MSCS with Distinction in Research Final Report
Scene Text Recognition with Convolutional Neural Networks

Tao Wang
Stanford University, 353 Serra Mall, Stanford, CA 94305
twangcat@cs.stanford.edu
Primary Advisor: Andrew Y. Ng
Secondary Advisor: Daphne Koller

Abstract

Full end-to-end text recognition in natural images is a challenging problem that has received much attention recently. Traditional systems in this area have relied on elaborate models incorporating carefully hand-engineered features or large amounts of prior knowledge. In this work, we take a different route and combine the representational power of large, multilayer neural networks together with recent developments in unsupervised feature learning, which allows us to use a common framework to train highly-accurate text detector and character recognizer modules. Then, using only off-the-shelf methods with simple engineering, we integrate these two modules into a full end-to-end, lexicon-driven, scene text recognition system that achieves state-of-the-art performance on standard benchmarks, namely Street View Text and ICDAR 2003.

1 Introduction

Extracting textual information from natural images is a challenging problem with many practical applications. Unlike character recognition for scanned documents, recognizing text in unconstrained images is complicated by a wide range of variations in backgrounds, textures, fonts, and lighting conditions. As a result, many text detection and recognition systems rely on cleverly hand-engineered features [7, 6, 18] to represent the underlying data. Sophisticated models such as conditional random fields [15, 26] or pictorial structures [24] are also often required to combine the raw detection/recognition outputs into a complete system.

In this paper, we attack the problem from a different angle. For low-level data representation, we use an unsupervised feature learning (UFL) algorithm that can automatically extract features from the given data. Such algorithms have enjoyed numerous successes in many related fields such as visual recognition [4] and action recognition [11]. In the case of text recognition, the system in [3] achieves competitive results in both text detection and character recognition using a simple and scalable feature learning architecture incorporating very little hand-engineering and prior knowledge.

We integrate these learned features into a large, discriminatively-trained convolutional neural network (CNN). CNNs have enjoyed many successes in similar problems such as handwriting recognition [12], visual object recognition [2], and character recognition [21]. By leveraging the representational power of these networks, we are able to train highly accurate text detection and character recognition modules. Using these modules, we can build an end-to-end system with post-processing techniques which only requires simple engineering like non-maximal suppression (NMS)[17] and beam search [20]. Despite its simplicity, our system achieves state-of-the-art performance on standard test sets.

2 Our Previous Work

Prior to this work, we went through a number of related investigations, which allow us to gain enough evidence and insight to develop the final end-to-end system. Here we briefly discuss some of our previous works.

Our approach to build detection and recognition modules is largely inspired by the previous work of [3], in which we achieved state-of-the-art results on cropped character recognition using unsupervised feature learning and a one-layer CNN. The first layer filters are pre-trained using a variant of K-means, and the top-layer

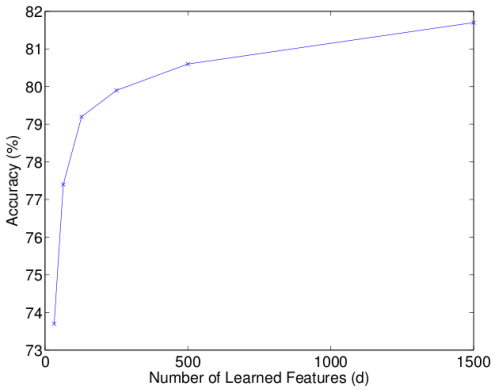
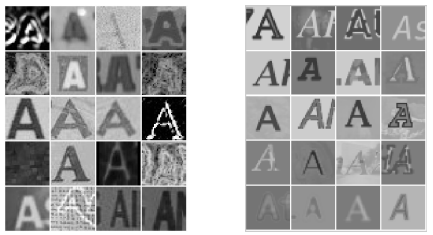


Figure 1: Character classification accuracy on ICDAR as a function of number of first layer filters in [3].



(a) Distorted ICDAR examples.

(b) Synthetic examples

Figure 2: Distorted and synthetic data used in [3].

classifier is trained using supervised learning with labeled examples. Experimental results in [3] suggest that classification accuracy consistently increases as a function of the number of learned features, as shown in Figure 1. We also experimented with different ways of data augmentation, including distortions and artificially synthesized examples (See Figure 2).

As another form of contribution to the scene text recognition field as well as the more general computer vision community, we introduced the Street View House Numbers (SVHN) dataset in [16], which focuses on a restricted instance of the scene text recognition problem: reading digits from house numbers in street level images. As shown in Figure 3, SVHN contains images of small cropped digits obtained from house numbers in Google Street View images. It has similar style as MNIST [5], but incorporates an order of magnitude more labeled data (over 600,000 digit images) and represents the significantly harder, unsolved problem of recognizing digits and numbers in natural scene images. In order to obtain the large number of cropped and labeled digits with a relatively low cost, we leveraged the Amazon Mechanical Turk (AMT) annotation toolbox introduced in [23]. We pass the raw images through

Figure 3: Examples from the SVHN dataset

Figure 4: Example annotations by AMT workers

multiple AMT tasks to obtain first the word level, and then the character level bounding boxes. Some example result annotations by workers on AMT are shown in Figure 4.

Apart from introducing the new SVHN dataset, we also perform a number of experiments to compare different classifier models in [16]. As shown in Figure 5, the model pretrained with our variant K-means feature learning approach outperformed other models such as HOG and stacked auto-encoders (SAE). We also noticed that classification accuracy increases steadily as more training examples are used. With findings in these two works, we continue to explore the potential of larger CNN models with multiple stacked layers, as well as larger training set sizes.

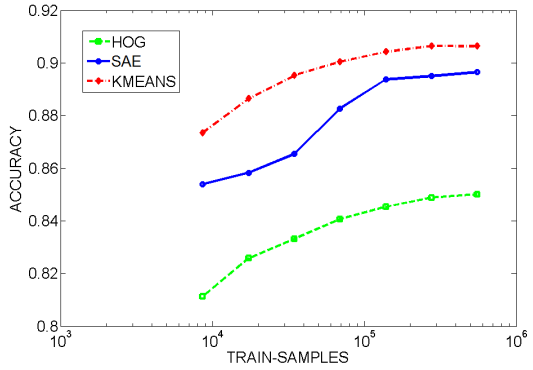


Figure 5: 10-way Digit classification accuracy on SVHN as a function of number of training examples in [16].

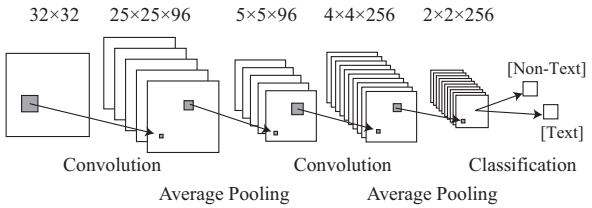


Figure 6: CNN used for text detection.

3 Learning Architecture

In this section, we describe our text detector and character recognizer modules, which are the essential building blocks of our full end-to-end system.

Given a 32-by-32 pixel window, the detector decides whether the window contains a centered character. Similarly, the recognizer decides which of 62 characters (26 uppercase, 26 lowercase letters, and 10 digits) is in the window. As described at length in Section 4, we slide the detector across a full scene image to identify candidate lines of text, on which we perform word-level segmentation and recognition to obtain the end-to-end results.

For both detection and recognition, we use a multi-layer, convolutional neural network (CNN) similar to [12, 21]. Our networks have two convolutional layers with n_1 and n_2 filters respectively. The network we use for detection with $n_1 = 96$ and $n_2 = 256$ is shown in Figure 6, while a larger, but structurally identical one ($n_1 = 115$ and $n_2 = 720$) is used for recognition.

3.1 Dataset

For text detection, we train a binary classifier that decides whether a single 32-by-32 subwindow contains text or not. Unlike our previous work [3], here 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.

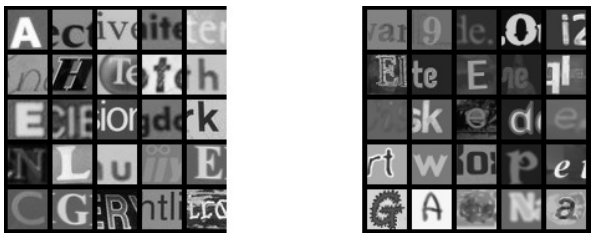
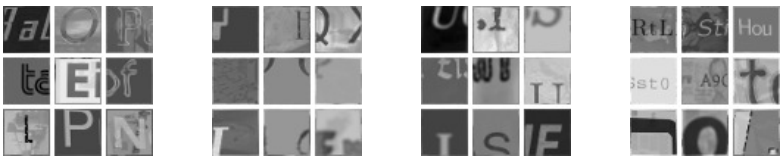


Figure 7: Examples from our training set. Left: from ICDAR. Right: synthetic data

Synthetic Training Data In our previous work [3], we have used large synthetic datasets to achieve better classification results. In this paper, we improved our text synthesizer to generate higher quality synthetic training images using a wider range of fonts. The number of images per character class are distributed according to the unigram frequency obtained from the Brown Corpus [8] so as to simulate the natural distribution of character classes. In [3] we generate grayscale levels of characters and the background from a uniform distribution. In this work, these grayscale levels 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 7. Our improved synthetic examples are much more realistic compared to the artificial data (Figure 2) we used in [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 illustrated in Figure 8. As such, we have compiled a dataset consisting of examples from the ICDAR 2003 training images [14], the English subset of the Chars74k dataset [6], and the sign-reading dataset from Weinman, *et al*. [25], as well as synthetically generated examples. In training the detec-



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

Figure 8: Synthetic images used to train the detector and classifiers.

tor, our training set generally consists of 75,000 positive examples and 150,000 negative examples; for the classification task, we use about 63,000 examples in total.

3.2 Unsupervised Feature Learning

We begin by using an unsupervised learning algorithm to pretrain the filters used for both detection and recognition. In recent years there has been many successes of unsupervised feature learning (UFL) algorithms in numerous domains. These algorithms attempt to learn feature representations directly from the data, in contrast to hand-engineered features that incorporates prior knowledge. Some well-known UFL algorithms include sparse auto-encoder [1], sparse coding [19], Restricted Boltzmann Machine (RBM) [9] and Independent Component Analysis (ICA) [10].

Here, we use a UFL pipeline that resembles that described in [3, 4]. We briefly outline the key components of this system:

1. Collect a set of m small image patches from the training set. As in [3], 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 preprocessed patches $x^{(i)}$ to build a mapping from an input image I to its features z. In this paper, we adhere to the variant of the K-means algorithm described in [3] where we learn a dictionary $D \in \mathbb{R}^{64 \times n_1}$ containing n_1 normalized basis vectors. Having learnt D, we describe how to compute features z in Section 3.3.

In Figure 9, we have shown some of the learnt first layer filters.

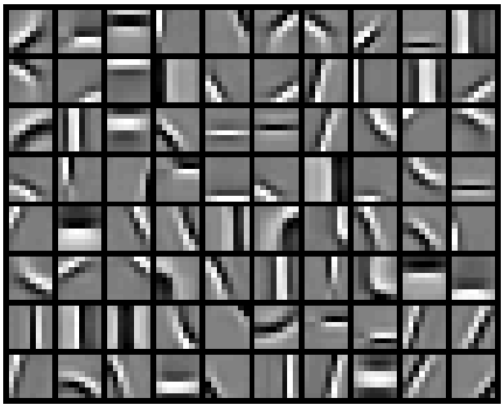


Figure 9: Examples of the our learnt first layer filters

3.3 Convolutional and Pooling Layers

The two-layer convolutional architecture we use for text detection is given in Figure 6. Note that we use the same architecture to train the character classifier, except that $n_1 = 115$ and $n_2 = 720$. The larger number of features 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 the set of learnt filters D. More specifically, given a filter (kernel) D_j (which is obtained by reshaping one row of D to 8-by-8 pixels) and a 32-by-32 input image I, the response is given by $r_j = I \star D_j$ where $\star$ denotes the 2-D convolution operator. Note that we only evaluate r_j's 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. After evaluating each filter, we stack all r_j's to arrive at a 25-by-25-by-n_1 response r as the output for the first layer. As in [3], we apply a scalar, nonlinear activation function to the responses to obtain the first layer features $z = \max\{0, |r| - \alpha\}$ where α is a hyperparameter. In this work, we take $\alpha = 0.5$.

As is standard in the literature on convolutional ar-

chitectures, we now apply a spatial pooling step. This has the benefit of reducing the dimensionality of the features 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-n_1 features. 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-n_2 response. This output feeds into a fully connected classification layer which is trained with supervised fine-tuning.

3.4 Supervised Fine-tuning

In [3], we only train the top-layer classifier with supervised learning, and the low-level features were trained with unsupervised learning In this work, we discriminatively train the network by backpropagating the L_2-SVM classification error in the form of a squared hinge loss: $\max\{0, 1 - \theta^\mathsf{T} x\}^2$, while we fix the filters in the first convolution layer (D learned from K-means). Given the size of the networks, fine-tuning is performed by distributing data across multiple machines using a map-reduce framework.

4 End-to-End Pipeline Integration

Our full end-to-end system combines a *lexicon* with our detection/recognition modules using post-processing techniques including NMS and beam search. Here we assume that we are given a lexicon (a list of tens to hundreds of candidate words) for a particular image. As argued in [24], this is often a valid assumption as we can use prior knowledge to constrain the search to just certain words in many applications. The pipeline mainly involves the following two stages:

(i) We run sliding window detection over high resolution input images to obtain a set of candidate lines of text. Using these detector responses, we also estimate locations for the spaces in the line.

(ii) We integrate the character responses with the candidate spacings using beam search [20] to obtain full end-to-end results.

These two stages are described in details below.

4.1 Text Line Detection

First, given an input image, we identify horizontal lines of text using multiscale, sliding window detection. At each scale s, we evaluate the detector response $R_s[x, y]$ at each point (x, y) in the scaled image. As

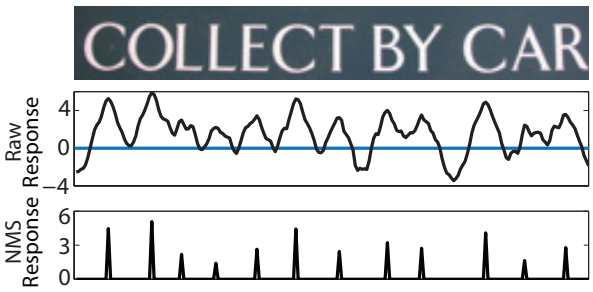


Figure 10: Detector responses in a line.

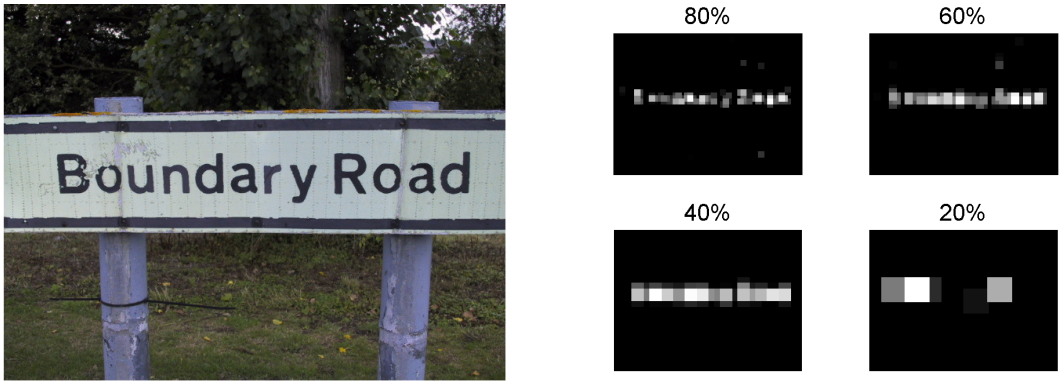
shown in Figure 11b, windows centered on single characters at the right scale produce positive $R_s[x, y]$, represented by the bright squares. We apply NMS [17] to $R_s[x, r]$ in each individual row r to estimate the character locations on a horizontal line. In particular, we define the NMS response

$$\tilde{R}_s[x,r] = \begin{cases} R_s[x,r] & \text{if } R_s[x,r] \geq R_s[x',r], \\ & \quad \forall x' \; s.t. \; |x'-x| < \delta \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where δ is some width parameter. For a row r with non-zero $\tilde{R}_s[x, r]$, we form a line-level bounding box L_s^r (see Figure 10) with the same height as the sliding window at scale s. The left and right boundaries of L_s^r are defined as $\min\{x : \tilde{R}_s[x, r] > 0\}$ and $\max\{x : \tilde{R}_s[x, r] > 0\}$, which are the locations of the left and right most peaks of $\tilde{R}_s[x, r]$. This yields a set of possibly overlapping line-level bounding boxes. We score each box by averaging the nonzero peak values of $\tilde{R}_s[x, r]$ over the number of peaks. We then apply standard NMS to remove all L's that overlaps by more than 50% with another box of a higher score, and obtain the final set of line-level bounding boxes $\tilde{\mathbf{L}}$. Since gaps between words produce sharply negative responses, we also estimate possible space locations within each L_s^r by applying the same NMS technique as above to the negative responses. Note that this heuristic assumes that text run horizontally, which is true in most real cases. In practice, we found our approach capable of detecting text lines with angles that are less than 10 degrees to the x-axis of the image.

4.2 Joint Word Segmentation and Recognition

After identifying the horizontal lines of text, we jointly segment the lines of text into words and recognize each word in the line. Given a line-level bounding box L and its candidate space locations, we evaluate each possible word-level bounding boxes using a



(a) Sample image from ICDAR dataset.

(b) Detector responses from multiple scales.

Figure 11: Multi-scale response from text detector.

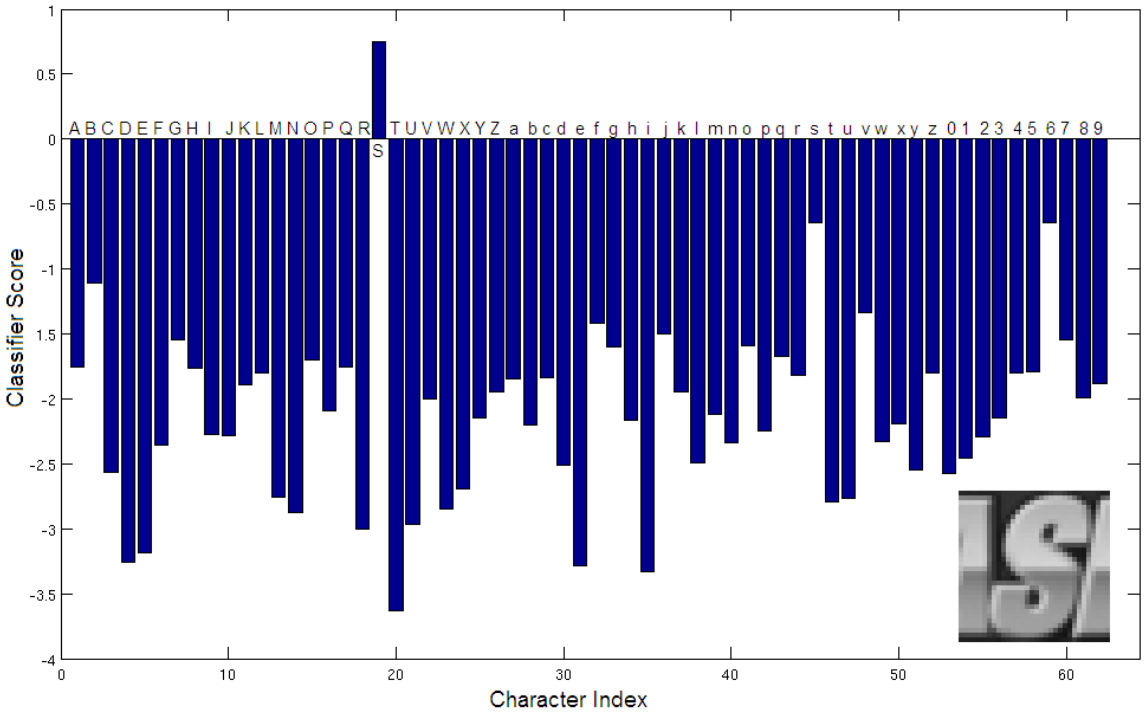


Figure 12: Character classifier scores over an example patch

Viterbi-style algorithm and find the best segmentation scheme using a beam search technique similar to [13]. To evaluate a word-level bounding box B, we slide the character recognizer across it and obtain a $62 \times N$ score matrix M, where N is the number of sliding windows within the bounding box. Figure 12 shows the plot of the elements in one column of M, which corresponds to the 62-way classifier outputs in one sliding window. Intuitively, a more positive $M(i, j)$ suggests a higher chance that the character with index i is centered on the location of the j^{th} window. Similar to the detection phase, we perform NMS over M to select the columns where a character is most likely to be present. The other columns of M are set to $-\infty$. We then find the lexicon word w^* that best matches a score matrix M as follows: given a lexicon word w, compute the alignment score

$$S_M^w = \max_{l^w \in L^w} \left(\sum_k^{|w|} M(w_k, l_k^w) \right) \quad (2)$$

where l^w is the alignment vector[1] between the characters in w and the columns of M. S_M^w can be computed efficiently using a Viterbi-style alignment algorithm similar to [22]. In practice, we also augment S_M^w with additional terms that encourage geometric consistency. For example, we penalize character spacings that are either too narrow or vary a lot within a single word. We compute S_M^w for all lexicon words and label the word-level bounding-box B with the highest scoring word w^*. We take $S_B = S_M^{w^*}$ to be the *recognition score* of B.

Having defined the recognition score for a single bounding box, we can now systematically evaluate possible word-level segmentations using beam search [20], a variant of breadth first search that explores the top N possible partial segmentations according to some heuristic score. In our case, the heuristic score of a candidate segmentation is the sum of the S_B's over all the resulting bounding boxes in a line of text L.

Figure 13 illustrates two beam search steps on one of the text lines. At the top of the figure is a line-level bounding box generated by the procedure described in Section 4.1. The vertical blue lines denote the candidate space positions. We have set the threshold for generating candidate spaces relatively low, so that we capture almost every true space, at the expense of many false spaces. Given the number of candidate spaces p ($p = 5$ in Figure 13), there are $p + 1$ candidate segments. Each segment can stand alone as a single word, or join with adjacent segments to form a longer word. For every predicted word, we compute its associated recognition

[1]For example, $l_4^w = 6$ means the 4^{th} character in w aligns with the 6^{th} column of M, or the 6^{th} sliding window in a line of text.

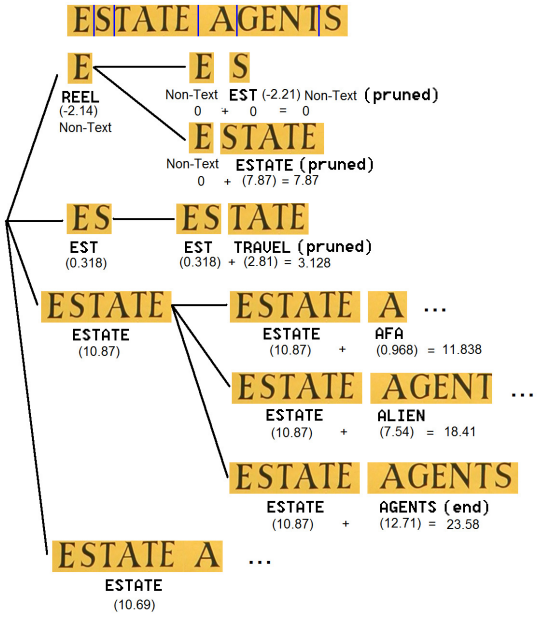


Figure 13: Illustration of beam search steps on a single line of text. Segmentations that are clearly wrong are pruned early due to low recognition scores

score S_B. Words with S_B's lower than a threshold t are treated as "Non-Text", and their S_B's are set to t. In Figure 13 we used $t = 0$, but in practice we vary t to obtain precision-recall curves. In the first branch of Figure 13, the segment with actual letter 'E' is recognized as word 'REEL' with $S_B = -2.14$ (the number in brackets), and thus assigned as Non-Text. In the second branch, segments 'E' and 'S' are joint and recognized as the word 'EST' with $S_B = 0.318$.

The beam search algorithm tries to find an assignment to the segments so that the sum of S_B's is maximized. In principle, the search algorithm has to enumerate exponential number of assignment schemes in the worst case. However, we find that keeping the top $N = 60$ search paths works well. In Figure 13, we use $N = 4$ for clarity. Notice that at depth 2, only the top 4 scoring search paths survive, and the rest are pruned. The beam search algorithm stops when all paths are pruned or reach the end, and the highest scoring path gives the predicted words.

5 Experimental Results

5.1 Results on ICDAR 2003 and SVT Benchmarks

In this section we present a detailed evaluation of our text recognition pipeline. We measure cropped charac-

Table 1: Cropped word recognition accuracies on ICDAR 2003 and SVT

Benchmark	I-WD-50	I-WD	SVT-WD
Our approach	**90%**	**84%**	70%
Wang, *et al*. [24]	76%	62%	57%
Mishra, *et al*. [15]	82%	-	**73%**

Figure 14: Examples from the I-WD benchmark

ter and word recognition accuracies, as well as end-to-end text recognition performance of our system on the ICDAR 2003 [14] and the Street View Text (SVT) [24] datasets. Apart from that, we also perform additional analysis to evaluate the importance of model size on different stages of the pipeline.

First we evaluate our character recognizer module on the ICDAR 2003 dataset. Our 62-way character classifier achieves state-of-the-art accuracy of 83.9% on cropped characters from the ICDAR 2003 test set. The best known previous result on the same benchmark is 81.7% reported by [3]

Our word recognition sub-system is evaluated on images of perfectly cropped words from the ICDAR 2003 and SVT datasets, as illustrated in Figure 14 We use the exact same test setup as [24]. More concretely, we measure word-level accuracy with a lexicon containing all the words from the ICDAR test set (called I-WD), and with lexicons consisting of the ground truth words for that image plus 50 random "distractor" words added from the test set (called I-WD-50). For the SVT dataset, we used the provided lexicons to evaluate the accuracy (called SVT-WD). Table 1 compares our results with [24] and the very recent work of [15].

We evaluate our final end-to-end system on both the ICDAR 2003 and SVT datasets, where we locate and recognize words in full scene images given a lexicon. For the SVT dataset, we use the provided lexicons; for the ICDAR 2003 dataset, we used lexicons of 5, 20 and 50 distractor words provided by the authors of [24], as well as the "FULL" lexicon consisting of all words in the test set. We call these benchmarks I-5, I-20, I-50 and I-FULL respectively. Like [24], we only consider alphanumeric words with at least 3 characters. Figure 16 shows some sample outputs of our system. We follow the standard evaluation criterion described in [14] to compute the precision and recall. Figure 15 shows precision and recall plots for the different benchmarks on the ICDAR 2003 dataset.

As a standard way of summarizing results, we also

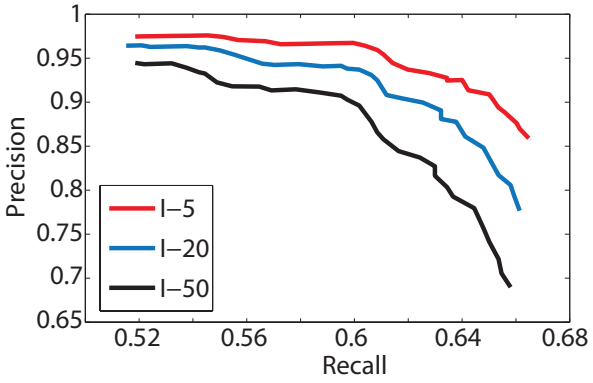


Figure 15: End-to-end PR curves on ICDAR 2003 dataset using lexicons with 5, 20, and 50 distractor words.

report the highest F-scores over the PR curves and compare with [24] in Table 2. Our system achieves higher F-scores in every case. Moreover, the margin of improvement is much higher on the harder benchmarks (0.16 for I-FULL and 0.08 for SVT), suggesting that our system is robust in more general settings.

5.2 Extension to a General Lexicon

In addition to settings with a known lexicon, we also extend our system to the more general setting by using a large lexicon $\mathbb{L}$ of common words. Since it is infeasible to search over all lexicon words in this case, we limit our search to a small subset $P \in \mathbb{L}$ of "visually plausible" words. We first perform NMS on the score matrix M across positions and character classes to obtain a set of response peaks similar to the ones seen in Figure 10, but with each peak representing an appropriate character class at that location. We then threshold this response with different values to obtain a set of raw strings. The raw strings are fed into Hunspell[2] to yield a set of suggested words as our smaller lexicon P. Using this simple setup, we achieve scores of 0.54/0.30/0.38 (precision/recall/F-score) on the ICDAR dataset. This is comparable to the best known result 0.42/0.39/0.40 obtained with a general lexicon by [18].

5.3 Control Experiments on Model Size

In order to analyze the impact of model size on different stages of the pipeline, we also train detection and

[2]Hunspell is an open source spell checking software available at http://hunspell.sourceforge.net/. We augment Hunspell's default lexicon with a corpus of English proper names to better handle text in scenes.

Figure 16: Example output bounding boxes of our end-to-end system on I-FULL and SVT benchmarks. Green: correct detections. Red: false positives. Blue: misses.

Table 2: F-scores from end-to-end evaluation on ICDAR 2003 and SVT datasets.

Benchmark	I-5	I-20	I-50	I-FULL	SVT
Our approach	**.76**	**.74**	**.72**	**.67**	**.46**
Wang, *et al*. [24]	.72	.70	.68	.51	.38

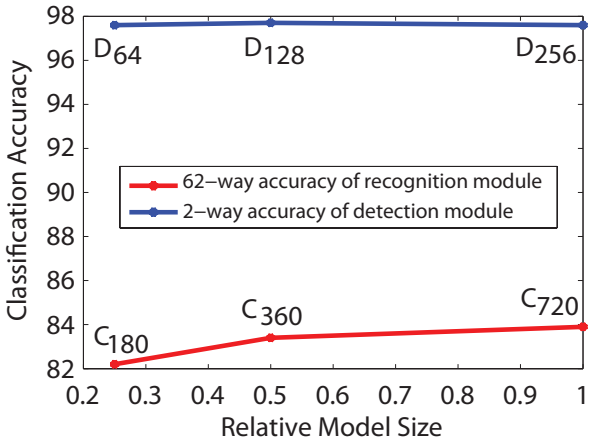


Figure 17: Accuracies of the detection and recognition modules on cropped patches

recognition modules with fewer second layer convolutional filters. The detection modules have $n_2 = 64$ and 128 compared to 256 in our full model. We call the detection modules D_{64}, D_{128} and D_{256} respectively. Similarly, we call the recognition modules C_{180}, C_{360} and C_{720}, which corresponds to $n_2 = 180$, 360 and 720. The smaller models have about $^1/_4$ and $^1/_2$ number of learnable parameters compared to the full models.

To evaluate the performance of the detection modules, we construct a 2-way (character vs. non-character) classification dataset by cropping patches from the ICDAR test images. The recognition modules are evaluated on cropped characters only. As shown in Figure 17, the 62-way classification accuracy increases as model size gets larger, while the 2-way classification results remain unchanged. This suggests that larger model sizes yield better recognition modules, but not necessarily better detection modules.

Table 3: Classification and end-to-end results of different recognition modules

Recognition module	C_{180}	C_{360}	C_{720}
Classification accuracy	82.2%	83.4%	83.9%
End-to-end F-score	.6330	.6333	.6723

Finally, we evaluate the 3 different recognition modules on the I-FULL benchmark, with D_{256} as the detector for all 3 cases. The end-to-end F-scores are listed against the respective classification accuracies in Table 3. The results suggests that higher character classification accuracy does give rise to better end-to-end results. This trend is consistent with the findings of [16] on house number recognition in natural images.

6 Conclusion

In this paper, we have considered a novel approach for end-to-end text recognition. By leveraging large, multi-layer CNNs, we train powerful and robust text detection and recognition modules. Because of this increase in representational power, we are able to use simple non-maximal suppression and beam search techniques to construct a complete system. This represents a departure from previous systems which have generally relied on intricate graphical models or elaborately hand-engineered systems. As evidence of the power of this approach, we have demonstrated state-of-the-art results in character recognition as well as lexicon-driven cropped word recognition and end-to-end recognition. Even more, we can easily extend our model to the general-purpose setting by leveraging conventional open-source spell checkers and in doing so, achieve per-

formance comparable to state-of-the-art.

Acknowledgements

This work is acomplished in close colaboration with David J. Wu. David developed the text detection modules which are the key compenents of our final end-to-end system. I would also like to thank Adam Coates, who gave us many invaluable advices. This work would not be possible without the great support from Adam and David.

References

[1] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Neural Information Processing Systems*, 2006.

[2] 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.

[3] 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.

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

[5] Courant Institute, NYU and Google Labs, New York. *The MNIST Database*.

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

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

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

[9] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

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

[11] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011.

[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.

[13] C.-L. Liu, M. Koga, and H. Fujisawa. Lexicon-driven segmentation and recognition of handwritten character strings for japanese address reading. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(11):1425–1437, Nov. 2002.

[14] S. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young. ICDAR 2003 robust reading competitions. *ICDAR*, 2003.

[15] A. Mishra, K. Alahari, and C. V. Jawahar. Top-down and bottom-up cues for scene text recognition. In *CVPR*, 2012.

[16] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[17] A. Neubeck and L. Gool. Efficient non-maximum suppression. In *ICPR*, 2006.

[18] L. Neumann and J. Matas. A method for text localization and recognition in real-world images. In *ACCV*, 2010.

[19] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[20] S. J. Russell, P. Norvig, J. F. Candy, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

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

[22] S. Sarawagi and W. W. Cohen. Semi-markov conditional random fields for information extraction. In *NIPS*, pages 1185–1192, 2004.

[23] A. Sorokin and D. Forsyth. Utility data annotation with amazon mechanical turk. In *First IEEE Workshop on Internet Vision at CVPR*, 2008.

[24] K. Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. In *ICCV*, 2011.

[25] 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.

[26] J. J. Weinman, E. Learned-Miller, and A. R. Hanson. A discriminative semi-markov model for robust scene text recognition. In *ICPR*, Dec. 2008.