作业 5

任务一： 在作业 4 第二步完成的基础上，切割出红绿蓝这些区域内的数字(分别用不同颜色框 标记)，并识别这些数字(可以用 OpenCV 已经训练好的模型).

上个实验已经实现了识别标尺和下面的数字，这里主要加入了实现上面和下面的数字。

1. 实现识别顶部数字

识别顶部的数字主要思路是先找到上面数字的位置，注意到上面的数字基本都在上面 1/4 的位置，而且顶部数字的排列比较整齐，有相近的纵坐标。所以可以用纵坐标来定位数字的位置。具体实现如下：

1) 先把在图片上面 1/4 位置的连通块的上下边界纵坐标找到，可以用之前实现的 get_diagonal 函数，获得的是左上和右下的点的坐标。代码如下:

```
vector<int>top;
vector<int>bottom;
for (int i = 0; i < link_area.size(); i++)//把连通块的上边界都找到，然后排序，找到上边界剧烈变化的下标把 vector 分割成几个部分，每个部分基本在一行，找到一行中元素个数最多的一行即为上面的数字区域
{
    top.push_back(get_diagonal(link_area[i])[0].y);//连通块的左上角的 y 值，即连通块的上边界
    bottom.push_back(get_diagonal(link_area[i])[1].y);//连通块的右下角的 y 值，即连通块的下边界
}
```

找到上边界后将上边界排序，即可把有相同上边界的连通块的纵坐标分在一起。然后找到纵坐标变化较为剧烈的位置，这个位置表示换行。对

下界也进行同样的操作：

```
sort(top.begin(),top.end());//从小到大排序
sort(bottom.begin(), bottom.end());
vector<int>top_indexs;
vector<int>bottom_indexs;
top_indexs.push_back(0);
bottom_indexs.push_back(0);
for (int i = 1; i < top.size(); i++)
{
    if (top[i] > Src._height*0.25)//上面的数字一般在图片的顶端，这里设定为1/4图片高度
    {
        top_indexs.push_back(i);
        break;
    }
    if (top[i] - top[i - 1] > 5)//找到高度剧烈变化的位置
        top_indexs.push_back(i);
}
for (int i = 1; i < bottom.size(); i++)
{
    if (bottom[i] > Src._height*0.25)//上面的数字一般在图片的顶端，这里设定为1/4图片高度
    {
        bottom_indexs.push_back(i);
        break;
    }
    if (bottom[i] - bottom[i - 1] > 4)//找到高度剧烈变化的位置
        bottom_indexs.push_back(i);
}
```

找到每个剧烈变化的位置后计算两行之间的间隔，这个间隔表示有多少个元素位于同一行。多于一定数量的元素位于同一行可以认为是数字行。另外由于数字是竖着排列，所以可能有多个数字行，我们的上界找到第一个数字行即可停止，而下界要找完数字行，这样才能把数字行整个框起来，代码如下：

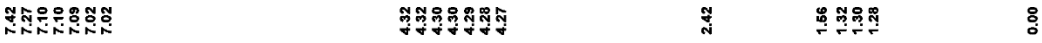
```
int max = 0;
int max_index = 0;
```

```
for (int i = 1; i < top_indexs.size(); i++)
{
    if (top_indexs[i] - top_indexs[i - 1] > max)
    {
        max = top_indexs[i] - top_indexs[i - 1];
        max_index = top_indexs[i - 1];//第 i-1 行是最多的那一行
        if (max > 10)//因为数字是竖着排列，所以可能有很多连通块的数量都是最大的，找到最上面的连通块的位置即可
            break;
    }
}
int top_ = get_diagonal(link_area[max_index])[0].y;
for (int i = 1; i < bottom_indexs.size(); i++)
{
    if (bottom_indexs[i] - bottom_indexs[i - 1] > 10)
    {
        max_index = bottom_indexs[i - 1];//第 i-1 行是最多的那一行
    }
}
int bottom_ = get_diagonal(link_area[max_index])[1].y;
```

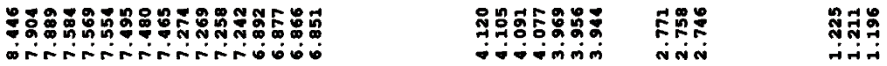
这样我们就可以把数字行单独拿出来，测试结果如下：

H-Img0

H-Img1



H-Img2



然后用上次作业实现的 Number_Detect 去识别，由于上次是单个数字进行识别，这次是要识别一个完整的小数，而且数字是竖着放的，所以要对之前的识别代码进行更改。

首先还是对数字进行分割。首先进行上下分割，与之前的分割方法不同的是这次分割是从上到下遍历到像素和不为 0 的位置就停止，然后从下到上再进行遍历，这样就找到了一行完整的数字，代码如下：

```
void cut_top(Mat& src, Mat& dstImg)
{
    int top, bottom;
    top = 0;
    bottom = 0;
    for (int i = 0; i < src.rows; i++)
    {
        int colValue = getRowSum(src, i);//统计所有行像素的总和
        if (colValue > 0)//扫描直到行像素的总和大于 0 时，记下当前位置 top
        {
            top = i;
            break;
        }
    }
    for (int i = src.rows - 1; i >= 0; i--)
    {
        int colValue = getRowSum(src, i);//统计所有行像素的总和
        if (colValue > 0)//扫描直到行像素的总和大于 0 时，记下当前位置 top
        {
            bottom = i;
            break;
        }
    }
    int height = bottom - top;
    Rect rect(0, top, src.cols, height);
    dstImg = src(rect).clone();
}
```

然后再对每组数字进行左右分割得到一个 Mat 类型的 vector，接着对 vector 里面的每个图根据需要进行反转（因为有些数字是正着的不用反转），代码如下：

```
vector<Mat> Number_Detect::cutImg(Mat & src,bool top_bottom)//0 for top number,1 for bottom number
{

    vector<Mat>imgs;
    Mat temp;
    cut_top(src, temp);
    int left, right;
    left = 0;
    right = 0;
```

```
    //namedWindow("Image");
    //imshow("Image", temp);//显示分割后的图片
    //waitKey(0);
    //destroyWindow("Image");
    Mat grayImage;
    cvtColor(temp, grayImage, COLOR_BGR2GRAY);
    threshold(grayImage, grayImage, 100, 255, 1);

    while (1)
    {
        int i;
        for (i = right; i < grayImage.cols; i++)
        {
            int colValue = getColSum(grayImage, i);//统计所有列像素的总和
            if (colValue > 0)//扫描直到列像素的总和大于 0 时，记下当前位置 left
            {
                left = i;
                break;
            }
        }
        if (i == grayImage.cols)
            break;
        //继续扫描
        for (; i < grayImage.cols; i++)
        {
            int colValue = getColSum(grayImage, i);
            if (colValue == 0)//继续扫描直到列像素的总和重新等于 0 时，记下当前位置 right
            {
                right = i;
                break;
            }
            right = grayImage.cols;
        }
        int width = right - left;//分割图片的宽度则为 right - left
        Rect rect(left, 0, width, temp.rows);//构造一个矩形，参数分别为矩形左边顶部的 X 坐标、Y 坐标，右边底部的 X 坐标、Y 坐标（左上角坐标为 0，0）
        Mat img = temp(rect).clone();
        if (!top_bottom)//if top number
        {
            Mat t, dst;
            transpose(img, t);
            flip(t, dst, 1);
            imgs.push_back(dst);
```

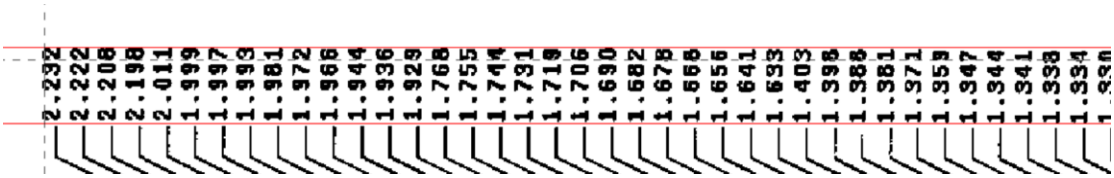
```
        }
        else
            imgs.push_back(img);
        //namedWindow("Image");
        //imshow("Image",img);//显示分割后的图片
        //waitKey(0);
        //destroyWindow("Image");
    }
    return imgs;
```

然后对 img 里面的每张图片进行识别即可：

```
void Number_Detect::getTopNumber()
{
    vector<Mat>number = cutImg(Src,0);
    for (int i = 0; i < number.size(); i++)
    {
        //imshow("temp", number[i]);
        //waitKey();
        get_Number(number[i]);
        cout << endl;
    }
}
```

识别结果如下：

框出上面的数字（以 H-Img0 为例）：



识别结果：

H-Img0

(这里由于数字有点多一个控制台窗口放不下)

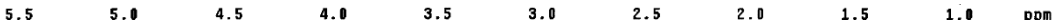
H-Img1

H-Img2

2. 标尺数字识别

我在上次的代码基础上稍微进行了修改，思路是标尺这样的大型连通块一般都在图片的下部，而我找连通块的方式是从上到下，所以遍历大型连通块，得到的最后一个就是标尺。对三张图片进行实验，得到的结果如下：

H-Img0:



H-Img1

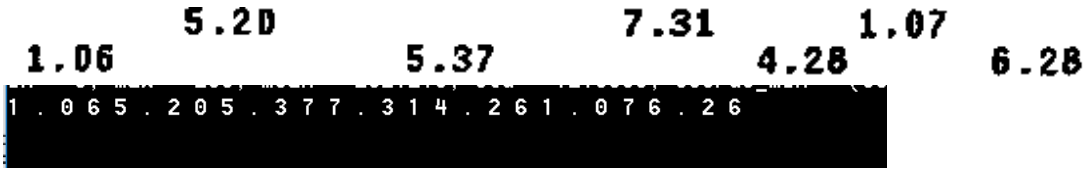
H-Img2

找到数字之后可以按照上面的方法进行识别，识别结果如下（这里只展示 H-Img0）：

3. 识别下面的图片

识别下面的数字的流程跟识别上面的基本类似，但是下面的数字比较难定位，能做的基本上就是把四周位置都限制好再找，但是这样显得很针对，而且对

图片 H-Img2 来说数字和旁边的括号连在了一起成了一个连通块，这样识别起来就很困难。我在任务二中用了另一种方法进行这些数字的识别，这里只放用上面的方法完成的第一张图的识别结果：



任务二：分割并识别水平括号(包括水平括号下边的数字和左右两端刻度值):

1. 先分割出水平括号，并计算出左右两端在标尺刻度中对应的值.(例如上图中最左边的水平括号，左右两边对应的刻度假设为 5.60 和 5.50)

2. 识别水平括号下边的数字(例如上图中最左边的水平括号下边的数值为 1.00).

3. 输出所有的水平括号的特征，即每个水平括号输出三个值：1.00, 5.60, 5.50(水平括号下边的数字，水平括号左边刻度，水平括号右边刻度).

由于括号有三种类型，所以在 Link_area::get_blocks 函数中分了三种情况进行处理，通过传入参数的方式进行选择。

1. 首先要找到括号，第一张图和第二张图中，括号的长宽比比较大，可以根据这个结合连通块大小找到所有括号，代码如下：

```
1.  vector<vector<Area_point>> blocks;
2.  vector<vector<int>> scales;
3.
4.  for (int i = 0; i < link_area.size(); i++)
```

```
{
    int width = get_diagonal(link_area[i])[1].x - get_diagonal(link_area[i])[0].x;
    int height = get_diagonal(link_area[i])[1].y - get_diagonal(link_area[i])[0].y;
    int ratio = width / height;//计算长宽比，长宽比大的是括号
    if (type == 1)
    {
        if (ratio >= threshold && link_area[i].size() > 120 && link_area[i].size() < 2000)
        {
            blocks.push_back(link_area[i]);
        }
    }
```

而第二种括号是长宽比比较小，所以要改一下筛选条件：

```
if (type == 2)
{
    if(ratio<=threshold&& link_area[i].size() > 550 && link_area[i].size() < 2000)
        blocks.push_back(link_area[i]);
}
```

找到括号后接下来要找到括号的刻度。由于找连通块的顺序是左上到右下，而且 type1 和 type3 的刻度起点都在两边，所以 type1 和 type3 的连通块纵坐标可以用连通块中第一个元素的纵坐标进行标定。找到第一个刻度之后再遍历连通块剩下的部分，找到纵坐标与起点相近但横坐标相差较大（刻度是有宽度的，避免宽度影响刻度的数量）的点的横坐标即可，代码如下：

```
for (int i = 0; i < blocks.size(); i++)
{
    vector<int>temp;
    base = blocks[i][0];//因为遍历是从左上到右下进行的，所以刻度的基准为连通块第一个点
    temp.push_back(blocks[i][0].x);//先把基准的横坐标存着
    for (int j = 1; j < blocks[i].size(); j++)
    {
```

```
8.          if (blocks[i][j].y - base.y <= 2  && blocks[i][j].x && blocks[i][j].
    x - temp[temp.size()-1] > 5)
9.          {
10.             temp.push_back(blocks[i][j].x);//如果连通块里面有跟基准差不多的纵坐
    标且这个点与基准点的横坐标相差较大，那这个点也是刻度
11.             cout << blocks[i][j].x << endl;
12.         }
13.     }
14.     scales.push_back(temp);
15. }
```

找到每个刻度的横坐标后根据标尺的两头的坐标和刻度值（刻度值手动输入）用线性变换即可得到刻度值：

```
1.  cout << "刻
    度:" << left + (scales[i][j] - Scaleplate_left)*(right - left) / (Scaleplate
    _right - Scaleplate_left) << endl;
```

对与 type2 的括号，由于括号的左右刻度是分开成两个连通块的，所以找到每个连通块纵坐标最小的点的横坐标即为刻度的横坐标（窄的一头在上面），代码如下：

```
1.  if (type == 2)
2.  {
3.      Area_point base = Area_point(0,0,0);
4.      Area_point bottom = Area_point(0, 0, 0);
5.      vector<int>temp_bottom;
6.      vector<int>temp_base;
7.      for (int i = 0; i < blocks.size(); i++)
8.      {
9.          base = blocks[i][0];//因为遍历是从左上到右下进行的，所以刻度的基准为连通
    块第一个点
10.         int max = 0;
11.         int max_index;
12.         for (int k = 0; k < blocks[i].size(); k++)
13.         {
14.             if (blocks[i][k].y > max)
15.             {
16.                 max = blocks[i][k].y;
17.                 max_index = k;
```

```
            }
        }
        bottom = blocks[i][max_index];//连通块最下面的就是括号的底边
        temp_bottom.push_back(bottom.x);

        temp_base.push_back(blocks[i][0].x);//把基准的横坐标存着
        //由于括号是竖着的而且括号两边不连着，所以一个连通块只有一个刻度
    }
    sort(temp_bottom.begin(),temp_bottom.end());
    sort(temp_base.begin(), temp_base.end());
    for (int i = 0; i < temp_bottom.size(); i++)
    {
        cout << "刻度:" << left + (temp_base[i] - Scaleplate_left)*(right - left) / (Scaleplate_right - Scaleplate_left) << endl;
    }
```

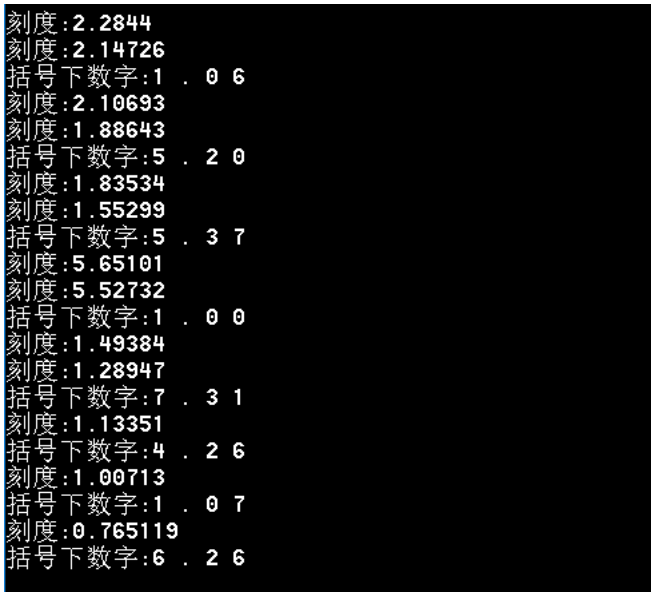
2. 然后是识别标尺下面的数字，对于 type1 和 3 识别比较容易，数字就在标尺下方，根据标尺的位置往下找即可分割出数字，然后用上面的方法即可完成识别：

```
if (j >= 1)
{
    CImg<unsigned char>temp_img(scales[i][j] - scales[i][j - 1], 100, 1, 1);

    cimg_forXY(temp_img, x, y)
    {
        temp_img(x, y) = result(scales[i][j - 1] + x, blocks[i][0].y + y + 13);
    }
    temp_img.save("temp.bmp");
    Number_Detect test_temp("temp.bmp", "");
    cout << "括号下数字:";
    test_temp.getBottomNumber();
    cout << endl;
}
```

由于切割用的是 CImg，识别用的是 openCV 所以要先把切割的图片存起来再用 openCV 读取，有点麻烦。

比较难的是 type2 的数字，由于刻度比较窄数字比较宽，所以不能根据括号上界定位数字，而是应该根据下界进行定位。所以首先要定位连通块的下界，对连通块中的每个点的纵坐标进行排序，找到最大的，然后获得其横坐标即可，对每两个下界之间的部分进行切割即可切割出数字，但是这样也有一个问题就是有的两个下界之间有数字有的没有，而且存在两个连通块连在一起的现象，这让我有点头疼。不过没有数字的连通块识别出来是…或者是空的，这样大概也能获得括号的数据。Type1、3 的识别结果如下：

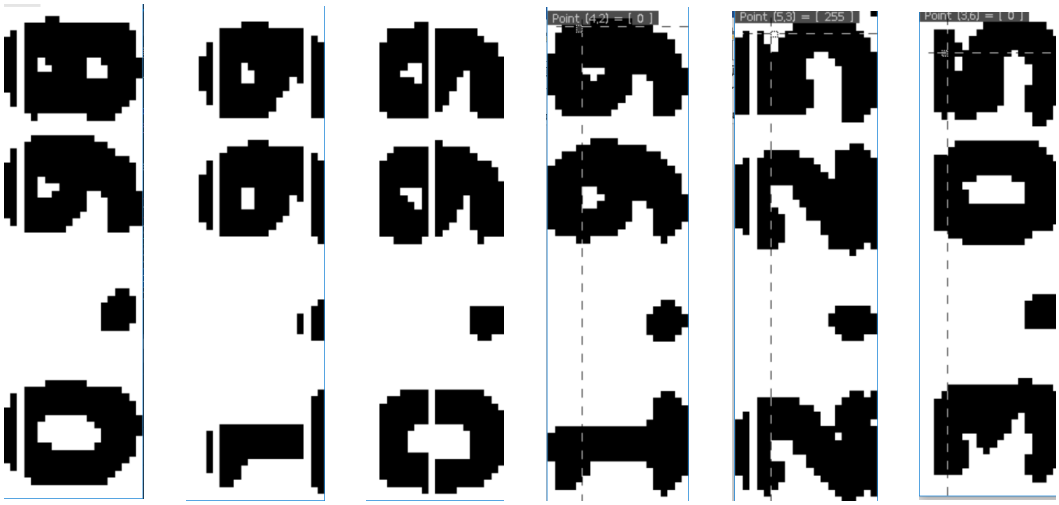
H-Img0（type1）

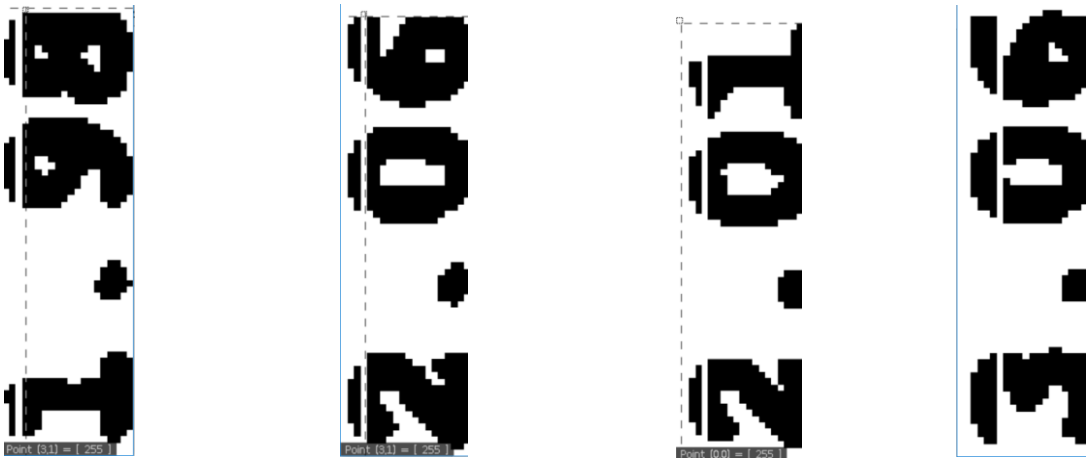


H-Img1(type3)

```
刻度:7.4841
刻度:7.37038
括号下数字:4.00

刻度:7.13927
刻度:7.06223
括号下数字:0.97

刻度:6.95217
括号下数字:2.04

刻度:4.3769
刻度:4.20448
括号下数字:4.98

刻度:2.50598
刻度:2.30788
括号下数字:2.99

刻度:1.62921
刻度:1.4678
括号下数字:2.99

刻度:1.36875
刻度:1.18899
括号下数字:3.07
```

Type2 的数字分割和识别结果如下：





```
括号下数字:
刻度:7.54351
括号下数字:...
9.72

刻度:7.44175
括号下数字:1.99

刻度:7.29835
括号下数字:.
...
...

刻度:7.22434
括号下数字:...
2.25.

刻度:6.92829
括号下数字:
刻度:6.81265
括号下数字:3.05

刻度:4.13433
括号下数字:
刻度:4.06495
括号下数字:...
1.98
```

可以看出分割和识别确实没什么问题，但是划分的时候确实出现了问题。

另外由于这次的操作比较多我把操作整合到一个类 Test 中：

```
class Test {
private:
    string Src_name;
    string otsu_name;
    string delate_name;
    string link_area_name;
    string top_number_name;
    string bottom_number_name;
    string Scaleplate_name;
    CImg<unsigned char>img;
```

```
    int bottom_left;
    int bottom_right;
    int bottom_top;
    int bottom_down;
    int threshold;
    int type;
    double left_scale;
    double right_scale;
public:
    Test(string file, string src_name)
    {
        this->Src_name = file +'/'+ src_name;
        this->otsu_name = file + '/' + "otsu.bmp";
        this->delate_name = file + '/' + "delate.bmp";
        this->Scaleplate_name = file + '/' + "Scaleplate.bmp";
        this->top_number_name = file + '/' + "top_number.bmp";
        this->bottom_number_name = file + '/' + "bottom_number.bmp";
        this->link_area_name = file + '/' + "link_area.bmp";
        img.load_bmp(Src_name.c_str());
    }
    void Do_Operate(int left, int right, int top, int bottom, int thres, int
 typ, double s_l, double s_r)
    {
        otsu Otsu(img, otsu_name.c_str());
        Otsu.seg();
        img.load_bmp(otsu_name.c_str());
        Delate delate(img, delate_name.c_str());
        delate.Do_delate();
        img.load_bmp(delate_name.c_str());
        Link_area link(img, link_area_name.c_str());
        link.get_Link_Area();
        link.filtrate();
        link.get_Top_Number(top_number_name.c_str());
        //link.get_Bottom_Number(top,bottom,left,right,bottom_number_name.c_
str());
        link.get_Scaleplate(Scaleplate_name.c_str());
        Number_Detect detect_scale(Scaleplate_name.c_str(), "");
        detect_scale.getBottomNumber();
        Number_Detect detect_top(top_number_name.c_str(),"");
        detect_top.getTopNumber();
        Number_Detect detect_bottom(bottom_number_name.c_str(), "");
        detect_bottom.getBottomNumber();
        link.get_blocks(thres, typ, s_l, s_r);
    }
```

```
};
```

只用传入图片所在的文件夹和图片名以及一些参数即可。

测试环境：

与 Hw4 相同，环境是 visual studio2017，openCV 的版本是 3.4.5

测试数据是老师给的图片转为的 bmp 格式图片，测试时要更改文件路径，并修改代码中数字模板的位置

测试代码：

```
int main() {
    Test t1("C:/Users/HP/Desktop/HW5", "H-Image0.bmp");
    t1.Do_Operate(1427, 1994, 1458, 1500,3, 1, 5.85, 0.55);
    Test t2("C:/Users/HP/Desktop/HW5", "H-Image1.bmp");
    t2.Do_Operate(-1, -1, -1, -1, 0.3, 2, 9.75, -0.2);
    Test t3("C:/Users/HP/Desktop/HW5", "H-Image2.bmp");
    t3.Do_Operate(-1, -1, -1, -1, 3, 3, 7.95, -0.15);
    return 0;
}
```

对测试结果的分析：

测试结果基本正确，出现的主要问题是：

1. 数字识别不准，可能的原因是膨胀让数字变形，加上训练数据只有一组导致训练准确度不高。
2. 识别 type2 的括号时两个括号之间不一定有数字，这里没有区分，导致识别出来为空或者为…，看起来也不整齐