End-to-End Scene Text Recognition

Kai Wang, Boris Babenko and Serge Belongie
Department of Computer Science and Engineering
University of California, San Diego
{kaw006,bbabenko,sjb}@cs.ucsd.edu

Abstract

This paper focuses on the problem of word detection and recognition in natural images. The problem is significantly more challenging than reading text in scanned documents, and has only recently gained attention from the computer vision community. Sub-components of the problem, such as text detection and cropped image word recognition, have been studied in isolation [7, 4, 20]. However, what is unclear is how these recent approaches contribute to solving the end-to-end problem of word recognition.

We fill this gap by constructing and evaluating two systems. The first, representing the de facto state-of-the-art, is a two stage pipeline consisting of text detection followed by a leading OCR engine. The second is a system rooted in generic object recognition, an extension of our previous work in [20]. We show that the latter approach achieves superior performance. While scene text recognition has generally been treated with highly domain-specific methods, our results demonstrate the suitability of applying generic computer vision methods. Adopting this approach opens the door for real world scene text recognition to benefit from the rapid advances that have been taking place in object recognition.

1. Introduction

Reading words in unconstrained images is a challenging problem of considerable practical interest. While text from scanned documents has served as the principal focus of Optical Character Recognition (OCR) applications in the past, text acquired in general settings (referred to as scene text) is becoming more prevalent with the proliferation of mobile imaging devices. Since text is a pervasive element in many environments, solving this problem has potential for significant impact. For example, reading scene text can play an important role in navigation for automobiles equipped with street-facing cameras in outdoor environments, and in assisting a blind person to navigate in certain indoor environments (e.g., a grocery store [15]).

Figure 1. The problem we address in this paper is that of word detection and recognition. Input consists of an image and a list of words (e.g., in the above example the list contains around 50 total words, and include 'TRIPLE' and 'DOOR'). The output is a set of bounding boxes labeled with words.

Despite its apparent usefulness, the scene text problem has received only a modest amount of interest from the computer vision community. The ICDAR Robust Reading challenge [13] was the first public dataset collected to highlight the problem of detecting and recognizing scene text. In this benchmark, the organizers identified four subproblems: (1) cropped character classification, (2) full image text detection, (3) cropped word recognition, and (4) full image word recognition. The work of [6] addressed the cropped character classification problem (1) and showed the relative effectiveness of using generic object recognition methods versus off-the-shelf OCR. The works of [4, 7] introduced methods for text detection (2). The cropped word recognition problem (3) has also recently received attention by [21] and in our previous work [20]. While progress has been made on the isolated components, there has been very little work on the full image word recognition problem (4); the only other work we are aware of that addresses the problem is [16].

In this paper, we focus on a special case of the scene text problem where we are also given a list of words (i.e., a lexicon) to be detected and read (see Figure 1). While making

2011 IEEE International Conference on Computer Vision
978-1-4577-1102-2/11/$26.00

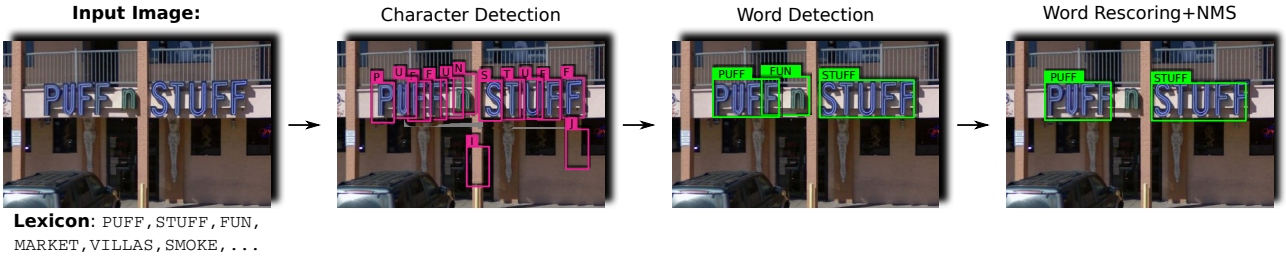

Figure 2. An overview of our word detection and recognition pipeline. Starting with an input image and a lexicon, we perform multi-scale character detection. The words 'PUFF' and 'STUFF' appear in the image while the other words in the lexicon can be thought of as "distractors". Next we perform word detection using a Pictorial Structures framework, treating the characters as "parts" of a word. Finally, we re-score detected words using features based on their global layout, and perform non-maximal suppression (NMS) over words.

the problem more manageable, this framework leaves ample room for useful applications. Consider again the example of assisting a blind person to navigate a grocery store; in this scenario a grocery list can serve as the lexicon. In many other applications it is reasonable to assume that one can use context to limit the search to certain words of interest.

Our contributions are two-fold: (1) We evaluate the word detection and recognition performance of the two-step approach consisting of a state-of-the-art text detector and a leading OCR engine. (2) We construct a system rooted in modern object recognition techniques by extending our work from [20]. We show that our object recognition-based pipelines perform significantly better than one using conventional OCR. We also show that, surprisingly, an object recognition-based pipeline achieves competitive performance *without* the need for an explicit text detection step. This result provides a significant simplification of the end-to-end pipeline and blurs the line between word recognition and the more common object recognition problems studied in computer vision.

2. Overview of Full Image Word Recognition

We discuss each step in detail. Figure 2 shows an overview of our approach.

2.1. Character detection

The first step in our pipeline is to detect potential locations of characters in an image. We perform multi-scale character detection via sliding window classification; this approach has been extremely successful in face [19] and pedestrian [5] detection. However, since our problem requires detection of a large number of categories (62 characters), we must be mindful in our choice of classifier. In this respect Random Ferns [17, 2, 18] are an appealing choice as they are naturally multi-class and efficient both to train and test. In the following sections we will review the basics of Random Ferns and how we use them for detection, and discuss the details of our training data.

Character Detection with Random Ferns For each location ℓ in an image we will extract some feature vector x, and compute a score, $u(\ell, c)$, that tells us the likelihood of character c being in this location, as opposed to the background c_{bg}:

$$u(\ell, c) = \log\left(\frac{\mathbf{p}(c|x)}{\mathbf{p}(c_{bg}|x)}\right) \tag{1}$$
$$= \log\Big(\mathbf{p}(x|c)\Big) - \log\Big(\mathbf{p}(x|c_{bg})\Big) + \log\left(\frac{\mathbf{p}(c)}{\mathbf{p}(c_{bg})}\right).$$

We will assume a uniform prior over categories which means the last term in the second line becomes a constant and can be ignored for our purposes. For the simplicity of the model we will assume that our feature space consists of N binary features (i.e., $x \in \{0,1\}^N$). Notice that storing a representation of the joint probability $\mathbf{p}(x|c)$ would require a table of size 2^N. A common simplification of this model is to assume that all features are conditionally independent (i.e., the Naive Bayes model [1]):

$$\mathbf{p}(x|c) = \prod_{i=1}^{N} \mathbf{p}(x[i]|c).$$

Random Ferns, introduced in [17], can be interpreted as a compromise between the above oversimplification and a fully joint probability table: the features are partitioned into M groups, $x_1, \ldots, x_M$, of size $S = N/M$, and an independence assumption is made for these groups rather than individual features. This results in the following formula for the conditional probability:

$$\mathbf{p}(x|c) = \prod_{i=1}^{M} \mathbf{p}(x_i|c).$$

Notice that the conditional probability for each group, or *Fern*, x_i can be computed using a table of size $2^S \times M$ per category. At run time we must simply compute our binary features, look up the corresponding fern probabilities

Figure 3. Top: synthetic data generated by placing a small random character (with 1 of 40 different fonts) in the center of a 48×48 pixel patch and two neighboring characters, adding Gaussian noise and a random affine deformation. Bottom: "real" characters from the ICDAR dataset. To train our character detector we generated 1000 images for each character.

in stored tables, and multiply the results (or take a log and add). In our present implementation the features consist of applying randomly chosen thresholds on randomly chosen entries in a HOG descriptor [5] computed at the window location. This framework scales well with the number of categories, and has been incorporated in real-time systems for keypoint matching [17] and object recognition [18].

The final step of character detection is to perform non-maximal suppression (NMS). We do this separately for each character using a simple greedy heuristic (similar to what is described in [9]): we iterate over all windows in the image in descending order of their score, and if the location has not yet been suppressed, we suppress all of its neighbors (i.e., windows that have an overlap over some threshold).

The character detection step can be applied directly to the image or after a generic text detector has identified regions of interest.

Equipped with this simple but robust classification module we must now face the task of collecting enough training data to achieve good detection performance.

Synthetic Training Data Collecting a sufficiently large dataset is a typical burden of using a supervised learning method. However, some domains have enjoyed success by training and/or evaluating on synthetically generated images: fingerprints [3], fluorescent microscopy images [12], keypoint deformations [17], and even pedestrians [11, 14]. Beyond the obvious advantage of having limitless amounts of data, synthesizing training images allows for precise control over alignment of bounding boxes – an important property that is often critical to learning a good classifier.

We synthesized about 1000 images per character using 40 fonts. For each image we add some amount of Gaussian noise, and apply a random affine deformation. Examples of our synthesized examples are shown in Figure 3, along with examples of "real" characters from the ICDAR dataset.

2.2. Pictorial Structures

To detect words in the image, we use the Pictorial Structures (PS) [10] formulation that takes the locations and scores of detected characters as input and finds an optimal configuration of a particular word. More formally, let $w = (c_1, c_2, ..., c_n)$ be some word with n characters from our lexicon, $\mathcal{L}_i$ be the set of detected locations for the i^{th} character in w, and $u(\ell_i, c_i)$ be the score of a particular detection at $\ell_i \in \mathcal{L}_i$, computed with Eqn. (1). We seek to find a configuration $L^* = (\ell_1^*, \ldots, \ell_n^*)$ by optimizing the following objective function:

$$L^* = \underset{\forall i, \ell_i \in \mathcal{L}_i}{\operatorname{argmin}} \left(\sum_{i=1}^{n} -u(\ell_i, c_i) + \sum_{i=1}^{n-1} d(\ell_i, \ell_{i+1}) \right), \quad (2)$$

where $d(l_i, l_j)$ is a pairwise cost that incorporates spatial layout and scale similarity between two neighboring characters[1]. In practice, a tradeoff parameter is used to balance the contributions of the two terms.

The above objective can be optimized efficiently using dynamic programming as follows. Let $D(l_i)$ be the cost of the optimal placement of characters $i+1$ to n with the location of the i^{th} character fixed at ℓ_i:

$$D(l_i) = -u(l_i, c_i) + \min_{l_{i+1} \in \mathcal{L}_{i+1}} d(l_i, l_{i+1}) + D(l_{i+1}). \quad (3)$$

Notice that total cost of the optimal configuration L^* is $\min_{\ell_1 \in \mathcal{L}_1} D(\ell_1)$. Due to the recursive nature of $D(\cdot)$ we can find the optimal configuration by first pre-computing $D(\ell_n) = -u(\ell_n, c_n)$ for each $\ell_n \in \mathcal{L}_n$ and then working backwards toward the first letter of the word. For improved efficiency we also include a pruning rule when performing the minimization in Eqn. (3) by only considering locations of ℓ_{i+1} that are sufficiently spatially close to ℓ_i.

Pictorial Structures with a Lexicon. The dynamic programming procedure for configuring a single word can be extended to finding configurations of multiple words. Consider for example the scenario where the lexicon contains the two words {'ICCV','ECCV'}. The value of $D(\ell_2)$ is the same for both words because they share the suffix 'CCV', and can therefore be computed once and used for configuring both words. We leverage this by building a trie structure out of the lexicon, with all the words reversed. Figure 4 shows an example of a trie for five words, with the shaded nodes marking the beginning of words from the lexicon (the rest of string is formed by tracing back to the root

[1] The deformation cost measures the deviation of a child character to the expected location relative to its parent, which is specified as one character-width away, as in [20].

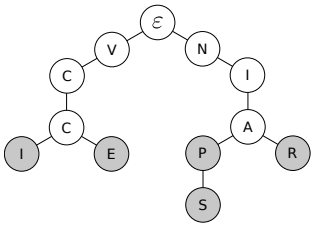

Figure 4. An example of a trie data structure built for a lexicon containing the words {'ICCV','ECCV', 'SPAIN','PAIN','RAIN'}. Every node in the trie that is the beginning of a word is shaded in gray. To efficiently perform Pictorial Structures for all words in the lexicon, we traverse the trie, storing intermediate configuration solutions at every node. When a shaded node is reached, we return the optimal configurations for the corresponding word.

of the tree). To find configurations of the lexicon words in the image, we traverse the trie and store intermediate solutions at every node. When we reach nodes labeled as words (the grayed out nodes), we return the optimal configurations as word candidates. In practice, since an image may contain more than one instance of each word, we return a few of the top configurations for each word. In the worst case, when no two words in the lexicon share a common suffix, this method is equivalent to performing the optimization for each word separately. In practice, however, performing the optimization jointly is typically more efficient. In the remainder of the paper we will refer to the above procedure as "PLEX".

2.3. Word Re-scoring and NMS

The final step in our pipeline is to perform non-maximal suppression over all detected words. Unfortunately, there are a couple problems with the scores returned by PLEX. First, these scores are not comparable for words of different lengths. The more important issue, however, is that the Pictorial Structures objective function captures only pairwise relationships and ignores global features of the configuration. While this allows for an efficient dynamic programming solution to finding good configurations, we would like to capture some global information in our final step. We therefore re-score each word returned by PLEX in the following manner. We compute a number of features given a word and its configuration:

- The configuration score from PLEX (i.e., cost of L^*)
- Mean, median, minimum and standard deviation of character scores (i.e., $u(\ell_i)$)
- Standard deviation of horizontal and vertical gaps between consecutive characters
- Number of characters in a word.

These features are fed into an SVM classifier, the output of which becomes the new score for the word. To train the classifier we simply run our system on the entire training dataset, label each returned word positive if it matches the ground truth and negative otherwise, and feed these labels and computed features into a standard SVM package[2]. Parameters of the SVM are set using cross validation on the training data. Once the words receive their new scores, we perform non-maximal suppression in the same manner as we described for character detection in Section 2.1.

The full system, implemented in Matlab, takes roughly 15 seconds on average to run on a 800×1200 resolution image with lexicon size of around 50 words. We expect the runtime to be much lower with more careful engineering (e.g., [18] showed real-time performance for Ferns).

3. Experiments

In this section we present a detailed evaluation of our PLEX pipeline, as well as a two-step pipeline consisting of Stroke Width Transform [7] (a state-of-the-art text detector) and ABBYY FineReader[3] (a leading commercial OCR engine). We used data from the Chars74K[4] dataset, introduced in [6] for cropped character classification; the ICDAR Robust Reading Competition dataset [13], discussed in Section 1; and Street View Text (SVT), a full image lexicon-driven scene text dataset introduced in [20][5].

3.1. Character Classification and Detection

We begin with an evaluation of character classification on the Chars74K-15 (where there are 15 training examples per character class) and the ICDAR-CH (character classification sub-benchmark). We measure performance of Ferns trained on synthetic data and Ferns trained on the real images from the respective datasets (labeled 'NATIVE'). We also compare to previously published results of HOG+NN and ABBYY [20], as well as MKL [6].

Table 1 lists the character classification results on the two datasets. We see that NATIVE+FERNS outperforms other methods on the ICDAR-CH dataset. However, its performance on the Chars74K-15 benchmark is below that of previous results using HOG+NN. Upon further inspection, we noticed significant similarity between the images in the training and testing sets from Chars74K (in some cases near duplicates) which work to the advantage of a Nearest Neighbor classifier. In contrast, the training and testing split in ICDAR-CH was done on a per image basis, making it highly unlikely to have near duplicates across the split – this helps account for the drop in performance of HOG+NN on ICDAR-CH. Finally, we see that training on purely syn-

[2] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[3] http://finereader.abbyy.com

[4] http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[5] The dataset has undergone revision since originally introduced.

Method	Chars74K-15	ICDAR-CH
SYNTH+FERNS	.47	.52
NATIVE+FERNS	.54	.64
HOG+NN [20]	.58	.52
MKL [6]	.55	-
ABBYY [20]	.19	.21

Table 1. Character classification accuracy of Ferns versus previously published results on the Chars74K and ICDAR benchmarks. The SYNTH+FERNS method was trained on synthetic data while the NATIVE+FERNS was trained on data from their respective datasets.

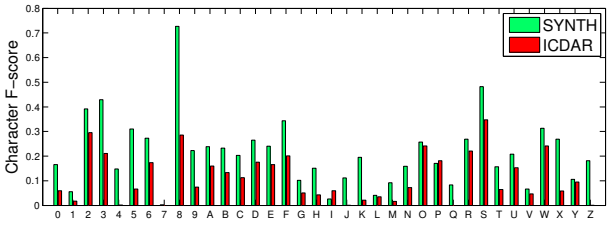

Figure 5. Character detection performance (F-score) comparing Fern classifiers trained on synthetic data versus data from ICDAR.

thetic data shows competitive performance to training on the native data.

While the character classification accuracy of SYNTH+FERNS appears lower than NATIVE+FERNS, our end goal is to use this classifier in a sliding window fashion for character detection. We therefore evaluated the character detection performance of SYNTH+FERNS and ICDAR+FERNS (trained using real characters from the ICDAR data) on the full images from ICDAR. Figure 5 shows the F-score, defined as $(\frac{1}{0.5\times precision} + \frac{1}{0.5\times recall})^{-1}$, for each character. From this plot we see that although ICDAR+FERNS performed better on cropped character classification, SYNTH+FERNS is more effective when used for sliding window character detection. A possible explanation for this is that training on synthetic data benefits both from a larger volume of training examples, and from more consistent alignment of the data.

3.2. Cropped Word Recognition

Next, we evaluate cropped word recognition on the ICDAR-WD and SVT-WD (the cropped word benchmarks of the respective datasets). This is akin to measuring recall of a system that has a "perfect" text detector. In the SVT-WD case, a lexicon of about 50 words is provided with each image as part of the dataset. For the ICDAR dataset, we measure performance using a lexicon created from all the words that appear in the test set (we call this ICDAR-WD(FULL)), and with lexicons consisting of the ground truth words for that image plus 50 random "distractor" words added from the test set (we call this ICDAR-WD(50)). The latter benchmark allows for direct comparison to SVT-WD. For simplification, we ignore all words that contain non-alphanumeric characters, as well as words with 2 or fewer characters.

Method	ICDAR(FULL)	ICDAR(50)	SVT
SYNTH+PLEX	.62	.76	.57
ICDAR+PLEX	.57	.72	.56
ABBYY	.55	.56	.35

Table 2. Accuracy of cropped word recognition comparing Pictorial Structures-based methods (trained on synthetic data and data from ICDAR) to ABBYY FineReader.

Table 2 shows our results for word recognition on cropped images for three methods: 1) PLEX with Ferns trained on ICDAR-CH data, 2) PLEX with Ferns trained on synthetic data, and 3) ABBYY. The latter is a generic OCR system and does not take a lexicon as input. To simulate lexicon driven OCR, we return the word in the lexicon that has the smallest edit distance to the raw output of ABBYY (i.e., a type of spell checking). It is important to note that evaluating the raw output from ABBYY results in poor performance and some form of post-processing is essential – without spell checking, the accuracy of ABBYY is .21 on the ICDAR-WD(FULL).

Comparing the results in Table 2 to our previous results from [20], we notice that ABBYY performs considerably better on ICDAR-WD(FULL) than before. This difference was observed after expanding word boxes by 25% in both dimensions.

These results show that training our system with synthetic data indeed leads to better performance. They also suggest that the SVT dataset is significantly more challenging than ICDAR.

3.3. Word Detection and Recognition

Our main experiment consists of evaluating end-to-end word detection and recognition on the ICDAR and SVT datasets. We follow the evaluation guidelines outlined in [13], which are essentially the same as the evaluation guidelines of other object recognition competitions, like PASCAL VOC [8]. A bounding box is counted as a match if it overlaps a ground truth bounding box by more than 50% and the words match (ignoring case).

ICDAR Evaluation We compare performance of several end-to-end pipelines on the ICDAR dataset. Our first pipeline is a combination of a Stroke Width Transform (SWT) and ABBYY (naming this pipeline SWT+ABBYY). We acquired a set of bounding boxes returned by SWT from the authors of [7]; these regions are then fed into ABBYY. As we did before, we correct results from ABBYY by converting its output to the word in the lexicon with the smallest

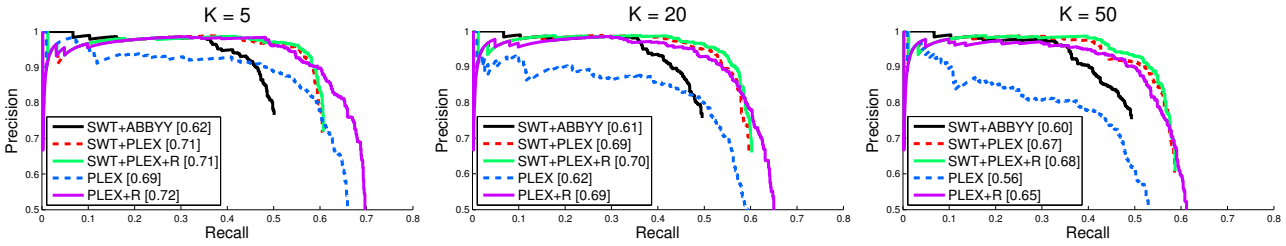

Figure 6. Precision and recall of end-to-end word detection and recognition methods on the ICDAR dataset. Results are shown with lexicons created with 5, 20, and 50 distractor words. F-scores are shown in brackets next to pipeline name.

edit distance. In this case, we throw out all bounding boxes for which ABBYY returns an empty string, or for which the smallest edit distance to a lexicon word is above some threshold – this helps reduce the number of false positives for this system.

Next, we apply PLEX to full images without a text detection step (named PLEX). Finally, we combine SWT with PLEX as the reading engine (named SWT+PLEX). This hybrid pipeline serves as a sanity check to see if text detection improves results of PLEX. To show the effect of the re-scoring technique presented in Section 2.3, we evaluate the latter two pipelines with and without this step (adding '+R' to the name when re-scoring is used). Motivated by our earlier experiments, we all PLEX-based systems were trained on synthetic data.

We construct a lexicon for each image by taking the ground truth words that appear in that image and adding K extra distractor words chosen at random from the test set, as well as filtering short words, as in the previous experiment.

Figure 8 shows select examples of output; Figure 6 shows precision and recall plots for different values of K as we sweep over a threshold on scores (or maximum edit distance for ABBYY, as described above). From these results, we make the following observations. (1) Re-scoring significantly improves performance of PLEX, especially for larger lexicons. (2) The performance of PLEX-based pipelines is significantly better than SWT+ABBYY. While the gap in F-scores of these methods shrinks as the lexicon increases, the PLEX based systems obtain a considerably higher recall at high precision points in all cases. (3) PLEX+R, a system that *does not rely on explicit text detection*, is not only comparable to SWT+PLEX+R, but actually outperforms it for smaller length lexicons.

While an explicit text detection step could in principle improve the precision of a system, the recall is also limited by that of the text detector. Improving the recall of such a two stage pipeline would therefore necessitate improving the recall of text detection. Upon further examination of our results, we found a strong positive correlation between the words that ABBYY was able to read and the words that were detected by the SWT detector. Recall that in the cropped word experiment, ABBYY achieved .56 accuracy on the ICDAR-WD(50) benchmark (correctly reading 482 words). In the end-to-end benchmark of ICDAR with $K = 50$, SWT+ABBYY correctly read 438 words (very close to its performance on cropped words, which simulates a "perfect" text detector). This shows that improving the recall of SWT would not have a big impact on the performance of SWT+ABBYY, unless ABBYY was improved as well.

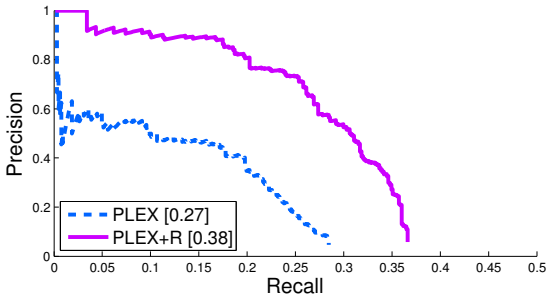

Figure 7. Precision and recall of end-to-end word detection and recognition methods on the Street View Text dataset. F-scores are shown in brackets next to pipeline name.

While ABBYY is a black box, the PLEX pipeline is constructed using computer vision techniques that are well understood and constantly improved by the community. We believe this paves a clearer path towards improving reading accuracy.

The work of [16] reported word recognition results on the ICDAR dataset of 0.42/0.39/0.40, for precision, recall and F-score. In our experiments, we created word lists for every image, however word lists were not provided in the experiments in [16], making the results not directly comparable. The closest comparison in our framework is to provide the entire ground truth set (> 500) as a word list to each test image. In that case, our PLEX+R pipeline achieves 0.45/0.54/0.51.

SVT Evaluation For the SVT dataset, we evaluated only PLEX and PLEX+R because we were unable to obtain SWT output for this data (and the original implementation is not publicly available). Recall that this dataset comes with a lexicon for each image (generated from local busi-

Figure 8. Selected results of end-to-end methods on the ICDAR dataset (for a lexicon with $K = 20$ distractors). Results from PLEX+R are shown in green and results from SWT+ABBYY are shown in blue. In the first two images ABBYY has trouble reading text with noisy image conditions and unusual fonts; the last image is more well suited for ABBYY as it is more similar to a scanned document.

ness searches in Google Maps). Figure 9 shows examples of output and Figure 7 shows precision and recall plots for this experiment. Again we see that re-scoring makes a dramatic improvement in the results. As with the cropped word recognition results, comparing the performance on the ICDAR(50) to the performance on SVT exposes the relative difficulty of SVT. One difference between ICDAR and SVT that may contribute to this difficulty is that for each ICDAR image *all* of the words in that image are contained in the lexicon. On the other hand, in SVT, many of the images contain irrelevant text that leads to a higher number of false positives for our system. Notice, however, that this problem would not be alleviated by the use of a text detector – the burden still lies with the reading module.

4. Conclusion

These results establish a baseline for using generic computer vision methods on end-to-end word recognition in the wild. We show that we can outperform conventional OCR engines and do so *without* the explicit use of a text detector. The latter is a promising new direction, significantly simplifying the recognition pipeline.

5. Acknowledgements

We thank Piotr Dollár for helpful conversations and for making his toolbox available. We thank Boris Epshtein for generously sharing output from previous work [7]. This material is based upon work supported by an NSF Graduate Research Fellowship, Google Fellowship, and the Amazon AWS in Education Program.

References

[1] C. Bishop. *Pattern recognition and machine learning*. Springer, 2006. 2

[2] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *ICCV*, 2007. 2

[3] R. Cappelli, D. Maio, D. Maltoni, and A. Erol. Synthetic fingerprint-image generation. In *ICPR*, 2000. 3

[4] X. Chen and A. L. Yuille. Detecting and reading text in natural scenes. In *CVPR*, 2004. 1

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005. 2, 3

[6] T. de Campos, B. Babu, and M. Varma. Character recognition in natural images. In *VISAPP*, Feb. 2009. 1, 4, 5

[7] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *CVPR*, 2010. 1, 4, 5, 7

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, 2010. 5

[9] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE TPAMI*, 2009. 3

[10] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 61:55–79, 2005. 3

[11] K. Grauman, G. Shakhnarovich, and T. Darrell. Inferring 3d structure with a statistical image-based shape model. In *CVPR*, 2008. 3

[12] A. Lehmussola, P. Ruusuvuori, J. Selinummi, H. Huttunen, and O. Yli-Harja. Computational framework for simulating fluorescence microscope images with cell populations. *IEEE Trans. Med. Imaging*, 26(7):1010–1016, 2007. 3

Figure 9. Selected results on the Street View Text dataset. PLEX+R results are shown in green and words from the corresponding lexicons are shown in dashed pink (recall that these images can contain other irrelevant text).

[13] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young. ICDAR 2003 robust reading competitions. *ICDAR*, 2003. 1, 4, 5

[14] J. Marin, D. Vazquez, D. Geronimo, and A. Lopez. Learning appearance in virtual scenarios for ped. detection. In *CVPR*, 2010. 3

[15] M. Merler, C. Galleguillos, and S. Belongie. Recognizing groceries in situ using in vitro training data. In *SLAM*, 2007. 1

[16] L. Neumann and J. Matas. A method for text localization and recognition in real-world images. In *ACCV*, pages 770–783, 2010. 1, 6

[17] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *CVPR*, 2007. 2, 3

[18] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *CVPR*, 2008. 2, 3, 4

[19] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001. 2

[20] K. Wang and S. Belongie. Word spotting in the wild. In *ECCV*, 2010. 1, 2, 3, 4, 5

[21] J. J. Weinman, E. Learned-Miller, and A. R. Hanson. Scene text recognition using similarity and a lexicon with sparse belief propagation. *IEEE TPAMI*, 31:1733–1746, 2009. 1