使Tesseract开源OCR引擎适应于多语言OCR

Adapting the Tesseract Open Source OCR Engine for Multilingual OCR

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

Ray Smith　　Daria Antonova　　Dar-Shyang Lee

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

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

Abstract

本文描述了使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%。

We describe efforts to adapt the Tesseract open source OCR engine for multiple scripts and languages. Effort has been concentrated on enabling generic language support so that negligible customization is required for a new language beyond providing a corpus of text. Although change was required to the page layout analysis module, and a few limitations of the linguistic post-processing, no change was required to the character classifier beyond changing a few limits. The Tesseract classifier has adapted easily to Simplified Chinese. Testing on a random sample of books, shows a reasonably consistent word error rate between 3.72% and 5.78%, and Simplified Chinese has a character error rate of only 3.77%.

关键字Tesseract，多语言OCR。

Keywords

Tesseract, Multi-Lingual OCR.

1. Introduction

简介：一些基于拉丁语的OCR的研究兴趣比十年前减少了，转而支持中文，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。的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。印地语将多个字母组合成代表音节的数千种形状。当字母结合在一起时，它们形成缀线，其形状仅与原始字母模糊不清。Hindi then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。

Research interest in Latin-based OCR fades away more than a decade ago, in favor of Chinese, Japanese, and Korean (CJK) [1,2], followed more recently by Arabic [3,4], and then Hindi [5,6]. These languages provide extra challenges, specifically to the page layout analysis, segmentation, character classifiers, and also to the other components of OCR systems. Chinese and Japanese share the Han script, which contains thousands of different characters. Korean uses Hangul script, which has several thousand more of its own, as well as using Han characters. The number of characters is one or two orders of magnitude greater than Latin alphabets. Arabic is written with connected characters, and its character shape according to the position in a word. Hindi combines a small number of basic symbols into thousands of shapes, that represent syllables. As the letters combine, they form ligatures whose shape only vaguely resembles the original letters. Hindi then combines the problems of CJK and Arabic, by joining all the symbols in a word with a line called the shiro-reka.

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，汉语和印地文的大字符集中要求的数量少得多的组件组成，汉族中的部首，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

Hindi takes this a step further by combining all the symbols in a word with a line called the shiro-reka. Arabic is written with connected characters, and its character shape according to the position in a word. Hindi combines a small number of basic symbols into thousands of shapes, that represent syllables. As the letters combine, they form ligatures whose shape only vaguely resembles the original letters. Hindi then combines the problems of CJK and Arabic, by joining all the symbols in a word with a line called the shiro-reka.

The research methods use language specific solutions that avoid the problems in some way, since that is simpler than trying to find a solution that works for all languages. For instance, the large character sets of Han, Hangul, and Hindi are mostly made up of a much smaller number of components, known as radicals in Han, Jamo in Hangul, and letters in Hindi. Since it is much easier to develop a classifier for a small number of classes, one approach is to recognize these components [1,2,5] and infer the actual characters from the combination of radicals. This approach is easier for Hangul than for Han or Hindi, since the

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Han，the radicals often are squashed to fit in the character and mostly touch other radicals. Hindi takes this a step further by combining all the symbols in a word with a line called the shiro-reka. Arabic is written with connected characters, and its character shape according to the position in a word. Hindi combines a small number of basic symbols into thousands of shapes, that represent syllables. As the letters combine, they form ligatures whose shape only vaguely resembles the original letters. Hindi then combines the problems of CJK and Arabic, by joining all the symbols in a word with a line called the shiro-reka.

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]。Model that models the character boundaries [3].

Google致力于提供尽可能多的语言版本[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结果。我们的方法是使用语言通用的方法，以尽量减少覆盖多种语言的人工工作。

Google has dedicated to making it available in as many languages as possible [7], so we are also interested in adapting the Tesseract Open Source OCR Engine [8, 9] to many languages. This paper discusses our efforts so far in fully internationalizing Tesseract, and some surprising results of this effort. Our approach is use language generic methods, to minimize the manual effort to cover many languages.

2. Review Of Tesseract For Latin

拉丁文Tesseract的评论

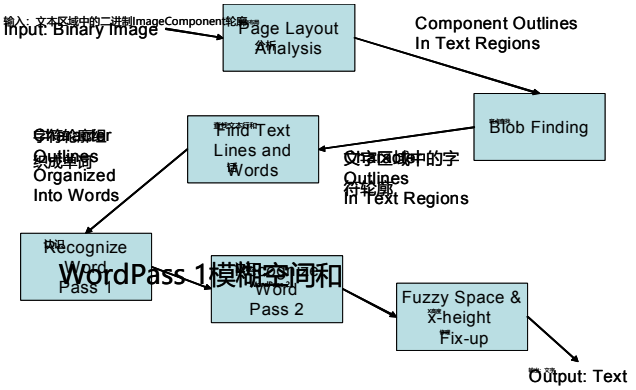

图1 Tesseract的顶级框图。

Figure 1. Top level block diagram of Tesseract.

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，特别是黑白文本之后，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）一般连接组件（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

Figure 1 is a block diagram of the basic components of Tesseract. The new page layout analysis [10] was designed from the beginning to be language-independent, but the rest of the engine was developed for English, without a great deal of thought as to how it might apply to other languages. Looking at the commercial engines at the time were strictly for black-on-white text, one of the original design goals of Tesseract was that it should recognize white-on-black (inverse video) text as easily as black-on-white. This led to the design (fortuitously as it turned out) in the direction of connected component (CC) analysis and operating on outlines of the components. The first step after CC

©ACM, 2009。这是作品的作者版本，经ACM许可将其张贴在此处供您个人使用，不进行重新分配。最终版本发布于2009年7月25日在西班牙巴塞罗那举行的2009年多语言OCR国际研讨会论文集上。http://doi.acm.org/10.1145/1577802.1577804。

©ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10/1145/1577802.1577804.

分析是在文本区域中找到斑点，斑点是指定的可分类单位，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。A problem is detecting inverse text inside a box, i.e. the holes inside a character. For English, there are very few characters (maybe © and ®) that have more than 2 levels of outline, and it is very rare to have more than 2 holes, so any blob that breaks these rules is a character, or is part of a character that is broken into multiple CCs. 反字符的框，甚至是黑色字符周围框架的内部或外部。the inside or outside of a frame around black-on-white characters.

在确定哪些轮廓组成斑点后，文本行查找器[11]检测文本行。Tesseract [11] detects (horizontal only) text lines by virtue of the vertical overlap of adjacent characters on a text line. For English the text finder is now performing very well, and can handle skew very precisely to a very large angle. After finding the text lines, a fixed-pitch detector checks for fixed pitch character layout, and if found, marks the lines for the word segmenter to use algorithms according to the fixed pitch decision. The bulk of the recognition process operates on each word independently, in which uncertain spaces are decided.

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，最好的结果被传递到自适应分类器。然后将结果与字典进行比较，并将字典与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，具有最佳的整体评级的字符串。对于English version，大多数标点符号规则都是硬编码的。

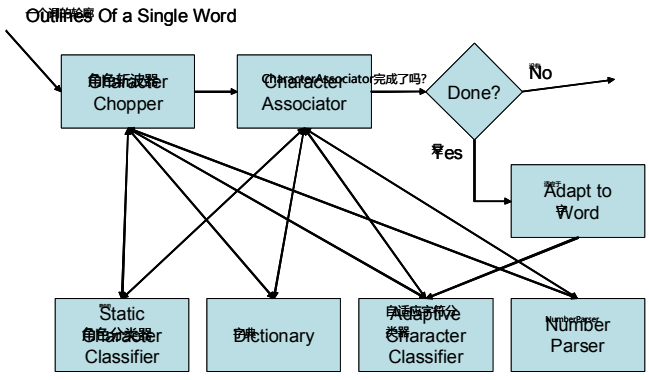

Figure 2. Block diagram of Tesseract word recognition.

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。As soon as the adaptive classifier has sufficient samples, it can contribute to the recognition in the second pass. On the second pass, words that were not good enough on pass 1 are processed for a second time, in case the adaptive classifier has gained more information since the first pass over the word.

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字符串是更简的整数代码代替？最初，我们将Tesseract适应于UTF-8，并将字符代码更改为UTF-8字符串，因为这是最灵活的，但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. Layout Preprocessing

Several aspects of the "textord" (text ordering) module of Tesseract required changes to make it more language-independent. This section discusses these changes.

3.1 Vertical Text Layout

Chinese, Japanese, and Korean (CJK) can have text lines either horizontally or vertically, and often mix directions on a single page. This problem is not unique to CJK, as English text lines are also sometimes vertical, for example in column headings. Vertical text is detected by the page layout analysis. If a majority of CCs on a page have their top-left coordinate to the left of their right side on a right tab, then everything between the tab-stops could be a line of vertical text. To prevent false-positives in tables, the CCs in vertical text must have a reasonable vertical gap between CCs to be less than the mean width of the CCs. If the majority of CCs on a page are vertically aligned, the page is rotated 90 degrees and the layout analysis is run again. The minority originally horizontal text will then become vertical text, and is rotated back before word recognition. The majority of the text will be horizontal.

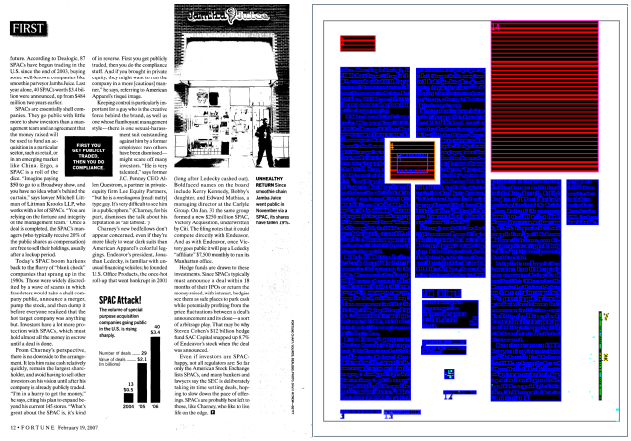

Figure 3. (a) A page containing a verical text region.
(b) The detected regions with image in red, horizontal text in blue, and vertical text in yellow.

©ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10/1145/1577802.1577804.

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上。

Tesseract在特征的整数坐标空间中的CC，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，解决方案只是通过旋转页面中的垂直和水平文本块，并旋转每一个字符，来简单地区分出垂直和水平文本行，以满足分类所需的字符。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4中检测到的文本块。在图4中，垂直文本区域旋转了90度（图中图像的右下角），因此它看起来与原始图像一样，但处于水平方向。

Tesseract originally was designed with the assumption that it would not be necessary to deal with vertical text, and there are a lot of places in the code where some assumption is made over characters being arranged on a horizontal text line. Fortunately, Tesseract operates on connected components in a signed integer coordinates space, which makes rotations by multiples of 90 degrees trivial, and it doesn't care whether coordinates are positive or negative. The solution is therefore simply to differentially rotate the vertical and horizontal text blocks on a page and rotate the characters as needed for classification. Fig. 3 shows an example for English text. The page in Fig. 3(a) contains vertical text at the lower-right, which is detected as a text block, so in Fig. 4 the vertical text region is rotated 90 degrees clockwise (the text in the bottom-left of the image), so appears well below the original image, but in horizontal orientation.

Figure 4. The vertical text is differentially rotated so it is oriented horizontally.

图4 垂直文本是不同的旋转以使其水平定向。

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。

Figure 5 shows an example for Chinese text. The vertical body text is rotated out of the image, to make it horizontal, and the header, which was originally horizontal, stays where it started. Vertical and horizontal text lines exist in different blocks, but Tesseract cares only that the text lines are horizontal. The data structure for a text block records the rotations that have been performed on it, so that the reverse rotation can be applied to the characters as they are passed to the classifier, to make them upright. Automatic orientation detection [12] can be used to ensure that the text is upright when passed to the classifier, as vertical text could have characters that are in at least 3 different orientations relative to the reading direction. After Tesseract processes the rotated text blocks, the coordinate space is re-rotated back to the original image orientation so that reported character bounding boxes are still accurate.

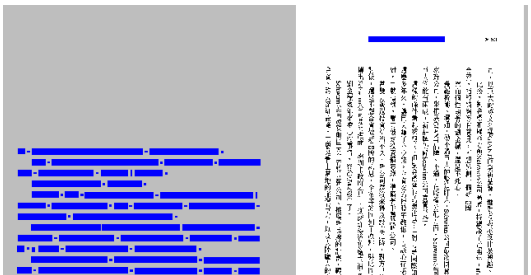

Figure 5. Horizontal text detection for Traditional Chinese. Since the majority of the text is vertical, inside Tesseract it is rotated anticlockwise 90 degrees so it lies outside the image, but the lines are horizontal. The page header, which was already horizontal, remains behind.

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的，仍然落后。

3.2 Text-line and Word Finding

Tesseract's original text-line finder [11] assumed that CCs of characters mostly vertically overlap the bulk of the text line. The one real exception is i dots. For general languages this is also true, including many languages that have diacritics, which are placed above and/or below the bulk of the text-line. For that, for example, the distance from the body of the text line to the diacritics can be quite extreme. The page layout analysis for Tesseract is designed to simply locate text lines, and then to subdivide text regions into blocks of uniform text size and line spacing. This makes it possible to force-fit a line-spacing model, so the text-line finding does not have to deal with skew. Tesseract's page layout analysis also estimates the residual skew of the text regions, which means the text-line finder no longer has to be insensitive to skew.

Tesseract原始的文本行查找算法[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主体上方和/或下方，例如带有变音符号的语言。例如，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偏斜，这意味着文本行查找器不再必须对偏斜不敏感。

修改后的文本行查找算法可以在用分析中为每个文本区域独立工作，并从搜索较小CC的邻域（比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。

The original Tesseract text line finder [11] assumed that CCs of characters mostly vertically overlap the bulk of the text line. The one real exception is i dots. For general languages this is also true, including many languages that have diacritics, which are placed above and/or below the bulk of the text-line. For that, for example, the distance from the body of the text line to the diacritics can be quite extreme. The page layout analysis for Tesseract is designed to simply locate text lines, and then to subdivide text regions into blocks of uniform text size and line spacing. This makes it possible to force-fit a line-spacing model, so the text-line finding does not have to deal with skew. Tesseract's page layout analysis also estimates the residual skew of the text regions, which means the text-line finder no longer has to be insensitive to skew.

The modified text line finding algorithm can work independently in each text region from layout analysis, and begins by searching the neighborhood of small CCs (relative to the estimated text size) to find the nearest body-text-sized CC. If there is no nearby body-text-sized CC, then a small CC is regarded as likely noise and discarded. (An exception has to be made for dotted/dashed leaders, as typically found in a table of contents.) Otherwise, a bounding box containing the enclosed CC and its larger neighbor is constructed and used in place of the bounding box of the small CC in the following projection.

使用小型CC的修改后的框，对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。成本函数是配置文件条目在切点加上它们之间间距变化的度量的总和。对于大多数文本，配置文件条目的总和为零，并且方差为零，并且只有少数切点非零，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。

Using the modified boxes for small CCs, a dynamic programming algorithm selects the best set of partitions of the projection profile. The cost function is the sum of profile entries at the cut points plus a measure of the variance of the spacing between them. The sum is zero for perfectly fitting text, and the variance helps to choose the most regular line spacing. For more complex situations, the variance and the modified bounding boxes for small CCs combine to help discriminate the best cuts to maximize the number of diacriticals that stay with their appropriate body characters.

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。

Once cut lines are determined, the whole connected components are placed in the text line that they vertically overlap the most, (still using the modified boxes) except where a component strongly overlaps multiple lines, as CCs are assumed to be either characters from multiple lines that touch, and so need cutting at the cut line, or drop-caps, in which case they are placed in the top overlapped line. This algorithm works well, even for Arabic.

提取文本行后，将行上的斑点组织成识别单元。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母的语言模型。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，鉴于汉字由多个字形（基本组成部分）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能地检测到单词。虽然这并不完全

After text lines are found, they are processed into recognition units. For Latin languages, the logical recognition units correspond to space-delimited words, which is naturally suited to alphabetic script models. For languages that are not space-delimited, such as Chinese, it is less clear what the corresponding recognition unit should be. One possibility is to treat each symbol as a word. However, given that Chinese symbols are composed of multiple glyphs (radicals), it would be difficult to get the correct character segmentation without recognition. Given the limited amount of information that is available at this early stage of processing, the solution is to break up the blob sequence at punctuations, based on their size and spacing to the next blob. Although this does not completely

©ACM, 2009年。这是作品的作者版本，经ACM许可将其张贴在此处供您个人使用，不进行重新分配。最终版本发布于2009年7月25日在西班牙巴塞罗那举行的2009年多语言OCR国际研讨会论文集上。http://doi.acm.org/10/1145/1577802.1577804。

©ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10/1145/1577802.1577804.

解决了很长的词点序列问题。这是确定分割图的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、倾面的黑色区域或侧边或反向视频区域中的大型容器。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裹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 Estimating x-height in Cyrillic Text

在完成文本行查找和单词查找步骤后，Tesseract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的，Tesseract无法为大多数线找到正确的x高度。结果，在Russian books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混淆率高。例如，在33种大写俄语字母中，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，当使用西文字体训练Tesseract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

4. CHARACTER WORD RECOGNITION

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，(4) 识别出以上都不是的线（即最有可能包含噪声的斑点过小、过大或尺寸不一致的线）的估计值。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。如果该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，上面在一词

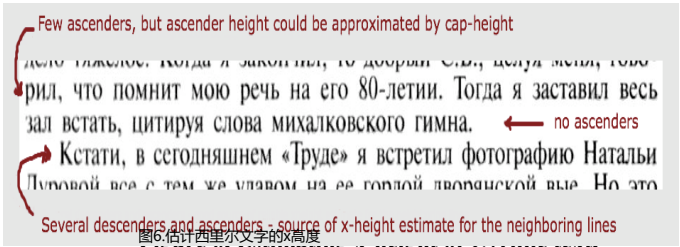

图6.估计西里尔文字的x高度

套俄文书籍的测试中降到了7.6%。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 Character / Word Recognition

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，这对针对从几个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是符号分隔的短语。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在分割图上采用最佳优先搜索策略，按Blob序列的长度呈指数增长。

©ACM, 2009年。这是作品的作者版本。经ACM许可将其张贴在此处供您个人使用。不进行重新分配。最终版本发布于2009年7月25日在西班牙巴塞罗那举行的2009年多语言OCR国际研讨会论文集上。http://doi.acm.org/10.1145/1577802.1577804。

©ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10/1145/1577802.1577804.

segmentation points and a termination condition when the result is found in the dictionary, it often exhausts available resources when classifying a Chinese phrase. To resolve this issue, we need to dramatically reduce the number of segmentation points evaluated in the permutation and devise a termination condition that is easier to meet.

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。

In order to reduce the number of segmentation points, we incorporate the constraint of roughly constant character widths in a mono-spaced language like Chinese and Japanese. In these languages, characters mostly have similar aspect ratios, and are either full-pitch or half-pitch in their positioning. Although the normalized width distribution would vary across fonts, and the spacing would shift due to line justification and inclusion of digits or Latin words, which is not uncommon, by and large these constraints provide a strong guideline for whether a particular segmentation point is compatible with another. Therefore, using the deviation from the segmentation model as a cost, we can eliminate a lot of implausible segmentation states and effectively reduce the search space. We also use this estimate to prune the search space based on the best partial solution, making it effectively a beam search. This also provides a termination condition when no further expansion is likely to produce a better solution.

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。

Another powerful constraint is the consistency of character script within a phrase. As we include shape classes from multiple scripts, confusion errors between characters across different scripts become inevitable. Although we can establish the dominant script or language for the page, we must allow for Latin characters as well, since the occurrence of English words inside foreign language books is so common. Under the assumption that characters within a recognition unit would have the same script, we would promote a character interpretation if it improves the overall script consistency of the whole unit. However, blindly promoting script characters based on prior could actually hurt the performance if the word or phrase is truly mixed script. So we apply the constraint only if over half the characters in the top interpretation belong to the same script, and the adjustment is weighted against the shape recognition score, like any other permutation.

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 Shape Classification

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。

Classifiers for large numbers of classes are still a research problem; even today, especially when they are required to operate at the speeds needed for OCR [13, 14]. The curse of dimensionality is largely to blame. The Tesseract shape classifier works surprisingly well on 5000 Chinese characters without requiring any major modifications, so it seems to be well suited to large class-size problems. This result deserves some explanation, so in this section we describe the Tesseract shape classifier.

The features are components of a polygonal approximation of the outline of a shape. In training, a 4-dimensional feature vector of (x, y-position, direction, length) is derived from each element of the polygonal approximation, and clustered to form prototypical feature vectors. (Hence the name: Tesseract.) In recognition, the elements of the polygon are broken into shorter pieces of equal length, so that the length dimension is eliminated from the feature vector. Multiple short features are matched against each prototypical feature, making the classification process more robust against broken characters.

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。

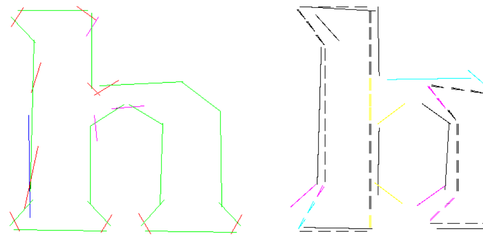

Figure 7. (a) Prototype of h for Times Roman, (b) Match of a broken h against prototytype.

图7.（a）Times Roman的h原型，（b）残缺的h与原型的匹配。

Fig. 7(a) shows an example prototype of the letter ‘h’ for the font Times Roman. The green line-segments represent cluster means of *significant clusters* that contain samples from almost every sample of ‘h’ in Times Roman. Blue segments are cluster means that were merged with another cluster to form a significant cluster. Magenta segments were not used, as they matched an existing significant cluster. Red segments did not contain enough samples to be significant, and could not be merged with any neighboring cluster to form a significant cluster.

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。

Fig. 7(b) shows how the shorter features of the unknown match against the prototype to achieve insensitivity to broken characters. The short, thick lines are the features of the unknown, being a broken ‘h’ and the longer lines are the prototype features. Colors represent match quality: black -> good, magenta -> reasonable, cyan -> poor, and yellow -> no match. The vertical prototypes are all well matched, despite the fact that the h is broken.

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。

The shape classifier operates in two stages. The first stage, called the *class pruner*, reduces the character set to a short-list of 1-10 characters, using a method closely related to Locality Sensitive Hashing (LSH) [13]. The final stage computes the distance of the unknown from the prototypes of the characters in the short-list.

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LocalitySensitiveHashing（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。

Originally designed as a simple and vital time-saving optimization, the class pruner partitions the high-dimensional feature space, by considering each 3-D feature individually. In place of the hash table of LSH, there is a simple look-up table, which returns a vector of integers in the range [0, 3], one for each class in the character set, with the value representing the approximate goodness of match of that feature to a prototype of that character class. The vector results are summed across all features of the unknown, and the classes that have a total score within a fraction of the highest are returned as the shortlist to be classified by the second stage. The class pruner is relatively fast, but its time scales linearly with the number of classes and also with the number of features.

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。

The second stage classifier calculates the distance d_f of each feature from its nearest prototype, as the squared Euclidean distance d of the (x,y) feature coordinates from the prototype line in 2-D space, plus a weighted (w) difference of the angle θ from the prototype:

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）着原型：

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

©ACM, 2009. *This is the authors' version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10/1145/1577802.1577804.*

©ACM，2009年。这是作品的作者版本。经ACM许可将其张贴在此处供您个人使用。不进行重新分配。最终版本发布于2009年7月25日在西班牙巴塞罗那举行的2009年多语言OCR国际研讨会论文集上。http://doi.acm.org/10/1145/1577802.1577804。

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E_f：

$$E_f = \frac{1}{1 + kd_f^2}$$

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E_f被复制到原型E_p。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L_p，结果转换回距离：

$$d = k\left(1 - \frac{\sum_f E_f + \sum_p E_p}{N_f + \sum_p L_p}\right)$$

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整数缩放常数，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。这是维基百科[15]背后的概念。只是当前的弱分类器尚未加权。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. Contextual Post-Processing

Tesseract's training process supports partially extending the language model by providing a way to generate dictionaries for new languages from training data [10] compatible with fast search. these dictionaries are represented by directed acyclic word graphs (DAWGs). In the original implementation the following DAWG dictionaries, including the pre-generated system dictionary, document dictionary (dynamically constructed from the words in the OCRed document) and a user-provided word list.

Originally each edge in the DAWG stored a unichar ID similar to the letter used in the corresponding transition in the DAWG. This representation, however, was limiting, since it made handling languages with many characters awkward. To overcome this limitation, the DAWG data structure was modified to store the *unicharset IDs* used by the character classifier instead. This significantly simplified the

5.1 Constraint Patterns

The punctuation and number state machines in Tesseract were hard-coded and did not generalize beyond the Latin scripts. Even for the Latin scripts, a significant portion of valid punctuation and number patterns were not accepted by the state machines. To help Tesseract handle punctuation and numbers in non-Latin scripts Tesseract's training was enhanced to collect and encode a set of frequently occurring punctuation and number patterns. The step for collecting these patterns was implemented to run in parallel, making use of multiple CPUs and the existing code for generating and searching the dictionaries. To represent and match the generated patterns, the already existing code for dictionaries was reused. To this end, the DAWG was considered to be completely data-driven and so far requires no special casing for any language in particular.

5.2 Resolving Shape Ambiguities

Tesseract's shape classifier includes an adaptive component that learns the patterns of the characters seen in the OCRed document in order to ensure that it adapts to the unambiguous dictionary words. The OCRed word is "dubbed" unambiguous if it satisfies two constraints: First, the margin by which the best choice beats the next choice of a different word must be sufficiently large. Second, every character in the word must be sufficiently close to the best choice for that character. These constraints ensure that no dictionary word can exist that is ambiguous in shape to the best choice for the word. This requirement is also important for recognition speed, since (depending on the classifier) once Tesseract finds such a word choice, it could accept the recognition result and stop further processing of the word.

For Latin script, Tesseract contains a built-in table (referred to as "dangerous ambiguities" file), specifying which letter combinations are inherently ambiguous in the majority of the Latin fonts. A scalable solution to enable this functionality for

©ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10/1145/1577802.1577804.

language. A set of n-grams (in this case uni- and bi-grams) whose combined weight accounts for 95% of all n-grams in the language was collected from a large text corpus. The n-grams were rendered with a set of commonly used fonts in four degradation modes and exposures. Then Tesseract's shape classifier was run on the rendered images to obtain a set of top scoring classifications for each of the n-grams. The resulting classification scores statistics was aggregated for each of the n-grams and the outliers with low classifier scores were discarded (in some fonts and degradation modes the characters were rendered beyond recognizable, and such cases would only pollute the data). Then for each of the incorrectly OCRed n-grams and the corresponding correct n-gram pair an ambiguity score was computed. The ambiguity score was defined as a function of the shape classifier's perceived similarity between the wrong and correct n-grams (aggregated across all fonts and degradation modes) and the frequency of the correct n-gram in the language. To achieve a balance between Tesseract's speed and accuracy, it was necessary to pick a threshold of the expected number of errors allowed to occur due to ambiguity (computed from the n-gram frequency and classifier error statistics). To generate the "dangerous ambiguities" file, the n-gram ambiguity pairs were sorted in the non-increasing order of their ambiguity scores and the appropriate number of top-scoring ambiguities that ensured the desired expected error rate were included in the file.

Using this automatic method of generating the file, it was possible to achieve similar improvements on Latin scripts (EFIGS data set) as compared to using the hand-crafted "dangerous ambiguities" file (although the results for some languages were slightly weaker). Using the automatically-generated file on the Russian data set resulted in a 10% reduction in word error rate. Examining the files generated for other languages also showed that the automatically generated files contained a fair number of commonly confused shapes, but further tests on the corresponding data sets will be needed to quantify the improvement.

5.3 Handling Highly Inflected Languages

Tesseract's speed and accuracy are strongly related to the quality of the dictionary, and it is always a challenge to maximize these, while minimizing the space consumed to store the dictionary. Obtaining a good coverage of the language from a limited corpus of a highly inflected language is a particularly difficult task. The frequency of words in highly inflected languages is more evenly distributed, and thus to achieve the same language coverage a larger dictionary is needed. Moreover, many of the word forms of even the more frequent words might not occur enough times in the training corpus to be included in the dictionary, and thus the dictionary might not generalize well beyond the training corpus. Fig 8 illustrates this problem on a collection of languages by graphing the coverage of the corpus against the number of most frequent words chosen to form the dictionary.

Because of the head and tail compaction of the DAWG data structure, adding an inflected form of a word that already exists in the DAWGs might result in a very small increase in the overall size of the dictionary. This is because the beginning and the ending of the word might already be stored in the DAWG (for example it would be relatively cheap to add the word "talking" to the dictionary if "talk" and "making" have already been inserted).

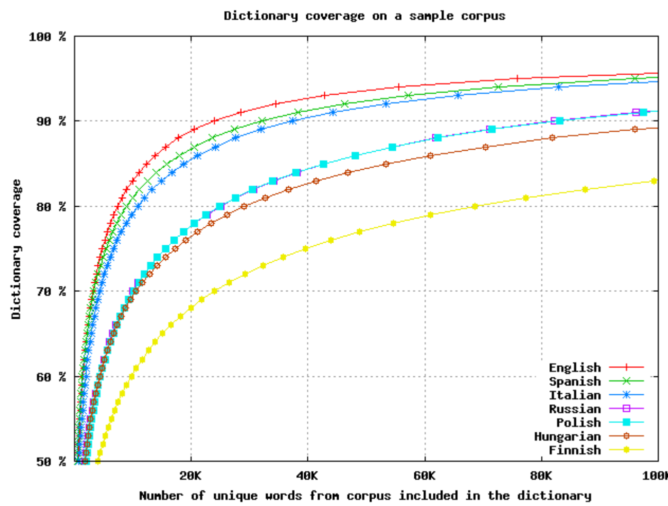

Figure 8. Corpus coverage of varying-size dictionaries in a collection of languages.

To combat the problem of capturing more inflected word forms, the dictionary generation process was amended with a step to generate word variants (that were not present in the word list) and add them to the dictionary. First, as previously done, the DAWG is constructed from a word list. Then for each word root in the DAWG a set of suffixes is collected. The sets are clustered using a group-average hierarchical agglomerative clustering algorithm. The suffix sets in the resulting clusters are merged to form expanded suffix sets. Then for each word root and the corresponding suffix set (pre-computed during the first traversal of the DAWG) the closest expanded suffix set is identified. The new words formed by the suffixes from the expanded set are inserted into the DAWG.

6. Status / Experimental Results

Our test data set consists of pages from randomly selected books collected by the Google Book Search project. For each language, 100 random books were selected, and 10 pages were randomly selected from each book for manual ground truthing. Therefore, these pages cover a large variety in every aspect from layout, typeface, image quality, to subject and term usage.

The data set is then broken into training, validation and test sets, where the training and validation sets are used for learning and benchmarking the algorithms during development, and the test set is reserved for final evaluation. Table 1 summarizes the size of the data set and current accuracy for a few languages. For alphabetical languages, we report the error rates at both the character and word level. For Chinese, where the meaning of word is ambiguous, we report only the character substitution rate. EFIGSD is a combination of English, French, Italian, German, Spanish and Dutch.

For Simplified Chinese, we noticed the large deviation of error rates across different books. The difference can be mainly attributed to variation in fonts and quality. Where the page quality and accuracy are reasonable, the errors are mainly due to confusions between similar or near-identical shape classes. We have plans to increase the capacity of the feature space in the shape matcher, which should help distinguish between similar

©ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10/1145/1577802.1577804.

形状。在几乎相同的情况下，跨字体的类内变体可能大于跨间变体。更重要的是，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。

shapes. Of almost identical cases, font variation can be greater than inter-class variation. Fortunately, their usage and priors are so different that they could easily be corrected when we introduce a language model for CJK.

Table 1. Error rates over various languages

语言数量	No. of chars (millions)	字数 (百万)	Char error rate (%)	Word error rate (%)
English	39	4	0.5	3.72
EFIGS	213	26	0.75	5.78
Russian	38	5	1.35	5.48
Simplified Chinese	0.25	NA	3.77	NA
Hindi	1.4	0.33	15.41	69.44

7. Conclusions & Future Work

结论和未来的工作我们已经测试了使用Tesseract适用于多种语言的实验，并已经证明这样做的主要是工程问题，没有任何对当前架构进行重大更改，我们能够获得多种基于拉丁语的语言结果，包括各种口音变体，以及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许多旧字体的混合，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。这项工作当然涵盖从右到左书写的语言，这本身是另一个工程问题，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，它带来了高度模糊的字符问题，并且与中文一样，单词之间没有空格。

We have now tested the feasibility of using Tesseract for multiple languages, and found that it was surprisingly mostly a matter of engineering. Without any changes of substance to the current architecture, we are able to obtain results for a variety of Latin-based languages, Russian, and even Simplified Chinese. The results for Hindi have so far been disappointing, but we have discovered that our test set contains a mix of new and old typography, and a significant proportion of errors are due to the fact that the training set does not contain characters from the old typography. This work does not cover languages that are written from right to left, which is mainly another engineering issue, but Arabic has its own set of problems that Tesseract may not be able to address, namely character segmentation. Another language that we have not discussed is Thai, which poses problems of highly ambiguous characters, and like Chinese, does not have spaces between words.

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。只管理复合功能会增加大小写更改插入中的复杂性，但也很有用。

An important future project is to improve the training process to be able to use real data for training instead of just synthetic data with character bounding boxes. This will greatly help accuracy on Hindi. We also need to test Arabic and Thai, where we anticipate more problems. For Chinese, Japanese, and Thai, we need to allow the language model to search the space of arbitrarily concatenated words, since there is no whitespace between the words. Just managing compound functions would increase the complexity of case changes and inserted letters, but it is also useful for German, although German compounding has the additional complexity of case changes and inserted letters.

8. References

[1] Nagy, G., "Chinese character recognition: thirty-five-year perspective" *9th Int. Conf. on Pattern Recognition*, Nov 1988, pp163-167.

[2] Xia, F. "Knowledge-based substroke segmentation of Chinese characters" *Image Processing 1994* Proc. ICIP-94, IEEE Int. Conf. vol.1, 13-16 Nov 1994, pp179-182.

[3] Schwartz, R., Nartey, P., Abi-Ezzi, J., Matsoukas, S., "Advances in the BBN BYBLOS OCR system" *Proc. 5th Int. Conf on Document Analysis and Recognition*, 1999, pp337-340.

[4] Kanungo, T., Marton, G.A., Bulbul, O., "Omnipage vs. Sakhr: paired Model Evaluation of Two Arabic OCR Products" *Proc. SPIE* **3651,** 7 Jan 1999, pp109-120.

[5] Bansal, V., Sinha, R.M.K., "A complete OCR for printed Hindi text in Devanagari script" *Proc. 6th Int. Conf on Document Analysis and Recognition*, 2001, pp800-804.

[6] Govindaraju, V., et al. "Tools for Creating Digital Access to multi-lingual Indic documents" *Proc. 1st Int. Workshop on document Image Analysis for Libraries*, 2004, pp122-133.

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

[8] Smith, R., "An Overview of the Tesseract OCR Engine" *Proc. 9th Int. Conf on Document Analysis and Recognition,* 2007, pp629-633.

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

[10] Smith, R. "Hybrid Page Layout Analysis via Tab-Stop Detection, Document Analysis and Recognition" *Proc. 10th Int. Conf. on Document Analysis and Recognition,* 2009.

[11] Smith, R., "A simple and efficient skew detection algorithm via text row accumulation" *Proc. 3rd Int. Conf. on Document Analysis and Recognition*, 1995, pp1145-1148.

[12] Unnikrishnan, R., Smith, R., "Combined Script and Page Orientation Estimation using the Tesseract OCR Engine" *Submitted to International Workshop on* Multilingual OCR, 25th July 2009, Barcelona, Spain.

[13] Gionis, A., Indyk, P., Motwani, R., "Similarity Search in High Dimensions via Hashing" *Proc. 25th Int. Conf. on Very Large Data Bases*, 1999, pp518-529.

[14] Baluja, S., Covell, M., "Learning to hash: forgiving hash functions and applications" *Data Mining and Knowledge Discovery* **17**(3), Dec 2008, pp402-430.

[15] Schapire, R.E., "The Strength of Weak Learnability" *Machine Learning,* **5**, 1990, pp 197-227.

©ACM, 2009年。这是作品的作者版本，经ACM许可将其张贴在此处供您个人使用，不进行重新分配。最终版本发布于2009年7月25日在西班牙巴塞罗那举行的2009年多语言OCR国际研讨会论文集上。http://doi.acm.org/10.1145/1577802.1577804。

©ACM, 2009. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the International Workshop on Multilingual OCR 2009, Barcelona, Spain July 25, 2009. http://doi.acm.org/10.1145/1577802.1577804.