

图像处理课程设计报告

1600012731 徐可涵

任务内容

车票序列号检测与识别

对于一系列给定的火车票扫描图像，确定出火车票中7位码和21位码的位置，并分别识别出每一位数字/字母。

算法原理

(1) 车票票面检测

实现对车票位置的检测、分割和旋转，这部分代码见 `crop.py`。

初步考虑用Canny算子提取图像的边缘，再用Hough变换找出图中所有直线。车票的边缘应该为分别最靠上下左右的四根直线，据此检测出票面。

实际情况中，由于所有图片的右侧均有一小条白边，故只取图片从左侧开始像素宽为1000的部分（图片尺寸统一为1080*1200），截去这部分白边。由于扫描得到的图片在车票外侧的黑色部分中留下了断断续续的一些白线，故用高斯模糊去除这些噪声后再进行边缘检测，防止这些白线被认作车票边缘。

`lines1` 数组中存放了检测到的所有直线的两个端点的坐标，`up, down, left, right` 分别记录所有直线中点里最靠上下左右的横/纵坐标。由于所有的样例里车票的位置都非常接近于竖直方向，故我们直接按水平竖直方向进行切割（`crop = img[up:down, left:right]`）而不考虑倾斜的细微角度。

要把切割后的图像旋转到水平方向，只有顺时针旋转90度和旋转270度两种可能性。在进行Hough变换时发现，由于车票下半部分有一个小的虚线方框且字体更为密集，故检测到的直线比上半部分多。因此开一个 `average` 变量记录所有检测到的直线的中点在x轴上的均值，根据与它更靠近的是 `left` 还是 `right` 量来判断应如何旋转。

(2) 车票序列号定位

对车票中7位码和21位码进行定位，这部分代码见 `detection.py`。

为了方便进行定位，先将灰度图进行二值化。由于在车票的灰度图中7位码和21位码的灰度值并不相同，分别采用不同的阈值对图片进行二值化后再检测。

21位码检测

- 左侧：在x轴值为 `[50, 500)` 的区间从左往右检测，若在一定高度范围 `[height-100, height-10)` 内出现 ≥ 3 个黑色像素则说明遇到黑色字符，将该x轴值确定为左侧位置。
 - 将黑色像素个数规定为 ≥ 3 个而非1个是为了排除扫描时在白色区域出现噪声小黑点的可能性。（事实上确实有样例因此而影响判断）

- 考虑到可能左侧位置比50更小，如果在x值为 [50,53] 间（不仅限50是考虑到字符之间的空隙）检测到 ≥ 3 个黑色像素，则把左侧的检测范围更改到x值在 [20,60] 间重新开始检测。
- 下侧：在y轴值为 [height-50,height) 的区间从上往下检测，若在一定水平范围 [50,500) 内都没有黑色像素则说明检测到21位码的下侧，将该y轴值确定为下侧位置。
 - 小部分图中21位码的位置较靠下，其最上方位置都低于 height-50，这样也会检测到y轴值为 height-50 时水平方向像素均为白色，但实际是把21位码的上侧检测成了下侧。因此仿照左侧的做法，如果在x值为 [height-50,height-45] 间检测到水平方向像素全白，则把x的区间变为 [height-20,height) 重新开始检测。
- 上侧：在已经检测出下侧位置的基础上，在y轴值为 [down-200,down-10) 范围内从下往上检测（即反过来），找到的第一个在一定水平范围 [50,400) 内都没有黑色像素的y值就确定为上侧位置。
 - 有样例在x轴像素500处已经到了21位码右侧的字，故水平范围不能采用 [50,500)
- 右侧：在确定了上侧和下侧位置以后，在x轴范围 [400,600] 内从左向右搜索竖直方向上在上下侧范围内都是白色像素的列。考虑到字符之间有空隙，当连续搜到10个全是白色像素的列以后才确定找到的是右侧边界。
 - 把右侧放在最后是因为21位码右侧结束处上方很短的距离内就有别的黑色像素，无法像左侧一样规定一个宽泛的范围搜索，故一定要等到上侧位置确定后才能搜索。

总体来说，需要先确定下侧，才能确定上侧，继而确定右侧，而左侧的位置可以独立确定。

7位码检测

大致流程与21位码检测相同。下侧位置的确定依赖于上侧，右侧则依赖于下侧。把左侧的检测放在上下侧检测之后虽然不是必要的，但可以缩小查看像素的范围。

在实际情况中7位码的大致位置相较21位码更为固定，在给出的固定范围内搜索即能找到，不会出现21位码检测中左侧和下侧里不在范围内的情况，因此代码更为简洁。

(3) 车票序列号分割

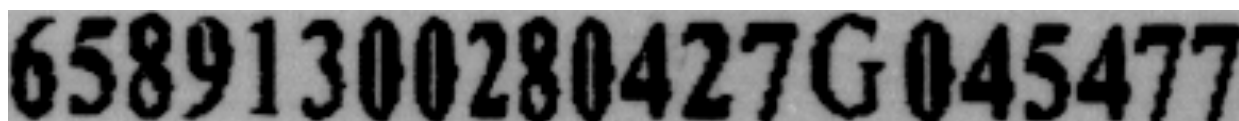
将之前定位到的7位码和21位码分割成单个字符，这部分代码见 `segmentation.py`。

将7位码图像放大后会发现字符周边的像素坑坑洼洼，故对图像采取先dilate再erode，对边缘进行平滑。

（由于是白底黑字，实际在程序里需要先erode再dilate）



21位码不存在上述问题。



字符分割采用投影算法，计算每一列上黑色像素的个数，如果 ≥ 5 则说明这一列上有字符，否则认为是字符间的空隙。同时用变量 `inline` 记录此时处于空白区还是字符区。这样扫遍每一列，可以记录下每个字符的起始点和结束点。再取每个字符的结束点和下一个字符的起始点的中点处作为每一个分割点，即完成分割。

(4) 训练能识别数字字母的神经网络

网络模型见 `model_10.py` 和 `model_62.py`。训练的代码见 `train_10.py` 和 `train_62.py`。测试的代码见 `test_10.py` 和 `test_62.py`。

训练的数据来自the Chars74K dataset:<http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/>。由于车票上的数字字母都是最标准的打印体，故只使用了数据集中电脑合成的数字字母部分。由于是电脑合成的，所以数据集非常规整，62类字符，每一类都有1000张左右的训练数据。

由于字符识别是比较基础的任务，采用了最简单的神经网络模型。`model_10.py` 用于识别10个数字，有两层conv，都采用relu激活函数和2*2max pooling，再接两层fc，最后过一个softmax输出一个10维向量。为了防止过拟合，在第一层fc处进行一次dropout。loss函数采用cross entropy，并用AdamOptimizer进行优化。`model_62.py` 用于识别62个数字+大小写字母，把conv从两层变成了三层。

遇到的问题是计算cross entropy时，由于输出的向量有分量为0，导致表达式中出现了log 0，loss值变为Nan。解决办法是计算log时取向量的分量和1e-3中较大的一个。

`train_10.py` 和 `train_62.py` 中保证了训练被打断后会保存当前graph中的参数，使得下次开始时能够继续之前的训练。

`test_10.py` 和 `test_62.py` 读入在训练时保存的graph及其参数，将要识别的图片传进网络中，得到的就是识别结果。返回时将编号0~61依次转换为相应的数字和字母。注意python中字母之间的转换不能像C++中一样直接加上ASCII码的差值，而是必须要通过ASCII码与字符相互转换的函数，例子如下：`chr(ord('A') + distance)`。

(5) 识别车票序列号中的数字和字母

将（3）中分割出来的单个字符图片传给 `test_10.py` 或 `test_62.py` 中的函数进行测试，代码见 `segmentation.py` 的后半部分。

由于训练数据中数字和字母周围都有一圈白边，而分割出的字符图片中黑色像素可能两边紧挨着边界、上下方离边界有不确定的距离，所以考虑对图片再进行上方和下方的裁剪，得到一个字符紧挨着四个边界的图片，再把它resize到20 * 20，在四周分别添加长度为4的白边，得到大小为28 * 28的图像，送进训练好的神经网络结构里。上方和下方的裁剪由 `crop_piece` 函数实现，分别从图像高度的1/2处向上和向下扫到边界。

在识别的过程中发现只识别数字的模型对序列号中数字的识别十分准确，但用识别数字+字母的模型去识别序列号中的数字则会有很多数字被误识别成字母。由于21位码和7位码都各只有一个字母且位置固定，故把这个字母单独用 `test_62.py` 识别，而让 `test_10.py` 识别剩下的数字。为了提高识别速度，把所有数字字符的图像都concat在一起，作为一个batch送入网络。

实验结果和分析

程序的运行方法及在程序上测试新数据时需要做的改动见 `README.md` 文件。

程序运行的分割结果较好，有90%的图像都达到了预期的分割效果。



被正确分割的图像的数字识别准确率高达95%以上。

尽管21位码的后7位与7位码相同，还是对它们分别进行了识别。7位码里的字母识别的准确率要高于21位码中相应的字母，且更能够将此字符识别为大写字母。

由于程序中的裁剪、分割都是水平竖直方向的，所以对于扫描时略微有倾斜的车票并不能进行倾斜度调整，导致识别的框中有不少冗余的空白。程序也无法应对倾斜度较大的车票。

程序中对字符位置的判断完全由像素是否为黑色决定，因此要求7位码和21位码周围不能有笔的划痕，对干扰不够健壮。

最花时间的部分是序列号定位。由于每张图上7位码和21位码位置都不同，还有一些很靠近页面边界的情况，最开始设想的简单算法有很多情况不能很好地应对，需要不停地根据定位效果不佳的图片来分析问题并改进。而且采用的算法较为笨拙，只是单纯地检测每一行/列是否有黑色像素，没有想到更好的检测方法。