An Overview of the Tesseract OCR Engine
Tesseract OCRエンジンの概要

Ray Smith
レイ・スミス
Google Inc.
グーグル株式会社
theraysmith@gmail.com

Abstract
要約

The Tesseract OCR engine, as was the HP Research Prototype in the UNLV Fourth Annual Test of OCR Accuracy[1], is described in a comprehensive overview. Emphasis is placed on aspects that are novel or at least unusual in an OCR engine, including in particular the line finding, features/classification methods, and the adaptive classifier.

ネバダ大学ラスベガス校での第4回OCR正確さテスト[1]の当時はHP研究所のプロトタイプであったTesseract OCRエンジンは、包括的な概要で説明されます。特にライン発見、特徴/分類方法、および適応分類器を含む、OCRエンジンにおいて新規または少なくとも異例である局面に重点が置かれている。

1. Introduction – Motivation and History
1. はじめに － 動機と歴史

Tesseract is an open-source OCR engine that was developed at HP between 1984 and 1994. Like a supernova, it appeared from nowhere for the 1995 UNLV Annual Test of OCR Accuracy [1], shone brightly with its results, and then vanished back under the same cloak of secrecy under which it had been developed. Now for the first time, details of the architecture and algorithms can be revealed.

Tesseractは、1984年から1994年の間にHPで開発されたオープンソースのOCRエンジンです。それはそれが開発されたのと同じ秘密の隠蔽の下に戻ってきた。今回初めて、アーキテクチャとアルゴリズムの詳細を明らかにすることができます。

Tesseract began as a PhD research project [2] in HP Labs, Bristol, and gained momentum as a possible software and/or hardware add-on for HP's line of flatbed scanners. Motivation was provided by the fact that the commercial OCR engines of the day were in their infancy, and failed miserably on anything but the best quality print.

Tesseractは、ブリストルのHP Labsで博士課程の研究プロジェクト[2]として始まり、HPのフラットベッドスキャナのラインナップの可能なソフトウェアおよび/またはハードウェアアドオンとして勢いを増しました。動機は、その日の市販のOCRエンジンが初期の段階にあり、最高品質の印刷物以外のもので惨めに失敗したという事実によってもたらされました。

After a joint project between HP Labs Bristol, and HP's scanner division in Colorado, Tesseract had a significant lead in accuracy over the commercial engines, but did not become a product. The next stage of its development was back in HP Labs Bristol as an investigation of OCR for compression. Work concentrated more on improving rejection efficiency than on base-level accuracy. At the end of this project, at the end of 1994, development ceased entirely. The engine was sent to UNLV for the 1995 Annual Test of OCR Accuracy[1], where it proved its worth against the commercial engines of the time. In late 2005, HP released Tesseract for open source. It is now available at http://code.google.com/p/tesseract-ocr.

HP Labs Bristolとコロラド州にあるHPのスキャナー部門との共同プロジェクトの後、Tesseractは商用エンジンよりも正確性において大きなリードを築いたが、製品にはならなかった。その開発の次の段階は、圧縮のためのOCRの調査としてHP Labs Bristolに戻りました。作業は、基本レベルの精度よりも拒否効率の改善に集中しました。このプロジェクトの終わり、1994年の終わりには、開発は完全に中止されました。このエンジンは、1995年のOCR精度テスト[1]のためにUNLVに送られ、そこで当時の商用エンジンに対してその価値が証明されました。2005年末、HPはオープンソース向けにTesseractをリリースしました。現在http://code.google.com/p/tesseract-ocrから入手できます。

2. Architecture
2. アーキテクチャー

Since HP had independently-developed page layout analysis technology that was used in products, (and therefore not released for open-source) Tesseract never needed its own page layout analysis. Tesseract therefore assumes that its input is a binary image with optional polygonal text regions defined.

HPは、製品で使用されていた(したがってオープンソース用にはリリースされていなかった)独自に開発されたページレイアウト分析テクノロジを持っていたので、Tesseractは独自のページレイアウト分析を必要としませんでした。したがって、Tesseractはその入力がオプションの多角形テキスト領域が定義されたバイナリイメージであると仮定します。

Processing follows a traditional step-by-step pipeline, but some of the stages were unusual in their day, and possibly remain so even now. The first step is a connected component analysis in which outlines of the components are stored. This was a computationally expensive design decision at the time, but had a significant advantage: by inspection of the nesting of outlines, and the number of child and grandchild outlines, it is simple to detect inverse text and recognize it as easily as black-on-white text. Tesseract was probably the first OCR engine able to handle white-on-black text so trivially. At this stage, outlines are gathered together, purely by nesting, into *Blobs*.

処理は伝統的な段階的なパイプラインに従いますが、ステージのいくつかは当時は珍しく、おそらく今でもそう残っています。最初のステップは、コンポーネントの概要が格納されている連結コンポーネント分析です。これは当時は計算コストのかかる設計上の決定でしたが、大きな利点がありました。アウトラインのネスティング、子と孫のアウトラインの数を調べることで、反転テキストを検出してブラックオンと同じくらい簡単に認識できるからです。- 白いテキスト Tesseractはおそらく白黒のテキストをそれほど簡単に処理できた最初のOCRエンジンでした。この段階では、アウトラインは純粋に入れ子になってBlobにまとめられます。

Blobs are organized into text lines, and the lines and regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Fixed pitch text is chopped immediately by character cells. Proportional text is broken into words using definite spaces and fuzzy spaces.

ブロブはテキスト行に編成され、行と領域は固定ピッチまたはプロポーショナルテキストについて分析されます。テキスト行は、文字間隔の種類によって異なる方法で単語に分割されます。固定ピッチのテキストは、文字セルによってすぐに切り刻まれます。比例テキストは、一定のスペースとあいまいなスペースを使用して単語に分割されます。

Recognition then proceeds as a two-pass process. In the first pass, an attempt is made to recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down the page.

認識は2パスのプロセスとして進行します。最初のパスでは、各単語を順番に認識しようとします。満足のいく各単語は、訓練データとして適応分類器に渡される。そして、適応分類器は、ページの下にあるテキストをより正確に認識する機会を得ます。

Since the adaptive classifier may have learned something useful too late to make a contribution near the top of the page, a second pass is run over the page, in which words that were not recognized well enough are recognized again.

適応分類器は、ページの上部近くで貢献するには遅すぎる有用な何かを学んだ可能性があるので、十分に認識されなかった単語が再び認識される、2番目のパスがページ上で実行されます。

A final phase resolves fuzzy spaces, and checks alternative hypotheses for the x-height to locate smallcap text.

最後の段階では、あいまいなスペースを解決し、xの高さに関する対立仮説を調べて、小さな大文字のテキストを見つけます。

3. Line and Word Finding
3. 行と単語の検索

3.1. Line Finding
3.1. 行の発見

The line finding algorithm is one of the few parts of Tesseract that has previously been published [3]. The line finding algorithm is designed so that a skewed page can be recognized without having to de-skew, thus saving loss of image quality. The key parts of the process are blob filtering and line construction.

行の発見アルゴリズムは、以前に公開されたTesseractの数少ない部分の1つです[3]。直線検出アルゴリズムは、歪曲されていないページが歪曲を解消する必要なく認識されるように設計されているので、画像品質の損失を防ぐことができます。このプロセスの重要な部分は、ブロブフィルタリングとライン構築です。

Assuming that page layout analysis has already provided text regions of a roughly uniform text size, a simple percentile height filter removes drop-caps and vertically touching characters. The median height approximates the text size in the region, so it is safe to filter out blobs that are smaller than some fraction of the median height, being most likely punctuation, diacritical marks and noise.

ページレイアウト分析がすでにほぼ一様なテキストサイズのテキスト領域を提供していると仮定すると、単純な百分位数の高さフィルタは、ドロップキャップと垂直方向のタッチ文字を削除します。中央値の高さはその領域のテキストサイズに近いので、中央値の高さの一部よりも小さいBlob、つまり句読点、発音区別符号、およびノイズを除外するのが安全です。

The filtered blobs are more likely to fit a model of non-overlapping, parallel, but sloping lines. Sorting and processing the blobs by x-coordinate makes it possible to assign blobs to a unique text line, while tracking the slope across the page, with greatly reduced danger of assigning to an incorrect text line in the presence of skew. Once the filtered blobs have been assigned to lines, a least median of squares fit [4] is used to estimate the baselines, and the filtered-out blobs are fitted back into the appropriate lines.

フィルタ処理されたブロブは、重ならない、平行であるが傾斜した線のモデルに合う可能性が高い。ブロブをx座標でソートし処理することにより、斜面が存在する場合に誤ったテキスト行に割り当てる危険性を大幅に低減しながら、ページを横切る傾きを追跡しながら、ブロブを一意のテキスト行に割り当てることが可能になる。フィルタリングされたブロブがラインに割り当てられると、最小二乗平均平方根[4]がベースラインを推定するために使用され、フィルタリングされたブロブは適切なラインにフィッティングされます。

The final step of the line creation process merges blobs that overlap by at least half horizontally, putting diacritical marks together with the correct base and correctly associating parts of some broken characters.

行作成プロセスの最後のステップでは、重なり合うブロブを水平方向の半分以上マージして、発音区別符号を正しいベースと一緒に配置し、一部の破損した文字の一部を正しく関連付けます。

3.2. Baseline Fitting
3.2. ベースラインフィッティング

Once the text lines have been found, the baselines are fitted more precisely using a quadratic spline. This was another first for an OCR system, and enabled Tesseract to handle pages with curved baselines [5], which are a common artifact in scanning, and not just at book bindings.

テキスト行が見つかったら、2次スプラインを使用してベースラインをより正確に合わせます。これはOCRシステムにとってもう1つの最初の試みであり、Tesseractが湾曲したベースライン[5]を持つページを処理できるようにしました。湾曲したベースラインは、製本に限らず、スキャニングにおいて共通して現れる影響です。

The baselines are fitted by partitioning the blobs into groups with a reasonably continuous displacement for the original straight baseline. A quadratic spline is fitted to the most populous partition, (assumed to be the baseline) by a least squares fit. The quadratic spline has the advantage that this calculation is reasonably stable, but the disadvantage that discontinuities can arise when multiple spline segments are required. A more traditional cubic spline [6] might work better.

ベースラインは、ブロブを元の直線ベースラインに対して適度に連続的な変位でグループに分割することによってフィットされます。二次スプラインは、最小二乗フィットによって、最も人口の多いパーティション（ベースラインと見なされる）にフィットします。二次スプラインは、この計算が適度に安定しているという利点を有するが、複数のスプラインセグメントが必要とされるときに不連続性が生じ得るという欠点を有する。より伝統的な3次スプライン[6]がうまくいくかもしれません。

Fig. 1. An example of a curved fitted baseline.
図.1. 曲線で近似されたベースラインの例

Fig.1 shows an example of a line of text with a fitted baseline, descender line, meanline and ascender line. All these lines are “parallel” (the y separation is a constant over the entire length) and slightly curved. The ascender line is cyan (prints as light gray) and the black line above it is actually straight. Close inspection shows that the cyan/gray line is curved relative to the straight black line above it.

図1は、フィットされたベースライン、ディセンダライン、ミーンライン、およびアセンダラインを含むテキスト行の例を示しています。これらすべての線は「平行」（y方向の間隔は全長にわたって一定）で、わずかに湾曲しています。アセンダラインはシアン（薄いグレーとして印刷）で、その上の黒いラインは実際には直線です。よく見ると、シアン/グレーの線はその上の黒い直線に対して曲線を描いています。

3.3. Fixed Pitch Detection and Chopping
3.3. 固定（等幅）ピッチ検出と切り分け

Tesseract tests the text lines to determine whether they are fixed pitch. Where it finds fixed pitch text, Tesseract chops the words into characters using the pitch, and disables the chopper and associator on these words for the word recognition step. Fig. 2 shows a typical example of a fixed-pitch word.

Tesseractはテキスト行をテストして、それらが固定ピッチかどうかを判断します。固定ピッチのテキストが見つかった場合、Tesseractはピッチを使用して単語を文字に切り分け、単語認識ステップでこれらの単語のチョッパーとアソシエータを無効にします。固定ピッチ単語の典型例を図2に示します。

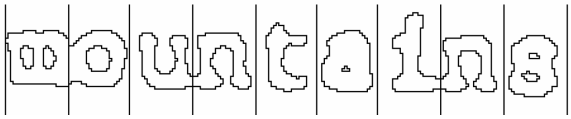

Fig. 2. A fixed-pitch chopped word.
図.2. 固定ピッチの切り分けられた単語

3.4. Proportional Word Finding
3.4. プロポーショナル(可変)ピッチ単語の発見

Non-fixed-pitch or proportional text spacing is a highly non-trivial task. Fig. 3 illustrates some typical problems. The gap between the tens and units of '11.9%' is a similar size to the general space, and is certainly larger than the kerned space between 'erated' and 'junk'. There is no horizontal gap at all between the bounding boxes of 'of' and 'financial'. Tesseract solves most of these problems by measuring gaps in a limited vertical range between the baseline and mean line. Spaces that are close to the threshold at this stage are made fuzzy, so that a final decision can be made after word recognition.

非固定ピッチまたはプロポーショナルテキストの間隔は極めて簡単ではない作業です。図3は、いくつかの典型的な問題を示しています。「11.9%」の数十と単位の間のギャップは、一般的なスペースと同様のサイズであり、「評価済み」と「ジャンク」の間のカーニング済みスペースよりも確実に大きいです。「of」と「Financial」のバウンディングボックスの間に水平方向のギャップはまったくありません。Tesseractは、ベースラインと平均線の間の限られた垂直範囲でギャップを測定することによって、これらの問題のほとんどを解決します。この段階でしきい値に近いスペースはあいまいにされるため、単語認識後に最終的な決定を下すことができます。

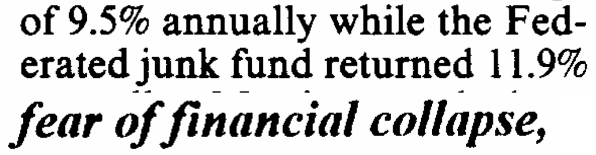

Fig. 3. Some difficult word spacing.
図.3. スペーシングが難しい単語のいくつか

4. Word Recognition
4. 単語の認識

Part of the recognition process for any character recognition engine is to identify how a word should be segmented into characters. The initial segmentation output from line finding is classified first. The rest of the word recognition step applies only to non-fixed-pitch text.

あらゆる文字認識エンジンのための認識プロセスの一部は単語がどのように文字に分割されるべきであるかを識別することです。ラインファインディングからの初期セグメンテーション出力が最初に分類されます。単語認識ステップの残りは、非固定ピッチテキストにのみ適用される。

4.1 Chopping Joined Characters
4.1 結合された文字の分割

While the result from a word (see section 6) is unsatisfactory, Tesseract attempts to improve the result by chopping the blob with worst confidence from the character classifier. Candidate chop points are found from concave vertices of a polygonal approximation [2] of the outline, and may have either another concave vertex opposite, or a line segment. It may take up to 3 pairs of chop points to successfully separate joined characters from the ASCII set.

単語からの結果(セクション6を参照)は満足できるものではありませんが、Tesseractは文字分類子からの信頼性が最も低いブロブを分割することによって結果を改善しようとします。分割点の候補は、輪郭の多角形近似[2]の凹状の頂点から見つけられ、反対側の別の凹状の頂点、または線分を持つことができます。ASCII文字列から結合文字を正しく分離するには、最大3組の分割点が必要です。

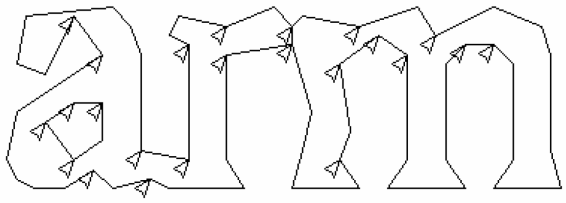

Fig. 4. Candidate chop points and chop.
図.4. 分割点の候補と、分割の様子

Fig. 4 shows a set of candidate chop points with arrows and the selected chop as a line across the outline where the 'r' touches the 'm'. Chops are executed in priority order. Any chop that fails to improve the confidence of the result is undone, but not completely discarded so that the chop can be re-used later by the associator if needed.

図4は、「r」が「m」と接触する輪郭を横切る線として、一組の分割点の候補と、選択された分割とを矢印で示す。分割は優先順位に従って実行されます。結果の信頼性を向上させるのに失敗した分割は元に戻されますが、必要に応じてアソシエータが後でその分割を再利用できるように、完全には破棄されません。

4.2. Associating Broken Characters
4.2. 分断された文字の関連付け

When the potential chops have been exhausted, if the word is still not good enough, it is given to the associator. The associator makes an A* (best first) search of the segmentation graph of possible combinations of the maximally chopped blobs into candidate characters. It does this without actually building the segmentation graph, but instead maintains a hash table of visited states. The A* search proceeds by pulling candidate new states from a priority queue and evaluating them by classifying unclassified combinations of fragments.

分割の可能性が使い果たされてなお、その単語が不十分な場合、それは関連付け器に与えられる。関連付け器は、最大に切り刻まれたブロブの候補文字への可能な組み合わせのセグメンテーショングラフのA*(最良から始める)検索を行います。実際にセグメンテーショングラフを作成せずにこれを行いますが、代わりに訪問状態のハッシュテーブルを維持します。A*検索は、優先待ち行列から候補の新しい状態を引

き出し、未分類のフラグメントの組み合わせを分類することによってそれらを評価することによって進行する。

It may be argued that this fully-chop-then-associate approach is at best inefficient, at worst liable to miss important chops, and that may well be the case. The advantage is that the chop-then-associate scheme simplifies the data structures that would be required to maintain the full segmentation graph.

完全に分割した後に関連付けを行う、この方式は、この上なく非効率的であるし、最悪の場合は重要な分割を見逃す恐れがある点についても議論になるかもしれませんが、完全なセグメンテーショングラフを維持するために必要となるデータ構造を単純化できる利点があります。

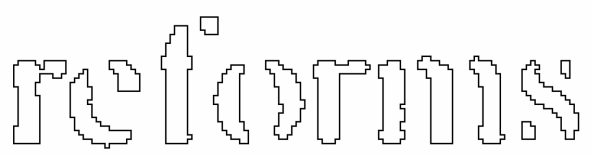

Fig. 5. An easily recognized word.
図.5. 容易に認識された単語

When the A* segmentation search was first implemented in about 1989, Tesseract's accuracy on broken characters was well ahead of the commercial engines of the day. Fig. 5 is a typical example. An essential part of that success was the character classifier that could easily recognize broken characters.

1989年頃にA*セグメンテーション検索が初めて実装された頃、壊れた文字に対するTesseractの正確さはその日の商用エンジンを大きく上回っていました。図5はその代表例である。その成功の重要な部分は、壊れた文字を簡単に認識できる文字分類子でした。

5. Static Character Classifier
5. 静的文字分類器

5.1. Features
5.1. 特徴

An early version of Tesseract used topological features developed from the work of Shillman et. al. [7-8] Though nicely independent of font and size, these features are not robust to the problems found in reallife images, as Bokser [9] describes. An intermediate idea involved the use of segments of the polygonal approximation as features, but this approach is also not robust to damaged characters. For example, in Fig. 6(a), the right side of the shaft is in two main pieces, but in Fig. 6(b) there is just a single piece.

Tesseractの初期のバージョンは、Shillmanら[7-8]の研究から開発された位相的特徴を使用しました。Bokser[9]が述べているように、フォントやサイズにはほとんど依存しないが、これらの機能は現実の画像に見られる問題に対して頑強ではない。中間的なアイデアは、特徴として、多角形近似のセグメントの使用を含みました、しかし、このアプローチはまた損傷した文字に対して頑健ではありません。例えば、図6(a)では、シャフトの右側は2つの主要なピースになっていますが、図6(b)では1つのピースしかありません。

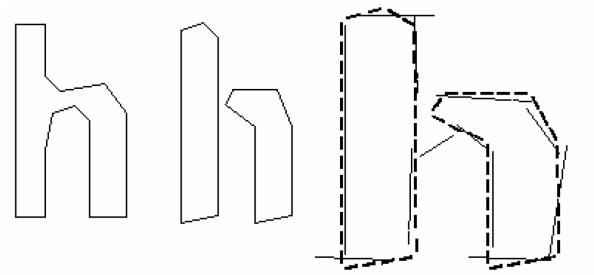

Fig. 6. (a) Pristine 'h, (b) broken 'h', (c) features matched to prototypes.
図.6. (a) 原型の「h」、(b) 壊れた「h」、(c) 見本と照合された特徴

The breakthrough solution is the idea that the features in the unknown need not be the same as the features in the training data. During training, the segments of a polygonal approximation [2] are used for features, but in recognition, features of a small, fixed length (in normalized units) are extracted from the outline and matched many-to-one against the clustered prototype features of the training data. In Fig. 6(c), the short, thick lines are the features extracted from the unknown, and the thin, longer lines are the clustered segments of the polygonal approximation that are used as prototypes. One prototype bridging the two pieces is completely unmatched. Three features on one side and two on the other are unmatched, but, apart from those, every prototype and every feature is well matched. This example shows that this process of small features matching large prototypes is easily able to cope with recognition of damaged images. Its main problem is that the computational cost of computing the distance between an unknown and a prototype is very high.

画期的な解決策は、未知の特徴が訓練データの特徴と同じである必要はないという考えです。訓練中、多角形近似のセグメント[2]がフィーチャに使用されますが、認識では、小さい、固定長(正規化単位)のフィーチャがアウトラインから抽出され、クラスタ化されたプロトタイプフィーチャに対して多対1で一致します。トレーニングデータ図6(c)では、細くて太い線が未知から抽出された特徴であり、細くて長い線がプロトタイプとして使用される多角形近似のクラスタ化されたセグメントである。2つの部分を橋渡しする1つのプロトタイプは完全に比類のないです。一方の側に3つの機能と他方の側に2つの機能が並ぶものはありませんが、それ以外のすべてのプロトタイプとすべての機能はよく一致しています。この例は、大きなプロトタイプと一致する小さな機能のこのプロセスが、損傷した画像の認識に容易に対処できることを示しています。その主な問題は、未知のものとプロトタイプの間の距離を計算するための計算コストが非常に高いことです。

The features extracted from the unknown are thus 3-dimensional, (x, y position, angle), with typically 50-100 features in a character, and the prototype features are 4-dimensional (x, y, position, angle, length), with typically 10-20 features in a prototype configuration.

未知から抽出された特徴は、したがって、文字内に典型的には50～100個の特徴を有する3次元（x、y位置、角度）であり、プロトタイプ特徴は4次元（x、y、位置、角度、長さ）である。プロトタイプ構成では、通常10～20個のフィーチャーがあります。

5.2. Classification
5.2. 分類

Classification proceeds as a two-step process. In the first step, a class pruner creates a shortlist of character classes that the unknown might match. Each feature fetches, from a coarsely quantized 3-dimensional look-up table, a bit-vector of classes that it might match, and the bit-vectors are summed over all the features. The classes with the highest counts (after correcting for expected number of features) become the short-list for the next step.

分類は2段階のプロセスとして進行します。最初のステップでは、クラスプルーナが、未知の文字クラスと一致する可能性がある文字クラスのショートリストを作成します。各特徴は、粗く量子化された3次元ルックアップテーブルから、それがマッチするかもしれないクラスのビットベクトルを取り出し、そしてビットベクトルは全ての特徴にわたって合計される。（予想される数のフィーチャーを補正した後で）最もカウント数が多いクラスは、次のステップの候補リストになります。

Each feature of the unknown looks up a bit vector of prototypes of the given class that it might match, and then the actual similarity between them is computed. Each prototype character class is represented by a logical sum-of-product expression with each term called a *configuration*, so the distance calculation process keeps a record of the total similarity evidence of each feature in each configuration, as well as of each prototype. The best combined distance, which is calculated from the summed feature and prototype evidences, is the best over all the stored configurations of the class.

未知の各特徴は、それがマッチするかもしれない与えられたクラスのプロトタイプのビットベクトルを調べ、そしてそれらの間の実際の類似性が計算されます。各プロトタイプ文字クラスは、構成と呼ばれる各項を持つ論理積和式で表されるため、距離計算プロセスでは、各構成内の各フィーチャおよび各プロトタイプの総合的な類似性の証拠の記録が保持されます。合計された特徴とプロトタイプの証拠から計算される最良の組み合わせ距離は、クラスのすべての格納された構成に対して最良です。

5.3. Training Data
5.3. トレーニングデータ

Since the classifier is able to recognize damaged characters easily, the classifier was not trained on damaged characters. In fact, the classifier was trained on a mere 20 samples of 94 characters from 8 fonts in a single size, but with 4 attributes (normal, bold, italic, bold italic), making a total of 60160 training samples. This is a significant contrast to other published classifiers, such as the Calera classifier with more than a million samples [9], and Baird's 100-font classifier [10] with 1175000 training samples.

分類器は損傷した文字を容易に認識することができるので、分類器は損傷した文字について訓練されていなかった。実際、分類子は単一のサイズで8つのフォントから94文字のわずか20のサンプルでトレーニングされましたが、4つの属性（normal、bold、italic、bold italic）を持ち、合計60160のトレーニングサンプルを作成しました。これは、百万を超えるサンプルを持つCalera分類器[9]や、1175000個のトレーニングサンプルを含むBairdの100フォント分類器[10]など、他の公開されている分類器とはかなり対照的です。

6. Linguistic Analysis
6. 言語分析

Tesseract contains relatively little linguistic analysis. Whenever the word recognition module is considering a new segmentation, the linguistic module (mis-named the permuter) chooses the best available word string in each of the following categories: Top frequent word, Top dictionary word, Top numeric word, Top UPPER case word, Top lower case word (with optional initial upper), Top classifier choice word. The final decision for a given segmentation is simply the word with the lowest total distance rating, where each of the above categories is multiplied by a different constant.

Tesseractは比較的小さな言語分析を含んでいます。単語認識モジュールが新しいセグメンテーションを検討しているときはいつでも、言語モジュール（permuterと誤称されている）は、以下のカテゴリーのそれぞれにおいて最も利用可能な単語列を選びます:トップ頻出単語、トップ辞書単語、トップ数字単語、トップ大文字ケースワード 、上の小文字の単語（オプションの頭文字を含む）、上の分類子選択の単語。与えられたセグメンテーションのための最終的な決定は単に最も低い合計距離評価を持つ単語です、そこでは上記のカテゴリのそれぞれは異なる定数によって乗算されます。

Words from different segmentations may have different numbers of characters in them. It is hard to compare these words directly, even where a classifier claims to be producing probabilities, which Tesseract does not. This problem is solved in Tesseract by generating two numbers for each character classification. The first, called the confidence, is minus the normalized distance from the prototype. This enables it to be a "confidence" in the sense that greater

numbers are better, but still a distance, as, the farther from zero, the greater the distance. The second output, called the rating, multiplies the normalized distance from the prototype by the total outline length in the unknown character. Ratings for characters within a word can be summed meaningfully, since the total outline length for all characters within a word is always the same.

異なるセグメンテーションからの単語は、それらの中に異なる文字数を持つことができます。分類子が確率を生成していると主張している場合でも、これらの単語を直接比較するのは困難です。この問題はTesseractで各文字分類ごとに2つの数を生成することによって解決されます。1つ目は信頼度と呼ばれ、プロトタイプからの正規化距離を引いたものです。これは、数字が大きいほど良いという意味での「自信」になることを可能にしますが、それでも、ゼロから遠くなるほど距離は大きくなります。評価と呼ばれる2番目の出力は、プロトタイプからの正規化距離に、未知の文字のアウトラインの全長を掛けたものです。単語内のすべての文字のアウトラインの合計の長さは常に同じなので、単語内の文字の評価は有意義に合計できます。

7. Adaptive Classifier
7. 適応分類器

It has been suggested [11] and demonstrated [12] that OCR engines can benefit from the use of an adaptive classifier. Since the static classifier has to be good at generalizing to any kind of font, its ability to discriminate between different characters or between characters and non-characters is weakened. A more font-sensitive adaptive classifier that is trained by the output of the static classifier is therefore commonly [13] used to obtain greater discrimination within each document, where the number of fonts is limited.

OCRエンジンにとって、適応分類器の使用が有益であることが示唆され[11]、実証されている[12]。静的分類器はあらゆる種類のフォントに一般化するのが得意でなければならないので、異なる文字間または文字と非文字との間を区別するその能力は弱まっている。したがって、静的分類器の出力によって訓練される、よりフォントに敏感な適応型分類器は、フォントの数が限られている各文書内でより大きな識別を得るために一般的に使用される[13]。

Tesseract does not employ a template classifier, but uses the same features and classifier as the static classifier. The only significant difference between the static classifier and the adaptive classifier, apart from the training data, is that the adaptive classifier uses isotropic baseline/x-height normalization, whereas the static classifier normalizes characters by the centroid (first moments) for position and second moments for anisotropic size normalization.

Tesseractはテンプレート分類子を使用しませんが、静的分類子と同じ機能と分類子を使用します。静的分類子と適応分類子の唯一の重要な違いは、学習データとは別に、適応分類子は等方性ベースライン/x高さ正規化を使用するのに対し、静的分類子は位置と2番目のセントロイドで文字を正規化することです。異方性サイズ正規化のためのモーメント

The baseline/x-height normalization makes it easier to distinguish upper and lower case characters as well as improving immunity to noise specks. The main benefit of character moment normalization is removal of font aspect ratio and some degree of font stroke width. It also makes recognition of sub and superscripts simpler, but requires an additional classifier feature to distinguish some upper and lower case characters. Fig. 7 shows an example of 3 letters in baseline/x-height normalized form and moment normalized form.

ベースライン/x高さの正規化により、大文字と小文字の区別が容易になり、ノイズスペックに対する耐性が向上します。文字モーメントの正規化の主な利点は、フォントの縦横比とある程度のフォントストローク幅の削除です。下付き文字や上付き文字の認識も簡単になりますが、大文字と小文字を区別するための追加の分類機能が必要になります。図7は、ベースライン/x高さ正規化形式およびモーメント正規化形式の3文字の例を示す。

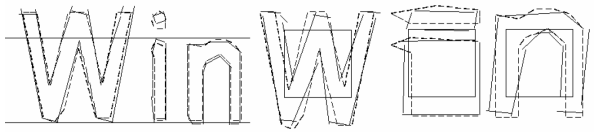

Fig. 7. Baseline and moment normalized letters.

図.7. ベースラインとモーメントの正規化文字

8. Results
8. 結果

Tesseract was included in the 4th UNLV annual test [1] of OCR accuracy, as “HP Labs OCR,” but the code has changed a lot since then, including conversion to Unicode and retraining. Table 1 compares results from a recent version of Tesseract (shown as 2.0) with the original 1995 results (shown as HP). All four 300 DPI binary test sets that were used in the 1995 test are shown, along with the number of errors (Errs), the percent error rate (%Err) and the percent change relative to the 1995 results (%Chg) for both character errors and non-stopword errors. [1] More up-to-date results are at http://code.google.com/p/tesseract-ocr.

Tesseractは、「HP Labs OCR」として、UNLVにおける4回目の年次OCR精度テスト[1]に含まれていましたが、それ以降、コードはUnicodeへの変換や再トレーニングを含めて大きく変わりました。 表1では、最近のバージョンのTesseract（2.0と表示）の結果と元の1995年の結果（HPと表示）を比較しています。1995年のテストで使用された4つの300DPIバイナリテストセットすべてが、エラー数（Errs）、エラー率（%Err）、および1995年の結果に対するパーセント変化（%Chg）とともに示されています。文字エラーおよびノンストップワードエラー[1]最新の結果はhttp://code.google.com/p/tesseract-ocrにあります。

Table 1. Results of Current and old Tesseract.
表1. 新旧Tesseractの結果

		Character			Word		
Ver	Set	Errs	%Err	%Chg	Errs	%Err	%Chg
HP	bus	5959	1.86		1293	4.27	
2.0	bus	6449	2.02	8.22	1295	4.28	0.15
HP	doe	36349	2.48		7042	5.13	
2.0	doe	29921	2.04	-17.68	6791	4.95	-3.56
HP	mag	15043	2.26		3379	5.01	
2.0	mag	14814	2.22	-1.52	3133	4.64	-7.28
HP	news	6432	1.31		1502	3.06	
2.0	news	7935	1.61	23.36	1284	2.62	-14.51
2.0	total	59119		-7.31	12503		-5.39

9. Conclusion and Further Work
9. 結論とさらなる研究

After lying dormant for more than 10 years, Tesseract is now behind the leading commercial engines in terms of its accuracy. Its key strength is probably its unusual choice of features. Its key weakness is probably its use of a polygonal approximation as input to the classifier instead of the raw outlines.

Tesseractは10年以上休止した後、精度の面で今や主要な商用エンジンの背後にあります。 その主な強みは、おそらくその珍しい機能の選択です。 その主な弱点は、生のアウトラインの代わりに分類子への入力として多角形近似を使用することです。

With internationalization done, accuracy could probably be improved significantly with the judicious addition of a Hidden-Markov-Model-based character n-gram model, and possibly an improved chopper.

国際化が行われれば、隠れマルコフモデルベースの文字プログラムモデルと、おそらくは改良された分割器を慎重に追加することで、おそらく精度が大幅に向上するでしょう。

10. Acknowledgements
10. 謝辞

The author would like to thank John Burns and Tom Nartker for their efforts in making Tesseract open source, the ISRI group at UNLV for sharing their tools and data, as well as Luc Vincent, Igor Krivokon, Dar-Shyang Lee, and Thomas Kielbus for their comments on the content of this paper.

著者は、John Burns氏とTom Nartker氏がTesseractをオープンソースにしたこと、UNLVのISRIグループがツールとデータを共有してくれたことに感謝します。この論文の内容に関する彼らのコメント。

11. References
11. 参考文献

[1] S.V. Rice, F.R. Jenkins, T.A. Nartker, *The Fourth Annual Test of OCR Accuracy, Technical Report 95-03*, Information Science Research Institute, University of Nevada, Las Vegas, July 1995.

[2] R.W. Smith, *The Extraction and Recognition of Text from Multimedia Document Images*, PhD Thesis, University of Bristol, November 1987.

[3] R. Smith, "A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation", *Proc. of the 3rd Int. Conf. on Document Analysis and Recognition* (Vol. 2), IEEE 1995, pp. 1145-1148.

[4] P.J. Rousseeuw, A.M. Leroy, *Robust Regression and Outlier Detection*, Wiley-IEEE, 2003.

[5] S.V. Rice, G. Nagy, T.A. Nartker, *Optical Character Recognition: An Illustrated Guide to the Frontier*, Kluwer Academic Publishers, USA 1999, pp. 57-60.

[6] P.J. Schneider, "An Algorithm for Automatically Fitting Digitized Curves", in A.S. Glassner, *Graphics Gems I*, Morgan Kaufmann, 1990, pp. 612-626.

[7] R.J. Shillman, *Character Recognition Based on Phenomenological Attributes: Theory and Methods*, PhD. Thesis, Massachusetts Institute of Technology. 1974.

[8] B.A. Blesser, T.T. Kuklinski, R.J. Shillman, "Empirical Tests for Feature Selection Based on a Pscychological Theory of Character Recognition", *Pattern Recognition* **8**(2), Elsevier, New York, 1976.

[9] M. Bokser, "Omnidocument Technologies", *Proc. IEEE* **80**(7), IEEE, USA, Jul 1992, pp. 1066-1078.

[10] H.S. Baird, R. Fossey, "A 100-Font Classifier", *Proc. of the 1st Int. Conf. on Document Analysis and Recognition*, IEEE, 1991, pp 332-340.

[11] G. Nagy, "At the frontiers of OCR", *Proc. IEEE* **80**(7), IEEE, USA, Jul 1992, pp 1093-1100.

[12] G. Nagy, Y. Xu, "Automatic Prototype Extraction for Adaptive OCR", *Proc. of the 4th Int. Conf. on Document Analysis and Recognition*, IEEE, Aug 1997, pp 278-282.

[13] I. Marosi, "Industrial OCR approaches: architecture, algorithms and adaptation techniques", *Document Recognition and Retrieval XIV*, SPIE Jan 2007, 6500-01.