) || (hierarchy[i][3] != 0)){
            hash[i] = false;
            counter --;
        }
```

结果如下：



3. 去除杂质

在观察图像特质之后我们可以得到杂质如下特征：①颜色和细胞颜色较为接近，二值化后为黑色。②面积远小于细胞。

方法一：统计学处理

根据分析很容易得出如下结论，细胞的面积基本相差不大，服从 N(μ, σ)的正态分布，其中 μ 为面积均值， σ 为标准差。根据三倍标准差检验，即分布在[μ -3 σ , μ +3 σ]外的数据为不符合要求数据。

存在问题：

1. 没有总体数据

   在没有总体的情况下，如果强行算出所有数据的均值和标准差，会发现混入了杂质后，细胞的均值下降，方差大幅度增加，导致了杂质基本都存在于[μ -3 σ , μ +3 σ]区间内。

2. 无法迭代操作

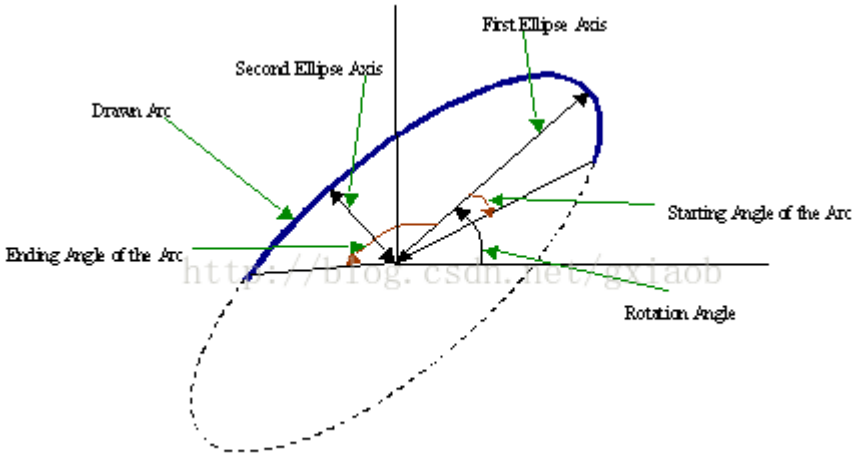
   如果采用将符合要求的数据逐个插入总体中，进行迭代，则涉及到最刚开始的数据必须是符合要求的细胞的数据，否则如果一开始都是杂质，那么以杂质建立总体样本，就会出现细胞变成了不符合要求的数据，并且被踢出。

方法二：

法二较为简单，利用杂质面积远小于细胞的特性，设置一个参数，为杂质和细胞均值的面积比例，可以通过实验得出，当比例小于 0.3 的时候就可以很好的区分出细胞和杂质了。但是这一种方法的误差依然存在。

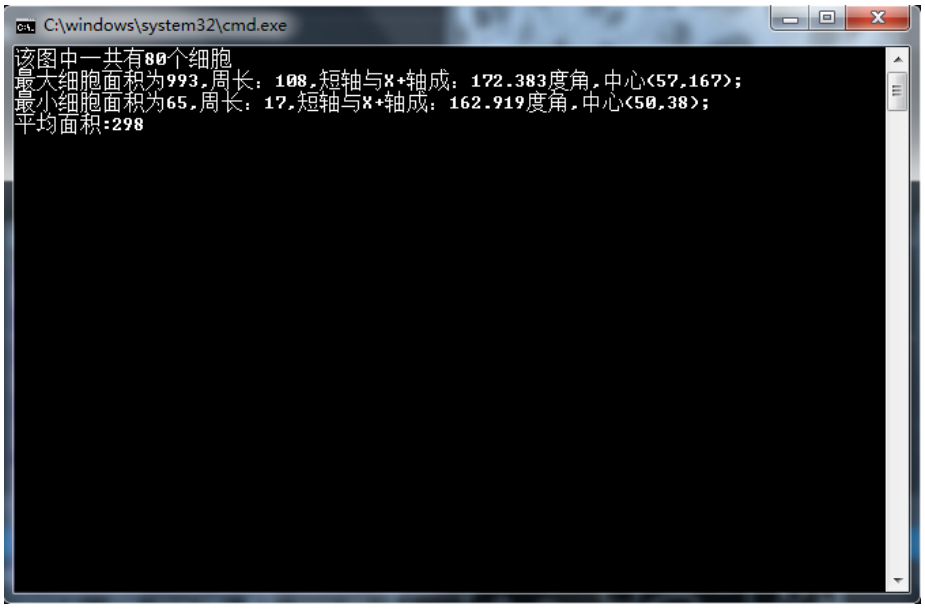
4. 椭圆拟合

椭圆拟合较为简单，注意角度的表示，如下图：

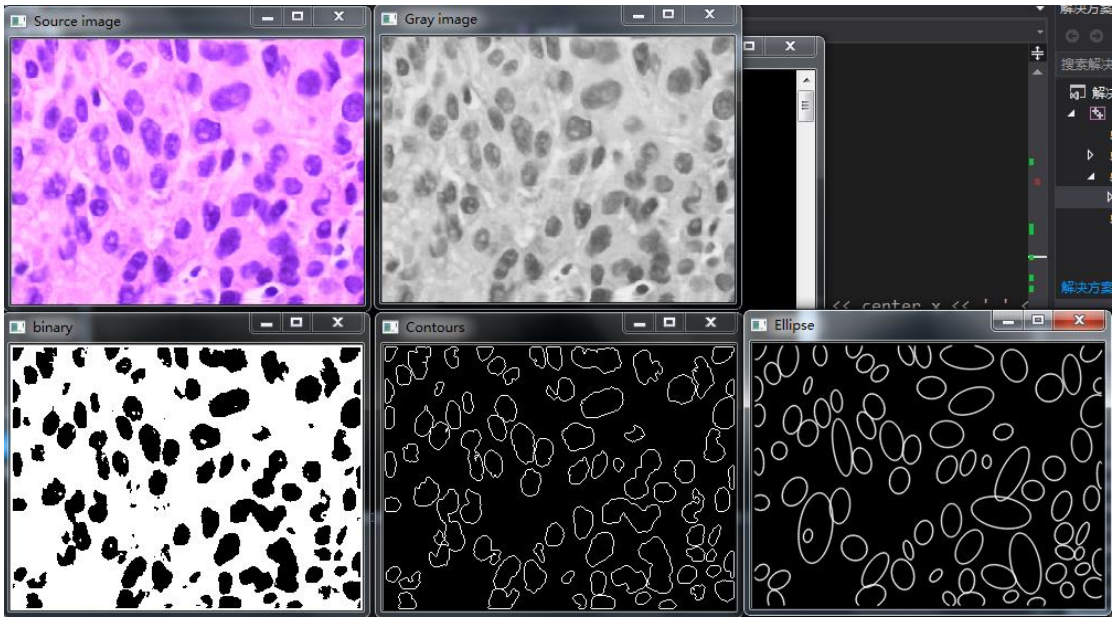


实验结果展示

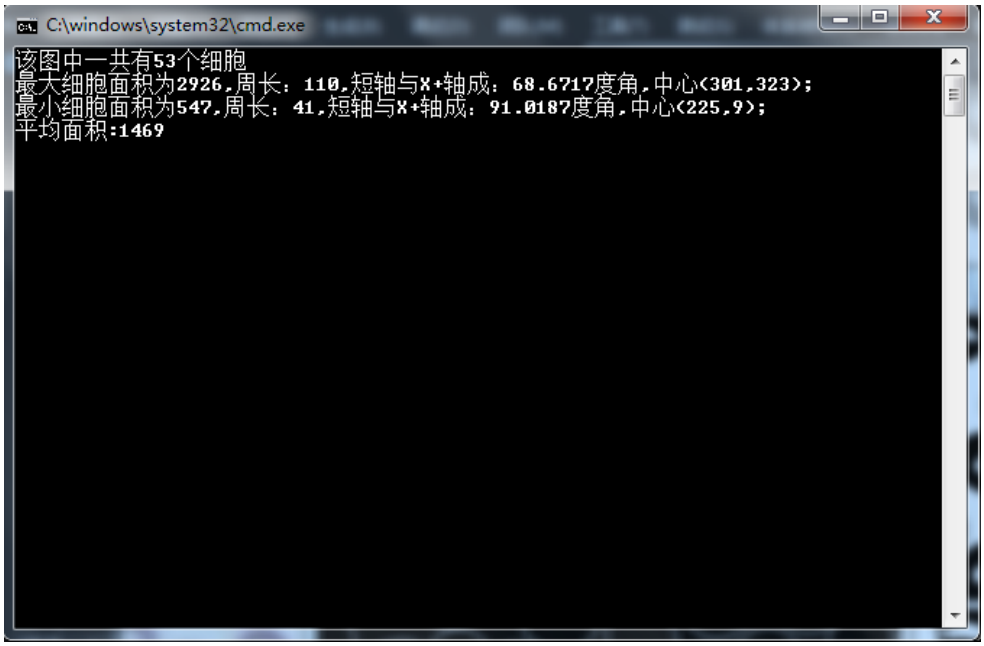
1. cell1.bmp



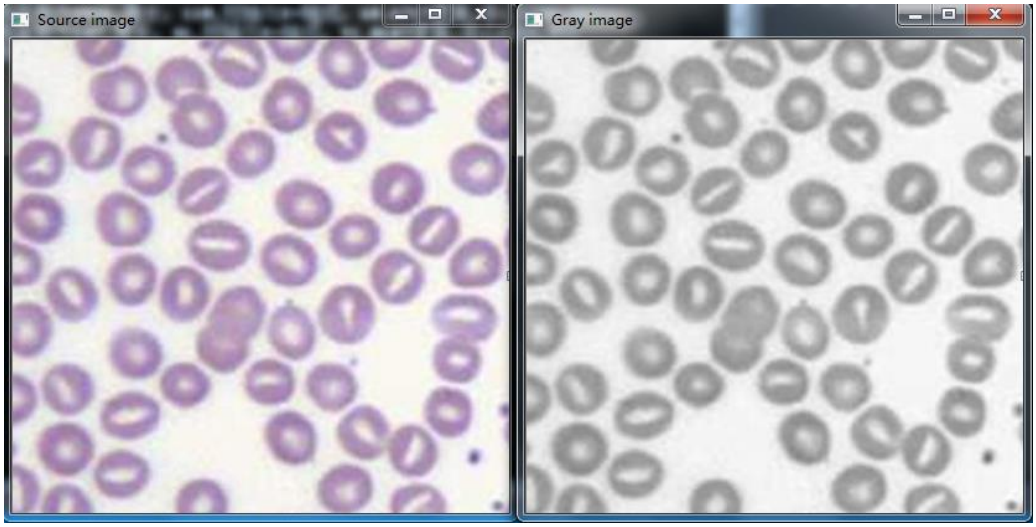
中间步骤结果：

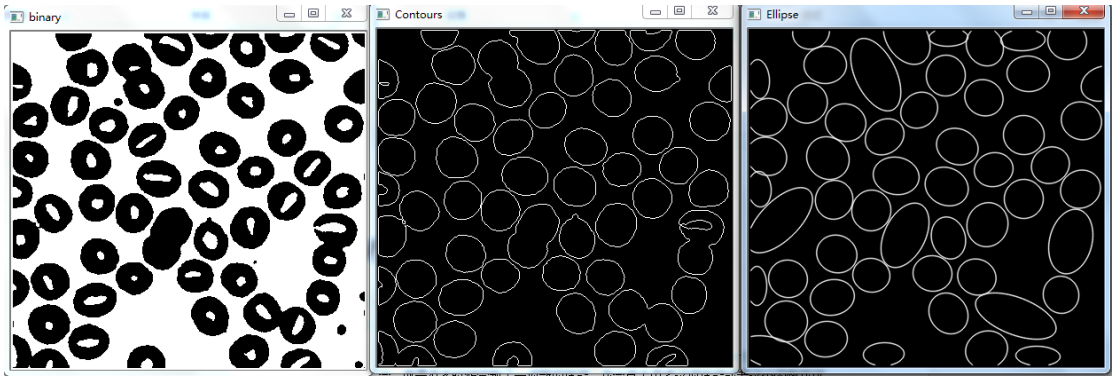


2. cell2.jpg

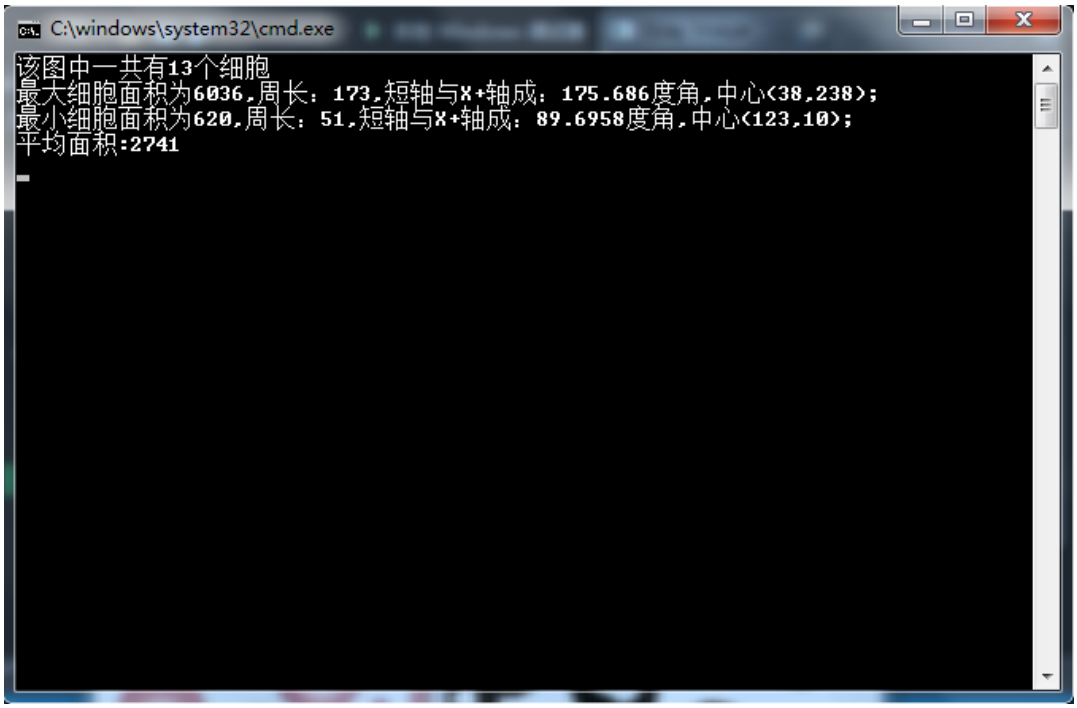


中间步骤结果：

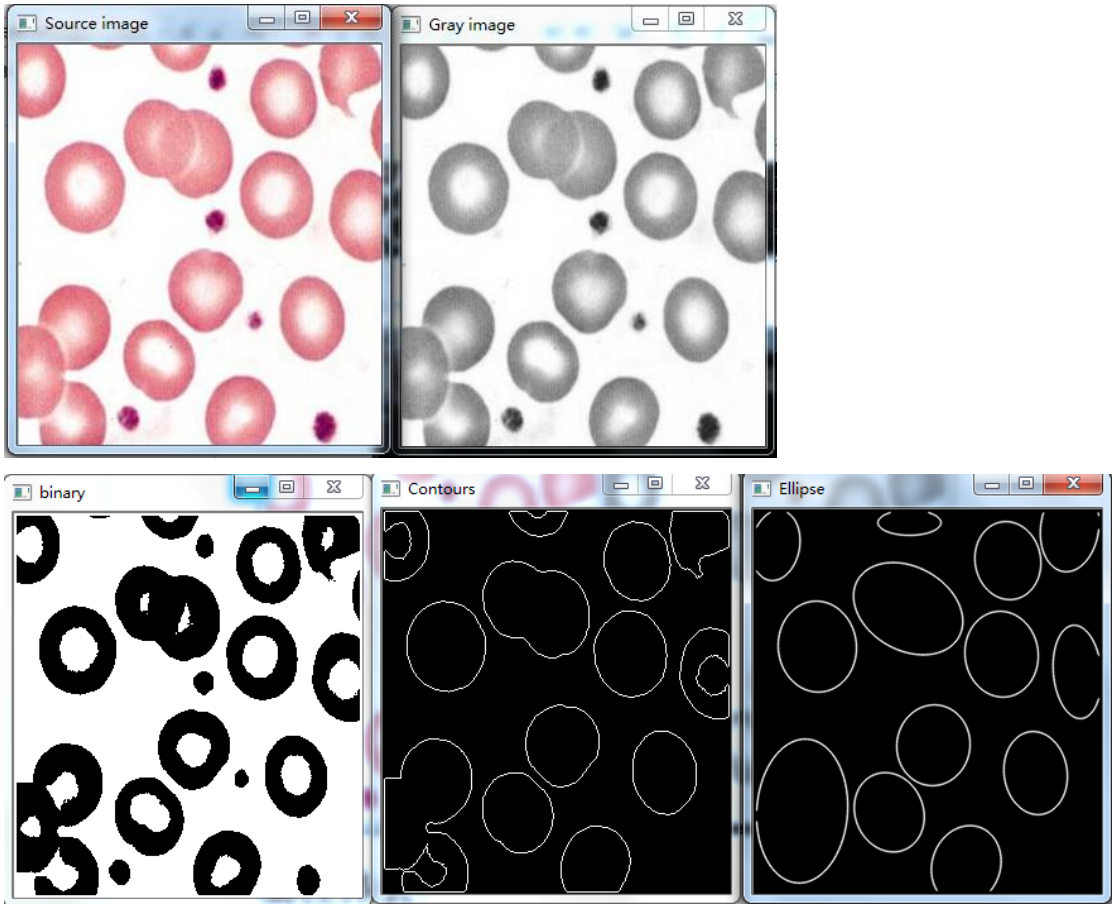




3. cell3.jpg



中间步骤结果：



编程体会

1. **要善于查阅资料文献以及学习他人经验**

   在这次作业过程中我开始感到毫无头绪，但是后来在网上查阅了一些相关的文献之后我得到了算法的大体步骤，而一些具体的做法，包括每一个 OpenCV 函数怎么使用，要得益于很多人在网络上的博客的分享。所以在我看来善于利用这些知识十分重要

2. **学会利用所学知识，并且有所创新**

   在处理杂质的时候，我灵光一现想起了“概率论与数理统计”课当中所学的知识，希望能使用统计学知识来解决，虽然最后结果没有实现，但是在提取外层轮廓等步骤中我都采用的自己思考所得到或者优化出的算法，因此我觉得很有成就感而且学到的东西也很多