

本 科 生 毕 业 设 计（学 术 论 文）



题 目 **基于KNN算法的验证码分类和识别系统**

学 院  **软件学院**

专 业 **软件工程**

学生姓名  **祁旭翔**

学 号  **2012141463033** 年级 **2012**

指导教师  **胡晓勤**

教务处制表

二Ο一六年五月三十一日

**基于KNN算法的验证码分类和识别系统**

软件工程

学生 祁旭翔 指导老师 胡晓勤

**[摘要]** 验证码已经成为当前互联网广泛使用的验证手段，它在我们上网的时候几乎无处不在。验证码识别的研究不仅能让我们更好地理解机器识别图像的原理，也可以及时发现验证码设计的缺陷与不足。

论文从原始图片开始，先进行了灰度化、去噪点、二值化等预处理操作，然后进行图像切割和数据提取，分别用异或算法、KNN算法和BP神经网络算法（可选）进行了训练和识别。在每个步骤中，我们都使用了不同的算法进行对比和分析，然后希望在文章结束的时候能分析出验证码设计的那些细节可以防止验证码被破解，并提出一些关于验证码防破解的建议。

**[主题词]** 验证码识别；机器学习；图像处理；

**The system of classification and recognization of CAPTCHA based on K Nearest Neighbors algorithm**

Software Engineering

Student：Qi Xuxiang Adviser: Hu Xiaoqin

**[Abstract]** CAPTCHA is a important validate method which has been widely used in the Internet,it is almost in everywhere when we surf in the Internet.The research on CAPTCHA can not only enable us to understand how a computer recognize the image , but also can find the defects of the CAPTCHA code. The These begins with the original pictures.Firstly, I do some preprocessing,including gray degree transformation，cleaing the noise and binary . Then I cut the picture and extrat data from characters .Fianlly I use the data to train with the Exclusive-OR algorithm、K Nearest Neighbors algorithm and Backpropagation neural algorithm(selectable) for recognization.I want to analyze the details of design of the CAPTCHA code to find suggestions to prevent the CAPTCHA code being break.

**[Key Words]** break CAPTCHA；machine learning；image processing；.

**目 录**

[1 绪论 1](#_Toc449624737)

[1.1 研究背景 2](#_Toc449624738)

[**1.1.1** **验证码概念** 2](#_Toc449624739)

[**1.1.2** **验证码作用** 2](#_Toc449624740)

[**1.1.3** **验证码验证流程** 2](#_Toc449624741)

[**1.1.4** **机器学习** 3](#_Toc449624742)

[1.2 研究现状 3](#_Toc449624743)

[1.2.1 国外研究现状 3](#_Toc449624744)

[1.2.2 国内研究现状 3](#_Toc449624745)

[1.3 论文主要工作 3](#_Toc449624746)

[1.4 论文组织与结构 3](#_Toc449624747)

[2 验证码识别技术概述 6](#_Toc449624748)

[2.1 数据预处理 6](#_Toc449624749)

[2.2 图片切割 6](#_Toc449624750)

[2.3 识别 6](#_Toc449624751)

[3 图像预处理 7](#_Toc449624752)

[3.1 灰度化 7](#_Toc449624753)

[**3.1.1 分量法** 7](#_Toc449624754)

[**3.1.2 分量法** 7](#_Toc449624755)

[**3.1.3 均值法** 7](#_Toc449624756)

[**3.1.4 加权均值法** 7](#_Toc449624757)

[**3.1.4 对比选择** 8](#_Toc449624758)

[3.2 去噪点 8](#_Toc449624759)

[**3.2.1** **注入填充法** 8](#_Toc449624760)

[**3.2.2** **相邻对比法** 9](#_Toc449624761)

[**3.2.3** **对比选择** 10](#_Toc449624762)

[3.3 二值化 10](#_Toc449624763)

[**3.3.1 猜值法** 10](#_Toc449624764)

[**3.3.2 均值法** 11](#_Toc449624765)

[**3.3.3 直方图统计法** 11](#_Toc449624766)

[**3.3.4 对比选择** 12](#_Toc449624767)

[3.4 本章小结 13](#_Toc449624768)

[4 图像切割 14](#_Toc449624769)

[4.1 概述 14](#_Toc449624770)

[**4.1.1 投影切割法** 14](#_Toc449624771)

[4.1.2 均值分割法 15](#_Toc449624772)

[4.1.3 滴水切割算法 15](#_Toc449624773)

[4.2 本章总结 17](#_Toc449624774)

[5 数据提取和识别 17](#_Toc449624775)

[5.1 数据提取 17](#_Toc449624776)

[5.2 识别思想 17](#_Toc449624777)

[5.3 异或识别算法 17](#_Toc449624778)

[**5.3.1 算法思想和实现** 18](#_Toc449624779)

[**5.3.2 代码测试** 19](#_Toc449624780)

[**5.3.2 算法分析和改进** 20](#_Toc449624781)

[5.4 KNN识别算法 20](#_Toc449624782)

[**5.4.1 算法思想和实现** 20](#_Toc449624783)

[**5.4.2 算法测试** 21](#_Toc449624784)

[**5.4.3 算法改进** 21](#_Toc449624785)

[**5.4.4 结果分析** 24](#_Toc449624786)

[**5.4.5 和BP神经网络算法的对比** 24](#_Toc449624787)

[5.5 本章总结 27](#_Toc449624788)

[6 小结 27](#_Toc449624789)

[6.1 工作总结 27](#_Toc449624790)

[6.2 心得体会 27](#_Toc449624791)

[6.3 进一步的工作 28](#_Toc449624792)

[参考文献 29](#_Toc449624793)

[致 谢 30](#_Toc449624794)

[附录1 需求规格说明书 31](#_Toc449624795)

[**1.** **运行环境规定** 31](#_Toc449624796)

[1.1 运行支持 31](#_Toc449624798)

[1.2 支持软件 31](#_Toc449624799)

[1.3 支持包安装 31](#_Toc449624800)

# 绪论

**Python语言：**

Python语言不仅简洁优雅，还有大量优秀的第三方支持库，这使得它成为了图像处理、机器学习和科学统计领域里大家乐意选择的一门编程语言。

**研究技术概述：**

验证码识别涉及到的核心技术就是图像处理和机器学习。

**图像预处理：**

Python Image Library(PIL)是python里面带的一个图形处理库，功能比较强大，是我们图像处理的首选。图像预处理包括图像灰度化，二值化和去噪点。

将彩色图像转化成为灰度图像的过程成为图像的灰度化处理,彩色图像中的每个像素的颜色有R、G、B三个分量决定，而每个分量有255中值可取，这样一个像素点可以有1600多万（255\*255\*255）的颜色的变化范围。[1]而灰度图像是R、G、B三个分量相同的一种特殊的彩色图像，其一个像素点的变化范围为255种，所以在数字图像处理种一般先将各种格式的图像转变成灰度图像以使后续的图像的计算量变得少一些，灰度图像的描述与彩色图像一样仍然反映了整幅图像的整体和局部的色度和亮度等级的分布和特征。[2]

二值化是图像分割的一种方法。在二值化图象的时候把大于某个临界灰度值的像素灰度设为灰度极大值，把小于这个值的像素灰度设为灰度极小值，从而实现二值化。

现实中的数字图像在数字化和传输过程中常受到成像设备与外部环境噪声干扰等影响，称为含噪图像或噪声图像。减少数字图像中噪声的过程称为图像去噪。

一般验证码为了防止破解会加入各种噪点。而我们去噪点则是为了提取出验证码图片中所含的有用信息，利于我们后面切割字符。去噪之后，会有效提高图片的质量，更好的体现原来图像所携带的信息，所以去噪点操作是一种重要的图片预处理手段。

**机器学习：**

欧氏距离：在数学中，欧几里得距离或欧几里得度量是欧几里得空间中两点间“普通”（即直线）距离。

在欧几里得空间中，点x =(x1,...,xn)和 y =(y1,...,yn)之间的欧氏距离为：

 (1.1)

K最近邻（k-Nearest Neighbor，KNN）分类算法可以说是最简单的机器学习算法。其基础思想是在同一维度内欧氏距离越小的两个点相似度越高。所以相似度最高的K个点就在同一类别里面。它采用测量不同特征值之间的距离方法进行分类。它的思想很简单：如果一个样本在特征空间中的K个最相似（即特征空间中最邻近）的样本中的大多数属于某一个类别，则该样本也属于这个类别。一般而言，欧氏距离越小，二者相似度越高。

BP(Back Propagation)神经网络是使用最为广泛的一种神经网络，全称为基于误差反向传播算法的人工神经网络。它具有分层结构，每层包含多个节点，节点输入来自前一层的输出，而节点的输出则作为后一层的输入，同层节点之间不相连。对于输入层，其输入数据就是输出数据；对于隐含层与输出层，输入数据之和经过激活函数，得到的结果就是该节点的输出。

BP神经网络算法更像人脑工作的方式：它会不断更新权值以使输出结果与期望结果之间的误差变得越来越小，这个算法相比于KNN算法，更加先进。正向传播学习，逆向传播误差，在经过一系列的学习之后，BP神经网络算法的识别结果会变得很准确。

## 研究背景

* + 1. **验证码识别过程**

抛却维基百科对验证码识别的概念不谈，我尝试从人脑理解和视觉的角度来理解验证码识别。不得不说，人脑是世界上最神奇的最神奇的系统之一。考虑我们人类对下面这张验证码的识别过程。

**1**

一般人能毫不费力地把这个验证码识别为”xl4b1”。这是因为我们从小被教育，我们的大脑也一直在学习。在经过一次次错误之后，我们开始归纳出了字符的规律，从一个字母、数字都不会到秒识别验证码。在我们看到图像到意识到这个验证码是”xl4b1”这么短时间内，我们大脑里面发生的要比计算机处理图像的过程复杂得多。简单描述就是：大脑去除掉干扰，提取了该验证码图片的有效信息，再对比这些字符和我们之前学过的字母数字，匹配度最高的就是识别出来的结果！

我们每个人都自身携带一台无与伦比的“超级计算机” ——大脑，这台“超级计算机”进化了数百万年，然后还在一直进化。比起机器处理图像，我们人类具有更让人不可思议的能力：那就是能马上理解看到的东西是什么。

* + 1. **机器学习**

上面简要叙述了人脑识别验证码的过程，我们再尝试用计算机来识别验证码，顺便说下我对机器学习概念的认识和理解。人识别验证码易如反掌，可让计算机做同样的事情却稍微有点难。假设我写个计算机程序来识别第四个字符“B”，我可能建个模型：字符是字母，且上面有个环形，下面也有个环形，左边有一条垂线，两个环形基本一样大。。。。。。大概归纳特点就这些，我们很容易能用语言描述这些特点，却难以用算法表达出来，更别说写程序实现了。当然还有小写字母’b‘,还有各种不规则的从未见过的‘b’。。。。。。如此一来，计算机识别验证码似乎成了一件极其困难的事情。

但是机器学习却用另外的方法解决了“计算机识别验证码”这一难题，先用一系列的切割字符作为训练用例，然后组建一个能从这些训练用例中学习的系统。在这个系统里面，通过不断的学习，计算机可以逐渐掌握一些规律，从而学会识别验证码。换言之，机器学习就是能利用这些用例来自动学习和分类的学习机制。这里面当然涉及到概率统计、微积分、极限论等诸多数学知识。机器学习绝对不仅限于图像处理和验证码识别，还有决策、分类和预测等各种例子。

* + 1. **验证码作用**

验证码作为一种辅助安全手段在计算机世界中几乎无处不在：无论是防止暴力破解，还是减轻服务器的压力，亦或是在论坛中防止恶意垃圾消息。其终极作用就是为了区分用户是人还是机器（程序）。其作用主要有下：

1.防止密码被暴力破解：QQ、支付宝、邮箱等含有较多私人信息的软件或app，在用户尝试登陆失败多次后会要求输入验证码，这是检测到了登陆过程可能是在暴力破解密码；

2.减轻服务器的压力：有些电商网站在访问量极大的情况下，为了减轻服务器的压力，会在用户登录时设置验证码；

3.防止恶意垃圾消息：论坛网站或者留言系统为了防止恶意发布垃圾信息，也会设置验证码。

4.防止恶意注册：黑客会用恶意注册方式攻击网站，验证码设置会有效防止这一行为。

* + 1. **验证码验证流程**

当用户输入验证码之后，将用户名、密码和验证码等信息提交到服务器，服务器端获取用户的提交之后，判断用户提交的验证码字符与服务器端保存的字符是否相同。如果相同，则通过对用户提交信息的验证；否则将提示没有通过验证的提示信息。

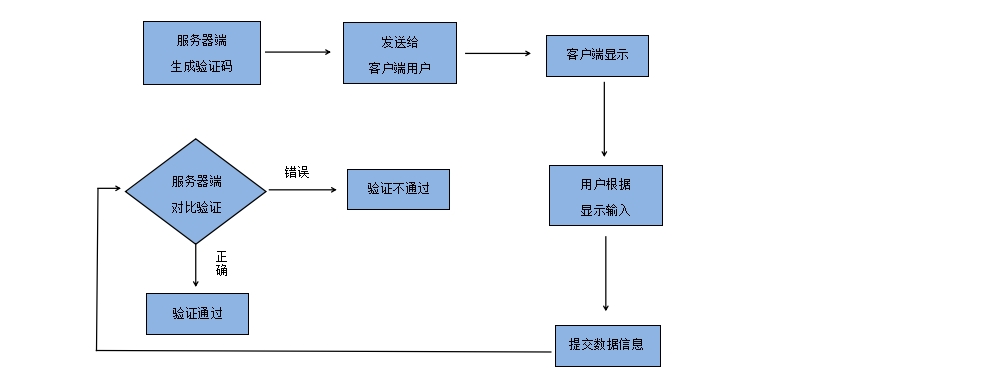


图1.1.1 验证码验证流程图

## 研究现状

### 国外研究现状

在国外，验证码识别起步较早，验证码识别技术也相对比较完善。Mori和Malik通过使用复杂对象识别算法已经成功的识别了EZ-Gimpy 和Gimpy两种相似类型的验证码，其中对EZ-Gimpy验证码的识别率高达92%，对Gimpy验证码的识别率为33%，Chellapilla和Patrice通过机器学习方法破解了几个从网站上获取的验证码，识别率从4.89%到66.2%不等。[6]

### 国内研究现状

在国内，2007年张淑雅通过使用近邻法，神经网络学习等相关知识对SMTH-BBS验证码进行了分割识别，识别率率达到95%以上。[7]

## 论文主要工作

本论文主要研究验证码识别，文章从最原始的验证码图片开始，经过预处理（灰度化、二值化和去噪点）阶段，然后把图片进行切割，把切割后的图片信息提取成数字信息，再用相应的算法对计算机进行训练，尝试进行识别。识别过程分别利用了异或算法、KNN算法和BP神经网络算法。

本论文实现了以上所述的识别过程所有步骤，并在实际过程中对部分算法的提出了改进和延伸。最后根据研究的结果，总结出对验证码识别影响最大的因素，从而提出一种防破解性好的验证码设计方案。

## 论文组织与结构

第一部分：绪论。主要阐述了本文的研究背景。包括验证码的概念、验证原理和作用等。顺便说明了机器学习的概念，其中概念性的东西都来自于维基百科的词条。

第二部分：验证码识别的技术概要和总结，说明了本论文各项的基本流程。

第三部分：介绍了图像的预处理过程。这步作为起始步骤，包括了图像灰度化、去噪点和二值化处理，是为后面的切割字符做准备。

第四部分：进行图像切割。该步骤用了三种不同的切割算法，对比了它们的优劣，我们都要在尽可能保留原图片信息，且不能掺加干扰因素。

第五部分：数据提取和识别。这里我们把图像提取为计算机可以识别的数字信息，然后用异或算法和KNN算法对计算机进行训练。引出了BP神经网络算法，简单实现了一个BP神经网络算法，并用上面提取的数据进行训练和测试。

第六部分：工作总结和结论的整理。

论文结构图如下：

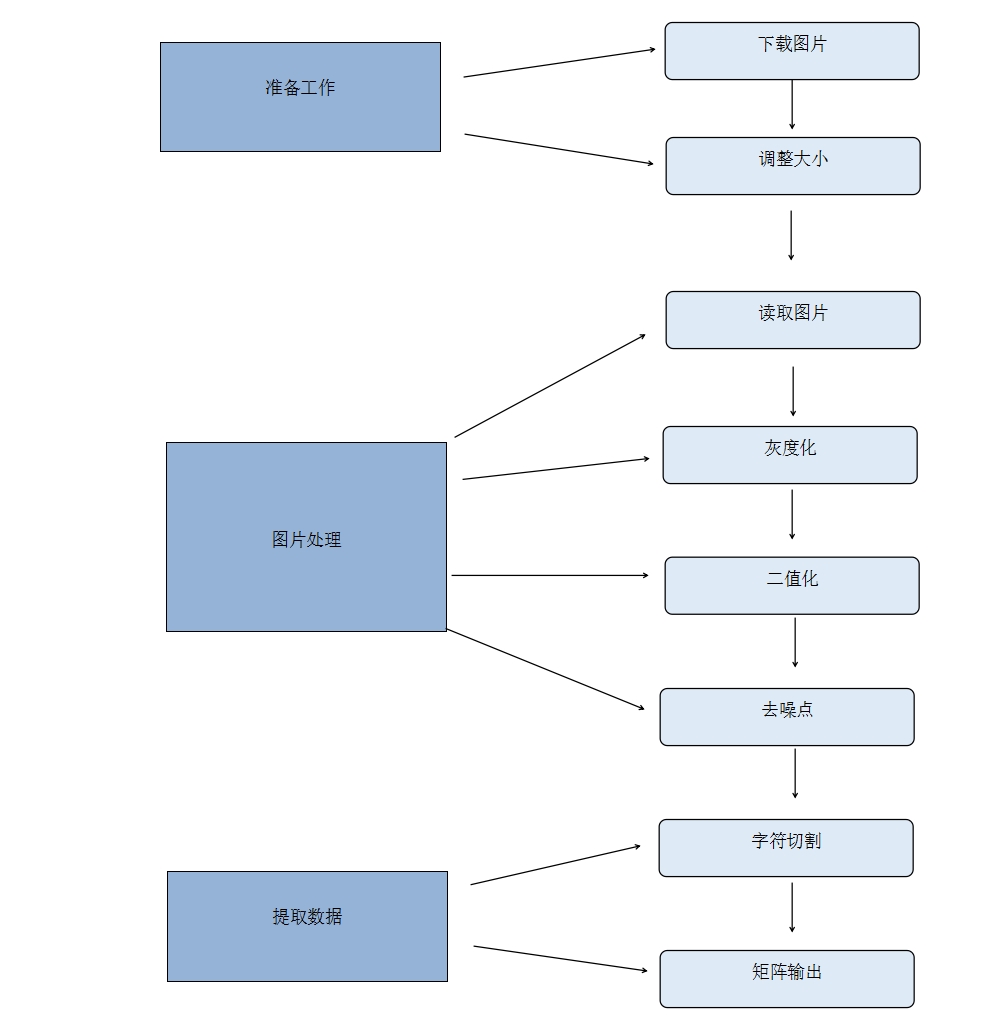


图1.4.1流程图1

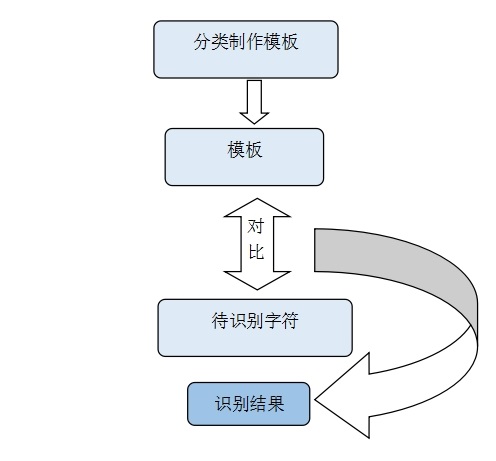


图1.4.2流程图2

# 验证码识别技术概述

验证码识别技术其实是计算机识别图像技术的一个延伸：其核心思想是经过一系列的过程把图片中所含的信息转化为计算机所能识别的数字信息。然后再根据相应的机器学习算法训练让计算机进行识别。

## python图像处理库

python PIL是一个极其成熟的图像处理库，打开Python\Lib\site-packages\PIL文件夹下，我们看到180个python文件，当然绝大多数是关于图像的操作，每个python文件里面又有不止一个方法，所以几乎可以说：这个库里面几乎包含了你能想到的所有图片操作的方法。

**2.1.1 python PIL库的安装**

官方PIL库只支持python 2.x版本，尚不支持python 3.x版本。但是有一堆狂热的python粉丝还是在github上放出了能支持python 3.x的图形库，叫做pillow。当然，可以理解为python 3.x的PIL。Pillow项目git地址:https://github.com/python-pillow/Pillow,官网地址是http://python-pillow.org/。

下载后安装包是.whl文件，可以同pip来安装这个可执行包文件，安装方法在附录中有介绍。

**2.1.2 PIL常见操作图片方法**

来回忆下常见的图片操作：

导入包操作是from PIL import Image

1. 打开图片：img = Image.open(‘test.png’)
2. 灰度化：img=Image.open(‘test.png’).convert(‘L’)
3. 保存图片：img.save(‘test01.png’) #保存为test01.png
4. 调整尺寸：img=Image.open(‘test.png’).resize(140,140)
5. 创建缩略图:img.thumbnail((128,128)) #创建长度为128的缩略图
6. 旋转图片：img.rotate(45) #逆时针旋转45度
7. 打印图片格式、模式和尺寸：print(img.mode,img.mode,img.size)
8. 打印图片下所有信息：print(img.info)

**2.1.2 PIL进阶操作**

## 数据预处理

常见的验证码是彩色图片，里面所含信息维度数较高，直接提取数据会很麻烦，所以首先是一个信息降维的过程，我们称之为灰度化，这会把验证码图片从彩色图转化为灰度图。

灰度化之后验证码图片还有些无用的噪点信息，这些无用信息会对后面的识别产生影响。噪点一般是很小的信息块，在去噪点时只需要用相应的算法找出这些主干信息之外的“小信息块”即可，只是在去噪点的时候也要注意不能把主干像素点去掉。

因为电脑识别的数据是“0”和“1”,所以去噪点之后的灰度图提取信息依然无法满足计算机输入的需求。所以还需要继续对图片进行降维，该步骤称之为“二值化”。二值化操作之后整个验证码图片就只剩下字符有效信息的那块了，而且还是纯黑白图片。

## 图片切割

切割时我们需要把图片原始的信息完整切割下来，当然还不能掺加入干扰信息，这样才能保证后面识别的准确性。对于比较简单且规范的验证码，可以直接遍历像素点就能得到较好的分割效果。但对于有粘连的字符、倾斜等非规范的验证码字符并不可行。

切割之后就把每个字符中所含的数据提取出来以备后面识别之用。提取数据的原则很简单：遍历切割后的字符，如果有像素点则标记值为1，反之值为0。

## 识别

提取数据完毕后我们开始识别，识别算法中最容易想到的是挨个遍历对比，相似度最高的显然异或结果最小。这种算法最简单、复杂度最高，但识别效果最好。

另外比较好的识别算法是机器学习，包括KNN算法和BP神经网络算法。前者是计算二者欧氏距离来比较相似度，而后者是通过不断学习调整最后输出可能的识别结果。

验证码识别本身是一个比较大的话题，不同网站的验证码有不同的特征和形态，同一网站的验证码也会呈现不同的特征。鉴于此，本论文以csdn网站找回密码网站（http://passport.csdn.net/account/fpwd?action=forgotpassword）中的验证码为例子，展开了讨论和研究。

# 图像预处理

图像预处理包括图像灰度化、二值化和去噪点操作，我们希望预处理能够为接下来的切割提供一个信息较为完整的图片。

## 灰度化

将彩色图像转化成为灰度图像的过程成为图像的灰度化处理。彩色图像中的每个像素的颜色有R、G、B三个分量决定，而每个分量有255中值可取，这样一个像素点可以有1600多万（255\*255\*255）的颜色的变化范围，而灰度图像是R、G、B三个分量相同的一种特殊的彩色图像，其一个像素点的变化范围为255种，所以在数字图像处理种一般先将各种格式的图像转变成灰度图像以使后续的图像的计算量变得少一些。[8]灰度图像的描述与彩色图像一样仍然反映了整幅图像的整体和局部的色度和亮度等级的分布和特征。[9]

常见的图像灰度化方法有四种：分量法、最大值法、均值法和加权均值法。

**3.1.1 分量法**

将彩色图像中的三分量的亮度作为三个灰度图像的灰度值，可根据应用需要选取一种灰度图像。

F1(i,j) = R(i,j) R分量灰度 (3.1)

F2(i,j) = G(i,j) G分量灰度 (3.2)

F3(i,j) = B(i,j) B分量灰度 (3.3)

**3.1.2 分量法**

将彩色图像中的三分量亮度的最大值作为灰度图的灰度值。

F(i,j) = max(R(i,j), G(i,j), B(i,j)) (3.4)

**3.1.3 均值法**

将彩色图像中的三分量亮度求平均得到一个灰度值。

F(i,j) = (R(i,j) + G(i,j) + B(i,j)) / 3 (3.5)

**3.1.4 加权均值法**

根据重要性及其它指标，将三个分量以不同的权值进行加权平均。由于人眼对绿色的敏感最高，对蓝色敏感最低，因此，按下式对RGB三分量进行加权平均能得到较合理的灰度图像。[10]

F(i,j) = 0.30R(i,j) + 0.59G(i,j) + 0.11B(i,j)) (3.6)

**3.1.4 对比选择**

以上四种灰度算法中，加权平均法对图像信息的完整度保留最好，因此本项目中采用该方法进行灰度化操作。在python 的PIL库中查看时得知convert(‘L’)就是对加权平均值方法。代码演示如下：

from PIL import Image #导入PIL库

img = Image.open(‘picture.jpg’).convert('L') #将picture.jpg进行灰度化

img.save(‘gray.jpg’) #灰度化后的文件保存为gray.jpg

效果如下所示：



图3.1.1 原图（示例图） 图3.1.2灰度图



图3.1.3 原图 图3.1.4 灰度图

## 去噪点

噪点是图片中对主干信息产生干扰的像素点，也可以理解成无用信息点。一般验证码为了防止破解会加入各种噪点。而我们去噪点则是为了提取出验证码图片中所含的有用信息，利于我们后面切割字符。去噪之后，会有效提高图片的质量，更好的体现原来图像所携带的信息，所以去噪点操作是一种重要的图片预处理手段。

去噪点常见算法有注入填充算法和相邻对比法。

**3.2.1 注入填充法**

介绍注入填充算法之前我们先回忆下word或Windows中的画图工具里面的“油漆桶”功能：在一个闭合的区域里面我们使用了该功能之后整个区域都会被染成同一种颜色。注入填充法的思想和“油漆桶”功能很相似：从一个点开始，在指定的区域内内的某种指定颜色的点都替换成另一种颜色，从而实现填充效果。这个方法一般用于计算一个图的极大连通子图。

设想一个无向图，我们从这个图中一个未标号（“标号”可以理解为“染色”）的结点开始，将此结点和从这个结点出发可达的所有结点都赋予相同的标号（染上相同的颜色），那么，我们就得到了这些被标号的结点所组成的一个极大连通子图，搜索下一个未标号的结点并重复上述过程我们便可以找到所有的极大连通子图。“染色”的过程可以用深度优先搜索或者广度优先搜索实现，如果结点数为V，边数为E，因为我们在Flood Fill过程中“造访”每个结点两次，“造访”每条边两次，所以得到所有极大连通子图的时间复杂度为。[11]

给出一张未知连通性的图，如下图：

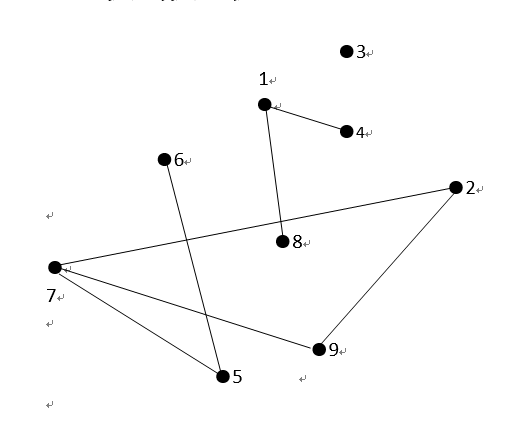


图3.2.1 未知连通图

该图的极大连通子图是：{1,4,8}、{2,5,6,7,9} 和 {3}。

使用该算法的去噪做法是：先用填充算法计算每块的面积,然后把面积最小的块当作噪点去除。

**3.2.2 相邻对比法**

在实际验证码中，噪点一般集中分布在主干图像的边缘位置。所以去噪点使用合理的算法就可以达到除去干扰并保留主干信息的效果。常用的两种去噪点算法是色差检测法和像素点检测法。色差检测法主要思想是：噪点是被主干信息孤立的点，所以噪点相对于主干信息的点，和周围相邻点的色差更大，毕竟含主干信息的点和相邻点的色差几乎为0，因此只要有个点周围有不止6个点色差都大于某个值，我们就认为该点是噪点。

像素点检测法思想更简单：遍历某个点周围的像素点，如果该点周围的像素点个数小于8，就认为该点是噪点，可以全部删除。

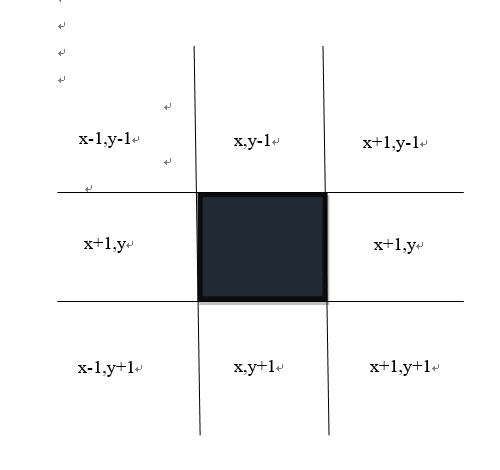


图3.2.2相邻对比法

**3.2.3 对比选择**

虽然注入填充法在实际表现中去噪点的效果也比较好，但由于该算法是遍历计算相邻点计算面积，故算法复杂度高于相邻对比法。且在实际操作中体积参数设小了，噪点没去干净；设大了，数字部分可能也去了一小块。最重要的是这里噪点的大小不是很规律,很难找到一个不错的面积参数。

综上两个原因，我在去噪点操作中舍弃了注入填充法，选择用了代码更简洁、算法复杂度更低而且效果更好的相邻对比法。

## 二值化

二值化会把图像中包含的有用信息提取出来。在本实验中为了让计算机识别图像，将验证码图片转化为黑白图像。

在二值化图象的时候把大于某个临界灰度值的像素灰度设为灰度极大值，把小于这个值的像素灰度设为灰度极小值，从而实现二值化。所以二值化核心问题就是找出临界灰度值。灰度化阈值查找算法一般分为猜值法、均值法、直方图统计法和最大类间方差法。

**3.3.1 猜值法**

猜值法是基于经验设置阈值，是最简单的二值化方法。一般把127（2^7-1）作为阈值，具体做法是：扫描图像的每个像素值，值小于127的将像素值设为0(黑色)，值大于等于127的像素值设为255(白色)。该方法的特点是处理简单，因为不需要考虑图像本身的信息分布。但反而是越简单的处理方式越会造成麻烦。这种方法的坏处在于很可能会把我们需要的信息分离殆尽，导致我们后面识别时根本无法找到有用的信息。

猜值法将灰度阈值设为127。

**3.3.2 均值法**

均值法首先计算像素的平均值K，扫描图像的每个像素值，如果像素值大于K，则把像素值设为255(白色)，值小于等于K像素值设为0(黑色)。[12]相比于之前的猜值法，这里进行了遍历计算，稍微科学一点。当然该方法有点简单粗暴，是因为可能会导致图片中主干信息的流失，这会间接影响到我们后面识别的准确率。

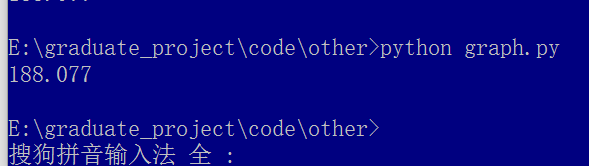


图3.3.1相邻对比法

如图所示，用均值法求出的阈值是188。

**3.3.3 直方图统计法**

图像的直方图用来表征该图像像素值的分布情况。用一定数目的小区间来指定表征像素值的范围,每个小区间会得到落入该小区间表示范围的像素数目。

直方图是用来寻找灰度图像二值化阈值常用而且是有效的手段之一，如果一幅灰度图像的直方图显示为两个波峰，则二值化阈值应该是这两个波峰之间的某个灰度值，一般可以求出两个波峰的均值作为阈值。

python可用matplotlib包下的hist()函数直接绘制直方图。

绘制直方图代码如下：

from PIL import Image

from pylab import \*

# 读取图像到数组中，并灰度化

im = array(Image.open('./source/test.jpg').convert('L'))

# 直方图图像

hist(im.flatten(),128)

# 显示

show()

下面用一张示例图来展示其直方图：



图3.3.2示例图

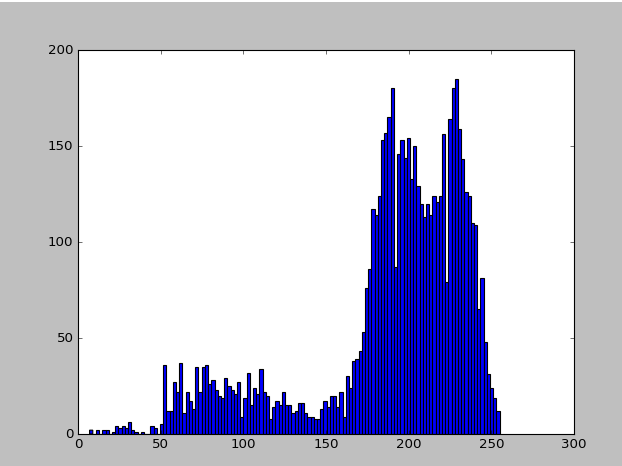


图3.3.3直方图统计法

在上图中我们可以看到两个波峰分别为230和190，即阈值是210。

**3.3.4 对比选择**

用上述四种方法求得的二值化阈值如下表：

|  |  |  |
| --- | --- | --- |
| 猜值法 | 均值法 | 直方图统计法 |
| 127 | 188 | 210 |

表3.3.4

二值化步骤不像上面的去噪点操作，需要考虑算法复杂度。我们更加重视使用该阈值后图片的处理效果。所以我们分别把三个阈值（127、188和210）带入进行二值化操作，选取最合适的阈值。

二值化操作代码如下：

def binary(im):

for x in range(im.width):

for y in range(im.height):

pixels[x,y] = 255 if pixels[x,y]>阈值 else 0

在实际测试中，四个阈值效果都差不多，在实际项目中我把阈值设为200，是为了凑个整数。二值化前后效果如下：



图3.3.4灰度图 图3.3.5二值化图

至此，我们终于把所有的干扰信息排除掉，图片预处理阶段正式结束，接下来是准备切割字符。

## 本章小结

在本章中，我着重研究了图像的灰度化、去噪点和二值化操作及其算法的实现。对python处理图像的方式也有了比较升入的了解。比起C++和java更加深入底层的代码实现，python有得天独厚的优势：强大的封装PIL库让我更加专注于算法本身和操作过程而非底层实现细节，简洁的语法也使得代码行数锐减。

灰度化、去噪点和二值化操作也让我熟悉了python语言中常见的图片操作，对图像的理解更加深入。三个操作将图片的信息成功降维下来，使得图片中所含的主干信息被完全保留下来，而且只剩下黑色像素点，这对后面的切割和提取数据极其有利。接下来就要对处理后的图片进行切割，使用不同的切割算法来将图片的有用信息分割开来。当然分割的时候必须保证图像信息的完整性和有效性：既不能添加入可能产生干扰的信息，也要切割出字符。

# 图像切割

## 概述

在图像灰度化、去噪点和二值化之后，我们需要对验证码图片进行分割，此处必须保证切割后图像信息的完整性。图像切割需要留意的地方在于要保证切割后的信息完整性，同时不能掺入额外的信息进行干扰，即保证原图片的“信息完整性”。图像切割算法一般有投影切割法、均值分割法和水滴法。

**4.1.1 投影切割法**

投影切割算法需要遍历图片像素点，一般对非黏连的验证码图片才能起到良好的切割效果。具体做法如图所示：



图4.1.1投影切割法

这里我们把所有的竖直方向上的黑像素点投影到X轴，然后两个谷底便是两条切割垂线A和B，如下图：

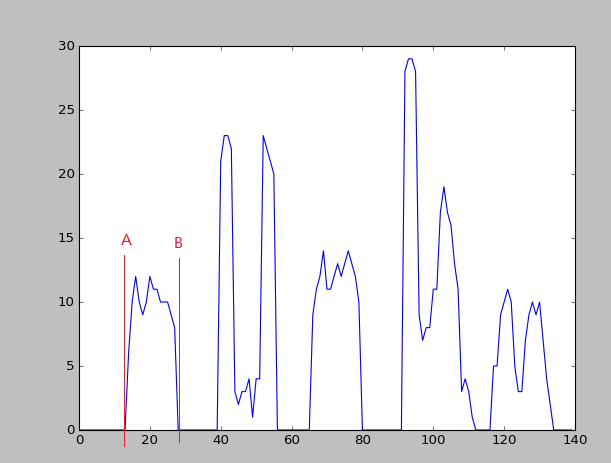


图4.1.2切割线

然后沿垂线A和垂线B进行切割，在切割后的新图像中从下往上遍历像素点，找到第一个有像素的点记录位置，做一条横线C，这样就找到了我们的第一条标记线。然后继续向上扫描遍历像素点，找到第一个没有像素点的位置，做一条垂线D，这样就找到了我们第二条标记线，然后就可以切割出我们所需要的字符。其余四个字符也按同样的方法提取出来。当然从描述看来，这个算法需要遍历找到四个边界线，算法复杂度相对较高。

投影切割算法在切割时表现出良好的性能：在1000个字符中错误分割的字符只有38个，切割准确率高达96.2%！但是当如果验证码图片粘连的时候，就无法正确切割，如下图：

E:\graduate_project\img_cut\20160322104952561366_.png

·

图4.1.3

这张图片本该分解成两张图片，可被扫描时X轴方向上有交集的情况，这时会把两个字符误判成一个字符就未被分割开，这会影响到后面的识别。鉴于此，我又探索了另外针对粘连字符一种算法——均值分割算法和滴水切割算法。

### 4.1.2 均值分割法

均值分割法的思想是：记录原验证码图片的长度，切割后挨个对比字符间的长度，如果某个字符长度大于其他字符长度的均值，就从该字符的平均字符长度处向两侧找竖直方向上有效像素个数的极小值点，然后从极小值点进行分割。

在实际测试中，这种方法虽然在一定程度上可以解决粘连字符的问题，但是更多可能会破坏字符，这样可能对之后的识别造成干扰。如下面字符中‘J’字母长度远小于‘M’，按此算法会把‘M’切割拆开。



图4.1.4

### 4.1.3 滴水切割算法

滴水切割算法是先在验证码图片的顶部找到像素点个数的极小值位置，然后沿着图片边缘的轨迹像水滴一样逐渐往下渗入，然后“水滴”的轨迹就可以作为切割的边界。在实际操作中需要不断寻找像素点个数的极小值位置。

首先计算出粘连字符在竖直方向投影的极小值点个数，字符块的最大宽度为MaxW，字符块的平均宽度为AveW，则如果图片宽度小于最大字符宽度则认为是单个字符。然后从平均字符宽度处向两侧找极小值点，从极小值点处进行分割即可。粘连字符在粘连处竖直投影个数很小，所以可以按此算法对粘连字符实现很好的切割效果。

和上面投影法不同的地方在于：投影法是找到不含像素点的轴线，而滴水切割法是先计算字符平均宽度，然后从该处往两侧找到含像素点少的极小值点。认为该处即是分割线。

核心代码如下：

def Cutgraph(Graph):

Pixels = Graph.load()

(W, H) = Graph.size

MaxW = 20 # 最大字符宽度

AveW = 14 # 平均字符宽度

if W < MaxW: # 若小于最大字符宽度则认为是单个字符

return [Graph]

Blocks = []

PixelCount = []

for i in range(W): # 统计竖直方向像素个数

Count = 0

for j in xrange(H):

if Pixels[i, j] == TEXTCOLOR:

Count += 1

PixelCount.append(Count)

for i in range(W): # 从平均字符宽度处向两侧找极小值点，从极小值点处进行分割

if AveW - i > 0:

if PixelCount[AveW - i - 1] > PixelCount[AveW - i] < PixelCount[AveW - i + 1]:

Blocks.append(Graph.crop((0, 0, AveW - i + 1, H)))

Blocks += Cutgraph(Graph.crop((AveW - i + 1, 0, W, H)))

break

if AveW + i < W - 1:

if PixelCount[AveW + i - 1] > PixelCount[AveW + i] < PixelCount[AveW + i + 1]:

Blocks.append(Graph.crop((0, 0, AveW + i + 1, H)))

Blocks += Cutgraph(Graph.crop((AveW + i + 1, 0, W, H)))

break

return Blocks

## 本章总结

在图像切割时，首先用最简单的遍历像素点办法，这依然取得了很好的效果；然而考虑到有些字符可能会产生粘连的情况，我又测试了均值分割法和滴水切割法。当然算法的核心是为了确保图片信息的完整性以及不会掺加干扰因素。在实际测试中如果是没有粘连的验证码，投影切割法表现的性能最好。

# 数据提取和识别

## 数据提取

上面验证码已经被切割成只含有单个字符的黑白图片。接下来就把图片信息转化为数字信息。先调整图片大小，把图像规范化为 24\*32 像素的图片，然后遍历图片信息中的所有像素点，如果遇到黑像素置为1，否则便置为0。这样便生成一个 24\*32 的矩阵作为每个字符的特征值。

## 识别思想

让计算机识别图像的思想无非就是：先把图像里面的信息提取成计算能识别的数据，再给计算机输入数据（特征值），然后“告诉”计算机这是什么数字或者字母。在相应的训练算法结束之后，我们可以输入数据进行测试：在输入数据后计算机会根据相应的算法进行判断和识别。识别算法由简到难分别有：异或识别算法、KNN识别算法和BP神经网络识别算法。

## 异或识别算法

异或算法算法使用异或操作遍历。我们先从切割后的后的字符里面分拣出出1到10和a到z的所有共计36个字符制作字库。鉴于有些大小写字母差别很大，我们也单独提出来分拣，这些字母有：A(37),B(38),D(39),E(40),F(41),G(42),H(43),N(44),Q(45),R(46),T(47)和Y(48)，所以字符匹配库中共有48个样本。

如下图：

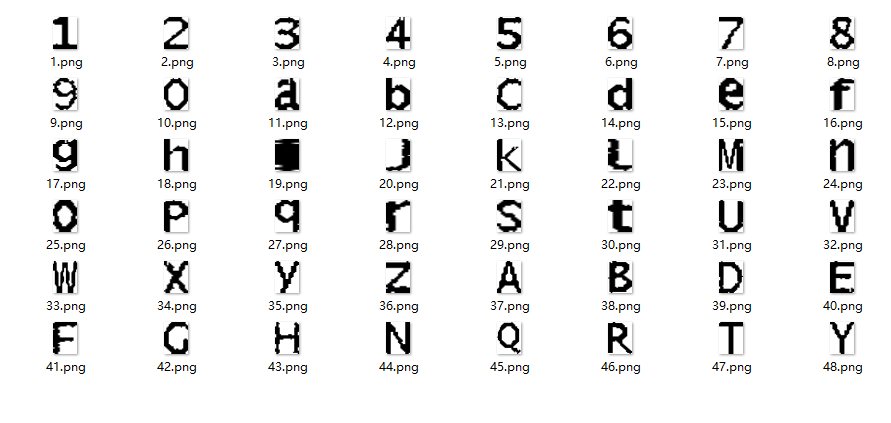


图5.3.1 模板库

然后把字符按4.1中的提取方式提取出数据，如下：

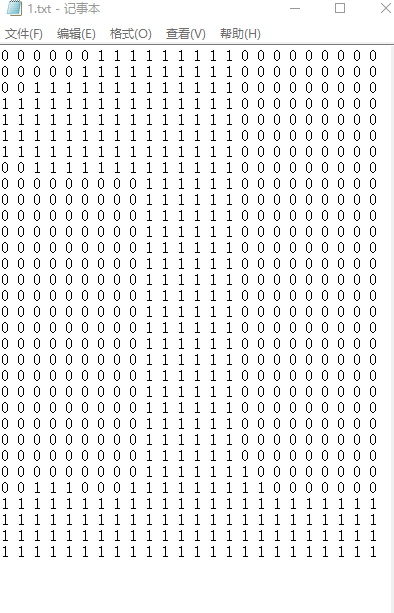


图5.3.2 模板库

我们可以看到提取的数据中所有的“1”组成了数字的轮廓。

**5.3.1 算法思想和实现**

匹配相似度算法里面，异或操作是能想到的最简单的算法：即遍历对比矩阵的值，然后进行一个计数，不同的值越少，相似度越高，例如下图：



图5.3.3 对比图

异或比较后得到：

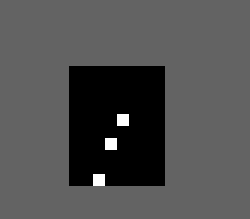


图5.3.4 对比图

我们可以得到计数count是3，此时相似度很高；而如果换做其他任何其他字符，相似度对比时计数count会很大。

代码实现：

my\_dic={1:'1',2:'2',3:'3',4:'4',5:'5',6:'6',7:'7',8:'8',9:'9',10:'0',#建立一个自己的匹配字典

11:'a',12:'b',13:'c',14:'d',15:'e',16:'f',17:'g',18:'h',19:'i',

20:'j',21:'k',22:'l',23:'m',24:'n',25:'o',26:'p',27:'q',28:'r',

29:'s',30:'t',31:'u',32:'v',33:'w',34:'x',35:'y',36:'z',37:'A',

38:'B',39:'D',40:'E',41:'F',42:'G',43:'H',44:'N',45:'Q',46:'R',

47:'T',48:'Y'}

#第一种识别思想：亦或(^)操作结果越小，相似度越高

img=Image.open(os.path.join(ori\_path,'20160322104949889456.png' )).convert('L')

l=img2vector(img)

#print(len(l))

arr=[]

for n in N[1:49]:

if len(l)!=len(n):

print("wrong!!")

exit()

amount=0

for i in range(len(l)):

if n[i]^l[i]:

amount+=1

arr.append(amount)

v=min(arr)

#print(arr)

print(my\_dic.get(arr.index(v)+1))

**5.3.2 代码测试**

我使用该算法来测试之前已经处理好的二值化图：



图5.3.5 待测试图

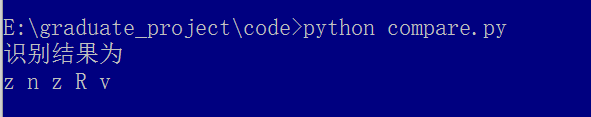


图5.3.6 识别测试

**5.3.2 算法分析和改进**

在实际测试中这种算法识别率最高，而这种高识别率背后却是极高的算法复杂度：不仅需要遍历48个字模板，还需要在每个模板里面遍历对比24\*32 的矩阵，算法复杂度已经达到！借鉴前面的“投影切割法”，对算法进行改进：计数在竖直方向上的像素个数。这样我们在遍历对比的时候数据是一个长度为24的数组，算法复杂度马上变成了，算法复杂度整整降低了一个维度。

## KNN识别算法

**5.4.1 算法思想和实现**

KNN算法可以简单理解为：在同一特征空间内，欧式距离最接近的K个样本属于同一个类别。当然这里的K值一般都比较小，依自己的需求而定。

假定所有的实例对应于维欧氏空间中的点。一个实例的最近邻是根据标准欧氏距离定义的。更精确地讲，把任意的实例表示为下面的特征向量：

 (5.1)

其中表示实例的第个属性，那么两个实例和之间的距离定义为，

其中：

 (5.2)

当越小时，两个实例和的欧氏距离越近，也即二者相似度越高。

在实际代码中我们把可能的识别结果序号作为标签（label），然后再把所有模板库里面的所有提取的数据集合到一块作为类别集合（group），计算测试数据与所有数据集之间的欧氏距离，欧式距离最小的便是所求的结果。

核心代码实现：

def classify0(inX,dataSet,labels,k):

#先计算各自的欧氏距离

dataSetSize=dataSet.shape[0]

diffMat=tile(inX,(dataSetSize,1))-dataSet

sqDiffmat=pow(diffMat,2)

sqDistance=sqDiffmat.sum(axis=1)

distances=sqDistance\*\*0.5

sortedDisIndicies=distances.argsort()

classCount={}

#排序欧式距离最小的前K个数

for i in range(k):

voteIlable=labels[sortedDisIndicies[i]]

classCount[voteIlable]=classCount.get(voteIlable,0)+1

sortedClassCount=sorted(classCount.items(),

key=operator.itemgetter(1),reverse=True)

#距离最小的即是最可能的结果

return sortedClassCount

group=array(N[1:49])

labels=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,

22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,

40,41,42,43,44,45,46,47,48]

print(my\_dic.get(classify0(l,group,labels,2)))

**5.4.2 算法测试**

我依然使用该算法来测试之前已经处理好的二值化图：



图5.4.1

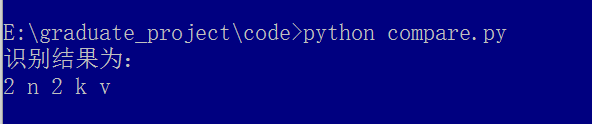


图5.4.2 识别结果

可见这次识别结果实在很一般，比起上次异或算法的全部识别成功，这次才识别出了两个，算法识别效果很差。

**5.4.3 算法改进**

KNN算法识别效果很一般，这次5个字符识别错误3个。当然，结果也是意料之中：因为：

1. 数据输入维度（24\*32）太大，数据分类时很容易出错，如下图：

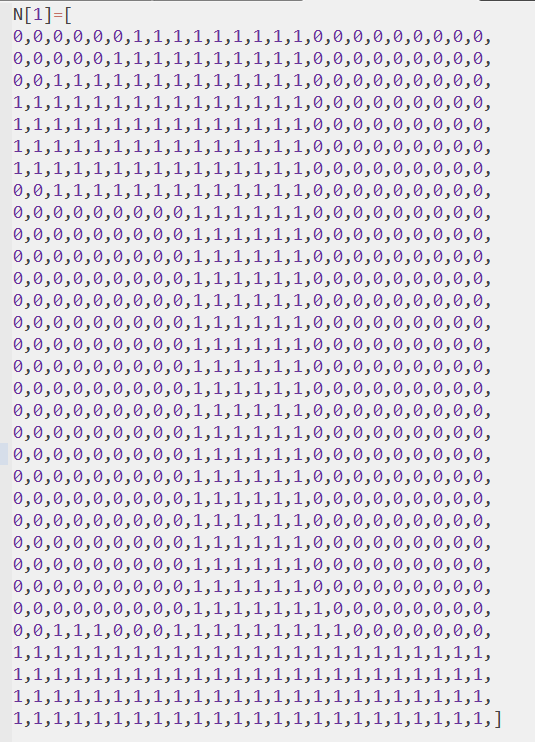


图5.4.3 数据1

2.类域的交叉部分较少，这会导致分类决策很容易出错。

我继续用之前的投影法改进下算法，让输入数据维度变小，具体做法是把图像的像素点进行垂直投影然后计数。这样每个切割字符的提取数据就变成了一个长度为24的列表，相比于上面的提取方法，维数也变为1维。我们继续用投影图作为例子来解释：

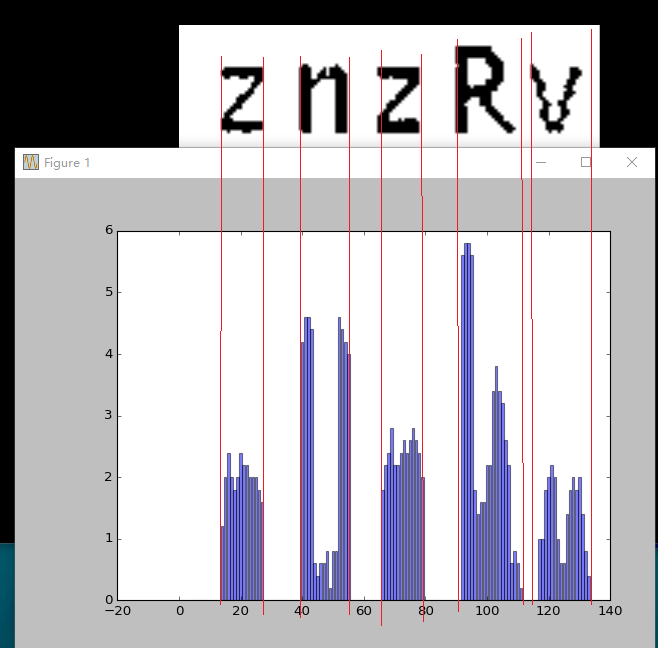


图5.4.4 投影直方图

然后我们提取的数据便是每个纵行投影的1的个数：

[8, 8, 11, 11, 11, 11, 12, 12, 13, 32, 32, 32, 32, 32, 32, 6, 5, 4, 4, 4, 4, 4, 4, 4]

经过改进后，数据维度大大减小，计算欧氏距离时我们只需算一维空间的欧氏距离计即可，这会让分类识别准确性更高。

提取数据的代码如下：

for i in range(1,49):

#先从导出的txt里面加载数据

b = np.loadtxt((os.path.join(file\_path,'%d.txt')%i),delimiter=" ",dtype=int32)

#规范化

c =b.reshape(32,24)

#print(b)

#计算“投影”下来的1的数目

d= np.sum(c,axis=0)

lst=d.tolist()

tmp\_lst=[str(j) for j in lst]

#写到sample01.py准备后面使用

with open(os.path.join(file\_path,'sample01.py'),'at') as f:

f.write('[')

f.write(', '.join(tmp\_lst))

f.write('],')

f.write('\n')

把数据写入sample01.py之后，再次测试，结果如下所示：

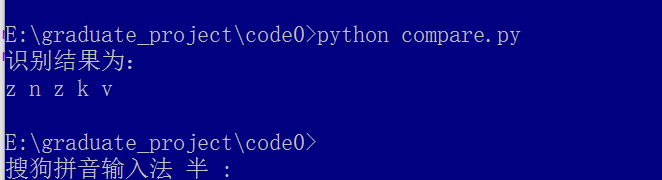


图5.4.5 识别结果

**5.4.4 结果分析**

在改进算法之后，发现识别率提高了，同时也大大降低了算法复杂度。原因也在意料之中：数据维度变小之后，分类匹配时误差会更小。但是由于“R”和“k”在提取特征值时候二者重叠度很大，这也导致了这里的识别错误。但总体来说，这次提升仍然让人很开心！当然在实际测试中诸如“u”和“n”等字母的识别也会相互产生混淆，是因为这些字母在投影后的像素点个数很接近。鉴于KNN算法在决策时候的失误无法用算法本身减小，我们就用一个可以通过自身学习来减小误差的算法——BP网络神经算法。

**5.4.5 和BP神经网络算法的对比**

KNN算法是最简单的神经网络算法，在决策时会因为维度过大而造成误判。上面算法改进中我们通过数据降维改善了识别率，然而如果数据无法降维失误率将无法降低。这里就引出了一个能学习归纳然后降低失误的算法——BP神经网络算法。该算法更像人脑的工作方式，可在不断的学习过程中调整权值，使得输出失误率越来越小。

BP神经网络算法实现作为本论文的拓展，具体详细原理和数学推导就不赘述。我在实验中简单实现了一个python版本的BP神经网络算法，用此算法进行图像识别的思想如下：

1.输入数据是每个切割字符竖直投影下1的个数，如[8, 8, 11, 11, 11, 11, 12, 12, 13, 32, 32, 32, 32, 32, 32, 6, 5, 4, 4, 4, 4, 4, 4, 4]。

2.输出数据是一个代表字符的一维矩阵，比如1的输出矩阵为： [1, 0 , 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]。总共有48个可能识别的字符，所以这里数据输出是48维的向量。

我在输入测试数据的时候输入了每个字符的竖直投影计数个数，即输入层节点数是24，而由于字符分类有48个，所以输出层节点数是48，每100次计算一下错误率。随着学习深入，失误率会慢慢降低，学习识别率会一直升高，最后逼近98.5%！

失误率曲线和测试结果图分别如下：

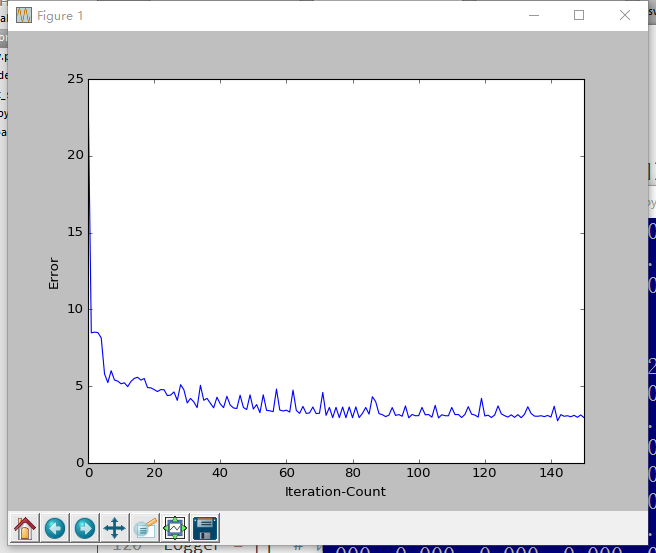


图5.4.6 失误率曲线

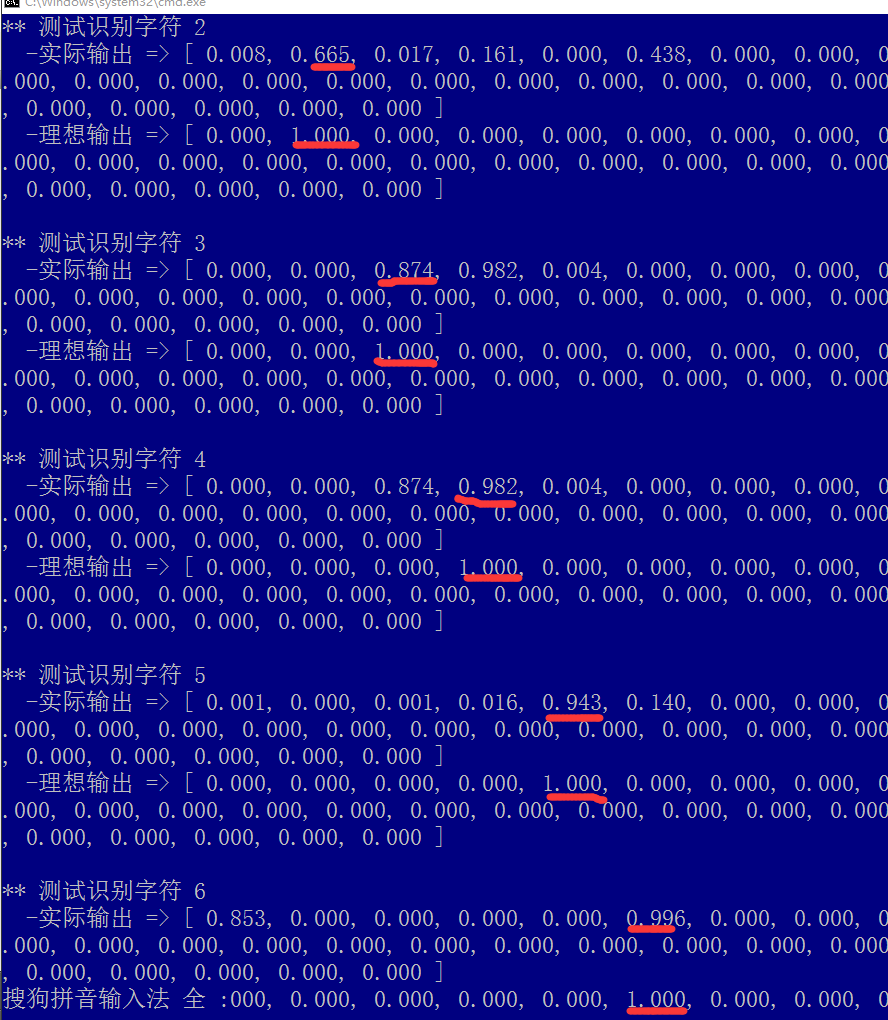


图5.4.7 测试结果

可以看出，实际输出与理想输出非常接近。

我用BP神经网络算法训练数据并进行测试，也取得了较好的识别效果。在实验里面吧学习率设置为0.5时学下效果最好，在这里面如果把学习率设置不合适会导致过度学习，当然一般在0.2到0.8之间学习效果都不错。

BP神经网络算法比起前面的对比相似度的两种算法显然更胜一筹：这个算法除了让计算机学习之外还有归纳和纠正错误的能力：该算法会把误差反向传播，然后一直调整输入的权值而让识别率变得越来越高，可以该算法更接近人脑细胞的工作方式。然而由于涉及到较多的数学原理，所以该算法较难理解。

## 本章总结

本章我们用异或算法和KNN算法来比较识别效果。异或算法复杂度高，但识别效果要好；KNN算法复杂度相对较低，但识别效果比较差。

我们当然想在保证算法复杂度低的情况下又能有较好的识别效果。所以又把输入数据的维度变小了，达到了预期效果。类似的地方在于，异或算法和KNN算法都是先分类数据让计算机进行学习，然后输入测试数据分析计算相似度。当然前者只是普通的遍历对比算法，而后者已经属于机器学习的范畴。

简单实现了用BP神经网络算法识别验证码，这个算法可以不断更新权值来提高识别准确率。

# 小结

## 6.1 工作总结

本文总结了验证码识别的处理步骤极其常见算法，实现了用不同的的识别算法来训练识别验证码的功能。分析了影响验证码处理过程和识别过程中关键的影响因素，对机器学习进行了一次简单的尝试。现将本文主要工作总结如下：

1.介绍了图像灰度化的原理及其常见的实现方法，对比了不同实现方式对保留图像对比度和特征的不同；

2.对比了不同的图像去噪点的方法，注入填充算法和相邻对比法在去噪表现方面各有千秋，实现方式也不近相同，只是前者算法复杂度更高。

3.阐述了二值化的方法，即找到一个合理的阈值，大于此阈值把像素点的灰度值设置为255，反之设置为0。研究了不同的阈值寻找方法及其效果表现。

4.介绍了三种切割字符的方法，在试图尽可能保留原图片信息的基础上还不能掺加干扰因素。投影切割法在没有字符粘连的情况下表现很好，可对于有字符粘连的情况却会出现切割失误。均值切割粗略地估计字符长度，然后平均分开，这当然在一定程度可以避免投影切割法的不足，然而可能会造成信息的损失。这就引出了滴水切割法，该算法对粘连字符切割有较好的表现。

5.阐释了简单提取图像信息数据的方法，并用提取的数据进行训练和测试，异或算法和KNN算法在识别的原理很相似，但在实际识别效果和算法复杂度方面差异很大。

6.简单实现了BP神经网络算法，用该算法进行了数据训练和测试。该算法把错误反向传递，然后均摊，最大程度减小了误差。

## 6.2 心得体会

至少从目前的人工智能和机器学习的发展程度看，没有简单的做法能自动处理各种不同的验证码。这是因为不同的验证码有不同的特点，即使某个破解系统的破解能力很强，那么系统自然也十分复杂。

前面的各个步骤分析过来，其实对验证码识别影响最大的是切割！因此对验证码防破解的建议如下：

1.加入复杂的干扰因素（如“五角星”）；

2.让验证码字符无规律地倾斜也会让切割变得复杂；

3.噪点很容易会被破解者用去噪点算法过滤掉，但是一些面积略大的块或干扰线条也会对切割破解产生较大影响。

4.让验证码有效时间变得更短，防止被破解。

5.避免让验证码所含的核心信息过久暴露，即可以用轮廓变化的图片作为验证码，让验证码图片反复消失和出现。

## 6.3 进一步的工作

在今天验证码识别技术变得很成熟，无论是“抢小米”系统还是“12306火车票抢票系统”，都能在更复杂环境下实现对更复杂验证码的识别。这其实也在侧面说明了验证码本身存在漏洞，像本人用异或算法，识别率几乎高达100%。当然，除了在本实验遇到的粘连字符切割外，其他的诸如gif合适的动态验证码、12306购票网站的实物验证码本论文暂时无法识别，基于此，我提出了以下的展望：

1. 验证码开始逐渐从“静图”往“动图”转换，所以希望能解决动图gif验证码的字符切割。具体思路是捕捉动态验证码的快照，把动图转化为静图。

2.结合网上常用的识图API，尝试去破解“12306”购票网站的验证码。这个网站的验证码比起其他网站传统的“数字字母组合”验证码更加复杂，但我们可以借助识图API去尝试破解。

3.尝试使用支持向量机或贝叶斯分类器来训练识别验证码。之前的识别算法都存在一些缺陷，无法很准确对数据集进行分类，识别时也会受到干扰。

4.对一些汉字、简单计算值之类的验证码也可以尝试用机器学习的方式来尝试实现一下。

# 参考文献

[1] 郭铁桥,郑海明;机动车牌照图像二值化提取的研究[J];计算机时代;2008年12期

[2] 马宏锋,李颖,王宏斌.基于FPGA的交通监控图像高速预处理［J］.兰州交通大学学报,2012,31(3):112-115

[3] https://zh.wikipedia.org/wiki/机器学习 维基百科 机器学习词条

[4] 王巍;基于Haar特征的高分辨率遥感影像地物识别方法研究[D];中国地质大学(北京);2012年

[5] 王淑珍;机器学习算法的Weka嵌入[D];华南理工大学;2013年

[6]Greg Mori, Jitendra Malik, 2003,Recognizing Objects in Adversarial Clutter:Breaking a Visual CAPTCHA[C],IEEE Conference on Computer Vision & Pattern Recognition,IEEE Computer Society,1(124-141)

[7] 张淑雅，赵一鸣，赵晓宇等.2007,认证码字符识别方法的研究[J].宁波大学学报:理工版，12(4):429-433.

[8] 郭铁桥,郑海明,机动车牌照图像二值化提取的研究[J];计算机时代;2008年12期

[9] 马宏锋,李颖,王宏斌.基于FPGA的交通监控图像高速预处理［J］.兰州交通大学学报,2012,31(3):112-115

[10] 汪世刚.黑白图像自动彩色化方法研究 [D].上海大学;上海大学,2010

[11] http://baudelaire.is-programmer.com/posts/35352.html Baudelaire is Programmer博客

[12] 李瑞霞,周熠,用C^＋＋builder实现图像的黑白处理[J].教育技术导刊,2008,7(10):94-96.

# 致 谢

首先我在这里向养育和培养我的父母表示深深地思念和感谢，也向关心我，帮助我的老师和同学表示感谢！

能顺利圆满地完成我的设计，离不开自身的努力，离不开指导老师胡晓勤老师的指点，更离不开四川大学这个大环境四年来在学习和生活的熏陶和培养。

大四下学期开始不久，我开始了我的毕业设计。在这几个月里，我得到了导师胡老师的关心和教导，胡老师在学术上的渊博知识让我在这几个月的学习和研究中深受其益。我向于老师表示深深地谢意，感谢胡老师悉心指导；向帮助我的同学表示感谢；向软件学院实验室中，陪着我们度过几多光阴的管理员和老师表示谢意。

该论文的完成，为我在川大的学习生涯画上了一个圆满的句号。在四川大学度过的美好时光将使我终生受益。

作者签名：

2016年05月 日

指导教师签名：

2016年05月 日

**附录1 需求规格说明书**

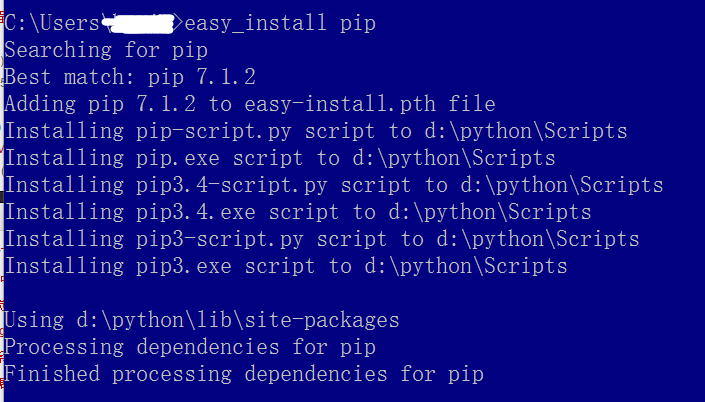
1. **运行环境规定**
2. 1. 运行支持

由于本人使用了python语言下众多的包，因而先需要安装.whl后缀名的python支持包，包括pillow、numpy、scipy、matplotlib等，都是用来数据处理或图像处理。

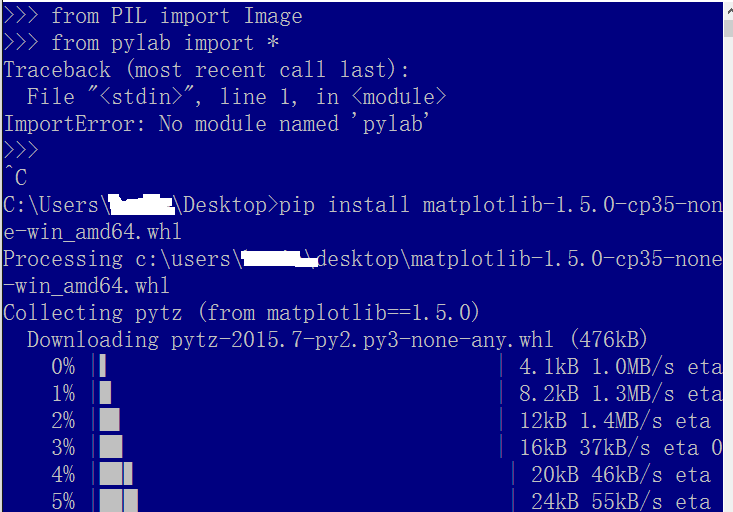
* 1. 支持软件

1. 操作系统：windows
2. 编程语言：python 3.x（未用open cv，暂时不支持python3.x版本）
3. Python支持包：matplotlib、PIL、scipy、numpy
   1. 支持包安装

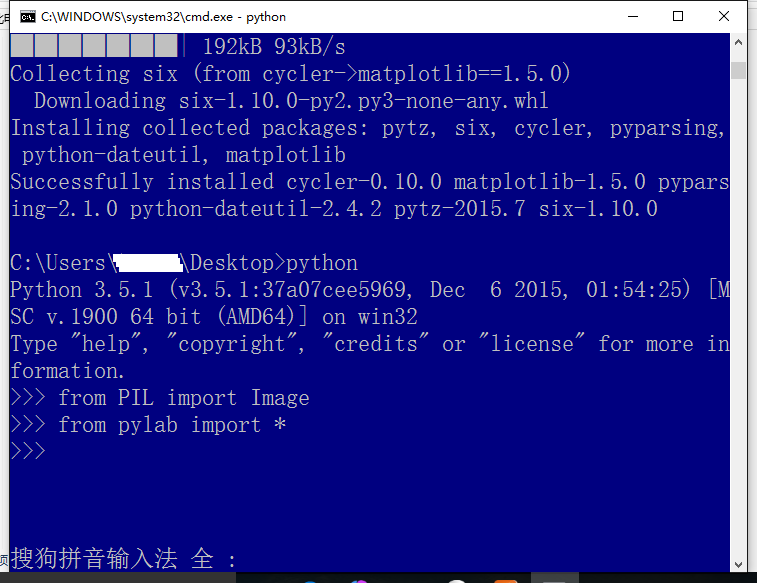
以hl为后缀的python支持包安装略复杂，我选择了用最简单的pip安装方法，如下，在cmd窗口下输入easy\_install pip，就可以安装pip了，如下图所示：



本来在python中导入包会报错：



然后我们用pip install命令安装matplotlib包，接下来就可以发现能正常导入包了：



**附录2 部分源码**

2.1 下载验证码图片 load\_img.py

'''

@Author qixuxiang

@Date 2016.03.01

load\_im.py

验证码图片下载和尺寸调整

'''

import urllib.request

import os

from PIL import Image

url = 'http://passport.csdn.net/ajax/verifyhandler.ashx'

ori\_path = 'E:\graduate\_project\img'

if not os.path.exists(ori\_path):

os.mkdir(ori\_path)

if \_\_name\_\_ == '\_\_main\_\_':

for i in range(1,201):

#csdn验证码下载

with open((os.path.join(ori\_path,'%d\_ori.png' %i)), 'wb') as f:

f.write(urllib.request.urlopen(url).read())

#调整图片大小

img = Image.open((os.path.join(ori\_path,'%d\_ori.png' %i)))

img.resize((140,45),Image.ANTIALIAS).save(os.path.join(ori\_path,'%d.png' %i))

2.2 图片预处理 img\_proc.py

'''

@Author qixuxiang

@Date 2016.03.05

ima\_proc.py

图片预处理，包括调整大小、灰度化、二值化和去噪点

'''

import sys,os

from PIL import Image,ImageDraw

#二值化处理

def binary(im):

pixels =im.load()

for x in range(im.width):

for y in range(im.height):

pixels[x,y] = 255 if pixels[x,y]>125 else 0

#二值判断,如果确认是噪声,用改点的上面一个点的灰度进行替换

def getPixel(image,x,y,G,N):

L = image.getpixel((x,y))

if L > G:

L = True

else:

L = False

nearDots = 0

if L == (image.getpixel((x - 1,y - 1)) > G):

nearDots += 1

if L == (image.getpixel((x - 1,y)) > G):

nearDots += 1

if L == (image.getpixel((x - 1,y + 1)) > G):

nearDots += 1

if L == (image.getpixel((x,y - 1)) > G):

nearDots += 1

if L == (image.getpixel((x,y + 1)) > G):

nearDots += 1

if L == (image.getpixel((x + 1,y - 1)) > G):

nearDots += 1

if L == (image.getpixel((x + 1,y)) > G):

nearDots += 1

if L == (image.getpixel((x + 1,y + 1)) > G):

nearDots += 1

if nearDots < N:

return image.getpixel((x,y-1))

else:

return None

def clearNoise(image,G,N,Z):

draw = ImageDraw.Draw(image)

for i in range(0,Z):

for x in range(1,image.size[0] - 1):

for y in range(1,image.size[1] - 1):

color = getPixel(image,x,y,G,N)

if color != None:

draw.point((x,y),color)

ori\_path = 'E:\graduate\_project\img'

file\_path = 'E:\graduate\_project\img\_proc'

if not os.path.exists(ori\_path):

os.mkdir(ori\_path)

if not os.path.exists(file\_path):

os.mkdir(file\_path)

if \_\_name\_\_ == '\_\_main\_\_':

for i in range(1,201):

#先进行灰度化

gray\_im = Image.open((os.path.join(ori\_path,'%d.png' %i))).convert('L')

gray\_im.save((os.path.join(file\_path,'%d\_g.png' %i)))

#再进行去噪点操作

noise\_im = Image.open((os.path.join(file\_path,'%d\_g.png' %i)))

clearNoise(noise\_im,50,4,4)

noise\_im.save((os.path.join(file\_path,'%d\_n.png' %i)))

#最后进行二值化

binary\_im = Image.open((os.path.join(file\_path,'%d\_n.png' %i)))

binary(binary\_im)

binary\_im.save((os.path.join(file\_path,'%d\_b.png' %i)))

**2.3** 切割图片 load\_img.py

'''

@Author qixuxiang

@Date 2016.03.08

cutgraph.py

图片切割并规范化成24\*32图片，利于后面提取特征值

'''

from PIL import Image

from numpy import \*

import os

import datetime

import time

ori\_path = 'E:\graduate\_project\img\_proc'

file\_path = 'E:\graduate\_project\img\_cut'

re\_path = 'E:\graduate\_project\img\_re'

if not os.path.exists(ori\_path):

os.mkdir(ori\_path)

if not os.path.exists(file\_path):

os.mkdir(file\_path)

if not os.path.exists(re\_path):

os.mkdir(re\_path)

#图片x轴的投影，如果有数据（黑色像素点）值为1否则为0

def get\_projection\_x(image):

p\_x = [0 for x in range(image.size[0])]

for w in range(image.size[1]):

for h in range(image.size[0]):

if image.getpixel((h,w)) == 0:

p\_x[h] = 1

length=len(p\_x)

return p\_x

'''测试函数

def count\_size(im):

print(im.size)

'''

#获取分割后的x轴坐标点

#返回值为[起始位置, 长度] 的列表

def get\_split\_seq(projection\_x):

res = []

for idx in range(len(projection\_x) - 1):

p1 = projection\_x[idx]

p2 = projection\_x[idx + 1]

if p1 == 1 and idx == 0:

res.append([idx, 1])

elif p1 == 0 and p2 == 0:

continue

elif p1 == 1 and p2 == 1:

res[-1][1] += 1

elif p1 == 0 and p2 == 1:

res.append([idx + 1, 1])

elif p1 == 1 and p2 == 0:

continue

return res

#分割后的图片，x轴分割后，同时去掉y轴上线多余的空白

def split\_image(image, split\_seq=None):

if split\_seq is None:

split\_seq = get\_split\_seq(get\_projection\_x(image))

length = len(split\_seq)

imgs = [[] for i in range(length)]

res = []

for w in range(image.size[1]):

line = [image.getpixel((h,w)) for h in range(image.size[0])]

for idx in range(length):

pos = split\_seq[idx][0]

llen = split\_seq[idx][1]

l = line[pos:pos+llen]

imgs[idx].append(l)

for idx in range(length):

datas = []

height = 0

for data in imgs[idx]:

flag = False

for d in data:

if d == 0:

flag = True

if flag == True:

height += 1

datas += data

#print(datas)

#print(len(datas))

#二值化图片被切割后被规范化为24\*32的图片

child\_img = Image.new('L',(split\_seq[idx][1], height))

child\_img.putdata(datas)

#生成了时间戳，不重复随机数，精确到微秒

nowTime=datetime.datetime.now().strftime("%Y%m%d%H%M%S%f")

j=int(nowTime)

child\_img.save(os.path.join(file\_path,'%d\_.png' % (j)))

img=Image.open(os.path.join(file\_path,'%d\_.png' % (j)))

new\_img = img.resize((24,32),Image.BILINEAR)

new\_img.save(os.path.join(re\_path,'%d.png'% (j)))

time.sleep(0.1)

res.append(child\_img)

return res