

目录

一、 软件开发背景	1
1. 引言	1
2. 软件主要开发环境	1
3. 软件概要设计	1
4. 软件开发基本流程	2
二、 软件开发过程	3
1. 图像采集	3
2. 图像处理	4
2.1 灰度化、尺寸预处理	4
2.2 二值化	4
2.3 抽取图像骨架	5
2.4 去除干扰线	6
2.5 去噪	7
2.6 字符切割	8
2.7 字符归一化	8
3. 图像识别	10
3.1 素材标记	10
3.2 字符特征提取	11
3.3 训练样本、建立模型	12
3.4 测试、评估模型	12
3.5 特征改进、模型择优	13
4. 开发 API、输出结果并评估	14
4.1 开发 API	14
5. 开发简单图形界面	15
5.1 开发 UI 界面	15
5.2 接入软件 API	16
5.3 测试界面与输出结果	16
三、 软件开发总结	18
1. 后期改进方案	18
1.1 加入字符倾斜度校正	18
1.2 图像切割后进行修复，提高字符完整性	18
1.3 加入英文字母和汉字的识别	18
1.4 选择用更优的特征值，增加数据集数量	18
2. 开发心得	19
四、 参考文献	19

一、 软件开发背景

1. 引言

验证码是网上非常常见的一个事物，充当着很多网站的防火墙功能。尤其是对于经常写爬虫在网上爬取数据的伙伴来说，验证码有时候是一个比较头痛的问题。但是随着 OCR 技术的发展，验证码暴露出来的安全问题也越来越严峻。

本文介绍了一套字符验证码识别的完整流程，利用图像处理的基本知识，对验证码图片采集、处理并建模识别，对于验证码安全和 OCR 识别技术都有一定的借鉴意义。

同时本软件实现的验证码主要针对我校教务处、信息门户等的验证码，对于同学自己写脚本获取教务处的成绩、考试情况等和信息门户预约羽毛球、健身等体育活动，都能提供更好的帮助。

2. 软件主要开发环境

系统：Windows 10 旗舰版
Python IDE：JetBrains PyCharm 2017.1.2
Python 版本：Python 3.6.0 32 位
开源的 svm 机器学习库：Libsvm-3.22
图片处理库：PIL
自动化测试工具：webdriver
Http 库：requests
科学计算包：numpy
Python 访问操作系统模块：os
Python 标准 GUI 工具包：tkinter

3. 软件概要设计

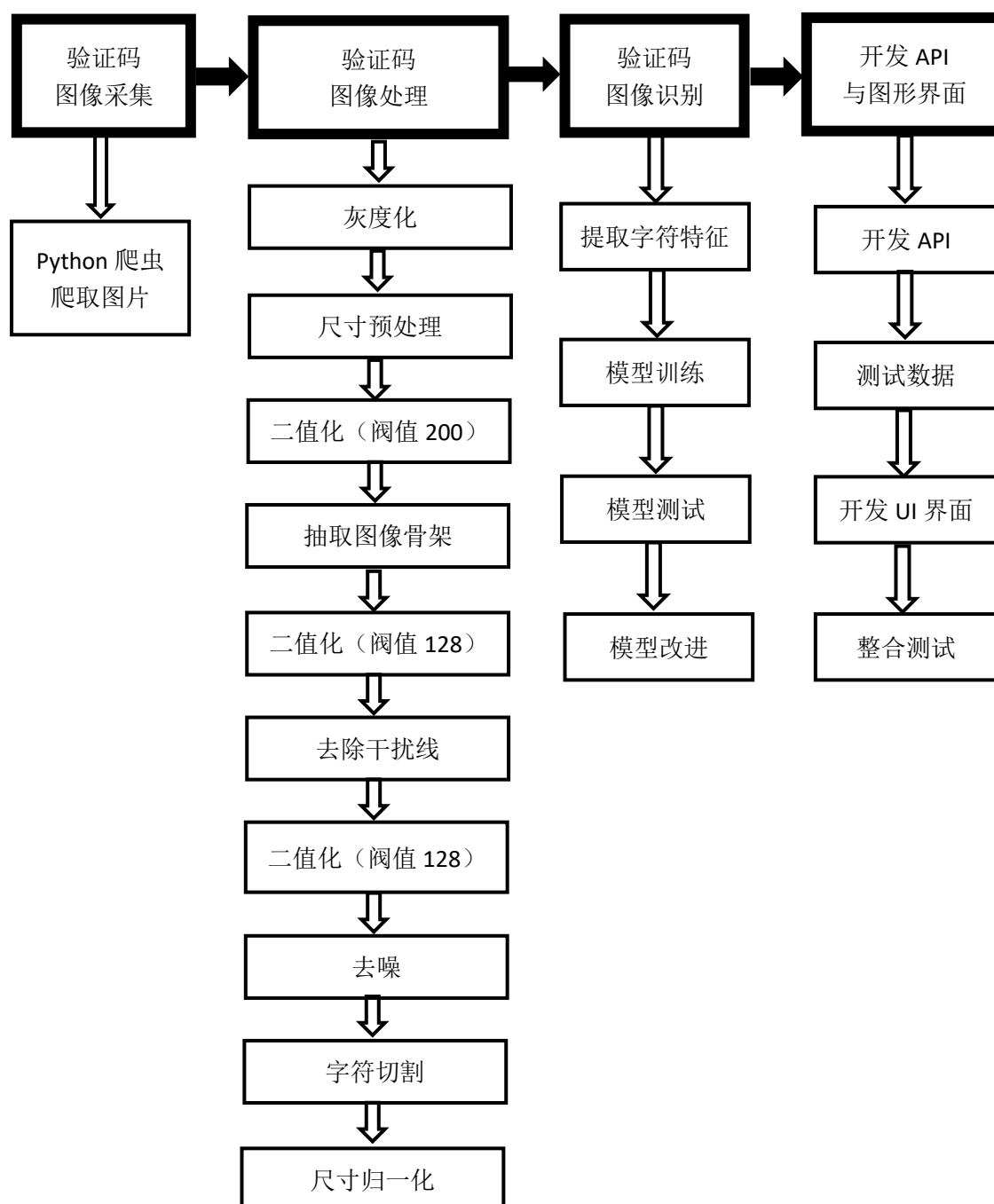
机器识别图片主要的三个步骤为消去背景、切割字符、识别字符，而现有的字符验证码也针对这三个方面来设计强壮的验证码。本软件主要通过验证码图像采集、图像处理、图像识别、开发 API 与 UI 界面四个步骤来实现。

首先从学校教务处和信息门户用 Python 脚本爬取验证码图片约 20000 张，然后对图片进行灰度化、二值化、抽取图像骨架、去除干扰线、去噪、字符切割、归一化等等处理。对处理后的字符，进行人工判断后提取合适的特征值。接下来采用 LibSVM 向量机读入特征值训练模型并测试模型。测试之后，根据测试的吻合度对特征值进行改进，进而优化模型。

接着开发软件的 API 和 UI 界面，然后整合 API 和 UI 界面，并进行最终的整合测试。

4. 软件开发基本流程

开发过程根据以上概要设计，建立识别验证码的逻辑过程，如以下流程图所示：



二、软件开发过程

1. 图像采集

通过 Python 脚本模拟浏览器，访问我校教务处查询成绩登录处和信息门户登录处的网址，然后得到验证码图片的地址。根据分析，教务处和信息门户的验证码都是动态地址，访问同一个链接，每次访问都能从服务器获得不同的验证码，于是对该网址各访问 10000 次，并保存每次访问得到的验证码图片。如下图：



其中，上面一种验证码较为简单，定义为第一类验证码；下面一种验证码较为复杂，定义为第二类验证码。

2. 图像处理

2.1 灰度化、尺寸预处理

由于图像可能为 RGB 图片，所以需先对图像灰度化，以便于后面操作。本次开发利用了 PIL 库的灰度化函数对图像进行灰度化，代码如下：

```
1. image = image.convert('L') # 灰度化
```

对尺寸进行预处理，可以将不同尺寸的图片缩放到一个范围后，以便于对图片作统一的操作。本次开发以 60×20 为基准，对图片进行缩放，为了不改变原图的宽高比，并且是图片缩放后不小于 60×20，缩放倍率取原图宽和高与 60 和 20 的比值中较小的 1 个，代码如下：

```
1. width, height = image.size # 获取图像的宽和高
2. zoom_k = min(width/60, height/20)# 缩放倍数
```

然后采用双线性内插法缩放图像。主要代码如下：

```
1. first_n_pix = (image.getpixel((m,n+1))-
    image.getpixel((m,n))) * float_y + image.getpixel((m,n))#第 1 条线值
2. second_n_pix = (image.getpixel((m+1, n+1)) - image.getpixel((m+1,n))
    ) * float_y + image.getpixel((m+1,n))#第 2 条线对应值
3. image_matrix[y_row][x_column] = int((second_n_pix - first_n_pix) * f
    loat_x + first_n_pix)#双线性内插对应值
```

2.2 二值化

二值化时对灰度级为 8 的图像共 256 个像素值先设定一个阈值，然后建立一个长度为 256 的容器，对于小于阈值的像素，设定为 0，大于等于阈值的设定为 13。遍历图像，以像素值对下标寻找对应的二值并覆盖原像素值。代码如下：

```
1. ### No.4 二值化
2. def binarization(image):
3.     img_grey = image.convert('L') # 灰度化
4.     threshold = 128 # 二值化阈值
5.     table = [] #二值化依据
6.     for i in range(256):
7.         if i < threshold:
8.             table.append(0)
9.         else:
```

```

10.         table.append(1)
11.     img_bin = img_grey.point(table, '1') # 二值化
12.     return img_bin

```

在抽取图像骨架时, 为了更好突出图像轮廓, 阈值设置为 200, 在其余需要二值化时, 阈值设置为 128。

2.3 抽取图像骨架

细化抽取图像骨架时, 根据某点的八个相邻点的情况^[1]来判断该点是否能删除。由正方形的骨架是它的中心点; 圆的骨架是它的圆心, 直线的骨架是它自身, 孤立点的骨架也是自身的原理, 事先做出一张表, 从 0 到 255 共有 256 个元素, 每个元素要么是 0, 要么是 1。我们根据某点 (要处理的黑色点) 的八个相邻点的情况查表, 若表中的元素是 1, 则表示该点可删, 否则保留。

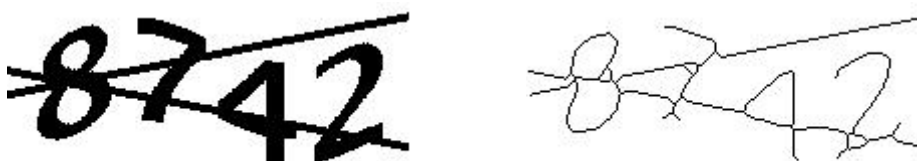
主要代码如下:

```

1. #事先做好的表
2.     array = [0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, \
3.             1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, \
4.             0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, \
5.             1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, \
6.             1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
7.             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
8.             1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, \
9.             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
10.            0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, \
11.            1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, \
12.            0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, \
13.            1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, \
14.            1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
15.            1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, \
16.            1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, \
17.            1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0]

```

图像细化后的效果如下图:



2.4 去除干扰线

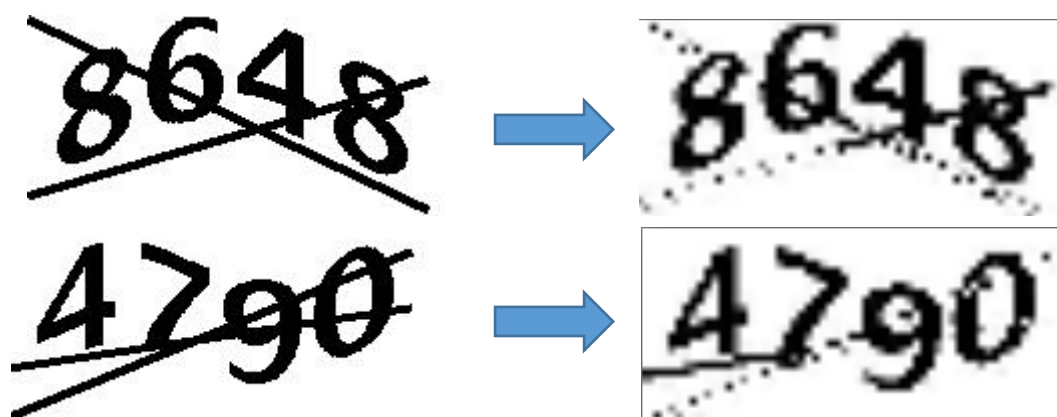
细化图像之后，参考论文《一种通用的去除干扰线的方法》^[2]采用深度优先的递归算法找出图像的干扰线并去除。由于细化后的图像只有单像素宽度，所以最终将找出的干扰线应用于原图像，以使图形有更好的完整度。而随之带来的情况则是可能干扰线宽度很大，一次不能完整去除，所以可以将去除干扰线的原图再次细化，然后找出干扰线并去除，直到不能再找到干扰线为止。同时干扰线一般是与验证码字符交叉的，所以对深度搜索过程中发生分叉的地方(在交叉处才回分叉)记录下来，在去除干扰线时不覆盖这些记录的点。主要函数截图：

```

7      ## 去除干扰线
8      ## 采用深度优先算法
9
10     class RemoveNoiseLine:
11         ##### 初始化
12         def __init__(self, image_refine, image_ori):...
39
40         # 开始搜索
41         def start(self):...
116
117         # 深度优先搜索干扰线
118         def dfsLine(self, cur_node, node_list, line_length,
195
196         ## 得到两个结点之间的方向和角度
197         def getTheta(self, cur_node, nebr_node):...
209
210         # 获得每个节点的邻居节点
211         def getNeighborNodes(self, cur_node):...

```

去除干扰线的效果如下图：



2.5 去噪

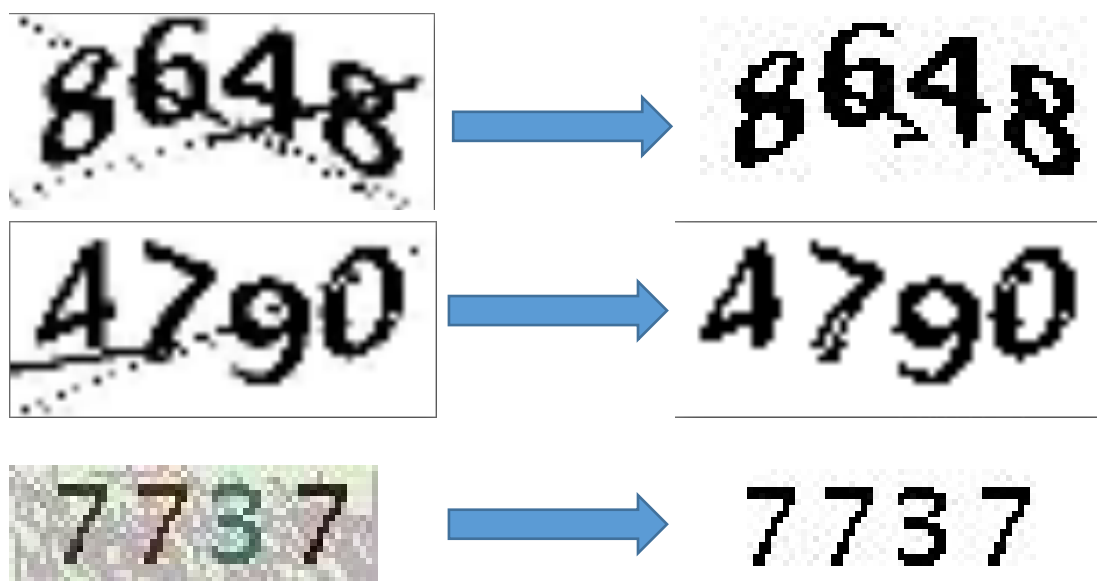
经过去除干扰线之后，剩下的大部分噪点也是最简单的那种孤立点，所以可以通过检测这些孤立点就能移除大量的噪点。此处参考博客《字符型图片验证码识别完整过程及 Python 实现》^[3]决掉这个问题：

- 对某个黑点周边的九宫格里面的黑色点计数
- 如果黑色点少于 2 个则证明此点为孤立点，然后得到所有的孤立点
- 对所有孤立点一次批量移除。

代码如下，其中 findIsolatedPoints 函数返回黑点 8 领域的黑点数。

```
1. ### No.5 去除噪声
2. def removeNoise(image):
3.     width, height = image.size # 获取宽和高
4.
5.     is_once = False #是否继续去噪
6.     for x_column in range(width):
7.         for y_row in range(height):
8.             # 黑点周围黑点数（包括自己）少于 3 的 去掉黑点 设为白色
9.             if 0 < findIsolatedPoints(image, x_column, y_row) < 3:
10.                 image.putpixel((x_column, y_row), 1)
11.                 is_once = True
12.
13.     return is_once, image#返回去噪声的图片
14.
15. # 寻找孤立点 返回黑点 8 领域的黑点数量
16. def findIsolatedPoints(image, x_column, y_row):
```

去除噪声后效果如图：



2.6 字符切割

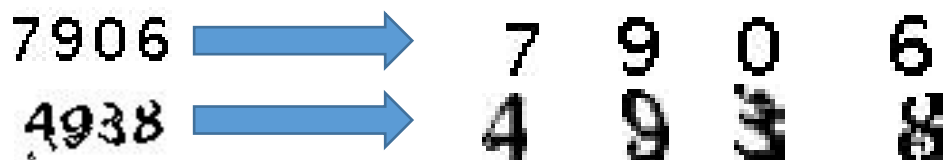
字符切割时,首先对去噪后的图像进行预处理,去除纵向的“毛刺”,使得图像左右光滑且不粘连。然后先竖着扫描,将图像纵向切开;接着横向扫描,去除图像上下的白边,再对切割后的字符作去“毛刺”处理,使得图像上下光滑。主要函数截图如下:

```

7      ### 根据去噪后的二值化图像切割
8      ### 得到验证码的每个字母图像
9
10     +def main():...
39
40     ### 保存验证码切割后的图像
41     +def saveCropImages(img_list, ori_name):...
44
45     ### 去除图像每列黑点小于等于2的噪声
46     +def beforCrop(image):...
61
62     ### 切割后去除周围噪声
63     +def afterCrop(image):...
97
98     ### 切割验证码图像
99     +def getCropImages(image):...

```

切割后的效果如图:



2.7 字符归一化

切割后的字符,由于尺寸大小不统一,实时还不适合用于提取特征。于是对字符的大小作归一化处理,本次开发选择宽×高为 10×16。对于宽高比不为 10×16 的,宽和高作变形缩放。

```

1.  ### 切割后的图像归一化 10 × 16
2.  def imageZoom(image):
3.      image = image.convert('L')
4.      width, height = image.size # 获取图像的宽和高
5.      zoom_kx = width/10# 宽缩放倍数
6.      zoom_ky = height/16# 高缩放倍数

```

```

7.      # 缩放后的长宽
8.      new_w = 10
9.      new_h = 16
10.     # 缩放后的新数组
11.     image_matrix = zeros([new_h, new_w])
12.     for x_column in range(new_w):
13.         for y_row in range(new_h):
14.             # 新图像坐标对应原图像的坐标
15.             x = x_column * zoom_kx
16.             y = y_row * zoom_ky
17.             # 向下取整
18.             m = int(x)
19.             n = int(y)
20.             float_x = x - m # 浮点数 x
21.             float_y = y - n # 浮点数 y
22.             # 边界判断
23.             m = width - 2 if m+1 >= width else m
24.             n = height - 2 if n+1 >= height else n
25.             m = 0 if m < 0 else m
26.             n = 0 if n < 0 else n
27.             first_n_pix = (image.getpixel((m,n+1))-
                image.getpixel((m,n))) * float_y + image.getpixel((m,n))
28.             second_n_pix = (image.getpixel((m+1, n+1)) - image.getpixel
                ((m+1,n))) * float_y + image.getpixel((m+1,n))
29.             image_matrix[y_row][x_column] = int((second_n_pix - first_n
                _pix) * float_x + first_n_pix)
30.     new_data = numpy.reshape(image_matrix, (new_h,new_w))
31.     new_im = Image.fromarray(new_data)
32.     # new_im.show()## 显示图片
33.     new_im = new_im.convert('L')
34.     return new_im

```

归一化后的图像：

8	4	7	5	1	5	2
9963-2.jpg	9963-3.jpg	9963-4.jpg	9964-1.jpg	9964-2.jpg	9964-3.jpg	9964-4.jpg
0	8	6	2	3	6	4
9965-1.jpg	9965-2.jpg	9965-3.jpg	9965-4.jpg	9966-1.jpg	9966-2.jpg	9966-3.jpg
2	8	7	0	0	2	3
9966-4.jpg	9967-1.jpg	9967-2.jpg	9967-3.jpg	9967-4.jpg	9968-1.jpg	9968-2.jpg
5	0	8	5	1	3	4
9968-3.jpg	9968-4.jpg	9969-1.jpg	9969-2.jpg	9969-3.jpg	9969-4.jpg	9970-1.jpg
7	4	0	3	8	7	1
9970-2.jpg	9970-3.jpg	9970-4.jpg	9971-1.jpg	9971-2.jpg	9971-3.jpg	9971-4.jpg
3	9	4	7	2	1	4
9972-1.jpg	9972-2.jpg	9972-3.jpg	9972-4.jpg	9973-1.jpg	9973-2.jpg	9973-3.jpg
3	1	0	6	3	3	2
9973-4.jpg	9974-1.jpg	9974-2.jpg	9974-3.jpg	9974-4.jpg	9975-1.jpg	9975-2.jpg
8	9	9	3	3	7	0
9975-3.jpg	9975-4.jpg	9976-1.jpg	9976-2.jpg	9976-3.jpg	9976-4.jpg	9977-1.jpg

3. 图像识别

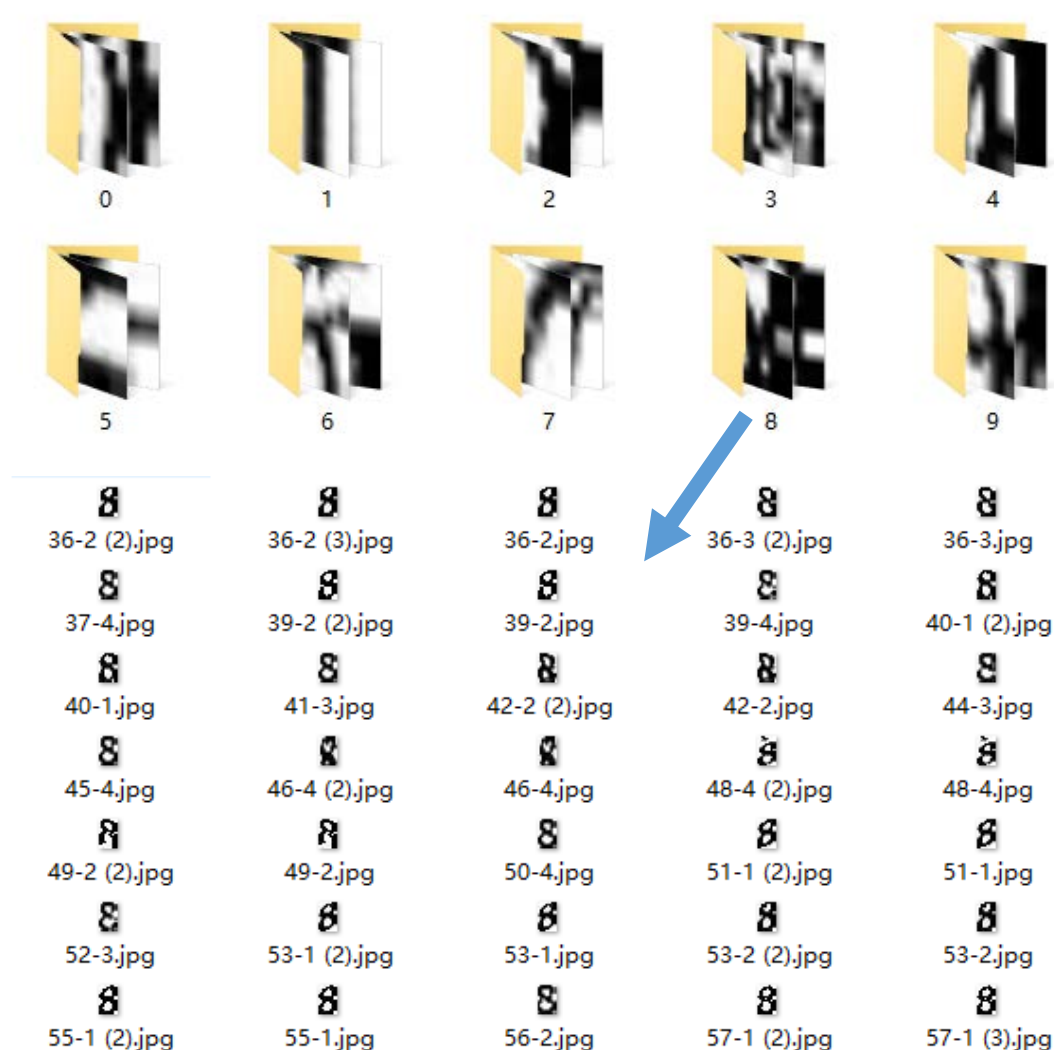
3.1 素材标记

由于本文使用的这种识别方法中，机器在最开始是不具备任何数字的观念的。所以需要人为的对素材进行标识，告诉 机器什么样的图片的内容是 1……。这个过程叫做 “标记”。

具体打标签的方法是：

- 为 0~9 每个数字建立一个目录，目录名称为相应数字（相当于标签）
- 人为判定 图片内容，并将图片拖到指定数字目录中
- 每个目录中存放 5000 张左右的素材

一般情况下，标记的素材越多，那么训练出的模型的分辨能力和预测能力越强。例如本文中，标记素材为十多张的时候，对一类图片识别率基本为零，但是到达 100 张时，则可以达到近乎 100% 的识别率。但是对二类图片的识别率并不是很高，当数量到达上千张时，对二类图片的识别有所改观。



3.2 字符特征提取

从宏观上看，不同的数字图片的本质就是将黑色按照一定规则填充在相应的像素点上，所以这些特征都是最后围绕像素点进行。

字符图片宽 10 个像素，高 16 个像素，于是我先根据博客《字符型图片验证码识别完整过程及 Python 实现》给出一种简单粗暴的特征定义：

- 每行上黑色像素的个数，可以得到 10 个特征
- 每列上黑色像素的个数，可以得到 16 个特征

最后得到 26 维的一组特征，实现代码如下：

```

1. #获取特征值 数量 = 行数 + 列数
2. def getFeatureValue26(image):
3.     width, height = image.size # 获取宽和高
4.     feature_value_list = [] # 特征值列表
5.     ## 第一次遍历 得到每行的特征值
6.     for y_row in range(height):
7.         pix_cnt_x = 0 # 黑点数量
8.         for x_column in range(width):
9.             if image.getpixel((x_column, y_row)) == 0: # 黑色点
10.                 pix_cnt_x += 1
11.             feature_value_list.append(pix_cnt_x)
12.     ## 第二次遍历 得到每列的特征值
13.     for x_column in range(width):
14.         pix_cnt_y = 0 # 黑点数量
15.         for y_row in range(height):
16.             if image.getpixel((x_column, y_row)) == 0: # 黑色点
17.                 pix_cnt_y += 1
18.             feature_value_list.append(pix_cnt_y)
19.
20.     return feature_value_list

```

```

0 1:4 2:7 3:6 4:5 5:4 6:4 7:4 8:4 9:4 10:4 11:4 12:4 13:5 14:9 15:8 16:9 17:8
18:14 19:6 20:5 21:5 22:5 23:6 24:7 25:14 26:15
0 1:4 2:9 3:7 4:5 5:4 6:4 7:4 8:4 9:4 10:4 11:4 12:4 13:5 14:9 15:8 16:9 17:8
18:15 19:7 20:6 21:5 22:5 23:6 24:7 25:14 26:15
0 1:4 2:9 3:9 4:5 5:4 6:4 7:4 8:4 9:4 10:4 11:4 12:4 13:4 14:5 15:6 16:6 17:9
18:15 19:6 20:5 21:3 22:3 23:3 24:6 25:15 26:15
0 1:4 2:6 3:4 4:3 5:4 6:5 7:4 8:4 9:4 10:4 11:3 12:4 13:3 14:5 15:5 16:5 17:2
18:9 19:12 20:4 21:5 22:4 23:5 24:9 25:11 26:6
0 1:4 2:6 3:4 4:5 5:4 6:4 7:5 8:4 9:5 10:5 11:5 12:3 13:4 14:7 15:4 16:4 17:4
18:10 19:13 20:6 21:5 22:5 23:4 24:9 25:10 26:7
0 1:1 2:4 3:4 4:4 5:3 6:5 7:5 8:7 9:6 10:5 11:5 12:5 13:5 14:6 15:6 16:6 17:7
18:11 19:13 20:4 21:4 22:5 23:7 24:13 25:9 26:4
0 1:4 2:6 3:5 4:4 5:4 6:5 7:5 8:4 9:6 10:6 11:5 12:5 13:5 14:3 15:5 16:6 17:5
18:10 19:13 20:6 21:4 22:4 23:4 24:14 25:13 26:5
0 1:4 2:6 3:5 4:3 5:4 6:5 7:4 8:4 9:4 10:4 11:4 12:4 13:4 14:5 15:5 16:5 17:3
18:9 19:12 20:5 21:5 22:5 23:5 24:9 25:11 26:6
0 1:3 2:6 3:4 4:6 5:5 6:5 7:4 8:5 9:4 10:4 11:5 12:4 13:5 14:3 15:5 16:5 17:1
18:10 19:12 20:7 21:4 22:4 23:4 24:11 25:12 26:8
0 1:2 2:3 3:2 4:3 5:3 6:4 7:5 8:5 9:5 10:5 11:5 12:5 13:5 14:5 15:6 16:6 17:6

```

3.3 训练样本、建立模型

向量文件说明如下：

- 第一列是标签列，即此图片人为标记值，后续还有其它数值 1~9 的标记
- 后面是 16 组特征值，冒号前面是索引号，后面是值
- 如果有 1000 张训练图片，那么会产生 1000 行的记录

用到的几个主要函数：

```
1. #读取 LibSVM 格式的训练数据：
2. y, x = svm_read_problem('../AllFile/feature_3.txt')
3.
4. #由训练数据 y,x 创建 svm_problem 对象
5. model = svm_train(y, x, '-q')# -q 为静默模式
6.
7. #将训练好的 svm_model 存储到文件中：
8. svm_save_model('../AllFile/model_file_3', model)
9.
10. #读取存储在文件中的 svm_model：
11. model = svm_load_model('../AllFile/model_file_2') # 加载模型
12.
13. #这个函数不仅是测试用的接口，也是应用状态下进行分类的接口。
14. #需要输入测试标签 y 才能进行预测，因为 y 不影响预测结果可以用 0 向量代替。
15. p_labs, p_acc, p_vals = svm_predict(y, x, model
    [, 'predicting_options'])
16. p_label, p_acc, p_val = svm_predict([0], list, model, '-q') # 预测识别
17.
18. ### 参数说明 ###
19. # y 测试数据的标签
20. # x 测试数据的输入向量
21. # model 为训练好的 SVM 模型。
22. # p_labs 是存储预测标签的列表。
23. # p_acc 存储了预测的精确度，均值和回归的平方相关系数。
24. # p_vals 在指定参数 '-b 1' 时将返回判定系数(判定的可靠程度)。
```

3.4 测试、评估模型

第一次定义的 26 个特征值，训练出来的模型，预测一类验证码准确率能达到 100%，但是二类只有 70% 左右。由于二类验证码的复杂性较大，所以这个模型并不具有普适性。于是想出了另外一种特征定义的方法：

3.5 特征改进、模型择优

- 字符图片宽 10 个像素，高 16 个像素，共计 160 个像素
- 对每个像素值，以其自身与周围 8 邻域的黑点数总和为特征
第一种定义中，需对图像遍历两次，改进后的特征定义只需遍历一次，这次在时间复杂度和预测准确度上都得到了优化。主要代码如下：

```

1. #获取特征值 数量 = 行数 × 列数
2. def getFeatureValue160(image):
3.     width, height = image.size # 获取宽和高
4.     feature_value_list = [] # 特征值列表
5.
6.     ## 遍历图像
7.     for y_row in range(height):
8.         for x_column in range(width):
9.             feature_value_list.append(find0Points(image, x_column,
10.                                                    y_row))
11.
12.     return feature_value_list
13.
14. # 寻找孤立点 返回黑点 8 邻域的黑点数量加自己
15. def find0Points(image, x_column, y_row):
16.     ###函数具体代码详见附件 py 文件

```

然后就将图片素材特征化，按照 libSVM 指定的格式生成一组带特征值和标记值的向量文件。内容示例如下：

```

feature_3.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0 1:2 2:4 3:6 4:5 5:5 6:4 7:3 8:1 9:0 10:0 11:4 12:7 13:8 14:6 15:6 16:6 17:6
18:3 19:1 20:0 21:5 22:8 23:7 24:4 25:3 26:5 27:7 28:5 29:2 30:0 31:6 32:9
33:6 34:3 35:1 36:4 37:7 38:7 39:4 40:1 41:6 42:8 43:5 44:2 45:0 46:3 47:6
48:8 49:5 50:2 51:6 52:7 53:4 54:1 55:0 56:3 57:6 58:9 59:7 60:4 61:6 62:7
63:4 64:1 65:0 66:3 67:6 68:9 69:8 70:5 71:6 72:8 73:5 74:2 75:0 76:2 77:5
78:8 79:9 80:6 81:6 82:9 83:6 84:3 85:0 86:1 87:4 88:7 89:9 90:6 91:6 92:9
93:6 94:3 95:0 96:0 97:3 98:6 99:9 100:6 101:6 102:9 103:6 104:3 105:0 106:1
107:4 108:7 109:9 110:6 111:5 112:8 113:6 114:3 115:0 116:2 117:5 118:8 119:8
120:5 121:4 122:7 123:7 124:4 125:1 126:3 127:6 128:9 129:7 130:4 131:2 132:4
133:6 134:5 135:4 136:5 137:7 138:8 139:5 140:2 141:1 142:2 143:5 144:6 145:7
146:7 147:8 148:7 149:4 150:1 151:0 152:0 153:2 154:4 155:6 156:6 157:6 158:4
159:2 160:0
0 1:2 2:4 3:6 4:5 5:5 6:4 7:3 8:1 9:0 10:0 11:4 12:7 13:8 14:6 15:6 16:6 17:6
18:3 19:1 20:0 21:5 22:8 23:7 24:4 25:3 26:5 27:7 28:5 29:2 30:0 31:6 32:9
33:6 34:3 35:1 36:4 37:7 38:7 39:4 40:1 41:6 42:8 43:5 44:2 45:0 46:3 47:6
48:8 49:5 50:2 51:6 52:7 53:4 54:1 55:0 56:3 57:6 58:9 59:7 60:4 61:6 62:7
63:4 64:1 65:0 66:3 67:6 68:9 69:8 70:5 71:6 72:8 73:5 74:2 75:0 76:2 77:5
78:8 79:9 80:6 81:6 82:9 83:6 84:3 85:0 86:1 87:4 88:7 89:9 90:6 91:6 92:9
93:6 94:3 95:0 96:0 97:3 98:6 99:9 100:6 101:6 102:9 103:6 104:3 105:0 106:1
107:4 108:7 109:9 110:6 111:5 112:8 113:6 114:3 115:0 116:2 117:5 118:8 119:8
120:5 121:4 122:7 123:7 124:4 125:1 126:3 127:6 128:9 129:7 130:4 131:2 132:4
133:6 134:5 135:4 136:5 137:7 138:8 139:5 140:2 141:1 142:2 143:5 144:6 145:7

```

4. 开发 API、输出结果并评估

4.1 开发 API

整合所有步骤的代码，调用每个步骤的接口函数，组成一个对外的开放接口。主要代码如下：

```
1. ## 识别 API
2. def startValidate(image):
3.     ### Step1 图像预处理
4.     res = '' # 存储结果的字符串
5.     image = Step_1_0_Pretreatment.imageZoom(image) # NO.1 NO.2
6.     # NO.3
7.     image_refine = startRefine(image) # 抽取骨架
8.     remove_noise_line = RemoveNoiseLine(image_refine, image)
9.     line_has, image_result = remove_noise_line.start()
10.    image = binarization(image_result) # No.4
11.    is_once, image = removeNoise(image) # No.5
12.    ### Step2 图像切割
13.    image = binarization(image) # 二值化
14.    image = beforCrop(image) #切割前处理
15.    is_once, image = removeNoise(image)#去噪
16.    res_img_list = getCropImages(image) # 切割
17.    same_img_list = []
18.    # 大小归一化
19.    for img in res_img_list:
20.        width, height = img.size # 获取图像的宽和高
21.        # 删除异常图片
22.        if width < 3 or height < 4:
23.            continue
24.        # img.show()
25.        same_img_list.append(imageZoom(img))
26.    ### Step3 提取特征值
27.    for img in same_img_list:
28.        img = Step_3_GetFeatures.binValue(img)
29.        feature_list = getFeatureValue160(img) # 获取特征值
30.    ### Step4 识别字符
31.    model = svm_load_model('../AllFile/model_file_3') # 加载模型
32.    p_label, p_acc, p_val = svm_predict([0], list, model, '-q') # 预测识别
33.    # print(p_label)
34.    res += str(int(p_label[0])) # 添加到结果字符串
35. return res
```

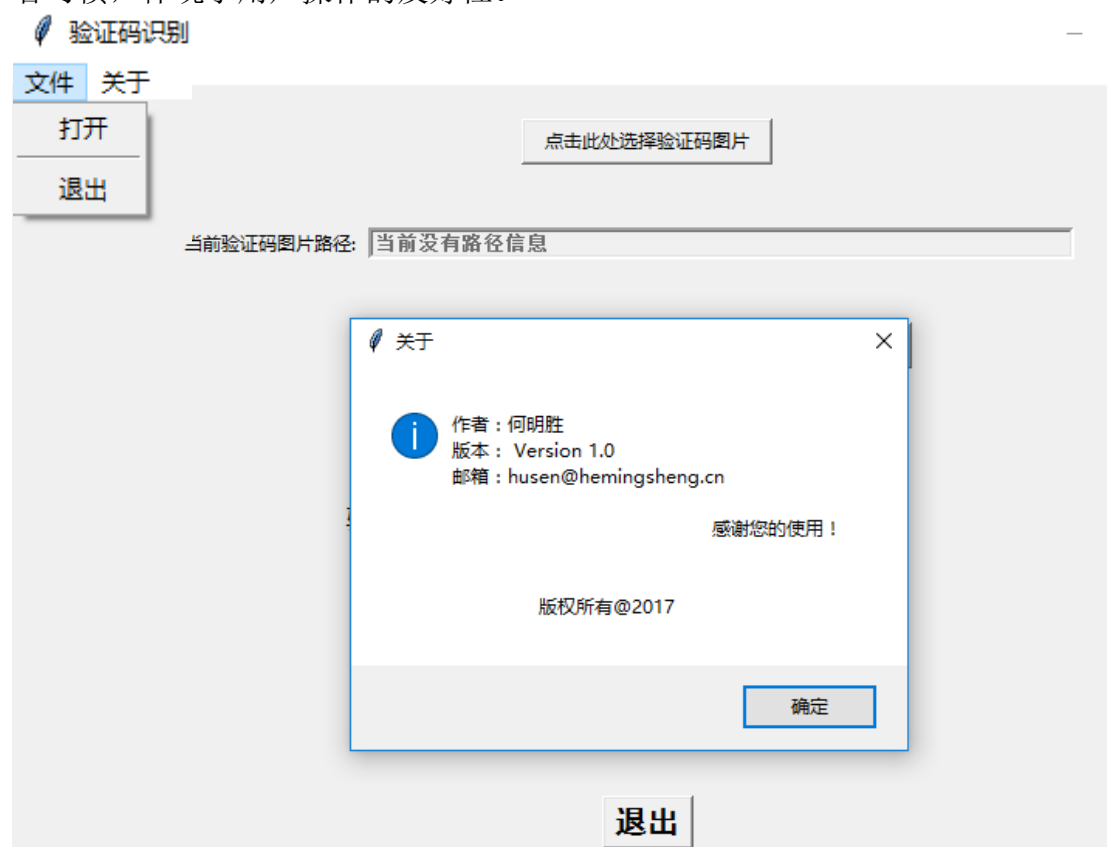

5. 开发简单图形界面

5.1 开发 UI 界面

设计开发简单的图形界面，可以选择电脑上的某一个存有一张或者多张 JPG 或者 PNG 格式的图形，并在 UI 界面显示原图和识别的结果。如果该文件夹有多张图片，可在界面选择上一张和下一张对其他图片进行识别。如下图：



对 UI 界面还有菜单栏，可以选择文件打开图片或者退出，也可点击关于查看软件版本和开发者信息。点击界面下方的退出可退出程序，此操作设计符合使用者习惯，体现了用户操作的友好性。



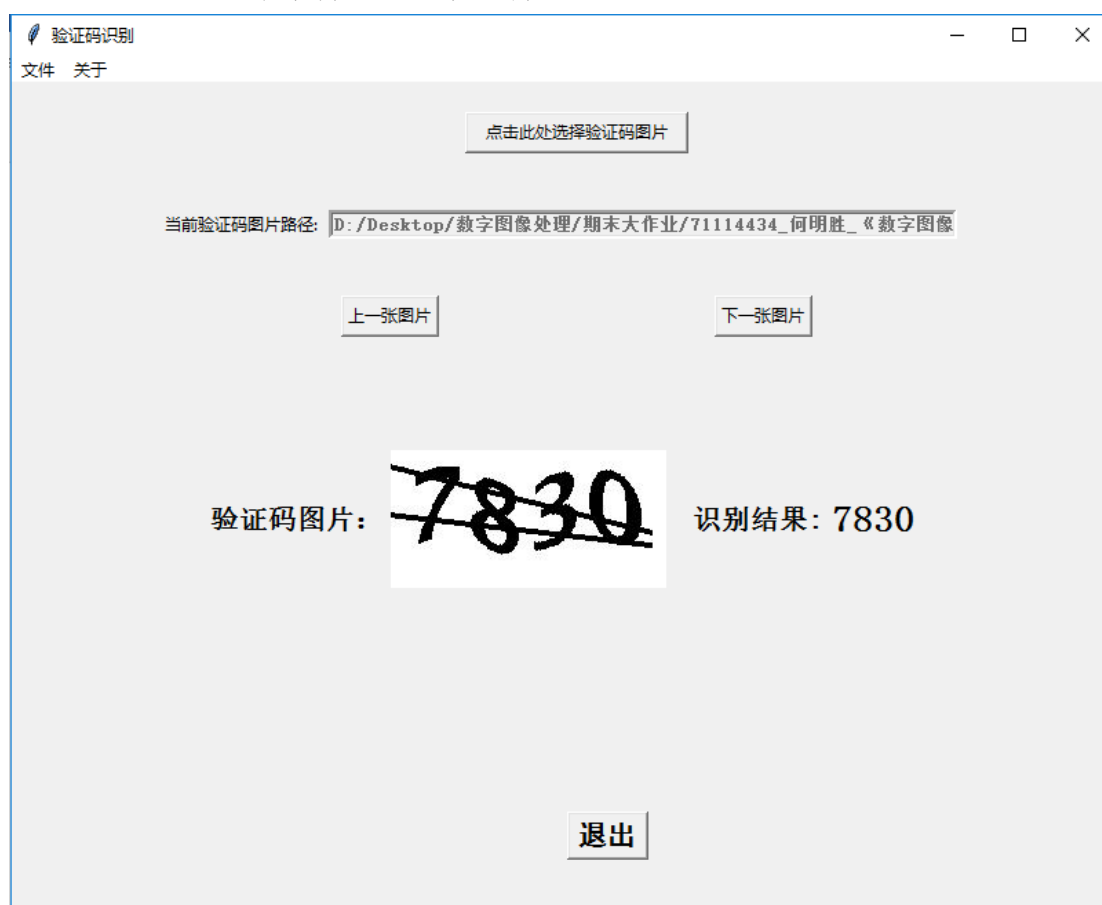
5.2 接入软件 API


UI 界面接入软件 API

```
1. #调用逻辑程序识别
2. ori_image = Image.open(dirname)
3. result = startValidate(ori_image)
4. self.res_txt.set(result)
```

5.3 测试界面与输出结果

在 UI 界面进行最后的开发测试，最终识别准确率接近 100%，如下所示只有一个字符识别失败，将 7 识别成了 2。



验证码图片:  识别结果: 9548

点击此处选择验证码图片

当前验证码图片路径: D:/Desktop/数字图像处理/期末大作业/软件相关/图片集/历史图片及展

上一张图片

下一张图片

验证码图片:  识别结果: 7461

验证码图片:  识别结果: 5323

验证码图片:  识别结果: 3922

验证码图片:  识别结果: 8424

验证码图片:  识别结果: 6704

三、 软件开发总结

1. 后期改进方案

由于时间关系，这次开发还有很多需要完善的地方。后期改进方案主要从以下四个方面进行：

1.1 加入字符倾斜度校正

由于有的验证码字符会有小幅度的倾斜，从而会影响到预测的准确性。在后期可以对每个字符进行一定角度的旋转，其依据为：虽然不知道每个字符旋转了多少度，但是每个字符正常（即不旋转时）一定是最“瘦”的。可根据此原理对字符进行各种角度的旋转，如利用矩阵的知识，最终找到旋转后最“瘦”的一个，以此对倾斜度进行校正。

1.2 图像切割后进行修复，提高字符完整性

在对图像进行预处理时，在去噪和去干扰线的同时，可能会丢失一部分字符原始像素，使得得到的字符不够完整，如出现空缺。在后期可利用连通和像素连续的原理，对预处理后字符进行连通处理，即将孤立的部分连接起来，以提高字符完整性。

1.3 加入英文字母和汉字的识别

本次开发由于时间关系，只识别了数字 0-9，在后期可以增加英文字母和汉字的识别，以增强本软件的识别广度。英文字母以及汉字和数字的识别是大同小异的，只是在预处理和特征值上面，可能需要将目前的函数做一些通用性的扩展，并增强一些其他的算法来优化预处理的过程。

1.4 选择用更优的特征值，增加数据集数量

虽然目前采用的 160 个特征值取得了不错的效果，但是仍旧不能达到 100% 的识别率，一方面是预处理的时候得到的图像不是 100% 的清晰，另一方面特征值的选取和数据集的数量级还不够。所以后期可以考虑选用更优的特征值，同时增加数据集的数量级。

2. 开发心得

通过这次软件开发，对图像处理的知识有了综合的运用。通过自己的摸索，对图像灰度级、RGB 原理、二值化、缩放、切割、去噪、滤波、去除干扰线、图像细化等等都有了更加深刻的理解。对 Python 的使用更加熟悉，特别是一些强大的库。如 PIL、numpy 这些强大的库，使用起来非常方便。通过 LibSVM 的使用，开始接触到了机器学习的一些东西，算是走进了机器学习的大门。

由于没有经验，在开发的过程中组了很多弯路。比如对 python 语言函数参数的引用传递和值传递理解得不够透彻，所以在写去除干扰线的算法时，一直不能得到正确的结果。后来才发现，python 的函数在传递参数时，除了 int 和 string，其余一律为引用传递，所以每次赋值都会改变原始变量的值。

最后本来想将程序文件打包成 exe 文件，但是由于程序利用了一些第三方的库，如 LibSVM、numpy，以及 python 自带的 tkinter 库，导致在使用 cxfreeze 打包程序后，无法正常运行。又由于马上要去实习了，所以这个想法只能后期才实现了。

四、 参考文献

^[1] <http://www.cnblogs.com/slysky/archive/2011/10/16/2214015.html>

^[2] 程治国，刘允才. 一种通用的去除文字图像中干扰线的算法[J]. 上海交通大学学报, 2005, 39(8):1288-1291.

^[3] <http://www.cnblogs.com/beer/p/5672678.html>