Fast Two–Level HMM Decoding Algorithm for Large Vocabulary Handwriting Recognition

Alessandro L. Koerich
Department of Computer Science, Pontifical Catholic University of Paraná
R. Imaculada Conceição, 1155, Curitiba, PR, 80215–901, Brazil
alekoe@ppgia.pucpr.br

Robert Sabourin
Departement de Génie de la Production Automatisée, École de Technologie Supérieure
1100, rue Notre–Dame Ouest, Montreal, QC, H3C 1K8, Canada
robert.sabourin@etsmtl.ca

Ching Y. Suen
Centre for Pattern Recognition and Machine Intelligence, Concordia University
1445, Maisonneuve Blvd. West, Montreal, QC, H3G 1M8, Canada
suen@cenparmi.concordia.ca

Abstract

To support large vocabulary handwriting recognition in standard computer platforms, a fast algorithm for hidden Markov model alignment is necessary. To address this problem, we propose a non–heuristic fast decoding algorithm which is based on hidden Markov model representation of characters. The decoding algorithm breaks up the computation of word likelihoods into two levels: state level and character level. Given an observation sequence, the two level decoding enables the reuse of character likelihoods to decode all words in the lexicon, avoiding repeated computation of state sequences. In an 80,000–word recognition task, the proposed decoding algorithm is about 15 times faster than a conventional Viterbi algorithm, while maintaining the same recognition accuracy.

1 Introduction

Considerable progress has been made in handwriting recognition over the last few years [10]. Many systems are already available, however they usually work on well–defined application environments, such as ZIP code recognition [1, 7, 9] and bankcheck processing [3, 4, 8], where a number of constraints can be imposed to facilitate the recognition task such as the restricted size of the vocabulary or the reduced number of classes [5, 6, 10]. Most of the research effort on handwriting recognition has primarily been in improving recognition accuracy of such systems. The simple extension of these techniques used in this small–scale but complex problems to deal with large vocabularies is not efficient. While recognition rates as high as 90% are achieved for vocabularies of hundreds of words, such an accuracy drops to less than 80% for vocabularies of ten thousands of words [5, 6]. The negative effects are also observed on the recognition speed. While the recognition time is on the order of tens of seconds for small vocabularies, it increases to few minutes for very large vocabularies.

In order to be practically useful, large vocabulary handwriting recognition systems have to be efficient in their usage of computational resources and deliver reasonable recognition accuracy as well. Therefore, there are two basic problems in large vocabulary handwriting recognition: recognition speed and recognition accuracy. Recognition speed is defined as the time required to recognize an input word and recognition accuracy is defined as the percentage of input words correctly recognized among all input words presented to the recognizer. However, these two aspects of performance are in mutual conflict. It is relatively easy to improve recognition speed while trading away some accuracy and vice versa.

Speed concerns are related to the decoding algorithms as well as to the dimension of many parameters (vocabulary size, models, length of the feature vector, etc) [5]. Dynamic programming (DP) methods are the search strategies

Proceedings of the 9th Int'l Workshop on Frontiers in Handwriting Recognition (IWFHR-9 2004)
0-7695-2187-8/04 $20.00 © 2004 IEEE

used more often in small and medium vocabulary applications. Although, calculating the probabilities in this manner is computationally expensive, particularly with large models or long observation sequences. Other search techniques such as A*, beam search, and multi–pass have not been used widely in handwriting recognition. Because they are faster, generally they are less accurate, providing sub–optimal solutions [5]. Most of the techniques that have been used are based on pruning mechanisms that attempt to reduce the number of word hypotheses prior to the recognition [5].

The aim of this paper is to introduce a novel decoding algorithm to speedup the recognition process while maintaining the recognition accuracy. The idea is to take into account the particularities of the architecture of the hidden Markov models (HMM), feature extraction, segmentation and lexicon–driven recognition to eliminate the repeated computation steps and develop fast search strategies.

This paper is organized as follows. Section 2 presents the modeling of characters by hidden Markov models. Section 3 states the recognition problem and presents the fast two–level HMM decoding algorithm. Section 4 presents the experimental results of using the proposed algorithm in the recognition of unconstrained handwritten words considering different sizes of vocabularies. The conclusions of the paper are presented in the last section.

2 HMM Modeling

We assume that the front–end parameterization of the recognition system provides high–level features in the form of a sequence of observations. After a preprocessing step, a segmentation–recognition strategy is used to loosely segment (oversegmented) words into sub–word units (characters or pseudo–characters). These sub–word units are modeled in a probabilistic framework by elementary HMMs. The sequence of segments obtained by the segmentation process is transformed into a sequence of symbols by considering two sets of features where the first feature set is based on global features, namely loops, ascenders, and descenders and the second feature set is based on the analysis of the bidimensional contour transition histogram of each segment in the horizontal and vertical directions [2]. There are also five segmentation features that try to reflect the way segments are linked together.

We assume that observations are produced by transitions rather than by states. The character models use discrete HMMs where transitions with no output are also incorporated into the model. The compact notation is $\lambda = \{A, A', \Pi\}$, where $A = \{a_{ij}^z\}$ is the probability distribution associated with transitions from state s_i to state s_j and at the same time producing observation symbol z, $A' = \{a_{ij}'^{\Phi}\}$ is the probability distribution associated with null transitions from state s_i to state s_j and at the same time producing null observation symbol Φ, and $\Pi = \{\pi_i\}$ is the initial state distribution. Note that a_{ij}^z and $a_{ij}'^{\Phi}$ obey the stochastic constraint:

$$\sum_{j=1}^{N}[a_{ij}'^{\Phi} + \sum_{z=1}^{M} a_{ij}^z] = 1 \qquad i = 1, 2, \ldots, N \tag{1}$$

Considering a discrete symbol observation with M symbols, a character HMM where the number of states is denoted as N, and a recognition vocabulary represented by a lexicon $\mathcal{L}$ which contains V words in which the average length is L characters, word models, denoted as $\hat{\lambda}$, regarded as a "super–HMM" can be built by the concatenation of L sub–word HMMs, i.e.:

$$\hat{\lambda} = \lambda_1 \oplus \lambda_2 \oplus \ldots \oplus \lambda_l \oplus \ldots \oplus \lambda_L \tag{2}$$

The goal of the training phase is to estimate the best parameter values of the character models, say A and A' for all models λ, given a set of training examples and their associated word labels. Since the exact orthographic transcription of each training word image is available, the word model, denoted as $\hat{\lambda}$, is made up of the concatenation of the appropriate character models (Equation 2), where L is the number of characters that form a word. In such a scheme, the final state of an HMM becomes the initial state of the next one, and so on. A variant of the Baum–Welch algorithm is used for training in which the segments produced by the segmentation algorithm need not be manually labeled [2].

3 Recognition

The basic problem in large vocabulary handwriting recognition is given a word to recognize represented by a sequence of observations $o_1^T = (o_1 o_2 \ldots o_T)$ where T is the number of observations in the sequence, and a recognition vocabulary represented by $\mathcal{L}$ corresponding to V unique words, find the word $w \in \mathcal{L}$ that best matches to the input pattern. The standard approach is to assume a simple probabilistic model of handwriting production whereby a specified word, w, produces an observation sequence o_1^T with probability $P(w|o_1^T)$. The goal is then to decode the word, based on the observation sequence, so that the decoded word has the maximum *a posteriori* (MAP) probability, i.e.:

$$\hat{w} \ni P(\hat{w}|o_1^T) = \max_{w \in \mathcal{L}} P(w|o_1^T) \tag{3}$$

The *a posteriori* probability of a word w can be rewritten using Bayes' rule:

$$P(\hat{w}|o_1^T) = \max_{w \in \mathcal{L}} P(o_1^T|w)P(w) \tag{4}$$

where $P(w)$ is the prior probability of the word occurring, which depends on the vocabulary used, and it is determined by frequency counts in the training dataset. The way we compute $P(o_1^T|w)$ for large vocabularies is to build statistical models for sub–word units (characters) in an HMM framework, build up word models constrained to spellings in a lexicon, and then evaluate the model probabilities via standard concatenation methods. However, in HMM the T–length observation sequence o_1^T is connected to the L–length character sequence via the state sequence s_1^T, then we compute $P(o_1^T|c_1^L)$ as follows.

$$\begin{aligned} P(o_1^T|c_1^L) &= \sum_{s_1^T} P(o_1^T, s_1^T|c_1^L) \\ &= \sum_{s_1^T} \prod_{t=1}^{T} P(o_t, s_{t+1}|s_t, c_1^L) \end{aligned} \tag{5}$$

and applying the maximum approximation we obtain:

$$P(o_1^T|c_1^L) = \max_{s_1^T} \prod_{t=1}^{T} P(o_t, s_{t+1}|s_t, c_1^L) \tag{6}$$

However, the $P(o_1^T|c_1^L)$ can also be computed as:

$$P(o_1^T|c_1^L) = \max_{s_1^T} \prod_{l=1}^{L} \prod_{e=1}^{T} \prod_{b=1}^{T} P(o_b^e, s_b^e|c_l) \tag{7}$$

where $P(o_b^e, s_b^e|c_l)$ corresponds to the likelihood of each individual character model c_l for each possible state sequence starting at a given initial state (s_I) and ending at a given final state (s_F) against each possible portion of the observation sequence delimited by a range of beginning observations $1 \leq b \leq T$ and ending observations $1 \leq e \leq T$. Figure 1 illustrates the decoding of a 3–state HMM and a 7–observation sequence.

The main advantage of using Equation 7 instead of Equation 6 in a lexicon–driven word recognition scheme is that given an observation sequence, character likelihoods can be computed once and reused further to compute the likelihoods of all words in a lexicon with no loss of optimality. This reusability of character likelihoods is very interesting in the case of large vocabularies and brings about a fast recognition scheme.

3.1 The Fast Two–Level Decoding Algorithm

We have developed a novel algorithm that finds recursively the $P(o_1^T|c_1^L)$ by computing Equation 7 and which is named *fast two–level decoding algorithm* (FTLDA). This algorithm breaks up the computation into two levels. At the first level character HMMs are decoded considering each possible entry and exit point and the result (best state sequences and likelihoods) is stored in an array for further use. At the second level words are decoded but reusing the likelihoods pre–computed at the first level. So, only the character boundaries are decoded without the necessity of going through the HMM states.

3.1.1 First Level: Decoding of Character Models

The idea underneath the proposed search strategy lies in avoiding repeated computation of state sequences of sub–word HMMs. Given a set of sub–word models $\Upsilon = \{\lambda_1, \ldots, \lambda_m, \ldots, \lambda_M\}$ where λ_m models character classes (letters, digits, and symbols) and a sequence of observations o_1^T, at the first level we evaluate the matching between o_1^T and each λ_m. To do that, we assume that each character HMM has only one initial state (or entry state) and only one final state (or exit state) and we compute the best state sequences between initial and final states, considering one possible beginning frame at each time, which is denoted as b, for all possible ending frames, which are denoted as e. Furthermore, we store in an array the best state sequences and likelihoods of all pairs of beginning and ending frames (b, e). The complete decoding algorithm for finding the best state sequences of each character HMM is presented as follows.

1. Initialization: for $1 \leq b \leq T+1$, $i = 1$:

$$\begin{aligned} \delta_b(1) &= 1 \\ \omega_b(1) &= 0 \\ \delta_b(j) &= \max_{1 \leq i \leq N} \left[\delta_b(i) a_{ij}'^{\Phi}\right] \\ \omega_b(j) &= \arg\max_{1 \leq i \leq N} \left[\delta_b(i) a_{ij}'^{\Phi}\right] \end{aligned} \tag{8}$$

2. Recursion: for $b \leq t \leq T+1$, $1 \leq j \leq N$:

$$\delta_t(j) = \max \left\{ \max_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}^{o_{t-1}}\right] ; \max_{1 \leq i \leq N} \left[\delta_t(i) a_{ij}'^{\Phi}\right] \right\}$$

$$\omega_t(j) = \begin{cases} \arg\max_{1 \leq i \leq N} \left[\delta_t(i) a_{ij}'^{\Phi}\right] & \text{if } ij \text{ is null} \\ \arg\max_{1 \leq i \leq N} \left[\delta_{t-1}(i) a_{ij}^{o_{t-1}}\right] & \text{otherwise} \end{cases}$$

$$\zeta_t(j) = \begin{cases} 1 & \text{if } ij \text{ is null} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

3. Termination: if $j = N$:

$$\begin{aligned} e &= t \\ \chi(b, e) &= \delta_t(N) \\ q_e^* &= N \end{aligned} \tag{10}$$

4. Path Backtracking: for $t = e-1, e-2, \ldots, b$, q_t^* is determined recursively by:

$$q_t^* = \omega_{t+1}(q_{t+1}^*) \tag{11}$$

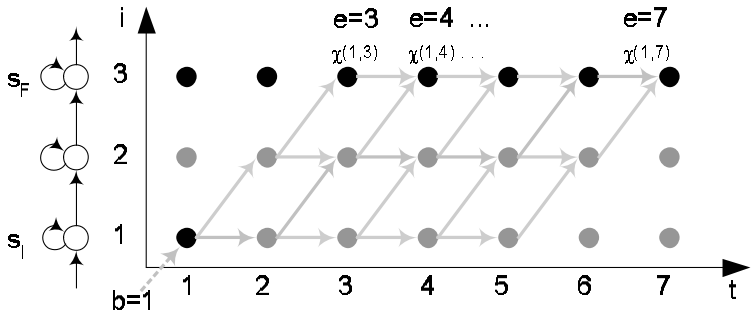

Figure 1. Decoding of a 3–state HMM for T=7, b=1, e=3,4,. . . ,7, s_I=1, and s_F=3. The resulting likelihoods are $\chi(1,3), \chi(1,4) \ldots \chi(1,7)$.

However, a new variable that accounts for the presence of null transitions, denoted as q'_t is introduced. After computing q^*_t we have to check if the variable ζ indicates the presence of a null transition, that is, $\zeta_t(q^*_t) = 1$. In such a case, without decreasing the time index t, q'_t is computed as:

$$q'_t = \omega_t(q^*_t) \quad \text{if} \quad \zeta_t(q^*_t) = 1 \tag{12}$$

Following the presence of a null transition during the backtracking procedure, that is $\zeta_{t+1}(q^*_{t+1}) = 1$, the computation of q^*_t is modified slightly as follows (now with a decrease in the time index t):

$$q^*_t = \omega_{t+1}(q'_{t+1}) \quad \text{if} \quad \zeta_{t+1}(q^*_{t+1}) = 1 \tag{13}$$

So, the best state sequence denoted as $\varepsilon(b,e)$ will be given by:

$$\varepsilon(b,e) = (q^*_e q'_e q^*_{e-1} q'_{e-1} \ldots q^*_{b+1} q'_{b+1} q^*_b q'_b) \tag{14}$$

So, for a given observation sequence o_1^T we end up with M arrays that keep the best likelihoods $\chi(b,e)$ and the best state sequences $\epsilon(b,e)$. Doing that, the likelihoods of the character models are totally independent of the context (the word within the character may appear) and may be reused unrestrictedly to decode all words in a lexicon.

3.1.2 Second Level: Decoding of Word Models

Considering our primary problem, that is to find the word $w \in \mathcal{L}$ that best matches with the sequence of observations o_1^T. Words are formed by the concatenation of single character HMMs (Equation 2), but we no longer have observation sequences and character models but instead pre–decoded character arrays $\chi_l(b,e)$ that hide the notion of observations and states.

The decoding of words is carried out from left to right and since words have well–defined initial and terminations, the initialization condition specifies that for the first character of the word represented by the array $\chi_1(b,e)$ the only valid entry point is at $b = 1$. Here we make use of the index l to denote the position of the character models within a word, and now we can throw away the index b and keep only the paths for which $\hat{\delta}_t(l)$ is maximal. The complete decoding algorithm for the second level is given as follows.

1. Initialization: for $1 \le l \le L$, $1 \le t \le T$:

$$\hat{\delta}_t(l) = \begin{cases} \chi(1,t) & \text{if } l = 1 \\ \max\limits_{1 \le b \le T} \left[\hat{\delta}_b(l-1)\chi(b,t) \right] & \text{if } 2 \le l \le L \end{cases}$$

$$\hat{\omega}_t(l) = \begin{cases} 1 & \text{if } l = 1 \\ \arg\max\limits_{1 \le b \le T} \left[\hat{\delta}_b(l-1)\chi(b,t) \right] & \text{if } 2 \le l \le L \end{cases} \tag{15}$$

2. Termination: for $l = L$, $t = T$:

$$\begin{aligned} \hat{P}^* &= \hat{\delta}_T(L) \\ \hat{q}^*(L) &= \hat{\omega}_T(L) \end{aligned} \tag{16}$$

3. Character Backtracking: for $L-1, L-2, \ldots, 1$:

$$\hat{q}^*(l) = \hat{\omega}_{\hat{q}^*(l+1)}(l) \tag{17}$$

4. State Backtracking: for $L-1, L-2, \ldots, 1$:

$$\begin{aligned} \hat{\epsilon}_L &= \epsilon[T, \hat{q}^*(L)] \\ \hat{\epsilon}_l &= \epsilon[\hat{q}^*(l+1) - 1, \hat{q}^*(l)] \end{aligned} \tag{18}$$

At the initialization step, $\hat{\omega}_t(l)$ records the frame t in which the preceding character ended. To recover the character boundaries, we need to rely on the character output backpointer, $\hat{q}^*(l)$ that records the time t at which the previous character ended and it is determined recursively at the Step 3. Recall that the whole MAP state sequence for each single character was stored in an array $\epsilon(b,e)$. So, to recover the MAP state sequence we need the additional Step 4.

Table 1 shows the approximate computational complexity and storage requirements for the FTLDA and a conventional Viterbi algorithm. The main difference is that the first level of the FTLDA is independent of the lexicon, but it is repeated for each possible beginning and end frame and character models. A rough comparison of the computational complexities shows that the FTLDA performs better than a conventional Viterbi algorithm when $T < N^2$. However, at the second stage, T^2LV can be reduced to $T(T-D)LV$, where D is an estimation of the duration of the character models, without loss of accuracy.

4 Experiments and Results

A database of 20,172 unconstrained handwritten words extracted from postal envelopes was used to evaluate the performance of the proposed decoding algorithm. A dataset of 12,023 unconstrained handwritten words was used to train seventy–two HMMs corresponding to 26 uppercase letters (A–Z), 26 lowercase letters (a–z), 10 digits (0–9) and 8 special symbols. A validation set of 3,475 unconstrained handwritten words was also used during the training procedure. A test set with 4,674 unconstrained handwritten words was used for evaluation and comparison of proposed decoding algorithm with a conventional Viterbi algorithm. A vocabulary with 85,092 city names was used in all recognition experiments and dynamic lexicons with different sizes (10, 1,000, 10,000, 40,000, 80,000) were randomly generated from this vocabulary.

The criteria to evaluate the recognition accuracy is the recognition rate while the recognition time, defined as the time in seconds required to recognize one word and measured in CPU–seconds was used as criteria to evaluate the recognition speed. The experiments were carried out on a PC AMD Athlon 1.1GHz, 512MB of RAM memory and all results are averaged over the test set of $4,674$ words and ten runs for ten different lexicons which were generated randomly. It is important to notice that the recognition time also depends on the software implementation. To ensure a fair comparison, the implementations of all search algorithms were kept as similar as possible, so they share many parts of code, including disk operations.

Table 2 summarizes the word recognition rates obtained by using the two versions of the FTLDA, one with a flat lexicon (FlatLex) and other with a lexical tree (LexTree). This table also presents the results for the conventional Viterbi algorithm with a flat lexicon. The bottom line is that there is no difference in accuracy between the FTLDA and the Viterbi implementation. However, these results were already expected because the FTLDA maintains the optimality in terms of maximum likelihood of the conventional Viterbi algorithm. Table 2 also shows the recognition times for the 5 dynamically generated lexicons. The performance of the Viterbi algorithm is completely flawed on very–large vocabulary tasks (80,000 words) since it needs more than 3 minutes to recognize a single word.

There is a significant improvement in recognition time relative to the Viterbi implementation and speedup factors between 6.3 and 20.6 were achieved for the FTLDA with a lexicon tree and between 7.0 and 15.9 for the FTLDA with a flat lexicon. It is worth to notice that the FTLDA somewhat overlaps the advantages of using a lexical tree instead of a flat lexicon. Moreover, even for small and medium–size lexicons, the fast search strategy is advantageous. In summary, there is a significant improvement in recognition time relative to the Viterbi implementation while preserving exactly the same word recognition rates.

5 Conclusion

This paper has focused on the problems relating to the computational efficiency of large vocabulary handwriting recognition. The main concern is that with a novel fast two–level decoding algorithm it is possible to speedup significantly the recognition process while maintaining the recognition accuracy. As a result, it is now possible to recognize writer–independent, unconstrained handwritten words in reasonable time, using a very–large vocabulary with relative accuracy.

Improvements in recognition speed were achieved by using the FTLDA which breaks up the computation of word likelihoods into two levels: state level and character level. This enables the reuse of the character likelihoods to decode all words in the lexicon, avoiding repeated computation of state sequences. The main results achieved by using the FTLDA is a 15 speedup factor relatively to a conventional Viterbi algorithm for an 80,000–word vocabulary task. Good results were also achieved for lexicons with a smaller number of words. Nevertheless in our experiments the high speedup achieved was due the segmentation algorithm and high–level features which have produced short observation sequences (30 observations in average) and HMMs with 10 states.

Although, the recognition times presented in this paper for the FTLDA might not meet the throughput requirements of many practical applications since it still needs approximately 15 seconds to recognize a single word in an 80,000–word lexicon. However, the results are very encouraging and hopefully, this work will stimulate other researchers to pursue interesting research into this subject since many applications require large vocabularies. In practice, the recognition times could be further reduced by using some code optimization and programming techniques.

Table 1. Computational complexity and storage requirements for the FTLDA and Viterbi algorithm

Algorithm	Computational Complexity	Storage Requirements
FTLDA	$T^2N^2M + T^2LV$	$2T^2N + 2MT^2$
Viterbi	TN^2LV	$2TN^2LV$

Table 2. Word recognition rates and recognition times for the system based on the FTLDA and based on a conventional Viterbi algorithm.

Lexicon Size (#)	Recognition Rate (%)	Recognition Time (sec/word)		
		FTLDA LexTree	FTLDA FlatLex	Viterbi FlatLex
10	98.84	0.010	0.009	0.063
1,000	91.01	0.273	0.321	4.685
10,000	81.06	1.992	2.576	41.06
40,000	73.23	7.516	9.741	139.6
80,000	68.65	14.46	19.52	216.4

References

[1] E. Augustin, O. Baret, D. Price, and S. Knerr. Legal amount recognition on french bank checks using a neural network–hidden markov model hybrid. In *Proc. 6th Int. Workshop on Frontiers in Handwriting Recognition*, pages 45–54, Taejon, Korea, 1998.

[2] A. El-Yacoubi, M. Gilloux, R. Sabourin, and C. Y. Suen. Unconstrained handwritten word recognition using hidden markov models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):752–760, 1999.

[3] D. Guillevic and C. Y. Suen. Cursive script recognition applied to the processing of bank cheques. In *Proc. 3rd International Conference on Document Analysis and Recognition*, pages 11–14, Montreal, Canada, 1995.

[4] G. Kim and V. Govindaraju. Bankcheck recognition using cross validation between legal and courtesy amounts. In S. Impedovo, P. S. P. Wang, and H. Bunke, editors, *International Journal of Pattern Recognition and Artificial Intelligence*, pages 657–673. World Scientific, 1997.

[5] A. L. Koerich, R. Sabourin, and C. Y. Suen. Large vocabulary off–line handwriting recognition: A survey. *Pattern Analysis and Applications*, 6(2):97–121, 2003.

[6] A. L. Koerich, R. Sabourin, and C. Y. Suen. Lexicon–driven hmm decoding for large vocabulary handwriting recognition with multiple character models. *International Journal on Document Analysis and Recognition*, 6(2):126–144, 2003.

[7] Y. LeCun, O. Matan, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten zip code recognition with multilayer networks. In *Proc. 10th International Conference on Pattern Recognition*, pages 35–40, Atlantic City, USA, 1990.

[8] L. L. Lee, M. E. Lizarraga, N. R. Gomes, and A. L. Koerich. A prototype for Brazilian bankcheck recognition. In S. Impedovo, P. S. P. Wang, and H. Bunke, editors, *Automatic Bankcheck Processing*, pages 549–569. World Scientific, 1997.

[9] U. Mahadevan and S. N. Srihari. Parsing and recognition of city, state, and zip codes in handwritten addresses. In *Proc. 5th International Conference on Document Analysis and Recognition*, pages 325–328, Bangalore, India, 1999.

[10] R. Plamondon and S. N. Srihari. On–line and off–line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):68–89, 2000.