END-TO-END TEXT RECOGNITION WITH CONVOLUTIONAL NEURAL NETWORKS

AN HONORS THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
OF STANFORD UNIVERSITY

David J. Wu
Principal Adviser: Andrew Y. Ng
May 2012

Abstract

Full end-to-end text recognition in natural images is a challenging problem that has recently received much attention in computer vision and machine learning. Traditional systems in this area have relied on elaborate models that incorporate carefully hand-engineered features or large amounts of prior knowledge. In this thesis, I describe an alternative approach that combines the representational power of large, multilayer neural networks with recent developments in unsupervised feature learning. This particular approach enables us to train highly accurate text detection and character recognition modules. Because of the high degree of accuracy and robustness of these detection and recognition modules, it becomes possible to integrate them into a full end-to-end, lexicon-driven, scene text recognition system using only simple off-the-shelf techniques. In doing so, we demonstrate state-of-the-art performance on standard benchmarks in both cropped-word recognition as well as full end-to-end text recognition.

Acknowledgements

First and foremost, I would like to thank my adviser, Andrew Ng, for his advice and mentorship throughout the past two years. His class on machine learning first sparked my interest in the field of artificial intelligence and deep learning; his willingness to let me work in his lab has helped me refine my own research interests and convinced me to pursue graduate studies.

Special thanks also to Adam Coates for the tremendous support and guidance he has provided me over these past two years. From him, I have learned an immense amount about the practical side of machine learning and how to get algorithms to work well in practice. I also want to thank him for giving me the opportunity to work on so many projects, even when I was just a freshman with absolutely no background in machine learning. I would also like to thank Tao Wang for the invaluable advice he provided through the course of this project. The work presented in this thesis is the joint work of our collaboration; none of it would have been possible without his input and ideas. Both Adam and Tao have contributed countless hours to bring this project to a successful completion. It has truly been both a pleasure and a privilege for me to have worked with them. For that, I thank them both.

I would like to thank Hoon Cho for being a great source of insights and lively discussion, and most of all, for being a great friend. Without his input, I would probably not have pursued this thesis in the first place. I would also like to thank Will Zou for allowing me the opportunity to continue working in the lab after the completion of this project. Thanks also to Mary McDevitt for her insightful comments and proofreading of an early draft of this thesis.

Finally, I would like to thank my family for their unconditional support both before and during my undergraduate career. To my grandparents, I thank you for sparking my curiosity and for instilling within me a passion for learning so early on in my childhood. To

my parents, I thank you for your ever-present advice and encouragement. Without your support, this work would not have been possible. This thesis is for you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A system that can automatically locate and recognize text in natural images has many practical applications. For instance, such a system can be instrumental in helping visually impaired users navigate in different environments, such as grocery stores [28] or city landscapes [3], or in providing an additional source of information to an autonomous navigation system. More generally, text in natural images provides a rich source of information about the underlying image or scene.

At the same time, however, text recognition in natural images has its own set of difficulties. While state-of-the-art methods generally achieve nearly perfect performance on object character recognition (OCR) for scanned documents, the more general problem of recognizing text in unconstrained images is far from solved. Recognizing text in scene images is much more challenging due to the many possible variations in backgrounds, textures, fonts, and lighting conditions that are present in such images. Consequently, building a full end-to-end text recognition system requires us to develop models and representations that are robust to these variations. Not surprisingly, current high-performing text detection and character recognition systems have employed cleverly hand-engineered features [10, 11] to both capture the details of and represent the underlying data. In many cases, sophisticated models such as conditional random fields (CRFs) [32] or pictorial-structure models [38] are also necessary to combine the raw detection or recognition responses into a complete system.

In this thesis, I describe an alternative approach to this problem of text recognition based upon recent advances in machine learning, and more precisely, unsupervised feature learning. These feature-learning algorithms are designed to automatically learn low-level

representations from the underlying data [8, 15, 16, 19, 23, 33] and thus present one alternative to hand-engineering the features used for representation. Such algorithms have already enjoyed numerous successes in many related fields, such as visual recognition [42] and action classification [20]. In the case of text recognition, the system in [7] has achieved solid results in text detection and character recognition using a simple and scalable feature-learning architecture that relies very little on feature-engineering or prior knowledge.

By leveraging these feature-learning algorithms, we were able to derive a set of specialized features tuned particularly for the text recognition problem. These learned features were then integrated into a larger, discriminatively-trained convolutional neural network (CNN). CNNs are hierarchical neural networks that have immense representational capacity and have been successfully applied to many problems such as handwriting recognition [21], visual object recognition [4], and character recognition [35]. By tapping into the representational power of these architectures, we trained highly accurate text detection and character recognition modules. Despite the inherent differences between the text detection and character recognition tasks, we were able to use structurally identical network architectures for both the text detector and character classifier. Then, as a direct consequence of the increased accuracy and robustness of these models, it was possible to construct a full end-to-end system using very simple and standard post-processing techniques such as non-maximal suppression (NMS) [29] and beam search [34]. In spite of the simplicity, however, our system achieved state-of-the art performance on the standard ICDAR 2003 [25] and Street View Text (SVT) [38] benchmarks. Our results thus demonstrate the viability of our alternative construction of a complete end-to-end text recognition system that does not rely extensively on hand-engineered features or hard-coded prior knowledge.

The following thesis begins with a survey of the relevant literature, continues with a description of our end-to-end text recognition system, and concludes with a detailed analysis of the proposed system. More concretely, Chapter 2 discusses background and related work on scene text recognition, unsupervised feature learning, and convolutional neural architectures. Chapter 3 provides a high-level description of the different components in the full end-to-end recognition system. As part of this discussion, I provide a careful description of the text detection module. The character recognition module and final end-to-end integration of the two modules is primarily the work of Tao Wang and thus, is not elaborated upon as extensively in this thesis. Chapter 4 presents some experimental analysis of the text detection module as well as the full end-to-end recognition system.

Finally, Chapter 5 summarizes the key results outlined in this thesis and provides some concluding remarks.

As a final note, the construction of the full end-to-end text recognition system was done in collaboration with Tao Wang and advised by Adam Coates and Professor Andrew Ng. Thus, in describing the system, I generally refer to it as "our system" and "our work" to reflect the joint nature of this work.

Chapter 2

Background and Related Work

2.1 Scene Text Recognition

Text recognition is a problem in machine learning and computer vision that dates back several decades. At the high level, the general problem of *end-to-end* text recognition consists of two primary components: text localization and word recognition. First, in text localization, the goal is to locate individual words or lines of text. Then, once we know where the regions of text are located in the image, we seek to identify the actual words and lines of text in those regions. An illustration of the end-to-end recognition task is presented in Figure 2.1.

Over the years, much time and effort have been invested in solving different components of the text-recognition problem. As a direct result, there now exist algorithms that achieve extremely high performance on specialized tasks such as digit recognition in constrained settings. For instance, the system in [5] is able to achieve near-human performance on handwritten digit recognition. Similarly, the system in [7] attains very high accuracy on the task of recognizing English characters. Despite these advances in the field of text recognition, however, the more general task of detecting and recognizing text in complex scenes still remains an open problem.

Much of the literature on scene text recognition tends to focus on a sub-component of the full end-to-end text recognition system. The three primary subsystems that have received much of this attention are text detection, character/word segmentation, and character/word recognition. Below I describe each of these subsystems individually.

Figure 2.1: Illustration of the end-to-end text recognition problem. Given an input image, we first *localize* the text. Then, we *recognize* the words in the image. This particular image is taken from the Street View Text (SVT) dataset [38].

2.1.1 Text Detection

As mentioned above, the goal of text detection or localization is to identify candidate regions of text in a given input image. Typically, the detection task corresponds to identifying a bounding box or rectangle for each word or for each line of text in the image. Many different methods have been proposed for text detection. These methods range from using simple off-the-shelf classifiers with hand-coded features [3] to much more sophisticated multi-stage pipelines incorporating many different algorithms and processing layers [31, 32]. One example of an elaborate multi-stage pipeline is the system proposed by [32], which employs extensive pre-processing stages such as binarization of the input image, followed by connected component analysis via a conditional random field (CRF) to identify lines of text. Still others in the field of text detection have developed clever hand-engineered features and transformations well-suited for the task at hand. For example, the system in [11] exploits the uniformity of stroke width in characters to develop a robust, state-of-the-art text detection system.

2.1.2 Text Segmentation and Recognition

The story is similar in the case of text segmentation and recognition. I begin by briefly outlining the problem of text segmentation and recognition. In the case of segmentation, the task is to take a single word or single line of text and produce individual characters or

individual words. In the context of an end-to-end system, this input line or word would correspond to a region of text identified by the text detection system. In turn, the character or word recognizer would identify these segmented characters and words. It follows that if we can recognize each character in the word, we can concatenate the characters together to identify the underlying word. Thus, the final result of the segmentation and recognition stages is a set of *annotated* bounding boxes, similar to those shown in Figure 2.1. The label or annotation for each bounding box is just the identified word.

As in the case of detection, a wide variety of techniques has been applied to the problem of segmentation and recognition. These techniques include various flavors of probabilistic graphical models for joint segmentation and recognition [13, 40, 41], a multi-stage hypothesis verification pipeline [30] that leverages geometric models, language models, and other forms of prior knowledge in a complete end-to-end recognition system, as well as a pictorial structure model [38] that combines a simple character classifier along with geometric constraints on character locations that also achieves end-to-end recognition. Because of the nature of the recognition problem, many of these systems incorporate varying degrees of prior knowledge in the form of geometric models or language models. Geometric typographic models, like the one used in [30], encode knowledge such as the fact that the height of certain characters is greater than that of others (e.g., the letter 'b' versus the letter 'c') or that certain characters are interchangeable because they have similar appearances (e.g., uppercase 'S' and lowercase 's'). Similarly, language models provide information about how characters are typically distributed within words. For instance, a language model might encode the fact that certain bigrams, or sequences of two characters, occur more frequently than others in the English language.

2.1.3 Lexicon-Driven Recognition

Since the ultimate goal of a text recognition system is to detect and localize text in images, one common component used to improve the performance of recognition systems is the inclusion of a *lexicon* [30, 38, 40]. The lexicon is simply a list of candidate words that may appear in the scene or image. For instance, if we are focused on reading the English text in images, it is reasonable to provide a listing of words that appear in the English language, perhaps augmented with a list of common names, places, abbreviations, and so forth. By including a lexicon, we give the model the ability to correct some of its own mistakes; for instance, the model can misread a character in a word, but still identify the correct word by

searching through a list of possible words and choosing the one most similar to the predicted word.

Naturally, the smaller the set of possible words, the better the performance of the text recognizer. At first, requiring that there be a lexicon may seem like a severe limitation of the system; in many applications, however, it is possible to obtain a small and restrictive lexicon. As argued in [38], in the case where the system is used to assist someone at a grocery store, the number of words that may be present in the scene would certainly be much smaller than the total number of words in the English language. In a case where a user is trying to recognize street names, store signs, and so forth, we can often obtain additional context based upon location data. For instance, given a user's location, the system can perform an Internet search for the streets and stores in the immediate vicinity of the user and then use these search results to construct an appropriate lexicon. Thus, while requiring that a small lexicon (consisting of 50-500 words) be provided for each image would certainly reduce the generality of the model, in many applications, this requirement is in fact a reasonable one. In constructing our system then, we focused more heavily on this simplified lexicon-constrained environment. I refer to this framework as the *lexicon-driven* or *lexicon-constrained* recognition framework. Note, however, that in this thesis, I also present a method to relax this assumption and obtain decent recognition results in the more general setting where specialized lexicons are not readily available.

2.2 Unsupervised Feature Learning

In the field of machine learning, the performance of a model is oftentimes strongly influenced by the way the data is represented. Thus, devising effective representations of the data is an important component of constructing a high-performing model. As the survey of scene text recognition in Section 2.1 demonstrates, it is evident that many of the techniques that have been successfully applied in the area of text detection and recognition have generally relied on carefully hand-engineered features [3, 11, 30] for data representation. These specialized features are essential for constructing a representation for the data that is robust to the high degree of variation present in natural images. As mentioned in Chapter 1, these variations include, but are not limited to, variations in lighting, texture, font, background, and image resolution. While hand-engineering is certainly one way of approaching this problem of data representation, in many cases, hand-engineering is also a difficult and expensive process.

Recent work in machine learning has focused on the development of algorithms that can automatically learn these underlying representations, or features, from the data itself. In the case of *unsupervised* feature-learning algorithms, these features are learned from *unlabeled* data. These feature-learning systems thus present an alternative method of devising very specialized features for use in the problems of text detection and recognition. Furthermore, these algorithms allow us to generate much larger and richer sets of features than would be possible by hand-engineering alone. In turn, these larger feature banks may be used to achieve higher performance in standard classification algorithms. Indeed, the system in [8] achieves state-of-the-art performance in character recognition using upwards of 4,000 features learned from an unsupervised learning algorithm.

Many different types of unsupervised feature-learning algorithms have been applied to various fields across machine learning, such as image classification [42], sentiment analysis [26], and text recognition [7]. Some of the different types of feature learning algorithms featured prominently in the literature include restricted Boltzmann machines (RBMs) [16, 19], sparse autoencoders [14, 33], sparse coding [15, 23, 42], and K-means [8]. One differentiating factor among these various algorithms is their computational complexity and scalability. Unfortunately, most of these algorithms tend to be very computationally expensive and cannot be scaled to work effectively with large images such as those that would be input to a text detection system. In our work, we employed the variant of the K-means algorithm described in [7, 8], which is known for its simplicity (requiring almost no hyperparameters) and speed.

Overall, the ability to automatically extract domain-specific features from the underlying data provides a viable alternative to the method of hand-engineering that has been the more traditional approach in machine learning and computer vision. As I demonstrate in this thesis, the use of these learned features in conjunction with the representational power of a convolutional neural network enables us to design a high-performing and robust model with virtually no hand-tuning. As such, this work represents a departure from some of the more conventional methods that have been applied to the problems of text detection and recognition.

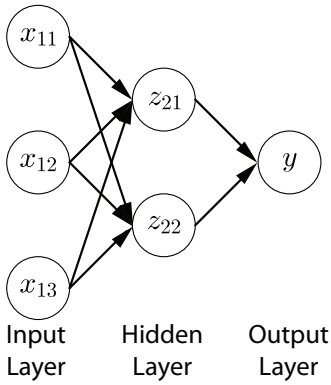


Figure 2.2: A simple feed-forward neural network.

2.3 Convolutional Neural Networks

Thus far in my exposition, I have described some of the existing methods researchers have used for detection and recognition. In many cases, these methods combine sophisticated models with cleverly designed features for the problem at hand. In this thesis, my goal is to illustrate an alternative design that does not require clever, hand-designed features or very intricate models incorporating vast amounts of prior knowledge. Unsupervised feature-learning techniques present a viable alternative to the hand-engineering of features. In our work, we then integrated these learned features into a convolutional neural network [21, 22]. This section provides a brief overview of both neural networks as well as convolutional neural network (CNN).

2.3.1 Feed-Forward Neural Networks

Before describing the convolutional neural network, I begin with a description of the basic, or feed-forward, neural network. A more thorough treatment of this material is provided in [1]. Consider a supervised learning scenario where we are given a set of labeled data $\left\{\left(x^{(i)}, y^{(i)}\right)\right\}$. Here, $x^{(i)}$ and $y^{(i)}$ denote the features and label, respectively of the i^{th} training example. At a high-level then, neural networks provide a way of representing a complex, nonlinear function $h_W(x)$ of our input variable x. The function $h_W(x)$ is parameterized by a weights matrix W that we can tune to fit our data. Figure 2.2 shows a simple neural network consisting of three input units or *neurons*, denoted $x_{11}, x_{12},$ and x_{13}, and one output unit $y = h_W(x_{11}, x_{12}, x_{13})$.

A neural network is generally organized into multiple *layers*. For example, the network in Figure 2.2 consists of three layers: the input layer, the hidden layer, and the output layer. As evidenced in the diagram, we also have a set of edges connecting neurons between adjacent layers. While Figure 2.2 shows a fully connected network where each neuron is connected to every neuron from the preceding layer, this is not a necessary condition in the construction of a neural network. The connectivity pattern of a neural network is generally referred to as the network's *architecture*.

Aside from the neurons in the input layer, each neuron x_i in the neural network is a computational unit that takes in as input the values of the neurons from the preceding layer that feed into x_i. As a concrete example, the inputs to the neuron labeled z_{21} in the sample neural network is $x_{11}, x_{12},$ and x_{13} and the input to y is z_{21} and z_{22}. Given its inputs, a neuron first computes a weighted linear combination of those inputs. More precisely, let $x_1, \ldots, x_n$ denote the inputs to a neuron z_j. Then, we first compute

$$a_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

where w_{ji} is a parameter describing the interaction between z_j and the input neuron x_i. The b_j term is a bias or intercept term associated with neuron z_j. We then apply a nonlinear activation function [18] to a_j. Some common activation functions include the sigmoid and the hyperbolic tangent functions. In particular, the *activation* or value of the neuron z_j is defined to be

$$z_j = h\left(a_j\right) = h\left(\sum_{i=1}^{n} w_{ji} x_i + b_j\right)$$

where h in this case is our nonlinear activation function. Given a set of input variables x, and weights W (one term for each edge and a bias term for each node excluding the ones in the input layer), we can compute the activation of each neuron by following the above steps. Since the activation of each neuron depends only upon the values of neurons in preceding layers, we compute the activations starting from the first hidden layer (which depend only upon the input values) and proceed layer-wise through the network. This process where information propagates through the network is called the *forward-propagation* step. At the end of the forward-propagation step, we obtain a set of outputs $y = h_W(x)$. In the case where we are performing binary classification, we can view the output y as a classification result for the input x.

The parameters W in the neural network are comprised of the weight terms for each of the edges as well as a bias term for each of the nodes, excluding the ones in the input layer. Given our labeled training set $\left\{\left(x^{(i)}, y^{(i)}\right)\right\}$, the objective is to learn the parameters W so as to minimize some objective or loss function. In the case where we are performing binary classification, a simple objective function might the classification error over the entire dataset (e.g., the mean squared difference between the predicted label $\hat{y} = h_W(x)$ and the true label y). The standard approach to learning the parameters W in order to minimize the desired objective is the *error backpropagation* algorithm [1]. Because the backpropagation algorithm is a standard approach in the literature on neural networks and is not essential to understanding the work presented in this thesis, I omit the details here.

2.3.2 Convolutional Neural Networks

With this preparation, I now describe the structure of the convolutional neural network. At the most basic level, a convolutional neural network is just a multilayer, hierarchical neural network. There are three principal factors that distinguish the CNN from the simple feed-forward neural networks described in Section 2.3.1: local receptive fields, weight sharing, and spatial pooling or subsampling layers. I consider each of these three properties individually in the context of a visual recognition problem. In particular, suppose that the input to the CNN consists of a single 32-by-32 image patch. For instance, this input can be a 32-by-32 grid of pixel intensity values.

In the simple neural networks described in Section 2.3.1, each neuron was fully connected to each of the neurons in the subsequent layer. More concretely, each neuron in the hidden layer computed a function that depended on the values of every node in the input layer. In visual recognition, however, it is often advantageous to exploit local substructure within the image. For example, pixels that are close together in the image (e.g., adjacent pixels) tend to be strongly correlated while pixels that are far apart in the image tend to be weakly correlated or uncorrelated. Not surprisingly then, many standard feature representations used in computer vision problems are based upon local features within the image [9, 24]. In the CNN architecture, we capture this local substructure within the image by constraining each neuron to depend only on a spatially local subset of the variables in the previous layer. For example, if the input to the CNN is a 32-by-32 image patch, a neuron in the first hidden layer might only depend on an 8-by-8 subwindow within the overall 32-by-32 window. The set of nodes in the input layer that affect the activation of a neuron is referred to as the

neuron's *receptive field*. Intuitively, this is the part of the image that the neuron "sees." Thus, in a CNN, individual neurons generally have a *local* receptive field rather than a *global* receptive field. In terms of network architecture, this translates to a sparser set of edges since adjacent layers are not always fully connected.

The second feature that distinguishes CNNs from simple neural networks is the fact that the edge weights in the network are shared across different neurons in the hidden layers. Recall that each neuron in the network first computes a weighted linear combination of its inputs. We can view this process as evaluating a linear *filter* over the input values. In this context, sharing the weights across multiple neurons in a hidden layer translates to evaluating the *same* filter over multiple subwindows of the input image. In this regard, we can view the CNN as effectively learning a set of filters $\mathcal{F} = \{F_i \mid i = 1, \ldots, n\}$, each of which is applied to all of the subwindows within the input image. Using the same set of filters over the entire image forces the network to learn a general encoding or representation of the underlying data. Constraining the weights to be equal across different neurons also has a regularizing effect on the CNN; in turn, this allows the network to generalize better in many visual recognition settings. Another benefit of weight sharing is the fact that it substantially reduces the number of free parameters in the CNN, making it markedly easier and more efficient to train. As a final note, evaluating a filter F over each window in the input image I amounts to performing a convolution of the image I with the filter F (we *convolve* the image I with the filter F). Thus, in the *convolutional step* of the CNN, we take the input image and convolve it with each filter in $\mathcal{F}$ to obtain the convolutional response map.

The final distinguishing component in a CNN is the presence of subsampling or pooling layers. The goal here is twofold: reduce the dimensionality of the convolutional responses and confer a small degree of translational invariance into the model. The standard approach is through spatial pooling [2]. In spatial pooling, the convolutional response map is first divided into a set of $m \times n$ blocks (generally disjoint). We then evaluate a *pooling function* over the responses in each block. This process yields a smaller response map with dimension $m \times n$ (one response for each block). In the case of max pooling, the response for each block is taken to be the maximum value over the block responses, and in the case of average pooling, the response is taken to be the average value of the block responses. Figure 2.3 shows an example of average pooling. In this case, the convolutional response map is a 4-by-4 grid and we average pool over four 2-by-2 blocks (the shaded regions in Figure 2.3)

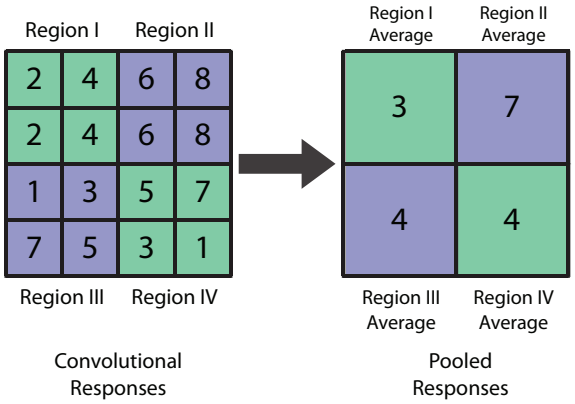


Figure 2.3: Average pooling in a convolution neural network. In this case, we average pool over a 2-by-2 grid over the 4-by-4 convolutional responses. The shaded regions denote the blocks over which we are average pooling. We compute the average over the values in each block, yielding the 2-by-2 pooled response map shown on the right.

arranged in a 2-by-2 grid. The pooled response is taken to be the average of the values in the block. After applying this average pooling procedure, we arrive at a final 2-by-2 pooled response map. Compared to the original 4-by-4 convolutional response map, this represents a significant reduction in dimensionality of the response map.

In a typical CNN, we have multiple layers, alternating between convolution and pooling. For example, we can stack another convolution-pooling layer on top of the outputs of the first convolution-pooling layer. In this case, we simply treat the outputs of the first set of convolution-pooling layers as the input to the second set of layers. In this way, we can construct a multilayered or *deep* architecture. Intuitively, the low-level convolutional filters, such as those in the first convolutional layer, can be thought of as providing a low-level encoding of the input data. In the case of image data, these low-level filters may consist of simple edge filters. As we move to higher layers in the neural network, the model begins to learn more and more complicated structures. By using multiple layers and large numbers of filters, the CNN architecture can thus provide vast amounts of representational power. To train a CNN, we can use the standard technique of error backpropagation used to train neural networks [1].

Convolutional neural networks have enjoyed a series of successes in many problems related to text classification such as handwriting recognition [21], visual object recognition [4] and character recognition [35]. Coupled with the rapid advancements in distributed and

GPU (graphics processing units) computation, it is now possible to train much larger and more powerful CNNs that achieve state-of-the-art performance on standard benchmarks [4, 27]. Thus, by leveraging the representational capacity contained within these networks in conjunction with the robustness of features derived from unsupervised algorithms, we are able to construct simple, but powerful and robust, systems for both text detection and recognition. Using these robust and highly-accurate components renders it possible to obtain full end-to-end results using only the simplest of post-processing techniques.

Chapter 3

Methodology

In this chapter, I first describe the learning architecture used to train our text detection and recognition modules. These were the essential building blocks of our full end-to-end system. I then describe how we integrated these two individual components into a complete end-to-end system.

To provide context, I begin with a high-level description of our text recognition system. Our system consisted of two principal modules: the text detector and the character classifier. To construct the text detector, we first trained a binary classifier that decided whether a single 32-by-32 image patch contained a well-centered character or not. To compute the detector responses over a full image then, we took this binary classifier and evaluated its response over every 32-by-32 window in the image. By using this sliding-window approach where we essentially slid the fixed-size detection window across the full image, we identified candidate lines and groups of text. Next, we trained a character classifier that decided which of 62 possible characters (26 uppercase letters, 26 lowercase letters, 10 digits) was present in a 32-by-32 input patch. Note that by construction, the character classifier always assumed that the input image patch contained a single character; in particular, we did not introduce a non-character class. Then, by sliding the character classifier across the regions of text identified by the text detector, we identified the characters in each region or line. Finally, using a beam search algorithm, we combined the outputs from the text detection and character recognition modules to obtain the final end-to-end results: a set of annotated bounding boxes for the words in the image. Figure 3.1 provide a simple illustration of this recognition pipeline.

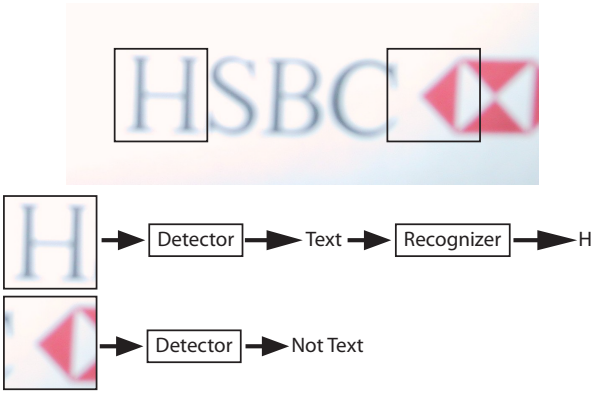


Figure 3.1: Operation of detection and recognition modules. The detector decides whether a single 32-by-32 image patch contains text or not. The recognizer identifies the character present in a text-containing input patch.

3.1 Detection and Recognition Modules

In this section, I elaborate on the construction of the text detection and recognition modules, and conclude with a few remarks on the datasets used to train the detector and recognizer.

At the most fundamental level, both the text detector and character recognizer consisted of a two-layer convolutional neural network. In both cases, we first applied an unsupervised feature-learning algorithm to learn a set of low-level features to represent the data. Armed with this low-level representation, we trained the neural network discriminatively by back-propagation of the L_2-SVM classification error.

3.1.1 Unsupervised Pretraining

First, we used an unsupervised learning algorithm similar to [7, 8] to extract features for use in the first convolutional layer of the CNN. Our specific goal was to develop a robust low-level representation of the input data (a single 32-by-32 image patch). We followed the basic procedure outlined below.

1. We began by extracting a set of m small image patches from the training set. As in [7], we used 8-by-8 grayscale patches. Each such image patch can be regarded as a vector of $8 \times 8 = 64$ pixel intensity values. Thus, we obtained a set of m vectors of pixels $\tilde{x}^{(i)} \in \mathbb{R}^{64}, i = 1, \ldots, m$.

2. We then normalized each vector $\tilde{x}^{(i)}$ for brightness and contrast by subtracting the mean and dividing by the standard deviation. This process yielded a set of normalized vectors $\hat{x}^{(i)}$:

$$\hat{x}^{(i)} = \frac{\tilde{x}^{(i)} - \mu_{\tilde{x}_i}}{\sigma_{\tilde{x}_i}}$$

where $\mu_{\tilde{x}_i}$ and $\sigma_{\tilde{x}_i}$ denote the mean and standard deviations of $\tilde{x}_i$, respectively.

3. Next, we centered and whitened the normalized vectors $\hat{x}^{(i)}$ using ZCA whitening [17] to obtain a new set of vectors $x^{(i)}$. More specifically, given normalized data $\hat{x}^{(1)}, \ldots, \hat{x}^{(m)}$, we computed the mean M and covariance Σ along each component. We then took the eigendecomposition of $\Sigma = VDV^T$, where V is the orthogonal matrix consisting of the eigenvectors of Σ and D is the diagonal matrix consisting of the eigenvalues of Σ. Mathematically, the centering and whitening transforms are given by

$$x^{(i)} = VD^{-1/2}V^T\left(\hat{x}^{(i)} - M\right) = P\left(\hat{x}^{(i)} - M\right)$$

where $P = VD^{-1/2}V^T$. This step essentially had the effect of decorrelating adjacent pixels in the image.

4. We then applied an unsupervised learning algorithm to the preprocessed patches $x^{(i)}$ to construct a mapping from the input patches to feature vectors $z^{(i)} = f(x^{(i)})$. For our unsupervised feature-learning algorithm, we used the variant of the K-means algorithm described in [7] where we learn a dictionary $D \in \mathbb{R}^{64\times d}$ of normalized basis vectors. In this case, the mapping f is simply the projection of $x^{(i)}$ onto the subspace spanned by the columns of D: $z^{(i)} = f\left(x^{(i)}\right) = D^T x^{(i)}$. Note that d represents the number of basis vectors or filters in the dictionary D (recall that our input patches $x^{(i)} \in \mathbb{R}^{64}$ so the dictionary D has dimension $64 \times d$). In our particular setup, we learned $d = 96$ filters for use in the text detection module and $d = 115$ filters for use in the character recognition module. To compute D, we solved the following optimization problem:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} \left\| Ds^{(i)} - x^{(i)} \right\|_2^2 \\ \text{subject to} & \left\| s^{(i)} \right\|_1 = \left\| s^{(i)} \right\|_\infty, \ i = 1, \ldots, m \\ & \left\| D^{(j)} \right\|_2 = 1, \ j = 1, \ldots, d. \end{array}$$

As in [7], the minimization was performed over the variables D and $s^{(i)}$. In this

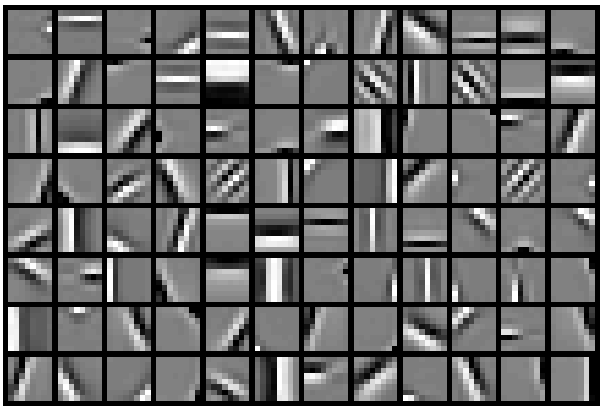


Figure 3.2: A visualization of the dictionary elements learned from whitened grayscale 8-by-8 image patches.

optimization problem, the $x^{(i)}$ represent the preprocessed input examples, and the $s^{(i)}$ represent the scaled canonical basis vectors (the constraint $\left\|s^{(i)}\right\|_1 = \left\|s^{(i)}\right\|_\infty$ forces each $s^{(i)}$ to have at most one non-zero component). The second constraint in the optimization problem is simply the normalization condition on the columns $D^{(j)}$ of D where $j = 1, \ldots, d$. Thus, we effectively approximate each example $x^{(i)}$ using a scalar multiple of one of the basis vectors (one of the columns of D). As in the standard K-means algorithm [1], we solved the optimization problem by alternately minimizing the objective over D given the encodings $s^{(i)}$ and minimizing the objective over $s^{(i)}$ given the dictionary D.

Figure 3.2 presents a visualization of the set of basis vectors learned using the above method. Visually, these features are not significantly different from the typical set of edge filters learned by other unsupervised algorithms [16, 17].

Using D, we proceeded to define the feature representation for a single 8-by-8 image patch $\tilde{x}$. First, we applied the same normalization and whitening transforms from Step 3 above to obtain x:

$$x = P\left(\frac{\tilde{x} - \mu_{\tilde{x}}}{\sigma_{\tilde{x}}} - M\right).$$

Next, we projected x onto the vector space spanned by the columns of D to obtain a new representation $z' \in \mathbb{R}^d$. More precisely, we computed $z' = D^T x$. As will be described in Section 3.1.2, the filters in the dictionary D were used as the filters in the first convolutional layer of the CNN. Thus, as was standard in the literature on neural networks, we applied a

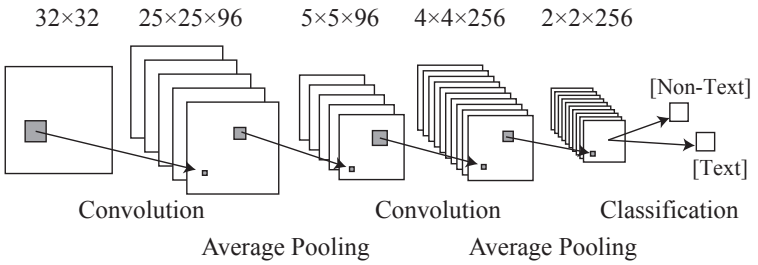


Figure 3.3: Convolutional neural network used for detection. An analogous network is used for recognition.

nonlinear activation function to the responses z'. Here, we adopted the activation function used in [7]:

$$z = h(z') = \max\left\{0, \left|z'\right| - \alpha\right\} = \max\left\{0, \left|D^T x\right| - \alpha\right\} \in \mathbb{R}^d$$

where α is a hyperparameter. For all experiments described in this thesis, we took $\alpha = 0.5$.

3.1.2 Convolutional Neural Network Architecture

For both the text detection and character recognition modules, we used a two-layer, convolutional neural network similar to [21, 35]. As discussed in Section 2.3.2, each layer of the CNN consisted of a convolutional layer, followed by a spatial pooling layer. The outputs of the second spatial pooling layer fed into a fully connected classification layer that performed either binary classification in the case of the text detection module or 62-way classification in the case of the character recognition module.

The CNN we used for text detection is shown in Figure 3.3; a larger, but structurally identical network was used for character recognition. As mentioned at the beginning of Chapter 3, the input to both the detector and the recognizer consisted of a single 32-by-32 grayscale image patch. In the first convolutional layer, we used the encoding described in Section 3.1.1. More precisely, for every 8-by-8 subwindow $\tilde{x}$ in the input image, we first normalized and whitened it to obtain an 8-by-8 patch x. Using x, we then computed the activated response $z = h\left(D^T x\right) = \max\left\{0, \left|D^T x\right| - \alpha\right\}$. By evaluating the activation over each 8-by-8 subwindow, we thus obtained the final 25-by-25-by-d representation of the input image. In the case of text detection, we used a dictionary of $d = 96$ basis vectors and in the case of character recognition, we used a dictionary of $d = 115$ basis vectors in the first

layer. Note that to efficiently compute this representation, we convolved the input image with each of our basis elements or filters.

We next introduced a spatial pooling layer [2] on top of the convolutional layer. As done in [7], we used average pooling which both reduced the dimensionality of the response map as well as conferred a degree of translational invariance to the model. More precisely, in the first average-pooling layer, we averaged over the activations of 25 blocks arranged in a 5-by-5 grid over the 25-by-25-by-d response map from the first convolutional layer. As expected, this process yielded a 5-by-5-by-d response map.

We then stacked an additional layer of convolution and average pooling on top of the outputs from this first layer. In this second convolutional layer, we used a set of 2-by-2-by-d filters. As was the case in the first layer, we convolved each 2-by-2-by-d filter with the 5-by-5-by-d response from the first layer. After performing these convolutions, we arrived at a 4-by-4-by-d_2 response map where d_2 denotes the number of filters used in the second convolutional layer. In the case of detection, we used $d_2 = 256$ filters, and in the case of recognition, we used $d_2 = 720$ filters. We used a larger number of filters in the case of recognition because the recognizer was performing 62-way classification rather than binary classification as in the case of the detector. Thus, to achieve similar performance, the character recognizer required greater expressive power, and correspondingly, a larger number of filters. As was the case in the first convolutional layer, we again applied the activation function h to the convolutional responses in the second layer. Finally, we averaged over the responses over four blocks arranged in a 2-by-2 grid over the image, yielding a final 2-by-2-by-d_2 representation as the output of the second layer. This 2-by-2-by-d_2 representation served as the input to a one-versus-all, multiclass support vector machine (SVM). In the case of detection, the SVM performed binary classification and in the case of recognition, 62-way classification.

To discriminatively train the CNN, we minimized the L_2-SVM classification error. Given a training set $\left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{1 \le i \le n}$ with inputs $x^{(i)} \in \mathbf{R}^n$ and output labels $y^{(i)} \in \{0, 1\}$, the standard L_2-SVM optimization problem is

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \xi_i^2 \\ \text{subject to} & y^{(i)}\left(w^T x^{(i)} + b\right) \ge 1 - \xi_i \\ & \xi_i \ge 0 \end{array}$$

where we minimize over the weights $w \in \mathbf{R}^n$ and the bias $b \in \mathbf{R}$. Here, $C > 0$ is the

standard SVM regularization parameter that controls the tradeoff between classification error minimization and margin maximization [1]. Equivalently, we may formulate this as an unconstrained optimization problem with a hinge loss objective:

$$\text{minimize} \quad \tfrac{1}{2}\|w\|_2^2 \; + C\sum_{i=1}^{n}\left[\max\left\{0, 1 - y^{(i)}\left(w^T x^{(i)} + b\right)\right\}\right]^2.$$

We then applied error backpropagation [1] to fine-tune, or update, the weights of the CNN so as to minimize this objective. Note that we only fine-tuned the parameters in the classification layer and the second set of convolutional and average-pooling layers. The first layer of filters, namely the pretrained ones, were kept fixed throughout the process. This was motivated by the fact that K-means features tended to work well as a low-level encoding of the data [7, 8] as well as the fact that it was computationally less intensive to train only a subset of the layers in the CNN.

It is important to note that due to the relatively large networks we were using, we performed the fine-tuning over multiple graphics processing units (GPUs) in a distributed setup. The use of GPUs and distributed computing is gradually becoming commonplace in large-scale machine learning [4, 6] and is often essential to constructing and training larger, more powerful, representational models.

3.1.3 Datasets

Before proceeding to a discussion of the end-to-end integration process, I briefly describe the datasets used to train both the text detector and character recognizer. In the case of the text detector, we defined a "positive" example to be a single 32-by-32 image patch containing a well-centered character (some examples are shown in Figure 3.4). Examples where the character was off-center or just contained parts of a character were taken to be negatives. This distinction provided a relatively unambiguous definition of what constituted a positive example. By training the detector using this definition of "positive" and "negative," the detector was thus able to select for and identify windows containing well-centered characters. This detail played an important role later in the space estimation process described in Section 3.3.1.

In much of the recent literature in text recognition, researchers have augmented their datasets with synthetic training examples in order to improve model performance [7, 30, 38]. Likewise, in our system, we generated high quality synthetic training images using a total

(a) Examples from ICDAR 2003 (b) Synthetically Generated Examples

Figure 3.4: Comparison of training images from the ICDAR 2003 dataset and synthetically generated examples.

of 665 fonts for training both the text detector and character recognizer[1]. To construct the synthetic examples, we first sampled the grayscale level of the character as well as the background from Gaussian distributions with the same mean and standard deviation as the examples in the ICDAR 2003 training images [25]. We then applied small amounts of Gaussian blurring and projective transformations to random portions of the image. Finally, we blended the synthesized images with natural backgrounds to simulate background clutter. A comparison of our synthetically generated examples with actual training examples from the ICDAR 2003 training images is shown in Figure 3.4.

Last, to train the detector and recognizer, we compiled a dataset consisting of examples drawn from the ICDAR 2003 training images [25], the English subset of the Chars74k dataset [10], and a very large number of synthetically generated examples. In training the detector and recognizer, we used over 100,000 examples in total.

3.2 Text Line Detection

In this section, I describe the process of generating a set of candidate lines of text given a high-resolution input image. I begin with a high-level overview of our process. First, we took the text detector and ran sliding-window detection across the image. This yielded a response map that encoded the likelihood of text occurring at every 32-by-32 window in the image. We repeated this process at multiple scales of the image to obtain a set of multiscale

[1]Note that the synthetic data generator described here was largely the work of Tao Wang. I present the key details to provide useful context for the subsequent sections of this thesis.

response maps. From these response maps, we inferred and scored a set of bounding boxes for candidate lines of text in the original image. We then applied non-maximal suppression (NMS) [29] to this set of candidate bounding boxes to obtain our final set of bounding boxes.

3.2.1 Multiscale Sliding Window

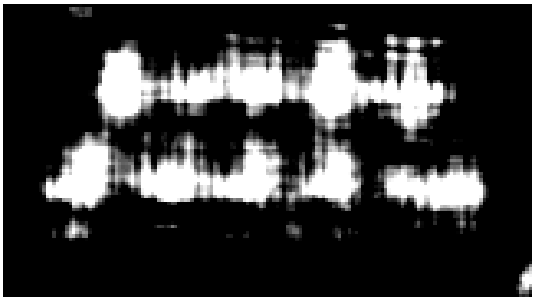
Given a high-resolution input image (resolutions typically ranged from 640-by-480 pixels to 1600-by-1200 pixels), we began by evaluating the detector response for every 32-by-32 window in the image. This kind of *sliding-window* detection has proven to be effective in a wide variety of related vision tasks, such as pedestrian detection [9] or face recognition [37]. Computing the detector response for every 32-by-32 window in the image amounted effectively to a forward propagation step in the CNN used for detection. Thus, the detector effectively assigned a score or confidence to each window in the image. More concretely, a large positive score from the detector translated to a higher degree of confidence that the window contained text and a large negative value from the detector translated to a low degree of confidence that the window contained text. Note that according to our specifications outlined in Section 3.1.3, for a window to "contain text," the window must contain a well-centered character.

Because the detection window was fixed at 32-by-32 pixels, detecting text at different scales required us to apply this sliding-window process to multiple scales of the image. Specifically, we considered both upsampling and downsampling the image in order to capture text at a wide range of scales. In total, we used 13 different scales, ranging from as high as 150% the size of the original image to as low as 10% the size of the original image. For example, a 32-by-32 detection window oven an input image at 50% scale would translate to a 64-by-64 detection window over the original image. By performing this sliding-window detection at each of these scales, we obtained a multiscale response map. Figure 3.5 illustrates an example of this multiscale sliding-window response process.

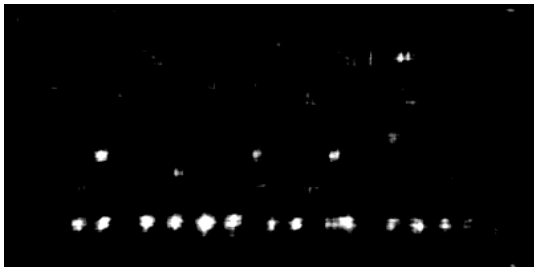
3.2.2 Text Line Formation

With the multiscale response map for an input image, we were able to compute bounding boxes for each line of text in the image. To do so, we first made the simplifying assumption that text tends to lie along horizontal lines in natural images. This was generally a reasonable assumption since most of the text in natural images do indeed fall along horizontal

(a) Original image. Numbers denote scale of detection windows.

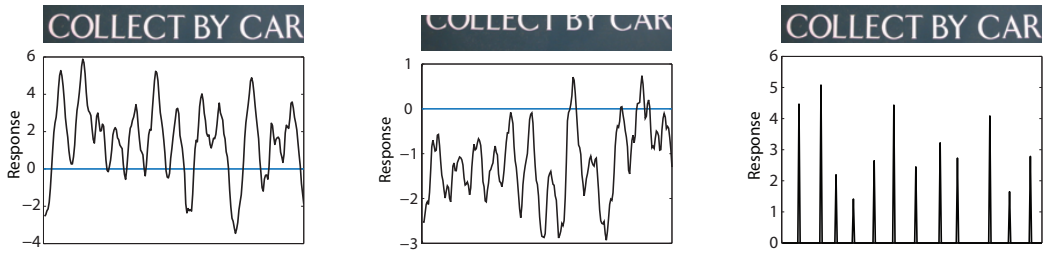


(b) Response map at 40% scale.



(c) Response map at 90% scale.

Figure 3.5: Detector response maps at different scales. The top image shows the *effective* size of the detector window on the original image (a 32-by-32 detection window applied to an image at 50% scale translates to a 64-by-64 window applied to the image at normal scale). The bottom two images show detector response maps at two different scales. In the response maps, the brightness of a pixel measures the detector's confidence for a window centered at that position. Thus, a black pixel indicates a negative prediction and a white pixel indicates a positive prediction. In particular, note that the response map at 40% scale appears to capture the words `SLUSH PUPPIE` while the response map at 90% scale appears to capture the words `AS COOL AS IT GETS`. Finally, note that because the CNN is invariant to small translations, we tend to see clusters of positive responses around the locations of each character.



(a) Detector responses across a centered line of text.

(b) Detector responses across an uncentered line of text.

(c) Non-maximally suppressed responses across a single line of text.

Figure 3.6: Detector and NMS responses across lines in the image.

or slightly slanted lines. The detector responses across a single line in the image (e.g., a 32-by-w window where w is the width of the image) corresponded to a single row in the response map for the image. Figures 3.6a and 3.6b show the detector responses across two different lines in the input image. From these examples, we noted two things. First, the detector responses across a line of *centered* text tended to be positive, while those across a line of *uncentered* text tended to be negative. This particular behavior from the detector was expected because we had defined a positive example to be a window containing a well-centered character. Thus, windows where the character was only partially visible, such as those in the line of uncentered text, would not have produced a positive signal. Second, we noted that the peaks of the detector responses generally corresponded to the positions of the characters in the line of text. Again, this particular behavior was not surprising in light of our particular definition of what constituted a positive example.

Because a line of text would generally contain multiple characters and characters generally appeared as peaks in the detector response profile across a line, we applied non-maximal suppression (NMS) [29] to the response profile to identify the peaks. Then, given the number of peaks that occurred in a line along with the magnitudes of the peak responses, we could decide whether the line contained text or not. More precisely, let $R(x)$ denote the detector response at every position x along a single line. From $R(x)$, we constructed the non-maximally suppressed response $R'(x)$ by computing

$$R'(x) = \begin{cases} R(x) & R(x) \geq R(y), \forall y : |x - y| < \delta \\ 0 & \text{otherwise} \end{cases}$$

where δ is the width of the interval over which we performed NMS. We chose δ by cross-validation. Intuitively, NMS filtered out all responses that were not *maximal* in their immediate neighborhood. An example of NMS applied to a centered line of text is shown in Figure 3.6c.

Using the NMS responses across a single line, we computed the average peak response $\bar{R}_{\text{peak}}$ and compared it to a fixed threshold τ. If $\bar{R}_{\text{peak}}$ exceeded the threshold τ, then the line was taken to be a positive (contains text), and if $\bar{R}_{\text{peak}}$ was below the threshold, then we disregarded the line (does not contain text). If the line contained text, then we took the positions of the left and rightmost peaks in the NMS response map to be the extents of the bounding box for the text in that line. As usual, we selected τ by cross-validation. There were two potential problems with this method of bounding box estimation. First, there were cases where two or more groups of words resided on the same line, but were separated by a large gap or non-text region. One example of this is shown in Figure 3.7. In this case, the words `Ipswich` and `London` roughly reside on the same line, as do `Gt Yarmouth` and `Chelmsford`. However, it does not make sense to have one large bounding box spanning the width of the image; there are clearly two regions of text in this line.

Another problem was the fact that the response maps could contain many false positives, and in turn, produce spurious peaks in the NMS response across a line. This caused non-text regions to be misidentified as text regions. What distinguished these spurious detections from those corresponding to a true region of text was the variance of the spacing between peaks in the NMS profile. In a typical line of text, the spacing between characters tends to be consistent; correspondingly, the spacing between peaks would tend to be roughly uniform. Not surprisingly, the spacing between spurious peaks did not exhibit this uniformity. Thus, we exploited this consistency to both distinguish between groups of words that reside along the same line as well as filter out some of the spurious responses. In particular, we computed the distance between adjacent peaks from the NMS profile across the line. We then took the median m of these peak separation distances. If the separation distance between two adjacent peaks exceeded some scalar multiple of the median distance m, then we split the line at that peak. If either of the two resultant segments contained just a single peak after the split, then we discarded that segment altogether. This was because it was far more likely for a single, isolated peak to be due to a spurious detection response than to a single, isolated character in the image.

I illustrate this line-splitting process with a more concrete example. Consider again the

Figure 3.7: Estimated bounding boxes from text detector after all post-processing steps. There are two things to notice. First, the bounding boxes capture groups of words such as `Leisure World` or `Town Centre`. Word-level segmentation is done during end-to-end integration. Secondly, words that appear on the same "line" such as `Ipswich` and `London` are split into two boxes because of the large gap between the last character in `Ipswich` and the first character in `London`.

image in Figure 3.7. In particular, focus on the line containing `Ipswich` and `London`. The NMS profile in this line would contain peaks for the characters in `Ipswich` as well as for `London`. In the region between the two words, however, there are no significant activations. Thus, the median distance between peaks would be comparable to the distance between adjacent characters in the words `Ipswich` and `London`. Thus, when we consider the peaks across the line, we would observe a large gap between the peak corresponding to the last character in `Ipswich` and the peak corresponding to the first character in `London`. As a result, we split the line into two segments, one ending on the last character in `Ipswich` and the other starting from the first character in `London` (the location of the next peak in the line). Thus, we see that this heuristic has the desired effect of partitioning the line into groups of words. In cases where there are spurious responses, the peaks would tend to stand isolated in the NMS profile across a line and thus, be filtered out as part of this process.

Next, given a single or a set of bounding boxes for a line, we scored each bounding box by averaging the magnitudes of the peaks contained within the bounding box. As noted earlier, strong peak responses were indicative of higher confidence that a character was present at the underlying position. Thus, higher scores generally corresponded to better-fitting boxes.

We applied the above line-finding and splitting algorithm to each of the response maps to estimate bounding boxes at each scale of the image. More concretely, we applied another iteration of non-maximal suppression to the set of bounding boxes over all the different scales in order to obtain the final set of bounding boxes. In this particular case, given two bounding boxes B_1 and B_2, possibly from different scales, and corresponding scores $s_1 < s_2$, we suppressed the box with lower score (B_2 in this example) if

$$\frac{|B_1 \cap B_2|}{\min\{|B_1|, |B_2|\}} > \frac{1}{2}$$

where $|B_1|$, $|B_2|$ denotes the areas of B_1 and B_2, respectively, and $|B_1 \cap B_2|$ denotes the area of the intersection of B_1 and B_2. In words, we suppressed the bounding box with the smaller score if the bounding box significantly overlapped with another bounding box with higher score. After performing NMS, we arrived at a final set of estimated bounding boxes for the lines of text in the image. Figure 3.7 shows a sample image together with a set of bounding boxes for the detected lines and groups of words.

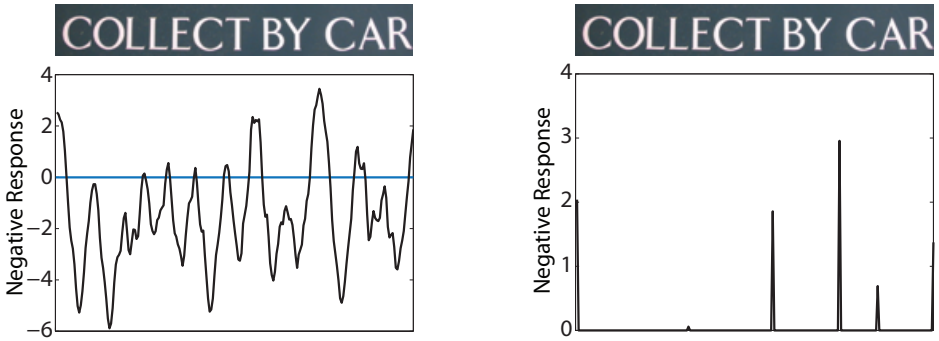
3.3 End-to-End Integration

In the preceding section, I described a method for identifying candidate lines and groups of text from an image using a multiscale, sliding-window approach. In this section, I complete the discussion by presenting our method of obtaining final end-to-end text recognition results from these candidate bounding boxes. At a high-level, this process consisted of three steps. First, given a bounding box, we *estimated* where the word boundaries were within the bounding box. Next, using the character recognizer, we identified the characters that occur within the bounding box. Finally, using both the estimated locations of the word boundaries (spaces) as well as the identities of the characters along this line, we used a beam search to jointly determine the most likely segmentation of the line into words as well as the identity of the individual words within each segment. This overall process resulted in a set of annotated bounding boxes for the image. Figure 2.1 shows a sample image along with the set of annotated bounding boxes returned by our system.

3.3.1 Space Estimation

The first step towards obtaining an end-to-end result was to identify the word boundaries in a line of text. To do so, we leveraged the same technique used to identify candidate lines of text described in Section 3.2.2 where we examined the detector response profile across a single line of text. Furthermore, as was described in Section 3.1.3, the text detector was trained to select for images of well-centered characters. In particular, images where the characters were only partially visible or off-centered were taken to be negative examples. Thus, a window centered on a space between characters should produce a negative response. Noting this, we were able to estimate the locations of the spaces by considering the *inverse* or negative detector response across a line of text. Just as we selected for the positive peak responses in order to locate characters in a line, we selected for the negative peak responses across a line to identify spaces. More precisely, we applied NMS to the negative responses across a line.

Unfortunately, it was difficult to distinguish between a space between two individual characters and a space between two individual words. In natural images, there tends to be high variability in the types of fonts and styles that can be present, and so, correspondingly, there is a high degree of variation in the amount of spacing that exist between words and between characters. In general, however, the amount of spacing between two words tends to be larger than the amount of spacing between two characters. Thus, to reduce the number of false positives (mistaking a space between two characters for a space between two words), we further imposed a threshold on the peak responses. In this case, from the negative NMS response profile $\tilde{R}'(x)$, we computed $R'(x) = \max\left\{\tilde{R}'(x) - \tau, 0\right\}$ where τ is a threshold parameter selected through cross-validation to offer a good tradeoff between number of false positives (too low threshold) and number of false negatives (too high threshold). After applying NMS, the non-zero entries in the thresholded NMS response $R'(x)$ corresponded to the predicted word boundaries in the line. The magnitudes of these peaks were used as confidences or scores for the prediction. In other words, larger peaks at a position x in the response profile indicated a higher likelihood that there was a space at position x in the line. Figure 3.8 illustrates an example of this space-estimation process.



(a) Negative detector responses across a line of text.

(b) Thresholded negative NMS response.

Figure 3.8: Negative responses across a line of text. Because it is difficult to distinguish between a space between two characters and a space between two words, this method tends to produce false positives. For instance, in the above image, we have a small peak between the `C` and the `A` in `CAR`.

3.3.2 Cropped Word Recognition

The next two sections conclude the discussion of the full end-to-end system by describing how we integrated the text detection and space estimation procedures with the character recognition module. Note that most of the work described in this section and the following section was done by Tao Wang; I include it here to provide necessary context.

I begin by describing a more constrained setting where we were provided a well-cropped bounding box containing a *single* word. Furthermore, we assumed that we were provided a lexicon $\mathcal{L}$ containing the words that could appear in the bounding box. The goal in this *word-recognition* task was to identify the word that appeared in the bounding box. Our approach in this problem was very similar to the approach we took in detection. In particular, we used sliding window classification, in which we slid the character classifier across the confines of the bounding box. Because we were provided a word-level bounding box in this scenario, we only needed to slide the character classifier horizontally across the bounding box. This was in contrast to the case of detection where we needed to slide the binary classifier both vertically and horizontally across the full image. At each horizontal position in the bounding box, the character classifier produced a 62-way classification response (sample shown in Figure 3.9). As usual, a positive prediction (the window contained a character of that class) was encoded by a positive response, and a negative prediction was encoded by a

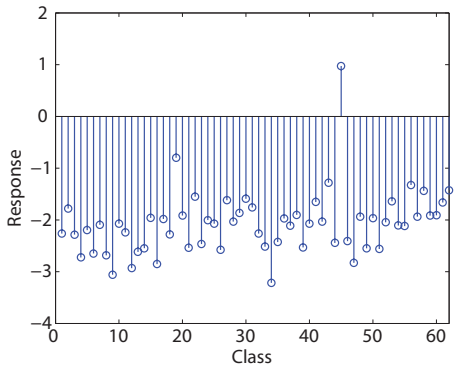


Figure 3.9: 62-way character classifier response for a single 32-by-32 image patch.

negative response.

Evaluating the character classifier response at every point in the bounding box yielded a 62-by-N scoring matrix M. In this expression, N denotes the number of sliding windows contained within the bounding box. Intuitively speaking, a larger value of $M(i, j)$ indicates a higher likelihood that the character with class i appears at the position of the j^{th} window. At each position j, we defined the confidence margin c_j to be the difference between the largest and second largest values in $M^{(j)}$ where $M^{(j)}$ denotes the j^{th} column of M. For example, a large confidence margin c_j indicates that the classifier is fairly certain of the class of the character in the j^{th} window (e.g., there is only one dominant peak), while a small confidence margin indicates more uncertainty (e.g., there are at least two peaks with similar magnitude, so it is unclear to which class the character belongs). In this way, we computed the confidence margin for each window in the bounding box to obtain a response profile $\boldsymbol{c} = (c_1, c_2, \ldots, c_N)$. As in the case of detection, the peaks in $\boldsymbol{c}$ corresponded to windows where the character was well-centered and thus, produced the largest confidence margin. We then selected for these peaks using NMS. More concretely, from the confidence margin $\boldsymbol{c}$, we computed

$$c'_j = \begin{cases} c_j & c_j \geq c_i,\ \forall i :\ |j - i| \leq \delta \\ 0 & \text{otherwise} \end{cases}.$$

Note that we did not use the peaks from the detection profile to estimate the locations of each characters since the window that maximally activated the detector did not necessarily

coincide with the window that had the greatest confidence margin with respect to the character recognizer. Since the goal was to identify the characters in the word, relying on the classifier responses was the more appropriate choice.

Using the non-maximally suppressed confidence margins $\boldsymbol{c}'$, we were able to estimate the location of each character in the bounding box. Then, for each position j, if $c'_j = 0$, we set the entries in $M^{(j)}$ to $-\infty$. In other words, if there was no character at position j in the bounding box, we effectively eliminated that column from the matrix M. We did this by assigning a score of $-\infty$ to each class. With this preparation in hand and given a lexicon $\mathcal{L}$, we defined the alignment score S_M^w of a word $w \in \mathcal{L}$ with respect to the modified score matrix M (the original score matrix with the non-character columns set to $-\infty$) to be

$$S_M^w = \max_{\ell^w \in L^w} \left(\sum_{k=1}^{|w|} M\left(w_k, \ell_k^w\right) \right).$$

In the above, ℓ^w is a $|w|$-dimensional alignment vector of the characters in w with the columns in M. As a more concrete example, suppose $w = \texttt{CAT}$. Then $\ell_2^w = 6$ means that the 2^{nd} character in w (‘A’) is aligned with the 6^{th} column in M, or equivalently, the 6^{th} sliding window within the provided word-level bounding box. Furthermore, since ℓ^w is an alignment vector, it must satisfy the ordering constraints: $\ell_k^w < \ell_{k+1}^w$ for all $k = 1, \ldots, |w|-1$. The term $M\left(w_k, \ell_k^w\right)$ is precisely the classifier response for class w_k at position ℓ_k^w in the word-level bounding box; this reflects how likely the character at position ℓ_k^w is actually w_k. To compute the final score between a word and the scoring matrix, we took the maximum score over the set L^w of possible alignments of the characters in w with the columns in M. For a given word w and score matrix M, computing S_M^w was done using an efficient Viterbi-style dynamic programming algorithm similar to the one presented in [36].

To improve the scoring metric defined above, we also introduced a few terms to encourage geometric consistency of the alignment. For instance, we noted that while the spacing between characters was not consistent across different images, the spacing tended to be consistent within a single word. We thus introduced a penalty term that was proportional to variations in the character spacings within a word. Introducing these additional consistency factors, we arrived at the augmented recognition score $\hat{S}_M^w$. Then, for all words w in the lexicon $\mathcal{L}$, we computed the augmented alignment score $\hat{S}_M^w$ and labeled the bounding box

with the highest scoring word $w^\star$:

$$w^\star = \arg\max_{w \in \mathcal{L}} \hat{S}_M^w.$$

3.3.3 Full End-to-End Integration

This section describes how we jointly performed segmentation and recognition on a line or a group of words. Recall from Section 3.2 that the output of the text detector generally consisted of a line of text or a group of words. Then, in Section 3.3.1, I described how we estimated a set of candidate spaces for the groups of words identified by the text detector.

In order to integrate the detection and classification components into a complete end-to-end system, we first assumed that for each bounding box B computed by the text detector, the true segmentation of the words in B could be constructed from the set S of estimated spaces for the bounding box. In other words, if there were multiple words contained in B, then the space between every adjacent pair of words in B must be contained in the set S of estimated spaces. Thus, it was acceptable for S to be an over-segmentation (contain a space at a location that does not contain a space in the ground truth), but *not* an under-segmentation (miss a space between two words) of the words in B. With this assumption, we were able to systematically evaluate different word-level segmentations using a beam search [34], a variant of breadth-first search that explores the top N possible partial segmentations of the line or group of words according to some heuristic score. In our case, we took the heuristic score of a candidate segmentation to be the sum of the augmented alignment score (described in Section 3.3.2) over each individual segment. By searching over this set of possible segmentations as allowed by the list of candidate spaces, we were able to recover the true segmentation of the words in the bounding box. To deal with possible false positives from the text detection stage, we also thresholded individual segments based upon their alignment scores. Segments with low recognition scores were thus filtered out as being "non-text." At the end of the beam search, we thus obtained a segmentation of the words in the bounding box, along with the identity of the words in each segment. In this manner, we achieved complete end-to-end recognition.

3.3.4 Recognition without a Specialized Lexicon

Much of the work described so far has hinged on the existence of a lexicon. As elaborated upon in Section 2.1, specialized lexicons are available in many different scenarios. Nonetheless, a truly general system should not make the assumption that a specialized lexicon would always be available. In this section, I describe how we generalized our system to the setting where we do not have access to a specialized lexicon.

One approach to generalizing the system to the case where a specialized lexicon is not readily available is to simply use the entire English dictionary, perhaps augmented with a list of common names, abbreviations, and proper nouns, as the lexicon. However, if we apply the method described in Sections 3.3.2 and 3.3.3, computing the alignment score for each candidate segment requires a full pass over the lexicon. Thus, using the full English language as the lexicon would not be computationally feasible. We therefore considered an alternative approach.

Our approach was to construct a lexicon by leveraging a spell-checking system. As in the lexicon-constrained setting, we first ran the text detection system to obtain a candidate set of bounding boxes for the regions of text in the image. For each bounding box, we computed the character classifier responses across the bounding box. As in Section 3.3.2, we next applied NMS to the character classifier responses in order to estimate the locations of each character in the bounding box. We then extracted the most-likely character at each of these identified positions and concatenated the corresponding characters together to form our initial guess for the underlying word. We next provided the candidate word to Hunspell, an open source spell-checker[2] to obtain a set of possible corrections. Note that in practice, we augmented the default Hunspell English lexicon with lists of common names, places, and other proper nouns that tended to occur in natural images. Using Hunspell then, we were able to obtain a set of suggested words, which we used to dynamically construct a lexicon. Armed with this lexicon, we simply applied the algorithm described in Section 3.3.3 to obtain our final end-to-end results. Using this method, we were able to extend our system to work in more general environments where we did not have access to a specialized lexicon.

[2]Hunspell is available for download at `http://hunspell.sourceforge.net/`.

Chapter 4

Experiments

In this chapter, I describe our evaluation of the text detection, character recognition, and the full end-to-end system. In addition, I show how our systems compare to other text recognition systems when measured against standard benchmarks. For all of these experiments, we evaluated our end-to-end recognition system on the ICDAR 2003 Robust Reading [25] and Street View Text (SVT) [38] datasets.

Before proceeding to the evaluation results, I briefly describe the characteristics of our particular evaluation sets. The ICDAR dataset is a fairly standard dataset used to assess text detection and recognition systems. The ICDAR images are taken from multiple sources, and include images of book covers, street signs, as well as store signs. On the other hand, the SVT dataset consists of images taken from Google Street View. As a result, images from this dataset tend to exhibit much higher variability and oftentimes have lower resolution compared to the images in the ICDAR dataset. Because of the increased variability and lower resolution, recognition over the SVT dataset is a much more challenging task compared to recognition over the ICDAR dataset. At the same time, compared to the ICDAR dataset, the SVT dataset is a much better approximation of the kinds of real-life conditions under which which a text recognition system would operate. As an additional note, the SVT dataset is designed purposefully for lexicon-driven end-to-end text recognition; as such, a lexicon is included with each image. One caveat, however, is that not all of the text in the image is represented in the lexicon. The goal on this dataset is to *only* identify the words in the image that appear in the lexicon. Other words that appear in the scene, but not in the lexicon should not be identified. An example of an image from the SVT dataset along with its ground truth annotations is shown in Figure 4.1.

Figure 4.1: Sample example from the Street View Text dataset [38]. The words within the green boxes are ones that are labeled in the ground truth, and therefore, included in the provided lexicon. The other words in the scene are not a part of the ground truth annotations and thus should *not* be recognized in the context of the SVT task. These extraneous words do *not* appear in the provided lexicon.

4.1 Text Detection

This section describes our evaluation of the text detection system. Recall that the output of the detector was a set of bounding boxes corresponding to groups of words in the image. Based upon the true set of bounding boxes for the text in the image, we assessed the performance of the detector. Like the authors of [11, 30, 32], we evaluated using the modified precision and recall metrics prescribed by [25]. For context, I briefly describe this criterion here. First, given two bounding boxes r_1, r_2, we take the match m_p to be the ratio of the area of $r_1 \cap r_2$ to the area of the smallest rectangle that contains both r_1 and r_2. Then, the match m between a rectangle and a *set* of rectangles R is given by

$$m(r, R) = \max \left\{ m_p(r, r') \mid r' \in R \right\}.$$

Effectively, we compute the score of the best match between the rectangle r and some rectangle r' in R. Next, given the set of predicted bounding boxes E and true bounding boxes T, we take the precision p and recall r of the detector to be

$$p = \frac{\sum_{r_e \in E} m(r_e, T)}{|E|} \qquad r = \frac{\sum_{r_t \in T} m(r_t, E)}{|T|}.$$

Algorithm	Precision	Recall	F-Score
Pan et al. [32]	0.67	0.71	0.69
Epshtein et al. [11]	0.73	0.60	0.66
Neumann et al. [30]	0.59	0.55	0.57
Ashida [25]	0.55	0.46	0.50
HWDavid [25]	0.44	0.46	0.45
Wolf [25]	0.30	0.44	0.35
Our approach	**0.42**	**0.36**	**0.39**
Our approach (coalescing lines)	**0.55**	**0.56**	**0.55**

Table 4.1: Text detector performance on the ICDAR 2003 dataset.

We can combine the two measures by taking the F-score, defined to be the harmonic mean of the precision and recall:

$$f = 2 \cdot \left[\frac{1}{p} + \frac{1}{r}\right]^{-1} = \frac{2pr}{p+r}.$$

We evaluated the detector performance over the ICDAR 2003 test set. To report results, we selected the detection threshold that corresponded to the highest F-score. Table 4.1 compares our method to other methods on the ICDAR 2003 dataset.

From the table, it is evident that our standard approach performed poorly compared to other methods. One reason for this was that the ICDAR 2003 Robust Reading ground truths consisted of bounding boxes for individual words, while our detection system outputted bounding boxes for entire lines of text. As a result, even if the detector successfully identified the text in the image, the match scores between a bounding box for an entire line and a bounding box for a single word tended to be very low. Thus, in order to assess the detector's ability to identify lines and groups of text, we made a small adjustment to the ground truth bounding boxes. In particular, we coalesced bounding boxes that resided along the same line; we then measured the precision and recall of the detector using these coalesced boxes as the ground truth. An example of this process is illustrated in Figure 4.2.

By testing against the coalesced bounding boxes, we observed a significant jump in the overall performance of the detector; in particular, the F-score increased by 0.16 on the ICDAR test set. Because of this adjustment to the ground truth bounding boxes, the performance metrics were no longer appropriate to use as a source of comparison with other work in this area. However, in terms of understanding how well the detector was at identifying lines of text, this adjusted metric was much more representative. Nonetheless,

(a) Original ICDAR ground truths. The ground truth bounding boxes are for individual words. Detector performance on this image: 0.3137 (precision), 0.2836 (recall).

(b) Ground truth bounding boxes after coalescing ones that reside on the same line. Detector performance on this image: 0.5001 (precision), 0.6251 (recall).

Figure 4.2: Visualization of ICDAR ground truth bounding boxes and coalesced ground truth bounding boxes. The green bounding boxes are the ground truths and the blue bounding boxes are the detector outputs.

the results still suggested that the detector's performance was still below that of several other text-detection systems. One of the primary limitations of our text detection system was the fact that it did not precisely crop out the text. As evidenced by the bounding boxes in Figure 4.2, the estimated boxes tended to be larger than the ground truth bounding boxes. In general, there were small amounts of padding between the extents of the words and the boundaries of the estimated bounding box.

I describe two possibilities that could have led to suboptimal performance with respect to the ICDAR metric. First, the examples we used to train the detector consisted of well-centered characters, and not necessarily perfectly-cropped characters. Moreover, the positive examples used to train the detector (examples shown in Figure 3.4) generally did not have uniform padding. As a result, the detector was tuned for recognizing the presence of a well-centered character and was relatively robust to the amount of padding present. Thus, the detector did not favor a tightly-cropped bounding box over a bounding box with small amounts of padding; both would receive similar scores as long as the characters in each were well-centered). Another factor that contributed to this suboptimal performance was the small number of scales we considered. Because we only considered a limited range

of scales, the possible heights of the estimated bounding boxes were confined to a small set of possible values (one for each scale). As a result of this limitation, it may simply have been impossible for the detector to produce a better cropped bounding box for a given line or set of words.

The limitations of our detection system described here did not fully apply to the purpose-built systems in [11, 30, 32], which generally employed additional processing steps to obtain well-cropped bounding boxes. Thus, it should not be too surprising that these more sophisticated systems were able to achieve better-cropped bounding boxes and thus, superior performance on this particular metric. It is important to note, however, that our goal here was not to achieve perfectly cropped bounding boxes, but rather, to obtain full end-to-end results. As long as the character recognizer was robust to these variations in padding, the bounding boxes returned by the detector were sufficient for the task.

4.2 Character and Word Recognition

In this section, I present a detailed evaluation of the complete text recognition pipeline. I begin by briefly describing the performance of the system on the tasks of cropped-character recognition and word recognition. For these two tasks, we evaluated our system on the ICDAR 2003 Robust Reading dataset [25].

4.2.1 Cropped Character Recognition

We first evaluated our character recognition module on the ICDAR 2003 test set, which consisted of 5198 test characters taken from 62 classes (10 digits, 26 uppercase and 26 lowercase letters). In the cropped character recognition task, we are given a single bounding box containing a well-centered character; the goal then is to identify the character in the bounding box. This is effectively the same 62-way classification problem described in Chapter 3. As described in Section 3.1.2, the character recognition module consisted of a two-layer CNN trained on a mix of real and synthetic data. Table 4.2 compares the performance of our system on the ICDAR 2003 test set to that of other character-recognition systems. As the table shows, our system achieved state-of-the-art accuracies in the task of cropped character recognition. This performance was a testament to the representational power of our convolutional neural architecture. By applying supervised fine-tuning over a large, multilayer convolutional architecture, we were thus able train a very robust and

Algorithm	Accuracy on ICDAR 2003 Character Test Set
Wang et al. [39]	52%
Wang et al. [38]	64%
Neumann et al. [30]	67.0%
Coates et al. [7]	81.7%
Our approach	**83.9%**

Table 4.2: Character recognition accuracy on the ICDAR 2003 character test set. Note that the result by Neumann et al. [30] was achieved without pre-segmented characters.

highly-accurate character recognition system. In turn, this robustness enabled us to use simpler algorithms to complete the integration of the detection and recognition subsystems and obtain a final end-to-end result.

4.2.2 Cropped Word Recognition

For the subsequent experiments that are described in the following sections, we adopted the lexicon-constrained framework described in Section 2.1.3. In particular, for the set of experiments in cropped word recognition and full end-to-end recognition, we assumed the existence of a lexicon. To provide necessary context, I compare the performance of our system with the similarly constructed lexicon-driven system described in [38]. In this section in particular, I assess the performance of the word recognition system from Section 3.3.2.

For the task of cropped word recognition, we evaluated on the ICDAR 2003 Robust Word Recognition dataset. In this case, the input images consisted of perfectly-cropped words such as the ones shown in Figure 4.3; the goal, of course, was to identify the word in the image. In the context of a full end-to-end system, the word recognition task would be analogous to measuring the best-possible recall assuming we had a perfect text detection system. Thus, the performance of our method (as described in Section 3.3.2) on this word recognition task would generally translate to an upper bound on the performance of our complete end-to-end system. We assessed the performance of our word recognition method by computing the case-insensitive word-level accuracy (percentage of words that we correctly identified, disregarding case). This case-insensitive metric represented a departure from the previous experiment with character recognition where we measured the performance under a case-sensitive setting. In this more challenging problem, we thus relaxed the case-sensitive

Figure 4.3: Sample images from the ICDAR 2003 Robust Word Recognition dataset. In this task, we are provided a perfectly cropped bounding box and the objective is to recognize the word in the image.

Algorithm	ICDAR (50)	ICDAR (Full)
Wang et al. [38]	76%	62%
Our approach	**90%**	**84%**

Table 4.3: Word recognition accuracy on the ICDAR 2003 dataset.

constraint and focused on just identifying the given word without trying to also distinguish the case of the underlying letters. We should note that in practice, it is generally sufficient to just identify the correct word and not to be concerned about the precise case of the characters. Thus, switching to a case-insensitive setting did not necessarily represent a significant loss of information or generality in our final end-to-end system.

Because we were now working in the lexicon-constrained setting, in each of our experiments, therefore, we provided the word recognition system a lexicon of candidate words. In the first experiment, denoted ICDAR (50), we provided the end-to-end system a lexicon consisting of the ground truth words (the words that actually appeared in the image) augmented by a list of 50 random "distractor" words (words that do not appear in the image). Thus, given an input image, the goal was to identify the correct word among a list of 50 or so possibilities. In the second experiment, denoted ICDAR (Full), we constructed a lexicon consisting of all of the words that appeared in the ICDAR 2003 test set, yielding a lexicon of more than 500 words. To compare performance with the lexicon-constrained end-to-end system in [38], we used the same lexicons that those authors used in their experiments. Table 4.3 compares the performance of our word-recognition system with the corresponding system in [38]. As of this writing, we do not know of any other word recognition system that has been evaluated on the ICDAR 2003 Word Recognition dataset.

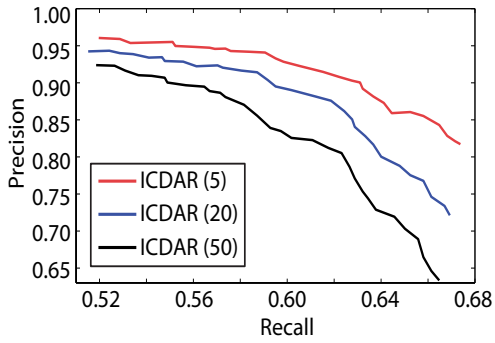
As evidenced in Table 4.3, our word recognition system achieved much higher accuracies than the system in [38] (an improvement of 14% and 22% on the ICDAR (50) and ICDAR (Full) setups, respectively). Moreover, the decrease in performance as we increased the size of the lexicon from 50 or so words to 500+ words was substantially smaller: a 6% decrease in

performance in the case of our system compared to a 14% decrease in the case of the system in [38]. This difference suggests that our model is more robust in a more general setting. In particular, our end-to-end recognition system did not require an extremely specialized lexicon to achieve high performance. Part of this gain in performance was due directly to the higher accuracy of our character classifier. Compared to the character classifier in [38], which obtained 64% accuracy on the ICDAR 2003 character sets, our character classifier obtained an accuracy of almost 84%. Naturally, with more accurate character recognizers, we were better able to discern the characters in the word and, thus, form better estimates regarding the identity of the underlying word. More generally, this result shows that by constructing a high-performing character recognition system, we can obtain state-of-the-art word recognition without needing to leverage more sophisticated methods such as using probabilistic inference or pictorial structure models.

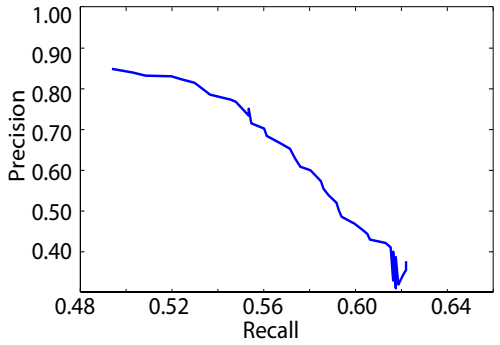
4.3 Full End-to-End Text Recognition

This section completes the experimental analysis by describing the performance of the full end-to-end system on the ICDAR 2003 Robust Reading dataset as well as the Street View Text dataset. In the end-to-end setting, the input to the recognition system consisted of a full image and the task was to locate and identify all of the words in the image. To do so, we began by using the text detection system to identify a set of bounding boxes for lines and groups of text. Then, as described in Section 3.3.1, we estimated the locations of the spaces. We next evaluated these estimated space locations using a beam search to jointly determine the optimal segmentation of the line and the identity of the underlying words.

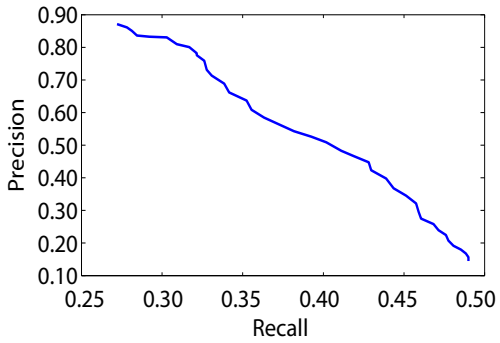
As in the case of word recognition, we worked under the lexicon-constrained setting. In the case of the SVT dataset, we used the provided lexicons, and in the case of the ICDAR dataset, we used the lexicons consisting of 5, 20, and 50 distractor words, as provided by the authors of [38]. We also conducted the end-to-end experiment using the full lexicon consisting of all of the words in the ICDAR test set. In the discussion below, I refer to these experiments as SVT, ICDAR (5), ICDAR (20), ICDAR (50), and ICDAR (Full), respectively. In all cases, we used the standard evaluation criterion described in [25] and used in [38]. These evaluation criterion are very similar to those used in other object recognition competitions, such as the PASCAL Visual Object Recognition Challenge [12]. More precisely, a predicted bounding box is considered to be a "match" if it overlaps a



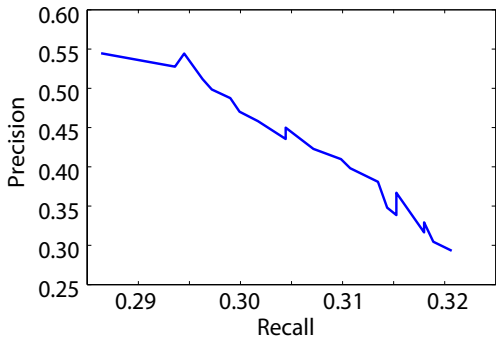
(a) Precision and recall curves on ICDAR 2003 using lexicons with 5, 20, and 50 distractor words. Best F-scores for ICDAR (5), ICDAR (20), ICDAR (50): 0.74, 0.73, 0.70, repectively.



(b) Precision and recall curve on ICDAR 2003 using a complete lexicon consisting of all words from the ICDAR test set. Best F-score: 0.64.



(c) Precision and recall curve on SVT using provided lexicons. Best F-score: 0.46.



(d) Precision and recall curve on ICDAR 2003 using Hunspell. Best F-score: 0.38.

Figure 4.4: Precision and recall curves for end-to-end evaluation on the ICDAR and SVT datasets.

ground truth bounding box by more than 50% and the predicted word matches the ground truth. As in the case of word recognition described in the preceding section, we ignored the case when comparing the predicted word to the ground-truth word. Using this convention, we measured the precision and recall of the full end-to-end system. Plots of the precision and recall curves for these experiments are presented in Figure 4.4.

To summarize results, we also computed the highest F-scores across the precision and recall curves for each of the end-to-end experiments. Table 4.4 compares the F-scores obtained by our end-to-end system with those obtained by the system in [38]. As evidenced by the table, our system achieved higher F-scores in every case. Furthermore, the margin

Algorithm	ICDAR (5)	ICDAR (20)	ICDAR (50)	ICDAR (Full)	SVT
Wang et al. [38]	0.72	0.70	0.68	0.51	0.38
Our approach	**0.74**	**0.73**	**0.70**	**0.64**	**0.46**

Table 4.4: F-scores from end-to-end evaluation on the ICDAR and SVT datasets.

of improvement between the performance, as measured by the F-score, of our end-to-end system and that of their system was much higher on the more difficult benchmarks: a difference of 0.13 on ICDAR (Full) and 0.08 on SVT. Once again, this result suggests that our system is more robust in a general setting where we either do not have access to a very specialized lexicon or in images where there is additional background clutter and increased variation in fonts and lighting conditions.

4.3.1 Recognition without a Specialized Lexicon

As a final experiment, we extended our model to the more general environment where we did not have access to a specialized lexicon. As described in Section 3.3.4, we leveraged an open-source spell checker, Hunspell, to dynamically construct a lexicon based upon the raw classifier responses. Figure 4.4d shows a plot of the precision and recall of this general-lexicon system on the ICDAR dataset. Table 4.5 compares the performance of our system to the system in [30], which notably does not rely on any lexicon or dictionary. As of this writing, I am not aware of any other full end-to-end recognition system that has been evaluated on the complete ICDAR 2003 Robust Reading dataset.

When we compare the precision and recall curves for the experiments where we supplied the recognition system a specialized lexicon to the experiment where we used Hunspell to dynamically construct a lexicon, we observed that the performance was significantly lower. This drop in performance was not surprising because the end-to-end system was designed to operate in a lexicon-constrained framework. Nonetheless, it is important to note that by using this simple extension with a freely available spell-checking tool, we are able to extend our end-to-end system to work in settings where there is no specialized lexicon. Furthermore, we observe that the performance of our system in this case is still comparable to state-of-the-art.

To conclude this section, I present some example outputs of the system from the ICDAR 2003 Robust Reading and SVT dataset. These are shown in Figure 4.5.

Algorithm	Precision	Recall	F-Score
Neumann, et al. [30]	0.42	0.39	0.40
Our approach	**0.54**	**0.30**	**0.38**

Table 4.5: Results from end-to-end evaluation on the ICDAR dataset in the general lexicon setting. Note that the results from [30] used case-sensitive comparison and did not leverage any form of lexicon or dictionary. Their work is the only other work that I am aware of that performs full, lexicon-free, end-to-end recognition on the ICDAR 2003 dataset.

(a) Example outputs of our end-to-end system on the ICDAR (Full) dataset.

(b) Example outputs of our end-to-end system on the SVT dataset.

Figure 4.5: Sample outputs from the full end-to-end system on the ICDAR and SVT datasets.

Chapter 5

Conclusion

To conclude, I first summarize the work presented in this thesis. I then describe some of the limitations of the current system and possible directions for future work.

5.1 Summary

The work described in this thesis presents an alternative means of approaching the problem of end-to-end text recognition using techniques from unsupervised feature learning and large scale convolutional architectures. In particular, our system integrates the specificity of learned features for capturing the underlying data with the vast representational power of a two-layer convolutional neural network. Using the *same* convolutional architecture, we are able to train highly accurate and robust text detection and character recognition modules. Then, by applying simple non-maximal suppression techniques in conjunction with a beam search, we are able to weave these two components into a complete, end-to-end system. This approach represents a departure from previous text detection and recognition systems which have generally required exquisite hand-engineering, prior knowledge, or sophisticated, multistage pipelines. As evidence of the robustness of our approach, I have presented our state-of-the-art results in character recognition, lexicon-based word recognition, and lexicon-based end-to-end recognition. In addition, I have shown that it is possible to extend the lexicon-constrained system to work in the absence of a specialized lexicon by simply leveraging freely available, open-source spell checking tools such as Hunspell. In this more general setting, the system again obtained results comparable to the state-of-the-art. Our results thus demonstrate the feasibility of using large, multilayer convolutional neural

networks as an alternative to purpose-built, hand-engineered systems for the problem of text recognition.

5.2 Limitations of the Current System and Future Directions

In this section, I highlight some of the limitations of our current end-to-end recognition system and propose some directions for future work. First, we observed a substantial gap between the performance of our system on lexicon-driven, cropped word recognition versus the full end-to-end text recognition task. Several reasons contributed to this performance gap. First, as noted in Section 4.1, the bounding boxes estimated by the text detection system were not perfectly cropped. Since the word-recognition system was specially tuned to operate on perfectly cropped word-level bounding boxes, part of the observed decrease in performance in the full end-to-end system could have been due to the poorer quality of the bounding boxes provided by the detector. Another contributing factor was the fact that the recall of the full end-to-end recognition system was limited by the recall of the text detection system. For instance, if the text detector failed to detect a word or line in the image, then it was impossible for the overall end-to-end system to recover from the detection failure. Thus, improving the recall of the detector would certainly improve performance. One possible direction for future work, then, is to continue improving the performance of the text detection system and aim for better-cropped bounding boxes. For instance, one possible method to achieve better-cropped bounding boxes is to train the binary classifier used in text detection on consistent and well-cropped characters. That way, the text detection system becomes more selective for well-cropped bounding boxes.

Another limitation of the text detection system was the fact that the detection system focused on finding horizontal lines of text (described in Section 3.2.2). While this was a reasonable assumption in many cases and turned to work well in practice, it was, nonetheless, still a limitation in our system. This constraint became particularly problematic in cases where the text was slightly slanted, and thus appeared to span multiple lines. In these cases, the detector was unable to provide a well-cropped bounding box for the entire word or line. In more extreme cases where the characters in a word were vertically aligned or arranged in a curved shape, the text detector was simply unable to localize the word. Thus, generalizing the end-to-end system so that it is able to handle cases where the text is not horizontally aligned is another possible direction for future work.

A third limitation of our current end-to-end system was the fact that we did not have an effective means of estimating word boundaries in a single line of text. In other words, we did not have a good solution to the problem of word-level segmentation. In the lexicon-constrained setting, we were able to use the simple space estimation routine described in Section 3.3.1 along with a beam search over the possible segmentations to decent effect. However, in the more general setting where we relied on Hunspell, this space estimation and beam search approach did not generalize very well. Thus, better means of segmentation that is not heuristic-based would certainly improve the performance of our full end-to-end system, especially in the setting where we do not have access to a specialized lexicon.

Finally, we can also consider developing a system that relies on a specialized lexicon, but is also able to identify words that are not present in the lexicon. Even though in the real world, it is often the case that we have access to specialized lexicons, it is not always safe to assume that all the words that could appear in the scene is necessarily contained within the provided lexicon. In this case then, it would be useful to have a specialized lexicon, but also allow the model to predict words that appear in the scene but are not present in the lexicon. Such a system would be a blend of the lexicon-constrained framework and the spell-correction framework I have described in the course of this thesis. This kind of hybrid framework would in turn be a better approximation of real-world operating conditions than either the fully lexicon-driven framework or the fully general, lexicon-free framework.

To conclude, I hope that the above description provides a sense of direction for future research in the area of end-to-end text recognition. In this thesis, I have presented our approach to the problem of end-to-end recognition. We have made some progress in this field, but the full end-to-end text recognition problem remains unsolved. It is my hope that this research shall stimulate and inspire future work in this exciting area.

Bibliography

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA, 2006.

[2] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In *Computer Vision and Pattern Recognition*, 2010.

[3] Xiangrong Chen and A.L. Yuille. Detecting and reading text in natural scenes. In *Computer Vision and Pattern Recognition*, volume 2, 2004.

[4] 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.

[5] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. Technical Report IDSIA-04-12, Dalle Molle Institute for Artificial Intelligence, 2012.

[6] Adam Coates, Paul Baumstarck, Quoc Le, and Andrew Y. Ng. Scalable learning for object detection with GPU hardware. In *IROS*, 2009.

[7] Adam Coates, Blake Carpenter, Carl Case, Sanjeev Satheesh, Bipin Suresh, Tao Wang, David J. Wu, and Andrew Y. Ng. Text detection and character recognition in scene images with unsupervised feature learning. In *ICDAR*, 2011.

[8] Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

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

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

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

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

[13] X. Fan and G. Fan. Graphical Models for Joint Segmentation and Recognition of License Plate Characters. *IEEE Signal Processing Letters*, 16(1), 2009.

[14] I.J. Goodfellow, Q.V. Le, A.M. Saxe, H. Lee, and A.Y. Ng. Measuring invariances in deep networks. In *NIPS*, 2009.

[15] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, 2010.

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

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

[18] Bekir Karlik and A. Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. In *International Journal of Artificial Intelligence and Expert Systems*, 2010.

[19] A. Krizhevsky. Learning multiple layers of features from Tiny Images. Master's thesis, Dept. of Comp. Sci., University of Toronto, 2009.

[20] Quoc V. Le, Will Y. Zou, Serena Y. Yeung, and Andrew Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011.

[21] 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.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *IEEE*, 1998.

[23] H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Neural Information Processing Systems*, 2007.

[24] David Lowe. Distinctive image features from scale-invariant keypoints. In *IJCV*, volume 20, pages 91–110, 2003.

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

[26] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *ACL*, pages 142–150, 2011.

[27] U. Meier, D.C. Ciresan, L.M. Gambardella, and J. Schmidhuber. Better digit recognition with a committee of simple neural nets. In *ICDAR*, 2011.

[28] Michele Merler, Carolina Galleguillos, and Serge Belongie. Recognizing groceries in situ using in vitro training data. In *CVPR*, 2007.

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

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

[31] Yi-Feng Pan, Xinwen Hou, and Cheng-Lin Liu. A robust system to detect and localize texts in natural scene images. In *International Workshop on Document Analysis Systems*, 2008.

[32] Yi-Feng Pan, Xinwen Hou, and Cheng-Lin Liu. Text localization in natural scene images based on conditional random field. In *ICDAR*, 2009.

[33] Marc'Aurelio Ranzato, Christopher Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *NIPS*, 2007.

[34] Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik, and Douglas D. Edwards. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

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

[36] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *In Advances in Neural Information Processing Systems 17*, pages 1185–1192, 2004.

[37] P. Viola and M.J. Jones. Robust real-time face detection. *IJCV*, 2004.

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

[39] K. Wang and S. Belongie. Word spotting in the wild. In *ECCV*, 2010.

[40] Jerod Weinman, Erik Learned-Miller, and Allen 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.

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

[42] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas S. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009.