作业 3: 字符识别：从图片中计算加减算式结果

姓名 王明　学号 161220124　邮箱 1090616897@qq.com　联系方式 13851085654

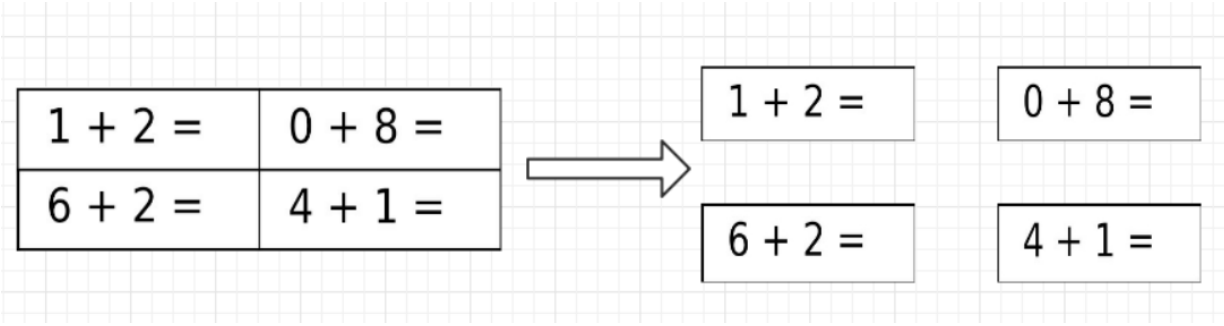
(南京大学 计算机科学与技术系，南京　210093)

1　实现细节

1.1 扣取算式

1.1.1　思路设计

根据提供测试用的图片，首先需要将每一张图的每个算式提取出来，需要达成如下图所示的效果。



为了达成这样的目标，我们需要对于黑色边界进行判断，我的做法是依据横着的黑线将图片进行横向的切割，然后使用同样的方法来对图片进行纵向的切割，这样就可以将一个包含 M*N 算式的图片里的算式分别切割出来。对于那些黑色像素是分界线/那些黑色像素是数字或者运算符，是通过设置一个标志位和给每一行或每一列设定阈值来确定的。

1.1.2　代码实现

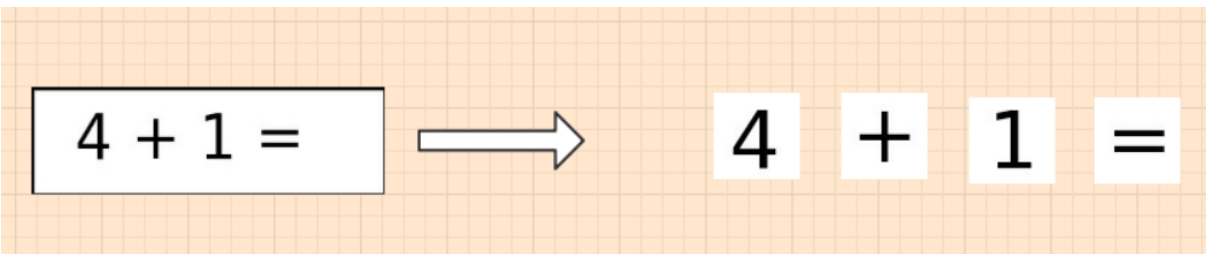
由于行的切割和列的切割是一样的实现，因此这里仅仅贴出行的切割代码实现。

```
%设置一个标志位
flag = 0;
%定义一个一维矩阵来记录切割点的坐标
y_points = [];
%遍历每一行，对每一行的黑白像素进行计数
for j = 1 : y
    count_black = 0;
    count_white = 0;
    for i = 1 : x
        if in(i,j) == 0
            count_black = count_black + 1;
        else
            count_white = count_white + 1;
        end
    end
    %黑色像素多于70%，可以认定是刚刚触碰到黑色边界
    if flag == 0 && count_black >= 0.7 * x
        flag = 1;
        y_points = cat(1,y_points,j - 1);
    %白色像素多于50%，可以认定是刚刚越过了黑色边界
    elseif flag == 1 && count_white >= 0.5 * x
        flag = 0;
        y_points = cat(1,y_points,j);
    end
end
```

1.2 扣取数字和运算符

1.2.1 思路设计

在得到一个算式的切片之后，需要对算式进行进一步切分，得到分离的数字和运算符，需要达成如下图所示的效果。



这一步是从左往右扫描图片，每次遇到黑色像素以及再次遇到纯白的之间就是扣取的数字或者运算符，按这种方式就可以将一个算式进行分解。

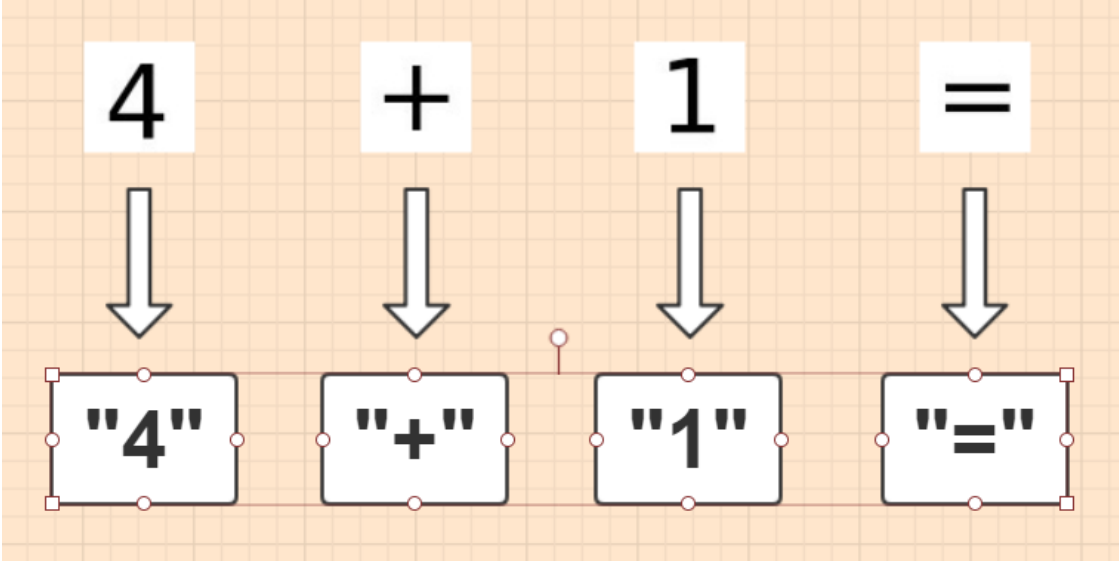
1.2.2 代码实现

```
%定义标志位以及存储切割点的矩阵
col_points = [];
flag = 0;
%从左到右进行扫描，计算每一列的黑色像素的个数
for m = 1 : col
    count_black = 0;
    for n = 1 : row
         if block(n,m) == 0
             count_black = count_black + 1;
         end
    end
    %遇到黑色像素，则遇到一个数字或者运算符
    if flag == 0 && count_black > 0
        flag = 1;
        col_points = cat(1,col_points,m - 1);
    %遇到纯白，那么一定是刚刚越过一个数字或者运算符
    elseif flag == 1 && count_black == 0
        col_points = cat(1,col_points,m);
        flag = 0;
    end
end
```

1.3 识别数字和运算符

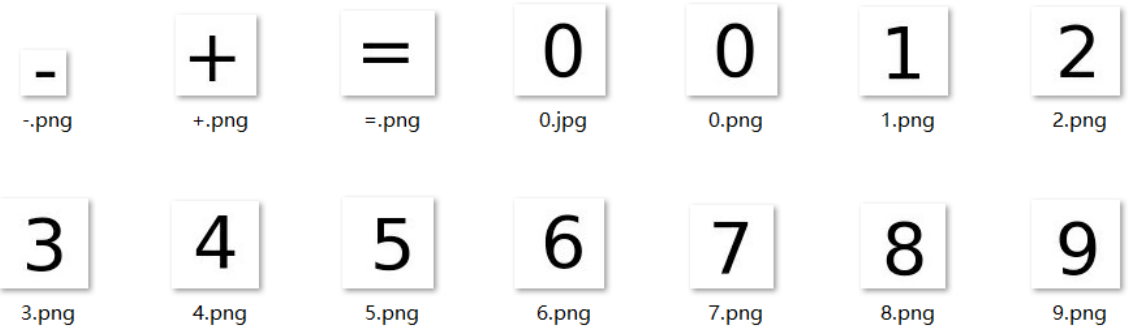
1.3.1 思路设计

接下来就是最关键的一步，对于单个数字或者运算符进行识别，能够识别出图片上的数字是几、运算符是加、减、等号，需要达成如下图所示的效果。

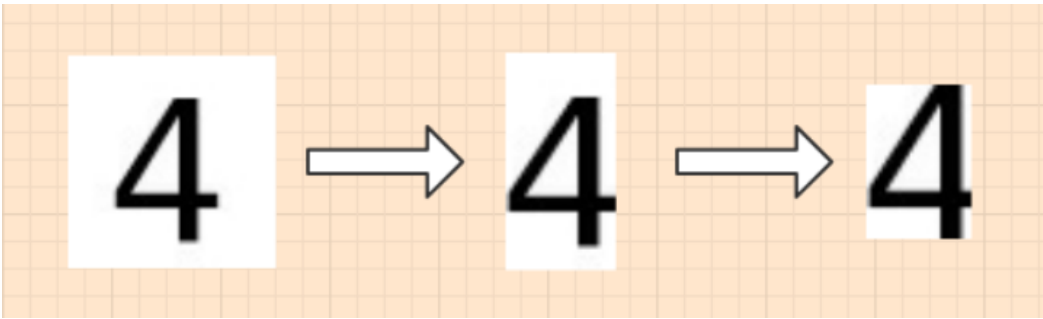


这本质上是一个分类问题，那么分类问题必须有训练集，而且必须是有标签的训练集。考虑到不同图片中的同一个数字和运算符的字体风格十分相近，那么我们可以考虑手动创建训练集，我们直接通过截取样例

图片中的 0 到 9 以及+、-、=等，标签体现在图片的命名上，以它本身的含义命名，截取的效果如下图所示。



但是对于这样的粗样本是不够的，为了之后的对比，我们需要对样本做进一步处理，把每个数字图片的边框去掉（竖直切割和水平切割），缩小到一个包含数字的不能再小的矩形。效果如下图所示。

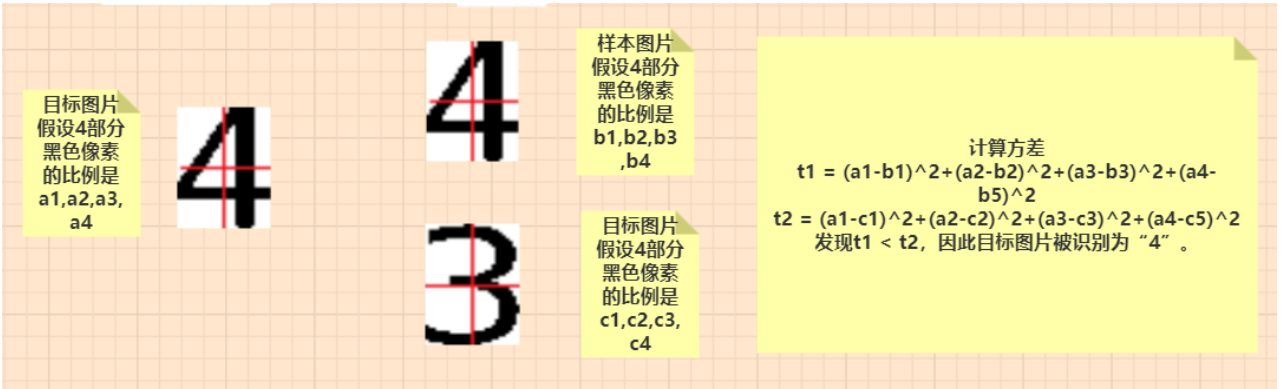


按照这样的方式就可以得到处理后的标准的用来对比识别的样本，如下图所示。

digit_sample:

operator_sample:

现在我们有了标准的样本，那么对于在算式中切片出来的数字或者运算符怎么知道他是多少呢。我采用的方法是，对于目标数字和样本数字，均十字交叉分成 4 个部分，然后计算每个部分黑色像素所占的比例，用目标数字和每个样本数字的对应部分计算方差，所有方差中最小的那个就是识别出来的数字，对于运算符也是一样的算法。这样做的原理是如果是相同的数字，那么切割后每个部分黑色像素所占的比例应该是差不多的，而不同的像素之间却相差很大，因此这种方法可以正确的识别出来，过程如下图所示（这里仅仅以 2 个样本的判别为例，实际操作的时候需要比较所有的样本）：



至此，我们已经可以成功得识别出所有的数字和运算符。

1.3.2 代码实现

将粗样本处理为标准样本：

```
[row,col] = size(img_bw);
%寻找上下左右四个边界的坐标
for i = 1 : row
    flag = 0;
    for j = 1 : col
        %从上往下扫描，遇到某一行有黑色的像素，则找到上边界
        if img_bw(i,j) == 0
            top = i;
            flag = 1;
            break;
        end
    end
    if flag == 1
        break;
    end
end
for i = row : -1 : 1...
for j = 1 : col...
for j = col : -1 : 1...
%找到上下左右所有的边界后，直接切割出来即可
img_out = imcrop(img_bw,[left top (right - left) (down - top)]);
```

数字识别（运算符识别也类似）：

```
%读取所有的样本图片
file_inpath = 'digit_sample/';
img_path_list = dir(strcat(file_inpath,'*.png'));
img_num = length(img_path_list);
%遍历所有的样本图片
for k = 1 : img_num
    img_name = img_path_list(k).name;
    img_bw = imread(strcat(file_inpath,img_name));
    %进行相似度计算，得到方差
    different = similarity(output_image,img_bw);
    %比较方差，取最小的方差为最终识别出来的数字
    if different < diff
        diff = different;
        name = strsplit(img_name,'.');
        output = str2num(char(name(1)));
    end
end
```

相似度的计算：

```
%对目标图片和样本图片十字切割成4部分
[image1_p1,image1_p2,image1_p3,image1_p4] = cut_picture(image1);
[image2_p1,image2_p2,image2_p3,image2_p4] = cut_picture(image2);
%计算相对应的4个部分黑色色素所占比例的方差
diff = power(proportion(image1_p1) - proportion(image2_p1),2) + power(proportion(image1_p2) - pr
```

每一部分比例的计算：

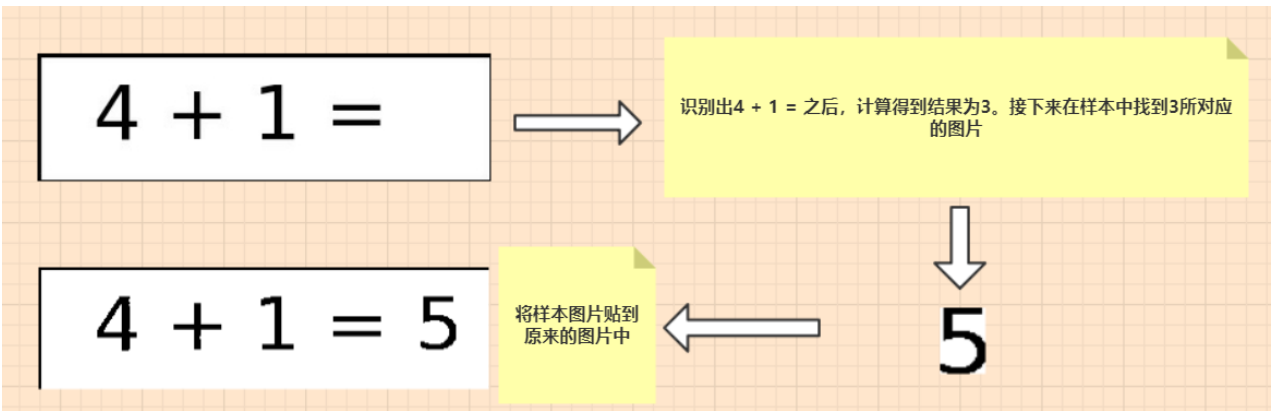
```
num_black = 0;
[row,col] = size(image);
%遍历整个图片，统计黑色像素的数目
for i = 1 : row
    for j = 1 : col
        if image(i,j) == 0
            num_black = num_black + 1;
        end
    end
end
%用黑色像素的数目除以总个数即得到比例
output = num_black * 1.0 / (row * col);
```

1.4 运算得结果并且将结果拼接到图片中

1.4.1 思路设计

我们已经完成了式子的图片切割功能，以及对于单个图片进行识别，那么接下来的运算就很简单，由于规定了式子的格式为“数字+运算符+数字+等于号”的格式，比如“4 + 1 =”的图片，我们按照“数字+运算符+数字+等于号”识别出来之后，可以直接进行运算得到结果，比如说 4 + 1 得到结果 5。

得到结果之后就是把运算结果贴到原来的式子当中。这一步我们的做法是，根据得出的结果在样本中找到与结果相对应的图片，然后将这个图片贴到等于号后面的空白处（只需找到空白处的左上角对应的起始坐标，然后按照像素赋值的方式将样本图片贴到原图片上即可）。这样就实现了将结果拼接到图片上的要求，具体的示例如下图。



1.4.2 代码实现

运算得到结果：

```
%得到切片之后，按照“数字+运算符+数字+等于号”的格式进行识别
for t = 1 : 2 : x3
    a = a + 1;
    clip = imcrop(block,[col_points(t,1) 1 (col_points(t + 1,1) - col_points(t,1)) row - 1]);
    if mod(a,2) == 1
        digit = cat(1,digit,my_digit(clip));
    else
        operator = cat(1,operator,my_operator(clip));
    end
end
result = 0;
%判断运算符是什么并进行相应的计算得到结果
if operator(1,1) == -1
    result = digit(1,1) + digit(2,1);
elseif operator(1,1) == -2
    result = digit(1,1) - digit(2,1);
end
```

最后一步将样本图片贴到原来的式子里：

```
%根据得到的结果读取相应的样本图片
img_num = imread(strcat('digit_sample/',num2str(result),'.png'));
[x4,y4] = size(img_num);
%得到左上角的相应坐标
left_top = [x_points(i,1) + round((row - x4) / 2) - 6 y_points(j,1) + col_points(x3,1) + rou
%利用双层的循环将样本图片的像素值赋值到原来的式子中
for ii = 1 : x4
    for jj = 1 : y4
        out(left_top(1,1) + ii,left_top(1,2) + jj) = img_num(ii,jj);
    end
end
```

2 结果

2.1 实验设置

输入：所给的测试用的图片，放在“test”文件夹下。

输出：计算好结果后的图片，放在“result”文件夹下，命名为输入的文件名+_result。

2.2 实验结果

输入			输出		
1 + 2 =	0 + 8 =	9 - 1 =	1 + 2 = 3	0 + 8 = 8	9 - 1 = 8
6 + 2 =	4 + 1 =	9 + 0 =	6 + 2 = 8	4 + 1 = 5	9 + 0 = 9
8 - 3 =	7 - 2 =	1 + 5 =	8 - 3 = 5	7 - 2 = 5	1 + 5 = 6
9 - 6 =	9 - 0 =	0 - 0 =	9 - 6 = 3	9 - 0 = 9	0 - 0 = 0
5 - 4 =	1 + 7 =	6 - 5 =	5 - 4 = 1	1 + 7 = 8	6 - 5 = 1
2 + 2 =	4 + 0 =	8 - 0 =	2 + 2 = 4	4 + 0 = 4	8 - 0 = 8
5 + 4 =	8 - 1 =	9 - 7 =	5 + 4 = 9	8 - 1 = 7	9 - 7 = 2
7 - 2 =	7 - 0 =	4 + 1 =	7 - 2 = 5	7 - 0 = 7	4 + 1 = 5
9 - 2 =	0 + 8 =	8 - 6 =	9 - 2 = 7	0 + 8 = 8	8 - 6 = 2
2 - 2 =	3 - 2 =	7 - 7 =	2 - 2 = 0	3 - 2 = 1	7 - 7 = 0

9 - 1 =	4 + 3 =	6 - 2 =
0 + 7 =	5 - 0 =	9 - 9 =
9 - 7 =	6 - 0 =	7 - 2 =
2 + 2 =	2 + 4 =	9 - 0 =
6 - 3 =	5 + 4 =	0 + 1 =
6 - 3 =	8 + 1 =	6 - 5 =
4 - 4 =	3 - 3 =	1 - 0 =
6 - 6 =	2 - 1 =	2 + 0 =
9 - 3 =	9 - 2 =	8 + 0 =
2 + 4 =	2 + 4 =	8 - 3 =

9 - 1 = 8	4 + 3 = 7	6 - 2 = 4
0 + 7 = 7	5 - 0 = 5	9 - 9 = 0
9 - 7 = 2	6 - 0 = 6	7 - 2 = 5
2 + 2 = 4	2 + 4 = 6	9 - 0 = 9
6 - 3 = 3	5 + 4 = 9	0 + 1 = 1
6 - 3 = 3	8 + 1 = 9	6 - 5 = 1
4 - 4 = 0	3 - 3 = 0	1 - 0 = 1
6 - 6 = 0	2 - 1 = 1	2 + 0 = 2
9 - 3 = 6	9 - 2 = 7	8 + 0 = 8
2 + 4 = 6	2 + 4 = 6	8 - 3 = 5

3 + 5 =	8 - 0 =	1 + 5 =
6 - 1 =	9 - 4 =	1 - 0 =
8 - 4 =	5 + 0 =	4 - 3 =
8 - 4 =	5 - 2 =	1 - 0 =
7 - 5 =	9 - 1 =	1 - 0 =
0 + 6 =	7 + 1 =	9 - 0 =
1 + 2 =	3 - 3 =	6 + 0 =
7 - 5 =	1 - 0 =	9 - 3 =
8 - 8 =	8 - 6 =	3 - 2 =
4 - 2 =	4 - 3 =	5 - 3 =

3 + 5 = 8	8 - 0 = 8	1 + 5 = 6
6 - 1 = 5	9 - 4 = 5	1 - 0 = 1
8 - 4 = 4	5 + 0 = 5	4 - 3 = 1
8 - 4 = 4	5 - 2 = 3	1 - 0 = 1
7 - 5 = 2	9 - 1 = 8	1 - 0 = 1
0 + 6 = 6	7 + 1 = 8	9 - 0 = 9
1 + 2 = 3	3 - 3 = 0	6 + 0 = 6
7 - 5 = 2	1 - 0 = 1	9 - 3 = 6
8 - 8 = 0	8 - 6 = 2	3 - 2 = 1
4 - 2 = 2	4 - 3 = 1	5 - 3 = 2

9 - 3 =	4 - 0 =	9 - 9 =
6 - 6 =	2 + 5 =	3 - 1 =
5 + 1 =	4 - 4 =	4 - 1 =
8 - 1 =	3 - 3 =	1 + 6 =
9 - 9 =	9 - 9 =	3 - 1 =
6 - 0 =	7 - 0 =	2 + 6 =
4 + 2 =	9 - 1 =	9 - 8 =
1 + 7 =	9 - 1 =	8 - 2 =
9 + 0 =	0 - 0 =	8 + 0 =
5 - 2 =	6 + 1 =	9 - 1 =

9 - 3 = 6	4 - 0 = 4	9 - 9 = 0
6 - 6 = 0	2 + 5 = 7	3 - 1 = 2
5 + 1 = 6	4 - 4 = 0	4 - 1 = 3
8 - 1 = 7	3 - 3 = 0	1 + 6 = 7
9 - 9 = 0	9 - 9 = 0	3 - 1 = 2
6 - 0 = 6	7 - 0 = 7	2 + 6 = 8
4 + 2 = 6	9 - 1 = 8	9 - 8 = 1
1 + 7 = 8	9 - 1 = 8	8 - 2 = 6
9 + 0 = 9	0 - 0 = 0	8 + 0 = 8
5 - 2 = 3	6 + 1 = 7	9 - 1 = 8

4 + 5 =	7 - 2 =	5 - 1 =
3 + 5 =	3 + 0 =	8 - 8 =
5 - 2 =	8 - 5 =	5 + 2 =
0 + 5 =	3 - 0 =	7 - 6 =
2 + 5 =	4 - 4 =	9 - 2 =
5 - 5 =	1 - 1 =	6 - 2 =
1 + 5 =	0 + 5 =	1 + 2 =
9 + 0 =	8 + 0 =	8 - 8 =
5 - 1 =	7 + 2 =	1 + 8 =
5 - 5 =	3 + 6 =	8 - 2 =

4 + 5 = 9	7 - 2 = 5	5 - 1 = 4
3 + 5 = 8	3 + 0 = 3	8 - 8 = 0
5 - 2 = 3	8 - 5 = 3	5 + 2 = 7
0 + 5 = 5	3 - 0 = 3	7 - 6 = 1
2 + 5 = 7	4 - 4 = 0	9 - 2 = 7
5 - 5 = 0	1 - 1 = 0	6 - 2 = 4
1 + 5 = 6	0 + 5 = 5	1 + 2 = 3
9 + 0 = 9	8 + 0 = 8	8 - 8 = 0
5 - 1 = 4	7 + 2 = 9	1 + 8 = 9
5 - 5 = 0	3 + 6 = 9	8 - 2 = 6

2 - 2 =	7 - 6 =	6 - 3 =
9 - 0 =	3 - 3 =	5 - 3 =
7 - 2 =	3 - 2 =	1 + 1 =
6 - 1 =	2 + 6 =	6 - 5 =
8 - 3 =	9 - 4 =	6 - 4 =
7 - 5 =	8 - 6 =	8 - 3 =
1 - 0 =	0 + 8 =	8 - 5 =
6 - 2 =	4 - 4 =	6 - 3 =
7 + 0 =	6 - 5 =	3 - 2 =
9 - 3 =	6 + 2 =	1 - 1 =

2 - 2 = 0	7 - 6 = 1	6 - 3 = 3
9 - 0 = 9	3 - 3 = 0	5 - 3 = 2
7 - 2 = 5	3 - 2 = 1	1 + 1 = 2
6 - 1 = 5	2 + 6 = 8	6 - 5 = 1
8 - 3 = 5	9 - 4 = 5	6 - 4 = 2
7 - 5 = 2	8 - 6 = 2	8 - 3 = 5
1 - 0 = 1	0 + 8 = 8	8 - 5 = 3
6 - 2 = 4	4 - 4 = 0	6 - 3 = 3
7 + 0 = 7	6 - 5 = 1	3 - 2 = 1
9 - 3 = 6	6 + 2 = 8	1 - 1 = 0

2.3 实验体会

总体上感觉此次实验没有想象中的困难，主要的难点在于数字和运算符识别这一块，这一块的做法是利用了相同数字的图片的相同部分的像素比例差不多而不同数字的相同部分的像素比例差别较大这一特征来完成数字和运算符的识别。其他剩下来的事情就是把整张图片分解成一个一个算式，把一个一个算式再切分成单个数字和运算符进行识别，最后将结果再贴回原来的图片即可。所以总体而言，实验的思路还是比较清晰的，每个步骤该怎么解决也都比较好实现。虽然实验难度不是很高，但是代码量也不少，算是一次大型 MATLAB 编程训练。

2.4 关于代码运行的说明

所有代码均放在 code 文件夹下，其中 test 文件夹里存放的是输入，result 文件夹里存放的是输出。在 my_test.m 中可以修改输入（只需修改 imgName 为对应的图片名即可）。

因此只需在 MATLAB 中打开 code 文件夹，将需要测试的图片（比如 a.png）放入到“test”文件夹中，修改 my_test.m 中的 imgName 为’a.png’，然后运行程序，在“result”文件夹中查看结果（结果命名为 a_result.png）。

2.5 关于代码结构的说明

code 文件夹中一共包含了 10 个.m 文件，除了 my_test.m 是脚本之外，其他的.m 文件每个都是一个函数，下面对于每个函数功能进行说明：

my_test.m	函数启动入口，控制整个程序执行
my_calculator.m	输入一张待计算的图片，输出一张计算后的图片
my_digit.m	输入单个数字图片，输出识别后的数字
my_operator.m	输入单个运算符图片，输出识别后的运算符
cut_picture.m	将单张图片按照横竖切割均匀分成 4 个子图片
getblock.m	将一张图片切分到一个一个的算式
getsamples.m	生成用于比较的样本

proportion.m　　计算每一个二值图片中黑色像素占总像素的比例
similarity.m　　利用 4 个子部分的比例差的平方和来表示目标图片和样本图片的相似度
single_process.m 对于一个数字或者运算符图片，进行切割处理，得到能包含数字和运算符的最小的矩形