

合肥工业大学

# 毕 业 设 计

设计题目 验证码识别中预处理算法的研究与实现

学生姓名 李 嘉

学 号 20093086

专业班级 软件工程 09-2 班

指导教师 郭 骏

院系名称 软件学院

20 13 年 5 月 28 日

## 目 录

摘 要 .....	1
Abstract.....	2
1 引言 .....	3
1.1 验证码概述 .....	3
1.2 验证码分类 .....	3
1.2.1 图片验证码 .....	4
1.2.2 非图片验证码.....	4
1.3 研究现状.....	5
1.4 验证码的作用和意义.....	6
1.5 研究验证码识别的意义.....	7
1.6 验证码识别的步骤 .....	7
2 通用的验证码预处理过程.....	9
2.1 灰度化.....	9
2.2 二值化.....	11
2.3 去边框和反色 .....	13
2.4 滤波去噪和平滑 .....	14
2.5 连通性去噪 .....	17
2.6 细化.....	18
2.7 去干扰线.....	19
3 分割和识别 .....	21
3.1 分割.....	21
3.1.1 投影法 .....	21
3.1.2 连通性分割 .....	21
3.2 识别.....	22
4 系统设计与实现.....	23
4.1 实用性论证 .....	23
4.2 系统结构与实现 .....	26
结论.....	29
致谢.....	30
参考文献 .....	31

# 验证码识别中预处理算法的研究与实现

**摘 要：**验证码（CAPTCHA）凭借其简单的技术、极小的数据流量和高效性迅速推广，已经成为了网络生活中不可分割的一部分。当前的验证码绝大多数是基于图片中字符识别的验证码（让用户辨识一段扭曲的字符串）。对于这一类验证码的预处理方法，本文对灰度化、二值化、滤波平滑、连通性去噪、细化、去干扰线的基本方法进行了研究并给出了伪代码或原理。接下来借助开源 OCR 引擎以一种微粘连和微扭曲的验证码为例验证了完整的预处理过程并得到了 18.3% 的识别率。同样的流程对于只含噪点的验证码则可以完美识别。文章最后描述了一个图形界面的通用验证码识别演示系统的实现过程。

**关键词：**验证码，预处理，伪代码，图像处理

# Study and Implementation of Pre-process Algorithms in Recognition of CAPTCHA

**Abstract :** Based on the simple technique, tiny volumes of data and efficiency, CAPTCHA has become an integral part of the Internet. Nowadays most captcha is image-based captcha(implemented as distorted text which the user must correctly transcribe). This thesis focus on the basic implement of pre-process algorithms, including gray processing, binarization, smoothing filter, connectivity denoising, thinning and the removal of interferential curve. For a kind of existed captcha, we put forward an overall pre-process procedure. With the help of an open-source OCR system, the broken rate is 18.3%. This procedure can deal with easy types of captcha which contains only noise perfectly. Finally, we write a common demo program of the recognition of captcha with graphical interface.

**Keywords:** CAPTCHA, Pre-processing, Pseudocode, Image Process

# 1 引言

## 1.1 验证码概述

验证码 (CAPTCHA) 全称为 “Completely Automated Public Turing test to tell Computers and Humans Apart”, 暂且译作 “全自动区分计算机和人类的公开图灵测试”。它最早由卡内基·梅隆大学 (简称 CMU) 的 Luis von Ahn、Manuel Blum、Nicholas J. Hopper 以及 John Langford 于 2000 年提出<sup>[2]</sup>并于 2002 年公开发表<sup>[1]</sup>。它是应当满足两个要求<sup>[3]</sup>的程序<sup>[1]</sup>:

- 1、人类可以通过测试;
- 2、当前的电脑程序无法通过。

这与阿兰·图灵在 1950 年提出的“图灵测试”<sup>[5]</sup>很类似。由于在这个测试中计算机是测试者, 受试者是人类或计算机, 而标准图灵测试中的人类是测试者, 受试者是人类和计算机, 因而人们也称 CAPTCHA 是一种反向图灵测试。CAPTCHA 经常与另一个新概念混用: 真人交互证明 (Human Interactive Proofs, 简称 HIPs)。由于 CAPTCHA 中的“公开”是指产生和测试验证码的程序应当公开, 因而严格来讲, 现在所用的验证码很大一部分只能算作“HIPs”<sup>[3]</sup>。虽然 HIPs 还包括人类安全认证 (Secure Human Authentication, 或 HUMANOIDs) 的范畴<sup>[3]</sup>, 但通常所说的 HIPs 都是指 CAPTCHA<sup>[4]</sup>。

Yahoo! 是 CAPTCHA 的第一个用户。自此, 验证码凭借其简单的技术、极小的数据流量和高效性迅速推广, 已经成为了网络生活中不可分割的一部分。验证码的用途主要有: 阻止机器人散布广告、防止资料被机器人抓取、阻止免费服务的批量注册 (如免费电子信箱)、限制用户行为的速度、防止暴力破解 (若干次不正确后需要验证码)、保障交易安全性和投票真实性 (确定为真人操作)。

## 1.2 验证码分类

当前的验证码实现绝大多数是基于图片的验证码, 其中最主要的是基于字符的验证码 (让用户辨识一段扭曲的字符串), 此外还有不常见的基于内容和语义的验证码、中文验证码、数学公式验证码等等。此外还有语音验证码、基于领域专有知识的验证码等。

### 1.2.1 图片验证码

#### 1 基于字符的验证码

基于字符的验证码是最常见的一类验证码。它易产生、形式简单、组合多变，不受知识背景的影响。用户需要正确输入验证码图像中包含的扭曲变形的数字、字母或文字。它是验证码领域的主要研究对象。我们提到的验证码默认指此类验证码。

#### 2 其他图片验证码

欢乐验证码<sup>1</sup>就是一种基于理解的验证码，它的问题涵盖面广，除非人工智能可以产生理解力甚至幽默感，否则很难破解。此外还有诸如在图片上数出指定颜色的圆点数、在一群动物图片中找到指定的动物等等五花八门的形式。



图 1.1 欢乐验证码

### 1.2.2 非图片验证码

#### 1 语音验证码

针对残障人士设计的验证码。正有越来越多的大型网站提供这一贴心功能。由于语音识别技术越来越成熟，这一形式的验证码前景并不明朗。

---

<sup>1</sup> <http://captcha.tw>

## 2 基于领域专有知识验证码

给出只有相关领域人士才能回答出的问题，往往用于注册时的验证问题。例如数学论坛给出一个算式要求用户输入结果，编程论坛给出一段程序语句要求给出结果等等，可以很好的阻止恶意注册。但由于问题库有限且特定，无法随机生成问题，因而还是可以轻易通过收集问题来破解。

注册问题：从前我和你一样也是是一名冒险家，直到我(....)中了一箭

回答：膝盖 (两字中文) 

图 1.2 bilibili.tv 注册问题

## 3 其他新颖的尝试

不得不提到 MotionCAPTCHA 项目<sup>2</sup>，它通过鼠标绘图的方式完成验证，你可以在这里<sup>3</sup>看到一个演示；另外还有一个针对触屏手机的验证码项目<sup>4</sup>，通过拖动完成拼图来达到验证目的。NuCaptcha 项目<sup>5</sup>尝试使用视频流（如 flash）里动态的字符代替现有的静态旋转和重叠来达到目的，减小了理解难度，增加了识别难度，但对于无法嵌入视频的设备显得很无力，对于低带宽网络仍需优化。

## 1.3 研究现状

验证码识别与 OCR 系统非常相似，但它有自己的特殊性，其粘连、噪声、扭曲的复杂程度使单纯的 OCR 系统不能直接使用到验证码识别上来。

2003 年，G. Mori<sup>[12]</sup>破解了雅虎使用的 EZ-gimpy 以及 gimpy 验证码，识别率分别达到了 92%和 33%。2004 年，Hocevar. S 发布了他的验证码识别程序 PWNtcha(Pretend We are Not a Turing Computer but a Human Antagonist, 即“假设我们不是图灵机，而是人类的对手”），但其针对的是背景易分离和非粘连的验证码。随着越来越

---

<sup>2</sup> <https://github.com/josscrowcroft/MotionCAPTCHA>

<sup>3</sup> <http://www.josscrowcroft.com/demos/motioncaptcha/>

<sup>4</sup> <http://capy.me/>

<sup>5</sup> <http://www.nucaptcha.com/>

多的粘连、旋转、扭曲的加入，验证码越来越难以识别。2005 年，斯坦福大学的 Chellapilla. K 等人<sup>[14]</sup>的研究表明单个的验证码字符可以很好的被计算机识别。2010 年，S. Li<sup>[13]</sup>等人破解了包含中文的几种网上银行验证码，识别率几乎全部为 100%。另外还有很多用神经网络方法高效识别的大量论文。

但是目前对字符严重粘连或扭曲重叠、无法分割的验证码如 Google 验证码尚无法破解。此外，由于每种验证码都不同，无法设计一个能够自动识别所有验证码的程序，即验证码识别需要有针对性，即便是上文提到的大名鼎鼎的 PWNtcha 也是针对每种验证码单独写的处理步骤。现有的研究也都针对某个网站的一种验证码。可以肯定的说，没有统一的方法可以一次性解决所有验证码的预处理和识别问题。

反观验证码的产生技术，当前加大验证码难度的主要思路就是增加验证码被分割的难度。粘连、噪声、旋转、扭曲、干扰线是主要手段。然而这一技术越来越遇到瓶颈，即人类也越来越难以识别验证码了。

## 1.4 验证码的作用和意义

值得一提的是由 CAPTCHA 的发明者之一 Luis von Ahn 所发起的著名的 reCaptcha<sup>™</sup>项目<sup>6</sup>，使用了在书籍电子化过程中实际遇到的目前的 OCR (Optical Character Recognition, 光学字符识别) 系统无法辨识的文本信息作为验证码，通过统计识别结果得到最佳答案，既得到了很好的验证码图片，又突破了技术限制，推动了纸质资料的电子化过程，更降低了电子化成本。

由于用户在填写验证码的过程中，会仔细阅读验证码图片，思考并输入正确信息来完成验证，因而加以利用，加入广告内容，即可达到非常好的广告效果。验证码广告在国外已初具规模，被誉为“最具前景的广告模式”之一。知名市场调研与数字媒体测评机构 comScore 对验证码广告进行调研后指出它的效果较普通广告有 8-16 倍的提升<sup>[6]</sup>。多家知名厂商如 Microsoft 都曾投放过验证码广告。它已经成为一种性价比极高的互联网广告形式。

虽然验证码为网络安全做出了突出贡献，然而未来字符验证码的道路是否一路

---

<sup>6</sup> <http://www.google.com/recaptcha>



顺风呢？福布斯网站的一篇文章<sup>7</sup>指出，正是由于日渐增强的验证码破解技术，人们不得不加大验证码字符的扭曲程度，甚至达到了真人都难以辨识的程度，这种恶性循环已经导致验证码越来越成为一种负担而不是保护措施。然而也应当看到人们为生成有效的字符验证码做出的努力，以及随之产生的一个非常有趣的“黑客经济”形式：“打码”服务（captcha human bypass），即人工破解服务。由于缺乏针对性的法律法规，有的国外站点<sup>8</sup>甚至长盛不衰，已经营数年之久。

## 1.5 研究验证码识别的意义

要设计出计算机无法识别的理想验证码，就要搞清楚计算机可以识别什么特征的验证码。验证码作为一种图灵测试，灵活性高、意义重大，不仅综合了人工智能、模式识别、图像处理、机器视觉等多领域研究，许多研究成果更可以推广到其他领域。

技术从来都是一把双刃剑，验证码识别除了可被用作恶意用途，当然也可以用于办公自动化领域。对网络服务的性能测试、压力测试环节也有益处。对基于图像的搜索引擎的发展也有推动作用。

预处理是验证码识别过程中不可跳过的一步。简而言之，预处理的目的是为了复原图像所含信息。预处理结果的好坏直接影响到整个流程的识别质量。其中，灰度化的好坏将直接影响灰度图所含信息的完整性；二值化阈值的选择更是直接反映在字符是否完整、清晰；去噪对于特征提取至关重要，去噪结果的清晰与否将直接影响到字符分割的质量。而所有步骤的叠加关系着识别步骤特征的强弱乃至最终识别率的高低。

## 1.6 验证码识别的步骤

验证码识别过程可分为明显的三步<sup>[15]</sup>：预处理（pre-processing）、分割（segmentation）、分类（classification）。其中，预处理步骤是本文的讨论核心，

---

<sup>7</sup> <http://www.forbes.com/sites/singularity/2012/08/28/artificial-intelligence-will-defeat-captcha-how-will-we-prove-were-human-then/>

<sup>8</sup> <http://www.deathbycaptcha.com/>

第二章遵循的步骤可以表示如图：

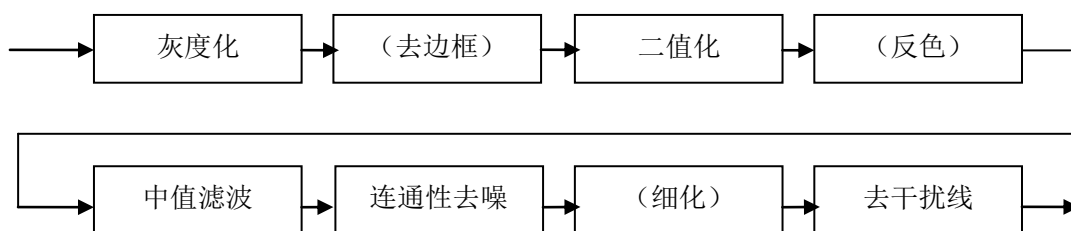


图 1.3 预处理流程

## 2 通用的验证码预处理过程

### 2.1 灰度化

计算机中处理的图像像素的颜色遵循 RGB 颜色模型 (RGB color model)，或可以转化<sup>[7]</sup>为 RGB 模型 (由 HSL、YUV 模型转换)。RGB 即三基色红 (red)、绿 (green)、蓝 (blue)，每个分量可以取 255 个值，这样一个像素点可以有 1677 万多 ( $255 \times 255 \times 255$ ) 种颜色表示。如果图像的颜色空间是一维的，即  $R = G = B$ ，则称其为灰度图像。在数字图像处理中一般先将各种格式的图像转变成灰度图像以减小计算量，在验证码识别中则是为二值化做准备。颜色信息对常用的验证码识别通常没有帮助，灰度化不影响识别 (指公式 2-2)，但这并不绝对。事实上，中科院一篇论文<sup>[8]</sup>中的方法直接对彩色图像进行了一次彩色去噪过程，而不是传统的二值化后再去噪。对于数字图像处理来说，灰度化 (指公式 2-2) 虽然确实丢失了一些颜色信息，但是图像的整体和局部的色彩以及亮度等级分布特征与彩色图的描述是一致的，灰度图像 (指公式 2-2) 的描述与彩色图像一样都能反映图像整体和局部的色度和亮度的分布和特征。

回到验证码预处理上来，没有一种万能的灰度化方法，而且传统灰度化方法可能不能很好地反映原图亮度等信息<sup>[9]</sup>，具体问题需要具体分析。

灰度化的原理很简单，即对 RGB 三个分量 (或称作通道) 进行加权平均得到最终的灰度值。常用的几种灰度化方法有 (设某像素坐标为  $(x, y)$ ):

- 1、加权平均值法：最常用最普遍的图像灰度化方法，给三种基色 R、G、B 赋予不同的权值，即：

$$\text{gray}(x, y) = \frac{W_r \times R(x, y) + W_g \times G(x, y) + W_b \times B(x, y)}{W_r + W_g + W_b} \quad (\text{公式 2-1})$$

其中， $W_r$ 、 $W_g$ 、 $W_b$  分别为 R、G、B 的权重。一个经实验和理论推导证明了的权重分配：

$$\begin{cases} W_r = 0.299 \\ W_g = 0.587 \\ W_b = 0.114 \end{cases}$$

被广泛应用，已经成为一种事实上的默认标准，即：

$$\text{gray}(x, y) = 0.299 \times R(x, y) + 0.587 \times G(x, y) + 0.114 \times B(x, y) \quad (\text{公式 2-2})$$

它是图像灰度化处理最常用的一个公式，也是 YIQ 颜色模型的基础。查看代码就会发现，著名的开源计算机视觉库 OpenCV<sup>9</sup>也是用了这一权重。当权重是浮点数时，可以将其转化为定点整数乘除法甚至位移运算<sup>[10]</sup>以提高运算效率。

2、平均值法：R、G、B 三者求和取均值，即：

$$\text{gray}(x, y) = \frac{R(x, y) + G(x, y) + B(x, y)}{3} \quad (\text{公式 2-3})$$

平均值法会产生比较柔和的灰度图像。

3、单通道法：取 RGB 中的一个值作为灰度化值，即

$$\text{gray}(x, y) = \begin{cases} R(x, y) \\ G(x, y) \\ B(x, y) \end{cases} \quad (\text{公式 2-4})$$

4、最大值法：取 R、G、B 三个值中最大的一个，即：

$$\text{gray}(x, y) = \max(R(x, y), G(x, y), B(x, y)) \quad (\text{公式 2-5})$$

最大值法会使图像的整体亮度增强。

5、最小值法：取 R、G、B 三个值中最小的一个，即：

$$\text{gray}(x, y) = \min(R(x, y), G(x, y), B(x, y)) \quad (\text{公式 2-6})$$

以加权平均值法为例，伪代码如下：

加权平均值法灰度化的算法

输入： in, 指向原始图像像素矩阵的指针

out, 保存灰度化后图像矩阵的指针变量

n, 图像行数

m, 图像列数

输出： out, 指向灰度化后图像矩阵

convert-to-gray(in, &out, n, m)

```

1 out ← 新建单通道灰度图像
2 for row ← 1 to n
3     for col ← 1 to m
4         out[row][col] ← (0.299×in.R[row][col] + 0.587×in.G[row][col]
5                             + 0.114×in.B[row][col]);
6 return
```

<sup>9</sup> <http://www.opencv.org>

以 linuxfr.org 所用验证码为例，几种灰度化效果如下：

表 2.1 灰度化效果对比

原图	加权平均(公式 2-2)	平均值	最大值
最小值	R 通道	G 通道	B 通道

可以看到，加权平均（公式 2-2）对图像整体分布保存较好。而在这一特殊样例中，B 通道灰度化方法直接起到了去掉背景图像杂块的作用。

## 2.2 二值化

二值图像是每个像素只有两个可能值的数字图像。二值化是指将图像的灰度级减少到 2。假设为 0、1，则二值图像中的像素灰度值只取 0 和 1 两种值。一般所说的二值化，是将 256 个亮度等级的灰度图像通过选取特定的阈值将灰度值大于等于阈值的像素被判定为前景，小于阈值的像素判定为背景。二值化的过程很简单，选取一个阈值<sup>10</sup>，把大于这个临界值的像素设为 1，小于这个值的像素设为 0，从而实现二值化。设  $t$  为阈值，有：

$$F(x, y) = \begin{cases} 1 & F(x, y) \geq t \\ 0 & F(x, y) < t \end{cases} \quad (\text{公式 2-7})$$

根据阈值选取的方式不同，二值化算法可分为固定阈值和自适应阈值。

固定阈值的  $t$  取某一特定值。一些针对性很强的简陋的验证码识别系统通常选取特定值来简化实现。而在数字图像处理中得到广泛应用的是自适应阈值。其中最广泛的要数日本科学家大津展之提出的大津算法，或称 OSTU 算法，又叫最大类间方差法。它被认为是图像分割中的最佳阈值选取算法，以不受图像亮度、对比度影响和计算简单而著称。假设按图像的灰度特性，将图像分成背景和前景两部分。则背景和前景之间的类间方差越大，说明构成图像的两部分的差别越大，因此，使类间方差最大的分割也就意味着错分概率最小，OSTU 也就是求得使类间方差最大的阈值并

<sup>10</sup> 阈 yù。翻阅的很多资料会误写作“阀(fá)值”，是错别字，“阈”无临界、边界之意。

将其作为最佳阈值的过程。

设阈值  $t$  将灰度图分成 A、B 两类，即背景和前景。在实际应用中，往往取下述公式：

$$D(t) = P_a \times (\mu_a - \mu)^2 + P_b \times (\mu_b - \mu)^2 \quad (\text{公式 2-8})$$

其中， $D(t)$  为两类间方差， $P_a$  为 A 类概率， $\mu_a$  为 A 类平均灰度， $P_b$  为 B 类概率， $\mu_b$  为 B 类平均灰度， $\mu$  为图像总体平均灰度。

伪代码如下：

OSTU 求阈值算法

输入：img, 指向原始图像像素矩阵的指针

输出：int 型阈值 (0-255)

OSTU (img)

```

1  for i ← 1 to img->height
2      for j ← 1 to img->width
3          灰度计数器 grayCnt[img[i][j]]++
4      计算平均灰度 aveGray 和总灰度 sumGray
5  for i ← 1 to 255
6      for j ← 0 to i-1
7          a += grayCnt[j]
8          asum += j*grayCnt[j]
9      for j ← i to 255
10         b += grayCnt[j]
11         bsum += j*grayCnt[j]
12     double tmp ← a*1.0/sum*(asum*1.0/a - aveGray)*(asum*1.0/a - aveGray)
13             + b*1.0/sum*(bsum*1.0/b - aveGray)*(bsum*1.0/b - aveGray)
14     if (tmp > maxTheta)
15         maxTheta ← tmp
16         threshold ← i
17 return threshold

```

以 thedailywtf.com 所用验证码为例，效果如下：

表 2.2 OTSU 二值化(1)

原图

灰度化

OTSU 二值化 (此时阈值 45)


可以看到，OTSU 算法对背景和前景区别明显的图片效果尤其突出，即图像的灰度直方图呈现“双峰”特征。而对于前景背景较为接近的情况，效果不好。

表 2.3 OTSU 二值化(2)

原图

灰度化

OTSU 二值化 (此时阈值 62)


## 2.3 去边框和反色

需要注意的是，有些验证码故意加上了边框，使得自动计算的阈值偏离真实阈值，且为后续分割制造难度，一定要去掉。去边框的方法可以采用 flood-fill 算法从左

上角开始将同色像素染成反色，这样只要前景与边框无粘连且不同色，无论边框颜色和厚度如何变化，只要边框同色均可有效除去。最好在二值化后再次使用画白色矩形框的方法去除某些变色边框。flood-fill 的深度优先搜索（DFS）实现伪代码如下：

Flood-fill 去边框

输入： `img`, 指向原始图像像素矩阵的指针

`x`, 当前搜索像素的 `x` 坐标

`y`, 当前搜索像素的 `y` 坐标

输出： 无

`clear-frame (img, x, y)`

1 **for** 上下左右四个方向

2 **if** 未被搜索且同左上角同色

3 标记此位置 (`tx`, `ty`)

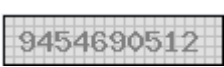
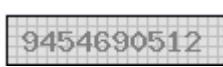
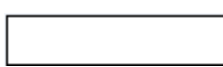

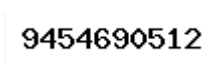
4 `img[tx][ty] ← 255 - img[tx][ty]`

5 `clear-frame(img, tx, ty)`

6 **return**

以 SCODE<sup>11</sup>验证码为例，效果如图：

表 2.4 去边框的作用

1、原图	2、灰度化	3、OTSU 二值化
		
4、去边框	5、去边框后二值化	
		

一些验证码是黑底白字，为了方便处理，我们判断当黑色像素数量多于白色像素数量时使图像灰度值取反，伪代码和样例略。

## 2.4 滤波去噪和平滑

在数字图象处理中，我们常常使用平滑操作达到去噪的效果。数字图像的平滑处

<sup>11</sup> 原图摘自 <http://caca.zoy.org/wiki/PWNtcha>



理就是利用一个  $n \times n$  ( $n$  为奇数) 的窗口和一个逻辑表达式，在像素矩阵上移动，依次过滤每个像素点的过程。平滑处理尤其是中值滤波往往也能达到消除孤立噪声的作用。常用的平滑处理方式有：

1、均值滤波。均值滤波是典型的线性滤波算法。它用窗口中像素灰度均值代替中心像素的灰度值，以  $3 \times 3$  窗口为例：

$$p(i,j) = \frac{\sum_n \sum_m p(i+m,j+n), (m,n=-1,0,1)}{3 \times 3} \quad (\text{公式 2-9})$$

均值滤波存在着固有缺陷：它不能很好地保护图像细节，使图像变得模糊，而且不能很好地去除噪声点。对二值图像来说，它会破坏二值图像（因为灰度值可能取到 0 和 255 的中间值）。实现方法参考中值滤波。

2、中值滤波。中值滤波是典型的非线性平滑技术。它使用窗口中像素的中值代替中心像素灰度值，以  $3 \times 3$  窗口为例：

$$\begin{cases} X = \{p'(i+m,j+n) \mid m,n = -1,0,1\} \\ P'(i,j) = \text{median}(X) \end{cases}, \text{median}(X) \text{ 表示对集合 } X \text{ 取中值} \quad (\text{公式 2-10})$$

中值滤波是使用频率最高的消除“盐椒噪声”的方法，可以很好的保存图像边缘。中值滤波用于图像修复效果显著，而在验证码识别中可以起到去细线的作用。原理如图所示：

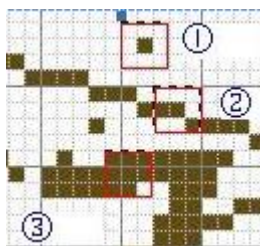


图 2.1 中值滤波的去噪点和细线原理

当窗口中黑色像素少于 5 个时，取中值操作就会把中央像素点标记为白色。因此，在二值图像中完全可以用窗口中黑色像素数的“阈值法”代替。

中值滤波伪代码如下：

中值滤波

输入：img, 指向原始图像像素矩阵的指针

输出：out, 指向滤波后图像矩阵的指针

median-smooth (img)

```
1 for (int i ← 0; i < img->height; ++i)
```



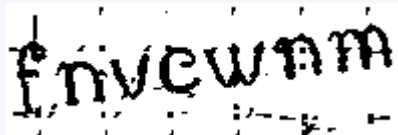
```

2     for (int j ← 0; j < img->width; ++j)
3         for 3×3 像素窗口
4             灰度值排序
5             out[i][j] ← 灰度中值
6     return

```

以 slashdot.org<sup>12</sup>所用验证码为例，中值滤波的效果如图：

表 2.5 中值滤波

原图

OTSU 二值化

中值滤波


可以看到，消除了所有的盐椒噪声和所有宽度为 1 像素的细线的非交点部分。

3、Unger 平滑<sup>[11]</sup>。Unger 于 1959 年提出的一种对二值图像的平滑算法。Unger 平滑检查窗口的细节而不是简单统计分析。Unger 平滑不能去掉细线，但是它可以用来填补凹槽和空洞、去除孤立点和边缘突起。遗憾的是，它对验证码的平滑作用并不明显，但可以用作补充处理。假设  $3 \times 3$  窗口矩阵如下，

$$\begin{array}{ccc}
 n3 & n2 & n1 \\
 n4 & p & n0 \\
 n5 & n6 & n7
 \end{array}$$

平滑方法为：

(1) 当  $p$  为白色时，如果  $n0, n2, n4, n6$  至少三个为黑色，则将  $p$  改为黑色，否




<sup>12</sup> 原图摘自 <http://caca.zoy.org/wiki/PWNtcha>

则  $p$  不变。这时  $p$  可用逻辑表达式表示为： $p = n_2n_6(n_0 + n_4) + n_0n_4(n_2 + n_6)$

- (2) 当  $p$  为黑色时，如果  $n_2, n_3, n_4$  中至少有一个是黑色，且  $n_0, n_6, n_7$  中至少也有一个为黑色时，或  $n_0, n_1, n_2$  中至少有一个为黑点且  $n_4, n_5, n_6$  中至少有一个为黑点时，则  $p$  不变，否则变为白色。这时  $p$  可用逻辑表达式表示为： $p = (n_2 + n_3 + n_4)(n_0 + n_6 + n_7) + (n_0 + n_1 + n_2)(n_4 + n_5 + n_6)$

伪代码略。以某验证码的中间处理步骤为例，经过 Unger 平滑后，小凹槽和突起明显平滑了。而这种状态下若使用中值滤波处理则会丢失某些主干特征。

表 2.6 Unger 平滑

某中间步骤	Unger 平滑	中值滤波
		

## 2.5 连通性去噪

此外，对于平滑操作无法消除的孤立噪声，我们可以用判断连通域面积的方法消除小面积噪声，效果显著。伪代码如下（其中 `dfs` 函数是深度优先搜索，并用数字 `cnt` 标记像素所在连通域，伪代码略）：

连通性去噪（原地操作）

输入： `img`, 指向原始图像像素矩阵的指针

`t`, 噪声面积阈值

输出： 无

```
anti-noise (img, t = 5)
1  for (int i ← 0; i < img->height; ++i)
2      for (int j ← 0; j < img->width; ++j)
3          if (map[i][j] == -1)
4              dfs(img, cnt, dir, map, i, j)
5              ++cnt
6  for (int i ← 0; i < cnt; ++i) area[i] ← 0
7  for (int i ← 0; i < img->height; ++i)
8      for (int j ← 0; j < img->width; ++j)
9          ++area[map[i][j]]
```





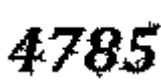
```

10 for (int i ← 0; i < cnt; ++i)
11     if (area[i] > t) area[i] ← 0
12 for (int i ← 0; i < img->height; ++i)
13     for (int j ← 0; j < img->width; ++j)
14         if (area[map[i][j]])
15             ptr[i*img->widthStep+j] ← 255
16 return

```

以 ibc168.com 所用验证码为例，连通性去噪效果如图：

表 2.7 连通性去噪

原图	灰度化	OTSU 二值化	反色后	连通性去噪
				

## 2.6 细化

有些验证码往往将不同粗细的字符和干扰线掺杂在一起。细化即找出图像的骨架，减少字符粗细变化带来的不便。但细化也可能将原本明显的杂质与字符主干拼接起来难以分别。目前没有针对验证码的通用细化算法。对于纯数字组成的验证码来说，细化过程可以提升识别的准确率<sup>[16]</sup>，减少干扰。下面以数字图象处理中最常用的 Hilditch 算法介绍细化。

Hilditch 算法<sup>[18]</sup>是 N. J. Naccache 和 R. Shinghal 于 1984 年提出的<sup>[17]</sup>，是二值图像细化的经典算法。这一算法通过一个  $3 \times 3$  的窗口在图像上扫描并由一组规则来处理。如果一个像素满足所有规则，则将其标记为删除。这一扫描过程重复执行直到某次循环没有删除任何点。

P9	P2	P3
P8	P1	P4
P7	P6	P5

$3 \times 3$  窗口如上。定义几个函数：

- 交叉数 (crossing-number) 为:  $X(P_i) = \sum b_i$  ( $i$  从 2 到 5)。其中若  $P(2i-2)$  为白色并且  $P(2i-1)$  和  $P(2i)$  二者至少有一个是黑色则  $b_i=1$ ，否则  $b_i=0$ ；

- 2、  $B(P_i)$  为  $P_i$  的八邻域中黑点的个数；
- 3、  $N(P_i)$  是  $P_i$  的 8 邻域中未标记删除的黑点数；
- 4、  $X_m(P_i)$  为假设  $P_m$  为白色得出的  $P_i$  的交叉数；

判断是否标记删除  $P_1$  的六个规则为：

$$H1: P_2 + P_4 + P_6 + P_8 \leq 3$$

$$H2: B(P_1) \geq 2$$

$$H3: N(P_1) \geq 1$$

$$H4: X(P_1) = 1$$

$$H5: P_4 \text{ 未被标记删除或者 } X_4(P_1) = 1$$

$$H6: P_6 \text{ 未被标记删除或者 } X_6(P_1) = 1$$

这些条件保证了删除时只从边界点开始删除且不会过度腐蚀。伪代码略。

以 sohu 和 thedailywtf 网站的验证码为例：

表 2.8 细化算法

原图	细化后
	
	

## 2.7 去干扰线

较短的干扰线可以通过连通性去掉。稍长一些的干扰线则很难处理。单纯找最长直线的方法可能会把字符的一部分也去掉导致其特征丢失。对于宽度为 1 的贯穿验证码的干扰线来说，可以发现这种干扰线的斜率通常变化不大，因此可以设计一个估价函数：

$$\text{cost}(x, y) = \max\{\text{cost}(k, y - 1) + \text{Height} - |y - k|\}, \text{ 其中 } 1 \leq k \leq \text{Height} \quad (\text{公式 2-11})$$

逐列扫描后，选择每列函数值最大的像素作为干扰点去掉。伪代码如下：

去干扰线（原地操作）




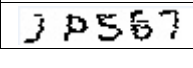
输入： `img`, 指向原始图像像素矩阵的指针

输出： 无

```
anti-line(img)
1  for (int y ← 1; y < img->width; ++y)
2      for(int x ← 0; x < img->height; ++x)
3          if (img[x][y] == 0)
4              int max ← map[0][y-1] + img->height - (x-0)
5              for (int k ← 1; k < img->height; ++k)
6                  int tmp ← map[k][y-1] + img->height - abs(x - k)
7                  if (tmp > max) max ← tmp
8              map[x][y] ← max
9          else map[x][y] ← 0
10 for (int y ← 0; y < img->width; ++y)
11     int maxx ← 0, max ← 0
12     for(int x ← 0; x < img->height; ++x)
13         if(map[x][y] > max)
14             max ← map[x][y]
15             maxx ← x
16     img[maxx][y] ← 255
17 return
```

以飞信客户端和新浪微博手机网页端所用验证码为例：

表 2.9 去干扰线

原图	去干扰线后
	
	

如果加上对删除点上下像素的颜色判断则可以解决字符内部被切割的问题。

## 3 分割和识别

### 3.1 分割

常用的字符分割有投影法和连通性分割法等等。

#### 3.1.1 投影法

投影法对无粘连的验证码效果良好。投影法在数字识别以及车牌识别中应用广泛。一般的做法是先在 X 轴上投影，根据投影值为 0 的列确定单个字符所占的宽度；再向 Y 轴投影，同理，得到字符高度，从而最终将整个字符提取出来。对于在投影方向上有粘连的验证码也可以使用，此时投影的波谷一般即可作为分割点，再加上字符宽度范围限制就可很好的把字符准确分割出来。投影法对图像倾斜度以及预处理的要求很高，某些看似孤立的噪声或倾斜过大的字符都可能导致相邻字符在投影图上的重合，使谷底不明显，从而影响切割精确度和效果。一个验证码在 X 轴上的投影图如图所示，伪代码略：

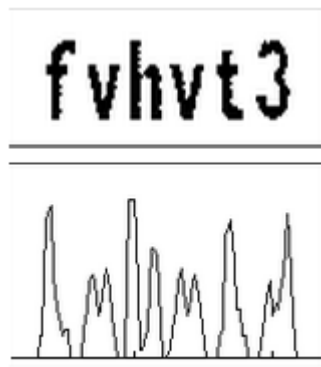


图 3.1 投影图



图 3.2 按 X 轴分割图

#### 3.1.2 连通性分割

连通性分割法利用了每个字符或字符的一部分构成连通域的特征，通过求连通区

域把字符分割开。这种方法不受图像倾斜或旋转的影响，但要求图像中字符的主体部分应当连通且不与噪声和其他字符连通，否则字符的特征会丢失，导致难以识别。其原理与连通性去噪相同，此处不详述，伪代码略。

## 3.2 识别

虽然 OCR 技术已经比较成熟，但往往即便经过预处理的验证码字符形变依然严重，很难直接使用 OCR 系统识别。验证码识别系统常常借鉴 OCR 系统的工作原理“训练”一个新系统。常见的识别方法主要有两类：对图像小、字符少的验证码使用基于模板的匹配识别，根据特征相似度识别；以及应用各种各样的人工神经网络来识别，例如 BP、SVM、KNN 等等，效果比模板好得多。



## 4 系统设计与实现

### 4.1 实用性论证

我们以识别弹幕视频网站 bilibili.tv 的注册验证码为例验证用上文各算法和 OCR 引擎结合来识别验证码的正确率，这类验证码很有代表性，可以很好地验证我们的验证码识别演示程序的实用性。

我们批量下载了 300 张验证码，下载程序代码（Python 语言）如下：

```
import urllib
import urllib2

def write_file(file, s):
    f=open(file, 'wb')
    f.write(s)
    f.close()

def save_image(i):
    url = "https://secure.bilibili.tv/captcha"
    user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
    values = {}
    headers = { 'User-Agent' : user_agent }
    data = urllib.urlencode(values)
    req = urllib2.Request(url, data, headers)
    try:
        response=urllib2.urlopen(req, timeout = 10)
        a = response.read()
        write_file('downloadCAPTCHA/Image%03d.png'%i, a)
    except Exception, e:
        print "ERROR, rerun later...", e
        save_image(i)
    else:
        print "Complete: ",i

for i in range(1, 301):
    print 'Starting:',i
    save_image(i)
```

首先观察这种验证码，发现前景和背景的颜色差距很大，因而用 OTSU 二值化应该可以完美分离出前景；背景上的干扰线很细，虽然形状毫无规律，但应当可以通过中值滤波去掉（事实证明也如此）；字符大小无变化，不需要细化处理。



图 4.1 部分用例

我们从前文的讨论中可以得到这一验证码的主要预处理步骤：灰度化、OSTU 二值化、反色、中值滤波、连通性去噪、Unger 平滑。上图对应的处理后图像如下：



图 4.2 处理后的部分用例

而粘连字符的分割和识别则比较棘手。总的来看影响到识别的倾斜和粘连率并不高，我们可以放心的把它交给开源 OCR 引擎 Tesseract-OCR。Tesseract-OCR 最先由 HP 实验室于 1985 年开始研发，至 1995 年时已经成为 OCR 业内最准确的三款识别引擎之一。然而，HP 不久便决定放弃 OCR 业务，直到 2005 年由 Google 接手并重新以开源软件方式发布。Tesseract 被认为现今世界上最精确的免费 OCR 引擎。像所有 OCR 系统一样，Tesseract 对于含有噪点，噪线，旋转，扭曲，粘连的验证码的识别率极低。

对所有 300 个样本的识别函数代码（C++）如下：

```
void capt ()
{
    FILE* fout = fopen("out1.txt", "w");
    for (int i = 1; i <= 300; ++i)
    {
        char s[50];
        sprintf(s, "bbed\\Image%03d.png", i);
        img = cvLoadImage(s);
        tesseract::TessBaseAPI tess;
        tess.Init(NULL, "eng");
        IplImage* imgh = img;
        tess.SetImage((uchar*)imgh->imageData, imgh->width
                      , imgh->height, imgh->nChannels, imgh->widthStep);
        tess.SetPageSegMode(tesseract::PSM_SINGLE_WORD);
        tess.SetVariable("tessedit_char_whitelist"
                        , "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ");
        char* out = tess.GetUTF8Text();
        fprintf(fout, "%d:%s", i, out);
    }
    fclose(fout);
}
```

看看它在我们预处理后的图像上工作得如何：

表 4.1 试验结果

	个数	比例
正确	55	18.3%
错误	245	81.7%

其中，由于字体中 1、7、T；2、z；0、Q；4、A 的相似性带来的混淆很多。可以设想，通过字体训练可以大大提高识别正确率。但可惜的是，一方面我们不可能训练到所有的字体；另一方面，Tesseract-OCR 对训练集中字体的宽高比有大小要求，小尺寸的验证码无法训练。

总之，这一实验结果证明上文的预处理算法和 OCR 引擎的“合作”可被用于演示程序。而对于无粘连和微倾斜验证码，如上网提到的 SCODE 和 ibc168.com 的验证码，使用相同的过程则可以达到 100% 的识别率。

## 4.2 系统结构与实现

验证码识别演示系统由图像处理端和图形界面端构成。前者使用 C++ 语言以加快处理速度，并且针对不同图像格式的问题，借助 openCV 1.0 库中的 `IplImage` 结构实现不同格式图像数据在内存中的统一，其他算法实现同第二章。最终打包为 DLL 文件方便与界面端的通信；后者使用 Python 写成。系统结构如图所示：

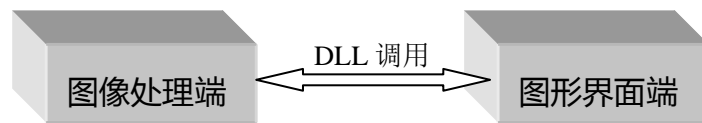


图 4.3 系统结构图

根据对上文实验的总结，验证码识别演示系统逻辑流程图如下：

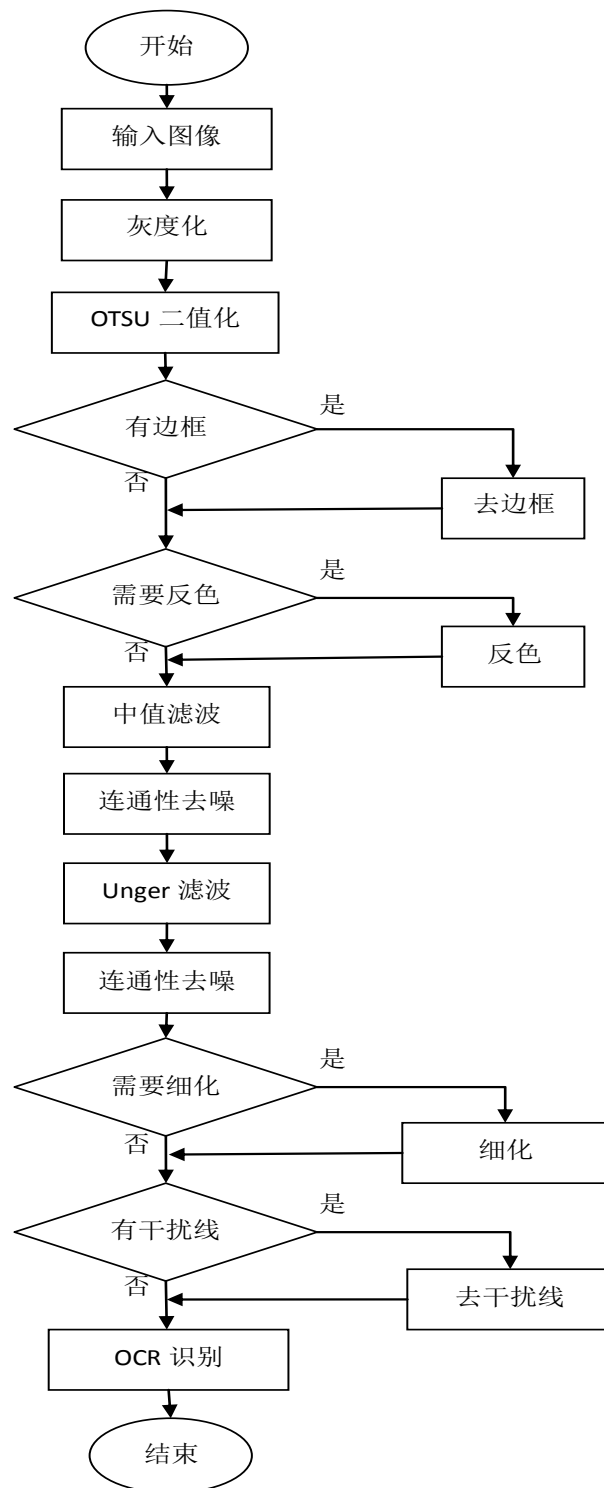


图 4.4 逻辑流程图

由此编写的验证码识别 DEMO 程序可以成功识别无粘连、含噪点的验证码。各算法实现方法同文中对应伪代码，此处略。仅给出程序界面截图如下：



图 4.5 界面截图

## 结论

验证码的生成和识别技术相辅相成，互相推动，成就了验证码今天的重要地位。其中，应用最多的要数基于字符的图片验证码。

本文研究了此类验证码识别预处理步骤的相关算法。主要包括灰度化、二值化、滤波去噪、连通性去噪、细化、去干扰线的基础算法，以及相关步骤中的处理细节。给出了包括五种灰度化、三种滤波平滑、OTSU 二值化、Hilditch 细化算法在内的所有步骤的原理及伪代码，并借助开源 OCR 引擎给出了可以成功识别无粘连、含噪点的验证码识别演示程序。

这些算法中的多数在验证码识别的文献中很常见，但大都一笔带过，并无对算法原理的详细说明，本文是对它们的总结。同时，去干扰线的算法是一个创新。

实验证明上述预处理方法的综合使用可以成功解决含噪点、无粘连、只有一根贯穿型细干扰线的验证码。并且再次印证，对验证码而言，没有一个统一的算法可以一次性解决所有的验证码预处理和识别问题。

本文总体上只是在验证码识别领域进行的小小尝试，使用的技术也是最基础、最通用的简单技术。通过研究，使我对于数字图像处理的相关技术有了初步了解。

## 致谢

大学四年是我成长的四年，既丰富了专业知识、锻炼了动手能力，又结交了很多朋友。衷心感谢关心、帮助、支持过我的家人、老师和同学们。

感谢各位认真负责的任课老师们，你们的严谨求实、兢兢业业、认真负责和一丝不苟的态度值得我学习；感谢所有同学们，你们对于学业和人生的态度深深影响了我，令我不断反思、不断改正、不断进步；感谢合肥工业大学软件学院为我提供的良好学习环境；

另外，我还要真诚感谢我的家人对我生活和学习上的支持和鼓励。

由于本人知识所限和时间仓促，论文中会难免有一些疏漏以及不妥之处，恳请各位老师批评指正。



## 参考文献

- [1]: L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart (automatically) [J]. CMU Tech Report, February 2002, CMU-02-117.
- [2]: Carnegie Mellon University. CAPTCHA: Telling Humans and Computers Apart Automatically. <http://www.captcha.net> .
- [3]: National Science Foundation. Human Interactive Proofs (HIPs). <http://www.aladdin.cs.cmu.edu/hips/> .
- [4]: T. Pavlidis. A TUTORIAL ON CAPTCHA. <http://www.theopavlidis.com/technology/captcha/tutorial.htm>, March 2008.
- [5]: Alan M. Turing. Computing machinery and intelligence [J]. Mind, October 1950, 59(236):433 - 460.
- [6]: 百度推广. 百度联盟验证码推广 抓住“7 秒钟”的推广机会. <http://e.baidu.com/news/content/2012-07-27/1347354782.html>, 2012.
- [7]: 维基百科. HSL 和 HSV 色彩空间. [http://zh.wikipedia.org/wiki/HSL\\_色彩空间](http://zh.wikipedia.org/wiki/HSL_色彩空间), 2013.
- [8]: 文晓阳, 高能, 夏鲁宁, 荆继武. 高效的验证码识别技术与验证码分类思想, 计算机工程 [J], 2009, 35 (8): 186-188.
- [9]: 吕刚. 带干扰的验证码识别研究 [D]. 杭州: 浙江工业大学, 2009.
- [10]: 博客园. 由图像的灰度化看基本图像处理. <http://www.cnblogs.com/Thinknet/archive/2009/01/13/1375106.html>, 2009.
- [11]: S. H. Unger. Pattern detection and recognition [J]. Proceedings of the IRE, 1959, 1737~1752.
- [12]: Greg Mori, Jitendra Malik, 2003, Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA [J], IEEE Conference on Computer Vision & Pattern Recognition, IEEE Computer Society, 1(124-141)
- [13]: Li, Shujun, Syed Amier Haider Shah, Muhammad Asad Usman Khan, Syed Ali Khayam, Ahmad-Reza Sadeghi and Roland Schmitz. Breaking e-Banking CAPTCHAs [A]. Proceedings of 26th Annual Computer Security Applications Conference (ACSAC 2010) [C]. New York, USA: ACM. 2010, 171 - 180.
- [14]: Kumar Chellapilla, Kevin Larson, Patrice Y. Simard, Mary Czerwinski. Computers beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (HIPs). In CEAS 2005 - Second Conference on Email and Anti-Spam, July 21-22, 2005, Stanford University, California, USA. 2005.
- [15]: Kurt Alfred Kluever, Independent Study Report Character Segmentation

and Classification, Department of Computer Science, Golisano College Of Computer and Information Sciences, Rochester Institute of Technology, Februray 28, 2008.

- [16]: 王虎, 冯林, 孙宇哲. 数字验证码识别算法的研究和设计[J]. 计算机工程与应用, 2007, 43(32): 86-87.
- [17]: N. J. Naccache and R. Shinghal. SPTA: A proposed algorithm for thinning binary patterns[J]. IEEE Trans. Systems, Man, Cybernetics, 14:409 - 418, 1984.
- [18]: Skeletonization Algorithm. <http://www.iasj.net/iasj?func=fulltext&aId=34184>