History of the Tesseract OCR Engine: What Worked and What Didn't

How to Build a World-Class OCR Engine in Less Than 20 Years

Ray Smith[a]

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

ABSTRACT

This paper describes the development history of the Tesseract OCR engine, and compares the methods to general changes in the field over a similar time period. Emphasis is placed on the lessons learned with the goal of providing a primer for those interested in OCR research.

Keywords: OCR, Machine learning, Structural pattern recognition, Multi-language OCR.

1. INTRODUCTION

The Tesseract OCR Engine is an open-source system that was developed originally at HP between 1985 and 1995, shelved for 10 years, open-sourced in 2006 and now developed mostly at Google. Its accuracy was among the top 3 in the 1995 UNLV Test[1], and with recent work is is again catching up with the commercial OCR engines. Despite its age, some of the components of Tesseract are surprisingly similar to more modern approaches. Among other things, the history of the development of Tesseract is a microcosm of the debate over statistical vs non-statistical classification methods. This paper provides a historical perspective, with concentration on the important lessons learned during Tesseract's development, covering both successes and failures, with a view to guiding others how to build an OCR system. It also compares the methods used with those that have been fashionable in recent times.

Nothing was published on Tesseract during its initial development. The project was run in "stealth mode" as a joint project between HP Labs in Bristol, and the HP Scanner Division in Colorado, with the aim of creating a differentiating feature for HP scanners. The resulting technology almost became a product in the early 1990s. It was much more accurate on poor quality images than the first software-only OCR engines, but a lot slower, so it would require hardware assist in the scanner itself. Due to various reasons, one of which was the hurdle of internationalization, the HP Scanner Division decided at the end of 1990 not to develop Tesseract into a product. HP Labs continued development until the end of 1994, initially with the aim of pushing the limits of accuracy on degraded images, and later with the aim of using OCR for document compression. Even during the latter development phase, very little was published

The body of this paper is laid out as follows: Sec. 2 describes the overall system architecture of Tesseract, Sec. 3 introduces the feature space by means of how it developed over time, and compares with currently common methodologies. Likewise, Sec. 4 covers the classifier, placing it in the perspective of some recent publications, and Sec. 5 covers testing. Languages are discussed in Sec. 6, and finally Sec. 7 covers the ad-hoc nature of the word classifier.

2. SYSTEM ARCHITECTURE

The overall architecture, shown in Fig.1, has stayed mostly unchanged over the years, and shows its origin in the traditional pipelined approach. The actual character recognition component has a two-pass format, being used in both "Recognize Word Pass 1" and "Recognize Word Pass 2," which allows an on-the-fly adaptive classifier to be trained and utilized in the first pass, and to re-visit unsatisfactory words in the second pass.

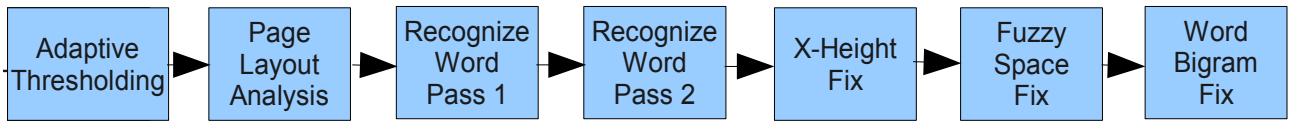

Figure 1. Block diagram of the overall architecture of Tesseract.

Document Recognition and Retrieval XX, edited by Richard Zanibbi, Bertrand Coüasnon,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 8658, 865802 · © 2013 SPIE-IS&T
CCC code: 0277-786/13/$18 · doi: 10.1117/12.2010051

Further "fixing" passes have been added to firm-up decisions that were left fuzzy or need correcting from an earlier step, like spacing, x-heights, or words that need multi-word context to resolve. In many ways, it is these additional phases that separate the styles of OCR architecture, which can be categorized as follows:

- Traditional, naive. Traditional pipelined "feed forward" systems start as a series of steps that make hard decisions in one domain, and pass the results on to the next part of the pipeline.
- Traditional, mature. Matured pipelined systems are characterized by additional steps that revisit some of the earlier decisions with additional information from other parts of the pipeline. Examples include adaptive character classification, adaptive font spacing/character size models, and document dictionaries.
- Modern, naive. More recent approaches try to avoid premature decision-making by pushing all the hard problems into a monolithic statistical module, such as a Hidden Markov Model (HMM) and expect it to resolve everything at once. These systems began with segmentation-free, also-known-as sliding window classification, in which decisions over character boundaries are made in parallel with character classification. Between-word spacing falls out naturally from such systems, but it is more difficult to incorporate knowledge such as words tend to use only a single font. The pioneering work in this area is the Document Image Decoding(DID) system, and more recently, the BBN system.
- Modern, mature. It would be against the principles of an HMM-based OCR system to patch it with post-processing modules that re-visit earlier decisions, so mature modern systems will be characterized by increasingly complex models that take into account all the structure of printed information, much as the original DID system did, but without requiring a hand-coded model for each type of page layout.

The Tesseract word recognizer, shown in Fig. 2 searches for the optimal segmentation of a word into isolated characters to feed to the character classifier. In terms of the Casey and Lecolinet survey on segmentation, Tesseract is a hybrid of the classical approach and recognition-based segmentation, but does not apply the over-segmentation algorithm described therein. An over-segmenting word recognizer would maximally segment a word (or a text-line) and then apply a beam search to choose the best few segmentation paths through the resulting lattice. The segmentation search in Tesseract takes a minimalist approach. Starting with the initial connected-component-based segmentation, while the word is unsatisfactory, it chops components that have poor confidence from the character classifier. At each step a beam search combines the character classifier results with the language model, as described in Sec. 7. If chopping fails to produce a satisfactory result, it then searches the segmentation lattice, by connecting adjacent character fragments, based on hints from the classifier and language model, such as where in the word the dictionary search dead-ended. The beam search is re-run for each segmentation hypothesis, again only until a satisfactory word is produced.

The principle behind this minimalist approach is that in most languages, the initial segmentation is close to correct, so minimal segmentation biases the search in favor of stopping at a state close to the initial segmentation. This reduces the probability of hallucinating garbage, i.e. chopping a perfectly good character into 'iii' or falsely merging perfectly good characters into 'm', as well as increasing computational efficiency. The disadvantage is that it can occasionally miss the correct segmentation due to finding a satisfactory result before it gets there.

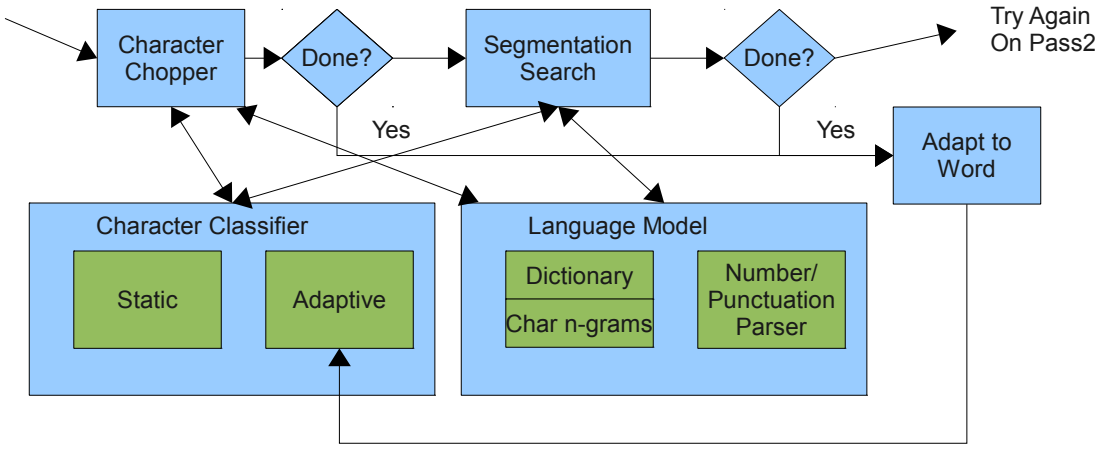

Figure 2. Block diagram of the Tesseract word recognizer.

HP had an independent layout analysis technology that was used in a product, so the only layout analysis component that was originally required was detecting text lines and words within a previously-identified text block. For this reason, layout analysis was not part of the open source release. Full layout analysis based on tab-stop detection, that can cope with non-text, multi-columns, and tables was added to Tesseract recently. The layout analysis and character segmentation processes make use of outlines of binary connected components, but there is nothing fundamental that says the classifier has to extract features from the binary outlines. In a recent change, aimed at Chinese and low-resolution input, the ability to extract features from greyscale was added.

An important component of Tesseract that helps improve accuracy on unseen fonts is the adaptive classifier. It typically reduces error-rates on a large enough document by 30-60% relative to the static classifier alone. The adaptive classifier is identical to the static classifier, except for the normalization that is applied to the outlines, (See Sec.3.4) and the fact that it is trained on-the-fly. The individual characters of each confidently recognized word are presented to the adaptive classifier as training data, as the words are recognized in Pass 1, and the adaptive classifier starts responding with answers (still in Pass 1) after it has seen only 3 matching samples of a character. Thus if the word "infinity" were recognized, the adaptive classifier would start recognizing 'i' on the next word.

3. FEATURE SPACE

Any textbook on machine learning or pattern recognition will discuss feature space as an n-dimensional vector space in which each training sample and each unknown occupies a single point, represented by a single n-dimensional feature vector. Conventional machine learning classifier methods either find the nearest training sample in this n-dimensional space, (e.g. kNN classifiers) or divide the space into regions which correspond to class labels (eg SVMs) and return the corresponding label for a given unknown. Tesseract took a different approach.

3.1 Early Work – Statistical Beginnings

An unconventional aspect of Tesseract was a consequence of an early choice of feature space. Early work on Tesseract was strongly influenced by psychological studies that had shown that human perception makes use of structural or topological features, such as "There is a stick on the left, and a loop at the bottom." This kind of description does not map well to a fixed n-dimensional feature space! Thus most published research has abandoned structural features and used simple pixel-level features or other quantized features that can be expressed as a fixed-dimension vector. In contrast, Tesseract originally extracted topological features from the skeleton. Character skeletons are appealing, partly because the skeleton should correspond to the path of the pen in handwriting, and partly because the aforementioned psychological studies indicated that human perception of characters is based largely on the skeleton. Unfortunately, in machine print, the ink is not created by pen strokes, and characters contain serifs, making the skeleton hard to define robustly. This difficulty in definition is reflected in the large number of papers on skeletonization, and yields the first lesson: ***If some required process in your system has a large number of published papers describing different solutions, choose an alternative process, as it probably means that there is no good solution.***

The decision was made however, and Tesseract went down the path of classification from a collection of low-dimensional feature vectors, instead of a single high-dimensional feature vector. The classifier was based on a Bayesian parametric Gaussian Mixture Model that treats each font/character combination as a separate class. It had to assume statistical independence in its probability calculations, and there were many discussions in the team over the invalidity of the independence assumption. Symbolically, in training, sample features, each of n dimensions, of a single font/character combination, $k \in [1, K]$, are clustered to J_k cluster means, with models $N(\mu_{ijk}, \sigma_{ijk}) : i \in [1, n], j \in [1, J_k]$. An unknown with M n-dimensional feature vectors $\vec{X}_l = \{x_{il} : i \in [1, n], l \in [1, M]\}$, was classified using:

$$argmax(k) \prod_{l,i} \frac{1}{\sigma_{ijk}} \exp\left[-\frac{1}{2}\left(\frac{x_{il} - \mu_{ijk}}{\sigma_{ijk}}\right)^2\right], \quad (1)$$

with the indices being: i=feature dimension, j=cluster, k=font/character class, and l=unknown's feature index defined such that l and j are matched to minimize $(x_{il} - \mu_{ijk})^2$.

After dropping skeletonization, features continued to be topological, but were extracted from outlines, until they showed themselves to be brittle in the presence of degraded images, and lesson 2 emerged: ***Features must be as invariant as possible to as many as possible of the expected degradations.***

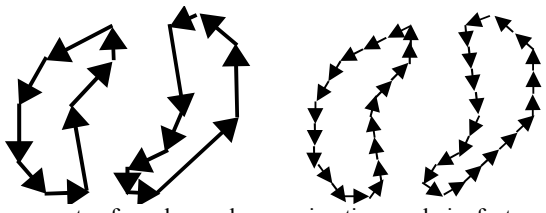

Figure 3. Nanofeatures (left) were segments of a polygonal approximation, and picofeatures (right) cut these into segments of a fixed length.

3.2 Shrinking Features – Statistics Abandoned

After discovering that topological features were too brittle, features became smaller and smaller fragments of outlines, (they were even named microfeatures, nanofeatures, and finally picofeatures!) at one point being 4-dimensional, (hence the name Tesseract) but eventually reducing to just 3-dimensions: x, y position and direction in the range [0, 2pi). The fourth dimension, which had been length, was reduced to a small constant. The last two steps of this process are shown in Fig. 3. The nanofeatures are just the segments of a polygonal approximation of the outline, and the picofeatures cut these segments down further into pieces of a constant length. In the current (v.3.02) implementation of Tesseract, only picofeatures are extracted from the unknown, but nanofeatures are clustered during training. This asymmetric arrangement provides better clustering during training, and the desired robustness to noise. Since a nanofeature is equivalent to one or more collinear picofeatures, the difference is ignored in the rest of the description.

This reduction in size of the features mirrors the changes in OCR research over the last twenty years or so, in which structural features have largely been abandoned in favor of simpler features, like pixels. A notable exception to this trend is the emergence of Deep Belief Nets as systems that derive their own higher-level/structural features from pixel-level input through multiple levels of convolution.

In Tesseract, the number of features had increased by an order of magnitude over the original topological features, and the assumption of statistical independence had become completely untenable as a result. Failure of the independence assumption was blamed for the "probabilities" being unrelated within a single classification, making the ranking of alternate answers unreliable. The pretense that Tesseract was dealing with probabilities was therefore dropped, and replaced with simple Euclidean distance. Upon re-casting the distance as spatial distance of a fragment of the outline of the unknown from a fragment of the outline of a training character, it quickly became obvious that the distance needed to be symmetrized, i.e. that it was also necessary to measure the distance of each fragment of the training character from the unknown character. As an equation, the classification had become:

$$argmin(k)\frac{1}{M+J_k}\left(\sum_{l,i}(x_{il}-\mu_{ijk})^2+\sum_{j,i}(x_{il}-\mu_{ijk})^2\right) \quad . \tag{2}$$

Using the same index notation as for Eq (1), the matching condition in the first sum, becomes best *j* for each *l,* and in the second sum, best *l* for each *j*. The classifier thus finds for each feature in the unknown the nearest cluster mean and vice-versa, summing all the distances and dividing by the total number of features in the unknown and the training sample, analogous to the Hausdorff distance, except that Eq (2) computes the mean distance instead of the maximum. The key difference from Eq (1) is the symmetrization in finding the nearest matching feature both ways instead of one-way. This is extremely important for a couple of reasons.

Firstly, the concept that 'e' has "more features" than 'c' exists in Tesseract and is encoded in this symmetrized distance metric. Conventional approaches have to somehow encode this difference in the many dimensions of their fixed dimension feature space.

Secondly, dropping the probability pretense does not make the problem of statistical independence go away! Because the features are so small, groups of them are likely to occur together, and move together as characters change shape. This makes it as difficult to make judgments about the absolute distances, as it is make inferences using the probabilities. The big advantage of the symmetrization though is that it mitigates the problem of statistical independence in the second summation, which specifies a collection of features that are *required*, whereas the first summation specifies features that are *allowed*. Continuing with the 'c'/'e' example, (almost) all the features in 'c' are allowed by 'e' so using Eq (1) an unknown of 'c' could get an almost equal score for 'c' or 'e', but in Eq (2), 'e' *requires* a cross-bar, so an unknown 'c' would get a worse match (greater distance) for 'e' than for 'c'. Statistical independence bites when the position of the

cross-bar of an unknown 'e' is different to any seen in training, and multiple features are penalized for being out of position, when in reality these features are connected so they are destined to move together with statistical dependence.

3.3 Variable Dimensions or Fixed? – A Comparison with Conventional Methods

Conventional (some would say modern) classifiers, for example SVMs, require a fixed-dimension feature space. A conventional way to convert from a structural feature set, as with Tesseract, to a fixed dimension space, is to quantize the feature space to a space of binary features in high dimensions, where each bit represents the presence of a feature in each quantum cell. Thus *n* Tesseract features would give rise to a sparse high-dimension binary feature vector with *n* set bits. This method is employed in the Tesseract Class Pruner (See Sec. 4), but it has its disadvantages. The biggest problem is that Euclidean distance in feature space becomes Hamming distance when each dimension is binary, so a single feature that has changed its value slightly to move from one quantum cell to the next has a hamming distance of 2, which is the same as if the feature had disappeared and been replaced by one anywhere in feature space. Features that were proximate in the original space thus have no special connection in the quantized space, which means that the generalization power is greatly reduced.

Another approach to mapping a structural space to a fixed-dimension feature space is by computing histograms on a coarser quantization grid. (e.g. Histogram of Gradients.) This approach has the problem of handling features that naturally sit near a quantum boundary. Recent work has been done to address this problem by optimizing the coarse quantum boundaries.

The modern approach to solving the problem of reduced generalization is to throw more training data at the classifier during training. Theoretical analysis and experimental tests in this area have concluded that discriminative models asymptotically beat generative models, (or the naïve Bayes generative model at least) but that more training data is usually required. This is another way of saying that discriminative models have less generalization power, but are better at learning more complex spaces than generative models.

Hidden Markov Model (HMM) classifiers, if anything, are the closest conventional systems to Tesseract. Their state transitions with self-loops are analogous to handling multiple variable numbers of features, albeit with the severe limitation of sequence, and usually with an order of magnitude less states than the number of features in a typical character in Tesseract. The main advantage of HMM classifiers is that the internal state transitions contain finer-grained detail of the required features, and thus a better solution for statistical independence than can be expressed by Eq (2).

Principled is a word that is often associated with the statistical methods, particularly those with HMMs, and non-statistical methods, such as Tesseract are described as *ad-hoc,* yet these so-called principled methods often use the log-linear model to combine probabilities, which are derived from a Gaussian Mixture Model. In a log-linear model, the classification is based on tuning a set of weights α_i:

$$argmax(k)\sum_i \alpha_i \log(p(x_i|k)), \tag{3}$$

for a collection over i of some *feature functions* x_i, using k as the class as before, and the Gaussian Mixture Model provides probabilities based on the Normal distribution:

$$p(x_i|k)=\frac{1}{\sigma_{ik}\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{x_i-\mu_{ik}}{\sigma_{ik}}\right)^2\right] \quad . \tag{4}$$

Substituting Eq(4) into Eq(3) gives:

$$argmax(k)\sum_i \alpha_i\left[\log\left(\frac{1}{\sigma_{ik}\sqrt{2\pi}}\right)-\frac{1}{2}\left(\frac{x_i-\mu_{ik}}{\sigma_{ik}}\right)^2\right], \tag{5}$$

which looks rather like Eq. (2), with the addition of an additive and a multiplicative constant, but the loss of the symmetry, which is made up by the combinations of states that are allowed in a character. This is a particularly interesting result, as it shows just how closely related are the ad-hoc, statistics-free approach of Tesseract and the models used by the so-called principled statistical methods. The addition of the standard deviation is hardly of rigorous importance, with a learned parameter that overrules it. The real difference here is the additional set of learned parameters, (the α_i) but it should be noted that these are not derived from a rigorous statistical model.

3.4 Adaption and Generalization via Normalization

Sec. 2 mentioned that Tesseract uses an adaptive classifier. The only real difference between the adaptive classifier and the static classifier is the type of normalization that is applied to the outline at feature extraction. In the static classifier, the centroid of the outline of the unknown is centered in feature space, and scaled anisotropically to normalize the second moments of the outlines. This centering and scaling aims to eliminate some font differences, such as aspect ratio, and allows sub/superscripts to be recognized as normal characters, but also introduces some ambiguities. The adaptive classifier normalizes the unknown by centering the horizontal centroid of the outline, but the vertical center of the text-line. The scaling is isotropic to normalize the x-height of the character. This normalization retains font differences, and improves immunity to pepper noise, but makes sub/superscript differ from normal text. The combination of the different normalizations helps improve overall accuracy.

4. CLASSIFIER

The classifier is essentially an optimized k-Nearest Neighbor (kNN) classifier. It returns the closest matching training sample, at the level of granularity of the grapheme-cluster, font pair, nominally using Eq.(2) to compute the distances. The brute-force time is O(J_kKMn). While the features were topological, this was not so computationally intensive, but with the features shrunk down there are 50-100 present in most characters, just for English, with J_k,M ~ 50-100, K = 3520, (32 training fonts * 110 character-set), and n = 3, makes for O(10^8) distance calculations, per character classification, which was prohibitively expensive on the machines of the time.

The primary solution to reduce the squared-order feature matching to linear is quantization, as described in Sec. 3.3, and the inverted index. This is combined with two-stage classification to reduce the total CPU load dramatically. The first stage classifier, called the *Class Pruner,* shown in Fig. 4, indexes the quantized value of each feature vector in the unknown to obtain a set of classes that allow the feature. The number of such feature hits for each class is summed over the features and the best few matching classes become a short-list of classes for the second stage. This process is identical to a linear classifier, except that no multiplication need be performed. Symbolically, given a quantization function $f:\mathbb{R}_n \rightarrow \{1,\ldots n'\}$ the quantization process can be described by:

$$\vec{X}_l=\{x_{il}: i\in[1,n], l\in[1,M]\}\rightarrow\vec{Q}=\{q_i, i\in[1,n'], q_i=1 \; if \; \exists l: f(\vec{X}_l)=i, q_i=0 \; otherwise\} \quad .$$

In the Tesseract Class Pruner each of the 3-dimensions is quantized to 24 cells, so $n'=24^3=13824$. The linear classifier in the class pruner is then easily described by: $argmax(k)\sum_{i=1}^{n'} w_{ik}\, q_i$, where the weights w_{ik}, take on 2-bit values. The 2-bit weights define the acceptance neighborhood in the inverted index around the mean of each feature cluster and are computed using arbitrary constants of distance rather than the standard deviation. With sufficient training data, there is no particular reason not to use the standard deviation to define the neighborhoods.

The secondary classifier computes the distance as in Eq (2), using a second inverted index to match features in the unknown to the features of the training samples, known as prototypes. Computation time is thus reduced to O(*KM*) for the Class Pruner and O(*Mn*) for the second stage for each class proposed by the Class Pruner. In English, the Class Pruner takes about 10% of total CPU (about 60μs per classification), and the secondary classifier about 45%. In Chinese, these roles are reversed, since the class pruner is linear in the number of classes, and the class pruner takes around 60%, with the secondary classifier at about 30%. Even these speed improvements left Tesseract about 10x slower than the commercial engines of the mid 1990s, but in the intervening years, in improving accuracy, the commercial engines have slowed down to match it, making Tesseract's speed comparable today

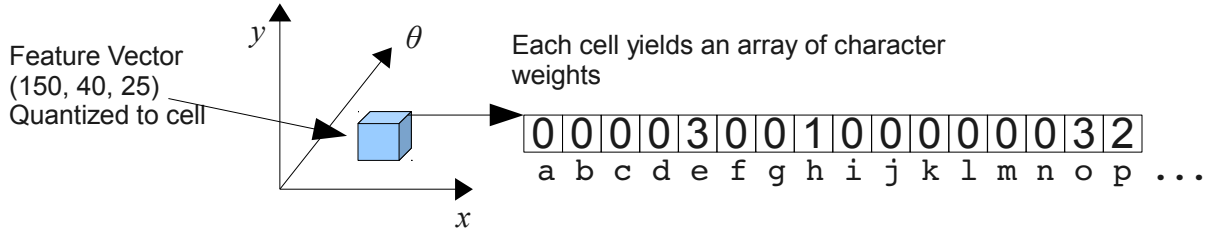

Figure 4. The Tesseract Class Pruner uses quantization and an inverted index for speed.

5. TESTING

The importance of testing to the development of Tesseract and other OCR systems cannot be over-emphasized. The initial development was done on a set of 30 page images that were scanned at 250 pixels per inch with 4-bits of grey per pixel. The scanner was custom-built using a linear CCD array attached to the drawing head of a small HP Pen Plotter. At 3MB per image, there wasn't even room for a single whole image in a 4MB machine. Accuracy development really took off after the development set was expanded to 400 pages at 8 bits of grey and 300 pixels per inch, and a compute-server system was built consisting of around 20 obsoleted machines built from scrap parts left over from system upgrades. The system was trained on a separate set of images that were printed in a selection of fonts and scanned to obtain realistic image degradation. Several important lessons emerged from the testing of Tesseract over the years:

- ***Small test sets are meaningless.*** It is easy to demonstrate accurate OCR on a small test set, but to demonstrate "industrial strength" takes a significant size of test set with a realistic variation in material. Taking an example from thresholding, which itself is an example of lesson 1, Minimum Error Thresholding uses a statistical model of the foreground and background pixels of an image to construct a thresholding solution, and then tests the solution using a small number of images that were artificially generated using the same model. Not surprisingly, it works well on this small test set, but the solution fails miserably on many real images of text because the model itself is flawed. Real images of thin text do not fit the model at all, due to the foreground pixels being heavily out-numbered by pixels on the edges of the characters, which makes the foreground peak much wider than it really should be.
- ***Test on different data to the training data.*** Recognizing data like the training data, for either single or multi-font OCR is easy. Shrink-wrapped commercial OCR engines are called "omni-font" because they could be asked to recognize anything printed in any font and are designed to do so. Omni-font OCR is hard, so if you want to demonstrate industrial strength OCR you have to test on completely different data to the training data. ***This makes the common methodology of randomly dividing a data set into training and test/validation sets flawed.*** Such a random division will place similar or even the same fonts in training and test sets, which in (omni-font) OCR terms is cheating. Even the commonly used cross-validation methods only serve to minimize the over-training and don't evaluate generalization beyond the data set from which the samples are drawn. Tesseract is now trained on synthetic data and tested on real data, so there is no doubt that the test results show some generalization. The development test set is different from the blind test set.
- ***Test every change.*** Any code change is capable of causing a regression, so the more often it is tested, the easier it is to identify what caused a problem. At a finer-grained level, unit tests provide confidence that changes do not break assumptions or cause regressions on previously-fixed problems.
- ***What you measure improves.*** The more dimensions of both test data and metrics that you have, the more readily you can identify what module is to blame for a particular error. For example, just measuring character error-rate doesn't show whether errors are due to the character classifier, language model or layout analysis, but add in a bag-of-words word error-rate as well as a longest-common-substring (edit distance) word error-rate, and the difference between them gives an indication of the layout analysis error-rate. On the data side, just looking at scanned printed material tells little about how well a system will perform on camera imagery.
- ***If it can break it will.*** Software as complex as a full OCR system must be tested very thoroughly. One of the successes of Tesseract at the UNLV test of 1995 was zero crashes. This robustness was obtained by testing on over 80000 images generated by thresholding the 400 page set at 200 different thresholds, which created a lot of badly degraded images. Every resulting crash was fixed.
- ***Generous compute power helps a lot.*** Fast turn-around leads to more rapid development. The 80000 pages above took the order of 400 cpu-days, which was only possible with a distributed test environment. We now regularly test on around 1500 books in about 30 minutes.
- ***If you can write faster code in the time it takes a test to run, do so!*** The commonly-held belief that premature optimization is wasteful or somehow narrows the research direction needs to be balanced with the fact that a faster train/test cycle leads to more rapid progress. One of the benefits of avoiding iterative training algorithms is that the core shape training for Tesseract takes just a few minutes instead of the hours or days that it commonly takes to train, for example, a neural network-based classifier.

6. LANGUAGES

Right up until 2007, Tesseract was designed only for English. It was more by luck than good judgment that Tesseract turned out to be fairly simple to upgrade to handle most of the world's languages. The upgrade path started with extension to the Western European languages, followed by the East Asian languages, then Indic, and finally Hebrew. A key lesson from this section is: ***A language-specific OCR system doesn't contribute much to OCRing the world's languages, as there are very many languages in use even before including historical variants.***

The first design decision in developing a multi-lingual OCR system is internal representation of *Recognition Units* (RU). An RU is an individual shape that the OCR engine recognizes. RU is used here as a generalization of alphabet, character set and grapheme cluster, each of which already has a specific meaning. A different term is needed, as, by design, one OCR system may choose to recognize different sets of shapes from another. For instance, one OCR system, like Tesseract, may choose to recognize whole Chinese characters, where another may recognize the individual radicals within each character. It is important to realize at this early stage of design that for some languages, most notably the Indic group, a single Unicode character is not an adequate representation of an RU. In the Indic languages, multiple consonants can combine with an optional vowel to make a grapheme cluster (representing a syllable) in either a ligature or a group of isolated connected components that may take a different shape from the individual Unicodes. Some of the Indic languages use as many different grapheme clusters as there are characters in use in Chinese, with the additional complexity that they can take 6 or more Unicodes to represent, and they are not all the same size and shape.

In the spirit of providing guidance to future OCR practitioners, the following languages provide in some sense a spanning set that covers most of the orthogonal difficulties:

- **English:** Believe it or not, English is on this list because it is *the most difficult language on which to achieve state-of-the-art.* It is easy to obtain 90% or greater character accuracy on English, but the commercial engines achieve 99%+ due to decades of work on the "long tail" of formatting issues: Drop-caps, small-caps, pair kerning (eg. *Of goods* vs. 11), foreign words (with foreign characters), multi-language documents, double single quote vs. single double quote, em-dash vs. hyphen, soft hyphen vs. hard hyphen, curly quotes vs straight quotes, bullets and in-line logos, bracket matching, Helvetica/Arial I vs. l, Times Roman 1 vs l, difficult fonts (especially italics and script-like), multiple sizes on one line, text on image, vertical text, inverse text, non-rectangular blocks, line numbers, tables, equations, sub and superscript, underline, strike-through. To achieve such accuracy, the language model has to be throttled back, and the model of the document, whether part of the functional code, a hand-crafted Markov Model, or somehow learned, has to attend to these issues.
- **German:** The difficulty with German is arbitrary noun compounding, e.g. "Straßenbahnhaltestelle," made from "Straße," "Bahn," "Halt," "Stelle." Note the inserted compounding letters and the dropped capitals. Tesseract manages to get by in German without any special treatment for noun compounding.
- **Hungarian:** Language model complexity is prominent in Hungarian, Polish, and Russian. A simple word-list dictionary is not sufficient for a language that has 1200 variants of the word "Table," with prepositions and other grammatical elements added in combination as prefixes and suffixes to the base word. Tesseract doesn't lean too heavily on its language model, so the simple word-list has remained sufficient for now.
- **Russian:** The most common example from the group of languages that use the Cyrillic alphabet. Severe problems with case errors abound due to upper-case and lower-case looking the same in most of the alphabet. Documents that contain mixed Russian and English, add the difficulty that some Cyrillic letters look exactly like Latin equivalents. Cyrillic lower-case looks just like small-caps in Latin, but Russian also uses small-caps. Tesseract's treatment of small-caps and x-height finding is still an area for improvement.
- **Japanese/Traditional Chinese:** Both introduce the problems of vertical text-lines, large character-set, highly detailed characters with little difference between them, and no space between words. Traditional Chinese has a larger character-set, but Japanese has the additional problem of some of the character-set occurring in two sizes. Since Tesseract's internal representation of a page is entirely vector outlines, a trivial rotation about the origin of blocks of vertical text-lines is sufficient to allow code written for horizontal text-lines to operate on vertical text-lines using negative coordinates. When an individual character is to be classified, it is rotated upright. The classifier works well for Simplified Chinese, but starts to struggle with ambiguities for Traditional Chinese.

- **Hindi:** There are many languages used in India and the surrounding countries, but from an OCR perspective, they can be broken into three groups. A representative of the Northern Indic group, Hindi has the difficulties of a header line running through all the letters in a word, a large set of Grapheme clusters, and a lot of ligatures. Tesseract cuts the header line of Hindi words for the benefit of page layout analysis, but relies on its normal character segmentation system to separate the characters during actual recognition.
- **Kannada/Telugu:** Representatives of the Southern Indic group, and very similar, Kannada and Telugu grapheme clusters are formed from several isolated connected components, which typically change shape when in combination. Some of the individual letters have a very similar appearance.
- **Tamil:** Without such a high combinatorial explosion of grapheme clusters, Tamil is really in a separate category, but has the additional problem of vowels in two components that appear to both the left and right of the consonant. Together with Myanmar, this brings the problem that the sequence on the text-line does not match the sequence of the Unicode representation.
- **Thai/Vietnamese:** Large numbers of stacking diacritics are common in both Thai and Vietnamese. Thai poses significant challenges to page layout analysis because the text-lines tend to be widely spaced, with the diacritics sitting between them.
- **Hebrew:** Hebrew is in this list to highlight the fact that right-to-left writing is not unique to the Arabic family of connected-script languages. Tesseract handles right-to-left with a relatively small code change. In layout analysis, the coordinate space is briefly reflected in the y-axis, so that columns are extracted in the correct reading order (right-to-left). In recognition, *there is no change.* Everything is still processed left-to-right, but the language model is reversed during training. This approach enables Hebrew to be mixed with English and recognized correctly. On output, Hebrew words are reversed, (both letters in the word and words on a line), and English are not, so that down-stream applications see the output text in the correct sequence. In addition, this bi-directional text output requires that some characters, like '(' are stored with their mirror and switched around according to whether the context of the current word is left-to-right or right-to-left, which itself is dependent on the script of the characters in the word, and may be undefined.
- **Arabic:** Well documented, the difficulties with Arabic are mainly the boundaries between letters, and the fact that letters change shape according to their position within a connected component. Farsi and Urdu are more challenging than Standard Arabic, with Urdu in particular having ligatures that are not handled by most computer typesetting systems, which makes training from synthetic data rather difficult. Tesseract currently handles Arabic with an add-on word recognizer, called Cube. Cube uses a convolutional neural network character classifier and a maximal segment-and-classify approach with a beam search. Although Cube does well for Arabic and Hindi, it has disappointing accuracy and speed for the European languages.

Some examples of non-latin scripts are shown in Fig. 5.

Figure 5. Some samples of non-Latin scripts, and some of the languages that use them.

7. THE AD-HOC UNDERBELLY: LANGUAGE MODEL AND WORD RECOGNITION

A severely ad-hoc component of Tesseract is the word recognition module. This module is responsible for searching the segmentation space (of a word into characters), and combining information from the static and adaptive classifiers, together with the language model to form the optimal interpretation of each word. The language model began as a simple state machine that would accept certain sequences of character types, such as upper-case, lower, lower, but not upper, lower, upper.

A word-list was quickly added using a Directed Acyclic Word Graph (DAWG) as a compact representation, but how should a dictionary word be compared to a non-dictionary word? How should two words of different length be compared? If we had not abandoned the pretense that the classifier returns a probability, we could have somehow combined the word frequency with the classifier probability to obtain some overall probability with which to compare candidate words, but even that would have been fraught with ad-hoc hacks. A non-dictionary word would have required some arbitrary out-of-dictionary probability. A 4-letter word would have multiplied 4 not-statistically-independent probabilities together, and might have to be compared to the product of 3 not-statistically-independent probabilities for a 3-letter word.

So in this section is another debate over the value of non-rigorously applied statistics. There is no sound theory of probability and statistics that can be applied rigorously here. Is it more or less principled to abandon the pretense and use machine learning? Although Tesseract's word recognizer is not built around statistics, its principles can be clearly and succinctly stated:

1. Character classifier distances for a word are combined by weighting according to the amount of material in the character – in this case the length of the outline. In a word of n RUs, with classifier distances d_i, and the outline length in each RU of length l_i, the overall word distance (known as rating,) r is:

$$r = \sum_{1}^{n} l_i d_i \quad .$$

   This allows words of different length to be compared fairly without the use of arbitrary constants that would have to be derived from a-priori probabilities that may not be truly constant.

2. There are several word sources, including the top choice word, the dictionaries: system, frequent word, user words, document and number parser. Each word source has a weight. If word source j has weight w_j and produces a word with rating r_j, then the result word is the word from the source given by:

$$argmin(j) w_j r_j \quad .$$

3. The weights w_j should be trained from data.

4. The final result is the word that is found with the minimum weighted rating during segmentation search

Ideally the weights should be trained by machine learning. Prior to the UNLV trial, the weights were optimized using a simple genetic algorithm. Since then the weights have been tuned by hand, but a new machine learning scheme is also being applied now to these weights to improve language-specific accuracy and to allow easy inclusion of new character classifiers.

Back to the issue of whether a system based on statistics could be any better, depends on the language and the quality of input. If the language presents problems for OCR, and/or the input image quality is poor, then a more complex language model and a heavier weight on language model frequency can be of significant benefit. In the case of English with high quality input, the language model has to be used carefully to obtain optimal accuracy, which leads to the final lesson: ***Rigorously applied statistics beats rigorously applied data-driven machine learning beats inappropriate use of statistics beats non-data-driven methods.*** Unfortunately, there are very few circumstances in which statistics can be applied rigorously, and therefore it is difficult to find examples that show the full inequality. Taking thresholding as an example, inappropriate use of statistics (Minimum Error Thresholding) can be worse than non-data-driven methods (fixed threshold at 50% of the gray-scale), and a proprietary thresholding algorithm that used rigorously applied statistical classification beats everything else. Tesseract's accuracy certainly improved after the switch from inappropriate statistics to data-driven machine learning, but as a counterpoint HMM-based systems with their dubious use of statistics do quite well.

8. RESULTS

Tesseract is regularly tested on around 30 languages, using test sets created by various means. Most of the Latin-based languages and Russian have test data that was created from parallel scans of books and PDF text layers. Most of the non-Latin test data was created by humans typing the text from 10 consecutive pages chosen randomly from books scanned under the Google Books project. None of the ground-truth text is perfect, least of all the PDF-originated text, which contains a large number of errors in whitespace (broken and merged words). These errors cause the reported word error rates reported in Table 1 to be significantly higher than would be expected given the character error rate, which excludes added/dropped spaces. Word error rates for Simplified Chinese and Japanese are calculated using a word segmentation system that itself is affected by character errors. The word error-rate for Thai is particularly high due to it not being a space-delimited language, and the word segmentation system is not applied in calculating the word error-rate. Another factor that makes word error rates unexpectedly high is that page segmentation errors are included in the word error rate, but the character error rate excludes these errors.

Table 1. Current error rates on various languages. Columns show the size of the test set in characters and words, and the character-level substitution rate, as well as the word-level error-rate.

Language	No. of Chars (million)	No. of Words (million)	Char error rate (%)	Word error rate (%)
English	271	44	0.47	6.4
Italian	59	10	0.54	5.41
Russian	23	3.5	0.67	5.57
Simplified Chinese	0.25	0.17	2.52	6.29
Hebrew	0.16	0.03	3.2	10.58
Japanese	10	4.1	4.26	18.72
Vietnamese	0.41	0.09	5.06	19.39
Hindi	2.1	0.41	6.43	28.62
Thai	0.19	0.01	21.31	80.53

9. CONCLUSION

Industrial strength, high-accuracy, generic OCR is incredibly difficult to achieve, as there are many components that all have to be world-class to compete with the best commercial systems. Over the last twenty years or so, traditional OCR approaches have gained the label “ad-hoc” as the more statistical systems have attempted to distance themselves by using the label “principled.” This paper has described how Tesseract moved away from statistics explicitly in an attempt to base its classifier on a more rigorous foundation than poorly applied statistics. In some respects this makes Tesseract's “ad-hoc” approach more principled than the “principled” methods, yet the end-result is remarkably similar. The ad-hoc nature of Tesseract and the commercial OCR systems really comes from the list of “long tail” issues in Sec. 6 that have to be dealt with, to achieve competitive accuracy on English. In contrast, statistics-based translation systems have beaten grammar-based translation because of their sheer scalability, but these systems have yet to reach an accuracy level where a long list of exceptions needs to be addressed. Taking these points into consideration, alternative terms for “ad-hoc” and “principled” might be “mature” and “naive.”

Although Tesseract has evolved over time, its direction has been towards the mature traditional architecture: Evolution, not Revolution. Further advances might be achieved through significant changes in the character and/or word classifier. Of the most promising recent (revolutionary) approaches, applying Hidden Markov Models as character classifiers (in addition to their long-established use in language models) have the advantage over conventional machine learning of doing a better job of matching variable length feature descriptions. Deep Belief Nets have the additional advantage of automatically deriving higher-level features (in 2-dimensions, unlike HMMs) from unsupervised training data.

ACKNOWLEDGMENTS

The author would like to thank all those who have ever worked on Tesseract, and contributed so much to the system described herein. Major contributors number about 25, and several more have contributed bug fixes or minor extensions.

REFERENCES

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

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

[3] Kopec, G. E. Chou, P. A., "Document image decoding using Markov source models," *IEEE trans. Pattern Analysis and Machine Intelligence* **16**, 602-617, IEEE (1994).

[4] Schwartz, R. LaPre, C.Makhoul, J. Raphael, C. Zhao, Y., "Language-independent OCR using a continuous speech recognition system," *Proc. 13th int. conf. on Pattern Recognition*, **3,** 99-103, IEEE (1996).

[5] Casey, R. G. Lecolinet, E., "A Survey of Methods and Strategies in Character Segmentation," *IEEE trans. Pattern Analysis and Machine Intelligence* **18**, 690-706, IEEE (1996).

[6] Smith, R., "Hybrid Page Layout Analysis via Tab-Stop Detection," *Proc. of the 10th Int. Conf. on Document Analysis and Recognition,* 241-245, IEEE (2009).

[7] Marsland, S., [*Machine learning: An Algorithmic Perspective*], CRC Press, (2009).

[8] Duda, R. O. Hart, P. E. Stork, D. G., [*Pattern Classification*], Wiley, (2001).

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

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

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

[12] Tesseract open source site: http://code.google.com/p/tesseract-ocr (2012).

[13] Coates, A. Carpenter, B. Case, C. Satheesh, S. Suresh, B. Wang, T. Wu, D. J. Ng, A.Y., "Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning," *Proc. of the 11th Int. Conf. on Document Analysis and Recognition,* 440-445, IEEE (2011).

[14] Kahan, S. Pavlidis, T. Baird, H. S., "On the Recognition of Printed Characters of Any Font and Size," *IEEE Trans. Pattern Analysis and Machine Intelligence,* **9**(2), 274-288, IEEE (1987).

[15] Impedovo, S. Pirlo, G. "Tuning between Exponential Functions and Zones for Membership Functions Selection in Voronoi-Based Zoning for Handwritten Character Recognition," *Proc. of the 11th Int. Conf. on Document Analysis and Recognition,* 997-1001, IEEE (2011).

[16] Ng, A. Y. Jordan, M. I., "On Discriminative vs. Generative classifiers: A comparison of logistic regression and naïve Bayes," *Advances in Neural Information Processing Systems 14: Proc. 2002 Conference,* **2**, 841-848, MIT Press, (2002).

[17] Rabiner, L., "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE,* **77**(2), 257-286, IEEE (1989).

[18] Och, F. J., "Minimum error rate training in statistical machine translation," *Proc. 41st Annual Meeting on Association for Computational Linguistics,* 160-167, ACL (2003).

[19] Kittler, J. Illingworth, J., "Minimum error thresholding," *Pattern Recognition,* **19,** 41-47, Elsevier (1986).

[20] Shi, D. Damper, R. I. Gunn, S. R., "Offline handwritten Chinese character recognition by radical decomposition," *ACM trans. on Asian Language Information Processing (TALIP)* **2**(1), 27-48, (2003).

[21] Smith, R., "Limits on the application of frequency-based language models to OCR," *Proc. of the 11th Int. Conf. on Document Analysis and Recognition,* 538-542, IEEE (2011).

[22] Menasri, F., "Shape-based Alphabet for Off-line Arabic Handwriting Recognition," *Proc. 9th Int. Conf. on Document Analysis and Recognition,* 969-973, IEEE (2007).

[23] Smith, R. Newton, C. Cheatle, P., "Adaptive Thresholding for OCR: A Significant Test," HPL-93-22 HP Laboratories Technical Report, March (1993).