基于OCR方法的字符识别

姓名：刘琰 学号：161220087 联系方式：893473663@qq.com
(南京大学　计算机科学与技术系，南京 210093)

1. 实验概述

对字符形态的理解，是数字图像处理的重要任务之一。本次实验中通过自己实现的OCR方法，对给定字符进行平面向量化，并于提前生成的模板进行测距的方法，实现了简单的数字、字符的识别。

2. 运行说明

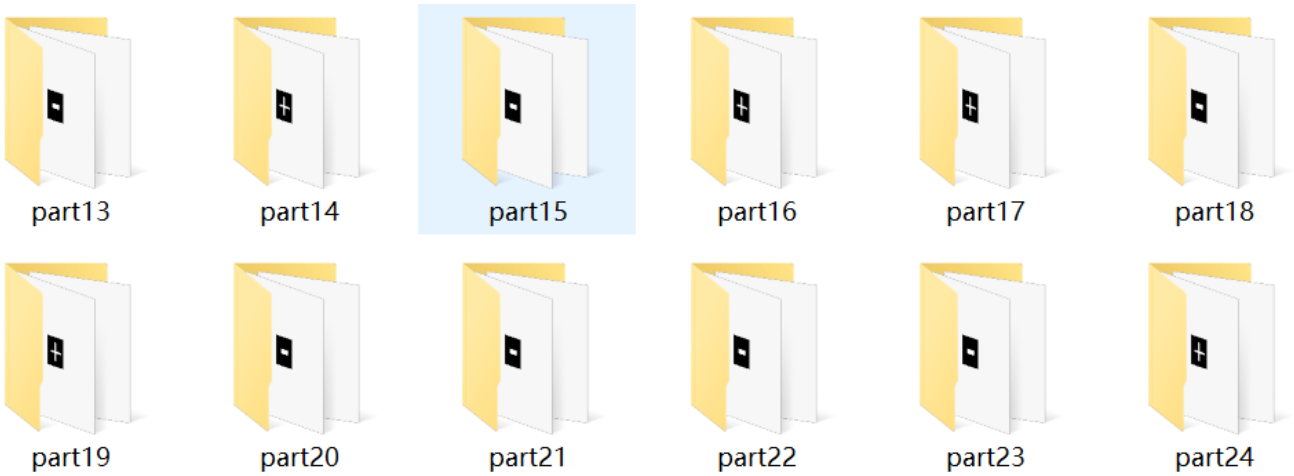
2.1 整体运行说明

运行my_test.m脚本即可，默认对位于'../asset/image/'的6幅测试图像进行处理，输出结果在'../asset/image/result/'中。除测试图像和template文件夹外，其余结果文件夹和结果图片均为临时生成，每次运行均会删除并再次生成。

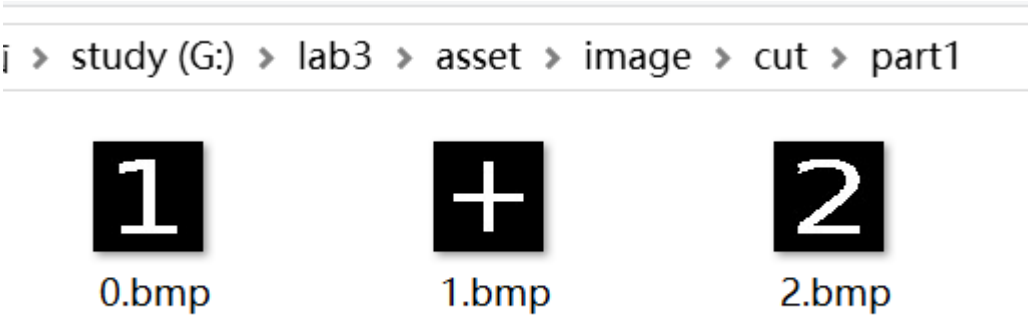
如果需要单独测试某张图片，可以注释26 ~ 34行，并取消40 ~ 48行的注释，结果将直接输出在屏幕上。

2.2 my_digit和my_operator的单独运行

本实验中my_digit和my_operator要求55x55像素大小且背景色黑的图片作为输入，为了测试的方便，预处理中将单独切割出来的字符已经保存至本地'../asset/image/cut'，如下：



每个文件夹表示一个算式（共30个），每个文件夹内容如下：



可以利用其中的字符对my_digit和my_operator进行测试，不支持直接输入图像进行识别。其中my_operator**不支持（也无必要）等于号**的识别。

3. 实现细节

本实验中将整个实验流程细分为如下流程：

- 预处理（包括图像二值化和去噪）
- 裁剪（将整个图片中的每个字符单独裁剪并保存至本地）
- 模式识别（计算图像与模板的矩阵相似度以完成分类）
- 计算结果
- 结果的写入（将结果写入原图像并打印在屏幕上）

3.1 预处理

3.1.1 二值化

由于我们进行字符识别的时候并不关心字符本身的颜色, 而是字符的形状, 并利用其形状对其代表的符号进行识别, 所以我们需要将原来的RGB颜色矩阵转换成01矩阵, 以此得到最有用的信息, 消去不必要的信息。

为了进行二值化, 我们首先要得到一个灰度矩阵(二维矩阵)而非RGB这样一个三维矩阵. 对于彩色图片, 我们首先需要进行灰度变换将原来的三维矩阵转换成二维矩阵以进一步二值化, 在matlab中我们可以直接调用 `rgb2gray` 函数进行灰度化。

```
if size(imgInput,3)==3 %RGB image
    imgInput = rgb2gray(imgInput);
end
```

得到二维的灰度矩阵后, 我们需要找到一个二值化的阈值以对灰度图像矩阵进行0,1变换, 在matlab中, 我们可以调用 `graythresh` 得到一个灰度图像比较合适的二值化阈值, 之后就可以利用该阈值对图像进行二值化.

```
threshold = graythresh(imgInput);
BW = ~im2bw(imgInput, threshold);
```

其中将原图像取反有两个考虑：1. 便于图像的去噪　2. 便于连通分支的识别

3.1.2 去噪

去噪在给出的测试图片中需求并不明显，但我们依然可以预料到，对于低清晰度的图片，去噪显然是必要的。

```
BW = bwareaopen(BW, 30);
```

`bwareaopen`命令接受一个二值化矩阵及最小连通量阈值, 其会将连通量小于该阈值的连通块直接消去, 得到的效果就是细小的白色噪声被消去.

3.2 图像的裁剪

由于一个待识别图像上并非只有一个字符, 我们需要将每一个字符单独提取出来进行识别，步骤如下：

- 去掉图像中无用的边框
- 文本行切割
- 对一行文本逐个切割出单个字符

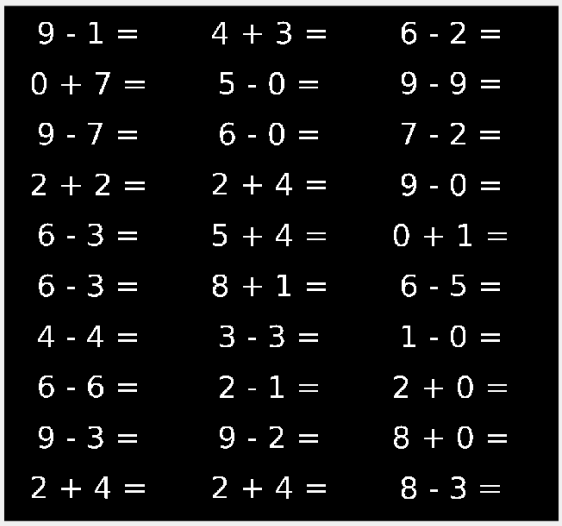
3.2.1 去除边框

由于我们在以上第3步中将（除等于号外）每个字符看作一个连通分支，所以显然不能存在额外的边框，否则边框也将被当作字符进行识别，这严重影响了识别的准确度。

```
L = bwlabel(BW);
[r, c] = find(L==1);
rsz = size(r);
rc = [r c];
for i=1:rsz
    row = rc(i,1);
    col = rc(i,2);
    BW(row, col) = 0;
end
```

我们利用`bwlabel`将图中所有连通分支找出来，而所有边框组成一个连通分支，直接将其在原图像修改为黑色即可。

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 =



3.2.2 文本行切割

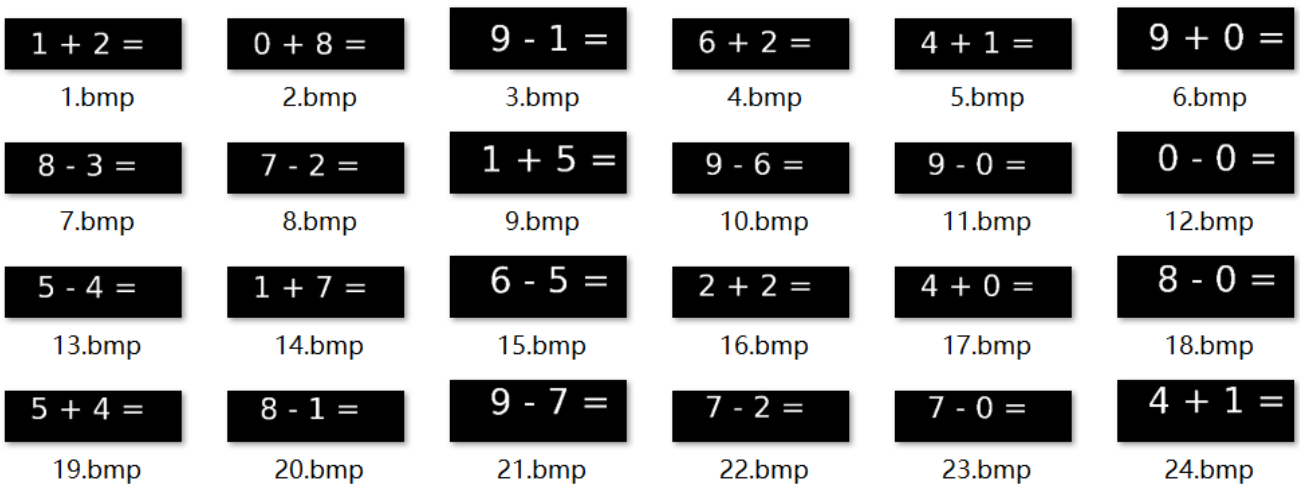
由于文本图像上的字符都是一行一行的, 所以我们可以一行一行的逐行识别, 这里定义一个函数将第一行字符切割出来, 我们单独在`cutImg`函数中实现，如下所示：

```
X=1;Y=1;
[m, n] = size(BW);
%计算每一行每一列的长宽
XInc = floor(n/3);
YInc = floor(m/10);

while(Y+YInc < n)
    %裁剪某一行的所有文本
    imgLine = imcrop(BW, [1, Y, m, YInc]);
    while(X < m)
        %将一行中三个算式单独裁剪出来
        imgCell = imcrop(imgLine, [X, 1, XInc, YInc]);
        X = X + XInc;
        %将每个算式单独保存到本地以便后续处理
        nameNum = nameNum + 1;
        address = strcat(relative, num2str(nameNum));
        address = strcat(address, '.bmp');

        imwrite(imgCell, address);
    end
    X=1;
    Y=Y+YInc;
end
```

处理结果如下：



3.2.3 字符切割

有了每一个行的字符, 更进一步的, 我们需要得到每一个字符以进行匹配, 我们可以利用`bwlabel`函数得到一行文本上的每一个字符的图像矩阵.

```
for i=1:num-2
    %找出第i个字符
    [r,c]=find(L==i);

```

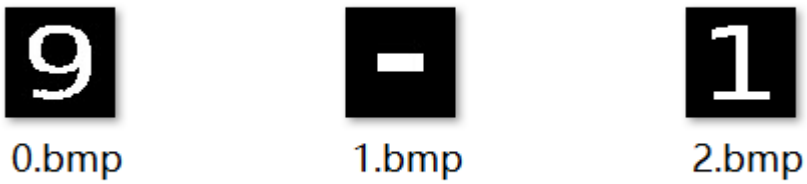
```
    %找出每个连通分支（也就是每个字符的最左上角的起始点）
    rmax = max(r);
    rmin = min(r);
    cmax = max(c);
    cmin = min(c);
    %计算每个字符的长宽，以便将它准确地切割出来
    height = rmax - rmin;
    width = cmax - cmin;

    imgCharacter = imcrop(imgLine, [cmin-10, rmin-10, width+20, height+20]);
end
```

同时，因为下一步跟模板比较时，要求图像尺寸一直，所以我们一律将图像更改为55x55的图像：

```
imgCharacter = imresize(imgCharacter, [55 55]);
```

结果如下：



0.bmp　　1.bmp　　2.bmp

这里我们存在一个问题，也就是"等于"的识别，因为等于号是两个连通分支。其实硬要说的话，单独利用以下方法可以解决：

```
[r1, c1] = find(L==num-1);
[r2, c2] = find(L==num);
lmaxc =max([c1, c2]);   lminc = min([c1, c2]);
lmaxr = max([r1, r2]);  lminr =  min([r1, r2]);
imgCharater = imgLine(lminr:lmaxr, lminc:lmaxc);
```

但是一来破坏了代码整体的结构美，而来做的完全是无用功，等于号根本不需要识别，所以刻意删去这部分代码。

3.3 字符的识别

其实数字的识别和符号的识别是一致的，我们选取数字的识别为例讲解。

在以上**3.2**的完成后，我们可以获得一系列的数字和符号的模板：

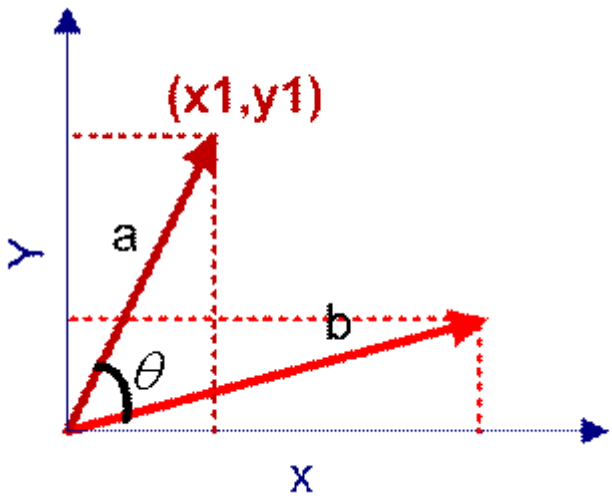
于是，我们可以通过一个字符与已存的模板的相似度判断该字符到底分属与哪个类别。

当然，判断矩阵相关系数，matlab已经给出了一个很好的方法，即利用

$$r=\frac{\sum_m\sum_n(A_{mn}-\overline{A})(B_{mn}-\overline{B})}{\sqrt{(\sum_m\sum_n(A_{mn}-\overline{A})^2)(\sum_m\sum_n(N_{mn}-\overline{B})^2)}} \tag{1}$$

但是，由于我们的模板是从结果中选取出来的，也就是说会存在$A_{mn}=\overline{A}$的情况而使得函数返回NaN，为了避免这一情况，我们采用另一种方法，即利用**空间向量的余弦定理**：

将图像矩阵转化为列向量a,b：



显然，它们的夹角θ代表了它们的相似程度，又$\cos\theta=\frac{(v_1v_2)}{||v_1||||v_2||}$，所以可以用$r=\cos\theta$的大小代表它们的相似度，$r$越大越相似，$r\in[-1,1]$.

```
a = double(img1(:));
b = double(img2(:));
r=dot(a,b) / (sqrt( sum( a.*a )) * sqrt( sum( b.*b )));
```

对于一个输入的数字图像而言，我们只需要分别求出它与9个模板的相似度即可。

```
rmax = -1;  %r ranges [0,1]
rmax_i = -1;
for i=1:10
    r = cmpMatrix(input_image, t{1, i});
    if r > rmax
        rmax = r;
        rmax_i = i;
    end
end
digit = rmax_i -1;
```

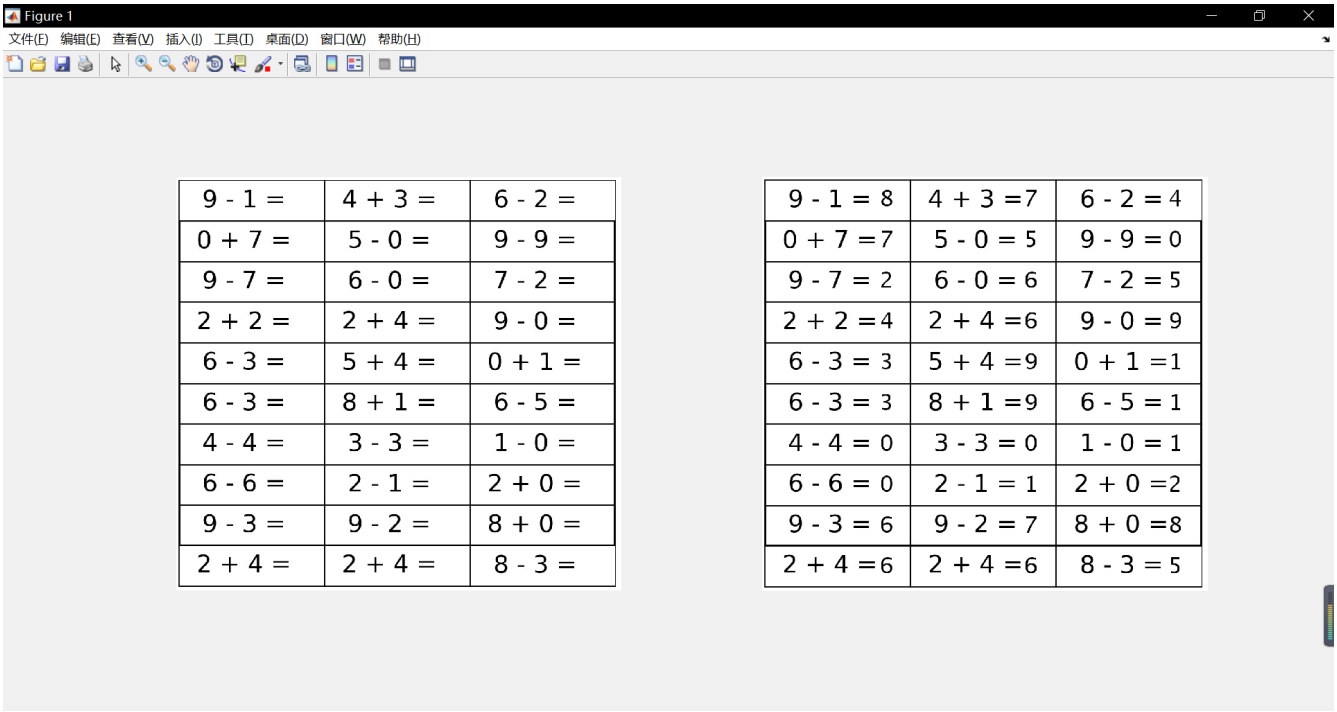
至此，我们已经完成了对于字符的识别。

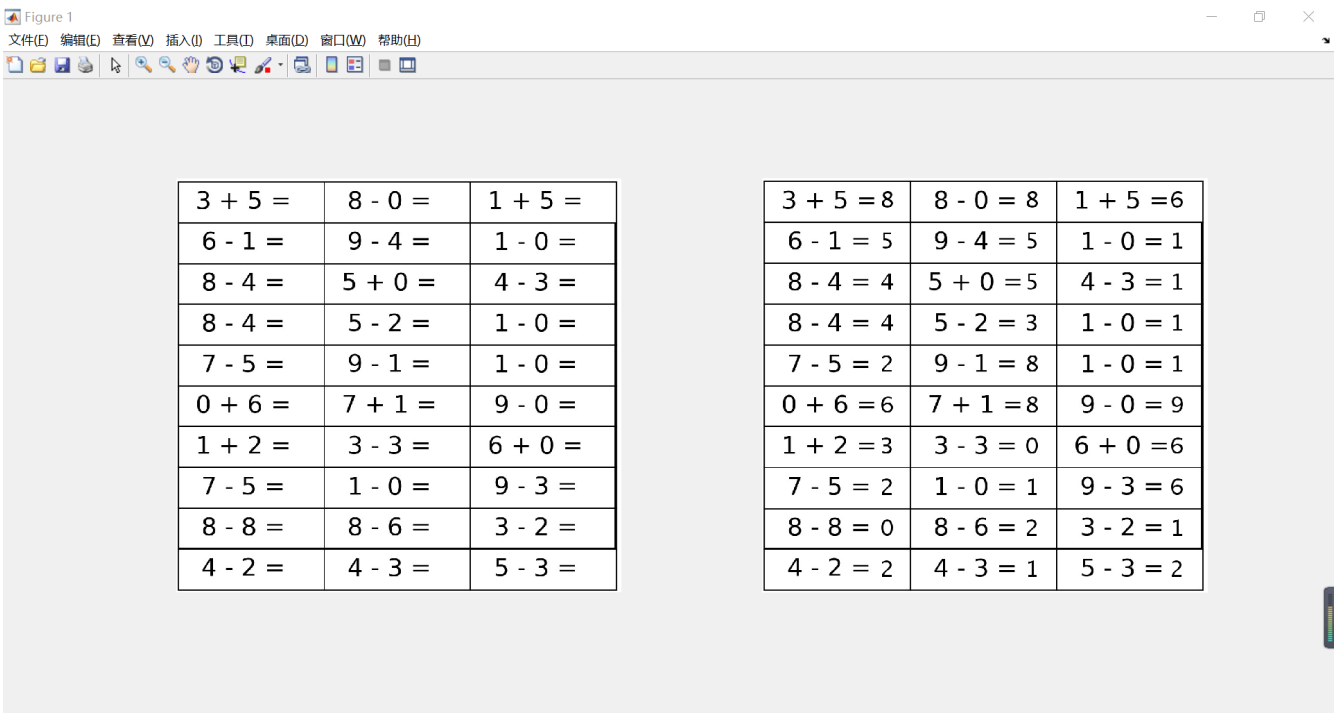
将结果写入原图像只需要调用函数`insertText`即可:

```
%找出输出结果的位置
position = cell(1, 30);
XLen = floor(n/3); YLen = floor(m/10);
X = 253; Y = 4;
cnt = 1;
for i=1:10
    for j=1:3
        position{1, cnt}=[X+(j-1)*XLen, Y+(floor(i-1))*YLen];
        cnt = cnt +1;
    end
end
%输出结果
for i=1:30
    ......
    newimg = insertText(imgInput, position{1, i}, res, 'FontSize', 50, 'BoxColor',
'w', 'BoxOpacity', 0);
    ......
end
```

4. 实验结果

具体的结果图片保存在'../asset/image/result/'中，下面展示两个例子。





5. 参考

https://blog.csdn.net/hjq376247328/article/details/68510125

http://www.matpic.com/esp/matlab/ocr.html