AILP (2010) Assignment 2 Report

Key1	*ute cYe 26F*
Key2	*MhlCPrpChsl*

Abstract

While isolated character recognition is an important task to investigate, most handwriting recognition systems - such as those employed by touch sensitive devices - use whole word input rather than individual characters. This report carries on from previous work in isolated character recognition, examining the larger task of continuous handwriting recognition. Three main approaches are identified - word-level matching, segmentation-based matching and a One-Stage dynamic programming algorithm. These approaches are examined and evaluated in terms of their accuracy and computational complexity on a dataset of numbers and English words, and an optimal system is created from the best testing system.

1. Introduction

In the previous part of this project, I looked at online isolated character recognition. While not unimportant, the task of online isolated character recognition is not directly useful in a real world situation. It is unlikely that a user would wish to enter singular characters one after another, indicating when they are finished after every character they write. Instead, written input is usually entered in entire words or even sentences. This report details an investigation into the recognition of both whole number and word input - the task of splitting a sentence into its component words is not considered here.

Some of the major challenges facing a continuous handwriting recognition system are similar to those faced by an isolated character recognition system, like the one implemented in the previous section of this project. The quality of the input data (the words/numbers) is usually poor, often worse than with isolated characters due to the greater freedom for personal style (see Fig 1 for an example.) Other challenges include the larger search space for matching input words and numbers to[1], and the higher potential for mistake by the input writer.

The importance of the task - combined with the challenges listed above - has resulted in much research in the area of continuous handwriting recognition. This has lead to a large and varied set of approaches for solving this problem, from Dynamic Time Warping (DTW) matching to the use of Hidden Markov Models (HMMs)[1, 2, 3]. Of these many approaches I will be considering three in this report. The first of these is a simple adaption of the system I developed previously for isolated character recognition, and serves as the baseline system for the report. The second approach examined is segmentation based recognition, where the input sequence is split into its component symbols which are then individually recognised (again using the isolated character recognition system developed previously) and rebuilt into an output word. Due to the complexity of this approach there are many different methods for attempting it[3-9] - I will examine and evaluate four of these. Finally I will investigate a dynamic programming approach to the problem, the One-Stage dynamic programming algorithm. This approach attempts to combine both segmentation and recognition of component symbols into a single task.

[1] In the latter case this search space can be effectively infinite.

Figure 1: *Examples of the word "Menu" written by four different writers.*

As with my previous work on isolated character recognition systems, I will consider the approaches detailed here with regards to the two main criteria for a handwriting recognition system - accuracy and processing time. While accuracy is obviously of prime importance to any recognition system, we must also balance this against time taken to recognise input words - an online recognition system is of little use if the user does not find the length of time required for recognition acceptable. their input to be recognised.

Note that throughout this paper the term 'word' is often used ambiguously to refer to any sequence of continuous handwriting - that is, both numbers and English words. Any usage of this term to refer purely to English words will be made explicit, otherwise the reader should assume the general meaning.

2. Experimentation

2.1. Overview

The input data set used for this project is a version of the MIT UNIPEN handwriting database, converted to the IIPL format for simplicity. Both number and word patterns are used for testing, with the isolated character and digit patterns used as reference patterns where needed.

All experiments performed within this report were carried out using 5-fold cross validation. The set of writers was split into 5 sets, based on their remainder when divided by 5 - 0, 1, 2, 3 and 4. Experiments were carried out separately on the word and number sets of characters, and independent results are given for each experiment. In each experiment both the accuracy and time taken were monitored, and the average and standard deviation for both are given in the results table for the experiment. Note that the time listed in the results tables is the average time to recognise a single word, calculated (approximately) by dividing the time taken per fold by the number of input patterns recognised in that fold.

Apart from the One-Stage dynamic programming approach, all other approaches considered in this report use 8 different reference templates, using the randomised generation of reference writers that I investigated in my previous work on isolated character recognition. Although not the best method of multi-templating, randomised choosing of writers was found to give accuracies of at worst only 5% less than the best method (pre-selection based on a scoring approach), and is far simpler to implement than the overly complex pre-selection method. 8 writers was previously found to be the optimal choice in terms of accuracy increase versus time taken.

2.2. Pre-processing

As all of the methods explored within this report use some form of matching between two patterns, various pre-processing techniques are used to increase the accuracy of the matching (see my original work on normalisation techniques in an isolated character recognition system for details.) Mean normalisation translates an input pattern to be centred at the point (0,0), solving the problem of variance in the position of the patterns we wish to match. Scale normalisation re-sizes a pattern to fit within a set bounding box, in order to correct for the different sizes that people write text. Additionally, the One-Stage dynamic programming approach uses velocity feature vectors instead of points for its inputs, as this implicitly applies mean normalisation.

2.3. Baseline System

As mentioned in the introduction, the baseline system for this project is simply an extension of the isolated character recognition system developed in the first assignment. The system was adapted to use numbers and words as patterns to match instead of digits or characters. Given an input sequence, the baseline system computes distances from the sequence to a list of reference patterns. The reference pattern with the smallest distance is chosen as the correct classification for the input.

As it is basically just a version of my previous isolated character recognition system, the baseline system for this project uses mean normalisation to centre its input and reference patterns for improved matching. However, unlike the isolated character systems, scale normalisation is not used as the dynamic size of words means that a fixed sized bounding box may lead to over or under compression of the pattern points. A fixed-height, variable-width bounding box scaling could be used, but this would still suffer with things like sloped input patterns. Given that my previous work found that scale normalisation added at most 2-3% to the accuracy, it was decided that its inclusion here was unnecessary.

The potential downsides to this method are fairly obvious. While differences in writing style do not matter too much with isolated character recognition (as I found in my previous report), words written by different people can vary greatly in terms of shape, size, stroke order, and so on. An example of just a few of the different writing styles found in the dataset is given in Fig 1. Using multi-templating as I have done here may help to alleviate this problem somewhat, but I do not predict a high level of accuracy for the baseline system. An additional point to consider is that even if the system was accurate enough for real world use, the use of entire word reference templates means that we need a template for every word we wish to be able to recognise - indeed, if using the multi-template system discussed here we need 8 templates per word. Given that the English language contains at least a quarter of a million words, and that the average English speaking adult uses around twenty thousand words regularly[10] (and in terms of numbers the potential input is effectively unlimited), it is unlikely that this system would scale without a large increase in storage requirements and a similarly large decrease in accuracy.

Figure 2: *An example of a word with both 'correct' boundaries (between the characters) and 'incorrect' boundaries (within the 'k').*

2.4. Segmentation

The second main approach to continuous handwriting recognition that was examined is segmentation-based recognition. Instead of attempting to recognise an input directly against a potentially massive set of words or numbers, segmentation-based approaches try to split an input pattern into its component symbols. These symbols are then recognised individually, using the isolated character recognition system developed previously, and are concatenated to form the classification output for the input.

There are two main types of segmentation - hard and soft. Both attempt to locate boundaries between symbols and to separate the input sequence accordingly. The difference arises in how the detected boundaries are applied. In hard segmentation we split the input over all of the boundaries that we detect, without any consideration about whether these are 'correct' boundaries. Soft segmentation attempts to determine whether the detected boundaries are actual boundaries between symbols, or simply gaps within a symbol. Fig 2 shows an example of a word with both 'correct' boundaries and 'incorrect' boundaries. In this section I will examine multiple methods for hard segmentation, and one method that can be implemented as either hard or soft segmentation.

Regardless of their type, all of the segmentation techniques investigated in this report use both the mean and scale normalisation pre-processing techniques when recognising the individual symbols detected, and they all also use a dictionary look-up as a post-processing step. The dictionary look-up - which is further discussed later - is used in order to correct the small mistakes that segmentation methods almost always end up with even with a good input pattern.

2.4.1. Stroke-Based Segmentation

In stroke-based segmentation we make a simplifying assumption - that each stroke represents a different character. This assumption is obviously naive, as it will separate out any character written in more than one stroke into multiple characters. When applied to the word in Fig 2, stroke-based segmentation would recognise the 'k' as two different symbols which would be most likely classified as an 'l' and a 'c', making the overall classification 'Baclcing' (assuming perfect character recognition).

This separation of characters means that stroke-based segmentation tends to over-segment its input pattern, and thus the output recognition will usually have too many characters. However, the use of the dictionary search mentioned previously can

help to clean up many of these small errors - for example, passing 'Baclcing' through it will result in an answer of 'Backing', which is the correct recognition. This is not always a sure proof method, however - some characters such as 'x' or 'E' usually cause more severe over-segmentation that the dictionary search cannot correct. This form of segmentation also fails to properly segment (or segment at all) multiple characters written in a single stroke, such as with writer C in Fig 1. Despite these flaws, I expect that stroke-based segmentation will be fairly accurate, as many of the test writers within the UNIPEN dataset write in mostly one stroke characters. It should also be fairly similar to the baseline system in terms of time taken, as the only extra work (the segmentation) requires just a single pass through the input pattern which is fairly computationally cheap.

Stroke-based segmentation (as implemented in this project) is a hard segmentation technique. While it may be possible to implement it as a soft segmentation method, this has not been attempted.

2.4.2. Centre of Mass Heuristic

We can attempt to fix some of the problems that we identified with stroke based segmentation by using a centre of mass heuristic. This heuristic attempts to join together distinct strokes that are actually part of the same symbol, in order to reduce the problems with multi-stroke characters such as 'E'. As most words are written horizontally, our calculations are based purely on the x-axis of the input pattern. A first pass is made through the input pattern in order to determine the width of the word and to record the mean x-coordinate of each stroke, and then a threshold distance is calculated using the following formula (where α is the threshold parameter):

$$dist\_threshold = \frac{maxX - minX}{\alpha}$$

A second pass through the input pattern is then made, examining the stroke number for each point. When the stroke changes we examine the mean x values for the two strokes - if the distance between them is less than the threshold value they are combined into a single symbol, otherwise we take the new stroke as the start of a new symbol.

This method does solve some of the problems that stroke based segmentation suffers from, but unfortunately it is not without problems of its own. It is very sensitive to the value of α, and I was unable to identify either a good general value for this parameter, nor a way to dynamically determine it. This is problematic as if α is too low for an input pattern this method is prone to under-segmenting, joining together adjacent characters that should be separated - but if α is too high then this method mostly becomes just a computationally more expensive stroke segmentation. Additionally, it is not unusual for parts of a single character to be further apart than adjacent characters, especially in the case of letters such as 'i'. Finally the algorithm in its current form only looks at sequential strokes - therefore, if a symbol is made up of strokes that are not sequential in numbering (as is the case for the letter 't' in Fig 3), this heuristic will not correctly join it into one symbol.

Figure 3: *An example of a word where a character (the first 't') has non-sequential strokes.*

Figure 4: *An example word with its histogram. Note that histogram segmentation would fail to seperate the two 'z's.*

Given these problems, I do not expect the centre of mass heuristic to increase the accuracy of stroke based segmentation by much, if at all. Computationally wise it is should not be much more expensive than stroke based, as it requires only one more pass through the input pattern to gather information.

As it is just a slight alteration of the stroke based algorithm, this is also a hard segmentation technique.

2.4.3. Histogram Segmentation

The final segmentation technique examined in this paper, histogram segmentation is technically a form of offline segmentation, though it can be applied reasonably quickly to online data. In this method a histogram (along the x-axis) is created for the input pattern, and then examined to find boundary points between symbols - points where no data from the input pattern is present. The histogram is created by iterating through the points in the input sequence, traversing along the line created by any two sequential points and increasing the histogram count for the x-coordinates that are passed through. An example input word and its histogram are given in Fig 4.

Histogram segmentation can be implemented as either hard or soft segmentation. Both methods generate the histogram as above, but differ in how they then apply it in order to segment the input pattern. In hard histogram segmentation the histogram is searched for boundary regions, which are recorded. All of these boundary regions are then used to split the input pattern into separate characters, which are then recognised individually as normal.

The soft segmentation method is slightly more complex. In it, each boundary region is given a score based on its width - the assumption being that the wider a boundary, the more likely that it is an actual character split rather than a small gap within a character. The regions are then ordered according to their score, and incrementally added as definite boundaries. Every time a boundary is added the distances for the segments so far are calculated, and the sum of the lowest distances recorded. If this sum is lower than the previous distance calculated, the boundary is taken to be a 'good' boundary and kept. However if

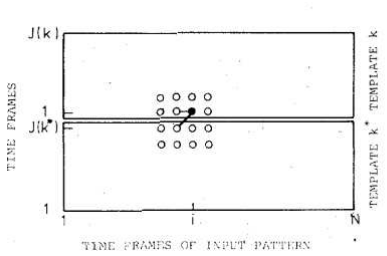

Figure 5: *A sub-section of the table used in the One-Stage algorithm, showing how we can visualise it as a 2 dimensional table.*

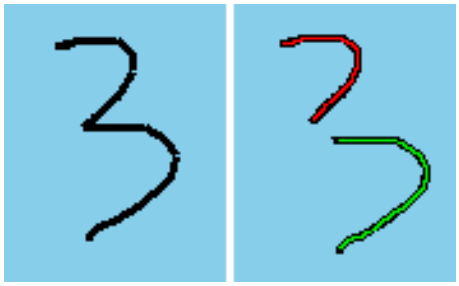

Figure 6: *An example of how the One-Stage algorithm might be able to find two '7' sub-symbols within a '3'.*

the distance increases after a boundary is added, we assume that we have reached the end of the good boundaries. The boundary that was just added is removed, and we split the input pattern over our set of finalised boundaries.

Both of these methods have advantages and disadvantages. If the input pattern is neatly spaced (as many of the patterns in the UNIPEN dataset actually are), both methods will work very well, with the hard method usually segmenting better than the soft. Unfortunately, as with the previous segmentation methods, neither hard nor soft histogram segmentation can deal with characters written in a single stroke. Additionally, if two characters overlap at all - even if written in two separate strokes - this method of segmentation will fail to segment them, combining them into a single symbol. Despite these deficiencies, I expect that histogram segmentation will perform at least as well as stroke segmentation in terms of accuracy, if not better, due to the fairly common nature of multi-stroke characters. Computationally wise, hard histogram segmentation is not much more expensive than the baseline system or the stroke-based system. Soft segmentation, on the other hand, is more expensive as it requires multiple segmenting and computing of distances (using the DTW algorithm) to segment an input pattern.

2.5. One-Stage DP

The final approach that will be examined in this paper is the One-Stage dynamic programming method for continuous word recognition, introduced by Ney in his 1984 paper[11]. This is a more complex method than those previously examined in this paper, as it attempts to combine both segmentation and recognition into a single task. It is able to identify characters made from multiple strokes as well as characters that are part of the same stroke.

This algorithm essentially works by trying to find characters contained within the input pattern, just as segmentation does. However, unlike segmentation it does not consider specific boundary regions based on strokes or blank spaces, but instead attempts to recognise all potential characters contained anywhere in the input pattern and then extracts the most likely sequence of these characters. This means that it calculates the likelihood of a section of the input pattern being a reference pattern as it 'segments', essentially segmenting and recognising at the same time.

In order to track these potential characters we use a three dimensional table D, whose (x, y, z) dimensions are the input pattern points, the reference patterns and the points for each reference pattern. As a useful visual analogy we can actually consider the table to be two dimensional, where the y axis contains each reference pattern split into their points - see Fig 5.

The table is built from the 'bottom left' up, using the following two formula:

$$D(i,1,k) = d(i,1,k) + min(D(i-1,j,k); \\ D(i-1,J(k^*),k^*) + \alpha : k^* = 1,...,K)$$

$$D(i,j,k) = min(\beta * d(i,j,k) + D(i-1,j,k); \\ d(i,j,k) + D(i-1,j-1,k); \\ \gamma * d(i,j,k) + D(i,j-1,k))$$

The first of these two formulas is used for between-template transitions, where we could either have come from the previous input point - D(i-1, j, k) - or the final point of *some* other reference pattern k^*. The second formula is the within-template step rule, and simply determines whether we came from just the previous input point, the previous input and reference point, or just the previous reference point.

Once the table has been built according to these rules, we must then traverse it starting from the far right column, following the minimum distance path to find the closest matching characters for the input pattern. This gives out a list of reference patterns that match to the input pattern, in reverse order. Reversing this list gives us the recognition for the input pattern.

This approach has multiple benefits over those examined above. As mentioned, it is able to deal with many different situations, such as cursive (joined-up) handwriting and multi-stroke characters. It also scales well, as the recognition time for one symbol does not depend on the number of other symbols within the input pattern.

Despite these benefits, this approach is not without problems. It is very dependent on the angle of the path it takes through the distances table. If the path is too steep it will over-segment, and recognise many different characters within a single symbol - for example finding two '7's within a '3' (see Fig 6). Conversely, if the path is too shallow it will combine multiple symbols into a single one instead. We can modify the parameters α, β and γ above in order to favour one path over the other, but I was unfortunately unable to determine an optimum setting for these parameters. The method may also be overly complex for the UNIPEN dataset, which (despite some oddities) largely contains straight forward words which can be recognised by simpler means. For example, there are very few cursive writers in the dataset.

Despite these disadvantages this method should show good accuracy with a reasonable computational cost (the building of the table is not much more expensive than the DTW algorithm used by the other approaches).

2.6. Post-processing

Many of the approaches examined in this project often give out a 'nearly correct' solution which is only slightly off from the target word. In order to help correct these small errors, a case insensitive search (using Levenshtein distance) against a provided dictionary of a thousand English words was used to try and match alphabetic output from the recognisers (a similar method was used for numbers, although due to the fact that any combination of digits is a valid number this largely consists of truncating the output number to have 5 digits). This dictionary search also deals with cases where the original writer mis-wrote or mis-spelt the prompted word.

3. Results

3.1. Baseline

As expected, the baseline system performed fairly poorly in terms of accuracy, giving only 8% accuracy for numbers and 6% for words - between just three and four times the accuracy one would get from just randomly guessing in each case. The time taken to recognise a word was fairly reasonable, although of course this can at best be used for a comparison with the other systems investigated here - it is no indication of real-world performance. The results for the baseline system can be found in Table 1.

	Avg Acc	Std Dev of Acc	Avg Time	Std Dev of Time
Numbers	8.05%	1.232	0.297s	0.003
Words	6.10%	0.654	0.998s	0.017

Table 1: *Baseline system results.*

3.2. Segmentation

3.2.1. Stroke-Based Segmentation

The results for stroke-based segmentation were not entirely as expected, showing a large difference in the accuracies for recognising numbers and words. The average accuracy for numbers, 21.74%, might have been expected due to the problems with over-segmenting that stroke-based segmentation suffers from, but the high accuracy for words (66.11%) is unexpected. I theorise that the dictionary post-processing that is applied to the recogniser must correct for the small errors caused by over-segmentation far better than expected, giving the system a far higher accuracy than might be expected otherwise. In order to confirm this, experimentation should be carried out to determine how many of the correctly recognised words were recognised only because of the dictionary search that the recogniser applies as its final step (and how much correction was necessary). The time taken for stroke based segmentation is not much more than for the baseline system, which was expected as stroke decisions are fairly computationally cheap and they both perform roughly the same amount of work for the DTW algorithm.

The results for this experiment can be found in Table 2.

	Avg Acc	Std Dev of Acc	Avg Time	Std Dev of Time
Numbers	21.74%	2.442	0.509s	0.007
Words	66.11%	1.622	1.094s	0.017

Table 2: *Stroke-based segmentation system results.*

3.3. Centre of Mass Heuristic

As with the stroke-based segmentation method, the results for the system using the centre of mass heuristic were expected in some areas and surprising in others. The increase in accuracy for numbers (from 21.74% in the stroke-based segmentation system, to 29.42% with the centre of mass heuristic) seems to confirm that applying a centre of mass calculation helps the recognition of multi-stroke symbols. Additionally, the potential decrease in accuracy for English word recognition was not surprising, due to the problems with under-segmentation of input words. However, the size of decrease in accuracy is unusual. It is most likely caused by under-segmentation of the input words - collecting together multiple characters as a single character. The time taken for this experiment was much the same as for without the centre of mass heuristic, which is not surprising as it is a very computationally insignificant heuristic.

The results for this experiment can be found in Table 3.

	Avg Acc	Std Dev of Acc	Avg Time	Std Dev of Time
Numbers	29.42%	3.664	0.500s	0.005
Words	17.03%	1.568	0.973s	0.022

Table 3: *Stroke-based segmentation (with centre of mass heuristic) system results.*

3.4. Histogram

Unlike the previous two methods, the results for histogram segmentation are largely as expected. Both hard and soft segmentation show a considerable increase in accuracy over the baseline system, in both number and word recognition. Hard segmentation actually shows a slightly higher accuracy than soft, with an average accuracy of 54.39% as compared to 47.98%. This is likely due to the fairly 'nice' nature of the UNIPEN data, which is mostly cleanly segmented, thus leading to hard segmentation making many "good" choices while soft segmentation did not always pick the best segmentation for the input data. An interesting comparison is the time taken - while hard segmentation takes roughly the same amount of time as the previously examined methods, soft segmentation takes around 33% more time than hard segmentation for English word recognition (10% for numbers). This was again expected, due to the multiple times that soft segmentation has to segment its input and compute the distance to the reference patterns.

The results for hard segmentation can be found in Table 4, and the results for soft segmentation can be found in Table 5.

	Avg Acc	Std Dev of Acc	Avg Time	Std Dev of Time
Numbers	52.14%	5.417	0.497s	0.009
Words	56.64%	5.363	1.019s	0.013

Table 4: *Histogram-based (hard) segmentation system results.*

	Avg Acc	Std Dev of Acc	Avg Time	Std Dev of Time
Numbers	50.69%	6.703	0.543s	0.035
Words	45.28%	4.685	1.359s	0.056

Table 5: *Histogram-based (soft) segmentation system results.*

3.5. One-Stage DP

The outcome of the One-Stage dynamic programming approach was unexpected. Both numbers and words show a very low

accuracy - 2.19% on average for numbers, 10.49% for words. These results are far lower than should be observed with this method, which indicates that some flaw existed within its implementation - most likely with the choices for α, β and γ. More work must be done to identify the problem and correct it.

An interesting point to note is the running time of the One-Stage approach. Despite the failings in accuracy presented here, the time for recognising a word is only 0.709 seconds on average - nearly a 0.3 second (or 30%) increase in speed from even the second fastest system. This highlights one of the main benefits from the dynamic programming method - the time saved by segmenting as you recognise.

The results for this experiment can be found in Table 6.

	Avg Acc	Std Dev of Acc	Avg Time	Std Dev of Time
Numbers	2.19%	1.450	0.572s	0.006
Words	10.49%	4.817	0.709s	0.019

Table 6: *One-Stage dynamic programming system.*

4. Conclusion

In this report I have examined three main approaches to the problem of continuous handwriting recognition - word matching, segmentation based and a One-Stage dynamic programming approach - and evaluated them in terms of their accuracy and speed of recognition.

Of these approaches, the one that showed most promise was hard segmentation using a histogram, with a 54.39% accuracy on average while retaining a recognition time of around 0.5 seconds for numeric input or 1.0 seconds for English words (on average). An interesting point is that if we only wish to recognise English words the simple stroke-based segmentation might be a better choice, as it achieved high accuracy than the histogram segmentation system for alphabetic input.

In terms of future work, there are many potential improvements for the approaches identified within this report. The centre of mass heuristic could be re-worked to handle non-sequential strokes, as well as to address the problem of its parameterisation. It could then be combined with the histogram segmentation in order to try and split touching symbols that the histogram segmentation failed to seperate. The analysis of the histograms could be extended to include local minimas in order to attempt to identify the case where multiple characters are written with a single stroke. Finally, the One-Stage dynamic programming approach presented here must be analysed to find the root cause of its failure which should be addressed.

5. References

[1] Mysers, C. S., and Rabiner, L. R., "A Comparative Study of Several Dynamic Time-Warping Algorithms for Connected Word Recognition", The Bell System Technical Journal, September 1981

[2] Bercu, S., and Lorette, G., "On-line Handwritten Word Recognition: An Approach Based on Hidden Markov Models", Pre-proceedings IWFHR III, May 1993

[3] Casey, R. G., and Lecolinet, E., "A Survey of Methods and Strategies in Character Segmentation",

[4] Paszkowski, B., Bieniecki, W., and Grabowski, S., "Preprocessing for Real-Time Handwritten Character Recognition",

[5] Loy, W. W., and Landau, I. D., "An On-Line Procedure for Recognition of Handprinted Alphanumeric Characters", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.PAMI-4, No. 4, July 1982

[6] McInerney, S., and Reilly, R. B., "Hybrid Multiplier/CORDIC Unit for Online Handwriting Recognition", 1999

[7] Brown, M. K., and Ganapathy, S., "Preprocessing Techniques for Cursive Script Word Recognition", Pattern Recognition, Vol. 16, 1983

[8] Groner, G. F., "Real-Time Recognition of Handprinted Text", 1966

[9] Tappert, C. C., Suen, C. Y, and Wakahara, T., "The State of the Art in On-Line Handwriting Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, August 1990

[10] "How Many Words are there in the English Language?", (`http://www.oxforddictionaries.com/page/93`)

[11] Ney, H., "The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition", IEEE Transactions on Acoustics, Speech and Signal Processing, April 1984

[12] Gader, P., Whalen, M., Ganzberger, M., and Hepp, D., "Handprinted Word Recognition on a NIST Data Set", Machine Vision and Applications, 1995

[13] Mohamed, M., and Gader, P., "Handwritten Word Recognition Using Segmentation-Free Hidden Markov Models and Segmentation-Based Dynamic Programming Techniques", IEEE Transactions on Pattern Analysis and Machine Intelligence, May 1996

[14] Gader, P., Mohamed, M., and Chiang, J., "Handwritten Word Recognition with Character and Inter-Character Neural Networks", IEEE Transactions on Systems, Man and Cybernetics, February 1997

[15] Myers, C. S., and Rabiner, L. R., "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition", IEEE Transactions on Acoustics, Speech, and Signal Processing, April 1981

[16] Sakoe, H., "Two-level DP-Matching - A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition", IEEE Transactions on Acoustics, Speech, and Signal Processing, December 1979