Robust Text Reading in Natural Scene Images

Tao Wang, David Wu
Stanford Computer Science Department
353 Serra Mall, Stanford, CA 94305
twangcat@stanford.edu, dwu4@stanford.edu

Abstract

In this paper, we consider using two-layer, convolutional neural networks (CNNs) to construct an end-to-end text recognition system for natural images. The approach we take is similar to traditional approaches to text recognition in which we first train a text detector that finds regions in the image that contain text, and then train a character classifier and apply it to the regions of interest to perform word-level recognition. While the basic two-stage system we adopt is similar to other text recognition systems, we note that in all existing systems, the detector and classifier are treated as two distinct systems with different sets of hand-crafted features. Here, we take a decisively different approach by leveraging the power of unsupervised feature learning and fine-tuning to construct a common framework for both the detector and recognizer. Such a system demonstrates the possibility of using a single, unified architecture for both detection and recognition in the construction of a full end-to-end system. In constructing such a system, we present state-of-the-art results in cropped word recognition and reputable results in full end-to-end recognition.

1. Introduction

Extracting textual information from natural images is a challenging problem with many practical applications. While current state-of-the-art methods achieve nearly perfect performance on Object Character Recognition (OCR) for scanned documents, the more general problem of recognizing text in unconstrained images is not so simple. Recognizing text in scene images is more challenging due to the many possible variations in backgrounds, textures, fonts, and lighting. As a result of these variations, many high-performing text detection and character recognition systems combine cleverly hand-engineered features [3, 8] or carefully-designed multi-step pipelines [11, 15].

In this paper, we approach the problem from a different angle by using systems that can learn the underlying features best suited for the problem. Like many previous text recognition systems, we postulate a multi-stage approach. We begin by training a text detector that performs a binary classification task: determine whether a given image patch contains text or does not contain text. Then, given a full image, we take a sliding window approach across multiple scales to identify regions of text. Given this multiscale response map, we apply several post-processing stages to estimate candidate bounding boxes for regions of text. Then, we train a 62-way classifier (one class for each lowercase and uppercase letter and one for each digit) for character recognition. Given the candidate bounding boxes generated by the text detector, we again use sliding windows to compute the character classifier response at each point in the bounding box. Given the character responses at each point, we leverage a lexicon-backed weighted string distance metric to determine the word within the candidate bounding box along with its recognition score. We then greedily make predictions on the top-scoring candidate boxes and rule out overlapping ones with lower scores. We also filter out false positives by thresholding the recognition score.

Figure 1: Illustration of the end-to-end word recognition problem we address. The input to our system is an image and a lexicon (e.g. here, the lexicon contains ≈ 50 words, including "SPIDER" and "MAN"). The output is a set of bounding boxes labeled with predicted words.

2. Background

Many of the proposed methods for text recognition are based upon sequential pipelines consisting of several disjoint components such as text detection, segmentation, and recognition. In each case, top-performing systems have generally combined simple classifiers with hand-tuned or domain-specific features, such as stroke width [3] or background-color consistency, aspect ratio, and number of holes [14]. Others have applied probabilistic models incorporating various forms of prior knowledge to good effect [18]. Additionally, many of these systems are targeted towards one particular part of the full text recognition problem. The challenge has thus been in cohesively integrating the different systems. For examples, features that work well for detection will likely not work well for recognition and vice versa; this is evidenced by the general lack of end-to-end text recognition systems that incorporate both a detection and a recognition component.

Recent work in feature learning algorithms offer a way to integrate the two systems. Such algorithms have enjoyed numerous successes in many diverse fields such as visual recognition [7]. In the field of text recognition, the system in [6] has achieved decent text detection and state-of-the-art character recognition using a simple, but scalable feature learning architecture incorporating virtually no hand-engineering or prior knowledge. In their work, Coates, *et al*. [6] adopt a purely unsupervised approach to learn a dictionary of features and then use those features, evaluated convolutionally over the image, as the inputs to a simple classifier. In this paper, we diverge from their purely unsupervised framework by applying supervised fine-tuning to a multi-layered convolutional neural network. Such networks have enjoyed many successes in similar problems such as handwriting recognition [12], visual object recognition [5], and character recognition [16]. In this paper, we consider using the same convolutional architecture for both the detection and recognition phases of the pipeline and in doing so, demonstrate that by using a single feature-learning algorithm, we can construct both a text detector as well as a character recognizer and integrate them together into a high-performing, complete end-to-end system.

3. Methodology

3.1. Learning Architecture

First, we discuss the learning architecture we adopt for the detection and recognition phases. We consider a multi-layer convolutional neural architecture similar to [5, 12, 16] for both components of the system. In particular, we adopt a two-layer convolutional structure (shown in Figure 2), with an average-pooling layer following each convolutional layer. The responses from the second pooling layer are then combined in a fully connected classification layer, where

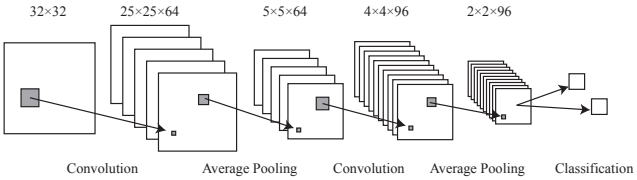


Figure 2: Convolutional neural network used for detection and recognition. The only difference between the CNN used for detection and recognition is the sizes of the convolutional layers.

we have one output unit for each class (binary in the case of text detection, 62-way in the case of character recognition). For text detection, we pretrain the first convolutional layer with filters generated by a feature learning algorithm and fine-tune the overall network by backpropagation of the classification error. In the 62-way character classification task, there are considerably more parameters to tune and thus, it is prohibitively expensive to perform full backpropagation over the network. Consequently, we again use unsupervised pretraining to learn the second convolutional layer and apply supervised training to just the top layer classifier, keeping the first and second layer weights fixed. Finally, we integrate the text detector and character recognizer to construct a complete end-to-end text recognition system.

3.1.1 Unsupervised pretraining

We begin by using an unsupervised learning algorithm to pretrain the filters used for both detection and recognition. Here, we use a pipeline that resembles the architectures described in [6, 7]. We briefly outline the key components of this system:

1. Collect a set of m small image patches from the training set. As in [6], we use 8x8 grayscale patches. This yields a set of m vectors of pixels $\tilde{x}^{(i)} \in \mathbb{R}^{64}, i \in \{1, \ldots, m\}$.

2. Normalize each vector $\tilde{x}^{(i)}$ for brightness and contrast (subtract out the mean and divide by the standard deviation). We then whiten the patches $\tilde{x}^{(i)}$ using ZCA whitening [10] to yield a new set of vectors $x^{(i)}$.

3. Apply an unsupervised learning algorithm on the pre-processed patches $x^{(i)}$ to build a mapping from input patches to feature vectors $z^{(i)} = f(x^{(i)})$. In this paper, we adhere to the variant of the K-means algorithm described in [6] where we learn a dictionary $D \in \mathbb{R}^{64 \times d}$ containing d normalized basis vectors.

3.1.2 Convolutional layers

The two-layer convolutional architecture we use for text detection is given in Figure 2. Note that we use the same architecture to train the character classifier, with the exception that the convolutional layers contain 300 and 3000 filters instead of 64 and 96. The larger number of maps is due to the fact that the character recognizer is performing 62-way classification rather than binary classification, and thus, requires a more expressive model. For the first convolutional layer, we use a set of 8-by-8 filters. More specifically, given a filter (kernel) K and an 8-by-8 window centered on (x, y), the response R at (x, y) will be given by $R(x, y) = K \star I(x, y)$ where $I(x, y)$ denotes the 8-by-8 window in the input patch centered at (x, y) and $\star$ denotes the convolution operator. The input to our convolutional network is a 32-by-32 image patch. Note that we only evaluate K over windows that are completely within the bounds of the patch. The application of each filter over the image patch yields a 25-by-25 response map. After evaluating each filter, we arrive at a 25-by-25-by-64 response map as the output for the first layer. As in [6], we apply a scalar nonlinear activation function to the responses $R(x, y)$: $z = \max\{0, |R(x, y)| - \alpha\}$ where α is a hyperparameter. In this paper, we take $\alpha = 0.5$. As is standard in the literature on convolutional architectures, we now apply a spatial pooling step. This has the benefit of reducing the dimensionality of the response maps at each layer as well as providing the model a degree of translational invariance. Here, we opt for average pooling, in which we sum over the values in a 5-by-5 grid over the 25-by-25-by-64 response map. We then stack another convolutional and average pooling layer on top of the outputs from the first layer. The outputs of this second layer consist of a 2-by-2-by-96 response map in the case of detection. This output feeds into a fully connected classification layer. To train the network for the text detector, we backpropagate the classification error in the form of the L_2-SVM loss through the network to fine tune the parameters of the network. As noted above, for the character classifier, we only train the top-level classifer without backpropagating through the entire network. Instead of fine-tuning, we adopt wide network structures (25-by-25-by-300 on the first layer and 2-by-2-by-3000 on the second layer) and pretrain each layer using the response map from the previous layer.

3.2. Dataset

For text detection, we train a binary classifier that decides whether a single 32-by-32 subwindow contains text or not. Unlike [6], we consider positive examples to be examples where a complete character appears centered in the window. Examples where the character is occluded, cropped, or off-center are considered negatives. This particular distinction is made so that the detector will focus on identifying regions where the text is centered since such windows are preferred for word-level recognition in the subsequent part of the system.

Synthetic Training Data Coates *et al*. [6] have used large synthetic datasets to improve classification results. Likewise, in this paper, we generate high quality synthetic training images using a total of 665 fonts for training both the text detector and character classifier. The number of images per character class are distributed according to the unigram frequency obtained from the Brown Corpus [9] so as to simulate the natural distribution of character classes. The grayscale level of the characters and the background are generated from Gaussian distributions with the same mean and standard deviation as those in the ICDAR training images. We also apply small amounts of Gaussian blurring and projective transformations to a random portion of the images and finally blend the images with natural backgrounds to simulate background clutter. The resulting synthetic images are shown alongside with real-world characters cropped from the ICDAR 2003 dataset in Figure 3. One advantage of using synthetic data is that we have full control of the location of the text in the image, so we can easily generate many types of negative examples (e.g. improperly scaled, improperly aligned, etc.) to use as hard negatives. As such, we have compiled a dataset consisting of examples from the ICDAR 2003 training images [13], the English subset of the Chars74k dataset [8], and the sign-reading dataset from Weinman, *et al*. [17], as well as synthetically generated examples. In training the detector, our training set generally consists of 75,000 positive examples and 150,000 negative examples; for the classification task, we use about 213,000 examples in total.

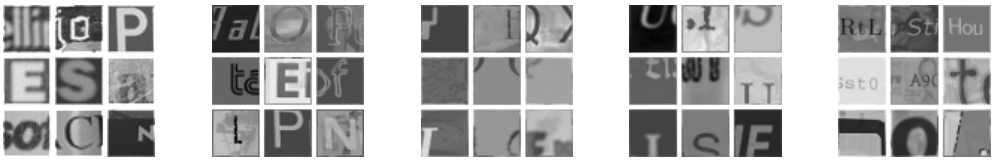
4. End-to-End Pipeline Integration

As discussed before, we use a two-pass pipeline where we first run sliding window detection over higher resolution input images to narrow down a set of candidate regions containing text. Then, we locate and recognize words using the responses of the sliding windows. There are several ways to obtain word-level bounding boxes from these sliding window responses. Wang *et al*. [11] use pictorial structures together with a lexicon to locate the target words, while others have used variants of conditional random fields (CRFs) [17]. Since both our detector and character classifier have high accuracies, we rely on simple non-maximal suppression (NMS) based heuristics to obtain our final end-to-end results.

4.1. Text detection

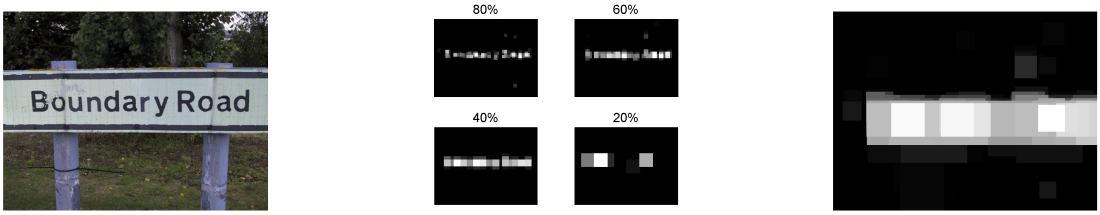
4.1.1 Pixel-level responses

Given an input image, we begin by identifying regions of text using a sliding window approach performed over multiple scales. In this paper, we consider ten different scales,



(a) ICDAR positives. (b) Synthetic positives. (c) Synthetic negatives (spaces). (d) Synthetic negatives (bad alignment). (e) Synthetic negatives (bad scale).

Figure 3: Images used to train the detector and classifiers.



(a) Sample image from ICDAR dataset. (b) Detector responses from multiple scales. (c) Pixel level responses over full image.

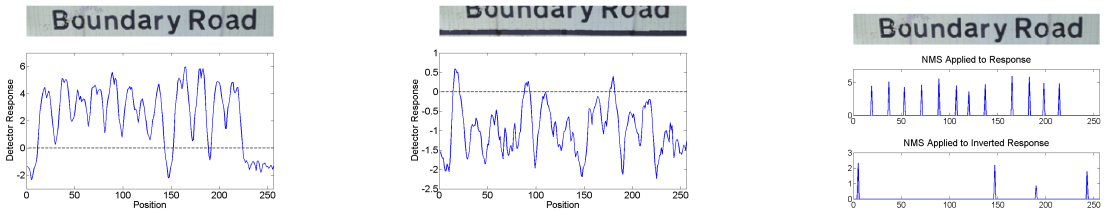
Figure 4: Multi-scale response from text detector.

starting from 100% and proceeding down to 10% in decrements of 10%. Due to the large numbers of windows in a given image and the computational infeasibility of computing the activations for every possible window, we slide our windows with a column stride of 3 and a row stride of 8. We show a visualization of the responses for four different scales in Figure 4b where for each window, we simply fill the region spanned by the window with the corresponding detection score. We also present a pixel-level response map $R(x, y)$ (4c) where we aggregate the scores from each scale according to $R(x, y) = \max_s R_s(x, y)$ where $R_s(x, y)$ denotes the response at position (x, y) in the response at scale s. While pixel-level responses maps present a viable way of assessing the performance of the detector [6], it is not immediately clear how to compute a proper bounding box for regions of text. Noting that the responses tend to be positive for regions of text and negative otherwise, one possibility is to construct the rectangle that maximizes the sum of responses. This is a simple optimization problem that may be solved efficiently using a dynamic programming algorithm [2] over the response map. While this method does construct reasonable bounding boxes, it also tends to cluster multiple lines of text into a single bounding box, especially if the separation distance between lines is small compared to character height, thereby causing the maximum-sum rectangle to encapsulate multiple lines. This is a particularly inconvenient format for the character and word recognizer, which are designed to operate on a single word.

4.1.2 Bounding box estimation

Due to the difficulties in line segmentation discussed above, we consider an alternative approach for estimating the bounding boxes for regions of text. Noting that words will generally align themselves to a particular line in the image, we instead apply the sliding window approach to individual lines in the response map at each scale. Here we take a line to be a single 32-by-w window in the image where w is the width of the image. Note that by imposing this constraint, the predicted bounding boxes will all have a height given by $32/s$ where s denotes the scale of the input. We show two example lines with their respective detector responses in Figures 5a and 5b. In lines where the text is well-centered (5a), the detector responses are generally positive while for lines in which the text is off-centered (5b), the responses are negative. We now score the line by computing the maximum-sum subarray over the detector responses for the given line. We notice that this is effectively the 1D analog of the problem encountered earlier with computing the maximum-sum submatrix. As before, this may be efficiently computed using a dynamic programming algorithm [2].

By imposing a threshold on the scores for each line, we have a set of candidate lines of text. The next task is to de-



(a) Response for centered line of text.

(b) Response for uncentered line of text.

(c) NMS responses.

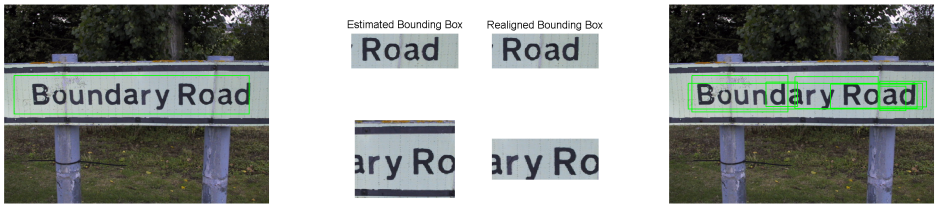
Figure 5: Single line sliding window and NMS for bounding box estimation. Note that in this visualization, the score at a position x denotes the score for the window between x and $x + 32$. In particular, the scores are *not* for a window centered at x.

termine the left and rightmost extents of the text. One possibility is to take the left and right edges for the maximum-sum submatrix, but in practice, this approach does not work well. Noting that our detector is trained on patches consisting of characters center-aligned in an image, a window centered upon a space would tend to get a low, negative score. Since the character spacing in a word is variable, in many cases, the detector will produce a large negative value for a space between characters within a word. In turn, the maximum-sum subarray across that line would not include the space; as a result, the estimated rectangle would only bound a portion of the word. To address this problem, we use a metric that discerns the locations of the *characters* in the line and construct the bounding box such that it contains all the characters. Here, we make the observation that since the detector activates strongly on centered characters, the peaks of the line-response map will generally translate to the positions of characters. To pick out the peaks, we apply non-maximal suppression (NMS) [1] to the detector responses. More precisely, for a response $R(x)$ at position x in the response map, the NMS response map $R'(x) = R(x)$ if and only if $R(y) \leq R(x)\ \forall x - \epsilon \leq y \leq x + \epsilon$ and 0 otherwise. Noting that characters generally occur together, we take ϵ to be a small number, generally on the order of one to two units in the response map. Sample NMS responses for the lines of text are shown in Figure 5c. Using the NMS responses, we take the leftmost peak (leftmost nonzero response) and the rightmost peak to be the extents of the bounding box. Note that this method does not perform word-level segmentation, and instead outputs a series of boxes that should encapsulate all the text on a given line.

4.1.3 Word-level segmentation and post-processing

The input to the character and word recognizer modules must consist of a single word and not a full line of text. Looking at the responses across a line, it is evident that when the detector sees a space, the response falls sharply to a local minimum. In general, it is difficult to distinguish between a space between words and a space between characters, primarily due to the high degree of variability of both cases across the dataset. Since our end goal is not to build an optimal word detector, but rather an end-to-end system, we have some room for error. In particular, our goal is to achieve high recall on the dataset and tolerate some false positives, the reason being that we can always filter out the wrong candidates in the subsequent stages of the pipeline. Using this simplification, we proceed to estimate spaces and perform word-level segmentation. Here, we leverage the same NMS framework as above, except now, we apply it to the inverse (negative) of the detector response across the region of interest. Since spaces between words will tend to be further apart, we suppress across a larger window ($\epsilon \approx 15$ units). The size of the peak denotes the confidence that there is a space at the specified location. For each predicted space, up to a maximum of 5-6, we split the box containing the space into two regions and add both candidates to the list of predicted bounding boxes for the particular image. Notice that we do *not* remove the original region from the list of possible candidates. This means that if we falsely predict a space to be in the middle of a word, the original region containing the word will not be removed from consideration. This way, we can improve recall by introducing additional candidates while sacrificing precision in the form of introducing additional false positives into the mix. Note that when we split a region into two regions about a space, we recompute the score for both regions, so as to maintain a consistent scoring scheme across all regions.

In the previous section, we noted that because we are performing a sliding window over individual lines in the response map, our predicted bounding boxes will always have a height of $32/s$ where s is a scale that we have considered. Since it is definitely not the case that the height of the bounding boxes takes on a small, finite set of possible val-



(a) Bounding box before word segmentation.

(b) Example realignments.

(c) Sample bounding boxes passed to recognizer.

Figure 6: Post-processing process: space estimation and realignment.

ues, some of the estimated bounding boxes are going to be of the wrong scale. To remedy this, we propose a realignment algorithm that effectively takes each of the predicted bounding boxes and adjusts the top and bottom boundaries of the bounding box so as to maximize the detector score over the region. More precisely, if the top t of the bounding box is originally at location y, we consider values of t ranging from $t - \epsilon$ to $t + \epsilon$. In our experiments, we have found that using 8 evenly spaced intervals spaced between $t - \frac{h}{3}$ and $t + \frac{h}{3}$, where h denotes the height of the image offers a good tradeoff between performance and speed.

After realigning all of the candidate bounding boxes, we apply one final step of NMS, this time to reduce the number of candidate bounding boxes. Effectively, given two overlapping bounding boxes A and B, with scores $s_A > s_B$, respectively, we suppress B if the ratio of the area of the intersection $A \cap B$ to the area of the union $A \cup B$ exceeds some ratio r (here, we have taken r to be between 0.4 and 0.5). Using this, we arrive at a candidate set of bounding boxes that we may use in the word recognizer.

4.2. Word-level recognition

Here we consider the scenario in which the ground truth word-level bounding box is given, and aim to build a word-level recognizer that gives the correct labeling to the input bounding box along with a recognition score reflecting the confidence of its prediction. All word bounding boxes are first resized to have a height of 32 pixels while maintaining the aspect ratio. This way, the character classifier can slide horizontally across the bounding box to generate a 62-way classifier score at each window position. The confidence score c_w of a sliding window w is defined as the difference between the highest and the second highest 62-way score in that window. Intuitively, a more positive c_w indicates a higher confidence that a centered character is present in window w. Since the c_w's are usually noisy, we apply NMS on the c_w's to obtain a set of candidate character locations, each associated with a 62-way classification score s_i. Our initial guess of the word will simply be a string consisting of characters with the highest 62-way score at each candidate location. In order to match the initial guess with a word in the lexicon, we use a variant of the weighted string distance (WSD) described in [4]. Unlike ordinary string distance where all edit actions have the same cost, each edit action in WSD is associated with a weight that is equal to the difference between the 62-way score of the source character and the target character at that location. To enable insertion and deletion, a null character λ is defined, and its weight is calculated as the average of the best 62-way score at each location minus the best 62-way score at a particular location. In other words,

$$s_i(\lambda) = \left(\frac{1}{W_{\text{NMS}}} \sum_{j=1}^{W_{\text{NMS}}} \max s_j \right) - \max s_i \qquad (1)$$

Where W_{NMS} denotes the number of window locations after NMS.

As with ordinary unweighted string distance, the weighted distance between two arbitrary strings can be efficiently computed using a dynamic programming algorithm in $O(mn)$ time, where m and n are the lengths of the two strings [4]. By using WSD, we have a simple way of comparing our initial guessed string with all entries in the lexicon while taking the character classifier score into consideration. The output string of the word recognizer will be the lexicon word with smallest weighted edit distance to the initial guessed word.

Finally, the location of each character in the final output string can be determined by backtracing the dynamic programming table for the WSD. For a word bounding box b, we define the recognition score $S_b(str)$ of its output string str as the sum of the classification scores associated with each predicted character at their respective locations:

$$S_b(str) = \sum_{j=1}^{\text{length}(str)} s_{L(str(j))} \qquad (2)$$

where $L(str(j))$ is the sliding window index of the j^{th} character in the final predicted string. The recognition score

(a) Candidate bounding boxes with their predicted label and recognition score.

(b) Final output after filtering and NMS.

Figure 7: Removing false positive bounding boxes using recognition scores.

$S_b(str)$ will be useful for removing false positives generated by the detection stage as well as generate word-level segmentation in the end-to-end recognition stage.

4.3. End-to-end word recognition

The word recognition approach described in Section 4.2 assumes a word-level bounding box is given. As shown in Figure 7, the candidate bounding boxes returned by the text detector may contain a fair amount of false positives, and it is the job of the word level recognizer to filter these out. The word level recognizer evaluates every candidate bounding box, assigning each of them a recognition score. Bounding boxes with classifier scores lower than a threshold are discarded to reduce the false positive rate. We then sort the bounding boxes by their recognition scores, and make predictions on the top-ranked ones, while removing bounding boxes with lower scores which overlap with the predictions using the same NMS approach described in Section 4.1.3.

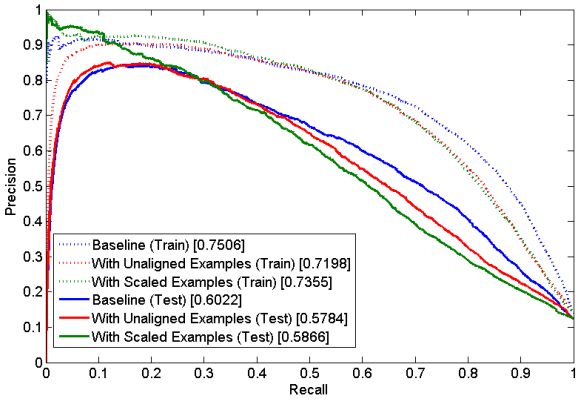


Figure 8: Pixel level precision-recall curves for text detector. Numbers in brackets denote average precision.

5. Experiments

In this section we present a detailed evaluation of our end-to-end system, as well as the subsystems in our pipeline using the ICDAR 2003 dataset [13]. In particular, we compare our text detection results with other similar systems and our character classifier results with [6]. We also evaluate our system on cropped word recognition as well as end-to-end word recognition on full images and compare our results with [11], which has the best known state-of-the-art results on the ICDAR dataset.

5.1. Text detection

We use two metrics to evaluate the performance of the text detector: pixel-level average precision as in [6] as well as the quality of the word-level bounding boxes using precision and recall as defined in [13]. The latter is the standard metric used to assess the performance of text localization and recognition systems over the ICDAR dataset. We first consider the effect of using different datasets to train the detector. The end goal here is to train a detector that is effective at picking out centered and properly scaled characters suitable for the word recognizer. Thus, we consider augmenting the standard dataset described in Section 3.2 with two additional types of synthetic negative examples: examples where the text is off-centered (Figure 3d) and examples where the text is of the incorrect scale (Figure 3e). Using these augmented training sets, we train a CNN and evaluate the average precision of the detector over the full ICDAR dataset. These results are summarized in

Method	Precision	Recall	F-Score
Baseline	0.4686	0.4416	0.4348
Word Segmentation	0.3901	0.6425	0.4702
Realignment	0.4875	0.5831	0.5165
Best Recall	0.3687	0.6626	0.4536

(a) Bounding box estimation steps.

Method	Precision	Recall	F-Score
Pan *et al*. [15]	0.67	0.71	0.69
Epshtein *et al*. [3]	0.73	0.60	0.66
Our approach	**0.49**	**0.58**	**0.52**
HWDavid [13]	0.44	0.46	0.45

(b) Text detection results comparison.

Table 1: Precision, recall and F-scores for different systems on the ICDAR test set [13].

the precision-recall curves in Figure 8. The results indicate that incorporating these additional negative examples tend to hurt the performance of the detector with average precision dropping by about 2-4% in each case. While these examples are better at defining the notion of a "positive," they are also increasing the difficulty of the detection problem. By imposing a strict scale and position requirement on the positive examples, we are also working against the spatial invariance implicit to the structure of the CNN, which in turn, may lead to a weaker detector. Given that, we fall back to the baseline detection system for the remainder of this paper. This detector attains an average pixel-level precision of 0.60, which is comparable to the best performance (0.62) of the much larger network (1000 input filters) in [6].

Having trained a detector, we now apply the full detection pipeline to construct word-level bounding boxes. We assess each step of the post-processing pipeline separately and summarize the results in Table 1a. We begin with the baseline system, which emits a set of bounding boxes for lines of text. Note that we also apply non-maximal suppression to these boxes to reduce the number of candidate boxes. Because the ICDAR test metric is for word-level bounding boxes, the system will be heavily penalized whenever there are multiple words in a single line. Nonetheless, this method yields a combined F-score (harmonic mean of precision and recall) of 0.4348 on the ICDAR test set. A comparison with other text detection systems is provided in Table 1b. The first step in the post-processing aims to slice a line-level bounding box into word-level bounding boxes. Introducing this yields a 4% boost in the F-score to 0.4702. Notice that there is a substantial improvement in the recall of the system from 0.4416 to 0.6424 since the system is now localizing the individual words. There is a corresponding reduction in the precision of the system since the system will emit both the entire line as well as the segmented regions as candidate bounding boxes. Thus, some of them are guaranteed to be false positives; however, we note that we can prune out these false positives in the recognizer stage, so the lowered precision is not a significant problem. Finally, we apply the realignment process, which increases the detector's F-score to 0.5165, which is still below from the state of the art in detection, but reputable nonetheless.

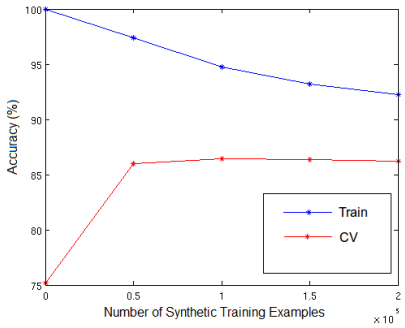


Figure 9: Cross-validation accuracy with varying number of synthetic data examples.

Method	Accuracy
REAL+SYNTH	**84.0%**
REAL	78.2%
Coates, *et al*. [6]	81.7%
Neumann and Matas [14]	67.0%

Table 2: Test recognition accuracy on ICDAR 2003 Character Dataset.

However, as noted earlier, what we are optimizing for is the end-to-end recognition results, in which case recall is more important than precision. Thus, we select the minimum value to be the detection threshold, which translates to a recall of 0.6626.

5.2. Character Classification

To prevent overfitting, we first train our character classifier using a mixture of the ICDAR 2003 sample images [13], the Chars74k dataset [8], the Weinman dataset [17] and varying amounts of synthetic data, and then run cross-validation (CV) over ICDAR 2003 train images. The training (blue) and CV (red) learning curves are shown in Figure 9. We can see that the best CV results are obtained using 100,000 synthetic examples. We then train our final character classifier using the best CV settings with all 113,000 examples (100,000 synthetic examples + all real training examples; we call this setup REAL+SYNTH), and achieve a

Figure 10: Example images in the ICDAR 2003 Robust Word Recognition Dataset.

Method	WD-50	WD-FULL	WD-NO-LEXs
Our approach	**86%**	**73%**	**52%**
Wang, *et al*. [11]	76%	62%	-

Table 3: Cropped word accuracy on ICDAR 2003 Dataset.

Method	K = 5	K = 20	K = 50
Our approach	**.67**	**.65**	**.63**
Wang, *et al*. [11]	.72	.70	.68

Table 4: F-scores for end-to-end evaluation on ICDAR 2003 Dataset.

62-way classifier accuracy of 84% on the ICDAR 2003 test set, which consists of 5198 test images. We also train on the 13,000 real examples alone and achieve an accuracy of 78.2% on the test set (we call this REAL). As shown in Table 2, our result trained with the REAL+SYNTH setup is superior to all other (purpose-built) systems tested on the same problem. We also observe that training with a large number of good-quality synthetic data can significantly improve classification results.

5.3. Cropped Word Recognition

This part of our system is evaluated on the ICDAR 2003 Robust Word Recognition dataset, which contains images of perfectly cropped words as illustrated in Figure 10. We use the exact same testing set-up as [11]. More concretely, we measure word-level accuracy with a lexicon containing all the words from the ICDAR test set (called WD-FULL), and with lexicons consisting of the ground truth words for that image plus 50 random "distractor" words added from the test set (called WD-50). In addition, we also evaluate the performance of our system without a known lexicon. In that case, we feed the raw classifier output string into Hunspell[1] to obtain a set of suggested words, and run the same WSD method using those suggested words as our lexicon (called WD-NO-LEX). Note that in [11], the authors ignored all words shorter than 3 characters, as well as words containing non-alphanumeric characters. We perform the same pre-processing on our data. Table 3 compares our results with [11]. Apart from achieving significantly better results on WD-FULL and WD-50, our system can achieve decent accuracy without relying on a known lexicon in the WD-NO-LEX setup, which is a missing feature from [11].

[1]Hunspell is an open source spell checking software available at http://hunspell.sourceforge.net/

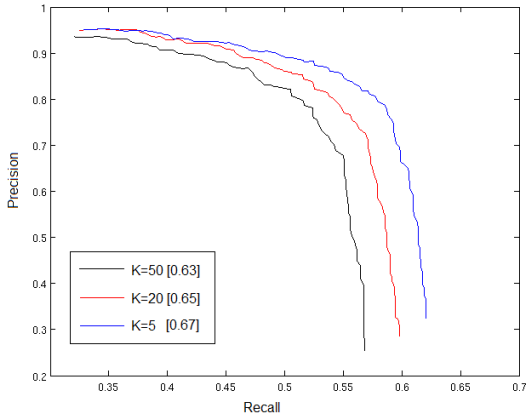


Figure 11: Precision-Recall Curves of our end-to-end word detection and recognition on the ICDAR dataset, with lexicons containing 5, 20, and 50 distractor words. Best F-scores are shown in square brackets.

5.4. End-to-end Word Detection and Recognition

To construct a fair comparison of our end-to-end system, we again test using the exact same settings as [11]. We construct a lexicon for each image by taking the ground truth words in that image and adding K ($K = 5, 20,$ and 50) extra distractor words chosen at random from the test set. Figure 12 shows some sample outputs of our system. As in the previous experiment, words shorter than 3 characters or containing non-alphanumeric characters are removed. We follow the standard evaluation criterion described in [13], where a predicted bounding box is considered to be a true positive if its area intersects a ground truth bounding box by more than 50% of the area of the smallest rectangle that can contain these two bounding boxes, and the words match (ignoring case). Precision and recall can then be calculated as usual: precision is equal to the fraction of true positives in the set of predictions, and recall is the fraction of true positives in the set of all ground truth bounding boxes. Figure 11 shows precision and recall plots for different values of K. As before, we also compute the standard F-score.

As a standard way of summarizing results, we report the highest F-score across the precision-recall curves and compare our results with [11] in Table 4. Our system attains slightly lower F-scores than [11]. However, do note that our system has better results on cropped word recognition, and thus, making further improvements to the text detector can potentially improve upon the full end-to-end system.

6. Conclusion and Future Considerations

In this paper, we consider a new end-to-end scene text recognition system that uses essentially the same architecture for both the detection and recognition subsystems. By

Figure 12: Example outputs of our end-to-end system with $K = 20$.

applying NMS-based heuristics in conjunction with sliding window responses, we outperform state-of-the-art systems in cropped word recognition, and achieve decent results in end-to-end text recognition on the ICDAR 2003 dataset. This demonstrates the possibility of using a single, unified architecture for both detection and recognition in the construction of a full end-to-end system.

As discussed above, to further improve the system, we should focus on improving the detector. In particular, we notice in Figure 11 that recall is significantly lower than precision, which means that many words are simply not captured by the detector. This could be because the candidate bounding boxes generated by the text detector are not bound tightly (of the wrong scale). Very often, these boxes are assigned low scores by the word recognizer, and subsequently removed by NMS. This problem can potentially be fixed by more careful tuning of the NMS during the text detection stage or by training context-dependent classifiers that take into account not only the responses of a single window, but also that of its neighboring windows. By increasing the amount of information available to the detector, it should be able to make more informed decisions, thus yielding higher quality bounding boxes.

References

[1] L. G. A. Neubeck. Efficient non-maximum suppression. In *ICPR*, 2006. 5, 11

[2] S. An, P. Peursum, W. Liu, and S. Venkatesh. Efficient algorithms for subwindow search in object detection and localization. In *IEEE*, 2009. 4, 11

[3] Y. W. B. Epshtein, E. Oyek. Detecting text in natural scenes with stroke width transform. In *CVPR*, 2010. 1, 2, 8

[4] C. Barat, C. Ducottet, É. Fromont, A.-C. Legrand, and M. Sebban. Weighted symbols-based edit distance for string-structured image classification. In *ECML/PKDD (1)*, pages 72–86, 2010. 6

[5] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. High performance neural networks for visual object classification. Technical Report IDSIA-01-11, Dalle Molle Institute for Artificial Intelligence, 2011. 2, 11

[6] A. Coates, B. Carpenter, C. Case, S. Satheesh, B. Suresh, T. Wang, D. J. Wu, and A. Y. Ng. Text detection and character recognition in scene images with unsupervised feature learning. In *ICDAR*, 2011. 2, 3, 4, 7, 8, 11

[7] A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011. 2, 11

[8] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *VISAPP*, February 2009. 1, 3, 8

[9] W. N. Francis and H. Kucera. Brown corpus manual. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island, US, 1979. 3

[10] A. Hyvarinen and E. Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000. 2

[11] S. B. K. Wang, B. Babenko. End-to-end scene text recognition. In *ICCV*, 2011. 1, 3, 7, 9, 11

[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989. 2, 11

[13] S. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young. ICDAR 2003 robust reading competitions. *ICDAR*, 2003. 3, 7, 8, 9

[14] L. Neumann and J. Matas. A method for text localization and recognition in real-world images. In *AACCV*, 2010. 2, 8

[15] Y. Pan, X. Hou, and C. Liu. Text localization in natural scene images based on conditional random field. In *ICDAR*, 2009. 1, 8

[16] Z. Saidane and C. Garcia. Automatic scene text recognition using a convolutional neural network. In *Workshop on Camera-Based Document Analysis and Recognition*, 2007. 2, 11

[17] J. Weinman, E. Learned-Miller, and A. R. Hanson. Scene text recognition using similarity and a lexicon with sparse belief propagation. In *Transactions on Pattern Analysis and Machine Intelligence*, volume 31, 2009. 3, 8

[18] J. J. Weinman, E. Learned-Miller, and A. R. Hanson. A discriminative semi-markov model for robust scene text recognition. In *Proc. IAPR International Conference on Pattern Recognition*, Dec. 2008. 2

7. Appendix

This project was done in collaboration with Adam Coates in Professor Andrew Ng's lab. In this section, we summarize the components of this project that directly pertain to computer vision as well as our own contributions, as per the requirements.

7.1. Computer Vision Components

This project focuses on the task of scene text recognition, which is a core problem in computer vision. Convolutional neural networks have been applied with great success to many problems in computer vision [5, 12, 16], as has unsupervised feature learning [6, 7]. In this project, we have elected to take an existing and high-performing framework and apply it to the field of text recognition. Once we have trained a text detector and recognizer, the way we evaluate it over full images draws upon standard techniques in computer vision: sliding windows across multiple scales, finding maximum-sum subregions for bounding box estimation [2] and applying non-maximal suppression to filter results [1]. In a sense, the entire post-processing pipeline we described for text detection is a standard object recognition system where we try to apply a single, highly accurate object detector to full images in order to localize objects of interest (the object of interest being text in this case). The word-level recognizer uses a lexicon-based search strategy not too different from the ideas proposed in [11] to go from character-level responses to full word recognition. Overall, the primary focus of this project has been to use the low-level representations from a CNN to construct a complete high-level system that can extract and recognize text from natural images. The majority of the project was spent constructing this high-level, text-recognition system using a single, unified low-level framework.

7.2. Project Contributions

This project is a major part of the Photo OCR project that we are working on in Professor Ng's lab. As part of this, we hold weekly meeting with Adam Coates, during which we discuss potential ideas and further experiments. Apart from Adam however, there are no other collaborators on this project at this time. Additionally, this research is not part of any other course projects that we have done, either past or present. The two of us are the main collaborators on this project and the vast majority of the code has been written by us. There are two main exceptions to this: for training the convolutional neural network, we used Jiquan Ngiam's CNN toolbox and for setting up distributed jobs, we used Adam Coates' distributed MapReduce framework. Both Jiquan and Adam are students in Professor Ng's lab. Note that in the case of training the CNN, we wrote the code that computes the L_2-SVM loss for our particular setup. There is also some legacy code that we wrote before this quarter that we incorporated into the project, most notably the code for K-means pretraining and some of the code for generating synthetic data. For the new system (CNN, bounding box computation, word recognition, and full end-to-end), all components were written in the course of this quarter.

To some extent, our work is an extension of the system described in our previous ICDAR paper [6]. At the same time, we have made substantial modifications to it since this summer, particularly with the focus on the construction of an end-to-end system. For instance, in the case of the text detector, we are using a new training set based on synthetic examples and are training a CNN rather than using just a single fully connected network. In the case of the character recognizer, we have also moved towards using a multi-layered convolutional neural network rather than a single layer of pretrained filters. Furthermore, all of the post-processing pipeline that we have described above, the word-level recognition using a lexicon-backed WSD, and the various applications of NMS were formulated and implemented by us during the course of this project. All of the analysis described in this paper is largely our own, although Adam did offer some insights into improving the system. This project constitutes a portion of the honors thesis (David) and Master's thesis (Tao) that we are working on this year.

We thus affirm that we are the sole authors of the work described in this paper.

7.3. Note on Code Submission

Note that in our code submission, we have omitted Jiquan's CNN toolbox and Adam's distributed system framework. All the code that we have included has been written by us. Please let us know if you would like to discuss the CNN and distributed systems codes.

7.4. Note on Future Distribution

Since we may write a research paper based on ideas discussed in this project, please DO NOT post this report on the course website.