# Design and Implementation of a CAPTCHA Cracker

*Course Project of Computer Vision, Winter, 2009*

张驰原 20921038, ZJU

2010 年 1 月 10 日

## 1 任务简介

CAPTCHA (completely automated public Turing test to tell computers and humans apart) 是一种区别人和计算机的测试，也就是我们常常说的“验证码”。他们通常出现在网站的登录对话框，游戏的防外挂系统中，用来防止使用计算机来进行作弊或者攻击。这种测试通常是由一台计算机（Server）问用户一个 CAPTCHA 问题，由于这种问题对计算机来说往往是很难解的，而对人来说却是非常容易的，所以只要能够通过这种测试，那么被测试者就会被认为是一个人，而不是一台机器。

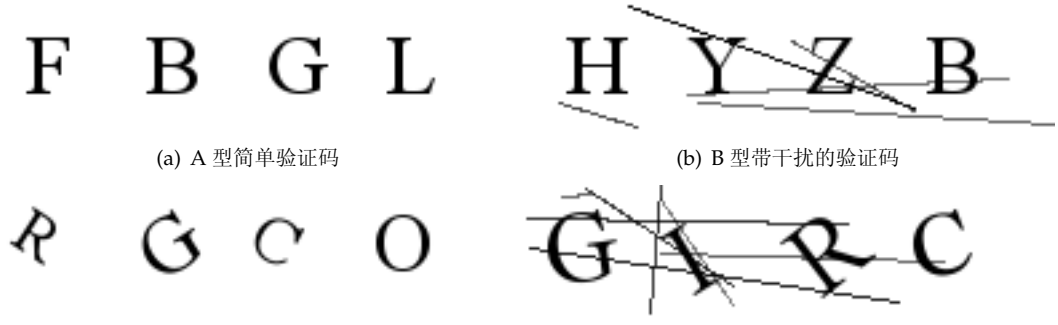
早期的验证码大多是由简单的字母组成的图像，但是随着 OCR 技术的日益完善，这种技术已经无法再判断是否是机器还是人了。于是新的验证码尝试在验证码的背景中加入各种噪音线条，并把内容并在一起，大大加大了计算机识别的难度。不过，随着计算机视觉等相关领域的技术发展，对于一些经过简单变换的 CAPTCHA 的识别也渐渐变得可行。本 Project 的目的就是设计并实现一个简单的验证码识别器，以期望在这个过程中能了解并熟悉相关的各种计算机视觉的算法和工具。

为了避免情况过于复杂，我将从 MSTC 第四届趣味程序设计竞赛中 gbb21 同学出的一道题目出发，那道题目的目标正是要对验证码进行识别，需要处理的验证码主要分为四种类型：

- A 型：最简单的验证码，无任何干扰和变换。
- B 型：带干扰的验证码，干扰是随机加入的一些线条。
- C 型：带旋转缩放的验证码，每个字母被随机地旋转（-60°∼ 60°）和缩放（50% ∼ 150%）。
- D 型：综合验证码，在经过旋转和缩放变换之后又加入了随机线条的干扰。

四种类型的验证码如图 1 所示。

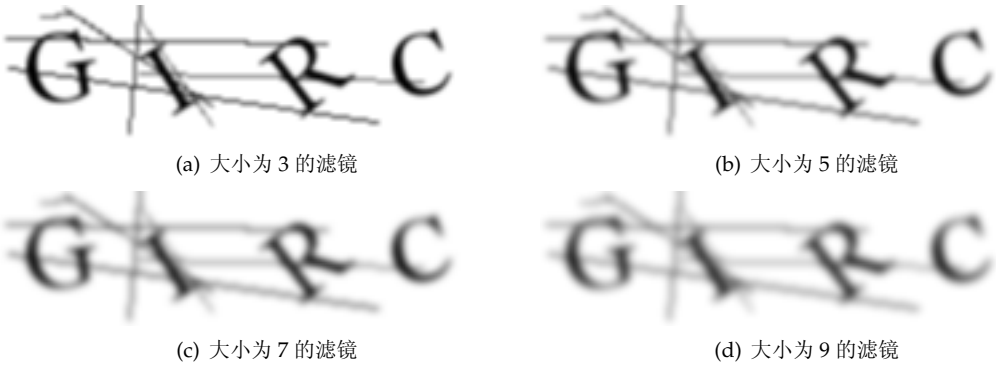
我直接采用该题目所提供的 Judge 程序来生成测试和训练用的验证码图片，该程序可以生成上面所描述的四种验证码图片，并且可以保证：



(a) A 型简单验证码　(b) B 型带干扰的验证码

(c) C 型旋转缩放验证码　(d) D 型综合验证码

图 1: 验证码示例



(a) 大小为 3 的滤镜　(b) 大小为 5 的滤镜

(c) 大小为 7 的滤镜　(d) 大小为 9 的滤镜

图 2: 高斯滤波的结果

- 每张图片都是大小位 200 × 50 的 BMP 图片，背景为白色，内容和干扰均为灰阶颜色，由于字体渲染引擎的平滑效果可能会在字体边缘出现一些其他颜色。
- 每张图片中内容为 4 个大写字母，在图片上从左往右排列且互不相连。
- 字母字体为 Times New Roman，为变换前字号为 24 。

# 2 算法设计与对比

对于 A 型验证码的识别是很简单的，而 D 型其实是 B 型和 C 型的结合，因此我主要将注意力集中在 D 型验证码的识别之上。我将这个问题分解为两个步骤：

1. 提取单个字母，要提取出字母，难点在于干扰背景的过滤。
2. 识别单个字母，这一步的难点在于如何处理经过了旋转和缩放的字母。

可以看出以上两个步骤其实可以说是专门针对了 B 型和 C 型验证码的解决方案，结合起来就可以完成 D 型验证码的识别了。

## 2.1 噪音过滤与字母提取

注意到噪音都是一些简单的直线，我首先想到的办法是通过滤波的方法将图片模糊化以达到去除线条的办法，另外较为复杂的方法就是采用直接检测直线的方法。下面分别来试验各种方法的效果。

### 2.1.1 滤波方法去除噪音

最常用的滤波方法就是高斯滤波了，OpenCV 提供了现成的实现，图 2 给出了对图 1(d) 中的图片进行高斯滤波的结果，这里给出了滤镜大小分别为 3、5、7 和 9 时的结果。可以看出高斯滤波器并不能直接把线条噪音去掉，不过确实可以将噪音变淡，对比一下各个参数的结果，在大小为 5 的滤镜下得出的结果里字母和噪音之间的区分度较大一些，图 3 中的结果也证实了这一点，从这张图来看，采用大小为 5 的高斯滤波器加上一个 threshold 操作[1]是一个不错的选择，有效地去除了噪音，同时也保留了字母的重要细节使其仍然较易辨认。当然，这个方法也并不总是完美的，如图 4 所示，有时候字母本身一些较细的线条也会被当作“噪音”给过滤掉。

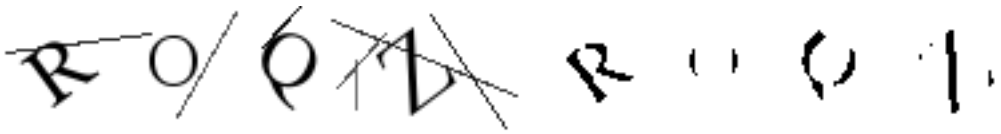
除了高斯滤波之外，还有一种滤波器是中值滤波器，它不是一个线性滤波器，由于他选取一个中值，这是图片中原本有的颜色，而不是像普通线性滤波器那样通过一个公式求出一个颜色值，因此许多时候有很好的性质。对于这里的情况，通常在噪音线条的周围都是大片的空白区域，因此求中值之后有很大的几率会被“抹去”。因此，除了高斯滤波之外，我还尝试了中值滤波，结果如图 5 所示。

[1]这里采用的是 threshold 为 120 的二值化 threshold。

(a) 大小为 3 的滤镜　　(b) 大小为 5 的滤镜

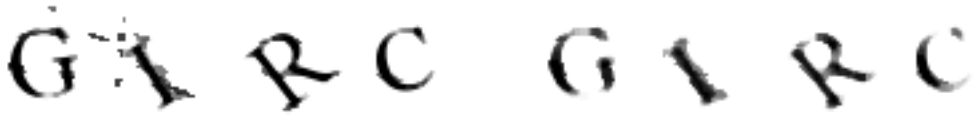
(c) 大小为 7 的滤镜　　(d) 大小为 9 的滤镜

图 3: 高斯滤波加上 thresold 的结果



(a) 原始图片　　(b) 高斯滤波的结果

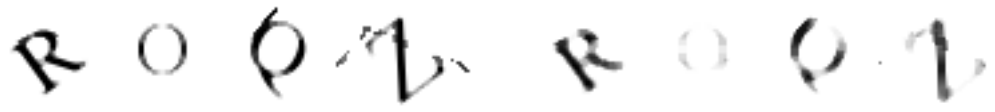
图 4: 高斯滤波器将字母 O 和 Z 的许多细节都抹去了



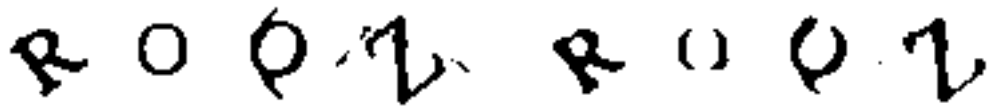
(a) 大小为 3 的滤镜　　(b) 大小为 5 的滤镜

(c) 大小为 7 的滤镜　　(d) 大小为 9 的滤镜

图 5: 中值滤波的结果

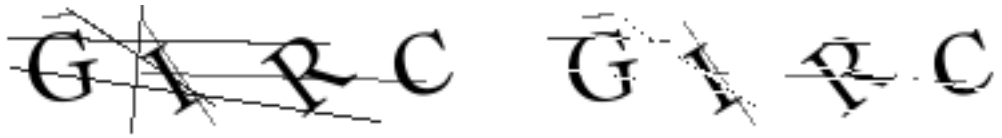


(a) 大小为 3 的滤镜　　(b) 大小为 5 的滤镜



(c) 大小为 3 的滤镜，二值化以后　　(d) 大小为 5 的滤镜，二值化以后

图 6: 中值滤波对于字母本身细节的损伤



(a) 原图　　(b) 去掉检测出的线段后的结果

图 7: 线段检测去除噪音的效果

需要注意的是，中值滤波不需要像高斯滤波那样还需要一个后续的 threshold 操作就可以直接去掉噪音线条了，可以看到同样是在滤镜大小为 5 时效果最好。不过，同高斯滤波器一样，有些时候中值滤波器对字母本身的损伤也比较大，如图 6 所示，不过，好在许多地方只是变淡了，并没有完全抹去，我在对图片进行二值化处理[2]之后可以看到整体效果还是比高斯滤波器好的。

### 2.1.2 其他去除噪音的方法

除了用滤波的方法之外，考虑到所有的噪音都是随机直线，因此也可以尝试用线段检测的方法。OpenCV 提供了现成的线段检测实现，只要将图片转换为二值图（黑色背景白色线段）就可以很方便的检测出线段来，然后再在原图中将检测出来的线段去掉，效果如图 7 所示。这里给出的结果是经过参数调整之后较得到的较好的情况，一方面要尽可能多地检测出噪音线条，另一方面又要保证不要误把字母的部分当作线条。结果可以看到有一些线条被很完美地去除了，但是还有一些线条却没能处理好，总体效果似乎并不理想。

另外，趣味程序设计竞赛中的那道题为了降低难度，还附加了一个条件就是干扰线条的颜色有很大的概率会淡于字体颜色，根据这个条件，其实我们可以简单地做一个 threshold 把非纯黑的颜色全部都去掉，结果如图 8 所示，虽然字体平滑的边缘都没有了，但是干扰线条也去得很干净。不过，这是该题目中故意设置的降低难度的条件，而且如果线条的颜色也是纯黑色就完全无法处理了，所以在这里我并没有采用这个方法来做噪音过滤。

最后综合上述算法的结果，我选择中值滤波的办法，由于在识别的过程中字体的形状是主要因素，并且在不同缩放尺度下做滤波会导致同一个字母的颜色深浅差异很大，因此我决定在中值滤波之后再加上一步二值化处理。

在去除噪音之后，由于事先限定了字母之间不存在重叠的情况，所以提取出单个字母就变得很简单了，我在这里直接使用了找出相对较大的空白区域然后进行分割的办法，将图片分为 4 份。当然，为了尽可能减少残留的噪音的影响，添加了一些简单规则和 threshold ，详细可以参考实现代码。

[2]实际上是进行了一次 threshold 为 250 的 threshold 操作。

(a) 原图　　(b) 去掉非纯黑的颜色后的结果

图 8: 直接去掉非纯黑颜色的效果

### 2.2 字母的识别

经过前面的预处理之后，问题变成了对单个字母的识别了，当然，不能简单的用得到的结果与标准字体进行比对，因为一来虽然经过了噪音过滤，但是并不能保证能过过滤完全了，也不能保证字符本身应有的细节没有被殃及；另一方面字符有可能被旋转或缩放过了。因此在这里考虑使用机器学习的方法来训练字符识别的模型，这实际上是一个多类的分类问题，这里我直接使用简单的 knn 分类器来进行识别。

knn 分类器的训练过程很简单，只要把训练数据存储下来即可，在对一个未知的字母进行识别的时候，只要在数据库中寻找 k 个与其最接近的数据，然后看这些数据中属于哪一类的占大多数，就选择这个类别作为结果。

分类器确定之后还需要确定的就是用什么 feature 来表达数据，我首先决定尝试近年来在 vision 领域提出的 *visual word* 的方法，也就是通过在图片中提取关键特征点，并使用原本处理文本数据的方法来进行图片处理的一套方法。

而具体到特征点的提取和表达，结合这个问题中字母经过了旋转和缩放这个性质，很自然地就想到了 SIFT 特征。SIFT 特征的全称是 Scale-invariant feature transform ，它的良好性质就是旋转和缩放的不敏感性。不过，由于专利的原因，OpenCV 里并没有包含 SIFT 的实现，因此，我在这里使用一个类似于 SIFT 的 SURF (Speeded Up Robust Features) 特征，它也具有旋转和缩放的不变性，并且根据作者的实验，该特征计算速度比 SIFT 要快，而且效果也很好。

由于每个字母的图片通常在 $50 \times 50$ 这个大小左右，提取出来的 SURF 特征点其实很少，一般不超过十个，我把所有训练数据收集到的特征点通过 kmeans 聚类到 M 个 cluster ，构成一个 M 个元素的 vocabulary 。而把图片转化为向量的过程为：

1. 构建一个 M 维的零向量，向量的每一维表示一个 visual word 在这张图片中的“词频”，
2. 提取图片的 SURF 特征点，
3. 对每个特征点，在 vocabulary 里寻找最接近的单词，增加该 word 的词频。

这样，对于每个图片都能得到一个 M 维的向量。因此，在 knn 求最小距离的时候，只需要直接按照向量的欧氏距离来求解就可以了。实验的结果是，经过各种参数调整，最好的情况识别的正确率大概能达到 70% 左右。

这个结果并不是特别好，因此我又尝试了传统的 Hand Written Digit 识别中经典的方法，大致就是首先将字符缩放到占满整个 $50 \times 50$ 的框，然后将这个二维灰度图片直接拉伸为一个向量作为原始 feature ，之后再使用 PCA 对其进行降维，得到一个较低维的向量作为最终识别用的 feature 向量。实验结果证明确实是很不错的，能达到大约 90% 左右的正确率。两种方法的性能对比如图 9 所示，可以看到 PCA 对于特征维度的变化并不太敏感，而且都比 SURF 的方法要好。

## 3 总结

关于噪音过滤算法的选择和对比，在上一小节中已经讨论得很详细了，虽然线段检测的方法更具有针对性一些，但是实际结果看来中值滤波相对于其他几种方法来说更好一些。而在于特征的选取方面，可以从结果看到老牌的 PCA 完美打败了新提出来的 visual word 和 SURF 特征结合的办法，一开始还是令我有些意外的，不过经过分析之后，其实这样的结果也并不是那么意外的。

注意到一开始我提到对于单个字母抽取 SURF 特征时能得到的特征点的个数几乎都是在 10 个以下，现在用 visual word 的方法来处理，就相当于在对一些只有 10 个单词不到的短文档进行处理，这就变成一个很困难的问题了，经过词频映射之后，及时我可以把 vocabulary 的大小设置为数百，可是得到的特征向量其实还是非常稀疏的，只有零星几个非零元素。事实上，再回到平常见到的 visual word 用于对象识别的应用上，对比这个问题，可见不

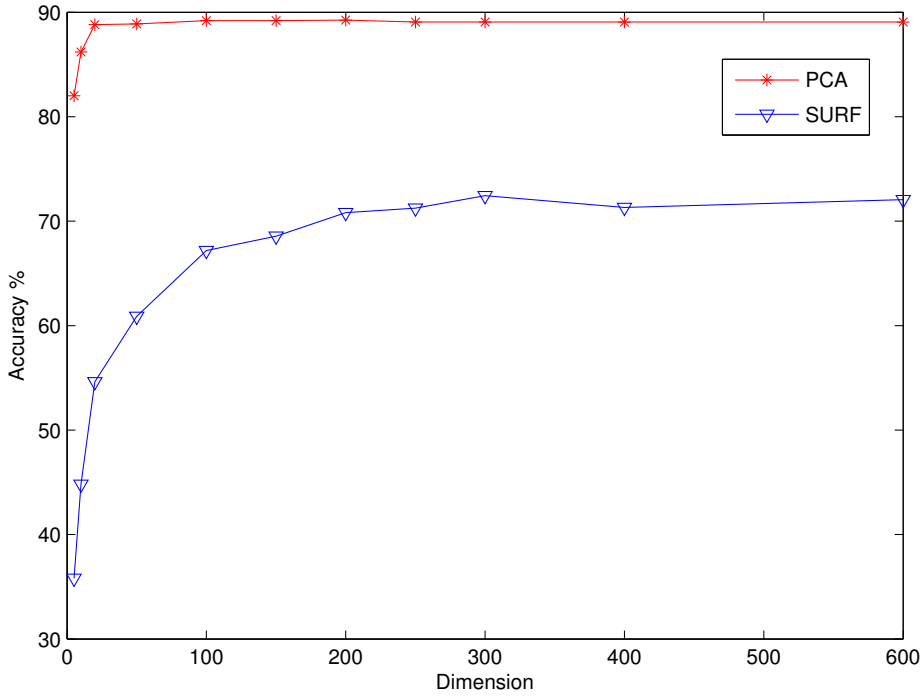



图 9: 不同特征维度下两种方法的精确度对比

管是单个图片里抽取出来的 SURF 特征点的个数还是最后 vocabulary 的大小，都完全不是一个数量级上的，SURF 善于处理那些场景复杂轮廓分明的大图，却不太适合于这里的结构简单并且还经过了滤波器模糊过的小图。

当然，对于真正的验证码来说，可能会有字母重叠，非线性的扭曲、变换以及各种更为复杂的噪音，要真正做到高精度地识别这样的验证码，仅靠 PCA 还是不够的，仍然需要挖掘更多视觉方面的理论和算法。当然，这本身就是一个非常困难且还未能较好地解决的难题。

通过这个 Project 的研究于实践，我了解了各种滤波器的工作原理和实现方式，熟悉了 OpenCV 提供的包括线段检测、SURF 特征提取等丰富的 CV 算法的编程接口，并通过和传统 PCA 方法的对比，加深了对 SURF 特征点算法和适用性的认识。

# 4 附录

## 4.1 代码说明

本实验的代码大部分使用 Python 写成，主要使用了 OpenCV 2.0 的 Python 接口，不过由于 OpenCV 提供的接口使用起来都非常麻烦，因此我在聚类和 knn 分类部分使用了 Numpy 和 Scipy 的接口。代码中部分文件介绍如下：

- `Judge.exe` 这个是原始的趣味程序设计竞赛的 Judge 程序，可以用于生成 4 种类型的验证码图片，使用方法详见 `CapJudge Readme.txt`。
- `gen_sample.py` 这个程序用于生成单个字母的图片集合，训练集和测试集都是通过它来生成的，它首先会调用 `Judge.exe` 来生成原始的图片，然后使用 `preprocess.py` 中的方法对其进行噪音过滤，最后分割为 4 个部分，保存到指定的目录中，文件名类似于 `A-2-1234.png`，第一部分是该图片所包含的实际字母，第二部分是指该图片是属于 4 种验证码中的哪一种，最后是一个保证唯一性的数字 ID。打开该文件在头部附件可以找到一些可调整的选项，包括生成的数量，生成哪些类型的验证码以及结果存放在哪个目录。
- `preprocess.py` 该文件包含了对图片进行预处理的代码，主要是用中值滤波器进行降噪和字母的分割。

- knn.py 该文件实现了一个简单的 knn 分类器。
- feature.py 该文件包含了 SURF 特征的提取以及将提取出来的 SURF 特征点根据 vocabulary 转化为特征向量的代码。使用 build_vocabulary.py 可以来构建一个 SURF visual word 的 vocabulary，它会从 raw 目录读取分割好的字母图片并将结果保存在 vocabulary.mat 中。
- pca_feature.py 该文件包含了 PCA 特征相关的代码，可以用来将一张图片转化为一个经过 PCA 降维后的特征向量。另外，构建 PCA 的降维投影系数时也是需要先用它来准备 matlab 格式的数据，由于已经有现成的 matlab PCA 代码可以很方便的使用，所以我在 matlab 中进行了 PCA 系数的计算。
- train_model.py 该文件用于根据训练数据生成一个模型。
- validate_model.py 该文件用于根据测试数据来测试模型的性能。

如果要运行测试，首先使用 gen_sample.py 生成分割好的 sample 集合，训练集保存在 raw 目录下，测试集保存在 test 目录下。生成训练集的时候，我使用如下配置：

```
nrepeat = 4
case_model = [0]
dest_dir = 'raw'
```

其中 case_model 可以选 $1 \sim 4$ 表示 4 种类型的验证码，0 表示包含了 4 种混合的。生成测试集的时候改一下目标目录，可以改一下 nrepeat 让它多生成一些数据作为测试用。

接下来是训练模型，如果要训练 SURF 模型，首先是构建 vocabulary，直接运行 build_vocabulary.py 即可，然后再构建模型：

```
python train_model.py surf raw model.mat vocabulary.mat
```

训练好的模型会保存在 model.mat 里。如果是要构建 PCA 模型的话，首先需要将训练集中的图片进行缩放：

```
python pca_feature.py scale raw raw_scale
```

缩放过的图片存放在 raw_scale 目录中，然后将其转化为 matlab 的矩阵格式：

```
python pca_feature.py prepare raw_scale pca_raw.mat
```

然后在 matlab 里加载 pca_raw.mat，并使用 PCA 求解降维系数，并将结果保存在 pca_coef.mat 中（使用变量 eigvec）即可。之后可以同样调用 train_model.py 来训练 PCA 模型：

```
python train_model.py pca raw model_pca.mat pca_coef.mat
```

之后可以分别使用以下两条命令来对训练好的两个模型进行测试：

```
python validate_model.py test surf model.mat vocabulary.mat
python validate_model.py test pca model_pca.mat pca_coef.mat
```