2011

Enforcing similarity constraints with integer programming for better scene text recognition

David L. Smith
Jacqueline Feild
Erik G Learned-Miller, *University of Massachusetts - Amherst*

Enforcing Similarity Constraints with Integer Programming for Better Scene Text Recognition

David L. Smith Jacqueline Feild Erik Learned-Miller
Department of Computer Science
University of Massachusetts Amherst
Amherst MA, 01003
{dlsmith,jfeild,elm}@cs.umass.edu

Abstract

The recognition of text in everyday scenes is made difficult by viewing conditions, unusual fonts, and lack of linguistic context. Most methods integrate a priori appearance information and some sort of hard or soft constraint on the allowable strings. Weinman and Learned-Miller [14] showed that the similarity among characters, as a supplement to the appearance of the characters with respect to a model, could be used to improve scene text recognition. In this work, we make further improvements to scene text recognition by taking a novel approach to the incorporation of similarity. In particular, we train a ***similarity expert*** *that learns to classify each pair of characters as equivalent or not. After removing logical inconsistencies in an equivalence graph, we formulate the search for the maximum likelihood interpretation of a sign as an integer program. We incorporate the equivalence information as constraints in the integer program and build an optimization criterion out of appearance features and character bigrams. Finally, we take the optimal solution from the integer program, and compare all "nearby" solutions using a probability model for strings derived from search engine queries. We demonstrate word error reductions of more than 30% relative to previous methods on the same data set.*

1. Introduction

Scene text recognition, the recognition of arbitrary text in the environment, such as grocery carton labels, license plates, and business placards, is a surprisingly difficult problem. Relative to the more structured recognition of text from machine-printed documents, known as optical character recognition (OCR), scene text recognition is more difficult for several reasons. First, viewing angle and lighting are typically not carefully controlled. Second, there is usually very little linguistic context, making it more difficult to apply sophisticated language models. Third, since most signage and other environmental text is just a few words in length, it is difficult to benefit from repeated occurrences of the same symbol to help decode a letter. For example, in a long document, the most common symbol is usually the letter "e", but in a road sign, any letter could be most common. This makes it difficult to use cryptographic techniques [13, 3, 2, 4, 18]. Finally, many signs are meant to be eye-catching, and are for this reason designed with stylized, whimsical, or otherwise unusual fonts for which we are unlikely to have matching training data. In other words, the scene text recognition task frequently requires interpreting versions of letters and digits that are significantly different than those seen in training. All of these factors collude to help this problem stubbornly resist resolution.

Despite its difficulty, there have been many approaches developed for scene text recognition [8, 22, 9, 23, 24, 19, 7]. We believe it is critical to use as many sources of information as possible, and to integrate them in a unified decision process (see, e.g. [23]). Two obvious sources of information are prior models of the appearance of each character, and prior models of the likelihood of each character string. These models can be learned using traditional vision techniques (for the appearance models), and statistical language modeling techniques, such as letter bigram models, letter trigram models, frequency-weighted dictionaries, or some combination of these techniques.

Another less obvious source of information is found in the similarity and dissimilarity between pairs of characters. As discussed by Weinman and Learned-Miller [14], a traditionally built scene text recognition system using only the two *a priori* sources of information above may well conclude that two characters which have nearly identical appearance have different labels. When there is high ambiguity in the label due to appearance, language information may "tip" the label for one character toward one letter and "tip" the other character toward another letter. A person

Figure 1. An example sign from the data set. Due to the specialized font, the character 'A' is particularly difficult to recognize.

will usually conclude, however, that two characters which appear highly similar must represent the same letter within a particular sign. This comes from the knowledge that while characters may vary in appearance across fonts, most signs use a consistent or small set of fonts, and within a single font, we expect consistency of character appearance.

1.1. Prior work in similarity

Weinman et al. [14, 23] successfully incorporated similarity for the scene text recognition problem in the following way. First, a raw similarity score was computed for each pair of characters A and B by computing

$$1 - f_A \cdot f_B, \tag{1}$$

where f_A and f_B were unit feature vectors for the characters A and B. This yields a number between 0 (when the vectors are identical) and 2 (when the vectors point in opposite directions) which is then put through a learned monotonic non-linearity. This similarity score was then used to define factors between each pair of characters in a factor graph, and integrated into a general belief propagation framework using other types of appearance and language information.

While the similarity score in Eq. (1) encodes some important information about similarity, our hypothesis was that by learning a more flexible function that was specialized to evaluate the similarity of two characters, we could develop a better score, and that this score would lead to better recognition rates. Recent work in the face verification problem [11, 15] has shown relatively high accuracy rates in determining whether two faces are of the same person or not, a problem much more difficult than determining whether two characters represent the same letter in the same font under natural viewing conditions.

This motivated us to return to the use of similarity in scene text recognition and see if we could do more. We started by assuming we had an oracle to tell us, for each pair of characters in a sign, whether they had the same label. We would then refuse to accept solutions that violated the constraints of the oracle. We viewed this as an upper bound on the benefit for word accuracy that we could get from estimated similarity information. Surprisingly, we were able to achieve nearly the same result by estimating similarity as by using the oracle-based similarity.

The system we used to achieve this result was designed as follows.

1. First, we trained a *similarity expert* to classify pairs of characters as same or different.

2. We then formulated the scene text recognition problem as an integer program. The optimization criterion of the integer program is a log likelihood derived from appearance features for each character class and bigram probabilities governing the likelihood of character transitions. The same or different classifications from the similarity expert are used as additional constraints in the integer program.

3. After running the integer program to find an approximate solution, we use a search engine to define an implicit probability distribution over strings that are close to this solution, and reevaluate these strings using a separate model.

These techniques together lead to a significant improvement in performance on a data set that has previously been used to study the scene text recognition problem.

The remainder of the paper is structured as follows. In Section 2, we discuss our method for evaluating the similarity of character pairs. In Section 3, we discuss our formulation of the scene text recognition problem as an integer program, including the criterion of optimization, and using equivalence and differences of character pairs as constraints in the integer program. In Section 4, we present a method of using a search engine to correct some errors in the signs in our database, similar to the work of others [20, 10]. In Section 5 we detail our experiments, and present results. We conclude with a discussion in Section 6 about how the techniques used in this work might be combined with methods used previously by other authors.

2. Similarity

Figure 1 shows one of the signs used in our experiments. Since it is unreasonable to expect a pre-trained character recognizer to correctly identify the triangles as capital As, it is very helpful to at least establish the equivalence of these characters. In that case, a search over a single label for both characters that satisfies reasonable case transitions, bigram probabilities, and language constraints is much more likely to be fruitful. Toward this end, it is important to have an accurate and robust way of identifying whether characters are equivalent (i.e. whether they have the same label) or not.

Our goal is to create a classifier that takes two character images as input and predicts whether the characters should have the same label. As discussed in Section 3.2, we choose to use an integer program to perform our search over possible interpretations of a sign. We incorporate similarity information via hard constraints in our integer program. We can contrast this with previous approaches that used belief propagation and "soft" similarity information. In general,

solving an integer program is an NP-hard problem, but in practice our problem instances are small enough that we can solve them efficiently. This allows us to guarantee a solution that is optimal with respect to adjustable numerical tolerances (discussed in Section 3.3). On the other hand, loopy belief propagation is an approximate inference technique that is neither guaranteed to converge, nor to find the optimal solution. Our use of "hard" equivalence information could be viewed as a weakness, in that it does not allow for poor estimates of equivalence to be overcome. We have two methods for recovering from incorrect classifications of equivalence, which are discussed in Sections 3.4 and 4. Since we can recover from errors, the disadvantage of using a "hard" decision is mitigated.

We developed our general approach using a subset of the ICDAR 2003 Robust OCR Dataset[1] [17] as training data. This allowed us to explore a wide range of methods for classifying equivalence and to experiment with parameter values. In our actual tests, we retrained the best methods on portions of our evaluation set using a cross-validation scheme. This is discussed in more detail in Section 5.

To ensure that the distribution of image characteristics in the ICDAR data was similar to our evaluation test set, we created a subset by removing images with significantly lower resolution or lower contrast than the poorest quality images in our test set.

2.1. Building a similarity expert

Our equivalence classifier is a support vector machine (SVM) [5, 6, 12] trained on a set of feature vectors extracted from pairs of character images. Each pair of characters is labeled as equivalent or different.

The features of the character images are formed using the following process. For each pair of characters, we standardize their size by taking the maximum of the width and height of both images. We then resize both images to be square images of this size. Next, we normalize the intensities of each image by scaling them to fill the range of the image, ignoring a small percentage from both extremes to account for outliers. We then extract one SIFT descriptor [16, 21] from each image by placing it in the center of the image, scaled to cover the entire image size.

We form two feature vectors for each pair of images. One is created by subtracting the SIFT descriptor of the first image from the second and the other is created by subtracting the SIFT descriptor of the second image from the first. We append the ratios of the original image widths and the original image heights to both difference descriptors. We add these additional features because they are good predictors of dissimilarity. If two images vary significantly in their original size, then they are more likely to have different labels than two images with similar sizes.

[1] http://algoval.essex.ac.uk/icdar/Datasets.html

(a) Equivalent image pairs that are classified as different. Problems giving difficulty to the similarity classifier include 3-dimensional layering effects (left), perspective distortion (center), and lighting effects (right).

(b) Different image pairs that are classified as equivalent. The leftmost pair represents a capital S and a lowercase s, which are considered to be different according to our evaluation criterion.

Figure 2. Examples of classification errors made by the similarity classifier using four SIFT descriptors.

For each pair of training images A and B, we then end up with two training sample vectors: $f(A, B)$ and $f(B, A)$, where f is the augmented SIFT vector described above. While we use both feature vectors in training, at test time we use only the first to represent a pair of images.

We also create an alternate version of this classifier by using the same process, except extracting a two by two non-overlapping grid of four SIFT descriptors from each image rather than a single SIFT descriptor.

We use five-fold cross validation to evaluate this classifier on the test data (which is further described in Section 5).[2] We extract all pairs of same and different images originating from the same sign from the data set and divide them into five groups, making sure that all pairs from the same sign are in the same group. We train an SVM with a quadratic kernel using four of the folds as the training set and one fold as the test set.

The result is a classification for each of the 10,290 pairs of similar and dissimilar characters in the test set. Using the classifier with one SIFT descriptor we correctly classify 10,215 pairs for an accuracy of 99.27%. Using the classifier with four SIFT descriptors, we correctly classify 10,230 pairs for an accuracy of 99.42%. Examples of classification errors made by the classifier using four SIFT descriptors are shown in Figure 2.

While we achieve over 99% accuracy for our equivalence classification for both types of features. This result is not quite as great as it sounds. In particular, for each string of words with k characters in our sign database, there

[2] One must take care in using cross-validation methods on an evaluation data set not to "cheat" by adapting parameters through multiple runs or using multiple runs on the test data to evaluate a wide variety of different algorithms. Put simply, it is important that no element of the evaluation data be used to set a parameter which is ultimately used in its own evaluation. We assiduously avoid this situation. In other words, we do not "train on the test data".

are $\mathcal{O}(k^2)$ similarity comparisons. Since we are making hard decisions about equivalence, an error in any one of these $\mathcal{O}(k^2)$ equivalence determinations would result in at least one incorrect word coming out of our integer program. While we mitigate this problem to some extent by eliminating equivalence constraints which are inconsistent with each other (discussed in Section 3.4), our IP accuracy is still highly sensitive to errors in equivalence determination. Thus, the error rate needs to be extremely low for this source of information to be helpful. As we shall see, the similarity information, even with some errors, does indeed improve the accuracy of our integer program (see Table 1). With our second processing step using search engine correction, it proves even more helpful.

3. Maximum Posterior Probability Via Integer Programs

Like many other authors, we take a probabilistic approach to recognition, finding the interpretation of a sign that maximizes the conditional probability of the labels given the observations. Given a set of N character image observations $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$, our task in recognition is to assign the best set of labels $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$ for these characters subject to a set of consistent equivalence and difference constraints $\mathcal{C}$. That is, we want to compute

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}}\, p(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmax}}\, p(\mathbf{y}, \mathbf{x}) \tag{2}$$

subject to $\mathcal{C}$. For our initial IP evaluation, we assume a Markov model over the labels,[3] leading us to express $p(\mathbf{y}, \mathbf{x})$ as[4]

$$p(\mathbf{y}, \mathbf{x}) = \prod_{i=1}^{N} p(x_i|y_i) \prod_{i=1}^{N} p(y_i|y_{i-1}). \tag{3}$$

Rather than maximizing Eq. 3, we can equivalently minimize its negative log. Thus,

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmin}}\; -\sum_{i=1}^{N} \log p(x_i|y_i) - \sum_{i=1}^{N} \log p(y_i|y_{i-1}). \tag{4}$$

Let $\mathcal{A}$ denote our alphabet. For simplicity, let $\phi_{i:j} = -\log p(x_i|y_i = \mathcal{A}_j)$, the negative log probability that character i takes on the label $\mathcal{A}_j$. Similarly we will let $\phi_{i(i+1):jk} = -\log p(y_{i+1} = \mathcal{A}_k|y_i = \mathcal{A}_j)$, the negative log probability that character i takes on the label $\mathcal{A}_j$ and character $i+1$ takes on the label $\mathcal{A}_k$. Using this notation, we have

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmin}} \sum_{i=1}^{N} \phi_{i:j} + \sum_{i=1}^{N-1} \phi_{i(i+1):jk}. \tag{5}$$

3.1. Features

Our appearance features were developed by training class conditional weights over a vector of edge-like features. The weights were trained to maximize the classification performance on a set of synthetic fonts. These appearance features were given to us precomputed from the authors of [14] with minimal modifications.

Our pairwise language features combine bigram and case transition statistics. The bigram statistics were trained on a selection of books from Project Gutenberg. Inter-word case change statistics (i.e. changing from upper to lower or lower to upper) were trained on the press-related sections of the Brown Corpus of American English.[5] For the first transition, we assume a uniform probability of transitioning from upper case to lower case and upper case to upper case. We now show how to express the optimization in Eq. (2) as an integer program.

3.2. Integer program formalization

An integer program (IP) is an optimization problem of a linear objective function over integer-valued variables $\mathbf{y}$, where the space of solutions is bounded by a set of linear constraints. The goal is to find the assignment to these variables that minimizes the objective function. An IP in standard form [1] is written

$$\text{minimize } \mathbf{c}^T\mathbf{y} \tag{6}$$
$$\text{subject to } \mathbf{A}\mathbf{y} = \mathbf{b} \tag{7}$$
$$\mathbf{y} \geq \mathbf{0} \tag{8}$$
$$\mathbf{y} \in \mathbb{Z}^n, \tag{9}$$

where $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^m \times \mathbb{R}^n$, and $\mathbb{Z}$ is the integers. Here we are now using $\mathbf{y}$ to denote the set of variables in the optimization criterion, rather than the set of labels for our characters. The connection between these uses will become clear below. We will solve our optimization problem by posing it as an IP[6] over binary valued variables.

Using the notation defined for Eq. (5), let $y_{i:j} = 1$ if variable $y_i = \mathcal{A}_j$ and 0 otherwise. Let $y_{i(i+1):jk} = 1$ if

[3] Later in processing, we will replace our Markov model over strings with a more complex model.

[4] Here we write the prior probability $p(y_1)$ of the first term as $p(y_1|y_0)$ for notational simplicity.

[5] http://icame.uib.no/brown/bcm.html.

[6] We chose to represent our IP in non-standard form. In standard form, all constraint types (with the exception of the non-negativity constraints) are expressed as equalities, whereas for clarity, we expressed one of our constraints as an inequality. However, one can easily convert to and from equality and inequality constraints by introducing additional constraints or slack variables. See [1] for more details.

variables $y_i = \mathcal{A}_j$ and $y_{i+1} = \mathcal{A}_k$ and $y_{i(i+1):jk} = 0$ otherwise. Our optimization problem from Eq. (5) (before integrating the equivalence and non-equivalence constraints $\mathcal{C}$) can then be written

$$\text{minimize} \sum_{i=1}^{N} \sum_{j=1}^{|\mathcal{A}|} \phi_{i:j} y_{i:j} + \sum_{i=1}^{N-1} \sum_{j=1}^{|\mathcal{A}|} \sum_{k=1}^{|\mathcal{A}|} \phi_{i(i+1):jk} y_{i(i+1):jk} \tag{10}$$

$$\text{subject to} \sum_{j=1}^{|\mathcal{A}|} y_{i:j} = 1 \tag{11}$$

$$\sum_{k=1}^{|\mathcal{A}|} y_{i(i+1):jk} = y_{i:j} \tag{12}$$

$$\sum_{j=1}^{|\mathcal{A}|} y_{i(i+1):jk} = y_{(i+1):k} \tag{13}$$

$$y_{i:j}, y_{i(i+1):jk} \geq 0 \tag{14}$$

$$y_{i:j}, y_{i(i+1):jk} \in \mathbb{Z}. \tag{15}$$

Eq. (11) ensures that we choose exactly one label for each character. Eqs. (12) and (13) ensure that we choose exactly one pairwise factor for each pair of characters and enforce consistency between assignments. Lastly Ineq. (14) and Eq. (15) ensure that our variables are restricted to non-negative integers.

We will enforce the equivalence and non-equivalence constraints $\mathcal{C}$ as follows. Let $\mathcal{C} = \{\mathcal{C}_s, \mathcal{C}_d\}$, where $\mathcal{C}_s$ is the set of equivalence constraints and $\mathcal{C}_d$ is the set of non-equivalence constraints. In order to enforce these constraints, we add the following to our IP constraint set:

$$y_{i:j} - y_{i':j} = 0, \forall (i, i') \in \mathcal{C}_s \tag{16}$$

$$y_{i:j} + y_{i':j} \leq 1, \forall (i, i') \in \mathcal{C}_d \tag{17}$$

The equivalence constraints expressed in Eq. (16) enforce that whenever either $y_{i:j}$ or $y_{i':j}$ is set to 1, the other must be set to 1 as well. The non-equivalence constraints expressed in Ineq. (17) enforce that both $y_{i:j}$ and $y_{i':j}$ cannot be set to 1 at the same time. Note that non-equivalence constraints such as (17) can be incorporated into an IP in standard form by including both the constraint and its negation.

3.3. Optimization considerations

We use the Mosek[7] optimization toolbox, which uses a variant of the branch-and-cut method, to efficiently solve our integer programs. Branch-and-cut works by first relaxing the integer program to a linear program. The optimization proceeds, eliminating non-integral solutions by adding constraints that remove these solutions from consideration. Once no more constraints can be added, the optimization uses the branch and bound strategy, which incrementally adds integer constraints on the variables. A branch in the optimization tree corresponds to choosing 0 or 1 for a specific variable. A lower bound on the optimization criterion is maintained by checking conditions on LP relaxations solved throughout the algorithm. An upper bound is maintained by noting the cases where the solution to an LP relaxation has binary values. These bounds allow the algorithm to prune subtrees of the optimization. For more information on linear optimization, see [1].

The Mosek solver uses finite error tolerances on both the integer feasibility and the optimization criterion in order to improve performance. Therefore, we do not have a formal guarantee of optimality. In our experiments, the solver's optimum was always the true optimum.[8]

3.4. Handling inconsistent constraints

If we assume ground-truth equivalence and non-equivalence constraints, our optimization space is guaranteed to be feasible. When we estimate equivalence and non-equivalence, we need to ensure that our constraints are consistent. For example, suppose we estimate that characters i and j are equivalent, characters j and k are equivalent, and characters i and k are different. These constraints are contradictory and hence there is no solution that satisfies them. We resolve this by removing constraints that violate consistency. An example of this scenario is shown in Figure 3.

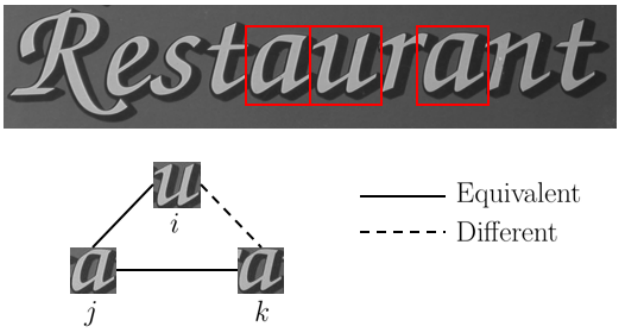

Figure 3. An example of inconsistent constraints. The similarity classifier labels i and j as equivalent, j and k as equivalent, but i and k as different. Through transitivity we know that i should be equivalent to k, which is inconsistent with the classifier output. In order to make our constraints consistent, we remove the constraints associated with this clique.

The root of this problem is that under certain conflicting constraints, we can determine that two characters should be equivalent through transitivity (i.e. $i = j = k$), and this equivalence conflicts with a non-equivalence constraint. To detect such a conflict, we compute a graph over characters,

[7] http://mosek.com/

[8] Here the *true optimum* refers to the best solution with respect to the optimization criterion, not necessarily to the *correct* solution.

where the graph contains an edge between characters i and j if i and j are equivalent under transitivity. All connected components of this graph will be fully connected. If we find that we have included an edge between two nodes that we have estimated to be different, we have a conflict. We can remove this conflict by removing all constraints in the relevant clique.

4. Error Correction

The final stage of recognition involves using information from a search engine[9] to incorporate language statistics that are more global than our bigram model. We use search engine results to model the distribution of common strings, as in [10]. We incorporate these results as follows.

Given a labeling h_{init} from our IP optimization, we create a set of 1-character substitutions of h_{init} over all characters in our alphabet $\mathcal{A}$. We add to this set any suggestions made by the search engine when these candidate strings are submitted to the search engine. This results in a set of hypotheses $\mathcal{H}$ for the true string. For each $h \in \mathcal{H}$, we record the number of hits from the search engine. We induce a probability distribution $p_{\mathcal{H}}(h)$ over $\mathcal{H}$ by normalizing the search hit counts with add-1 smoothing.

Rather than relying solely on search hit counts to correct errors in h_{init}, we wanted to combine this information with appearance information $p_{\mathbf{x}}(h)$ to produce a final probability for each h. If $p_{\mathbf{x}}$ and $p_{\mathcal{H}}$ are both probabilities, we could assume independence and multiply them together. However, as occurs frequently when combining language models with appearance models, these two distributions were "imbalanced" in the sense that the appearance term dominated the product, rendering $p_{\mathcal{H}}$ useless. To address this, we introduced a correction factor α to balance the terms:

$$f(h) = p_{\mathbf{x}}(h)^{\alpha} \cdot p_{\mathcal{H}}(h)^{(1-\alpha)}, \tag{18}$$

for $0 \leq \alpha \leq 1$. We compute $p_{\mathbf{x}}$ by evaluating each hypothesis $h \in \mathcal{H}$ according to a linear function in the form of Eq. (10), with bigram factors removed, and then normalizing. We remove the bigram factors to rely solely on language information from the search engine. After setting α on held out data (Section 5), the best hypothesis $h^* \in \mathcal{H}$ is simply the one maximizing Eq. (18).

5. Experiments and Results

In this section, we first describe the data sets we used for training and evaluating results. We discuss in detail our cross-validation scheme which allows us to use the same data for training and testing by using different folds. We report results for a variety of experiments that compare accuracies of sign recognition with no similarity, with estimated similarity, and with ground truth similarity as given by an oracle. We report accuracies with and without post-processing using the search engine-based language model.

[9] http://www.bing.com/

IP Solution without Equivalence Information	IP Solution with Equivalence Information
Via Wia	Via Via
KELLOGE	KELLOGG
ALAN N SHAREE	ALAN N SHARPE

Figure 4. Sample signs from the data set where equivalence or difference information improves recognition performance. The first two examples show how equivalence information can improve recognition while the third example shows how difference information can improve performance.

5.1. Data sets

There are a variety of data sets that have been used in work on scene text recognition. These include the sign data set of Weinman and Learned-Miller [14], the ICDAR reading competition data set [17], and others [8, 22, 9]. We chose to use the data set of Weinman and Learned-Miller [14] to evaluate our algorithms because a consistent body of work has been evaluated on this data set. We shall refer to this data set as the WLM data set. As discussed above, we also used the ICDAR data set as a source of exploratory data for our initial experiments on equivalence classification.

The WLM data set, obtained from the original authors, consists of 95 signs with a total of 215 words and 1209 printable characters, including digits, lowercase letters, and uppercase letters. The average number of words per sign is 2.26 and the average number of letters per word is 5.62. While scene text recognition requires finding text in an image, possibly segmenting it, and finally recognizing, we adopted a common simplification by starting with hand segmentations of each character in the form of a rectangular bounding box. *This is a substantial simplification of the full scene text recognition problem,* and the difficulty of solving the initial stages of detection and segmentation should not be underestimated. Nevertheless, we felt we could better assess our contributions by deferring the solution of these initial stages. We compare results to others that have made the same assumptions.

5.2. Cross validation

While we developed the general form of our similarity expert using the ICDAR data, after we settled on the form of our model, we wanted to adapt the parameters (for the

		No error correction	Error Correction
No Similarity	Word Accuracy	75.35	88.37
	Char. Accuracy	91.81	94.21
Similarity Classifier	Word Accuracy	78.60	92.56
(1 SIFT feature)	Char. Accuracy	93.05	96.20
Similarity Classifier	Word Accuracy	**78.60**	**92.56**
(4 SIFT features)	Char. Accuracy	**93.30**	**96.28**
Ground Truth	Word Accuracy	83.72	93.02
Similarity	Char. Accuracy	94.46	96.44

Table 1. A table of word and character accuracies for each experiment. Results are shown with and without error correction.

similarity SVM and the α parameter for balancing appearance and language information in the IP) to the properties of the WLM data set. To do this, we split the WLM data set into five folds of approximately the same size. No characters from a single sign appeared in more than one fold. The reason for avoiding having some characters from a sign go into one fold and some go into another is that this would make the similarity classification artificially easy, since the training data and test data might have pairs of characters that were virtually equivalent.

After splitting the WLM data into five folds, we used four folds for training the similarity SVM and used this SVM to rate all of the pairs in the other fold as equivalent or different. This resulted in five independent sets of estimated equivalences and differences. Again using four folds at a time for training, we estimated the parameter α from Eq. (18). We then solved the IP for the test fold and applied the error correction procedure of Section 4, often resulting in dramatic increases in word accuracy.

Table 1 shows a variety of results, compiled across folds, for word accuracy and character accuracy. Word accuracy is simply the percentage of words that are completely correct, including the proper case. A single character error, even if just a case disagreement, renders a word incorrect.

We show an improvement in word recognition accuracy due to similarity. With and without error correction we attain larger than 3% improvement in word accuracy over the equivalent method with similarity removed. Our best result of 92.56% achieves close to the same accuracy as our technique using ground-truth similarity. Furthermore, this result is higher than the state of the art result of 86.05% reported in [14]. See Figure 4 for examples of cases where the use of equivalence information improved performance.

6. Discussion

Perhaps the most immediate question about our results is what caused the improvement? It is tempting to conclude that our large gain in performance was due only to the search engine-based correction. However, a closer examination suggests that we are squeezing more information out of similarity than was demonstrated in previous work.

In particular, in previous work [23], it is shown that similarity can be beneficial when there is a poor language model, but that when language information is added in the form of a lexicon, the similarity information, as implemented, is of little additional benefit. Specifically, without a dictionary, similarity information increases word accuracy from 75.35% to 78.60%, for a gain of about 3.25%. But when a lexicon is added, it seems to reduce the benefits of adding similarity. With a lexicon, similarity raises the accuracy only about 0.50%, from 85.58% to 86.05%.

However, in our work, even with the sophisticated language model implicitly defined by the search engine queries, we still see a 4% gain in word accuracy from adding similarity: from 88.37% to 92.56%. It is interesting to note that this occurs despite the significantly smaller gain in character accuracy of about 2%. We hypothesize one reason this may occur. If a word has exactly one error, and it violates an equivalence constraint, then this constraint effectively forces the algorithm to choose a single label for the equivalent characters. If the algorithm is correct in this guess 50% of the time, then the character accuracy would not change, but the word accuracy would be increased, since some of the single error words would be converted to zero errors, and others would be converted to two errors.

Despite this analysis, it is likely that the system of Weinman et al. [23] would benefit significantly from post-processing using the search engine technique. Hence, it is difficult to conclude from our current experiments which combination of components would lead to the best overall system. In future work, we plan to systematically vary factors and study trade-offs between belief propagation with soft equivalence constraints and integer programming with hard equivalence constraints. We also believe that our similarity results can be further improved by incorporating better alignment algorithms before using our similarity expert. In any case, at a word accuracy rate of 92.56%, we have made a significant step forward, and we hope this will stimulate further work on this difficult problem.

Acknowledgements

The authors thank Jerod Weinman for the precomputed feature data and several helpful discussions. J. Feild was supported by an NSF Graduate Research Fellowship. This work was also supported by NIH grant 1R21EY018398-01.

References

[1] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997. 76, 77

[2] T. M. Breuel. Classification by probabilistic clustering. In *Int. Conf. on Acoustics, Speech and Signal Processing*, 2001. 73

[3] T. M. Breuel. Modeling the sample distribution for clustering OCR. In *SPIE Conf. on Document Recognition and Retrieval*, 2001. 73

[4] T. M. Breuel. Character recognition by adaptive statistical similarity. In *Int. Conf. on Document Analysis and Recognition*, 2003. 73

[5] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. 75

[6] C. Chang and C. Lin. LIBSVM: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. 75

[7] X. Chen, J. Yang, J. Zhang, and A. Waibel. Automatic detection and recognition of signs from natural scenes. *IEEE Transactions on Image Processing*, 13(1):87–99, 2004. 73

[8] X. Chen and A. L. Yuille. Detecting and reading text in natural scenes. In *IEEE Computer Vision and Pattern Recognition*, pages 366–373, 2004. 73, 78

[9] T. E. De Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Int. Conf. on Computer Vision Theory and Applications*, 2009. 73, 78

[10] M. Donoser, H. Bischof, and S. Wagner. Using web search engines to improve text recognition. In *Int. Conf. on Pattern Recognition*, 2008. 74, 78

[11] M. Guillaumin, J. Verbeek, and C. Schmid. Is that you? metric learning approaches for face identification. In *Int. Conf. on Computer Vision*, 2009. 74

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. In *SIGKDD Explorations, Volume 11, Issue 1*, 2009. 75

[13] J. D. Hobby and T. K. Ho. Enhancing degraded document images via bitmap clustering and averaging. In *Int. Conf. on Document Analysis and Recognition*, 1997. 73

[14] J.Weinman and E. G. Learned-Miller. Improving recognition of novel input with similarity. In *IEEE Computer Vision and Pattern Recognition*, volume 1, pages 308–315, 2006. 73, 74, 76, 78, 79

[15] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and simile classifiers for face verification. In *Int. Conf. on Computer Vision*, 2009. 74

[16] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 2003. 75

[17] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young. Icdar 2003 robust reading competitions. In *Int. Conf. on Document Analysis and Recognition*, pages 682–687, 2003. 75, 78

[18] G. Nagy and S.Veeramachaneni. Adaptive and interactive approaches to document analysis. In *Int. Conf. on Document Analysis and Recognition*, volume 2, pages 629–633, 2007. 73

[19] Z. Saidane and C. Garcia. Automatic scene text recognition using a convolutional neural network. In *Int. Workshop on Camera-Based Document Analysis and Recognition*, 2007. 73

[20] C. M. Strohmaier, C. Ringlstetter, K. U Schulz, and S. Mihov. Lexical postcorrection of OCR-results: The web as a dynamic secondary dictionary? In *Int. Conf. on Document Analysis and Recognition*, 2003. 74

[21] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008. 75

[22] K. Wang and S. Belongie. Word Spotting in the Wild. *European Conference on Computer Vision*, pages 591–604, 2010. 73, 78

[23] J. Weinman, E. G. Learned-Miller, and A. Hanson. Scene text recognition using similarity and a lexicon with sparse belief propagation. *IEEE Pattern Analysis and Machine Intelligence*, 2009. 73, 74, 79

[24] M. Yokobayashi and T. Wakahara. Segmentation and recognition of characters in scene images using selective binarization in color space and GAT correlation. In *Document Analysis and Recognition*, pages 167–171. IEEE, 2005. 73