

计算机视觉期末大作业

朱明杰 15331441

一. 实验任务

1. 输入普通 A4 打印纸，上面可能有手写笔记或者打印内容，但是拍照时可能角度不给出正。输出图像四个角点的坐标，以及校正后的图像。
2. 提取上面 A4 打印纸的学号、手机号码和身份证号。

二. 实验工具

Visual studio 2017

CImg Library

LibSVM

xInt

三. 算法流程

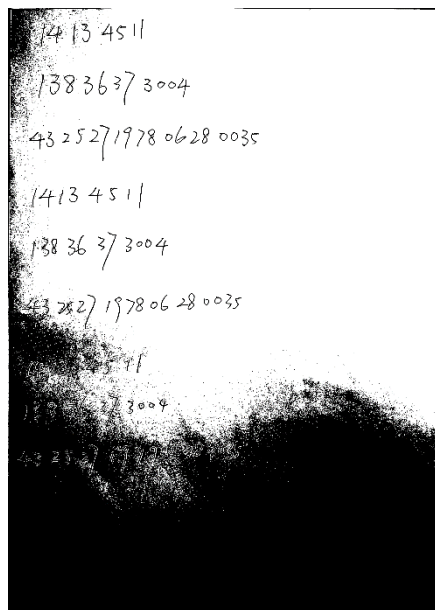
1. 图像角点坐标以及校正后图像。

尊师嘻嘻卡，直接抄第四次作业就完事儿了噯铁汁，欧力给！

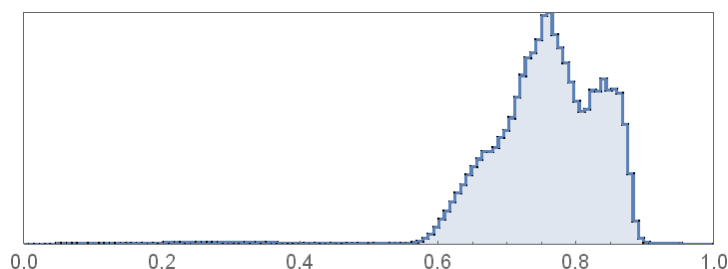
2. 图片的分割以及数字的提取。

直接在校正后的图片上进行。需要注意的是，校正后的图片边缘有可能有轻微的对不齐，所以我们需要设置一个可以容忍对不齐边缘的 margin，margin 内的不予考虑。

然后我们要对图片进行二值化。一开始我想尊师嘻嘻卡，简单地套一个 Otsu 过来，结果一下子就……

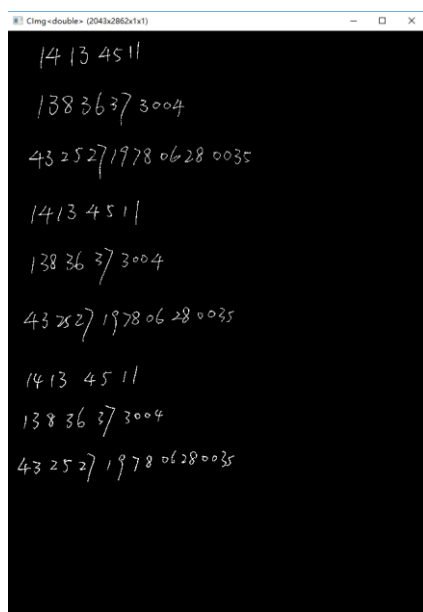


可以看到下面有很多黑色的东西，这是坠痛苦的。后来我经过理性分析，发现原因竟然是纸张下部有阴影较暗。所以我们来看一下纸张的灰度直方图：

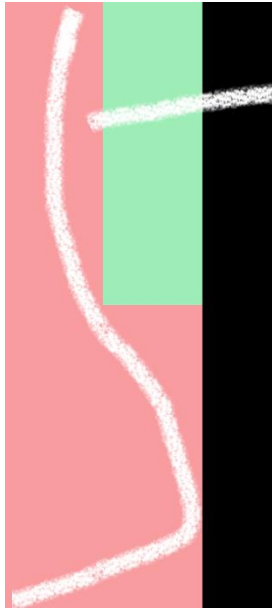


可以看到，图像的灰度确实呈现出一个双峰。但是我们的文字肯定是非常黑的（假设我们用黑笔），而一大片阴影的灰度值在 0.6~0.8 之间，纸张较亮的部分灰度值在 0.8~0.9 之间。直接跑 otsu 让类间方差最大化，肯定给出的阈值就是 0.8 左右了，然后纸张就被二值化成那样。所以，具体情况，具体分析，我们不能只从图像的全局统计特征（灰度直方图）来入手。局部的方法可能会得到更好的效果。

这里我们借鉴 Canny 算子的双阈值方法，来进行 Floodfill 搜索：定义两个阈值 t_1 （严格）和 t_2 （宽松），对整个图像进行遍历，当遍历到灰度值小于 t_1 的点的时候开始搜索；对于每个搜索队列中的点，可以扩展的邻域点是那些灰度值小于 t_2 的点。这样一来我们就只会在灰度较小（有数字）的地方开始搜索，而不会在那些灰度较大的地方（阴影处）进行搜索。这样我们就能得到一个比较好的结果：



在 Floodfill 搜索的同时，我们就能顺便把数字切割给做了。在理想状况下，然而这还是太理想化了，除了数字 5，其他的数字都是一笔画、单连通的，所以需要为了判断数字 5，我们需要在已经寻找出来的连通域的外接矩形的右上部分额外去寻找有没有那个横，而且为了保证每次先搜到的是钩而不是横，我们需要把图像遍历顺序变成从下到上、从左到右。如下图所示，白色为数字 5，有 2 个连通域。红色矩形为第一次搜到的钩的外接矩形。绿色矩形为我们横线的判定区域。



对于接连两个数字，如果它们的外接矩形在宽度的投影交集非空，那么我们则认为这两个数字是在同一行的。对于同一行的数字，对它们的中心横坐标排序，我们便得到了同行数字的顺序。

3. 分类器的训练。

如果不考虑上深度学习的话，分类器一般可以上逻辑回归(LR)和支持向量机(SVM)。我选择了后者，并且选择了 [LibSVM](#) 这个开源库。手写数据集直接选用 MNIST 来做。借助[一个 MATLAB 的程序](#)，我们能够很简单地加载 MNIST 数据集。然后将数据转化成 LibSVM 钦定的格式，并且调用 LibSVM 目录下的 tools/easy.py，就能够无脑得到局部最优参数。我得到的最好参数是

```
[local] 5 -7 98.055 (best c=32.0, g=0.0078125, rate=98.055)
```

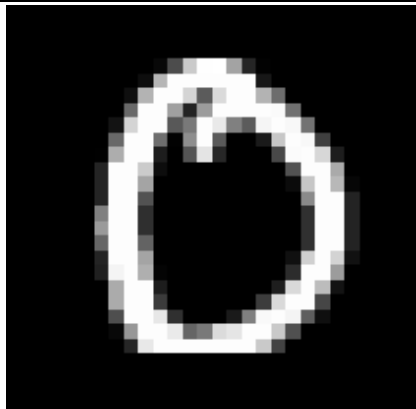
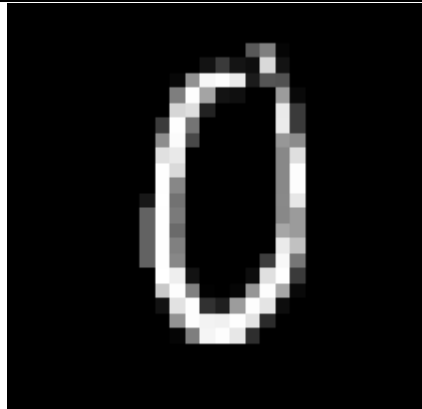
这告诉我们 $c=32.0$, $g=0.0078125$ 的时候，MNIST 能在测试集上做到 98.055% 的识别率。可以说，如果这个模型能够在测试集上做到这么高的识别率的话，如果我们的手写体与 MNIST 的差异不大的话，做到正确识别应该不会很难。我们将分类器预训练好，并且将模型文件放到我们项目文件夹下。

4. 数字的进一步处理与识别。

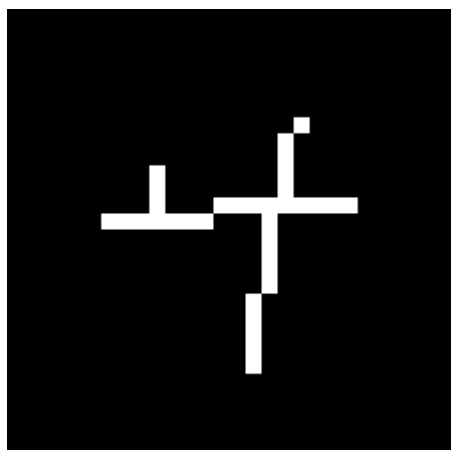
至于识别的话，尊师嘻嘻卡，带上启迪系的第一个符文行窃预兆，偷就完事了。稍微将 LibSVM 里面的 svm_predict 修改一下，就可以写出我们的预测代码。

当然第二步中数字提取的结果还需要进一步的处理。我们注意到 MNIST 数据集手写体都是 28×28 的灰度图像，而且是按照列优先的方法展开成的向量，这是需要我们注意到的。将 MNIST 数据集打开分析，发现手写体边缘跟图像边界差不多留了 4 个像素的宽度。所以我们的做法是先将之前提取出来的数字按长边做出一个正方形的图（譬如之前是 150×200 的数字，就做出一个 200×200 的数字，两幅图的中心要重合，多余部分用黑色填充），这样做的目的是保持数字的长宽比不变。然后重设大小为 20×20 。最后加入长度为 4 的黑色外圈边框，就生成了 28×28 的数字图像。

然而直接提取的数字可能会产生一些问题，如老师上课的时候讲到的我们的笔触大小与 MNIST 数据集笔触大小不一样的问题。这个问题其实 MNIST 数据集本身也有，例如 MNIST 测试集中以下两个样本：

测试样本#89 (数字 0)	测试样本#233 (数字 0)
	

可以看到，测试样本#89 的笔触明显要比测试样本#233 的笔触要粗一些。当然我们的笔触大小也不能跟 MNIST 数据集的产生太大偏差。为了解决这个问题，我尝试先用二值化加上细化算法抽出数字的骨架，然后再进行膨胀操作，最后将原图和二值化图用 3:1 的权重混合起来。再尝试的过程中，我还试了用二值化→细化→膨胀操作的结果作为掩码，来抽取原数字在结果为 1 上的像素灰度值。但是后来我查看了一下二值化→细化→膨胀操作的结果，发现结果欠理想，主要是很难找到一个非常令人满意的细化算法。



上图是某个 4 经过这二值化→细化下来的结果。可以看到这样弄了之后，数字骨架已经有了比较严重的变形，再膨胀之后对于之前的预测价值也不是很大。所以我最后决定将原图和二值化图用 3:1 的权重混合起来，再用一个阈值卡掉那些灰度值过低的点来构成最后待预测的图片。

再次观察的样本，在 MATLAB 中查看 MNIST 训练集上每幅图最亮的地方是什么样的：

```
>> min(max(x_train'))
```

```
ans =
```

```
0.9961
```

对 MNIST 测试集也做一遍：

```
>> min(max(x_test'))
```

```
ans =
```

0.9961

所以我们也需要将最后待预测的图片做一次归一化，来让我们待预测的图片与 MNIST 的训练集更加地吻合。

还有一个问题，就是由于我们的 `resize` 是选用的线性插值算法而非最近邻算法（肯定不能用最近邻算法，因为在事先不知道数字大小的情况下贸然用最近邻算法很容易得到纯黑图），由于插值算法的问题，非 0 的地方会比我们想象中的要大。这个时候我们先要用一个非常小的值（譬如 0.05）去卡掉这些假的非 0 点。

5. 输出结果至 `xlsx`。

首先我看了一下 MSDN，里面建议用 ODBC 来操作 excel 表。后来我找了一个叫做 [xlnt](#) 的开源库，然后对着 `xlnt` 文档中的样例代码改一改就行了。值得注意的是，由于 `xlnt` 是基于 `unicode` 实现的，所以直接用普通常量字符串（例如“图像名”）来填充 `xlsx` 的某个单元格会报错 `xlnt::serialization`。后来我结合 [issue#215](#) 中讨论的内容，摸索出了解决方法：加 `u8` 前缀来指明常量类型（例如 `u8"图像名"`）就行了，`xlnt` 不会报错，而且输出的结果也是正确的。

四. 算法代码 (CImg) 实现

见压缩包下代码。

五. 实验结果

数据文件见 `Dataset/Dataset` 中文件。

结果见 `Dataset/myresult` 中文件。其中以数字命名的文件夹是 `debug` 过程中将识别到的数字逐一输出的结果。

六. 分析与评价

整个过程下来，识别率还是不是很能令人满意。原因有以下几点：

1. `row data` 所造成的信息丢失与对齐。

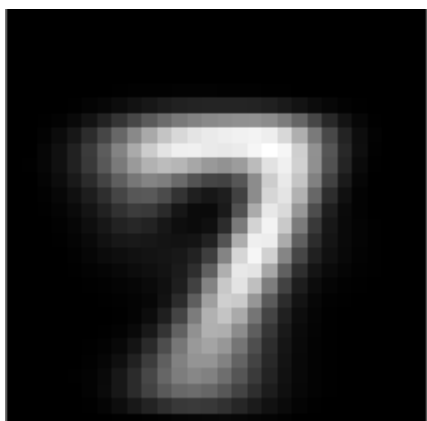
训练 MNIST 的过程是直接将 28×28 的图片按照列优先的方式展开成一个向量，再根据这个向量进行训练。这样看上去比较自然，实际上会带来一些问题。这样处理一个图片，很容易因为一些平移或者旋转上的细小差别而带来向量意义上的较大差别（维数上的错位），从而极大地影响训练与识别效果。对于图像的特征抽取与识别，使用 CNN 来往往能达到较好的效果。我找到了一个 C++ 的 [torch 库](#)，但是由于精力与金钱限制（龙鸣显卡没办法 QAQ）而没有去尝试一下，还是比较可惜。

2. MNIST 与日常手写体差异过大。

譬如 MNIST 里面所有数字为 7 的样本的均值如下图所示：

```
>> clearlove = reshape(mean(x_train(y_train==7,:),[28,28]);
>> imshow(clearlove,[])
```

结果如下：



可以看到，总体来说，MNIST 测试集中 7 的横都是比较明显的，而我们平常在写字的时候，我们的 7 都倾向于写得非常的修长，横也写得不是很明显，结果 resize 到 20×20 的时候，那一横一下子就……然后因为这个横太不明显所以直接被 SVM 判成 1，这是挺痛苦的。同理还有数字 9，也是一个不注意分分钟就被判成 1。目前我还没有想到比较好的通过图像处理来改进识别率的解决办法。

3. 连笔字的问题。

这个基本上属于无解。这是图像切割的范畴，而除了利用 Floodfill 来切割之外，我还找到了一个利用投影直方图来切割的[讨论](#)，但是实际上投影直方图并不会因为连笔而呈现出多峰，也并不会在连笔处有明显的凹陷。

4. 写字中途断笔的问题。

考虑下面的问题：有一个小鹏在用铅笔写字的时候铅笔突然断了，然后这个小鹏与接着铅笔断之前的地方写下来，但是这个时候由于断笔的顿了那么一下，这个小鹏与之后写的部分和之前写的部分没能完全连起来，一个数字变成了两个连通域了。这个问题在现在看来也是基本上属于不能通过图像处理来解决的问题。