Improving State-of-the-Art OCR through High-Precision Document-Specific Modeling

Andrew Kae[1] Gary Huang[1] Carl Doersch[2] Erik Learned-Miller[1]

[1]Department of Computer Science
University of Massachusetts Amherst
{akae,gbhuang,elm}@cs.umass.edu

[2]Department of Computer Science
Carnegie Mellon University
carl.doersch@gmail.com

Abstract

Optical character recognition (OCR) remains a difficult problem for noisy documents or documents not scanned at high resolution. Many current approaches rely on stored font models that are vulnerable to cases in which the document is noisy or is written in a font dissimilar to the stored fonts. We address these problems by learning character models directly from the document itself, rather than using pre-stored font models. This method has had some success in the past, but we are able to achieve substantial improvement in error reduction through a novel method for creating nearly error-free document-specific training data and building character appearance models from this data.

In particular, we first use the state-of-the-art OCR system Tesseract to produce an initial translation. Then, our method identifies a subset of words that we have high confidence have been recognized correctly and uses this subset to bootstrap document-specific character models. We present theoretical justification that a word in the selected subset is very unlikely to be incorrectly recognized, and empirical results on a data set of difficult historical newspaper scans demonstrating that we make only two errors in 56 documents. We then relax the theoretical constraint in order to create a larger training set, and using document-specific character models generated from this data, we are able to reduce the error over properly segmented characters by 34.1% overall from the initial Tesseract translation.

1. Introduction

Despite claims to the contrary, getting optical character recognition (OCR) systems to consistently obtain very high accuracy rates continues to be a challenging problem, due to difficult cases arising from factors such as noise, low-resolution, and unusual fonts [17]. One existing approach to address this problem is to incorporate *document-specific modeling* [4, 7, 8]. The idea is to specifically model the document being processed by adapting to the fonts, noise model, or lexicon in the document. However, such approaches have had only limited success.

If one had some method for finding a sample of words in a document that were *nearly all* correct, one could effectively use the characters in such words as training data from which to build document-specific font models. Obtaining such a sample of correctly labeled words is not easy, and to our knowledge, there are no existing techniques to perform this task with very high accuracy. There are many methods that could be used to produce lists of words that are mostly correct, but contain some errors. Unfortunately, such lists are not much good as training data for document-specific models since they contain significant numbers of errors, and these errors in training propagate to create more errors later.

In this paper, we present a novel method for identifying a subset of words that are nearly all correctly recognized (with at most one error per document) from an initial translation produced by a third-party OCR system.[1] We give both theoretical justification that our method is highly unlikely to incorrectly identify a word as being correctly recognized, and empirical results for which we make very few errors (less than 1 error per 2000 words chosen). We demonstrate the utility of document-specific modeling when such data is available by dramatically reducing character error on a set of difficult historical newspaper scans.

This is accomplished by using the characters in the identified words as high-precision training data to learn character appearance models for the specific documents. Using these models we can correct errors made in the initial translation, achieving a substantial error reduction of 34.1% over properly segmented characters. Moreover, we show that using training data from a naive approach based on thresholding confidence values leads to a lower error reduction of 9.5%. This demonstrates the utility of our method for iden-

[1]http://code.google.com/p/tesseract-ocr/

tifying training data with very few errors.

We begin in Section 2 with an overview of our approach and present related work. In Section 3, we present the specifics of our method for creating nearly error-free training sets, and give theoretical bounds on the probability of error in these sets in Section 4. We then describe how we use the document-specific model to reduce the error rate in Section 5, and give experimental results in Section 6. Finally, we conclude with directions for future research.

2. Document-Specific Modeling

In this paper, we consider a new approach to obtaining a "clean word list" for use in document-specific modeling. We consider a set of hypotheses about each word in the document where each hypothesis poses that one particular word of the initial OCR system is incorrect. We then search for hypotheses that we can reject with high confidence. More formally, we treat a third party OCR system as a null hypothesis generator, in which each attempted transcription T produced by the OCR system is treated as the basis for a separate null hypothesis. The null hypothesis for word T is simply "*Transcription T is incorrect.*" Letting W be the true identity of a transcription T, we notate this as $T \neq W$.

Our goal is to find as many hypotheses as possible that can be rejected *with high confidence*. In this paper, we take high confidence to mean with fewer than 1 error in a thousand rejected hypotheses. As we mention later, we only make an 2 errors in 4465 words in our clean word lists, even when they come from quite challenging documents.

This process allows us to form high-precision training sets with very few errors, at the cost of lower recall. We argue that this is preferable to the alternative of a larger amount of training data at lower precision. Although some classifiers may be robust to errors in the training data, this will be very dependent on the number of training examples available. For characters such as `'j'` that appear less frequently, having even a few errors may mean that more than half of the training examples are incorrect. While we can tolerate some errors in character classes such as `'e'`, we cannot tolerate them in all classes, as is supported later in our experimental results. Thus, it is essential that our error rate be very low in our clean word list.

2.1. Related Work

There has been significant work done in post-processing the output of an OCR system, although all different from the work we present here. Kukich surveyed various methods to correct words, either in isolation or with context, using natural language processing techniques [13]. Kolak developed a generative model to estimate the true word sequence from noisy OCR output [12]. They assume a generative process that produces words, characters, and word boundaries, in order to model segmentation and character recognition errors of an OCR system.

Our work differs from the work above in that we examine the document images themselves to build document-specific models of the characters. A similar idea was used by Hong and Hull, who examined the inter-word relationships of character patches to help constrain possible interpretations [9]. Our work extends these ideas to produce clean, document-specific training data that can then be used along with more traditional supervised learning methods,

Our work is also related to a variety of approaches that leverage inter-character similarity in documents in order to reduce the dependence upon a priori character models. One method for making use of such information is to treat OCR as a cryptogram decoding problem [2, 16]. After performing character clustering, decoding can be performed by a lexicon-based method or using hidden Markov models [5, 14]. However, such methods are limited by the assumption that characters can be clustered cleanly into pure clusters consisting of only one character. This particular problem can be overcome by solving the decoding problem iteratively [11].

An alternative approach to obtaining document-specific character models is presented in [3] using an iterative algorithm to extract character templates from high confidence regions. One major difference is that we provide a theoretical bound on the number of errors expected using our algorithm to identify highly confident words. Another significant difference is that the authors provide a small amount of manually defined training data in their application, whereas we provide none.

Another method for leveraging inter-character similarity is to perform some type of character clustering, for example to replace individual, potentially degraded character images with a smoothed image over the cluster or to perform nearest-neighbor classification of characters [1, 6].

The inability to attain high confidence in either the identity or equivalence of characters in these papers has hindered their use in subsequent OCR developments. We hope that the high confidence values we obtain will spur the use of these techniques for document-specific modeling.

3. Method for Producing Clean Word Lists

In this section, we present our method for taking a document and the output of an OCR system and producing a so-called *clean word list, i.e.* a list of words which we believe to be correct, with high confidence. Our success will be measured by the number of words that can be produced, and whether we achieve a very low error rate in the clean list. Ideally, we must produce a clean word list which is large enough to provide a sufficient amount of training data for document-specific modeling.

We assume the following setup:

- We are provided with a document D in the form of a grayscale image.
- We are provided with an OCR system.
- We further assume that the OCR system provides an *attempted* segmentation of the document D into words, and that the words are segmented into characters. *It is not necessary that the segmentation be entirely correct, but merely that the system produces an attempted segmentation.*
- In addition to a segmentation of words and characters, the system should produce a best guess for every character it has segmented, and hence, by extension, of every word (or string) it has segmented. Of course, we do not expect all of the characters or words to be correct, as that would make our exercise pointless.
- Using the segmentations provided by the OCR system, we assume we can extract the gray-valued bitmaps representing each guessed character from the original document image.
- Finally, we assume we are given a lexicon. Our method is relatively robust to the choice of lexicon, and assumes there will be a significant number of non-lexicon words in the document.

We define a few terms before proceeding. The *Hamming distance* between two strings of the same number of characters is the number of character substitutions necessary to convert one string to the other. The *Hamming ball* of radius r for a word W, $H_r(W)$, is the set of strings whose Hamming distance to W is less than or equal to r. Later, after defining certain equivalence relationships among highly confusable characters such as 'o' and 'c', we define a *pseudo-Hamming distance* which is equivalent to the Hamming distance except that it ignores substitutions among characters in the same equivalence class. We also use the notions of edit distance, which extends Hamming distance by including joins and splits of characters, and pseudo-edit distance, which is edit distance using the aforementioned equivalence classes.

Our method for identifying words in the clean list has three basic steps. We consider each word T output by the initial OCR system.

1. If T is not in the lexicon, we discard it and make no attempt to classify whether it is correct. That is, we do not put it on the clean word list.[2]
2. Given that T is a lexicon word, we evaluate whether $H_1(T)$ is non-empty, *i.e.* whether there are any lexicon words for which a single change of a letter can produce T. If $H_1(T)$ is non-empty, we discard T and again make no attempt to classify whether it is correct.
3. Assuming we have passed the first two tests, we now perform a *consistency check* (described below) of each character in the word. If the consistency check is passed, we declare the word to be correctly recognized and include it in the clean list.

[2]Why is it not trivial to simply declare any output of an OCR system that is a lexicon word to be highly confident? The reason is that OCR systems frequently use language models to project uncertain words onto nearby lexicon words. For example, suppose the original string was "Rumpledpigskin", and the OCR system, confused by its initial interpretation, projected "Rumpledpigskin" onto the nearest lexicon word "Rumplestiltskin". A declaration that this word is correct would then be wrong. However, our method will not fail in this way because if the true string were in fact "Rumpledpigskin", the character consistency check would never pass. It is for this reason that our method is highly non-trivial, and represents a significant advance in the creation of highly accurate clean word lists.

3.1. Consistency Check

In the following discussion, we refer to the bitmap associated with a character whose identity is unknown as a *glyph*. Let W_j be the true character class of the jth glyph of a word W, and let T_j be the initial OCR system's interpretation of the same glyph. The goal of a consistency check is to ensure that the OCR system's interpretation of a glyph is reliable. We will assess reliability by checking whether other similar-looking glyphs are usually interpreted the same by the OCR system.

To understand the purpose of the consistency check, consider the following situation. Imagine that a document contains a stray mark that does not look like any character at all, but was interpreted by the initial OCR system as a character. If the OCR system thought that the stray mark was a character, it would have to assign it to a character class like 't'. We would like to detect that this character is unreliable. Our scheme for doing this is to find other characters that are similar to this glyph, and to check the identity assigned to those characters by the initial OCR system. If a large majority of those characters are given the same interpretation by the OCR system, then we consider the original character to be reliable. Since it is unlikely that the characters closest to the stray mark are clustered tightly around the true character 't', we hope to detect that the stray mark is atypical, and hence unreliable.

More formally, to test a glyph g for reliability, we first find the M characters in the document that are most similar to g. We then run the following procedure:

1. Initialize i to 1.
2. Record the label of the character that is ith most similar to g. (We use normalized correlation as the similarity measure.)
3. If any character class c has matched g a number of times c_n such that $\frac{c_n}{i+1} > \theta$, then declare the character g to be **θ-dominated** by the class c, and terminate the procedure.
4. Otherwise, add 1 to i. If $i < M$, go to Step 2.
5. If, after the top M most similar characters to g are evaluated and no character class c dominates the glyph, then we declare that the glyph g is **undominated**.

There are three possible outcomes of the consistency check. The first is that the glyph g is dominated by the same class c as the OCR system's interpretation of g, namely T_j. The second outcome is that g is dominated by some other class that does not match T_j. The third outcome is that g is undominated. In the latter two cases, we declare the glyph g to be *unreliable*. The interpretation of glyph g is reliable only if g is dominated by the same class as the original OCR system. Furthermore, a word is included in the clean list only if all of the characters in the word are reliable.

The constants used in our experiments were $M = 20$ and $\theta = 0.66$. That is, we compared each glyph against a maximum of 20 other glyphs in our reliability check, and we insisted that a "smoothed" estimate of the number of similarly interpreted glyphs was at least 0.66% before declaring a character to be reliable. We now analyze the probability of making an error in the clean list.

4. Theoretical Bound

The goal of this section is to show that, under certain simple assumptions that hold for a very large class of documents, our method for producing clean lists has a very low probability of error. This formal bound on the error probability gives us some confidence that such a method will work not just for the documents tested in this paper but for a much larger class of documents.

The following is a condensed version of our analysis. More complete derivations can be found in the technical report [10]. Let C be the event that the word passed the consistency check. When w_t is a lexicon word and has an empty Hamming ball of size 1, we want to upper bound

$$\begin{aligned}
&\Pr(W \neq w_t | T = w_t, C) \\
&\quad = \sum_{w \neq w_t} \Pr(W = w | T = w_t, C) \\
&\quad = \sum_{\substack{w \neq w_t \\ w \in \text{Dict}, |w| = |w_t|}} \Pr(W = w | T = w_t, C) \\
&\qquad + \sum_{\substack{w \neq w_t \\ w \in \text{Dict}, |w| \neq |w_t|}} \Pr(W = w | T = w_t, C) \\
&\qquad + \sum_{\substack{w \neq w_t \\ w \notin \text{Dict}}} \Pr(W = w | T = w_t, C).
\end{aligned}$$

Let ϵ be an upper bound on the probability that noise has caused a character of any given class to look like it belongs to another specific class other than its own class. The probability of the consistency check giving a label c_2 when the true underlying label is c_1 is then less than 2ϵ, for any classes c_1, c_2. Let δ be a lower bound on the probability that a character consistency check will succeed if the OCR system's label of the character matches the ground truth label.

Let D_i be the number of lexicon words of pseudo-Hamming distance i away from w_t. Let r_D be the rate of growth of D_i as a function of i, *e.g.* $D_{i+2} = r_D^i D_2$. Assume, since $\epsilon \ll 1$, that $r_D(\frac{2\epsilon}{\delta}) < \frac{1}{2}$. Similarly, let E_i be the number of lexicon words w with a pseudo-edit distance i away from w_t and $|w| \neq |w_t|$, and also assume that r_E, the rate of growth of E_i, satisfies $r_E(\frac{2\epsilon}{\delta}) < \frac{1}{2}$.

Let N_i be the set of non-lexicon words with a pseudo-edit distance i from w_t, and let $p_i = \frac{\Pr(T=w_t|W\in N_i)\Pr(W\in N_i)}{\Pr(T=w_t|W=w_t)\Pr(W=w_t)}$. Assume the rate of growth of r_N of p_i satisfies $r_N(\frac{2\epsilon}{\delta}) < \frac{1}{2}$. We will make the assumption that the individual character consistency checks in $\Pr(C|T = w_t, W = w)$ are independent, although this is not exactly true.

We will make use of the following inequality:

$$\begin{aligned}
&\Pr(W = w | T = w_t, C) \\
&\quad \leq \frac{\Pr(C | T = w_t, W = w)}{\Pr(C | T = w_t, W = w_t)} \\
&\quad \times \frac{\Pr(T = w_t | W = w) \Pr(W = w)}{\Pr(T = w_t | W = w_t) \Pr(W = w_t)}. \qquad (1)
\end{aligned}$$

For lexicon words w, we will assume that

$$\begin{aligned}
&\frac{\Pr(T = w_t | W = w) \Pr(W = w)}{\Pr(T = w_t | W = w_t) \Pr(W = w_t)} \\
&\quad = \frac{\Pr(W = w | T = w_t)}{\Pr(W = w_t | T = w_t)} < 1. \qquad (2)
\end{aligned}$$

Bounding lexicon words. Applying Eq. 2 to Eq. 1, we get

$$\begin{aligned}
&\Pr(W = w | T = w_t, C) \\
&\quad \leq \frac{\Pr(C | T = w_t, W = w)}{\Pr(C | T = w_t, W = w_t)}. \qquad (3)
\end{aligned}$$

For a word w that is a pseudo-Hamming distance i from w_t, we can simplify Eq. 3 as $\Pr(W = w | T = w_t, C) \leq \frac{(2\epsilon)^i}{\delta^i}$ and bound the sum over all Hamming distance words as

$$\begin{aligned}
&\sum_{w \neq w_t, w \in \text{Dict}, |w| = |w_t|} \Pr(W = w | T = w_t, C) \\
&\quad \leq \sum_{i=2} D_i \frac{(2\epsilon)^i}{\delta^i} \leq 8 D_2 \frac{\epsilon^2}{\delta^2}.
\end{aligned}$$

For a word w that is a pseudo-edit distance i from w_t, we can simplify Eq. 3 as $\Pr(W = w | T = w_t, C) \leq \frac{(2\epsilon)^{i+1}}{\delta^i}$ and bound the sum over all such edit distance words as

$$\begin{aligned}
&\sum_{w \neq w_t, w \in \text{Dict}, |w| \neq |w_t|} \Pr(W = w | T = w_t, C) \\
&\quad \leq \sum_{i=1} E_i \frac{(2\epsilon)^{i+1}}{\delta^i} \leq 8 E_1 \frac{\epsilon^2}{\delta^2}.
\end{aligned}$$

Bounding non-lexicon words. Rearranging Eq. 1 and summing over all non-lexicon words,

$$\sum_{w \neq w_t, w \notin \text{Dict}} \Pr(W = w | T = w_t, C)$$
$$\leq \sum_{i=1} \sum_{w \in N_i} \frac{(2\epsilon)^i}{\delta^i} \frac{\Pr(W = w | T = w_t)}{\Pr(W = w_t | T = w_t)}$$
$$= \sum_{i=1} \frac{(2\epsilon)^i}{\delta^i} p_i \quad \leq \quad 4p_1 \frac{\epsilon}{\delta^2}.$$

Final bound. This gives us a final error probability of

$$\Pr(W \neq w_t | T = w_t) \leq \frac{(8D_2 + 8E_1)\epsilon^2 + 4p_1\epsilon}{\delta^2}.$$

For $\epsilon < 10^{-3}$, $8D_2 + 8E_1 < 10^2$, $4p_1 < 10^{-1}$, $\delta^2 > 10^{-1}$,

$$\Pr(W \neq w_t | T = w_t) \leq 2 \cdot 10^{-3}.$$

The bounds for the constants chosen above were selected conservatively to hold for a large range of documents, from very clean to moderately noisy. Not all documents will necessarily satisfy these bounds. In a sense, these inequalities define the set of documents for which our algorithm is expected to work, and for heavily degraded documents that fall outside this set, the character consistency checks may no longer be robust enough to guarantee a very low probability of error.

Our final bound on the probability of error, 0.002, is the result of a *worst case analysis* under our assumptions. If our assumptions hold, the probability of error will likely be much lower for the following reasons. For most pairs of letters, $\epsilon = 10^{-3}$ is not a tight upper bound. The quantity on the right of Eq. 2 is typically much lower than 1. The rate of growths r_D, r_E, r_N are typically much lower than assumed. The bound on p_1, the non-lexicon word probabilities, is not a tight upper bound, as non-lexicon words mislabeled as lexicon words are rare. Finally, the number of Hamming and edit distance neighbors D_2 and E_1 will typically be less than assumed.

On the other hand, for sufficiently noisy documents, and certain types of errors, our assumptions do not hold. Some of the problematic cases include the following. As discussed, the assumption that the individual character consistency checks are independent is not true. If a document is degraded or has a font such that one letter is consistently interpreted as another, then that error will likely pass the consistency check (*e.g.* ϵ will be very large). If a document is degraded or is very short, then δ may be much smaller than $10^{-\frac{1}{2}}$. (The character consistency check requires a character to match to at least a certain number of other similarly labeled characters, so, for example, if that number isn't present in the document to begin with, then the check will fail with certainty.) Finally, if the lexicon is not appropriate for the document then $4p_1 < 10^{-1}$ may not hold. This problem is compounded if the initial OCR system projects to lexicon words.

5. Character Recognition

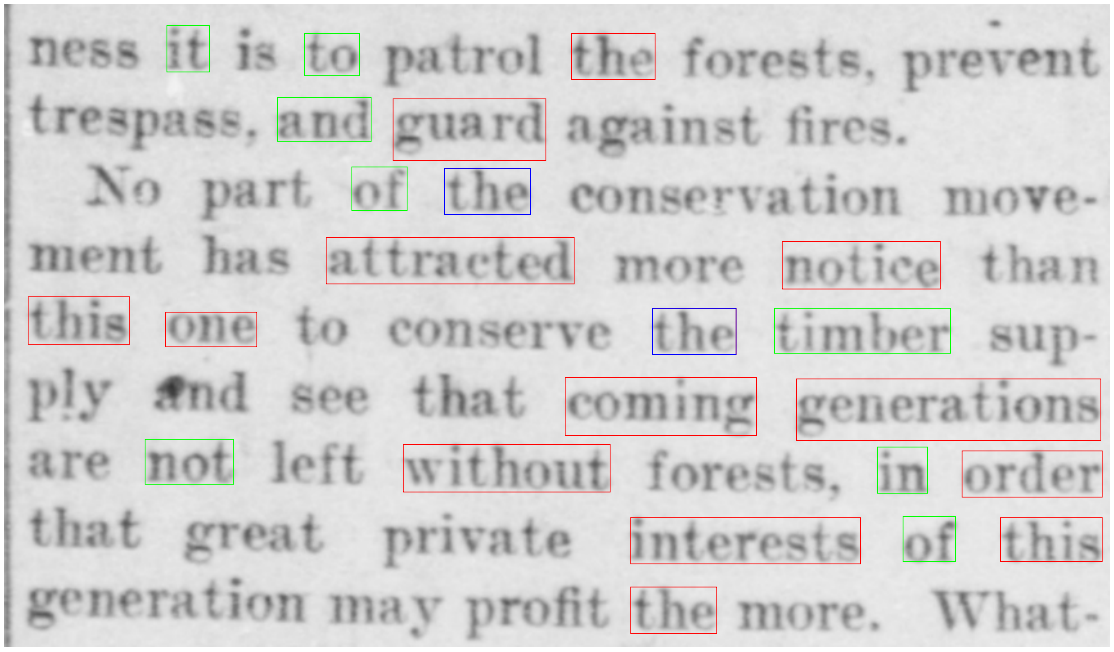
Using the clean list generated from the approach above, we build document-specific character models using SIFT features [15]. We refer to our algorithm as SIFT_Align. We use the traditional SIFT descriptor without applying the Gaussian weighting because we did not want to weight the center of an image more highly than the rest. In addition, we fix the scale to be 1 and orientation to be 0 at all times. The SIFT_Align procedure is presented below:

1. Compute the SIFT descriptor for each character image in the clean list, at the center of the image.
2. Compute the component-wise arithmetic mean of all SIFT descriptors for each character class in the clean list. These mean descriptors are the "representations" (or character models) of the respective classes.
3. For each character image in the clean list, compute a SIFT descriptor for each point in a window in the center of the image (we use a 5x5 window) and select the descriptor with smallest L2 distance to the mean SIFT descriptor for this character class. This aligns each character's descriptor to the mean class descriptor.
4. Test images are all character images *not* in the clean list (since we don't want to test on images we trained on). In addition, test images must be labeled as one of the clean list character classes by the initial OCR system, and actually be one of the clean list characters.[3] Align each test image as in the previous step, except select the descriptor with smallest L2 distance to *any* of the mean descriptors. This aligned descriptor is the final descriptor for the test image.
5. Pass the SIFT descriptors for the training/test images found in the previous steps to a multiclass SVM with a C value of 5,000,000.

We use the $SVM^{multiclass}$ implementation[4] of multiclass SVM [18] and use a high C value of 5,000,000, which was selected through cross-validation. This makes sense since we generally do not have many instances of each character class in the clean list, and so we want a minimum of slack, which a high C value enforces.

[3]Note that if a character is labeled as a clean list character class but its true label is actually not one of these classes, then it must be incorrectly labeled by the initial OCR system and and so our approach can do no worse by excluding it from the test set.

[4]`http://svmlight.joachims.org/`



ness it is to patrol the forests, prevent
trespass, and guard against fires.
No part of the conservation move-
ment has attracted more notice than
this one to conserve the timber sup-
ply and see that coming generations
are not left without forests, in order
that great private interests of this
generation may profit the more. What-

Figure 1. Red boxes indicate clean list words. Green boxes indicate Tesseract's confident word list. Blue boxes indicate words in both lists. "timber" is incorrectly recognized by OCR system as "timhcr". All other words in boxes are correctly translated. (Best viewed in color)

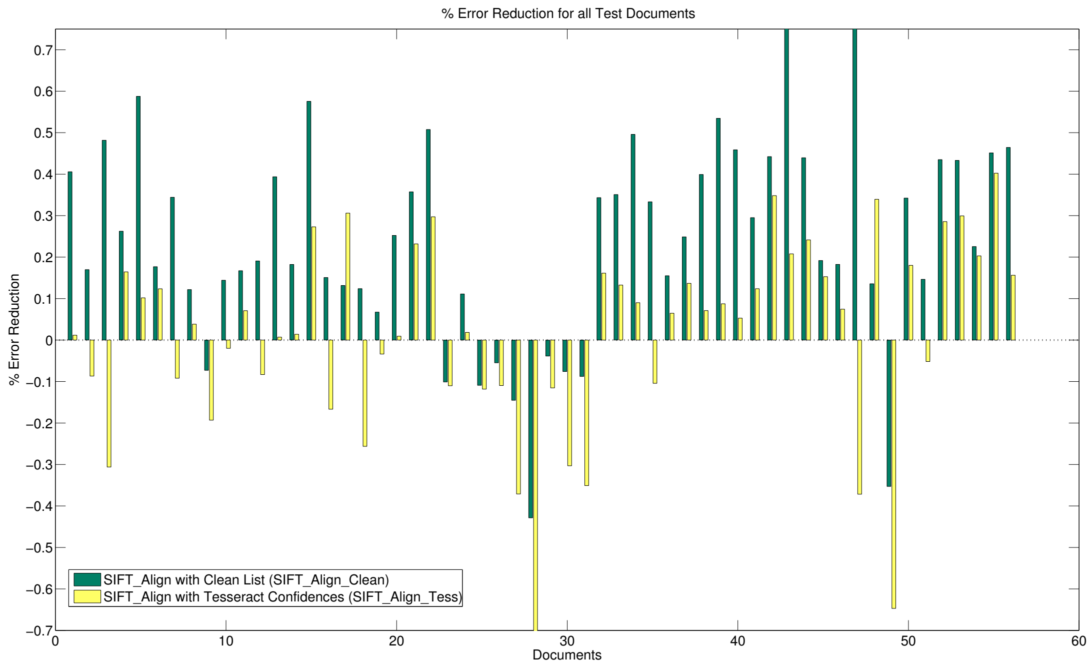
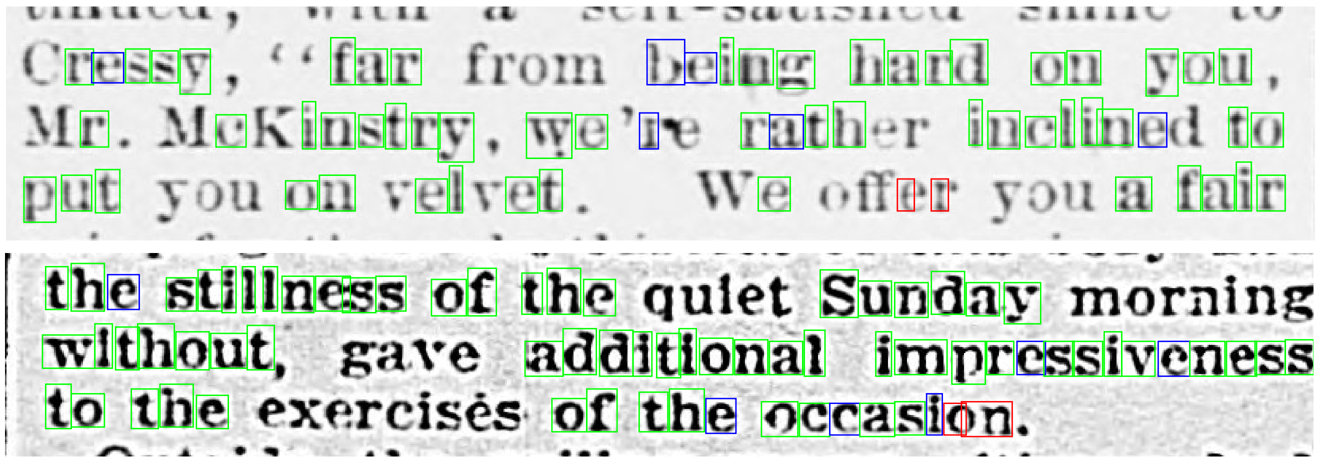


Figure 2. Character error reduction rates for SIFT_Align using the clean list (SIFT_Align_Clean) and Tesseract's confident word list (SIFT_Align_Tess) on the test sets of 56 documents. SIFT_Align_Clean increases error in 10 documents whereas SIFT_Align_Tess increases error in 21 documents.


Cressy, ''far from being hard on you,
Mr. McKinstry, we're rather inclined to
put you on velvet. We offer you a fair

the stillness of the quiet Sunday morning
without, gave additional impressiveness
to the exercises of the occasion.

Figure 3. Sample of results from two documents. A green box indicates both the initial OCR system (Tesseract) and SIFT_Align correctly classified the character. A red box indicates both systems misclassified the character, and a blue box indicates that SIFT_Align classified the character correctly and Tesseract misclassified it. In this example, there are no cases shown where Tesseract correctly classified a character and SIFT_Align misclassifies it. (Best viewed in color)

6. Experiments

We experimented with two sets of documents. The first set consists of 10 documents from the JSTOR archive[5] and Project Gutenberg[6]. This initial set of documents was used to evaluate our clean list generation algorithm and develop our algorithm for producing character models from the clean lists [10]. In this work, our clean list results selected an average of 6% of the words from each document. *These clean list did not contain a single error*, *e.g*. the precision of our clean list was 100%. This strongly supports our theoretical bounds established in Section 4.

The second set of documents, used for performance evaluation of the SIFT_Align algorithm, are 56 documents taken from the Chronicling America[7] archive of historical newspapers. Since our initial OCR system (Tesseract) can only accept blocks of text and does not perform layout analysis, we manually cropped out single columns of text from these newspaper pages. Other than cropping and converting to TIFF for Tesseract, the documents were not modified in any way. There are on average 1204 words per document. The clean list contains 2 errors out of a total of 4465 words, within the theoretical bound of .002 mentioned earlier.

In an effort to increase the size of the clean lists beyond 6% per document, we experimented with relaxing some of the criteria used to select the clean lists. In particular, we allowed the Hamming ball of radius 1 for a word to be non-empty as long as the words within the ball did not appear within the document. By making this small change, we were able to increase the size of the clean lists to an average of 18% per document while introducing at most one error per document. We refer to the original clean lists as *conservative clean lists* and to the modified, larger, and slightly less accurate clean lists as *aggressive clean lists*. We decided to use the aggressive clean lists for our experiments because they contain few errors and there are more character instances. From this point, our use of "clean list" refers to the aggressive clean list.

We then ran Tesseract on all documents, obtaining character bounding boxes[8] and guesses for each character. Next, we used Mechanical Turk[9] to label all character bounding boxes to produce a ground truth labeling. We instructed annotators to only label bounding boxes for which a single character is clearly visible. Other cases (multiple characters in the bounding box or a partial character) were discarded.

After the initial OCR system was used to make an initial pass at each document, the clean list for that document was extracted. Character recognition was then performed as in Section 5 on any characters from the classes defined by the clean list. Even though many of the characters were already recognized correctly by OCR system, our approach improves the recognition to produce an even higher accuracy than the original OCR system's accuracy, on average. As shown in the next section, in most cases, this resulted in correcting the classifications of a significant portion of the characters in the documents.

7. Results

In Figure 1, we show a portion of a document and the corresponding subset of clean list words (generated by our process) and highly confident Tesseract words within this

[5] `http://www.jstor.org`
[6] `http://www.gutenberg.org/`
[7] `http://chroniclingamerica.loc.gov/`

[8] This feature is not available out of the box; we edited the source code.
[9] `https://www.mturk.com/mturk/welcome`

portion. In this example, the only mistranslated word by Tesseract is “timber” in the latter set, while our clean list does not have errors.

In order to judge the effectiveness of using our clean list, we also generated another confident word list using Tesseract’s own measure of confidence.[10] To generate the confident word list, we sort Tesseract’s recognized words by certainty and take the top n words that result in the same number of characters as our clean list. We refer to the SIFT_Align algorithm using our clean list as SIFT_Align_Clean and the SIFT_Align algorithm using Tesseract’s confidences as SIFT_Align_Tess.

In Figure 2, we show the character error reduction rates for both SIFT_Align_Clean and SIFT_Align_Tess. In 46 of the 56 documents, SIFT_Align_Clean results in a reduction of errors whereas SIFT_Align_Tess reduces error in 35 documents. Note this figure shows percent error reduction, not the raw number of errors. SIFT_Align_Clean made a total of 2487 character errors (44.4 errors per document) on the test set compared to 7745 errors (138.3 errors per document) originally made by Tesseract on those same characters. For the 10 cases where SIFT_Align_Clean increased error, SIFT_Align_Clean made 356 character errors and Tesseract made 263 errors. Thus, overall, the error reductions achieved by SIFT_Align_Clean were much greater than the errors introduced.

SIFT_Align_Clean outperforms Sift_Align_Tess. Average error reduction for SIFT_Align_Clean is 34.1% compared to 9.5% for Sift_Align_Tess. Error reduction is calculated as $(TT - ST)/TD$ where TT is # Tesseract errors in the test set, ST is # SIFT_Align errors in the test set and TD is # Tesseract errors in the document. SIFT_Align_Clean also reduces the character error in more documents than does Sift_Align_Tess.

Our test cases only consider properly segmented characters which account for about half of all the errors in these documents. The error reduction for SIFT_Align_Clean over all characters (segmented properly or not) is 20.3%.

We are able to achieve these significant reductions in average character error, for properly segmented characters, using only simple appearance-based character recognition techniques with the clean lists. We believe that further improvements can be achieved by using the clean lists in conjunction with more sophisticated models, such as document-specific language models, as suggested by [19]. In addition, we believe that the clean lists can also be used to re-segment and fix the large percentage of initial errors that result from incorrect character segmentation.

Acknowledgements. Supported by NSF Grant IIS-0916555.

[10]There are two measures of word confidence in Tesseract: rating and certainty, of which we use certainty.

References

[1] T. Breuel. Character recognition by adaptive statistical similarity. In *International Conference on Document Analysis and Recognition*, 2003.

[2] R. Casey. Text OCR by solving a cryptogram. In *International Conference on Pattern Recognition*, 1986.

[3] J. Edwards and D. Forsyth. Searching for character models. In *Neural Information Processing Systems*, 2005.

[4] T. K. Ho. Bootstrapping text recognition from stop words. In *International Conference on Pattern Recognition*, 1998.

[5] T. K. Ho and G. Nagy. OCR with no shape training. In *International Conference on Pattern Recognition*, 2000.

[6] J. Hobby and T. Ho. Enhancing degraded document images via bitmap clustering and averaging. In *International Conference on Document Analysis and Recognition*, 1997.

[7] T. Hong and J. Hull. Character segmentation using visual inter-word constraints in a text page. In *Proceedings of SPIE (International Society for Optics and Photonics)*, 1995.

[8] T. Hong and J. Hull. Improving OCR performance with word image equivalence. In *Symposium on Document Analysis and Information Retrieval*, 1995.

[9] T. Hong and J. J. Hull. Visual inter-word relations and their use in OCR post-processing. In *International Conference on Document Analysis and Recognition*, 1995.

[10] A. Kae, G. Huang, and E. Learned-Miller. Bounding the probability of error for high precision recognition. Technical Report UM-CS-2009-031, University of Massachusetts Amherst, 2009.

[11] A. Kae and E. Learned-Miller. Learning on the fly: Font free approaches to difficult OCR problems. In *International Conference on Document Analysis and Recognition*, 2009.

[12] O. Kolak. A generative probabilistic ocr model for NLP applications. In *North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2003.

[13] K. Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[14] D. Lee. Substitution deciphering based on HMMs with applications to compressed document processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12), 2002.

[15] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[16] G. Nagy. Efficient algorithms to decode substitution ciphers with applications to OCR. In *International Conference on Pattern Recognition*, 1986.

[17] G. Nagy. Twenty years of document image analysis in PAMI. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1), 2000.

[18] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning*, 2004.

[19] M. Wick, M. Ross, and E. Learned-Miller. Context-sensitive error correction: Using topic models to improve OCR. In *International Conference on Document Analysis and Recognition*, 2007.