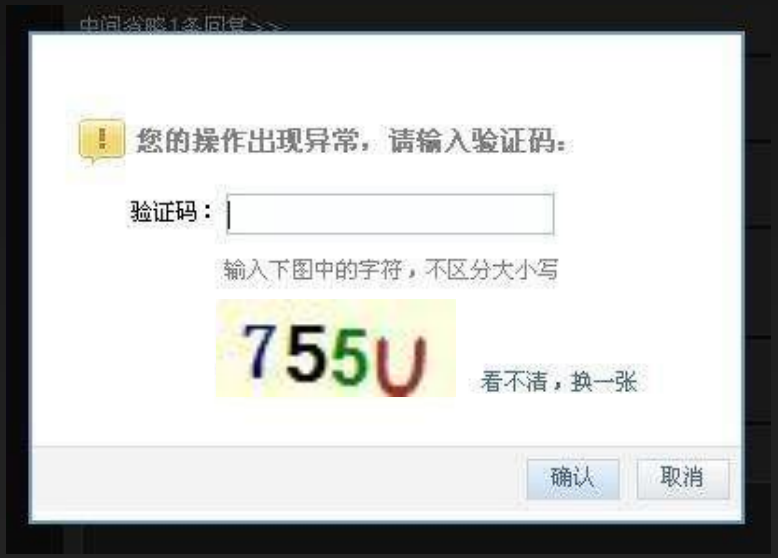
验证码的识别

何晓艺 150410430
尧帆　 150410428
袁明嵩 150410413

目录

我们为什么需要验证码识别器？

1.网络爬虫中，应对反爬虫机制



爬虫操作过于频繁或使用代理服务器容易触发验证码提示，需要输入验证码继续操作

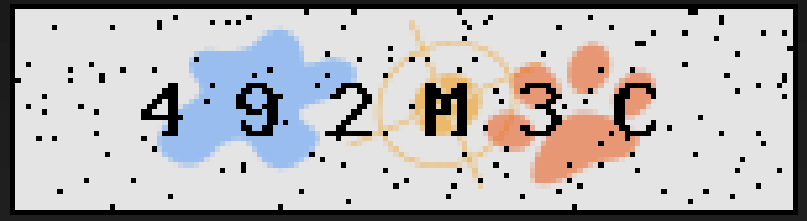
我们为什么需要验证码识别器？

2.实现一些批量自动化操作

例如，数据库里有几百几千个人名，要逐个查询是否失信。

验证码都有什么种类？

1.最常见的：文字验证码



把文字进行变形、变色、拉伸、加噪点、加背景图案等操作，增加机器识别的难度

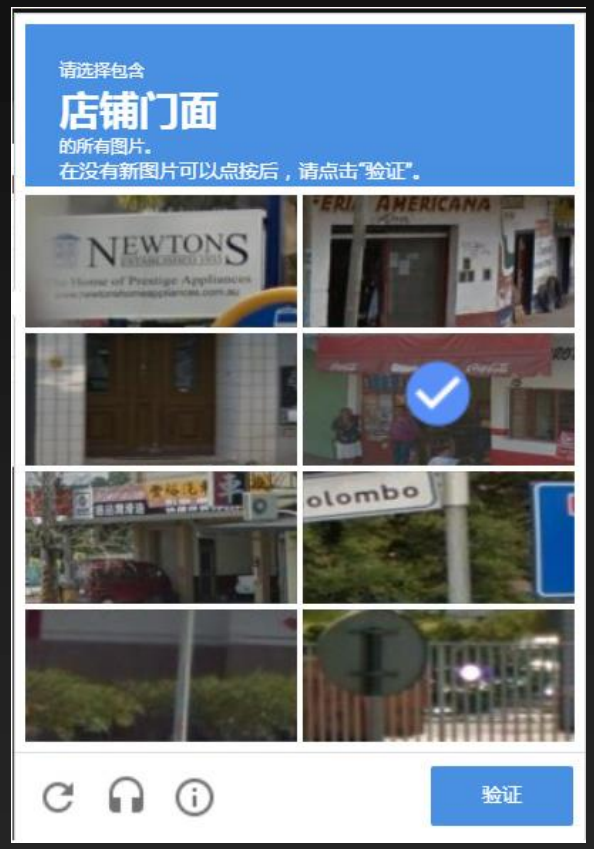
验证码都有什么种类？

2.滑动验证码

要求用户把图块滑动到某一指定位置，例如把拼图拼接完整。

验证码都有什么种类？

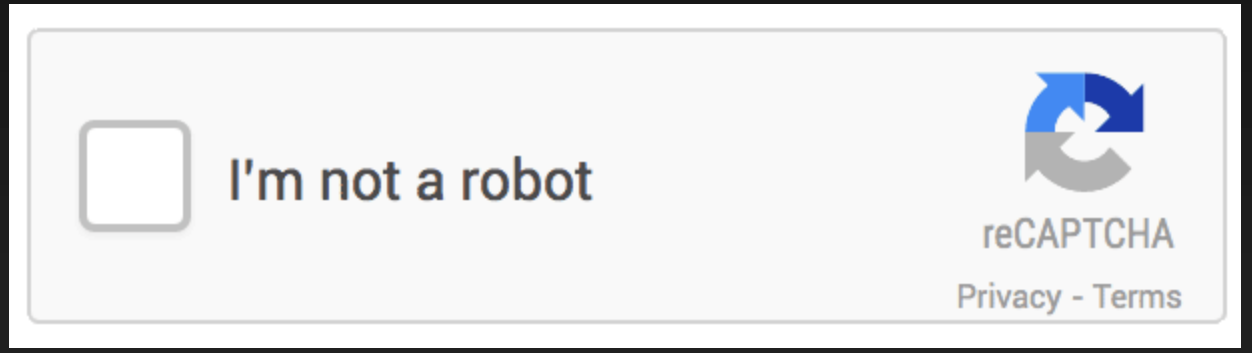
3.答题验证码



要求用户回答问题以证明自己不是机器人。

验证码都有什么种类？

4.reCAPTCHA



记录用户在网站上的鼠标移动和点击等动作，通过识别动作是否可疑来判定是否为机器人。

这几类验证码的破解难度

1.图文验证码

可以通过对图像进行二值化、去噪等处理后，用OCR方法处理。

2.滑动验证码

可以通过OCR方法获得图块边界信息后获知滑动距离。

3.答题验证码

使用机器学习手段识别问题和图片中的物体来自动解答问题。但是目前这项技术的破解效果不理想。

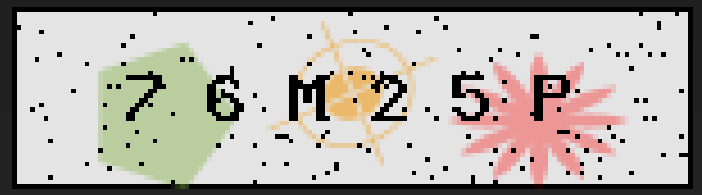
4.reCAPTCHA

目前为止尚无行之有效的破解手段。

本次研究的问题：简单图文验证码

图文验证码在现有的网站中最为常见（因为生成更容易），且破解难度相对最低。
本次研究的内容是简单图文验证码，其特点为：

1）有噪点
2）有背景干扰
3）无拉伸变形，或者拉伸变形种类有限，可遍历列出
4）字符均为大写英文字母



简单图文验证码破解步骤

破解图文验证码需要的步骤:

1.预处理

1）灰度化（彩色图片转黑白）
2）二值化（把黑白图片进一步转换成只有纯黑、纯白两种灰度)
3）去噪（去除图片中的噪点、干扰线等噪声）
4）字符分割（把图裁剪成一个一个字符分开的样子）

2.文本识别

使用OCR方法识别图片中的文本，获得验证码

预处理

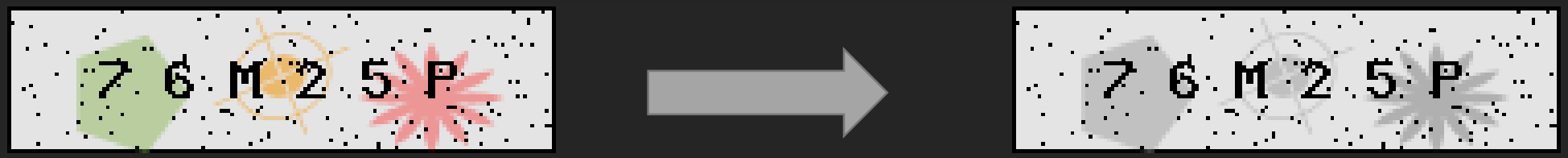
第一步：灰度化

把彩色的图片转换成黑白图片。因为颜色信息在验证码识别的过程中没有用。

对于每个像素点均执行如下转换操作即可：

设r,g,b为这个像素的红，绿，蓝三个分量，然后得亮度值

$$a = \frac{r+g+b}{3}$$



预处理

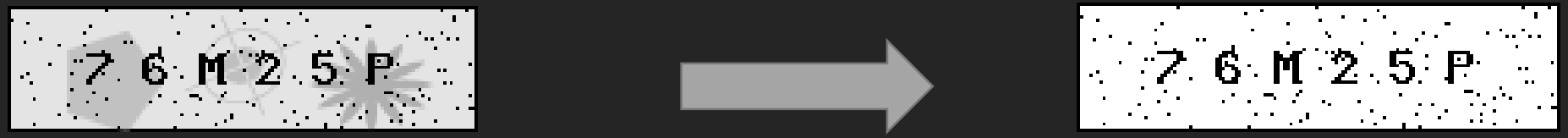
第二步：二值化

把黑白图片进一步转换成只有纯黑和纯白的图片。

对于每个像素点均执行如下转换操作即可：

```
if 颜色深度>阈值
        该点设为黑色
else
        该点设为白色
```

（阈值的具体数值根据具体情况酌情设定）



预处理

第三步：降噪

把图片中的噪点去除。
这里采用“8邻域算法”进行处理。

设图片尺寸为pxq，坐标从(0,0)到(p-1,q-1)，降噪率N为整数，N∈[1,8]，那么，对于x∈[1,p-2]，y∈[1,q-2]的每一个点(p,q)，进行这样的操作：

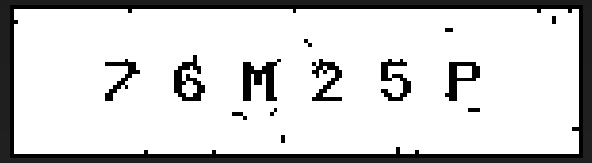
```
if A的颜色与该点周围相同的点数>=N
        该点视为不是噪点，维持原状
else
        认为这一点是噪点，设成白色
```

预处理

第四步：找出文本所在区域

我们需要知道文本在图片中处在什么位置。

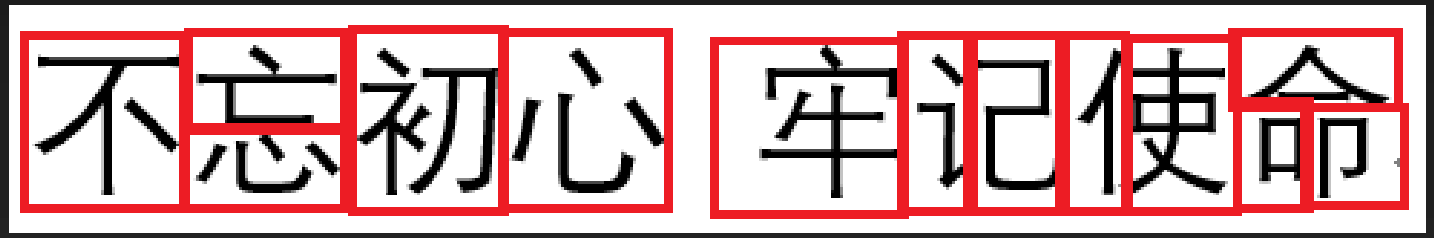
对二值化和降噪后的特征图进行连通区域搜索，每个连通区域对应的就是一个字母。这样，我们就可以对每个字母逐个考察了。



预处理

第四步：找出文本所在区域

注意：此法只适用于英文等字符，要求每个字都是连通区域。如果对中文使用这种方法将会出现如下所示的错误：一个字被分成了多个区域。



对于中文字符的情况，要使用其他的算法。

文本识别

我们使用了Google推出的Tesseract OCR文本识别库。这个库的原理在《Adapting the Tesseract Open Source OCR Engine for Multilingual OCR》这篇论文中有介绍。

文本识别

第一步：分辨形状、提取特征向量

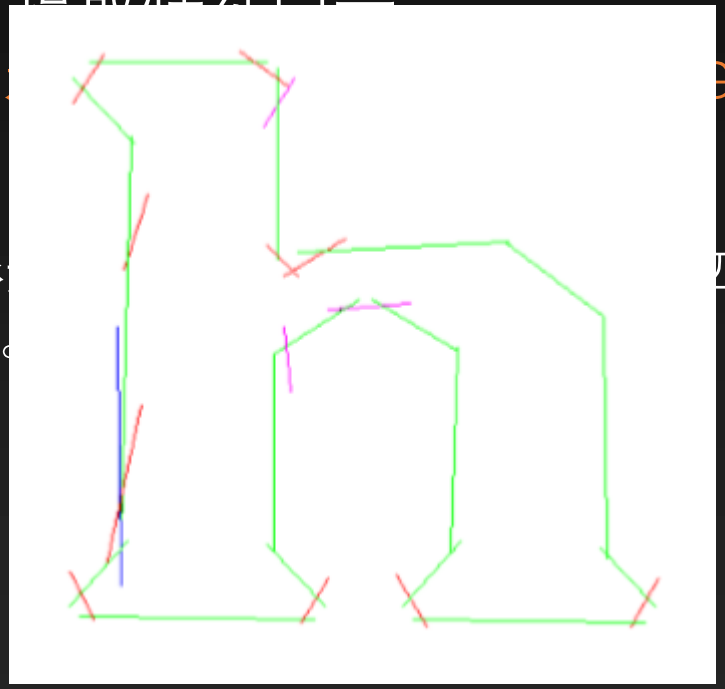
注：这个问题学术界仍在进一步研究中。目前Google Tesseract库中的形状分类器工作情况尚可。

我们从经过多边形近似处理过的字符轮廓中导出四维特征向量（x,y,d,l）x,y:位置坐标 d:方向 l:长度。

文本识别

第一步：分辨形状、提取特征向量

注：这个问题学[illegible]Google Tesseract库中的形状分类器工作情况尚可。



我们从经过多边形[illegible]四维特征向量（x,y,d,l）x, y:位置坐标 d:方向 l:长度。

文本识别

第二步：字符集映射

我们需要把要识别的字符映射到某一个范围内（例如，英文字母和数字这个集合{A,B,C,…Y,Z,a,b,c,…y,z,0,1,…9}），这样可以尽可能缩减可能产生的字符数量，也避免了出现异常字符。

如果我们事先已经知道这个范围，那就只需人工输入进行一一映射即可（如果这个验证码没有拉伸变形，则字形有限）。但是，有些情况下，我们不知道，或者识别范围是无穷的（例如带有扭曲、拉伸变形等变换的验证码，变形方法无限多）。那么，我们就需要用一种称为“局部敏感哈希”(Locality-Sensitive Hashing, LSH)的算法来完成这个动作。

文本识别

第二步：字符集映射

LSH的基本思想是：将原始数据空间中的两个相邻数据点通过相同的映射或投影变换（projection）后，这两个数据点在新的数据空间中仍然相邻的概率很大，而不相邻的数据点被映射之后仍然相邻的概率很小。也就是说，如果我们对原始数据进行一些hash映射后，我们希望原先相邻的两个数据能够被hash到相同的桶内，具有相同的桶号。对原始数据集合中所有的数据都进行hash映射后，我们就得到了一个hash table，这些原始数据集被分散到了hash table的桶内，每个桶里会落入一些原始数据。同一个桶里的数据有很大概率相邻。这时，问题的重点就在于，我们要找一个满足这个需求的hash算法。

文本识别

第二步：字符集映射

那具有怎样特点的hash functions才能够使得原本相邻的两个数据点经过hash变换后会落入相同的桶内？这些hash function需要满足以下两个条件：

1）如果$d(x,y) \leqslant d1$，则$h(x) = h(y)$的概率至少为p1；

2）如果$d(x,y) \geqslant d2$，则$h(x) = h(y)$的概率至多为p2；

其中d(x,y)表示x和y之间的距离，$d1 < d2$，h(x)和h(y)分别表示对x和y进行hash变换。

满足以上两个条件的hash functions称为(d1,d2,p1,p2)-sensitive。而通过一个或多个(d1,d2,p1,p2)-sensitive的hash function对原始数据集合进行hashing生成一个或多个hash table的过程称为Locality-sensitive Hashing。

文本识别

第二步：字符集映射

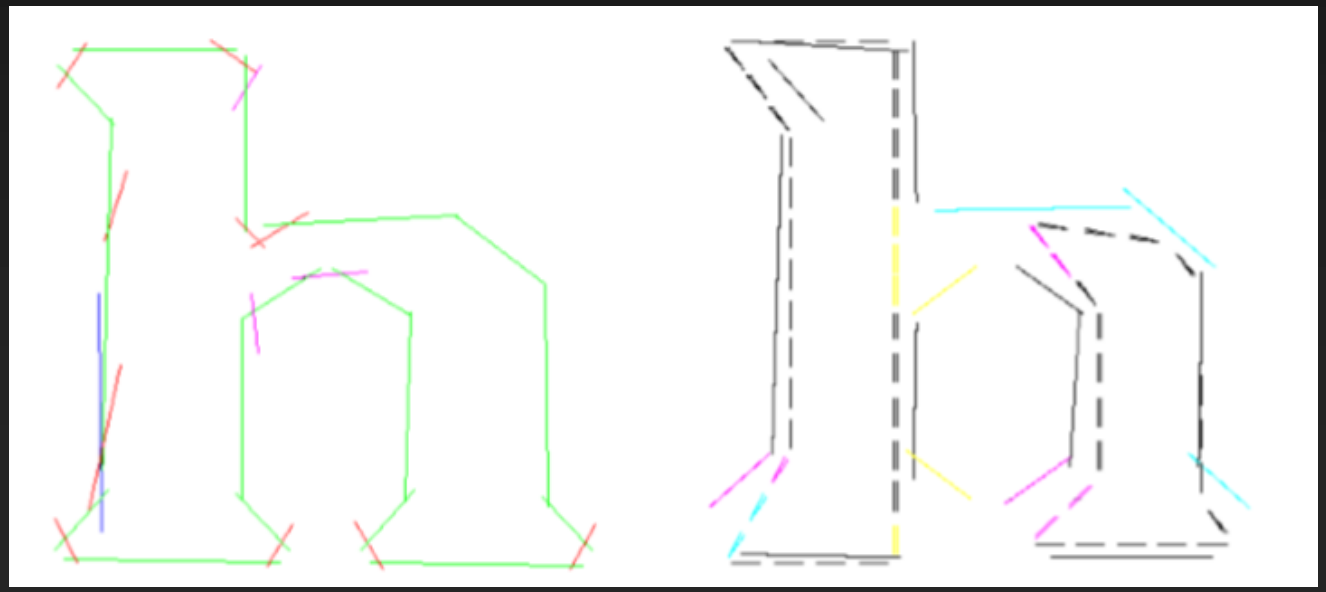
LSH算法的操作如下：

1.选取若干个符合上文所述条件的hash函数。
2.对每一个数据进行哈希计算，映射到对应的桶内，构成若干个hash table。
3.把查询数据对应的桶号x用hash函数算出。
4.把第x号桶中对应的数据取出。
5.计算想要查询的数据与桶中数据的相似度，返回最相近的数据。

文本识别

第二步：字符集映射

在验证码识别过程中，我们使用LSH算法辨别每个特征向量与字符集中字符的匹配程度，找出相对来说最匹配的那个特征向量。



如右边的图中，黑色—很好吻合 紫红色—吻合度尚可 青色—勉强吻合 黄色—几乎不匹配

文本识别

第三步：计算距离

接下来，我们计算每个字符与学习样本中字符样本之间的距离。使用如下公式：

$$d_f = d^2 + w\theta^2$$

其中d是平面上的坐标距离，w是权重，θ 是角度差

文本识别

第四步：计算置信度

接下来，我们计算每个字符与字符集中字符样本相对应的置信度：

$$E_f = \frac{1}{1 + \mathrm{k}d_f^2}$$

其中k是一个可调变量用于控制识别区间（k越大，同样距离下置信度越低）。
为了速度更快，

文本识别

第四步：计算置信度

接下来，我们计算每个字符与字符集中字符样本相对应的置信度：

$$E_f = \frac{1}{1 + \mathrm{k}d_f^2}$$

其中k是一个可调变量用于控制识别区间（k越大，同样距离下置信度越低）。
为了速度更快，

文本识别

第五步：计算最终距离

接下来，我们计算字符的最终距离：

在算出E_f的同时，我们把它复制一份称为E_{p}。因为样本中的每个字符通常都会被匹配多次，且OCR程序通常是多线程运行的，这就导致不同线程算出的 E_f 不相等。

于是，我们可以计算出一个最终的距离：

$$d_{final} = 1 - \frac{\sum_f E_f + \sum_p E_p}{N_f + \sum_p L_p}$$

其中L_p是各个样本特征向量中L维度的和，N_{f}是样本总数。

程序测试

运行环境：Python 3.7
*因Pytesser已很久没有更新，使用的是C++版Tesseract，自行实现Python接口
图像预处理：
灰度化：Python PIL Image库
二值化：自行实现
降噪：自行实现
文本区域查找：Tesseract内置
分辨形状：Tesseract内置
字符集映射：Tesseract内置
距离计算：Tesseract内置
置信度计算：Tesseract内置

源代码见https://github.com/lenovotcldellhp/ocrtest

程序测试

样本来源：百川PT登录页面
类型：6个大写英文字母，有噪点，有背景干扰，无拉伸变形

识别正确字符数	0	1	2	3	4	5	6
图片数量							

字符识别正确率：
整个验证码的识别准确率：

*用于生产环境时可以使用多线程的方法加快识别

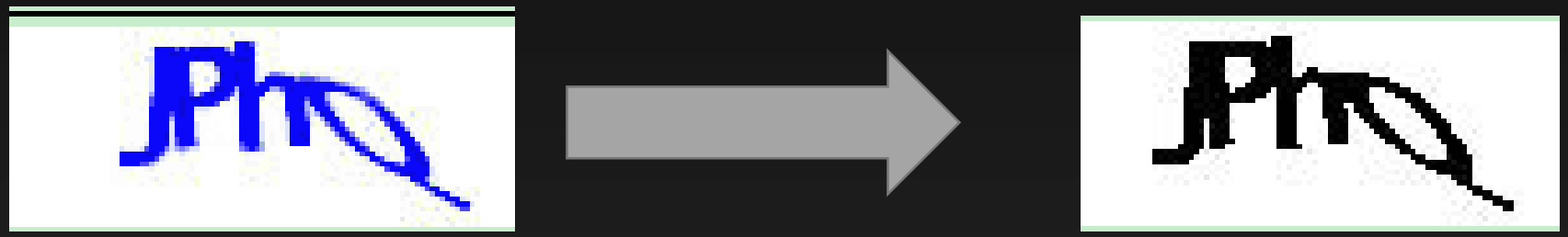
可以尝试的进一步升级

1.字符粘连的验证码识别

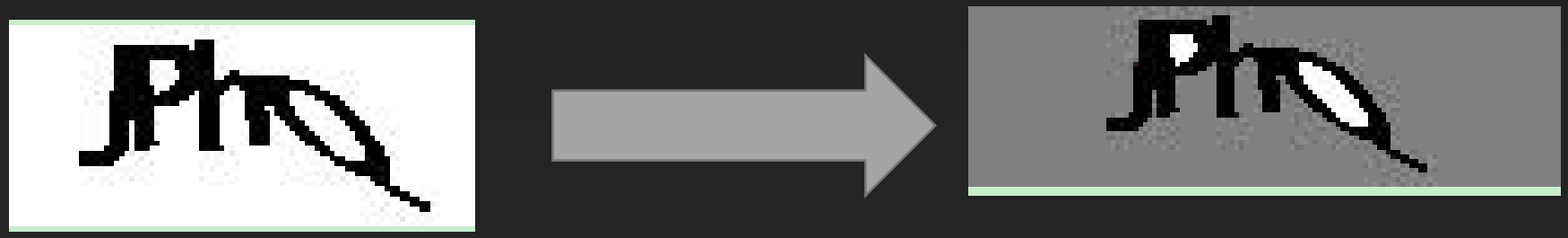
这类验证码中，字符是互相粘连的，无法使用上文所述的连通区域法来分割字符。这种验证码的识别方法在《The Robustness of “Connecting Characters Together” CAPTCHAs》中有描述。但难度较大，我们未能实现其代码，也未能找到相应的开源库，下面简要说明一下其原理。

可以尝试的进一步升级

1.字符粘连的验证码识别



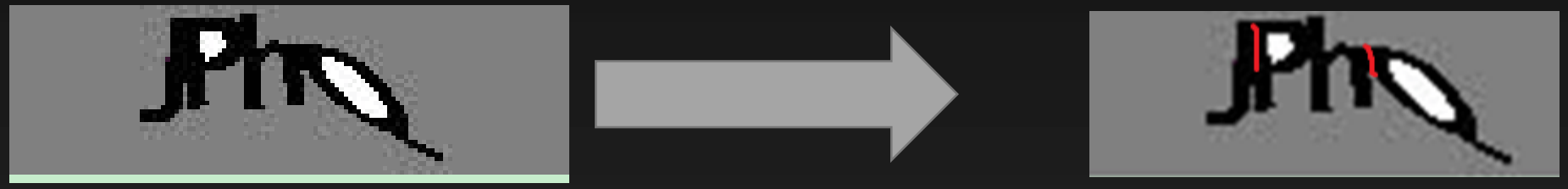
首先，和前面一样，我们仍然进行灰度化和二值化，去除多余的信息。
接下来，进行背景填充，给它填上一个背景色。



可以看到，有未被填充颜色的封闭区域。这个时候，我们就可以把这个封闭区域与旁边的字符分割开。

可以尝试的进一步升级

1.字符粘连的验证码识别



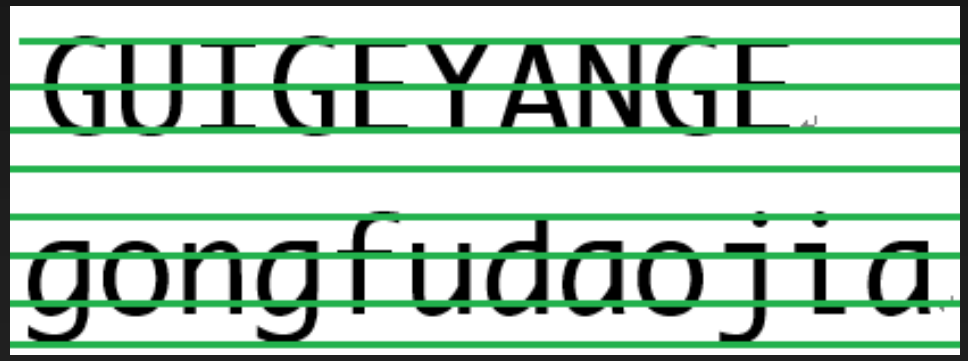
对于本例来说，已经顺利分开四个字母了。但是，有些情况下，这种填色法并不能完全分开每个字符。例如下面这个例子：

这时要使用另一种方法：拼音本特征(Guideline Principle)。

可以尝试的进一步升级

1.字符粘连的验证码识别

拼音本特征法来源于我们熟悉的四线三格拼音本：



我们模仿拼音本的样子，给不规则字符也标上水平标线：

可以尝试的进一步升级

1.字符粘连的验证码识别

水平标线标注算法如下：

1）首先，从字符像素的最左侧到最右侧，绘制固定像素长度的水平线，这些水平线的垂直位置由每一列字符像素的最低点决定。

2）为每一段水平线赋值。该值是该水平线的坐标点与其四邻域的坐标点的差值。

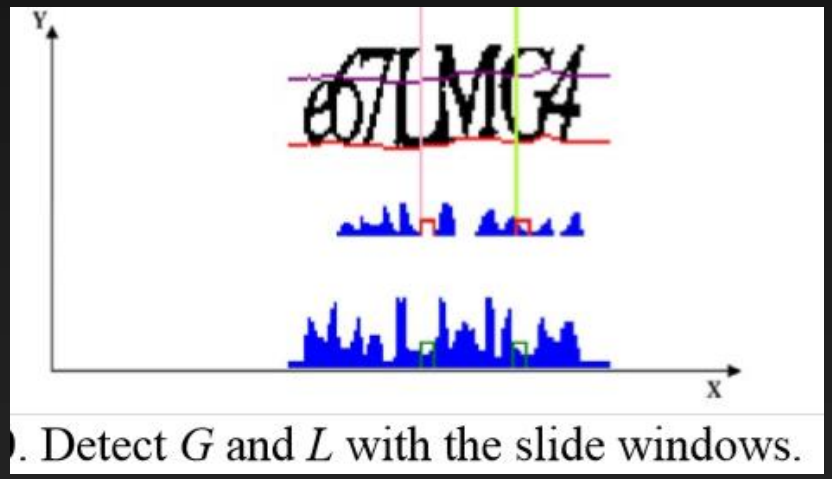
3）将特征值低于某经验阈值的水平线链接起来，其余高于阈值的水平线将被忽略。

4）最后，最左侧与最右侧的点均水平连接起来，该线作为我们的base line。在base line的每一列像素中找到垂直距离最大的字符像素点，将base line与该最大点距离的4/5作为mean line的绘制点，这样我们即可得到mean line。

可以尝试的进一步升级

1.字符粘连的验证码识别

接下来进行竖直标线的标注。



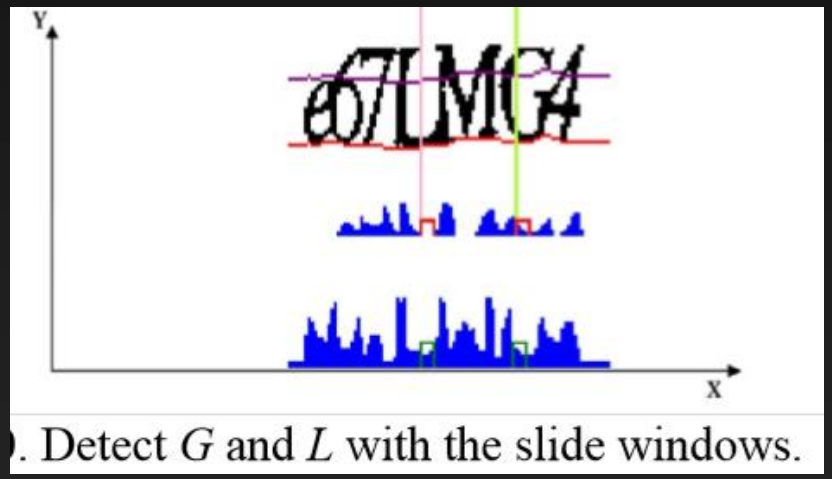
). Detect *G* and *L* with the slide windows.

把顶层和中层分别投影到mean line和base line上，如上图所示。上下两个蓝色区域分布是mean line和base line的投影。然后，用一个5像素高的红色矩形和一个8像素高的绿色矩形分别在两个投影区域同时从左到右滑动，直到滑到一个点，这个点上红色矩形中无投影像素且绿色矩形中的投影低于8像素高。我们可以认为这个点处可以画竖直分割线，如上图中的标线。

可以尝试的进一步升级

1.字符粘连的验证码识别

接下来进行竖直标线的标注。



. Detect G and L with the slide windows.

水平和竖直标线均标注完毕后，就可以进行下一步的识别了。

可以尝试的进一步升级

2.有扭曲变形的验证码识别

这类验证码的识别比上文所说的简单图文验证码困难，因为其中的字符是扭曲变形的。这种验证码需要使用TensorFlow进行卷积识别。因为我们没有足够的GPU计算力，暂时无法进行尝试。

参考资料

1.Adapting the Tesseract Open Source OCR Engine for Multilingual OCR，Ray Smith, Daria Antonova, Dar-Shyang Lee Google Inc.,

2. Locality-Sensitive Hashing for Finding Nearest Neighbors [Lecture Notes] ,Malcolm Slaney, Michael Casey.

3.局部敏感哈希算法的研究，史世泽，西安电子科技大学

4. Practical and Optimal LSH for Angular Distance，Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, Ludwig Schmidt, MIT

分工情况：

何晓艺150410430（组长）

论文查找与分析、目标网站选择、算法分析、源代码编写

尧帆150410428

论文分析、测试样本获取、测试程序

袁明嵩150410413

测试样本获取、测试程序