HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Engineering Physics and Mathematics

Handwritten Character Recognition: A Palm-top Implementation and Adaptive Committee Experiments

Matti Aksela

The thesis belongs to the special subject of information science

Supervisor Professor Erkki Oja
Instructors Dr. Jorma Laaksonen and Docent Jari Kangas

Espoo, Finland 22nd May 2000

HELSINKI UNIVERSITY OF TECHNOLOGY | **ABSTRACT OF THE MASTER'S THESIS**

Author and name of the thesis:

Matti Aksela

Handwritten Character Recognition: A Palm-top Implementation and Adaptive Committee Experiments

Date: 22nd May 2000 **Number of pages:** 120

Department: Department of Engineering Physics and Mathematics
Professorship: Tik-61 Information Sciences

Supervisor: Professor Erkki Oja
Instructors: Dr. Jorma Laaksonen and Docent Jari Kangas

The work presented in this thesis addresses on-line recognition of handwritten characters. An adaptive on-line handwritten character recognition system has been developed earlier in the Laboratory of Computer and Information Science. This thesis is focused around and expands the system.

The literature survey part of the thesis is two-fold, as the first part deals with pen computing hardware and applications and their development as well as the use of handwritten text as an input method in such devices. The second part of the survey is focused on various approaches to handwritten character recognition. Methods for both individual classifiers and committee combination are explored, with the focus on methods suitable for on-line operation.

For the practical part of this thesis two separable fractions can also be identified, one focusing on the palm-top implementation of the recognition system and the other on the implementation and testing of an adaptive committee classifier based on the Dynamically Expanding Context (DEC) principle. The adaptive on-line recognizer developed earlier is implemented on the palm-top platform and a suitable interface for data collection and recognizer performance evaluation is created. The interface is in the form of a questionnaire program to which the user answers by using handwriting. Also some speed-up methods necessitated by the platform have been investigated. The DEC committee has been implemented to join the results from a varying number of classifiers in an adaptive committee operation. Its performance has been evaluated in comparison to some reference committee methods, both adaptive and nonadaptive.

Results from the palm-top implementation show that the system is capable of adapting to the user's style of writing. Also the speed-up methodologies provide noticeable gains in recognition speed. The DEC committee established a notable performance increase over any member classifier and also outperformed all the reference classifiers.

Keywords:
pattern recognition, isolated handwritten characters, on-line recognition, adaptation, learning systems, committee recognition, pen-based computing, palm-top;
k Nearest Neighbour rule, Learning Vector Quantization (LVQ), Dynamic Time Warping (DTW), elastic matching, Dynamically Expanding Context (DEC);

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä ja työn nimi:

Matti Aksela

Käsinkirjoitettujen merkkien tunnistus: kämmentietokonetoteutus ja adaptiivisia komitea-kokeita

Päivämäärä: 22. 5. 2000 **Sivumäärä:** 120

Osasto: Teknillisen fysiikan ja matematiikan osasto
Professuuri: Tik-61 Informaatiotekniikka

Valvoja: Professori Erkki Oja
Ohjaajat: TkT Jorma Laaksonen ja Dosentti Jari Kangas

Työ liittyy käsinkirjoitettujen merkkien reaaliaikaiseen tunnistukseen. Se perustuu ja toimii lisänä Informaatiotekniikan laboratoriossa toteutettuun adaptiiviseen reaaliaikaiseen käsinkirjoitettujen merkkien tunnistusjärjestelmään.

Työn kirjallisuustutkimusosuus on kaksiosainen. Ensimmäinen osa käsittelee kynäpohjaiseen tietojenkäsittelyyn liittyviä laitteistoja sekä sovelluksia ja niiden kehitystä kuten myös käsinkirjoitettujen merkkien käyttöä tiedonsyöttömenetelmänä tällaisissa laitteissa. Toinen osa kirjallisuustutkimusta puolestaan keskittyy erinäisiin käsinkirjoitettujen merkkien tunnistukseen liittyviin lähestymistapoihin. Menetelmiä sekä yksittäisille luokittimille että luokittimien yhdistämiseen komiteatunnistimeksi on tutkittu pääpainon ollessa menetelmillä, jotka soveltuvat reaaliaikaiseen tunnistukseen.

Käytännön osuus koostuu myös kahdesta osakokonaisuudesta. Niistä ensimmäinen keskittyy tunnistimen kämmentietokonetoteutukseen ja toinen Dynamically Expanding Context (DEC)-periaatteeseen pohjautuvan adaptiivisen komitealuokittimen toteutukseen ja testaamiseen. Adaptiivisen reaaliaikaisen käsinkirjoitettujen merkkien tunnistusjärjestelmä toteutettiin kämmentietokoneelle ja järjestelmälle luotiin tiedonkeruuseen ja tunnistimien toimivuuden tarkasteluun soveltuva käyttöliittymä. Käyttöliittymä on toteutettu kyselyn muodossa, johon käyttäjä vastaa käsinkirjoitettujen merkkien avulla. Myös toteutuksen vaatimia nopeutusmenetelmiä on tutkittu. DEC-komitea toteutettiin yhdistämään tuloksia vaihtelevasta määrästä luokittimia adaptiivisella komiteaperiaatteella. Sen suorituskykyä on arvioitu vertaamalla saatuja tuloksia toisiin, sekä adaptiivisiin että epäadaptiivisiin, komitealuokittimiin.

Kämmentietokonetoteutuksesta saadut tulokset osoittavat, että järjestelmä kykenee adaptiotumaan käyttäjän kirjoitustapaan. Myös nopeutusmenetelmillä saatiin huomattavia parannuksia tunnistusnopeuteen. DEC-komitealla tehdyt kokeet osoittivat sen saavuttavan merkittävästi sekä jäseniään että vertailuluokittimia paremman tunnistustarkkuuden.

Avainsanat:
hahmontunnistus, yksittäiset käsinkirjoitetut merkit, tosiaikainen tunnistus, adaptaatio, oppivat järjestelmät, komitealuokittelu, kynäpohjainen tietokone, kämmentietokone; k:n lähimmän naapurin menetelmä, oppiva vektorikvantisaatio (LVQ), Dynamic Time Warping (DTW), elastinen sovitus, Dynamically Expanding Context (DEC)

Acknowledgments

All research presented in this thesis has been conducted in the Laboratory of Computer and Information Science (CIS) in the Department of Computer Science and Engineering at the Helsinki University of Technology. Without the excellent resources available, obtaining the results presented here would not have been possible. Overall, the CIS laboratory has provided a very pleasurable working environment, for which my gratitude goes out to the entire staff.

First, I would like to thank Professor Erkki Oja for supervising my work and the effort he has put into making this project possible. I would also like to express my deepest gratitude and respect for Academician Teuvo Kohonen for the extensive research resources and traditions he has made possible. My thanks also go to Nokia Research Center (NRC) for the funding of this research.

The work presented here could not have been possible without the expert guidance and insight of Dr. Jorma Laaksonen, who functioned as the instructor for this thesis as well as a coordinator of efforts in the On-line Recognition of Handwritten Characters-project. Without his help and knowledge many of the problems encountered during this work might have never been solved. Also the guidance of Docent Jari Kangas from NRC has been vital to the completion of this thesis. The other members of this research project, M.Sc. Vuokko Vuori from CIS laboratory and Mr. Jukka Yrjänäinen from NRC have also provided great help and support.

Otaniemi 22nd May 2000

Matti Aksela

Contents

List of Abbreviations

AIME	Adaptive Integration of Multiple Experts
ASSOM	Adaptive-Subspace Self-Organizing Map
BKS	Behavior-Knowledge Space
CBL	Current-Best-Learning
DEC	Dynamically Expanding Context
DTW	Dynamic Time Warping
EM	Expectation-Maximization
FPU	Floating Point Unit
GSM	Global System for Mobile communications
HCR	Handwritten Character Recognition
HMM	Hidden Markov Model
KA	Kind of Area
k-NN	k Nearest Neighbor
LAN	Local Area Network
LVQ	Learning Vector Quantization
MCBL	Modified Current-Best-Learning
ML	Maximum Likelihood
MLP	Multilayer Perceptron
NN	Neural Network
NPL	normalized point-to-line
NPP	normalized point-to-point
OCR	Optical Character Recognition
PAC	Probably Approximately Correct
PDA	Personal Digital Assistant
PDL	Picture Description Language
PL	point-to-line
PP	point-to-point

ppmm	points per millimeter
RAM	Random Access Memory
ROM	Read Only Memory
SA	Simple Area
SCM	Supervised Clustering and Matching
WAP	Wireless Application Protocol
WIN32	32-Bit Windows
WinCE	Windows CE
wpm	words per minute

List of Symbols

$\overline{x}$	input feature vector
ω_j	pattern class number j
M_{ω_j}	mean of feature vectors in class ω_j
Σ_{ω_j}	covariance matrix of feature vectors in class ω_j
$G(\overline{x}\vert n, s)$	nth Gaussian for the feature vector $\overline{x}$ in state s
V_N	set of non-terminal symbols
V_T	set of terminal symbols
P	set of writing rules
$\#$	blank symbol
X'	left-right flip of the string X
$\mathrm{OCC}(c, X)$	number of occurrences of character c in X
$\mathrm{REM}(X, n)$	string obtained from X by removing its first n elements
D_k	total distance between prototype k and the input
$d(\cdot)$	distance function
$w(\cdot)$	warping function
ϕ_i	slope angle of the tangent to the curve at the point i
$a_{h,k}$	fuzzy activation value
μ_h^H	horizontal fuzzy set triangular membership function
μ_k^V	vertical fuzzy set triangular membership function
Δ_{ji}	hardening of probabilities $P(\omega_j\vert\overline{x}_i)$ to binary values
N	total number of classifiers
$P_c(N)$	probability of a consensus of N classifiers being correct
p_e	classifier error rate
Λ	set of output labels of a classifier
$E(\overline{x})$	final decision of committee
$P_E(\overline{x} \in \omega_j\vert\overline{x})$	probability of $\overline{x}$ belonging to the class ω_j in the combiner
$d_k(\overline{x}, \omega_j)$	distance of $\overline{x}$ to ω_j with measure k

$P_k(\overline{x} \in \omega_j \vert \overline{x})$	probability of $\overline{x}$ belonging to class ω_j in classifier k
$f_j(\overline{x}, w_j)$	output of the jth expert
L	log likelihood
$F_R(e_k(\overline{x}))$	rank of the output from recognizer number k for the input pattern $\overline{x}$
$r_k^i(\overline{x})$	the output rank by recognizer k for class i and input $\overline{x}$
$C_F(i)$	number of votes for class i
$BEL(m)$	the Bayesian probability for m
C	total number of classes
b_j	assertment from critic number j
K	number of dimensions of a BKS
$e(j)$	output from classifier number j
$w_{jl}(\overline{x})$	critic's confidence in its experts decision
$BKS(e(1), \dots, e(K))$	the unit in the BKS where the classifiers give the outputs $e(1), \dots, e(K)$
$n_{e(1)\dots e(K)}(\omega)$	total number of samples belonging to class ω in BKS$(e(1), \dots, e(K))$
$T_{e(1)\dots e(K)}$	total number of incoming samples in BKS$(e(1), \dots, e(K))$
$R_{e(1)\dots e(K)}$	the best representative class of BKS$(e(1), \dots, e(K))$
λ	threshold parameter to control BKS solution reliability
l	length of the longer side of a character's bounding box
$P(j)$	time warping path point j
p_j	data point j
S_j	stroke number j
N_j	the number of points in stroke number j
$D_{\mathrm{PP}}(S_1, S_2)$	the point-to-point distance between strokes S_1 and S_2
$D_{\mathrm{NPP}}(S_1, S_2)$	the normalized point-to-point distance between strokes S_1 and S_2
$D_{\mathrm{PL}}(S_1, S_2)$	the point-to-line distance between strokes S_1 and S_2
$D_{\mathrm{NPL}}(S_1, S_2)$	the normalized point-to-line distance between strokes S_1 and S_2
(x_i, r)	the nearest point to the point y on the line between points x_i and x_{i+1}
r	location parameter
A_{SA}	area for polygons when the lines do not intersect

A_{KA}	Kind-of-Area distance
w_{add}	symbol string addition cost function
w_{rep}	symbol string replacement cost function
w_{rem}	symbol string removal cost function
d	number of discretization directions
ℓ	discretization distance threshold
λ_1	cost of an addition
λ_2	benefit of adding the same symbol as just one of its neighbors
λ_3	benefit of adding the same symbol as both its neighbors
λ_4	cost of adding a pen-up symbol
κ_1	factor for the effect of length differences in replacement
κ_2	cost of replacing with a pen-up symbol
ρ	ratio between the costs of additions and removals
$J(j)$	cluster splitting criterion function
$g(j)$	goodness value of prototype number j
$N_{corr}(P)$	count of the times prototype P belonged to the true class
$N_{err}(P)$	count of the times prototype P did not belong to the true class
$\overline{m}(t)$	reference vector at state t
α	learning coefficient
$\delta(i,j)$	Kroenecker's delta function
c_r	removal constant for stray point removal
$d_{est}(i)$	predicted total stroke distance at point i
d_i	the calculated column minimum distance from the dynamic programming algorithm at the point i
$D(S)$	total distance of matching stroke S
$x(A)y$	the context of A
$c_j(i)$	the confidence of classifier j in its decision for i

Chapter 1

Introduction

The basic problem of handwriting recognition has been studied for over a century. Initial excursions date back to as early as 1870, when automatic character recognition was proposed as an aid to the visually handicapped (Govindan 1990). Automatic on-line handwriting recognition has also been an actively researched problem already for four decades.

One could assume the recognition of handwriting not to be a really difficult task, as it is basically just a question of deciding which character of the alphabet the input matches with, or is most similar to. In theory, that is precisely the case. But even for Optical Character Recognition (OCR) systems dealing with machine-printed characters there exists great variation in sizes, alignments and shapes of possible fonts, more than enough to make the task definitely non-trivial.

As for Handwritten Character Recognition (HCR), the variation is even more diverse. It can safely be said that probably no two persons' handwriting styles are exactly identical. And as each human writes in several different manners depending on a number of factors such as haste, need for precision and the surrounding environment, the different ways of writing any particular character can be taken to be extraordinarily many.

Granted, there are some consistent features in handwritten characters that allow also human readers to distinguish between them. But it is also well known that the human brain excels in such recognition tasks (Wang and Gupta 1991), and surely everyone has come across simply indecipherable handwritten notes. This is accepted as simply someone having written in a way that cannot be read. But when the recognition is to be done by a computer, also extremely difficult forms of writing are in general expected to be recognized, or the user will be disappointed with the performance. Thus the task of automatic handwriting recognition is far from trivial.

1.1 Handwriting as an input method

With the introduction of small palm-top devices capable of most common tasks previously reserved for either desk or lap-top computers or the traditional pen and paper, a need to find suitable, natural ways of inputting data have arisen. As the size of a Personal Digital Assistant (PDA) device is too small for traditional keyboards to be practical, handwriting has been taken on as a viable alternative. Handwriting is a good way of data input, as it is consistent with what people are used to. It can be assumed that everyone using a computer these days has also learned to read and write, so using handwriting as the means of data input would be quite natural.

It has been a long-time dream for pattern recognition enthusiasts to obtain a recognition accuracy good enough for the pen-based input to even to an extent replace the traditional keyboard. But still recognition accuracy remains the main problem. When an experienced typist only tolerates an error of up to 1% (Guyon and Warwick 1996), the recognition accuracy for isolated handwritten characters by humans has been observed to be in the range of 94.9% to 96.5% (Neisser and Weene 1960). It has also been found that the user acceptance threshold for handwriting recognition accuracy is approximately 97% (Chang and MacKenzie 1994). This can be interpreted as the need for handwriting recognizers to at the very least reach, if not surpass, human accuracy before the recognition accuracy is deemed acceptable by the users.

In addition, handwriting input speed is in the range of 15-18 words per minute (wpm) (Guyon and Warwick 1996) whereas an average engineer can obtain much higher speeds of approximately 60 wpm when typing a memorandum (Ward and Blesser 1985). Thus it cannot be seen anywhere in the foreseeable future that handwritten input would replace the keyboard in a situation where space is not a concern but speed and accuracy of input are. But for some applications where space is an issue, as in palm-top devices, handwriting input is practically the only viable solution. Thus handwritten character recognition is much in need of continued research.

1.2 Aims and overview of this thesis

This thesis is focused on the adaptive on-line handwriting recognition system developed in the Laboratory of Computer and Information Science at the Helsinki University of Technology in co-operation with Nokia Research Center. The recognition system and its features are explained in Chapter 4.

The main objective of this thesis was two-fold. First, a survey and implementation for the palm-top platform were conducted. Second, a survey and research in adaptive committee recognition were performed.

The first part of the thesis consists of a literature survey. In Chapter 2, pen computing and palm-top devices, their features and development, are discussed. The history, current state, usability in terms of problems and benefits and future views of pen computing are examined. Then in Chapter 3 methods relating to handwritten character recognition are examined with focus on on-line handwriting recognition. Methods of recognition for both individual classifiers and classifier combination methods are included.

After outlining the principles of the on-line recognition system in Chapter 4, the implementation of the recognition system on a palm-top platform is discussed in Chapter 5. Some comparisons with the palm-top platform and an also previously used large-scale platform are made and the implemented user interface is explained. Also the results from some experiments are shown.

Experiments performed with an adaptive committee combination method based on the Dynamically Expanding Context (Kohonen 1986, Kohonen 1987) are discussed in Chapter 6. The principle behind and functionality of the implemented adaptive committee are explained and its performance is compared with that of some reference classifiers. Finally, in Chapter 7 main results are gathered and drawn conclusions explained.

Chapter 2

Pen computing

With recent developments in computer hardware it has become possible to manufacture computers not much larger than the human palm. Implementing regular keyboards with devices of such a small size is understandably very impractical. With extremely small keyboards typing becomes very cumbersome, and it requires extreme precision from the user to hit the desired keys. Thus pen-based input, and consequently handwriting recognition, seems to be the most logical form of human to computer information transfer for palm-top devices.

Palm-top devices have several advantages in comparison to their larger desktop counterparts, most notably portability and capability of being used nearly anywhere. An additional benefit is the possibility to connect palm-top devices to desktop computers and transfer data between one another making appointments scheduled or notes written more widely available, all without any need of retyping.

In this chapter the development of pen computing devices and their usual characteristics, both beneficial and restraining, are explored and examined.

2.1 Palm-top computers

The development of pen computing can be seen as having began when Alan Kay envisioned the Dynabook in 1968 (Meyer 1995). While this was the initial design for a pen-based computer, only a cardboard model was produced. A strong competitor for the title of the first commonly available palm-top device is the Psion Organizer in 1984 (Bray 1999), which lead the way for a multitude of products with varying success.

The Dynabook actually saw the light of day when a prototype very similar to the original design was presented by Apple in 1987. This was called the Knowledge Navigator. From the foundation of the Knowledge Navigator finally in 1992 appeared the Newton, which was the first in this series to reach mass production. (Meyer 1995)

The original devices were quite a distance from the current state of pen computing devices. Nowadays palm-top devices with color displays and tailor-made operating systems such as EPOC32 or PalmOS are commonly available (Bray 1999), but still many of the original problems persist.

According to Bray (1999) the choice of hand-held PCs is increasing constantly but the struggle for the dominance in operating systems is still going on between EPOC, PalmOS and WinCE. It was noted that WinCE devices were the first ones to be equipped with color displays. Also significant improvements to the speed and battery life of the WinCE devices have recently been seen. But still the large and loyal user base of the 3Com palm-top devices enables them to compete with WinCE-based solutions. In the review, the Casio Cassiopeia E-105, a palm-top running on a 131 MHz NEC MIPS R4000 processor which makes is the fastest-running WinCE palm-top device, was deemed the best of the currently available palm-top computers (Bray 1999).

A comparison of some features of selected palm-top computers available is shown in Table 2.1 (Bray 1999, Everex 2000). Only differentiating characteristics have been entered into the table. All the devices featured a docking cradle for PC connectivity, data synchronization software, stylus-based input, infrared ports and the possibility for mobile phone connectivity with additional equipment. The difference in approach for the PalmOS and WinCE-based devices is clear. The PalmOS-based palm-tops are designed as basic PDAs with limited functionality while WinCE-based devices in general offer more functionality. The cost of the extra features in WinCE devices is significantly shorter battery life, and in all but the Everex device, larger size and additional weight.

2.2 Benefits of pen-based computers

The first obvious benefit of pen-based palm-top computers is their size. Currently several quite small and light-weight solutions are commercially available. The sizes of most solutions are acceptable in the manner that they are not much larger than an average calendar or notebook one might think of being replaced by them.

With the increasing networking and connectivity of computers, it is only natural for users to desire connectability also from their mobile devices. This is a feature that has generally been very well implemented for pen-based computers. The connection methods include serial or parallel cables, infrared links, device-specific docking stations and wireless local area networks (LAN) (Meyer 1995). Currently also telephone networking in the form of both modems using GSM networks and WAP applications is becoming

Table 2.1: A comparison of palm-top device features

Manufacturer	3Com	3Com	Casio	Compaq
Model	Palm IIIx	Palm V	Cassiopeia E-105	Aero 2130
OS	PalmOS	PalmOS	WinCE 2.11	WinCE 2.11
CPU type CPU speed	Dragonball 16 MHz	Dragonball E2 16 MHz	MIPS R4000 131 MHz	MIPS R4000 70 MHz
ROM	2 MB	2 MB	16 MB	12 MB
RAM	4 MB	2 MB	32 MB	16 MB
Display Resolution	Monochrome 160×160	Monochrome 160×160	Color 320×240	Color 320×240
Dimensions (mm) Weight (g)	120×80×15 172	115×77×10 115	131×84×20 255	134×85×20 260
HCR Software	Graffiti	Graffiti	Jot	Jot
Battery life	2-4 weeks	2-4 weeks	6 h	10 h
Voice activation Voice recorder	no no	no no	no yes	no yes
Modem supplied	no	no	no	no
Upgrade options	Internal memory	-	CompactFlash cards	-

Manufacturer	Everex	HP	Philips
Model	Freestyle Executive A20	Jornada 420	Nino 500
OS	WinCE 2	WinCE 2.11	WinCE 2.11
CPU type CPU speed	MIPS R4000 66 MHz	Hitachi SH-3 100 MHz	MIPS R4000 75 MHz
ROM	8 MB	8 MB	16 MB
RAM	16 MB	8 MB	16 MB
Display Resolution	Monochrome 320×240	Color 320×240	Color 320×240
Dimensions (mm) Weight (g)	122×82×16 155	130×81×22 250	132×86×19 227
HCR Software	Jot	Jot	Calligrapher
Battery life	7-8 h	4 h	8 h
Voice activation Voice recorder	no yes	no yes	yes yes
Modem supplied	33.6 K	no	no
Upgrade options	CompactFlash cards	CompactFlash cards	CompactFlash cards, 19.2K modem

commonplace adding yet another dimension to connectivity. Especially wireless LANs and telephone networking offer a great deal in the terms of freedom of movement for any portable device, as the connection can be established anywhere within the range of the network.

The third notable benefit, the use of a pen or stylus for input, is also the most problematic one. A pen is a very natural way for humans to input information, since writing is generally taught in schools and learnt at a young age, at least for people in developed countries, who coincidently are also the most likely to use such appliances. Also reasons based on human physiology attribute to the use of a pen as a very effective and precise method for input. For humans positioning the pen-tip is highly accurate due to the high number of degrees of freedom provided by the fingers and the relatively large portion of the motor-control areas in the brain dedicated to finger movement (Schomaker 1998). This can clearly be observed by anyone when comparing the precision of input while drawing with a pen in comparison to drawing with a regular mouse. Movement originating from the wrist is much more coarse as is the quality of the final output.

When comparing handwriting input with another viable option for palm-top computer input, speech recognition, it can be seen that handwriting offers several advantages. Most notable of these is the fact that handwriting is not sensitive to surrounding noise, that can be a real problem for speech recognition (Park and Lee 1999). In addition, writing input can be used in situations when speech is deemed inappropriate, for example in meetings. Speech recognition also poses problems when editing inputted text, which on the other hand is quite natural when using pen-based input (Mel et al. 1988).

2.3 Problems with pen-based computers

With pen-based computing and handwritten character recognition having gained a great amount of interest, why have such devices not become commonplace? The main problem is currently associated with the usability of these devices. This can mainly be attributed to two important aspects hindering user acceptability of pen-based computers, namely recognition accuracy and user interface problems (Schomaker 1994). Also various metaphors, both deliberately and accidentally laid on, can cause false expectations on the actual functionality of the device. One example of common misleading metaphors is the pen-and-paper metaphor.

2.3.1 Recognition accuracy

It has often been stated, that the mediocre quality of handwriting recognition has been the most notable obstacle to the success of pen computers (Guyon and Warwick 1996). While this certainly is a major problem, it can also be seen that even reaching human recognition levels may not be acceptable for handwriting recognition (Schomaker 1994). Actually recognition performance might not be the most important factor.

It has been noted that users are likely to judge the value of an application primarily on terms of its appropriateness for the task rather than any individual aspect of functionality (Frankish et al. 1995). Thus, even if the recognition accuracy is not adequate for writing a peer-reviewed paper, the device and application might be completely acceptable and useful for filling out forms, for example.

When comparing handwriting recognition with other viable pen-based input methods, it has been established that the use of an on-screen soft keypad still offers both higher input speeds and lower error rates, 23 wpm versus 16 wpm and 1.1% character errors versus 8.1%, respectively (MacKenzie et al. 1994). The keyboard used the standard QWERTY layout, but it was also noted that tapping on a keyboard with an ABC layout produced still better accuracy at 0.6% errors, but the input speed was notably slower at 13 wpm (MacKenzie et al. 1994). This can clearly be seen as an indication that there is still much room for improvement in recognizer performance. When clearly better figures are obtained with other input methods, the value of the comfortability and naturalness of handwriting might not be enough to tip the scales in its favor.

Another approach taken by many is to constrain the writing by defining a specific set of alphabets designed in a way as to minimize the possibility of confusion between letters. Examples of such are the unistrokes alphabet (Goldberg and Richardson 1993) and its commercial derivation, Graffiti by Palm Computing (Meyer 1995). These specific alphabet schemes can obtain recognition accuracies very near to 100%, but the significant drawback is the need for the user to learn the modified alphabet. This in turn rises the level of motivation needed to begin using such a device. In addition it can be expected that such alphabets are rather easy to forget during prolonged periods of disuse.

As can be seen, efforts should be made to optimize recognition performance for natural handwriting. This has the notable benefit of enabling a lower initial acceptance threshold by removing the need of learning a specific writing style. The improvement of the recognition performance can be seen as being still quite viable, as a multitude of recent research continues to show that improvements are still possible (Chan and Yeung 1998, Brakensiek et al. 1999, Vuori et al. 1999).

2.3.2 User interfaces

Even though the most prominent problem regarding pen-based computing is in the general opinion poor recognition performance, the fact remains that it alone is not responsible for problems in the use of pen-based computers. Another important factor in the usability of pen-based computers and their software is naturally the user interface. It has been stated that, aside from recognition accuracy, the lack of maturity of user interface technology is the primary factor holding back the acceptance of pen-based computers (Schomaker 1994).

The design of a functional interface for a pen-based application is by no means a trivial task. Even though the use of a stylus for both pointing and textual input is possible, problems may easily arise from determining which input type is intended by the user. For example, when the user draws a horizontal line on some text written, is the intention to activate the text or append a dash?

Such problems can be resolved by constraining the handwriting input to specific areas of the user interface (Chang and MacKenzie 1994, MacKenzie et al. 1994, Meyer 1995). Although in the optimal case there should be no constraints on what and where the user writes and text should be writable along with non-textual input like graphics and gestures (Meyer 1995), in practice this is understandably very problematic.

The user interface should not be a handicap aside recognition performance, but rather it should be used to improve on and compensate for the defects in the recognition process. As a matter of fact, four methods of helping the recognizer perform better through the user interface can be stated (Schomaker 1994). These methods are:

1. Constraining the writer where possible
2. Giving the writer more control
3. Providing more powerful mechanisms for error handling
4. Clarifying to the user what is going on

The first approach includes, for example, forcing the user to write in boxes or combs for isolated hand-printed characters, or the use of specific gestures, an ”ok-button” or time-outs for input validation in the case of word recognition. This can be very beneficial for recognition accuracy since the need for specific segmentation, a task far from trivial in itself, is removed. Giving the user more control indicates that developers should include new methods for handling input material which is unacceptably poorly recognized, for example punctuation marks in many cases. This can be solved by offering the possibility of using tool bars for such symbols. Recovery from errors is also a problem which requires the inclusion of a deep "undo" stack. The last important point mentioned is giving the user more information on the actual occurrences inside the system. Thus, the interface must allow input validation by the user in an unintrusive manner. (Schomaker 1994)

Error recovery is an especially difficult problem for adaptive recognizers, as errors are the most prominent cause for mislearning leading to unnecessary errors. Also discriminating between real errors and changes of mind can be very problematic when functioning with an adaptive recognition system.

In general the user interface is perhaps at a tie with recognition performance for the title of the most important factor in the usability of pen-based computers. As such much more attention should be given to thorough user interface implementations for such devices.

2.3.3 The pen-and-paper metaphor

Ever since the first explorations into the world of pen computing, the pen-and-paper metaphor has been very prominent in assessing pen-based computers. This traditional comparison continues to cause confusion. Paper is still the most popular medium for sketching, note taking and form filling, as it possesses some unrivaled features including, but not limited to, its cheapness, availability, reliability and ease of use. But in larger quantities paper does have its problems, it is much harder to store masses of paper than files on a computer. Also editing previous handwritten notes can be quite tedious. (Guyon and Warwick 1996)

The general perception of the pen-and-paper metaphor indicates a great degree of freedom of use, and the use of this metaphor can give rise to the comparison between a pen-based computer and a rather unco-operative piece of paper (Schomaker 1998). The fact is simply that a computer processes information and is not merely a medium for sketching. As such pen-based computing poses constraints but also offers benefits completely out of the range of paper.

2.4 Palm-top HCR systems

Claims on effective handwriting recognition systems for the palm-top devices have also been made but sadly very few scientific publications or other reliable data could be found. For example Advanced Recognition Technologies Inc. boasts very high out-of-the-box recognition accuracy and adaptation to the users style of writing during use (ART 2000). But, as the recognition system is commercial in nature, no information on how the actual recognition is performed is given. It is merely stated that no dictionary is used, but the "unique linguistic layer enhances accuracy for the English languages" and that the recognizer adapts during use through a special algorithm (ART 2000). With a quick test, the system seems functional and adaptive.

Another example of available commercial recognizers is Jot from Communication Intelligence Corp. This recognizer has gained wide distribution, as it is the standard recognizer on palm-size PCs. The Jot recognition systems uses writing in different areas for different character sets to enhance performance. The commercially available Jot Pro offers also a specific trainer to enable the adaptation of the recognizer and the possibility for the user to write anywhere on the screen. (CIC 2000a, CIC 2000b)

Although commercial handwriting recognition systems are still not very numerous, the ones encountered are promising forerunners of functioning handwriting recognition-based input methods available to the general public. It can be expected that the variety and features will increase and be enhanced as competition in the sector increases.

2.5 Future views

It is clear that pen-based computers need to be approached as a field totally its own; the devices have their own merits and their own problems. As long as users have incorrect or unrealistic expectations for the functionality of pen-based computers, they will inevitably be disappointed. This is a dilemma that can only be corrected by either fulfilling these expectations or lowering them by providing a correct frame of reference.

Also the fast pace of progress in microprocessor technology will undoubtedly offer benefits of increased computational power and lower power consumption also to palm-top devices. A noteworthy recent release in the processor market is the Transmeta Crusoe, which would seem to be able to provide very competitive performance especially when scaled by power consumption (Laird 2000).

With the perennial growth of computational power even more complex and better performing algorithms can be implemented also on small platforms. Thus it is evident that research on handwriting recognition should be continued with the objective of creating new algorithms and optimizing old ones for higher recognition performance rather than fulfilling the speed requirements of current hardware (Guyon and Warwick 1996). As the computational speed increases, algorithms currently seen as being too complex will shortly cause no complications in implementation.

But even when acceptable recognition rates will be obtained, the speed of input will inevitably be much lower than that obtainable with traditional keyboard input. This necessitates the need to implement other effective methods of assisting the completion of common tasks. Especially tasks requiring precision such as password entry for user identification would benefit greatly from for example using signature verification instead of traditional passwords, as proposed for the PCS Smart Phone (Narayanaswamy et al. 1999). Such simple additions augmenting the usability of a pen-based device may just be one of the deciding factors in their advance to user acceptance.

Chapter 3

Handwritten character recognition

Systems for handwriting recognition can be examined from several perspectives. One major distinction can be made between systems performing the actual recognition either during the writing process (on-line) or from data collected earlier (off-line). Another important distinction can be made between writer dependent and writer independent recognition approaches. The former refers to a situation where the recognition is for example adapted to suit the style of the particular writer even at the cost of generic capabilities. Writer independent classifiers naturally do not develop such characteristics.

Other viewpoints include distinguishing between systems using a single classifier and ones combining several more or less distinct classifiers. Individual classifiers can also have many fundamentally different approaches to the recognition task. Also several varying approaches to representing the data can be isolated. These approaches to handwriting recognition, as well as some aspects of variation in handwriting, will be dealt with in the following sections.

3.1 Time of recognition

Perhaps the most prominent and obvious distinction to be made for recognition systems is regarding the time when the actual recognition is performed. The use of off-line or on-line recognition is usually dictated by the application, but methods from both fields can also be combined to enhance recognition performance (Guberman 1998, Tanaka et al. 1999).

3.1.1 Off-line handwriting recognition

Off-line handwriting recognition can be viewed as a direct subset of Optical Character Recognition (OCR), since recognition is in practice performed from images of written characters (Tappert et al. 1990). In off-line operation no feedback from the user, speed, pressure or directional information is generally available for the recognition process. In most cases the data in use is that what can be obtained by scanning writing from a piece of paper. When off-line recognition is performed, the recognition process by default deals with less data than its on-line counterpart. This in turn enables the use of computationally more expensive algorithms in all but time-critical systems dealing with extremely large quantities of data, such as postal address reading systems. Also, in off-line recognition the writing order or overwriting of strokes does not confuse the classifier, which is in contrast to on-line recognition where these are notable problems (Tanaka et al. 1999).

Off-line recognition has several useful applications, for example in postal address reading, processing documents to a computer readable form, automatic writer recognition and signature verification. OCR machines have been commercially available since the middle of the 1950s. (Govindan 1990)

3.1.2 On-line handwriting recognition

On-line recognition has some benefits over off-line recognition, mostly due to the extended amount of information that is obtainable. In addition to the spatial properties of the loci of the stylus also the number of strokes, their order, the direction of writing for each stroke and the speed of writing are available (Tappert et al. 1990). On applicable writing surfaces the pen pressure can also be measured. Another notable advantage for on-line operation is interactivity, which enables instant recognition mistake correction and also more effective adaptation of the system (Tappert et al. 1990).

Due to the abovementioned factors it can be expected that on-line recognition provides better recognition accuracy than off-line. This has been shown in experiments by Mandler et al. (1985), where the Euclidean distance measure was used with dynamic information. In comparison to rasterized images the worst and best writer final error rates decreased from 16.6% to 11.4% and 4.5% to 1.2%, respectively. Another study using the same base data for off-line and on-line methods states recognition rates of 73.8% and 84.8%, respectively (Tanaka et al. 1999). Further discussion is focused on on-line handwriting as that is in consensus with the nature of the application in question in this thesis.

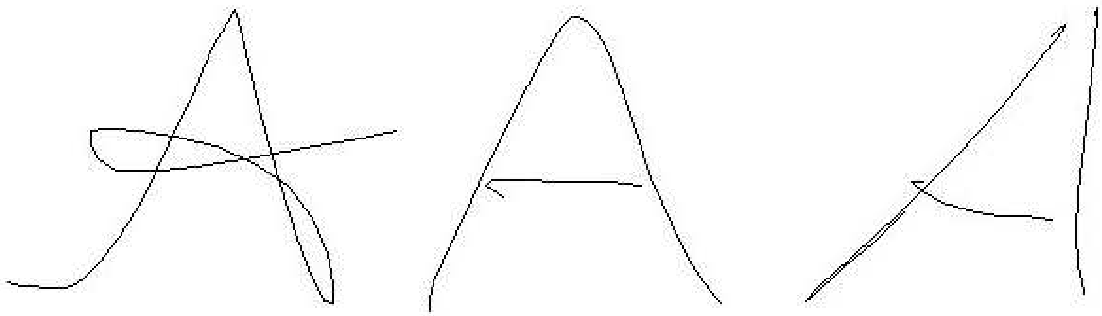


Figure 3.1: Three examples of the character "A" with a varying number strokes

3.2 Variation in writing

As is intuitively known, there are probably as many ways of writing as there are writers. In addition to individual writing styles, also several completely different character sets are widely used. It is imperative to be aware of the existence and possible manifestations of variation in order to be able to design a natural handwriting recognition system with efficiency. Here some commonly identifiable sources and forms of variation in handwriting are examined.

3.2.1 Strokes

Especially in on-line handwritten characters it is very common to consider a character as consisting of a varying number of elements of writing called strokes. In this thesis strokes are considered to be parts of the character separated by pen-ups, ie. when the pen is lifted from the writing surface. In Figure 3.1 three examples of the character "A" are presented. The leftmost example has been written with one stroke, as the pen has not been lifted during writing the letter. The second and third examples consist of two and three strokes, respectively.

The number and direction of the strokes is a significant cause for variation in handwritten characters in on-line recognition. For example, the character "E" is commonly written with an number of strokes ranging from 1 to 4. As each of them can also be written in either direction and in any order, it instantly produces $8 * 6 * 4 * 2 = 384$ different permutations for characters with no apparent difference when seen written.

3.2.2 Character sets

The existing character sets are inherently very different in both their appearance and features. Recognizers have been implemented at least for Latin, Japanese, Chinese, Indian, Arabic, Korean, Cyrillic, Hebrew, Thai, Greek and Berber characters. (Govindan 1990)

Of these perhaps the most challenging are Chinese characters due to the sheer amount of characters in use: approximately 5000 are commonly used of a total of over 50000 (Govindan 1990). Another challenge is posed by the fact that Chinese characters contain an average of 8–10 strokes, the most complicated having more than 30, and the characters can be written in either block or cursive style causing variation even in the number of strokes of the character (Tappert et al. 1990). In comparison, the average number of strokes for Latin uppercase letters is 2 and for lowercase letters one (Tappert et al. 1990). Another problematic writing system is Japanese, since they daily employ up to 5 different character sets, namely Hiragana, Katakana, Kanji, Roman letters and Arabic numerals (Govindan 1990).

Each character set has its general features, which demand the need for specialized recognition techniques. For example variation in stroke direction and retrace is more common in English than in Chinese (Tappert et al. 1990). Thus the target character set has a great effect on the desired properties of a given recognition system. In further discussion Latin characters will be focused on since the application at hand only deals with this set of characters.

3.2.3 Writing style

Variation in writing style for individual characters can be grouped to variations in both the static and dynamic properties of the written characters. Static properties are, for example, size and shape, and dynamic properties include for example the number and ordering of strokes. (Tappert et al. 1990)

Variation in the size of characters is very common in different mediums. For example it is very natural to write small letters in small boxes whereas the size is much more prone to vary when no such guidelines are present. Overall, the size of the letters can be controlled greatly by the underlying material or form properties. The shape of the characters may vary depending on the writers origin; distinct styles can be identified for example for writers from North America or Europe (Nouboud and Plamondon 1990). Also affecting the shape of characters is the inherent tendency for nominally straight strokes to become curved when written by hand (Ward and Kuklinski 1988).

Several factors affect the number of strokes. It is generally acceptable that a specific archetype can be formed using a different number of strokes (Kuklinski 1984). This is mainly due to the human perception that the most important factor in writing is the final appearance, as humans generally only do off-line recognition. Also connecting strokes often occur due to pen lift failures, which are most common in sequential similarly aligned strokes or strokes whose endpoints are near one another (Ward and Kuklinski 1988). Stroke retrace is a problem effecting mainly on-line recognition, as it is not necessarily visible in the final appearance of the character. Two types of retrace can be identified, intentional due to not seeing the output, and connecting segment retrace which takes place when the pen is repositioned without lifting it (Ward and Kuklinski 1988).

It can be also be stated that in most cases writers tend to draw vertical strokes before horizontal ones and left components before right ones (Nouboud and Plamondon 1990). The order of writing strokes is also dependent on the handedness of the writer simply due to the muscle and tendon groups used in writing. Generally, it can be said that most left handers tend to draw horizontal strokes from right to left whereas right handers prefer writing strokes from left to right (Kuklinski 1984).

In addition to inter-writer variation also several writing styles are probable to exist for a single writer. One usually writes in a different way when making personal notes in comparison to writing down something very important for others to read. Especially the speed and motivation to carefulness can greatly affect the writing style (Kuklinski 1984).

Due to these factors the variation in characters is extreme. It has been estimated that the amount of identifiable ways to write just four stroke upper-case "E"'s would be approximately 23 thousand (Kuklinski 1984), and variants of the upper-case "A"'s have been predicted to near 16 thousand (Ward and Kuklinski 1988).

3.3 Data preprocessing

Handwriting data is usually collected using either electromagnetic/electrostatic or pressure-sensitive tablets onto which the characters are written using a stylus. In on-line recognition the data is most commonly collected into the form of a vector of coordinates, possibly including also pressure information. The points of the vector are equidistant in time due to most data collection devices having a fixed sampling rate. (Nouboud and Plamondon 1990, Tappert et al. 1990)

Due to both the intrinsic variation in handwriting data and inadequacy in the precision of data gathering instruments there is usually need for preprocessing and normalization of gathered data. Preprocessing is mainly used to simplify the tasks of the recognition algorithms (Guerfali and Plamondon 1993). The purposes can be divided into three sub-categories; reducing the amount of information, eliminating imperfections and normalizing handwriting (Guerfali and Plamondon 1993).

As tablets and other data collection devices generally sample information at a fixed frequency, also redundant data is produced. The most obvious case of such are duplicate points, which generally provide no information of interest. Also points closer than a prespecified threshold are often removed, sometimes taking into account regions of greater curvature and avoiding removal in them. (Guerfali and Plamondon 1993)

Re-sampling the data to be equidistant in space rather than time is another common approach (Nouboud and Plamondon 1991, Nathan et al. 1993, Bellegarda et al. 1994, Hu et al. 1996, Rigoll et al. 1996, Connell and Jain 1999). This is a two-edged sword, in some cases it can ease recognition but equidistancing the points also results in the loss of pen speed information. Especially information on the variations in writing speed might be useful in detecting meaningful points of strokes. Generally equidistant sampling will be more beneficial in writer independent systems due to individual writers often writing with different speed characteristics (Bellegarda et al. 1994). This naturally results in the speed information being beneficial especially in writer recognition or identification applications (Yuen 1996).

Imperfection elimination is necessitated by the fact that data acquisition is rarely sufficiently precise. One commonly used method is smoothing, which is used to eliminate hardware problems or erratic hand motion (Guerfali and Plamondon 1993). Smoothing is usually performed by using some averaging scheme over neighboring points (Nouboud and Plamondon 1990, Guerfali and Plamondon 1993). Some examples of smoothing approaches include the use of a Gaussian smoothing filter (Connell and Jain 1999), a spline smoothing operator (Hu et al. 1996) and a mobile average filter (Nouboud and Plamondon 1991).

Another more extreme problem due to hardware inadequacies is the appearance of wild points, which are points a notable distance off the actual writing trace. They can usually be detected by high-velocity variations (Guerfali and Plamondon 1993). Wild point removal can be considered always useful, since the erroneous points do not contain any beneficial information. This preprocessing method is present in many studies (Bellegarda et al. 1994). Thresholding with limits based on hand motions is generally suitable for wild point removal (Guerfali and Plamondon 1993). Also a spatial filter that removes points too far from surrounding points can be successfully used.

One type of imperfection removal is the removal of component connections and hooks at the beginnings or ends of characters. The former manifests itself mainly due to imprecision in pen-up and pen-down detection of the data gathering device (Guerfali and Plamondon 1993). Hooks neither contain any additional information and are thus beneficial to remove, as often has been done (Bellegarda et al. 1994).

In most cases the area of digitization is larger than the actual letters, so moving the letters to the same origin is an often-used and beneficial step. Commonly used methods are to subtract the mean from all coordinate points or move the center of the bounding box of the character to the origin, which is also called bounding box subtraction. (Srinivasan and Ramakrishnan 1999, Laaksonen et al. 1998b)

It has been stated that size invariance is the key to robust recognition. Three basic approaches to normalization can be identified as multirate-based normalization, ratio-based normalization and simple scaling normalization. In multirate-based normalization signal re-sampling is based on multirate filter theory and can be implemented by a cascade of interpolation and decimation filters. Ratio-based normalization is implemented by stretching or compressing the image by a given ratio. Simple scaling normalization on the other hand is comprised of using the bounding box character and scaling the bounding box, and thus the also the character, to a predefined size. (Srikantan et al. 1995)

3.4 Prototype selection

When using a classifier based on prototype matching, selection of the prototypes is critical to recognition accuracy. Some important characteristics for efficient prototypes are (IBM 1991):

1. Sufficient coverage
2. Reasonable separation in prototype space
3. Each prototype should be a good representation of a way of writing a character
4. Mavericks should be avoided

Sufficient coverage dictates that the set of prototypes should include at least one prototype for each distinct way of writing a character, which is not an easy requirement to fulfill due to the extreme variation possible (Kuklinski 1984). Mavericks are prototypes formed from distorted characters and as such have very poor generalizational capabilities. Such prototypes can be very harmful to recognition, as they are most likely to cause only false recognitions. The problem arises when attempting to detect mavericks; simply rarely used prototypes might be accurate, though rare, representations, and eliminating such will hamper recognition accuracy.

3.5 Recognition methods for individual recognizers

Here some approaches to character recognition for individual classifiers are discussed. The techniques have been divided into statistical, syntactic and structural, time warping, neural network and fuzzy recognition methods due to the fundamental differences between the approaches.

3.5.1 Statistical methods

The general approach in statistical classification is to choose the class with the highest probability of correctness for the input sample. Some ways of decision rule formulation for statistical classifiers include converting the *a priori* probability of a class $P(\omega_i)$ into the *a posteriori* probability $P(\omega_i|\overline{x})$ and formulating a measure of expected classification error, or risk, and a decision rule to minimize that risk (Schalkoff 1992).

Perhaps the most common statistical approaches are methods related to Bayesian decision rules (Cao et al. 1992, Cheung et al. 1998). One example by Arakawa (1983) uses Fourier coefficients of pen-point movement loci in strokes as feature vectors. The actual classification is then based on the Bayesian decision rule by assuming that the distribution of the feature vector $\overline{x}$ consisting of the Fourier coefficients of all the characters strokes is multivariate normal and using a discriminant function of the form

$$g(\overline{x}, \omega_j) = \frac{1}{(2\pi)^{n/2}|\Sigma_{\omega_j}|^{1/2}} \exp\left[-\frac{1}{2}(\overline{x} - M_{\omega_j})\Sigma_{\omega_j}^{-1}(\overline{x} - M_{\omega_j})^T\right], \tag{3.1}$$

where M_{ω_j} is the mean and Σ_{ω_j} the covariance matrix of the feature vector $\overline{x}$ in class ω_j. In that study two discriminant conditions were used. First, it was assumed that there is no correlation among strokes composing a character, but the Fourier coefficients expressing each stroke have a correlation among the components. In the second discriminant condition, component variances in a feature vector are taken into account but the covariances between components are neglected.

This approach was tested with 25 writers in the set of learning samples and 10 other writers were used for testing. Each subject wrote five characters for each category. The recognition rate was 99% for the first discriminant form and 97% for the second.

As is with most methods, statistical recognition can also be combined with other approaches. For example Loy and Landay (1982) used a structural, chain-code-based, approach to prune the potential matches. After this a statistical classification scheme based on Gaussian distributions was used to decide between various classes in the category. With both the learning and test samples written by the same writer, final recognition rates of above 98% were obtained.

Hidden Markov Models

The concept of the Hidden Markov model (HMM) was introduced already in the late 1960's but has become increasingly popular in the 1980's. The attribute "hidden" in the name originates from the fact that a HMM is a doubly embedded stochastic process in which the underlying stochastic process is not observable. (Rabiner 1989)

According to Rabiner (1989) there are three problems in constructing a HMM useful in real-world applications. First, given a model and a sequence of observations, how can the probability that the observed sequence was produced by the model be calculated? This is called the evaluation problem, and the most practical solution is to test the available models and choose the one that best matches the observations. Second, how is a state sequence that is optimal in some meaningful sense chosen? To this problem no real answer can be given, as the hidden nature of HMMs inhibits the detection of the correct sequence in all but the most degenerate models with a single state. In practice this can be solved as well as possible by using a optimality criterion. Unfortunately, due to the sheer amount of possible criteria, choosing the right one, while of utmost importance, is no easy task. Finally, how should the model parameters be adjusted to maximize the probability of the observation sequence? This can be done by using various algorithms for training the models.

The use of HMMs in handwriting recognition has followed from successful speech recognition applications (Rabiner 1989). Speech recognition is in principle quite similar to connected cursive script recognition; both deal with a continuous signal over time, the items to recognize are well defined and the realized shape of a single object, whether a phoneme or character, depends on its neighbors (Starner et al. 1994). Thus it is only natural to apply method from one field to the other.

Usually HMMs are created for each allograph, or model for a way of writing a character, sufficiently represented in the training set (Nathan et al. 1993, Bellegarda et al. 1995). According to Brakensiek et al. (1999) HMM modeling approaches can, in general, be divided into three categories. The most commonly used approach is the continuous HMM. In the continuous form the feature vector $\overline{x}$ emission probability in the state s is usually described by a mixture of Gaussians,

$$p(\overline{x}|s) = \sum_{n=1}^{N} p(n|s) \cdot G(\overline{x}|n, s), \tag{3.2}$$

where $p(n|s)$ is the weighting factor for the nth Gaussian $G(\overline{x}|n, s)$ (Brakensiek et al. 1999). Especially in small databases the interpolating effect of continuous HMMs can be very beneficial (Rigoll et al. 1996).

The second approach mentioned by Brakensiek et al. (1999) is to use discrete HMMs by first processing the feature vector by a vector quantizer and approximating the probability with the label generated by the quantizer. Discrete parameterization helps reduce the amount of training data needed at the cost of the loss of information during the quantization step (Bellegarda et al. 1995). It has also been noted that discrete models can, surprisingly enough, lead to better results than their continuous counterparts when the size of the database increases (Rigoll et al. 1996).

The third approach is to use a hybrid method, where the HMM can be augmented by, for example, a neural network working as either the approximator of the probability function in the continuous case, or the vector quantizer in the discrete case. Such an approach has been explored by Brakensiek et al. (1999), where a neural network vector quantizer was used. Rather encouraging results of error rate reductions by 40% in on-line recognition and 20% in off-line recognition were received when replacing the k-means-based vector quantizer with a neural-network-based one.

3.5.2 Syntactic and structural methods

A major benefit of using syntactic or structural approaches to recognition comes from their hierarchical nature. They inherently divide a large, complicated pattern recursively into smaller sub-patterns until given primitives are found. This enables the use of powerful data structures using grammatical rules, trees or directed labeled graphs. This approach is intrinsically similar to powerful artificial intelligence problem solving approaches, where a large, complicated problem is divided into subproblems for easier solving. This methodology can be considered closest to the intuition of humans, a merit based on the fact that humans possess the best pattern recognition skills when dealing with handwriting recognition. (Wang and Gupta 1991)

According to Wang and Gupta (1991) common methods for describing the relations between line-drawing patterns include the Picture Description Language (PDL), tree grammars and array grammars. In principle PDL consists of labeling each primitive at two distinguishing points, the tail and the head. The primitive can be linked or concatenated to other primitives only at either of these points. The primitives consist of short and long straight lines in different orientations. An example for the character "A" is shown in Figure 3.2.

Strings can be generalized to trees by extending one-way concatenation to multiple linkings. If a pattern can conveniently be described by a tree, it is also easily generated by a tree grammar. An example of tree grammars can be found in (Wang and Gupta 1991).

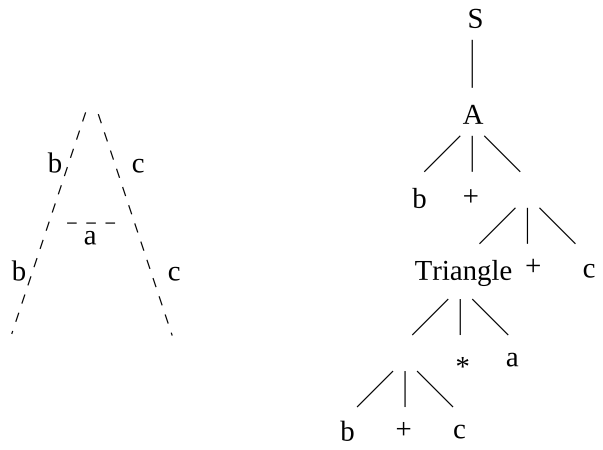


Figure 3.2: Naming of the PDL symbols and the PDL expression for "A"

An isometric array grammar is a quintuple

$$G = (V_N, V_T, P, S, \#), \tag{3.3}$$

where V_N is a finite non-empty set of non-terminal symbols, V_T is a finite non-empty set of terminal symbols, $V_N \cap V_T = 0$ and $\# \notin (V_N \cup V_T)$ is the blank symbol. P is a finite non-empty set of writing rules $\alpha \rightarrow \beta$, where array α and array β are geometrically identical over $V_N \cup V_T \cup \{\#\} : \alpha$ not all $\#$'s, and $S \in V_N$ is the starting symbol.

The distance between two arrays over V_T can be defined as the smallest number of error transformations (insertions, substitutions and deletions) required to derive Y from X. An example of this process is shown in Figure 3.3. The distance definition is generally not sufficient for efficient and unambiguous calculation of the distance, so further refinements in the forms of augmented grammars have been developed. (Wang and Gupta 1991)

Many structural methods are based on the existence of a predefined set of primitives to which the character is attempted to be divided. Commonly used primitives include the five Berthod and Maroy primitives (Wang and Gupta 1991):

1. straight line
2. positive (counterclockwise) curve
3. minus (clockwise) curve
4. pen lift
5. cusp

```
                      b
  a a a a a           a   a a a
      a                   a
X =   a       and Y =     a
      a                   a
      a                   b
```

```
                          b                     b                        b
      a a a a a           a a a a a             a   a a a                a   a a a
          a                   a                     a                        a
X =       a       after       a       after         a       after            a       =Y
          a       addition    a       deletion      a       substitution     a
          a                   a                     a                        b
```

Figure 3.3: Array grammar example of the derivation of Y from X

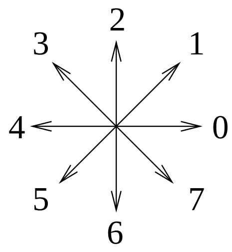


Figure 3.4: Freeman's octal chain code primitives

In addition to these, some new primitives have been introduced and used. They include quarter-circles, semi circles, full circles and points. (Chan and Yeung 1998, Khan and Hegt 1998)

To perform the actual difference calculation, in most cases matching methods deal with the representations directly. Due to variation in handwriting, finding absolute matches is in many cases impossible, and thus several matching schemes which allow variation have been developed. Two main approaches are elastic structural matching schemes (Chan and Yeung 1998, Khan and Hegt 1998, Sherkat et al. 1999) and deforming models (Filatov et al. 1995, Cheung et al. 1998). Actually the process is basically very similar in both cases, either the inputted characters or models are modified according to a defined set of rules and the amount or total cost of the modifications is calculated to be the difference between the matched characters.

A commonly used subset of structural representations is Freeman's code and its extensions and modifications (Wang and Gupta 1991). The original approach is to code the given character as a symbol string by using the eight directions depicted in Figure 3.4. An example of a Freeman-coded 'C' with the resulting code word 34566701, are shown in Figure 3.5. The downside to the use of basic equidistant chain code methods is the loss of the dynamic velocity data of the writing (Loy and Landay 1982).

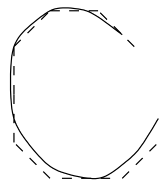


Figure 3.5: The character 'C' and it's Freeman's code word 34566701

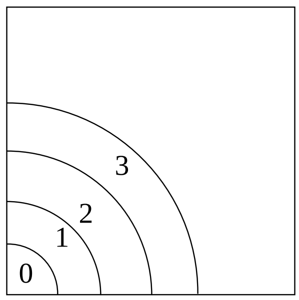


Figure 3.6: Position coding used by Nouboud et al.

One approach to chain code used by Nouboud and Plamondon (1991) uses an adaptation of Freeman's code using 12 evenly spaced directions and an additional scheme for coding positions (Figure 3.6). The string comparison is performed by a specialized string comparison processor, the PF474. The comparison value between the words S and T is

$$\operatorname{comp}(S,T) = \frac{2C(S,T) + 2C(S',T')}{|S|^2 + |S| + |T|^2 + |T|}, \tag{3.4}$$

where

$$C(S,T) = \sum_{n=0}^{\infty} \sum_{c \in CS} \min(\mathrm{OCC}(c, \mathrm{REM}(S,n)), \mathrm{OCC}(c, \mathrm{REM}(T,n))) \tag{3.5}$$

and $|X|$ denotes the length of X, X' the left-right flip of X, $\mathrm{OCC}(c,X)$ is the number of occurrences of character c in X and $\mathrm{REM}(X,n)$ is the string obtained from X by removing its first n elements. In a test with 4 writers and 59 character classes, an average recognition accuracy of 96% was obtained when the position code was not used. This improved to 98% with the addition of positional coding.

3.5.3 Time warping

Time warping is another method of elastic matching that originates from speech recognition (Kurtzberg and Tappert 1981). Time warping is often used as a distance calculation method for template matching (Lu and Brodersen 1984). Generally a sequence of metric points is constructed for the input character and it is matched against those of the prototypes. D_k, the overall distance between prototype k and the input, can be obtained as (Kurtzberg and Tappert 1981)

$$D_k = \min_{w(i)} \sum_{i=0}^{N} d(i, w(i); k), \tag{3.6}$$

where w is a warping function that maps the time index of the input model to that of the prototype and d is the distance function. Equation (3.6) can be efficiently and optimally solved with dynamic programming by using the recursion relation (Kurtzberg and Tappert 1981)

$$D(i, j; k) = d(i, j; k) + \min \begin{cases} D(i-1, j; k) \\ D(i-1, j-1; k) \\ D(i-1, j-2; k) \end{cases}, \tag{3.7}$$

where $D(i, j; k)$ is the cumulative distance up to the point (i, j), and thus the overall distance $D_k = D(N, w(N); k)$. By starting with $D(1, 1; k) = d(1, 1; k)$ and $D(i, j; k) \to \infty$ elsewhere the minimum of all possible paths through the model can be obtained.

An example of this approach from Tappert (1984) used four parameters to describe each point. The parameters were the slope angle ϕ_i of the tangent to the curve at the point i, the height of the point y_i, measured from the current baseline, and the horizontal and vertical offsets from the center of gravity, $\overline{x}_i$ and $\overline{y}_i$, respectively. Based on these parameters the point distance was defined as

$$d(i, j; k) = \min \{|\phi_i - \phi_{j,k}|, 360^\circ - |\phi_i - \phi_{j,k}|\} + |y_i - y_{j,k}| + |\overline{x_i} - \overline{x}_{j,k}| + |\overline{y}_i - \overline{y}_{j,k}| \tag{3.8}$$

This approach was tested using 26 upper and 26 lower case Latin characters and 10 digits. Six writers were used and all wrote four specified text fragments of a total of 325 characters. The first text was used for initial training and the last one for measuring the accuracy with the middle two used for updating. An average recognition accuracy of 94.1% was obtained, which was increased to 98.3% when using only upper case, lower case or digits at a given time.

3.5.4 Neural Networks

Neural Networks (NNs), a commonly used approach in pattern recognition in general, are also a popular method in handwritten character recognition. NNs can be thought of as storing prototype data, but instead of using actual prototypes the information is implicitly stored in the weights of the individual neurons.

A notable benefit of this approach is the fact that neural networks can be used without preprocessing the data (Mozayyani et al. 1998, Zhang et al. 1999). The black-box nature of NNs is not entirely advantageous, as this also makes it nearly impossible to break down possible sources of errors. Also the need for a very large training set is a severe drawback to NN use, as especially adaptivity is very hard to implement.

A traditional approach is to use Multilayer Perceptron (MLP) networks and back-propagation for training, but this suffers from a slow convergence rate (Annadurai and Balasubramaniam 1996). Several improved schemes for faster training and better recognition results have been proposed, for example a supervised feed-forward fuzzy neural classifier (Annadurai and Balasubramaniam 1996), enriching artificial neurons with spatio-temporal coding (Mozayyani et al. 1998) and using an Adaptive-Subspace Self-Organizing Map (ASSOM) (Zhang et al. 1999).

3.5.5 Fuzzy classification

Due to its capability of managing uncertainty and indecision fuzzy set theory can be used for both feature extraction and classification. Most common approaches break handwritten characters into component features, which are described in terms of linguistic variables and fuzzy sets. After this fuzzy rules are created to describe prototype characters and recognition is performed by comparing features derived from the input model with features of the models defined in the fuzzy rule base. (Frosini et al. 1998, Lazzerini and Marcelloni 1999)

A fuzzy approach deriving linguistic expressions describing individual characters from a fuzzy model of a set of character samples is described by Frosini et al. (1998). First the image is partitioned two dimensionally. Then both horizontal and vertical linguistic models are built, and an activation value $a_{h,k}$ is calculated for each rectangle defined by the supports of the Cartesian product of the horizontal and vertical fuzzy set

$$a_{h,k} = \frac{\sum_{i=1}^{P} \mu_h^H(x_i)\mu_k^V(y_i)}{\sum_{i=1}^{P} \mu_h^H(x_i)}, \tag{3.9}$$

where μ_h^H and μ_k^V are the triangular membership functions of the horizontal and vertical fuzzy sets, respectively.

3.6 Adaptation methods for recognizers

The most common way of adapting a recognizer has been through an initialization phase at the start of use. During it the prototype set is adapted for that individual user. This can be done either by having the user input all of the prototypes to be used (Lu and Brodersen 1984, Tappert 1984, Mandler et al. 1985, Nouboud and Plamondon 1991) or by retraining the available prototype set (Connell and Jain 1999).

The adaptation of a recognizer and its viability depend greatly on the method of recognition used. In most cases there is a prototype set which is adapted to suit the writing style of the user. Adaptation could also be implemented in systems based on other approaches, for example by adjusting weights of underlying neurons or modifying HMMs. The problems in adapting such systems arise mainly from the fact that a much larger data set is needed as a single example is not sufficient for estimating model parameters (Rabiner 1989, Annadurai and Balasubramaniam 1996). Also the time needed for retraining would probably be unacceptable. However, a scheme where individual HMMs are retrained if the number of examples matched to that particular model is above a threshold has been shown to be effective (Connell and Jain 1999).

The adaptation techniques for a prototype or template set can be divided into three sub-categories. First, prototypes can be added. Second, the existing prototypes may be modified according to some appropriate scheme. Last, prototypes can be removed from the set or inactivated.

In general, the addition of new prototypes is necessary when ways of writing not yet included in the prototype set are introduced. Tappert (1984) used an approach where, while the recognizer was in update mode, prototypes were added if the character was misrecognized or the ratio between the matching distance to the first and second prototype candidates was within a given range of tolerance.

Nouboud and Plamondon (1991) suggest a scheme, where the user can alter the prototype set during use. This is done by allowing the user to define a new character by writing 16 specimens. The user can also delete any character from the prototype set.

In (Schomaker et al. 1994) a probabilistic scheme is utilized. A prototype usage histogram is constructed and by using discriminant analysis this histogram is reduced in dimension, thus removing less used, or not probable, prototypes. It was noted that even with the reduced amount of prototypes recognition accuracy remained good, which clearly suggests that the number of prototypes in use could be reduced without negative effects on recognition performance.

Laaksonen et al. (1998a) and Liu and Nakagawa (1999) have proposed prototype modification schemes based on Learning Vector Quantization (LVQ) (Kohonen 1997). It was noted by Liu and Nakagawa (1999) that two prototype modification schemes yielded exceptionally good results. The first was a scheme based on generalized LVQ, where the closest correct prototype and the closest incorrect one are always updated in an optimization frame. The second learning scheme was based on minimizing classification error by gradient descent where a restricted number of prototypes were updated for each input pattern.

A scheme of adaptation where the system can learn the user's writing after both correct and incorrect recognition can be found in Qian (1999). Due to the word-based nature of the procedure in question, the adaptation scheme is based on improving the best interpretation of the word in question. Three adaptation approaches, adding a template, replacing a template and adjusting a template are used.

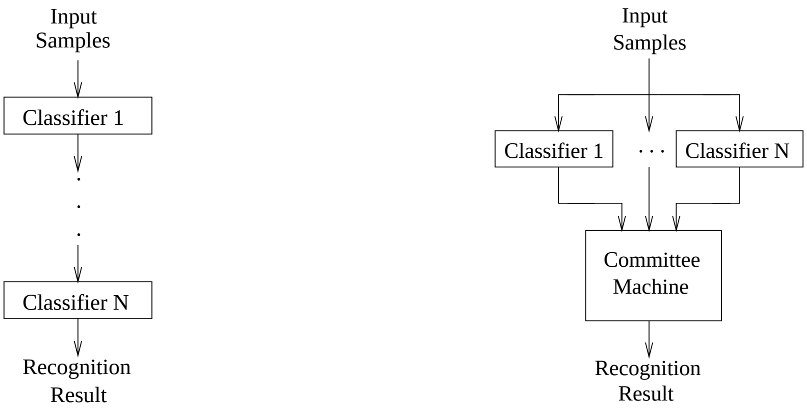
3.7 Recognizer combination methods

When one studies the outputs of several classifiers it can often be noted, that the errors are not necessarily overlapping. Since the primary objective of any recognition system is to achieve the best attainable performance, the possibility of combining different classifiers to enhance overall performance has arisen. The two main reasons for combining classifiers are desired increases efficiency and accuracy. (Kittler et al. 1998)

In order to increase efficiency, multistage combination rules where the input is first classified by a simple classifier using a small set of cheap features and a reject option, and then a more powerful classifier to handle the more difficult samples (Kittler et al. 1998) is often used. Another approach is to use the simple classifier to prune the possible amount of matches and have a more complex approach decide the final matching (Rahman and Fairhurst 1997c). A schematic diagram of such multistage, or serial, approaches is show in Figure 3.7(a).

For increasing accuracy results from parallel classifiers using the same input can be combined. Designing and combining the classifiers to enforce each other is of utmost importance in attempting to improve performance. (Suen et al. 1992). A schematic diagram of the parallel approach is show in Figure 3.7(b).

In the following sections some recognizer combination methods are examined. As an all-inclusive selection is out of the scope of this thesis, only some of the most common and promising approaches are focused on.



(a) General serial committee structure (b) General parallel committee structure

Figure 3.7: General structures of parallel and serial committees

3.7.1 Majority voting

Majority voting is a very simple and elegant scheme of classifier combination for performance increase. The basic approach is to have several independent classifiers output their results and then run a vote on these results, the result having most classifiers behind it being the output of the committee. Despite its simple nature majority voting has been shown to be very effective (Lam and Suen 1994).

The majority voting decision scheme can be written as (Kittler et al. 1998) assigning $Z \to \omega_j$ if

$$\sum_{i=1}^{N} \Delta_{ji} = \max_{k=1}^{C} \sum_{i=1}^{N} \Delta_{ki} \tag{3.10}$$

in which the sum on the right hand side equals to the count of the votes received for the class k of a total of C classes from the N recognizers. Δ_{ki} is defined by hardening the *a posteriori* probabilities $P(\omega_k|\overline{x}_i)$ to binary values as

$$\Delta_{ki} = \begin{cases} 1 & \text{if } P(\omega_k|\overline{x}_i) = \max_{j=1}^{C} P(\omega_j|\overline{x}_i) \\ 0 & \text{otherwise} \end{cases} . \tag{3.11}$$

Lam and Suen (1994) studied the theoretical foundations of majority voting in order to gain a deeper understanding on why it works. It is presumed that N classifiers are used and each is assumed to vote and the vote to have either of two values, correct or incorrect. The basic benefit of the majority voting scheme is that in order for the voting decision to be incorrect a majority of the votes must be for the same incorrect answer. Due to the large number of possible mistakes, it can be expected that the majority of the recognizers would not often simultaneously make the same mistake.

Assuming that the N individual experts have the same and mutually independent probability p of being correct, the probability $P_c(N)$ of the consensus being correct can be computed using the binomial distribution

$$P_c(N) = \sum_{m=k}^{N} \binom{N}{m} p^m (1-p)^{N-m}, \tag{3.12}$$

where $k = N/2$ is the margin of majority. It is also shown, that if the individual recognition accuracy probabilities are above a threshold of $p_u = 0.8090$, the ordering $P_c(2N) < P_c(2N-1) < P_c(2N+2) < P_c(2N+1) < P_c(2N+4)$ holds for all N. It has also been shown that if all the experts have error rates below $p_e < 0.5$, the overall correct decision rate increases with N (Miller and Yan 1999b).

3.7.2 Probabilistic combination methods

The Bayesian approach can be considered to be the most basic probabilistic combination method. As the name already suggests, the basis for Bayesian probability calculations is Bayes' rule

$$P(\omega_i|\overline{x}) = \frac{P(\overline{x}|\omega_i)P(\omega_i)}{\sum_{j=1}^{m} P(\overline{x}|\omega_j)P(w_j)} \tag{3.13}$$

The use of Bayesian combining for classifiers in general has been examined by Xu et al. (1992). For the Bayesian approach the classifiers must output results in the measurement level, ie. classifier e attributes each label j in the set Λ of output labels a measurement value that addresses the degree that the sample $\overline{x}$ has the label j.

With the Bayesian approach the final decision $E(\overline{x})$ is generally of the form

$$E(\overline{x}) = j, \text{ with } P_E(\overline{x} \in \omega_j|x) = \max_{i\in\Lambda} P_E(\overline{x} \in \omega_i|\overline{x}), \tag{3.14}$$

where $P_E(\overline{x} \in \omega_i|\overline{x})$ is the probability of $\overline{x}$ belonging to the class ω_i in the combiner. Generally any classifiers with some kind of apparent post-probabilities computable can be combined by the means of (3.14). For example, in the case of a distance classifier e_k where x is classified according to a distance measure $d_k(\overline{x}, \omega_i)$, the probability of $\overline{x}$ belonging to ω_i could be calculated, for example, with

$$P_k(\overline{x} \in \omega_i|\overline{x}) = \frac{1/d_k(\overline{x}, \omega_i)}{\sum_{j=1}^{M} 1/d_k(\overline{x}, \omega_j)}. \tag{3.15}$$

The probabilities could then be, for example, averaged to produce the combiner probability,

$$P_E(x \in \omega_i | x) = \frac{1}{N} \sum_{k=1}^{N} P_k(\overline{x} \in \omega_i | \overline{x}), \tag{3.16}$$

and these values again used through (3.14). This would be even simpler with Bayesian classifiers, as they directly produce the needed probabilities in (3.16).

The aspect of producing probabilities from recognizer scores has also been addressed by Bouchaffra and Govindaraju (1999). Due to the difficulty in calculating the state conditional probability $P(\overline{x}|\omega_i)$ in (3.13), the classic probability estimate as the frequency of an event in a number of trials is used. This is done by counting the number of times ω_i is the top choice in $|\mathcal{X}|$, the number of input patterns in the set $\mathcal{X}$, trials. Thus the conditional probability can be estimated as

$$P(\omega_i|\overline{x}) \approx \hat{P}(\omega_i|\overline{x}) = \frac{\sum_{u \in \mathcal{X}} \xi_\omega^u(\omega_i)}{|\mathcal{X}|}, \tag{3.17}$$

where

$$\xi_\omega^u(\omega_i) = \begin{cases} 1, \text{ if } \omega_i \text{ is the top choise given } u \in \mathcal{X} \\ 0, \text{ otherwise} \end{cases} \tag{3.18}$$

Kang and Lee (1999) present an approach to combine classifiers by minimizing the Bayes error rate using higher order dependencies. The presented Δ second order approach achieved a classification performance of 98.1%, a notable improvement in comparison to the best individual classifier used (96.0%), a voting approach (97.6%) or a standard first order Bayesian (97.1%).

The Dempster-Schafer theory of evidence can be seen as a kind of generalization to Bayesian combining. It is applicable also when handling weak evidence that does not fulfill the rather strict assumptions of probability theory. A computationally very efficient method can be derived from using binary voting, for or against membership, for every expert for one class and feature subspace. Each vote can be handled as an independent source of evidence for the class membership of the input patter. Thus it is not necessary to compute the combined belief for all of the possible subsets, but merely for the sets in focus for the final decision. (Franke and Mandler 1992)

3.7.3 EM algorithm

The Expectation-Maximization (EM) algorithm is an iterative procedure for Maximum Likelihood (ML) parameter estimation of a finite mixture model. The development of the EM algorithm can be seen as having started sometime in the 1960's. (Xu and Jordan 1993)

In Xu et al. (1995) a mixture of experts and EM learning algorithm is shown as follows. The mixture of experts model is based on the conditional mixture

$$P(y|x,\Theta) = \sum_{j=1}^{K} g_j(x,\nu)P(y|x,\theta_j), \tag{3.19}$$

where

$$P(y|x,\theta_j) = \frac{1}{(2\pi)^{n/2}|\Gamma_j|^{1/2}} \exp\{-\frac{1}{2}[y - f_j(x,w_j)]^T\Gamma_j^{-1}[y - f_j(x,w_j)]\}, \tag{3.20}$$

where $x \in R^n$, Θ consists of ν, $\{\theta_j\}_1^K$ and θ_j consists of $\{w_j\}_1^K, \{\Gamma_j\}_1^K$. $f_j(x,w_j)$ is the output of the jth expert and $g_j(x,\nu)$ is given by the softmax function

$$g_j(x,\nu) = \frac{\exp\beta_j(x,\nu)}{\sum_i \exp\beta_i(x,\nu)}, \tag{3.21}$$

where $\beta_j(x,\nu)$, $j = 1,\dots,K$ are the outputs of the gating network. The parameter Θ is then estimated using ML, where the log likelihood is given by $L = \sum_t \ln P(y^{(t)}|x^{(t)},\Theta)$. The estimate can be found iteratively through the EM algorithm consisting of two steps, given the current estimate $\Theta^{(k)}$.

The first step is called the E-step. For each pair $\{x^{(t)}, y^{(t)}\}$ compute $h_j^{(k)}(x^{(t)}|y^{(t)}) = P(j|x^{(t)}, y^{(t)})$ and then form a set of objective functions

$$Q_j^e(\theta_j) = \sum_t h_j^{(k)}(x^{(t)}|y^{(t)}) \ln P(j|x^{(t)}, y^{(t)}), j = 1,\dots,K \tag{3.22}$$

$$Q^g(\nu) = \sum_t \sum_j h_j^{(k)}(y^{(t)}|x^{(t)}) \ln g_j^{(k)}(x^{(t)}, \nu^{(k)}) \tag{3.23}$$

The second step, or the M-step, consists of finding a new estimate $\Theta^{(k+1)} = \{\{\theta_j^{(k+1)}\}_{j=1}^K, \nu^{(k+1)}\}$ with

$$\theta_j^{(k+1)} = \arg\max_{\theta_j} Q_j^e(\theta_j), j = 1,\dots,K \tag{3.24}$$

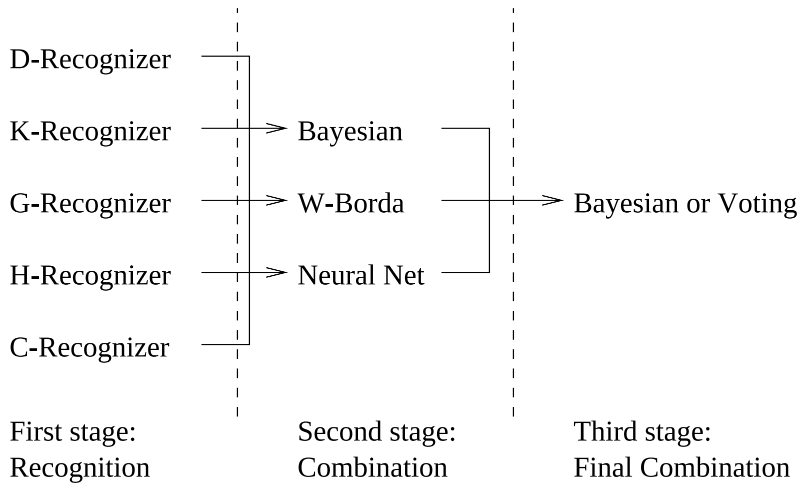


Figure 3.8: A block diagram of the combination system from Paik et. al (1996)

$$\nu^{(k+1)} = \arg\max_{\nu} Q^g(\nu) \tag{3.25}$$

Generally algorithms performing a full maximization during the M-step are referred to as EM algorithms.

3.7.4 Multi-stage combinations

Even though possibly the most common way of using multiple recognition stages is to speed up recognition by first performing a faster classification and then refining the product with more complex and effective methods (Kittler et al. 1998, Rahman and Fairhurst 1997c), it is not the only option. It is also feasible to combine classifiers in several stages in striving for better recognition performance.

Paik et al. (1996) introduce a two-level combination approach where there are five individual classifiers in the first stage, then three combining methods in the second and a final combination in the last stage. An overall diagram of the classification process is depicted in Figure 3.8. All the first-level recognizers are based on MLP neural networks using different feature vectors as input. The first uses a dynamic mesh feature (M-Recognizer), the second directional features extracted with a Kirsch mask (K-Recognizer), the third directional change features through the use of gradient vectors (G-Recognizer), the fourth histogram-based features (H-Recognizer) and the last one contour chain code for boundary information (C-Recognizer).

These five recognizers are then combined in the second stage with the three independent combiners. The first one is based on a Bayesian method. The second approach is based

Table 3.1: Some results from Paik et. al (1996)

Stage	Database			
	CENPARMI		CEDAR	
	Worst	Best	Worst	Best
Recognizers	91.65%	96.20%	95.43%	97.89%
First Combination	97.40%	97.60%	98.36%	98.40%
Final Combination (voting)	97.65%		98.50%	
Final Combination (Bayesian)	97.80%		98.62%	

on the Borda function, where the rank F_R is

$$F_R(e_k(\overline{x})) = \max_{i \in A}(B_i(e_k(\overline{x}))), \tag{3.26}$$

where $e_k(\overline{x})$ is the output result of recognizer number k for the input pattern $\overline{x}$ and

$$B_i(e_k(\overline{x})) = \sum_{k=1}^{N} w_k(C - r_k^i(\overline{x})), \tag{3.27}$$

in which C is the number of classes and $r_k^i(\overline{x})$ is the output rank by recognizer e_k for class i and N the total number of recognizers. The weighting factor w_k can also be unity for a non-weighted scheme. The third combiner in this stage is also based on a MLP neural network, where the number of output neurons equals the number of classes.

In the final stage the outputs of the second stage combiners are combined using either majority voting or a Bayesian combiner. The majority-voting approach is formulated as

$$S_V(F_i(e_k(\overline{x}))) = \begin{cases} j & \text{, if } C_F(j) = \max_{i \in A} C_F(i) > 1 \\ Rej\text{, otherwise} & \end{cases}, \tag{3.28}$$

where $C_F(i)$ is the number of votes for class i. The Bayesian approach is then defined as

$$S_B(F_i(e_k(\overline{x}))) = \begin{cases} j & \text{, if } BEL(j) = \max_{m=0}^{C} BEL(m) > \alpha \\ Rej\text{, otherwise} & \end{cases}, \tag{3.29}$$

where $BEL(m)$ is again the probability for m. The system was tested with the CEDAR (Srihari 1997) and CENPARMI (Nadal 1998) digit databases. Some results have been gathered into Table 3.1. The first row entitled Recognizers shows the best and worst percentages obtained with the individual first stage recognizers for each database. The second row indicates the best and worst results from the second stage, or equivalently the first combinatory stage, recognizers. The last two rows then show the overall results using both the voting and Bayesian final combiners, again for both databases individually.

3.7.5 Group-wise classification

Another way of increasing overall classification is to use or design classifiers specialized in specific character types or classes. The classifiers can be specialized in a particular sub-domain, obtaining exceptional performance there even when performing poorly in the entire domain (Teow and Tan 1995). Another option is to use specialized classifiers to re-process classes causing a notable amount of confusion (Rahman and Fairhurst 1997b).

According to Rahman and Fairhurst (1997a) two approaches to multiple expert configurations can be identified. The first is to develop, formalize and implement formal methods for combining multiple experts. This approach is thus mostly concerned with how the combination can improve on the rates of the member classifiers. Such methods are most commonly based on functional mathematics or statistical reasoning.

The second approach is to develop and implement specialized and task-oriented methods. Group-wise classification methods are a good example of such, as *a priori* knowledge often plays a major role in tailored methods. Using specialized classifiers for specific classes that are known to cause problems is indeed group-wise classification using *a priori* knowledge. This can be expected to be a productive way of dealing with the classification problem, as distinguishable confusion classes are a substantial source of error in the overall performance of a classifier (Rahman and Fairhurst 1997b).

The approach suggested by Rahman and Fairhurst (1997a) is informal in nature and incorporates *a priori* second-order knowledge from the training set. A general schematic of this type of strategy is presented in Figure 3.9. The basic classifier first performs an initial separation of the input characters. Based on the *a priori* knowledge, groups of character classes probable to cause confusion undergo group-wise classification, whereas the structurally dissimilar characters are directed to the general classifier. The final decision is then obtained by combining the decisions of the general and specialized classifiers.

A note should be made that when designing very specialized group-wise classifiers, the probability of such being capable of good recognition among other classes than those they were initially designed for will probably be very limited. Due to this factor the initial assignment of input samples to the specialized classifiers becomes a crucial point,

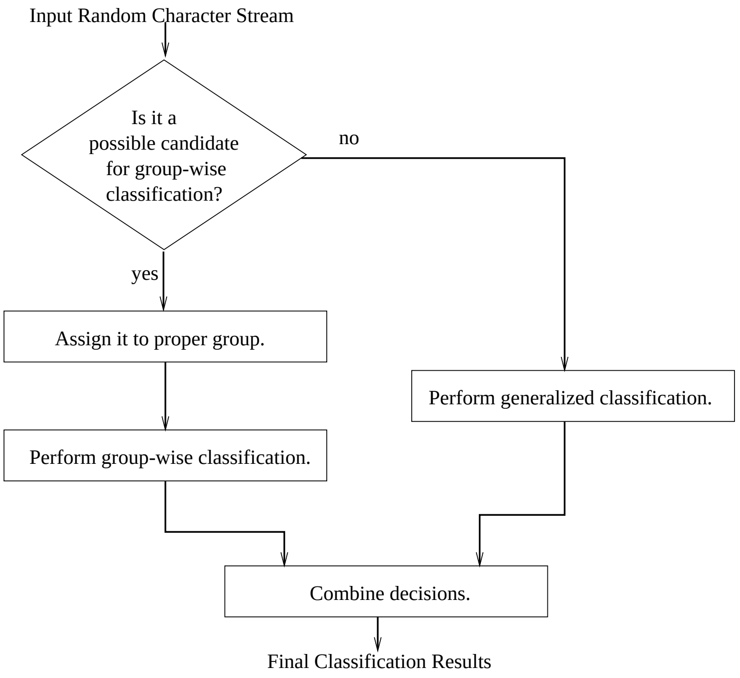


Figure 3.9: Generalized group-wise multi-expert structure

and also a possible source of errors. One way to solve this problem is to implement the specialized classifiers in a way that they are capable of group-wise recognition and trained rejection. So, they will automatically reject classes not within their area of expertise. Another way of dealing with this problem is to implement specialized filters with trained acceptance and rejection for each specialized classifier so that they can accurately extract the classes to be recognized by that particular classifier. (Rahman and Fairhurst 1997b)

An experiment on a group-wise classification system has been performed by Rahman and Fairhurst (1996). The best individual classifiers achieved recognition accuracy of 92.4% for numerals and 81.1% for alpha-numeric characters. With the use of the group-wise classification scheme accuracies of 94.8% and 84.2%, respectively, were achieved.

Teow and Tan (1995) propose a fundamentally different approach resulting in rather similar operation. A notable benefit of the approach is the lack of need for *a priori* knowledge. The Adaptive Integration of Multiple Experts (AIME) system proposed employs a gating network implemented by a fuzzy neural logic network. It is a neural network model capable of both pattern processing and logical inferencing, trained with the Supervised Clustering and Matching (SCM) algorithm (Tan and Teow 1995), which is a match-based learning method.

The gating module learns to assess the performance of individual experts based on their performance in the data space. It is thus capable of giving precedence to experts which have priorly performed well in certain situations. Also an exceptions expert is

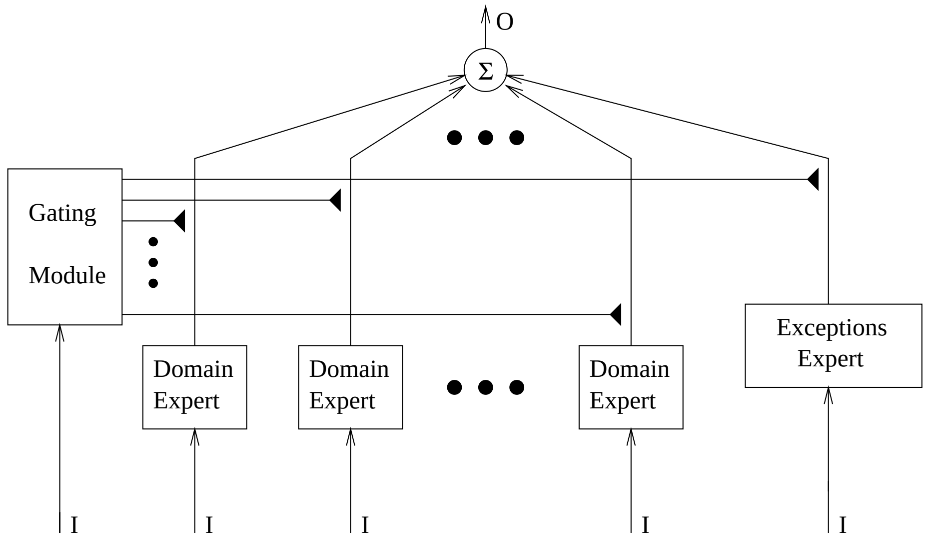


Figure 3.10: A schematic diagram of the AIME system

trained with SCM to capture patterns that fail to be classified correctly by any of the domain experts.

The AIME system is capable of both off-line and on-line learning, and it can thus be initially trained and then the performance boosted during operation. A schematic diagram of the system is shown in Figure 3.10.

3.7.6 Critic-driven combining

An interesting addition to committee classification strategies is the inclusion of a critic into the decision scheme. Basically the task of the critic is just to decide whether the classifier the critic has been assigned to is correct or incorrect. Due to the fact that the critic only has two classes to decide from, its predictions are generally more reliable than those of the classifiers taking on a multi-class problem. (Miller and Yan 1999a)

Two approaches to critic-driven combinations are identified by Miller and Yan (1999a), namely critic-driven voting and critic-driven averaging of probabilities. Critic-driven voting is performed through a standard voting scheme with the exception that if the critic deems the expert's prediction to be incorrect, the expert abstains from voting. As it has been shown that the probability for a correct voting result with $2N$ expert's is lower than with $2N-1$ experts (Lam and Suen 1994), so when the number of votes is even, the expert whose critic has the least confidence in the result is neglected. If one class receives more than half of the accepted votes, it is deemed the result.

For averaging properties it is noted in (Miller and Yan 1999a) that simple averaging is outperformed by using either geometric or arithmetic averaging but still more information is to be gained from the critic. Especially the situation where zero probability is obtained from the critic is highly informative, as this means that the expert's predicted class should be excluded. This can be taken into account by the following conditioning

$$\tilde{P}_e^{(j)}(\omega_j|\overline{x}, b_j = 0) = \begin{cases} \frac{1}{C-1} & \text{if } \omega_j \neq \omega^* \\ 0 & \text{otherwise} \end{cases}, \tag{3.30}$$

where C is the total number of classes, $b_j \in \{0, 1\}$ is the assertment from critic j, and $\omega^* = \arg\max_{\omega_i} P_e^{(j)}(\omega_i|\overline{x})$. After averaging the cross entropy cost sums over $2N$ terms the resulting probability estimate for the input $\overline{x}$ belonging to class ω_j, assuming experts as posteriors, is

$$P(\omega_j|\overline{x}) = \sum_{j=1}^{N} \sum_{l=0}^{1} w_{jl}(\overline{x}) \tilde{P}_e^{(j)}(\omega_j|\overline{x}, l), \tag{3.31}$$

where the weighting function $w_{jl}(\overline{x})$, $l \in \{0, 1\}$, is a probabilistic measure of the critic's confidence in its expert,

$$w_{jl}(\overline{x}) = \frac{P_c^{(j)}(l|\overline{x})}{\sum_{n=1}^{N} P_c^{(n)}(l|\overline{x})}. \tag{3.32}$$

When dealing with majority voting, if the experts' common error rate $p < 0.5$, the overall correct decision rate increases with the amount of experts N. In the case of critic-driven voting, the expected increase in overall correct decision rate is extended so that the overall correct decision rate increases with N if $p + q < 1$, where q is the critics error rate. (Miller and Yan 1999b)

3.7.7 Behavior-knowledge space method

The Behavior-Knowledge Space (BKS) method introduced by Huang and Suen (1995) is a classifier combination method theoretically sound even when independence among the member classifiers is not assumed. For independence not to be needed, information needs to be derived from a knowledge space that can concurrently record the decisions of all the classifiers on each learned sample. An example of such knowledge spaces is the BKS.

A BKS is a K-dimensional discrete space, where each dimension corresponds to the decision of one classifier, each of whom have $C + 1$ possible decision values to choose from. The intersection of the decisions of the classifiers occupies one unit in the BKS, and each unit accumulates the number of samples for that combination. The intersection unit of the classifiers' decisions for an input sample is called the focal unit. An

Table 3.2: A 2-D Behavior-knowledge space

$e(1)\backslash e(2)$	1	…	j	…	1
1	(1,1)	…	(1,j)	…	(1,11)
⋮	⋮	⋮	⋮	⋮	⋮
i	⋮	⋮	(i,j)	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮
11	(11,1)	…	(11,j)	…	(11,11)

example of a two dimensional BKS is shown in Table 3.2, where (i, j) is the focal unit when the results from classifiers $e(1)$ and $e(2)$ are i and j, respectively.

To examine the BKS, some symbols need to be defined. First, $BKS(e(1), \ldots, e(K))$ is the unit in the BKS where the classifiers give the outputs $e(1), \ldots, e(K)$. $n_{e(1)\ldots e(K)}(\omega)$ is the total number of incoming samples belonging to class ω in $BKS(e(1), \ldots, e(K))$. $T_{e(1)\ldots e(K)}$ is the total number of incoming samples in $BKS(e(1), \ldots, e(K))$, defined as

$$T_{e(1)\ldots e(K)} = \sum_{m=1}^{C} n_{e(1)\ldots e(K)}(m). \tag{3.33}$$

$R_{e(1)\ldots e(K)}$ is the best representative class of $BKS(e(1), \ldots, e(K))$,

$$R_{e(1)\ldots e(K)} = \{j \mid n_{e(1)\ldots e(K)}(j) = \max_{m} n_{e(1)\ldots e(K)}(m)\}. \tag{3.34}$$

The actual operation of the BKS is divided into two stages. First is the knowledge modeling, or learning stage, and then the decision making, or classification, stage. In the learning stage the BKS is constructed from both the true and recognized labels and the values for $T_{e(1)\ldots e(K)}$ and $R_{e(1)\ldots e(K)}$ are calculated from (3.33) and (3.34). In the classification stage the decisions are made in the focal unit according to the BKS and based on the rule

$$E(x) = \begin{cases} R_{e(1)\ldots e(K)}, \text{ when } T_{e(1)\ldots e(K)} > 0 \text{ and } \frac{n_{e(1)\ldots e(K)}(R_{e(1)\ldots e(K)})}{T_{e(1)\ldots e(K)}} \geq \lambda \\ C+1 \quad \text{, otherwise} \end{cases}, \tag{3.35}$$

where $0 \leq \lambda \leq 1$ is a threshold to control the reliability of the final decision. Threshold finding and optimality are considered further in (Huang and Suen 1995).

This approach was tested by Huang and Suen (1995) on a database of numerals from approximately 1000 writers. 5074 examples were used for training and the remaining 46451 numerals for testing. The results showed the BKS to perform better than either the Bayesian or voting approach used, and much better than any individual recognizer used in it. Also Khotanzad and Chung (1994) performed a test on 3000 numeral samples which showed a decrease of nearly 50% in the error rate from any of the three MLP classifiers used to construct the committee.

3.7.8 Boosting

Boosting is a method designed for converting a single learning machine with a finite error rate into an ensemble with arbitrarily low error rate (Drucker et al. 1993). Boosting is committee method especially designed to increase the performance of neural networks.

Boosting is based on the Probably Approximately Correct (PAC) learning model. In the standard ”strong” model the learner must be able to produce a hypothesis with an error rate at most ϵ, for arbitrarily small positive values of ϵ. Also a variation, sometimes called the ”weak” learning model can be used. In the weak model the need of correctness for the learner has substantially decreased to an error rate of slightly less than $\frac{1}{2}$, which in turn is only slightly better than random guessing. It has been shown that the learning models are actually equivalent through the use of ensemble combination. (Drucker et al. 1994)

Drucker et al. (1993) describe the boosting algorithm for an ensemble of neural networks as follows. First a set of training samples is used to train the first network. For the training set of the second network, the training samples are passed through the first network and the patterns for the second network's training set are collected so that the first network has classified half of them correctly and the other half incorrectly. Then the third network will be trained with patterns that the first and second network disagree on. The same training approach can then, if desired, be iterated in a recursive manner to produce 9, 27 and so on networks.

During the recognition phase, the patterns are passed through all the three networks. If the first two networks agree, that is the output label. Otherwise the label from the third network is used. In (Drucker et al. 1994) it is also shown, that as the training set size increases, the training error decreases until it asymptotes to the test error rate.

Experiments on boosting showed a decrease from a 9.0% error rate for the individual networks to a 6.3% error rate when using boosting on a 120 000 handwritten digit database, of which 2000 were exclusively used for testing and the remaining 118 000 for training. (Drucker et al. 1994)

3.8 Adaptation methods for recognizer combinations

Although the most common way of adaptation is to adapt a single recognizer to the user's writing style, it is also possible to construct a committee that adapts to the user as a whole. The members of such a committee can be adaptive or non-adaptive themselves.

A simple scheme for introducing some adaptation into committee operation is to have a committee performing weighted majority voting. The weights could then be adjusted based on the performance of the classifiers. The weighting would thus adjust the decision in favor of the classifier performing best at that time or for the current writer.

Another very simplistic committee adaptation approach was presented as a reference classifier in (Laaksonen et al. 1999). In this approach, the correct results for each classifier were tracked and used in adjusting the committee. The result from the classifier having the most correct applications so far was considered the output of the committee.

The AIME system discussed in Section 3.7.5 is also an example of an adaptive committee. It is capable of refining its operation first in the beginning of operation and then improving performance during use. (Teow and Tan 1995)

Also no reason can be seen as to why the BKS method (Huang and Suen 1995) examined in Section 3.7.7 could not perform in an adaptive fashion. The BKS could also store the decisions in the decision making stage, assuming that the true label can be obtained during operation. This could easily enhance performance by allowing the committee to adapt on-line according to the current situation. The implementation of writer-dependent adaptation would although require personal knowledge spaces for all writers.

Generally on-line adaptation poses problems in applications where the true class of the sample is not readily available. On-line adaptation and the Dynamically Expanding Context (DEC) (Kohonen 1986, Kohonen 1987) method used will be discussed in detail in Chapter 6.

Chapter 4

Description of the recognition system

The results in this thesis are based on an on-line recognition system for isolated handwritten characters developed at the Laboratory of Computer and Information Sciences of Helsinki University of Technology. It performs stroke-wise prototype matching using Dynamic Time Warping (DTW) distance calculations based on a variety of distance measures and the k nearest neighbor (k-NN) rule for the final decisions. Here the various aspects of the classifier are described.

Most of the following description is based on (Vuori 1999) and (Laaksonen et al. 1998a), but the author actively participated in latter stages of the recognition system development process. An example of the contribution are the symbol string distance measures explored in Section 4.3.4.

The initial, referred to as the large-scale, implementation was on the UNIX based central computer of the laboratory. To complement this, the recognition system has also been implemented onto a smaller-scale platform. This implementation is discussed in Chapter 5.

4.1 Data representation

The data in the large-scale implementation was collected using a pressure sensitive Wacom ArtPad II tablet attached to a Silicon Graphics workstation. The resolution of this tablet is 100 lines per millimeter and the maximum sampling rate 205 data points per second.

The data collected is the loci of the pen point movements in x and y coordinates, the pen's pressure on the writing surface and a time stamp. The data was then saved in UNIPEN 1.0 format (Guyon et al. 1994), a hierarchical, platform independent format for storing gathered data into a text file.

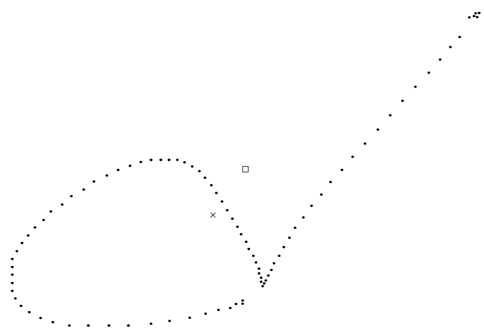


Figure 4.1: An example of the two available center operators for the character 'd'. The '□' denotes the center of the bounding box and the '×' the mass center.

4.2 Preprocessing and normalization

The preprocessing operations applied have mainly been used to adjust the sampling method and frequency. The sampling frequency can be altered with the operations *Decimate*(n) and *Interpolate*(n). *Decimate*(n) preserves every $(n + 1)$th data point while discarding the intermediate n points. *Interpolate*(n) performs the opposite action, interpolates n equally-spaced points between every two successive original data points. Also an operator called *EvenlySpacedPoints*(d) has been introduced. The operator can be used for simulating equidistant sampling, as it interpolates new data points using the original ones so that the distance between the points becomes constant. After equidistancing, the distance between adjacent points is $dl/1000$, where l is the length of the longer side of the bounding box of the character. The minimal bounding box of a character is a rectangular frame around the minimum and maximum values of the x and y coordinates of the character.

The normalization methods used were necessitated by the fact that the users only have a defined area to write in, and no specific location or size for the character is indicated. Thus it is necessary to move the characters to the same location prior to matching. This is implemented by moving the center of the character to the origin of the coordinate system. Since the "center" is not an unambiguous concept for handwritten characters, two alternative operations have been used. In the first one, the center is defined as the mass center of the character, and the movement of this point to the origin is performed through the normalization operation *MassCenter*. The second approach available is to determine the minimal bounding box for the character. The center of the bounding box is then moved to the origin using the normalization operation *BoundingBoxCenter*. The two centers for an example glyph are shown in Figure 4.1.

In addition to relocating the characters, also size variance needs to be controlled. As no strict guidelines regarding the size of the characters to be written have been introduced, size normalization is a necessity for stable operation. Size normalization is performed through the use of the same bounding box as in *BoundingBoxCenter*, by means of scaling the longer side of the bounding box to a predefined length and keeping the aspect ratio unchanged. This is called the normalization operation *MinMaxScaling*. As the length can be arbitrarily selected, the value 1000 was chosen for its evenness.

4.3 Distance measures

Due to the variation in writing speeds and lengths of strokes, it is inevitable that the number of data points in characters varies a great deal. Also the count of strokes can vary considerably. Thus it is necessary to use a distance measure and a calculation algorithm which can operate regardless of the variance in the number of points. Also symmetry, meaning that the same result is obtained when matching a to b as b to a ie. $d(a,b) = d(b,a)$, is highly desirable.

The distance metrics described in Sections 4.3.1, 4.3.2 and 4.3.3 are based on DTW and thus defined between curves with a varying number of data points. They all are also stroke-based and symmetric. If the number of strokes is different between characters, the distance is taken to be infinite. If the number of strokes matches, the total distance between the characters is the sum of the inter-stroke distances.

4.3.1 Point-to-point distances

The point-to-point (PP) distance simply uses the squared Euclidean distance between two data points as the cost function. The distance between the two strokes S_1 and S_2 and the optimal time warping path are found through the use of a dynamic programming algorithm (Sankoff and Kruskal 1983). The time warping path $P(h) = (i(h), j(h)),\ h = 1, 2, \ldots, H$, where h is an order index for matching the $i(h)$th and $j(h)$th data points of strokes S_1 and S_2, respectively, describes the point-to-point correspondence. The PP matching is depicted in Figure 4.2.

The boundary conditions for the time warping path are that the first and last data points of the strokes $S_1 = (p_1(1), \ldots, p_1(N_1))$ and $S_2 = (p_2(1), \ldots, p_2(N_2))$ are matched against each other, or

$$P(1) = P(i(1), j(1)) = (1, 1) \tag{4.1}$$

$$P(H) = P(i(H), j(H)) = (N_1, N_2). \tag{4.2}$$

The continuity condition is that all data points are matched at least once and several

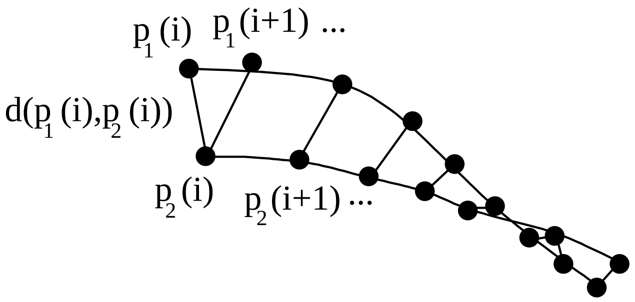


Figure 4.2: An example of PP matching

data points can be matched against one, for $h = 2, 3, \ldots, H$

$$(\Delta i(h), \Delta j(h)) = (i(h) - i(h-1), j(h) - j(h-1)) = \begin{cases} (1,0) \\ (0,1) \\ (1,1) \end{cases} . \tag{4.3}$$

The distance d between two data points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is the squared Euclidean distance,

$$d(p_1, p_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2. \tag{4.4}$$

The total distance D_{PP} between the strokes S_1 and S_2 is found by minimizing the sum of the matching costs for all data points,

$$D_{\mathrm{PP}}(S_1, S_2) = \min_{P(h)} \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))). \tag{4.5}$$

Due to the fact that the boundary and continuity conditions and the cost function are symmetrical, the distance metric D_{PP} is also symmetrical. (4.5) can be solved using dynamic programming,

$$D_{\mathrm{PP}}(i, j; h) = d(p_1(i), p_2(j)) + \min \begin{cases} D_{\mathrm{PP}}(i-1, j; h-1) \\ D_{\mathrm{PP}}(i, j-1; h-1) \\ D_{\mathrm{PP}}(i-1, j-1; h-1) \end{cases} , \tag{4.6}$$

with the initial conditions of setting $D_{\mathrm{PP}}(0, 0; 0)$ to zero and the undefined distances to infinity, or

$$D_{\mathrm{PP}}(0, 0; 0) \quad = 0 \tag{4.7}$$

$$d(p_1(i), p_2(j)) \quad = \infty \quad \text{, if } i, j \leq 0, j > N_1, \text{ or } j > N_2. \tag{4.8}$$

The minimum total cost in (4.5) is given by

$$D_{\mathrm{PP}}(S_1, S_2) = D_{\mathrm{PP}}(N_1, N_2; H) \tag{4.9}$$

Due to the fact that the number of points matched does indeed affect the sum of (4.5), the distances between seemingly similar strokes written at different speeds is still notable. So in addition to the shape of the stroke, also its dynamic properties affect the total distance. This might be an advantage when adapting to the style of a certain user, but is surely a disadvantage when dealing with several users.

Another aspect is that the relative weight of the cost of a stroke with more points is much larger in the final sum than that of a stroke with less points, even though in true interpretation of characters it might not be so. These possible problems have been countered by defining a normalized version of the PP distance, the normalized point-to-point (NPP) distance. The normalization is performed by scaling the distance of two matched strokes with the total number of matchings performed, H, so that the total cost becomes

$$D_{\mathrm{NPP}}(S_1, S_2) = \frac{D_{\mathrm{PP}}(S_1, S_2)}{H}, \ \max(N_1, N_2) \leq H \leq N_1 + N_2 - 2. \tag{4.10}$$

4.3.2 Point-to-line distances

The point-to-line (PL) distance is a modification of the PP distance where the points of a stroke can also be matched to lines interpolated between the successive points of the stroke they are being matched against (Sankoff and Kruskal 1983). This helps reduce the effects of phase differences in the sampling of two curves of similar shape. The effects are especially seen in cases where the sampling frequencies differ notably, while the actual shapes of the characters are relatively similar. The calculated PP distance between characters of similar shape, but of which one has a much higher sampling frequency is much larger than the difference in the actual appearance of the characters. The difference should naturally not be dependent on sampling frequency.

Problems may arise with the PL distance when the sampling frequency is very low, as the linear interpolation between points might cause shape distortions, especially in areas of high curvature. But when the sampling frequency is high enough, the effects of phase differences are insignificant.

In the PL distance calculations, all points except the first and last ones, whose matchings are dictated by the boundary conditions below, are matched once. If a stroke being matched contains only one point, it is duplicated. Each point is matched to the closest point on the lines interpolated between the data points of the opposite curve. The cost

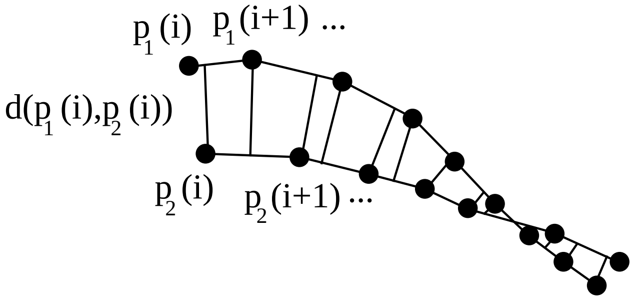


Figure 4.3: An example of PL matching

of an individual point matching is the squared Euclidean distance to the nearest point on the line. The PL matching is depicted in Figure 4.3.

In order to define the continuity constraints, some functions need to be defined. The index $k \in \{1, 2\}$ is the number of the stroke in question. $I_k(h)$ indicates whether the point of the stroke S_k involved in the hth matching is a data point or a point on an interpolated line. The function $i_k(h)$ is the index of the first data point that has not been used in the first $h-1$ matchings. The function $r(h) = [0, 1]$ indicates the relative position of the interpolated point involved in the hth matching. The continuity conditions can now be stated as

1. If both $I_k(h)$ and $I_k(h+1)$ indicate data points, the points are adjacent and thus $i_k(h+1) = i_k(h) + 1$.
2. If $I_k(h)$ indicates a data point and $I_k(h+1)$ an interpolated point, $i_k(h+1) = i_k(h)$.
3. If $I_k(h)$ indicates an interpolated point and $I_k(h+1)$ a data point, $i_k(h+1) = i_k(h) + 1$.
4. If both $I_k(h)$ and $I_k(h+1)$ indicate interpolated points, $i_k(h+1) = i_k(h)$.

The boundary conditions are defined as, with $a, b \in \{1, 2\}$ and $a \neq b$, that the first point $p_a(1)$ of the stroke S_a is matched to a point interpolated between the first and second points, $p_b(1)$ and $p_b(2)$, and point $p_b(1)$ is not matched at all. Similarly, the last point $p_a(N_a)$ of the stroke S_a is matched to a point interpolated between the last and second last points, $p_b(N_b)$ and $p_b(N_b - 1)$, and point $p_b(N_b)$ is not matched at all. With the functions $I_k(h)$ and $i_k(h)$ these conditions can be written as

1. $i_a(1) = i_b(1) = 1$
2. If $I_a(N_a + N_b - 2)$ indicates a data point, then $i_a(N_a + N_b - 2) = N_a + 1$ and $i_b(N_a + N_b - 2) = N_b$.

As the boundary and continuity conditions and the associated cost function are symmetric, the PL distance is also symmetric.

The nearest point (p_i, r) on the line between points p_i and p_{i+1} to the point q can be written as

$$(p_i, r) = (1 - r)p_i + rp_{i+1}, \tag{4.11}$$

where the relative location parameter r is defined as

$$r = \begin{cases} 1, \text{ if } \overline{r} \geq 1 \\ 0, \text{ if } \overline{r} \leq 0 \\ \overline{r}, \text{ otherwise} \end{cases}, \tag{4.12}$$

where

$$\overline{r} = \frac{(q - p_i) \cdot (p_{i+1} - p_i)}{(p_{i+1} - p_i) \cdot (p_{i+1} - p_i)}. \tag{4.13}$$

The PL distance is also solved by using dynamic programming. The recursive equation is

$$D_{\mathrm{PL}}(i, j, r; h) = \min \begin{cases} D_{\mathrm{PL}}(i - 1, j, r; h - 1) + d(p_1(i), (p_2(j), r)) \\ D_{\mathrm{PL}}(i, j - 1, r; h - 1) + d((p_1(i), r), p_2(j)) \end{cases} \tag{4.14}$$

With similar initial conditions as with the PP distance of setting $D_{\mathrm{PL}}(i, j, r; 0)$ to zero and the undefined distances to infinity, or

$$D_{\mathrm{PL}}(i, j, r; 0) = 0 \tag{4.15}$$

$$d((p_1(i), r), p_2(j)) = d(p_1(i), (p_2(j), r)) = \infty, \text{ if } i, j \leq 0, i > N_1, \text{ or } j > N_2, \text{ or } j > N_2 \tag{4.16}$$

The minimum total cost is then given by

$$D_{\mathrm{PL}}(S_1, S_2) = D_{\mathrm{PL}}(N_1 - 1, N_2 - 1, r; N_1 + N_2 - 2) + \min \begin{cases} d(p_1(N_1), (p_2(N_2 - 1), r)) \\ d((p_1(N_1 - 1), r), p_2(N_2)) \end{cases} \tag{4.17}$$

The normalized point-to-line (NPL) distance, the PL distance normalized by the total number of matches performed, H, is defined in a similar way as the NPP distance

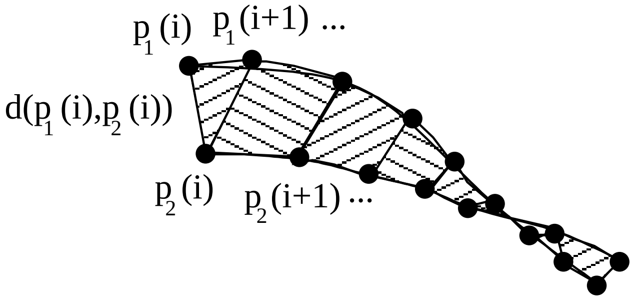


Figure 4.4: An example of SA matching

in (4.10),

$$D_{\mathrm{NPL}}(S_1, S_2) = \frac{D_{\mathrm{PL}}(S_1, S_2)}{H},\ H = N_1 + N_2 - 2. \tag{4.18}$$

4.3.3 Area-based distances

The main problem with the PP and PL distances is the fact that they do not compensate for the density of data points. This results in heavy emphasization of points where the pen is moved slowly, usually end points and points of high curvature.

Aside from the normalization in the NPP and NPL distances, the density of data points can be taken into account by using an area-based distance. This results in the lengths of the strokes being more meaningful, as the area depends on the distances between points and thus also the length of the stroke.

Two area-based distances have been implemented, the simple-area (SA) distance and the kind-of-area (KA) distance. Both area-based distances have similar boundary and continuity constraints as the PP distance, and the overall distances can also be solved using dynamic programming.

Simple-area distance

The cost function associated with the SA distance is defined by the area spanned by three or four data points. After the first matching, each matching produces a triangle or a quadrilateral between the strokes. The SA matching is depicted in Figure 4.4.

When matching two strokes, $S_1 = (p_1(1), \dots, p_1(i(h)), \dots, p_1(N_1))$ and $S_2 = (p_2(1), \dots, p_2(j(h)), \dots, p_2(N_2))$, in every point-to-point matching the indexes of the matched data points change in one of the following ways:

1. $(\Delta i(h), \Delta j(h)) = (1, 0)$ and a triangle is spanned by $p_1(i(h)), p_1(i(h-1))$ and $p_2(j(h))$, or

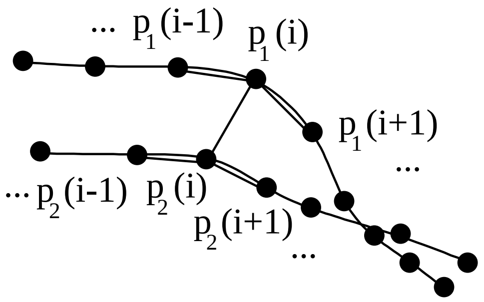


Figure 4.5: An example of KA matching

2. $(\Delta i(h), \Delta j(h)) = (0, 1)$ and a triangle is spanned by $p_1(i(h)), p_2(j(h-1))$ and $p_2(j(h))$, or

3. $(\Delta i(h), \Delta j(h)) = (1, 1)$ and a quadrilateral is spanned by $p_1(i(h)), p_1(i(h-1)), p_2(j(h))$ and $p_2(j(h-1))$.

The area, and thus the associated cost, for polygons with n points when the lines do not intersect, can be calculated as (Råde and Westergren 1990)

$$A_{\mathrm{SA}} = |x_n y_1 - x_1 y_n + \sum_{i=1}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)|. \tag{4.19}$$

Intersecting of the lines occurs only if the strokes cross each other. If the sampling rate is high the resulting error is very small in proportion to the total cost. This effect has been neglected in order to keep the computational complexity reasonable.

Kind-of-area distance

With the KA distance, the cost associated with matching the points $p_1(i(h))$ and $p_2(j(h))$, is

$$\begin{aligned} &A_{\mathrm{KA}}(p_1(i(h)), p_2(j(h)); n, m) \\ &\quad = |p_1(i(h)) - p_2(j(j))|^n (|p_1(i(h)) - p_1(i(h)+1)| + |p_1(i(h)) - p_1(i(h)-1)| \\ &\qquad + |p_2(j(h)) - p_2(j(h)+1)| + |p_2(j(h)) - p_2(j(h)-1)|)^m. \end{aligned} \tag{4.20}$$

Prior to matching, strokes S_1 and S_2 are augmented by replicating the second and next to last points, resulting in the lengthened stroke $S = (p(2), p(1), p(2), \ldots, p(N-1), p(N), p(N-1))$. The new starting and ending points are not matched. The SA matching is depicted in Figure 4.5.

The parameter $m \geq 0$ controls the effect of the point density on the total distance between strokes. If $m = 0$, high density parts are emphasized. When $m \geq 1$, the significance of points which are close to their neighbors is increasingly reduced. The other parameter, $n \geq 1$, affects the penalization for point distortions. When $n > 1$, points far apart are penalized more harshly than those close together. With $n = 2$ and $m = 0$ the KA distance is reduced to the PP distance.

4.3.4 Symbol-string-based distances

An entirely different approach implemented in the recognition system are the symbol-string-based distances, an extension of Freeman's code discussed in Section 3.5.2. In the creation of the symbol strings, the direction between the beginning and end points of the symbol were discretized to 2^i, $i \in \{2, 3, 4, 5\}$, distances. The end point was deemed a point either far enough, according to a threshold, from the beginning point or when the end of the stroke was reached. The symbols were stored either with or without length information in their codings. Also different codings for the pen-up symbols, where the pen was off the tablet, were used.

A sharp corner detection method was implemented to take into account rapid changes in the direction of the writing in order to add additional symbols to points of high curvature. The equation weighs changes near the origin of the polar coordinate system somewhat more. This can be expected to be beneficial in the sense that directional changes near the origin generally contain more information. Directional changes in the far parts of the character, on the other hand, often correspond to loops at the ends of strokes, and are thus of little interest. The change in direction is estimated as

$$\phi_i = \arctan\left(\frac{x_{i-2} + x_{i-1} - x_i}{y_{i-2} + y_{i-1} - y_i}\right), \tag{4.21}$$

where x_i and y_i are the x and y coordinates of point number i. The value of ϕ_i is calculated at points i and $i - 3$. If the difference $\Delta\phi_i$ between these values exceeds the threshold d/π, where d is the number of discretion directions, the point i is deemed the beginning point for the next symbol.

Two examples of the symbol string creation are shown in Figure 4.6. The first symbol string was created without the use of the corner detection option and the second one with the option active.

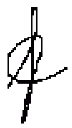
The original character	cdlldo	cdlldhna
		

Figure 4.6: Examples of symbol string created

The used cost function was defined based on the Levenshtein distance (Sankoff and Kruskal 1983) through the existence of additions, replacements and removals in the symbol string constructed as

$$d(a^i, b^j) = \min \begin{cases} d(a^{i-1}, b^j) \quad + w_{add}(a_{i-1}, b_j, a_i) & \text{(addition)} \\ d(a^{i-1}, b^{j-1}) + w_{rep}(a_i, b_j) & \text{(replacement)} \\ d(a^i, b^{j-1}) \quad + w_{rem}(a_{i-1}, a_i, a_{i+1}) & \text{(removal)} \end{cases} , \tag{4.22}$$

where a^i and b^i are the ith symbols in the two symbol strings being matched. The individual w-functions were defined as

$$w_{add}(a_{i-1}, b_j, a_i) = \lambda_1 \frac{\ell_{b_j}}{\ell} + \begin{cases} -\lambda_2, \text{ if } b_j \in \{a_{i-1}, a_i\}, a_{i-1} \neq a_i \\ -\lambda_3, \text{ if } b_j = a_{i-1} = a_i \\ \lambda_4 \quad , \text{ if } b_j \in U \\ 0 \quad\ \ , \text{ otherwise} \end{cases} , \tag{4.23}$$

$$w_{rep}(a_i, b_j) = \kappa_1 \frac{|\ell_{a_i} - \ell_{b_j}|}{\ell} + \frac{\angle(a_j - b_j)}{2\pi} + \begin{cases} \kappa_2 & , \text{ if } a_i \subset U, b_j \not\subset U, \\ & \text{ or } a_i \not\subset U, b_j \subset U \\ 0 & , \text{ otherwise} \end{cases} , \tag{4.24}$$

and

$$w_{rem}(a_{i-1}, a_i, a_{i+1}) = \rho w_{add}(a_{i-1}, a_i, a_{i+1}) \tag{4.25}$$

where ℓ_{b_j} is the length of the symbol b_j and U the set of pen-up symbols. d is the number of discretization directions and ℓ the discretization distance threshold. The cost parameters are identified in Table 4.1.

The total distance was then calculated using dynamic programming. This approach offers valuable benefits in computational cost in comparison to the distance measures

Table 4.1: The costs of the symbol-string distance calculations

λ_1	The cost of an addition
λ_2	The benefit of adding the same symbol as just one of its neighbors
λ_3	The benefit of adding the same symbol as both its neighbors
λ_4	The cost of adding a pen-up symbol
κ_1	The factor for the impact of the symbol length in replacement
κ_2	The cost of replacing with a pen-up symbol
ρ	The ratio between the costs of additions and removals

mentioned in Sections 4.3.1, 4.3.2 and 4.3.3. This benefit has a serious downside, namely the loss of information in the discretization stage. This was seen to result in a significantly poorer overall recognition accuracy than with the other distance metrics.

4.4 Prototype set

Generally the performance of a k-NN classifier improves with the increase in the number of samples to match the input against. But as the needed computation time also increases, it is very impractical to use all available samples for matching. With careful selection of the training set samples, the deterioration in accuracy resulting from using fewer samples can be minimized. Thus a prototype set consisting of samples representing different ways of writing can be constructed from the training samples and used in the recognition process.

As discussed in Section 3.4, the prototype set should have sufficient coverage and separability. All prototypes should be "good" in the way that they are not distorted, but accurate representations of the intended characters. To achieve good prototype sets in our system a semiautomatic clustering algorithm is used to cluster the training samples. This clustering results in groups that contain samples similar to each other, according to the selected similarity measure. Then the center-most sample from each cluster is selected to become the prototype representing that cluster.

The clustering algorithm is semiautomatic in the sense that the number of clusters must be determined beforehand. Thus, the number of clusters needed for each character class must be examined manually. Generally, due to the stroke-wise nature of the application, there must be at least as many clusters for each character as there are possible numbers of strokes to use in writing it. Also noticeably different writing styles need to have a cluster of their own. As an example, 7 prototypes for the character "E" are shown in Figure 4.7.

When the number of prototypes for each class and number of strokes therein have been decided upon, the clustering is carried out by an automatic iterative algorithm. The same preprocessing and normalization operators, as well as the same distance measure, as during recognition are used in the clustering phase. The clustering algorithm works with a splitting principle. At first all samples with the same label and number of strokes are in one cluster and this cluster is divided until the predetermined number of prototypes is reached or there are already as many groups as characters. The splitting is performed by first finding the central sample in the cluster, the one minimizing the squared distance to all others. Then the samples are ordered in the cluster according to increasing distance from the center. The most distant samples of the cluster are then

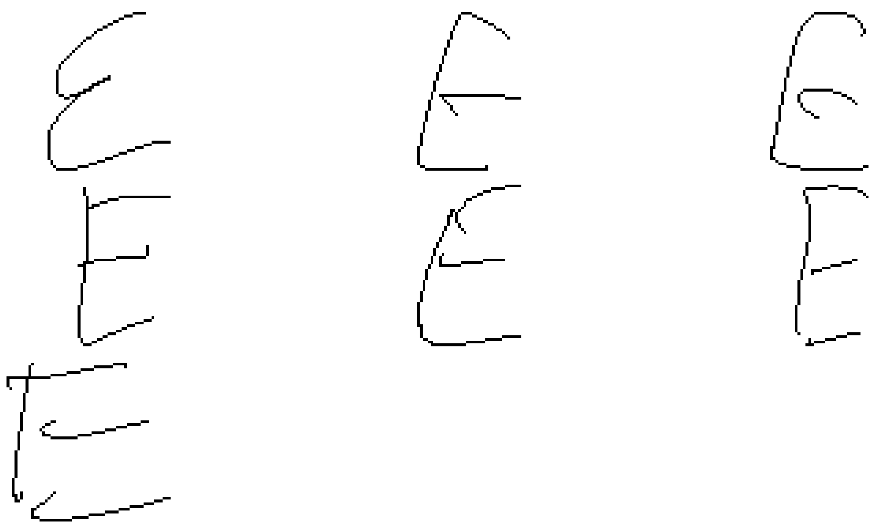


Figure 4.7: 7 sample prototypes for the character “E”

chosen to form a new cluster. The number of items in the new cluster is determined by minimizing the splitting criterion function $J(i)$ defined as

$$J(i) = d(x_{i-1}, \overline{x}_{old}(i-1)) + \max_{i \leq j \leq N} d(x_j, \overline{x}_{new}), \tag{4.26}$$

where $d(x, y)$ the squared distance between two samples, and $\overline{x}_{old}(i)$ and $\overline{x}_{new}$ the item x_i's old and new cluster centers, respectively. In the case that two distinct writing styles exist, the splitting function is roughly U-shaped. If more writing styles exist, several local minima may be seen. The samples are extracted into the new cluster by finding the index i^* so that

$$J(i^*) = \min_i J(i), \tag{4.27}$$

and the samples with indexes $i \geq i^*$ form the new cluster. Then the cluster centers are recalculated and items assigned to the cluster center nearest to them iteratively until a stable settlement is reached. After this the splitting procedure is re-entered unless the predefined number of clusters has been reached or the number of clusters equals the number of samples. (Laaksonen et al. 1998b)

4.5 Prototype pruning and ordering

In order to decrease the computational burden, the prototypes are pruned and ordered prior to actual classification. The pruning is based on the number of strokes. As the distance between characters with a different number of strokes is defined to be infinite, characters are only matched against prototypes with the same number of strokes.

The actual calculation loops have been implemented in a way that the calculation is stopped when the shortest distance already attained is exceeded. Due to this fact appropriate ordering of the prototypes helps speed up the computation considerably. The ordering is performed by placing each prototype into one of sixteen categories. The categories are defined after the preprocessing and normalization phases. For each category a four-bit binary value $b_4b_3b_2b_1$ is constructed according to the coordinates of the first and last points in the first stroke,

$$\begin{aligned} b_1 &= (x_1(1) \geq 0) \\ b_2 &= (y_1(1) \geq 0) \\ b_3 &= (x_1(N_1) \geq 0) \\ b_4 &= (y_1(N_1) \geq 0) \end{aligned} \tag{4.28}$$

where N_1 is the number of data points in the first stroke, and a true value of an expression corresponds to one and false to zero. The distance $d_c \in \{0, 1, 2, 3, 4\}$ between the categories is the count of bit differences in their binary representations. Prototypes are matched against the input sample in the order of increasing category distance.

4.6 Adaptation

Four adaptation strategies have been implemented into the basic classifier. They are all based on modifying the prototype set by either adding, altering or inactivating prototypes, or on a combination of these.

4.6.1 Prototype addition

The adaptation strategy $Add(k)$ is a strategy where new prototypes are added to the prototype set if any of the k nearest prototypes, as decided by the classifier, are incorrect. This is done even if the final classification result was correct.

Prototype addition is the only method of those presented here that can actually help the recognizer learn entirely new styles of writing, as long as the correct label can be extracted for the character. The downside to prototype addition can be seen when a user writes similar-looking characters for different classes. This results in more and more prototypes being added but the prototypes similar to the users input still have different class labels. In such cases adding prototypes will mainly only increase computation time while providing no enhancement to the recognition performance.

4.6.2 Prototype inactivation

The adaptation strategy $Inactivate(N, G)$, on the other hand, is based on inactivating prototypes in the prototype set. After each recognition the nearest prototype to the incoming sample is checked. For this purpose the goodness value $g(P)$ is defined, for a prototype P, as

$$g(P) = \frac{N_{corr}(P) - N_{err}(P)}{N_{corr}(P) + N_{err}(P)}, \tag{4.29}$$

where $N_{corr}(P)$ and $N_{err}(P)$ are the counts of how many times the class of the prototype P and the true class of input sample are the same and different, respectively, when the prototype has been the nearest one. If the goodness value $g(P)$ is below a given limit G and the prototype P has been the nearest prototype at least N times, P is removed from the set of prototypes in active use, ie. inactivated.

The usefulness of inactivation is most evident in situations, where the user writes a specific character in a way as to easily cause confusion between classes but characters of the class confusion is caused with in a distinct style. For example, the "b"'s written by the user are consistently closest to a prototype for the class "h", but none of the "h"'s written by the user match that particular prototype. In such a situation the prototypes causing confusion will be inactivated, thus improving the overall performance. Prototype inactivation can also easily and effectively be used in conjunction with any of the other adaptation methods presented here.

4.6.3 Prototype modification

When a character is close to the prototype it is matched to but of slightly different shape, reshaping of the prototype is a viable alternative to adding a new prototype. Reshaping existing prototypes is more effective for overall recognition speed as each additional prototype causes the need for more computation in every recognition round.

The prototype modification strategy used, $Lvq(\alpha)$, is based on Learning Vector Quantization (Kohonen 1997). If the classification is correct, the points of the nearest prototype are moved towards those of the input sample. If the classification resulted in an incorrect decision, the prototype points are moved in the opposite direction.

In the original LVQ rule, the reference vector $\overline{m}$ nearest to the input vector $\overline{x}$ is updated

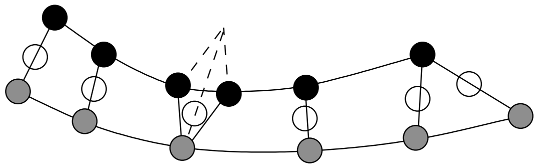


Figure 4.8: The operation of the modified LVQ training. The prototype stroke being moved is identified by the gray circles and the input sample by the black circles. The new locations for the prototype strokes points are shown as open circles.

based on a positive learning coefficient α as in

$$\overline{m}(t+1) = \begin{cases} \overline{m}(t) + \alpha(\overline{x} - \overline{m}(t)), \text{ if } \overline{m} \text{ and } \overline{x} \text{ belong to the same class} \\ \overline{m}(t) - \alpha(\overline{x} - \overline{m}(t)), \text{ otherwise} \end{cases} . \tag{4.30}$$

For the purposes of handwriting recognition, where the prototype and input sample generally do not have the same number of points, a modified rule has been derived (Laaksonen et al. 1998a). The point-to-point correspondence is established using DTW and the modified LVQ training is defined as:

Let $P(h) = (i(h), j(h))$ be the optimal time warping path between strokes $S_1 = (p_1(1), \ldots, p_1(N_1))$ and $S_2 = (p_2(1), \ldots, p_2(N_2))$. The PP distance between the strokes is

$$D_{\mathrm{PP}}(S_1, S_2) = \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))), \tag{4.31}$$

where $d(p_1, p_2)$ is the squared Euclidean distance. If the points of stroke S_2 are moved, their new locations are

$$S_2^{new} = \begin{cases} S_2^{old} - \alpha \frac{\partial D_{\mathrm{PP}}(S_1,S_2)}{\partial S_2}|_{S_2^{old}}, \text{ if } S_1 \text{ and } S_2 \text{ belong to the same class} \\ S_2^{old} + \alpha \frac{\partial D_{\mathrm{PP}}(S_1,S_2)}{\partial S_2}|_{S_2^{old}}, \text{ otherwise} \end{cases} , \tag{4.32}$$

where α again is the positive leaning coefficient, and

$$\frac{\partial D_{\mathrm{PP}}(S_1, S_2)}{\partial S_2}|_{S_2^{old}} = -2 \begin{pmatrix} \sum_{h=1}^{H} \delta(1, j(j))(p_1(i(h)) - p_2(1)) \\ \vdots \\ \sum_{h=1}^{H} \delta(k, j(j))(p_1(i(h)) - p_2(k)) \\ \vdots \\ \sum_{h=1}^{H} \delta(N_2, j(j))(p_1(i(h)) - p_2(N_2)) \end{pmatrix}^T , \tag{4.33}$$

where $\delta(i, j)$ is Kroenecker's delta function. The operation of the modified LVQ training rule is shown in Figure 4.8.

4.6.4 Hybrid approaches

The adaptation strategy $Hybrid(k, \alpha)$ is a combination of the adaptation strategies $Add(k)$ (Section 4.6.1) and $Lvq(\alpha)$ (Section 4.6.3). In this strategy, the k nearest prototypes are examined, and if any of them belong to the correct class for the input character, the nearest prototype is modified using $Lvq(\alpha)$. If none of the prototypes are correct, the input sample is added to the prototype set.

This strategy combines the benefits of being able to learn entirely new writing styles by adding prototypes while minimizing the growth of the prototype set through the use of prototype modification whenever possible. Thus the strategy can be expected to perform better than either adding or modifying the prototypes in itself. This was also shown to be the case (Vuori 1999).

Chapter 5

Palm-top device implementation

The recognition system described in Chapter 4 was originally implemented on a large-scale platform. While being in view of the computational power very effective for running tests, this is in fact quite far from the intended target system scale. Here the creation of an implementation for an actual palm-top device and the implementations features are described.

5.1 Motivation

At a certain point in the development of the recognition system it was found that an implementation on a portable platform would benefit our testing needs. This was because the objective of the research has been to create a recognition system usable in such a context. Also the data collection is expected to be easier to conduct when the writer need not be sitting at a workstation during collection. This can also be presumed to produce characters written in a more natural style as the writer is functioning in a situation similar to that where the device would be intended to be used; in a regular real-life environment.

Implementing the recognition system on a platform with much more limited computational resources also gave rise to several questions regarding the possibility of operating at a tolerable speed level. As it is known, Dynamic Time Warping is a quite computationally demanding approach (Lu and Brodersen 1984), and as such it was expected that it might pose some problems regarding the operational speed. Thus also methods for improving execution times on the small-scale platform became of great interest.

Table 5.1: Palm-top device features

Manufacturer	**Philips**	**Everex**
Model	**Nino 300**	**Freestyle Manager**
Processor	Philips PR31700	NEC VR4102
Processor clock	75 MHz	66 MHz
RAM	8 MB	8 MB
ROM	8 MB	8 MB
Display resolution	$240 \times 320 \times 4$	$240 \times 320 \times 2$
Dot Pitch	0.24 mm	0.24 mm

Table 5.2: Palm-top device writing resolutions

	Philips Nino 300	**Everex Freestyle Manager**
Horizontal point spacing	4.3 ppmm	4.8 ppmm
Vertical point spacing	2.9 ppmm	3.7 ppmm

5.2 Palm-top platform description

For the test beds, two different palm-top devices have been used. The first was a Philips Nino 300 and the other an Everex Freestyle Manager. Some general properties of the devices have been gathered into Table 5.1 (Philips 1998, Everex 1998, Everex 2000). Both of the palm-top devices use the Windows CE (WinCE) operating system version 2.0.

The actual writing resolutions were tested by drawing several vertical and horizontal lines of the same length and averaging the results to estimate the actual resolution. The results of this evaluation are presented in Table 5.2, as measured in points per millimeter (ppmm).

5.3 Implementational Issues

In addition to the expected differences in performance, also the application development environment for the small-scale platform is very different from the standard C++ implementation the recognizer was originally developed on. The development tools used on the palm-top platform were the Microsoft's Visual C++ and the Windows CE Toolkit for Visual C++, of which versions 5.0 and, when they became available, 6.0 were used. The main difference is caused by the approach taken by Microsoft for WinCE Programming.

As the nature of C++ is that things can generally be done in several ways, programmers tend to develop a style of their own when writing program code. With the WinCE Toolkit, a totally different approach has apparently been taken; for the most part, the number of choices available for the programmer has been greatly diminished. This would probably be not so much of a problem if the code being ported had initially been designed for a WIN32 platform. It seems that those two are reasonably compatible in terms of basic classes and functions available. But since the original implementation was written on an UNIX-based system, the need to redirect several basic functions arose. The final outcome was that a new intermediate function interface was constructed, allowing the majority of the recognizer code to work unaltered with the intermediate interface layer. This layer then either performs the necessary operations with functions defined there or redirects the calls to existing WinCE functions. Examples of missing features include standard input and output streams, file streams in general and some basic functions such as atoi, calloc, exit, sscanf and strtol.

All this might not even have been a problem with previous familiarity with WIN32 programming. Without prior experience on these platforms, this basically simple phase of transporting the code and getting it to work on the new platform posed quite a challenge. After getting the software to work on the new platform, the performance was noted to be unacceptably low. This was no surprise, as the phenomenon had been expected.

5.3.1 Stray point removal

When the Wacom tablet was used on the large-scale platform, stray points were not an issue. But with the transition to the smaller-scale platform, the occurrence of stray points became a real concern. This is probably due to both imprecision in the pen point loci capture and the fact that the device is generally used while held in the palm, and as such is not very stable. So, to negate the adverse effects of stray points on recognition performance, a method of stray point removal needed to be implemented for the palm-top platform.

Stray point removal was applied in a simple manner of comparing the distances between three consecutive points, p_1, p_2 and p_3. If the sum of the squared Euclidean distances to p_2 from its prior and next points, p_1 and p_3, respectively, was greater than the squared Euclidean distance between the points p_1 and p_3 multiplied by a removal constant c_r, the point was deemed stray and removed. This can be written as removing the point p_2 if

$$|p_1 - p_2|^2 + |p_2 - p_3|^2 > c_r |p_1 - p_3|^2, \tag{5.1}$$

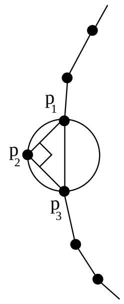


Figure 5.1: Stray point removal checking. When $c_r = 1$, removing the point corresponds to checking if the angle depicted is $\leq 90°$, which also corresponds to the point p_2 being outside the circle

Table 5.3: Effects of stray point removal

Removal constant c_r	**Percentage correct**	**Average recognition time (ms)**
∞ (no removal)	57.8	757
2.5	57.8	781
2	58.0	779
1.5	58.1	755
1.2	57.8	786

where $|\cdots|^2$ is the squared Euclidean distance. The function of stray point removal is shown in Figure 5.1. The functionality of this approach was tested with a prototype set of 273 characters and a test set of 709 characters all written on the Philips Nino. Both character sets included lower and upper case Latin characters and digits. The results are shown in Table 5.3.

As can be seen, using stray point removal with too large values of c_r produces no gains, as points are not removed, but slows down the recognition slightly due to having to do the distance calculations on each input sample. The optimal value found in these tests was $c_r = 1.5$, which produced some gains in both speed and accuracy. As the benefits were small, the number of stray points was evidently not large in the data used, which was also verifiable by visually examining the characters. But this is highly writer dependent, as low pen pressure used by the user is a major factor in creating stray points. For writers applying low pen pressure the effects might be much more obvious and beneficial.

5.4 Computational enhancements

The mere change in platform caused some significant difficulties in achieving acceptable performance, in terms of both accuracy and speed. This required some alterations to the main body of the code. In Table 5.4 the increase in recognition speed after varying

Table 5.4: Performance enhancements achieved with computational rearrangements

Operation performed	**Average time (ms)**	**Speedup factor**	**Total speedup factor**
None (Initial situation)	7084		
Compiler optimizations	5640	1.26	1.26
Distance calculation calls inlined	4543	1.24	1.56
Static memory allocation for matrices	1889	2.40	3.75
Added matrix value presetting	1805	1.05	3.92
Activated category-based ordering	1585	1.14	4.47
Moved to all-integer calculations	145	10.93	48.86

operations are shown. The results have been obtained with a prototype set of only one character for each class, so the results are not fully realistic. But as the same set was used on all runs, they remain perfectly comparable to one another. The very small prototype set was used simply due to the time the entire recognition process took, especially in the early stages of the code. The testing set used consisted of 464 characters.

The initial situation in the table is the result obtained by running the compiled code successfully for the first time. At this time the recognizer code had just been modified to be runnable on the small-scale platform. The first performance improvement was seen merely by activating a suitable set of compiler optimization parameters.

The third level of performance obtained was the result of removing the excessive amount of function call overhead in distance calculations. As compiler inlining of functions was not working properly, this was done manually. As can be seen, also this manual inlining resulted in a notable time decrease of 19.5%.

The first of the most significant benefits was obtained through moving from dynamic to static memory allocation for the most important calculation matrices. This was then combined with pre-setting some of the values on the matrices to provide even more speed gains. These operations are discussed in more detail in Section 5.4.1 below.

The activation of the category-based ordering of prototypes prior to recognition, discussed in Section 4.5, again added to the performance. The ordering had been initially disabled due to some functionality issues which were finally solved at this time.

The final, and clearly most significant boost in performance was seen when moving from floating point to integer calculations. This resulted in a decrease of approximately 91% in recognition time from the previous state. The procedures involved are discussed in more detail in Section 5.4.2 below.

5.4.1 Dynamic memory allocation

Already at the beginning of the development it was known, that the use of dynamic memory allocation on the WinCE platform might cause efficiency problems. Some speed decrease was expected to be amountable to the use of dynamic memory allocation.

For this purpose, the allocation of the tables needed for the dynamic programming routine was moved to the initialization of the program, and the same tables were used in all calculations. The values were just reset between recognition rounds without any freeing or reallocation of memory. The option to increase the table size when needed was naturally implemented but after some initial testing the sizes of the tables were set to a level that rarely needs adjustment. In addition to allocating the memory, also some steps to improve performance by presetting values that could be anticipated prior to calculation, especially in the initial corner of the matrix, were taken prior to the actual classification loop.

An overall decrease of computation time of 58.4% from the static matrices was instantly seen due to allocating the matrices only once during the initialization phase. A little additional gain of 4.4% could be realized through the anticipative value settings.

5.4.2 Floating point calculations

The most notable difference regarding the functionality of the large and small-scale processors is that the small-scale platform's processor has no specialized floating point unit (FPU) for floating point number calculations. With the original routines all being very FPU intensive, as in the large-scale platform this posed no difficulties, it is easy to imagine what the resulting speed of the operations was when the floating point instructions were emulated with integer numbers.

Once this problem was recognized, the program code was modified to use almost exclusively integer operations in calculations in the main recognition loop. It was noted that on the average the most time was clearly spent in this section, over 80% of the total recognition. The use of integer calculations naturally results in slightly reduced precision and the need for more careful overflow control. Still, the speed gains obtained made the switch the only viable option. A comparison using the lower and upper case Latin characters and digits was performed. A prototype set of a total of 273 characters was written with the Philips Nino by one writer. The test set consisted of 709 characters also written by one, but different, writer. They were used to compare the effects of integer and floating point calculations.

Table 5.5: Recognition accuracy and speed comparison between floating point and integer calculations

Calculation method	Recognition time (ms)	Error percentage
floating point	5173	36.2%
integer	757	42.2%

Table 5.6: Comparison of computational power available on different platforms

Platform	Integer result (ms)	Floating point result (ms)
large-scale	1282	2283
Everex Freestyle	3435	60232
Philips Nino	2791	109140

As can be seen in Table 5.5, with the introduction of integer calculations into the dynamic programming loop, the recognition time dropped by approximately 85% from the time needed with floating point calculations. As a downside, the recognition error count also increased by 17% from the error count with floating point use. This drop in accuracy is regrettable but from a usability point of view, the compromise had to be made.

5.4.3 Computational power

As is evident by just looking at the large and small-scale platforms used, there will be differences in computational power. It was expected that the small-scale platform would introduce difficulty especially performance-wise. A mobile MIPS R4000-based processor running at well below 100 MHz can in no way compare to the large-scale platform based on 16 MIPS R10000 processors running at 195 and 250 MHz.

A simple comparison of the computational power involved is shown in Table 5.6. The results were obtained by calculating the first 1000 decimals of π using an algorithm presented in (Rabinowitz and Wagon 1995). The results shown are an average of a total of 30 calculation runs. The difference in integer results, nearly 120% more for the Nino than the large-scale platform, is large but expectable. The floating point difference of almost 50 times more time used, on the other hand, is simply unacceptable.

5.4.4 Data storage space

Another issue encountered at a late point during the actual testing of the system was the limitation of the total of 8 MB RAM available in both palm-top platforms. As the intention is to gather information to further enhance the performance of the system, quite a lot of data is stored during the test phase. The total size of the output files is often in the range of several megabytes.

Thus the 8 MB base memory initially in the device proved too small. Luckily this problem was rather easy to circumvent, as the use of flash memory storage cards is very simple in these devices. Installing a 16 MB CompactFlash storage card in the Philips Nino posed no difficulties but for some reason the cards did not function with the Everex device. This issue is yet to be resolved but it should be just a matter of time before a driver update or another resolution suggestion from the manufacturer will be obtained, as they have been contacted on this issue. Meanwhile the data collected successfully with the Everex will thus be of slightly lower volume than that with the Nino.

5.5 The questionnaire application

We developed a data collection and testing environment for the palm-top platform. The environment was implemented as a questionnaire program. The user interface of the questionnaire application is shown in Figure 5.2. The basic idea is that the program shows questions to be answered in the top region called the question area. The user then uses either one of the two text-input boxes at the bottom, and the buttons on their right, to input the desired answer. The answer appears in the middle section referred to as the answer text area as it is being inputted. The insertion cursor is a blinking vertical line. If a portion of the text has been activated, it is shown on a black background, as is common in text-editing interfaces.

The questions were designed to provoke answers with more than just one or two letters. To enforce the gathering of sufficiently many characters, a minimum number of characters required before proceeding to the next question has also been implemented. The minimum number is question specific, and has been designed to be realistic in terms of an average answer. The limits range from two to twenty input characters. In between questions the question area is also used to show a message of what the device is doing, for example “Processing, please wait...”.

5.5.1 Inputting text

To input text, the user can type in either of the text-input boxes at the bottom in Figure 5.2. Characters are separated with timeouts or when the pen touches area outside the box. Thus, when desired by the user, text can be inputted at a high speed by simply alternating between the two text-input boxes. The recognizer was designed to be able to recognize all upper and lower case letters from a to z and digits. In

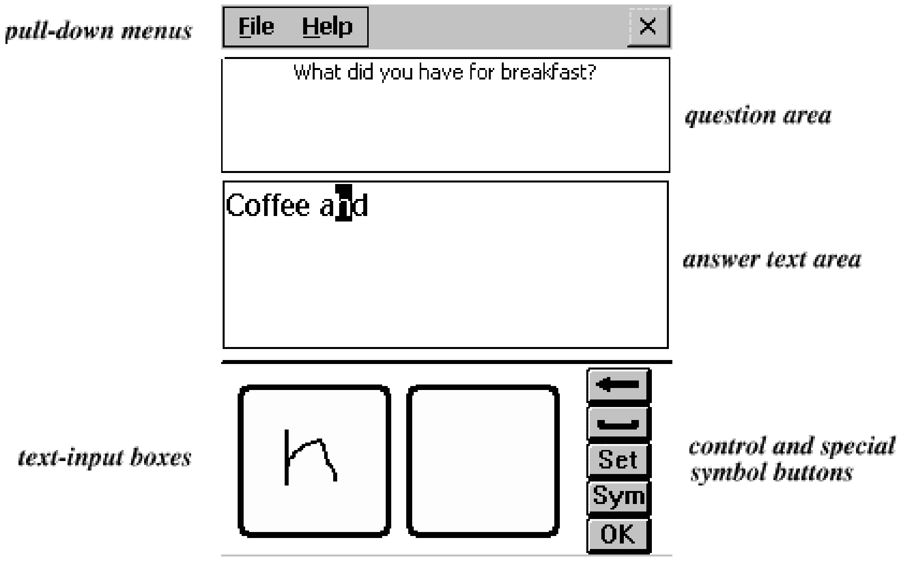


Figure 5.2: User interface of the palm-top questionnaire program

addition a single horizontal stroke from left to right corresponds to space. A stroke drawn in the opposite direction is interpreted as a backspace.

The topmost button to the right of the writing boxes, marked with an '←', also produces a backspace, and the button directly below it is for inserting a space. The *Set* button is used when the recognizer is simply unable to correctly label the input. It pops up a chart with all available labels and the user can select the desired label for the active character. The chart is also used to prompt the user for the labels for characters that the system was unable to recognize. Cases where no recognition result is obtained occur when the number of strokes in the input character exceeds that of any character in the prototype set.

The *Sym*, short for symbol, button is used to input special characters not recognized, such as '?', '% ', '(', ',' etc. They are available mainly for pleasing the user's eye, as they have no effect on the functionality of the recognizer and are actually internally treated as spaces. The last button, *OK*, indicates that the user has answered the question and is ready to move on to the next one. Pressing it first instigates a check whether enough characters have been written. If there is enough input for that question, the adaptation phase and movement onto the next question take place.

Text in the answer area, the middle section in Figure 5.2, is inputted by using the control mechanisms described above. The text can be edited by deleting and replacing using standard Windows text-editing mechanisms, with the exception that edit commands, such as cut, copy and paste, have not been included. Text can be activated by dragging the pen point over the desired area of text, and the cursor can be relocated by tapping in the edit window. For the *Set* operation, if no single character has been

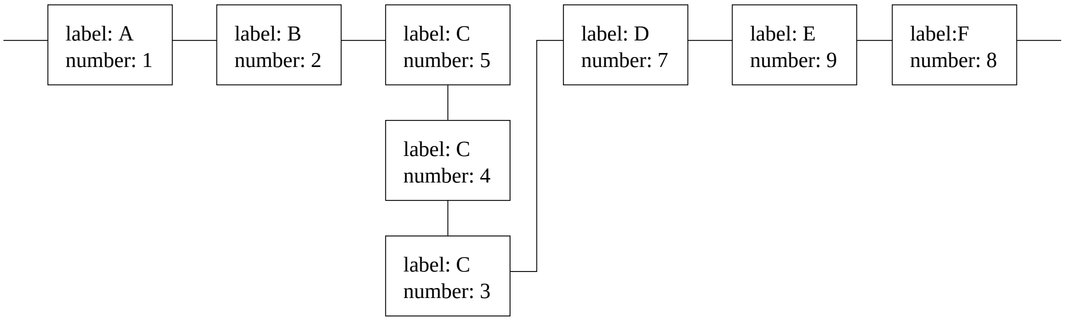


Figure 5.3: A diagram illustrating the data structure for the data collection program

selected, the active character is taken to be the one directly on the left of the cursor. An active area can be deleted by using the backspace character or button, or replaced by a single character inputted by writing or pressing the space or *Sym* button.

5.5.2 Determining labels

The underlying data structure to store characters and their labels is a simple linked list, where each character is stored along with its label, input index and on-screen position information. This list is modified during the writing process, and the first item with a given position always corresponds to the character currently seen in the editing section. If more than one character has been inputted and retained for a given position, they all are assigned the label of the topmost character in the "pile" of that position.

A schematic example of the data structure is shown in Figure 5.3. The example depicts the user having written the the first six capital letters. The first two were inputted successfully and in order, but the character "C" has needed three attempts for the correct recognition. Then the user has inputted a character that was later deleted, as the index number 6 is non-existent. Then the letters "D" and "F" were inputted, and "E" was inserted afterwards.

The most difficult task in the user-interface construction is that in order for successful adaptation to be possible, each character must be assigned a correct label. For this purpose, a number of cases can be isolated:

1. A character has been written and left unchanged. Such a character is considered to have been correctly recognized.
2. A single character has been replaced with another character. The replacement can be performed either through activating the character and replacing it implicitly or by deleting the character using the backspace and inputting a new character directly afterwards. In either case it is deduced that the initial character was

incorrectly recognized and the label of the new character is also assigned to the underlying character. Both characters are kept in the input character list.

3. A single character has been relabeled using the *Set* button. Then the label of the character is simply changed to that received from the user.

4. Several characters have been replaced with one input of any kind. This is thought to indicate the user's change of mind, and in such a situation nothing concerning the labels of the characters being replaced can be assumed. The replaced characters are thus discarded.

5. One or more characters have been replaced by a symbol. In such a case the writer has clearly desired to delete the characters. The characters are removed from the list of learning samples.

6. Several consecutive backspaces have been received. Also in this case the correctness of the characters being deleted cannot be established and as such they are removed from the list.

By the use of these rules the list containing the inputted characters is kept up to date and the labels therein are assumed correct for the adaptation process. Naturally the possibility of incorrect labels still exists, but through these principles the labels should be correct, if the user has noticed and taken care to correct all recognition errors before submitting the answer. The only unsolved situation is that of the user's change of mind for a single character, meaning that the user writes one character, deletes it and writes an entirely new one instead. Such situations are confused for case 2 above. It was thought that the error correction, the basis for case 2 above, is more important and common, so the possibility of error was deemed a reasonable risk.

5.5.3 Adaptation

Adaptation is performed after submitting an answer to a question. This has the drawback that users might be somewhat confused when the system at first does not seem to learn anything but then suddenly, when entering the next question, has indeed adapted. The problem is not an issue as long as the user is aware of this behavior. If adaptation were performed during writing, the correctness of the training samples could not be ensured. In many cases the need to revert to a previous stage would arise, increasing the requirements for both computational power and storage capabilities to beyond what is available for algorithms on the palm-top platform. When the adaptation is performed after the user is satisfied with the answer to the question, it can be quite reliably assumed that he or she left no errors. Then the adaptation can be confidently performed on the labels deduced according to the scheme presented in Section 5.5.2.

The actual adaptation is performed with the $Hybrid(k, \alpha)$ adaptation method described in Section 4.6.4. The values for k and α were previously determined the most effective at 3 and 0.3, respectively (Vuori et al. 1999), and these values were used also for the palm-top implementation.

5.5.4 Data collection

During operation, an exhaustive amount of data is stored in a log file for the entire process. The amount of information is in general enough to reproduce all events during data collection. Each entry in the log file holds a time stamp to help trace the progress of the experiment.

In addition to storing the actual characters written and their deduced labels, also all button presses and the contents of the editing section on any change are logged. The question being answered is also stored, as is information on any notable internal operation such as alterations in the contents of the storage list, adaption phase information on what adaption was performed, and, if applicable, what prototype was altered. The additional information is stored as comments in the UNIPEN-format file containing the input data. Due to the log file adhering to the UNIPEN file format, glyphs stored in the log can be browsed with a UNIPEN viewer.

5.6 Speedup methodologies

Due to insufficient recognition speed being a major issue in the palm-top platform implementation, methods for speed increase have been examined. Some of them have yet to be implemented on the actual platform since they have been tested on the large-scale system. The experiments have mainly been performed in the large-scale system as running extensive tests on the small-scale system is very cumbersome and difficult due to both storage space and computational power insufficiencies. Two speedup approaches are examined in more detail below.

5.6.1 Decimation

The operator $Decimate(n)$, described in Section 4.2, was an obvious means for improving recognition speed. Due to the nature of removing information-containing points

Table 5.7: Performance effects of decimation

Decimation level	Percentage correct	Average recognition time (ms)
None	58.1	754.8
1	57.0	473.4
2	53.0	387.0
3	42.0	347.3
4	44.1	312.5

from the incoming data, an extensive amount of decimation produces notable decreases in the recognition performance (Vuori 1999).

The effects of decimation were tested on the palm-top platform with a prototype set of 273 prototypes and a test set of 709 characters. Both upper and lower case Latin characters and digits were included. The results have been gathered into Table 5.7.

As can be seen, the recognition speed increases notably when moving from no decimation to 1 or further to 2 point decimation. Also the accuracy loss is acceptable for the system, as the adaptive recognition, which was not included in these simulations, helps improve the final accuracy. But with 3 or 4 point decimation, the loss in recognition accuracy is very prominent, and the recognition speed gains are no longer worth the loss. Thus the operation $Decimate(2)$ was chosen for use on the palm-top platform.

5.6.2 Predictive calculation aborting

The second speedup method examined focuses on trying to predict whether the result of the distance calculation will exceed the maximal allowed value, the minimal distance calculated so far. As the calculation algorithm already includes an option to abort calculation if the shortest distance already found is exceeded, the prediction merely causes the calculation to be terminated at an earlier stage.

The form of the prediction function was chosen based on modeling previously gathered data sequences of distance cumulation at varying points of matching. The prediction function $d_{est}(i)$ at point i of the stroke S with a total of I points is in the form of

$$d_{est}(d_i, i) = d_i(\frac{\alpha I}{i} + \beta), \tag{5.2}$$

where d_i is the calculated column-minimum distance from the dynamic programming algorithm at the point i. The form of the prediction needs to be selected so that it does not enormously overestimate the total distance, which would give rise to excessive abortings.

Figure 5.4(a) shows the average behavior of $D(S)/d_i$ as a function on $\frac{i}{I}$, where $D(S)$

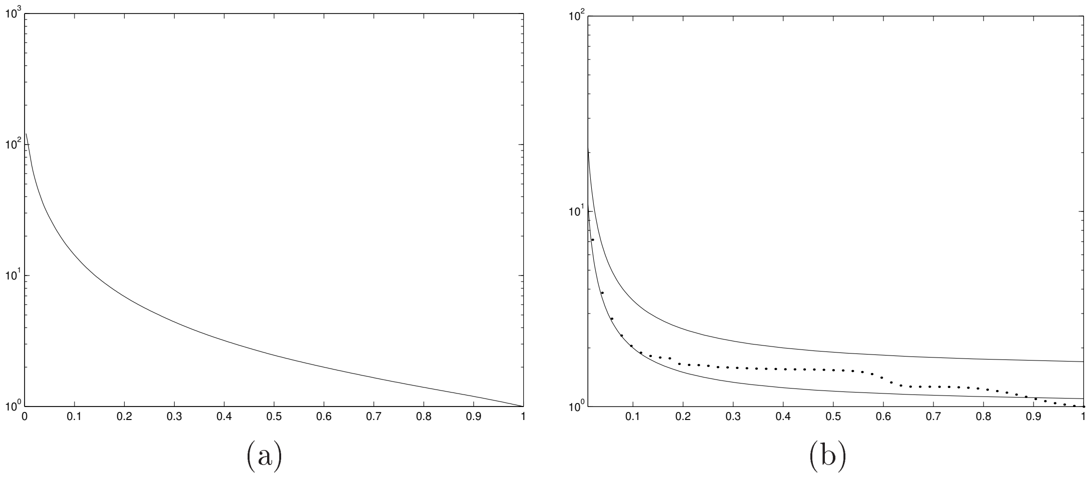


Figure 5.4: Plots of $D(S)/d_{colmin}(i/I)$; a) Averaged and b) Minimal (points) along with the functions $0.1/x + 1$ and $0.2/x + 1.5$

is the total distance resulting from the matching of S. The averaged matchings are all final correct results for each character, ie. the ones with the smallest total distance with the requirement that the closest match was indeed correct. In Figure 5.4(b), 0.1% of the smallest values have been removed and then the minimum of the remaining ones has been plotted. Two example fittings, the functions $\frac{0.1I}{i} + 1$ and $\frac{0.2I}{i} + 1.5$, have also been included to demonstrate the behavior of the minimums.

As the plots are the relation between $D(S)$ and d_i, it can be noted that precisely the minimum is the value of most interest, as that is the deciding one in the final prediction. The removal of the 0.1% can be expected to directly translate to errors, but it enables calculation aborting with some probability to take place.

This method was tested using a test set consisting of a total of 29292 characters from 24 writers. They were classified with a prototype set containing 7 prototypes for each class, a total of 476 prototypes produced with the semiautomatic clustering method described in Section 4.4. The sets included the lower and upper case Latin characters and digits. Some results of average recognition times per character, recognition accuracy and the fraction of prediction-based calculation interruptions obtained with this prediction method are presented in Table 5.8. The last column entitled "Activation percentage" represents the percentage of the matchings aborted due to the prediction algorithm of all matchings initiated.

As can be seen, the computational speed can be improved noticeably, and in some cases also the recognition accuracy improves. The improvement of the recognition accuracy is probably more of a coincidence, as this suggests that there are some character-prototype-pairs in the set where the end part produces a very good match even though the prototype belongs to the incorrect class. But as can be seen, the recognition time

Table 5.8: Performance effects of prediction

Variables α	β	Average recognition character (ms)	Error percentage	Activation percentage
Reference		304.6	34.3	0.0
0.1	0.0	179.6	34.5	43.0
0.1	1.0	165.5	33.9	81.7
0.2	1.0	127.7	33.8	83.1
0.3	0.75	113.5	34.3	82.4
0.2	1.5	112.6	36.1	85.2
0.325	0.75	108.7	34.5	82.9
0.4	0.75	95.8	35.0	83.8
0.45	0.75	89.4	35.8	84.2
0.5	1.0	80.9	37.0	85.3
0.9	0.0	65.9	40.2	84.4
1.0	1.0	57.3	43.5	87.0

can be reduced by 63% with no negative effects on the recognition performance, and by 71% with just an increase of 4% in the recognition error.

If the prediction algorithm can be implemented in a way feasible in the small-scale platform, these speed increases are extremely promising. The main problem is that for a function of the form $1/x$ the calculation of the values is by no means trivial for the integer processor. This will likely cause less significant benefits in terms of faster recognition. One possible implementation would be through the use of a look-up table for the values at varying points. When using slightly less strict predictions the unavoidable rounding error should not have any significant effect on the final outcome.

5.7 Current level of performance

The current performance difference between the large and small-scale platforms was tested by running an identical batch run on both systems. The prototype set consisted of 273 prototypes and a test set of 709 characters. All characters were written on the Philips Nino and included the upper and lower case Latin characters and digits. The results have been gathered into Table 5.9.

As is evident from both Tables 5.6 and 5.9, the small-scale platform simply cannot

Table 5.9: Performance difference between the platforms

Platform	Percentage correct	Average recognition time (ms)
Palm-top	53.0	387.0
Large-scale	61.5	51.9

compete with the large-scale platform in terms of computational power. The need to avoid floating point operations results from the lack of a specialized FPU in the palm-top devices. The use of integer operations does help speed up computation, but also results in slight deterioration in recognition rate, as was evident in Table 5.5. Thus this comparison showing a 645% increase in computation time along with a 14% deterioration in recognition rate is of no surprise.

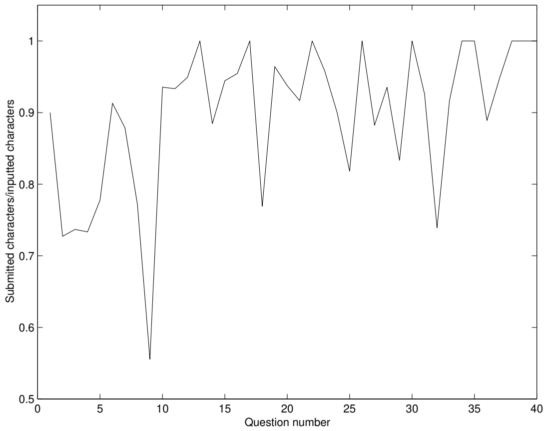
The speed of human handwriting input is approximately 18.5 wpm, with a word being defined as approximately five characters (MacKenzie et al. 1994). This translates to approximately 1.5 characters per second. The recognition should then be completed within approximately 650 milliseconds. As can be seen, both platforms reach this rate when operating on batch runs.

Sadly though, the actual recognition speed of the palm-top device drops quite noticeably from the batch run rate due to overheads from the user interface and data storage. The actual recognition rate during the use, the time it takes between the stopping of writing and receiving the result from the recognizer, was measured at approximately 450 ms during an actual test run. This attributes an approximately 17% overhead to the recognition performed on exactly the same input data and prototypes in batch mode. When the user-interface and data storeage overheads are added, the recognition time can be said to be still simply too long, as the user needs to wait for the recognition result to appear before continuing to write. A slight relief is provided by the user interface, as it allows characters to be written at a faster rate, with the results just appearing after a delay.

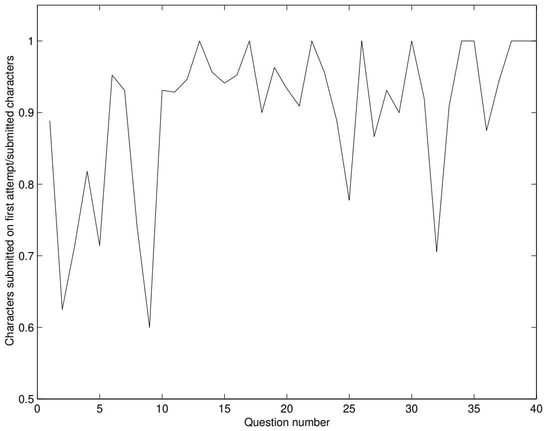
5.8 Experiments and results

Evaluation of the recognizer performance in a real-world application, such as the questionnaire application, described in Section 5.5, is not as straightforward as simply calculating the recognition rate in correct recognitions versus all recognition attempts. The true label of the input sample is not known at the time of recognition, and labelling the samples afterwards manually is a very laborious task. Also in some cases the label is also ambiguous for the human reader.

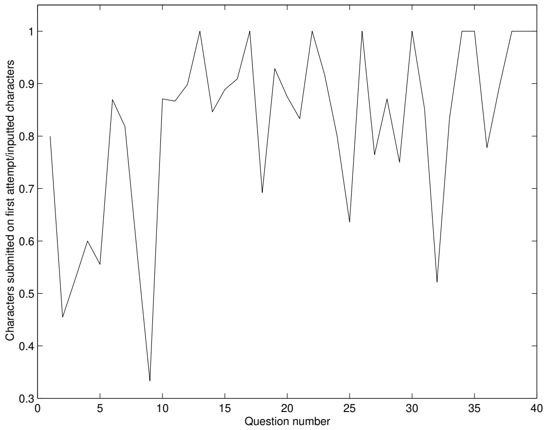
Some applicable measures identified for this purpose and are shown in Figure 5.5. These results were obtained with a prototype set consisting of 273 characters and a test set of 709 characters, written by different writers. All characters were written on the Philips Nino and included the upper and lower case Latin characters and digits. The criteria include the ratios between the counts of submitted characters and all input characters (Figure 5.5(a)), characters submitted with one attempt and all submitted characters



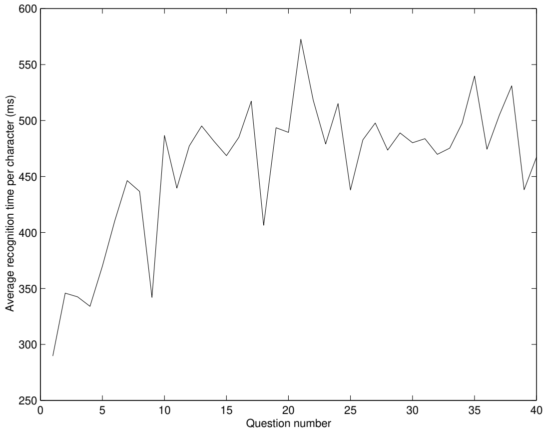
(a) Ratio of submitted characters to all inputted characters



(b) Ratio of characters submitted on the first attempt to all submitted characters



(c) Ratio of characters submitted on the first attempt to all inputted characters



(d) Average recognition time per character

Figure 5.5: Plots of the four criteria devised to evaluate recognizer performance

(Figure 5.5(b)) , or characters submitted with one attempt versus all input characters (Figure 5.5(c)). Here submitted characters refers to the input samples that were left in the storage list when the user deemed the question answered. Also the recognition time per character (Figure 5.5(d)), and its development, are naturally of interest. Additional information on the performance can be gained from simply questioning the subject as to whether performance at various points of the questionnaire was acceptable.

It can be noted, that the average recognition time overall was 452 ms, starting at approximately 300 ms and increasing, due to the addition of prototypes, to approximately 480 ms after the first 10 questions. Also all the ratios, on the average, improved towards the end of the questionnaire. This suggests that the adaptation of the system was successful, as the user did not need to re-input, or correct, as many input samples as in the beginning of the questionnaire.

Chapter 6

Adaptive committee recognition

This chapter deals with experiments performed with adaptive committee classifiers. Mainly the Dynamically Expanding Context (DEC) (Kohonen 1986, Kohonen 1987) approach has been used, but also some other methods have been examined mainly to serve as reference classifiers to evaluate the performance of the DEC-based committee. The classifiers and results obtained are discussed in the following sections.

6.1 Motivation

Even though rather impressive results have been obtained with the Dynamic Time Warping-based recognizer using single classifier adaptation (Vuori 1999), the question as to how these results could still be improved on was left open. As in general a combination of recognizers can be expected to perform better than any of its members, using an adaptive committee approach was a very logical step.

When searching for a suitable method of committee adaptation the idea of using the DEC principle, previously mainly used for speech recognition (Kohonen 1986, Torkkola and Kohonen 1988, Torkkola 1993, Kurimo 1998), arose. The method is described in detail below, as is its modified version used in our handwriting recognition project.

6.2 Dynamically Expanding Context

Here the main principle of the Dynamically Expanding Context is explained. Both the original principle and an illustrative experiment on context directions are described.

6.2.1 The original DEC principle

The original algorithm was developed to create transformation rules that would correct typical coarticulation effects in phonemic speech recognition. The notation for a DEC rule stands as

$$x(A)y \rightarrow B, \tag{6.1}$$

where A is a segment of the source string S, B is the corresponding segment in the transformed string T, and $x(\cdot)y$ is the context in string S where A occurs. So in other words A is replaced by B under the condition $x(\cdot)y$. (Kohonen 1986)

The main philosophy behind the approach is to determine just a sufficient amount of context for each individual segment A so that all conflicts in the set of training samples will be resolved (Kohonen 1987). Thus an optimal compromise between accuracy and generality is expected to be obtained.

In the DEC principle a series of stepwise expanding frames, each consisting of an increasing number of symbol positions on either or both sides of A, is created. The nth frame in the series is said to correspond to the nth level of context. Examples of contextual levels for the letter "s" in the string "eisinki", an erroneous form of "helsinki", are shown in Table 6.1.

The central idea of the method is to always first try to find a production of the lowest contextual level to sufficiently separate contradictory cases. Starting with context level 0, or context-free level, contexts of successively higher levels will be utilized until all conflicts are resolved.

In Kohonen (1987) it is stated that in order to define the context-dependent production rules one has to first find the context-independent production rules. This means finding out which segments in the transformed string T match best with the given segments in the source string S. This was performed using the weighted Levenshtein distance

Table 6.1: Two examples of contextual levels for the letter "s" in the string "eisinki"

Level	**Context I**	**Context II**
0	-	-
1	i(·)	(·)i
2	i(·)i	i(·)i
3	ei(·)i	i(·)in
4	ei(·)in	ei(·)in

defined as

$$D_{Levenshtein}(S, T) = \min(pa + qb + rc), \tag{6.2}$$

where T is obtained from S using a replacements, b insertions and c deletions. p, q and r are the weighting coefficients for each respective operation. The weights can be determined heuristically or obtained statistically from the confusion matrix of the alphabet as the inverse probability for a particular type of error. The confusion matrix is a matrix depicting for each character what characters it was confused with, and how often, during the recognition. The minimum of (6.2) is sought using an algorithm based on dynamic programming described in (Kohonen 1987).

Because the true size of the context cannot be known beforehand, the training data might have to be presented multiple times iteratively. To enforce correct rule creation, a conflict bit is used for each rule. This bit remains 0 as long as the rule is valid, and is changed to 1 if the rule needs to be invalidated. Using this knowledge, the final contextual levels required can be determined automatically in a hierarchical search. Estimation procedures for cases where the search for a suitable rule is unsuccessful are also presented in (Kohonen 1987).

This original approach was tested by Kohonen (1986) with several training sets consisting of 5000 to 9000 words. With the training samples, sets of correction rules with over 10000 productions were established. When using a familiar vocabulary the transcription accuracy rose from 68.0% to 90.5% as a total of 70% of the errors were corrected. With unfamiliar words the transcription accuracy improved from 66.0% to 85.1%, an improvement of 56% errors corrected.

6.2.2 Effects of specificity hierarchies

In (Torkkola 1993) the DEC principle is applied to the transformation of English text into phonemes on the basis of the local letter context. Different specificity hierarchies are explored, namely to left only, to right only, symmetrical starting from left, symmetrical starting from right, left-weighted and right-weighted. An example of these is shown in Table 6.2, where two alternatives for the context expansion of the word "abbreviation" are depicted.

After having been tested with a 18008 word training set and a 2000 word testing set, the specificity hierarchies were ordered in performance. The results are shown in Table 6.3. As can be seen, the direction of the learning hierarchy can have a surprisingly strong effect on the final performance.

Table 6.2: Examples of expanding learning hierarchies

Context level	**symmetrical expansion**	**right-weighted expansion**
0	(t)	(t)
1	(t)i	(t)i
2	a(t)i	(t)io
3	a(t)io	a(t)io
4	ia(t)io	a(t)ion
5	ia(t)ion	a(t)ion_

Table 6.3: The performance of different specificity hierarchies

Context expansion scheme	**Accuracy (%)**
to left only	78.1
to right only	86.6
symmetrical, starting from left	89.0
symmetrical, starting from right	90.6
1/3 right, 2/3 left	86.8
2/3 right, 1/3 left	90.8

6.3 DEC-based adaptive committee classifier

The DEC principle needed to be slightly modified to suit the use as a committee combinator of classifiers in the setting of this thesis. This section covers the used version of the algorithm and the implemented options, as well as the actual implementation of the algorithm.

6.3.1 Overview

In the DEC-based committee the classifiers are first initialized and then tested separately and ranked in order of decreasing performance. The outputs and the second-ranking results of the member classifiers are used as a one-sided context for the creation of the DEC rules. A schematic diagram of the DEC-based adaptive committee classifier is shown in Figure 6.1. In this example there are three member classifiers. The first-ranking results are denoted symbolically as a, b and c, and the second-ranking ones as d, e and f.

Each time a character is input to the system, the existing rules are first searched through. If no applicable rule is found, the default decision, which can be for example the result of the best individual classifier or a majority voting result, is applied. If the found rule, or default decision, results in an incorrect recognition result, a new rule is created.

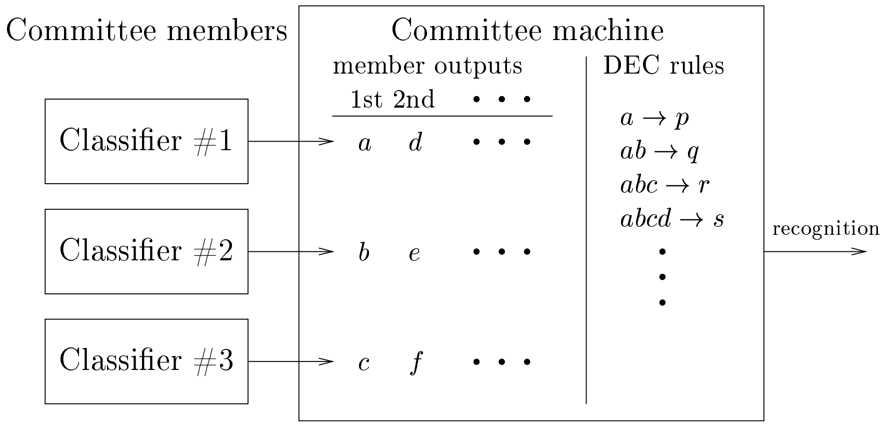


Figure 6.1: A block diagram of the DEC-based adaptive committee classifier

Whenever a new rule is created, it employs more contextual knowledge, if at all possible, than the rule causing the conflict. Eventually the entire context available will be used and more precise rules can no longer be written. For this situation a method for tracking the correctness of the rules can be used and the highest level rule most likely to be correct be applied.

In the case of off-line training the training set can be reiterated until rule consistency is ensured. But as the objective of this research is an on-line system, storing all previous input samples and using them in an iterative manner would be too expensive in terms of both computational performance and storage space. Thus it is assumed that prior samples will not be available after they have been recognized.

The introduction of a new writer always results in the re-initialization of the rule base, as the adaptation is aimed to be user-dependent. The previous rule sets can naturally be stored for use if input from the same writer will again be received.

6.3.2 Example of operation

An example of the operation of the DEC-based committee is presented in Table 6.4. The columns of the table show the correct class of the input character, the first outputs of the member classifiers, the existing rules, the output of the committee, its evaluation, and the action taken.

In this example it is presumed that no rules exist prior to receiving the first sample. The default decision is taken to be the result from the best individual classifier. A series of hardly distinguishable letters 't' and 'f' are input to the system, and the operation proceeds as follows.

On the first round, the input sample is a 't' and is classified as such by the best-ranked

Table 6.4: An example of DEC rule generation

Round	Input	Members	Rules	Output	Result	Action
1.	t	t f t	—	t	ok	—
2.	f	t f t	—	t	err	add rule: t f - $\rightarrow$ f
3.	f	t f t	t f - $\rightarrow$ f	f	ok	–
4.	t	t f t	t f - $\rightarrow$ f	f	err	add rule: t f t $\rightarrow$ t
5.	t	t f t	t f - $\rightarrow$ f t f t $\rightarrow$ t	t	ok	—

classifier. As no rules exist, this result is used for the result of the committee. Since the result is correct, no further actions need to be taken.

On the next round, the members' outputs are identical, but the correct label would have been 'f'. Thus the committee's result is incorrect, and a new rule is added. As no rules for such a case exist, the use of a one-character context is sufficient to resolve the conflict and the rule "t f - $\rightarrow$ f" is inserted. The third round presents an identical situation as the second, but as the rule created on the previous round matches the inputs, it is used to produce correct output.

The fourth round demonstrates the incorrect result from the previously created rule. This leads to the generation of the new rule "t f t $\rightarrow$ f". Using this new rule the committee is then again able to correctly classify the identical situation in round five, as the most context-having rule is used.

6.3.3 Options

Several options were explored in the search for the best achievable recognition result using the committee. Herein those variables are explained.

Default decision

The system's default decision rule is needed when no character-specific rules yet exist. Two methods for producing the default decision were experimented with. The first is to simply use the output of the best-ranked classifier as the default decision for the committee. The alternative is to perform majority voting on the results obtained from the classifiers to make the default decision.

Requiring the inclusion of the output

Another variation implemented was the possibility to require that the output symbol x for a rule of the form $A \to x$ must be included in the context A. In other words, one of the classifiers must produce the result for it to be the output of the committee. Activation of this option will hinder the creation of malicious transformation rules.

Use of second-ranking results

The committee can function either by using just the first-ranking results from its member classifiers or by also including the second-ranking results. The second-ranking results can be used in two ways, either horizontally or vertically.

The horizontal inclusion of the second-ranking results means that first the first-ranking result from the best-performing individual member is used. Then the second-ranking result is used from the same classifier, and after this the two results from the second-best performing classifier in the same order, then the third classifier and so on. In Figure 6.1, this corresponds to the order 'a', 'd', 'b', 'e', 'c' and 'f'.

The vertical approach to second-ranking result use is an indication of all first-ranked results being used prior to second-ranked results from any classifier. So the first-ranked result of the best classifier is followed by the first-ranked result from the second classifier and so on until all first-ranked results have been used. Then the second-ranked results are used in a similar fashion. This approach corresponds to the order 'a', 'b', 'c', 'd', 'e' and 'f' in Figure 6.1.

Correctness tracking

The initial version of the DEC implementation simply discarded rules as they resulted in an incorrect answer but this was quickly seen to be suboptimal. Hence three options were implemented to discriminate between conflicting high-level rules. These are 1) inactivation of the latest incorrect rule, 2) counting the correct applications and using the one with most correct results at the time, or 3) counting both the correct and incorrect applications and making the decision based on their difference.

6.3.4 Implementation

The DEC-based committee classifier has been implemented only on the large-scale platform described in Chapter 4. The committee classifier is a separate software component capable of using the recognition result information received from any number of individual classifiers to perform the committee operation.

The rule base for each writer is stored as a simple doubly-linked list and can be exported into a file for examination and restored for further use. In addition to the overall recognition performance information, the counts of correct and incorrect uses for each rule are also tracked.

For the experiments presented in this thesis, the committee was implemented and run in batch mode simulating on-line operation by taking data in its original order and disallowing reiteration. The individual member classifier outputs were created beforehand and stored in files with a format consisting of the information on the writer, the true label of the character, distance to the nearest prototype as well as its label and order number, and the same information for the second-nearest prototype.

6.4 Reference committee classifiers

In order to evaluate the performance of the DEC-based adaptive committee, some reference classifiers need to be used. The first two presented here are very simple adaptive committee combination methods. The third one uses a slightly more complex approach, and also incorporates information on the distances between the input characters and the prototypes into the decision-making scheme. All adaptation was performed in a user-dependent manner, and values were reset in between writers. The fourth, non-adaptive reference classifier used is a basic majority voting scheme, as presented in Section 3.7.1.

6.4.1 Adjusting best committee

The adjusting best committee is perhaps the most simple form of committee adaptation. The main idea is to select the best classifier for each individual writer by evaluating each classifier's performance during operation and use the result from the classifier that has performed the best up to that point.

The performance evaluation was conducted by simply keeping track of correct results obtained from each classifier. At any given time the committee's decision is thus the result from the classifier with the highest correct answer count at that point.

6.4.2 Adjusting majority voting committee

Another very simple approach to adaptive committee decisions is to use a variation of the traditional majority voting rule presented in Section 3.7.1. Adaptation was implemented by introducing weights based on an evaluation of correctness for each voting classifier as in

$$w_i = \frac{c_i + 1}{\sum_{j=1}^{N} c_j + 1}, \tag{6.3}$$

where w_i is the weight for the output of classifier number i, c_i is the count of correct recognitions from that classifier and N is the total number of classifiers. The addition of one in both the nominator and denominator has been performed merely to avoid both zero weights and divisions by zero before any correct recognition results have been obtained.

With this weighting, the final majority voting decision in (3.10) is modified to assigning $Z \to \omega_j$ if

$$\sum_{i=1}^{N} w_i \Delta_{ij} = \max_{k=1}^{C} \sum_{i=1}^{N} w_i \Delta_{ki}, \tag{6.4}$$

where the sum on the right hand side equals to the weighted sum of the votes received for the class k from the N recognizers. Δ_{ki} is defined by hardening the *a posteriori* probabilities $P(\omega_k|x_i)$ to binary values as in (3.11).

6.4.3 Modified Current-Best-Learning committee

The Current-Best-Learning (CBL) algorithm (Russell and Norvig 1995) strives for a consistent hypothesis for the entire set of samples by generalizing or specializing an initial hypothesis. The original algorithm uses backtracking to ensure that the hypothesis is also consistent with all prior samples. The specialization operation indicates that a unit, a location within the hypothesis space, that was previously positive must be deemed negative, and the generalization then refers to setting a previous negative to positive.

The actual algorithm used for this reference committee has actually grown quite far from that initial idea, but as the resemblance is still evident, it is here called the Modified Current-Best-Learning (MCBL). As in the original version, the data space is a two dimensional grid. The use of just a positive and negative value would require a separate class for each sample, and this would by no means be practical. So the values used here are in a way estimates of the confidence in a particular decision, and are defined as

$$c_j(\overline{x}) = 1 - \frac{d_j(\overline{x})}{d_1(\overline{x}) + d_2(\overline{x})}, \tag{6.5}$$

Table 6.5: Description of the hypothesis space using the confidence values

	Classifier 1	Classifier 2	Classifier 3	Classifier 4
a	$p_1(a)$	$p_2(a)$	$p_3(a)$	$p_4(a)$
b	$p_1(b)$	$p_2(b)$	$p_3(b)$	$p_4(b)$
c	$p_1(c)$	$p_2(c)$	$p_3(c)$	$p_4(c)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

where $c_j(\overline{x})$ is the confidence outputted for the sample $\overline{x}$. $j \in \{1, 2\}$ is the index indicating whether the confidence value is being calculated for the first or second-ranking result, and $d_1(\overline{x})$ and $d_2(\overline{x})$ are the distances to the first and second-ranked prototypes, respectively.

By collecting the values and combining them into class-wise confidence values $p_k(\omega_j)$, where k is the number of the classifier and ω_j the class, the data structure depicted in Table 6.5 is obtained. The decision of the committee is simply that member classifier's result obtained which has the largest confidence value. To modify the hypothesis, the values $p_k(\omega_j)$ are adjusted when the committee as a whole incorrect. So when an individual classifier k is correct, the confidence of the result $c_j(\overline{x})$, for that classifier is added to the overall confidence of the class for that classifier, $p_k(\omega_j)$ when the input sample i was classified as ω_j. On the other hand, when a classifier produces an incorrect result, its total confidence is reduced by the corresponding amount, but not below zero. When the committee produces the correct result, the current hypothesis has been effective and no changes are made.

The confidence values were initialized as the inverse of the ordering of the classifiers according to their decreasing recognition performance. Thus, the first classifier had an initial confidence of 1 for all classes, the second $\frac{1}{2}$ and so on.

6.5 Experiments and their results

Here the DEC-based committee's performance under various combinations of the available options is examined and the benefit of using each option evaluated. The DEC-based committee is also compared to the reference classifiers in order to see how the performance fares in comparison to other committee methods, adaptive and not.

6.5.1 Description of the data sets

The data used in these experiments was collected on the large-scale system and stored in UNIPEN format as described in Section 4.1. The details of the databases are sum-

Table 6.6: Summary of the databases used in the experiments.

Database	Subjects	Left-handed	Females	Characters	(a-z,0-9)
DB1	22	1	1	$\sim 10\,400$	8461
DB2	8	0	5	$\sim 8\,100$	4643

marized in Table 6.6.

Database 1 consists of characters which were written without any visual feedback. The pressure level thresholding of the measured data into pen up and pen down movements was set individually for each writer. The distribution of the classes (a-z, A-Z, å, ä, ö, Å, Ä, Ö, 0-9, (,), /, +, -, %, $, @, !, ?, :, ., and ,) was somewhat similar to that of the Finnish language.

Database 2 was collected with a program that showed the pen trace on the screen and recognized the characters on-line. The minimum writing pressure for showing the trace of the pen on the screen and detecting pen down movements was the same for all writers. The distribution of the character classes (a-z, A-Z, å, ä, ö, Å, Ä, Ö, and 0-9) was nearly even.

None of the writers of Database 1 appeared in Database 2. Only lower case letters and digits were used in the experiments. Database 1 was used for forming the initial prototype set which consisted of 7 prototypes per class and Database 2 was used as a test set.

6.5.2 Member classifiers

The adaptive committee experiments were performed using a committee consisting of four individual classifiers. All the classifiers are based on DTW calculations using either the point-to-line (PL) or normalized point-to-point (NPP) distances discussed in Sections 4.3.2 and 4.3.1, respectively. All classifiers used the *MinMaxScaling* operator and either the *MassCenter* or *BoundingBoxCenter* operation, which all were described in Section 4.2. The configurations and error rates of the member classifiers are shown in Table 6.7.

In general it can be stated that a committee could be expected to perform the better the lesser the errors made by its members are correlated. Unfortunate uncorrelatedness is not the case here. As the DTW-based classifier was the only one capable of acceptable recognition performance, all the member classifiers are rather similar. This can also be seen in Table 6.8, where the occurrence of instances when the members produce the same error is compared with the occurrence of different errors. The columns “First

Table 6.7: Recognition error rates of the four committee member classifiers

Number	**Distance measure**	**Bounding box**	**Mass center**	**Errors**
1	PL		•	14.9%
2	NPP		•	15.1%
3	NPP	•		18.2%
4	PL	•		19.6%

Table 6.8: Pairwise distribution of errors for the member classifiers

Classifiers	**Both correct**	**First correct**	**Second correct**	**Same incorrect**	**Different incorrect**
1 and 2	82.0%	3.0%	2.9%	9.7%	2.4%
1 and 3	78.5%	6.6%	3.3%	8.3%	3.3%
1 and 4	78.4%	6.7%	2.0%	10.8%	2.2%
2 and 3	79.3%	5.7%	2.5%	10.4%	2.2%
2 and 4	76.3%	8.7%	4.0%	8.1%	2.9%
3 and 4	77.0%	4.9%	3.4%	11.7%	3.1%

correct" and "Second correct" refer to situations where only the first or second classifier was correct and the other incorrect.

From Table 6.8 it is obvious that for all pair-wise combinations of the classifiers used the occurrence of the same error is much more common than the classifiers producing different errors. As when all the classifiers produce the same error the committee has a very limited chance of finding the correct answer, such a situation is highly unbeneficial for the committee operation.

6.5.3 Results

The results of the runs with the DEC committee for evaluating the performance of various options are shown in Table 6.9. These runs were performed using all combinations of the available options. Some averages of the effects of the different options on overall performance have been collected into Table 6.10. The tail error percentage in the tables corresponds to the error percentage calculated for the last 200 samples for each writer.

As can be seen from Tables 6.9 and 6.10, in general using the best individual classifier as the default rule outperformed majority voting in that role. This might be partially due to the relatively large difference in the accuracy between the best and worst classifier, with accuracies of 14.9% and 19.6%, respectively.

Table 6.9: Results from the DEC parameter variation experiments

Default decision	**Required inclusion**	**2nd-ranking results**	**Conflict resolving**	**Error percentage**	**Tail error percentage**
best	-	-	cor	14.8	15.1
best	-	-	c&w	15.1	15.4
best	-	-	ir	15.7	16.6
best	-	vertical	cor	11.1	11.3
best	-	vertical	c&w	11.1	11.3
best	-	vertical	ir	12.2	12.7
best	-	horizontal	cor	13.5	13.4
best	-	horizontal	c&w	13.5	13.4
best	-	horizontal	ir	15.6	16.4
best	yes	-	cor	12.7	13.4
best	yes	-	c&w	12.7	13.4
best	yes	-	ir	12.8	13.6
best	yes	vertical	cor	11.6	11.3
best	yes	vertical	c&w	11.6	11.3
best	yes	vertical	ir	12.3	12.4
best	yes	horizontal	cor	11.7	11.8
best	yes	horizontal	c&w	11.7	11.9
best	yes	horizontal	ir	11.9	12.4
majority	-	-	cor	15.1	15.4
majority	-	-	c&w	15.5	15.8
majority	-	-	ir	16.1	17.0
majority	-	vertical	cor	12.0	12.0
majority	-	vertical	c&w	12.0	12.0
majority	-	vertical	ir	13.0	13.3
majority	-	horizontal	cor	14.3	13.7
majority	-	horizontal	c&w	14.3	13.7
majority	-	horizontal	ir	16.6	17.2
majority	yes	-	cor	12.9	13.5
majority	yes	-	c&w	12.9	13.4
majority	yes	-	ir	12.9	13.5
majority	yes	vertical	cor	12.5	11.9
majority	yes	vertical	c&w	12.5	11.9
majority	yes	vertical	ir	13.0	12.9
majority	yes	horizontal	cor	12.5	12.3
majority	yes	horizontal	c&w	12.5	12.4
majority	yes	horizontal	ir	12.9	13.1

Table 6.10: Estimation of the effect of various individual options alone

Parameter	**With**	**Total error**	**Tail error**
best	vertical	11.7	11.7
	horizontal	13.0	13.2
	no 2nd	14.0	14.6
	inc	12.1	12.4
	no inc	13.6	14.0
	overall	12.8	13.2
majority	vertical	12.5	12.3
	horizontal	13.9	13.6
	no 2nd	14.2	14.8
	inc	12.7	12.8
	no inc	14.3	14.5
	overall	13.5	13.6
cor	best	12.6	12.7
	majority	13.2	13.1
	overall	12.9	12.9
c&w	best	12.6	12.8
	majority	13.3	13.2
	overall	13.0	13.0
ir	best	13.4	14.0
	majority	14.1	14.5
	overall	13.8	14.3
inc	best	12.1	12.4
	majority	12.7	12.8
	overall	12.4	12.6
no inc	best	13.6	14.0
	majority	14.3	14.5
	overall	14.0	14.2

Table 6.11: Comparison of DEC with reference classifiers

Combination method	**Error %**	**Tail error %**
DEC	11.1	11.3
MCBL	13.0	14.3
Adjusting Best	14.5	15.0
Non-adaptive Majority Voting	14.6	15.9
Adjusting Majority Voting	14.7	15.9

Requiring the output symbol to be included in the inputs seems to also improve accuracy somewhat. This option has the benefit of making absurd rules, such as for example "a a a $\rightarrow$ d", impossible. It can be expected to be beneficial in all cases but for those where the user truly writes a character in a way that results in the same incorrect recognition sequence every time, and the character the incorrect recognition results in causes a different sequence from the classifiers. As such errors are much easier to correct by modifying the prototype set, it would be expected that this inclusion requirement would be even more beneficial when using member classifiers that are also independently adaptive.

Also the use of the second-ranking results seems quite beneficial. In most cases the vertical ordering produces clearly better results than the horizontal. This would also be expected, as the vertical ordering should be the most likely to provide the most probably correct results with classifiers with recognition accuracies of over 50%. When that is the case, it is more likely for the first-ranking result of any classifier to be correct than any second-ranking result.

As for the method of resolving conflicting rules, the difference between using just the correct results (cor) and both correct and wrong results (c&w) for calculation of performance seems to have no significant effect. Using just the correct results is on the average slightly better, but the difference is not that significant. The strategy of inactivating rules after errors (ir), however, performs noticeably worse.

The results of the DEC committee runs are compared to the reference committees in Table 6.11. As can be seen, the DEC-based committee clearly outperforms all the reference classifiers used.

The adaption of the recognition error rate for an example writer is shown in Figure 6.2. The error rate has been calculated within a sliding window of 100 characters. The average error rate for the writer was 3.2%, but as can be seen the initial error rate is around 6-7%, and the final level is below 2%.

Another example of the evolution of the error rate can be seen in Figure 6.3. Also in

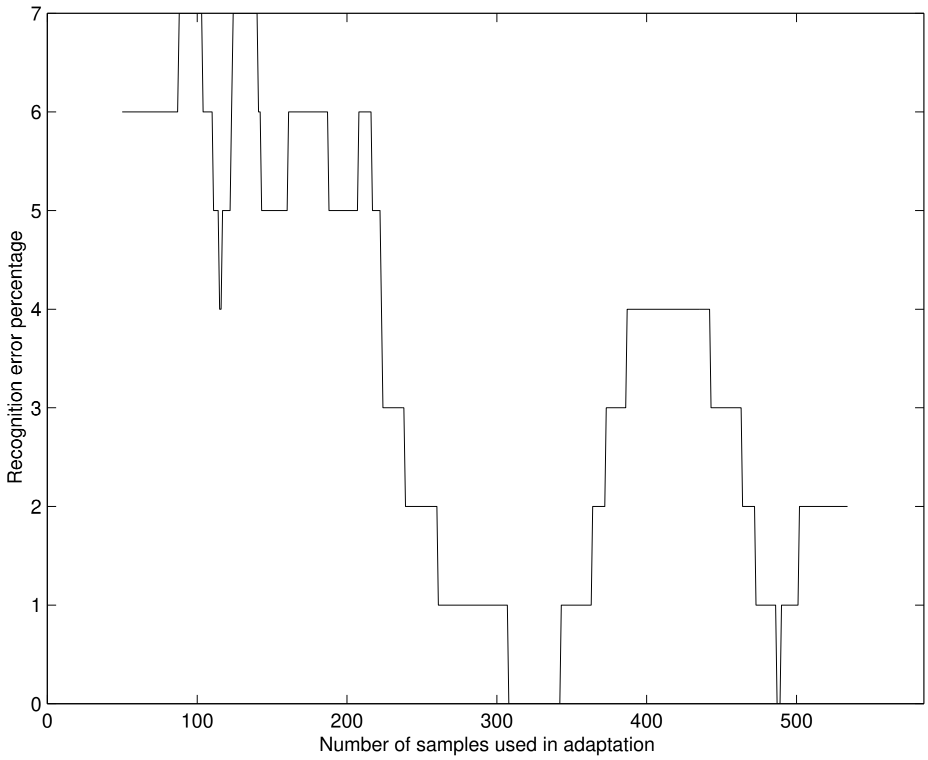


Figure 6.2: The evolution of the recognition error rate for one writer from the DEC-based committee

this case the error rate has been calculated within a sliding window of 100 characters. The average error rate for this writer was less impressive at 18.7%. The initial error rate was near 30%, but as can be seen, the final error rate improved considerably to end up under 15%. The result from the overall best individual classifier is also included and represented by the dotted line. As can be seen, the final error rate for the best non-adaptive member classifier was approximately 30%, which is twice as much as with the adaptive committee.

6.6 Future directions

The next logical stage in the experiments with committee classifiers will be the combining of the adaptive committee with adaptive member classifiers. This will pose some difficulties, as the adaptation in the individual classifiers makes the production of reliable rules for the DEC committee machine much more difficult.

The most noticeable difficulty is naturally that the classifier will not necessarily respond with the same classification result to an identical input after learning. Thus any rule created at an earlier stage may become worthless and possibly even hinder performance. A possible solution would be to keep track of the particular prototypes used to create a

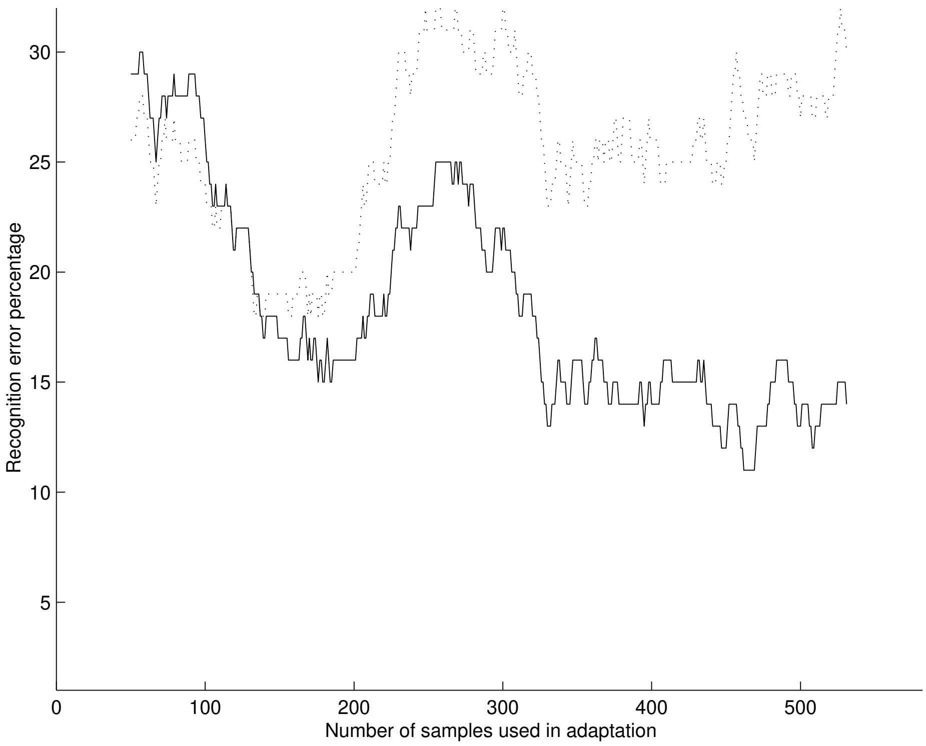


Figure 6.3: The evolution of the recognition error rate for another writer with the DEC-based committee. The result of the DEC committee is depicted with the solid line and the result from the best member classifier with the dotted line.

rule and simply delete the rule if the prototypes it is based on are altered or removed. As this could also result in rules incorporating a lower level of context being removed, the result would be a possibility for some inconsistency with the main DEC principle of minimal specificity to resolve conflicts. It might be possible to then generalize the higher-level rule, but as this approach is yet to be tested, nothing can be stated on the effectivity of such an approach.

Also the inclusion of the adapting best as the default decision rule for the DEC committee has been considered. This again poses slight difficulties as the adaptive best method inherently alters the ordering of the classifiers. Since the DEC rule context is formed based on the ordering, reordering of the classifiers should result in reordering of the contexts for all existing rules. This could naturally be done through storing the full context with each rule and just using the predesignated level of context. Discrepancy might also arise from reordering the context in the form of illogical rules arising, causing inconsistency especially when the inclusion option were in use.

Perhaps the simplest way to combine member classifier adaptation and committee adaptation would be to simply first adapt just the individual classifiers. Thus the actual creation of the adaptive rules would only be started when either a certain accuracy level has been reached or a predefined number of input samples have been processed. Thus the final level of accuracy of the classifiers could possibly be further enhanced by the addition of rules. The downside of this strategy would be a prolonged learning time, as the rule base can only be formed on classification errors.

Another interesting subject for further research would also be in loosening up the rules in situations where exact matches cannot be found. This might be implemented with some measure of nearness which could be applied in situations where a part of the context is different from an existing rule. If the context were similar enough, the rule could be used anyway. This might be able to produce more accurate results than resorting to the default decision, as is currently being done, to improve performance also in situations not yet encountered.

Chapter 7

Conclusions

Being an ongoing research problem, on-line handwriting recognition is still in need of enhancement to achieve a level of performance sufficient for commercial applications. Adaptation can be seen as the most suitable method of improving recognition performance especially in mobile solutions, where storage limitations hinder or even prohibit the use of a large dictionary, the other effective means of improving performance of algorithms. Also the general deployment of products to the international market makes the dictionary-based approaches even less practical, as the size of a dictionary to cover all languages the device might be used with easily becomes excessively large.

7.1 Palm computing

Pen-based computers in general are not a new concept but have become increasingly popular only recently, mainly with the success of the 3Com's Palm series of PDAs. The two main problems are the efficiency, or lack thereof, of handwritten character recognition components and the immaturity of the user interface technology.

But as the commercial sector gains knowledge and interest, or as the public demand for such applications increases, it is highly probable that more applications will emerge. Using natural handwriting is much more convenient for the user than having to learn a special alphabet in order to input writing successfully. As the computational power increases on even the smallest of platforms, more complex and better-performing algorithms should start to surface.

7.2 Handwriting recognition in literature

Handwritten character recognition has been an ongoing research topic for quite some time and has gained wide interest, as can be seen in the sheer number of publications on various aspects of the problem. A very large number of approaches to recognition have been undertaken, and only a few of them could be presented within the scope of the literature survey of this thesis.

There are clearly some basic differences in the recognition methods that enable a rough division into subclasses, as was done in Chapter 3. It seems that HMM and elastic-matching-based algorithms are perhaps the most popular, but also newer ideas, such as critic-driven and fuzzy classification, have gained noticeable interest. Adaption in the classifiers has also generally been seen as an interesting topic for research and has been widely studied.

7.3 True label deduction

An issue that must not be overlooked in implementing adaption in a real-life situation is how the true labels of the input samples are deduced. This is vital to the effectiveness of the adaptation, as otherwise the adaptation might easily result in the deterioration of the recognition performance instead of its improvement. As knowledge on the true labels cannot be received from the user explicitly, or the user would have to accept each character separately, the logic that reveals the labels is definitely something that cannot be afforded to be overlooked.

The approach presented in Section 5.5.2 is merely a starting point and its effectiveness will be seen in future experiments. It is probably impossible to determine the labels with 100% accuracy, as the user cannot be expected to always have the time or interest to correct all recognition mistakes. But due to the criticality of this stage, care must be taken in its implementation, or even the best of adaptation algorithms will eventually succumb to mislearning. Also feasible countermeasures for mislearning must be given consideration. Prototype inactivation is one method capable of dissipating the effects of incorrect learning, but also other methods should be investigated.

7.4 Future views for the palm-top implementation

A notable difficulty with the palm-top platform is the still too slow recognition performance. As the user has to wait for the characters to be recognized, the process of using the apparatus becomes quite frustrating. Clearly the already implemented speedup methodologies have helped, but the recognition speed simply must be improved on to make the device more usable. But due to the pace of hardware development, the effort put into devising and implementing speedup in software must be carefully considered, as it is merely a matter of time before the complexity causes no problems for palm-top computers available. Already the fastest of the devices covered in Section 2.1, the Casio Cassiopeia E-105, features a processor of running at 131 MHz, while the faster of our testing beds has the same processor architecture and a clock speed of 75 MHz. This increase in computational power alone may be enough to speed up recognition to an acceptable level without any further speed-related algorithm optimizations.

The palm-top implementation presented in this thesis is, mainly in terms of its interface, still a data collection and testing application. In the future the development of it to a more realistic application will be necessary in order to see how the performance and adaption fare when put into a true real-life test. This could be implemented by for example a simple note-taking application, or perhaps even as an additional operating system component that could intercept and forward all data written in the user interface and be used in a variety of tasks.

7.5 Adaptive committee recognition

The performance of the DEC-based committee classifier proved impressive if not astounding. The principle is effective, but still some modifications to further specify the committees actions to the case of handwritten character recognition may be necessary as constantly better performance is striven for.

The inclusion of adaptive member classifiers into the DEC committee will be a target for further research. If a method enabling concurrent adaptation for the committee and the individual classifiers can be found, the time to adaptation should be significantly reduced to produce a fast-adapting and well-performing complete system.

Bibliography

Annadurai, S. and Balasubramaniam, A. (1996). Classification of handwritten alphanumeric characters: a fuzzy neural approach, *International Conference on High Performance Computing*, pp. 36–41.

Arakawa, H. (1983). On-line recognition of handwritten characters – alphanumerics, hiragana, katakana, kanji, *Pattern Recognition* **16**(1): 9–16.

ART (2000). Smartwriter data sheet, http://www.artrecognition.com/artrecognition/download/smartwriter.pdf.

Bellegarda, E. J., Bellegarda, J. R., Nahamoo, D. and Nathan, K. S. (1994). A fast statistical mixture algorithm for on-line handwriting recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(12): 1227–1233.

Bellegarda, E. J., Bellegarda, J. R., Nahamoo, D. and Nathan, K. S. (1995). A discrete parameter HMM approach to on-line handwriting recognition, *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, IEEE, pp. 2631–2622.

Bouchaffra, D. and Govindaraju, V. (1999). A methodology for mapping scores to probabilities, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(9): 923–927.

Brakensiek, A., Kosmala, A. and Willett, D. (1999). Performance evaluation of a new hybrid modeling technique for handwriting recognition using identical on-line and off-line data, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 446–449.

Bray, J. (1999). Pocket computers, *PC Pro* pp. 76–96.

Cao, J., Shridhar, M., Kimura, F. and Ahmadi, M. (1992). Statistical and neural classification of handwritten numerals: a comparative study, *Proceedings of International Conference on Pattern Recognition Methodology and Systems*, pp. 643–646.

Chan, K.-F. and Yeung, D.-Y. (1998). Elastic structural matching for online handwritten alphanumeric character recognition, *Proceedings of International Conference on Pattern Recognition*, Vol. 2, pp. 1508–1511.

Chang, L. and MacKenzie, I. S. (1994). A comparison of two handwriting recognizers for pen-based computers, *Proceedings of Center for Advanced Studies Conference*, pp. 364–371.

Cheung, K., Yeung, D. and Chin, R. (1998). A bayesian framework for deformable pattern recognition with application to handwritten character recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(12): 1382–1388.

CIC (2000a). Jot pro 1.01 for palm-size pc, http://www.cic.com/software_store/product_details/jotpro_ppc.asp.

CIC (2000b). Jot pro user's guide/frequently asked questions, http://www.cic.com/support_center/Faq/CE/PPCJot/PPCJotUsersGuide.html.

Connell, S. and Jain, N. (1999). Writer adaptation of online handwriting models, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 434–437.

Drucker, H., Cortes, C., Jackel, L. D., LeCun, Y. and Vapnik, V. (1994). Boosting and other ensemble methods, *Neural Computation* **6**(6): 1289–1301.

Drucker, H., Schapire, R. and Simard, P. (1993). Boosting performance in neural networks, *International Journal of Pattern Recognition and Artificial Intelligence* **7**(4): 705–719.

Everex (1998). *Everex Freestyle Palm-size PC User's Manual*, Everex Systems Inc.

Everex (2000). Everex everywhere - tech support & customer service: F.a.q.: Freestyle[tm], http://www.everex.com/support/faq/freestyle_faq/.

Filatov, A., Gitis, A. and Kil, I. (1995). Graph-based handwritten digit string recognition, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 845–848.

Franke, J. and Mandler, E. (1992). A comparison of two approaches for combining the votes of cooperating classifiers, *Proceedings of the 11th International Conference on Pattern Recognition*, Vol. II, IAPR, Hague, pp. 611–614.

Frankish, C., Hull, R. and Morgan, P. (1995). Recognition accuracy and user acceptance of pen interfaces, *Proceedings ACM CHI'95 Conference on Human Factors in Computing Systems*.

Frosini, G., Lazzerini, B., Maggiore, A. and Marcelloni, F. (1998). A fuzzy classification based system for handwritten character recognition, *International Conference on Knowledge-Based Intelligent Electronic Systems*, pp. 61–65.

Goldberg, D. and Richardson, C. (1993). Touch-typing with a stylus, *Proceedings of International Conference on Human Factors in Computing Systems*, pp. 80–87.

Govindan, V. K. (1990). Character recognition – a review, *Pattern Recognition* **23**(7): 671–683.

Guberman, S. (1998). Off-line and online handwriting recognition-common approach, *IEE European Workshop on Handwriting Analysis and Recognition*, pp. 6/1–6/2.

Guerfali, W. and Plamondon, R. (1993). Normalizing and restoring on-line handwriting, *Pattern Recognition* **26**(3): 419–430.

Guyon, I. and Warwick, C. (1996). Joint EC-US survey of the state-of-the-art in human language technology, http://www.cse.ogi.edu/CSLU/HLTsurvey.htm.

Guyon, I., Schomaker, L., Plamondon, R., Liberman, M. and Janet, S. (1994). Unipen project of on-line data exchange and recognizer benchmark, *Proceedings of International Conference on Pattern Recognition*, pp. 29–33.

Hu, J., Brown, M. K. and Turin, W. (1996). HMM based on-line handwriting recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18**(10): 1039–1045.

Huang, Y. and Suen, C. (1995). A method of combining multiple experts for the recognition of unconstrained handwritten numerals, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(1): 90–94.

IBM (1991). Method that eliminates maverick prototypes in online handwriting recognition, *IBM Technical Disclosure Bulletin* **33**(11): 383–384.

Kang, H.-J. and Lee, S.-W. (1999). Combining classifiers based on minimization of a bayes error rate, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 398–401.

Khan, N. and Hegt, H. (1998). A flexible and robust matching scheme for character recognition to cope with variations in spatial interrelation among structural features, *International Conference on Systems, Man and Cybernetics*, Vol. 5, IEEE, pp. 4166–4171.

Khotanzad, A. and Chung, C. (1994). Hand written digit recognition using bks combination of neural network classifiers, *Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation*, pp. 94–99.

Kittler, J., Hatef, M., Duin, R. and Matas, J. (1998). On combining classifiers, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(3): 226–239.

Kohonen, T. (1986). Dynamically expanding context, with applications to the correction of symbol strings in the recognition of continuous speech, *International Conference on Pattern Recognition*, Vol. 2, pp. 1148–1151.

Kohonen, T. (1987). Dynamically expanding context, *Journal of Intelligent Systems* **1**(1): 79–95.

Kohonen, T. (1997). *Self-Organizing Maps*, Vol. 30 of *Springer Series in Information Sciences*, Springer-Verlag. Second Extended Edition.

Kuklinski, T. T. (1984). Components of handprint style variability, *Proceedings of the 7th International Conference on Pattern Recognition*, pp. 924–926.

Kurimo, M. (1998). Improving vocabulary independent hmm decoding results by using the dynamically expanding, *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Vol. 2, pp. 883–836.

Kurtzberg, J. M. and Tappert, C. C. (1981). Symbol recognition system by elastic matching, *IBM Technical Disclosure Bulletin* **24**(6): 2897–2902.

Laaksonen, J., Aksela, M., Oja, E. and Kangas, J. (1999). Dynamically Expanding Context as committee adaptation method in on-line recognition of handwritten latin characters, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 796–799.

Laaksonen, J., Hurri, J., Oja, E. and Kangas, J. (1998a). Comparison of adaptive strategies for on-line character recognition, *Proceedings of International Conference on Artificial Neural Networks*, pp. 245–250.

Laaksonen, J., Hurri, J., Oja, E. and Kangas, J. (1998b). Experiments with a self-supervised adaptive classification strategy in on-line recognition of isolated handwritten latin characters, *Proceedings of Sixth International Workshop on Frontiers in Handwriting Recognition*, pp. 475–484.

Laird, D. (2000). Crusoe processor products and technology, http://www.transmeta.com/crusoe/download/pdf/laird.pdf.

Lam, L. and Suen, C. Y. (1994). A theoretical analysis of the application of majority voting to pattern recognition, *Proceedings of 12th International Conference on Pattern Recognition*, Vol. II, IAPR, Jerusalem, pp. 418–420.

Lazzerini, B. and Marcelloni, F. (1999). Fuzzy classification of handwritten characters, *International Conference on North American Fuzzy Information*, pp. 566–570.

Liu, C.-L. and Nakagawa, M. (1999). Prototype learning algorithms for nearest neighbor classifier with application to handwritten character recognition, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 378–381.

Loy, W. and Landay, L. (1982). An on-line procedure for recognition of handprinted alphanumeric characters, *Pattern Recognition* **4**(4): 422–427.

Lu, P.-Y. and Brodersen, R. W. (1984). Real-time on-line symbol recognition using a DTW processor, *Proceedings of International Conference on Pattern Recognition*, Vol. 2, pp. 1281–1283.

MacKenzie, I. S., Nonnecke, B., Riddersma, S., McQueen, C. and Meltz, M. (1994). Alphanumeric entry on pen-based computers, *International Journal of Human-Computer Studies* **41**: 775–792.

Mandler, E., Oed, R. and Doster, W. (1985). Experiments in on-line script recognition, *Proceedings of Scandinavian Conference on Image Analysis*, pp. 75–86.

Mel, M. W., Omohundro, S. M., Robinson, A. D., Skiena, S. S., Thearling, K. H., Young, L. T. and Wolfram, S. (1988). Tablet: personal computer in the year 2000, *Communications of the ACM* **31**(6): 639–648.

Meyer, A. (1995). Pen computing, a technology overview and a vision, *ACM SIGCHI bulletin*.

Miller, D. and Yan, L. (1999a). Ensemble classification by critic-driven combining, *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Vol. 2, IEEE, pp. 1029–1032.

Miller, D. and Yan, L. (1999b). Some analytical results on critic-driven ensemble classification, *Proceedings of Neural Networks for Signal Processing*, pp. 252–263.

Mozayyani, N., Baig, A. and Vaucher, G. (1998). A fully-neural solution for on-line handwritten character recognition, *Proceedings of International Joint Conference on Neural Networks*, Vol. 2, pp. 160–164.

Nadal, C. (1998). Cenparmi home page, http://www.cenparmi.concordia.ca/index.html.

Narayanaswamy, S., Hu, J. and Kashi, R. (1999). User interface for a pcs smart phone, *International Conference on Multimedia Computing and Systems*, Vol. 1, IEEE, pp. 777–781.

Nathan, K., Bellegarda, J. R., Nahamoo, D. and Bellegarda, E. J. (1993). On-line handwriting recognition using continuous parameter hidden markov models, *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Vol. 5, pp. 121–124.

Neisser, U. and Weene, P. (1960). A note on human recognition of hand-printed characters, *Information and Control* (3): 191–196.

Nouboud, F. and Plamondon, R. (1990). On-line recognition of handprinted characters: survey and beta tests, *Pattern Recognition* **23**(9): 1031–1044.

Nouboud, F. and Plamondon, R. (1991). A structural approach to on-line character recognition: System design and applications, *International Journal of Pattern Recognition and Artificial Intelligence* **5**(1&2): 311–335.

Paik, J., bae Cho, S., Lee, K. and Lee, Y. (1996). Multiple recognizers system using two-stage combination, *Proceedings of International Conference on Pattern Recognition*, IEEE, pp. 581–585.

Park, K.-Y. and Lee, S.-Y. (1999). Selective attention for robust speech recognition in noisy environments, *Proceedings of International Joint Conference on Neural Networks 1999*, Vol. 5, pp. 3014–3019.

Philips (1998). *Nino 300 Owner's Guide*, Philips Electornics North America Corporation.

Qian, G. (1999). An engine for cursive handwriting interpretation, *Proceedings of International Conference on Tools with Artificial Intelligence*, pp. 271–278.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition, *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp. 267–295.

Rabinowitz, S. and Wagon, S. (1995). A spigot algorithm for the digits of π, *American Mathematical Monthly* **102**: 195–203.

Rahman, A. and Fairhurst, M. (1996). Recognition of handwritten characters with a multi-expert system, *IEE Workshop on Handwriting Analysis and Recognition - A European Perspective*, pp. 6/1–6/4.

Rahman, A. and Fairhurst, M. (1997a). A comparative study of decision combination strategies for a novel multiple-expert classifier, *International Conference on Image Processing and Its Applications*, Vol. 1, pp. 131–135.

Rahman, A. and Fairhurst, M. (1997b). Generalised approach to the recognition of structurally similar handwritten characters using multiple expert classifiers, *IEE Proceedings on Vision, Image and Signal Processing* **144**(1): 15–22.

Rahman, A. and Fairhurst, M. (1997c). Introducing new multiple expert decision combination topologies: a case study using recognition of handwritten characters, *Proceedings of International Conference on Document Analysis and Recognition*, Vol. 2, IEEE, pp. 886–891.

Råde, L. and Westergren, B. (1990). *BETA Mathematics Handbook for Science and Engineering, 2nd edition*, Chartwell-Bratt Ltd.

Rigoll, G., Kosmala, A., Rottland, J. and Neukirchen, C. (1996). A comparison between continuous and discrete density hidden markov models for cursive handwriting recognition, *Proceedings of International Conference on Pattern Recognition*, Vol. 2, IEEE, pp. 205–209.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, Prentice Hall.

Sankoff, D. and Kruskal, J. B. (1983). *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*, Addison-Wesley.

Schalkoff, R. (1992). *Pattern recognition: statistical, structural and neural approaches*, John Wiley & Sons, Inc.

Schomaker, L. (1994). User-interace aspects in recognizing connected-cursive handwriting, *IEE Colloquium on Handwriting and Pen-based input.*

Schomaker, L. (1998). From handwriting analysis to pen-computer applications, *Electrics & Communication Engineering Journal* pp. 93–101.

Schomaker, L., Abbink, G. and Selen, S. (1994). Writer and writing-style classification in the recognition of online handwriting, *IEE Colloquium (Digest) Proceedings of the European Workshop on Handwriting Analysis and Recognition: European Perspective.*

Sherkat, N., Whitrow, R. and Evans, R. (1999). Wholistic recognition of handwriting using structural features, *IEE Colloquium on Document Image Processing and Multimedia*, pp. 12/1–12/4.

Srihari, R. K. (1997). Cedar on-line handwriting database, http://www.cedar.buffalo.edu/Linguistics/database.html.

Srikantan, G., Lee, D.-S. and Favata, J. (1995). Comparison of normalization methods for character recognition, *Proceedings of International Conference on Document Analysis and Recognition*, Vol. 2, pp. 719–722.

Srinivasan, S. and Ramakrishnan, K. (1999). The independent components of characters are 'strokes', *Proceedings of International Conference on Document Analysis and Recognition*, pp. 414–417.

Starner, T., J. Makhoul, R. S. and Chou, G. (1994). On-line cursive handwriting recognition using speech recognition methods, *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Vol. 5, IEEE, pp. 125–128.

Suen, C., Nadal, C., Legault, R., Mai, T. and Lam, L. (1992). Computer recognition of unconstrained handwritten numerals, *Proceedings of the IEEE*, Vol. 7, IEEE, pp. 1162–1180.

Tan, A.-H. and Teow, L.-N. (1995). Learning by supervised clustering and matching, *Proceedings of International Conference on Neural Networks*, Vol. 1, pp. 242–246.

Tanaka, H., Nakajima, K., Ishigaki, K., Akiyama, K. and Nakagawa, M. (1999). Hybrid pen-input character recognition system based on integration of online-offline recognition, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 209–212.

Tappert, C. C. (1984). Adaptive on-line handwriting recognition, *Proceedings of International Conference on Pattern Recognition*, IEEE, pp. 1004–1007.

Tappert, C., Suen, C. Y. and Wakahara, T. (1990). The state of the art in on-line handwriting recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(8): 787–808.

Teow, L.-N. and Tan, A.-H. (1995). Adaptive integration of multiple experts, *Proceedings of International Conference on Neural Networks*, Vol. 3, pp. 1215–1220.

Torkkola, K. (1993). An efficient way to learn english grapheme-to-phoneme rules automatically, *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Vol. 2, pp. 199–202.

Torkkola, K. and Kohonen, T. (1988). Correction of quasiphoneme strings by the dynamically expanding context, *International Conference on Pattern Recognition*, Vol. 1, pp. 487–489.

Vuori, V. (1999). *Adaptation in on-line recognition of handwriting*, Master's thesis, Helsinki University of Technology.

Vuori, V., Laaksonen, J., Oja, E. and Kangas, J. (1999). On-line adaptation in recognition of handwritten alphanumeric characters, *Proceedings of International Conference on Document Analysis and Recognition*, pp. 792–795.

Wang, P. S.-P. and Gupta, A. (1991). An improved structural approach for automated recognition of handprinted character, *International Journal of Pattern Recognition and Artificial Intelligence* **5**(1&2): 97–121.

Ward, J. R. and Blesser, B. (1985). Interactive recognition of handprinted characters for computer input, *IEEE Computer Graphics and Applications* **9**(5): 24–37.

Ward, J. R. and Kuklinski, T. (1988). A model for variability effects in handprinting with implications for design of handwriting character recognition systems, *IEEE Transactions on Systems, Man, and Cybernetics* **18**(3): 438–451.

Xu, L. and Jordan, M. I. (1993). EM learning on a generalized finite mixture model for combining multiple classifiers, *Proceedings of the World Congress on Neural Networks*, Vol. IV, pp. 227–230.

Xu, L., Jordan, M. I. and Hinton, G. E. (1995). An alternative model for mixtures of experts, *in* G. Tesauro, D. S. Touretzky and T. K. Leen (eds), *Advances in Neural Information Processing Systems 7*, MIT Press, Cambridge, MA, pp. 633–640.

Xu, L., Krzyzak, A. and Suen, C. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition, *IEEE Transactions on Systems, Man and Cybernetics* **22**(3): 418–435.

Yuen, H. (1996). A chain coding approach for real-time recognition of on-line handwritten characters, *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Vol. 6, IEEE, pp. 3426–3429.

Zhang, B., Fu, M., Yan, H. and Jabri, M. (1999). Handwritten digit recognition by adaptive-subspace self-organizing map (assom), *IEEE Transactions on Neural Networks* **10**(4): 939–945.