Easily Adaptable Handwriting Recognition in Historical Manuscripts

John Alexander Edwards

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-76
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-76.html

May 29, 2007

Easily Adaptable Handwriting Recognition in Historical Manuscripts

by

John Alexander Edwards III

A.B. (Princeton University) 1997

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in

Computer Science

and the Designated Emphasis in
Communication, Computation and Statistics

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor David Forsyth, Co-Chair
Professor Jitendra Malik, Co-Chair
Professor Dan Klein
Professor Peter Bickel

Spring 2007

The dissertation of John Alexander Edwards III is approved:

Professor David Forsyth, Co-Chair — Date

Professor Jitendra Malik, Co-Chair — Date

Professor Dan Klein — Date

Professor Peter Bickel — Date

University of California, Berkeley

Spring 2007

Easily Adaptable Handwriting Recognition in Historical Manuscripts

by

John Alexander Edwards III

Abstract

Easily Adaptable Handwriting Recognition in Historical Manuscripts

by

John Alexander Edwards III

Doctor of Philosophy in Computer Science and the

Designated Emphasis in Communication, Computation and Statistics

University of California, Berkeley

Professor David Forsyth, Co-Chair

Professor Jitendra Malik, Co-Chair

As libraries increasingly digitize their collections, there are growing numbers of scanned manuscripts that current OCR and handwriting recognition techniques cannot transcribe, because the systems are not trained for the scripts in which these manuscripts are written. Documents in this category range from illuminated medieval manuscripts to handwritten letters to early printed works. Without transcriptions, these documents remain unsearchable. Unfortunately with existing methods, a user must manually label large amounts of text in the target font to adapt the system to a new script. Some systems require that a user manually segment and label instances of each glyph. Others provide for less costly training, allowing a user to segment and label entire lines of text instead of individual characters. Still, the collections we consider are extremely diverse, to the extent that in some cases almost every document may be in a different style. Because of this, the cost of manually transcribing dozens of lines of text for each font is prohibitively high.

In this dissertation, we introduce methods that significantly reduce the manual labor involved in training a character recognizer to new scripts. Rather than forcing a user to transcribe portions of each target document, our system leverages general

language statistics to identify regions of the document from which it may automatically extract new training exemplars. Unlike document specific transcriptions, these language statistics may be generated in a largely unsupervised manner, allowing our system to automate the process of building a model of scripts. We demonstrate the effectiveness of the model thus generated by using it to build a search engine for a Medieval illuminated manuscript.

Professor David Forsyth, Co-Chair Date

Professor Jitendra Malik, Co-Chair Date

Acknowledgements

First and foremost, I want to thank my advisor, David Forsyth, for seeing me through every step of this journey. His passion for Computer Vision, and his boundless curiosity have always inspired me, while his good nature and quick wit have made these past years a pleasure. I count myself exceedingly lucky to have been able to work with him. I also want to thank the rest of my dissertation committee, Jitendra Malik, Dan Klein, and Peter Bickel for their time and insightful comments.

I want to thank all my collaborators over the years. A particular note of thanks to Yee Whye Teh, who played such an integral role in getting this project off the ground, also Roger Bock and Michael Maire, for all their help building the original datasets. Thanks to Pinar Duygulu-Sahin and Kobus Barnard for helping me get my feet under me right at the beginning. Thanks to Dave Blei for all his machine learning insights. A special thanks to Tamara Berg, Deva Ramanan, Brian Milch, Ryan White and Okan Arikan for being both my sounding boards and sanity checks over the years.

Thanks to all the other current and former members of the vision/graphics corner of Soda Hall who've made the past six years such a pleasure: Leslie Ikemoto, Greg Mori, Alyoshoa Efros, Andrea Frome, Ashley Eden, Adam Kirk, Tony Lobay, Bryan Feldman, Pushkar Joshi, Hayley Iben and many others.

A tremendous note of thanks to my parents, John and Kathy, for supporting me in every possible way for as long as I can remember.

Above all, I want to thank my wife, Geraldine. She is everything to me, and without her none of this would seem meaningful.

— *Jaety*

Dedicated to my wife, my greatest source of strength

Contents

Chapter 1

Introduction

Optical Character Recognition is one of the original problems of computer vision, with the first examples dating back to the 1950s.[1] In the past 15 years, OCR has enjoyed a great deal of commercial success. For relatively high resolution scans of printed works, the leading commercial systems report near 100% accuracy rates on many modern fonts. A 1999 survey [Mello, 1999], for example, reported 99.6% rates with the OmniPage package on documents scanned at 250dpi. These high quoted rates mask the fact, however, that modern OCR is in one sense still very brittle. Systems perform well on only a certain number of modern printed fonts. Outside of this core competency, the performance of OCR systems rapidly degrades. The same 1999 survey also noted that accuracy rates on selected printed works from the late 19th century fell to between 85 and 90%. For handwritten manuscripts, word accuracy rates of around 65% are considered state of the art. [Lavrenko *et al.*, 2004]

Figure 1.1 provides extracts from a variety of historical manuscripts on which current OCR methods will fail. Images of texts such as these are increasingly available in electronic databases, but most do not have electronic transcriptions. Be-

[1] See Mori et al [1995] for an extensive historical review.

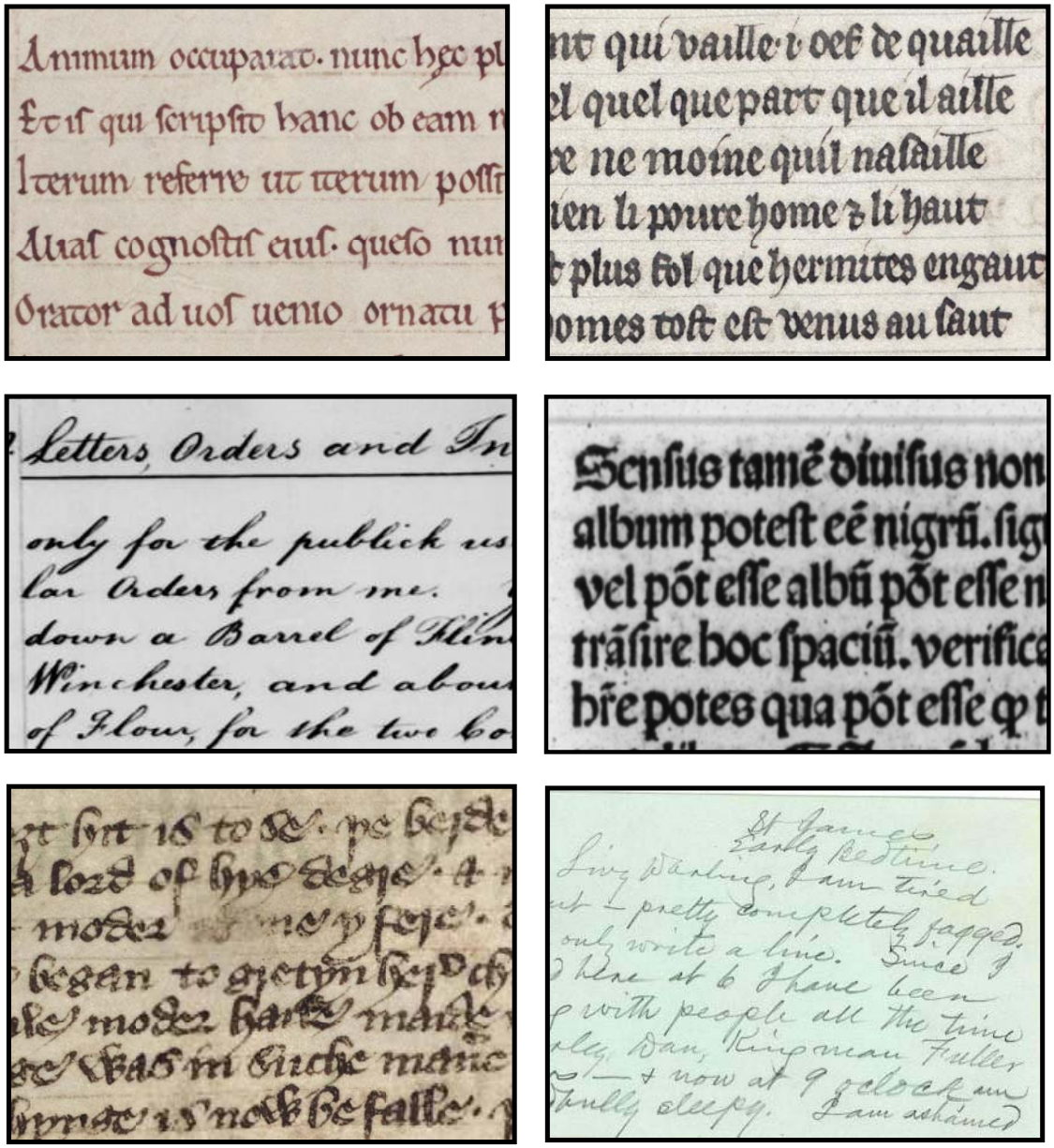


Figure 1.1: *Excerpts from various documents on which modern OCR will fail.* **Top Row***: A manuscript from the Bodleian Library of Terence's Comedies in Latin; The Romance of the Rose in French, also from the Bodleian.* **Middle Row***: An excerpt from a letter by George Washington and an early printed work, De Sensu Composito et Diviso by Heyte.* **Bottom Row***: A letter by Mark Twain; and an excerpt from a manuscript of Piers Plowman, Corpus Christi College.*

cause of this, their contents remain opaque to search. Examples of such manuscripts here at Berkeley range from the complete works and correspondence of Mark Twain to thousands of pages of zoological notes spanning two centuries. [Bancroft, 2006; Wilensky, 1995] The University of Oxford provides a portal to online collections of hundreds of Medieval manuscripts. [Oxford, 2007] Meanwhile, Google is pursuing the goal of scanning every book in many libraries around the world. Relative to the number of scanned pages printed in modern fonts, the number of historical and handwritten documents is minuscule. Business documents have historically and mostly likely will continue to represent the bulk of scanned materials. In absolute terms, however, the available collections of historical manuscripts is quite large and growing rapidly. In contrast to business or legal documents, there is no strong commercial incentive to provide manual transcriptions for documents where OCR fails. Because of this, most of these collections are opaque to search and will most likely remain so for the foreseeable future.

Accurately decoding Mark Twain's scrawled cursive or the degraded print of the Bodleian Library's Piers Plowman manuscript, illustrated on the last row of figure 1.1, is beyond the capability of current systems. However, the first two rows of the figure contain documents that while not printed, are still quite regular. The two Medieval manuscripts on the top line of figure 1.1 are good examples. The manuscripts produced by medieval scribes are notable for the regularity in their character forms. On the second line, George Washington's neat Copperplate hand is nearly as regular as that of the Medieval scribes. The second example on this line is an example of an early printed work, De Sensu Composito et Diviso by Heyte. Block prints and the glyphs from early movable type both show far greater variability between instances than modern printed fonts. For the purposes of OCR, they are more naturally grouped with handwritten scribal manuscripts than with later, more regular printed works. There is a wide spectrum of documents between those in clean, computer printed

fonts and those in scrawled handwritten cursive. The number of these intermediate documents is surprisingly large and many of their scripts seem sufficiently regular that one might imagine we could adapt current OCR techniques to decode them. However, no current OCR system can recognize them because we have no existing model for their fonts. What we require is an OCR system that can be easily trained on new scripts, even when those scripts are less regular than modern printed fonts.

The focus of research in OCR has however generally shifted away from the basic problem of character recognition to that of document layout. The field of Document Image Understanding (see Srihari [1986] for example) focuses on the broader problem of accurately representing the physical layout of a scanned document in an electronic form. Even in simple documents, lines of text must at the very least be ordered correctly in order to provide a coherent transcriptions. More complicated documents may contain many types of content apart from text, including images, graphs, maps and mathematical equations. Parsing documents into these constituent pieces is currently the most active area of OCR research. Still, any document understanding engine contains as a subproblem the task of transcribing a sequence of printed text. When we use the term *OCR* or *character recognition*, we are referring to this subproblem of Document Image Understanding. In Section 3.3 we will present methods for breaking documents in our collections into lines of text. In the rest of our work, however, we assume that the transcription of an input image can in fact be well represented as a sequence of ASCII or Unicode characters.

Document Image Understanding is an important problem, but as the documents of figure 1.1 illustrate there is still important work to be done in transcribing character strings, and adapting currently available OCR systems to new scripts is not an easy task. Traditionally, OCR research has emphasized efficiency over adaptability. This is historically understandable. Images are computationally expensive to work with, and thus most of the early challenges of OCR involved algorithms to work within the

speed and memory constraints of existing hardware. Moreover, the vast majority of scanned documents were then (and continue to be) business documents, and it was reasonable to assume that they were printed using a small set of known fonts. The unfortunate result of this historical path, however, is that OCR systems frequently employ many components manually optimized for a specific script. Because of this, it is generally difficult to tune them for new purposes.

In this dissertation, we reevaluate the problem of OCR from the perspective of adaptability. Collections of historical and handwritten manuscripts are far more heterogeneous than modern printed works. While the script within any one document may be regular, it is quite likely that any script model we learn may only apply to this single document. One of the core design assumptions of most OCR has been that the set of fonts to be recognized is known, and that system designers have access to an almost unlimited number of training exemplars, by which we mean labeled instances of character images. Providing the large number of character exemplars required by standard techniques is a practical impossibility if one must do this separately for each document in an archive.

1.1 An "Easily adaptable" OCR system?

We are interested in designing character recognition systems that can be easily adapted to recognize new, previously unknown scripts. Moreover, the ideal "easy" system could learn to transcribe this new font with very little manual intervention. To more crisply define "adaptability" and "manual intervention," we must introduce some terminology.

We can think of an OCR algorithm as a function mapping from images to character strings.

$$\text{OCR}(\text{image}) \Rightarrow (\text{char}_1, \text{char}_2, \ldots, \text{char}_N) \tag{1.1}$$

More generally, we may ask an OCR system to return a ranked list of word hypotheses. Let $f(\text{image}, \text{string})$ be a function that returns a real numbered value given an image and a candidate transcription. We rank alternate transcription hypotheses according to the scores provided by f. Equation 1.1 can be rewritten as the optimization problem

$$\text{OCR}(\text{image}) = \underset{s \in \text{strings}}{\text{argmax}}\, f(\text{image}, s) \tag{1.2}$$

An OCR algorithm also usually has a set of tunable parameters. For example, many OCR methods are designed to work with binary images. Given a grayscale image, pixels are assigned in a preprocessing step as either "ink" or "background," based on a threshold. This threshold value is a tunable parameter. Using θ to refer to the set of all tunable parameters for a system, equation 1.3 now becomes

$$\text{OCR}(\text{image}, \theta) = \underset{s \in \text{strings}}{\text{argmax}}\, f(\text{image}, s, \theta) \tag{1.3}$$

As we vary θ, we expect that the hypothesized transcriptions for a given image will also change. Thus the system's performance depends on finding good values for these parameters.

In modern OCR systems, these parameters are not generally set by hand, but rather by optimizing some performance criteria on a training set. Training is also an optimization problem, where the goal is to minimize some *loss function* L calculated over the training set. A natural training set for OCR might consist of a set of images and their corresponding correct transcriptions. Perhaps the easiest example of a loss function is the 0-1 loss, which is defined simply as the number of training exemplars that the system correctly transcribes. Let $T = ((I_1, s_1), \ldots, (I_N, s_N))$ be our training set pairs, and $\delta(\cdot, \cdot)$ be a function equal to 1 if its two inputs are identical and zero

otherwise. Under the 0-1 Loss function, the optimal parameters θ^* are[2]

$$\theta^* = \operatorname*{argmax}_{\theta} \sum_{(I,s)\in T} \delta(\mathrm{OCR}(I,\theta), s) \tag{1.4}$$

We can now provide a definition of an *adaptable* system. An OCR system is adaptable if it can be tuned to recognize a wide variety of fonts by providing an appropriate training set. This implies two things. First, for each font f, there exists a setting of the parameters θ_f^* such that the system accurately transcribes this font. Second, that provided a training set from some font f, we have a mechanism to find the optimal set of parameters. Most commercial OCR, for instance, is not adaptable under this criteria, at least for the end user. It provides no method to train the system for a different style. On the other hand, online handwriting recognition engines used on tablet PCs are generally adaptable. Plamondon and Srihari [2000] provide an extensive survey. These systems improve as the user provides more training data of his handwriting, but they also depend on knowledge of the pen trajectory available from the tablet input, and so are not directly applicable to the recognition of scanned historical manuscripts.

Adaptability has to do with the flexibility of the function $f(\cdot,\cdot,\theta)$. However, not all adaptable functions are equally easy to work with. We characterize a system as "easy" or "hard" based on the manual labor involved in providing a training set. This definition is qualitative but hopefully uncontroversial. Given an image of a line of text, for instance, we claim that it is easier to provide an ASCII transcription than it is to provide an ASCII transcription and the bounding box of each word. Providing the bounds for each word, in turn, is less cumbersome than manually segmenting the

[2]The 0-1 loss function is easily described, and thus a useful example. In practice, however, it is rarely used for OCR because its criteria for success is too strict. For instance, the 0-1 loss for a seven letter word makes no distinction between a system that correctly transcribes 6 out of 7 characters and one that gets all 7 wrong.

image into individual characters.

1.2 Decomposing the OCR Function

In designing an OCR system, we must balance three competing goals: adaptability, trainability and efficiency. We want a system with enough representational power to accurately recognize a wide variety of scripts under varied conditions. However, we also want a system that can be easily trained. Modern OCR systems often require hundreds or thousands of examples of each character, but these requirements are unreasonable for the historical manuscripts we consider. Finally, we must have computationally efficient methods for optimizing the OCR function to find the optimal parameters given a training set, and to find the optimal transcription given an image and values for the parameters.

To achieve the goals of trainability and efficiency, most systems decompose the OCR function of equation 1.3 into a set of smaller, interacting pieces. To a large degree, the history of OCR and handwriting recognition is a study of the effects of different decompositions. There are two broad families of decompositions that are perhaps the most fundamental, however, and crop up in most any OCR system.

- **Spatial Decomposition**: We can break the OCR function into smaller units that make use of only a portion of the input image. For example, if we can confidently segment a text image into individual words, we might try to translate each word separately.

$$f\big(g(\text{word image 1}, \text{string 1}), \ldots, g(\text{image N}, \text{string N})\big) \tag{1.5}$$

  Of course, breaking at word boundaries is not the only option. Alternative approaches attempt to segment the line into individual characters. Yet others

avoid segmentation altogether in favor of a series of measurements taken at a discrete set of locations along the line. As we will see in chapter 2, OCR systems can be roughly grouped by their approach to spatial decomposition.

- **Language and Visual Cues** The OCR function can be decomposed along different lines by making some functions language specific and others solely visual. For example, after breaking the line into individual characters, a system might try to recognize each character based solely on its visual appearance. Let $g(I, c)$ be the function that assigns a score for each character c given an image I.

  However, visual appearance is not the only feature humans use when reading text. A human reader also knows which strings are valid words and which are not. He or she can use this language knowledge to infer the correct transcription even if the word is blurry or otherwise visually ambiguous. We might devise a second function $h(\text{string})$ that gives a score based on whether this string is valid or not. This leads to an OCR function of the form

$$\text{OCR(image)} = \underset{s \in strings}{\text{argmax}}\, f\left(h(s), \sum_i g(I_i, s_i)\right) \tag{1.6}$$

  The final score will be some combination of a purely linguistic score from h and a purely visual score from the sequence of g's.

As we will see throughout the following chapters, finding a useful decomposition of the OCR function is the key challenge in handwriting and print recognition. We would like a decomposition that is adaptable and easily trained as defined in section 1.1. At the same time, we would like the family of functions to be such that it can achieve high accuracy given enough training data. Finally, we need to be able to efficiently compute results for the two core optimization problems of training to find optimal

parameters, and finding the optimal string, or ranking a set of strings.

Chapter 2

A Review of Adaptable Character Recognition

The history of OCR stretches back to the 1950s. However, the history of research into adaptable systems is considerably shorter. In general, OCR researchers and commercial systems have focused on improving accuracy and speed of recognition for a small set of of known modern fonts. Gary Kopec's Document Image Decoding work from the nineties is a notable exception to this trend, and provides one of the earlier attempts to frame OCR as an optimization problem over a family of functions. [Kopec and Lomelin, 1995] The question of adaptability has received broader attention from the handwriting recognition community, since the input to these systems has so much higher variability than print. In this chapter we review and draw comparisons between the major models presented in the existing literature. Major reviews of handwriting recognition include Steinherz et al. [1999] and Vinciarelli [2003]. Mori et al. [1995] provides an extensive historical review of OCR techniques.

2.1 Single Character Recognition

Perhaps the simplest formulation of the OCR problem assumes that the input images contain just a single character from a known alphabet. This is a classification problem from an input image to a small, fixed alphabet of characters A. In most systems, the input image is mapped to a fixed length vector, and a statistical classification method is trained from a set of labeled exemplar images.

The most widely tested dataset for the single character recognition problem is the MNIST collection [LeCun and Cortes, 1998], a set of handwritten digits drawn from postal address zipcodes. It consists of 60,000 training and 10,000 test images. These images have been size normalized and centered in the image window.

Lecun et al. [1998], report results on the MNIST collection for a wide variety of approaches. In the simplest setup, the image is reshaped into a vector whose values are the original pixels. Other methods preprocess the images in various ways including deskewing, blurring and denoising. K Nearest neighbors on the original images achieves a 5% error rate. Using preprocessing, this number is reduced to 1.22%. SVMs with a gaussian kernel on the original images achieve a 1.1% error rate, reduced to .8% with a 9^{th} degree polynomial kernel. The best performing method is the authors' special purpose neural network, the Convolutional Neural Net. With this system, they achieve an error rate of .4%.

Belongie et al. [2002] report comparable results using shape contexts. In this method, the set of edge points from an image are sampled at some fixed number of randomly selected points. The feature calculated at each of these points is a histogram of the locations of surrounding edge pixels, calculated with log-polar bins. Test images are classified by performing nearest neighbors to the test set, using a metric measuring the correspondence between these extracted points for each image. Performance for these datasets is extremely high. Their .4% error rate means that

only 40 out of 10,000 characters were misrecognized.

With test error rates this low, it seems that recognition of the postal address digits is basically a solved problem, and digit recognition has indeed made its way into real world applications for automatically recognizing postal addresses and the courtesy amounts on checks. [Srihari and Kuebert, 1997; LeCun *et al.*, 1997] However, providing 60,000 training exemplars involves *a lot* of manual labor. In addition, by providing images that are broken up into individual characters, this dataset makes the problem much easier. As we will see in the following sections, accurately segmenting a line into characters as a pre-processing step is difficult to accomplish automatically. In this dissertation, one major goal is to introduce algorithms to reduce the amount of manual supervision necessary to train an accurate OCR system on new scripts.

2.2 Holistic Word Recognition

Methods similar to those in section 2.1 have been applied to the recognition of larger strings as well. In this case, systems assume that the input consists of word images from some known vocabulary list. Some examples include [Kornfield *et al.*, 2004; Moreau, 1991; Leroux *et al.*, 1991] This vocabulary list serves the same role as the alphabet for single character recognition.

These systems extract a fixed length feature vector from the image, although the feature set is generally somewhat different than in single character recognition. For example, since word images are not of standardized size (as in the MNIST database), width and height are useful features. Rath and Manmatha [2003] treat various projections of the word image as 1-D signals, and convert them to fixed length vectors using the first few coefficients of their Discrete Fourier Transforms. Although the features are different that those used for single character recognition, the classification functions and training methods are similar. Lavrenko et al. [2004] fit a Gaussian mixture

model to training data, with one class for each word. Recognition of an entire text proceeds as a word level HMM, with this Gaussian model as the emission model.

The difficulty for holistic methods is that the range of the classification function grows linearly with the size of the vocabulary. In practice, this means that the training set must include multiple examples of each word the system is expected to classify. This requirement is frequently unachievable as the vast majority of words in a given language are very rare.[1] Holistic methods do not share parameters between different words, and because of this, have a serious problem with sparsity as the vocabulary size grows. Even for small vocabularies of common words, this method requires a fairly costly round of manual supervision.

In their "wordspotting" work, Lavrenko et al. [2004] use these methods on a large vocabulary, They report 65% word accuracy rates on a collection of letters by George Washington and Thomas Jefferson. They also introduce some interesting methods for reducing the cost of training. Most notably, they perform an initial unsupervised clustering of the word feature vectors. They can then label clusters instead of individual instances of words. Still, the overhead for a new document is substantial.

2.3 OCR as Sequence Alignment

To handle large or open vocabulary problems, non-holistic systems make use of the fact that both ASCII words and their visual representations are composed of smaller repeatable structures. This is explicitly true for printed works, and a reasonable assumption for handwritten ones. The basic units of printed text are generally referred to as *glyphs*. Historically, a glyph was the basic unit of movable type. For most ASCII characters, there is a one to one relationship between characters. Sometimes, however,

[1] See, for example, the discussion of Zipf's law in [Manning and Schütze, 1999]

character bigrams interact in a way that is not simply the concatenation of their two separate images. These are known as ligatures. Whistler et al. [2004] provide a useful discussion on the relationship between characters and glyphs for modern computer fonts. We will use the term glyph to refer to handwritten characters as well. It is worth noting that even when breaking a document into glyphs, the sparsity problem does not entirely disappear. For example, the capitalized version of some characters may be rare enough that the corresponding glyphs may happen only a few times even in reasonably long documents.

Since we are not usually provided with a segmentation of the image into glyphs, OCR systems must have some method for breaking an image into a discrete set of pieces. As we will see, these pieces may or may not correspond to our intuitive notion of a glyph. Some systems break a line into smaller pieces. Others refrain from segmenting the line at all and sample vectors of image features at a sequence of image points. They all, however, generate a discrete sequence of measurements and use this as the starting point of inference. We will refer to this family of approaches as *sequential models*, to contrast them with the holistic models of section 2.2. In section 2.4 we return to the question of segmenting the line. For the moment, however, we will assume that a correct segmentation is provided, and turn to the question of inferring a transcription given a sequence of glyph images.

2.3.1 Importance of Context

Assume for the moment that we could segment a line into a sequence of glyphs, and that the mapping between glyphs and characters was one to one. In this case, we could directly employ the single character recognition techniques of section 2.1 to generate a transcription by considering each character in isolation. However, systems will often be able to improve recognition rates by letting the identities of surrounding

characters influence the classification process for a glyph.

Visually, characters in some scripts have *contextual forms*, [Whistler *et al.*, 2004]. For example, in many early scripts, the first of a pair of s's was written in a form resembling an f, as in figure 2.1(a), an excerpt from a letter by George Washington. These effects are still present in modern fonts, for example in the connectors between cursive characters, figure 2.1(b). In these cases, the identity of adjacent characters directly influences the appearance of the current one.

(a) (b)

Figure 2.1: *2.1(a) the word "assigned", excerpted from a letter by George Washington. Note that the first 's' has a very different form than the second. This is called a contextual form. 2.1(b) A contextual form from a modern font, Lucida Handwriting. Note the connectors between "ca" in cat as opposed to "ba" in bat.*

A character's context is also useful because it allows us to bring *language knowledge* to bear. Humans do not need to accurately recognize every character in order to read, and this is because text in any language is not a random sequence of characters. If a reader sees the sequence "obv?ous", most English speakers will have no trouble filling in the missing character. In some situations, the identities of neighboring characters are so highly correlated, that given one a reader may predict another without even looking at the image. In an English language document, for example, given that the previous character is a "q", an OCR system could presumably predict the identity of the following character without ever examining the image itself.

2.3.2 Inference in Contextual Models

The major advantage of considering characters in isolation is that it is computationally efficient to infer a transcription. The computational cost grows only linearly with the size of the string. If we allow arbitrary interactions between characters, the complexity of inference becomes exponential in string length. However, if we restrict ourselves to pairwise interactions, optimization can once again be performed in polynomial time through dynamic programming. Most sequential models for character recognition are of this form. Many of these methods are variants of Hidden Markov Models and provide a generative model for an image of text. Others use undirected graphical models or energy-based models.[2] HMMs are a subfamily of this larger class of methods. Pairwise energy-based models for sequences have the following form:

Let $f_I(g, c)$ be a function that returns a real valued score given a glyph image and a candidate character label. Let $f_L(c_1, c_2)$ be a function that assigns scores to pairs of adjacent characters. For example, a good function would assign a high score to the pair ("q","u") and a low score to the pair ("q",anything else). We assign a *score* to a sequence of characters $c_{1...N}$ and an associated sequence of glyphs $g_{1...N}$ as

$$\text{score}(c, g) = f_L(\alpha, c_1) \prod_{i=1}^{N} f_I(g_i, c_i) \prod_{i=2}^{N} f_L(c_{v_1}, c_{v_2}) \tag{2.1}$$

Here α is a special dummy character that we assume begins any string.

Figure 2.2 provides a useful visual depiction of such a model for the word "cab". We define a *trellis graph* as a directed acyclic graph that can be depicted as a series columns of vertices with all edges running from left to right. We associate a label l_v and a glyph g_v with each vertex. If we assign the score $f_I(g_v, l_v)$ to each vertex, and the score $f_L(c_{v_1}, c_{v_2}$ to each edge $e = (v_1, v_2)$, then we can define the score of a

[2]The term energy-based model is used extensively by [LeCun *et al.*, 1998]

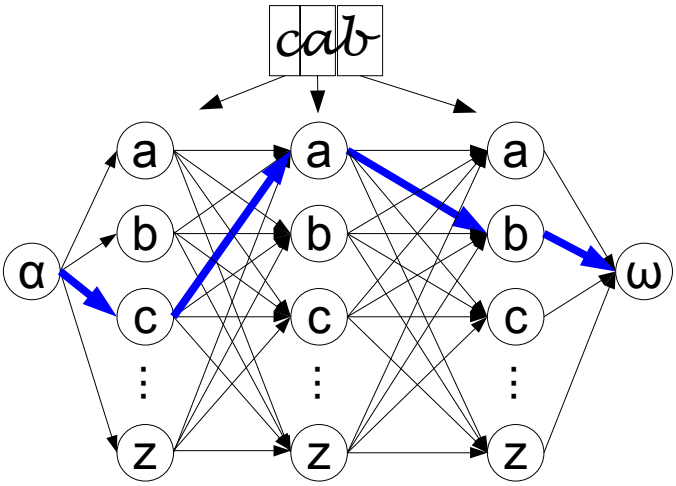


Figure 2.2: *Pairwise dependencies between adjacent characters can be represented as a trellis graph. A score is associated with each vertex and each edge. The vertex score rates the match between a glyph and a character. Edge scores allow adjacent characters to influence the classification of their neighbors. Each path through the graph defines a possible transcription. We add a special start state α and end state ω, and denote the correct path by the dark blue arrows. Finding the optimal route through such a graph is a single source, shortest path problem, and may be solved efficiently through dynamic programming*

particular transcription as the combination of those scores along a path in the graph from α to ω. Let V_p and E_p be the vertices and edges respectively of a given path p. The score of a path is

$$\text{score}(p) = \prod_{v \in V} f_I(g, c_v) \prod_{(v_1, v_2) \in E} f_L(c_{v_1}, c_{v_2}) \tag{2.2}$$

2.3.2.1 Hidden Markov Models

Models of the form in equation 2.2 are Hidden Markov Models if f_L is discrete, and both f_I and f_L are constrained to be conditional probability distributions. Impedovo [1993] provides an extensive review of the use of HMMs in handwriting recognition. We define f_I as the probability of the image g (or a feature x_g derived from the image) given the character label, and f_L as the probability of a character given the previous character. This represents a character bigram model. We can extend the model to character N grams by expanding f_L to depend on additional preceding characters. In this case, $f_L = \mathbb{P}(c|c_{\text{previous}})$ is a multinomial distribution over character labels.

A common method is to model f_I as a mixture of Gaussians. Vinciarelli et al. [2004] provide one example. A fixed length feature vector is generated from the image (as in single character recognition). The mixture parameters depend on the class label, and the Gaussian parameters are specific to the mixture.

$$f_I(g, c) = \mathbb{P}(x_g|c) = \sum_{i=1}^{K} \pi_c k \mathcal{N}(x_g; \theta_c k) \tag{2.3}$$

Here, the mixture has K components, feature vector x and character label c, the probability of a feature given a character is where π_k is the probability of the k^{th} mixture component, and $\mathcal{N}(f; \theta_k)$ is the probability of the feature f under a Gaussian with mixture specific parameters θ_k.

Hidden Markov Models are also extensively used in Speech Recognition. Models

are generally trained via Maximum Likelihood, although researchers have also trained these models using discriminative methods, see [Rabiner and Juang, 1993; Jelinek, 1997] for further references. The maximum likelihood path is calculated via the Viterbi algorithm, which is an instance of the single-source shortest path algorithm on a directed acyclic graph. [Cormen *et al.*, 2001]. See also [Jordan, Pre Print].The cost of the best path $c^*(v)$ from the start state α and ending at a given vertex v is given recursively in terms of the costs of v's direct ancestors A_v in the DAG

$$c^*(v) = f_I(v_g, v_c) + \min_{v' \in A_v} (c^*(v') + f_L(v', v)) \tag{2.4}$$

For probabilistic methods, the scores are log-likelihoods. Posterior probabilities for individual characters, i.e. $\mathbb{P}(c_i|\text{image})$ are calculated using the related forward-backward algorithm, which replaces the min in the Viterbi algorithm with

$$\log \sum_{v' \in A_v} \exp\left(c^*(v') + f_L(v', v)\right) \tag{2.5}$$

2.3.2.2 Energy-Based models and Undirected Graphical Models

Alternatively, we could forego a generative model, letting f_I and f_L be more general functions. Graph Transformer Networks [LeCun *et al.*, 1998], Conditional Random Fields [Lafferty *et al.*, 2001], and Max Margin Markov Networks [Taskar *et al.*, 2003] present various methods for training such models discriminatively. These methods assume that we are given a set of transcribed images as a training set. Optimization of these methods involves maximizing the *margin* between the true transcription of each training exemplar, and some set of alternatives.

The margin may be defined in a number of ways. We might, for example, maximize the difference between the score of the true transcription and that of the next best contender. Let $p* = (V^*, E^*)$ be the path of the true transcription in the trellis. Let

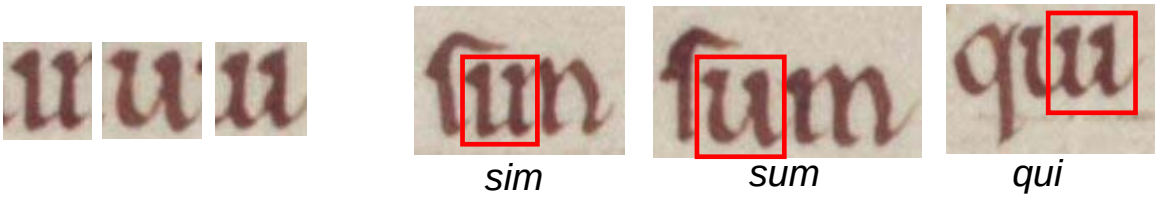


Figure 2.3: *In many scripts, characters are locally ambiguous. On the* **left**, *are three windows from the Terence manuscript. Only one of these is actually a "u". On the* **right** *we are given additional context. The first instance is in fact an "i" and the first third of an "m". The last is half a "u" and the following "i". Only the middle example is in fact a true "u".*

P_{alt} be the set of all other paths, and let T be a training set of images with their correct transcriptions. The optimization goal is then

$$\underset{\theta}{\text{argmax}} \sum_{t \in T} \left(\text{Score}(p*) - \max_{p' \in P_{\text{alt}}} \text{Score}(p') \right) \tag{2.6}$$

Alternately, in a probabilistic setting, the system may directly maximize $\mathbb{P}[p^*|\text{image}]$

$$\underset{\theta}{\text{argmax}} \sum_{t \in T} \left(\log \mathbb{P}(p*) - \log \sum_{p' \in P_{\text{alt}}} \mathbb{P}(p') \right) \tag{2.7}$$

2.4 Real World Segmentation Approaches

To use the above methods in the context of a longer string, we need some method to segment the image into characters. This is rarely easy. A rule of thumb that has come to be known as Sayre's paradox states that "it is impossible to segment without first recognizing characters, and it is impossible to recognize characters without a segmentation." [Sayre, 1973]

There are, in fact, many examples where even a human reader *must* rely on contextual cues to segment. Figure 2.3 shows three windows extracted from a 10$^{\text{th}}$

century manuscript of Terence's Comedies, written in Latin. This document is one of our main training sets, and we will return to it in chapters 3 and 4. All three windows appear to be a "u" when viewed in isolation, but as is clear when given additional context, only one actually is. The other two are not true characters, but combine portions of two adjacent glyphs, demonstrating mis-segmentations of the line.

At times, it may be theoretically possible to segment characters, but difficult to do so automatically. In the leftmost example of figure 2.3, the "i" in "sim" is nestled under the cross bar of the preceding "s". Historically, in movable type, these two characters would have been cast as a single unit, known as a *ligature*. There are many examples of ligatures in modern fonts as well. [Whistler *et al.*, 2004] OCR systems must choose how to manage ligatures. They may treat them as single characters, or break them into smaller pieces. The former approach more accurately models the true generative process, but the latter is often more computationally convenient as it preserves a one to one mapping between the ascii characters of a transcription and image pieces.

Since the segmentation is not normally provided, researchers have devised various methods of breaking a line of text into a finite sequence. In tandem with these approaches to segmentation, researchers have also had to introduce extensions to the inference mechanisms described in the last section in order to account for the fact that there is no longer a one to one correspondence between the ASCII characters of the transcription and the steps in the visual image sequence. In the following sections, we introduce examples of the major approaches to segmentation and the extensions to the sequential inference model that each method requires.

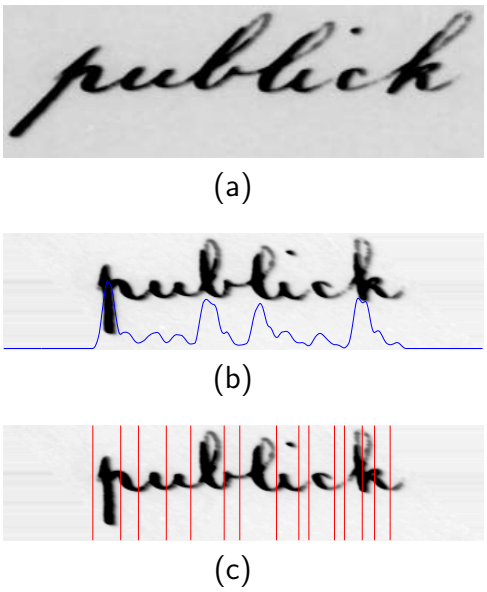


Figure 2.4: *In oversegmentation methods, the line is broken at a large number of places, in the hopes that the true glyph breaks will be contained in this larger set. In this example, the original image (a) is deslanted, so that strokes are roughly vertical. The projection profile (b) is computed by summing along columns and smoothing. Breaks are introduced (c) at local minima of this projection profile.*

2.4.1 Inference in Oversegmented Images

One method for working with images that cannot be easily split into single characters is to oversegment the line. This is also one of the earliest methods proposed. Burgest [1992] and Breuel [1994] illustrate two early examples.

The method entails identifying some repeatable feature at which to segment. Figure 2.4 shows an example. An image of text is first deslanted, so that strokes are roughly vertical. We derive a *vertical projection* from the image by first binarizing the text, then summing along each column of pixels, and finally smoothing the resulting signal with a Gaussian filter. The image is split at each local minima of this projection profile.

Oversegmentation methods rely on the assumption that the true breaks between

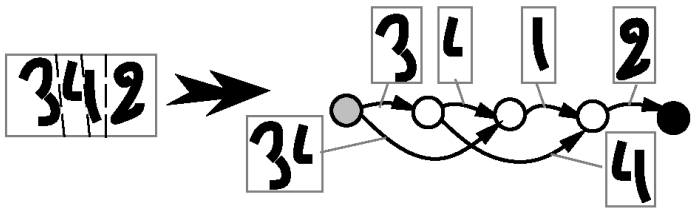


Figure 2.5: *This figure, borrowed from [LeCun* et al.*, 1998], demonstrates a partial lattice graph for oversegmentation. The system extracts a set of proposed break points. At each step along a path, one or more adjacent pieces may be combined as a single image, which the system then attempts to classify.*

characters will be contained in this larger set of segmentation points. If this assumption holds, and if the algorithm were told the correct subset of breaks to use, then the setup would be the same as in the previous section, with each image snippet corresponding to a single character. One approach is to exhaustively try different break subsets. For each subset, generate the best scoring transcription using one of the models from section 2.3.2, and choose as the final output the [break subset, transcription] pair with the best score. This is the method employed for example by Burges [1992] for example.

The trellis graph of figure 2.2 can be modified to allow this inference over different segmentations (See figure 2.5). First, the meaning of the vertices changes. Previously, each vertex represented a subimage. Now, each vertex represents a break. Each edge between adjacent columns represents one of the segments in the input image, but we now also allow longer edges, which correspond to the concatenation of multiple segments. In [Burges, 1992] and [Breuel, 1994], for example, each edge receives the score of a single character recognizer whose input is the corresponding image window. The output of these recognizers is a score between 0 and 1, with 0 representing a likely character. The optimal transcription is the shortest path in this new trellis, which can be computed using the same methods as in section 2.3.2. Inference is slightly

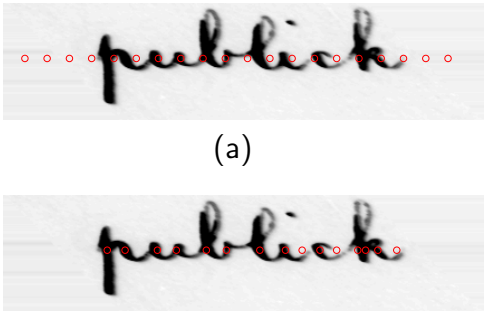
more expensive, because we have increased the number of edges. One caveat about this approach is that there is a bias toward shorter paths. The papers cited here safely ignore this wrinkle because their single character recognizers are sufficiently well behaved that they give very low scores to significantly too short segmentations.

These methods require a significant amount of training data. In [Breuel, 1994], 1000 characters are manually segmented from the NIST dataset [Wilkinson, 1994]. This dataset additionally provides 18,000 strings with their transcriptions, that they break (in an undisclosed percentage) between training and test. Given the initial 1000 characters, they train character recognizers, using these to segment the rest of the training set. Given this expanded set, they train new character recognizers.

2.5 HMM Based Methods

One major deficiency of oversegmentation methods is that they are fragile. If the original segmenter misses a glyph break, the subsequent inference algorithm cannot recover, because the single character recognizer is never presented with a valid character window. Vinciarelli et al. [2004] describe a method that avoids the problem of segmentation by using a *sliding window* technique. This is a model also extensively used in the speech recognition community. Instead of segmenting the image into subwindows, the image is treated as a 1-D signal, and feature samples are taken at regular points along the line.

The feature vector extracted at each interest point is modeled as a draw from a character specific mixture of Gaussians. Because characters may cover more than one sample point, the authors expand their alphabet, so that each character has 3 states associated with it c_1, c_2 and c_3. They experiment with different topologies. For example, in some c_1 may only transition to c_2 while in others it might also be able to transition directly to the next character. Characters may never transition



(a)

(b)

Figure 2.6: *In sliding window techniques, fixed length feature vectors are extracted from the image at a horizontal sequence of points. These points may be regularly spaced (a). Alternatively, one may define some interest point operator to create a sequence of image dependent feature points. In (b), for example, the interest points are the local maxima of the projection profile. The projection profile is shown in 2.4(c)*

backwards, however. In other words c_i may never transition to c_j for $j < i$ within the same character.

With this expanded alphabet, inference in the model proceeds using the same techniques as those in section 2.3.2.1. As with LeCun, the training data for this method is lines of text. It does not require that individual characters be hand segmented. This is a significant improvement in ease of training over earlier methods. However, the data requirements are still large. On two datasets of handwritten words, they use 3037 and 8062 labeled words respectively as training data.

2.5.1 Inference with Graph Transformer Networks

Lecun et al. [1998] introduce a different method to integrate segmentation more directly into inference. The authors' Convolutional Neural Networks achieved highly accurate performance on the MNIST single digit dataset, as described in section 2.1). In this work, they introduce a method, the Space Displacement Neural Network, which builds on their single character recognizer to infer characters in context. They

address the fragility of oversegmentation methods by exhaustively considering all possible locations for each character, at least to pixel resolution.

Their model expands the trellis of figure 2.2 so that there is a column of vertices for *every* column of pixels in the image. They allow edges between between widely separated columns, connecting every pixel with displacements in a small range about a manually provided mean width. In this work, they recognize the courtesy amounts on checks, which have been normalized for height, so they do not have to learn this spacing number. Moreover, CNNs show high robustness to translations of the position of the character in the input image, so this width parameter does not have to be finely tuned. There is an unaddressed problem in their work, namely that a character might, if the spacing is right, be counted as two separate instances. The digits in the courtesy check dataset was presumably fairly regularly spaced.

The key element of this work is that all of the parameters of the SDNN entire model, including both the neural net parameters of the CNN and the transition parameters between different character labels, can be optimized with respect to the transcription of the entire string via gradient descent. This addresses another deficiency of oversegmentation methods, where the segmenter and the single character recognizers are trained separately, and never optimized with respect to the ultimate goal of transcription accuracy. The authors also introduce Graph Transformer Networks, which provide an elegant framework for combining modules in such a way that the output remains differentiable with respect to the parameters, ensuring that the performance of the entire system remains optimizable through gradient descent. [Mohri *et al.*, 2000]

Performance on courtesy check amounts was quite high, and the system proved robust to various kinds of introduced noise. The system has been used in real world applications for recognizing the courtesy amounts on checks. [LeCun *et al.*, 1997] To our knowledge, however, it has never been extended for use in more general OCR or

handwriting recognition systems with full alphabets.

2.5.2 Document Image Decoding

Gary Kopec [1995] introduced the Document Image Decoding model. It is notable because it represents the most extensive early attempt to present OCR in a mathematically clean way. Moreover, it provides a quite natural generative model of printed texts, and it gives an early example of an adaptable system.

Its representation of space is very close to that which we use in our Overlap gHMM of chapter 4. Their model for a glyph was a binary image with ones at inked pixels, and zeros for the background. They called this binary image the *template*. Since they were concerned mostly with printed fonts, there is in fact an idealized glyph of this form. Paraphrasing the generative model, assume that each glyph template is printed on a transparency such that ink pixels are opaque and all others are perfectly transparent. We can then generate an image by placing a series of these transparencies down on a larger canvas, each at a different offset relative to a global coordinate system. The final template for a page is the logical OR of each of the transparencies. Finally, this page template may be corrupted by noise. Inference in this model is intractable, but Kopec introduced a greedy approximation method to allow its estimation.

2.6 Vocabulary and Language Models

We have discussed approaches to segmentation, but have not yet discussed uses of language knowledge. Traditional OCR makes very little use of language priors, depending instead on accurate single character recognizers. Handwriting recognition has had to rely on language to a greater extent because of the larger ambiguity in the input images. Vinciarelli [2004] shows marked improvements in an HMM based

sliding window model by moving from using only visual features to a model that incorporates trigram character statistics. In chapter 3, we see similar performance boosts in our research.

N-gram character models fit naturally into the dynamic programming models introduced in this chapter, although the computational costs for n-grams larger than trigrams becomes quite large. A different approach is to use a large wordlist and constrained inference. Any of the sequential models described in this chapter may be used to score individual words in this way. A system builds a modified trellis, constrained so that every path must contain the lexicon word as a substring. In this way, it can generate scores for each word in a wordlist, and pick the best from that set. This hearkens back to our discussion of holistic methods in that transcription is now treated as a classification problem over a known vocabulary, It has the benefit over holistic approaches that words can be added to the dictionary without retraining the model. Because the computational cost grows linearly with the size of the vocabulary, however, there are serious practical issues with using this method on large vocabulary problems.

An interesting exception to this rule is the method employed by Madhvanath et al. [1997] to recognize handwritten addresses on envelopes. They begin by recognizing the zip code. Using this information allows them to reduce the lexicon to the point where holistic methods can be employed to classify words. Holistic methods can be very useful even with very large dictionaries *if* you are working in a domain where other information can help you initially prune the vocabulary.

2.7 Moving Forward

All of the methods presented in this chapter require significant amounts of manual supervision during the training process. Character recognizers have traditionally been

trained using large numbers of manually segmented character exemplars. Vinciarelli, LeCun and Kopec each offer methods that alleviate the difficulty of training. In their works, one need only provide transcriptions of lines of text, not individual characters or even words. However, each of these approaches still require a manual transcription of many dozens of lines from the target document. If our goal is to OCR historical manuscripts en masse, we cannot assume that there will be the manpower to transcribe even a small set of lines for every document.

One of the biggest problems facing any OCR method is the question of segmentation. Each method presented in this chapter takes a somewhat different approach, and each choice impacts the ease with which their models may be trained. For example, sliding window techniques and the Space Displacement Neural Network both build robustness to translation directly into their appearance models for characters. This makes their inference step less sensitive to accurately localizing characters, but also increases the amount of training data they must provide to initialize their models. In the following chapter, we introduce a model, the generalized HMM, that is far easier to initialize than existing techniques.

Chapter 3

Searching Historical Documents with Generalized Hidden Markov Models

Statistical models of handwritten text rely on two distinct families of features: language cues that describe which strings of ASCII text are likely, and visual cues that determine how closely a given image window matches each of the possible character labels. For the types of historical manuscripts we consider, the script may be unique to a specific document, but the language statistics largely are not. We can thus fit language parameters much more easily than visual ones by training them against separate collections of ASCII texts that are easy to collect from the publicly available sources. In this chapter, we investigate the degree to which these easily trainable language cues may compensate for visual ones, particularly when the end goal is searching a text for query words (as opposed to providing a full transcription). To a surprising degree, we can use language cues to differentiate visually ambiguous text images. This is key since we are interested in building systems that easily adapt to new scripts, and language models generalize across documents more easily than visual character recognizers.

We introduce a statistical method, based on a generalized hidden Markov model, and demonstrate its efficacy in searching a medieval manuscript from the Bodleian collection. This is a twelfth century manuscript version of Terence's comedies.

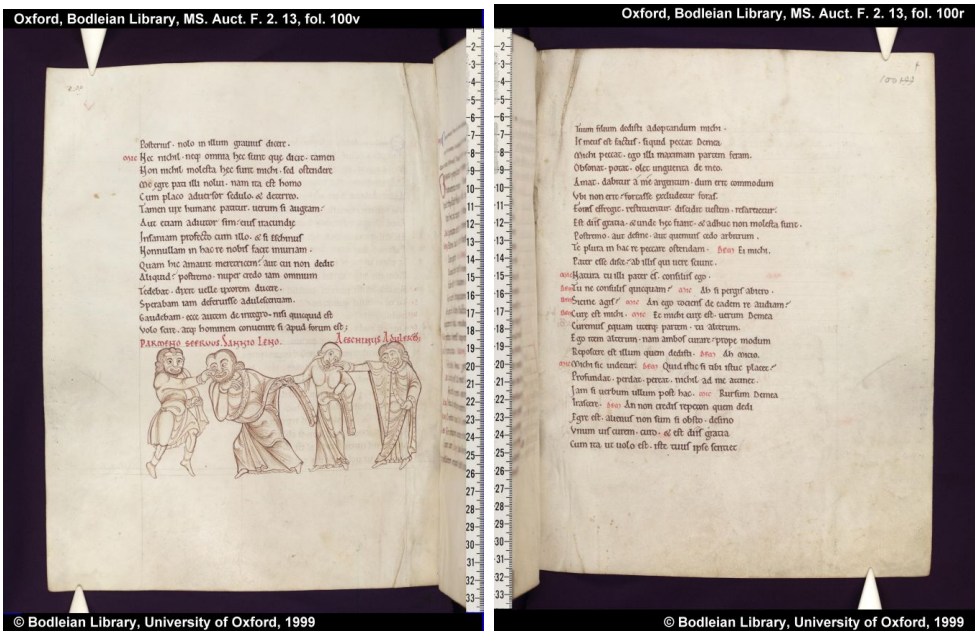
3.1 The Dataset

We selected the Terence manuscript for two reasons. First, its script is quite regular and thus seemed a natural starting point for moving beyond printed works. Second, Latin has not been extensively studied in natural language processing but large collections of ASCII works in Latin are available online to provide data to automatically learn language statistics.[1] In addition, the Latin language is heavily inflected compared to English. The large number of word forms mean that we must break words into smaller component pieces, or deal with very large dictionaries. We collected the images of these two manuscripts from the online repository of the Bodleian Library maintained by Oxford University. [Oxford, 2007]

We calculate statistics about the Latin language from this set of downloaded ASCII texts. In particular, we derive word-lists and unigram, bigram and trigram character statistics. We did not collect word n-gram statistics, because word choice in Latin is very flexible, and so we did not expect it to be very useful. This process involved little human intervention beyond designing the web spider.

Training data for the visual appearance of the script, on the other hand, is more difficult to provide, and so in this work we provided a minimal amount of supervised data. We manually segmented just a single example of each of the 22 most common glyphs from the Terence manuscript. These are basically the Latin lower case characters, although "u" and "'v," while distinct in ASCII were visually indistinguishable on the page. We also experimented with a second manuscript, a copy of the the Latin

[1] In this work, we used documents collected from www.thelatinlibrary.com



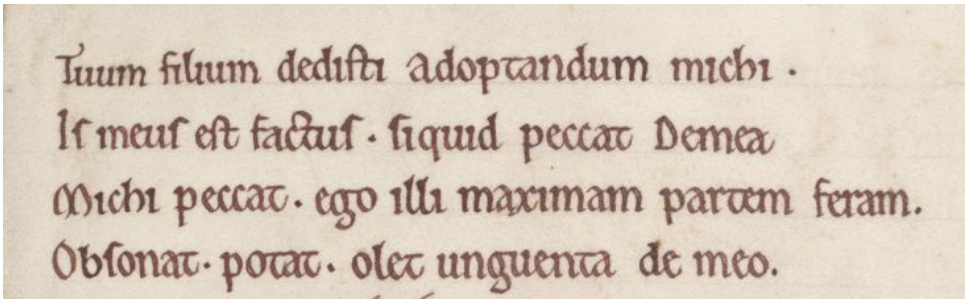


Figure 3.1: **Top**, *Two full pages from the main dataset of this chapter, a 12'th century manuscript of Terence's Comedies obtained from [Oxford, 2007].* **Bottom**, *The first four lines of the second page Note: (a) the richness of page layout; (b) the clear spacing of the lines; (c) the relatively regular handwriting.*

Gospels, also from the Bodleian library. For this manuscript, we provide no visual training data, but use the model trained from Terence to investigate its adaptability.

3.2 Introducing the gHMM

Twenty two visual exemplars is a very small amount of training data by the standards of the OCR and Handwriting Recognition communities. Generally OCR systems require hundreds of examples of each character. As noted in chapter 2, some researchers, [LeCun *et al.*, 1998; Vinciarelli, 2003; Kopec and Lomelin, 1995] have developed systems to train from transcriptions of whole lines, without providing individual character segmentations, but they still require a large number of these transcribed lines. As we will demonstrate, the generalized HMM is attractive because it can be initialized with small amounts of data, and yet still achieve quite strong results.

There are two additional aspects, in particular, that make the gHMM attractive. First, the parameters associated with language are distinct from those associated with visual appearance, allowing language models to be trained independently. Second, the gHMM explicitly models not only transcriptions but also different possible segmentations of the line into characters. This is necessary since in the manuscripts we consider, the individual glyphs are not generally well separated by whitespace, and by assumption, we will not have accurate character recognizers. Because of this, any segmentation based solely on visual cues will be quite inaccurate. The gHMM explicitly allows joint inference over transcriptions and alternate segmentations.

3.3 Preprocessing

Our input is the unprocessed scanned pages of a handwritten document. Figure 3.1 shows some examples from the Terence Manuscript, our main dataset for this chapter.

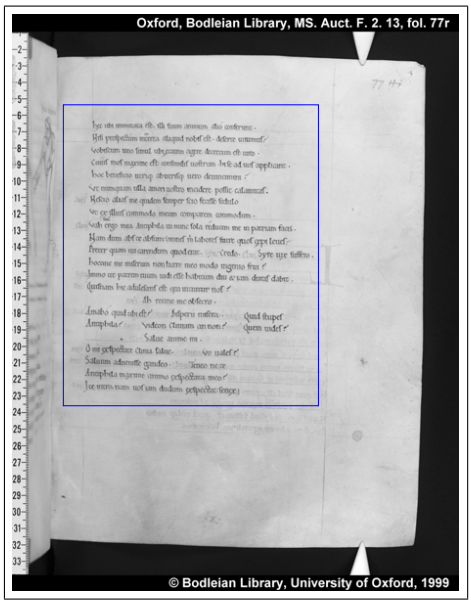
Figure 3.3 shows additional manuscripts from the Oxford Library collections. Our models assume that their input is a set of image windows and that the output for each image window will be a single sequence of characters. Our first goal is to extract a set of these sub windows from the document page that meet this criteria, and to order those windows so that we can recombine them into a full transcription once we have translated each one.

We have two methods for accomplishing this. The first makes the assumption that lines of text are largely uncurved, and identifies lines by analyzing the projection profile of a page. This is the method used to provide the training set used in this chapter from Terence and the Beast Headed Evangelist manuscripts. We also demonstrate a second method, based on identifying regions with "text-like" texture, that performs well on a wider array of document types.

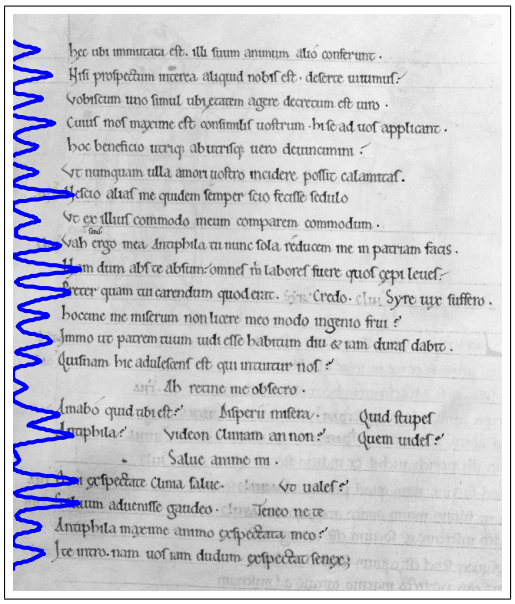
3.3.1 Identifying Lines by Projection Profile

In the projection profile method, we convert the color images to grayscale. The steps are illustrated in figure 3.2. We provide a manual bounding box to crop the image so that only the page remains. The pages of these manuscripts are scanned in a regular manner, so that a single bounding box sufficed.[2] For a simple page, what remains will be a set of roughly horizontal lines of text on an unmarked background Assume for the moment that the lines are perfectly horizontal. If we project onto the vertical axis by summing along each row of pixels, we are left with a highly peaked graph. If we project along any other direction, the resulting graph will be less peaked. Assuming the lines are basically linear, we can determine the correct orientation of the page by performing this projection at a variety of angles, and choosing the orientation

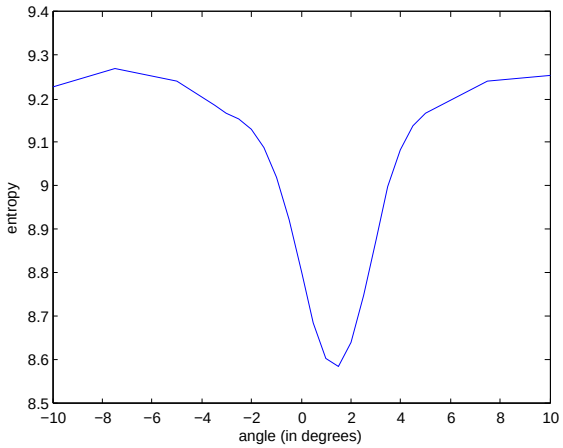
[2]The bounding box shown in figure 3.2 would normally include the whitespace of the page. We have reduced it for illustrative purposes.



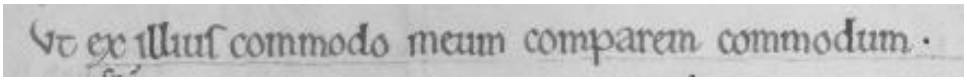
(a) Cropping Original Image



(b) Projecting onto Vertical Axis



(c) Entropy of Angled Projections



(d) Final Line Image

Figure 3.2: *We have two methods for extracting lines of text from scanned pages. In the* **projection profile** *method, shown here, we first crop the original image (a) to eliminate the non-textual border. We threshold the image, setting ink pixels to 1 and background pixels to 0. The plot in (b) is this thresholded image, summed along pixel rows, and smoothed with a Gaussian. Treating this plot as a distribution, we find its entropy for a range of rotations of the image (c). We extract lines (d) from the rotated image with minimal entropy.*

whose resulting graph has the lowest entropy.[3] Let $s_{\theta i} = \sum_j \text{image}_{ij}$ be the sum of foreground pixels along a row of pixels in a copy of the page image, rotated by θ. Let $p_{\theta i} = s_{\theta i} / \sum_i' s_{\theta i'}$ We estimate the rotation of the page as

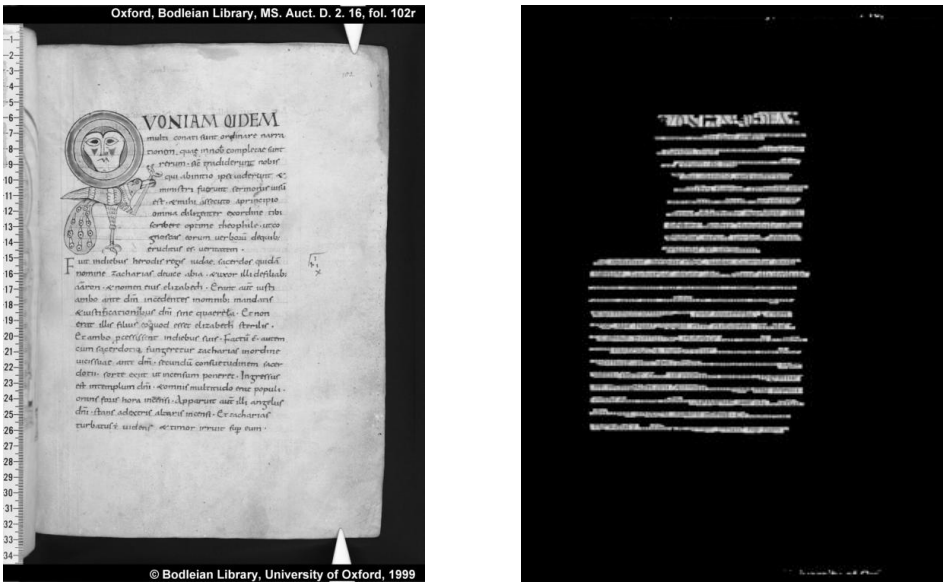
$$\text{Optimal Rotation} = \underset{\theta}{\text{argmin}} \sum_i -p_{\theta i} \log p_{\theta i} \tag{3.1}$$

3.3.2 Identifying Text Texture Regions

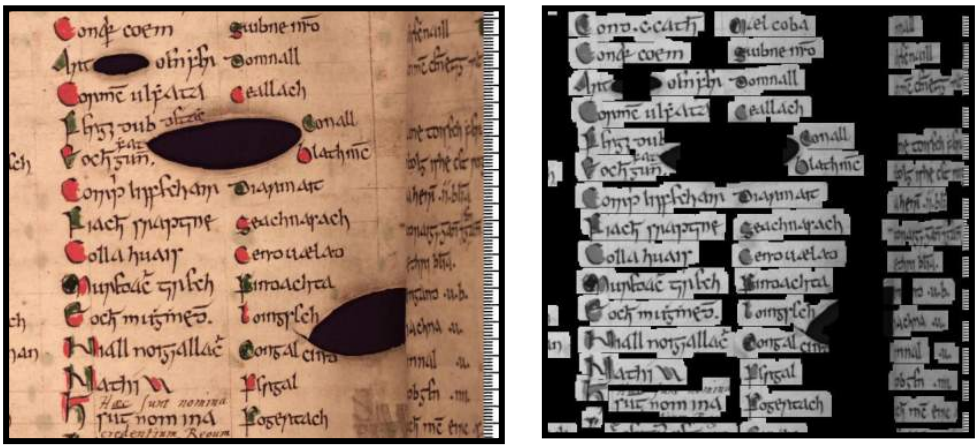
Most pages of the Terence manuscript were well segmented using the projection profile method. Unfortunately, many historical manuscripts violate the assumption that a page is composed only of unbroken and nearly linear strings of text. Documents frequently contain illustrations, curved lines, and various other artifacts such as dark streaks or holes. The method outlined above relies heavily on our ability to crop out or otherwise ignore these artifacts. A more robust method is to provide a local definition of a "text like" region, and to cut the document into text snippets that are close to locally linear. We define a text like region by its texture.

We designed a text texture operator. This operator assumes that text has two distinguishing features in scale space. First, at a low resolution, text appears as a horizontal bar. At a higher resolution, it has a significant number of vertical edges. Our text detector searches scale space for the maximal response to a horizontal bar detector, multiplied by the strongest response to a vertical edge detector at four times the scale. We then threshold these responses, blur them, and take the connected components. These become our text snippets. This method worked well on a variety of scripts in addition to the two used in this chapter. Further refinements might train the texture classifier automatically, but we have not experimented with this.

[3] see [Cover and Thomas, 1991]



(a) Beast Headed Evangelists



(b) Many documents are degraded, including holes in the page

Figure 3.3: **Extracting "text texture" regions:** *Most manuscript pages do not contain unbroken lines on an otherwise blank page. Many have embedded images or other artifacts, such as holes in the page or intra line annotations. Our second line finding method, section 3.3.2 handles such features, by identifying regions of text texture, and extracting locally linear text snippets.*

3.4 The Generalized HMM Model

We assume that each image provided by the line extractor has a transcription that is a single sequence of ASCII characters drawn from a known alphabet. Unfortunately, we do not know where in the image each character lies. If someone were to segment the image into N pieces such that each segment contained a single character, and then uniquely label each segment with a number from 1 to N we would have a one-to-one mapping between characters in the transcription and segments in the image. In this setting, an HMM is a fairly natural choice to model our data.

To use a Hidden Markov Model in this context, we assume that each character c_i in the transcription is drawn based on the previous k characters. Given c_i we generate the corresponding image segment s_i. Let g_i be the k-gram ending at c_i. Without yet specifying the details of these distributions, the probability of the image under such a model is

$$\mathbb{P}[\text{line}] = \mathbb{P}(g_1)\mathbb{P}(s_1|c_1)\prod_{i=2}^{N}\mathbb{P}(g_i|g_{i-1})\mathbb{P}(s_i|c_i) \tag{3.2}$$

The hidden Markov Model is attractive primarily for its computational efficiency. Because each state is independent of the rest of the sequence given its neighbors, we can use the forward-backward algorithm (section 2.3.2.1) to efficiently calculate the likelihood of different candidate transcriptions. It is also attractive because of its natural separation between language and visual parameters. The transition matrix $\mathbb{P}(g_i|g_{i-1})$ represents language knowledge in the form of frequencies over character n-grams, while our emission model $\mathbb{P}(s_i|c_i)$ will encode visual cues.

Unfortunately, the HMM model presented thus far requires that a segmentation be provided *a priori*. This may actually be reasonable for printed works without noise, and many early OCR systems did expect an independent character segmenter [Mori *et al.*, 1995]. However, this approach is fragile and fails on the handwritten historical manuscripts we consider. If the preprocessor fails, the system cannot re-

cover. It seems reasonable that if someone were to give us the transcription before we attempted to segment, then we could use this information to improve our segmentation guesses. What we would like is to delay making an initial hard decision on character boundaries, and instead try to infer segments and their labels jointly. To operationalize this idea, we now introduce an extension to our HMM model that allows us to include reasoning over segmentations.

Formally, our new method is derived from one frequently called a generalized or segment HMM [Murphy, 2002]. In a standard HMM, each hidden state g_i emits a single evidence variable s_i. In a generalized HMM, each hidden state is allowed to emit a variable length number of evidence variables $(s_{i1} \dots s_{iw_i})$, where w is drawn from some arbitrary distribution.[4] Additionally, the gHMM does not assume that the variables emitted by a single hidden state are independent. Mapping this to our specific problem, the s_i are individual columns of pixels from the image, and w_i represents the width of a character. Each character emits a variable width column of pixels. These columns partition the image pixels, and therefore represent a segmentation of the image.

Letting $x_i = \sum_{j=1}^{i-1} w_j$ denote the position of the leftmost column of pixels associated with character c_i, the probability of a line of text under the generalized HMM is

$$\mathbb{P}(\text{line}) = \mathbb{P}(g_1) \prod_{i>1} \mathbb{P}(g_i|g_{i-1}) \prod_{i \geq 1} \mathbb{P}(w_i|c_i)\mathbb{P}(s_{x_i} \dots s_{x_i+w_i-1}|x_i, w_i, c_i) \tag{3.3}$$

The final step is to expand $\mathbb{P}(s_{x_i} \dots s_{x_i+w_i-1}|x_i, w_i, c_i)$, i.e. the probability of a column of pixels given character label. Recall that we have provided a single example image of each character. Intuitively, the generative story for a column of pixels $s_{x_i} \dots s_{x_i+w_i-1}$ is simply that we choose a vertical position y_i, and place a copy of the

[4]The generalized HMM is equivalent to a standard HMM in the case where w is drawn from a geometric distribution.

corresponding example character into the image with its top left corner at (x_i, y_i).

To refine this intuitive story, we define a local coordinate system for the column with origin (x_i, y_i). At each pixel location, we assume that we have an associated feature vector f_{xy}. We discuss the details of these features in the next section. We model each f_{xy} as drawn from a multivariate normal distribution. The parameters of this normal distribution for a specific pixel in this column depend on the character label (c_i) and on the pixel's position relative to the local coordinate system in the following way:

1. We use each of our example character images to derive a corresponding *character template*. Figure 3.4. This template is a matrix of values the same size as the original character image, but whose "pixels" are the means and covariance matrices of a set of Gaussian distributions. Once again, we postpone the details of training these parameters until the next section.

2. Given c_i, we choose a character template. Given (x_i, y_i) we also know where to place the template in the image. This template then overlays a rectangle of pixels in the column. Each of these covered pixels is drawn from a normal distribution whose parameters are the corresponding entries in the template matrix. Any pixels not covered by the template are drawn from a single, shared normal distribution representing the background.

Letting t_i represent a character template, we expand the last term in equation 3.3 as

$$\mathbb{P}(s_{x_i} \dots s_{x_i+w_i-1} | x_i, w_i, c_i) = \sum_{y_i} \mathbb{P}(y_i|c_i)\mathbb{P}(t_i|c_i) \prod_{y'=1}^{Y} \prod_{x'=x_i}^{x_i+w_i-1} \mathbb{P}(f_{x'y'}|t_i, x_i, y_i) \quad (3.4)$$

3.4.1 Transition, Position and Emission Distributions

The gHMM model has five trainable distributions:

- The transition distribution, $\mathbb{P}(g_i|g_{i-1})$, is the probability of seeing a given ASCII character given k preceding characters. We generate character n-gram statistics by collecting a large number of ASCII Latin texts from [LatinLibrary, 2004]. This website includes a transcription of the Terence manuscript, but we exclude this document from the training set. We experiment with unigram, bigram and trigram models for this distribution, applying Kneser-Ney smoothing to the collected statistics.

- The width distribution, $\mathbb{P}(w_i|c_i)$, describes how many columns of pixels a character occupies. We restrict the values that w may take on. We assume each character has a mean width $\hat{w}_c$, and that the width distribution is only non-zero in the range $(\hat{w}_c - d_w \ldots \hat{w}_c + d_w$ for some small d_w. This is a fairly innocuous assumption as empirically the widths of a specific character do not vary greatly from instance to instance. The mean width for each character is initialized to the width of the corresponding example character image, and we assume a uniform distribution over values within the nonzero range.

- The vertical position, $\mathbb{P}(y_i|c_i)$ localizes a character template within a column. We assume a uniform distribution over possible y positions.

- In this chapter, the distribution over templates $\mathbb{P}(t_i|c_i)$ is a delta function. We assume a one-to-one correspondence between labels and character templates.

- The emission distribution, $\mathbb{P}(f_{x'y'}|t_i, x_i, y_i)$ defines the likelihood of pixel feature vectors given their position relative to a character template.

  - *Image Representation:* We associate a five dimensional vector with each pixel. The first element is the original gray-scale pixel value. For the other four, we convolve the image with two derivative of Gaussian filters oriented vertically and horizontally. We then separate the responses of

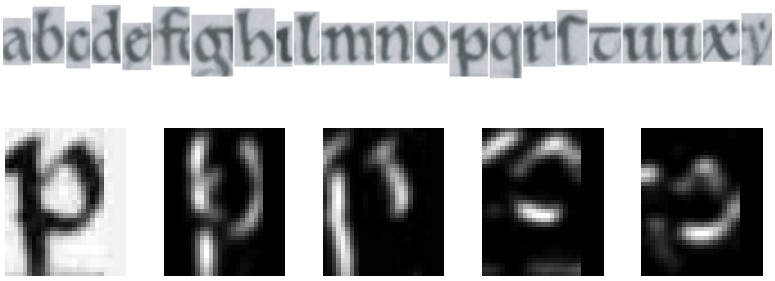


Figure 3.4: **Top**, the 22 instances, one per letter, used to train our emission model. These templates are extracted by hand from the Terence document. **Bottom**, the five image channels for a single letter.

these convolutions into positive and negative channels, invert the negative channels, and smooth each image. The resulting four response images are positive near edges in one of four axis aligned directions and zero elsewhere. Figure 3.4 shows examples of these 5 channels for our "p" exemplar.

– *Template Parameters:* We generate the 5-channel image described above for each of our example character images. We initialize the means of the corresponding character templates with the values from these feature images. We set the mean for the background distribution by providing one more example image of blank paper. We assume a diagonal covariance matrix for these features.

3.4.2 The Generative Process

Putting all the pieces together, we now have a full generative process for a line of text. At each sequence step, the hidden state of a a generalized HMM consists of a character label n-gram c, width w, horizontal and vertical position (x,y). In this model, the characters of c are drawn from the characters 'a'-'z', a space ' ', and a special end state Ω. (We do not treat capital letters separately.) Let T_c be the

template associated with character c, T_{ch}, T_{cw} be respectively the height and width of that template, and m be the height of the image.

Beginning at image column 1 (and assuming a dummy space before the first character),

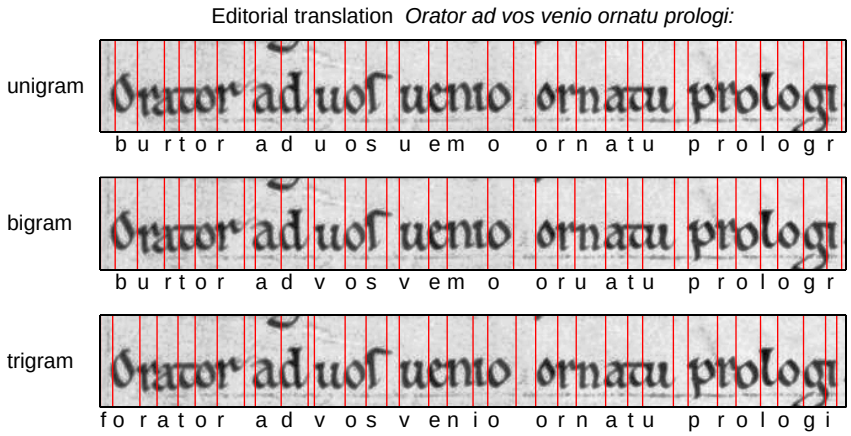
- choose a character $c \sim p(c|\text{previous n characters})$ (an n-gram label model)
- choose the corresponding template T_c This is deterministic given c
- choose a width $w \sim \text{Uniform}(T_{cw} - k, T_{cw} + k)$ (for some small k)
- choose a vertical position $y \sim \text{Uniform}(1, m - T_{ch})$
- *z,y and T_{ch} now define a bounding box b of pixels. Let i and j be indexed from the top left of that bounding box.*
  - draw pixel $(i, j) \sim \mathcal{N}(\mu(T_{cij}), \sigma(T_{cij}))$ for each pixel in b
  - draw all pixels outside of b from background Gaussian $\mathcal{N}(\mu_0, \sigma_0)$

    (See 3.4.1 for greater detail on pixel emission model)
- move to column $w + 1$ and repeat until we enter the end state Ω.

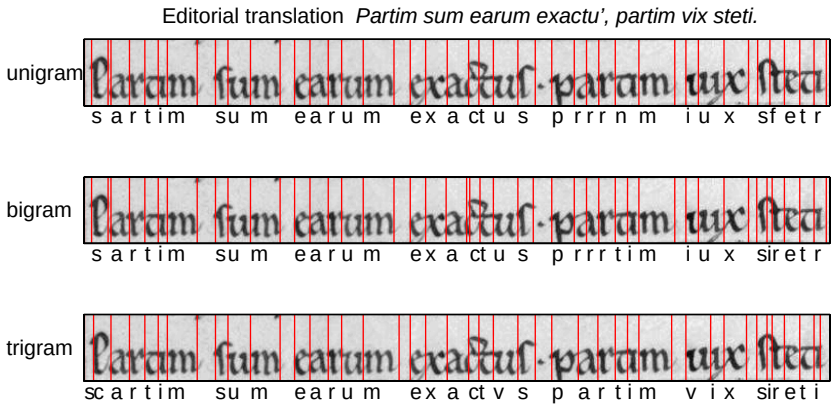
3.5 Results

3.5.1 Transcription

Transcription is not our primary task, but methods that produce good transcriptions are going to support good searches. The gHMM can produce a surprisingly good transcription, given the small number of exemplars used to train the emission model. We aligned an editors version of Terence with 25 pages from the manuscript by hand, and computed the edit distance between the transcribed text and the aligned text; as

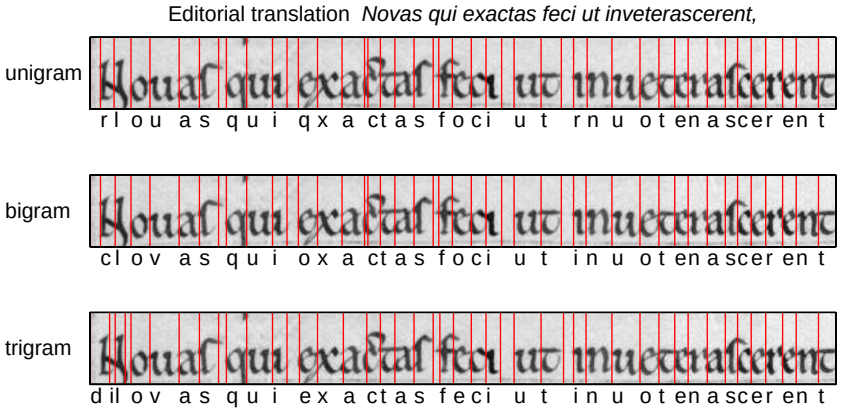


(a)

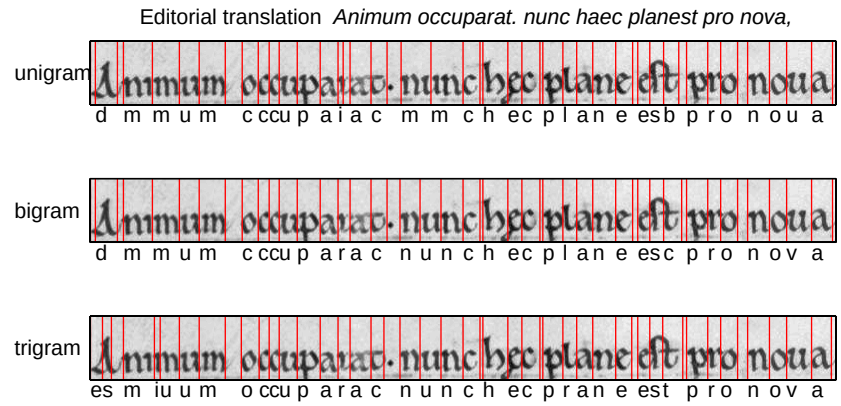


(b)

Figure 3.5: *We transcribe the text by finding the maximum likelihood path through the gHMM. The top line of text in each figure shows the standard version of the line (obtained by consensus among editors who have consulted various manuscripts; we obtained this information in electronic form from* `http://www.thelatinlibrary.com`*). Below, we show the line as segmented and transcribed by unigram, bigram and trigram models. In* **(a)** *the word "venio," for example is mistranscribed as "vemo" under the unigram and bigram models, but trigrams provide enough context to correctly transcribe this ambiguous piece of ink. This word also demonstrates the importance of transcribing and segmenting jointly. In* **(b)***, the trigram model also improves our transcriptions of "partim" and "vix", although all three models have trouble with the nested "t" in "steti."*



(a)



(b)

Figure 3.6: *Additional lines and their proposed transcriptions under unigram, bigram and trigram models.* **(a)** *The trigram model correctly transcribes 4 out of 6 words. Note that the model has no knowledge of the appearance of capital letters, and so the poor performance on the first character of each line is unsurprising.* **(b)** *Stronger language priors help, but are not a panacea. Note that the word "plane" is correctly transcribed by unigram and bigram models, but the model ignores the ink in favor of the prior under the trigram model and mistranscribes the word as "prane." On the other hand, more robust priors than trigrams would correct problems such as the final character of "occuparat". "Occuparac" is not a valid Latin word, but has acceptible trigram statistics.*

Model	matching chars	substitutions	insertions	deletions
Perfect transcription	21019	0	0	0
unigram	14603	5487	534	773
bigram	15572	4597	541	718
trigram	15788	4410	507	695

Table 3.1: Edit distance between our transcribed Terence and the editor's version. Note the trigram model produces significantly fewer letter errors than the unigram model, but that the error rate is still a substantial 25%.

table 3.1 indicates, approximately 75% of letters are read correctly. Note that we have no models of capital characters, and so the errors at these locations are unsurprising.

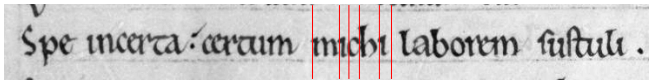
Figures 3.5 and 3.6 show transcription results for four lines from the Terence manuscript. Each line is transcribed using unigram, bigram and trigram character transition models. Our transcriptions significantly improve as we move from unigrams to bigrams. Stronger language models are clearly useful. Our transcriptions of almost all words are improved by their use. It is important to note, however, that strong language priors can occasionally mislead, by forcing the model to ignore evidence from the ink. In figure 3.5b, the word "plane" is correctly transcribed under unigram and bigram models, but changes incorrectly to "prane" under a trigram model, as "pra" is significantly more likely than "pla".

3.5.2 Searching

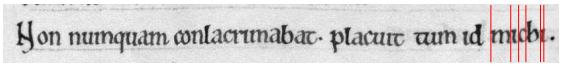
An attractive aspect of the generalized HMM is that it provides a true probabilistic interpretation. This in turn means that we can ask questions of the form, what is the probability that a given word exists in this image? and we can meaningfully rank the responses to such a question for each image in a large database. This allows us to perform search.

For search, we rank lines by the probability that each contains our search word.

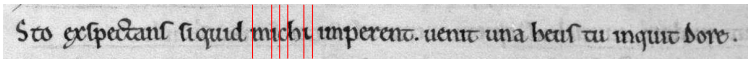
Spe incerta certum mihi laborem sustuli,



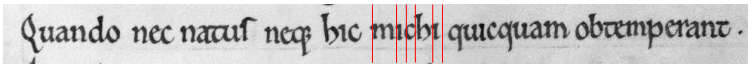
Nonnumquam conlacrumabat. placuit tum id mihi.



Sto exspectans siquid mi imperent. venit una, "heus tu" inquit "Dore,

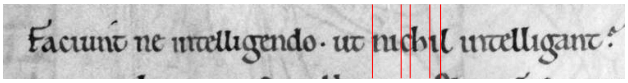


Quando nec gnatu' neque hic mi quicquam obtemperant,

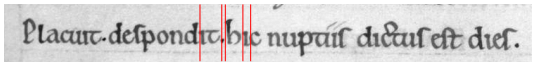


(a)

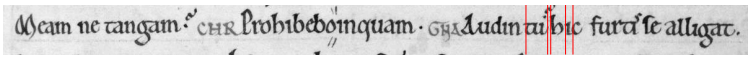
Faciuntne intellegendo ut nil intellegant?



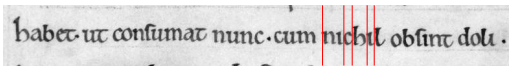
Placuit: despondi. hic nuptiis dictust dies.



Meam ne tangam? CH. Prohibebo inquam. GN. Audin tu? hic furti se adligat:



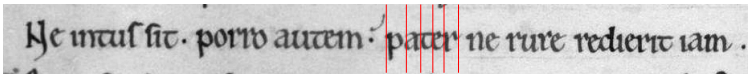
Habet, ut consumat nunc quom nil obsint doli;



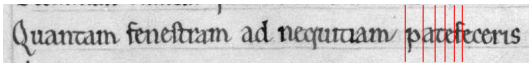
(b)

Figure 3.7: The handwritten text does not fully correspond to the transcribed version; for example, scribes commonly write "michi" for the standard "mihi". Our search process reflects the ink fairly faithfully, however. (a) the first four lines returned for a search on the string "michi"; (b) the first four lines returned for a search on the string "mihi", which does not appear in the document. Note that our search process can offer scholars access to the ink in a particular document, useful for studying variations in transcription, etc.

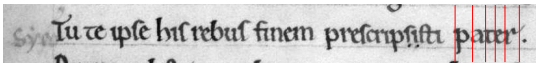
Ne intus sit; porro autem pater ne rure redierit iam.



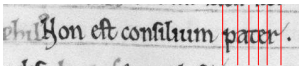
Quantam fenestram ad nequitiem patefeceris,



SI. "Tute ipse his rebu' finem praescripsisti, pater:

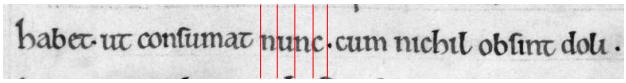


Iube illam redire. PA. Non est consilium, pater:

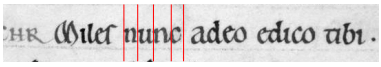


(a)

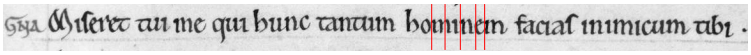
Habet, ut consumat nunc quom nil obsint doli;



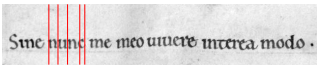
CH. Meam sororem. THR. Os durum. CH. Miles, nunc adeo edico tibi



GN. Miseret tui me qui hunc tantum hominem facias inimicum tibi.



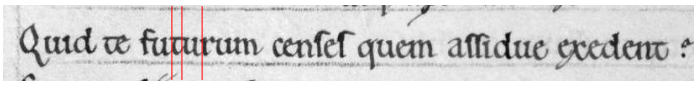
Sine nunc meo me vivere interea modo."



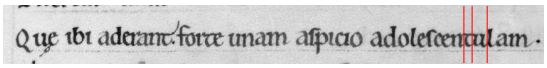
(b)

Figure 3.8: *Search Results for mid-length strings. The top four lines returned for the words* **(a)** *"pater" and* **(b)** *"nunc". Three of the top four are correctly identified and located in the line.*

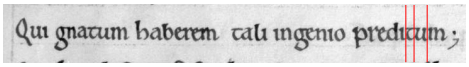
Quid te futurum censes quem adsidue exedent?



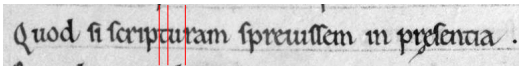
Quae ibi aderant forte unam aspicio adulescentulam



Qui gnatum haberem tali ingenio praeditum.

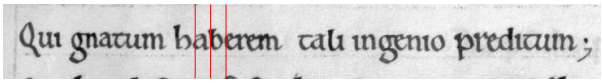


Quod si scripturam sprevissem in praesentia

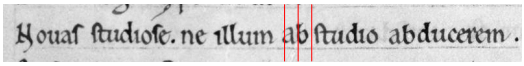


(a)

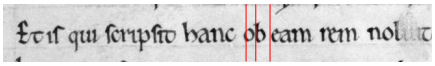
Qui gnatum haberem tali ingenio praeditum.



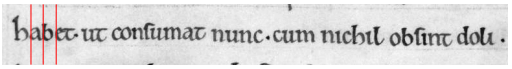
Novas, studiose ne illum ab studio abducerem.



Et is qui scripsit hanc ob eam rem noluit



Habet, ut consumat nunc quom nil obsint doli;



(b)

Figure 3.9: Our search method may be used to find substrings and not only whole words. In this figure we show search results for two small words, including substrings of longer words. **(a)** Top four lines returned for the search query "tu". **(b)** Results of a search for "ab."

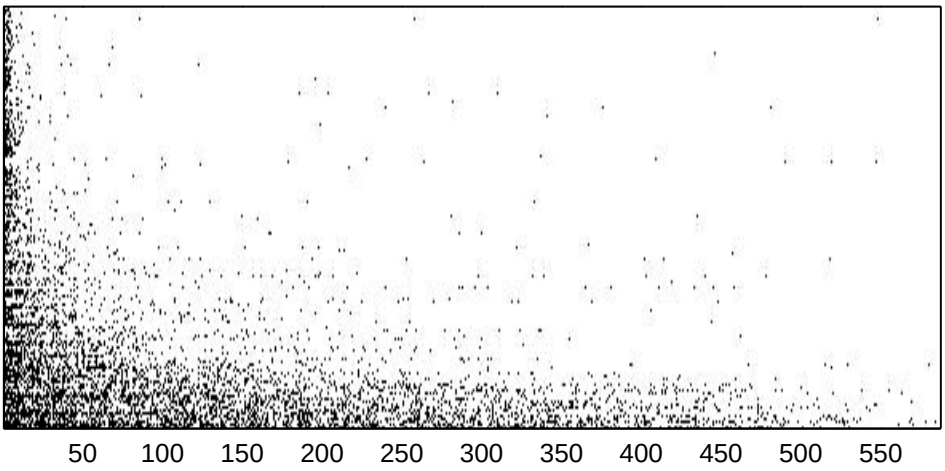


Figure 3.10: *Our search ranks 587 manuscript lines, with higher ranking lines more likely to contain the relevant term. This figure shows complete search results for each term that appears more than three times in the 587 lines. Each row represents the ranked search results for a term, and a black mark appears if the search term is actually in the line; a successful search will therefore appear as a row which is wholly dark to the left, and then wholly light. All 587 lines are represented. More common terms are represented by lower rows. More detailed results appear in figure 3.11 and figure 3.12; this summary figure suggests almost all searches are highly successful.*

Using Bayes rule,

$$\mathbb{P}[\text{word in line}|\text{image}] = \frac{\mathbb{P}[\text{image}|\text{word in line}]\mathbb{P}[\text{word in line}]}{\mathbb{P}[\text{image}]} \tag{3.5}$$

The denominator is simply the probability of the end state as calculated by the forward-backward algorithm. It must be calculated in order for the search score to be meaningful across different lines of text. The numerator is the probability of the image when we restrict inference to the subset of character sequences that contain our query word. This value can also be calculated using a second pass of the forward algorithm, constraining inference to paths containing the word.

Search results are strong. We show results for two documents. The first set of results refers to the edition of Terence's Comedies, from which we took the 22 letter

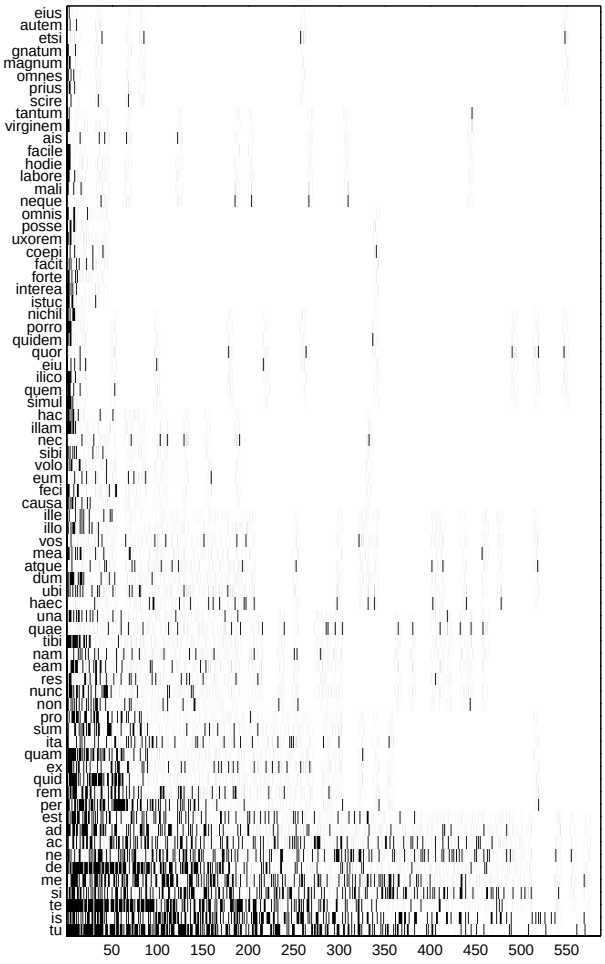


Figure 3.11: *Search results for selected words (indicated on the leftmost column). Each row represents the ranked search results for a term, and a black mark appears if the search term is actually in the line; a successful search will therefore appear as a row which is wholly dark to the left, and then wholly light. Note only the top 300 results are represented, and that lines containing the search term are almost always at or close to the top of the search results (black marks to the left).*

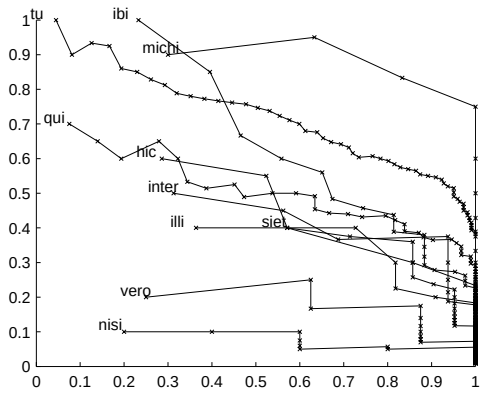


Figure 3.12: *Here we plot precision against recall for a set of different words by taking the top 10, 20, ... lines returned from the search, and checking them against the aligned manuscript. Note that, once all cases have been found, if the size of the pool is increased the precision will fall with 100% recall; many words work well, with most of the first 20 or so lines returned containing the search term.*

instances. In particular, for any given search term, our process ranks the complete set of lines. We used a hand alignment of the manuscript to determine which lines contained each term; figure 3.10 shows an overview of searches performed using every word that appears in the document more than three times, in particular, showing which of the ranked set of lines actually contained the search term. For almost every search, the term appears mainly in the lines with higher rank. Figure 3.11 contains more detailed information for a smaller set of words. We do not score the position of a word in a line (for practical reasons).

Figure 3.7 demonstrates (a) that our search respects the ink of the document and (b) that for the Terence document, word positions are accurately estimated. The spelling of medieval documents is typically cleaned up by editors; in our manuscript, the scribe reliably spells "michi" for the standard "mihi". A search on "michi" produces many instances; a search on "mihi" produces none, because the ink doesn't actually contain the editorial version of this word. Notice this phenomenon also in

the bottom right line of figure 3.7, the scribe writes "habet, ut consumat nunc cum nichil obsint doli" and the editor gives "habet, ut consumat nunc quom nil obsint doli." Figure 3.9 shows that searches on short strings produce many words *containing that string* as one would wish. Figure 3.13 shows two search results for an alternate manuscript, MS. Auct. D. 2. 16, a copy of the Latin Gospels from the Bodleian library. For this manuscript, we do not have an aligned text, so cannot measure recall and precision, but searches perform reasonably given the lack of any training data from this manuscript.

3.6 Search, Transcription and Language Models

Performance on transcription and search is clearly related. At one extreme, if a search engine had access to a perfect transcription, it would have no need to review the original document. Surprisingly, though, the results of this chapter demonstrate that we do not necessarily need to accurately transcribe in order to perform accurate searches. For Terence, our 75% character transcription accuracy rate means that we only rarely will transcribe an entire word without error. Our search results, on the other hand, are far stronger.

In part this is due to the fact that for search, we return a ranked list. If the best result is not first, it is still likely found in the top few ranked lines. For transcription, in contrast, we return only the best result. Still, this does not account for the largest boost in performance. We see examples where the top ranked search result is correct, but the transcription of that word image is wrong. Figure 3.14 shows one example. The word "animo" is incorrectly transcribed by the trigram model, and yet this line is the top ranked search result.

The answer here is that search implicitly uses a much stronger language prior than transcription. For transcription we must hypothesize the most likely string for

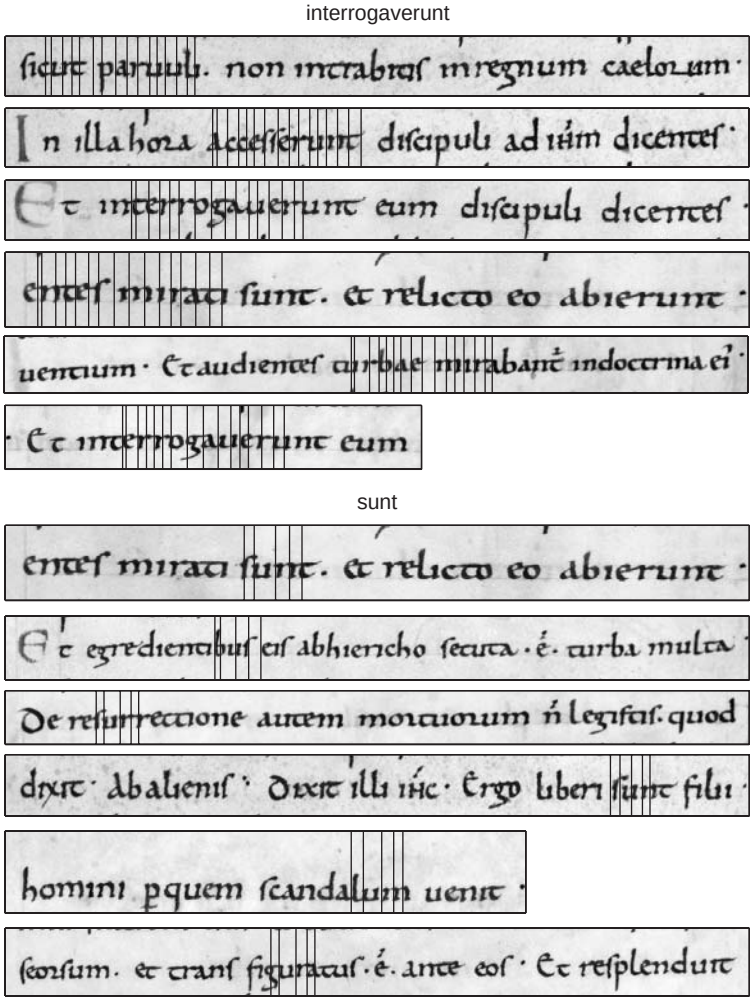
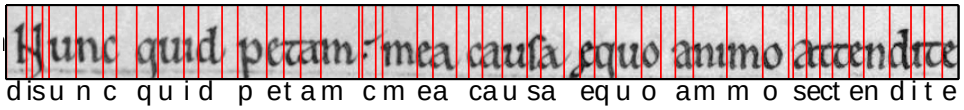
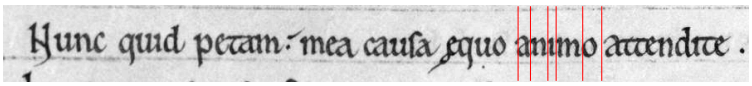


Figure 3.13: *The first six lines returned from the second manuscript, (MS. Auct. D. 2. 16, Latin Gospels with beast-headed evangelist portraits made at Landvennec, Brittany, late 9th or early 10th century, from [Oxford, 2007]), in response to the queries "interrogeraverunt" (*left*; lines three and six contain the word, which is localized largely correctly) and "sunt" (*right*; lines one and four contain the word). We do not have aligned text, so cannot measure the recall and precision for searches on this document. The recall and precision are clearly not as good as those for the Terence document, the search is reasonably satisfactory, given that* no training information from this document was available.

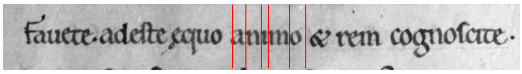


(a)

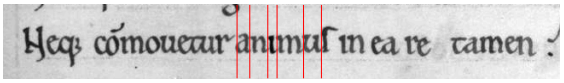
Nunc quid petam mea causa aequo animo attendite.



Favete, adeste aequo animo et rem cognoscite,



Neque commovetur animus in ea re tamen,



(b)

Figure 3.14: Searches for words often succeed even when the transcription fails. Figure **(a)** shows the maximum likelihood trigram transcription for a line from the Terence manuscript. It incorrectly transcribes "animo" as "ammo." In (b), this same line is still the best ranked result when searching the document for "animo".

an image. Since the model only has access to trigram statistics, this comparison includes many strings that are not valid Latin words.

In search, however, we are given a query string, such as "animo," and asked to hypothesize likely locations in the document. The important difference is that in this case we benefit from the fact that the author only writes valid Latin. Presumably, if the author had actually written the gibberish string "ammo," this would harm our search results for the term "animo," just as we saw with transcription. In search, we implicitly use that true distribution over Latin words in a way that we cannot when transcribing.

3.6.1 Improving Visual Classifiers

Clearly, language priors boost our performance. We see a significant boost in transcription performance when moving from unigram to trigram character models. However, trigram statistics still provide only a "weak" prior on acceptable Latin strings in that they still give high probability to many invalid transcriptions. Our search results are evidence that stronger language priors could improve performance still further. However, moving beyond trigrams presents two problems. First, it introduces a substantial computational burden into gHMM inference. Second, we run into sparsity problems in our training data. We do not have enough ASCII texts to derive good estimates of the frequency of longer strings. Finally, even if we had perfect knowledge of the distribution over Latin words, too much reliance on a language prior can force the model in some cases to ignore useful evidence from the ink, as we saw in figure 3.6 with the word "plane." Detrimental effects such as this are most likely for rare words, For search purposes, it is exactly these rare words that a user is often most interested in retrieving.

In some sense, however, we have taken our reasoning about language models to

an extreme. Our model would not need rely so heavily on language cues if our visual character recognizers were more discriminative. Our initial character emission model is weak, because it is trained on just one example of each character. It seems reasonable that if we had more examples of each character that we could train a more discriminative classifier. This, in turn, would lessen our reliance on a language prior, and improve our performance on all words, including uncommon ones.

We propose an iterative process, where we use the existing gHMM to propose segmentations and labels for lines of text, and then use those labeled segments as additional training data to improve the visual character classifiers. It turns out that the vanilla gHMM is not well suited to such a process. In the next chapter, however, we will examine the problems with the gHMM, and present an extension to the model that will allow us to effectively retrain the visual classifiers.

Chapter 4

Improving Visual Appearance Models

In the preceding chapter we demonstrated that language models can be used in a handwriting recognition system to greatly reduce the need for accurate visual character recognizers. We introduced the generalized HMM, and used this model to accurately search a medieval manuscript from the Bodleian collection.

Our reliance on language models came at a price, however. The stronger the language model, the less well the system recognizes out of vocabulary or uncommon words. This is less a problem for search than for transcription, since for search the user provides the query term and thereby removes the problem of out of vocabulary words. Still, in both cases, while the degree to which we can rely on language alone is impressive, it stands to reason that better character recognizers could only improve performance.

The imbalance between language and visual cues is a result of our decision to restrict the amount of document specific supervised data we would use to initialize the gHMM. For the language parameters, we collected a large set of ASCII documents and could therefore compile quite accurate n-gram character statistics and extensive wordlists. For the document specific script model, on the other hand, we provided

just one example each of 22 lower case characters. It is reasonable to expect that we could build more accurate models of character appearance if we were given a larger collection of exemplars with which to train them, but once again we would like to avoid the manual labor of cutting these additional examples out from the document by hand.

In this chapter, we present a method for automatically extracting additional exemplars from a document. We train a somewhat modified version of the gHMM with the same 22 exemplars as previously used. Once trained, the model provides candidate segmentations and labels for the document. Recall that our search results were more reliable than maximum likelihood transcriptions. To exploit this, we provide a generic dictionary of words in the target language. We restrict the portions of the document from which we extract examples by identifying "high confidence" regions. These are image regions for which exactly one word from our dictionary scores highly under our model. Given a set of high confidence regions, we effectively have a training corpus of text images with associated transcriptions. In these regions, we infer a segmentation and extract new character examples. Finally, we use these new exemplars to learn an improved character prediction model. As in chapter 3, our document in this work is a 12th century manuscript of Terence's Comedies obtained from Oxford's Bodleian library [Bodleian, 1100].

4.1 Extending the Segmentation Model

In the gHMM, visual recognition of a character is a local operation. Given a segmentation and the ASCII labels of surrounding characters, the probability of a character in a segment is independent of the appearance of the rest of the document. In a local model such as this, the natural training data is a set of image segments each containing an instance of a labeled character. Given a large number of these segments,

researchers have proposed a wide variety of methods for training better single character recognizers. See the discussion in section 2.1. The less variance in the appearance of a glyph, the fewer exemplars these methods need to train an accurate recognizer. We have no control over some sources of variance such as noise and pen strokes. We do, however, have control over segmentation of the line image into windows containing glyphs. The recognition task will be easier if we can train the system with a set of glyph examples that are well aligned with respect to each other.

In the previous chapter, we primarily focused on the influence of language models. We briefly discussed the importance of modeling segmentations and labels jointly, but did not investigate this thread of reasoning in great detail. However, with our new emphasis on accurate segmentation, we must reevaluate the gHMM's performance. Unfortunately, we discover that the gHMM frequently localizes characters poorly, even when it returns the correct transcription.

4.1.1 the gHMM, Segmentation Review

In the simplest application of an HMM to modeling handwriting, we might assume that every character has a fixed width. In this case, a hidden state represents a character and the evidence variable is some feature vector calculated for fixed blocks along the line. Recalling our notation from chapter 3, let c_i be the i^{th} character in a sequence, and g_i, the n-gram ending at c_i.

The probability of a line under this model is given as

$$p(line) = p(g_1|\alpha) \prod_{t>1} p(g_t|g_{t-1}) p(y_t|c_t) p(im_{[w*(t-1):w*t]}|c_t) \tag{4.1}$$

α is the special start state, w is the width of a character, $im_{[w(t-1)+1:wt]}$ the column of pixels beginning at column $w*(t-1)+1$ of the image and ending at column $w*t$, (i.e. the set of pixels spanned by c), and y_t is the vertical position of the character

inside this column.

As we have seen, character's widths do vary quite substantially and so in chapter 3 we introduced the gHMM to accommodate different possible segmentations. In this model a hidden state is allowed to emit a variable length series of evidence variables. We introduce an explicit distribution over the possible widths of a character. Letting x_t refer to the horizontal position of the left edge of a character on the line, the probability of a line under this revised model is

$$p(line) = p(c_1|\alpha) \prod_{t>1} p(c_t|c_{t-1}) p(w_t|c_t) p(y_t|c_t) p(im_{[x_t+1:x_t+w_t]}|w_t, c_t, x_t, y_t) \qquad (4.2)$$

The last term in the above equation gives the probability of a column of pixels. Within each column, a template covers a rectangular "region of influence," in which it defines the pixel emission parameters. Outside this region, pixels are drawn from a background distribution. Since columns are disjoint, each pixel has a single parent in the generative model.

The gHMM's assumption of non-interacting, rectangular bounding boxes is computationally attractive, but in practice quite inaccurate. Most commonly, this is a result of ligatures, where a small character is deliberately nestled inside the bounding box of a larger preceding character, as we see repeatedly in figure 4.1.

Figure 4.1 demonstrates the problems with the segmentations allowed by the gHMM. The restriction to non-overlapping rectangular bounding boxes means that in many cases, in order to accurately place one character, we must inaccurately place another. In fact, these effects can be quite far reaching. There are many incorrect segmentations that join half of two adjacent characters. This can potentially disrupt the segmentation of a line for large distances.

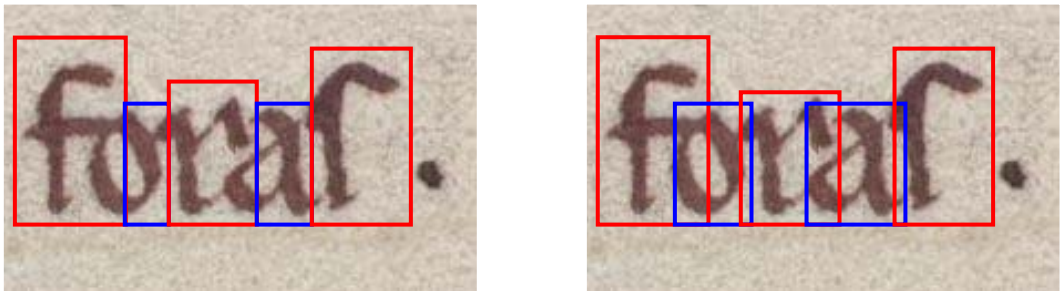


Figure 4.1: *Under the original gHMM, it is rarely possible to accurately localize all characters. This model was formulated so that the bounding boxes of adjacent characters abutted, but could not overlap. As the* **(left)** *image demonstrates, it is impossible to accurately segment all characters and meet this requirement. The goal in this chapter is to extract new exemplars with which to train character models. In this context, accurate localization is extremely important. By allowing adjacent characters to overlap* **(right)**, *we can achieve accurate localization while maintaining only pairwise dependencies between characters.*

4.1.2 Effect of segmentation on gHMM search results

Why didn't these segmentation problems have a larger adverse effect on our search results in the previous chapter? Recall that in search, we did not score segmentations. Our reported results were based on matching to a reference transcription, regardless of whether the component characters were accurately segmented or localized. Referring back to our discussion of section 3.6, we note that the edit distance between words is generally quite large, which in turn means that if we can correctly identify just a few characters in a word, and we know its approximate length, we may very well still be able to identify the word unambiguously. We hypothesize that search scores for a specific word under the gHMM frequently depend on correctly localizing only a few key characters in each word. This in turn creates inaccurate segmentations of other characters, but the large margins between scores for real words in a language prevent these inaccuracies from affecting the final transcription.

In the previous chapter, we treated inaccurate segmentations as additional noise, and this noise did not seem to have a large adverse effect on the final transcriptions. In this chapter, however, our goal is to build a library of glyph examples in order to retrain a more discriminative visual classifier. In order to accomplish this, we must accurately localize examples of each glyph, and thus we must reevaluate our segmentation model.

4.2 The Overlap gHMM

We would like to broaden the family of segmentations we consider, but in a manner that preserves the computational attractiveness of the original gHMM as much as possible. One possible approach is to allow arbitrarily shaped template boundaries. This approach is computationally unattractive, however, because it greatly expands the space of segmentations we need consider. An alternative method is to preserve rectangular boundaries, but allow templates to overlap. We adopt this second strategy, calling our modified model the *Overlap gHMM*

For computational reasons, it is imperative to limit templates' interactions and the size of their regions of influence. Without any restrictions, we approach the limiting case where every character on a page may influence the appearance of every pixel in the image. Clearly, inference in such a model is intractable. In the following model, we move one step beyond the gHMM's assumption of complete visual independence between characters, and allow pairwise interactions between templates. By pairwise interaction, we mean that at most two templates are allowed to cover any specific pixel.

This assumption is not completely accurate. For example, in documents with justified margins, space must be adjusted globally in order to line up both ends of each line. In modern fonts, this is generally accomplished by adjusting the interword

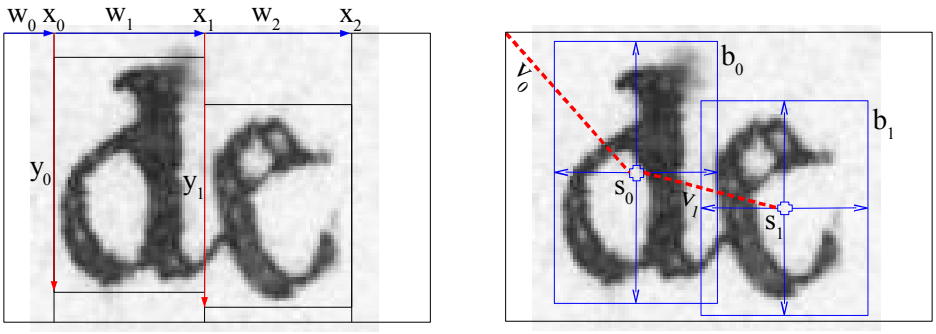


Figure 4.2: In the overlap gHMM, character templates may overlap. However, in the parameterization of space used by the original gHMM, there is no way to uniquely define the locations of overlapping templates. In that model (left), the location bounding box of a template is variable, and defined by an (x, y) position and width w. An alternate, more flexible representation (right), explicitly defines a template's "region of influence" with respect to an origin s and fixed size bounding box b. The origins of adjacent glyphs are related by an offset vector v.

spaces. In handwritten words, however, the effect is sometimes also accomplished by stretching the characters themselves. Other, more local examples include decorative ascenders or descenders which may extend past the immediately adjacent characters. However, these are pathological and infrequent cases. In general, allowing pairwise interactions covers most of the features we see in real world documents.

4.2.1 New parameterization of space

To define the Overlap gHMM, we introduce a somewhat different parameterization of character templates and their relative spatial relationships than presented in chapter 3. Recall that in the gHMM, a character template is a function mapping from $\mathbb{R}^2$ to the parameters of a normal distribution. The domain of this function is specified relative to a coordinate system local to the template. Pixel locations, on the other hand, are given relative to a global frame attached to the image. Their distributions are fully specified by the sequence of variables $((T_1, w_1, y_1), (T_2, w_2, y_2), \ldots, (T_N, w_N, y_N))$,

where T_i,w_i and y_i are the i^{th} template, width and vertical position respectively. Implicitly, these variables define an offset vector mapping from the global coordinate frame to that of each individual template, and that template's rectangular region of influence.

Once we allow templates to overlap, however, the sequence of widths no longer unambiguously defines the horizontal positions of the templates, nor their ROIs. To correct this, we replace w and y with two new variables v, an offset vector, and b, an explicit encoding of the template's bounding box. It proves convenient to use *relative* offset vectors. We define the sequence of v's as

- v_1: the position of the first template origin relative to the top left pixel of the image.
- v_i: the position of the i^{th} template relative to the origin of the $(i-1)^{th}$ template.

A template's absolute position in image coordinates is given by $v_i^* = \sum_{j=1}^{i} v_j$

The b variables are four element vectors $(b_{-x}, b_{+x}, b_{-y}, b_{+y})$. They define the rectangular region of influence for a template as the set of pixels $P \subseteq$ image such that

$$\text{region of influence} = \{p : b_{-x} \leq p_x \leq b_{+x}, b_{-y} \leq p_y \leq b_{+y}\} \tag{4.3}$$

Without loss of generality, we assume that the origin of the template is contained within this region of influence.

Clearly, this new parameterization (T, v, b) can be transformed back to our original variables (T, w, y). Moreover, we can enforce non-overlap of adjacent templates by restricting the allowed displacement vectors between pairs of templates. This alternate encoding can therefore represent the gHMM, but it is in fact more flexible. In particular, by changing the restrictions we place on displacement vectors, we can use this encoding to allow two templates, and no more than two, to overlap.

4.2.2 Compositing Overlapping Templates

In the original gHMM, a pixel feature was modeled as a draw from a Gaussian distribution, and these parameters were provided by the corresponding "pixel" of a character template, or by a background distribution if no template covered that pixel. The Overlap gHMM allows a third possibility: that two templates overlay the same pixel, and we must extend our emission model to accommodate this alternative.

- **No template covers pixel:** As in the gHMM, the pixel is drawn from a Gaussian with mean and variance determined by a single background distribution
- **One template covers pixel:** As in the gHMM, the pixel distribution is a normal distribution with parameters taken from the corresponding location in the character template.
- **Two templates cover pixel:** With overlapping templates, a pixel is drawn from a normal distribution whose mean and variance are the sum of those from the two corresponding template pixels.

4.2.3 Pixel Representation

Our method for compositing two templates is applicable because of the specific image representation we use in this work. In particular, we perform a soft binarization of the image. We fit a Gaussian mixture model to the pixels of a randomly selected collection of text images, positing two clusters: foreground (ink) and background (plain paper) pixels. We preprocess each text image by replacing each pixel in the raw image with the probability of its being foreground under the mixture model. In this feature image, then, pixels range from 0 to 1, with ink pixels close to 1. Our initial template glyphs are the 22 handcut lower case exemplars, after being passed

through this preprocessing step. Each pixel in the template represents the mean of a Gaussian describing the value we expect that pixel to take on.

With the above representation for a pixel, summing the means of two or more templates effectively serves as a logical "or", and allows us to accurately model effects such as an "i" nesting under an "f" in the bigram "fi". For locations where an ink stroke is shared by two characters, this is clearly not an accurate representation. When two ink strokes overlap, they may get slightly darker, but certainly not twice as dark. However, we also know that while characters do overlap slightly, significant amounts of overlap is to be avoided. Fortuitously, then, this bias away from the true representation of a pixel actually aids our recognition engine.

4.2.4 The Generative Process

We assume a set of labels L. Each label is represented electronically by an ASCII character. We associate a set of templates T_l with each label $l \in L$. We initialize these sets as in the gHMM by providing a single handcut example for each label. Each template $t \in T_l$ is a function mapping from $\mathcal{R}^2$ to the parameters of a normal distribution. Each template also has an associated bounding box b defining its region of influence. b is fully specified by the four element vector $(b_{-x}, b_{+x}, b_{-y}, b_{+y})$, defining the offsets from the origin of the template to the edges of the bounding box. The bounding boxes of our initial templates are derived from the widths and heights of the hand cut exemplars.

To ensure that no more than two templates overlap, we constrain the displacement vectors such that the bounding box of each template in a sequence lies entirely to the right of the preceding template's origin. For computational reasons, we also bound the maximum amount of whitespace allowed between two adjacent templates. We set this max-width parameter manually, and assume it is template independent. Let

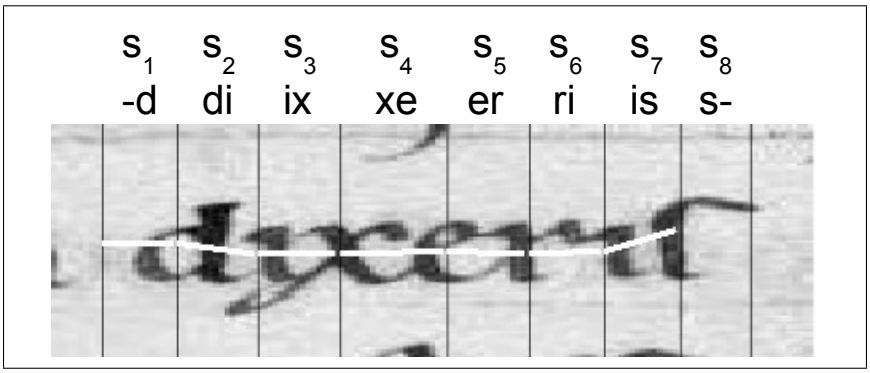


Figure 4.3: **A line, and the states that generate it.** *Each state s_t is defined by its left and right characters c_{tl} and c_{tr} (eg "x" and "e" for s_4). In the image, a state spans half of each of these two characters, starting just past the center of the left character and extending to the center of the right character, i.e. the right half of the "x" and the left half of the "e" in s_4. The relative positions of the two characters is given by a displacement vector v_t (superimposed on the image as white lines). Associating states with intercharacter spaces instead of with individual characters allows for the bounding boxes of characters to overlap while maintaining the independence properties of the Markov chain.*

t_{i-1} and t_i be two adjacent templates in a sequence, with corresponding bounding boxes b_{i-1} and b_i. Let $\sigma_{\max}$ be the maximum amount of whitespace allowed between templates, The constraints on v_i can be given in terms of b and σ as

$$\max(b_{(i-1,+x)}, b_{(i,-x)}) \leq v_{ix} \leq b_{(i-1,+x)} + b_{(i,-x)} + \sigma_{\max} \tag{4.4}$$

4.2.5 Overlap gHMM Generative Process

The generative process then, is

1. **Choose a label l_i.** The label is chosen from a multinomial distribution conditioned on the previous k labels. In this chapter we use character bigrams, so $k = 1$. As in chapter 3, this distribution is trained from a separate set of ASCII documents collected from the web.

2. **Choose a template t_i, and region of influence b_i** Given l, t is chosen uniformly at random from the set T_l. Each template deterministically defines a bounding box b_i

3. **Choose a displacement vector v_i** This vector is chosen such that no more than two templates overlap at any pixel.

   (a) The **horizontal component** v_{ix} is chosen uniformly at random from the range specified in equation 4.4

   (b) For the first template, the **vertical component** v_{iy} is chosen u.a.r. from the range $[1, \text{height of image}]$. For subsequent templates, we assume the mean vertical displacement is 0. v_{iy} is chosen u.a.r. from the range $[v_{i-1,y} - k, \ldots, v_{i-1,y} + k]$ for some small k.

4. Given T_{i-1}, T_i and v_i, we have fully specified the emission distributions for the column of pixels lying between the origins of these two templates. Each pixel may be covered by 0,1 or 2 templates.

   (a) The pixel is uncovered, and drawn from a background distribution

   $$p_{xy} \sim \mathcal{N}[\mu(\text{bg}), \sigma(\text{bg})^2]$$

   (b) Pixels covered only by template T_i are drawn from the distribution

   $$p_{xy} \sim \mathcal{N}[(\mu, \sigma^2) = T_i(p_x - x_i, p_y - y_i)]$$

   and those covered only by T_{i-1} from an equivalent distribution

   $$p_{xy} \sim \mathcal{N}[(\mu, \sigma^2) = T_{i-1}(p_x - x_{i-1}, p_y - y_{i-1})]$$

(c) Pixels covered by both templates are drawn from

$$p_{xy} \sim \mathcal{N}[(\mu, \sigma^2) = T_i(p_x - x_i, p_y - y_i) + T_{i-1}(p_x - x_{i-1}, p_y - y_{i-1})]$$

4.3 Efficiency Considerations

The state space of this model is not intrinsically larger than that of the gHMM. However, allowing symbols to overlap has increased the dependencies between adjacent steps in the chain. In chapter 3, adjacent characters were linked only through the language parameters. In this model, visual appearance is linked as well. The model remains a generalized HMM, and in principle inference can therefore be treated tractably as an instance of dynamic programming with the forward-backward algorithm, but the added visual dependencies eliminate our ability to use a number of optimizations that allowed for efficient inference in the vanilla gHMM.

4.3.1 Efficient Inference in the Overlap gHMM

The lattice graph, introduced in section 2.3.2, is a useful representation to describe inference algorithms for the gHMM and Overlap gHMM. Each vertex represents the possible position of a glyph template on the image. Edges are the allowed displacement vectors between adjacent glyphs. In the gHMM, a vertex is indexed by (x, y, w, c), the (x, y) position of the bottom left template pixel, its width w, and its character label c. We need w because the bounding box for a template is only implicitly defined. In the Overlap gHMM, the bounding box is explicitly given for each glyph template, so the (x, y) coordinate of the origin fully specifies a template's location. Unlike in chapter 3, however, we allow multiple glyphs g per character label c. A vertex in the Overlap gHMM is indexed by (x, y, g, c).

Inference in either model is as described in section 2.3.2. Each path through

the lattice represents a possible transcription and sequence of glyph locations. We can use the Viterbi algorithm to find the maximum likelihood path, and the related forward-backward algorithm to generate posterior probabilities. The computational complexity of these inference algorithms is directly related to the number of vertices and edges in the lattice graph. In the gHMM, we calculate a different likelihood for a column of pixels for each vertex in this graph. Let (m, n) be the size of an image in pixels, R_L be a bound on the computational cost of calculating this likelihood at a specific location, and G be the number of different glyphs in our model. The cost of calculating likelihoods for every position is $O(mnR_LG)$. When we allow character overlaps, pixel likelihoods now depend on the position of *two* adjacent glyphs, not just one. This makes the number of distinct values we must calculate depend not only on the identities of two glyphs instead of one, but also on the displacements between them. We can provide some bound on the allowable displacement vectors. Let D_Y be a bound on the number of different vertical displacements, and D_X a bound on horizontal ones. Still, we have drastically increased the number of distinct values we must calculate from $O(mnR_LG)$ to $O(mnR_LG^2D_YD_X)$

For inference in the gHMM, we can also reduce the size of the lattice graph without losing the ability to perform exact inference. The reason for this is that in the previous model, the vertical position of each glyph template, y_i, is chosen independently. In particular, it is independent of the y position of other characters. We can therefore sum over this variable before traversing the graph, and our vertex set collapses from (x, y, w, c) to (x, w, c). The score at a vertex becomes

$$\mathcal{L}(v(x, w, c)) = \sum_{y=1}^{m} \mathcal{L}(\text{image}|x, y, w) + \mathcal{L}(y) \tag{4.5}$$

In the overlap gHMM, because pixel likelihoods depend on the displacement vector, we can no longer reduce the lattice graph using this technique.

4.3.2 Pruning the State space

For each model, there is a computational cost C_e associated with calculating the score for an edge, and a cost C_v for calculating the score of a vertex. The total computational complexity of inference on a lattice graph is $O(EC_e + VC_v)$. In both these models, the number of edges grows roughly quadratically with the number of vertices. As the previous section explains, we have a far larger number of vertices in the overlap gHMM, because we cannot sum over y positions. The costs are also dominated by calculating pixel likelihoods. In the gHMM, this cost is associated with a vertex. In the overlap gHMM, it is an edge cost, because it depends on the relative positions of two templates. Because of these two differences, exact inference in the overlap gHMM, although still polynomial, is no longer practically possible. In this section, therefore, we introduce a series of approximations to prune the lattice by removing very unlikely vertices and edges before traversing the graph.

4.3.2.1 Pruning Candidate Locations

Restricting candidate displacement vectors is not sufficient, and so we also restrict the locations we consider. The true likelihood of the pixels covered by glyph s at position (x, y) depends on the position and template of both adjacent glyphs. However, we can approximate this likelihood with an independent likelihood identical to that used in the original gHMM. We prune our set of candidate (glyph,location) pairs by only considering sites where this independent value is sufficiently high and locally optimal. We only calculate the true bigram glyph likelihoods for pairs of locations where both unigram likelihoods are sufficiently high. In particular, our threshold for a given location is that $\mathcal{L}(block) < .7\mathcal{L}(background)$ (where $\mathcal{L}(x)$ represents the negative log likelihood of x). In other words, a location has to look at least marginally more like a given block than it looks like the background.

This heuristic pruning step will only effect our accuracy if it eliminates true character locations. In practice, this happens very rarely. Our hypothesis is that while overlap is necessary for accurate localization, it does not normally effect the pixel likelihood very significantly, because the number of foreground pixels participating in the overlap is generally small relative to the total number of pixels in the symbol. This reasoning is questionable for a few cases such as the character "f". In this case, following characters tend to nest under the right crossbar of the "f", significantly overlapping its bounding box. However, in practice, the true "f" locations still ended up as local maxima, and our pruning threshold was loose enough that even in this case we did not appear to have a problem.

With these pruning steps, we have a much reduced set of candidate locations for characters. While the forward-backward algorithm is generally presented in matrix form, it can also be thought of as a graph traversal. The corresponding graph has a vertex for each (symbol,location,timestep) tuple, and edges connecting any two vertices with a valid displacement vector. In the vanilla HMM, the vertices corresponding to adjacent timesteps are fully connected. The pruning steps outlined above make the graph very sparse. The forward-backward algorithm runs in time $O(E)$ where E is the number of edges in this sparse graph.

4.3.3 Search Efficiency

So far we have addressed efficiency problems introduced because of our decision to allow character overlaps. However, the original gHMM itself had a serious efficiency problem with regards to its search technique. The search score for each query word was calculated by a separate application of the forward backward algorithm. In this work, our approach depends on efficiently searching a large dictionary. To accomplish this, we introduce a second set of approximations to enable efficient search.

Recall that the search score we used in the original gHMM was the probability of a line of text containing the query word

$$\mathbb{P}[\text{word in line}|\text{image}] = \frac{\mathbb{P}[\text{image}|\text{word in line}]\mathbb{P}[\text{word in line}]}{\mathbb{P}[\text{image}]} \tag{4.6}$$

The denominator on the right hand side is independent of the query, and can be calculated just once per line of text in our database. The numerator may also be calculated with the forward pass of a forward-backward algorithm by defining a modified set of hidden states and transition matrix. Unfortunately, however, these modifications are different for each query word. Thus the computational cost of indexing for search scales linearly with dictionary size, and has a large constant factor.

Our first step for more efficient search is to replace true probabilities with maximum likelihood. Our search score becomes

$$\mathbb{P}[\text{word in line}|\text{image}] \approx \frac{\max \mathbb{P}[\text{sequence through line containing word}]}{\max \mathbb{P}[\text{any sequence through line}]} \tag{4.7}$$

This is not yet computationally more efficient than true probabilities, but it allows us two optimizations that made search sublinear in the size of the dictionary.

As a first step, we calculate the likelihood of the maximum path through each vertex in the lattice. This score is an upper bound on the numerator search score for any path using this vertex. We heuristically prune sites with excessively low likelihoods, keeping only the top 30%.

We reduce the computational burden of search still further, by introducing a prefix tree. This is a datastructure that allows work performed for one query be reused for other words that share the same starting characters.

The prefix tree is a data structure used to represent a dictionary of words in a

compact manner. It is a directed tree graph in which each vertex has one outgoing edge for each unique label in our alphabet. We associate a string with each vertex as the sequence of labels for each edge from the root to this node. A dictionary may be encoded by adding an extra bit to each vertex, indicating whether its associated string is a valid word. In what follows, each vertex in the prefix tree is given a unique tree id.

Using a prefix tree, we can search the entire dictionary at a fraction of the cost of sequentially searching each word. Actually, storage requirements might be less for a suffix tree, given that Latin is an inflective language, but we did not experiment with this alternative.

Search proceeds as follows:

Run the forward backward algorithm to determine max path score for each site.
Prune sites with insufficiently high maximum path scores.
For each site in topological order,
    For each candidate predecessor site
        Calculate transition score
        For each prefix tree vertex linked to predecessor site
            Traverse tree edge with current site's label.
            new path score = prior path score + transition score
            If (*new path score* > *best so far*) for this path
                add tuple (score,prefix tree id, prior site) to list at this site
                If this site in the prefix tree is a word
                    calculate the search score for this word instance
                    Return search score.

An instance of a word in our lattice is a path that passes through a series of sites whose labels spell that word. The final search score for an instance of a word is the likelihood of the best path for the line passing through this sequence. This score is easily calculable by compositing the score from the (treeid, site)record with those calculated from the initial max path run.

Letting $V_{\text{pre}}(s)$ be the likelihood of the best path from the beginning of the line to site s. Let $V_{\text{post}}(s)$ be the likelihood of the best path from s to the end of the line. Let V_{word} be the score reported from the record associated with a specific (site,prefix tree node) pair. Finally, let s_α be the first site in the word instance sequence and s_ω be the last site of the word instance. Then the search score for this instance is

$$\text{search likelihood} = V_{\text{pre}}(s_\alpha) + V_{\text{word}} + V_{\text{post}}(s_\omega) \tag{4.8}$$

4.4 Extracting New Templates

One approach to estimate new character templates is to calculate the posterior of each possible character location under our model. The traditional EM approach to estimating new templates would be to use these sites as training examples, weighted by their posteriors. Unfortunately, the constraints imposed by 3 and even 4-gram character models seem to be insufficient. The posteriors of sites are not discriminative enough to get learning off the ground.

We need to restrict ourselves to a smaller, less noisy subset of the entire document. We have shown that even when the posteriors of individual characters are not discriminative, one can still achieve very good search results with the same model. The search word significantly restricts the transitions between hidden states, only allowing paths through the state graph that actually contain the query term. The longer the query, the more it constrains the model. Whole words impose much tighter constraints than a 2 or 3-gram character model. If we are given a large dictionary of words and no alternative word explains a region of ink nearly as well as the best scoring word, then we can be extremely confident that this is a true transcription of

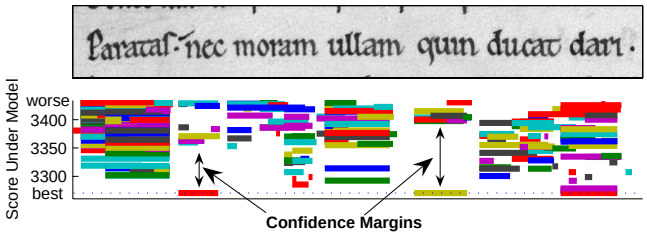


Figure 4.4: *Each line segment in the lower figure represents a proposed location for a word from our dictionary. It's vertical height is the score of that location under our model. A lower score represents a better fit. The dotted line is the score of our model's best possible path. Three correct words, "nec", "quin" and "dari", are actually on the best path. We define the* **confidence margin** *of a location as the difference in score between the best fitting word from our dictionary and the next best.*

that piece of ink.

Starting with a weak character model, we do not expect to find many of these "high confidence" regions, but with a large enough document, we should expect to find some. With the optimizations presented in the last section, we are able to efficiently return search scores for all words from a dictionary of 9500 words. Figure 4.4 gives a visual representation of search scores for a line, showing results for all words that score within a fixed percentage of the score of the best possible path. We mark two of these high confidence regions.

Restricting ourselves to the smaller set of high confidence document snippets, we can extract new, reliable templates with which to improve our character models. The most valuable of these new templates will be those that are significantly different from any in our current set. For example, in figure 4.7, note that our system identifies capital Q's, even though our only input template was lower case. It identifies this ink as a Q in much the same way that a person solves a crossword puzzle. We can easily

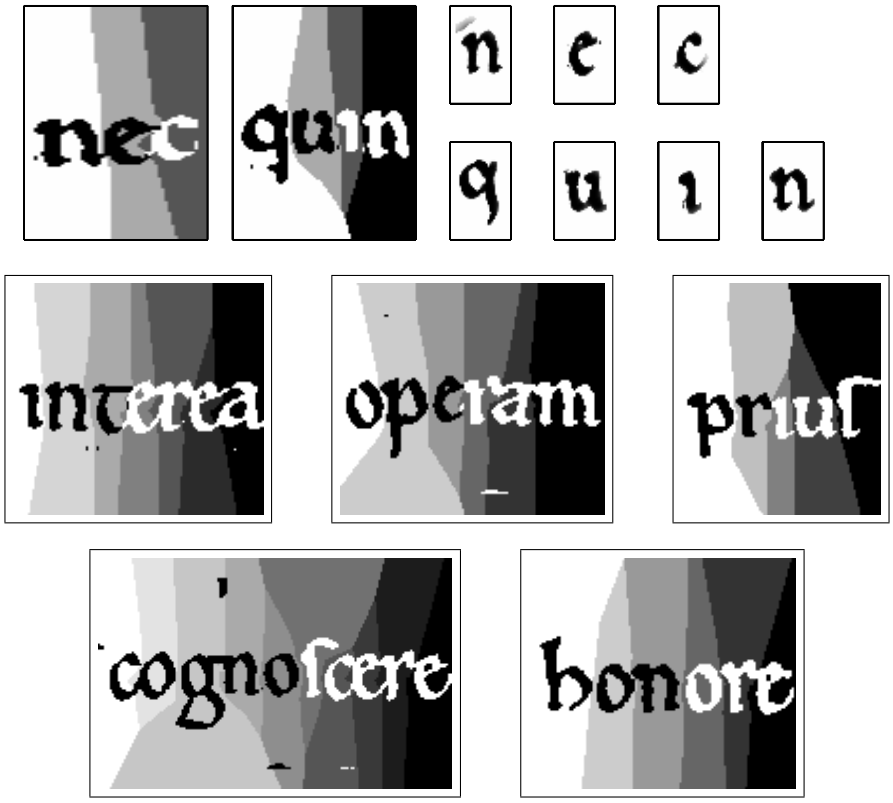


Figure 4.5: **Extracting Templates** *For a region with sufficiently high confidence margin, we construct the maximum likelihood template from our current exemplars.* **top left**, *and we assign pixels from the original image to a template based on its distance to the nearest pixel in the template image, extracting new glyph exemplars* **top right**. *These new glyphs become the exemplars for our next round of training.* **Lines 2 and 3** *show additional template masks from high confidence regions.*

infer the missing character in the string "obv-ous" because the other letters constrain us to one possible solution. Similarly, if other character templates in a word match well, then we can unambiguously identify the other, more ambiguous ones. In our Latin case, "Quid" is the only likely explanation for "-uid".

Within a high confidence region we have both a transcription and a localization of template centers. It remains only to cut out new templates. We accomplish this by creating a template image for the column of pixels from the corresponding

block templates and then assigning image pixels to the nearest template character (measured by Euclidean distance). Figure 4.5.

Given a set of templates extracted from high confidence regions, we choose a subset of templates that best explain the remaining examples. We do this in a greedy fashion by choosing the example whose likelihood is lowest under our current model and adding it to our set. Currently, we threshold the number of new templates for the sake of efficiency. Finally, given the new set of templates, we can add them to the model as additional character templates. Finally, we rerun our searches using the new model, potentially identifying new high confidence regions.

4.5 Results

Our algorithm improves the character model by gathering new training data from high confidence regions. Figure 4.7 shows that this method finds new templates significantly different from the originals. In this document, our set of examples after one round appears to cover the space of character images well, at least those in lower case. Our templates are not perfect. The "a", for instance, has become associated with at least one block that is in fact an "o". These mistakes are uncommon, particularly if we restrict ourselves to longer words. Those that do occur introduce a tolerable amount of noise into our model. They make certain regions of the document more ambiguous locally, but that local ambiguity can be overcome with the context provided by surrounding characters and a language model.

Figure 4.6: **Original Training Data** *These 22 glyphs are our only document specific training data. We use the model based on these characters to extract the new examples shown below*

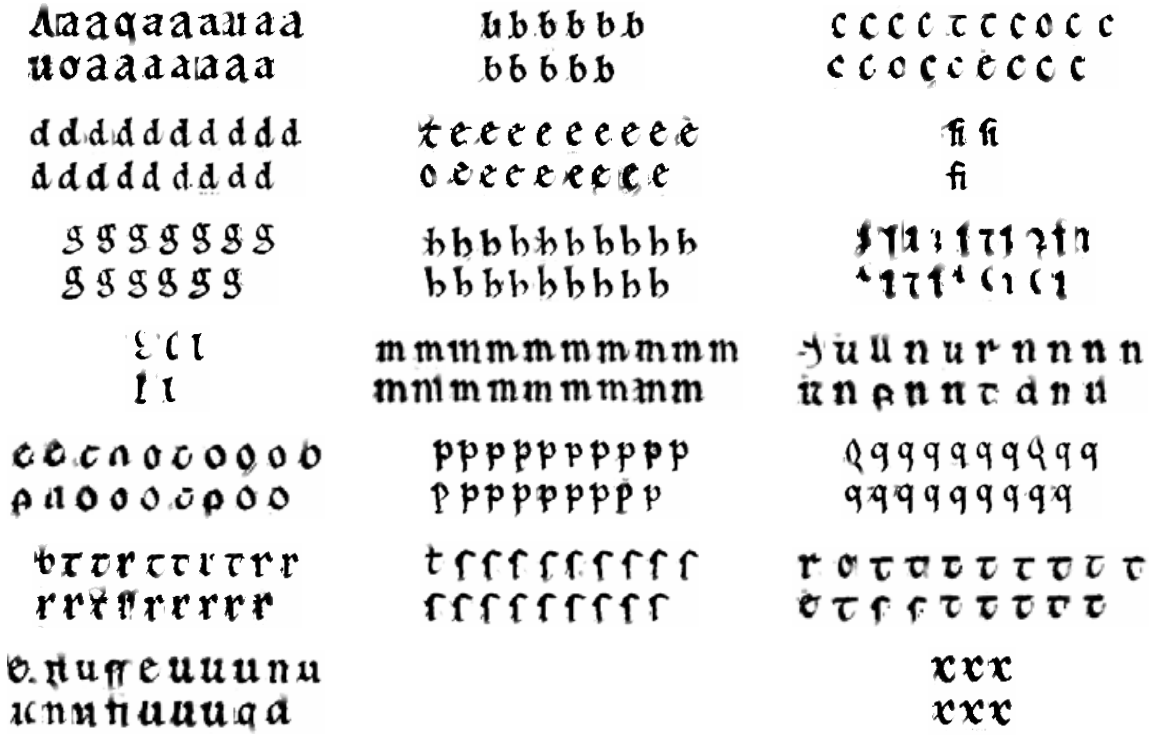


Figure 4.7: **Examples of extracted templates** *We extract new templates from high confidence regions. From these, we choose a subset to incorporate into the model as new exemplars. Templates are chosen iteratively to best cover the space of training examples. Notice that for "q" and "a", we have extracted capital letters, of which there were no examples in our original set of glyphs. This happens when the combination of constraints from the dictionary the surrounding glyphs make a "q" or "a" the only possible explanation for this region, even though its local likelihood is poor. The groups in this figure are the templates used in the second round search*

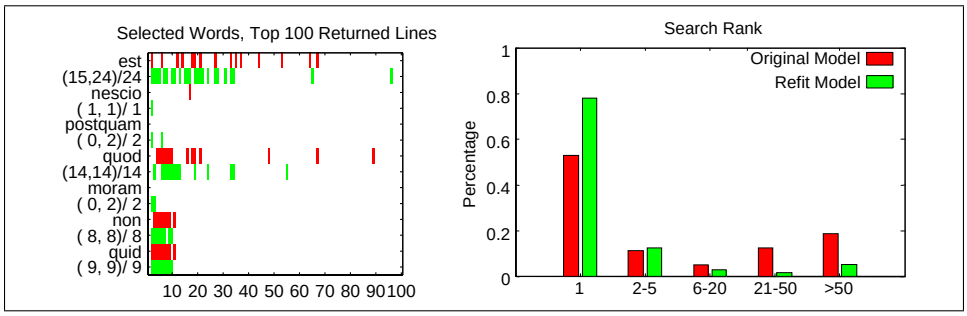


Figure 4.8: *The figure on the* **left** *gives a synopsis of search results for 7 selected words under our initial model, and after learning new templates. In each row, a colored dot at position i indicates that the i^{th} ranked search result actually contained the query term. In an ideal row, all colored points are to the left. The odd rows (in red) are the results using only the original templates. The even rows (in green) are the results after integrating templates from high confidence regions. Almost all search words in our corpus show a significant improvement. The numbers to the right (x/y) mean that out of y lines that actually contained the search word in our document, x of them made it into the top ten. The* **right** *figure shows a histogram of search ranks for 321 words that occur just once in the test set. In round 2, the number of words correctly returned as the first search result rises from 50% to 78%.*

4.5.1 Improved Search Results

We evaluate the method more quantitatively by testing the impact of the new templates on the quality of searches performed against the document. To search for a given word, we rank lines by the ratio of the maximum likelihood transcription/segmentation that contains the search word to the likelihood of the best possible segmentation/transcription under our model. The lowest possible search score is 1, happening when the search word is actually a substring of the maximum likelihood transcription. Higher scores mean that the word is increasingly unlikely under our model. In Figure 4.8, the figure on the left shows the improvement in ranking of the lines that truly contain selected search words. The odd rows (in red) are search results using only the original 22 glyphs, while the even rows (in green) use an additional 332 glyphs extracted from

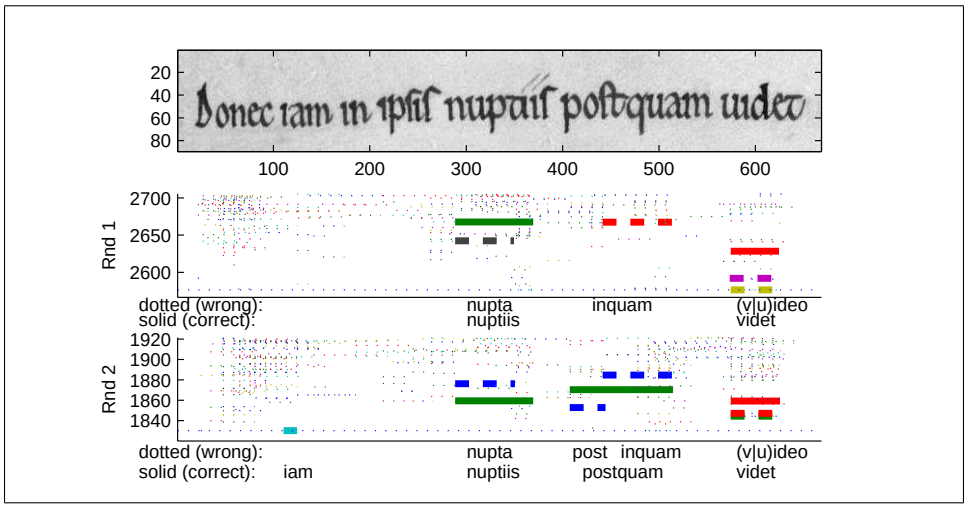


Figure 4.9: *Search Results with (Rnd 1) initial templates only and with (Rnd 2) templates extracted from high confidence regions. We show results that have a score within 5% of the best path.* **Solid Lines** *are the results for the correct word. Dotted lines represent other search results, where we have made a few larger in order to show those words that are the closest competitors to the true word. Many alternative searches, like the highlighted "post" are actually portions of the correct larger words. These restrict our selection of confidence regions, but do not impinge on search quality.*
Each correct word has significantly improved after one round of template reestimation. **"iam"** *has been correctly identified, and is a new high confidence region. Both* **"nuptiis"** *and* **"postquam"** *are now the highest likelihood words for their region barring smaller subsequences, and* **"videt"** *has narrowed the gap between its competitor "video".*

high confidence regions. Search results are markedly improved in the second model. The word "est", for instance, only had 15 of 24 of the correct lines in the top 100 under the original model, while under the learned model all 24 are not only present but also more highly ranked. The right of figure 4.8 shows that the new model also greatly improves performance for rare words. For 320 words occurring just once in the dataset, 50% are correctly returned as the top ranked result under the original model. Under the learned model, this number jumps to 78%.

Figure 4.9 shows the improved performance of our refitted model for a single line. Most words have greatly improved relative to their next best alternative. “postquam” and “iam” were not even considered by the original model and now are nearly optimal.

Chapter 5

Toward Extensible Holistic Searches

In the two preceding chapters, we have paid a great deal of attention to the problem of segmentation of a text image. The gHMM of chapter 3 attempted to segment a line into individual characters but often placed the characters inaccurately. In chapter 4, we introduced a substantially more complicated model, the Overlap gHMM, in order to accurately localize the bounding boxes of characters in the image. One way in which we could extend the overlap gHMM is by incorporating some of the more complicated single character recognizers introduced in chapter 2. This should allow us to more accurately localize and recognize characters, but at the cost of significantly more complicated training.

Another direction, however, is to recall that the goal of an OCR engine is not to segment an image into characters, but rather to transcribe entire words or lines of text. Since accurate segmentation into individual characters can be so difficult we might ask if it is really necessary. The work of Manmatha et al. [1995] argues that the answer may be no. Manmatha et al design a successful search engine that avoids the

complicated step of segmenting into characters, relying instead on features extracted from images of entire words. Unfortunately, the cost of simplifying segmentation is that their method is not extensible. It can only be used to search for words that are present in their training set. This chapter presents an outline for future work that we believe has the potential to blend the strengths of holistic methods such as Manmatha's wordspotting with the easy extensibility of sequential methods like the gHMM.

5.1 A Review of Wordspotting

Recall our search for "confidence regions" with the Overlap gHMM in chapter 4. We used the likelihood under the Overlap gHMM as a search score to find instances of words from a large dictionary. There are many parallels between the search for confidence regions and the wordspotting approach to search described by Manmatha [1995]. Manmatha uses the term wordspotting to describe a method that searches a document for any instances of words from a dictionary. It differs from traditional OCR in that it may simply refuse to predict at certain locations instead of hypothesizing an out of vocabulary transcription. Because the dictionary in a wordspotting method is fixed, the problem can be approached as a traditional classification task.

$$\mathbb{P}[\text{dictionary word}|\text{feature vector}]$$

Wordspotting methods are somewhat restrictive compared to traditional OCR methods in that they will not make hypotheses for "out of vocabulary" words. For search, however, this is not a serious drawback as long as the dictionary can be

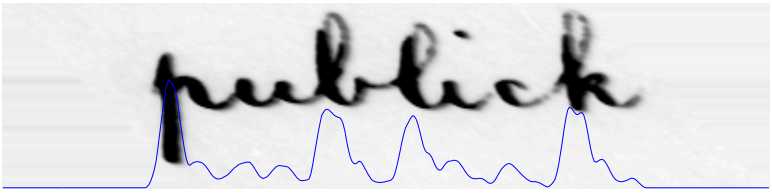


Figure 5.1: *One set of holistic features used in [Lavrenko* et al.*, 2004] are derived from the projection profile shown above. The authors extract the first 7 DFT coefficients from this 1D signal. Unfortunately, it is difficult to predict features such as these for words not found in the training set.*

easily extended when, for example, a user enters a never before seen query term. Unfortunately, this dynamic extension of the dictionary is not possible in the system presented by Manmatha et al. Their method can only search for queries that are present in their training set of word images. On this front, the gHMM is more appealing since its search dictionary may contain arbitrary words.

Still the holistic method introduced by Manmatha is attractive for numerous reasons. It avoids the difficult step of segmenting into individual characters. It is also easy to combine different types of features, measured on differently scaled windows, something considerably less straightforward in generative sequential models such as the gHMM. Manmatha's system also perform well despite the fact that its features encode only low resolution statistics about a word image, The use of low resolution features is attractive, because we believe it should make a system more robust to certain kinds of noise in the image, or moderate changes in font.

5.1.1 Wordspotting Features

For wordspotting, Manmatha uses features first described in work with Lavrenko [Lavrenko *et al.*, 2004]. These features can be reliably estimated even in quite low

resolution images. The authors measure simple global features such as the width and height of the bounding box. They also take various profiles of the word image. Figure 5.1 (repeated from section 2.4) shows one of these, a vertical projection profile. It is a smoothed 1D signal, derived by summing the number of ink pixels in each column of a deslanted image. Manmatha et al extract the first 7 coefficients of this signal's Discrete Fourier Transform as a feature vector. They extract similar vectors from the upper and lower word profiles, lines that roughly follow the outline of the top and bottom of the word. These coarse features prove quite discriminative. The authors report word accuracy rates of 65%, which ranks as state of the art on these types of manuscripts.

There are real benefits to working with low resolution features. Measurements based on low resolution features should be more robust to variations in segmentation or font style than the character templates we used with the gHMM or other more fine-tuned single character recognizers like those described in section 2.1.

The count of ascenders in a word, for instance, is unlikely to be affected by changing the word's bounding box by a few pixels. In the Overlap gHMM, we often faced the problem that the bounding box of characters was not well defined, and yet our features were very sensitive to getting the bounding box correct. Decoupling feature measurements from the need for exact segmentations is a very attractive feature.

Moreover, the type of profile measurements described above should be unaffected by small changes in the font. For example, the two fonts in figure 5.2 differ in their details, but their low resolution features are consistent with each other.

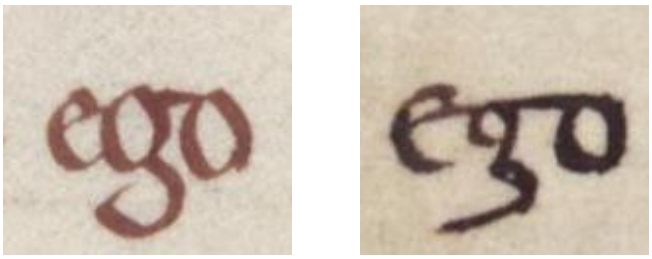


Figure 5.2: *The fonts of these two medieval manuscripts differ in subtle details, but their low resolution features are largely the same. Models like the gHMM that attempt to describe the appearance of individual characters must account for these subtle differences. Holistic models generate comparable transcription accuracy rates while working with more robust, low resolution features.*

5.2 Character Decomposable Features

The problem with the profile features used by Lavrenko is that there is no clear way to predict the expected feature vector for a word we have not seen. Contrast this with the gHMM, where we can search for any word without retraining, since our templates model individual characters and may be rearranged to model an arbitrary string without needing to retrain. To retain the strengths of both methods, we need features that can be measured without segmentation, but for which we can predict $\mathbb{P}$[feature vector|word] for words we have not seen. We now propose a family of features that meet both criteria.

5.2.1 Character Decomposable Holistic Features

Many holistic features are naturally modeled as the sum of contributions from individual characters. The width of a word, for example, is to a first approximation the sum of widths of the characters it contains. Given a training set of labeled word images, we can estimate these character widths without the need to segment the characters

themselves.

Let X be an $N \times A$ matrix, where N is the number of training words, and A is the number of characters in the alphabet. Let X_{ij} be the number of times the j^{th}character occurs in word i. Let y be the $N \times 1$ vector of word image widths and w an $A \times 1$ vector of parameters. The expected widths of individual characters may be estimated using non-negative least squares.

$$\underset{w}{\operatorname{argmin}}(Xw - y)^T(Xw - y) \quad \text{s.t.} \quad \forall i : w_i \geq 0$$

We know that this model of word widths is not entirely accurate. The width of certain character bigrams will be substantially different than the sum of their component characters because they form ligatures. We easily extend the above technique to include bigram terms by appending bigram counts to the matrix X. We would like the corresponding parameter vector to be sparse, since we know this effect is only important in a small set of cases. Since the effect of a ligature should always be to make the combined width smaller, we could constrain these bigram terms to be always negative which should also help to make the estimated parameter vector sparse.

Many other useful holistic features may be modelled in this manner. Counting the local maxima in the projection profile shown in figure 5.1 gives us an estimate of the number of vertical strokes, while counting the number of local maxima above some threshold would give an estimate of the total number of ascenders and descenders in the word.

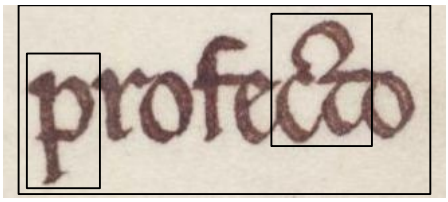


Figure 5.3: *When treating a word image holistically, it is easy to combine a wide variety of features measured on differently sized windows. From the full word window In this example, we might estimate the number of characters, as well as the number of ascenders and descenders, pruning to a small set of candidate words. Meanwhile, classifiers tuned to especially discriminative characters or bigrams allow us to further refine our predictions. It is much more difficult to combine such heterogenous features in sequential models.*

5.2.2 Templates Revisited

In fact, any method that returns a set of point features in the image can be trained as a decomposable feature. We could, for instance, take the local maxima of the responses to the gHMM's character templates. Unlike with the gHMM, however, we do not need to model every character. Holistic, low resolution features get us most of the way there, and we can opportunistically choose to use templates that are particularly discriminative.

In Terence, for example, a "p" or "q" is very distinctive, even if the image is somewhat blurry. Moreover, they are quite rare letters, and so correctly identifying them significantly constrains the set of possible words at that location. Other characters are less useful. An "i" for example, is a much more common letter and is not visually distinctive. In the Terence manuscript, an "i", seen in isolation, is virtually indistinguishable from pieces of other characters including "n", "m" and "u". It is therefore difficult to segment accurately, and not particularly informative even if we do succeed in locating it.

Additionally, this method may incorporate templates for discriminative *groups*

of characters. The bigram 'ct,' for example, is very discriminative. (Figure 5.3). We might consider automating the process of choosing discriminative windows by allowing the system could to randomly choose image windows from the training set. The ability to easily combine windows with variable length transcriptions is far more complicated in sequential models like the gHMM.

All of the features we've described are modeled as

$$x_w^T \phi + \epsilon$$

where x_w is the vector of character counts, ϕ our fit parameters, and ϵ a Gaussian noise term.

We are inspired in part by the successful tradition of "bag of words" feature models used in both computer vision and NLP. For example, NLP methods for clustering documents, generally work with vectors of word counts, discarding any information about word order. See, for example, [Manning and Schütze, 1999]. Researchers in Computer Vision have similarly built successful object recognition systems based on counts of local point features, discarding information about their relative positions. See for example, Sivic et al. [2005]. In each of these works, the input features are based on counts of local features, while the relative spatial location of these point features is largely ignored.

5.2.3 Extending the Dictionary

The decomposable features of this section are modelled as draws from a Gaussian distribution conditioned on the character count vector of a word. We can train their

parameters without ever segmenting more than word boundaries. Since we have a generative model, we are now in a position to estimate the expected feature vector for words that *are not in our training set*. This means that, as desired, the dictionary used by our wordspotting method may be extended on the fly, without the need to provide additional training data.

5.3 Equivalence Classes

Particularly given only low resolution features, we expect that maximum likelihood classification of words may often be unreliable. Clearly, there are some strings that will be more easily confused than others. We argue that there will be some distinctions that it is simply unwise to try and make. We define an *equivalence classes* of words as a set of words that are indistinguishable given a particular feature representation. In this section, we outline a method to identify such equivalence classes. Rather than train via maximum likelihood, we propose explicitly build binary classifiers that test for membership in these classes.

As an illustrative example, take the hypothetical case where our features are three perfect oracles. One gives the number of ascenders in an image, another the number of descenders, and the last gives the image's width in "stroke units" which we'll define as 1 for "i","j" and "l", 3 for "m" and 2 for all other characters. The vector $[3, 0, 5]$, for instance, is a 5 unit wide image, with three ascenders and no descenders.

Given only the vector $[3, 0, 5]$, we cannot uniquely reconstruct the true transcription. However, we can prune the possible strings down to a very small set. There are only six common English words or substrings of words that match these criteria.

They are "bill", "kill", "hill", "fill", "lift" and "till". These six features form an equivalence class under the given set of features. In a real world setting, the feature vectors for these 6 strings are unlikely to exactly coincide at a single point and one of these 6 will probably have a marginally higher likelihood than the others. However, it seems unreasonable to now try and differentiate them, simply because we've added noise to our inputs.

On the other hand, it seems likely that these six transcriptions will form a well defined cluster in feature space. Treating the expected feature vectors for our transcriptions as a point cloud in feature space, we can actually search for such clusters. The most useful of these clusters will be have few members, and be well separated from the nearest alternative transcription. We propose to search for such "useful" clusters, and to build binary classifers for membership. In the example above, for example, the classifier would predict that the transcription was one of these six words, but would not attempt to distinguish which one.

Most all other OCR methods, including the gHMM of previous chapters will always return a maximum likelihood string, but we feel that the concept of equivalence classes is very real. For a certain set of features, there are differentiations one should not try to make. Methods that return a single maximum likelihood transcription do not remove the ambiguity, they simply mask it.

5.4 Word Pieces and Efficient Search

As described thus far, our decomposable features throw away information about the relative spatial position of features in a word. It is possible to reintroduce some spa-

tial reasoning, however, without the need to return to sequential models, by building classifiers not only whole words, but also smaller pieces. As Lavrenko et al have demonstrated, holistic methods clearly benefit from from calculating features on windows much larger than characters. However, there is nothing in the method requiring that these windows contain entire words.

In the previous section, we learned equivalence class classifiers for word images based on decomposable features. These features serve equally well as an appearance model for word substrings. We will build equivalence class classifiers not just for whole words, but also for these substrings. Once we have pruned our set of hypotheses for an image down to a small set using whole word classifiers, we can further refine our hypotheses by examining the classifications for subwindows of the image, and throwing away words that are inconsistent.

Suppose that we have used the equivalence class classifiers of the previous section to narrow the hypotheses for an image to a set of two words: "mane' and "amen". If we have an accurate classifier for the snippet "me," we can differentiate the two by searching subimages of the input for this string. We additionally know that these snippets should also appear in a consistent ordering. In "amen," "am" must lie to the left, and partially overlap "me" etc. In some cases, it may be necessary to enforce this type of spatial consistency, and this leads us back to a sequential models. We hypothesize, however, that in many cases the simpler test of existence of the correct substrings may be enough.

One unfortunate aspect of this approach is that is still involves an exhaustive search over windows in the image. By focusing exclusively on whole word images, existing holistic methods avoid this costly search. Word boundaries are attractive

segmentation points because they are generally easy to identify, even in low resolution images. As we have discussed, however, word boundaries are not the only repeatable low resolution features. Segmenting a word into characters is difficult, but there are repeatable visual cues that we should be able to consistently identify. We can use these cues to extract subwindows corresponding to equivalence classes.

5.4.1 Inference with Substrings

The proposed system will select image subwindows opportunistically, and so the number of pieces per input image is not fixed. Because of this, we can no longer build a classifier by simply concatenating features from each window into a larger vector. One possible method to combine the evidence from these subimages is as follows:

For each word in our lexicon, let S_w be the set of all substrings of word w, and let F be the set of feature vectors for the windows in some word image. We choose the word w that maximizes

$$\prod_{f \in F} \max_{s \in S_w} \mathbb{P}(s|f) \tag{5.1}$$

This assigns each window in the word the score associated with the best match of any substring of a word. We choose the transcription with the highest score, and report "no match," if all dictionary words score below some cutoff.

5.4.2 Efficient Search with Equivalence Classes

The search method using Equation 5.1 must be calculated separately for each word in our dictionary, which presents a prohibitive computational burden for large dictionaries. We can use snippet classifiers to build a far more efficient search method.

Assume we have built a set of snippet equivalence class classifiers. For each of these, we have trained a binary classifier that can predict whether an image window is or is not a member of that class. Applying each of these classifiers to an input image, the model will identify a small set of windows that are members of one of the classes. In place of equation 5.1, our classification becomes basically a logical and. Each window must be explained by some substring of a candidate word.

Each equivalence class acts as an index into our dictionary of words, and our output wordset is simply the intersection of the lists from each window.

5.4.3 Ignoring Noise and Unknown Glyphs

Both holistic methods and sequential methods like the gHMM suffer if part of a word image is obscured by noise like the examples in figure 5.4 They have similar difficulties if a word includes glyphs that were not present in the training set. While working with the gHMM, for instance, we had no model of capital letters. The model was forced to predict labels for these regions, however, and this frequently hurt the transcription of the whole word. Holistic models face an equivalent problem, because noise or unknown glyphs will corrupt their feature measurements.

By opportunistically choosing snippets, our proposed method may be able to address this problem. We expect that if some region of the image is corrupted enough to significantly alter the measured feature vector, it is unlikely that the system will confuse it with a different string. It is far more likely that the corrupted feature will be so far from any possible transcription that our method will simply refuse to predict. This means that the set of image pieces considered in equation 5.1 will naturally exclude those that cover the problem region.

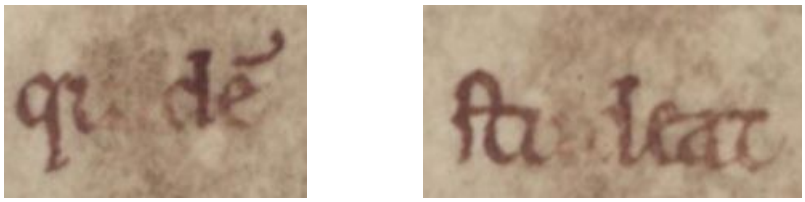


Figure 5.4: **left** *quidem,* **right** *studeat. are two examples of a corrupted word image. The gHMM of chapter 3 has trouble with such images because it is forced to predict a character for the noisy regions. The method of this chapter, on the other hand, should refuse to predict on any window including the noisy region. This has the desirable side effect of making it base its prediction on the portions of the word it can confidently transcribe.*

As a corollary, we can also use word piece predictions to say something about out of dictionary words, even if it can't propose a full transcription. For instance, the system might not be able to recognize the full word, but could say that it ends in "ing". We can cache this partial information to improve search speeds.

Chapter 6

In Conclusion

Optical Character Recognition is often falsely represented as a solved problem. However, there are a wide array of publicly available documents on which current OCR techniques fail. These provide ample evidence that the field still has open problems. In particular, current methods lack the ability to easily adapt to previously unseen fonts. Throughout this dissertation, we have focused our efforts on the design of such an adaptable system. With the gHMM and Overlap gHMM, we have successfully designed a system that may be initialized with a bare minimum of document specific training data.

This ease of initialization is the most notable feature of the gHMM when compared to leading alternatives such as sliding windows [Vinciarelli *et al.*, 2004], Space Displacement Neural Networks [LeCun *et al.*, 1998], or Document Image Decoding [Kopec and Lomelin, 1995]). We are able to initialize the system using only a single instance of each character from the target document, and language statistics learned from electronic texts spidered from the Internet. To our knowledge, this is the first

approach that does not require any line aligned transcriptions as training data.

In chapter 4, we extended the gHMM to allow more accurate localization of character templates. We showed that this Overlap gHMM can be used to automatically refine our initial character models. This in turn led to substantially improved search accuracy over the original gHMM.

Our work with the gHMM has provided ample evidence of the importance of language models. In the original gHMM, we provided only trigram character statistics. Such a model gives high probability to many strings that are not valid words. In order to automatically extract a clean training set of new character exemplars in chapter 4, we used a far more discriminative language model. We generated a large list of Latin words and searched the document for each one. We extracted characters only from "high confidence" regions, which we defined as a location where only one dictionary word had high probability under the model. Traditionally, OCR methods rely only lightly on language cues, the reason being that too tight a language prior will force the model to ignore the ink and mis-transcribe certain rare and out of vocabulary words. This is a valid concern, but the important insight of chapter 4 is that a model may be free to use very restrictive language models *during training*, in order to learn more accurate character models. Once the character models have been learned, we may relax the language constraints.

Our work with the gHMM has taught us a simple, but hard won, lesson. *Accurate localization of individual characters is difficult.* Not only the gHMM, but all models that decompose words into smaller pieces must deal with the problem of segmentation and localization of those smaller windows. The character models in the gHMM are quite sensitive to accurate localization, and our overlap gHMM of chapter 5 was de-

signed specifically to address this problem. However, the flexibility to accurately place characters came at the cost of a significantly more complicated inference process. In contrast, methods such as sliding windows and the SDNN [Vinciarelli *et al.*, 2004; LeCun *et al.*, 1998] are less sensitive to accurate localization when performing inference, but this is because they have built robustness to translation into their character appearance models. The cost for this approach is the requirement for substantially more training data to estimate appearance models.

A desire to avoid the cost and complexity of segmenting individual characters, has led us to reconsider holistic models such as the wordspotting technique presented in [Lavrenko *et al.*, 2004]. Moreover, our search for high confidence regions in chapter 4 illustrated the benefits of turning the task of transcription into a classification problem over a known dictionary. Holistic methods are also appealing because they appear to achieve comparable results to sequential methods while using lower resolution features, features that are more likely to be shared across different scripts.

Clearly, transcribing from a fixed dictionary is unacceptable if the system will encounter out of vocabulary words, but for search the queries are provided by a user. If the dictionary can be easily extended at query time, then there is never an out of vocabulary problem. Unfortunately, no existing holistic methods can be adapted on the fly in this way. They must be retrained for new words.

We wish to reemphasize the distinction between search and transcription in OCR. Most models in OCR and handwriting recognition, the gHMM included, are originally designed to transcribe documents. Clearly, if a model can provide a perfect transcription, then search is a trivial problem. But transcription is harder than search, because it must hypothesize transcriptions for words that are outside its vocabulary. In search,

the vocabulary is provided, although we may need to expand our dictionary on the fly. Treating transcription as a classification problem over a fixed dictionary allows for far greater flexibility in our choice of models than with transcription models. We can easily mix feature spaces and window sizes. This means that we can more easily focus on the pieces of a font that are both visually distinctive and linguistically informative, while ignoring more difficult regions.

In chapter 5 we outline a direction for future work that is explicitly aimed at search. We propose a set of methods designed to avoid the thorny issue of character segmentation. As in holistic models, our approach treats transcription as a classification task over a known vocabulary. Unlike existing holistic methods, however, this dictionary may be easily extended without retraining. We believe that this approach may achieve search accuracy rates comparable to existing methods, while preserving the easy adaptability of the gHMM.

We hope the reader will agree that adaptability in print and handwriting recognition is an important, and still quite open line of research. For example, the documents we have considered are in English and Latin, but many collections will be in languages such as Arabic or Chinese that will certainly present new challenges. In this dissertation, we have made a number of positive steps toward an easily adaptable system for OCR and handwriting recognition, but there are still many interesting challenges ahead.

Bibliography

[Bancroft, 2006] Bancroft. Mark twain's letters,1876-1880: An electronic edition. http://bancroft.berkeley.edu/MTP/publications.html, 2006.

[Belongie *et al.*, 2002] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence*, 2002.

[Bodleian, 1100] Bodleian. Terence's comedies, ms. auct. f. 2.13, 1100.

[Breuel, 1994] T. Breuel. A system for off-line recognition of handwritten text, 1994.

[Burges, 1992] C.J.C. Burges. Shortest path segmentation: A method for training a neural network. *Proc. Int'l Joint Conf. Neural Networks*, Oct. 1992.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[Cover and Thomas, 1991] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[Impedovo, 1993] S. Impedovo, editor. *Fundamentals of Handwriting Recognition*, chapter Hidden Markov Models in Handwriting Recognition. Springer-Verlag, 1993.

[Jelinek, 1997] Frederick Jelinek. *Statistical methods for speech recognition*. MIT Press, Cambridge, MA, USA, 1997.

[Jordan, Pre Print] Michael I. Jordan. An introduction to probabilistic graphical models. Unpublished manuscript, Pre-Print.

[Kopec and Lomelin, 1995] G. Kopec and M. Lomelin. Document image decoding approach to character template estimation, 1995.

[Kornfield *et al.*, 2004] E.M. Kornfield, R. Manmatha, and J. Allan. Text alignment with handwritten documents. In *Proceedings of Document Image Analysis for Libraries (DIAL)*, 2004.

[Lafferty *et al.*, 2001] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[LatinLibrary, 2004] LatinLibrary. The latin library website, 2004.

[Lavrenko *et al.*, 2004] V. Lavrenko, T. Rath, and R. Manmatha. Holistic word recognition for handwritten historical documents. In *Proceedings of Document Image Analysis for Libraries (DIAL)*, pages 278–287, 2004.

[LeCun and Cortes, 1998] Y. LeCun and C. Cortes. The mnist database. http://yann.lecun.com/exdb/mnist/, 1998.

[LeCun *et al.*, 1997] Y. LeCun, L. Bottou, and Y. Bengio. Reading checks with graph transformer networks. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 151–154, Munich, 1997. IEEE.

[LeCun *et al.*, 1998] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Leroux *et al.*, 1991] M. Leroux, J.C. Salome, and J. Badard. Recognition of cursive script words in a small lexicon. In *Proc. First Int'l Conf. Document Analysis and Recognition*, pages 774–782, 1991.

[Madhvanath *et al.*, 1997] Sriganesh Madhvanath, Evelyn Kleinberg, Venu Govindaraju, and Sargur N. Srihari. The hover system for rapid holistic verification of off-linehandwritten phrases. In *ICDAR '97: Proceedings of the 4th International Conference on Document Analysis and Recognition*, pages 855–860, Washington, DC, USA, 1997. IEEE Computer Society.

[Manmatha *et al.*, 1995] R. Manmatha, C. Han, and E. M Riseman. Word spottin: A new approach to indexing handwriting. Technical Report UM-CS-1995-105, , 1995.

[Manning and Schütze, 1999] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[Mello, 1999] Carlos De Mello. A comparative study on ocr tools, 1999.

[Mohri *et al.*, 2000] M. Mohri, F. Pereira, and M. Riley. Weighted finite state transducers in speech recognition. In *ISCA ITRW Automatic Speech Recognition: Challenges for the Millenium*, pages 97–106, 2000.

[Moreau, 1991] J. Moreau. A new system for automatic reading of postal checks. In *Proc. Int'l Workshop on Frontiers in Handwriting Recognition*, pages 121–132, Bonas, France, Sept 1991.

[Mori *et al.*, 1995] Shunji Mori, Ching Y. Suen, and Kazuhiko Yamamoto. *Historical review of OCR research and development*, pages 244–273. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[Murphy, 2002] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, MIT, 2002. Chair-Stuart Russell.

[Oxford, 2007] Oxford. Early manuscripts at oxford university, 2007.

[Plamondon and Srihari, 2000] Rejean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[Rabiner and Juang, 1993] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[Rath and Manmatha, 2003] T. M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 521–527, 2003.

[Sayre, 1973] K.M. Sayre. Machine recognition of handwritten words: a project report. *Patter Recognition*, 5(3):213–228, 1973.

[Sivic *et al.*, 2005] Josef Sivic, Bryan Russell, Alexei A. Efros, Andrew Zisserman, and Bill Freeman. Discovering objects and their location in images. In *International Conference on Computer Vision (ICCV 2005)*, October 2005.

[Srihari and Kuebert, 1997] S.N. Srihari and E.J. Kuebert. Integration of handwritten address interpretation technology into the united states postal service remote computer reader system. *icdar*, 00:892, 1997.

[Srihari, 1986] Sargur N. Srihari. Document image understanding. In *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*, pages 87–96, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[Steinherz *et al.*, 1999] Tal Steinherz, Ehud Rivlin, and Nathan Intrator. Offline cursive script word recognition - a survey. *IJDAR*, 2(2-3):90–110, 1999.

[Taskar *et al.*, 2003] B. Taskar, C. Guestrin, and D. Koller. Max margin markov networks, 2003.

[Vinciarelli *et al.*, 2004] Alessandro Vinciarelli, Samy Bengio, and Horst Bunke. Offline recognition of unconstrained handwritten texts using hmms and statistical language models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):709–720, 2004.

[Vinciarelli, 2003] A. Vinciarelli. *Offline Cursive Handwriting: From Word to Text Recognition*. PhD thesis, University of Bern, Bern, Switzerland, 2003.

[Whistler *et al.*, 2004] Ken Whistler, Mark Davis, and Asmus Freytag. Unicode technical report 17, character encoding model. Technical report, Unicode Technical Committee, 2004. [Online; accessed 24-January-07].

[Wilensky, 1995] Robert Wilensky. Uc berkeley's digital library project. *Commun. ACM*, 38(4):60, 1995.

[Wilkinson, 1994] R. Allen Wilkinson, editor. *The Second Census Optical Character Recognition System Conference*, 1994.