计算机视觉和模式识别 Final Project

*13331231*

*孙圣*

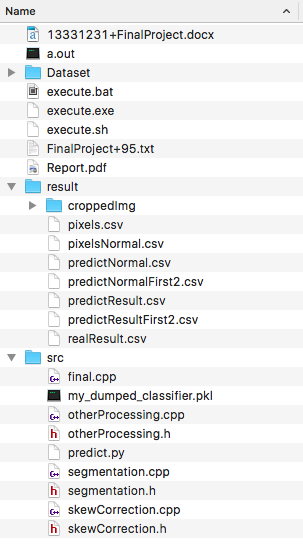
*计应2班*

一、使用说明

通过sh execute.sh或execute.bat直接编译运行即可（需要安装opencv,python3,numpy,scipy,scikit-learn）

测试环境：MAC 0SX 10.11

目录结构：



1.其中a.out和execute.exe为C++部分的可执行程序；

2.Dataset中保存着被测试的三张图片；

3.execute.bat和execute.sh为直接运行所有程序的脚本程序；

4.result文件夹中保存着结果：

4.1.croppedImg目录下保存着所有的截取的数字图像；

4.2.pixels.csv保存着图片的id和展开的像素值；

4.3.predictResult.csv保存着预测的值；

5. src文件夹中保存着所有源文件以及训练好的SVM模型。

二、实验过程

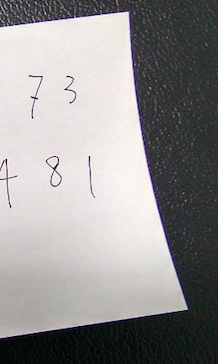
**1. 图像的矫正(skewCorrection.cpp)**

这次实验是把前几次的作业综合起来的实验。因此，最重要的是保证前几次的代码能够重用而不至于出现太多的BUG。

首先利用的是作业三中的A4纸矫正的代码。直接对3张图片进行测试发现，图片2和图片3都能够成功矫正，而图片1不能，原因是：图片1中存在一个明显的红黑分界线，在利用边缘检测和霍夫变换时，会检测到该分界线：



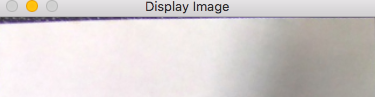
所以一开始的想法是在霍夫变换时，提高相应的阈值，使得这条额外的边不会被检测到。但是这样做会带来额外的问题，导致右侧的边无法检测到。右侧的边由于有所弯曲，并不是在一个完美的直线上，因此当提高霍夫变换的阈值时，该边缘由于包含的点较少，而被排除在外：



所以并不能通过简单的参数调节来解决这个问题。因此又考虑通过去重的方法解决。但是，以前所写的去重主要是负责去除与纸张明显不平行的线段。但在这里却并不适用，因为无关的边缘恰好与纸张的边缘平行，因此要利用其它特点。

考虑到这张图片的特性，即无关的边缘与纸张边缘距离相近，因此可以设置一个阈值来将近距离的边缘排除。在作业3中，考虑过排除相隔距离较远的边缘线段，因此只需要复用那段代码，将相应的参数修改一下即可。

完成了以上修改，对于图片一的矫正基本完成。还有一点需要注意的是，由于图片的边缘并不是一条完美的直线，因此矫正完成后很有可能在边缘处参杂了许多无关像素点，这会给之后的切割带来很大的麻烦，因此要去除：



方法就是通过把边缘处割去几个像素点：

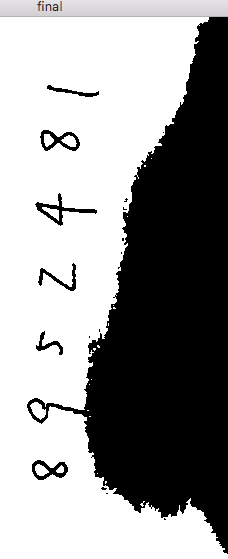


2.**图像切割(segmentation.cpp)**

对于图像的切割主要分为两部分，第一部分为竖直切割，先将纸张切割呈条带状，这里主要使用的方法是在x轴计算频率直方图然后分割。

**2.1 stripSegmentation：**

首先要做的是将图片二值化，一开始考虑使用作业5中完成的OTSU，但是发现OTSU并不适合这个场景,对图片1进行OTSU发现结果很不理想，右侧一大片区域被设为黑色：

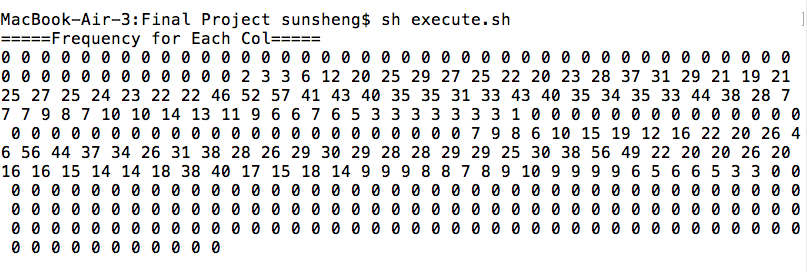


主要原因为光线问题，在拍摄的时候那部分已经很暗，因此使用OTSU就会把它划为黑色的部分。

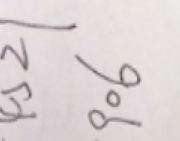
后来改用opencv提供的adaptiveThreshold()方法，其中最后两个参数需要不断调节测试，同时将黑白反转，与MNIST中的数据一致：



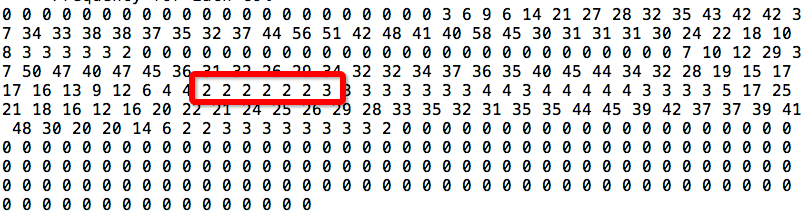
之后就要统计每一列中白色的像素点的数目，统计结果大致如下：



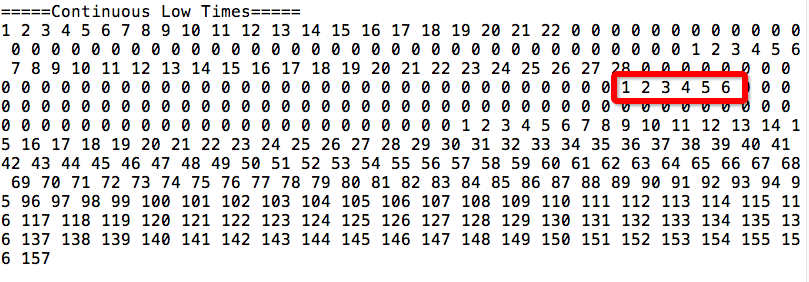
可以发现连续0的位置就是没有文字的位置，也是我们首选的切割点。但是对第3张图测试发现，情况并不是这么理想。因为图中第二列和第三列之间并没有空白：



相应的像素统计信息如下，红色框住的部分应该是分割点，但是并不是全为0的，而是连续的2：



因此，下一步考虑统计像素低于某一阈值(本次实验中设为3)连续出现的次数。对于图3，有以下结果，红色框框住的区域就是上面所提到的两列数字有重叠的区域：



由此可见，我们只需要将0->非0数字转变或者非0数字->0转变的点作为切割点即可。对图3进行统计可得：



之后就要进行切割。还有一个优化的地方：按照之前提到的方法寻找切割点，会导致数字的某些像素就在图片的边缘，与MNIST数据集相差较大，因此要补充一定的像素作为边界：



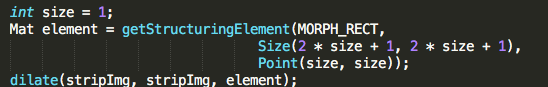
最后就是对竖直的图像进行旋转，转换成水平的图像。

**2.2 characterSegmentation**：

这一部分与前一部分相比难度更大，因为在图片2和图片3中，有部分数字挨的很近，到时按照投影的方法切割十分困难。因此这部分主要采用的是opencv中提供的findContours()方法：

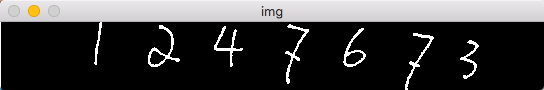


同时为了使得把数字作为一个整体识别出来而不是分为几个子区域，要对图片进行膨胀，增大白色的像素点：

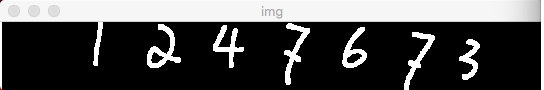


结果如下：

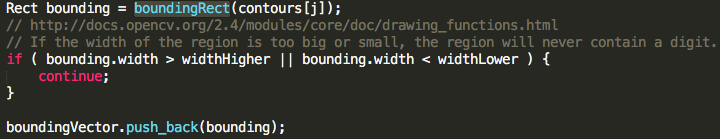
膨胀前：



膨胀后：



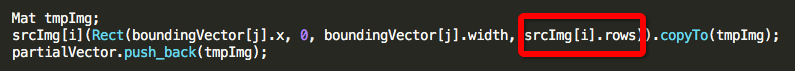
findContours()函数返回了一系列的点作为轮廓，首先我们要将其变为矩形，因此调用boundingRect()方法。之后要对矩形进行筛选，如果太宽或者太窄都不可能是包含了数字的矩形，应该排除：



得到了一系列的矩形之后，要根据x轴的坐标进行排序，保证数字是从左往右依次排列的。之后就是对图像进行切割。这里要注意的是切割时不能按照给出的高进行切割，因为findContours返回的区域并不能完整的包括所要的数字，例如：

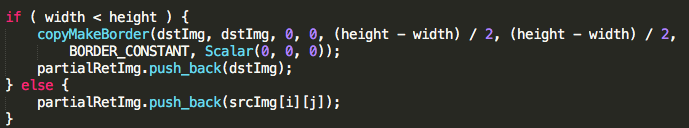
因此考虑将截取的高设为原图像的高：



将切割好的图片放入vector内返回。至此，对数字的切割基本完成，剩下的就是进行相应的处理，并完成预测。

**3. 图像的再处理(skewCorrection.cpp)**

首先，要给刚才截取的图片添加边界。对于大部分的图片，宽远远比高要小。当之后要缩小到28\*28的图像时，会带来很多的问题。所以就给图片填充纯黑色的边界：



将图片缩小之后要做的就是风格上的处理，使得图片尽可能的接近MNIST数据集中提供的图片，提高正确率。有几点可以尝试的，第一，因为MNIST中的数字都很粗，所以可以利用膨胀来加粗；第二，可以对图像进行平滑处理等等。

**4.数据的保存(final.cpp)**

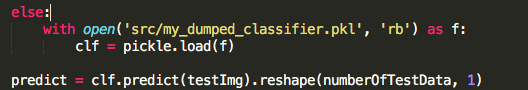
在训练之前，首先要把图片的数据都保存起来。首先利用imwrite()方法，将图片写入到result目录下的croppedImg文件夹下。命名规范为ImageID ID ID.jpg，三个ID分别为图片ID，条带ID和位置ID。

之后利用文件流的方法将ID和展开的像素值写入到pixels.csv中，以供python读取。

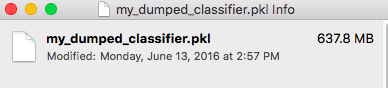
5.**训练与预测(predict.py)**

这部分的代码主要是依赖于作业6中的实现。首先利用numpy提供的genfromtxt()方法读取之前存下来的图片数据。之后进行归一化。

这部分代码最大的改进地方在于，将训练的模型持久化的保存下来。这里利用了pickle.dump()和pickle.load()方法。这样就可以避免每次预测时都需要花费时间重新训练：



这里采用的模型是SVM的模型，因为模型占用的空间较小，只有64KB，而使用了随机森林的adaboost模型却有600M左右：



因此尽管SVM模型在训练集上的正确率只有92%，不如adaboost的97%，考虑到空间因素，还是采用SVM模型。

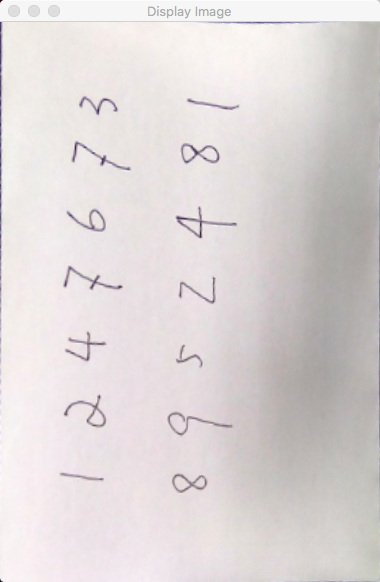
最后将预测的数据写入到文件中并在屏幕中输出。

至此所有步骤都成功完成，效果分析见实验结果部分。

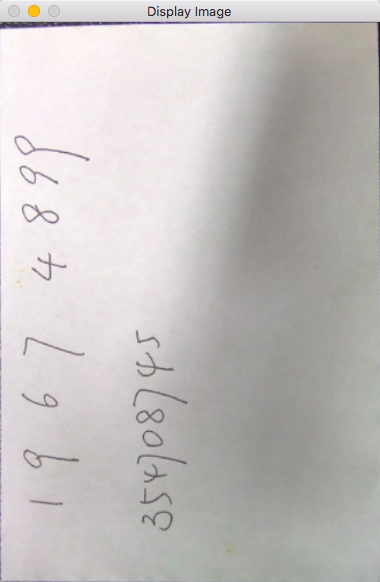
三、实现结果：

矫正结果：

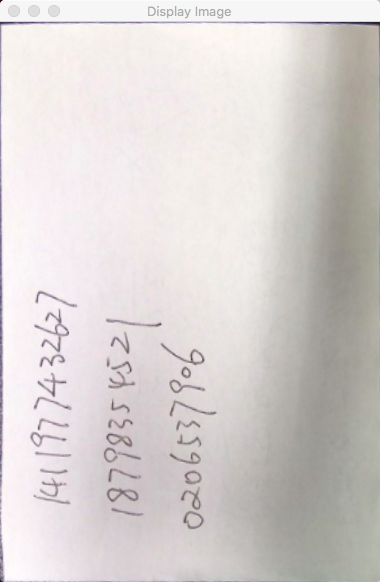
1.



2.

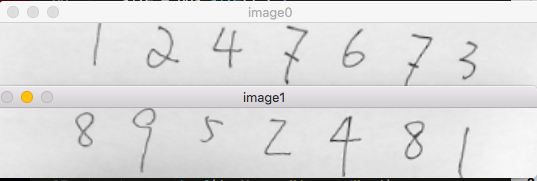


3.

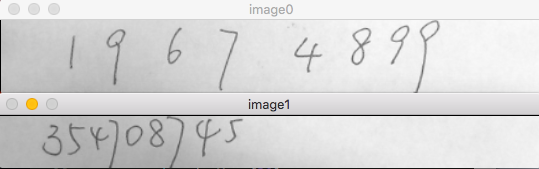


竖直切割：

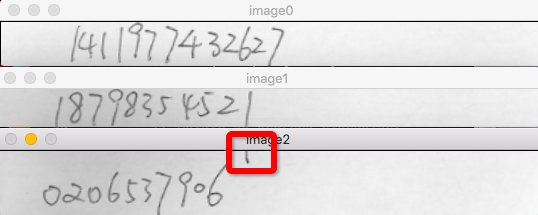
1.



2.

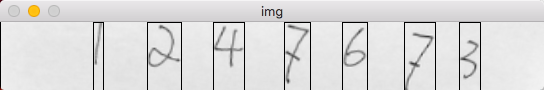


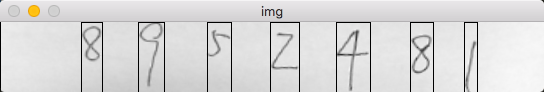
3.可见，由于重叠的问题，image2中还是有1的部分痕迹：



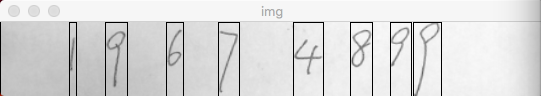
数字切割：

1.



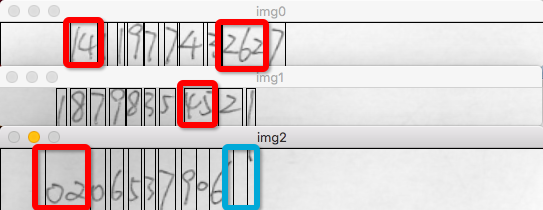


2.





3.可见红色框住的部分都是因为数字挨的太近，导致无法分割成功。而蓝色框住的部分由于有噪声导致误判：



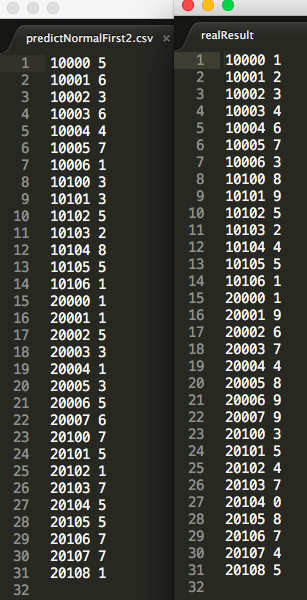
切割字符节选：

Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image1 0 0.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image1 0 1.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image1 0 2.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image1 1 1.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image1 1 2.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image2 0 2.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image2 0 3.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image2 1 4.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image2 1 5.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image3 0 7.jpg

不正确切割字符：

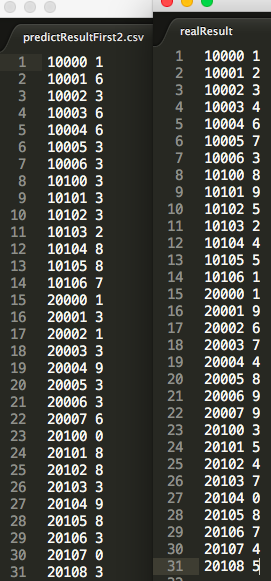
Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image3 0 0.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image3 0 8.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image3 1 7.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image3 2 0.jpg Macintosh HD:Users:sunsheng:Desktop:ComputerVision:Final Project:result:croppedImg:Image3 2 9.jpg

预测结果一般情况（仅截取前两张图的数据）：



共有31张图片，预测正确10个，正确率32%。

对图片进行膨胀操作的预测结果情况（仅截取前两张图的数据）：



共有31张图片，预测正确7个，正确率22%，比不做膨胀处理的正确率低了10%。

原因分析：

造成预测成功率较低的原因有以下几个方面：

1. 图像切割时的问题：

1.1. 图像无法完美切割，即部分像素点被排除在截取的图片之外，同

时又有其他噪声加入到截取的图片中；

1.2. 图像切割时位置的问题，数字像素并不是集中在图像的中部，有

一些集中在上半部分，而有一些又偏下；

2. 图像预测前处理的问题：

2.1. 图像缩小带来失真的问题；

2.2. 图像像素值与训练所用数据集内图像像素值差距过大的问题：尽

管已经做到使得图片的样式和训练样本中图片的样式差不多，但是像素的

值的分布还是有很大的差距的。训练集中的图像的值大多为0和255，有

少量的值是接近0或255。但是截取出来的图片中，有部分像素值是90-

170左右的中间值，这给之后的预测带来了很大的麻烦；

2.3. 数字粗细的问题，训练集中数字是相对较粗的，而我们截取出来

的数字是相对较细的。但是经过一次核为1的膨胀之后，由于图像本身较

小，整个数字的形态会发生巨大的变化，因此造成相应的问题；

3. 分类器的问题：

3.1. 由于分类器是运用MNIST数据集进行训练的，由于书写风格，环

境等种种因素的影响，很难推广到其他手写体识别的预测集上；

综合考虑以上几个方面，我们可以提出几个方法来改进，提高预测的成功率：

1. 使用更好的切割方法，然后对图像进行平移等变换，使得数字位于图像的中间；

2. 对图像做更好的预处理，这方面不局限于膨胀等基本方法，也可以是其他相对高级的方法；

3. 利用推广性能更好的数据挖掘算法，训练出适用面更广的方法。

老师/TA，您好。我未来有出国读研并在这方面发展的打算，在这门课上也投入了很多，希望能取得一个不错的成绩，谢谢！