使Tesseract开源OCR引擎适应于多语言OCR

雷·史密斯（Ray Smith Daria）Antonova Dar-Shyang Lee

Google Inc.，1600 Amphitheatre Pkwy，Mountain View，CA 94043，USA。

摘要我们描述了使Tesseract开源OCR引擎适应多种脚本和语言的工作。一直致力于启用通用的多语言操作，这样，除了提供文本集之外，新语言所需的自定义要求也可以忽略不计。尽管需要对包括物理布局分析和语言后处理在内的各个模块进行更改，但除几个限制外，无需对字符分类器进行任何更改。 Tesseract分类器已轻松适应简体中文。从随机的书籍样本中获得的英语（一种欧洲语言和俄语的混合）的测试结果显示，词错误率在3.72％和5.78％之间保持合理一致，而简体中文的字符错误率仅为3.77％。

关键字Tesseract，多语言OCR。

1.简介对基于拉丁语的OCR的研究兴趣比十年前减少了，转而使用中文，日文和韩文（CJK）[1,2]，随后是阿拉伯语[3,4]，然后是印地语[5]，6]。这些语言给分类器以及OCR系统的其他组件带来了更大的挑战。中文和日语共享汉字，其中包含成千上万种不同的字符形状。朝鲜语使用的汉字具有自己的数千种文字，还使用汉字。字符数比拉丁文大一到两个数量级。阿拉伯语大多是用连接的字符书写的，并且其字符的形状会根据单词中的位置发生变化。印地语将少量字母组合成代表音节的数千种形状。当字母结合在一起时，它们形成绑线，其形状仅与原始字母模糊不清。 Hindithen通过将所有符号和单词shiro-reka结合在一起，从而将CJK和阿拉伯语的问题结合在一起。

研究方法使用了特定于语言的解决方法，从而以某种方式避免了问题，因为这比试图找到适用于所有语言的解决方案要简单。例如，汉，韩文和印地文的大字符集主要由数量少得多的组件组成，汉族中的部首字母，韩文中的Jamo以及印地语中的字母。由于开发针对少量类别的分类器要容易得多，因此一种方法是识别部首[1,2,5]，并从部首组合中推断出实际字符。与韩文或印地文相比，这种方法对韩语来说更容易，因为汉字中的部首变化不大，而在汉族中，部首经常被压缩以适合角色，并且大多与其他部首接触。印地语在形成辅音辅音连字时，通过改变辅音的形状来进一步采取这一步骤。阿拉伯语的另一种针对特定语言的变通方法，其中很难确定字符边界以将连接的组件分割成字符。一种常用的方法是对单个垂直像素条进行分类，每个垂直像素条都是一个局部字符，然后使用对字符边界建模的隐马尔可夫模型将分类组合在一起[3]。

Google致力于提供尽可能多的语言服务[7]，因此我们也有兴趣将Tesseract开源OCR引擎[8，9]改编为多种语言。本文讨论了迄今为止我们在使Tesseract全面国际化方面所做的努力，以及其中一些可能令人惊讶的轻松。我们的方法是使用语言通用方法，以尽量减少覆盖多种语言的人工工作。

2.拉丁文Tesseract的评论

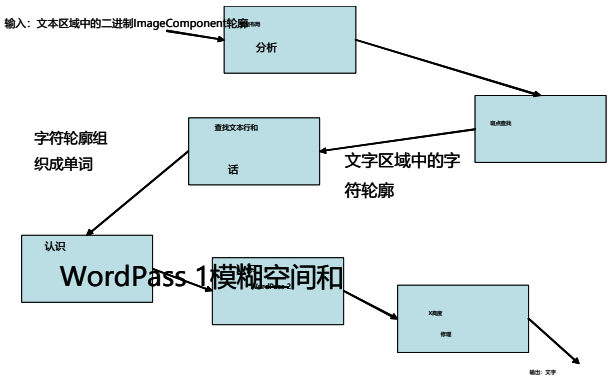


图1. Tesseract的顶级框图

图1是Tesseract基本组成部分的框图。Tesseract [10]的新页面布局分析从一开始就设计成与语言无关，但是该引擎的其余部分都是为英语开发的，没有太多思想。它可能如何适用于其他语言。在注意到当时的商业引擎严格用于黑白文本之后，Tesseract的最初设计目标之一是，它应该能够将黑白文本（逆向视频）识别为与黑白文本一样容易。这引导了设计（很可能是事实证明）朝着连接组件（CC）分析的方向进行，并按组件的轮廓进行操作。 CC之后的第一步

分析是在文本区域中找到斑点。斑点是推定的可分类单位，可以是一个或多个水平重叠的CC及其内部嵌套的轮廓或孔。 Aproble m正在检测框内的反文字与字符内的孔。对于英语，很少有字符（可能是©和®）具有超过2个级别的轮廓线，而且很少有2个以上的孔，因此，任何破坏这些规则的斑点都是“明显地”一个包含反字符的框，甚至是黑白字符周围框架的内部或外部。

在确定哪些轮廓组成斑点后，文本行查找器[11]依靠文本行上相邻字符的垂直重叠来检测（仅水平）文本行。对于英语，重叠和基线表现得很好，可以用来非常精确地检测到很大角度的偏斜。在找到文本行之后，固定音高检测器将检查固定音高字符布局，并根据固定音高决策运行两个不同的词分割算法之一。大部分的认知过程独立地作用于每个单词，
然后是最终的模糊空间解析阶段，其中确定了不确定的空间。

图2是单词识别器的框图。在大多数情况下，blob对应于一个字符，因此单词识别器首先对每个blob进行分类，然后将结果提供给字典搜索，以在单词中每个blob的分类器选择组合中找到一个单词。如果单词结果不够好，则下一步是将无法识别的字符切碎，从而提高分类器的置信度。在用尽所有可能的切分之后，最好首先搜索所得的分割图，将切好的字符片段或部分字符分解成原始图像中的多个CC。在最佳优先搜索的每个步骤中，将对任何新的斑点组合进行分类，然后将分类器结果再次提供给字典。单词的输出是在根据单词是否在字典中和/或在单词周围的标点排列进行加权后，具有最佳的基于距离的整体评级的字符串。对于Englishversion，大多数标点符号规则都是硬编码的。

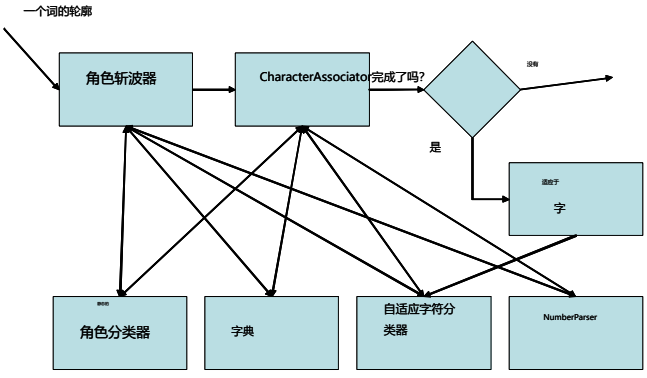


图2。Tesseract单词识别的框图

图像中的单词被处理两次。第一步，将成功的单词，即词典中的单词和不存在危险模棱两可的单词，传递给自适应分类器进行训练。自适应分类器一旦有足够的样本，就可以提供分类结果，即使是在第一次通过时。在第二遍，在第一遍的单词不够好是如果自适应分类器自第一次通过单词以来获得了更多信息，则进行第二次处理。

从前面的描述中，显然非拉丁语言的设计存在问题，第3、4和5节将处理一些更复杂的问题，但是其中一些问题只是复杂的工程。例如，字符类的一个字节代码不足，但是应该用UTF-8字符串还是更宽的整数代码代替？最初，我们将Tesseract适应拉丁语，并将字符代码更改为UTF-8字符串，因为这是最灵活的，但是结果却导致字典表示出现问题（请参阅第5节），因此我们最终使用了索引插入UTF-8字符串表作为内部类代码。

3。布局预处理Tesseract的“ textord”（文本排序）模块的几个方面都需要进行更改，以使其更加语言化，

独立。本节讨论这些更改。

3.1垂直文本布局中文，日文和韩文在不同程度上均水平或垂直读取所有文本行，并且经常在单个页面上混合方向。这个问题不是CJK独有的，因为英语杂志的页面通常在照片或文章的侧面使用竖排文字来表示摄影师或作者的信誉。通过页面布局分析可以检测到竖排文字。如果制表位上的大多数CC的左侧都在左侧选项卡上，而右侧都在右侧选项卡上，则制表位之间的所有内容都可能是一行垂直文本。为了防止假阳性的不稳定性，进一步的限制要求垂直文本的CC之间的垂直中间距离应小于CC的平均宽度。如果页面上的大多数CC垂直对齐，则将页面旋转90度并再次运行页面布局分析，以减少在垂直文本中发现错误列的机会，然后少数原来水平的文本将在旋转后的页面中变为Verticaltext ，并且文本的正文将为水平。

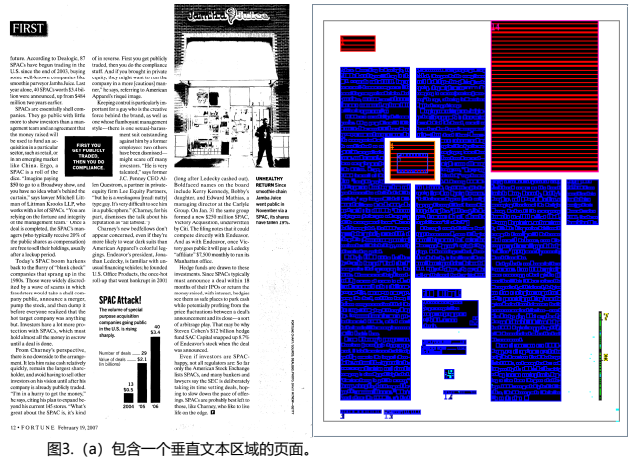


图3.（a）包含一个垂直文本区域的页面。
（b）检测到的区域带有红色的图像，水平文本为蓝色和黄色垂直文本。

按照最初的设计，Tesseract没有处理垂直文本的功能，并且在代码中有很多地方都进行了一些假设，将过度字符排列在

水平文本行。幸好，

Tesseract在带符号的整数坐标空间中的CCs轮廓上进行操作，这使旋转90度的次数变得微不足道，并且不必关心坐标是正还是负。因此，解决方案只是差速旋转

页面上的垂直和水平文本块，并旋转分类所需的字符。图3显示了一个针对英语文本的示例。图3（a）的页面在右下角包含垂直文本，即

图3（b）中检测到的文字以及其余文字。在图4中，垂直文本区域旋转了90度

顺时针方向（居中于图像的左下角），因此它看起来远低于原始图像，但处于水平方向。

图4.垂直文本是不同的

旋转使其定向

水平地。

图5示出了中文文本的示例。垂直主体文本从图像中旋转出来使其变为水平状态，而原先为水平的标题保留在其开始位置。垂直和水平文本块在坐标空间中分开，但是Tesseract关心的只是文本线是水平的。文本块的数据结构记录在块上执行的旋转，以便在将字符传递到分类器时可以将反向旋转应用于字符，以使它们直立。自动方向检测[12]可用于确保文本在传递到分类器时是直立的，因为垂直文本可能具有相对于阅读方向至少3个不同方向的字符。在Tesseract处理旋转的文本块之后，坐标空间将重新旋转回原始图像方向，以便报告的字符边界框仍然准确。

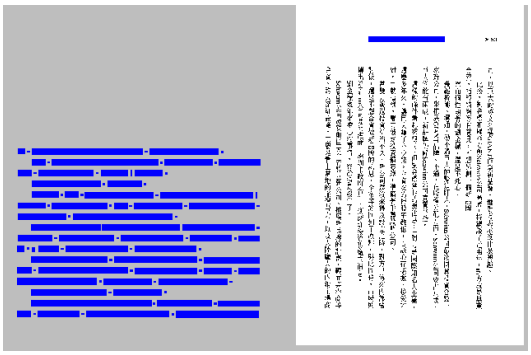


图5.繁体中文的水平文本检测。由于大部分文本是垂直的，因此在Tesseract内部将其逆时针旋转90度，使其位于图像的外部，

但线条是水平的。页面标题是

已经水平，仍然落后。

3.2文本行和单词查找最初的Tesseract文本行查找器[11]假定组成字符的CC大部分垂直重叠在文本行的大部分上。一个真正的例外是i点。对于一般语言，这是不正确的，因为许多语言的变音符号都位于文本行的大部分上方和/或下方。以泰语为例，从文本行正文到变音符号的距离可能相当大。 Tesseract的页面布局分析旨在通过将文本区域细分为统一的文本大小和行距的块来简化文本行查找。这使得强制拟合行距模型成为可能，因此对文本行查找进行了修改以利用这一优势。页面布局分析还估计了文本区域的残留歪斜，这意味着文本行查找器不再必须对歪斜不敏感。

修改后的文本行查找算法可从布局分析中为每个文本区域独立工作，并从搜索较小CC的邻域（□□相对于估计的文本大小）开始，以找到最接近正文文本大小的CC。如果附近没有正文文本大小的CC，则将一个小的CC视为可能的噪音，并将其丢弃。 （通常在目录中会发现虚线/虚线引线的例外。）否则，将构造并使用包含小CC及其较大邻居的包围盒，以代替小CC的包围盒。按照预测。

使用小型CC的修改后的框，从CC的边界框平行于估计的倾斜水平线构建“水平”投影轮廓。然后，动态编程算法在投影配置文件中选择最佳的分割点集。成本函数是切割点处的轮廓输入项的总和加上它们之间间距变化的度量。对于大多数文本，配置文件条目的总和为零，并且方差有助于选择最规则的行距。对于更复杂的情况，方差和小型CC的修改后的边界框结合在一起有助于引导行切割，从而最大程度地增加行距。带有适当身体特征的变音符号。

确定切割线后，将整个连接的零部件放置在文本线中，使其垂直垂直重叠最多（仍然使用修改的框），除非零部件与多条线强烈重叠。假定这些CC是来自多行触摸的字符，因此需要在剪切行处进行剪切，或者是首字下沉，在这种情况下，它们应位于重叠的顶部行中。即使对于阿拉伯语，该算法也能正常工作。

提取文本行后，一行上的斑点被组织成识别单元。对于拉丁语言，逻辑识别单元对应于以空格分隔的单词，这自然适用于基于字典的语言模型。对于没有空格的语言（例如中文），不清楚相应的识别单元应该是什么。一种可能性是处理每个中文符号作为识别单元。但是，鉴于汉字符号由多个字形（基）组成，如果不借助识别，将很难获得正确的字符分割。考虑到在此处理的早期阶段可用的信息量有限，解决方案是在标点处分解斑点序列，可以根据它们的大小和到下一个斑点的间距非常可靠地检测到斑点序列。虽然这并不完全

解决了很长的斑点序列问题，这是确定分割图时确定效率和质量的关键因素，这至少可以将识别单元的长度减少到更易于管理的大小。

如第2节中所述，黑白文本的检测基于轮廓的嵌套复杂度。同样的过程也可以拒绝非文本，包括半色调噪点，侧面的黑色区域或侧边栏或反向视频区域中的大型容器盒。过滤的一部分基于对斑点的拓扑复杂性的度量，并根据内部组件的数量，嵌套孔的层数，周长与面积之比等进行估算。但是，无论如何，繁体中文字符的复杂性通常都超过了包装在框中的英文单词的复杂性。解决方案是对不同的语言应用不同的复杂度阈值，并依靠后续分析来恢复任何错误拒绝的斑点。

3.3估算西里尔文字的x高度完成文本行查找步骤并将斑点块组织成行后，Tesseract估算每个文本行的x高度.x高度估算算法首先确定基于最大和最小可接受x高度的界限根据为该块计算的初始行大小。然后，对于每条线，分别量化在线上出现的斑点的边界框的高度并将其聚合为直方图。从该直方图中，x高度查找算法查找两个最常见的高度模式，它们相距足够远，以达到潜在的x高度和上升高度。为了获得对某些噪声的鲁棒性，该算法确保了高度相对于该行上的斑点总数，选择为x高度和上升高度的模式具有足够的数目或出现次数。

对于大多数拉丁字体，该算法效果很好。但是，当按原样应用于西里尔字母时，Tesseract无法为大多数线找到正确的x高度。结果，在Russianbooks的数据集上，Tesseract的单词错误率被证明是97%。如此高的错误率的原因是双重的。首先，西里尔字体的递增统计与拉丁字母显着不同。仅降低期望的每行上升数阈值并不是一种有效的解决方案，因为一行文本很少包含一个或不包含上升字母。如此差的性能的第二个原因是西里尔字体的大小写含混不清。例如，在33种大写现代俄语字母中，只有6种小写字母的形状与大多数字体中的大写字母明显不同，因此，当使用西里尔字母时，Tesseract会很容易被错误的x高度信息误导。会轻易将小写字母识别为大写字母。

我们解决西里尔字母x高度问题的方法是调整行上的最小上升数，考虑下降统计信息，并更有效地使用同一文本块中相邻行的x高度信息（一个块是由分页版面分析确定的文本区域，其文本斑点和行间距的大小一致，因此可能包含相同或相似字体大小的字母。

对于给定的文本块，改进的x高度查找算法首先尝试分别查找每行的x高度。基于计算结果的每一行都属于以下四类之一：（1）找到x高度和上升模式的行，（2）找到下降器的行，（3）可以使用的通用Blob高度作为对上限高度或上限x高度的估计值，（4）识别出以上所有内容的行（即最有可能包含噪音的线条，这些斑点的斑点太小，太大或大小不一致。如果在该块中找到第一类中具有可靠的x高度和上升者高度估计的行，则将其高度估计用于第二类中具有类似x高度估计值的行（存在降序的行）。对于第三类中的那些行（未找到上升或下降），使用了相同的x高度估计，其最常见的高度在x高度估计的一小部分内。如果逐行方法未找到任何可靠的x高度和上升高度模式，则将汇总文本块中所有斑点的统计信息，并使用此累积信息重复搜索x高度和上升高度模式。

作为上述改进的结果，错误一词

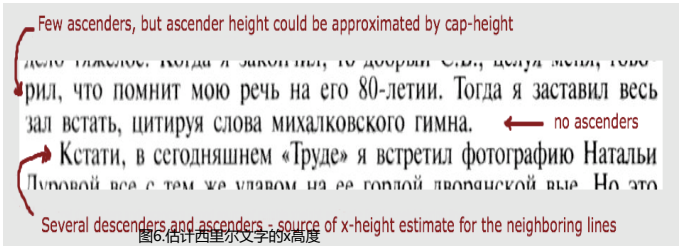


图6.估计西里尔文字的x高度

一套俄文书籍的测试率降到了6%。改进后，由于无法估计文本行的正确x高度，测试集仍然包含一些错误。但是，在许多这样的情况下，即使是人类读者也必须使用来自相邻文本块或有关书籍的共同组织的知识的信息来确定给定的行是大写还是小写。

4.字符/单词识别在使Tesseract适应多语言OCR方面要克服的主要挑战之一是将最初为字母语言设计的内容扩展为处理诸如中国和日语之类的表意语言。这些语言的特点是符号大量，缺少清晰的单词边界，这对针对从小字母很好界定的单词而设计的搜索策略和分类引擎进行了严格的测试。 4.1分段和搜索如第3.2节所述，对于像中文这样的非空格分隔的语言，与西方语言中的单词对等的识别单元现在对应于标点符号分隔的短语。处理这些短语时，需要考虑两个问题：它们比拉丁语中的典型单词涉及更深入的搜索，并且它们不对应于词典中的条目。 Tesseract在分割图上使用最佳优先搜索策略，该策略随blob序列的长度呈指数增长。

当在字典中找到结果时，切分点和终止条件会给汉语短语分类时经常耗尽可用资源。要解决此问题，我们需要大幅度减少排列中评估的分割点的数量，并设计出更容易满足的终止条件。

为了减少分割点的数量，我们将大致恒定的字符宽度的约束并入了诸如中文和日语之类的等间距语言中。在这些语言中，字符多数具有相似的长宽比，并且在其位置上是全角还是半角。尽管归一化的宽度分布会随字体而变化，并且间距会因行对齐和数字或拉丁词的包含而变化，这并不罕见，但总的来说，这些约束为特定的分段点是否与另一个分段点兼容提供了强有力的指导。因此，使用与分割模型的偏差作为代价，我们可以消除许多令人难以置信的分割状态，并有效地减少了搜索空间。我们还使用此估计值基于最佳局部解决方案来修剪搜索空间，从而有效地进行波束搜索。当没有进一步的扩展可能会产生更好的解决方案时，这也提供了终止条件。

另一个强大的约束是短语中字符脚本的一致性。当我们包含多个形状类

脚本之间，跨不同脚本的字符之间的混淆错误变得不可避免。虽然我们可以建立

页面的主要脚本或语言，我们也必须允许使用拉丁字符，因为外语书籍中英语单词的出现非常普遍。在一个识别单元内的字符具有相同脚本的假设下，如果改善整个单元的总体脚本一致性，我们将促进字符解释。但是，如果单词或短语是真正的混合脚本，则基于先验盲目地提升脚本字符实际上可能会损害性能。因此，仅当topexploration中超过一半的字符属于同一脚本，并且像其他排列一样，对形状识别分数加权调整时，才应用约束。

4.2形状分类用于大量类别的分类器仍然是一个研究问题;即使在今天，尤其是当要求它们以OCR所需的速度运行时[13，14]。的诅咒

维度是主要的责任。 Tesseract形状分类器在不需要任何大的修改的情况下，可以很好地在5000个汉字上很好地工作，因此它似乎非常适合大型班级问题。这个结果值得解释，因此在本节中，我们将描述Tesseract形状分类器。

特征是形状轮廓的多边形近似的分量。在训练中，从多边形逼近的每个元素中导出（x，y位置，方向，长度）的4维特征向量，并将其聚类以形成原型特征向量。 （因此，名称为Tesseract。）在识别中，多边形的元素被分解成等长的较短片段，因此长度维从特征向量中消除了。多个短特征相互匹配来自训练的原型特征，这使得分类过程对残破字符的鲁棒性更高。

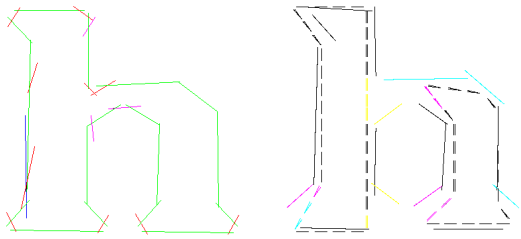


图7。（a）《罗马时报》的h原型，（b）中断的h与原型的匹配。

图7（a）显示了字体Times Roman的字母“ h”的示例原型。绿线段代表重要簇的簇均值，其中包含时代罗马时期几乎所有“ h”样本的样本。蓝色部分是簇，意味着与另一个簇合并形成一个重要的簇。未使用洋红色段，因为它们与现有的重要群集匹配。红色部分的样本数量不足以构成显着的，并且无法与任何相邻的聚类合并以形成显着的聚类。

图7（b）显示了未知字符的较短特征如何与原型匹配，以实现对残破字符的不敏感性。短而粗的线是未知特征（被打破的“ h”），而较长的字符则是原型特征。颜色代表匹配质量：黑色->良好，品红色->合理，青色->较差，黄色->不匹配。垂直原型都很好地匹配，尽管h被打破了。

形状分类器分两个阶段运行。第一阶段称为类修剪器，它使用与Locality SensitiveHashing（LSH）密切相关的方法将字符集缩减为1-10个字符的简短列表。最后阶段从候选清单中的字符原型计算未知对象的距离。

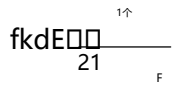
最初设计为节省时间的简单方法

通过优化，类修剪器通过分别考虑每个3-D特征来划分高维特征空间。代替LSH的哈希表，有一个简单的查找表，该表返回范围为[0，3]的整数向量，每个字符集对应一个字符集，该值表示该字符集的近似匹配度角色类原型的特征。向量结果在未知的所有特征之间求和，总分数在最高分数的一小部分之内的类别将作为第二阶段要归类的候选清单返回。类修剪器相对较快，但是其时间与类的数量以及要素的数量呈线性比例关系。

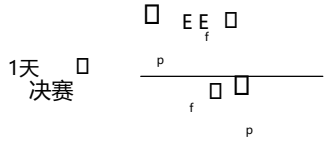
第二阶段分类器计算每个特征与其最近原型的距离df，作为二维空间中原型线的（x，y）特征坐标的平方欧几里得距离d的平方，再加上与该特征线成角度的加权（w）差原型：

22□ddf□

从本质上讲，这是一个生成分类器，它可以计算与理想目标的距离。使用以下公式将特征距离转换为特征证据Ef：



常数k用于控制证据随距离衰减的速率。当特征与原型匹配时，特征证据Ef被复制到原型Ep。由于原型期望多个特征与它们匹配，并且“最佳匹配”的收集是为了速度而独立完成的，因此特征和原型证据的总和可能不同。总和由特征数量和原型长度Lp归一化，结果转换回距离：



请注意，实际的实现使用定点整数算术运算，并且上述公式中省略了很多比例缩放常数，否则这些比例常数会混淆计算。

第二阶段分类器的部分优势在于，每个类标签中都可以有多个理想值（在Tesseract中称为configs），因此可以实现由字体或版式的任意差异引起的多模式分布。上述匹配过程在计算最终距离时会选择最佳配置。从这个意义上说，分类器因此实际上是最接近的邻居分类器。

我们假设类修剪器和辅助分类器对于大量类来说效果很好，这是因为它们在多个小尺寸的“弱分类器”之间使用投票，而不是依赖于一个高维度的分类器。这是提振[15]背后的概念，只是当前的财富分类器尚未加权。特征空间的尺寸被量化为256个级别，这提供了足够的精度来存储CJK字符和印度音节的复杂形状，并且最终计算避免了类似于类修剪器的尺寸诅咒。

5.上下文后处理

Tesseract的培训过程通过提供一种从任意单词列表生成新语言的字典的方法，支持部分扩展语言模型。为了紧凑和快速搜索，这些字典由有向无环词图（DAWG）表示。在原始实现中，DAWG数据结构用于顺序搜索多个词典，包括预生成的系统词典，文档词典（由OCRed文档中的单词动态构建）和用户提供的单词列表。

最初，DAWG中的每个边都存储了一个8位字符，代表用于DAWG中相应转换的字母。但是，这种表示是有限的，因为

以这种方式处理多字符字素和多字节Unicode字符很尴尬。修改了DAWG数据结构，以存储字符分类器使用的unicharset ID。这大大简化了建立和搜索DAWG的过程。另一个改进是并行搜索所有DAWG。要确定给定的字符串是否是有效的词典单词，现在从初始的一组“活动” DAWG开始。当考虑单词中的每个字母时，将此集合简化为仅包含那些仍“接受”部分字符串的字母。在该过程结束时，“活动的” DAWG组仅由包含单词的那些DAWG组成。这种重组使我们能够动态地加载任意数量的DAWG，而不必为搜索每个新添加的DAWG添加任何自定义支持。这也是允许Tesseract支持任意语言组合的必要修改之一-Tesseract处理多语言文本所需的功能。

5.1约束模式Tesseract中的标点和数字状态机是硬编码的，没有泛化到拉丁文字之外。即使对于拉丁文字，状态机也不接受很大一部分有效的标点和数字模式。为了帮助Tesseract在非拉丁文字中处理标点符号和数字，Tesseract的培训过程扩展了代码，以收集和编码一组频繁出现的标点符号和数字模式。收集这些模式的步骤被实现为与大型文本语料库的处理并行进行，以构造给定语言的字典。为了表示和匹配生成的模式，使用了已经存在的用于生成和搜索单词DAWG的代码。对确定给定分类器选择是否为启用语言的Tesseractto中的有效单词的算法进行了一些修改，对包含单词，标点和数字模式的所有DAWG进行了同时搜索。通过此修改，可以删除所有针对数字和标点的语言和脚本特定的硬编码规则。标点和数字模式的生成和搜索过程被设计为完全由数据驱动，到目前为止，对于任何一种语言，都不需要特殊的大小写。

5.2解决形状歧义除了预训练的形状模板外，Tesseract的shape classifier包括一个自适应组件，用于OCRed文档中看到的字符的模式。为了确保对自适应组件进行可靠数据的训练，分类器仅适应明确的字典词。如果满足两个约束条件，则将OCRed单词称为“明确”。第一个是形状分类器必须在单词的所有其他选择中识别出明显的赢家（即，最佳选择的分类器评级必须明显高于次优选择的评级）。第二个约束是不能存在字典词，其形状对于词的最佳选择而言是模棱两可的。此要求对识别速度也很重要，因为一旦（取决于分类器分数）Tesseract找到了这样的单词选择，它就可以接受识别结果并停止对该单词的进一步处理。

对于拉丁文字，Tesseract包含一个手工制作的数据文件（称为“危险歧义”文件），该文件指定大多数拉丁字体中哪些字母组合固有地模棱两可。一种使用其他脚本实现此功能的可扩展解决方案，是开发一种自动生成任意给定n-gram对列表的自动方法。

语言。从大型文本语料库中收集了一组n-gram（在这种情况下为uni-，bi-gram），其组合权重占该语言中所有n-gram的95％。 n-gram用一组常用字体以几种降级模式和曝光进行渲染。然后，在渲染图像上运行Tesseract的形状分类器，以获得每个n-gram的一组最高得分分类。汇总每个n-gram的最终分类得分统计信息，并丢弃具有低分类得分的离群值（在某些字体和降级模式下，字符被渲染为无法识别，并且在某些情况下只会污染数据）。然后，对于每个不正确的OCRed n-gram和相应的正确的n-gram对，计算歧义分数。模糊度分数被定义为形状分类器感知到的错误和正确的n-gram（跨所有字体和降级模式汇总）与语言中正确的n-gram频率之间的相似度的函数。为了在Tesseract的速度和准确性之间达到所需的平衡，有必要选择一个阈值，该阈值是由于n-gram形状歧义（根据n-gram频率和分类器错误统计信息计算）而允许发生的预期错误数。生成“危险歧义”文件将歧义n-gram对按歧义分数的不递增顺序进行排序，并确保文件中包含所需期望错误率的适当的得分最高歧义数。

使用这种自动方法生成的数据文件，与使用手工制作的“危险歧义”文件相比，有可能在拉丁文字（EFIGS数据集）上实现类似的改进（尽管在某些语言中结果稍差）。在俄罗斯数据集上使用自动生成的文件可以减少10％的单词错误率。对其他语言生成的文件进行检查也表明，自动生成的文件包含相当多的常见形状混淆，但是将对相应的数据集进行进一步测试需要量化改进。

5.3处理高语种的语言Tesseract的速度和准确性与词典的质量息息相关，要最大限度地增加这些语言同时最小化存储词典所消耗的空间始终是一个挑战。

从语料库以高度变化的语言生成字典是一项特别困难的任务。高屈折语言中单词的频率分布更均匀，因此要实现相同的语言覆盖范围，就需要使用更大的词典。此外，即使是更频繁使用的单词中的许多单词形式也可能在训练语料库中出现的次数不够多，无法包含在其中。字典，因此字典可能无法在训练语料库之外概括。图8通过将语料库的覆盖范围与选择形成词典的最常用单词的数量进行图形化比较，说明了语言集合上的这一问题。

由于DAWG数据结构的头部和尾部压缩，添加DAWG中已经存在的单词的变体形式可能会导致字典整体大小的很小增加。这是因为该单词的开头和结尾可能已经存储在DAWG中（例如，如果已经插入“ talk”和“ make”，则在字典中添加单词“ talking”会相对便宜）。

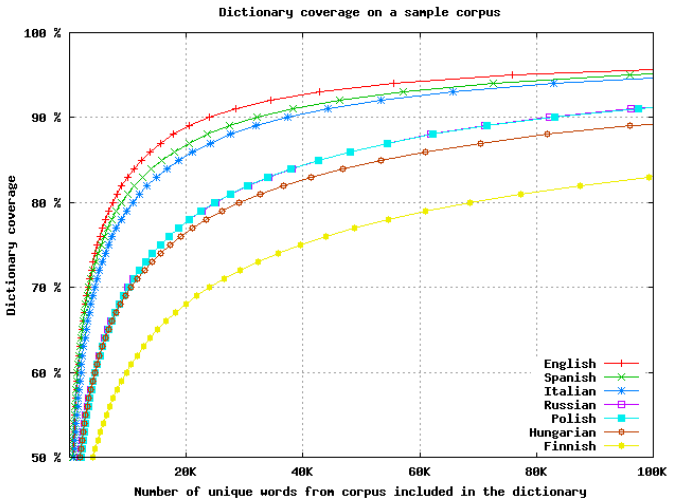


图8.词典中不同大小词典的语料库覆盖率

语言集合

为了解决捕获更多变形单词形式的问题，对词典生成过程进行了修改，以生成单词变体（单词列表中不存在）并将其添加到词典的步骤。首先，如先前所做的那样，DAWG是从单词列表中构建的。然后，对于DAWG中每个词根，都会收集一组后缀。使用组平均层次聚类聚类算法对集合进行聚类。合并结果簇中的后缀集以形成扩展后缀集。然后，针对每个词根和相应的后缀集（在DAWG的第一次遍历期间预先计算），确定最接近的扩展后缀集。由扩展集中的后缀组成的新词将插入到DAWG中。

6.状态/实验结果我们的数据集由Google图书搜索项目收集的随机选择的图书中的页面组成。对于每种语言，选择了100本随机书籍，并从每本书籍中随机选择10页进行手工地面练习。因此，这些页面涵盖了从布局，字体，图像质量到主题和术语使用的各个方面的各种内容。

然后将数据集分为训练，验证和测试集，其中训练和验证集用于在开发过程中学习和基准化算法，测试集保留用于发布期间的最终评估。表1总结了几种语言的数据集大小和当前准确性。对于字母语言，我们报告了字符级和单词级的错误率。对于单词含义不明确的中文，我们仅报告字符替换率。EFIGSD是英语，法语，意大利语，德语，西班牙语和荷兰语的组合。

对于简体中文，我们注意到不同书籍之间的错误率差异很大。差异主要归因于字体和质量的变化。在页面质量和准确性合理的地方，错误主要是由于相似或接近相同的形状类别之间的混淆所致。我们计划增加形状匹配器中特征空间的容量，这应该有助于区分相似的

形状。在几乎相同的情况下，跨字体的类内变体可能大于类间变体。幸运的是，它们的用法和先验差异如此之大，以至于当我们为CJK引入语言模型时可以很容易地对其进行纠正。

表1.各种语言的错误率

语言数量	（百万）	字数（百万）	字符错误率（%）	字误率（%）
英语39 4 0.5 3.72				
EFIGSD 213 26 0.75 5.78				
俄文38 5 1.35 5.48				
简体中文	0.25不适用3.77不适用			
印地语1.4 0.33 15.41 69.44				

7.结论和未来的工作我们已经描述了使Tesseract适用于多种语言的实验，并且令人惊讶地发现这主要是工程问题。没有任何

对分类器进行了重大更改，我们能够获得多种基于拉丁语的语言，俄语甚至简体中文的良好结果。到目前为止，印地语的结果令人失望，但是我们发现我们的测试集包含新旧字体的混合，并且很大一部分错误是由于训练集不包含旧字体的事实造成的。这项工作尚未涵盖从右到左书写的语言，这主要是另一个工程问题，但是阿拉伯语具有Tesseract可能无法解决的一系列问题，即字符分割。我们还没有讨论过的另一种语言是泰语，它带来了高度歧义的字符问题，并且与中文一样，单词之间没有空格。

一个重要的未来项目是改善训练过程，使其能够使用真实数据进行训练，而不仅仅是具有字符边界框的合成数据。这将大大提高印地语的准确性。我们还需要测试阿拉伯语和泰语，因为我们预计还会有更多问题。对于中文，日语和泰语，我们需要允许使用语言模型来搜索任意串联的单词的空间，因为这些语言的单词之间没有空格。尽管德语复合功能会增加大小写更改和插入字母的复杂性，但相同的功能对德语也很有用。

8.参考文献[1]

Nagy，G.，“汉字识别：二十五年的前景”，第9诠释。
Conf。关于模式识别，1988年11月，第163-167页。

[2]夏芳。“基于知识的子模式分割：汉字分解”，图像处理，1994。
程序ICIP-94，IEEE国际标准。
Conf。第1卷，1994年11月13日至16日，pp179-182。

[3]陆志东，施瓦茨，R。Natarajan，P。Bazzi，I。Makhoul，J。“BBN BYBLOS OCR系统的进展”，Proc。 5thInt。
Conf。文件分析与识别，1999，pp337-340。

[4] Kanungo，T.，Marton，G.A.，Bulbul，O.，“Omnipage vs. Sakhr：两种阿拉伯OCR产品的配对模型评估”，Proc.Natl.Acad.Sci.USA，87：3877-5。 SPIE 3651，1999年1月7日，第109-120页。

[5] Bansal，V .；R.M.K，Sinha，“以梵文书写的印地文文本的完整OCR”，Proc。第六国际Conf on Document Analysis and Recognition，2001，pp800-804。

[6] Govindaraju，V.等。等Proc 1st Int。“启用数字访问多语种印度文档的工具”。图书馆文献图像分析研讨会，2004年，pp122-133。

[7] Google官方博客：http：//googleblog.blogspot.com/2008/07/hitting-40-languages.html。

[8] Smith，R。，“Tesseract OCR引擎概述” Proc 9th Int。
Conf。文档分析与识别研究，2007，pp629-633。

[9] Tesseract开源OCR：http：//code.google.com/p/tesseract-ocr。

[10] Smith，R“通过Tab-Stop进行混合页面布局分析
检测，文档分析和识别”程序。第十名
诠释Conf。文件分析与识别，2009年。

[11] Smith，R.，“一种简单有效的偏斜检测算法
通过文本行累加”过程。第三国际Conf。文件分析与识别》，1995年，第1145-1148页。

[12] R. Unnikrishnan，R。Smith，“组合脚本和页面
使用Tesseract OCR引擎进行方位估计”，提交给多语言OCR国际研讨会，2009年7月25日，西班牙巴塞罗那。

[13] Gionis，A.，Indyk，P.，Motwani，R。，“在
通过散列实现高尺寸” Proc。
25日Conf。在VeryLarge数据库中，1999，pp518-529。

[14] Baluja，S.，Covell，M.，“学习哈希：宽恕哈希
功能和应用”，数据挖掘和知识发现17（3），2008年12月，pp402-430。

[15] R.E. Schapire，“弱学习性的力量”
机器学习，1990年5月，第197-227页