图片验证码识别算法的设计与实现

张扬

摘要：

图片验证码作为一种防止机器人使用重复手动暴力尝试获取网络资源的一种手段，其生成方式简单且种类多样，主要用于区别人和机器。但各种验证码生产程序生成的图片验证码保护强度不一。对验证码机器识别的研究可以为日后研究更强的验证码生成规则提供基础。本文主要针对一种样式的图片验证码，给出一系列图像处理的方法，包括二值化，修复，去噪。并针对性地设计一种简单高效且有较高正确率的字符切割方法最后使用支持向量机进行机器训练和预测。

1. 引言

图片验证码技术是现代网络安全技术的一部分，用于区分访问对象是人还是机器人，以防止使程序进行大流量的有目的的访问。一般的验证码可以阻止一些简单的机械试的访问，在如今机器识别已经成熟的时代，对于简单排列的数字和字母的图片识别已经不是什么难题了。国内外已经有很多学者对验证码的识别进行研究。尝试机器识别验证码的目的不是破解而是发明更有效的，难以机器识别的验证码来保持发明验证码的初衷。验证码的识别包括图片预处理阶段，包括图片灰度化，图片二值化，图片去噪，图片增强等；字符分割阶段，也是识别验证码最难最关键的阶段，字符分割的好与坏会直接影响到下一步分类学习识别阶段；机器学习与预测阶段，这一阶段常用的分类及预测的有KNN和SVM算法。现阶段的研究表明，字符分割是验证码处理的重难点，目前还没有通用的完美的算法去解决复杂粘连字符的分割问题，有些甚至人眼也需要一定时间去区分，这是阻碍机器识别验证码的主要障碍，也成为如今验证码依然可以成为人机区分手段的基础保障。国外学者指出，当一组样本的验证码能够被机器有效识别（完全识别正确）超过10%则该验证码（系统）已经不再安全。

本课题旨在以验证码识别为媒介学习图片处理图片分割以及机器学习的一些算法，尝试设计一套简单的图片验证码识别算法。

第一节对图片样本的特性作出一些分析，第二节主要对图像预处理各步骤算法作介绍和一些性能分析，第三节主要介绍和分析字符分割算法，第四节对实际预测效果和性能作出一些测试和评价。

1. 原始样本的特性分析

针对一种特定规则生成的图片验证码设计图像处理机器识别算法是改进其生成算法以防止后续扔被机器暴力破解的前提，因此针对特征图片进行图像处理时可以避免大量嵌套一系列通用图像处理的算法而针对性地设计特征处理算法，因此需要在算法设计前期针对原始的图片样本做一些图像的特征分析。本案例选取500张样式样本作为机器学习的学习样本。

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| 图1. 一些原始验证码图片样本 | |

通过肉眼观察上述验证码图片的样本，我们可以发现一个样本图片主要由背景，干扰，字符三部分组成，而且这类生成规则下的验证码字符元素的粘连不大甚至几乎没有，这对于图片后期的字符切割是极为有利的，同样也就预示着这类验证码极其容易受到机器的暴力破解。

* 1. 图像结构及描述的定义

|  |
| --- |
|  |
| 图2. 图像定义 |

本文中处理的图像坐标原点设在图像左上角，图像宽度方向为X轴，高度方向为Y轴对于其中的给定一点像素为方便表述及后续处理，定义像素位置向量

|  |  |
| --- | --- |
|  | （1） |

将图像看作复平面，定义像素位置复数

|  |  |
| --- | --- |
|  | （2） |

。

对于输出为实数的（离散）函数 ，其二元函数表达形式 二维向量表达形式，复变表达形式 在本文中表达相同意义，并取其表述简单的形式对算法进行描述。

图像区域

|  |  |
| --- | --- |
|  | （3） |

其中width表示图像宽度，height表示图像高度。

* 1. 样本集中主要图像结构元素的分布

我们将样本集中所有的样本图像叠加后求平均灰度值，便可以观察到整个样本集中干扰字符等的分布状况。

对样本集做如下处理

|  |  |
| --- | --- |
|  | （4） |

处理后我们可以得到如下结果：

|  |
| --- |
|  |
| 图3. 叠加后的灰度平均值分布情况 |
| 图中，蓝色矩形框表示主干扰存在区域，红色矩形框表示字符的主要存在区域。  我们可以发现，有较多分布的较大干扰反而对后期字符处理影响不大，由于与字符间隔较大，它们可以很容易得被去除。因此我们主要需要处理的噪声为穿过字符存在区域较细的长干扰线。 |

* 1. 图像背景灰度值分布的前期分析

通过对验证码图片的样例进行观察可以得知图片背景灰度在图片高度方向几乎没有变化，在沿着图片宽度方向逐渐变亮。我们选取几行不经过字符主要存在区域的行作为背景采样行，用500张样本进行灰度分布的采样。

做如下操作

|  |  |
| --- | --- |
|  | （5） |

|  |
| --- |
|  |
| 图4. 主要背景区域采样行灰度值分布情况 |

对上述采样图像及原始样本图像进行分析可以得知，在宽度20和45位置的两个波谷是由黑色的主干扰线引起的平均灰度值下降，第49行的平均灰度出现另外四个明显波谷是由于有些字符如p,g，向下延伸较大，在第49行仍有黑色的字符像素分布引起的。四个波谷几乎恰好对应四个字符的分布位置。

略去黑色干扰线和个别字符引起的灰度值分布波动，背景的灰度值从图像左侧到右侧明显呈上升趋势，从采用图像上粗略估计背景灰度的最小值应该是不低于185 。

1. 图像预处理算法设计及性能分析
   1. 灰度图像二值化

本案例中样本灰度图像的二值化主要需要讨论两个方面，使用常规的二值化阈值确定方法还是根据样本特性设计一种更简单的二值化阈值确定算法；另一方面对样本进行二值化时，每个样本确定一个二值化阈值还是所有样本使用一个全局的二值化阈值，两者效果是否有较大的差距。

下面我们考察一个样本及其灰度直方图的特性：

|  |
| --- |
|  |
| 图5. 样本及其灰度直方图 |
| 结合前期对于样本图片背景的研究，其中很大一部分灰阶值所拥有的像素个数少于10个，这表明这些灰度值对应的点是字符与背景的过渡点，由于字符始终是灰度为0的黑色点，而背景由左至右逐渐变亮，图片的生成规则为了使变化不唐突，使得字符与背景的过渡点随着背景的变化而变化。  我们显然不能通过二值化操作将字符过度略去，更不能将背景划为有效识别像素，因此那一系列低像素值的灰阶便是我们选取二值化阈值的范围。  传统的OSTU法取图像二值化需要多次迭代，比较耗费计算时间，主要用于处理那些构成复杂的图片，这类图片验证码背景区分鲜明，构成并不复杂，因此这里考虑不适用OSTU法，初步试探尝试使用低像素灰阶中位数法。   1. 先对原图像作出灰度直方图 2. 在定义域的子集中观测一个合理的界定值 3. 在区间中找到所有的 4. 求出的中位数 5. 记录二值化阈值   找到二值化阈值后对图像进行二值化处理  则有   |  |  | | --- | --- | |  | （6） |   我们将将作为学习样本的500张图片，分别用不同的二值化阈值计算方法计算其二值化阈值，并统计得到下图：   |  | | --- | |  | | 图6. 不同二值化阈值计算方法计算出的阈值结果 |   通过对比看出，OSTU计算出的二值化阈值较高，各样本的阈值相当接近，MLPG法计算出的二值化阈值较低，个样本比较接近，最大与最小的二值化阈值相差在20左右。但是MLPG法的计算算法复杂度远低于OSTU，不必要做重复的迭代，只需要扫描一遍像素点即可。  通过上述统计还可以发现，不同样本的二值化阈值相当接近，表明有条件进一半简化算法，对于所有的样本使用同一的二值化阈值，使得二值化前不必要针对每个样本进行计算阈值。实际上这是可行而且是较佳的选择。  下面我们给出两个样本使用不同的阈值进行二值化的效果。   |  | | --- | |  | |  | | 图7. 不同二值化阈值下的二值化效果 |   不难看出，二值化阈值越高，字符约粗实，字符边缘也越发平滑，但较高的二值化阈值也强化的干扰线部分，加大了后期的处理难度，而且对于某些样本如【mw3n】，较高的二值化阈值使得字符见出现了粘连，这对于后期字符分割是比较难以处理的，这也是较高二值化阈值的一个致命缺陷。低二值化阈值使得字符边缘凹凸不平，且字符骨干部分偏细，但是却很好地分开了各个字符，不对对后期字符分割造成困难。  由于二值化阈值选取的不同，会导致字符粗细不同，这对于后期机器学习样本字符的平均粗细不一是很不利的，使用一个全局的二值化阈值可以使得不同样本间的字符粗细均衡，对于机器学习需要的归一化条件是很有利的，这也是最后摈弃了每个样本分别计算二值化阈值的主要原因。  本案例中的二值化阈值尝试选取T=80,T=100,T=120 ，经过后期的去噪分析，字符分割分析，最终选取全局二值化阈值T=80。该阈值较好地保留了字符的骨干部分，会导致字符边缘出现一些孔洞，这使用后期的字符空洞修复算法可以完成；该阈值没有过度强调干扰线，使得干扰部分可以较为容易地去除；该阈值不易造成字符粘连，（原则上该样本集的字符间均有空隙，但可能较小，过高的二值化阈值将这些字符背景过渡点划为字符点使得出现了字符粘连）比较有利于后期的字符分割步骤。   * 1. 字符边缘孔洞修复   由于二值化算法选取的阈值会导致字符边缘出现孔洞，字符边缘不平滑的结果。这样的字符边缘在后期去噪可能会被误认为干扰线；其次，同一字符的不同样本在不同的位置出现孔洞与不平滑现象对于机器训练是不利的，会导致机器识别误判；再次，后期的字符分割默认认为字符是连续的，字符内部是不应该出现空隙的，字符边缘零散越出的白点或者黑点也可能会对字符分割造成干扰。因此字符边缘的修复这一步是必要的。  这里先介绍取一个像素周围四连通域或者八连通域的算法：  取四连通域的四个像素即取该像素位置的上下左右四个像素，在该案例中使用像素的位置复数表达较为简单。  由复数的有关知识可知在上述坐标系中 所表示的位置分别为复数 所表示的像素位置的 右下左上 四个像素。  对于取八连通区域的八个像素，即像素的上下左右，左上左下右上右下八个像素。  为方便处理定义旋转用模长为1的复数   |  |  | | --- | --- | |  | （7） |   同时为保证旋转45°时依旧是格点复数，即x,y是整数，需要在旋转45°时做模长伸缩变换。  像素八连通域的八个像素位置可以用如下方式取到   |  |  | | --- | --- | |  | （8） |   为方便后续使用，定义   |  |  | | --- | --- | |  | （9） |   在上述坐标系下 其中k=0,1,2,3,4,5,6,7时分别表示原像素的右，右下，下，左下，左，左上，上，右上 八个位置的像素位置复数。  这里我们额外定义一条规定：对于图像区域外即简单地定义此时的即白色点。  取四连通域像素进行判断时，我们认为一个白点周围有三个即以上的黑点，那么该白点应当被修正为黑点；   |  |  | | --- | --- | |  | （10） |   取八连通域像素进行判断时，我们认为一个白点周围有五个及以上的黑点，那么该白点应当被修正为黑点。   |  |  | | --- | --- | |  | （11） |   所有的修正后的信息直接输出在一个新建的图像上，对原始的输入图像不直接做修改。   |  |  | | --- | --- | |  |  | |  |  | |  |  | | 二值化后图像 | 边缘散点修复后图像 | | 图8. 字符边缘孔洞修复效果对比 | |   通过字符边缘孔洞修复后的图片明显得字符边缘变得平滑，且白点黑点零散分布的现象明显减少，有利于确定字符的主体位置，且有利于后续的去噪手段，不至于把应属于字符的黑色像素点过分地去除。   |  |  |  | | --- | --- | --- | |  |  |  | | 3pfg | 525d | cdg8 | | 图9. 修复掉的白色噪点 | | |   由效果图还可以得知，虽然这一算法步骤是去掉白点增加黑点，但主要是增加了字符边缘的黑色点，对干扰线并没有明显地加粗，因为干扰线周围的白点是不符合被黑色点大半包围这一判定条件的。  通过实际的测试，该算法的实现及相关阈值的设定，达到了不错的预期效果，可以为后续步骤的实施提供前提保障。   * 1. 图像去噪算法      1. 噪声来源   通过前文的描述可以得知，经过二值化后，背景几乎不会产生噪声，图像噪声来源主要在于厚度较大的左侧两条干扰线，其中厚度较大的部分在图像的主要字符存在区域外部，这一部分干扰后续可以在字符切割的时候直接切除实现，现处理阶段实际并不会造成过多的影响，这一阶段主要处理的是一段贯穿主字符存在区间的细干扰线，它会对后期机器训练识别字符造成一定的影响，比如字符误认；也可能对字符切割时造成困难，比如切割时机器认为这是字符粘连而错选了切割位置。  因此，图像去噪这一阶段的算法设计，主要是针对如何消除贯穿主字符区间的干扰线而特别设计的，避免了叠加使用通用图像处理方法。这里设计的集中针对性去线算法可以在每一个案例中单独使用，也可以进行设定不同的参数以组合使用达到更好的去噪效果。   * + 1. 深度（垂直）方向探索去噪算法   我们把在图像的一列中，沿着高度方向，图像的y坐标增大的方向叫做图像的深处方向。深度方向探索即检查当前像素位置深处方向的像素点。  根据我们要着重去除黑色干扰线部分的特点：连续较长，贯穿字符存在区间；大部分图像的干扰线较细（少数特例）厚度小于三个像素。同时，我们探查了字符厚度，大致在5个像素厚度以上。 |
| 对于大部分图片样本而言，生成的黑色干扰线厚度大致相同，因此我们选择几例干扰线和字符明显分离的样本在做厚度采样测试。  样本中的蓝色矩形框为我们对噪声区域像素厚度的采样，红色矩形区域为我们对字符区域像素厚度的采样。  记录的信息为我们每找到一个黑色像素点，这个黑色像素点深度方向的厚度。   |  |  | | --- | --- | | 其中 |  | |  |  | |  |  | |

|  |
| --- |
|  |
| 10-1 样本图片 |
|  |
| 10-2 干扰线像素点深度方向厚度 |
|  |
| 10-3 字符像素点深度方向厚度 |
| 图10. 样本深度方向厚度的一个采样 |
|  |

通过一个样例的采样分析我们可以看出，属于干扰线部分的黑色像素，深度方向的厚度绝大部分在3以下，少数达到4，而属于有效字符的黑色像素，只有少量在深度方向的厚度不达3，这些黑色点在字符下端片区，即“字符笔迹”在垂直方向的末端点。

由此我们给出一个尝试消去黑色干扰线的深度方向去噪算法：

D为给定的深度方向探索阈值，

若

|  |  |
| --- | --- |
|  | （11） |

则计算

|  |  |
| --- | --- |
|  | （12） |

则去噪修正后的灰度为

|  |  |
| --- | --- |
|  | （13） |

所有的修正后的信息直接输出在一个新建的图像上，对原始的输入图像不直接做修改。

我们在这里先讨论单一方式去噪的效果，后续再进行组合方式去噪的讨论。

下面我们对单一进行深度方向去噪的效果给出一些性能测试：

|  |  |  |  |
| --- | --- | --- | --- |
| 样本 |  |  |  |
| 阈值D=1 |  |  |  |
| 阈值D=2 |  |  |  |
| 阈值D=3 |  |  |  |
| 阈值D=4 |  |  |  |
| 图11. 单一深度方向去噪算法效果 | | | |

我们很显然地发现，使用这种深度方向去噪算法虽然可以有效消除掉薄的干扰线，但是也会削减掉字符的厚度，因此阈值的选择尤为重要，这里阈值的选择需要考虑能把大部分有影响的干扰线去除，尽量不把字符像素削减的过于薄以至于在字符内部出现空隙对后期字符分割造成不利的影响。

我们参照上述效果图可以看出，如果单一对这种验证码图片使用这种去噪算法，阈值D=2或者阈值D=3是可以考虑使用的阈值，这也与之前像素采样结果相一致。

该算法的有点为针对性强，专门针对薄干扰线设计，且算法执行简单，只需要遍历图片所有像素一次即可，没有迭代操作；该算法缺点也很明显，很明显地削薄了表示字符的像素，过度的去噪可能会产生字符内部不该有的空隙；如果要很好地保留字符像素就会去噪不当，干扰线去除不充分。该算法与其他去噪算法组合使用的效果将在后面测试。

* + 1. 八方向检索去噪算法

检索一个像素八连通域方向上的像素是否总体符合一定的要求来判断这是不是一个待去除的噪声干扰点。在这里，我们先对一个像素八方向及层数的定义做一些说明。

|  |
| --- |
|  |
| 图12. 一个像素的八方向及其层数 |

取一个像素第一次的八方向像素即取一个像素八连通域的八个像素的方法在上文已经介绍过，这里再说明一下取不同层数的八方向像素的方法。

一个像素的八方向像素与中心像素的辐射角满足如下的位置关系：

|  |  |
| --- | --- |
|  | （14） |

对应各数字分别是中心像素的右，右上，上，左上，左，左下，下，右下八个位置。那么待选取的第s层第k个像素可以如下表示：

|  |  |
| --- | --- |
|  | （15） |

其中zz为待选取像素，z为中心像素。

我们依旧选取一些样本对这些样本的干扰线区域和字符区域统计其中黑色像素点（待处理像素点）的八方向像素点信息，主要是统计八方向白色像素点的个数。

|  |
| --- |
|  |
| 图12. 八方向数据采集样例（蓝色为干扰区，红色为字符区） |

|  |  |
| --- | --- |
|  |  |
| 层数=2 干扰区域统计结果 | 层数=2 字符区域统计结果 |
|  |  |
| 层数=3 干扰区域统计结果 | 层数=3 字符区域统计结果 |
|  |  |
| 层数=4 干扰区域统计结果 | 层数=4 字符区域统计结果 |
| 图13. 对样例进行2,3,4层八方向白色像素个数统计结果 | |

我们给出如下的八方向多层检索去噪算法：

实际所取的层数S，和设定的参考阈值T为经验参数。

|  |  |
| --- | --- |
|  | （16） |

当一个黑色像素点检索到的多层八方向像素中白色像素点的个数大于设定的阈值时，这个黑色像素点就会被当做噪声点给予去除。

所有的修正后的信息直接输出在一个新建的图像上，对原始的输入图像不直接做修改。

在对设置探索层数及对应的处理阈值前，我们对样本每层及各个阈值去除掉的干扰线点比例和字符点比例进行统计。

|  |  |  |
| --- | --- | --- |
| 表1. 探索层数=2时各阈值上述样本去噪效果 | | |
| 阈值 | 干扰区黑色像素点去除率（应去率） | 字符区黑色像素点去除率（误去率） |
| 5 | 80.24% | 38.79% |
| 6 | 68.14% | 30.85% |
| 7 | 54.28% | 16.96% |
| 8 | 41.89% | 11.41% |
| 9 | 29.50% | 4.55% |
| 10 | 24.78% | 2.73% |
| 11 | 17.70% | 0.74% |

|  |  |  |
| --- | --- | --- |
| 表2. 探索层数=3时各阈值上述样本去噪效果 | | |
| 阈值 | 干扰区黑色像素点去除率（应去率） | 字符区黑色像素点去除率（误去率） |
| 8 | 88.50% | 45.33% |
| 9 | 84.07% | 36.97% |
| 10 | 71.39% | 26.22% |
| 11 | 59.88% | 18.94% |
| 12 | 50.74% | 12.82% |
| 13 | 37.76% | 7.94% |
| 14 | 30.97% | 4.30% |
| 15 | 24.48% | 3.06% |

|  |  |  |
| --- | --- | --- |
| 表3. 探索层数=4时各阈值上述样本去噪效果 | | |
| 阈值 | 干扰区黑色像素点去除率（应去率） | 字符区黑色像素点去除率（误去率） |
| 11 | 89.09% | 59.06% |
| 12 | 86.73% | 48.06% |
| 13 | 76.40% | 38.21% |
| 14 | 74.34% | 29.36% |
| 15 | 68.44% | 22.75% |
| 16 | 57.82% | 16.63% |
| 17 | 49.56% | 11.86% |
| 18 | 40.71% | 7.69% |
| 19 | 31.86% | 5.46% |
| 20 | 26.25% | 3.39% |
| 21 | 23.89% | 2.07% |

以上效果数据为从少数样本将干扰与字符分离后的测试结果，具有一定的参考性。同时针对不同样本，实际的效果可能会有所出入。

这种针对性去噪算法显而易见地可以得出，如果要增加去噪率，必然会提高误去率，因此使用该去噪算法的关键在于控制好两者比例，使得影响字符识别和字符分割的干扰线能够被充分去除，同时保证字符框架被较好保留，因为字符被错误去除一小部分几乎是无可避免的。

下面给出单一使用这种去噪方式（组合去噪方式将在后续讨论）对于几个测试样本的实际去噪效果。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 样本 | T=4 | T=5 | T=6 | T=7 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 图14. 层数为2时八方向检索去噪法各阈值的实际效果 | | | | |

通过上述实例效果我们可以看出，进行2层八方向检索去噪时，阈值设定为5或6为理想的阈值能够达到较好效果，阈值设定大于7时，虽然字符保留比较完整但可能会使得部分干扰线去除不是很彻底，但如果某些样本使用低阈值时字符去除过多，设定阈值为7也是可以参考的。

我们选用同样的样本做3层八方向去噪实验：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T=8 | T=9 | T=10 | T=11 | T=12 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 图15. 层数为3时八方向检索去噪法各阈值的实际效果（省略原始样本） | | | | |

使用三层八方向去噪时，去噪效果随阈值设定的变化渐变，从上述样本的直观视觉看来，阈值设定9,10,11,12的实验效果均可以接受，不同的阈值虽然对字符骨干的保留均尚可，但对字符厚度还是有些许影响，如果机器训练的样本和识别用的样本去噪阈值设定不同，字符的厚度可能会影响到机器识别的效果。再者，当3层去噪与2层去噪有相当效果时，尽量选择二层去噪来适当减少计算量。

最后我们还是选择上述样本做4层八方向去噪实验：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T=14 | T=15 | T=16 | T=17 | T=18 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 图16. 层数为4时八方向检索去噪法各阈值的实际效果（省略原始样本） | | | | |

使用四层八方向去噪时，阈值设定更加细化，去噪效果越发渐变，但实际上较较少层去噪的效果提升并没有质的变化，因此实际使用时不推荐使用四层以上的检索。

综合上述去噪效果的测试，我们后续默认处理时，使用效果比较好的2层阈值为6的去噪参数以作为一个对比效果。

八方向检索去噪也是根据干扰线厚度较小的特征，针对像素厚度的一种去噪方式的变形，因此这种算法也是不可避免地会削薄有效的字符像素的厚度，但通过合适的参数设定，可以让这种“削弱”是可控且可接受的。

* + 1. 模板统计选择滤波器

模板滤波器是图像处理中的一种通用方法，主要通过设定不同的模板窗口，卷积核，权重达到处理图像的效果。常见的模板滤波器有梯度算子滤波器，拉普拉斯滤波器等。统计模板滤波器是模板滤波器的一个分支，它不对领域各像素设置权重而是统计它们的数值最后做出选择，常见的统计模板滤波器有中值滤波，最大值滤波，最小值滤波等，选取的窗口通常为奇数的平方。

这里，我们新设计一种模板统计滤波器，针对要去噪的特征重新设定窗口和最终选择作为输出像素的位置。

我们下面给出模板统计选择滤波器的一个通用算法的设计：

|  |  |
| --- | --- |
| 设计函数 (sort and index)排序并标序，函数的输入为有穷数列，输出从小到大经过排序的有序数列  设计函数选择器，函数的输入为有序数列和选择位置，输出为实数（整数）。  像素的模板窗口区域  设计函数窗口取数器，输入模板，输出为一个数集。  则设计一个统计排序滤波器，输入像素位置，模板，选择位置，输出为一个实数（整数）。  其中i表示索引xi表示元素，元素可以重复  列表 |  |
|  |  |

对于窗口的选择，需要能在取数完后满足去噪的需要。我们这里需要去噪的特征为穿越主字符区域的，厚度一般不大于3个像素的主要为横线发展的干扰线。

我们尝试使用宽度W=3厚度H=5的窗口。选择此窗口测试的依据为假设该窗口完全覆盖三个宽度的干扰线，那么窗口中读入9个干扰线点，6个其他点，相比其他的奇数X奇数的矩形窗口，干扰点占比较为适中。我们依旧选取一个样本对其在窗口下的取数做一个样本检测。

|  |
| --- |
|  |
| 图17. 模板统计选择滤波器数据采集样例（蓝色为干扰区，红色为字符区） |

我们对统计区域内的每个像素取窗口，将取到的数（0黑色或者1白色）按从小到大的顺序进行排序，构成函数输出：

|  |  |
| --- | --- |
| 其中y表示像素序号，x表示该序号的像素模板窗口取数从小到大排列的位置 |  |

我们对下面的测试结果图像进行一些说明：

左侧第一幅图像为对所选区域内的所有点进行模板取数后排序，纵坐标表示这是第多少个像素，横坐标表示取数从小到大排序后取的第几个数。色域中蓝色表示为0，红色表示为1。第二幅图像表示仅针对所选区域内的黑色像素点进行取数排序，表现出的是区域内黑色像素点的取数特征，也就是我们需要处理的点的特征。第三幅图像表示对整个区域用取数排序后的第k的值作为模板卷积核位置的新值能将黑色像素点去除的比例，横坐标表示选择的位置，纵坐标表示黑色点的去除率。

|  |
| --- |
|  |
| 图18. 干扰区域内的窗口检测数据 |

|  |
| --- |
|  |
| 图19. 字符区域内的窗口检测数据 |

我们选择这种方式滤波需要尽可能去除干扰点同时尽可能地不破坏字符的点，同样需要折中考虑取数位置。

|  |  |  |
| --- | --- | --- |
| 表4. 5X3窗口的不同选择位置的黑点去除率 | | |
| 取数位置 | 干扰区域黑点去除率 | 字符区域黑点去除率 |
| 1 | 0 | 0 |
| 2 | 0 | 0 |
| 3 | 0.42% | 0 |
| 4 | 0.84% | 0 |
| 5 | 10.55% | 0.25% |
| 6 | 29.96% | 0.59% |
| 7 | 40.51% | 2.45% |
| 8 | 44.73% | 3.89% |
| 9 | 57.81% | 9.22% |
| 10 | 74.68% | 18.87% |
| 11 | 83.97% | 29.86% |
| 12 | 99.58% | 37.39% |
| 13 | 99.58% | 47.63% |
| 14 | 99.58% | 56.56% |
| 15 | 99.58% | 66.07% |

通过观察上述样本的去除率，我们发现取数位置选择在9,10或者11时能够较多的将干扰特征的黑点去除而且不错地保留字符特征的黑点。下面我们选择一些样本进行实际的去噪测试。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 样本 | S=8 | S=9 | S=10 | S=11 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 图20. 模板统计选择滤波器在[5X3]窗口下各取数位置的去噪效果 | | | | |

通过实际去噪测试发现，当取数位置S>=8的时候穿越字符区域的干扰线几乎能被完全去除，取数位置S=8与S=9的区别在于字符间如果间距较小，S=9时能够去除字符与字符间，字符与邻近干扰线散点的一些连通成分，使得字符间的间距更为明显一些，但是代价则为字符被削薄一层，一些很薄位置的字符像素被错误地去除。我们另外发现当S>=9时，对于干扰线主体部分的去除已经没有什么差别，S越大去除的字符越多反而得不偿失。因此我们实际使用[5X3]窗口取数时，滤波器的选择位置选择为9。这个位置能够兼顾较好的去噪效果和字符骨干保留效果。

* + 1. 形态学去噪

<保留>

* + 1. 主干扰线检测

<保留>

* + 1. 组合去噪器

<保留>

1. 字符分割与单字符样本后期处理
   1. 二次校正的垂直字符分割
      1. 基本垂直字符分割的适用性与不足

垂直字符分割是一种简单高效的字符分割方法，但是它对输入图像有着较高的要求，输入图像的字符间间隙要求为垂直方向，输入字符无粘连，输入字符在垂直方向上没有交错。因为垂直字符分割适用性的特典，通常被用于车牌字符，银行卡卡号数字等形状规则且几乎没有其他干扰的字符图片分割。

回到本案例的验证码图像样本，大部分图像字符间有空隙，但是会有一些字符粘连，同时个别样本由于去噪的不完全，部分干扰线挡住字符垂直分割的间隙，另有一些去噪过度在字符内部出现了空隙，因此不能直接套用标准垂直字符分割算法，需要在适用性上进行一些改进。

|  |  |  |
| --- | --- | --- |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF38.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF68.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF39.jpg |
|  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF113.jpg |  |
| 字符粘连 | 去噪不足 | 去噪过度 |
| 图21. 无法直接使用垂直字符分割的一些样例 | | |

* + 1. 基准线垂直分割

普通的垂直字符分割通常先对图片的水平方向进行扫描，扫描每一列有效字符像素的个数，并选取合适的有效字符像素为0的列作为分割列将不同字符区分开来。

沿着图片的宽度方向逐列扫描每一列中黑色像素的个数

|  |  |
| --- | --- |
|  |  |

理想状态下的进行垂直字符分割的样本如银行卡号的数字串其样本和P(x)函数图像大致如下：

|  |
| --- |
|  |
| 图22. 可以进行标准垂直分割的理想样本及其P(x)函数图像 |

对于上述理想状态下的样本进行字符分割，切分位置可以由如下的方程组解出：

字符起始位置切分轴：

|  |  |
| --- | --- |
|  |  |

将计算出的解集中元素从小到大排序构成每个字符的起始切分位置集。

字符结束位置切分轴：

|  |  |
| --- | --- |
|  |  |

将计算出的解集中元素从小到大排序构成每个字符的结束切分位置集。

然而本案例中经过去噪后的验证码图像样本无法做到上述可以直接进行切割的程度，因此需要对垂直切割算法进行一系列的改进措施。

|  |  |
| --- | --- |
|  |  |
| 图23. 一个前期处理较为理想的样本及其P(x)曲线 | |

|  |  |
| --- | --- |
|  |  |
| 图24. 一个前期处理后仍有字符粘连的样本及其P(x)曲线 | |

通过简单的样本采样可以看出本案例中经过前期处理的图像并不具有理想的垂直字符分割的条件，首先由于图像左侧原先有较粗的干扰线，为了较好得保留字符部分，这部分粗线可能不能完全被去除，虽然它们不会对主字符区域造成影响，但是在字符分割时需要考虑它们的存在；其次由于会出现不可预测的字符粘连（验证码生成时通常会形成字符粘连以增加机器识别的难度），甚至多字符粘连，本应该作为切分位置的地方其P(x)函数值不为零，机器会误认为此处不具有切割条件。

由此，我们可以简单得出一个结论，每张样本图片不一定都存在理想的切分位置，即机器可以直接识别的切分位置，且需要对左侧干扰线做出适当的处理，以去除他们在分割时的影响。

综上，我们先提出基准轴和距离阈值的概念。基准轴即判断每个字符切分位置的标准，通关对全体样本采样计算得出，距离阈值为经验数值。实际切分时，对在基准轴的正负距离阈值范围内查找是否存在满足表达式（XX）和表达式（XX）的切分位置。这项改进措施主要针对左侧干扰线的存在，通过这项措施就可以排除掉干扰线存在部分的扫描，也就不会误把干扰线作为有效字符处理；其次这项措施也为存在字符粘连的样本后续的校正处理提供一些基础信息。

基准线通过如下的操作产生：

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| J(x)表示当前样本在x列的黑色像素个数是否为0。 |  |

计算

|  |  |
| --- | --- |
|  |  |

我们对用于训练的500个样本进行上述计算，得到样本集的TB(x)函数曲线，其图像如下：

|  |
| --- |
|  |
| 图25. 用于训练的500个样本的样本集TB(x)曲线 |

我们知道本案例中的样本有四个字符，需要五条基准轴，分别是首字符起，一二字符间，二三字符间，三四字符间和末字符尾。

通过对样本集的观察我们可以得出一个字符存在的置信区间

|  |  |
| --- | --- |
|  |  |

观察上述曲线图像可以得出一个结论，在置信区间内，上述图像的峰值即大多数样本此处均不存在黑色像素即为可能的切分基准轴。我们由此基础进行计算并最后进行人为筛选（主要针对可能的左侧干扰区内的数据）出最后的切分准轴。

找出区间内图像峰值所在的位置即求函数的极大值。我们可以解如下离散方程组：

|  |  |
| --- | --- |
|  |  |

我们解上述离散方程组得到下面这个解集：

|  |  |
| --- | --- |
|  |  |

我们人为对比图像，删选确定出我们所需要的“高峰”极大值点为下面这个解集：

|  |  |
| --- | --- |
|  |  |

把上述集合定义为基准线集

|  |  |
| --- | --- |
|  |  |

确定完基准线，我们就可以给出基准线垂直分割的算法了。

先提出距离阈值，这是一个经验阈值，需要去尝试使用不同的值来观察实际的切分效果来最终确定。

求解各字符首分隔位置

|  |  |
| --- | --- |
|  |  |

通过解改进后的离散方程组得出：

|  |  |
| --- | --- |
| 如果每组不等式的解集 则否则进行后续校正处理 |  |

然后再求解各字符末分割位置

|  |  |
| --- | --- |
|  |  |

通过解下列改进后的离散方程组得出：

|  |  |
| --- | --- |
| 如果每组不等式的解集 则否则进行后续校正处理 |  |

我们下面进行不同阈值的没有经过错误校正（即没有进行后续处理，上述解集如果为空则以使用的基准线代替）的基准线垂直分割的分割效果实验。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 分割前 | TH=4 | | | | TH=10 | | | | TH=12 | | | |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF92.jpg |  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD92-2.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD92-3.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD92-4.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD92-1.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD92-2.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD92-3.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD92-4.jpg |  |  |  |  |
|  |  |  |  |  |  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD36-2.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD36-3.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD36-4.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD36-1.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD36-2.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD36-3.jpg |  |
|  |  |  |  |  |  |  |  |  |  |  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD74-3.jpg | PD74-4 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 图26. 不含后期校正的基准线垂直分割不同测试阈值下各种实验结果 | | | | | | | | | | | | |

上述展示的各种分割结果对比鲜明的几个样例可以看出，除了样例1不同的距离阈值都可以正常分割外，样例2,3,4对于不同的距离阈值或多或少有着不同的输出响应，有的甚至在某些距离阈值下无法分割。对于设定单一阈值进行无后期校正的垂直字符分割，我们经过测试，距离阈值设定为10时相对正确率最高，为472/500 。

* + 1. 分割失败样例的原因分析

虽然总体分割成功率仍然达到90%以上，但是这个分割算法仍待改进，因为字符的正确分割是后期机器识别字符的关键，这一步需要尽可能提高分割正确率，只对极少数意外情况可以忽略不计。因此，我们接下来对一些情况下分割失败的原因进行一些调查来探寻改进方法。

|  |
| --- |
|  |
| 图27. 样例2的P(x)曲线 |

从上述曲线可以看出样例2在TH=4时分割失败的原因，作为正确的切分位置75和135不在的范围内，增加切分点的搜索范围看似可以解决这个问题。

|  |
| --- |
|  |
| 图28. 样例3的P(x)曲线 |

从样例3的P(x)曲线来看，增大搜索范围即th的值就可以解决问题，从实际的效果来看也是如此th=12时字符分割结果就完美了。那么是不是取消基准线，直接使用垂直分割的效果反而会更好呢？因为去噪等原因造成的处理后的字符有时并不完美（字符粘连或者内部空缺）没有了基准线会使得某些样本根本无法分割。

|  |
| --- |
|  |
| 图29. 样例4的P(x)曲线 |

从样例4的TH=12时分割失败和上述曲线可以看出，在位置X=121处，由于本身字符厚度较小滤波器将点滤除造成了字符内部的空缺，而这个宽度为1的空缺对字符分割造成了不利的影响，被分割器误认为这里是字符间间隙处理而造成了错误的分割结果。

可以得出一个初步结论，光通过阈值的调整几乎是不可能达到一个完美的切分效果的，因为前期去噪的不确定性，不能对样本进行全局的直接垂直分割，基准线及其阈值对于有轻微字符粘连的样本的字符分割有着很重要的作用。

|  |  |
| --- | --- |
|  |  |
| 图30. 一个轻微粘连样本及其P(x)曲线 | |

我们可以看出 X=82处是字符1和字符2视觉上应该作为分割的地方，但是这里发生了轻微的字符粘连，如果没有基准线辅助，那么分割器就不会对这两个字符作出分割处理，因为分割器找不到切分位置方程组的解。

在有基准线辅助的情况下这个样本的分割结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD50-3.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD50-4.jpg |
| 图31. 上述轻微粘连的样本在基准线辅助下的分割结果 | | | |

我们可以看出，这种轻微粘连的样本，视觉上的分隔位置就落在了基准线附近，因此在找不到方程组的解的时候用用基准线作为分割位置有一个不错的效果，但并不是左右的样本都那么“幸运”。

* + 1. 两次校正处理

下面先给出一个奇怪的分割失败的样例：

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 分割前 | TH=4 | | | | TH=10 | | | | TH=12 | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 图32. 一个各种距离阈值都分割失败的样例 | | | | | | | | | | | | |

我们可以看到这个样本不同的阈值都分割失败了，或者说分割结果不清晰，但是样本本身并没有任何字符粘连现象，理论上分隔位置方程组是有解的，但是分割器却没有在正确的位置分割。

我们采集了这个样本的P(x)曲线：

|  |
| --- |
|  |
| 图33. 样本cdgm的P(x)曲线 |

我们从曲线图可以看出应该作为分割点位置方程的解是存在的，但是这个解离开基准线太远了超出了搜索区间。但是从前文得知这个搜索区间又不宜过大，因此我们给出修正处理的方法。

* + - 1. 第一次校正

我们先给出如下一个修正算法：

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

上述给出的修正算法解决了这样一个问题，字符间有一定空隙可以作为分割点，但是紧贴字符的最佳分割点不在分割点的搜索范围内，而这个搜索范围又不宜过大，原先遇到这种情况时直接使用基准线作为分割点，这一步修正后，检查了在基准线周围是不是有零黑色像素的备用分割点，这样避免了字符周围有间隙可以分割但是却直接使用基准线错误地将字符从内部分开。

* + - 1. 第二次校正

第一次校正解决了那些本来可以正常分割但是最佳分割点不在分割区域搜索范围内的情况，且给出了一个补偿措施，第二次校正主要针对一些意外和轻微粘连字符做出修正处理。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF307.jpg |  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD307-2.jpg | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD307-3.jpg |  |
| 图34. 一个由于去噪字符内部出现空缺的情况 | | | | |

由于第三个字符n中间出现空缺，原有的算法将其从字符中间分开，因为这是个满足条件的“最佳分割点”。

我们给出如下一个算法去修正这种字符内部切分的错误。通过前面给出的样例的P(x)曲线可以看出，由于去噪造成的字符内部空缺，其宽度通常为1.因此有如下算法：

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

同时针对第一次修正算法中的遗留问题给出第二次修正：

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

第二次修正主要针对有轻微粘连的字符分割给出一些补充处理，前提认为轻微粘连的字符，粘连处的厚度相对而言应该是区间内最薄的，并且对字符内部出现间隙做了一些简单处理。

* + 1. 校正后字符分割的处理效果

通过两次校正处理，可以解决轻微字符粘连问题和基准线搜索范围矛盾的问题，我们将一些进行修正前无法正确分割的样本使用具有修正的算法重新进行分割实验。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 样本 | 无后续修正处理 | | | | 二次修正处理 | | | |
|  |  |  |  |  |  |  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cut-1526056296679-3.jpg |  |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF125.jpg |  |  |  |  |  | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\cut-1526056437197-2.jpg |  |  |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF225.jpg |  |  |  |  |  |  |  |  |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\MF307.jpg |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| 图35. 带二次修正的基准线垂直分割算法改进效果 | | | | | | | | |

从上述举例的实验效果来看，进行二次补充修正的垂直字符分割算法能够更好地适应样本特征，针对一些没有补充修正时会出现错误的情况有了较好的改正，能够自适应处理更多的样本，大大增加了该字符分割算法针对这批样本的适用率，提高了字符分割的正确率。

* + 1. 字符分割的正确率

本案例中的字符分割算法需要设定几个参数，分别是基准线，搜索距离阈值，意外判断距离阈值，粘连判定阈值，参考权重 。

下文所叙述的分割正确率都由肉眼观察得知，分割成功即为当前分割块只含单字符内容且肉眼可辨认。

首先本案例中的基准线集合已经由前文算出这里不再赘述，为了简化处理，我们先使用无修正的基准线垂直字符分割来确定搜索距离阈值的选定。下面是针对同一组样本选取不同的值的分割正确率。

|  |
| --- |
|  |
| 图36. 无修正的基准线垂直分割设置不同th值时的正确率 |

因此我们设定搜索距离阈值为10并进行后续参数的实验。

这里我们所涉及的轻微粘连多数是由于去噪不够充分，两字符间间距过小而误把干扰线作为字符的一部分而产生的粘连，基于这一实际我们把粘连判定阈值设置为4 。

这里的参考权重a,b所反映的是针对存在粘连的字符的分割，是倾向于在最薄除分割还是距离基准线最近处分割。这里由于是相对参数，我们固定a=1，对b的取值进行测试。

在此之前，我们先使用a=1,b=0对dth的取值进行测试。dth值针对的是字符内部出现空缺的情况，这个阈值的设定主要是为了防止两个字符之间的间隙正好只有1个像素宽的“巧合”，因此这个参数的设定是针对有空隙字符的，对有粘连的情况影响不大。

|  |
| --- |
|  |
| 图37. th=10,w=4,a=1,b=0情况下不同的dth对应的分割正确率 |

从上图中分析看出，dth值设置为1~3来预防上文提到的“巧合”即可，实际上对于排除掉字符内部的空隙造成分割失败的影响效果还是可观的。

|  |
| --- |
|  |
| 图38. th=10,dth=2,w=4,a=1情况下不同的b对应的分割正确率 |

上述实验结果也印证了基准线只起分割参考作用，越倾向于重视基准线肉眼观察到的分割正确率越低，因此本案例的粘连分割方法选取在搜索区域内的最薄除切分，以取得不错的效果。

字符分割的正确率是后续机器字符识别的保证，后续的字符识别难免会因为字符块内尚有干扰去除不干净等原因进一步使最终正确率下降。这里字符分割的正确率达到97%还是可以接受的。

* 1. 单字符样本冗余部分去除

在字符分割完成后得到的单字符样本，在字符区域的上下都有部分空白区域，因为输入到机器训练的样本是需要归一化的，如果直接将得到的单字符样本进行大小归一化，空白部分所占比例未免过大，即有效信息所占比例太低，容易造成机器识别的误差过大，因此这里先将得到的单字符样本进行上下部分的冗余切除，让字符“顶格”。

|  |  |
| --- | --- |
| 对得到的字符分割后的单字符原始样本图片（宽度不定，高度50）样本逐行扫描每一行的黑色像素个数 |  |
|  |  |
|  |  |
|  |  |
|  |  |

我们绘制出如下的YB曲线：

|  |
| --- |
|  |
| 图39. YB曲线 |

根据YB的图像，可是获得切除冗余部分的首尾经验准轴

使得98%的样本字符在hcs和hce之间

换言之hcs和hce可以通过解方程

|  |  |
| --- | --- |
|  |  |
| 对于每一个样本的实际切割修正可以通过如下操作实现，设定一个容差阈值th |  |
| 顶部切割位置解如下方程组实现 |  |
|  |  |
| 如果解集则取否则 |  |
| 底部切割位置解如下方程组实现 |  |
|  |  |
| 如果解集则取否则 |  |

我们给出几个经过冗余切除后的单字符样本：

|  |  |  |  |
| --- | --- | --- | --- |
| 切除前 | 切除后 | 切除前 | 切除后 |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD22-4.jpg20x50 | 20x36 | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD39-1.jpg19x50 | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HD39-1.jpg19x38 |
| C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\PD45-4.jpg31x50 | C:\Users\eanjc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\HD45-4.jpg31x36 | 22x50 | 22x35 |
| 图40. 单字符高度方向冗余切除前后对比 | | | |

冗余切除后提高了字符有效信息（黑色像素）占总像素的比值，使得进行下一步图像归一化时不同样本的字符占据图像的比例更加均衡一些，一定程度上避免了在机器字符识别时由于字符占比大小的差别过大而导致的字符识别正确率偏低。

* 1. 单字符样本大小归一化