Recognition of Handwritten Responses on US Census Forms

Thomas M. Breuel
Institut Dalle Molle d'Intelligence Artificielle Perceptive (IDIAP)
C.P. 609
1920 Martigny, Switzerland
tmb@idiap.ch

ABSTRACT

The design and implementation new system for the recognition of handwritten responses on US Census forms, a hard real-world recognition task, is described. The system goes from raw, binary scanned images of census forms to ASCII transcriptions of the fields contained within the forms. The recognition of the handwritten text is based on extracting a large number of character subimages from the input that are then individually classified using a MLP (Multi-Layer Perceptron). A Viterbi algorithm is used to assemble the individual classified character subimages into optimal interpretations of the input, taking into account both the quality of the overall segmentation and the degree to which each character subimage of the segmentation matches a character model. The system uses two different statistical language models, one based on a phrase dictionary and the other based on a word dictionary. Results from recognition based on each language model are integrated using a decision tree classifier. The system has been tested on the NIST data base of handwritten responses on 1990 US Census forms and demonstrated high recognition rates. The effects of design decisions, remaining sources of errors, and comparisons with other systems are discussed.

1. Introduction

Handwriting recognition is an important practical problem. Its applications include processing bank checks, postal mail addresses, census forms, tax forms, office documents, library information systems, FAX routing, personal organizers, and many more.

It is also an interesting problem in computational vision, since it encompasses many of the major problems in computational vision: feature extraction, figure-ground problems, segmentation, learning, idiosyncratic and "natural" shape variation, and the integration of top-down knowledge. From an experimental point of view, the recognition of handwritten text is a particularly nice model system, since it is a well-defined problem and large amounts of training and test data are available.

This paper describes a new system for the recognition of handwritten text. There are a number of noteworthy features about this system:

- *complete forms-to-ASCII system*: The system starts with a raw scanned input form and outputs a segmentation and ASCII transcription of the form; this allows us to estimate overall recognition rates in a realistic setting.

- *strictly bottom-up processing*: Each processing stage computes an output that is then passed on to the next processing stage; information never flows backwards.
- *pre-stroke segmentation*: A novel segmentation algorithm particularly suited to the segmentation of handprinted text is used.
- *character subimage classification using a MLP*: A MLP (Multi-Layer Perceptron with sigmoidal activation functions) is used to estimate posterior probabilities $P(w_i|x)$, where w_i is the class of the character subimage and x is the corresponding input vector.
- *segmental recognition*: Many alternative segmentations of the input string are possible. In addition to classifying individual characters, the MLP is used directly to distinguish good segmentations from bad ones. To accomplish this task, the MLP has been trained to reject character subimages that are part of bad segmentations.
- *dictionary back-off strategies, hypothesis integration, and rejection via decision trees*: As part of postprocessing, the system generates multiple hypotheses for each input; these are integrated using a decision tree classifier.

An overview of the system is shown in Figure 1. Four major processing stages can be distinguished. *Preprocessing* starts with the raw input and computes images of isolated, normalized strings of handwritten text found within the raw input. The *segmentation* stage divides up the handwritten text into potential character subimages and describes the spatial relations among those character subimages compactly using a graph structure. The *recognition* stage determines how similar each character subimage is to known, well-segmented character subimages in the database. Finally, in the *postprocessing* stage, the individual character subimages are assembled into a globally optimal interpretation of the handwritten input string, taking into account constraints imposed by the language model.

2. Statistical Foundations

The statistical basis for the system described in this paper is similar to that of segment-based speech recognition systems. The presentation and notation used below follow the papers by Leung *et al.*[1] and Breuel[2] closely.

The goal of the system is to identify the character string $W = w_1 \ldots w_n$ that represents the most probable interpretation of the input image x. More precisely, in the set of all permissible strings Σ^*, we want to find the string $W \in \Sigma^*$ that maximizes $P(W|x)$. In a Bayesian framework, this represents the optimal decision under a zero-one loss function.

We can think of this interpretation of the input image as a character string as consisting of two parts: a segmentation $S = s_1 \ldots s_n$ of the input image, that is, a partitioning of the image into character subimages, and an interpretation of each character subimage x_i as a character w_i.

Consider now the joint conditional distribution $P(W, S|x)$. The desired probability $P(W|x)$ is related to this by summing over all possible segmentations:

$$P(W|x) = \sum_S P(W, S|x) \tag{1}$$

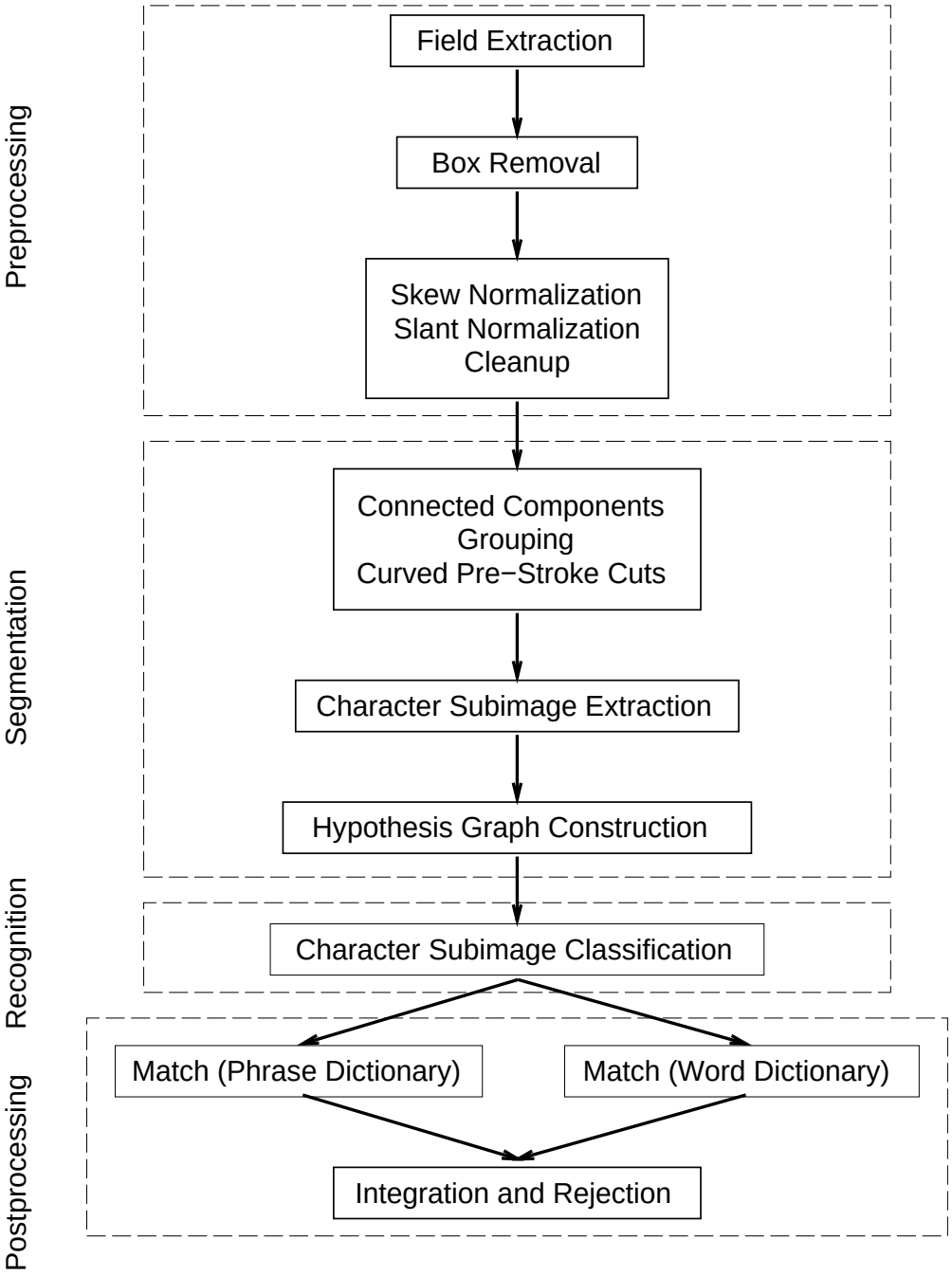

Figure 1: An overview of the system.

We impose the *a priori* constraint that $P(W, S|x) = 0$ if $|W| \neq |S|$. That is, the number of character subimages generated by the segmentation and the length of the hypothesized string must be the same.

In this system, we will be using context-independent models for the character subimages. Therefore, we want to express $P(W, S|x)$ in terms of the individual w_i and s_i:

$$
\begin{aligned}
& P(W, S|x) && (2)\\
= \; & \frac{P(x|W, S)\, P(W, S)}{P(x)} && (3)\\
\approx \; & P(W) \prod_i \frac{P(x_i|w_i, s_i)\, P(s_i|w_i)}{P(x_i)} && (4)\\
= \; & P(W) \prod_i \frac{P(w_i, s_i|x_i)}{P(w_i)} && (5)
\end{aligned}
$$

Here, the approximation is based on the assumption of context independence. Leung *et al.*[1], in contrast, are using context dependent models.

This leaves us with the problem of estimating and interpreting the factors $P(w_i, s_i|x_i)$. Our choice of model for these factors determines the nature of the overall probability model $P(W, S|x)$. In the current system, these factors are chosen to be proportional to the probability that segment x_i is an individual, well-segmented handwritten character of class w_i (up to a global normalization factor that ensures that $\sum_{W,S} P(W, S|x) = 1$). In practice, this choice is realized by the particular set of training examples chosen to train the neural network.

To understand the properties of this probability model better, it is useful to look at the contribution of the segmentation to the overall probability. By summing over all possible strings W, we obtain $P(S|x) \approx c \prod_i P(s_i|x_i)$. That is, the probability of a segmentation S is the product of the probabilities that each of subimages x_i represents an individual, well-segmented handwritten character.

Clearly, this is only an approximate probability model for $P(S|x)$. In particular, the implicit independence assumption is violated for the actual segmentation method used. It remains to be seen whether modeling these dependencies has any effect on recognition rates in practice. On theoretical grounds, it appears that the magnitude of these dependencies is small compared to other modeling errors.

3. Database and Input Data

The database used in this work was made available by NIST for participation in the Second Census OCR Conference[3]. In the work described here, a subset of 1200 forms images digitized from paper is used for training and testing. These forms images can be found in the subdirectories of the `data2` directory of the CD-ROM containing the "NIST Special Database 12/Miniform Database 2 of Binary Images from Microfilm and Paper". This set of 1200 forms images will henceforth be referred to simply as "the NIST database".

Each forms image consists of 5 miniforms. Each miniform contains the responses to three questions about the nature of the employment of the respondent. The forms in the database contained actual responses received by the census bureau. Hence, a wide variety of writing instruments, writing styles, and input qualities are represented. Input was digitized to bilevel images at a resolution of 200 DPI. An example of a miniform is shown in Figure 2.

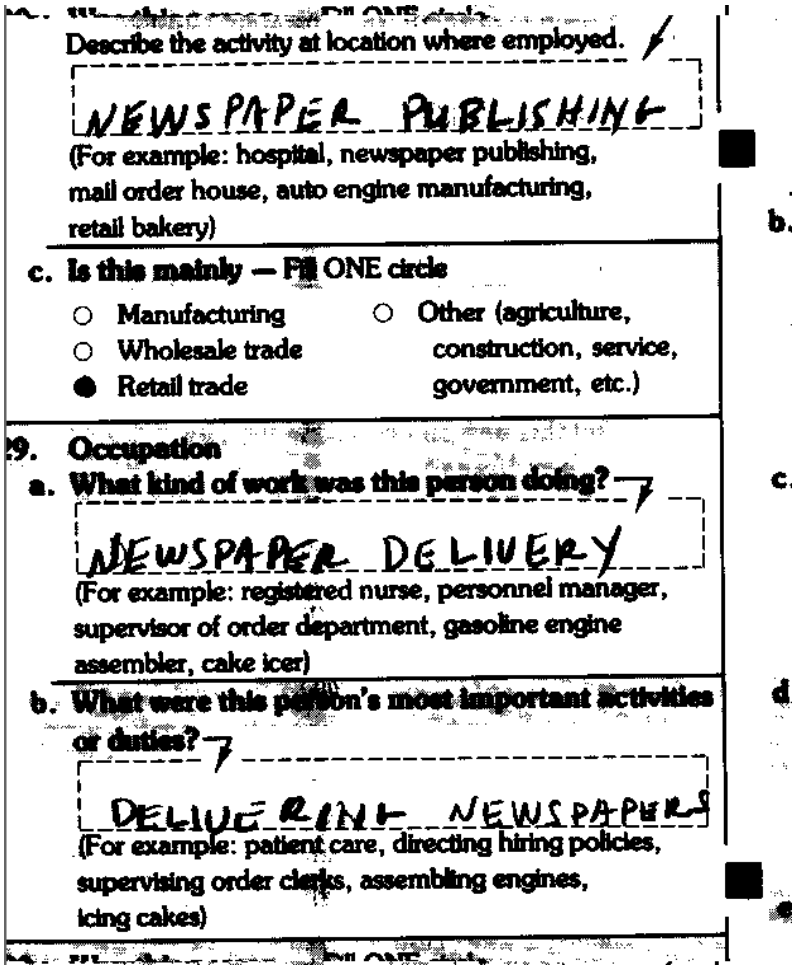

Figure 2: A sample miniform from the database written in an upper-case, well segmented style (see Table 1). The input to the system consists of a vertical concatenation of five such miniforms into a large image of about 600×3700 pixels. A significant fraction of miniforms have spurious markings outside the input field, have non-negligible skew, or have been binarized using a threshold that is slightly too low or too high (leading to fading of characters or the presence of noisy areas).

For each field, the database contains a transcription of the alphanumeric characters contained in that field. Non-alphanumeric characters (e.g., / and &) are not transcribed. Transcriptions contain spaces, but these often fail to represent physical spaces present in the input and sometimes transcribe space that is not actually present in the input.

There are a number of properties of input in the database that complicate recognition, among them:

- In some input fields, writing falls outside the box. Strokes reaching above the top or below the bottom of the printed bounding box are due to excessive textheight or excessive ascender/descender length. Strokes and characters falling outside the right margin of the printed bounding box occur because the writer has reached the right

style	NIST	CEDAR
handprinted, well-segmented, upper-case	22%	8%
handprinted, linked, upper-case	18%	9%
handprinted, well-segmented, mixed-case	21%	3%
handprinted, linked, mixed-case	28%	9%
cursive	6%	67%
two lines	3%	N.A.
faded	2%	4%
strike-through	<1%	N.A.

Table 1: Approximate frequencies of different writing styles in the NIST database, based on a sample of 580 randomly chosen fields. Generally, well-segmented and upper-case styles are easier to recognize, but there are frequent exceptions to this rule. For comparison, the same classification is shown for 200 arbitrarily chosen words from the CEDAR "bd" database of city names.

margin of the box without finishing his response. Strokes that fall significantly outside the top or bottom margin of the box are very difficult to recover, since they overlap printed instructions on the form.

- A non-negligible fraction of input fields contains two lines of text. These can be the result of corrections to a response using an insertion mark (often indicated using a $\vee$). In other cases, the writer seems to have anticipated giving a long response.

- Responses show a significant degree of pitch variability (i.e., variability in character spacing and character width). Probably the most common cause is that the writer is approaching the right margin of the box and anticipates not being able to complete his responses at the current pitch. Some responses also show significantly non-linear baselines and variable character slant.

- Some input fields contain very faded or broken up characters. Part of the reason for this is that the input images are binary and that the threshold for binarization was set based on the printed marking on the forms, not the handwritten input.

- Some input fields contain struck out handwriting, often using erratic scribbles; usually, struck out text is not transcribed. Occasionally, forms contain numerous extraneous markings, such as check marks marking each response that has been completed.

These phenomena could probably be reduced (though not completely eliminated) by some simple improvements in the design of the forms.

For a sample of 580 fields, the percentages of writing styles and exceptional input fields were determined; this is shown in Table 1.

It is interesting to compare the NIST database with the CEDAR database of handwritten US postal addresses, another widely used database for developing and testing recognition systems for handwritten text. What is striking is the very different distribution of writing

styles between the two databases (see Table 1). While the NIST database consists only to about 6% of input written in a cursive style, about 67% of the inputs in the CEDAR database are cursive.

4. Forms Segmentation

Before any character recognition can take place, handwritten character strings need to be extracted from the miniforms. The first step of this extraction process is to locate approximately the bounding boxes containing the handwritten responses.

Perhaps the most obvious approach is to try to locate the printed dashed boxes surrounding each field directly (see Figure 2). An algorithm based on mathematical morphology was implemented and located and extracted more than 90% of the fields correctly. Failures were mainly due to noise, fading of the box, or the obscuring of the sides of the box by touching strokes.

This rate was considered too low and a more robust extraction method was developed. This method uses the prominent black horizontal and vertical lines that form part of the miniform layout to locate each miniform in the input image. It then determines the position of the miniform input fields relative to the position of the layout lines. Raw layout lines are located using morphological operations and directional histograms.

Very rarely, layout lines are missed due to noise or fading, or, more frequently, spurious layout lines are detected inside textual or handwritten regions. Another complication is that several different layouts exist for the miniforms. To cope with both problems, a 1D analog of a search-based object recognition algorithm (see, for example, Grimson and Lozano-Perez[4]) is used to recognize layouts in the presence of insertions and deletions.

The recognition algorithm is given as input a set of manually constructed 1D models (relative positions of horizontal layout lines within a particular type of miniform) together with the positions of layout lines (found by morphological processing) and searches for a consistent interpretation allowing for some insertions and deletions.

Once the miniform layout lines have been identified and located, input fields are extracted from positions relative to those layout lines. Some additional space surrounding each box is extracted to allow strokes that touch the box or cross the box margins to be included. An example of extracted fields is shown in Figure 3.

The failure rate for miniform extraction by this method is quite small: out of a sample of 3000 miniforms, 2 miniforms failed to get extracted. These cases were detected automatically, and the corresponding fields were rejected.

5. Box/Underline Removal

As we noted above, each input field is surrounded by a dashed box. This box needs to be removed before segmentation and recognition can take place.

The first step is to identify those regions of the image that contain the dashes making up the box. These regions are found using morphological filtering, selecting small rectangles of approximately the right size that are arranged in long lines. In order to eliminate the bottom and top lines delimiting the handwritten input, within these regions, all vertical stretches of

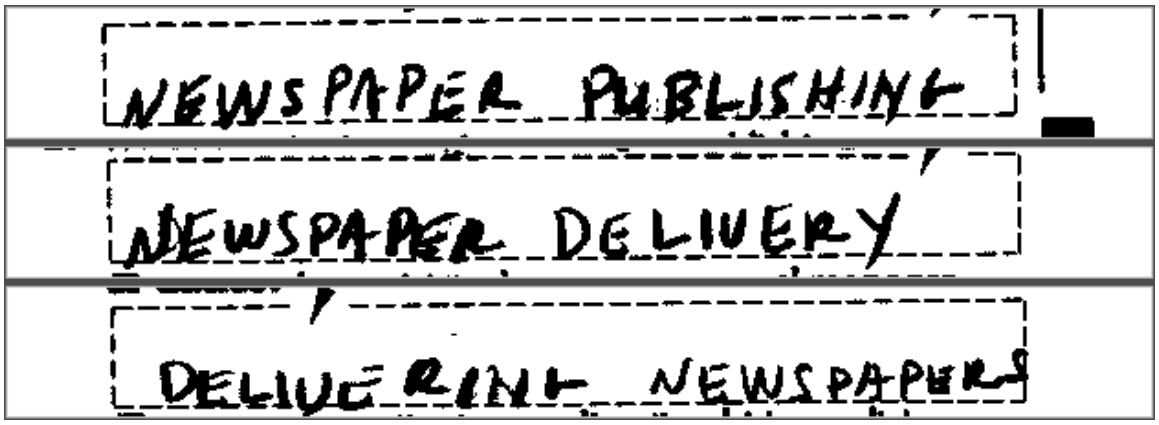

Figure 3: The fields extracted from the miniform shown in Figure 2. Note that they are still slightly skewed, that the dashed box is still present, and that there are arrows protruding into the dashed box.

pixels that are shorter than an empirically determined threshold are deleted. This operation almost completely eliminates parts of the box that do not touch any strokes, while leaving almost all strokes touching or crossing the box intact.

6. Skew and Slant Correction

Skew is the misalignment of the baseline of the handwritten text with respect to the pixel coordinate system of the field image. Slant is the deviation from the vertical of the long down strokes in letters like I, K, or T. Skew correction shears the input image in a direction parallel to the y axis to force the skew of the input to zero. Slant correction shears the input image in a direction parallel to the x axis to force the average slant of the handwritten input to zero.

In many systems, skew and slant correction are carried out for the benefit of the recognizer, to remove transformations of the input under which the identity of characters is largely invariant. In the system described here, skew and slant correction have a more immediate goal, however: they are carried out to simplify the subsequent segmentation stage. They give vertical lines through the image a distinguished status. After skew and slant correction, vertical lines will run parallel to down strokes, and, in the absence of kerning, segmentation cuts between characters will be vertical lines.

Skew and slant correction are each carried out by a generate-and-test procedure. We can think of this as generating different transformed instances of the image subjected to skewing or slanting, and picking the transformation that maximizes an evaluation function. In the actual system, a more efficient algorithm is used to compute the same result.

The evaluation function used in this work consists of the average value of the local maxima in the smoothed horizontal (skew correction) or vertical (slant correction) histogram.

This simple procedure reduces, but does not completely eliminate, variability in skew and slant. A significant fraction of the input strings exhibit non-linear baselines and/or variable slant.

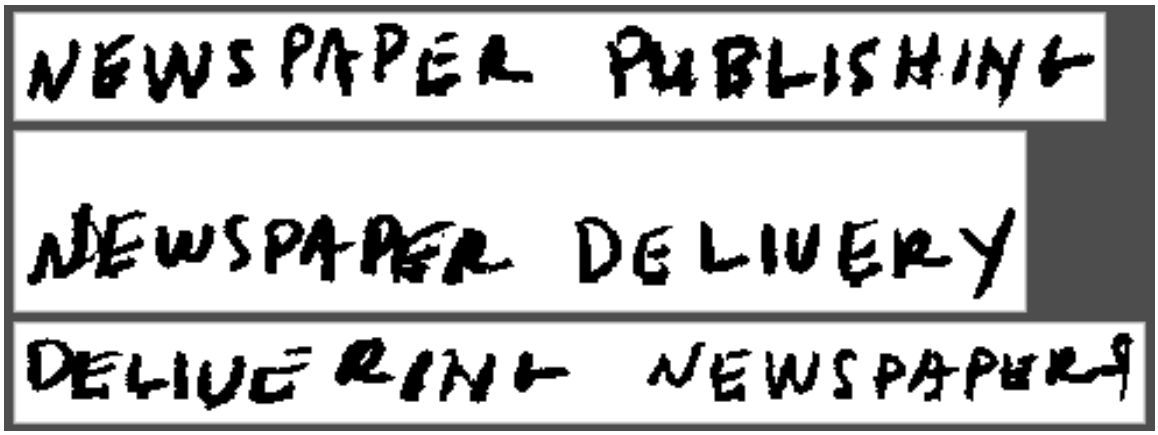

Figure 4: Cleaned-up versions of the fields shown in Figure 3. Note in particular that the writing has been slant corrected and that strokes touching or crossing the bottom of the dashed box have been preserved.

6.1. Removing Spurious Connected Components

We define a *central region* as those parts of the input image that lie between the 20th and 80th percentile of the horizontally projected histogram. Connected components in the input image that overlap the central region are automatically retained. Other connected components are retained only if their bounding boxes overlap significantly one of the connected components overlapping the central region. Connected components that are not retained during those previous two steps are deleted and not considered further by subsequent processing stages.

The result of the previous steps, box removal, skew and slant correction, and connected component analysis is shown in Figure 4.

7. Segmentation

The output of the previous stage is an image that contains mainly a handwritten input string. The goal of the segmentation stage is to determine a collection of character subimages that can then be individually classified.

In idealized handprinted text, each connected component corresponds to a letter (with the exceptions of i and j, which consist of two connected components). Unfortunately, this simple correspondence breaks down for real handwriting.

In many cases, even in "isolated" handprinted character writing styles characters are connected through ink trails or noise. Conversely, characters like E, T, or H often consist of two or more connected components. Because of pen fading, in particular during long, smoothly curved strokes, characters like N and U also can appear separated, i.e., as two or more connected components, in the image. Connected components are therefore subjected both to a grouping process and to a segmentation process.

7.1. Cut Hypotheses

The grouping process based on connected components and proximity described above divides the image of the handwritten input string into a number of groups. But because input characters frequently touch each other, connected components need to be subdivided further by cuts that separate each handwritten character from its neighbors. This is done by computing a set of explicit cuts through the connected components. Each cut is a line going from bottom to top through the image of the handwritten input string. Cuts must satisfy certain geometric regularity conditions and are chosen in a way that makes it likely that they correspond to boundaries separating individual handwritten characters in the input string.

Without taking advantage of character shape models and language models, the validity of many cuts cannot be determined unambiguously at the segmentation stage. A commonly seen example in cursive handwriting is the ambiguity between the single letter d and the pair of letters cl. In handprinted text, another common ambiguity exists between the letter m and the pair of letters rn. Many more such ambiguities exist in printed and cursive handwriting. The goal of the segmentation step is therefore to generate a set of cut hypotheses that is sufficient to cover all reasonable segmentations of the input string.

If we cannot determine the exact set of cuts needed to separate characters from one another, we have to make tradeoffs. Either we include too many cuts (oversegmentation), some of which will divide individual characters, or we include too few cuts (undersegmentaion) so that some character pairs are not separated. Oversegmentation is much less of a problem than undersegmentation. The reason is that oversegmentation leads to the inclusion of some spurious character subimages, but usually does not eliminate correct character subimages from the set of hypotheses. Undersegmentation, however, introduces unrecoverable recognition errors.

7.2. Valley Point Cuts

Two different approaches to generating cut hypotheses were tried. The first method was the method of valley-points (used in several systems, including Kimura *et al.*[5]). While it yielded good results on cursive handwriting in some preliminary experiments, its performance on handprinted text (in particular, upper case handprinted text) was disappointing.

The good performance of valley point segmentation on standard cursive handwriting (copper plate) is easily explained by the way characters are linked using festoon-like strokes, which regularly contain local minima between individual characters.

However, the linking strokes between handprinted characters, in particular uppercase handprinted characters, are more idiosyncratic. Characters are often linked at the top rather than at the bottom. Frequently, characters are even linked via two or three strokes (e.g., the ink of a very closely spaced pair of upper case letters EE, as in Œ, will form a connected region with two holes). If valley points exist at all between such character pairs, they are often in inappropriate places and do not determine a unique cutting path.

Typical examples of touching characters that cannot be separated properly using the valley point method are shown in Figure 5. Here, the FL, EN, EE, and ER pairs cannot be separated properly if valley points are used as cutting points.

Another problem with the valley point segmentation method is that it only indicates a single cutting point on the upper contour of a connected component. In the presence of

Figure 5: An example of a string that is difficult to segment.

multiply connected components or kerning, this leads to ambiguities as to how the component should be cut.

Since the majority of the data used in this work was handprinted rather than cursive (see Table 1), and because of the difficulties of extending a single cutting point into a complete cut, a different method of generating cuts was devised.

7.3. Pre-Stroke Cuts

The basic idea behind the segmentation method used in the system is to cut apart an input string just to the left of every vertical stroke. This approach works very well for letters that have a vertical stroke (possibly curved or slanted) on the left. It works acceptably well with the remaining letters. The only significant exceptions are letters that have a protruding horizontal stroke on the left (J T f j t), which, when cut right before its vertical stroke, will leave behind a small horizontal line with the preceding letter. Usually, this only results in the presence of a harmless, spurious cut.

Two different versions of pre-stroke segmentation were implemented. The first version used only straight, vertical cuts. Strokes were identified as local maxima in a vertical histogram, and stroke boundaries were found as inflection points, i.e., zeros of the second derivative of the smoothed vertical histogram. This method worked quite well for identifying good cutting points, but straight lines failed to cut a significant fraction of character pairs properly. For example, the letter pair Co is frequently linked (and hence represented by only a single connected component), and the letter o is frequently moved partially under the upper part of the letter C, as in Co, a phenomenon that is called *kerning*. Clearly, we cannot choose a single straight vertical line to separate such pairs of letters.

Nevertheless, even this segmentation method gave surprisingly good classification results. It appears that the classifier acquired models for letter variants arising from imprecise segmentation (contextual models would probably result in significant improvements when using straight cuts for segmentation).

In order to cope better with kerning, the system evaluates a large set of curved cuts and selects a small, "optimal" subset. The resulting set of curved cuts allowed most pairs of characters to be separated cleanly, even in the presence of kerning. This approach is somewhat similar to the deflection method[6, 7]. The details of this algorithm will be described elsewhere.

7.4. Character Subimage Hypotheses

The two segmentation stages described above find a set of connected components, and, for each connected component, a set of cuts. For a connected component, a subset of the set

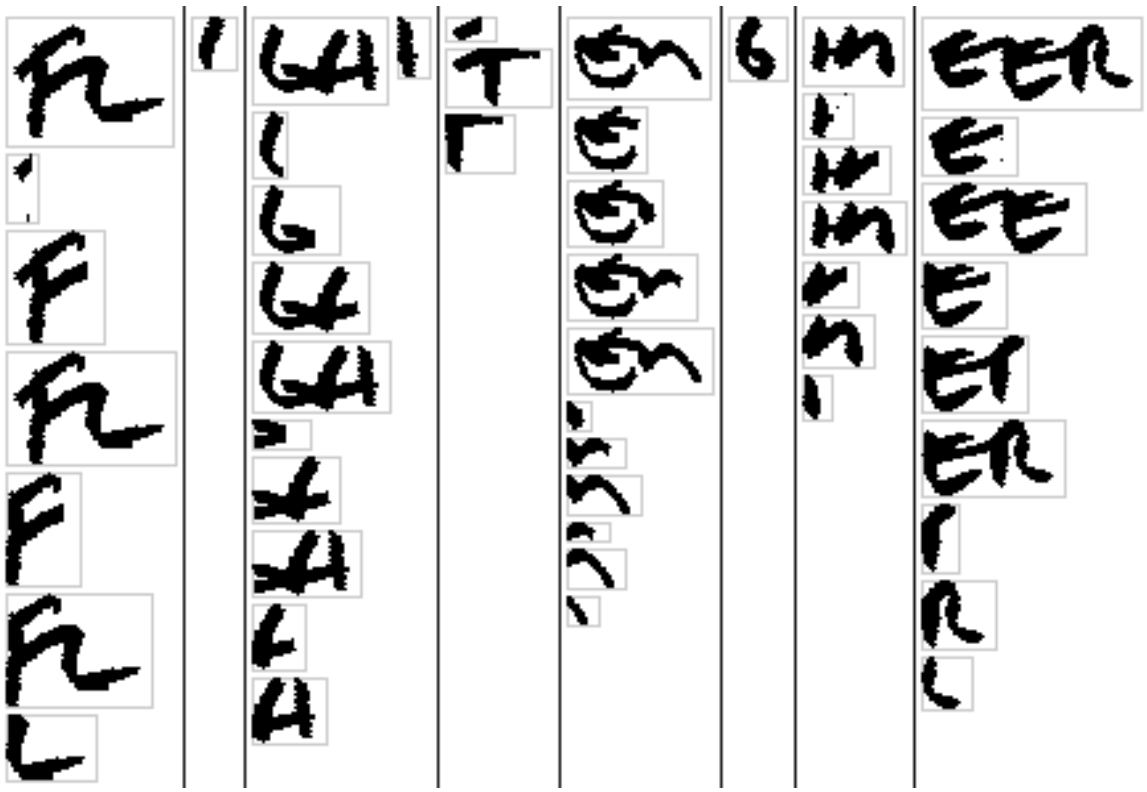

Figure 6: The character subimages corresponding to the segmentation of the string shown in Figure 5.

of cuts defines a partition of that connected component. Each element of such a partition corresponds to a character hypothesis.

The set of all possible partitions of a connected component generated by a set of cuts grows combinatorially in the number of cuts. This could make the recognition of long connected components prohibitively expensive. Fortunately, the number of possible character subimages itself only grows at most as the square of the number N_c of cuts.

However, the set of possible partitions of the input string can be represented compactly as a graph in the following way. We let each cut correspond to a vertex. Each (directed) edge connecting two vertices then represents a character subimage. A segmentation of the input is a simple path through this hypothesis graph, starting at the vertex corresponding to the leftmost cut and ending at the vertex corresponding to the rightmost cut. This structure is very similar to that of a *phone lattice* in speech recognition. A example of a set of character subimages is shown in Figure 6.

But even the extraction, processing, and classification of $\binom{N_c}{2}$ character subimages means a significant computational burden: the number of cuts N_c is usually somewhere between n and $3n$, where n is the length of the character string corresponding to the connected component. In practice, n can easily reach 20 characters, potentially making the recognition of a connected component between 400 and 3600 times slower than the recognition of the same number of isolated characters.

By imposing some additional constraints, we can reduce the number of character subimages greatly and even achieve linear complexity with a small constant factors. In order to do this, we take advantage of two properties of handwritten or handprinted characters.

First, we observe that each of the upper or lower case letters of the Roman alphabet span at most two pre-stroke cuts (the only letters that actually contain two cuts are M, m, W, and w). In principle, therefore, we need not consider any character subimage that spans more than two cuts. Allowing for some error, we might limit number of cuts falling inside a character subimage to three or four. This brings down the number of character subimages that need to be considered to at most $5N_c$.

Second, individual letters satisfy some size constraints. For example, they are very unlikely to be significantly wider than they are tall, and they are very unlikely to be significantly wider than a small multiple of the estimated text height. We need not consider any character subimages not satisfying these size constraints. This further cuts down on the number of character subimages that need to be processed.

Limiting the number of pre-stroke cuts spanned by the character subimage and its length also has another important function. As we have seen, the cost of matching an input image against a dictionary string consists of two parts: a cost computed for the segmentation/partition of the input image into characters, and a cost of making the correspondence between each character subimage and its corresponding character in the dictionary. Both of these costs are estimated as the logarithm of the posterior probability estimated using a MLP classifier.

But the classifier only has a limited amount of training data available, and without additional *a priori* knowledge, in a Bayesian framework, there is effectively a lower bound on the magnitudes of the probabilities it can estimate based on the training data alone. The effect is that even very long connected components, which appear impossible as part of an acceptable segmentation to a human observer in possession of *a priori* knowledge, could not incur a cost of more than around 23.0 nats in the current system. As a consequence, when excessively long character subimages were included in the segmentations in an early version of the system, long connected components would often form part of an optimal match where there was a sequence of degraded or poorly formed characters, since such long connected components incur comparatively small per-character costs. Eliminating very long character subimages from futher consideration essentially forces their prior probability to be zero and thereby avoids this problem.

8. Character Subimage Classification

Each character subimage found by the segmentation stage is individually classified using a MLP (Multi-Layer Perceptron with sigmoidal activation functions). As is well-known, the output of the classifier approximates a conditional probability[8, 9, 10, 11], $P(w_i|x)$

The possible classes $w_i \in \Sigma$ are the 26 letters A through Z (no distinction is made between upper and lower case) and a rejection class. The rejection class contains all non-alphabetic characters and all character subimages that do not represent a complete, individual character. For the recognition of US census forms, the system was not trained to recognize digits or special characters: the number of digits in the training set was too low to warrant inclusion, and special characters were not transcribed, making training difficult.

Figure 7: An example of the feature maps for a handwritten letter T. These feature maps form the input to the MLP.

Ideally, only character subimages belonging to the correct segmentation would be classified as one of the letters A through Z; all character subimages belonging to incorrect segmentations should be assigned to the rejection class by the classifier. Of course, as we observed above, this ideal is not achievable, since the same string frequently allows several plausible segmentations. Such ambiguities must be resolved at a later processing stage on the basis of top-down knowledge.

Training of the MLP was carried out using the backpropagation algorithm with a momentum term[12, 13]. Training was stopped when the cross-validated error did not decrease further.

Input to the classification stage consists of eight normalized feature maps, the first seven of which are 10×10 unit topographic representations of feature maps corresponding to the character subimage. The eighth feature map encodes the ascent, descent, width, height, and center relative to the baseline of the character subimage using a unary code.

The first four feature maps encode the local gradient of the character subimage. Each feature map is maximally sensitive to a particular gradient orientation; response to gradients differing from this preferred orientation decays like a Gaussian.

The next feature map encodes the presence of "holes", i.e., interior regions that are not connected to the background of the character subimage. Such regions occur frequently in letters like O, a, A, or e, and are almost always absent in letters like L, T, or I.

The last two feature maps encode the presence of singular points of the skeleton of the character subimage. The first of the two feature maps encodes endpoints of the skeleton, while the second encodes points where three or four branches of the skeleton meet. The skeleton is computed using a thinning algorithm[14].

Figure 7 shows an example of the input to the MLP for a handwritten letter T. The first four feature maps containing orientation information are clearly visible. The next feature map containing information about interior regions is black, since the letter T has no interior regions. The next two maps show the three endpoints and the point of intersection of the horizontal and vertical strokes respectively. Finally, the last map encodes information about the position and size of the character subimage relative to the rest of the input.

As we noted above, the output of the MLP can be interpreted as an approximation to a set of conditional probabilities. Subsequent processing uses the negative of the logarithm of each output of the MLP; we will refer to this as a "cost". The addition of costs corresponds to the multiplication of probabilities, as in Eq. (5).

9. Hypothesis Graph

The output of the segmentation stage is a collection of character subimages and an associated directed graph. As we observed above, each cut corresponds to a vertex in this graph, and each character subimage corresponds to an edge going from the cut delimiting its left side to the cut delimiting its right side. We can think of this graph equivalently as a Hidden Markov Model (HMM) or a Finite State Machine (FSM).

Each character subimage, and hence each edge of the hypothesis graph, is associated with a cost vector output by the classification stage. Each entry i in this 26 dimensional cost vector can be thought of as describing the cost of interpreting the corresponding character subimage as a character of class i.

The problem of interpreting the input string consists of picking a path through the hypothesis graph that starts at the vertex corresponding to the leftmost cut of the image of the handwritten input string and finishes at the vertex corresponding to the rightmost cut of the image of the handwritten input string. Each edge in such a path corresponds to the interpretation of a character subimage. If we take a HMM or FSM view, we can think of the vertices of the graph as states and its edges as transitions; the costs associated with each character class and edge describe the probabilities of outputting the corresponding symbol.

In practice, the hypothesis graph derived from the segmentation needs to be edited slightly to account for the possible insertion and deletion of character subimages from the input string. For example, a non-negligible fraction of the input images contain spurious markings before and after the actual handwritten string. In addition, some input strings contain insertions of special characters like /, &, and –. The unedited hypothesis graph above would force the interpretation of such extraneous character subimages as characters or as parts of other character subimages, resulting in recognition errors.

If we view the hypothesis graph as a representation of a FSM or HMM, we can overcome this problem by adding ϵ-transitions and wildcard self-loops to each state. In many HMM-based speech recognition, the estimation of the probabilities of such transition is very important, since they represent durational models for phonemes in the input. In this system, costs associated with ϵ-transitions and self-loops were picked on the basis of some simple experimentation. The reason why this seems to be sufficient is that ϵ-transitions and self-loops need to participate only rarely in a match; "durational" models (i.e., character width models) are already implicit in the segmentation and character subimage classification steps.

We have not yet discussed the processing of spaces in the input. They can be used in some of the matching operations. However, because they are often not transcribed, they are handled asymmetrically: a space in a dictionary entry is required to be present in the input, but a space in the input may be ignored. This is handled by allowing all spaces in the hypothesis graph to be skipped at no cost.

Figure 8 illustrates the process by which the raw hypothesis graph is transformed into the edited hypothesis graph. For illustrative purposes, we assume that each edge in the unedited graph is only associated with a single character subimage hypothesis. The unedited graph corresponds only to the string AB_C, where _ denotes a space or blank character, at a total cost of: 1.0 nats + 2.3 nats + 0.7 nats = 4.0 nats. In this example, the editing process has

- added wild-card self-loops with a cost of 15.0 nats,

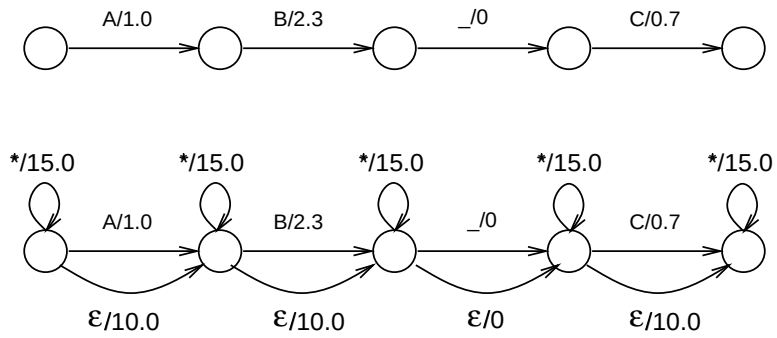

Figure 8: An illustration of the process of editing a hypothesis graph to allow skips, insertions, and the deletions of blanks from the input.

- short-circuited each edge corresponding to a non-blank character with an ϵ-transition at the cost of 10.0 nats, and
- short-circuited the edge corresponding to a blank with an ϵ-transition at a cost of 0.0 nats

This means that the new graph corresponds not only to the original string AB_C at a cost of 4.0 nats, but also to the string ABC at the same cost, or the strings A_C or AC at the cost of 11.7 nats, by taking the ϵ-transition skipping the B.

10. Language Models

In principle, we could simply pick the best path through the hypothesis graph and use that as our transcription of the input string. Unfortunately, recognition performance using such an approach is generally poor. The reason is that most handwritten input is ambiguous. These ambiguities can only be resolved using a language model that restricts the set of paths through the hypothesis graph to those that are compatible with strings given by the language model.

Mathematically, the language model corresponds to the factor $P(W)$ in Eq. (5). A good estimate of $P(W)$ is therefore important for approaching Bayes-optimal performance.

When trying to come up with a language model for this task, the following task-specific phenomena should be kept in mind:

- The handwritten text on the census forms classified by the system was given in response to questions related to the employment of the respondent. Furthermore, only limited space was provided for responses, resulting in responses consisting of usually one or two words. This makes language models for these responses considerably different from language models for general English prose.
- Abbreviations and spelling errors were frequent, and the particular task to be solved and the scoring is based on a letter-accurate transcription of each input field, including all spelling errors present in the handwritten input.

language model	size	coverage
common-phrases	19550	64%
all-words	14432	88%

Table 2: Coverage and size of language models constructed for the recognition system.

- Only very limited amounts of training data (approximately 75000 phrases) were available for the construction of language models.

A number of language models were considered and tried. A more detailed discussion of the different language models and their properties can be found in Breuel[15].

NIST supplied two kinds of data that could be used to build language models. The first kind of data consisted of dictionaries of words and phrases derived from a large set of responses on 1980 and 1990 US Census forms. However, these dictionaries were not provided with frequency information. The second kind of data was the set of transcriptions provided with the binary images of responses on 1990 US Census forms that were provided by NIST for training the handwriting recognition engines. This set of transcriptions was considerably smaller than the set of responses that the NIST dictionaries were based on, but it did allow frequency information to be derived by counting. (The responses on the test set used for obtaining the error rates reported in this paper were of course excluded from the construction of any language model.)

Based on the dictionaries supplied by NIST, a unigram language model (correct-words) was constructed. It consisted of all the correctly spelled words (as determined by UNIX `spell`) contained in the NIST dictionaries. Since probabilities for the individual words could not be estimated from the training data, all words were assigned equal probabilities. Under this language model, the cost, $-\log P(W)$, of a phrase W is therefore simply proportional to the number of words in it.

A phrase probability model (common-phrases) was also constructed. It consisted of all the phrases that occurred more than once in a set of approximately 75000 transcriptions provided with the NIST database. The probability $P(W)$ assigned to a phrase W was simply the number of times that W occurred in the set of transcriptions divided by the total number of transcriptions. Before probability estimation, phrases that occurred only once in the set of transcriptions were deleted since probabilities cannot be estimated reliably for them.

For each language model, a question of particular importance is its coverage, that is, the percentage of actual responses on the US census forms that are assigned non-zero probability by the language model. The recognition rate of a system is limited by its coverage, since no string that has been assigned zero probability can ever be returned by the language model. The coverages of the various language models are shown in Table 2.

11. Matching

The purpose of matching is to evaluate $P(W|x)$ for each string W and find the string with the largest posterior probability. Complicating this computation is the fact that multiple

paths through the hypothesis graph can correspond to the same string W. As is common in speech recognition, we use the Viterbi approximation; that is, we use the best path through the hypothesis graph as an approximation to the sum of the probabilities of all paths through the hypothesis graph that correspond to a string W.

Each path through the hypothesis graph corresponds to a string $W \in \Sigma^*$ and an associated cost $c_h(W)$ obtained by summing all the costs along the edges traversed by the path. Here, Σ is the alphabet used by the system, i.e., the set of symbols A...Z and the blank character _. If some string W does not correspond to any path in the hypothesis graph, we assign an infinite cost to it. Similarly, the language model assigns a cost $c_l(W)$ to every string W.

The matching stage finds a string $W \in \Sigma^*$ that has minimum total cost $c_h(W) + c_l(W)$. It is not difficult to see that this corresponds to minimizing Eq. (5) over all strings and all segmentations, under the Viterbi approximation.

Finding the best path through the hypothesis graph constrained by the dictionary is carried out using a Viterbi algorithm. To speed up matching, the dictionary is itself represented as a graph corresponding to an optimized form of the trie representing the words or phrases in the dictionary.

12. Hypothesis Integration

Since the system as described uses two different language models, the question remains of how to integrate the two hypotheses found by matching against the two dictionaries. A number of methods have been proposed in the literature[16, 17, 18, 19].

A related problem is that of deciding whether a hypothesis should be accepted at all. It could be argued that if our character models and segmentation models give statistically correct answers, rejection could be decided purely on the basis of a comparison of the posterior probabilities computed for each match. But we already know that the statistical information used by the system is only an approximation. Some aspects of the input could not be modeled accurately given the training data provided. Hence, rejection based purely on the computed costs is likely to be suboptimal.

For example, as we noted above, non-alphanumeric characters in the input are not transcribed. Similarly, the dictionaries that were provided with the training data did not contain any frequency information. Such information could only be approximated crudely by counting phrases in the much smaller set of actual transcriptions included with the NIST training data.

For both these reasons, the system includes a final processing stage that attempts to decide whether to accept the hypothesis based on the phrase dictionary, to accept the hypothesis based on the word dictionary, or to reject both.

Hypothesis integration and rejection is carried out using a decision tree classifier. The decision tree classifier was constructed using the CART method[20]. The input features to the decision tree classifier consist of properties of the input image, like pixel density, bounding box, text height, etc., as well as properties of the word- and phrase-based matches, such as length, per-character cost, etc.

The decision tree classifier can make four possible decisions: (1) reject both the phrase-based hypothesis and the word-based hypothesis, (2) accept the phrase-based hypothesis, (3) accept the word-based hypothesis, or (4) accept both hypotheses (clearly, the last case is

actual/predicted	reject both	accept phrase	accept word	accept both
reject both	29.99	2.90	0.40	2.36
accept phrase	4.45	11.05	0.40	0.00
accept word	4.99	0.81	2.90	0.00
accept both	0.20	0.00	0.00	39.56
total	39.62	14.76	3.71	41.91

Table 3: A confusion matrix for the decision tree post-processing stage (all entries are in percent). Note that the total percentage of fields, 39.62%, that are predicted to be rejected is the rejection rate for this confusion matrix. This is the rejection rate that the decision tree is intrinsically optimized for. Rejection at other rejection rates is done using the estimated terminal probabilities associated with the leaves of the decision tree.

only possible if both hypotheses are equal). The CART method optimizes the zero-one loss function for these decisions. This yields a decision tree that is suitable for recognition at a certain *intrinsic rejection rate*. In the case of the present system, this intrinsic rejection rate is approximately 40% (see Table 3).

However, depending on the application for the handwriting recognition system, we may wish to choose rejection rates that differ from this intrinsic rejection rate. It is possible to modify the CART method to accomplish this. However, in order to rely on existing software, a simpler approach was chosen for the current system. Associated with each terminal node in the decision tree is a set of estimates of posterior probabilities for each of the four different decisions. Based on these posterior probabilities, the system picks the more probable hypothesis among the word-based recognizer and the phrase-based recognizer, and it computes a probability of misclassification. The technical details of this are straighforward and left to the reader to work out. The probability of misclassification can be used to give a linear order to the set of all hypotheses. Now, in order to achieve, say, a 50% rejection rate, we pick those 50% of all hypotheses that are associated with the lowest estimated probabilities of misclassification.

13. Bootstrapping and Training

The MLP classifier that performs character subimage classification requires training data. Except for the decision tree classifier, all the other parameters of the system have been set either by estimating probabilities using counting (e.g., phrase priors and character priors), by simple statistics on geometric measurements of the input, or by inspection.

Since the NIST database contains only transcriptions but no alignment information, training had to proceed in two phases. In the first phase, manually segmented and aligned input was used to train a crude MLP character subimage classifier. This initial classifier was then used to segment and align input strings, and character subimages from the resulting alignment were used to train better classifiers for character subimages in a process similar to embedded training in speech recognition and the expectation-maximization (EM) algorithm

used in statistics.

For the initial training step, 1000 input fields from the `d00` directory of the NIST database segmented manually, resulting in approximately 14000 character subimages. Manual segmentation was accomplished within a few hours using specialized tools that themselves attempted an automatic segmentation and alignment based on some simple heuristics. In fact, using those tools, it would not have been difficult so segment the complete database over the span of a few days. However, this approach was not taken, since the character subimage classifier should be trained on character subimages obtained using the segmentation boundaries returned by the segmentation method actually used in the system, not some manual approximation.

For subsequent training steps, recognition was essentially carried out as described above, but using a dictionary consisting only of a single phrase, the correct answer. This yields a set of aligned character subimages. A heuristically chosen subset of these character subimages were used as training examples for the MLP classifier. Among the remaining character subimages, which did not participate in the best match, a subset was selected heuristically that was used as training examples for segments that did not form part of the correct segmentation. Subdirectories `d01` through `d05` of the NIST database were used for this training step, resulting in training of the MLP on approximately 117000 character subimages.

Subdirectories `d06` through `d10` of the NIST database were matched recognized with the full system, using either the phrase-based language model or the word-based language model. The output from these classifications were used to train the decision tree model for the hypothesis integration and rejection step.

14. Results

Subdirectory `d11` of the NIST database was never used for any training step, and was reserved instead for evaluating the performance of the system. All error rates and results presented below refer to the 1500 field images in this subdirectory.

Evaluating Error Rates The criterion used for evaluating the performance of the system is the percentage of fields classified correctly. A field was counted as classified correctly if the system returned a string that was an exact match against the transcription supplied with the NIST database, allowing for the insertion or deletion of spaces. This means, in particular, that any misspelling in the handwritten input must be recognized and transcribed by the system.

This exact-match criterion is rather pessimistic, in the sense that less stringent criteria are usually sufficient for actual applications. Therefore, an error rate of 5% under the exact-match criterion may represent nearly perfect performance for an application requiring, say, the classification of types of occupation into broader classes.

On the other hand, the evaluation criterion used in the NIST competition for comparing the performance of handwriting recognizers on the NIST database is slightly stricter, requiring exact matches also for spaces in the input[3]. This latter criterion was not used in the present work, since upon examination, almost all of the errors in spacing were found to be due to plausible insertions and deletions of spaces, or to spaces in the input that were not transcribed

rejected	system	phrase-based	word-based
75%	1.9%	1.9%	5.1%
50%	6.1%	8.9%	19.8%
25%	21.1%	27.0%	36.1%
0%	36.7%	42.0%	50.3%
	(n=1500)		

Table 4: Field error rates under an exact-match criterion after rejecting different fractions of the input. For example, 6.1% error at 50% rejection means that 750 input fields are rejected and 46 out of the remaining 750 accepted input fields are misclassified. The columns "phrase-based" and "word-based" refer to the performance of the subsystems that either only use the phrase-based dictionary or the word-based dictionary. Rejection for the word-based and phrase-based subsystems is based on the per-character cost of the answer, including the costs associated with the dictionary. The column "system" refers to the performance of the complete system, including decision-tree based integration and rejection.

(e.g., "DAY CARE" vs. "DAYCARE"). The difference in the 50% rejection error rate for the system if spaces are required to match exactly is approximately 1.5%.

In the evaluation, we consider the performance allowing various fractions of the input to be rejected. In actual applications, even a system that only recognizes, say, 50% of the inputs with high recognition rates may be economically interesting.

Intrinsic Errors Error rates for the complete system and the phrase-based and word-based subsystems are shown in Table 4. It is important to understand that there are some intrinsic limitations to the recognition rates achievable by the current system:

- A total of 12% of the input fields are very difficult to recognize, being either cursive or faded, consisting of two lines, or containing struck out characters (see Table 1). Since the system was not trained on cursive writing, and since it has no special mechanisms for coping with two-line input or writing falling outside the printed box, most of these input fields will be errors or rejects.

- The phrase-based recognizer has relatively low coverage (64%). However, for those phrases covered by its language model, the language model and its assignment of costs to strings is quite accurate.

- The word-based recognizer has relatively high coverage (88%). However, its language model is quite rudimentary, taking into account no grammatical or semantic constraints, and using a fixed cost per word to assign costs to strings.

- Other data shows that the errors of the phrase-based and the word-based recognizer are significantly correlated, so that the recognition rate of the word based recognizer on those input fields whose transcription is not covered by the phrase-based language model is significantly higher than on the database as a whole.

rejected	system	phrase-based	word-based
75%	1.5%	0.0%	3.7%
50%	3.4%	1.4%	10.6%
25%	12.6%	2.6%	21.0%
0%	29.5%	12.4%	36.5%
	(n=1311/87%)	(n=983/66%)	(n=1311/87%)

Table 5: This table is analogous to Table 4, but recognition rates are only reported for inputs whose transcription is contained in the language model for the given (sub-)system. The last row gives the number of phrases contained in the language model and the corresponding coverage of the language model relative to the complete set of 1500 transcriptions.

The best we can hope for is that if we consider the error rate of the system allowing for a certain fraction of the inputs to be rejected, the system will reject those inputs that correspond to strings that it can intrinsically not recognize.

Phrase-Based Recognizer Let us first consider the performance of the phrase-based recognizer. Its error rate at 0% rejection is 44%. An error rate of 36% is intrinsic, however, since 36% of the input strings have a transcription that is not contained in the phrase dictionary; the remaining difference of 8% is probably largely due to fields that are intrinsically difficult or impossible to recognize for the current system. We can see this when we look at Table 5, which evaluates the performance of each recognizer only on the set of phrases actually covered by its language model. At 0% rejection, the error rate of the phrase-based recognizer is 12.4%, which is nearly identical to the 12% of inputs that are written in a style that we know cannot be recognized by the system. When we allow the system to reject 25% of the input, the error rate drops to 2.6%, meaning that most of the inputs that are difficult or impossible to recognize are actually rejected.

Word-Based Recognizer The performance of the word-based recognizer is clearly much worse. For 25% rejection and strings contained in their respective language models, the error rate of the word-based recognizer of 21.0% is nearly 8 times as large as the error rate of 2.6% of the phrase-based recognizer. An inspection of the errors suggests that a large fraction of the incorrect hypotheses are grammatically or semantically implausible, and that therefore a more restrictive language model with similar coverage could be found that would greatly reduce the error rate of the word-based recognizer. Unfortunately, the construction of such a model would probably require a corpus of responses that is significantly larger than is available.

However, despite its relatively high error rate, combining the results of the word-based recognizer with the results of the phrase-based recognizer results in some reduction in the overall error rate under most conditions. We can see this by comparing the error rates in Table 4. For example, at 50% rejection, the system error rate is reduced by 31% over the phrase-based error rate (8.9% to 6.1%).

System Error Rate We can understand the effect of the decision-tree based integration and rejection step better if we look at a confusion matrix of the different possible outputs of the decision tree classifier. This is shown in Table 3. This table shows the way the hypotheses of the phrase-based and the word-based recognizers are integrated at the intrinsic rejection rate of the decision tree of 39.62%.

We notice that the word-based and the phrase-based recognizer agree on approximately 40% of the inputs. It appears that when the word-based and the phrase-based recognizer are in agreement, chances are good that the common hypothesis is actually correct. This integration of evidence from two slightly different sources alone probably contributes significantly to the overall reduction in error rate when the decision tree is used for integrating hypotheses.

The phrase-based recognizer alone contributes 11.05% to the correctly recognized input strings, and the word-based recognizer contributes another 2.90%. However, the decision tree does not appear to be very efficient in taking advantage of the hypotheses where only the word-based or only the phrase-based recognizer has the correct answer. For example, while 2.90% of the correct word-based hypotheses are actually used, another 4.99% are rejected unnecessarily. Therefore, by improving the decision tree stage of the system, we might be able to gain noticeable decreases in either rejection rates or error rates.

Reducing the Error Rate As we noted above, if we want to achieve low error rates, we have to live with a certain amount of rejection. The reasons are that dictionaries have limited coverage, and that some input fields are simply not recognizable. In order to reduce the error rate at 50% rejection, we can use two different strategies: decrease the number of correct hypotheses that are rejected, or reduce the number of incorrect hypotheses that are generated.

That is, first, we can try to distribute hypotheses better between the rejected set and the accepted set; the ideal is to have only incorrect hypotheses in the rejected set and to have only correct hypotheses in the accepted set. Rejection is generally based on the assignment of a numerical "quality of match" or "confidence" in each of the matches. As we saw above, in this system, this confidence is based on the estimated probability of misclassification derived from the terminal probabilities in the decision tree. These estimates are in turn derived primarily from the per-character cost of the match of the hypothesis string against the input field. An examination of correct hypotheses that are rejected by the system shows that rejection is often due to the use of unusually formed characters, uncommon writing styles, or a strongly linked writing style. These result in high costs for individual characters and for the segmentation. Improving the confidence of the system into these hypotheses probably requires increasing the training set for the MLP classifier.

Another approach, perhaps the more straightforward one, to reducing the error rate is to reduce the number of incorrect hypotheses returned by the recognizer. We will discuss this in the next paragraph.

Principal Causes of Incorrect Hypotheses To get some idea of what the sources of incorrect hypotheses are, the 46 input images (corresponding to the 6.1% error rate) that were misrecognized by the system at 50% rejection rate were examined manually and the likely causes of misclassification were guessed. The results of this work are shown in Table 6.

Error Rate	Cause
2.0%	The cost assigned to an incorrect interpretation of a character subimage is too low or the cost assigned to a correct interpretation of a character subimage is too high.
1.6%	The correct response was not in the dictionary; the system returned a close approximation from the dictionary.
0.8%	The handwritten string went past the right end of the input box and was truncated during pre-processing; the system returned a close approximation to the truncated string.
0.5%	The handwritten input consisted of two lines, one of which was (nearly) eliminated during pre-processing; the system returned a close approximation to the remaining line of text.
0.3%	The system returned a correct interpretation of the input that differed from the actual transcription.
0.3%	Spurious markings in the image were transcribed as a character (e.g., a trailing smudge as a plural S).
0.5%	Other.

Table 6: An analysis of the causes of misclassifications at 50% rejection. As we can see from Table 4, at 50% rejection, 93.9% of the input are classified correctly, 6.1% (46/750) fail to be classified correctly.

Nearly one third, or 2.0%, of the 6.1% error rate appear to be due to misclassifications of individual characters. In several cases, this is due to variants of characters that the system has not been trained on or that are intrinsically difficult to recognize. In some other cases, the system fails to assign a high segmentation cost to certain pairs of characters that vaguely ressemble another character; for example, in certain writing styles, the sequence nt can appear to the character recognizer like the single character m and will not necessarily incur a high cost for being mis-segmented.

In most of these cases, the distinctions that the character subimage recognizer fails to make hinge on the presence of small gaps or protrusions or other subtle differences in shape. Using larger training sets would probably improve the performance of the system significantly.

About one quarter, or 1.6%, of the 6.1% error rate are due to inputs not represented by either language model. Because we are looking at errors among those hypotheses retained at 50% rejection, the system had high confidence in these answers, and the difference between the hypothesis and the actual input should not be large. This is indeed what we find. An example of a typical error is returning the hypothesis "ADMINISTRATION" for the handwritten, misspelled input "ADMINSTRATION".

About another quarter, 1.3%, of the error rate is due to errors in pre-processing that resulted in the image of a partial or truncated image to the recognizer. Such errors are either due to truncating the input on the right when writing falls outside the box, or removing one of the two lines of a two-line input field. These errors are avoidable by heuristic methods that detect two-line input and writing that falls outside the box directly. A better approach

might be to design input forms in such a way that such inputs do not occur in the first place.

Throughput For many applications, it is not only important that the system returns correct results, but also operates fast. The current system is designed for flexibility and extensibility. It is composed of a number of separate UNIX programs that are connected via pipes and intermediate files. This results in significant file I/O overhead and the costly redundant computation of some intermediate results (e.g., the Gaussian-convolved input image).

Keeping these caveats in mind, here are some typical performance figures on a SPARCstation ELC (a low-end SPARCstation machine benchmarked at 20 SPECmarks, 21 MIPS, and 3 MFLOPS). Forms segmentation takes approximately 6 seconds per input field found. Box removal, skew correction, and slant correction take about 10-20 seconds per field. The recognition algorithm itself, including segmentation and matching against a language model, takes an average of 60 seconds per field (90% of all input fields are segmented and matched in less than 120 seconds, and the longest processing time among the 1500 input fields of the d11 subdirectory of the NIST database was 170 seconds). Hypothesis integration takes less than a second.

15. Discussion

The system described in this paper is using a segmental approach to off-line handwriting recognition. That is, it is based on a segmentation stage that precedes recognition and generates a relatively small set of candidates of individual characters. While some systems have used more speech-like (HMM-based) approaches to off-line handwriting recognition[21, 22], segmental approaches seem to be popular for this task[23, 24, 25, 5].

In speech recognition as well as on-line cursive handwriting recognition, segmental approaches seem to be less popular (with some exceptions[1, 10]). It is difficult to say at this point whether these differences are due to genuine differences in the respective domains, or whether they are grounded in the history of the fields.

One of the most important questions for any approach to a real-world pattern recognition system is how well it performs relative to other systems. A comparison of the system described here with other systems reported in the literature is difficult because the domains are substantially different: many other systems for handwritten text recognition have worked in the domain of postal address recognition, in which writing styles, language models, scoring, and image quality are all very different compared to the census form recognition task. For example, language models tend to consist of comparatively small, closed dictionaries, and scoring is based on whether the semantically correct answer is retrieved, not on spelling.

Keeping this in mind, Kimura *et al.*[5] report performance on a system for the recognition of handwritten addresses. They use closed dictionaries containing up to 1000 words. Their database probably contains writing styles in a distribution similar to those shown under "CEDAR" in Table 1. On this database, their first system achieved an error rate of 19.10%. That has been improved to 8.51% recently for a dictionary of size 1000. Perhaps the closest figure for comparison in the current system is the error rate of 12.4% at 0% rejection for the phrase-based recognizer applied to phrases contained in the phrase dictionary (a dictionary that is about 20 times as large).

Giloux *et al.*[22] describe a system for the recognition of handwritten amounts based on Hidden-Markov Models (HMMs). As we mentioned above, this system uses an approach to segmentation and recognition based on "pseudoletters". The word-recognition error rate for a vocabulary of size 27 achieved using this approach is 21%. On an city name recognition task, the error rate for a 100 phrase vocabulary was 22.4%. Because relatively small training sets were used, improvements of these rates are to be expected.

Of course, the best comparison is with other systems on the same task and test set. This was the purpose of the 2nd Census OCR Conference[3]. The official report has not yet become available from NIST, but preliminary results distributed at the conference show that the system described in this paper achieved recognition rates among the highest of systems participating in the conference.

Improvements in the recognition rates of the system may come from algorithmic improvements. For example, recent developments in language modeling[26] may allow us to build better language models from small corpora. Improvements may also be possible in the neural-network based recognition of segments and in the modeling of contextual dependencies.

Considerable improvements in performance are possible without even modifying the system itself. Experience with the system suggests that simply having larger corpora for building language models and more training data available will improve recognition rates signficiantly. Also, the task itself can be modified with little effort: forms layout and forms scanning can be improved considerably to reduce effects such as incomplete forms removal and fading. Furthermore, the evaluation criteria used to assess the performance of the system should be adapted to the task, rather than being based on the somewhat arbitrary and very strict measure of letter accurate transcriptions.

16. References

[1] Hon C. Leung, I. Lee Hetherington, and Victor W. Zue. Speech Recognition using Stochastic Explicit-Segment Modeling. In *EUROSPEECH 91. 2nd European Conference on Speech Communication and Technology Proceedings*, Genova, Italy, 1991. Instituto Int. Comunicazioni.

[2] Thomas M. Breuel. A system for the off-line recognition of handwritten text. Technical Report 94-02, IDIAP, Martigny, Switzerland, 1994. submitted to ICPR'94.

[3] NIST. The Second Census Optical Character Recognition System Conference. Technical report, U.S. Department of Commerce, National Institute of Standards, 1994. in preparation.

[4] W. Eric L. Grimson and Tomas Lozano-Perez. Localizing Overlapping Parts by Searching the Interpretation Tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(4):469–482, July 1987.

[5] F. Kimura, M. Shridhar, and Z. Chen. Improvements of a Lexicon Directed Algorithm for Recognition of Unconstrained Handwritten Words. In *International Conference on Document Analysis and Recognition*, pages 18–22. IEEE Computer Society Press, 1993.

[6] M. Shridhar and A. Badreldin. Recognition of isolated and simply connected handwritten numerals. *Pattern Recognition*, 19(1):1–12, 1986.

[7] M. Shridhar and A. Badreldin. Context-directed segmentation algorithm for handwritten numeral strings. *Image and Vision Computing*, 5(1):3–9, 1987.

[8] H. Bourlard and C. J. Wellekens. Links between Markov models and multilayer perceptrons. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 502–510, San Mateo, CA, 1989. Morgan Kaufmann.

[9] J. S. Bridle. Training stochastic model algorithms as networks can lead to maximum mutual information estimation of parameters. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 211–271, San Mateo, CA, 1990. Morgan Kaufmann.

[10] Herve Bourlard, Nelson Morgan, and Steve Renals. Neural Nets and Hidden Markov Models: Review and Generalizations. *Speech Communication*, 11:237–246, 1992.

[11] Steve Renals and Nelson Morgan. Connectionist Probability Estimation in HMM Speech Recognition. Technical Report TR-92-081, International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA 94704, USA, December 1992.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(9):533–536, October 1986.

[13] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[14] T. Pavlidis. *Algorithms for Graphics and Image Processing.* Computer Science Press, Rockville, MD, 1982.

[15] Thomas M. Breuel. Language modeling for a real-world handwriting recognition task. In L. Evett and T. Rose, editors, *AISB Workshop on Computational Linguistics for Speech and Handwriting.* The Society for the Study of Artificial Intelligence and Simulation of Behaviour, U. of Leeds, School of Computer Studies, U.K., 1994.

[16] David H. Wolpert. Stacked Generalization. Technical Report LA-UR-90-3460, LANL, Los Alamos, NM 87545, USA, 1990.

[17] Tin Kam Ho. *A Theory of Multiple Classifier Systems and Its Application to Visual Word Recognition.* PhD thesis, Department of Computer Science, State University of New York at Buffalo, Buffalo, New York 14260, USA, 1992.

[18] Michael Peter Perrone. *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization.* PhD thesis, Brown University, 1993.

[19] Thomas M. Breuel. Recognition of Handprinted Digits using Optimal Bounded Error Matching. In *International Conference on Document Analysis and Retrieval (ICDAR), Tsukuba Science City, Japan*, 1993.

[20] L. Breiman et al. *Classification and Regression Trees.* The Wadsworth statistics/probability series. Wadsworth, Belmont, CA, 1984.

[21] T. Caesar, J. M. Gloger, and E. Mandler. Preprocessing and Feature Extraction for a Handwriting Recognition System. In *International Conference on Document Analysis and Recognition*, pages 408–411. IEEE Computer Society Press, 1993.

[22] M. Giloux, M. Leroux, and J-M. Bertille. Strategies for Handwritten Word Recognition using Hidden Markov Models. In *International Conference on Document Analysis and Recognition*, pages 299–304. IEEE Computer Society Press, 1993.

[23] S. Edelman, S. Ullman, and T. Flash. Reading cursive handwriting by alignment of letter prototypes. *International Journal of Computer Vision*, 5:303–331, 1990.

[24] O. Matan, C. J. C. Burges, Y. LeCun, and J. S. Denker. Multi-Digit Recognition using a Space Displacement Neural Network. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan Kaufmann, 1992.

[25] C. J. C. Burges, J. I. Ben, J. S. Denker, Y. LeCun, and C. R. Nohl. Off Line Recognition of Handwritten Postal Words using Neural Networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4), 1993.

[26] Hermann Ney, Ute Essen, and Reinhard Kneser. On structuring probabilistic dependences in stochastic language modelling. *Computer Speech and Language*, 8:1–38, 1994.