计算机视觉与模式识别期末项目

13331321 殷明旺

一：项目实验环境

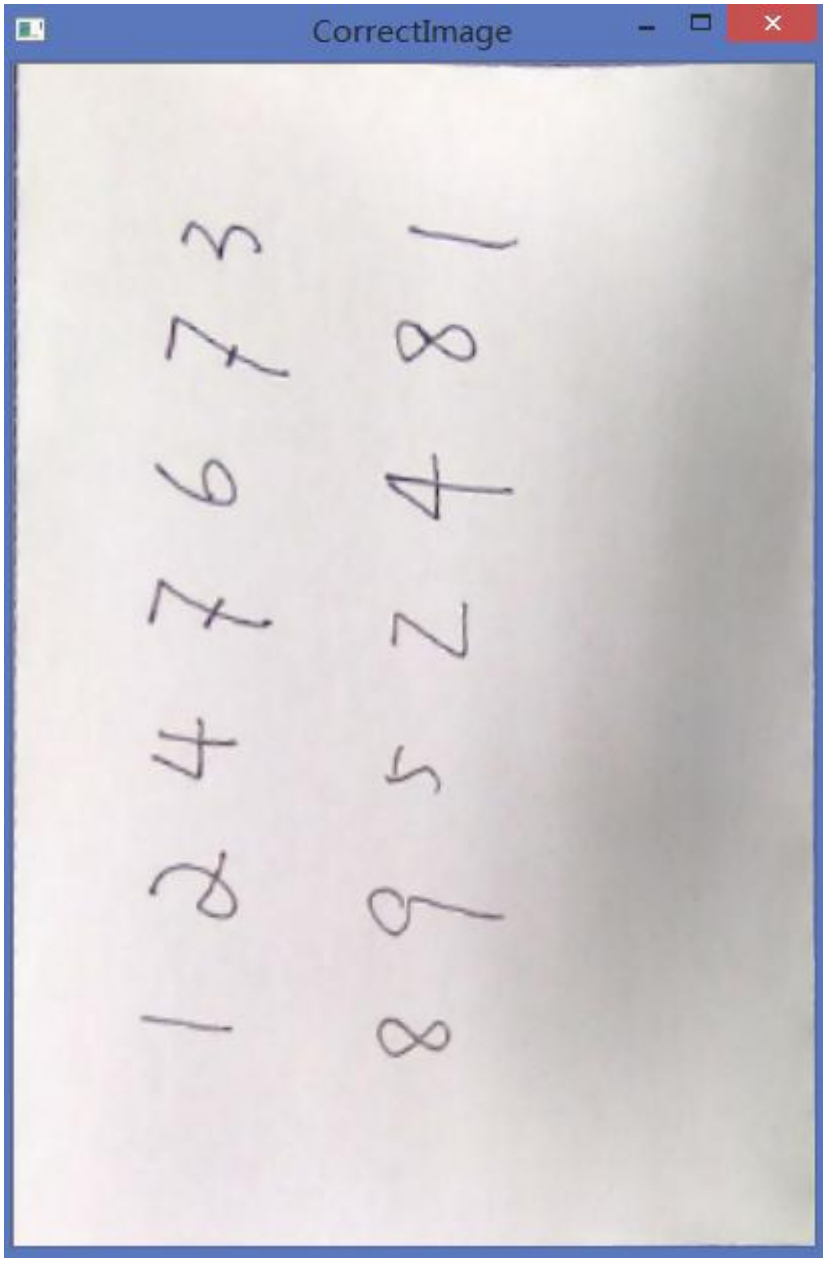
WIN8.1+VS2013+Opencv2.7

二：项目要求

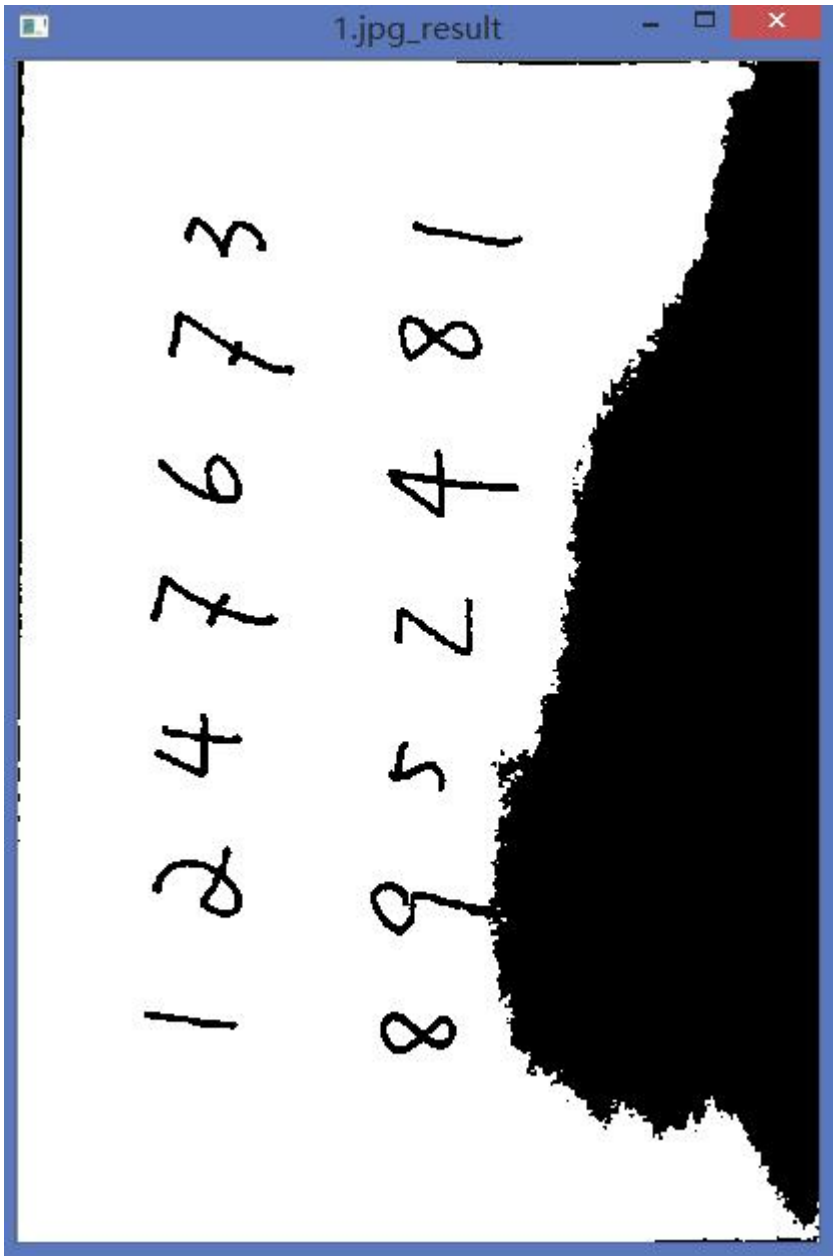
1. 校正图像为标准的 A4 图；
2. 用 Adaboost 或者 SVM 训练一个手写体数字的分类器(作业 6 的代码)；
3. 切割字符；
4. 识别并输出数字。

三：实现步骤

1.根据以前的作业三的代码来对图像进行矫正，可能要调节一下线段距离的参数来消除多余的线条，因为这图片会检测出 5 条直线，有一条是纸与桌相间的直线



2. 对图像进行前后景分离，最开始我使用了之前作业里边写的 OTSU 算法来分离前后景，但是有一个问题，图片上的光照是不均匀的，这样就导致 OTSU 在求阈值的时候会出现问题使用 OTSU 来分离得到结果如下图：



所以这里我使用了 opencv 自带的自适应阈值分割函数 adaptiveThreshold

adaptiveThreshold(InputArray src, OutputArray dst, double maxValue, int adaptiveMethod, int thresholdType, int blockSize, double C)

InputArray src：源图像

OutputArray dst：输出图像，与源图像大小一致

int adaptiveMethod：在一个邻域内计算阈值所采用的算法，有两个取值，分别为 ADAPTIVE_THRESH_MEAN_C 和 ADAPTIVE_THRESH_GAUSSIAN_C 。

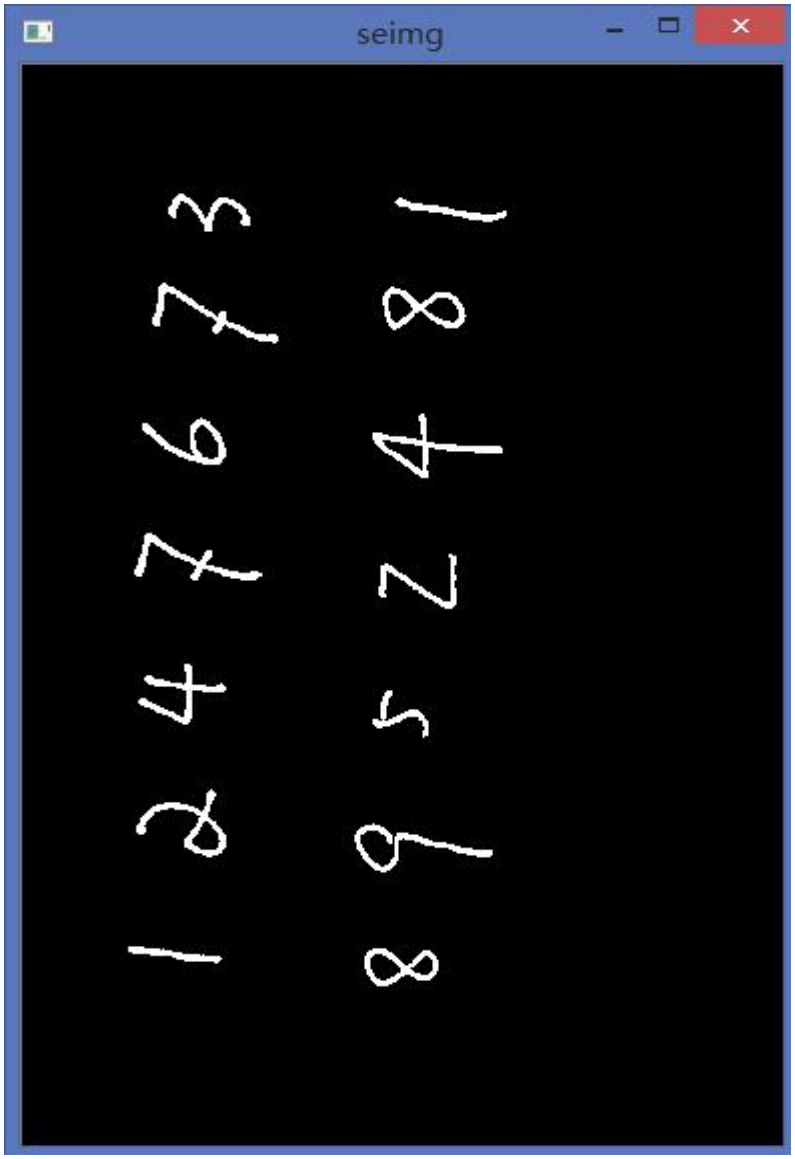
ADAPTIVE_THRESH_MEAN_C 的计算方法是计算出领域的平均值再减去第七个参数 double C 的值

ADAPTIVE_THRESH_GAUSSIAN_C 的计算方法是计算出领域的高斯均值再减去第七个参数

double C 的值
int thresholdType：这是阈值类型，只有两个取值，分别为 THRESH_BINARY 和 THRESH_BINARY_INV 具体的请看官方的说明，这里不多做解释
int blockSize：adaptiveThreshold 的计算单位是像素的邻域块，邻域块取多大，就由这个值作决定
double C：在对参数 int adaptiveMethod 的说明中，我已经说了这个参数的作用，从中可以看出，这个参数实际上是一个偏移值调整量
为了使分离效果更好，这里 blocksize 我取了 33

```
Mat segmentation(Mat srcimg) {
    Mat grayimg;
    Mat seimg;
    cvtColor(srcimg, grayimg, CV_BGR2GRAY);
    adaptiveThreshold(grayimg, seimg, 255, CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY_INV, 33, 5);
    //imshow("seimg", seimg);
    return seimg;
}
```

前后景分离后的结果如下：



3.对前后景分离好的图像做一些处理，便于识别和切割，这里先做旋转，然后再做形态学的膨胀和腐蚀

腐蚀：

```
Mat Erosion(Mat srcimg,int size) {
    Mat erodeimg;
    Mat element = getStructuringElement(MORPH_RECT, Size(size, size));
    /// 腐蚀操作
    erode(srcimg, erodeimg, element);
    //imshow("Erosion Demo", erodeimg);
    return erodeimg;
}
```

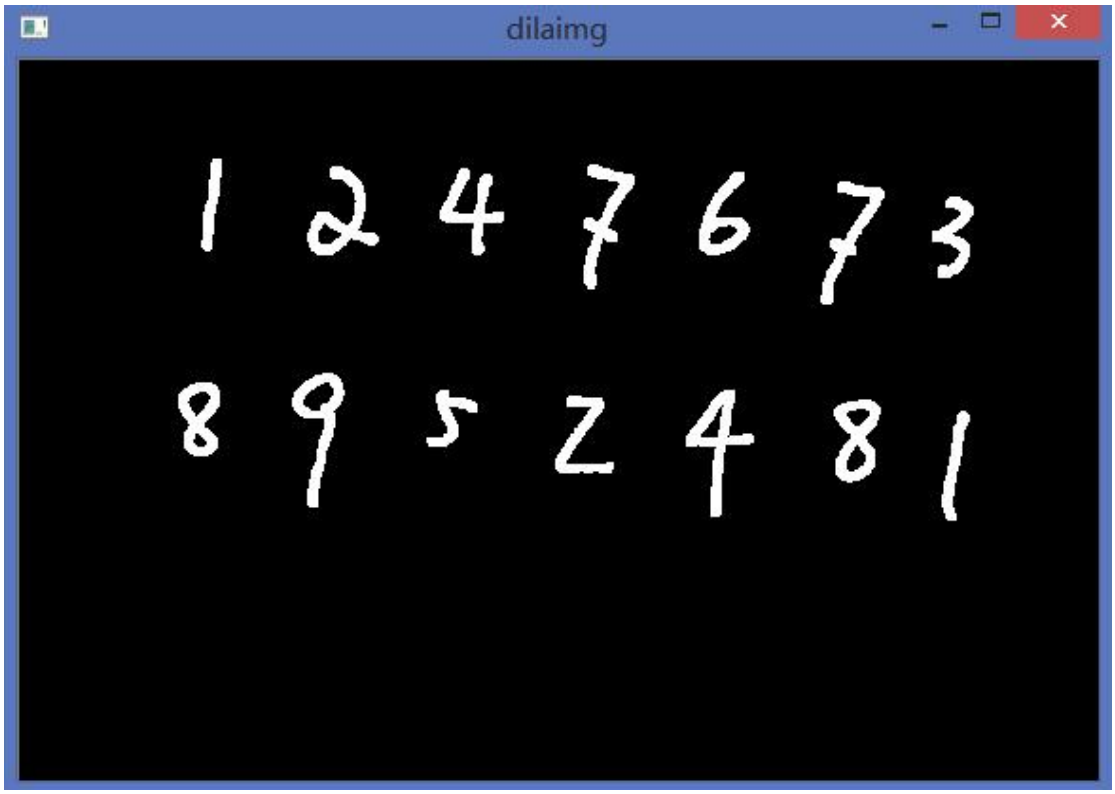
膨胀：

```
Mat Dilation(Mat srcimg,int size)
{
    Mat dilatimg;
    Mat element = getStructuringElement(MORPH_RECT, Size(size, size));
    /// 膨胀操作
    dilate(srcimg, dilatimg, element);
    //imshow("Dilation Demo", dilatimg);
    return dilatimg;
}
```

注：这里是通过改变 size 的大小来改变膨胀和腐蚀的程度

图片 2 和图片 3 为了查找轮廓和切割，要用到腐蚀

这里对图片 1 进行膨胀，膨胀的滤波器的矩阵大小为 3x3,结果如下



4 对上边图像进行切割

首先得找出每个数字的轮廓，然后调整参数去掉一些太小的轮廓
使用了 void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())API 来查找轮廓
定义最小和最大的轮廓值来去掉一些不需要的轮廓

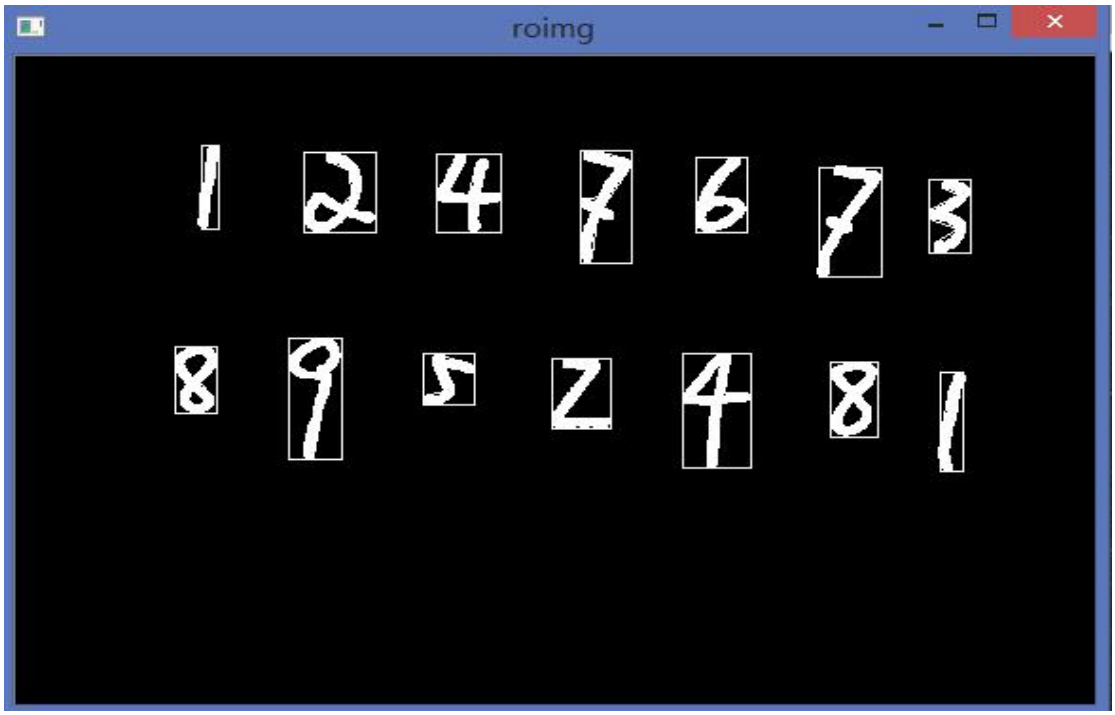
```
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
findContours(threshold_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
vector<vector<Point>>::const_iterator itc = contours.begin();
while (itc != contours.end()) {

    if (itc->size() < cmin || itc->size() > cmax)
        itc = contours.erase(itc);
    else
        ++itc;
}
```

然后根据求得的轮廓把数字用矩形框出来，并把数字用矩形切出来
使用函数 drawContour()和 rectangle 来框出轮廓，使用 Rect 切出数字，把大小变为 28X28 便于识别的时候使用，再把图片信息写到.csv 文件里边

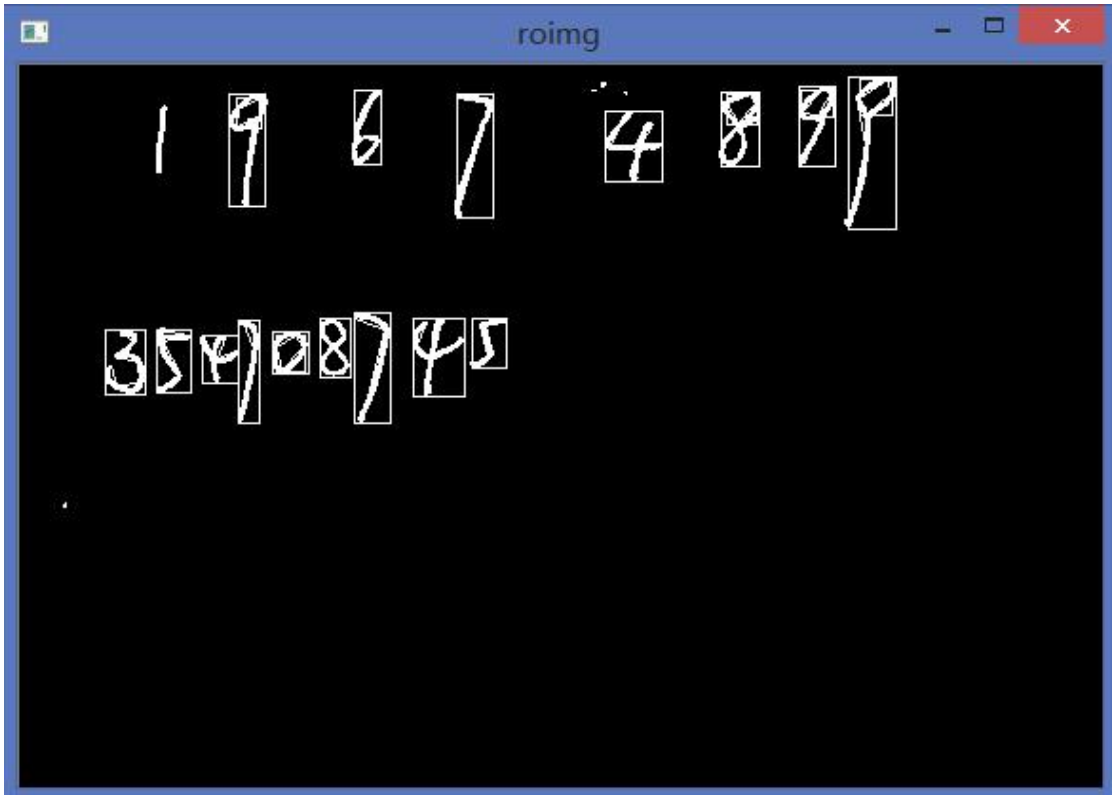
```
Mat drawing = srcimg.clone();
Mat src = srcimg.clone();
vector<Mat> roi(contours.size());
for (int i = 0; i < contours.size(); i++)
{
    stringstream stream;
    string str;
    stream << i;
    stream >> str;
    Scalar color = Scalar(255, 255, 255);
    drawContours(drawing, contours_poly, i, color, 1, 8, vector<Vec4i>(), 0, Point());
    rectangle(drawing, boundRect[i].tl(), boundRect[i].br(), color, 1, 8, 0);
    Mat temp(src, Rect(boundRect[i]));
    Mat img;
    resize(temp, img, Size(28,28), 0, 0, CV_INTER_LINEAR);
```

结果如下：
图片 1：

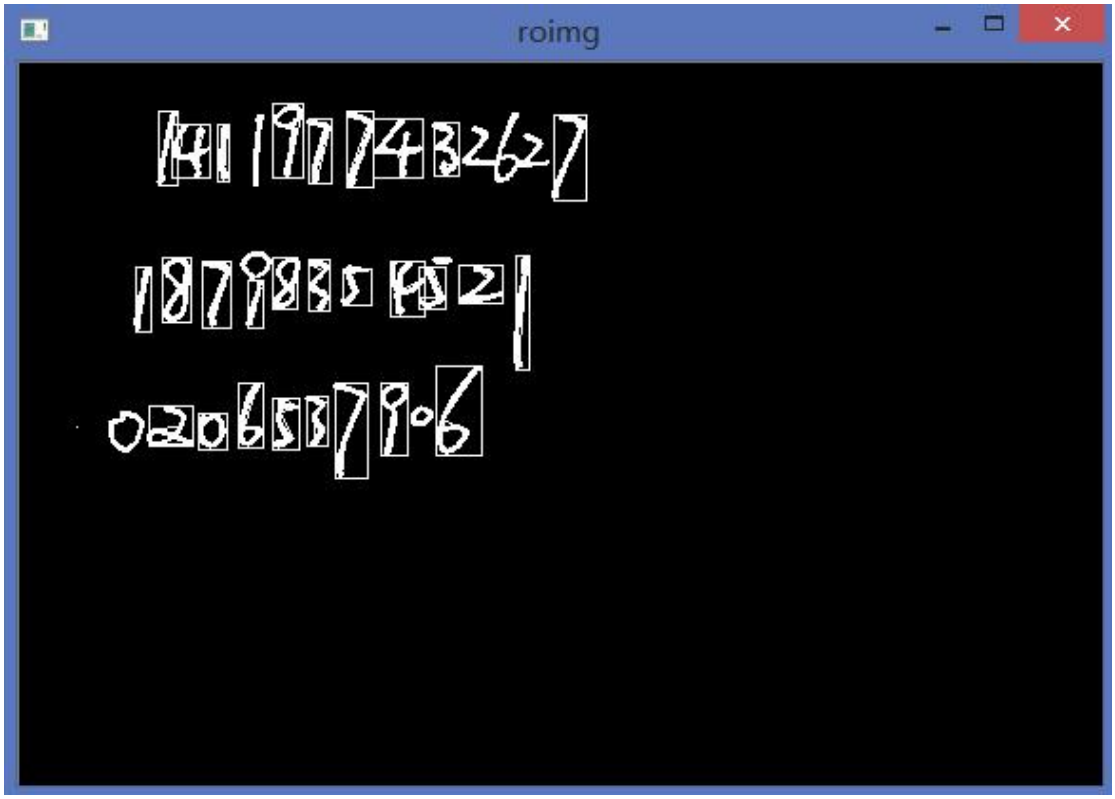


切出的图片如下：

图片二：
先腐蚀切割后再膨胀的



图片三：
也是先腐蚀再膨胀的



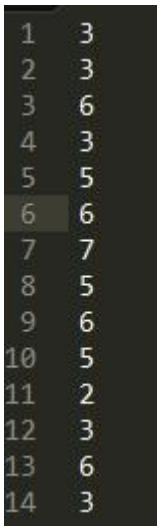
切割出来的图像以及写到.csv 文件里边用于识别，文件和对应得识别文件也一起打包交上来了，具体请看 Img1.csv，Img1Result.csv 等文件

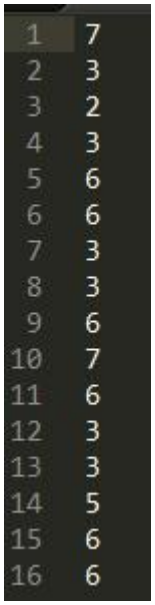
5 识别

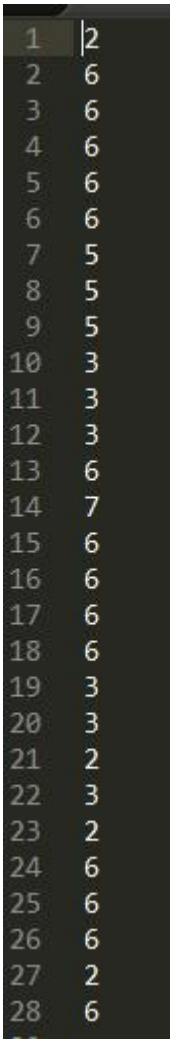
这里利用前一次作业里边训练好的模型来经行识别

部分识别结果如下：准确率很低，可能是把图片变为 28x28 大小的缘故

img1 img2 img3


1 3
2 3
3 6
4 3
5 5
6 6
7 7
8 5
9 6
10 5
11 2
12 3
13 6
14 3


1 7
2 3
3 2
4 3
5 6
6 6
7 3
8 3
9 6
10 7
11 6
12 3
13 3
14 5
15 6
16 6


1 2
2 6
3 6
4 6
5 6
6 6
7 5
8 5
9 5
10 3
11 3
12 3
13 6
14 7
15 6
16 6
17 6
18 6
19 3
20 3
21 2
22 3
23 2
24 6
25 6
26 6
27 2
28 6