

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI[®]
800-521-0600

UNCONSTRAINED HANDWRITTEN NUMERAL
RECOGNITION: A CONTRIBUTION TOWARDS
MATCHING HUMAN PERFORMANCE

RAYMOND LEGAULT

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY AT
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

SEPTEMBER 1997
© RAYMOND LEGAULT, 1997



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

Our file *Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-39794-7

Canada

Abstract

Unconstrained Handwritten Numeral Recognition: A Contribution Towards Matching Human Performance

Raymond Legault, Ph.D.
Concordia University, 1997

Intense activity and significant progress have characterized the last decade in the field of the recognition of unconstrained handwritten numerals by computer. The diversity and ingenuity of the methods proposed are carefully reviewed at the beginning of this thesis and the results achieved are compared.

Despite important advances, the very high reliability of human recognition has not been matched by these approaches and our work is intended as a contribution towards bridging this reliability gap. In recent years, the combination of several recognition methods has been a very fruitful idea in this regard. Here we explore another avenue: overcoming the limits of single methods, particularly structural model-based methods, to deliver much more reliable classification on their own. Lessons are drawn from past work and all stages of the recognition process which are typical of this approach (preprocessing, feature extraction, and classification) are revisited.

Achieving higher levels of reliability and robustness in the feature extraction stage was seen as key to achieving our goal. Hence much of our research was devoted to the solution of this problem including a comparative study of several curvature feature extraction schemes, the detailed ‘autopsy’ of a feature extractor previously devised by this author, and the meticulous construction of a new extractor to circumvent identified weaknesses. Compared to a preceding effort at creating a numeral recognition system, our conception of the development of classification rules was also deeply revised. Much care is taken to distinguish all the (global or local) shape variants to be identified by the system and to tightly model each of these variants in a more

refined and exhaustive manner. A special syntax and a development interface were designed to assist in this task.

Results for the CENPARMI database, from a partial classifier built upon these foundations, demonstrate the feasibility of creating a single-method numeral recognition system with a high recognition rate (around 90%) and a *very low* substitution rate. When applied to other databases, including some which incorporate markedly different writing styles, the very high reliability of the system is maintained.

Acknowledgements

I first enrolled into the doctoral program at Concordia University eight years ago. During the last four years, I have been working full-time outside of the university environment. Completing this work was not always straightforward and easy...

I would first like to thank my supervisor, Professor Ching Y. Suen for his guidance and encouragement throughout, for his financial assistance during the first years of the research, and for his careful review of the first version of this thesis.

During these years, I have had the opportunity to carry out joint research with a few CENPARMI colleagues. In particular, I would like to acknowledge the contributions of Christine Nadal and Louisa Lam, with whom I co-authored articles and had several interesting exchanges. I am also indebted to Louisa for a very thorough review of an earlier version of Chapter 5 and thoughtful suggestions. Several researchers kindly assisted in my investigation of curvature feature extractors and corner detectors for parts of Chapter 6. My thanks go to A. Commike and D.-S. Lee for their prompt and precise replies to my inquiries and more particularly to Louisa Lam for sharing her code (and experience with) the Beus-Tiu method, and to L. O’Gorman and A. Held for providing the full code of their method and letting me experiment with it.

Years of research and software development on computers necessarily carry along a number of technical problems. I could always count on the assistance of William Wong and Mike Yu to help with such difficulties. Thank you.

The initial years of my research were financially supported by fellowships from NSERC, FCAR, and Concordia University. This has allowed me to fully concentrate on my studies and research during this period.

A research laboratory is not only a place for work but also, to some extent, a living environment. I was glad to share lunch hours, and other moments of rest and casual exchange with several students, staff members and visitors, in particular with Christine, William, Myriam, Jurgen, and Simone.

Lastly, but most importantly, I would like to express my gratitude to my family and my close friends who provided crucial encouragement and support, especially during the last year, when it was not always clear to me that I had the motivation and stamina to bring this project to completion. In this respect, and several others, I owe them a lot and thank them for being there.

Contents

List of Tables	xiii
List of Figures	xv
1 Presentation	1
1.1 OCR: A Bit of History	1
1.2 Focus of Our Work	2
1.3 Outline of Thesis	4
2 State of the Art	6
2.1 More Sophisticated and Diversified Methods	7
2.1.1 Structural Methods	7
2.1.2 Artificial Neural Networks	10
2.1.3 Statistical Methods	15
2.2 Combination of Recognition Methods	20
2.2.1 Multistage Classification Methods	20
2.2.2 Multi-Expert Classification Methods	23
2.3 Comparative Results	34
2.3.1 Some Guidelines for Comparison	34
2.3.2 Some Published Results on CENPARMI Database	36
2.3.3 Some Published Results on CEDAR Database	38
2.3.4 The NIST Competitions	40
2.3.5 The IPTP Competitions	43
2.4 Machine vs Human Performance	44

2.4.1	About the Conclusions of NIST Conferences	45
2.4.2	Human Recognition is More Reliable	47
3	Overview of our Research	57
3.1	Limitations of the Expert E4	59
3.1.1	Lessons Learned on the ITRI Database	60
3.2	General Avenues	63
3.3	Design of the Recognition System	65
4	Preprocessing	69
4.1	Edge Extraction	70
4.2	Contour Extraction	73
4.3	Image Segmentation	74
4.3.1	Edge Chaining and Object Labeling	75
4.3.2	Finding Nesting Relationships Between Objects	77
4.4	Useful Global Features at Minimal Cost	79
4.4.1	Stroke Width Estimate From Edge Extraction	79
4.4.2	Computing Areas During Edge Chaining	80
4.4.3	Computing Minimum Bounding Boxes During Edge Chaining	82
4.5	Correction of Several Image Defects	83
4.5.1	Edge Smoothing	84
4.5.2	Hole and Cavity Opening	88
4.5.3	Extra Filling on First and Last Rows	90
4.5.4	Repairing Faulty Scanlines	90
4.5.5	Removing Isolated or Near-Isolated Black Runs	93
4.5.6	Trimming Protruding Black Runs	94
4.5.7	Removing Edges of Length 1	95
4.5.8	Removing Vertical Two-Pixel Stems	96
4.5.9	Frequency of Defects	97
5	Contour Smoothing	98
5.1	Contour Smoothing: A Brief Overview	100

5.1.1	Variety of Approaches and Related Problems	101
5.1.2	Our Own Work	106
5.2	Local Weighted Averaging	107
5.2.1	Geometric Interpretation	108
5.3	Optimum Results from a Simple Digitization Noise Model	109
5.3.1	Best Parameters to Minimize d'_{rms}	112
5.3.2	Best Parameters to Minimize m'_{rms}	113
5.3.3	Best Parameters to Minimize Deviation Angles	117
5.4	Verifying Results for Varying Curvature	121
5.4.1	Minimizing Error on Distances to Center	122
5.4.2	Minimizing Error on Tangent Directions	126
5.4.3	Minimizing Error on Deviation Angles	129
5.5	Conclusion	133
6	Curvature Feature Extraction	135
6.1	Feature Extraction in E4 System	136
6.1.1	Initial Processing of Holes	136
6.1.2	Structural Features From Outer Contours	137
6.2	Comparative Study	143
6.2.1	Problem Definition	143
6.2.2	Corners, Dominant Points and Curvature Features	144
6.2.3	Curvature Features in Recent OCR Literature	150
6.2.4	Experiment and Results	151
6.3	Detailed Autopsy of E4 Feature Extractor	163
6.3.1	Data Used for the Investigation	164
6.3.2	Problems with Endpoint Extraction	165
6.3.3	Problems with Extraction of Cavities and Bends	169
6.4	The New Feature Extractor	173
6.4.1	The Computation of Deviation Angles	175
6.4.2	The Extraction of Arc Regions	177
6.4.3	Finding the Focal Point of an Arc	182

6.4.4	From Arcs to More Global Features	182
6.4.5	From Arc to Endpoint (Function CHECK_END)	185
6.4.6	Endpoint Width Criterion	187
6.4.7	Conditional Endpoint Testing	188
6.4.8	Testing for End-Bend Composite Arc	188
6.4.9	Merging Consecutive Features	190
6.4.10	Computing Feature Directions	194
6.4.11	Assessment of New Feature Extractor	197
7	Development of Classification Rules	208
7.1	Steps Involved in Training for Each Class	209
7.2	Syntax for Classification Rules	211
7.2.1	Operator-Rules	211
7.2.2	Other Type of Rules	211
7.2.3	A Few Examples	212
7.2.4	Syntax, Classification, and Rule Generation	213
7.3	Tool for Rule Development	215
7.4	Overview of Classifier Development	218
7.5	Illustration of Approach for '2's	220
7.5.1	Assignment of Starting Feature	220
7.5.2	Clustering: Creation of Model Files	222
7.5.3	Rule Generation: Creation of Rule Files	230
7.5.4	Results With CENPARMI Training Samples	238
7.5.5	Relaxing Rules With CEDAR Training Data	239
8	Overall System & Results	244
8.1	Processing Multi-Component Samples	245
8.1.1	Deleting Tiny Pieces	245
8.1.2	Extracting New Endpoints from Small Pieces	246
8.1.3	Reconnecting the Pieces Together	247
8.1.4	Evaluation	251
8.2	Filtering Spurious Holes	255

8.3	First Recognition Pass	256
8.4	Filtering the Feature List	259
8.4.1	Filtering Wiggle-Pairs of Features	259
8.4.2	Filtering Small Convex/Concave Features in Large Concave/Convex Environment	259
8.4.3	Filtering Less Significant Isolated Features	262
8.5	Dropping Smallest Hole	262
8.6	Aiming for Higher Reliability	263
8.7	Results	267
9	Conclusion	273
9.1	Original Contributions	274
9.2	Future Work	276
	References	279
A	About Some Numeral Databases	298
A.1	CEDAR Database	298
A.2	CENPARMI Database	298
A.3	Concordia-Montreal Database	299
A.4	ITRI-Taiwan Databases	299
B	Derivation of ϕ_{rms} For Noisy Horizontal Border	300
C	Best Parameters to Minimize d'_{rms}	302
D	Minimizing m'_{rms}	304
E	Obtaining (dx'_i, dy'_i) Recursively	306
F	Syntax For Classification Rules	308
F.1	<u>B</u> ounding Box Rules	308
F.2	<u>C</u> omparison Rules	309
F.3	<u>F</u> eature Rules	310

F.4	<u>G</u> lobal Feature Rules	313
F.5	<u>H</u> ole Rules	314
F.6	<u>I</u> ndex Rules	314
F.7	<u>M</u> arking Rules	315
F.8	<u>O</u> r Rules	316
F.9	<u>P</u> iece Rules	316
F.10	<u>S</u> torage Modify Rules	317
G	Interface for Rule Generation	318
G.1	File Handling	320
G.2	The Option Menus	320
G.2.1	The VIEW Menu	320
G.2.2	The 1ST FEATURE Menu	322
G.2.3	The CLUSTERING Menu	325
G.2.4	The STATISTICS Menu	325
G.3	The Drawing Area	327
H	Holes in CENPARMI data	328
I	General Implementation Information	330
J	Number of Pieces in Numerals	333

List of Tables

1	Results of Individual Systems on CENPARMI Data	37
2	Results of Combinations of Systems on CENPARMI Data	38
3	Some Published Results on CEDAR Data	39
4	Top Ten Results for Digits at First NIST Conference	41
5	Machine and Human Performance on 360 Most Confusing Samples . .	49
6	Performance Results for Subsets of Original 360 Database	50
7	Required Rejection Levels for Very Low Error Rates	52
8	Edge Chaining and Object Segmentation	75
9	Edge Information for Above Figures	78
10	Percentages of Samples Affected by Preprocessing Operations	97
11	Best Parameters to Minimize $(y'_{i+1} - y'_i)^2$ and Fraction of Noise Removed	115
12	Best Parameters to Minimize $\left(\frac{y'_{i+1} - y'_{i-1}}{2}\right)^2$ and Fraction of Noise Removed	116
13	Mean Noise Reduction for $10 \leq R \leq 99$	126
14	Mean Noise Reduction for $4 \leq R \leq 99$	129
15	Mean Noise Reduction for $4 \leq R \leq 99$	132
16	Average Measures of Goodness for the Methods Tested	156
17	Average Measures of Goodness for Endpoint Regions Only	156
18	Average Measures of Goodness for the New Feature Extractor	201
19	Comparing Overall Parameters For Best Five Methods	202
20	Sample Counts Comparing Regions Detected by Methods vs Humans	203
21	Relative Weaknesses of Best Five Methods	205
22	Examples of Syntax Rules	212
23	Format of a Model File	216

24	Performance Matrix of E4 System	219
25	Models Based on Number and Position of Holes	223
26	Models for Bottom-Right Portion of '2's with a Bottom Hole	225
27	Models for Bottom-Right Portion of '2's Without a Bottom Hole	225
28	Models for Top Portion of '2's	228
29	Partial Performance Matrix on CENPARMI Sets A, B, T	239
30	Recognition of '2's Depending on Smoothing	241
31	Recognized Samples After Relaxing With cedar2-2.rlc	243
32	Partial Performance Matrix on Combined CENPARMI Sets A and B	267
33	Partial Performance Matrix on CENPARMI Test Set T	267
34	Partial Performance Matrix on CEDAR goodbs Test Set	268
35	Partial Performance Matrix on CEDAR bs Test Set	269
36	Partial Performance Matrix on Concordia-Montreal <i>f</i> -Set	270
37	Partial Performance Matrix on ITRI-Taiwan tw5-tw7 Sets	270
38	Partial Performance Matrix on ITRI-Taiwan t25-t29 Sets	271
39	Possible Values of ϕ_i^2 for Unsmoothed Border	301
40	Sample counts depending on number of holes	328
41	Program files in system	330
42	Numbers of rule files and rules	332
43	CEDAR database: Statistics on number of pieces	333
44	Concordia-Montreal database: Statistics on number of pieces	334
45	Taiwan tw1-tw4 database: Statistics on number of pieces	334
46	Taiwan t20-t24 database: Statistics on number of pieces	335

List of Figures

1	Difficult Samples from ITRI Database	61
2	Substitution Created by Partial Contour Information	64
3	Design of Recognition System	67
4	Edge Extraction From Binary Image	70
5	Sibling Objects With Parent Edge Connection	78
6	Computation of Total Enclosed Areas	82
7	Random Extraneous Cavities	84
8	Single-Pixel Deletion and Filling Masks	85
9	Edge Smoothing for '0' Displayed Above	86
10	Hole Split Into Many Holes by 8-Connectivity	88
11	Opening Narrow Hole or Downward Cavity	89
12	Spurious Hole Created by 8-Connectivity	89
13	Extra Filling on First and Last Rows	91
14	Rebuilding of Missing or Faulty Scanlines	92
15	Removing Isolated or Near-Isolated Black Runs	93
16	Trimming Protruding Black Runs	94
17	Removing Edges of Length 1	95
18	Removing Vertical 2-Pixel Stems	96
19	Contour Smoothing With Triangular Filter.	99
20	Freeman Chain Code.	101
21	Deviation Angle at p_i	101
22	Geometric Interpretation for $w = 3$	108
23	Noisy Horizontal Border	110

24	Best α to Minimize Deviation Angles for $w = 3$	118
25	Fraction of ϕ_{rms} Removed for $w = 3$	120
26	Minimizing $\overline{\phi_i'^2}$ for $w = 5$	121
27	Digital Circle of Radius $R = 7$	122
28	Best Smoothing to Minimize $\overline{(R - d_i')^2}$	124
29	Fraction of RMS Noise Removed: (a) $w = 3$; (b) $w = 5$	125
30	Best Smoothing to Minimize $\overline{(\varphi_i' - \theta_i')^2}$	127
31	Fraction of RMS Noise Removed: (a) $w = 3$; (b) $w = 5$	128
32	Best Smoothing to Minimize $\overline{(\delta_i' - \phi_i')^2}$	130
33	Fraction of RMS Noise Removed: (a) $w = 3$; (b) $w = 5$	131
34	RMS Noise Removed for $w = 5$: best case (solid); $\frac{2}{9}, \frac{1}{9}$ (dotted) . . .	132
35	The Preprocessing of Multiple Holes	137
36	Feature Regions Extracted by E4 system	140
37	Features of a Numeral '6'	141
38	Global Shape Features of a '3'	144
39	Illustration of DOS Methods	145
40	Numerals of Varying Styles and Sizes	154
41	Other Examples of Feature Regions Selected Manually	155
42	Weaknesses in Beus-Tiu (BT) Method	157
43	Weaknesses in D'Amato (DA) Method	158
44	Weaknesses in Legault-Suen (LS) Method	159
45	Weaknesses in Rosenfeld-Weszka (RW) Method	160
46	Idealized End-Regions	166
47	Defects in "E4" End-Region Extraction	166
48	More Defects in "E4" End-region Extraction	167
49	Large Curvature Regions Left Undetected	169
50	Missed or Fragmented Feature Regions	170
51	Merging Consecutive Bends or Cavities	171
52	Problematic Small Features	172
53	New Bend and Cavity Captured by GET_ARC_FROM_INTER_ARC	179
54	New Cavity Captured by SCRUTINIZE_INTER_ARC	182

55	Processing Composite Arcs	190
56	Cavities Not Merged With Neighbouring Cavities	192
57	Bends Not Merged With Neighbouring Bends	193
58	Four Different Direction Measurements	195
59	Samples Which Had End-Region Defects	198
60	Large Curvature Regions Previously Undetected	199
61	Previously Missed or Fragmented Feature Regions	200
62	Leftover Shortcomings of New Feature Extractor	207
63	Database Inspection and Rule Development Interface	217
64	Starting Feature and Discarded Samples for Class '2'	221
65	Problems with Starting Feature Definition	222
66	Example of Numerals for Model Files Based on Holes	224
67	Some Bottom-Right Feature Sequences for '2's With Bottom Holes	226
68	Some Bottom-Right Feature Sequences for '2's Without Bottom Holes	227
69	Some Less Common Top Feature Sequences for '2's	228
70	Samples of '2' With Unusual or Peculiar Shapes	229
71	Samples With Tiny Holes or Spurious Features	229
72	Samples Left Out of Rule Generation Process	231
73	Defining Safe Boundaries for Recognition	232
74	Classifier Output for a Specific Sample	234
75	Some of the Distances Involved in Rules	236
76	A '7' Misclassified as a '2'	242
77	Handling Small Blobs	246
78	Delimiting Region to be Filled	250
79	Reconnecting Broken Pieces: E-E Connections (Case 1)	251
80	Reconnecting Broken Pieces: E-E Connections (Cases 2 & 3)	252
81	Reconnecting Broken Pieces: E-B Connections (Cases 1 & 2)	253
82	Reconnecting Broken Pieces: Too Distant Pieces	254
83	Spurious Holes	255
84	Typical Samples With 2 Holes, After Spurious Hole Filtering	258
85	Filtering Out Small Isolated Features	260

86	Measuring Relative Perturbation Caused by Small Feature	261
87	Samples Incorrectly Classified as '0'	264
88	Samples Incorrectly Classified as '2'	265
89	Samples Incorrectly Classified as '6'	266
90	Examples of Width and Depth Feature Rule Measurements	311
91	Examples of unexpected number of holes	329

Chapter 1

Presentation

1.1 OCR: A Bit of History

Practical research in Optical Character Recognition (OCR) began in the 1950s, over 40 years ago. But the awareness of the possibility and potential usefulness of automatically processing text existed even earlier. Thus before the first commercial computer, UNIVAC I, was installed in the U.S. Bureau of Statistics in 1951, OCR patents based on template matching had been filed as early as 1929 in Germany and 1933 in the U.S.A. (see [116]). In the late 1950s, OCR systems had been designed in the U.S. and Japan and several were commercially available in the 1960s.

Initially, interest was focused on *printed* and very nicely *handprinted* alphanumeric characters but it did not take long before less constrained handwriting also received attention. In 1968, with the introduction of postal codes in Japan, the world's first machine able to directly read 3-digit handwritten codes was put into service; the numerals were entered within red frames and processed through OCR equipment.

Optical Character Recognition is probably the most researched area in Pattern Recognition, now addressing very diversified and sophisticated problems. Important research avenues include degraded omnifont printed text recognition, analysis and recognition of complete documents (including texts, images, charts, tables, etc.), and cursive handwriting recognition. Interest in this field is motivated by the challenging

nature of the problems to be solved, the fact that rapidly evolving computer technology allows practical solutions to more and more complex problems, and, above all, by its numerous commercial applications such as automatic processing of handwritten mail addresses, bank checks, credit card slips, payments of all kinds, income tax forms, etc.

In the past 10 years, there has been an explosion of research in all aspects of OCR prompted by faster computers, new approaches¹, and increased governmental and industrial involvement and cooperation. Several international forums and competitions were created, both reflecting this trend and acting as important driving forces on their own.

Postal applications in particular have played a major role with events such as the Advanced Technology Conferences of the U.S. Postal Service (held bi-annually between 1984 and 1992), the conferences of the Institute for Posts and Telecommunications Policy in Japan (1992 and 1993), and the First European Conference Dedicated to Postal Technologies (1993). Moreover, these sponsoring agencies were also important sources of funding for related research internationally.

Several industrial and university research teams took part in the International OCR competitions sponsored by NIST, the National Institute for Standards and Technology in the U.S. (1992, 1994), and by IPTP (1992, 1993). We also note the series of International Workshops on Frontiers in Handwriting Recognition (1990, 1991, 1993, 1994, and 1996) and the International Conferences on Document Analysis and Recognition (1991, 1993, 1995, and 1997). The IWFHR were first launched in 1990 at CENPARMI, Concordia University, and the third ICDAR was held in Montreal in 1995.

1.2 Focus of Our Work

The focus of the work described in this thesis is the *recognition of totally unconstrained isolated handwritten numerals*. In most applications, this is only one component of the solution of a complex problem.

¹ In particular, multi-layer perceptrons.

In automatic check processing, for example, the system begins with a full color image of the check; the background must be removed, the image binarized, the items of interest located and extracted. Individual numeral recognition will be the key in recognizing the courtesy amount, and will also play a role in recognizing the date. However, for the courtesy amount, touching or overlapping digits must first be segmented; furthermore, non-numeric symbols (period, comma, dollar sign, dash, etc.) must also be dealt with. The processing of the check may involve additional steps for the recognition of the cursive legal amount and some signature verification steps...

Among all steps just mentioned, our research will be concerned solely with recognizing already isolated handwritten numerals. Yet, even if this is only a part of the solution in most applications, it has been a very major area of research in its own right. Despite its apparent simplicity (there are, after all, only 10 different digits!), the problem is not easy to tackle because of the great variations in writing styles, instruments, and circumstances. By *totally unconstrained* handwritten numerals, we mean that people are not asked to write in pre-printed boxes, are not asked to write neatly, are not asked to write with a specific type of pen, etc. Hence, the goal is to recognize numbers written by people in real-life situations (as zip codes on envelopes, courtesy amounts on checks, ...).

For years, contradictory opinions have been voiced concerning the state of handwriting recognition. Here are two representative samples:

“The digit classification problem is as simple as many other classification tasks.” (see [92], page 447).

“For almost three decades, staff at the Census Bureau have heard claims that machine recognition of handwriting was just around the technological corner. However, a careful review of most claims showed that the corner was still a long way off.” (see [165], page 1).

The first statement deals specifically with digit classification; the second one purports to handwriting more generally, but it should be kept in mind that the first NIST competition focused entirely on recognizing already segmented individual digits. Thus

these affirmations are a reflection of the ‘already solved’ vs ‘still not satisfactorily solved’ opposite views on our research area.

Nevertheless, in their report on the second NIST competition, Geist et al. write:

“An important conclusion of the First Conference was that the OCR of isolated (properly segmented) characters was essentially a solved problem”. (see [57], page 5).

The last quote should be understood in light of the following context: for years, researchers had operated under the assumption that recognition of individual handwritten characters was the major stumbling block for cursive handwriting recognition in general. The NIST competitions proved otherwise; namely that segmentation is probably the most difficult subtask in the overall problem.

We do not question the very significant improvements made in recent years in handwritten character recognition nor the fact that, on some aspects of this task, machines can already outperform humans. But we still think that, in some crucial respect, especially when maximum reliability is required, human performance has not been matched. Our research efforts have been geared precisely towards making a contribution to bridging this reliability gap.

1.3 Outline of Thesis

In the next chapter, we will provide an in-depth overview of recent advances in unconstrained handwritten numeral recognition. Such a detailed account cannot be found elsewhere and is one of the contributions of this thesis. We are convinced that newcomers into this rapidly evolving field will appreciate such a complete ‘tour d’horizon’. The same chapter ends with an assessment of whether or not machine recognition has yet matched human performance, a question on which this author and his colleagues have also made contributions.

In light of the state of the art, an overview of our research will be presented in chapter 3. This will introduce our overall objective and discuss the orientations and guidelines which have shaped our work. It will also sketch the general design of the new recognition system which we have developed.

All following chapters will focus on particular aspects of our numeral recognition system. Chapter 4 will examine preprocessing steps, including edge and contour extraction, measurement of important global features and concurrent correction of several image defects. Chapter 5 will deal with contour smoothing by local weighted averaging methods, a specific aspect on which a detailed analytical investigation was conducted. Chapter 6 will consider the problem of curvature feature extraction, a very important part of our work including a comparative study of such feature extractors, a detailed analysis of the weaknesses of a particular method which we had previously developed, and the presentation of the newly created one to address these weaknesses. Chapter 7 will present our approach to the development of classification rules, making use of an interface for clustering samples by shape and gathering statistics about specific feature attributes, and using a special syntax for representing these attributes and their allowed ranges of values. Chapter 8 will discuss the overall system including the processing of multicomponent samples, and the filtering of small holes and of the curvature feature list leading to additional recognition passes; final results on several databases are also presented for a partial classifier. Finally, Chapter 9 will offer a conclusion, summing up the contributions made to this research field and indicating avenues for future work.

Chapter 2

State of the Art

In the past four decades, a wide variety of approaches have been proposed to try to capture the distinctive features of handwritten characters [154]. These approaches generally fell in two categories: global analysis and structural analysis. In the first category, we find techniques such as template matching, measurements of density of points, moments, characteristic loci, and mathematical transforms. Features of this type are generally used in conjunction with statistical classification methods (see [147]). In the second category, efforts are aimed at capturing the essential shape features of characters, generally from their skeletons or contours. Such features include loops, endpoints, junctions, arcs, concavities and convexities, and strokes. Most often a syntactical classification approach is used with structural features (see [17], [5], [65], and [16]).

For much more detailed accounts of historical developments in OCR, see the excellent surveys by Suen et al. [155] and by Mori et al. [116]. Of course, these surveys do not focus entirely on handwritten numeral recognition. Moreover, significant advances have been made in recent years which could not be accounted for in these reviews. These important gains seem primarily related to the availability of very large training databases, their use being facilitated by self-learning classification techniques in conjunction with more powerful machines.

Thus, in the next section, we provide an overview of approaches which were recently developed to tackle our specific problem. We have limited ourselves to methods

which reportedly produce high recognition rates and low error rates¹. In our condensed descriptions, we will focus on the diversity and sophistication of the schemes; we will simply quote recognition and substitution rates without discussing database issues which are nonetheless critical for any serious comparison of results. In a later section, we will discuss comparative results between methods, where statistics on the same testing databases are available.

2.1 More Sophisticated and Diversified Methods

2.1.1 Structural Methods

In Brown et al. [22], a number of expert modules are used to improve the quality of skeletons so that more reliable features can be derived. In Stringa, [152] and [153], a three-layer network performs image compression by Boolean functions, which are established automatically during the learning phase. The compression preserves the essential topological features, and classification is by standard parsing techniques. A 92.6% recognition rate with 4.6% substitution rate were obtained. Mitchell and Gillies [114] use the tools of mathematical morphology to extract cavity features as the starting input to their specialized digit recognizers. Thirty-three numeral models were “painstakingly crafted” with an iterative refine-and-test methodology over several thousand digits. Classification is performed by a symbolic model matching process resulting in 87.95% and 1.04% recognition and substitution rates respectively.

Four structural recognition methods were also developed at CENPARMI during the same period. They are described in [157] where they are labeled as Experts E1 through E4. Thorough discussions of each method can be found in [117], [82], [105], and [94] respectively.

Expert E1 begins with a skeleton which is then decomposed into branches which are vectorized and compressed. Several attributes are measured on each branch; others are computed as needed during classification. Eleven rule-based classification

¹ The *recognition rate* is the ratio of correctly recognized over total processed samples; the *substitution rate* is the ratio of misclassified samples over total processed samples. Samples which are neither recognized nor substituted are tallied as part of the *rejection rate*.

modules (one for samples with a single branch, and one per digit class for multi-branch patterns) were manually constructed. Unknown samples go through all modules which may result in multi-class assignments.

Expert E2 approximates the skeletons of numerals by line segments, later grouped into convex polygons, and by loops. For each primitive, features are extracted such as primitive type, direction, coordinates of starting and ending points, etc. The preceding and following primitives are also recorded. Simpler samples (about 80%) are classified by decision trees based on these primitives and the remaining ones are passed on to a relaxation algorithm trying to match the unknown sample with selected masks.

Expert E3 scans the result of skeletonization from top to bottom, left to right, and extracts the sequence of special points (end-, fork-, cross-points) as primary features. Secondary features based on human perception are then extracted from the skeleton *and the contour* of the numeral image. Frequencies of occurrence of these features on the training data are stored in a database. A recognition decision can be obtained by comparing unknown and stored feature vectors. An alternate inference method is realized by intersecting the hypothesis sets associated to each feature present in the unknown sample. Contrary to the other 3 experts, it should be noted that Expert E3 is 'trainable'. We shall return on this point later.

Expert E4 extracts concave and convex regions along a smoothed contour of the sample. These regions are then labeled as cavities, bends or endpoints. For each feature, its type, position, initial and final points, and general orientation are recorded. A tree classifier attempts to recognize the least complex numerals first (mostly 0s and 1s) and then proceeds based on the number and location of holes: the feature list is examined; often new measurements are made, as needed, to confirm or rule out the hypothesized identity. Classification ends as soon as the sample has met all the conditions for a particular class. If unsuccessful, a small hole (if present) may be dropped and recognition attempted a second time. If still unsuccessful, the feature list is filtered and another attempt may be made.

Recognition and substitution rates for the 4 Experts are respectively: 86.05% and 2.25% (E1); 93.10% and 2.95% (E2); 92.95% and 2.15% (E3); and 93.90% and 1.60%

(E4).

Abuhaiba & Ahmed [1] apply smoothing and thinning to the sample image and then find polygonal approximations to the skeletons. These approximations are converted to tree structures from which Character Graph Models (CGM) are derived. Constraints with fuzzy distributions are added to differentiate between the different characters recognized by the same CGM. When all learning samples have been recognized, a set of 105 Fuzzy Constrained Character Graph Models (FCCGM) is obtained. Reasonable results are reported (90.7% recognition and 2.9% substitution rates) but the database is neither very large nor widely used.

Nishida & Mori ([119] and [120] and Nishida [118] have made interesting contributions by proposing curve descriptions and methods to automatically construct structural models from these descriptions. The starting point is a model-based thinned image of the numerals. Points with at least 3 incident edges are called singular points. The description encodes the structure of singular points and the quasi-topological structure of each stroke, specified in terms of curve primitives, loops, and their connectivity. With this description method, structural models can be built automatically from the training data, by applying an *exact* string matching method. In [120], 2 experiments were conducted on this basis. The first experiment uses 13 400 digits neatly written by 251 writers. Half the data was used for training and the other half for testing. The training process produced 46 structural models². For the test set, the recognition rate was 98.7% and the substitution rate 0.3% (1% rejection). In the second experiment, all the neatly written samples were used as training material and the system was tested on 127 000 unconstrained handwritten numerals written by 319 other persons. The results achieved were recognition and substitution rates of 95.4% and 1.7% respectively. In Nishida [118], much more flexibility is brought to the curve description, in particular by allowing commonly occurring discontinuous transformations in numeral handwriting. Two such transformations are analyzed systematically and an algorithm is developed to infer super-classes under these transformations. As a result, few models are required to capture several shape variants of a digit. Using 14 000 samples from a public-domain database (ETL-1, Japan), half for training and

² Only 2 models were required for '6's but as many as 10 models for '8's.

the other half for testing, the recognition and substitution rates were 98.6% and 0.9% respectively. Note that only 12 descriptions were required for the 10 digits.

In closing this section, we note that most methods discussed above were devised in the late 1980s and early 1990s. Very few serious incursions on this terrain have been reported since then.

2.1.2 Artificial Neural Networks

In the past 6 or 7 years, there has been a tremendous increase of interest into artificial neural networks (also called multi-layer perceptrons, MLPs) as a possible solution to the problem of recognizing handwritten numerals. By far, this has been the preferred avenue of researchers and excellent results were achieved. The primary advantage of neural networks is their ability to be trained automatically from examples. Other assets include possible parallel implementation, and their good performance with noisy or incomplete data. Such networks are composed of several layers of interconnected elements, each of which computes a weighted sum of its input and transforms it into an output by a nonlinear function. In the backpropagation learning phase, the weights associated with each connection are modified until the desired outputs are obtained.

Handwritten numeral recognition with MLPs has followed a few major directions: feeding the input layer with results of more sophisticated feature extraction; or, on the contrary, simply feeding the input layer with normalized numeral images without any explicit feature extraction; experimenting with various network architectures in both approaches; etc...

With Explicit Features as Input

In Krzyzak et al. [75], 15 complex Fourier descriptors are extracted from the outer contours and simple topological features from the inner contours. They are presented as input to a fully connected three-layer network. Training is done by a modified back-propagation technique to avoid the problem of local minima in the gradient descent technique used to adjust the weights; this results in a convergence rate which is twice as fast. Recognition and substitution rates of 86.40% and 1.00% or 94.85%

and 5.15% are achieved.

Important research on the use of neural networks for handwriting recognition has been conducted at AT&T Bell Labs for several years. In LeCun et al. [84], two different methods are presented, and implemented on chips. The first method is discussed here and the second in the next section. In the first method, shape and skew normalization transform the input image into a 32x32 bit map; skeletonization is carried out with 5x5 windows; features are extracted based on 49 templates (inspired from experimental neurobiology and fine-tuned by hand) which check for the presence of oriented lines, end-stops, and arcs. This results in 49 32x32 maps which are reduced to 18 3x5 maps (270 bits). Classification is by a fully-connected 3-layer MLP with 270 input units, 40 hidden units, and 10 output units (approximately 11 000 weights in total). This system achieves 94% recognition with 6% error rate. When tuned for 1% error rate, recognition falls to 86% (13% of samples are rejected).

For a French postal code application, Lemarié uses morphological preprocessing³ to extract a vector of length 138 from the bitmap image of the character. The learning stage is in 2 phases: in phase #1, the width of each center and output weights are learned; in phase #2, the scalar variance of each center is replaced by a diagonal co-variance matrix and the diagonal elements are learned. For the diagonal RBF system, the no-rejection error rate is 2.03%, significantly better than the 3.71% figure obtained from the RBF network alone.

In Lee et al. [91], input images are size-normalized to 16x16 and directional features in horizontal, vertical, and both diagonal directions are extracted using Kirsch-like masks. Each 16x16 directional feature map is then compressed to 4x4; the original image is also compressed to 4x4 as a means to incorporate global features into the input. The input and the hidden layers both consist of five 4x4 clusters, a cluster in the hidden layer being fully connected only to its corresponding input cluster; the output layer is fully connected to all hidden nodes. A genetic algorithm is used to provide good seeds for backpropagation training and circumvent the local minima

³ The method is inspired from characteristic loci; for each background pixel, points crossing the sample in 8 directions are recorded.

problem. Three databases are experimented with, enhancing their generality by using the original as well as perturbed training data. The no-rejection error rates for each of the 3 databases are 2.20% (CENPARMI, Canada), 0.87% (ETL-1, Japan), and 0.60% (ETRI, Korea).

Using the same preprocessing and feature extraction as just presented, Lee & Kim [90] also experiment with a fully-connected 3-layer recurrent neural network (RNN) where each output node is connected not only to all hidden nodes but also to all output nodes, including itself. Fewer errors are generated on the CENPARMI test set by their proposed RNN (2.7%, without rejections), as compared to a feedforward neural net (7.7%) and two other types of RNNs, Jordan RNN (4.2%) and Elman RNN (3.4%). Again using the same preprocessing and feature extraction as Lee et al., Cho [32] obtains a no-rejection 3.95% substitution rate on the CENPARMI test set also. The classifier is a structure-adaptive self-organizing map (SOM) with an input layer directly connected to an output layer. Both the number of output nodes and the connection weights are learned in the training process.

Lee et al. [89] use a cluster neural network as in citeLeeSW:94. A multiresolution feature vector is obtained by convolving Haar wavelets with 16x16 normalized images (at 8x8 and 4x4 resolutions). This time, the network consists of eight sub-networks (instead of five above). The reported no-rejection error rates for each of 3 databases are: 3.20% (CENPARMI, Canada), 0.83% (ETL-1, Japan), and 0.75% (ETRI, Korea).

As input to their neural net, Matsui et al. [110] use a 182-component feature vector which combines three types of features: counts of segment (stroke) types which are based on the number of connected component above and below a given segment (22 components); background structure features⁴ detected within 16 regions equally subdividing the original image (32 components); finally a histogram based on 8 chain-code directions for each of the 16 regions and normalized in the (0,1) range (128 components). The fully connected network had 182 input units, 70 hidden units, and 10 output units. Performance is measured with $S = 10E + R$, where E is the error

⁴ These floating point features are based on object pixels encountered in a certain direction and then in a perpendicular direction.

rate and R is the rejection rate. For one database⁵, the best results are given as $S \approx 7.5$; recognition and substitution rates of approximately 96.1% and 0.4% can be inferred from a curve. For another database⁶, the no-rejection error rate is given as 1.43%.

In Strathy & Suen [151], remarkably good results are obtained using very simple pixel distance features (PDF). Numeral images are thinned and 2 features are measured for each pixel, namely the signed distance to the nearest black pixel in the horizontal and vertical directions. PDFs are averaged to downscale the image to size 12x14 and normalized in (-1,1) range. A fully-connected backpropagation network with 336 input nodes, 70 hidden nodes, and 10 output nodes is trained with a standard database *and also* with the same data to which random shearing is added. A no-rejection error rate of 3.69% is reported.

With Normalized Images as Direct Input

The second method presented in LeCun et al. [84] uses shape and skew normalization producing smaller 16x16 bit maps (instead of 32x32). They are fed directly as input to an MLP with 3 hidden layers: the first layer consists of 12 8x8 feature maps; the second of 12 4x4 feature maps and the third is fully connected to the preceding and the output layers. The network has 1 256 units and 64 660 connections, but only 9 760 free parameters because of weight sharing among the feature maps. This system achieves 95% recognition with 5% error rate. When tuned for 1% error rate, recognition falls to 86.9%.

The *local receptive fields* approach just described has been pushed further in LeCun et al. [83] using a more complex but more highly constrained network architecture. Size-normalized images are used as direct input. The network has four internal layers, two made of independent groups of feature extractors and two which perform averaging/subsampling. The last internal layer is fully connected to the ten-element output layer but all other connections are local and use shared weights. In total there are 4 635 units, 98 442 connections, but only 2 578 independent parameters. Typical

⁵ Using 15 000 numerals from the test set of the second ITPP competition (see section 2.3.5).

⁶ An unspecified NIST database.

recognition and substitution rates are: 92.00% and 2.00% respectively or 90.00% and 1.00%.

Le Cun et al. argued that the excellent generalization properties of their system was a direct result of the constrained architecture which incorporates high-level knowledge about the problem at hand. But Martin & Pittman [109] dispute this claim. In experiments using from 100 to 35 200 training samples, they feed size-normalized 15x24 grayscale images, with pixel values ranging from 0.0 to 1.0, directly into a basic fully-connected network. They report recognition and substitution rates of 96% and 4% respectively (or 89% recognition for 1% error). By modifying the number of hidden nodes, limiting connectivity to local areas, and sharing weights, they show that the effects on generalization are only marginal provided a net with sufficient capacity and a large enough training set are available. The latter factor, according to Martin & Pittman “may be the single, most important factor in achieving high recognition accuracy”. Constrained architectures could still be preferred for their training efficiency, bias towards position invariance, and more straightforward interpretation.

Using a very large database⁷, Bottou et al. [20] report on the results obtained by several classifiers recently developed at AT&T Bell Laboratories. Input images are size-normalized to 20x20 and then centered (using center of gravity) into a 28x28 frame, resulting in grayscale pixel values. A perceptron baseline linear classifier consisting of an input and an output layer with 7 850 weights yields a no-rejection 8.4% error rate. A 3-NN (nearest neighbour) classifier achieves 2.4% error rate. The method described above from Le Cun et al. [83] is called ‘LeNet 1’; it requires 140 000 multiply/add operations and gives 1.7% substitution rate. A large fully-connected multi-layer neural network with 300 hidden units attains 1.6% error.

Five other classifiers achieve a 1.1% error rate on the same test set. One is called ‘LeNet 4’ and is an expanded version of LeNet 1, including more feature maps and an extra hidden layer fully-connected to both the last feature-map and the output

⁷ Training and testing sets each containing 60 000 samples, combining the training and testing sets of the 1st NIST competition, described in section 2.3.

layers. It has 17 000 free parameters and requires 260 000 multiply/add steps. Another variant combines LeNet 4 with k-NN classification by using the output of the penultimate layer of LeNet 4 as a feature vector for a Euclidian distance search. With the same feature vector, a third variant trains simple linear classifiers (the training is local, based only on the k patterns in the training set which are closest to the test pattern). The fourth classifier achieving 1.1% error rate is a 3-NN classifier using a ‘tangent’ distance instead of a Euclidean distance. For more information, see Simard et al. [148]. The fifth classifier is an Optimal Margin Classifier, consisting of 10 classifiers oriented to two-group pattern classification with a 4th degree polynomial decision surface in input space. We note that all five methods have very high memory requirements (11 to 25 Megabytes) except LeNet 4 (60 Kilobytes).

2.1.3 Statistical Methods

Nearest Neighbour Classifiers

Increased computation power has caused simple but traditionally less efficient methods such as nearest neighbour classification to be revisited. Yan (see [171] and [172]) divides the image into a square grid and computes the percentage of black pixels in each square as a feature. Initially, training samples are clustered and clusters are averaged. These are used as initial prototypes for nearest neighbour comparison. Then the prototypes are optimized via a multi-layer perceptron with one hidden layer, where each node represents a prototype and its coordinates in feature space are adjusted. In [172], with the 500 initial prototypes, the no-rejection error rate is 8.3%; after prototype optimization, it is lowered to 3.9%. In [171], 1 000 optimized prototypes yield typical recognition and substitution rates of 94.68% and 0.66% respectively, or 96.15% and 1.00%. It should be mentioned that the data used is from the relatively easy SD3 NIST database; see 2.3. Speed up techniques are implemented which save about 40% of computation time.

Smith et al. [149] downsize all images in their minimum bounding box to 32x32 bitmaps. With a very large database⁸, for each writer, they use all samples from

⁸ All 223 125 segmented digits of the SD3 database are used.

all other writers for k-NN classification. The computation is performed on a massively parallel supercomputer (Connection Machine CM-2) with 3 distance metrics: a Hamming metric, a pixel distance metric, and a pen stroke (breakpoints defined at curvature maxima of a thinned image) metric. The no-rejection error rates are 1.9%, 1.1%, and 1.0% respectively. It is observed that for every 10-fold increase in the size of the learning set, the error rate is cut by half or more. In addition, the distance metric $1 - D_1/\bar{D}_1$, where D_1 is the distance to the nearest neighbour and \bar{D}_1 is the nearest distance to a sample of another class is shown to be much superior to the metric $1 - D_1/K$, where K is a constant which is larger than the maximum distance.

W.P. de Waard [36] also investigated nearest neighbour classification on the large datasets of NIST's first competition. A smoothed contour description of each numeral is obtained to which linear transformations are applied for translation, deskewing, and resizing. Features extracted are 40 projections, 8 directional histograms, 13 characteristic loci, and 4 topological features. A method is proposed to find discriminative prototypes with optimized weights for every feature. With only 10 prototypes, the no-rejection error rate is 4.5%. But, surprisingly, starting with many prototypes per class, made no difference at all. The author explains that the optimized distance did not work quite as expected and that due to the global nature of backpropagation, small clusters are not detected.

Other Distance Classifiers

Heutte et al. [59] combine 4 families of features: 138 based on concavity measurements extracted from normalized images; 20 from normalized histograms of black pixels in horizontal and vertical directions; 21 from polygonization of outer and inner contours; 30 from pixel extrema i.e. black pixels not 8-connected with pixels 'above' (in the 4 main directions). Despite the variability of some structural features, they manage to create an ordered fixed-length feature vector of numerical variables. The initial 209 features, normalized in (-1,1) range, are reduced to 157 by removing variables with insignificant discriminating power or highly correlated to others. The learning process defines a set of hyperplanes separating pairs of classes. No-rejection substitution rates are 1.95% and 1.84% for the full and reduced feature vector respectively.

In Revow et al. ([133] and [132]), a generative model is constructed for each numeral class using a uniform, cubic B-spline, with at most 8 control points, and with Gaussian ink generators uniformly placed along the spline. The elastic matching recognition process fits all digit models to the unknown sample, minimizing an energy function which weighs both the deformation energy of the model⁹ and the log probability that the model would generate the inked pixels. Once optimal model deformations are computed, the results (7 terms per model) are fed into a simple neural network which determines which model fits best. Using one generic model per digit class, a no-rejection error rate of 1.53% is reported in [132] on the well-segmented numerals of the CEDAR *goodbs* dataset. This is slightly improved to 1.50% when a mixture of local models are used for each class. However, computation costs constitute a serious drawback of this method: the authors mention a classification rate of only 5.5 numeral images *per minute* on a R4000 based workstation, even after downsizing the image to one quarter of its original size. Parallelization techniques can however speed up the process.

As a more practical alternative computation-wise, Hinton et al. [60] use principal component analysis (PCA) and factor analysis (FA) to develop locally linear low-dimensional approximations to the underlying manifolds of images of handwritten digits. The images are first scaled on an 8x8 pixel grid and smoothed with a Gaussian filter with a standard deviation of half a pixel. Mixtures of linear submodels are built for each class. For the results reported, they allowed up to 10 principal components or factors per submodel, and 10 submodels in each mixture. No-rejection substitution rates of 2.35% (PCA) and 2.17% (FA) are obtained on the *goodbs* test set. Adding tangent information in the submodels improves the performance to 2.17% for PCA but degrades it to 2.49% for FA.

Polynomial Classifiers

The AEG Daimler-Benz company has been an important manufacturer of large scale OCR equipment for years. Interestingly, they have shared some results of their research activity through several papers in journals and conferences. For handwriting

⁹ Moving the control points away from their home locations.

recognition, they have conducted diversified experiments on several databases based on their functional classifier approach.

The functional classifier is presented in [48]. Starting with a feature vector \mathbf{x} , these features are enhanced in a non-linear way using functions $f_j(\mathbf{x})$ and the enhanced features are combined linearly to yield:

$$d_i(\mathbf{x}) = \sum_j a_{ji} \cdot f_j(\mathbf{x}), \quad \text{for } j = 0, 1, \dots, 9 \quad \text{or} \quad \mathbf{d}(\mathbf{x}) = A^T \mathbf{f}(\mathbf{x}) \quad (1)$$

The author presents the advantages of the functional classifier over the multi-layer perceptron (MLP):

1. Once the functional components $f_j(\mathbf{x})$ have been generated, the mathematics of solving linear problems can be applied. Efficient techniques exist for matrix inversion and they are faster than the gradient descent method for the adaptation of the MLP.
2. Some matrix inversion techniques offer additionally a rank order for each functional term, showing how important it is for the classification task.
3. For learning sets larger than the number of coefficients of the functional approach, the generalization ability of the functional classifier increases dramatically and surpasses that of a MLP.
4. Moment matrices can be computed in advance and this is the most expensive part of the inversion process. Furthermore, combining subsets of training samples is equivalent to a linear combination of moment matrices with weights. Thus if statistical moments are computed separately for each class, they can be used very efficiently for different tasks (numeral, alphabetic, or alphanumeric classification) and different classifier hierarchies¹⁰.

A *polynomial classifier* is a special case of the functional classifier where the functions $f_j(\mathbf{x})$ are feature vector components or products of feature vector components.

¹⁰ Provided, of course, the functional structure is the same for all classes, tasks, or structures.

For example, $f_j(\mathbf{x}) = x_k$ is a linear feature; $f_j(\mathbf{x}) = x_k \cdot x_l$ is a second order feature, and as a special case, $f_j(\mathbf{x}) = x_k^2$ is a quadratic feature. In practical situations, features of order higher than 2 are rarely used.

In Franke [46], slant, stroke width and size normalization is applied to binary images, producing 16x16 pixel images in 8-bit gray levels. Using the 256 pixels as \mathbf{x} , an incomplete quadratic polynomial classifier of length 255 is constructed (labeled PIQ255_B_SW13) by retaining the 128 highest ranking single pixel features and the 127 highest ranking second order (but not quadratic) features. For this classifier, recognition and substitution rates of 93.10% and 3.10% respectively are reported. For another incomplete quadratic polynomial classifier of length 1 079, labeled PIQ1079_NB_SW13, the 175 highest ranking first degree features and the 904 highest ranking second order (including quadratic) features are retained. Recognition and substitution rates of 94.55% and 1.25% respectively are given.

Studies Focusing on Features

Bailey & Srinath [14] investigate the approximation of 2-D images by expansion in terms of orthogonal polynomials (Legendre, Zernicke, and pseudo-Zernicke polynomials); these orthogonal moments are used as features for Bayes quadratic, k-nearest neighbour, Parzen (kernel), and neural network classifiers. Various combinations of feature type and order, location invariance method based on centroid or minimum bounding circle, and classifier are evaluated. The training and testing sets includes 13 250 and 3 300 samples respectively. The best results achieved are a no-rejection error rate of 2.4% for the minimum bounding circle pseudo-Zernicke moments of order 4 with a Parzen (option 3) classifier, and 2.7% for the centroid-based pseudo-Zernicke moments of order 6 with a two-layer MLP. Overall, averaging performance over all moment types and orders, these two classifiers were the best and almost at par; conversely, averaging performance over all classifiers, the centroid-based and minimum bounding circle pseudo-Zernicke moments proved to be the most discriminant features, again almost at par.

Gader & Khabou [55] propose a method for automatically generating discriminating features for numeral recognition. All images are moment-normalized to size

24x18. For each class, a pixel sum image is obtained by averaging 1000 images and mapping the result in the $[-1,1]$ range. A feature detector for a given class is specified by 4 parameters: it is an $m \times n$ subimage of the pixel sum image for that class, with upper left corner at (x_o, y_o) . The height m is chosen randomly between 6 and 12; the width n is chosen randomly between 4 and 8. One hundred features (10 for each class) were generated as follows: a) randomly generate 10 features per class; b) perform 500 iterations during which the feature with the lowest evaluation can be replaced by a new feature for the same class if its evaluation is improved. An information measure and an orthogonality measure were used to evaluate features. The best 10 features per class are retained at the end of the generation/evaluation procedure. Their values are used as input to an MLP with 100 input nodes, 25 nodes in a first hidden layer, 15 nodes in a second hidden layer, and 10 output nodes. Training is by back-propagation. Orthogonality-based features produced better results. Results reported on the CENPARMI test set range from a recognition rate of 96.50% with a substitution rate of 3.50% to a recognition rate of 93.00% with a substitution rate of 1.55%.

2.2 Combination of Recognition Methods

The designing of high performance numeral recognition systems is generally not an easy task, and improving on already excellent systems is a very difficult proposition. However, it was realized a while ago that the *combination of different systems* offers a promising approach in this respect, especially when the systems to be combined are significantly different. In this last case, mis-classifications will tend to occur on different samples; thus the combination scheme may avoid many errors in this way. Recently, this avenue has received much theoretical and experimental attention.

2.2.1 Multistage Classification Methods

In multistage systems, more efficient methods are first used to reliably classify well-formed numerals while more complex and costly methods are applied in later stages

for characters of poorer quality.

Duerr et al. [43] reported a remarkable no-rejection recognition rate of 99.50% achieved with a four-stage classification scheme: a conventional statistical classifier and a fast structural classifier are first applied to all samples and correctly recognize approximately 95% of them; for the remaining samples, a structural hypothesis reducer and possibly a final heuristic matching stage are invoked.

The method of Lam & Suen, called Expert E2 in section 2.1.1, is actually a 2-stage classifier. Kuan & Srihari [76] obtained 93.3% recognition with 2.5% substitution with a three-level hierarchical classifier, based primarily on the topological structure of detected strokes and secondarily on contour profile information. Gader et al. [54] experimented with various multistage combinations involving four classifiers: a model-based classifier ([114]); a Jaccard/Yule template matcher; a fully connected four-layer neural network using coarse-coded versions of the image and of six basic cavity features as input; and, finally, a Fourier coefficient classifier. The pipeline consisting of the template matcher, followed by the model-based classifier and neural network, achieved the best results: recognition and error rates of 96.35% and 1.00% respectively on one database and 98.20% and 0.77% on another.

Franke [46] also investigates several polynomial classifier *hierarchies* such as binary trees, 2-stage classifiers, and classifier nets¹¹. These classifiers were trained on a very large proprietary database (over 1 000 000 samples) and tested on a totally different database¹². Some of the 6 classifiers investigated were adapted on the 10 numeral classes while others were adapted on 22 shape classes (some numeral classes were divided into 4 shape classes, for example). The best performances were by classifiers labeled SHAPE_N and SHAPE_N_SW13.

Both are classifier nets consisting of 231 2x2 classifiers for every pair of shape classes¹³; for the second system, stroke width normalization to 13% of the height and width of the character was applied to the test set data. Recognition and error rates

¹¹ Composed of $\binom{K}{2}$ pairwise classifiers, where K is the number of classes or subclasses to distinguish.

¹² The CENPARMI database.

¹³ Even pairs of shape classes belonging to the same numeral class.

are 95.05% and 1.30% for the SHAPE_N and 94.45% and 1.10% for SHAPE_N_SW13. Since these classifiers were not trained on CENPARMI data, they can be viewed as very sophisticated feature extractors for this data. Thus the 22-component output vectors obtained for the CENPARMI training samples by the SHAPE_N_SW13 classifier can be used as input to train a complete quadratic polynomial classifier of length 275. This CQ22 classifier adapted to SHAPE_N_SW13 yields improved recognition and error rates of 96.00% and 0.35%. Similarly, the no-rejection error rate was improved from 2.45% to 1.70%.

Franke [47] report on experiments with the BS (SUNY, Buffalo) database. The 16x16 images are size and slant normalized resulting in 8-bit grey levels. Using principal axis transformation, the input vector is reduced from dimension 256 to 40. Several classifier hierarchies are tried. They are trained on 10 000 images per class, totally independent of the BS dataset, to meet a target error rate of about 1%. The best results are obtained by a method labeled 'KH30F2ZTG', which is basically the same as SHAPE_N. When tested on the BS dataset, recognition and substitution rates are 97.02% and 0.90% respectively.

In Chi et al. [31], a 2 stage classifier is presented. Original binary images are rescaled and centered in a 64x64 region, and then sub-sampled into an 8x8 grey-scale image producing 64 features. The Kohonen self-organizing map (SOM) algorithm produces prototypes of the training set which are used to determine fuzzy regions and membership functions. The first stage of classification uses 10 401 learned fuzzy rules and yields recognition and substitution rates of 84.8% and 1.8% respectively, rejecting 13.4% of the data. In the second stage, rejected samples are fed to a SOM classifier resulting in 96.8% and 3.2% figures.

Cao et al. [25] present a 2-stage neural net classification scheme. After slant correction, binary images are scaled to 80x64 pixels. For the first stage, images undergo 6 passes of 3x3 neighbourhood averaging and are downsampled to 16x16. This input vector of length 256 is fed to a one-layer incremental clustering neural network. The distance between an unknown sample and the means of every cluster are computed: if the closest and second closest clusters belong to the same class and the inter-cluster distance is large, the sample is recognized; if the 3 closest clusters are

close, it is rejected; in all other cases the decision is taken by the appropriate subnet in the following stage. For stage 2, histograms of chain-code orientation (4 directions) are obtained for each of 16 rectangular zones with fuzzified borders. This serves as input to 45 pairwise neural net classifiers, each having 3 outputs: one for each class of the pair and one for rejection. If one output neuron is activated with large enough value while the other two values are low enough, the sample is recognized; otherwise, it is rejected. Playing with these thresholds, recognition and error rates ranging from 97.5% and 0.76% to 85.33% and 0.17% are reported. In [24], Cao et al. present another 2-stage neural net classifier. Preprocessing includes slant correction, scaling and smoothing as already explained, followed by downsampling to size 8x8. The first stage extracts 'principal components' using Oja's rule; optimum performance was obtained for 40 'principal components'. The second stage is a Bayes incremental clustering neural net; in the training process, this stage begins with 3 nodes (clusters) per class and a new sub-cluster is constructed if the conditional probability of the feature vector with respect to each existing sub-cluster is below a threshold. Merging of within class sub-clusters is also performed. The number of clusters stabilizes at 90. Results indicate that this architecture offers some improvement compared to a backpropagation neural network with a 50-node hidden layer using the preprocessed 8x8 image as direct input. Furthermore, learning time is in the order of minutes for the new system, in the order of days for the backpropagation network.

With preprocessing and features as described in Section 2.1.2, Cho [32] also proposes a hybrid Hidden Markov Model (HMM)/MLP classifier. Usually, one Markov model is trained for each class and recognition involves accumulating scores for an unknown input across the nodes in each class model, and selecting the class model with the highest score. Instead of this last maximum selection step, the hybrid classifier uses the likelihood patterns inside the HMM's as input layer to an MLP. This system obtains a no-rejection substitution rate of 3.45% on the CENPARMI test set.

2.2.2 Multi-Expert Classification Methods

In this section we discuss another avenue for the combination of classifiers. Here the images of numerals are fed, independently and possibly in parallel, to two or more

recognition systems which can be considered as ‘experts’. The combination takes place at the level of the decisions of these experts. This problem has received much theoretical and practical attention in recent years.

Theoretical Considerations

Given a sample x from pattern space $P = \cup C_i \forall i \in \Lambda = 1, 2, \dots, M$, where the C_i are mutually exclusive classes, the task of a classifier expert (denoted e) is to assign to x an integer label $j \in \Lambda \cup M + 1$ indicating that x belongs to class C_j . The label $M + 1$ is to indicate that the classifier cannot classify x and is the rejection label.

However the information that is output by a classifier is not always a single label j . In general, we can distinguish between three types of output information and the problem of combining classifier outputs will vary according to these categories. In the first level, called the *abstract* level, each classifier e_k , $k = 1, \dots, K$ outputs a single label j_k , or possibly a subset $J_k \subset \Lambda$. In the second level, called the *rank* level, each e_k ranks all the labels in Λ or in a subset $J_k \subset \Lambda$ in a queue with the label at the top being the first choice. In the third level, called the *measurement* level, each e_k assigns each label in Λ a measurement value indicating the degree to which x has that label. In the rest of this section, we will denote by $E(x)$ the combined result of the outputs $e_k(x)$, regardless of the information level.

Xu et al. [170] investigate the combination of *abstract level* output information. They first consider several variants of the simple voting principle:

- $E(x) = j$ only if all $j_k = j$; otherwise, $E(x) = M + 1$; this will achieve very high reliability at the expense of a high rejection rate.
- A somewhat less stringent combination is obtained with the combination $E(x) = j$ if all $j_k \in j, M + 1$.
- $E(x) = j$ if the count of votes for one class j is larger than $K/2$, otherwise $E(x) = M + 1$.
- $E(x) = j$ if the count of votes for one class j is larger or equal to $\alpha \cdot K$, which allows flexible control of the rejection level.

- $E(x) = j$ if the count of votes for class j exceeds all others by at least $\alpha \cdot K$, otherwise $E(x) = M + 1$.

They also examine the combination of multiple classifiers in the Bayesian formalism and in the Dempster-Shafer formalism. In the first case, they propose using the confusion matrices of the classifiers to compute conditional probabilities $P(x \in C_i / e_k(x) = j_k)$; these then serve to compute estimates of the belief function that $x \in C_i$ which is given by:

$$bel(i) = \eta \prod_{k=1}^K P(x \in C_i / e_k(x) = j_k). \quad (2)$$

In the Dempster-Shafer formalism, a basic probability assignment m is given not only to each class C_i as with the Bayesian formalism but to every subset of classes in the pattern space P and the belief that a sample belongs to a subset of classes $A \subset P$ is given by:

$$bel(A) = \sum_{B \subseteq A} m(B). \quad (3)$$

In this formalism, Xu et al. use only the global recognition, substitution and rejection rates of each classifier as prior knowledge.

Huang & Suen ([67] and [66]) present the Behavior-Knowledge Space (BKS) method which is claimed to be the best for the combination of multiple classifiers providing *abstract level* information. The BKS is a K -dimensional space which is constructed during a learning phase; in this space, the bin of coordinates (j_1, j_2, \dots, j_K) is called the focal unit for every sample for which the classifier decisions are $e_k(x) = j_k$; each bin keeps the counts of incoming samples belonging to each class, from which the best representative class can be deduced and used to determine $E(x)$ ¹⁴. This method does not assume classifier independence but suffers from high storage requirements. Often a large number of bins will not be used as focal units and storage problems can be alleviated with dynamic allocation schemes. Huge training sets are needed to take full advantage of this method.

Ho et al. [61] consider the problem of combining *rank level* decisions of classifiers

¹⁴ In conjunction with some threshold value for rejection/substitution trade-off.

and propose several methods. Here the application is machine printed word recognition for a 1365-word lexicon. Two methods are investigated for *class set reduction*, where one attempts to reduce the number of classes in the output list without losing the true class. And three methods (highest rank, Borda count, and logistic regression) are proposed for *class set reordering*, where the idea is to improve the rank of the correct class. *Dynamic classifier selection* is also examined, where an oracle is modeled to predict, before recognition, which classifier is best to use for the sample.

Lee & Srihari [88] propose a unified theory of classifier combination based on the neural network approach. A scheme is presented to transform classifier outputs of any of the 3 kinds described above into M -dimensional vectors of values in the $[0,1]$ range, where M is the number of classes. These vectors serve as input to a Decision Combination Neural Network (DCNN), a multi-layer perceptron without a hidden layer. The elimination of redundant classifiers and soft weight sharing are also considered. Finally, a dynamic selection network is integrated to the DCNN resulting in a Dynamic Selection Combination Network (DSCN). This last idea is similar to the dynamic classifier selection of Ho et al. The implementation is via a side network which takes the images as input and outputs weights in the $[0,1]$ range for each classifier being combined. The output vectors of each classifier are then multiplied by these weights before they are fed into the DCNN.

Lam & Suen [80] investigate specific questions concerning the addition of experts in the following simple majority voting scheme: assuming each of the K classifier experts outputs a single class label, the combined decision will be $E(x) = j$, provided at least $\lceil (K + 1)/2 \rceil$ of the classifiers agree on that label; otherwise, $E(x) = M + 1$ i.e. the sample is rejected. They show that *adding one vote* to an even number of decisions will mostly increase the recognition rate while adding one vote to an odd number of decisions will mostly increase the number of rejections. Also, *adding 2 votes* to an even number of votes increases the recognition rate with high probability (the new error rate being dependent on individual classifier substitution rates); and adding 2 votes to an odd number of decisions decreases the substitution rate with high probability. Interesting variations on majority voting are analyzed: a) to increase reliability (less substitutions) when we have an odd number of classifiers, is it better

to drop one classifier or to double one up? b) similarly, to increase the recognition rate when we have an even number of classifiers, what is the best course to adopt? The best solution depends on the relationship between products of the odds ratio $r_i r_j$ for pairs of experts, where $r_k \equiv p_k / (1 - p_k)$ and p_k is the probability that expert k 's prediction is correct.

Practical Results

Possibly the first instance of combining the results of different classifier experts can be found in Mandler & Schürmann [107] for an on-line script recognition application. The votes of 3 experts are combined together based on the Dempster-Shafer theory of evidence and the combined decision is seen to be much better than that of any individual classifier.

In Cohen et al. [33], four algorithms are used in parallel to recognize numerals in ZIP codes: a polynomial discriminant method applied to 16x16 normalized images; a method relying on statistical and structural analysis of a piecewise-linear approximation of the contours; a structural classifier using information about the size and placement of nearly horizontal and nearly vertical strokes in the images; and a contour analysis method based on eight feature types defines in terms of the amount of curvature present at any point. A decision tree composed of 17 rules is used to combine the results of these 4 algorithms. Recognition and substitution rates of 95.54% and 1.99% respectively are obtained; for non-touching digits free of artifacts, these rates climb to 97.10% and 0.96% respectively.

Kimura & Shridhar [73] combine the outcomes of 2 methods: the first is a modified quadratic discriminant function based on local histograms of 4-valued chain codes extracted for 16 rectangular zones of the images; the second is a tree classifier based on structural features extracted from the left and right profiles of the numerals: left and right peaks, location of minima and maxima etc. Recognition/substitution rate pairs ranging from 96.23% and 0.25% to 89.55% and 0.07% were obtained.

In Suen et al. [157], the 4 structural methods labeled as 'Expert E1', ..., 'Expert E4' and already presented in section 2.1.1 were combined using several variants of the simple majority voting scheme. With the same variant as that discussed under Lam

& Suen [80] in the preceding subsection, the average recognition/substitution rates achieved were 86.68% and 0.07% for all combinations of 2 of the 4 experts; 95.45% and 0.26% for all combinations of 3 of the 4 experts; and 93.05% and 0.00% for the combination of all 4 experts.

Using the same 4 experts and the same test set of 2 000 samples, Xu et al. [170] provide a variety of results for combination methods based on Dempster-Shafer formalism, Bayesian formalism, and the voting principle as presented in the preceding subsection. Most of these combination methods are parameter-dependent and only some of the best results will be discussed here. Using the first half of the test set to derive recognition, substitution and rejection rates for each method, they obtained recognition and substitution rates of 95.00% and 0.00% respectively on the other half of the test set. For the Bayesian formalism, using the confusion matrices of the first half of the test set, results of 91.5% and 0.30% respectively are obtained; averaging the results of 200 trials, leaving 10 samples out and using the other 1990 to derive the confusion matrices, recognition and substitution rates of 94.15% and 0.60% are achieved. For the voting principle, recognition and substitution rates of 97.95% and 0.35% and of 95.45% and 0.05% are possible for the last 2 variants presented above. The authors note that methods based on Bayesian formalism tend to degenerate unless the confusion matrices are well-learned. Dempster-Shafer and voting based methods are more robust when dealing with incomplete information.

Huang & Suen [67] also provide results for combining the same experts on the same database using their Behaviour-Knowledge Space method. Because of insufficient learning data, the results are not for a pure BKS application but for a mixed BKS-Bayesian combination method. Best recognition and substitution rates given (for certain values of a free parameter) are 97.65% and 0.10% respectively or 95.60% and 0.05%.

Using the CENPARMI database once again, Franke [49] provides results of combining the 4 CENPARMI experts with some of his best polynomial classifier methods labeled 'E5' and 'E6' by a simple voting method. 'E6' is identical to classifier CQ22 adapted to SHAPE_N_SW13 as presented in section 2.2.1. 'E5', also labeled 'MEAN_N' elsewhere, is a classifier net of 45 (2x2) classifiers adapted to each pair

of the 10 numeral classifiers. Results of combining 2, 3, 4, 5, and 6 classifiers are given. If minimal substitution rates are wanted, the best combination of 4 experts (E2, E3, E4, and E6) gives recognition and substitution rates of 95.65% and 0.00% respectively; the best combination of 5 experts (E1, E2, E3, E4, and E5) gives 97.40% and 0.05%; the combination of all 6 experts does not yield better results in this case: 97.00% and 0.05% respectively.

Lee & Srihari [87] present the results of 7 different classifiers and of 5 methods of combining their outputs for their own CEDAR¹⁵ database. The methods are:

1. Binpoly: a binary polynomial classifier using 1240 first- and second-order features;
2. Histogram: a two-layer feed-forward MLP receiving as input a histogram-based feature vector of dimension 72;
3. Gabor: a neural network of 104 input, 100 hidden, and 10 output units using as input the 52 complex coefficients obtained from Gabor feature extraction;
4. Gradient: a neural network of 192 input, 101 hidden, and 10 output units using as input gradient-based features extracted using Sobel operators;
5. Morphology: a neural network of 85 input, 50 hidden, and 10 output units using morphological features as input; for each of 9 non-uniform sub-regions of the images, 9 components are obtained based on the area of concave regions, the length of stroke features, and the number of endpoints and crosspoints; four global features (moments and number of holes) complete the input;
6. Chaincode: a neural network of 640 input, 100 hidden, and 10 output units using as input, for each of 16 equal-sized regions of the image, the percentage of boundary pixels with a particular slope (8 values) and a particular curvature (5 values);
7. GSC: a 6-NN classifier with a voting scheme using input from 3 separate feature generators: the 192 inputs of method #4 above; 128 inputs from extracted

¹⁵ Center of Excellence for Document Analysis and Recognition.

strokes and corners based on a gradient map; 128 other inputs from the extraction of coarse concavities in 4 directions, holes and large scale strokes.

The five methods of combination considered are: Bayesian; neural net using outputs of all 7 classifiers as input; logistic regression; fuzzy integral; and simple majority voting. On the same test set¹⁶, the numbering of the methods above corresponds to their individual ranking, with no-rejection recognition rates ranging from 96.12% for #1 to 98.09% for #7 (GSC). The no-rejection recognition rates for the combination of methods range from 98.04% (fuzzy integral) to 98.52% (logistic regression). Once more, the simple majority voting scheme performed very well, being the second best with 98.43%.

Sabourin et al. [139] investigate practical ways of speeding up k-NN (Nearest Neighbour) classifiers on the large (250 000 isolated digits) NIST-3 database. Pruning substantially reduces the size of the pattern reference set; and search optimization by the triangle inequality, using so well-chosen anchor points also helps a lot. It turns out that both methods improved the query time by a factor of 80! Using these techniques, the authors study combinations of Nearest-Neighbour classifiers based on 2 different feature sets: 1) tangents uniformly sampled on the character chain-coded contour; 2) Zernicke moments¹⁷. Requiring unanimity of decisions for both k-NN classifiers with $k = 1$ results in a recognition rate of 96.28% with an error rate of 0.20%. For $k = 4$, both rates drop to 86.12% and 0.022

More complex combination schemes based on Bayesian and Dempster-Shafer formalism were implemented but better results were obtained with a dynamic classifier selection (DCS) algorithm and a single parameter combination. Based on selected classifier parameters forming a new “meta” pattern space, the DCS is also a nearest neighbour classifier which selects the classifier most likely to give the correct answer; 60 000 distinct digits are used to select the right parameters for the DCS. Recognition and error rates of 98.97% and 1.03% respectively were obtained in this way. Finally, a joint 5-NN classifier, where the most frequent class amongst the 5 nearest neighbours from both classifiers is chosen, yielded 99.34% recognition with 0.66% substitution.

¹⁶ Results given here are for the so-called BHA test set.

¹⁷ Of the 250 000 samples, 118 000 are used for training and 60 000 for testing.

Thien & Bunke [162] discuss a multi-expert and multi-stage method based on different perturbations to the image data. Classical recognizers labeled 'C1' and 'C2' are linearly combined to create classifier 'C3'. Starting from normalized 32x32 binary images, the first classifier computes 4 black pixel histograms from the image and another 4 from the skeleton, plus contour profiles from 8 directions; 5 features are extracted from each of these 16 functions. These 80 features plus the aspect ratio are fed to a Parzen estimate based classifier. The second method begins with 64x64 normalized binary images; directions at each (outer and inner) contour point are quantized in 16 values; for each of 9 overlapping regions, the number of pixels in 16 directions is counted, weighted based on the distance to the center of the region. The input vector of length 144 is fed to a distance-weighted 4-NN classifier. Classifier 'C3' linearly combines 'C1' and 'C2', assigning 25% more weight to 'C2' compared to 'C1'.

Furthermore, eleven perturbation types can be applied to original images: rotation in 1 direction, slant in 2 directions, perspective and shrink in 4 directions each. For each perturbation type, 4 discrete values are tried resulting in a total of 45 potential perturbations (including 1 case for unperturbed data).

Here is how the multi-stage system works: In the first stage, unperturbed data is fed to 'C3' with a very high rejection threshold; rejected data goes through the second stage where perturbations are limited to the first 2 discrete values and the outputs from these perturbed images and the unperturbed image are combined. Finally, still rejected samples undergo second degree perturbations whose outcome are combined with previous outputs to make the final selection. We note that a penalizing factor is applied which increases with the amount of perturbation. Results of this sophisticated method, labeled 'P2', on the so-called 'goodbs' data set of SUNY (Buffalo) are said to be the best published: 99.09% recognition with 0.91% substitution. The main drawback of this method is the processing time: 6.0 seconds per sample on a SPARC 10 computer.

Huang & Suen [66] conducted some experiments to compare Bayesian, Dempster-Shafer, BKS, and voting combination approaches on an ITRI¹⁸ numeral database

¹⁸ The Industrial Technology Research Institute is a government-sponsored research institution in Taiwan.

consisting of 46 451 samples. They trained 3 classifiers on only 5 074 samples, using the remaining 41 377 for testing. Consequently, the no-rejection error rates of the classifiers were fairly high: 9.63%, 9.07%, and 7.86% respectively. In these conditions, the BKS method performed best, followed by the Bayesian approach. Dempster-Shafer and simple voting performed almost the same, in third position. However, it should be noted that for substitution rates of 0.77% or less, all 4 approaches got similar performances. Thus the simple voting scheme would then be a very suitable choice.

Lam & Suen [81] studied the extension of the voting scheme by performing a weighted voting. Optimal weights are sought to maximize the function

$$F = \text{Recognition} - \beta \cdot \text{error} \quad \text{for } \beta = 10, 15, 20, 25, \text{ and } 30.$$

They are derived in two ways: a) using a Bayesian formulation, one weight per classifier and per class is obtained; b) using a genetic algorithm, one weight per classifier is obtained. Seven classifiers (from CENPARMI and ITRI) are combined, using the ITRI database, from which 24 427 samples are used for training the individual classifiers; the remaining 22 024 samples are further subdivided into set A (13 272 samples) and set B (8 752 samples). Set A is used to compute the optimum weights which are then applied to set B. Of course, the Bayesian- and genetic-based weighted voting methods can be trained to reach F -values on set A which are higher than what is achieved by simple majority voting. However, it turns out that for any reasonable trade-offs between the recognition and the error rates ($\beta \geq 4$), simple majority vote produces the best results on set B. It appears that finding enough data to train in a representative way is not so easy: even if samples were collected under the same conditions, training parameters on set A can result in “overfitting” which induces more errors on set B. The authors conclude that in the absence of a truly representative training set, simple majority voting remains the easiest and most reliable solution among the ones studied.

In Strathy & Suen [151], 3 very similar standard backpropagation networks, labeled A, B, and C are combined. Method A was described above and yielded a substitution rate of 3.69% (with no rejection). Method B is the same as A but was

not additionally trained on randomly sheared data (as was the case for A); its no-rejection error rate is 3.73%. Method C uses 12x12 normalized images (instead of 12x14, for methods A and B) and the network structure is also modified: 288 input, 80 hidden, and 10 output units (vs 336-70-10 for A and B); its no-rejection error rate is 3.87%. A simple 2/3 majority voting combination (choosing A's decision when all 3 classifiers disagree) achieves a slight improvement: a no-rejection error rate of 3.47%.

In Bottou et al. [20], 'LeNet 4' was one of 5 methods achieving a 1.1% error rate on the modified NIST database (see section 2.1.2). Following Drucker et al. [41] a committee of 3 'LeNet 4' machines was devised. The idea is to train a second machine on a mix of samples, half of which were correctly recognized while the other half was misclassified; and to train a third machine on samples for which the first two disagree. Enormous amounts of data are needed for this training which are not available given the already excellent performance of 'LeNet 4'. The problem is solved by generating deformed training data with a set of affine transformations and line-thickness variations. The combination is by adding the scores of the 3 machines and choosing the class with the highest total score. This 'Boosted LeNet 4' combination achieved a no-rejection substitution rate of only 0.7%. When the first machine classifies with high confidence, the other 2 machines need not be evaluated which increases computation time by a factor of 1.75 (instead of 3).

Gader & Khabou [55] study the combination of up to 4 neural networks by simple average of their outputs; the best combination is for only two MLPs. The first uses automatically generated features (previously described) and the second is a shared weight network based on Le Cun et al. [83]. This combination achieved recognition and substitution rates of 98.3% and 1.7% respectively, or of 97.4% and 1.0%, on the CENPARMI test set.

Cho [32] compares 3 different methods for combining the outcomes of 3 two-layered MLPs. The first MLP uses size-normalized 16x16 images, compressed to 4x4, as input; the second MLP uses as input four 4x4 clusters of directional features obtained using Kirsch-like masks; and the third MLP uses as input 15 complex Fourier descriptors from the outer contours, plus simple topological features from the inner contours. The three combination methods are simple majority voting, averaging of the separate

networks, and fuzzy integral. The recognition and substitution rates achieved on the CENPARMI test set for each combination method are: 96.70% and 3.05% (voting); 97.15% and 2.35% (average); 97.35% and 2.30% (fuzzy integral).

2.3 Comparative Results

A discussion of important factors to be weighed when comparing recognition results and a compilation of some of the best results published in the literature up until the early 1990's can be found in Suen et al. [157]. However, as should be apparent from the preceding discussion, research has intensified a lot since that time and a large number of excellent systems have been produced. It is thus necessary to return on the topic once more...

2.3.1 Some Guidelines for Comparison

Raw percentage figures providing recognition, substitution and rejection rates of various systems on different databases should NOT be compared without much caution. A system obtaining a higher recognition rate than another is NOT necessarily 'better' and the one obtaining the highest is NOT necessarily 'the best'.

Before making comparisons, one should first have a good understanding of the requirements of the targeted application.

- What kind of data will the system have to deal with? Totally unconstrained in terms of writing conditions and instruments or somewhat constrained? Written by the general public or by a specialized sector of the population? Writers from a specific region or country or from anywhere on the planet? Etc... As much as possible, system performances should be compared precisely on the kind of data that will have to be dealt with. Drawing conclusions otherwise may be risky since system performance can be data-dependent to a large extent according to our own experience (see section 2.4 on machine vs human performance). Unfortunately, there are very few accounts in the literature of the performance

of recognition systems on databases that are very different from the ones used for training.

- What error level is tolerable for the application? Or, put in another way, what is the relative cost of an error and that of a rejection? When errors are not costly at all, one may consider having all data classified by machine, dealing with errors afterwards. However, in many applications, it may be preferable to reject an item and have it classified by human operators. Here a very low error rate is critical while the highest recognition is not so important.

Some measures of performance take this last factor into account such as the function $F = \text{Recognition} - \beta \cdot \text{error}$ in Lam & Suen [81] or an equivalent formulation in Matsui et al. [111]. Several recognition schemes are ‘adjustable’ in terms of the error/rejection trade-offs. In such cases, the best way to present performance is offer the error vs rejection curve. Unfortunately, this is often not provided: authors are often content with quoting system performance at 0% rejection; while this may offer some common basis for comparison, it is not always of practical interest for real-life applications.

Ideally, comparisons are most significant when results are available on common, large, uniform and representative databases. In the next sections, we will present such databases and results which have become available in the past few years. However, a number of databases do exist for which results have already been published; and more will no doubt come into existence in the near future. When trying to assess results, the following data-related and processing-dependent factors should always be weighed carefully:

- How large is the database?
- Is the data really unconstrained? What relative level of difficulty does it offer in terms of writing styles, conditions, etc?
- Is the data from a large number of different writers?
- Are the training and testing sets really distinct i.e. not containing samples from the same writers?

- Is the test set balanced i.e. containing the same number of samples from the 10 numeral classes?¹⁹
- What is the resolution of the data images?
- Was the data manually or automatically segmented when overlapping, touching, or even crossing each other?²⁰ Was any data discarded from the test set as a result?

One last word of caution is warranted. When test set results for parameter-dependent combinations of methods are given for several values of the parameter(s), one should not draw quick conclusions about the best value of such parameter(s). The test set is then being used somewhat as a training set to find the optimum parameter value(s) and this should normally be confirmed on other independent data sets—which is rarely the case.

2.3.2 Some Published Results on CENPARMI Database

One of the first databases to be used for comparative purposes is our own CENPARMI database, composed of approximately 17 000 isolated numerals. For a summary of technical information concerning this database, see Appendix A.2.

Results of individual recognition systems on the CENPARMI database are presented in Table 1. All figures are percentages and relate to test set T. *Reliability* is defined as:

$$Reliability = \frac{Recognition}{Recognition + Substitution}. \quad (4)$$

It is important to recall that all methods were trained on sets A and B, except for Experts 'E5' and 'E6' which were trained on a huge proprietary database of AEG

¹⁹ When databases contain a larger proportion of easier digits (such as 0's and 1's), performance statistics can be artificially inflated, unless of course such imbalance is typical of the targeted application.

²⁰ Machine and even human recognition performance will degrade on automatically segmented data.

Reference	Recog.	Substit.	Reject.	Reliab.
Krzyzak et al. [75]	86.40	1.00	12.60	98.85
Krzyzak et al. [75]	94.85	5.15	0.00	94.85
Nadal & Suen [117] (Expert E1)	86.05	2.25	11.70	97.45
Lam & Suen [82] (Expert E2)	93.10	2.95	3.95	96.98
Mai & Suen [105] (Expert E3)	92.95	2.15	4.90	97.74
Legault & Suen [94] (Expert E4)	93.90	1.60	4.50	98.32
Franke [46] PIQ1079_NB_SW13	94.55	1.25	4.20	98.70
Franke [49] Expert E5	96.95	3.05	0.00	96.95
Franke [49] Expert E6	98.30	1.70	0.00	98.30
Lee et al. [91]	97.80	2.20	0.00	97.80
Lee & Kim [90]	97.30	2.70	0.00	97.30
Lee et al. [89]	96.80	3.20	0.00	96.80
Gader & Khabou [55]	96.50	3.50	0.00	96.50
Gader & Khabou [55]	93.00	1.55	5.45	98.36
Cho [32]	96.55	3.45	0.00	96.55

Table 1: Results of Individual Systems on CENPARMI Data

Daimler-Benz composed of over 1 000 000 digits.²¹ Of those trained on the CENPARMI database, the incomplete quadratic polynomial classifier PIQ1079_NB_SW13 of Franke is probably offering the best high recognition and high reliability performance; the methods of Lee et al. achieve higher recognition rates but at the expense of a substitution rate which is almost twice that of Franke's method. Of course, training on a huge database does make a difference: Expert 'E6' is as reliable as 'E4' with an increase of 4.4% in the recognition rate.

Results of multi-expert combinations on the CENPARMI database are presented in Table 2.

Again caution should be exercised in the interpretation of these results. For instance, a substitution rate of 0.05% represents 1 sample out of 2 000 and such a figure cannot be said to be statistically significant. However a very clear tendency is observed; namely that the combination of different methods produces a dramatic reduction of substitutions while still allowing a very high recognition rate. In fact,

²¹ Results of 'E5' and 'E6' are given here to clarify the next table.

Reference	Combination Method	Recog.	Substit.	Reject.	Reliab.
Suen et al. [158] (E1 ... E4)	Voting	93.05	0.00	6.95	100.00
Suen et al. [157] (avg. 3 of 4)	Voting	95.45	0.26	4.29	99.73
Xu et al. [170] (E1 ... E4)	Dempster-Shafer	95.00	0.00	5.00	100.00
	Bayesian	94.15	0.60	5.25	99.37
	Voting	97.95	0.35	1.70	99.64
	Voting	95.45	0.05	4.50	99.95
Huang & Suen [67] (E1 ... E4)	BKS + Bayesian	95.60	0.05	4.35	99.95
	BKS + Bayesian	97.65	0.10	2.25	99.90
Franke [49] (E2-E3-E4-E6)	Voting	95.65	0.00	4.35	100.00
Franke [49] (E1 ... E5)	Voting	97.40	0.05	2.55	99.95
Franke [49] (E1 ... E6)	Voting	97.00	0.05	2.95	99.95
Gader & Khabou [55] (2 MLPs)	Output averaging	97.40	1.00	1.60	98.98
Cho [32] (3 MLPs)	Fuzzy integral	97.35	2.30	0.35	97.69

Table 2: Results of Combinations of Systems on CENPARMI Data

the final recognition rate can often exceed even the highest figure attained by any of the individual experts. Of course, the results are for this particular test set, which is rather small, and there is no guarantee that the ranking of the methods or the quantitative gains of the combinations would be the same for another database.

2.3.3 Some Published Results on CEDAR Database

CEDAR, the Center of Excellence for Document Analysis and Recognition (Buffalo, N.Y.), also has made a database of handwritten digits available and several researchers have published recognition results on it. The database consists of 21 179 binary images of digits extracted from U.S. ZIP codes. For a brief description of this data, see Appendix A.1.

Table 3 presents some of the published statistics on this database, the horizontal line separating results of single methods from results of multi-stage and multi-expert systems. Note that the *goodbs* test set is a subset of 2 213 well-segmented samples extracted from the 2711-sample *bs* test set. The methods of Revow et al. are described in section 2.1.3; those of Lee & Srihari, of Thien & Bunke, and of Strathy & Suen

Reference	'bs' Data Set			'goodbs' Data Set		
	Rec.	Sub.	Rej.	Rec.	Sub.	Rej.
Lee & Srihari [87] Binpoly	93.99	6.01	0.00	96.43	3.57	0.00
Lee & Srihari [87] Histogram	94.95	5.05	0.00	97.47	2.53	0.00
Lee & Srihari [87] Gabor	95.28	4.72	0.00	97.70	2.30	0.00
Lee & Srihari [87] Gradient	96.42	3.58	0.00	98.46	1.54	0.00
Lee & Srihari [87] Morphology	95.98	4.02	0.00	97.92	2.08	0.00
Lee & Srihari [87] Chaincode	96.39	3.61	0.00	98.33	1.67	0.00
Lee & Srihari [87] GSC	97.05	2.95	0.00	98.87	1.13	0.00
Revow et al. [132]	96.57	3.43	0.00	98.47	1.53	0.00
	96.86	3.14	0.00	98.50	1.50	0.00
Thien & Bunke [162] C1				97.69	2.31	0.00
Thien & Bunke [162] C2				98.19	1.81	0.00
Strathy & Suen [151] A	96.31	3.69	0.00			
Strathy & Suen [151] B	96.27	3.73	0.00			
Strathy & Suen [151] C	96.13	3.87	0.00			
Hinton et al. [60]	95.32	4.68	0.00	97.83	2.17	0.00
Franke [46] KH30F2ZTG				97.02	0.90	2.08
				98.33	1.67	0.00
Strathy & Suen [151] ABC(voting)	96.53	3.47	0.00	98.64 ²²	1.36	0.00
Thien & Bunke [162] C3(voting)				98.51	1.49	0.00
Thien & Bunke [162] P2				99.09	0.91	0.00

Table 3: Some Published Results on CEDAR Data

under 'Practical Results' in section 2.2.2; the KH30F2ZTG method of Franke in section 2.2.1. Note that the latter was trained on 100 000 digits totally independent of the CEDAR database. Also note that for results of Lee & Srihari the BS dataset is really more a validation set than a test set²³. Results on an independent test set are given under 'Practical Results' in section 2.2.2.

It should be noted that the *goodbs* test data set is not a balanced set. It contains a total of 2 213 samples, including 355 zero's, 289 one's, 245 sixe's, but only 117 five's, 180 nine's, 183 four's. Hence the global recognition figures presented above are probably higher than would be achieved on a balanced test set.

²³ The BS images were cross-tested several times in developing algorithms and in determining neural network convergence.

2.3.4 The NIST Competitions

In May 1992, the National Institute for Standards and Technology (U.S.) held the First Census Optical Character Recognition Systems Conference [165]. The event was organized to assess the state of the art in OCR, to learn what are the current limiting factors and to find out if the collection of new databases of handprinted characters could help further developments. It focused on a single step of document processing: machine recognition of individual, already segmented, characters (digits, plus uppercase and lowercase letters) without context. Twenty nine groups²⁴ from North America and Europe responded to the call for participation and they received training data in February 1992. Later they received the test data and only 3 groups did not submit test results and hence did not participate in the conference in May.

The training material²⁵, labeled NIST Special Database 3 (SD3), came from forms filled out by 2 100 permanent Census Field workers, as part of the 1990 Census program. It included 223 122 digits. The testing material, labeled NIST Test Data 1 (TD1) came from forms filled out by 500 math and science high school students. It included 58 646 digits.

For digits, NIST partitioned the TD1 set in 10 and results were provided in terms of the mean and the standard deviation of the no-rejection error rate calculated over these 10 partitions. About half of the systems recognized more than 95% of the samples. Most systems in the report combine several methods for preprocessing/filtering and feature extraction. All four combinations of rule-based and learning-based feature extraction and classification were represented at the conference and each yielded at least one low error rate system. But most common was some sort of math-based feature extractor with a multi-layer perceptron (MLP) used for classification. As for speed considerations, some methods used PC platforms and processed only 1 character per second; at the other end of the spectrum, other systems used parallel machines or dedicated hardware and processed up to 1 000 characters per second.

The 10 best results are presented in Table 4. All values represent percentages.

²⁴ 18 from private industry and 8 from universities and national research institutes.

²⁵ Participants could also choose to use their own training material or a combination of SD3 and their own.

Participating group	System label	No-rejection Error Rate
OCR Systems Inc.	OCRSYS	1.56 ± 0.19
AT&T Bell Laboratories	ATT_1	3.16 ± 0.29
ELSAG BAILEY INC.	ELSAGB_3	3.35 ± 0.21
ELSAG BAILEY INC.	ELSAGB_2	3.38 ± 0.20
AEG Electrocom GmbH	AEG	3.43 ± 0.23
IBM Almaden Research Center	IBM	3.49 ± 0.12
AT&T Bell Laboratories	ATT_2	3.67 ± 0.23
Thinking Machines Corp.	THINK_2	3.85 ± 0.33
Environmental Res. Inst. of Michigan	ERIM_1	3.88 ± 0.20
Environmental Res. Inst. of Michigan	ERIM_2	3.92 ± 0.24

Table 4: Top Ten Results for Digits at First NIST Conference

We will now briefly summarize the key aspects of the top 5 contenders. OCRSYS uses convolutions with hand-coded filters as features and a multi-layer perceptron (MLP) for classification. ATT_1 uses a gray level rescaled image directly as features and a k-NN classifier with a special distance measure compensating for common distortions. ELSAG_3 performs noise removal and size normalization to 24x36; features are shape functions of the character bit maps having the same size as the character; classification is in 2 stages: first a k-NN classifier using references representing clusters of shape functions in the training samples, followed by the same classifier using a more sophisticated distance measure and many more references. The description provided for ELSAG_2 is not apparently different from that of ELSAG_3. Preprocessing for the AEG entry includes their familiar normalization for size, stroke width and slant; a 256-component feature vector is obtained from KL-transform and an adaptive polynomial classifier is used.

Participants pointed out that SD3 did not constitute very appropriate training material to prepare for the TD1 test set. Indeed, cross validation studies performed after the competition indicated that TD1 was more diverse and more general. Mostly this is because the 2 100 Census Field Workers who provided the SD3 data were motivated and careful subjects, more so than the 500 high school students who wrote

the TD1 data²⁶. Furthermore, different segmenters were used for SD3 and TD1. The segmenter utilized on the training data failed on a much larger fraction of samples than the other; hence more of the difficult data made it into TD1

The low quality of SD3 as training material is confirmed in 2 other ways. OCRSYS, the top entry in the conference, has a substitution rate which is barely one half that of its nearest contender; and it was *not* trained with SD3 at all but with a proprietary database. Similarly, the AEG entry, when re-trained with their own proprietary database, saw its substitution rate reduced from 3.43% to 2.9%.

The major conclusion of the First Census Optical Character Recognition Systems Conference was that

“The state of the art of machine OCR of segmented, hand-printed digits is approaching human performance with respect to the zero-rejection-rate error rate. The results for upper case letters and lower case letters are probably not as good relative to human performance as the performance for digits, but no human classifications under the conditions of the Conference test have been conducted to address (the) question.” (see [165], p. 12).

In the introduction to the Second Conference report this statement was inflated to the following:

“An important conclusion of the First Conference was that the OCR of isolated (properly segmented) characters was essentially a solved problem.” (see [57], p. 5).

Perhaps the key is in the words ‘*properly* segmented’.

The report to the Second Census Optical Character Recognition Systems Conference was published in May 1994. The conference focused on a much broader and realistic recognition task: reading answers from digital images of forms scanned from microfilm and from paper. Initially, a large sample of 12 500 digital miniforms (containing 37 500 answer fields), scanned from microfilm copies of a non-sensitive portion

²⁶ The latter are probably more representative of the general public however.

of the Industry and Occupation section of the 1990 Census long form, was to be the only basis for the competition. But image quality was far inferior to that which could be obtained by scanning original paper forms. So another training CD-ROM was prepared consisting of 6 000 miniforms (18 000 answer fields) scanned from microfilm *and from paper*. The complete OCR task was considered: document handling, scanning, form identification, field isolation, character segmentation, recognition, and context-based field correction. Several dictionaries were provided with training materials to allow for this last task.

The focus of the second conference was indeed much more difficult. Twenty five groups responded to the call for participation but only ten submitted test results (some late) and attended the meeting in mid-February 1994. Results indicate that at field rejection levels between 40% and 60% several systems achieved accuracy levels that exceeded human performance levels at the 0% rejection level. More precisely, at the 60% field rejection rate, the top 4 systems²⁷ had field error rates of 3.6%, 3.9% (ERIM, Michigan), 10.7% (IDIAP, Switzerland), and 13.8% (CGK mbH, Germany) respectively. At 0% field rejection rate, the corresponding figures were 39.7%, 41.9%, 52.6%, and 50.5%. Systems not performing segmentation were not among the most accurate. Furthermore, except for NIST's own system, all systems attempting segmentation used intentional over-segmentation and the best ones had sophisticated means to recombine character fragments before dictionary-based correction.

2.3.5 The IPTP Competitions

The Institute for Posts and Telecommunications Policy (IPTP) in Japan also held 2 competitions in 1992 and in 1993. Matsui et al. [111] report on the results of the First IPTP Character Recognition Competition. The data consisted of 200 PPI binary numerals extracted from 3-digit postal codes written freely (but within frames). The training set had 7 500 digits and the testing set 29 883 digits (unbalanced, but not too biased). Thirteen recognition programs were submitted by 5 universities and 8 manufacturing companies.

²⁷ Their performance was markedly superior to the average.

The weighted sum $S = (10 \cdot \text{Errorrate}) + \text{Rejectionrate}$ was used as criterion for evaluation. The best individual system scored 96.22% recognition and 0.37% substitution, for an S -value of 7.09. The top 3 algorithms, labeled A, B, and C, all use position and scale normalization; pattern descriptions are by contour points (A) and contour chain code (B and C). For features, A uses crossing counts, topological features and contour features; B uses peripheral white run-lengths and line-direction histogram; C uses weighted line direction histogram. For classification, A uses subjective clustering and exhaustive decision trees; B uses fuzzy clustering and neural computing; C uses a modified quadratic discriminant function.

The authors then investigate several multi-expert combinations of the top 3 contenders including voting, minimal sum of rankings, and some scheme similar to Huang & Suen's BKS method, labeled "candidate appearance likelihood rule". For the latter, an S -value as low as 1.53 (recognition and substitution rates of 99.85% and 0.15% respectively) was achieved by considering a 9-component vector composed of the 2 most probable classes and a rejection flag as determined by each method.

Noumi et al. [121] report on the Second IPTP Character Recognition Competition. The training set consisted of one half of the first competition's test set plus 9 500 new 3-digit ZIP codes; the testing set consisted of the other half of the first competition's test set plus 5 000 new 3-digit ZIP codes. Fourteen recognition programs from universities and manufacturing companies were submitted (including 2 foreign participants). This time the best entry scored an S -value of 3.86 (97.94% recognition, 0.20% substitution, and 1.86% rejection). This time, the top 2 recognition methods were actually multi-expert systems.

2.4 Machine vs Human Performance

We begin this section with 2 quotes expressing opposite views about the comparison between machine and human performance in recognizing handwritten digits.

"Humans are no longer clearly better than machines at the recognition of isolated characters, and certainly not in any economically significant way." (see Geist et al. [57], p. 16)

“However, handwritten character recognition is still a difficult task in which human beings perform much better than any computer systems.”
(see Chi et al. [31], p. 59)

In our view, the true picture is probably somewhere in the middle: there are aspects on which humans still clearly outperform machines and others where the opposite is true. In a nutshell, we would say that human recognition is still clearly more robust and reliable while machine recognition can definitely be faster. And it is true that for many applications, a combination of both can already offer clear economic advantages compared to human recognition alone.

However, there are few in-depth analyses of the relative performances of humans and machines at recognizing digits. Occasionally, statements are made in some papers without evidence to substantiate them. For example, LeCun et al. [84] seem to postulate that humans must reject 5% of samples to achieve a 1% error rate on the kind of data used for their system; Bottou et al. [20] state a much lower estimate of 0.2% error rate without any rejection, for their modified NIST database. In this section, we will present evidence to support our own claims. First, we will examine the conclusions of the NIST conferences on this aspect, since their goal was to assess the state of the art in OCR.

2.4.1 About the Conclusions of NIST Conferences

The report on the first NIST conference [165] mentions that 10 000 samples from the TD1 testing set were presented to one human subject, at the rate of about 1 character per second, for periods of 10 minutes to over an hour with breaks of several hours between periods. This experiment yielded a zero-rejection error rate of 1.57% which is almost identical to the 1.56% figure achieved by the OCRSYS machine recognition method.

In the report of the second NIST conference [57], it is explained that after the first conference a technician classified all digits of TD1 in a 2-pass experiment. In the first pass, all samples which could not instantly be classified with confidence were to be rejected; this is reflexive classification and Geist labels it as ESHC (economically

significant human classification). This first pass provided a 3.34% rejection rate with a 0.28% error rate. In the second pass, only the rejected samples from the first pass were processed; the subject could take his time and decide to classify or to reject the samples again.

The overall result was a 0.56% rejection rate with a 0.35% error rate. One year after the first NIST conference, Mitek and CGK, two companies that had not taken part in the competition, sent results achieved on TD1 by their systems, which outperform all previous submissions. Still, for very low error rates, the human error rate²⁸ was more than 2 times lower than the lowest machine result (Mitek). At rejection levels of 5% and 10% respectively, the Mitek and the CGK systems produced error rates that were less than that of ESHC (reflexive). On this basis, Geist argues that processing a set of isolated handprint digits by machine and having humans reclassify a low confidence subset should produce a lower error rate at less cost than conventional ESHC on the same set.

Of course, machine recognition has the advantage of speed. The Mitek system, for example, processes 15.5 characters per second while the human experiment was conducted at a much slower pace. LeNet1 [83] with its single-board hardware can attain 1 000 characters per second! But what is said above tends to support our claim about higher reliability of human recognition. There are important additional comments to be made:

- The human experiments described above are very limited (2 subjects only); and already there are differences in the procedure followed between the 2 experiments and the results obtained. Experiments with a larger number of subjects would be needed for a serious comparison. But there is a more important flaw in the way the comparison is carried out.
- The comparison is unfair to human recognition because it forces human subjects to play the game on the machine's home ground: machines need isolated digits for their recognition programs but humans do not read multi-digit strings, digit by digit independently. For this reason, while humans can at times confuse a

²⁸ Estimated from a 'composite' of the 2 human subject experiments.

'4' for a '9', they are much less likely than a machine to confuse a '49' for a '44', or a '94', or a '99'. By cutting the context away, we are not evaluating full human potential for this task.

- Moreover, not only were the digits segmented but they were *machine*-segmented. Thus poor machine segmentation may induce what will be credited as human errors in the experiments. On this topic, Martin & Pittman [109] provide the following evidence:

“An independent person categorizing the test set of pre-segmented size-normalized digits achieved an error rate of 3.4%. This figure is considerably below the near-perfect performance of operators keying in numbers directly from bank checks, because the segmentation algorithm is flawed.” (p. 406)

In summary, for a fair comparison between machine recognition and ESHC, machines should deal with their own segmented digits and humans with the original digit strings.

2.4.2 Human Recognition is More Reliable

In many applications, errors are potentially very costly. In some cases, it may be so critical for recognition systems to achieve extremely low error rates that rejecting a sizeable portion of the processed data represents a sound choice. For example, for the processing of French Postal Service checks, Gilloux [58] argues that

“... a confusion rate not higher than 0.01% would be acceptable and compatible with that of human readers while a recognition rate not less than 50%²⁹ would ensure the usefulness of automatic reading.” (p. 29)

Thus machines would only deal with the easy half of the data, leaving the other, more difficult, half to human operators. This is clearly an admission that human recognition is more robust and reliable... In this section, we will offer various pieces

²⁹ These percentages are on complete check amounts, *not* individual digits.

of evidence supporting that claim. We will first summarize the findings of an experiment we conducted with some colleagues to compare human and machine performance on very confusing samples. Secondly, we will examine the available information concerning required rejection levels for machines to perform at particularly low error rates. Thirdly, we will present our qualitative assesment of samples misrecognized by various OCR methods. And finally, we will give an account of what happens when methods trained on a particular database are applied to another database without re-training.

An Extensive Experiment

Here we consider an extensive experiment conducted by Legault et al. (see [98] and [99]). The authors selected 360 of the most confusing samples from the CENPARMI database and presented them to human subjects for identification. The subjects (5 OCR researchers and 4 paid volunteers) were asked to distribute a total score of 100 among 11 boxes³⁰. They also were required to write for each sample the key factors which motivated their score distribution, trying to rank their arguments in decreasing order of importance. Human recognition was investigated in detail, but more important to us here, it was also compared to the performance of the 4 ‘Experts’ (E1, E2, E3, and E4) of Suen et al. [157] previously discussed and to their voting combination.

For the group of 5 OCR researchers, for each sample, the average score in each box was computed. Then let S_1 and S_2 be respectively the highest and second highest score, associated with class c_1 and c_2 , and let S_{nil} be the average score in the ‘not a digit’ box for the same sample. The sample was considered as *unambiguously classified* into class c_1 , if the following condition was satisfied:

$$(S_1 \geq 2.5 \cdot S_2) \text{ and } (S_1 \geq 1.75 \cdot S_{nil}). \quad (5)$$

Otherwise, the sample was considered as rejected. For 22 samples, the unambiguous classifications so determined were different from the original labels of the samples and were taken as their new truth value.

³⁰ Representing the 10 numeral classes, plus a ‘not a digit’ box.

For the 360 most confusing samples of the CENPARMI database, Table 5 provides recognition performance for the methods E1, E2, E3, and E4, as well as for their combination through majority voting; under the horizontal line, the corresponding statistics for both groups of humans are also given for comparison. All figures are percentages.

System/Group	Recognition	Substitution	Rejection	Reliability
Expert E1	44.2	19.2	36.6	69.7
Expert E2	56.1	31.4	12.5	64.1
Expert E3	55.0	19.4	25.6	73.9
Expert E4	54.2	17.2	28.6	75.9
E1,E2,E3,E4 (Voting)	47.8	6.4	45.8	88.2
Paid Volunteers	46.4	6.1	47.5	88.4
OCR Researchers	59.2	0.0	40.8	100.0

Table 5: Machine and Human Performance on 360 Most Confusing Samples

On this most confusing data, the relative ranking of the machine experts in terms of *reliability*, from highest to lowest, is E4, E3, E1, and E2. This is in keeping with the rankings on testing set T (see section 2.3.2). In terms of the *recognition rate*, the present ranking of the methods, from highest to lowest, is E2, E3, E4, and E1; on test set T, it was E4, E2, E3, and E1. But in both situations, we note that E2, E3, and E4 have quite close recognition rates.

As was the case for testing set T, the voting combination of E1 through E4 improves performance dramatically. The combined recognition rate, which was between the individual rates of E2 and E3 on test set T, is now only slightly better than that of E1. But in striking counterpart, the combined error rate is only one third that of E1! Moreover, the performance of the 4-expert voting scheme is roughly at par with that of the 4 human volunteers. This further validates the concept of multi-expert recognition systems for unconstrained handwritten data. However the authors caution against drawing the hasty conclusion that human performance has now been matched.

First, the machine results are clearly inferior to those of OCR researchers. But

also a more detailed examination reveals some shortcomings as compared to even the group of volunteers.

Based on the unambiguous classification criterion presented above, we can say that the group of OCR researchers has unambiguously recognized 213 of the 360 samples and has rejected the other 147. Consequently, we would expect, for both the combination of machine experts and for the group of human volunteers, a relatively better performance on the first subset as compared to the second.

Table 6 gives the percentage of those samples which are recognized, substituted and rejected by the volunteers and the machine. It is clear from Table 6 (a) that the expected improvement in performance is present for the volunteers but not for the 4-method combination, despite the relative sophistication of the member algorithms. We see that the machine can produce substitutions for samples whose identity is unambiguous to humans. Also the machine cannot differentiate between the samples which humans identify unambiguously and the samples which they find *confusing*.

	Rec.	Sub.	Rej.
Volunteers	70.0	0.5	29.5
Machine	49.3	6.6	44.1

(a) 213 easier samples

	Rec.	Sub.	Rej.
Volunteers	12.2	14.3	73.5
Machine	45.6	6.1	48.3

(b) 147 more difficult samples

Table 6: Performance Results for Subsets of Original 360 Database

The authors then examined more carefully the subset of 147 samples rejected by the OCR specialists to further distinguish between 2 categories which they labeled “confused” and “unrecognized”. To this end, they expanded the rule of equation 5 as follows:

If ($S_{nil} \geq 50$) then “Unrecognized”
 Else if ($S_1 < 2.5 \cdot S_2$) then “Confused”
 Else if ($S_1 \geq 1.75 \cdot S_{nil}$) then “Recognized”
 Else if ($S_2 \geq 10$) and ($S_1 + S_2 > S_{nil}$) then “Confused”
 Else “Unrecognized”

(6)

When the above rule is applied to the OCR specialists' group scores, 48 of the samples are considered *unrecognized* and 99 are *confused*. These 99 *confused* samples can be regrouped into 20 confusing pairs.

Of the 48 samples *unrecognized* by OCR specialists, only one third are also rejected by the machine experts' combination; another third are seen as *confusing* (algorithms disagree about their identity); and the last third are classified uniquely (12 correctly and 4 incorrectly). On the other hand, the group of volunteers agree much more with the specialists: they score 42 of the 48 samples as *unrecognized* as well.

Of the 99 samples *confusing* to the group of 5 OCR specialists, only one quarter are also confusing to the machine combination; another quarter are *unrecognized*; and more than one half are uniquely classified. For the samples which are *confusing* to both the OCR specialists and the machine combination, the confusion is around the same pair of classes in only 35% of the cases.

The above shortcomings of the multi-expert system are inherited from the individual experts which compose it. They can be overcome only by bringing more refinement into each (or at least some) algorithm.

Required Rejection Levels for Low Error Rates

We now examine the ability of individual methods and combination systems to perform at very high reliability levels. More precisely, what rejection levels must be tolerated when an application requires error rates lower than 1%, or lower than 0.5%, or lower than 0.1%?

Systematic information on this question is not easy to come by. In a few articles, authors produce error *versus* rejection curves for their methods (if they can be so tuned, of course). But often they will only provide performance statistics at a couple of error levels. In the worst case, we can only obtain the forced recognition error rate (i.e. at 0% rejection) which is useless to answer our present concern...

Table 7 presents gathered information concerning required rejection levels to achieve very low error rates for various individual methods (above horizontal line) and system combinations (below horizontal line). The methods and combinations have all been summarized earlier in this chapter. All figures are percentages.

Reference	Method or Combination	Rejection	Substitution
LeCun et al. [84]	MLP chip (local receptive fields)	12-13	1.0
NIST #1 [165]	Best participating method	15.0	0.3
	Best participating method	30-35	0.1
Lee & Srihari [87]	Best of 7 methods	5.0	0.5
Bottou et al. [20]	5 different methods	1.4-1.9	0.5
de Waard [36]	Nearest Neighbour	10.0	1.0
		15.0	0.5
Cao et al. [24]	Neural Network	13.0	1.0
		36.0	0.5
Lee & Srihari [87]	Best combo of 7 methods	5.0	0.25
Sabourin et al. [139]	Unanimity of two 1-NN	3.52	0.20
Bottou et al. [20]	Boosted LeNet 4	0.5	0.5
Thien & Bunke [162]	Multistage Multi-expert (on <i>goodbs</i>)	5.0	0.25
Strathy & Suen [151]	ABC voting (on <i>goodbs</i>)	12.0	0.22
		16.7	0.15
		26.0	0.0

Table 7: Required Rejection Levels for Very Low Error Rates

There is little data available concerning rejection levels required by human subjects to achieve very low error rates. Even when dealing with *machine*-segmented individual numerals (which unfairly penalizes human subjects), it appears clear to us that very few errors would be made if we allowed them to reject say 5.0% of examined samples (i.e. 1 in 20). And the errors, if any, would almost all be due to flawed segmentation.

When examining the results of Table 7, the figures of Bottou et al. are particularly impressive for machine recognition. This may raise the question of whether human performance has been matched with respect to very high reliability as well. However their testing data was a mix of NIST's SD3 and TD1 databases making it somewhat easier than unconstrained numerals from the general public. In their paper, they estimate that humans would attain a 99.8% recognition rate with only 0.2% as substitution rate (no rejection). If allowed, like boosted LeNet 4, to reject say 0.5% of the samples, they would certainly make very few errors on the remaining data, if any. Finally, the combinations of systems whose performance is reported for CENPARMI

data in section 2 are very impressive since many achieve precisely or near 0.00% error rates with low rejection rates. However, because of the small size of testing set T (2 000 samples), these figures are not statistically significant. In addition, even if they were, it is likely that humans would need to reject fewer samples to achieve the same result.

Examining Samples Misclassified by Machines

We begin this section with a quote from Noumi et al. [121] concerning the results of the first IPTP competition:

“However, many patterns which can be recognized easily by humans still remain as those which were substituted or rejected ones by the algorithms.” (p. 338)

We have studied the relevant literature published in recent years and carefully examined the samples *misclassified* by various methods when this information was available. It is our estimate after this investigation that the great majority of these samples are easily classified by humans. In most of the remaining cases, the identity of the digit is not clear (often as a result of artifacts created by segmentation); the proper decision would then have been a rejection, after which a human operator could examine the entire *unsegmented data field* to make the final identification. Finally, in rare instances, humans would also make the same misclassification; but this could also be the result of wrong segmentation...

We list the various instances on which the preceding conclusions have been drawn:

- In LeCun et al. [84], all 17 misclassified samples are shown. Of these, 11 are easily identified and 4 may be considered confusing.
- In Abuhaiba & Ahmed [1], 12 misclassified samples are shown. Of these, 8 are easily identified, 3 are potentially confusing and 1 could also have been mistaken by a human subject.
- Sabourin et al. [139] show 156 misrecognized samples of which the majority are easily identified and most others should be rejected because of confusion created

by segmentation artifacts.

- Matsui et al. [111] show 10 samples which were misrecognized by each of the 3 best systems participating in the first IPTP competition; among these, 4 are easily identified, 3 are potentially confusing and 3 may also have been mistaken by humans.
- Lee et al. [91] show 20 misrecognized samples of which 2 or 3 are potentially confusing and 1 may have also been misrecognized by humans; all other samples are easily recognizable.
- Noumi et al. [121] show 6 samples which were misrecognized by each of the 3 best systems participating in the second IPTP competition; 4 are easy to recognize and 2 are potentially confusing.
- Matsui et al. [110] show 10 substituted samples of which 6 are easily identified and 4 are potentially confusing.
- Lam & Suen [81] show all 14 samples which were misclassified by the voting combination of 7 classifiers (10 of these substitutions were common to all 7 classifiers). In our opinion again, 8 of these 14 samples are easily recognized, 4 are potentially confusing and 2 would likely also be misrecognized by human subjects.

Transferring Expertise Across Databases

We briefly discuss here another aspect which reveals shortcomings of machine recognition as compared to human recognition, namely the significant drop of performance which results from applying a recognition system on a different database from the one it was trained on. We feel it is significant that almost no accounts can be found on this topic in the literature

At CENPARMI, we have conducted some experiments on other databases with experts E1, E2, and E4. Without any additional training, we applied them to a 5000-sample test set of handwritten numerals from a Concordia-Montreal database

with a 200 PPI resolution. For technical information concerning this database, see Appendix A.3. The recognition rates of the methods dropped sharply to 75.10%, 78.80%, and 76.50% respectively; the substitution rates increased to 5.40%, 5.40%, and 5.10%. We note that some samples in this database were split in two or more pieces while the CENPARMI database is composed only of single-component digits; also, writing styles in Quebec are a mixture of North-American and European styles.

In a separate experiment, the same methods (E1, E2, and E4) were applied to a 3000-sample test set of handwritten numerals from an ITRI-Taiwan database with a 400 PPI resolution (see Appendix A.4). The recognition rates of the methods were 66.47%, 69.67%, and 79.83% respectively; the substitution rates were 5.84%, 8.54%, and 5.03% respectively. This data has a much higher resolution than the CENPARMI data (166 PPI) and is more difficult: as many as 13.3% of the samples were broken in 2 or more pieces (29.5% of five's, but also 18% of zero's, 16% of three's, etc ...); furthermore, the writing styles are often quite different from the U.S. with several two's resembling seven's etc The voting combination of the 3 methods improved the performance very significantly, but nowhere near what was achieved on the CENPARMI database. The combined recognition and substitution rates were 74.41% and 1.60% respectively.

Of course, one would expect a diminished performance with such important changes of data characteristics, especially for model-based classifiers such as E1 and E4 (and in good part E2 as well). But the drop in the recognition rate is much more easily justifiable than the equally important increase in substitution rate... The latter cannot all be attributed to data resemblance across numeral classes.

But such problems do not only exist for model- or rule-based classifiers. They are also encountered for neural networks. In other experiments at CENPARMI, Strathy³¹ trained a slightly improved version of a previously discussed MLP on one database and tested it on another. When trained on the 18 468 samples of the *br* training set from CEDAR, it yielded a forced recognition rate of 98.46% on the CEDAR *goodbs* testing set but only 90.32%³² on a 5000-sample test set from an ITRI database.

³¹ Private communication.

³² Thus an error rate of 9.68%.

Conversely, when trained with 46 922 samples from the ITRI database, it scored a forced recognition rate of 98.72% on the 5000-sample ITRI test set and only 90.74% on the CEDAR *goodbs* test set. Even the improved 3-net voting combination producing the 99.23% recognition rate on the *goodbs*, gave a forced recognition rate of only 92.86% on the 5000-sample ITRI test set. And it achieved 98.30% on the 2000-sample CENPARMI test set. For this last figure, one must bear in mind that both datasets are from U.S. postal codes; on the *goodbs* data, the forced error rate is 0.77% and on the *T* data it is 1.70% (more than double). If reliability is our primary concern, such variations in error rates are indeed problematic.

Chapter 3

Overview of our Research

The practical *starting point* for our Ph.D. was the research and development effort around the creation of Expert E4, briefly described in 2.1.1. And our *general goal*, as expressed in the thesis title, was to make a contribution towards matching human performance, more precisely in the area of very high reliability.

As previously exemplified, the combination of several complementary recognition schemes is one promising approach towards this goal, and a path down which many have walked in the past few years. But we felt that the apparent limits of single-approach recognition systems, particularly those of structural model-based methods, could be overcome and we wanted to see how far we could go in this (much less popular) direction.

In Legault et al. [99], we summarized the inherent difficulty of this problem as follows:

“The systems must then incorporate much more knowledge and their development can become a painstaking task. A situation of diminishing returns often arises, where efforts to repeatedly refine classification schemes produce smaller and smaller gains in overall reliability. In this respect, the combination of complementary algorithms into multi-expert systems appears to be a much more promising alternative (...).” (see p. 235).

Our objective was not necessarily to circumvent the “painstaking task” but to overcome the “diminishing returns” aspect of the problem. More precisely we set

out to test the *feasibility* of a structural model-based method which could achieve a recognition rate of approximately 90% with ‘close to 0%’ substitution rate on the CENPARMI database. To our knowledge, the most reliable single approach recognition methods for which results are known on the CENPARMI database are: PIQ1079_NB_SW13 of Franke [46] with a reliability of 98.70%, a recognition rate of 94.55% and substitution rate of 1.25%; and the method of Krzyzak et al. [75] with a reliability of 98.85% with a recognition rate of 86.40% and a substitution rate of 1.00%.

The new system would also be required to make the passage to other databases without suffering from the severe degradation of performance in the area of reliability which is observed for other systems: a decreased recognition rate of say 75-80% would be acceptable as long as the substitution rate remained very low. With such an outcome, we could conclude that what was learned on one database was robust and reliable since it did not generate all kinds of errors when applied to a different dataset; the decrease of recognition rate, for its part, is justifiable since the new database most likely contains writing styles and variations not encountered in the first one.

Before presenting our approach, we wish to stress that focusing on a single-method recognition system should not be seen as a ‘return to the past’ while the future lies in the combination of methods...

Firstly and obviously, combination schemes rely on the existence of individual methods; and the better are the individual building blocks, the better the combinations are likely to be.

Secondly, it is a relatively easy task to write programs which will combine the *outcomes* of classifiers according to voting, Bayesian, Dempster-Shafer or some other formalism. But it is not obvious at all that the algorithms so combined on paper are easily combined in a real-life system. If their preprocessing, feature extraction and classification methods are significantly different, one has to think of a system architecture where these stages are performed in parallel on several processors; and the synchronizing problems, associated with methods which may have very different computing time, must be solved. Otherwise, on a single-processor machine, a very severe drop in CPU time per sample will result from combinations of several methods.

Thirdly, in order to improve efficiency, it would be very handy to have a reliable and reasonably fast single method which could process, for example, a good 80% of the data with an extremely low error rate. The combination approach, which cannot be faster than the slowest of the combined methods, would then only deal with the remaining 20% of the data. When recognition is also used repeatedly to validate the proposed cuts of a numeral string segmenter¹, such efficiency considerations are even more important.

In the next chapters we will discuss in detail the preprocessing, feature extraction, classifier development and results of our recognition system. For each stage, we will present both the initial characteristics of the E4 system, specific problems discovered, and the major revision and expansion carried out. In the rest of the current chapter, we want to give an overview of the weaknesses we later identified in the general approach followed for developing E4, and an overview of new avenues we pursued for the new system.

3.1 Limitations of the Expert E4

On the positive side, Expert E4's strong points can be summarized as follows:

- it efficiently combines edge smoothing, contour extraction, and simple measurements on holes (area and minimum bounding box) in a single pass over the binary image;
- it uses a new technique to extract structural features from contours;
- it relies on a large set of specialized modules to handle different aspects of the recognition task;
- it can at times move back and forth between classification and further feature extraction when more refined measurements are needed.

In its original version, Expert E4 consisted of a Pascal module for the first item mentioned above and Fortran modules for the other items. The total size of the

¹ The best segmenters work in this manner.

source code was approximately 700 KB. The average processing time was 42 ms per numeral image on a VAX2 computer system.

E4 was created by the author without a vast experience of the area of OCR. Hence, it was developed from the bottom up and targeted towards solving the specific difficulties of a particular database (CENPARMI). This database has relatively low resolution (166 PPI) and is made up of single-component images. The extraction of structural features from the outer contours, for its part, was derived from a new method we had developed with another purpose in mind: the piecewise approximation of contours with parametric cubics and quartics.

The development of the large set of classification rules was done in several stages. With hindsight, we can say that it suffered all along from an initial misconception of the author, namely that of wanting to achieve the highest possible *recognition* rate with *as few rules as possible*. While this is not necessarily a wrong goal, it can be contradictory with our present aim for maximum *reliability*. We will expand more on this in the next section.

3.1.1 Lessons Learned on the ITRI Database

As part of a joint ITRI-CENPARMI project, experiments were conducted to adapt E4 and other methods to the ITRI database of 46 451 samples (24 427 for training and 22 024 for testing; 400 PPI). The initial recognition and substitution rates of E4 were given at the end of the preceding chapter: 79.83% and 5.03% respectively. The performance was gradually improved to new rates of 87.32% and 1.81% respectively. As a result of these improvements, performance on the CENPARMI database could also reach up to 94.65% recognition and 1.55% substitution. Important lessons were learned in the process.

This database was quite difficult for our systems. To begin with, 13.3% of all samples were broken in 2 or more components as already mentioned. These broken samples accounted for much more than their relative share of substitutions and rejections. Secondly, the database contained some writing styles which were quite different from typical North American writing styles and at times even close to North American writing styles for digits of another class. For examples of these difficulties,

see Figure 1.

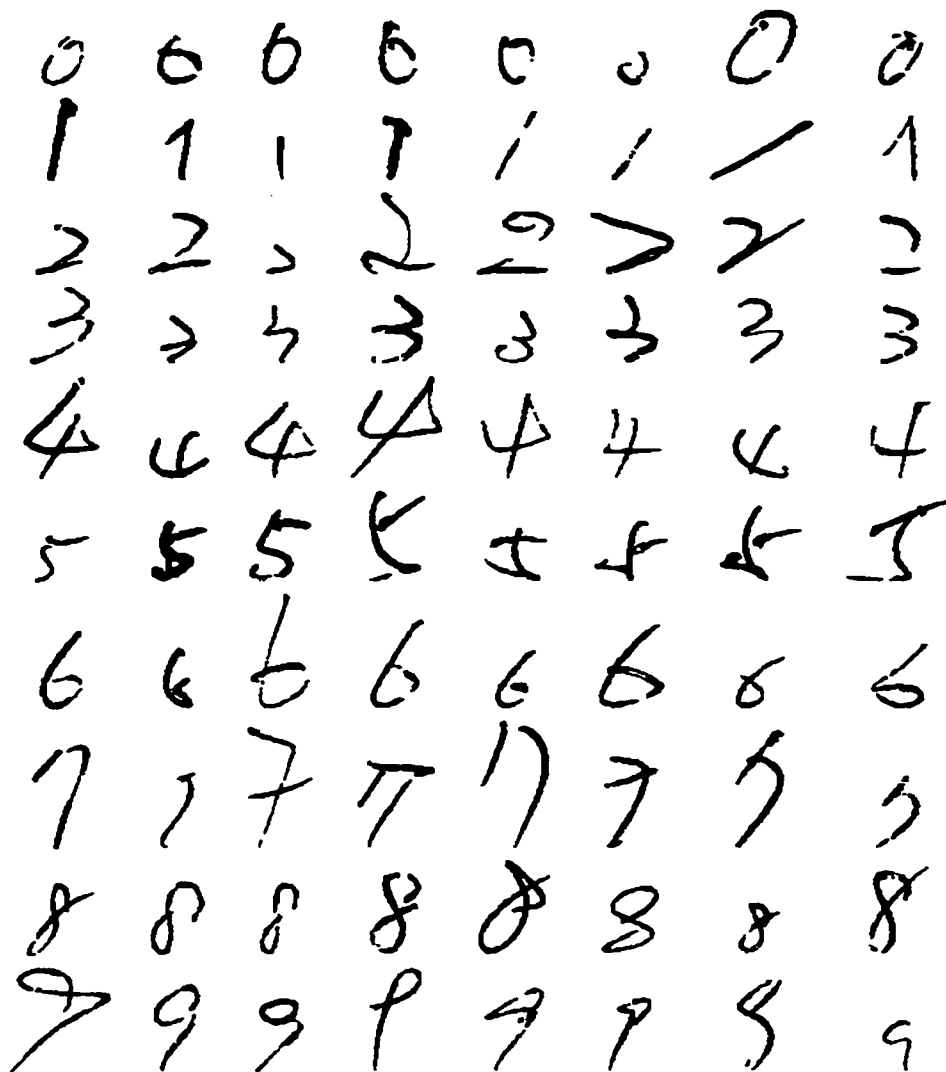


Figure 1: Difficult Samples from ITRI Database

In large part however, E4's own weaknesses were to blame for the serious deterioration in initial performance. Using the CENPARMI database, the classification rules were handcrafted by subjectively clustering samples with the same general shape and then extracting, from subjective scrutiny of these clusters, the sequence of rules

describing them. Aiming for the highest possible recognition with as few rules as possible can lead to problems of clustering too broadly and modeling too loosely:

- Samples which would be better described as part of 2 or more clusters are all put in the same cluster...
- Samples which are outliers and really poorly formed or confusing are put in a cluster with other samples instead of being ruled out of the training data entirely
- To accomodate such broad clustering, the recognition rules for each cluster must tolerate a much larger range of values for various feature attributes

This results in serious loopholes in the web of classification rules, loopholes which may not be very apparent based on the results of a limited test set such as set T of CENPARMI, but which are clearly revealed for the larger and relatively different test set of the ITRI database. The gain in performance later obtained on the ITRI database was achieved by splitting clusters into narrower sub-clusters and describing them with tighter sets of rules, and by creating new clusters corresponding to new handwriting styles etc...

The work on the ITRI database also revealed weaknesses at earlier stages of the recognition process, especially with feature extraction and even with preprocessing. The algorithms for the extraction of endpoints, concave regions and convex regions along the contour had been devised and tested on images digitized at 166 PPI. They were not robust enough for the huge jump to 400 PPI; at that resolution, significant features were missed more frequently and, conversely, spurious features were extracted more frequently also. The problem was identified but no work to improve the extraction of features was carried out in the ITRI project.

We believe that such shortcomings in feature extraction, and the difficulty in overcoming them, are key reasons for the limits of structural, rule-based recognition systems². Others who have worked in this area and have tried to automate the rule-learning process, such as Nishida & Mori [119] and Commike & Hull [34], have also

² And for their near-abandonment for the last few years.

voiced this opinion. We have devoted much of our attention in the current research to build the required solid feature extraction foundation to move beyond current limits.

3.2 General Avenues

We now summarize the general avenues we have followed in order to produce a very reliable and yet efficient recognition system for unconstrained handwritten numerals.

- With the knowledge accumulated in previous research and our new goal of maximum reliability, we revisited all stages of the recognition process to bring more knowledge and expertise into each.
- Preprocessing was revised and much expanded to correct for a broader class of image defects encountered with numeral images scanned at various resolutions.
- Feature extraction was completely redesigned to attain much higher levels of robustness and reliability. It is one thing to *identify* that curvature features along contours –such as endpoint, concave, and convex regions– are “very good features”; it is another to actually *extract* them reliably (ideally missing none of the significant features) and robustly (ideally extracting no spurious features at all) and to do so at various resolutions. More human expertise (essentially that of the author) was also integrated into feature selection and extraction.
- The general approach to the ‘training’ of the classifier was modified importantly.
 1. For E4, we were trying to recognize as much of the training data as possible; recognition and substitution rates of 98.95% and 0.20% respectively were achieved on CENPARMI’s training set A (97.55% and 0.20% for training set B). For the new system, several samples, confusing or rather unique in their style, were *discarded* from the start.
 2. As explained above, the subjective clustering operation on training samples was aimed at narrower clusters.

3. The classification rules for each cluster were fully developed to account for *all* features around the outer contour of the image. In Expert E4, we ‘relied’ on the natural redundancy between left and right profiles in some (parts of) numerals and rule development was only carried out ‘as far as required’ to recognize training samples and avoid substitutions on them. For an example of how one can be misled by relying on incomplete contour information, see figure 2.

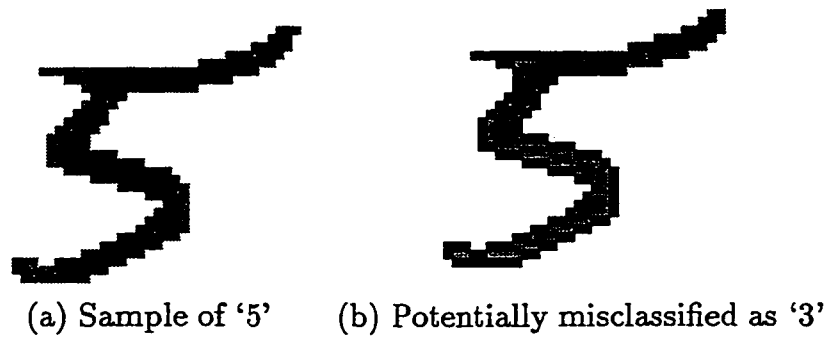


Figure 2: Substitution Created by Partial Contour Information

The image on the left is easily recognized as a ‘5’; considering 68% of the contour information highlighted in figure 2 (b), the numeral really looks like a ‘3’. We note also that the top right stroke could have been shorter (the highlighted portion would then represent a larger percentage of the contour) and the same misrecognition would be created.

4. *Interrelations* between attribute values of *different* features were investigated systematically and integrated into the subsets of recognition rules. For example, the opposite pairs of cavities defined at the crossing of the horizontal and the vertical strokes in many ‘4’s can have a broad range of orientations; but the difference of these directions falls within a narrower range. Furthermore, the presence of redundancy between left and right profiles was verified when typical of certain numeral classes or sub-classes,

both in terms of the orientations of related features and their relative distance.

5. The points listed above indicate that the rule generation process would become an even more lengthy and painstaking process. To help towards this task, we developed a sophisticated rule development environment. While this did not automate the process in any way, it facilitated
 - the visual inspection of databases based on various queries;
 - the clustering of samples by the human developer;
 - the gathering of statistics to investigate potential rules;
 - the verification (parsing) and writing of accepted rules in rule files.
6. Because of the considerable R & D effort involved in all preceding stages (preprocessing, feature extraction, rule development environment) and because of the still ‘painstaking’ character of developing classification rules, we limited the development of the classifier to 4 numeral classes. For these classes, we have been able to demonstrate unequivocally that our reliability targets were achievable with a single-method system.
- Finally, we took even more liberty than we had in Expert E4 with the *classical sequence* of pattern recognition operations:

preprocessing \longrightarrow *feature extraction* \longrightarrow *classification*

This will be explained in the next section.

3.3 Design of the Recognition System

In E4, we had already implemented feature extraction and classification as inter-related and cooperating processes. The rationale for this is explained in Suen et al. [156]:

“Consider a classification task which potentially requires a broad range of information, from coarse and easily extracted elements to others which are

much more subtle and costly to obtain. Furthermore, assume that only a relatively small subset of this information is really necessary to successfully classify any individual pattern. Rather than wasting computer resources measuring beforehand everything which could possibly be useful in all cases, it would seem more reasonable that recognition proceed on the basis of some smaller set of initial features and to let the early recognition stages establish what finer details to extract and when. The more diversified and sophisticated is the information that a system potentially makes use of, the more appropriate and efficient it would be to swap back and forth between feature extraction and classification subtasks, as recognition progresses.” (p. 309).

Since our new system makes use of more diversified and more carefully measured features, this approach has been carried to a higher level.

The flowchart of Figure 3 represents the general design of our new system. Beginning at the circled 1 at the top left, we have a preprocessing stage, followed by the general feature extraction required for classification to proceed.

When dealing with *single-connected-component numerals*, the system then proceeds to a third stage, where the inter-relation between classification and further feature extraction subtasks is clearly illustrated inside the shaded box. In this flowchart, the large ‘S’ and ‘NS’ indicate whether recognition was Successful or Not Successful. As can be seen, when recognition is not successful, we can have two other attempts at classification. These will proceed only when certain changes can be made to the feature list: the second pass is attempted only when small, potentially spurious, *curvature features* around the outer contours are detected and removed; the third pass is attempted only when small, potentially spurious, *holes* are detected and removed;

The 3-pass classification was carried over from Expert E4 to the new system. But the hole and feature filtering steps were changed in important ways to preserve the very high reliability achieved in the first pass; in E4, these steps were responsible for several errors.

The handling of *multi-connected-component numerals* is totally new, since E4 did not have to deal with this problem at all. It is done in 3 different ways.

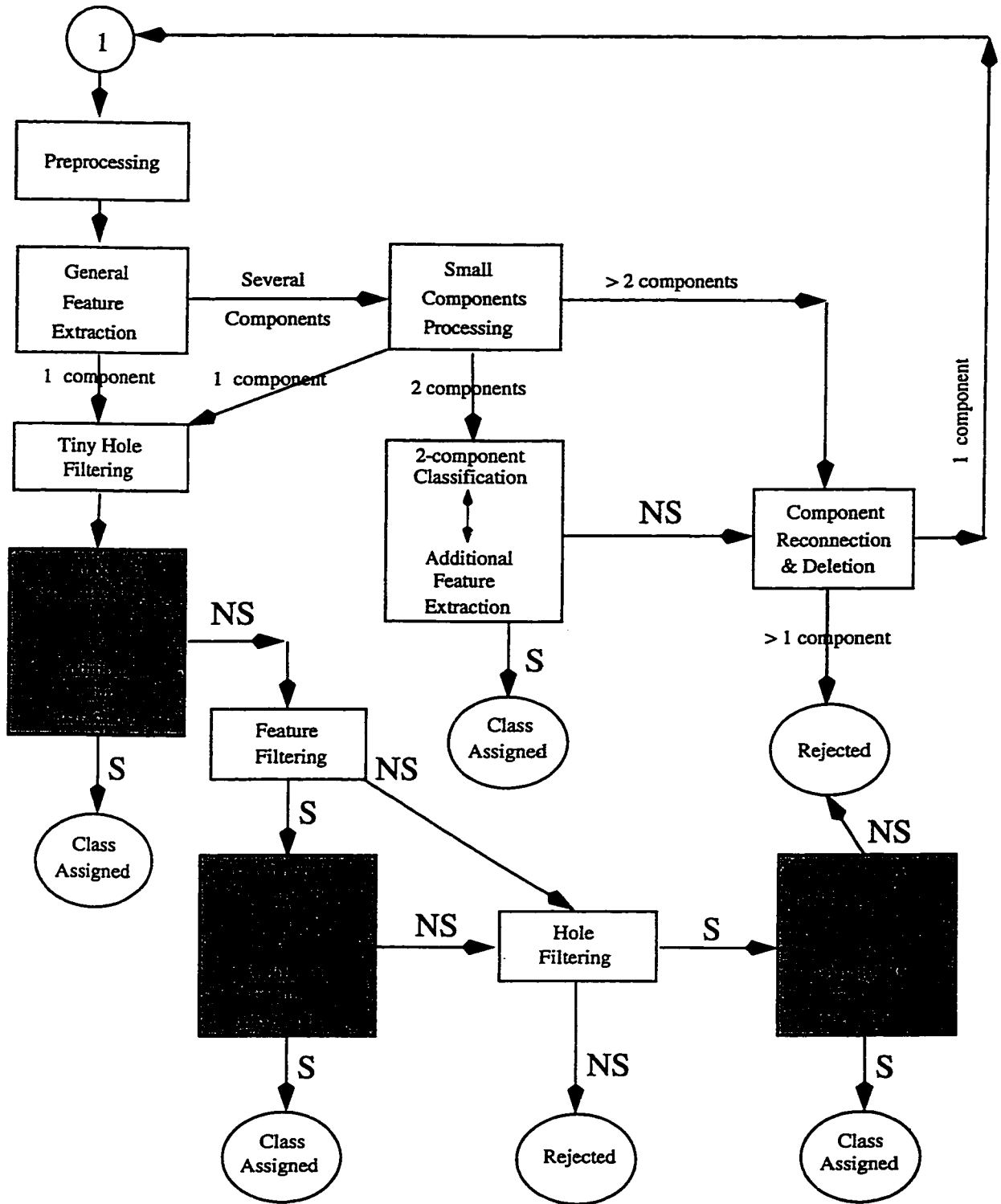


Figure 3: Design of Recognition System

First, a specialized module examines very small components; in some cases, they will be removed; in others, the feature extraction on these components will be improved. If the result of this processing is a single-component numeral, it will be processed as just described above.

Second, if exactly 2 components are present (after the small components module has performed its work), recognition will be attempted directly for certain styles of '5' (mostly), and '4', and '1'. These variants are more typical and easier to recognize this way than after reconnection.

Third, if the 2-component classification fails or if more than 2 components are present, a component reconnection and deletion module will take over. After this, if the result is not a single component, the sample is rejected. Otherwise, if component reconnection is performed and results in a single-component image, recognition starts completely anew with preprocessing, general feature extraction, and 1-component classification.

Chapter 4

Preprocessing

With this chapter, we begin the more detailed description of the methods and techniques used in the various stages of our numeral recognition system. However, we cannot present in this thesis *all the details* related to each aspect of a very intricate system. Often, we will refer to published articles and technical reports. And the final word really lies within the programs where all the details are encoded.

The input data to our recognition system is a file of *run-length encoded* binary images of single numerals. We recall that an image may consist of a single ‘piece’ or connected component, or it may be broken into several connected components. We now briefly describe the file format and define run-length encoding.

A *black run* is defined as a horizontal sequence of object pixels, preceded and followed by a background pixel or the image boundary. For example, row 19 in Figure 4 (a) has 3 black runs. Conversely, a *white run* is a horizontal sequence of background pixels, preceded and followed by an object pixel or the image boundary. For each binary image, the input data begins with a sequence of 5 integers followed by the run-length encoding. The 5 initial integers are a file number, a sample number, the numeral’s identity (0-9), the number of rows and the number of columns in the image. The run-length encoding consists of the length of alternating white and black runs on each row. By convention, each row is assumed to begin with a white run, which may be of length 0.

The fundamental preprocessing operations described in this chapter are edge and

contour extraction. These are efficiently combined, within the same single pass over the binary image, with edge smoothing, correction of several other small image defects, and computation of useful global features.

4.1 Edge Extraction

Edge extraction is performed, in the new system as it was in Expert E4, with an efficient sequential tracking algorithm which makes use of the edge-type concept of Ahmed & Suen [4].

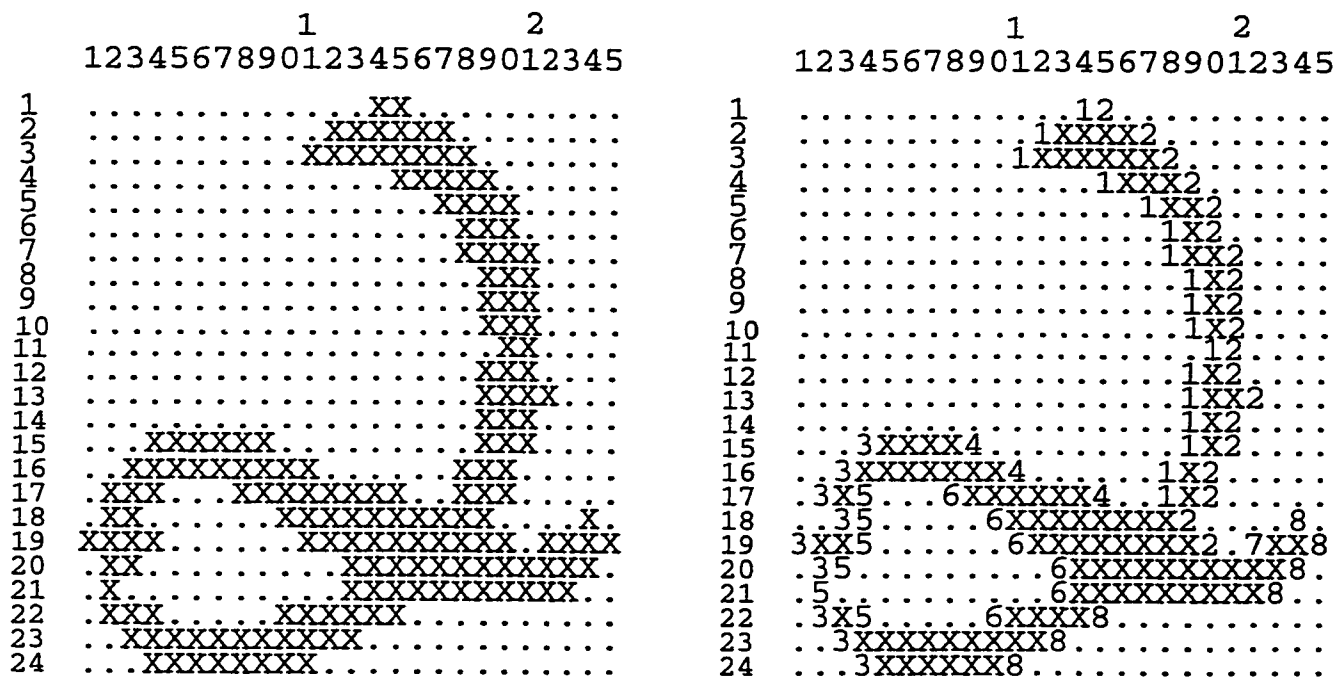


Figure 4: Edge Extraction From Binary Image

In summary, the extreme, leftmost and rightmost, pixels of a black run are defined as *edge points* and belong to different edges. Black runs are examined, from left to right, row by row. In this process, edges are *created*, *continued*, or *terminated* based

on black run connectivity from row to row¹. We now discuss this in more detail.

A new pair of edges is *created* in two circumstances:

- When a black run in one row is not connected to any black run in the preceding row, a new edge of type 1 is created and the leftmost pixel of the run is stored as its first point; and a new edge of type 2 is created and the rightmost pixel of the run is stored as its first point. Intuitively, this corresponds to the top of a new blob, or the top of one of its upward protrusions. In Figure 4 (a), this happens three times: in row 1, where edges #1 and #2 are created; in row 15, for the first black run, where edges #3 and #4 are created; and in row 18, for the last (single-pixel) black run, where edges #7 and #8 are created.
- When two consecutive black runs in one row are connected to the same black run in the preceding row (a *split*), a new edge of type 3 is created and the rightmost pixel of the first black run is stored as its first point; and a new edge of type 4 is created and the leftmost pixel of the second black run is stored as its first point. Intuitively, this corresponds to the top of a hole or the top of a downward cavity in the contour. This occurs only once for the chosen image for the first 2 black runs of row 17, where edges #5 and #6 are created.

At their creation, edges are labeled with a unique integer, giving their rank in the edge creation process. Edges are then *continued* from row to row until they eventually terminate.

A pair of edges is *terminated* also in two circumstances:

- When a black run in one row has no connected black run in the following row, the 2 edges to which its extreme points belong are terminated. Intuitively, this marks the bottom of a blob, or the bottom of one of its downward protrusions. This happens in the last row, with edges #3 and #8.
- When two consecutive black runs in one row are connected to the same black run in the following row (a *merge*), the edges to which the rightmost pixel of

¹ In our system we use 8-connectivity; thus 2 black runs in consecutive rows are considered connected if there is at least one pixel on one black run which is 8-connected with one pixel on the other black run.

the first run and the leftmost pixel of the second run belong are terminated. Intuitively, this corresponds to the bottom of a hole or the bottom of an upward cavity. This happens three times in our image: in row 17, where edges #4 and #1 are terminated; in row 19, where edges #2 and #7 are terminated; and in row 22, where edges #5 and #6 are terminated.

The extraction of edges from a binary picture is a process which advances row by row. When processing row k , for example, we have a linked list of black runs for row k and a linked list of black runs for row $(k - 1)$. The task is to establish the edge labels of the 2 extreme pixels of every black run on row k , based on their connectivity with the black runs in row $(k - 1)$, for which the associated edge labels have already been recorded in the associated linked list of black runs.

In Ahmed & Suen [4], connectivity is established with the help of a C-matrix, which has as many rows as there are black runs on row $(k - 1)$ and as many columns as there are black runs on row k . However, this requires additional storage and delays execution. As shown by Agrawala & Kulkarni [2] and Capson [26], a simple set of rules can be used to establish connectivity. Since this is more efficient, we have used such a set of rules, modified to implement 8-connectivity.

While each row is being processed, the run-lengths read from the input file are also used to create the rectangular array of the image shown in Figure 4. This image format is conveniently used for edge smoothing and correcting other image defects (see section 4.5). The result is illustrated in Figure 4 (b). Note that single-pixel black runs are also associated with 2 edges but only the largest of the two labels is shown. Thus pixel (18,24) is associated with edges #7 and #8 and pixel (21,2) is associated with edges #3 and #5.

For each edge, an EDGENODE data structure records the following information:

- its label;
- its type;
- its starting row;
- the labels of the 2 associated edges at creation and termination;

- the number of rows covered by the edge;
- an array containing the column numbers of the edge points;
- the label of the *parent edge*, which is used to unravel the relative nesting of blobs and holes in the image and which is a novelty of our method;
- the *object number* to which the edge belongs (see section 4.3).

Finally, we note that any empty rows at the top or bottom of the image, and any empty columns at the left or the right of the image are removed during the edge extraction process. The resulting picture is thus in its minimum bounding box.

4.2 Contour Extraction

It is easy to see that by following edge #1 downward, then edge #4 upward, then edge #3 downward again, then edge #8 upward again, then edge #7 downward again, and finally edge #2 upward, we have essentially followed the outer contour of the image in a counter-clockwise fashion. Similarly, going down edge #5 and then up edge #6, we obtain the inner contour. Thus contours are obtained essentially from chains of edges, which are easily created since for each edge the labels of the associated edges at creation (top) and at termination (bottom) have been recorded.

Chains of edges do not provide complete 8-connected contours since only the extreme pixels of every black run are edge points. Some gap filling is also required. For example, starting with edge point (1, 14) of edge #1, we must insert pixel (2, 13) before taking the next edge point (2, 12). Here the gap filling is done in the row of the next edge point to be inserted, row 2. Another situation is illustrated in rows 3 and 4: after including edge point (3, 11), we must add pixels (3, 12), (3, 13) and (3, 14) before we can insert the next edge point (4, 15). Similar situations occur when following an edge upward. Additional gap filling may also be required when chaining one edge to the next; it may be performed in the same row as the edge points to be connected (in row 15, to chain edges #4 and #3), in the preceding row (in row 16,

to chain edge #6 to #5 at the top), or in the next row (in row 23, to chain edge #5 to #6 at the bottom).

In our programming implementation, we keep all external contour points of all extracted blobs in a single array. We will see later how each blob can recover its own points.

4.3 Image Segmentation

In the most general case, a binary image may contain several blobs or connected components. These connected components may contain holes, which may themselves include islands, possibly also containing holes, all this to an arbitrary level of nesting.

The chaining of edges, as just described, can elegantly solve the problems of 1) segmenting the disconnected components and holes, and 2) finding all the relationships between them. The *object number* field of an edge is used for this purpose. It is initialized to 0 to indicate an edge which is not yet part of any object (blob or hole). Each object is assigned a unique integer label which is positive for blobs and negative for holes. The nesting is derived by making use of the parent edge. In the simplest case, the parent edge belongs to the enclosing object; in more complex cases, it will belong to a *sibling object*².

Now we consider the edges extracted in Figure 4 (b) and illustrate these notions concretely. The first 4 columns of Table 8 give the edge numbers, edge types, parent edge numbers and object numbers resulting from the edge extraction process. We now explain these figures.

First, the parent edge numbers. Parent edges are assigned when edge pairs are created. As seen above, this can occur in 2 ways.

In the first situation (a black run which has no connected black run in the preceding row), the parent edge is the edge associated with the leftmost pixel of the first black run in the preceding row which is beyond the current run. For example, the first black run in row 15 (from column 4 through column 9) has no connected black run in row 14; hence the creation of edges #3 and #4. The first black run in row

² Two objects are siblings if their immediate enclosing object is the same.

Edge Number	Edge Type	Parent Edge	Object Number	Object Number	Object Number
1	1	-1	0	1	1
2	2	-1	0	1	1
3	1	1	0	1	1
4	2	1	0	1	1
5	3	3	0	0	-1
6	4	3	0	0	-1
7	1	-1	0	1	1
8	2	-1	0	1	1
			Init.	Chain #1	Chain #2

Table 8: Edge Chaining and Object Segmentation

14 which is beyond column 9 begins in column 29 and that pixel is associated with edge #1. Hence, the parent edge of edges #3 and #4 is edge #1. When there is no next run in the preceding row (when edge pairs #1-#2 and #7-#8 are created), the value -1 is assigned to the parent edge, indicating that these edges definitely belong to an outermost blob.

In the second situation where an edge pair is created (black run 'split'), the parent edge is the edge associated with the leftmost pixel of the black run that 'splits'. For example, in row 16, the black run from column 3 through column 11 is connected to 2 black runs in the next row which creates edges #5 and #6 in row 17. Their parent edge will be edge #3, associated with the leftmost pixel of the black run in row 16.

4.3.1 Edge Chaining and Object Labeling

Now we examine how object labels are assigned to edges in the edge chaining process. The first edge chain begins, of course, with edge #1. Since it is necessarily of type 1, we know that this edge belongs to the outer contour of a blob. And since this is our first blob in the image, we use 1 for its object label. Following the chain leads successively through edges #1, #4, #3, #8, #7, and #2. For each of these edges, the object number assigned will thus be 1 (entered in bold in the column before last).

The next chain will always begin with the edge of lowest rank which is yet unchained i.e. which has an object number value of 0. In our example, this is edge #5. Because its edge type is 3, the new edge chain will necessarily be an inner contour, i.e. the contour of a hole³. Since it is the first hole in the image, we assign it an object number of -1. And edges #5 and #6 will be assigned this object number, completing the edge chaining process.

For each object (blob or hole), an OBJECTNODE data structure will record the following information:

- the *parent object*;
- the minimum bounding box of the object;
- the total enclosed area (including the areas of all enclosed objects);
- the area of the object per se;
- a pointer to the first enclosed object;
- a pointer to the last enclosed object;
- a pointer to the next *sibling object*;
- the number of contour points;
- the index of the first contour point into the all-contour-points array;
- the number of curvature features extracted;
- the index of the first feature into the all-features array;
- the number of endpoints;
- the number of cavities;

The last four items are explained in chapter 6.

³ Using the *edge type* of the yet unchained edge of lowest rank to determine whether the object is a blob or a hole is a new and much simpler criterion than the one used in Ahmed & Suen [4].

4.3.2 Finding Nesting Relationships Between Objects

When a new edge chain (object) is begun, we can deduce its *parent object* number from the *parent edge* information in the following manner. The first edge chain, resulting in object 1, begins with edge #1 whose parent edge is -1 . This means that object 1 is surely an outermost blob and we assign a *parent object* number of 0 to all such objects. The second edge chain (resulting in object -1) begins with edge #5 whose parent edge is #3. And edge #3 belongs to the object labeled 1. Thus object -1 has parent object 1.

Before examining the more complex case, we note the following general rules:

1. Blobs can only enclose holes and vice-versa; hence, except for outermost blobs, objects and their parent objects must have labels of opposite signs.
2. Edge # k necessarily has a parent edge of label less than k , since the parent edge was created earlier (above) in the process.
3. Because every new edge chain begins with the currently unchained edge of *lowest* rank, we can be sure that the parent edge has already been associated with an object number when a new edge chain is created.

When using the parent edge concept to deduce parent objects, the more complex case occurs when sibling objects are further related in the following manner: one object begins with an edge whose parent edge belongs to a sibling object. Two simple situations of this kind are illustrated in Figure 5 and the associated edge information is found in Table 9.

In Figure 5 (a), we have 2 sibling blobs enclosed within the same hole. Since edge #1 has parent edge -1 , we conclude that the parent object of object 1 has label 0. Since edge #3, which begins object -1 , has parent edge #1 which belongs to object 1, we conclude that object 1 is the parent object of object -1 . Since edge #5, which begins object 2, has parent edge #4 which belongs to object -1 , we conclude that object -1 is the parent of object 2. Finally, we arrive at the new situation: edge #7, which begins object 3, has parent edge #5 which belongs to object 2 Now object 3 cannot have object 2 as its parent because a blob cannot be immediately enclosed

```

1XXXXXXXXXXXXXXXXXX2
1XXXXXXXXXXXXXXXXXX2
1X3.....5X6..4X2
1X3.....5X6..4X2
1X3.....5X6..4X2
1X3..7X8..5X6..4X2
1X3..7X8.....4X2
1X3.....4X2
1XXXXXXXXXXXXXXXXXX2
1XXXXXXXXXXXXXXXXXX2
    
```

(a) Sibling Blobs

```

1XXXXXXXXXXXXXXXXXX2
1XXXXXXXXXXXXXXXXXX2
1X3...4XXXXXXXXX2
1X3...4XXXXXXXXX2
1X3...4X5...6X2
1X3...4X5...6X2
1X3...4X5...6X2
1X3...4X5...6X2
1XXXXXXXXXXXXXXXXXX2
1XXXXXXXXXXXXXXXXXX2
    
```

(b) Sibling Holes

Figure 5: Sibling Objects With Parent Edge Connection

Edge Number	Edge Type	Parent Edge	Object Number
1	1	-1	1
2	2	-1	1
3	3	1	-1
4	4	1	-1
5	1	4	2
6	2	4	2
7	1	5	3
8	2	5	3

(a) Sibling Blobs

Edge Number	Edge Type	Parent Edge	Object Number
1	1	-1	1
2	2	-1	1
3	3	1	-1
4	4	1	-1
5	3	4	-2
6	4	4	-2

(b) Sibling Holes

Table 9: Edge Information for Above Figures

within another blob. The solution is that both these blobs must then have the same enclosing hole; thus, object 3 also has object -1 as its parent.

In figure 5 (b), we have 2 sibling holes enclosed within the same blob. We easily find that object 1 has parent object of label 0 and that object 1 is the parent of object -1. Now edge #5, which begins object -2, has parent edge #4 which belongs to object -1. Again, this is not possible since a hole cannot be immediately enclosed within a hole. The solution is that both these holes must then have the same enclosing blob; thus, object -2 also has object 1 as its parent.

We have created 3 versions of our contour extraction scheme. The first version

extracts all contour points (counter-clockwise) only for the outermost blobs and extracts bounding boxes only for holes immediately enclosed in these blobs; all blobs and holes at a deeper level are ignored; this is the version used for our numeral recognition system. The second version extracts all contour points for both the outermost blobs and their immediately enclosed holes, also ignoring objects at deeper levels of nesting. Finally, the third version extracts all contour points for all objects at all levels of nesting.

4.4 Useful Global Features at Minimal Cost

Areas of body regions and holes, and their *minimum bounding boxes* can be derived at very little cost as we chain edges to extract contours. These techniques were carried over from Expert E4 to our new recognition system. The latter also computes a reliable estimate of the *stroke width*, a measurement which plays an important role in our new feature extraction and classification stages but which was not considered in E4.

4.4.1 Stroke Width Estimate From Edge Extraction

The stroke width, in pixels, depends on several factors among which we can mention: the resolution for data acquisition; the writing instrument (pen, ball pen, felt pen, pencil with sharp or dull point, ...), the porosity and softness of the paper, etc. Even within a single numeral image, there can be important variations in the stroke width, from region to region, depending on varying inclination and pressure exerted on the writing instrument. Thus, what we are trying to obtain is really a *mean* stroke width. An *estimate* of this feature can play an important role in several aspects. Our new recognition system makes use of it 1) in feature extraction, for endpoint detection; 2) in the reconstruction of broken digits; 3) and very intensively in the classification stage, to confirm redundancy between left and right profiles, to verify that a stroke goes around a hole, or to verify that a hole is filled, etc... Our new system derives its estimate as follows.

In the edge extraction process, as run lengths are read from the input files, separate counts are kept for black runs of lengths 1, 2, 3, etc. At the end, the total count of black runs for the entire image is computed from these separate counts. The stroke width estimate is obtained as $\max(3, j)$, where j is the largest integer such that:

$$\sum_{i=2}^j \text{separate_count}[i] < 0.25 \cdot \text{total_count}. \quad (7)$$

Thus the minimum value of the stroke width estimate is 3. What is the rationale behind the above estimate? Since the images were scanned horizontally, the lengths of black runs will adequately reflect the stroke width only for the portions where strokes are vertical or nearly vertical. For all other orientations, black run lengths are larger than the stroke width and should not be considered.

On the other hand, we cannot simply choose the length of the shortest black run as our estimate. Indeed, we know that a number of black runs will be shorter than the mean stroke width we are seeking. This is so because fainter regions, such as stroke endings and other parts where less pressure was exerted by the writer, will have been binarized narrower than their grey level width, as a result of the binarization threshold. Experimentally, a good compromise was realized with the 0.25 parameter in the above equation.

4.4.2 Computing Areas During Edge Chaining

Different methods have been proposed to measure the area of discrete binary objects ([50], [77], [142], [135], [134]). For a brief discussion of these approaches, see Legault & Suen [94], pages 9-10.

For our application, it is sufficient to compute areas as the *number of pixels* making up the objects. A black run beginning in column c_i and ending in column c_f will contribute $(c_f - c_i + 1)$ pixels to the area of the blob to which it belongs. The problem is knowing which blob it belongs to . . . One solution, proposed in Agrawala & Kulkarni [2] is to create temporary blob objects every time a pair of edges of types 1 and 2 are created in our method, and temporary hole objects every time a pair of edges of types 3 and 4 are created in our method. Partial areas are cumulated in these

temporary structures and merged when it is realized that these temporary objects merge. When external contours have many bays and peninsulas, this can represent significant overhead.

By first recording all edge information as discussed in section 4.1 and only subsequently chaining the edges to extract the contours, we obtain a simpler and more efficient method. First, we note that the column coordinates involved in $(c_f - c_i + 1)$ are those of edge points which are stored in different EDGENODEs. Second, since we know the number of rows covered by an edge, we can avoid the individual $+1$ additions, by adding this row count only once.

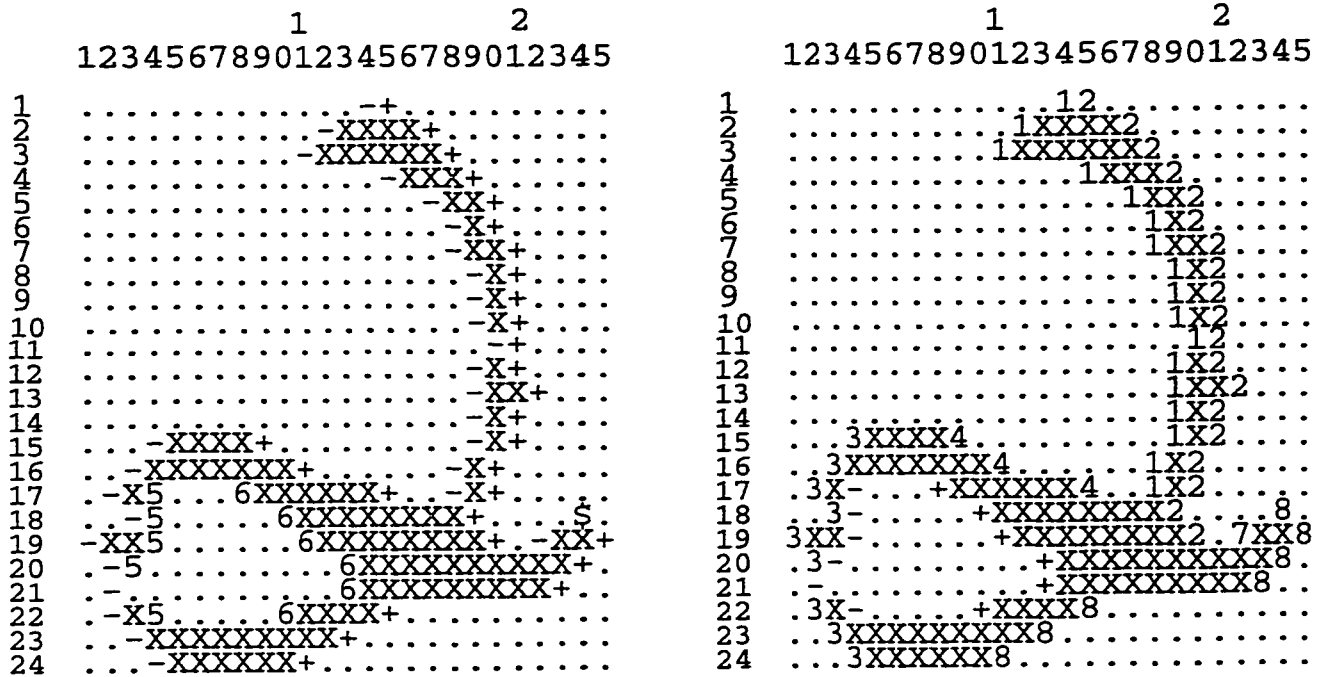
To obtain the area of blobs, whose edges we follow in the counter-clockwise direction, we simply follow these rules:

1. Initialize the area to 0.
2. For every edge followed *downward*,
 - (a) subtract the column number of every edge point from the area;
 - (b) add the row count of the 'downward' edge to the area.
3. For every edge followed *upward*, add the column number of every edge point to the area.

The same set of rules can be used to compute the number of background pixels making up the area of holes, with one minor change: in rule 2 (b), we would now *subtract* the row count of the 'downward' edge from the area.

Figure 6 illustrates the computation of blob and hole areas for the '2' of Figure 4. Column numbers are added to the area when marked with "+" and subtracted when marked with "-". In Figure 6 (a), both operations are performed at pixel (18, 24), marked "\$".

We note that the areas calculated with the above rules are the *total enclosed area* of the corresponding objects: the counts obtained include the areas of all enclosed objects. This results in a blob area of 212 pixels and a hole area of 38 pixels. The black pixel count for the '2' is simply the difference: 174 pixels. In general, when all total enclosed areas have been computed, the strict black pixel area of a blob (or the



(a) Total Enclosed Blob Area

(b) Total Enclosed Hole Area

Figure 6: Computation of Total Enclosed Areas

strict white pixel area of a hole) can be found simply by subtracting from its total enclosed area that of all holes (blobs) immediately enclosed within it.

4.4.3 Computing Minimum Bounding Boxes During Edge Chaining

The *minimum bounding box* of an object is simply the topmost and bottommost rows and the leftmost and rightmost columns reached by this object. It is a simple matter to extract this information during the edge chaining process.

The *topmost row* of any object is obtained immediately as the starting row of its initial edge. To update the *bottommost row*, we need only consider the last edge point of edges followed downward. Obtaining the leftmost and rightmost pixels is more costly but we need only consider half the edge points in each case.

The *leftmost* pixel of a blob must necessarily be on the left of a black run which means that it must belong to an edge of type 1 or of type 4; conversely, the leftmost pixel of a hole must necessarily be on the right of a black run which means that it must belong to an edge of type 2 or of type 3. In all cases, when outer or inner contours are followed in the counter-clockwise direction, these edge types are followed *downwards*. Under the same conditions, we can similarly deduce that the *rightmost* pixel of any object must belong to an edge which is followed *upwards*. Hence, the leftmost column of an object may have to be updated only for edges followed downwards, and the rightmost only for edges followed upwards.

4.5 Correction of Several Image Defects

In our approach, the classification of numerals will be based on structural features extracted from their external contours. As much as possible, we must ensure that the truly significant features are captured, while meaningless small bumps and cavities resulting from digitization or writing circumstances are discarded. Many small image defects which unfavorably alter the outer contours can be detected and corrected during the edge extraction process, at little additional cost.

In addition, the presence of holes and their location is used as an important indication, leading the classification stage into preferred directions. Here again we must distinguish between significant and meaningless holes and some useful filtering can be performed during the edge extraction process.

The image defects which are dealt with below were identified by scrutiny of numeral samples from 2 databases: the CENPARMI database and the Quebec-Concordia database. The preprocessing operations described in sections 4.5.1 and 4.5.2 were carried over from Expert E4. All other preprocessing operations described below were developed for our new recognition system.

4.5.1 Edge Smoothing

Edge smoothing is a preprocessing operation carried out to get rid of small extraneous cavities facing up, down, left, or right. More precisely, these cavities must meet the following 2 criteria:

- in either a horizontal or vertical scan, at least one of the 2 edges defining these cavities has only 1 edge point;
- the cavities can be removed by filling or deleting a single pixel.

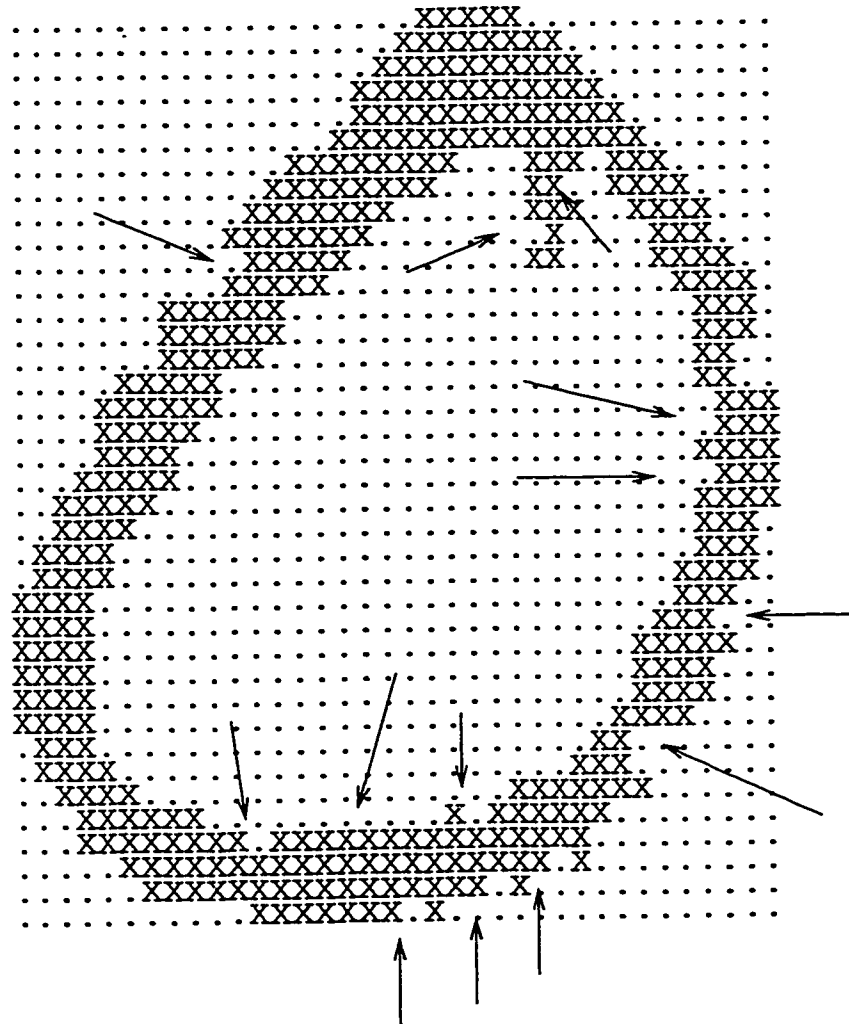


Figure 7: Random Extraneous Cavities

While an operation as simple as single-pixel filling or deletion is sufficient to get rid of these cavities, they represent the overwhelming majority of the insignificant features which tend to appear randomly along the contours of binary images. Several such cavities are indicated with arrows in Figure 7.

Furthermore, it is legitimate to assume that, when digitization is performed at an appropriate resolution, cavities which are not random but rather style-related should not disappear when single pixels are filled or deleted.

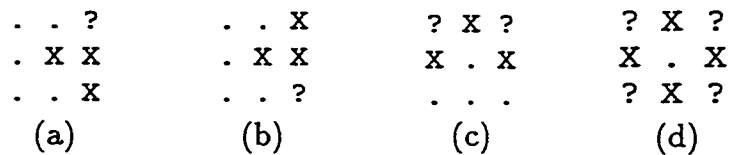


Figure 8: Single-Pixel Deletion and Filling Masks

Edge smoothing is performed based on the masks of Figure 8. In these figures, the ‘X’ represents a black (object) pixel, the ‘.’ represents a background pixel, and the ‘?’ represents either (i.e. a “don’t care” situation). When a 3×3 window matches the patterns of Figure 8 (a) or 8 (b) or any of the six equivalent patterns obtained with 90° , 180° , and 270° degree rotations of these patterns, the central pixel is deleted. Similarly, when a 3×3 window matches the pattern of 8 (c) or any of the three equivalent patterns obtained with 90° , 180° , and 270° degree rotations of this pattern, the central pixel is filled. Finally, when a 3×3 window matches the pattern of Figure 8 (d), the decision about filling the central pixel or not is delayed and its column position is recorded (explained in next section).

The result of applying the above smoothing masks is shown in Figure 9. In this figure, the empty little squares represent black pixels which were deleted.

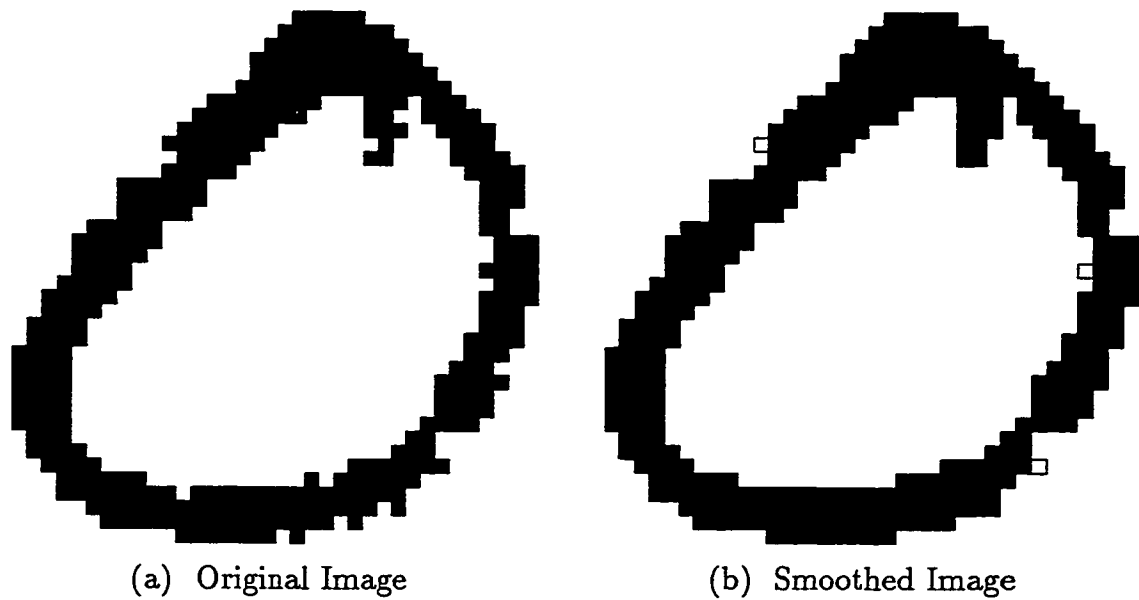


Figure 9: Edge Smoothing for '0' Displayed Above

Implementation

One obvious way to perform edge smoothing with the above masks is to perform a separate pass over the image, testing every black pixel for deletion and every background pixel for filling. However, the computation overhead in this approach is quite important. First, the input data is run-length encoded so we must 'unpack' it to obtain the rectangular binary array of the image; then we perform the edge smoothing pass; and finally we must re-extract the possibly modified black runs from the rectangular binary array in order to extract the edge points.

A much more efficient method consists in combining the edge smoothing operations with the edge extraction process in a single pass. To this end, information on three consecutive rows will be required at all times: row k , for which edge labels must be assigned to edge points; row $(k - 1)$, for which this task has already been performed; and row $(k + 1)$ for which this task will be performed later, but which must be unpacked already to process row k using the deletion and filling masks.

The chosen masks are such that the only black pixels that need to be examined for deletion are the edge points, i.e. the leftmost and rightmost pixels on every black

run; and their neighbouring background pixels are the only ones that should be tested for filling.

Furthermore, the masks to be tested can be narrowed down considerably, depending on the situation encountered.

Thus, the 2 templates of Figure 8 (a) and (b) can only match a 3×3 window centered on the *leftmost* pixel of a black run; similarly, the equivalent masks rotated by 180° need only be tested for a 3×3 window centered on the *rightmost* pixel of a black run; the equivalent masks rotated by 90° or 270° need only be tested for black runs of length 1⁴.

As for the filling, the mask of Figure 8 (c), or the equivalent one rotated by 180° , requires a white run of length 1 as a prerequisite condition; the mask obtained with a 90° counter-clockwise rotation need only be tested for a 3×3 window centered on the background pixel immediately following the *rightmost* pixel of a black run; the one obtained with a 270° counter-clockwise rotation need only be tested for a 3×3 window centered on the background pixel immediately preceding the *leftmost* pixel of a black run.

Effectiveness and Efficiency

In Legault & Suen [94], four edge smoothing methods, based on differing masks and different implementations, were compared in terms of their computation overhead and their ability to eliminate small random cavities while preserving style-related cavities. The tests were carried out using 200 digits of the CENPARMI database, 20 from each numeral class. Without any edge smoothing and using 4-connectivity, 148 random cavities, 116 non-random (style-related) cavities, and 33 open loops were counted on these samples.

In the study, the edge smoothing method described in this section was called 'Method #4'. It proved to be the most effective, removing 91 % of the small random cavities and preserving 71% of the style-related ones. Furthermore, the implementation proposed above provided a very sizeable gain in efficiency. For more information on this investigation, see Legault & Suen [94], pages 19–26.

⁴ Since the images are run-length encoded, this information is immediately available.

4.5.2 Hole and Cavity Opening

The investigation of Legault & Suen [94] also indicated that 13 of the 33 observed open loops could be corrected to ‘regular holes’ simply by using 8-connectivity instead of 4-connectivity. However this leads to new problems: among the 200 test samples, 3 cases of holes split into 2 or more holes were found; in 4 other instances, a hole is created by 8-connectivity where none is normally expected. For an example of the first kind of problem, see Figure 10 (a); here the hole in the ‘6’ is split into 4 holes of area 2 pixels, 6 pixels, 2 pixels, and 1 pixel respectively.

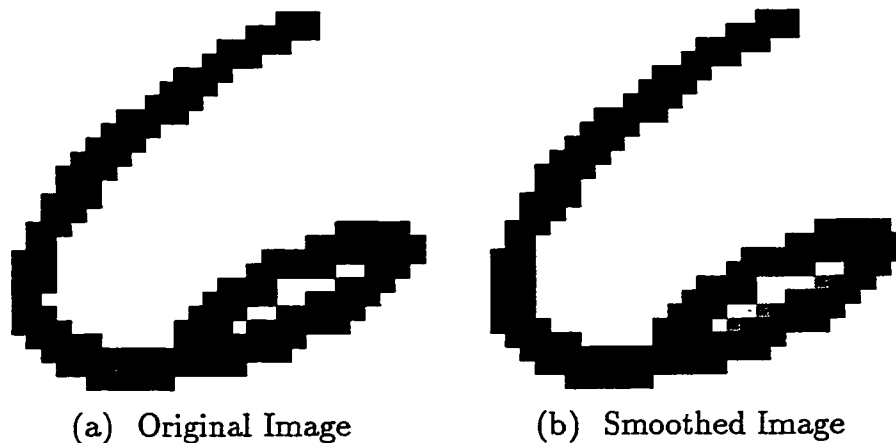


Figure 10: Hole Split Into Many Holes by 8-Connectivity

For an example of the second kind of problem, see Figure 12 (a): here the top hole in the ‘9’ is saved by 8-connectivity in rows 14–15; however, in rows 19–20, 8-connectivity creates a spurious hole of area 3 pixels.

A simple technique can be used, again concurrently with edge extraction, which solves most of the problematic cases. The solution consists in detecting instances of the situation depicted in Figure 11:

Three conditions must be met:

1. Consecutive black runs on one row, marked A and B, merge into another black run on the following row, marked D;
2. The leftmost pixel of black run D is one column to the right of the rightmost pixel

```

AAAAAA . .BBBBB
CCCCC . . .DDDDD

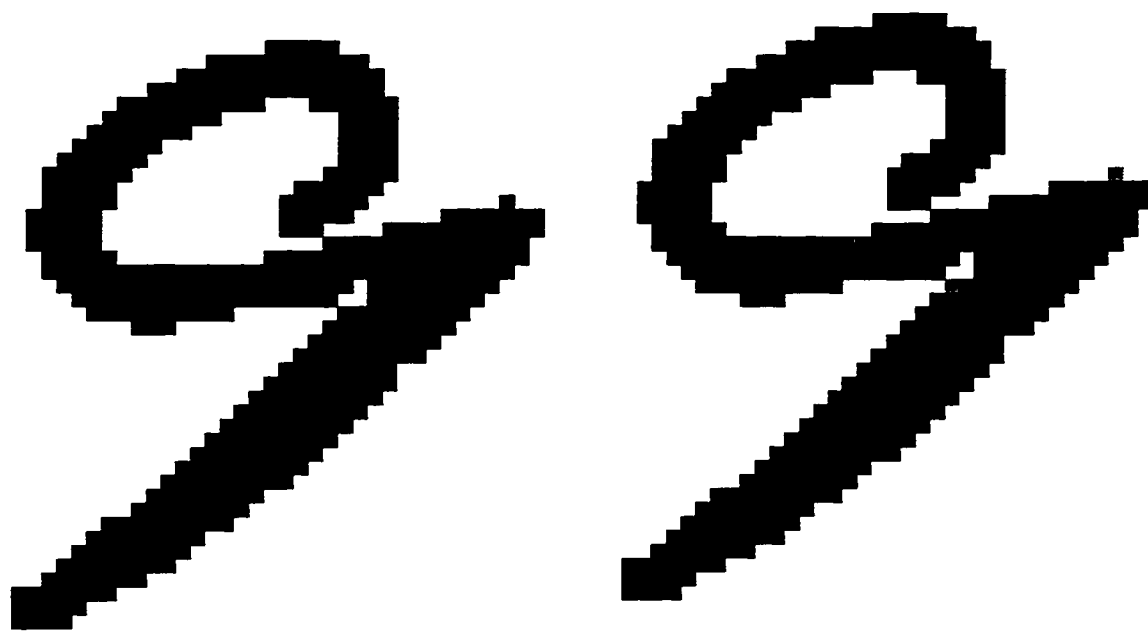
```

Figure 11: Opening Narrow Hole or Downward Cavity

of black run A;

3. Black run A further splits into two black runs, marked C and D.

When these 3 conditions are met, the leftmost pixel of component D is deleted. This cancels both the merging of black runs A and B and the splitting of black run A. The list of edges and edge points must be changed accordingly. The results of this operation on the 2 problematic cases are shown in Figures 10 (b) and 12 (b) respectively where the removed pixels are shown in gray.



(a) Original Image

(b) Smoothed Image

Figure 12: Spurious Hole Created by 8-Connectivity

When applied to the 2000 samples of training set A, this procedure prevented the

splitting of holes or creation of spurious holes for 32 samples (1.6%). It appears to be especially useful for 6's and 9's, fixing respectively 11 and 6 of them, out of 200.

We can now explain why edge smoothing by filling is delayed when a 3×3 window matches the template of Figure 8 (d). When processing the central row of this mask, it is not yet known whether the black run on the left of the middle pixel will split into 2 black runs in the next row; hence, we record the column number of the central pixel and wait until the following row is processed to make the final decision. If the conditions listed above are fulfilled, the pixel immediately below the central pixel is deleted, otherwise the delayed filling is performed on the central pixel.

4.5.3 Extra Filling on First and Last Rows

Strokes written in pencil or ink are often fainter at their edges than in their center. Hence, for horizontal strokes at the very top or very bottom of the image, thresholding for binarization may cause, in the first and last rows, gaps in black runs which are wider than 1 pixel. This can lead to spurious cavities unrelated to the writer's style.

The solution used is to allow, on the first and last rows only, the 'reconnection' of consecutive black runs, separated by up to 3 background pixels. The maximum amount of filling permitted, *maxfill*, actually depends on *n_cols*, the number of columns in the image. The exact formula is:

$$\text{maxfill} = \min(3, \lfloor (n_cols + 5)/10 \rfloor)$$

Examples of this operation are shown in Figure 13. On the first row of the '7', 2 white runs of lengths 2 and 3 respectively are filled. On the last row of the '8', a white run of length 3 is filled; this combined with regular single-pixel filling on the row before last, totally repairs the bottom of that image.

4.5.4 Repairing Faulty Scanlines

Occasionally, the scanner misses a scanline completely or almost completely. Empty rows or rows containing only single-pixel black runs are considered candidates for this defect. When such rows are detected, the black runs in the preceding and the

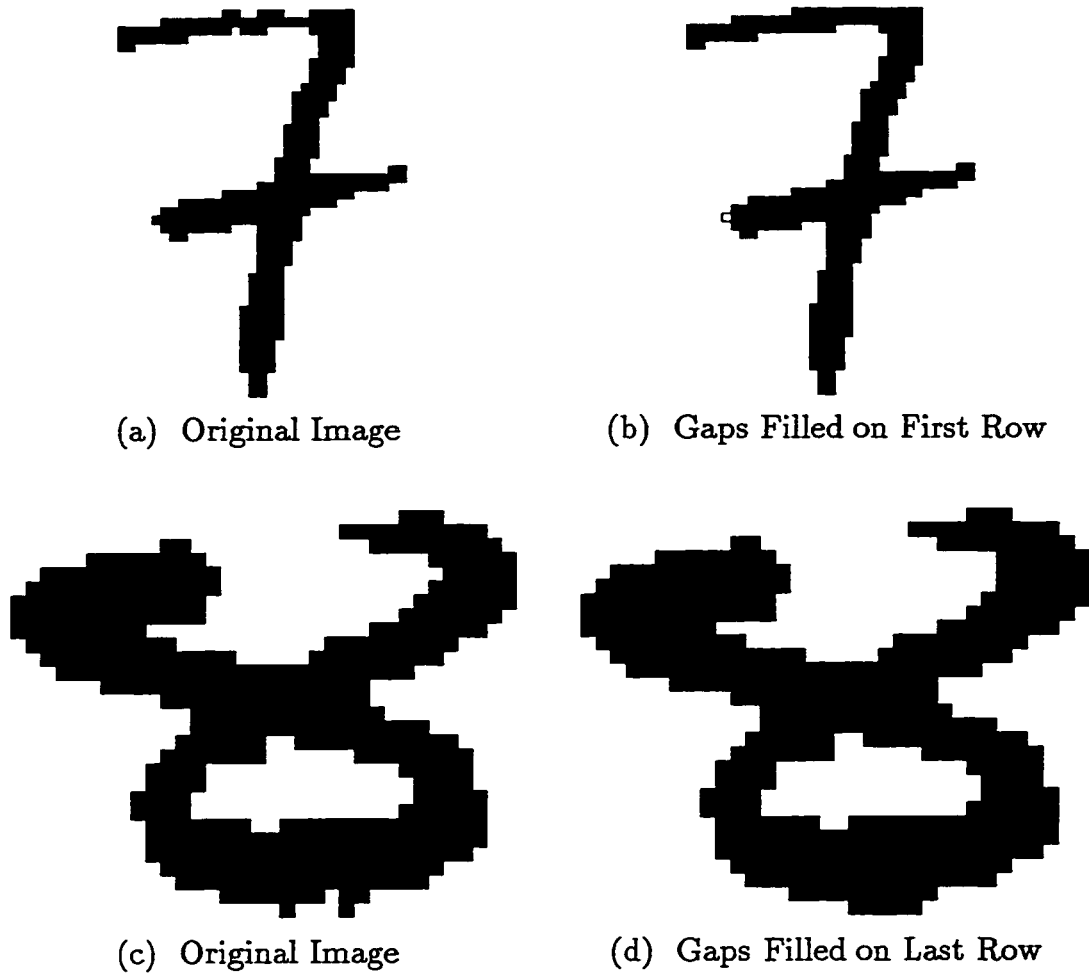


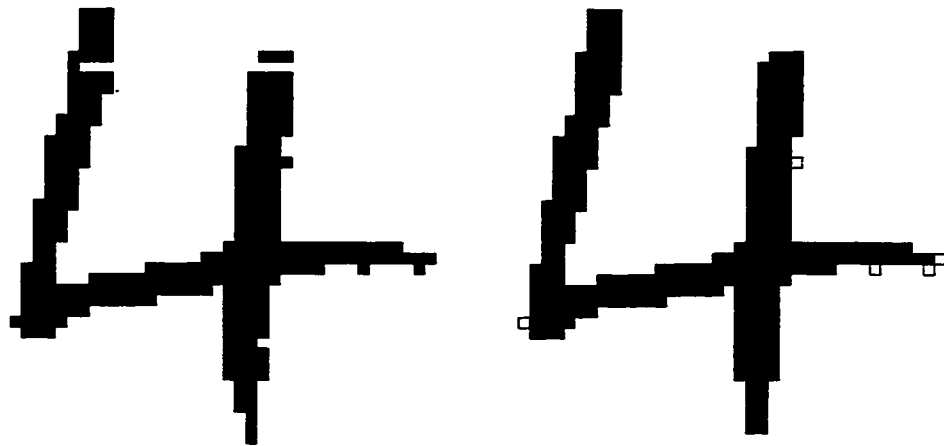
Figure 13: Extra Filling on First and Last Rows

following rows are examined to see if they could potentially be connected. If so, the black runs of the middle faulty row are completely re-built.

Let the faulty scanline be row r_1 . Now suppose that the left edge point of a black run in row $(r_1 - 1)$ and column c_1 , is seen as connected to the left edge point of a black run in row $(r_1 + 1)$ and column c_2 ; then the left edge point of the newly-built corresponding black run in row r_1 will be in column $(c_1 + c_2)/2$. And similarly for a right edge continuation.

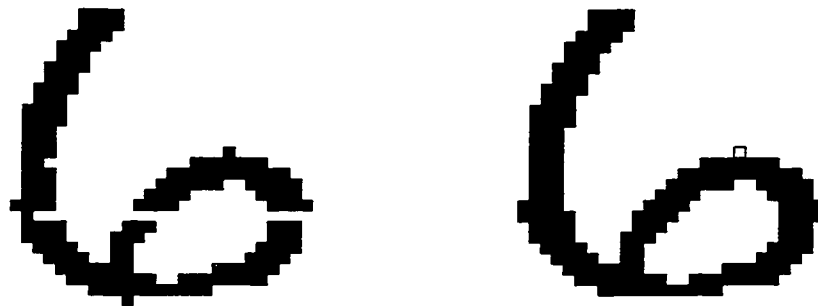
It is also possible that black run splitting or merging must be considered. For example, suppose a black run in row $(r_1 - 1)$ extends from column c_1 to column

c_2 ; and that it is seen as connected to *two* black runs in row $(r_1 + 1)$, extending respectively from column c_3 to column c_4 and from column c_5 to column c_6 . The rebuilding would then involve the creation a *one* new black run in row r_1 , extending from column $(c_1 + c_3)/2$ to column $(c_2 + c_6)/2$. Black run merging is handled in a similar fashion.



(a) Original Image

(b) Scanlines 6, 39 and 40 Rebuilt



(c) Original Image

(d) Scanline 20 Rebuilt

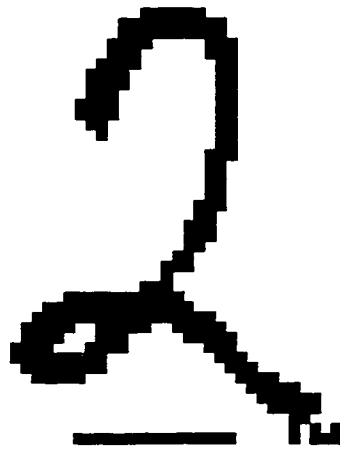
Figure 14: Rebuilding of Missing or Faulty Scanlines

Examples of this problem are shown in Figure 14. In Figure 14 (a), the obvious defects in row 6 are corrected; in addition, the two rows before the last row, rows 39 and 40, contain only single-pixel black runs and are also candidates for rebuilding.

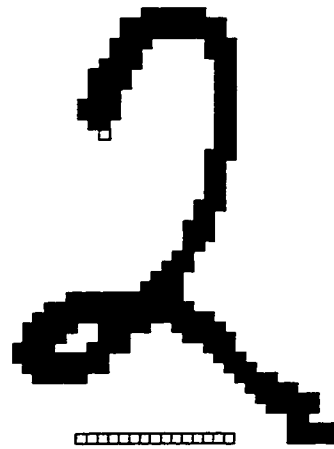
The result is a thickening of the narrow 1-pixel wide ending of the bottom stroke. Note also the apparent effect of simple edge smoothing operations, in particular the removed pixels (including the pixel on the very last row, not displayed as a small empty square).

4.5.5 Removing Isolated or Near-Isolated Black Runs

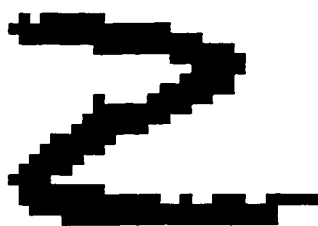
Another defect is the presence of an isolated black run, not connected to any other black run as in Figure 15 (a). This is easily detected as 2 edges which are created together and then immediately ended together, with a single edge-point each.



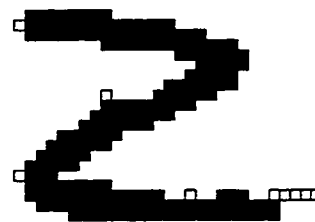
(a) Original Image



(b) Isolated Black Run Removed



(c) Original Image



(d) Near-Isolated Black Run Removed

Figure 15: Removing Isolated or Near-Isolated Black Runs

In other cases, a black run may be connected to another, but just barely, as shown at the bottom right of the '2' in Figure 15 (c). This may also be caused by some scanner defect and can be corrected easily.

4.5.6 Trimming Protruding Black Runs

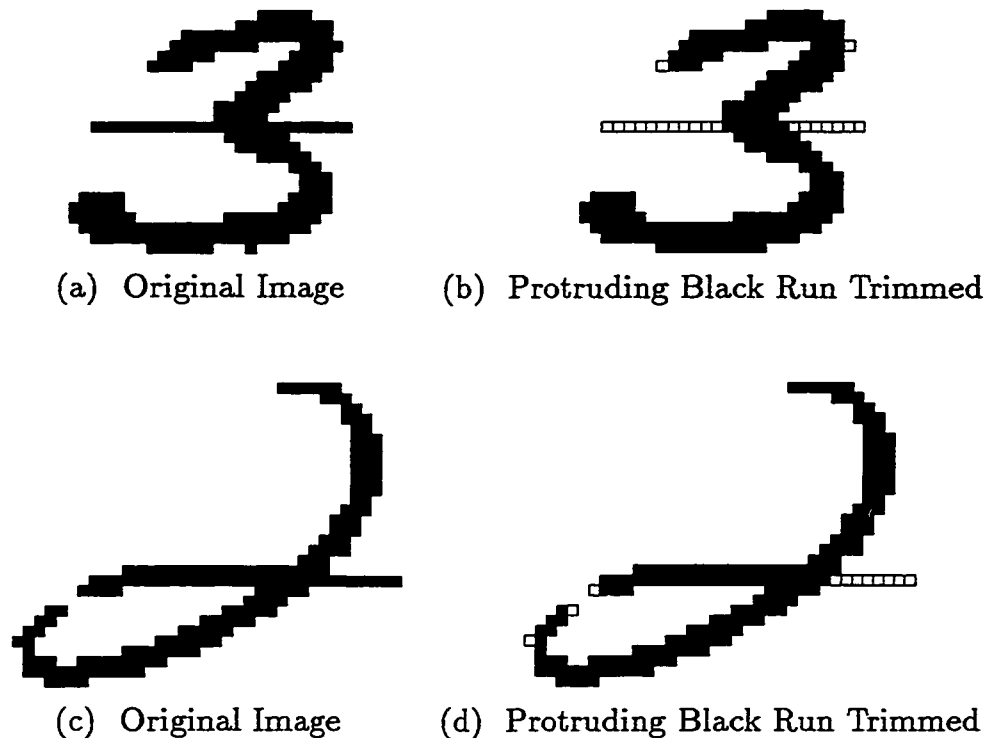


Figure 16: Trimming Protruding Black Runs

Another defect, probably due to scanning devices, is the occasional protrusion of a black run beyond its expected limits. This can be detected by comparing the column position of an edge point with the column positions of the connected edge points on the preceding and following rows. When this comparison reveals an excess of at least 2 pixels on the current row, trimming will take place. In the most general case, the new column position for the trimmed edge point will be the average of the column positions of the connected edge points on the preceding and following rows.

An example is shown in Figures 16 (a) and 16 (b).

Generally, the simple techniques which are used to detect and repair the image defects we are examining in this section will be beneficial. But on occasion, of course, they may not improve the image; they may even attenuate or remove some significant feature as shown in Figures 16 (c) and (d). In this case, the stroke in the region of interest is only 1 pixel wide, which produces the problem. If the imaging resolution is appropriately chosen, such a problem would only occur for exceptionally thin (or faint) characters.

4.5.7 Removing Edges of Length 1

As already mentioned in subsection 4.5.3, strokes tend to be fainter at their edges than at their center. In Figure 17 (a), this explains the bumpy lower side of the top horizontal stroke, which creates 3 insignificant down-facing cavities of depth 1.



(a) Original Image

(b) Length-1 Edges Removed

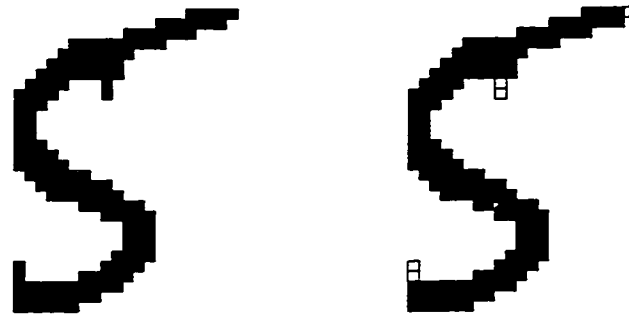
Figure 17: Removing Edges of Length 1

In our preprocessing program, we detect this problem by verifying the number of points belonging to an edge when it terminates or merges with another edge. In Figure 17 (a), the second and, after single-pixel filling, the third black run in the fifth row carry edges which terminate after a single edge point. To get rid of the resulting

parasitic cavities, we simply remove the guilty black runs as shown in Figure 17 (b). A similar situation can occur on the upper side of a horizontal stroke, when two newly-created edges associated with a single black run each merge with other edges in the very next row; the treatment applied is then the same: removing the black run responsible for the problem.

4.5.8 Removing Vertical Two-Pixel Stems

One last problem which is dealt with as the edge extraction proceeds from row to row is illustrated in Figure 18 (a). Here we see the presence of small (two-pixel long) downward and upward vertical stems. To avoid removing the meaningful endings of thin vertical strokes, we only remove such stems when they are attached to sufficiently wide black runs⁵.



(a) Original Image (b) Two-Pixel Stems Removed

Figure 18: Removing Vertical 2-Pixel Stems

In order to detect such defects, we now need information on 4 consecutive rows to be kept at all times. The result of the stem removal is shown in Figure 18 (b).

⁵ The exact criterion used is that such stems are only removed if the black runs to which they are attached are wider than one tenth of the image width.

4.5.9 Frequency of Defects

A careful investigation was conducted concerning the image defects described in subsections 4.5.2 to 4.5.8. For this study, we used the 2 000 samples of training set A from the CENPARMI database and the 5 000 samples of training set E from the Concordia-Montreal database (see section 2.4.2). The first set of data were digitized at a resolution of 166 PPI and the second set at 200 PPI.

In Table 10, we indicate the percentage of samples affected by every preprocessing operation discussed above. Concerning the repair of faulty scanlines, two numbers are given for each database. The first figure represents the percentage of samples affected by serious scanline defects. The second –much higher– figure also includes all samples which benefit from the broadening of 1-pixel wide strokes which comes as a side-effect of the repair procedure.

Operation	U.S. Data	Quebec data
Hole and Cavity Opening	6.9	0.7
Extra Filling on First/Last Rows	3.2	2.6
Repairing Faulty Scanlines	0.65 (5.1)	4.1 (10.7)
Removing (Near-)Isolated Black Runs	0.35	0.84
Trimming Protruding Black Runs	8.3	5.0
Removing Edges of Length 1	1.6	1.9
Removing Vertical 2-Pixel Stems	0.8	0.4

Table 10: Percentages of Samples Affected by Preprocessing Operations

The table shows that the relative frequency of some defects is database-dependent. For instance, the hole and cavity opening operation affects 10 times more samples from the US database than from the Quebec database. This is perhaps explained by the different digitization resolutions (166 PPI versus 200 PPI). Also, seriously damaged scanlines are 6 times more frequent for the Quebec database than for the US database. The scanner used to perform the data acquisition for the former database may have been ill-operating at times . . .

Chapter 5

Contour Smoothing

In the next chapter, we will describe our method for extracting regions of ‘significant curvature’ from the contours of binary images. This method begins with the computation of *deviation angles* ϕ_i (see Figure 21) from pixel to pixel along the contours. Without smoothing, ϕ_i only takes a few discrete values ($0, \pm 45^\circ, \pm 90^\circ, \pm 135^\circ, \pm 180^\circ$) with frequent local oscillations between these. This ‘wiggly’ nature of binary contours makes it difficult to extract curvature regions of significant extent directly from the ϕ_i ’s. To help solve this problem, smoothing can be applied.

In our first system E4, smoothing was performed¹ with a simple averaging method: for every point of the contour, its coordinates are replaced by averaging them with the coordinates of the preceding and following points. This was performed *twice* around the entire contour. Equivalently, we could have used a single-pass of local *weighted* averaging involving each point and its two immediate neighbours on each side, with weights $(\frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{2}{9}, \frac{1}{9})$. This is frequently referred to as a triangular filter.

The result is illustrated in Figure 19 for the external contour of a simple zero digit. In Figure 19 (a), line segments connect the grid points associated with the pixels of the original binary contour; the smoothed contour points are superimposed. In Figure 19 (b), only the smoothed points are presented.

¹ The implementation in our new system is quite different from what is described here and will be explained in the next chapter.

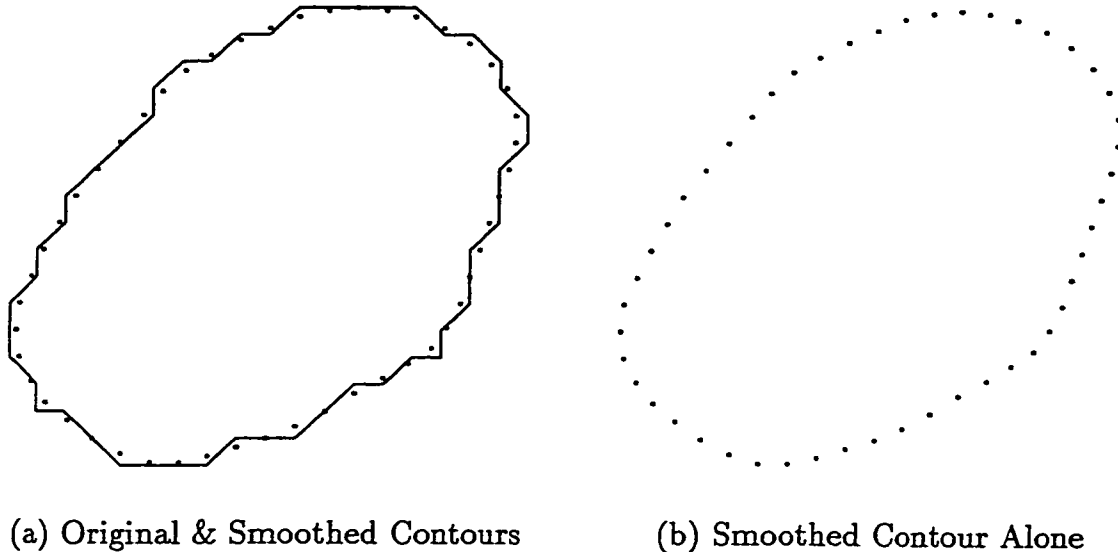


Figure 19: Contour Smoothing With Triangular Filter.

Satisfactory results were obtained from this simple smoothing procedure. Nevertheless, we were curious about the optimality of our choice of local weights. Initial review revealed widespread use of smoothing by local weighted averaging in practical applications, with quite diversified sets of weights. However, little theoretical investigation of these methods *per se* has been conducted. Thus, even if this represents a minor aspect of our overall numeral recognition system, we decided to conduct an analytical and experimental investigation into this matter.

Contour smoothing normally belongs to the preprocessing stage, discussed in the preceding chapter. However, in our recognition system, it is not performed explicitly, as a preprocessing operation; instead, it is done mostly *implicitly* as part of the computation of the ϕ_i 's in the feature extraction stage. In addition, we conducted quite extensive work on this topic. For these reasons, we have chosen to report on it in a separate chapter.

In the next section, we present a brief overview of approaches and problems related to the smoothing problem. Then, in section 5.2, we introduce local weighted averaging methods and offer a simple geometric interpretation. In section 5.3, the simple

model of a noisy horizontal border is used to derive *optimal* values of the smoothing parameters, in view of specific computational goals. Finally, the applicability of our findings for varying curvature is explored experimentally in section 5.4.

5.1 Contour Smoothing: A Brief Overview

There are numerous applications involving the processing of 2-D images, and 2-D views of 3-D images, where binary contours are used to represent and classify patterns of interest. Measurements are then made using the contour information (e.g. perimeter, area, moments, slopes, curvature, deviation angles etc.). To obtain reliable estimates of these quantities, one must take into account the noisy nature of binary contours due to discrete sampling, binarization, and possibly the inherent fuzziness of the boundaries themselves². In some cases, this can be done explicitly and exhaustively (see Worring & Smeulders [167] on curvature estimation). But more frequently it is done implicitly by smoothing. Following this operation, the measurements of interest can be obtained directly from the smoothed contour points, as in our case, or from a curve fitted to these points. For a recent example of this last approach, see Tsai & Chen [163].

In the rest of this chapter, the following definitions will be used. Let $\mathbf{p}_i = (x_i, y_i)$, for $i = 1, 2, \dots, N$, be the sequence of N points (4- or 8-connected) around the closed contour. Since the contour is cyclic, $\mathbf{p}_{N+i} = \mathbf{p}_i$ and $\mathbf{p}_{1-i} = \mathbf{p}_{N+1-i}$. Furthermore, let $\mathbf{v}_i = \mathbf{p}_i - \mathbf{p}_{i-1}$, and θ_i be the counter-clockwise elevation angle between \mathbf{v}_i and the horizontal x -axis. We have $\theta_i = c_i \cdot \frac{\pi}{4}$ where c_i is the Freeman [52] chain code (see Figure 20). For 4-connectivity, the values of c_i are limited to even values. We also define d_i , the differential chain code, as $d_i \equiv (c_{i+1} - c_i + 11) \bmod 8 - 3$, and the deviation angle $\phi_i \equiv \theta_{i+1} - \theta_i$ (see Figure 21).

Finally, the local weighted averaging method investigated in sections 5.2, 5.3, and 5.4 is defined as:

$$\mathbf{p}_i^{(k)} = \sum_{j=-n}^n \alpha_j \mathbf{p}_{i+j}^{(k-1)}, \quad k = 1, 2, \dots, K \quad (8)$$

² For example, the “borders” of strokes in handwriting.

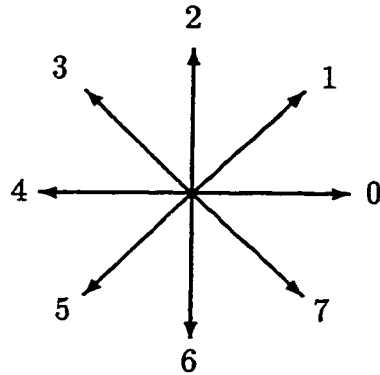
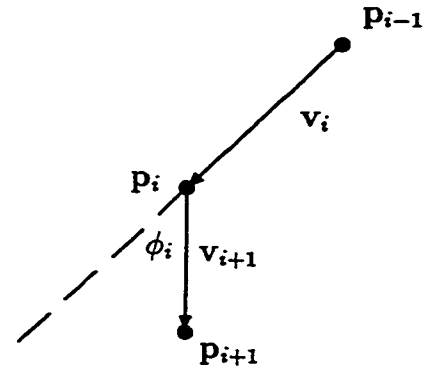


Figure 20: Freeman Chain Code

Figure 21: Deviation Angle at p_i

where $p_i^{(k)}$ is the contour point i after k smoothing steps ($p_i^{(0)} = p_i$) and the α_j 's are the smoothing coefficients. The window size is $w = (2n + 1)$.

5.1.1 Variety of Approaches and Related Problems

We now present various smoothing approaches used in pattern recognition applications. We will briefly consider their theoretical foundations and implementation difficulties. For a more detailed account of these aspects, see Legault & Suen ([96] and [97]).

Local Weighted Averaging

Due to limited computing power, early methods were quite simple and found justification in their "good results". Thus we find schemes removing/filling one-pixel wide protrusions/intrusions based on templates, or replacing certain pairs in the chain code sequence by other pairs or by singletons ([141]). However, from early on, local weighted averaging methods are the most frequently used. They are applied to differential chain codes ([44], [51], [56]), possibly with compensation for the anisotropy of the square grid ([112]); they are applied to cartesian coordinates ([13]), possibly with weights depending on neighbouring pixel configuration ([70]) or varying with successive iterations ([38]); they are applied to deviation angles ([45]).

Even today, in many practical applications, the smoothing operation is still performed by some local weighted averaging schemes because they are simple, fast, and effective (see for example [9], [19], [157], [159], [169]).

With advances in computing power and insight into the smoothing problem, more complex methods were developed with more solid theoretical foundations. In this process, “Gaussian smoothing” has become very popular (see next heading). One approach consists of applying local weighted averaging with Gaussian weights. For examples, see [9], [39], and [126].

Gaussian Smoothing and Multiscale Image Representation

Variable amounts of smoothing can be applied to the entire curve, taking the overall behaviour of the smoothed curve across scale as its complete description. When little or no smoothing is applied, all fine details of the image (plus noise) are retained; when more smoothing is applied, only the most salient features are preserved. The smoothing can be performed with various filters, the most popular being the Gaussian.

Witkin [166] convolves a signal $f(x)$ with Gaussian masks over a continuum of sizes:

$$F(x, \sigma) = f(x) * G(x, \sigma) = \int_{-\infty}^{\infty} f(u) \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-u)^2}{2\sigma^2}} du. \quad (9)$$

$F(x, \sigma)$ is called the *scale-space image* of $f(x)$ and it is analyzed in terms of its inflection points. For other results concerning scale-space and Gaussian smoothing, see Asada & Brady [10], Koenderink [74], Mokhtarian & Mackworth [115], and Wuescher & Boyer [169].

However, multiscale shape representations are not necessarily associated with Gaussian smoothing. For scale-space based on so-called *adaptive smoothing*, see Saint-Marc et al. [140]. See also the works of Maragos [108], Chen & Yan [29], and Bangham et al. [15] for multiscale shape representations based on non-linear filters.

Theoretical Foundations

Regularization theory and the study of scale-space kernels are the two main areas which have provided insight into the special qualities of the Gaussian kernel for

smoothing purposes. As will be seen, they do not warrant unqualified statements about the ‘optimality’ of Gaussian smoothing which are often encountered in pattern recognition literature.

Consider a one-dimensional function $g(x)$, corrupted by noise $\eta(x)$. The observed signal is then $y(x) = g(x) + \eta(x)$. Assume that the information available is a sampling of this signal y_1, y_2, \dots, y_n obtained for $x = x_1, x_2, \dots, x_n$. Here $x_i < x_{i+1}$. One approach to estimating $g(x)$ is to find $f(x)$ which minimizes

$$\frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2 + \lambda \int_{x_1}^{x_n} [f^{(m)}(u)]^2 du, \quad (10)$$

where λ is the *regularization parameter*. The solution is a *smoothing spline* of order $2m$ (see [131], [144]).

For *equally spaced data* and $m = 2$, Poggio et al. [128] have shown that the cubic spline solution is *very similar* to a Gaussian. Canny’s paper on edge detection [23] is also cited to support the optimality of Gaussian filtering. But the Gaussian is only an *approximation* to his theoretically obtained optimal filter.

Several authors have investigated smoothing kernels for multiscale representations of *continuous* signals. Depending on the imposed set of axioms representing the properties deemed desirable, the Gaussian filter may be single out as the optimal filter.

Babaud et al. [12] have considered the class of infinitely differentiable kernels $g(x, y)$ vanishing at infinity faster than any inverse of polynomial, and one-dimensional signals $f(x)$ that are continuous linear functionals on the space of these kernels. In this class, they have shown that only the Gaussian $g(x, y) = \frac{1}{\sqrt{2\pi}} y e^{-1/2(xy)^2}$ can guarantee that all first-order maxima (or minima) of the convolution

$$\phi(x, y) = f(x) * g(x, y) = \int_{-\infty}^{\infty} f(u) g(x - u, y) du \quad (11)$$

will increase (or decrease) monotonically as y increases. Yuille & Poggio [173] extended the previous result by showing that, *in any dimension*, the Gaussian is the only *linear* filter that does not create generic zero crossings of the Laplacian as the scale increases. For related results, see Anh et al. [7], Mackworth & Mokhtarian [104],

Pauwels et al. [125]³ and Wu & Xie [168].

For *discrete* signals, Lindeberg [101] found the unique one-parameter family of *scale-space kernels* with a *continuous* scale parameter; as the scale increases, the discretized Gaussian *approximates* these kernels better and better.

Practical Considerations

For practical applications, regardless of the smoothing method one decides to use, some concrete questions must eventually be answered. For the regularization approach, what value should be used for λ ? For Gaussian smoothing, what value of σ and what finite window size? When scale-space representation is used, if we say that significant features are those which survive over “a wide range of scale”, we must eventually put some actual figures on this ‘wide’ range. These decisions can be entirely data-driven or based on prior experience, knowledge of particular applications etc. Other practical issues may also need to be addressed such as the preservation of significant discontinuities in the contours and the problem of contour shrinkage caused by smoothing. In the end, they may play a significant role in both the performance of the selected method and its implementation cost. We now briefly consider some of these aspects.

In [145] [146], Shahraray & Anderson consider the regularization problem of equation 10, for $m = 4$, and they argue that finding the best value of λ is critical. For this purpose, they propose a technique based on minimizing the cross-validation mean square error (CVMSE):

$$\text{CVMSE}(\lambda) = \frac{1}{n} \sum_{k=1}^n (g_{n,\lambda}^{[k]}(x_k) - y_k)^2, \quad (12)$$

where $g_{n,\lambda}^{[k]}$ is the smoothing spline constructed using all samples *except* y_k , and is then used to estimate y_k . The method is said to provide a very good estimate of the best λ , *for equally-spaced periodic data assuming only a global minimum*. Otherwise, a so-called generalized cross-validation function must be used.

³ Here it is shown that imposing recursivity and scale-invariance on linear, isotropic, convolution filters is *not* restrictive enough to single out the Gaussian.

The presence of discontinuities to be preserved in the contours of interest brings more complexity into the optimal smoothing problem. One possible solution was already mentioned: the adaptive smoothing of Saint-Marc et al. [140]. For one-dimensional regularization which preserves discontinuities, see Lee & Pavlidis [85]. For two-dimensional regularization which preserves discontinuities, see Chen & Chin [28]. For Gaussian smoothing which preserves discontinuities, see the methods of Ansari & Huang [9] and of Brady et al. [21].

As already mentioned above, repeated convolution of a closed curve with a kernel may not yield a closed contour or may cause shrinkage. Different approaches have been considered to solve or attenuate this problem. See Horn & Weldon [64], Lowe [103], Mackworth & Mokhtarian [104], Oliensis [124], and Wheeler & Ikeuchi [164].

Li & Chen [100] investigate a possible solution to the high computation and storage requirements of generating “continuous” scale-space. They show that an optimal \mathcal{L}_1 approximation of the Gaussian can be obtained with a finite number of basis Gaussian filters:

$$G(x, \sigma) \approx \sum_{i=1}^n w_i(\sigma) G(x, \sigma_i) \quad (13)$$

from which scale-space can be constructed efficiently.

The above discussion, albeit brief, exemplifies the potential complexity involved in implementing ‘optimal’ methods. Clearly, in practice, one should not lose track of the cost of these operations and how much smoothing is really required by the application of interest. It is not always necessary to attain the ultimate precision in every measurement. In many situations, simple and fast methods such as local weighted averaging with fixed weights and a small window size, will provide a very satisfactory solution in only 2 or 3 iterations (see [38], [44], [45]). Moreover, there is often little difference in the results obtained via different methods. Thus, Dill et al. [39] report similar results when a Gaussian filter and a triangular (Gallus-Neurath) filter of the same width are applied to differential chain codes; in Kasvand & Otsu [72], rectangular, triangular, and Gaussian kernels, with the same standard deviation, yield comparable outcomes (especially the latter two) for the smooth reconstruction of planar curves from piecewise linear approximations.

5.1.2 Our Own Work

In the rest of this chapter, we assume local weighted averaging with constant weights *as a starting point* and we investigate how these smoothing methods handle small random noise. To this end, we propose a simple model of a noisy horizontal border. The simplicity of the model allows a very pointed analysis of these smoothing methods. More precisely, for specific computational goals such as estimating contour point positions, derivatives (slopes of tangents), or deviation angles from the pixels of binary contours, we answer the following questions: what are the optimum fixed weights for a given window size? and what fraction of the noise is actually removed by these optimum weights?

After deriving these results, we offer experimental evidence that their validity is not restricted to the limited case of noisy horizontal borders. This is done by considering digital circles. For each particular computational task, we find very close agreement between the optimum weights derived from our simple model and the ones derived numerically for circles over a wide range of radii.

An important side-result concerns the great caution which should be exercised in speaking of ‘optimal’ smoothing. Even for our simple idealized model, we find that the smoothing coefficients which best restore the original noise-free pixel *positions* are not the same which best restore the original local *slope*, or the original local *deviation angles*; furthermore, the best smoothing coefficients even depend on the specific difference method used to numerically estimate the slope. Hence, in choosing smoothing methods, researchers should probably first consider *what* it is they intend to measure after smoothing and *in what manner*.

In relation to this, we point out the work of Worring & Smeulders [167]. They analyze noise-free digitized circular arcs and exhaustively characterize *all centers and radii* which yield a given digitization pattern; by averaging over all these, an optimum measure of *radius* or *curvature* can be obtained. If radius or curvature is the measurement of interest and if utmost precision is required (with the associated computing cost to be paid), then their approach is most suitable.

Our work in contrast is not oriented towards measuring a single attribute. We focus on such measurements as position, slope and deviation angles because they are

often of interest in pattern recognition. But our model and approach can be used to investigate other quantities or other numerical estimates of the same quantities. The methods may be less accurate but they will be much less costly, and optimum in the category of local weighted averaging methods. The requirements of specific applications should dictate what is the best trade-off.

5.2 Local Weighted Averaging

We begin the investigation of local weighted averaging, as defined by equation 8, where n neighbours are considered on each side of \mathbf{p}_i . The window size of the operation is then $w = 2n + 1$. Of course, the smoothed contour points $\mathbf{p}_i^{(k)}$, after k smoothing iterations, can be obtained directly from the original points \mathbf{p}_i as

$$\mathbf{p}_i^{(k)} = \sum_{j=-n'}^{n'} \beta_j \mathbf{p}_{i+j}, \quad (14)$$

where $n' = kn$, corresponding to a window size $w' = k(w - 1) + 1$, and the β 's are functions of the α 's and of k . The form of Equation 8 is often computationally more convenient. However, as long as k and n are finite, the study of local weighted averaging need only consider the case of a single iteration with *finite* width filters. When this is done, we will use the simpler notation \mathbf{p}'_i instead of $\mathbf{p}_i^{(1)}$:

We now impose a simple requirement to this large family of methods. Since our goal is to smooth the small 'wiggles' along boundaries of binary images, it seems reasonable to require that when \mathbf{p}_i and its neighbouring contour pixels are perfectly aligned, the smoothing operation should leave \mathbf{p}_i unchanged. In particular, consider the x -coordinates of consecutive horizontally-aligned pixels from \mathbf{p}_{i-n} to \mathbf{p}_{i+n} . For $j \in [-n, n]$, we have $x_{i+j} = x_i + j$. Our requirement that $x'_i = x_i$ then becomes

$$x'_i = \sum_{j=-n}^n \alpha_j (x_i + j) = x_i \sum_{j=-n}^n \alpha_j + \sum_{j=1}^n (\alpha_j - \alpha_{-j}) j = x_i. \quad (15)$$

For this to hold for any value of x_i , we must have

$$\sum_{j=-n}^n \alpha_j = 1; \quad \text{and} \quad \alpha_{-j} = \alpha_j, \quad j = 1, 2, \dots, n \quad (16)$$

Thus our requirement is equivalent to a normalization condition and a symmetry constraint on the α 's.

5.2.1 Geometric Interpretation

Starting from equation 8 and using the above conditions, it is a simple matter to find a geometric interpretation for local weighted averaging. We can write:

$$\begin{aligned}
 \mathbf{p}_i^{(k)} &= \sum_{j=-n}^n \alpha_j \mathbf{p}_{i+j}^{(k-1)} \\
 &= \alpha_0 \mathbf{p}_i^{(k-1)} + \sum_{j=1}^n \alpha_j (\mathbf{p}_{i-j}^{(k-1)} + \mathbf{p}_{i+j}^{(k-1)}) \\
 &= (1 - 2 \sum_{j=1}^n \alpha_j) \mathbf{p}_i^{(k-1)} + \sum_{j=1}^n \alpha_j (\mathbf{p}_{i-j}^{(k-1)} + \mathbf{p}_{i+j}^{(k-1)}) \\
 &= \mathbf{p}_i^{(k-1)} + \sum_{j=1}^n 2\alpha_j \left[\frac{\mathbf{p}_{i-j}^{(k-1)} + \mathbf{p}_{i+j}^{(k-1)}}{2} - \mathbf{p}_i^{(k-1)} \right]. \tag{17}
 \end{aligned}$$

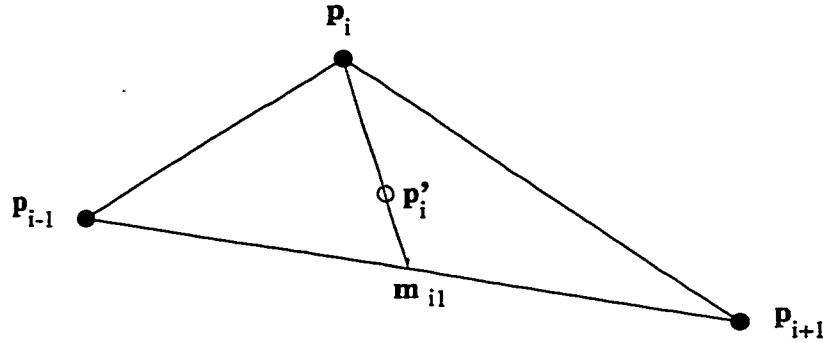


Figure 22: Geometric Interpretation for $w = 3$.

For a single iteration ($k = 1$) of the simplest method ($n = 1$), the last equation reduces to

$$\mathbf{p}'_i = \mathbf{p}_i + 2\alpha \left[\frac{\mathbf{p}_{i-1} + \mathbf{p}_{i+1}}{2} - \mathbf{p}_i \right]. \tag{18}$$

The points \mathbf{p}_{i-1} , \mathbf{p}_i , and \mathbf{p}_{i+1} are generally not aligned and the situation is illustrated in Figure 22, where $\mathbf{m}_{i1} = (\mathbf{p}_{i-1} + \mathbf{p}_{i+1})/2$ is the middle of the base of the

triangle. Equation 18 implies that the smoothed point \mathbf{p}'_i is always on the *median* of the triangle from point \mathbf{p}_i . Furthermore, the effect of the unique coefficient α is clear since $|\mathbf{p}_i\mathbf{p}'_i|/|\mathbf{p}_i\mathbf{m}_{i1}| = 2\alpha$. As α varies continuously from 0 to 0.5, \mathbf{p}'_i ‘slides’ from \mathbf{p}_i to \mathbf{m}_{i1} .

Similarly, in the more general situation, the vectors $[\frac{\mathbf{p}_{i-j}^{(k-1)} + \mathbf{p}_{i+j}^{(k-1)}}{2} - \mathbf{p}_i^{(k-1)}]$ are the *medians* from $\mathbf{p}_i^{(k-1)}$ of the triangles $\Delta\mathbf{p}_{i-j}^{(k-1)}\mathbf{p}_i^{(k-1)}\mathbf{p}_{i+j}^{(k-1)}$. Equation 17 indicates that the smoothed point $\mathbf{p}_i^{(k)}$ is obtained by adding to $\mathbf{p}_i^{(k-1)}$ a weighted sum of the medians of these triangles, using $2\alpha_j$ as weights. Thus, in a *geometric* sense, local weighted averaging as a contour smoothing method could be renamed *median smoothing*.

5.3 Optimum Results from a Simple Digitization Noise Model

This section addresses the question “If local weighted averaging is considered, what constant coefficients α_j should be used for smoothing binary contours in view of specific computational goals?”. We develop an answer to this question, based on a simple model: *an infinite horizontal border with random 1-pixel noise*.

Why use this model? Of course, we do not consider the horizontal line to be a very general object. Nor do we think that noise on any particular binary contour is a random phenomenon. We have noticed in our work that binary contours often bear small noise, commonly “1-pixel wiggles”. Our goal is to perform an *analytical study* of the ability of local weighted averaging smoothing methods to remove such noise. Since the filters are meant to be used with arbitrary binary contours, it seems reasonable to consider that over a large set of images noise can be considered random.

Furthermore, we do not make the very frequent implicit assumption that a smoothing filter can be optimal independently of the specific attributes one intends to measure or even the specific numerical estimation method used. For specific measurements and computation methods, we would like to find the best choice of smoothing coefficients for a given window size and an estimate of how much noise these coefficients

remove; if the window size is increased⁴, what are then the best coefficients and how much more is gained compared to the smaller window size?

These questions are very pointed and we have no workable expression for small random noise on an *arbitrary* binary contour which would allow to derive answers *analytically*. Thus we choose to look at an ideal object for which we can easily model random 1-pixel noise so that our study can be carried out. Similar approaches are often followed. For example, in studying optimal edge detectors, Canny [23] considers the ideal step edge. There is no implication that this is a common object to detect in practice; simply it makes the analytical investigation easier and can still allow to gain insight into the edge detection problem more generally. The practicality of our own findings concerning optimal local weighted averaging will be verified in section 5.4. We now give a definition for our simple model.

The infinite horizontal border with random 1-pixel noise consists of all points (x_i, y_i) , $i \in \mathcal{Z}$, satisfying

$$x_i = i, \quad \forall i \in \mathcal{Z} \quad (19)$$

$$y_i = \begin{cases} y_0, & \text{with probability } (1 - p) \\ y_0 + 1, & \text{with probability } p \end{cases} \quad (20)$$

An example of such a simple noisy boundary is shown in Figure 23.

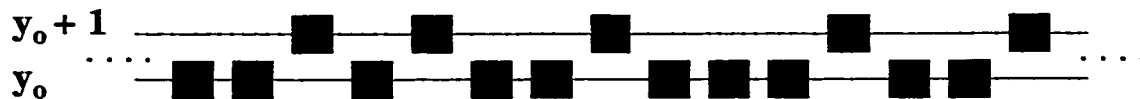


Figure 23: Noisy Horizontal Border

With this model, $x_{i-j} = x_i - j$ and $x_{i+j} = x_i + j$. It then follows from Equation 17 that the smoothing operation will not change the x -coordinates and the local weighted averaging will only affect the y -coordinates. The best fitting straight line

⁴ Assuming the feature structure scale allows this.

through this initial data is easy to obtain since it must be of the form $y = \tilde{y}$. It is obtained by minimizing the mean square distance

$$\overline{d^2} = (1 - p)(\tilde{y} - y_o)^2 + p(\tilde{y} - y_o - 1)^2 \quad (21)$$

with respect to \tilde{y} . The best fitting line is simply $y = y_o + p$. We will consider this to be the Equation of the ideal border which has been corrupted by the digitization process, yielding the situation of Equation 20.

We now examine the problem of applying “optimal” local weighted averaging to the data of our simple model. Our aim is to eliminate the ‘wiggles’ along the noisy horizontal border as much as possible. An alternate formulation is that we would want the border, after smoothing, to be “as straight as possible” and as close as possible to $y = y_o + p$.

Several criteria can be used to assess the straightness of the border and optimize the smoothing process:

- Minimize the mean square *distance* to the best fitting line after the data has been smoothed.
- Minimize the mean square *slope* along the smoothed data points (ideally, the border is straight and its slope should be 0 everywhere).
- Minimize the mean square *deviation angle* ϕ_i (see Figure 21) along the smoothed data (ideally, ϕ_i should also be 0 everywhere).

Each of the above criteria is sound and none can be said to be the *best* without considering the particular situation further. The first criterion is the most commonly used in the curve fitting literature. In this paper however, we want to derive optimal smoothing methods *tailored for specific computational tasks*; hence, we will consider each of the above criteria in turn. If our interest is simply to obtain numerical estimates of the *slopes* at contour points, the optimal α_j 's derived based on the second criterion should be preferred. And for estimating *deviation angles* ϕ_i , the optimal coefficients derived from the third criterion would be better.

For the first criterion, we will use d_{rms} , the root mean square (r.m.s.) distance to the best fitting line, as our measure of noise before and after the smoothing step; for

the second criterion, m_{rms} , the r.m.s. slope along the border; with the third criterion, ϕ_{rms} , the r.m.s. deviation angle along the border. For the original *unsmoothed* data, these noise measures can be computed using the probabilities of the possible configurations of 2 or 3 consecutive pixels. The derivation of ϕ_{rms} is given in Appendix B. For the original, unsmoothed data the results obtained are:

$$d_{rms} = \sqrt{p(1-p)} \quad (22)$$

$$m_{rms} = \begin{cases} \sqrt{2p(1-p)} & \text{using } m_i = y_{i+1} - y_i \\ \sqrt{\frac{p(1-p)}{2}} & \text{using } m_i = (y_{i+1} - y_{i-1})/2 \end{cases} \quad (23)$$

$$\phi_{rms} = \frac{\pi}{2} \sqrt{\frac{3p(1-p)}{2}} \quad (24)$$

Based on our simple model, we now derive the best smoothing parameters for each of the three criteria mentioned above. Once obtained, we will compute the corresponding noise measures for the *smoothed data* which we will denote by ${}_{[w]}d'_{rms}$, ${}_{[w]}m'_{rms}$, and ${}_{[w]}\phi'_{rms}$ respectively. In this notation the 'prime' indicates a single smoothing step and w is the window size used.

5.3.1 Best Parameters to Minimize d'_{rms}

The unsmoothed y -coordinate of our border points is a discrete random variable following a simple Bernouilli distribution for which the expected value is $y_o + p$ and the variance is $p(1-p)$. Obviously, the best fitting line is simply the expected value and d_{rms} in Equation 22 is the standard deviation of y_i . After smoothing, the expected value of y'_i will remain $y_o + p$ for *any* (finite) window size w . Denoting the expected value by E , we have from elementary probability theory:

$$E(y'_i) = E\left(\sum_{j=-n}^n \alpha_j y_{i+j}\right) = \sum_{j=-n}^n \alpha_j E(y_{i+j}) = \sum_{j=-n}^n \alpha_j E(y_i) = E(y_i). \quad (25)$$

Now we find the best choice of smoothing parameters and the corresponding noise measure ${}_{[w]}d'_{rms}$. Denoting variance by $\sigma^2()$, we have:

$$\overline{d_i'^2} = \sigma^2(y_i') = \sum_{j=-n}^n \sigma^2(\alpha_j y_{i+j}). \quad (26)$$

Each $\alpha_j y_{i+j}$ is a discrete random variable (with two possible values) and its variance is $p(1-p)\alpha_j^2$. Thus

$$\overline{d_i'^2} = p(1-p) \sum_{j=-n}^n \alpha_j^2. \quad (27)$$

We must now minimize $\sum_{j=-n}^n \alpha_j^2$, subject to the constraint $\alpha_0 + 2 \sum_{j=1}^n \alpha_j - 1 = 0$. This problem is typically solved using the Lagrange multipliers method (see [160], page 182). from which we obtain the simple result $\alpha_k = \alpha_0$, for each k . All coefficients are equal, hence of value $1/(2n+1)$. See derivation in Appendix C. Substituting this value into Equation 27, we obtain the corresponding $\overline{d_i'^2}$. Our findings can be summarized as follows:

- For a single smoothing iteration with arbitrary window size w , for *any* value of p , the best choice of parameters to minimize the mean square distance is to set all α_j 's to $1/w$, resulting in ${}_{[w]}d'_{rms} = \sqrt{\frac{p(1-p)}{w}}$.

The fraction of the noise which is removed by the smoothing operation is $1 - \sqrt{\frac{1}{w}}$. Hence, for $w = 3$, the noise is reduced by 42.3%; for $w = 5$, by 55.3%; for $w = 7$, by 62.2%. Finally, contrary to what one might expect, we note that the optimum 5-point smoothing operation is *not* to apply the optimum 3-point operation twice. The latter is equivalent to a 5-point window with $\alpha_j = (3 - |j|)/9$ which gives $d'_{rms} = \frac{1}{9}\sqrt{19p(1-p)}$. This would remove approximately 51.6% of the noise.

5.3.2 Best Parameters to Minimize m'_{rms}

In this section, we apply our second criterion for straightness and minimize the root mean square value of the *slope* after smoothing. We consider two different ways of computing the slope from contour points.

Based on $m'_i = y'_{i+1} - y'_i$

The simplest numerical estimate of the slope is given by the forward difference formula $m'_i = y'_{i+1} - y'_i$. Some algebraic manipulation (outlined in Appendix D) leads to an expression for $\overline{m_i'^2}$ involving only the n independent parameters $\alpha_1, \alpha_2, \dots, \alpha_n$:

$$\overline{m_i'^2} = 2p(1-p) \left[1 - 2\alpha_1 + 4(\alpha_1 - 1) \sum_{j=1}^n \alpha_j + 2 \sum_{j=1}^n \alpha_j^2 + 4 \left(\sum_{j=1}^n \alpha_j \right)^2 - 2 \sum_{j=1}^{n-1} \alpha_j \alpha_{j+1} \right]. \quad (28)$$

Differentiating this with respect to α_k , for $1 \leq k \leq n$, we obtain a system of n linear equations as shown below.

$$\begin{pmatrix} 10 & 5 & 6 & 6 & 6 & 6 & 6 & \cdots & 6 & 6 & 6 & 6 \\ 5 & 6 & 3 & 4 & 4 & 4 & 4 & \cdots & 4 & 4 & 4 & 4 \\ 6 & 3 & 6 & 3 & 4 & 4 & 4 & \cdots & 4 & 4 & 4 & 4 \\ 6 & 4 & 3 & 6 & 3 & 4 & 4 & \cdots & 4 & 4 & 4 & 4 \\ 6 & 4 & 4 & 3 & 6 & 3 & 4 & \cdots & 4 & 4 & 4 & 4 \\ 6 & 4 & 4 & 4 & 3 & 6 & 3 & \cdots & 4 & 4 & 4 & 4 \\ & & \vdots & & \vdots & & \vdots & & & & & \\ 6 & 4 & 4 & 4 & 4 & 4 & 4 & \cdots & 4 & 3 & 6 & 3 \\ 6 & 4 & 4 & 4 & 4 & 4 & 4 & \cdots & 4 & 4 & 3 & 6 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \vdots \\ \alpha_{n-1} \\ \alpha_n \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ \vdots \\ 2 \\ 2 \end{pmatrix} \quad (29)$$

Solutions are given in Table 11 for $1 \leq n \leq 6$. The column before last gives the fraction of the noise which is removed by the optimum smoothing method. For comparison purposes, the last column provides the equivalent result when all weights are set equal to $\frac{1}{w}$. Finally we note that for window size $w = 5$, the triangular filter using $\alpha_j = (3 - |j|)/9$ results in a noise reduction of 80.8%, slightly better than the equal-weights method.

n	α_0	$\alpha_{\pm 1}$	$\alpha_{\pm 2}$	$\alpha_{\pm 3}$	$\alpha_{\pm 4}$	$\alpha_{\pm 5}$	$\alpha_{\pm 6}$	$1 - \frac{m'_{rms}}{m_{rms}}$	$1 - \frac{m'_{rms}}{m_{rms}}$
								optimum	$\alpha_j = \frac{1}{m}$
1	$\frac{2}{5}$	$\frac{3}{10}$						0.684	0.667
2	$\frac{9}{35}$	$\frac{8}{35}$	$\frac{1}{7}$					0.831	0.800
3	$\frac{4}{21}$	$\frac{5}{28}$	$\frac{1}{7}$	$\frac{1}{12}$				0.891	0.857
4	$\frac{5}{33}$	$\frac{5}{35}$	$\frac{7}{55}$	$\frac{16}{165}$	$\frac{3}{55}$			0.922	0.889
5	$\frac{18}{143}$	$\frac{7}{286}$	$\frac{16}{143}$	$\frac{27}{286}$	$\frac{10}{143}$	$\frac{1}{26}$		0.941	0.909
6	$\frac{7}{65}$	$\frac{9}{455}$	$\frac{8}{91}$	$\frac{9}{91}$	$\frac{8}{455}$	$\frac{24}{455}$	$\frac{1}{35}$	0.953	0.923

Table 11: Best Parameters to Minimize $(y'_{i+1} - y'_i)^2$ and Fraction of Noise Removed

Based on $m'_i = \frac{1}{2}(y'_{i+1} - y'_{i-1})$

A more accurate⁵ estimate of the slope is given by $m'_i = \frac{1}{2}(y'_{i+1} - y'_{i-1})$. Expanding it in terms of the original coordinates and following the same approach as in Appendix D, we arrive at:

$$\overline{m_i'^2} = \frac{p(1-p)}{2} \left[\sum_{j=-n}^n \alpha_j^2 + \sum_{j=-n+1}^{n-1} \alpha_{j-1} \alpha_{j+1} \right]. \quad (30)$$

When expressed in terms of the n independent α 's, this becomes:

$$\begin{aligned} \overline{m_i'^2} = \frac{p(1-p)}{2} & \left[1 - 2\alpha_2 - \alpha_1^2 + 4(\alpha_2 - 1) \sum_{j=1}^n \alpha_j \right. \\ & \left. + 2 \sum_{j=1}^n \alpha_j^2 + 4 \left(\sum_{j=1}^n \alpha_j \right)^2 - 2 \sum_{j=1}^{n-2} \alpha_j \alpha_{j+2} \right]. \quad (31) \end{aligned}$$

Minimizing with respect to α_k , for $1 \leq k \leq n$, we obtain another set of n linear equations with the following form:

⁵ Provided the data resolution is fine enough.

$$\begin{pmatrix}
 5 & 6 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & \dots & 4 & 4 & 4 & 4 & 4 \\
 6 & 10 & 6 & 5 & 6 & 6 & 6 & 6 & 6 & 6 & \dots & 6 & 6 & 6 & 6 & 6 \\
 3 & 6 & 6 & 4 & 3 & 4 & 4 & 4 & 4 & 4 & \dots & 4 & 4 & 4 & 4 & 4 \\
 4 & 5 & 4 & 6 & 4 & 3 & 4 & 4 & 4 & 4 & \dots & 4 & 4 & 4 & 4 & 4 \\
 4 & 6 & 3 & 4 & 6 & 4 & 3 & 4 & 4 & 4 & \dots & 4 & 4 & 4 & 4 & 4 \\
 4 & 6 & 4 & 3 & 4 & 6 & 4 & 3 & 4 & 4 & \dots & 4 & 4 & 4 & 4 & 4 \\
 4 & 6 & 4 & 4 & 3 & 4 & 6 & 4 & 3 & 4 & \dots & 4 & 4 & 4 & 4 & 4 \\
 4 & 6 & 4 & 4 & 4 & 3 & 4 & 6 & 4 & 3 & \dots & 4 & 4 & 4 & 4 & 4 \\
 & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\
 4 & 6 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & \dots & 3 & 4 & 6 & 4 & 3 \\
 4 & 6 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & \dots & 4 & 3 & 4 & 6 & 4 \\
 4 & 6 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & \dots & 4 & 4 & 3 & 4 & 6
 \end{pmatrix}
 \begin{pmatrix}
 \alpha_1 \\
 \alpha_2 \\
 \alpha_3 \\
 \alpha_4 \\
 \alpha_5 \\
 \alpha_6 \\
 \alpha_7 \\
 \alpha_8 \\
 \vdots \\
 \alpha_{n-2} \\
 \alpha_{n-1} \\
 \alpha_n
 \end{pmatrix}
 =
 \begin{pmatrix}
 2 \\
 3 \\
 2 \\
 2 \\
 2 \\
 2 \\
 2 \\
 2 \\
 \vdots \\
 2 \\
 2 \\
 2
 \end{pmatrix}
 \tag{32}$$

The solutions are listed in Table 12 for $1 \leq n \leq 6$.

n	α_0	$\alpha_{\pm 1}$	$\alpha_{\pm 2}$	$\alpha_{\pm 3}$	$\alpha_{\pm 4}$	$\alpha_{\pm 5}$	$\alpha_{\pm 6}$	$1 - \frac{m_{rms}}{m_{rms}}$ optimum	$1 - \frac{m_{rms}}{m_{rms}}$ $\alpha_j = \frac{1}{w}$
1	$\frac{1}{5}$	$\frac{2}{5}$						0.553	0.529
2	$\frac{2}{7}$	$\frac{1}{7}$	$\frac{3}{14}$					0.733	0.717
3	$\frac{2}{15}$	$\frac{1}{5}$	$\frac{1}{10}$	$\frac{2}{15}$				0.817	0.798
4	$\frac{9}{55}$	$\frac{6}{55}$	$\frac{8}{55}$	$\frac{4}{10}$	$\frac{1}{5}$			0.865	0.843
5	$\frac{55}{91}$	$\frac{12}{91}$	$\frac{8}{91}$	$\frac{55}{10}$	$\frac{11}{5}$	$\frac{6}{91}$		0.895	0.871
6	$\frac{4}{35}$	$\frac{3}{35}$	$\frac{3}{28}$	$\frac{1}{14}$	$\frac{91}{35}$	$\frac{91}{70}$	$\frac{1}{20}$	0.915	0.891

Table 12: Best Parameters to Minimize $\left(\frac{y'_{i+1} - y'_{i-1}}{2}\right)^2$ and Fraction of Noise Removed

Note that the distribution of coefficients from α_{-n} to α_n is no longer unimodal. Furthermore, $\alpha_0 < \alpha_1$ for odd values of n . Finally we note that for window size $w = 5$, the triangular filter using $\alpha_j = (3 - |j|)/9$ results in a noise reduction of 66.7%, notably less than the equal-weights method.

5.3.3 Best Parameters to Minimize Deviation Angles

In this section, we examine the smoothing problem based on minimizing the deviation angles ϕ'_i . Here the problem is more complex and we will *not* obtain general expressions of the optimum smoothing parameters which are independent of the probability p involved in our model. We restrict our study to the cases $w = 3$ and $w = 5$.

Our definitions of section 5.1 imply that $\phi'_i = \theta'_{i+1} - \theta'_i$. From trigonometry, we have

$$\tan \phi'_i = \frac{\tan \theta'_{i+1} - \tan \theta'_i}{1 + \tan \theta'_i \tan \theta'_{i+1}}. \quad (33)$$

Now $\tan \theta'_i = y'_i - y'_{i-1}$ and $\tan \theta'_{i+1} = y'_{i+1} - y'_i$. Thus we can obtain $\tan \phi'_i$ from the smoothed coordinates and then ϕ'_i from the value of the tangent.

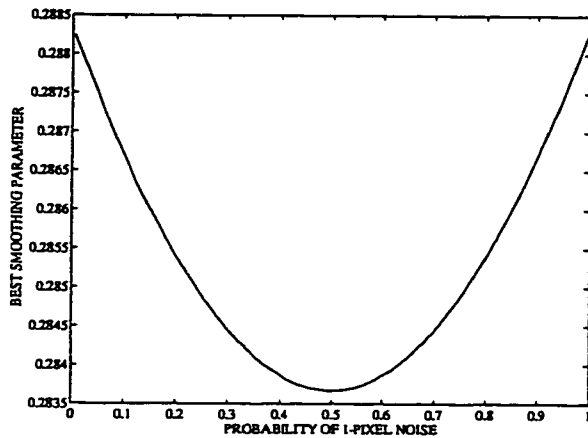
For $w = 3$

For $w = 3$, ϕ'_i at \mathbf{p}'_i will depend on the *original* contour points in a 5-point neighbourhood around \mathbf{p}_i . For our model of Equation 20, there are $2^5 = 32$ possible configurations for such a neighbourhood, which must be examined for their corresponding ϕ'_i . Of course, these computations need not be performed manually; they can be carried out using a language for symbolic mathematical calculation.

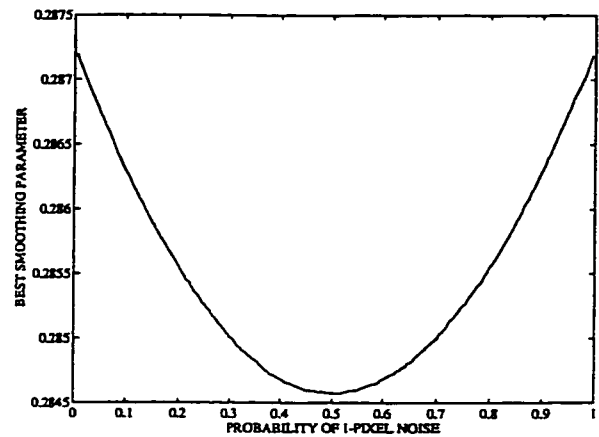
Adding together the contributions from the 32 possible configurations, weighted by the respective probabilities of these configurations, results in the following expression for $\overline{\tan^2 \phi'_i}$:

$$\begin{aligned} \overline{\tan^2 \phi'_i} = & 2p(1-p)(1-3p+3p^2) \left(\alpha^2 + \frac{(1-4\alpha)^2}{(1+\alpha-3\alpha^2)^2} + \frac{2(1-3\alpha)^2}{9\alpha^2(2-3\alpha)^2} \right) \\ & + 2p^2(1-p)^2 \left((1-3\alpha)^2 + (1-2\alpha)^2 + \frac{(1-3\alpha)^2}{(1+\alpha-2\alpha^2)^2} \right. \\ & \left. + \frac{(2-7\alpha)^2}{\alpha^2(7-12\alpha)^2} + \frac{2\alpha^2}{(1-\alpha^2)^2} + \frac{(1-4\alpha)^2}{32\alpha^2(1-2\alpha)^2} \right) \quad (34) \end{aligned}$$

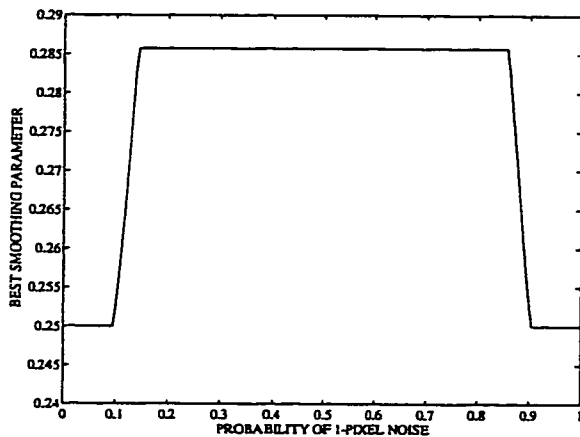
For simplicity we have dropped the subscript on the unique parameter α_1 . Numerical optimization was performed to find the value of α which minimizes Equation 34. No single value of α will minimize $\overline{\tan^2 \phi'_i}$ for all values of p . The results are shown



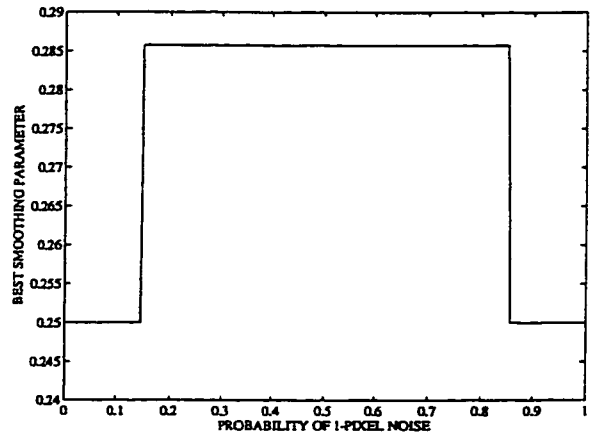
(a) Minimizing mean squared tangent



(b) Minimizing mean squared angle



(c) Minimizing mean absolute tangent



(d) Minimizing mean absolute angle

Figure 24: Best α to Minimize Deviation Angles for $w = 3$.

in Figure 24(a). The best value of α is now a smooth function of p . However we note that the domain of variation is very little.

We cannot compare the results obtained minimizing the mean squared tangent of ϕ'_i to the situation without smoothing, since $\overline{\tan^2 \phi_i}$ is infinite. By taking the *arc tangent* function of Equation 33, we can obtain the values of the angles ϕ'_i themselves and we can derive an expression for $\overline{\phi_i'^2}$ in the same manner. Numerical optimization

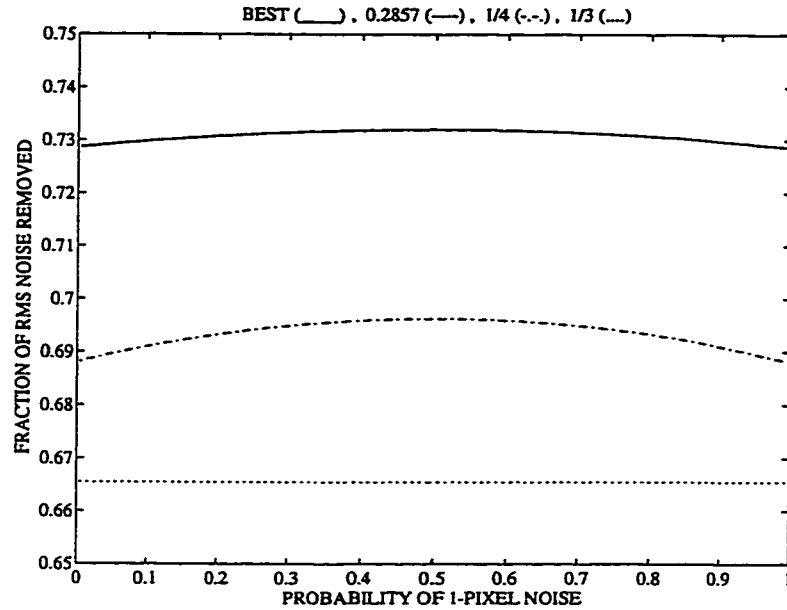
of this expression yields the results shown in Figure 24(b). As can be seen, they are almost the same as those of Figure 24(a). In a similar fashion, we can generate expressions for $\overline{|\tan \phi'_i|}$ and $\overline{|\phi'_i|}$, for which the best smoothing parameters are shown in Figures 24(c) and 24(d) respectively. Here there seems to be one predominant best parameter over a wide range of values for p .

All the results shown in Figure 24 were obtained numerically, for values of p ranging from 0.005 to 0.995, in steps of 0.005. As expected, all these curves are symmetric about $p = 0.5$, so we will limit our discussion to $p < 0.5$. In Figure 24(c), the best value of α for $p \in (0.005, 0.095)$ is $\alpha = 0.25$; then, for $p \in (0.145, 0.495)$, the best value is $\alpha = 0.2857$. Between these two intervals, p increases almost linearly. In Figure 24(d), the same values of α are found: $\alpha = 0.25$ is the best choice for $p \in (0.005, 0.145)$ and $\alpha = 0.2857$ is the best choice for $p \in (0.150, 0.495)$.

In Figures 24(a) and 24(b), the best value of α is a smoothly varying function of p . But we notice that $\alpha = 0.2857$ is an intermediate value of α in the narrow range of best values. In fact, choosing $\alpha = 0.2857$ for *all* values of p , the value of $\overline{\tan^2 \phi'_i}$ is always within 0.2% of the minimum possible.

Despite the differences in the actual curves of Figure 24, the corresponding ranges of best α 's are always quite narrow and very similar, independently of the exact criterion chosen. From now on, to maintain uniformity with the treatment of sections 5.3.1 and 5.3.2, we will restrict ourselves to minimizing the mean squared angle.

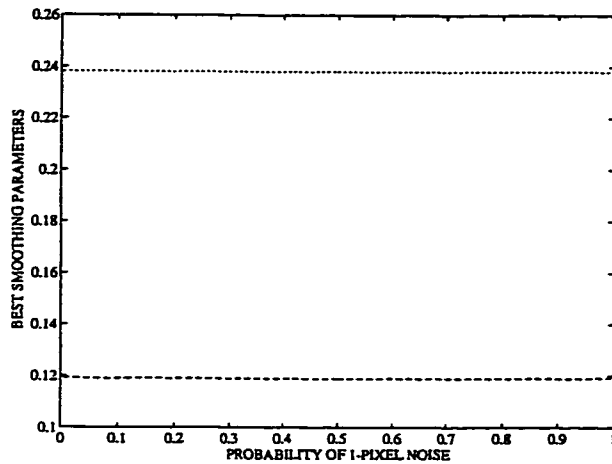
Equation 24 provided a measure of the noise before smoothing: $\phi_{rms} = \frac{\pi}{2} \sqrt{\frac{3p(1-p)}{2}}$. The fraction of this noise $(1 - \phi'_{rms}/\phi_{rms})$ which is removed by a simple smoothing operation with $w = 3$ was computed for different values of α . The results are displayed in Figure 25. The solid line represents the best case and we see that approximately 73% of the r.m.s. noise is removed. The dashed line, representing the case where $\alpha = 0.2857$ is used for *all* values of p , is *not* distinguishable from the best case at this scale. The dash-dotted and the dotted lines represent the fraction of noise removed for $\alpha = 0.25$ and $\alpha = \frac{1}{3}$ respectively. In this last case, this fraction is a constant equal to 0.6655.

Figure 25: Fraction of ϕ_{rms} Removed for $w = 3$.

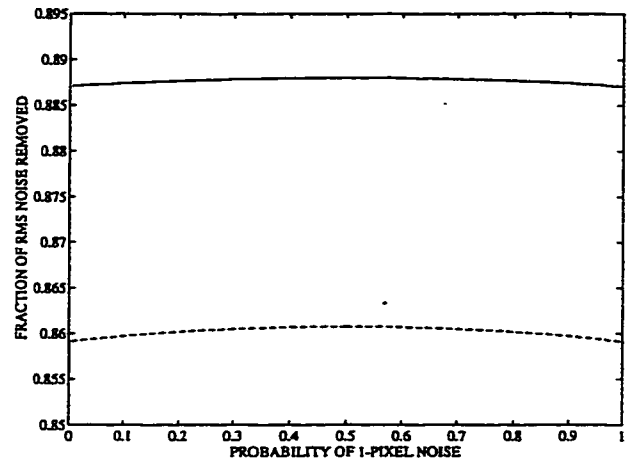
For $w = 5$

For $w = 5$, $2^7 = 128$ possible configurations of a 7-point neighbourhood centered on p'_i must be considered to obtain the values of $\phi'_i{}^2$ after smoothing. In Equation 34, there were 9 distinct terms involving p and α . Now the computation of $\overline{\tan^2 \phi'_i}$ results in 35 distinct terms in p , α_1 , and α_2 . We will not reproduce this lengthy expression here...

Figure 26(a) presents the best choice of parameters to minimize $\overline{\phi'_i{}^2}$. We notice that there is very little variation in their values over the range of values of p . The optimum parameters are approximately $\alpha_{\pm 1} = 0.2381$ and $\alpha_{\pm 2} = 0.1189$. These values are close to $\frac{2}{9}$ and $\frac{1}{9}$, the values for the triangular 5-point filter. Figure 26(b) shows the fraction of ϕ_{rms} removed by the smoothing operation. The solid line represents the case where the optimum parameters are used for each value of p . In this situation, approximately 88.75% of the noise is removed. The dashed line represents the case $\alpha_{\pm 1} = \frac{2}{9}$ and $\alpha_{\pm 2} = \frac{1}{9}$, for which 86% of the noise is removed approximately. We see that these results are close to the ideal situation.



(a) Optimum parameters
 $\alpha_{\pm 1}$ (dotted); $\alpha_{\pm 2}$ (dashed)



(b) Fraction of ϕ_{rms} removed
 Best (solid); $\alpha_{\pm 1} = \frac{2}{9}$ & $\alpha_{\pm 2} = \frac{1}{9}$ (dashed)

Figure 26: Minimizing $\overline{\phi_i^2}$ for $w = 5$.

5.4 Verifying Results for Varying Curvature

In the preceding section, we have studied optimum local weighted averaging extensively, based on a model of a horizontal border with random 1-pixel noise. Particular solutions were derived based on error criteria chosen in light of specific computational tasks to be performed after the smoothing operation. But can these results be relied upon to handle digitization noise along *arbitrary* contours?

Our results were obtained for a *straight, horizontal* border, i.e. a line of curvature 0. But for arbitrary contours, curvature may vary from point to point. Should optimum smoothing parameters vary with curvature and, if so, in what manner? For a given window size, can a fixed set of smoothing parameters be found which will give optimum (or near optimum) results across a wide range of curvature values? If so, how does this set of parameters compare with the one we have derived using our simple model?

In this section, we try to answer the above questions by performing some experiments with digital circles. It should be clear that our interest is not with digital circles per se but rather, as explained above, with the variation of optimum smoothing parameters with curvature. The approach will be to examine, for digital circles of various radii, situations which are equivalent to the ones studied for the horizontal straight border in sections 5.3.1, 5.3.2, and 5.3.3. Using numerical optimization, we will find the best choice of smoothing parameters for each situation, over a wide range of curvature values, and compare them with the values obtained previously. An example of a digital circle is shown in Figure 27 for a radius $R = 7$.

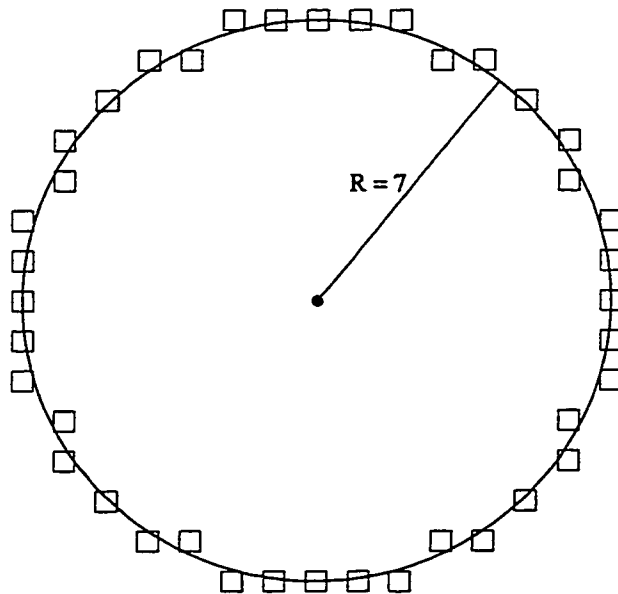


Figure 27: Digital Circle of Radius $R = 7$.

5.4.1 Minimizing Error on Distances to Center

In this section, we consider the distances d_i from the center to each pixel P_i as approximations to the radius R . See Figure 28(a). After smoothing, pixel P_i is replaced by pixel P'_i which is at a distance d'_i from the center of the circle. Our aim is to find the values of the smoothing parameters, for $w = 3$ and $w = 5$, which will minimize $\overline{(R - d_i)^2}$. Of course, these parameters might vary depending on the radius

of the circles.

For reasons of symmetry, it is only necessary to consider one quadrant; with special attention to the main diagonal, we can restrict our attention to the first octant of each circle. Let $N_{1/8}$ be the number of pixels which are *strictly* within the first octant. The mean value of $(R - d'_i)^2$ is obtained by adding twice the sum of $(R - d'_i)^2$ for these $N_{1/8}$ points, plus the value for the pixel at coordinates $(R, 0)$, plus the value for the pixel on the main diagonal (if present). This sum is then divided by $2N_{1/8} + 1$ (or $2N_{1/8} + 2$, if there is a pixel on the main diagonal).

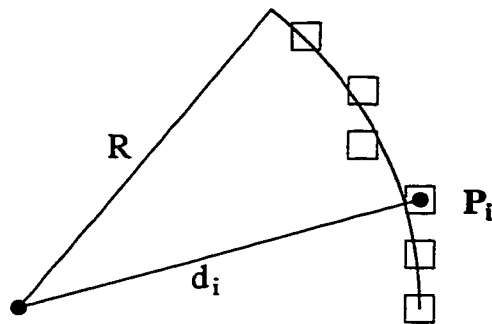
We now give a simple example, for $R = 4$ and $w = 3$. First we consider the situation before any smoothing is applied. For the point on the x -axis, the value of $(R - d_0)^2$ is always 0. For the next point $(4, 1)$, $(R - d_1)^2 = (\sqrt{17} - 4)^2$. For the next point $(3, 2)$, $(R - d_2)^2 = (\sqrt{13} - 4)^2$. Finally, for the point on the main diagonal $(3, 3)$, $(R - d_3)^2 = (\sqrt{18} - 4)^2$. The contributions for points $(4, 1)$ and $(3, 2)$ are counted twice and added to the contributions for the diagonal point $(3, 3)$. This sum is then divided by 6. Taking the square root of the result, we obtain $(R - d_i)_{rms} = 0.25832$.

After smoothing with the smallest window size, $w = 3$, we have the following values for $(R - d'_i)^2$:

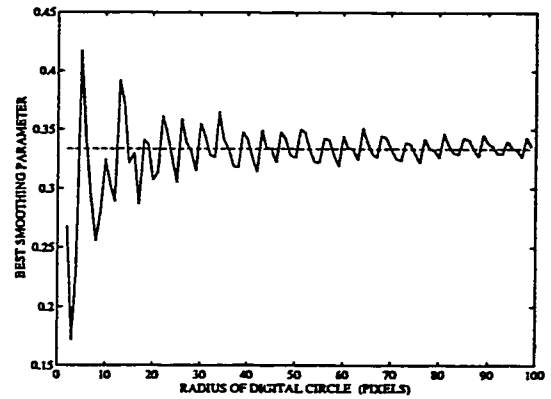
$$\begin{aligned}
 (R - d'_0)^2 &= 0 \\
 (R - d'_1)^2 &= (\sqrt{(4 - \alpha)^2 + 1} - 4)^2 \\
 (R - d'_2)^2 &= (\sqrt{(3 + \alpha)^2 + 4} - 4)^2 \\
 (R - d'_3)^2 &= (\sqrt{2(3 - \alpha)^2} - 4)^2.
 \end{aligned} \tag{35}$$

and we must minimize the expression $\frac{1}{6} (2(R - d'_1)^2 + 2(R - d'_2)^2 + (R - d'_3)^2)$. Thus the best smoothing parameter for $R = 4$ and $w = 3$ is found to be $\alpha = 0.23610$.

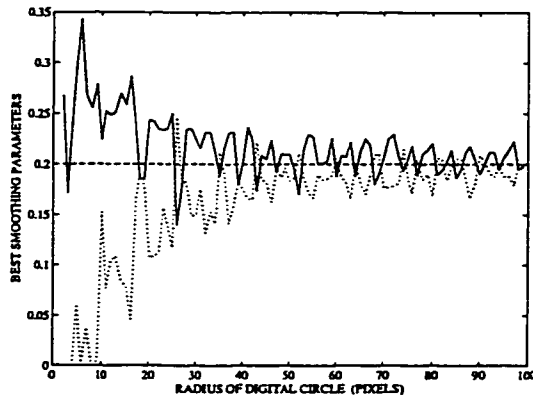
In the numerical computations it is possible to take advantage of the fact that, for small window sizes, the smoothing rarely affects the y -coordinates in the first octant; exceptions occur occasionally for the last pixel in the first octant (not on the diagonal) and for the pixel on the diagonal when the preceding pixel has the same x -coordinate. This last condition is found only for radii values of 1, 4, 11, 134, 373, 4552 etc... (see Kulpa [79]).



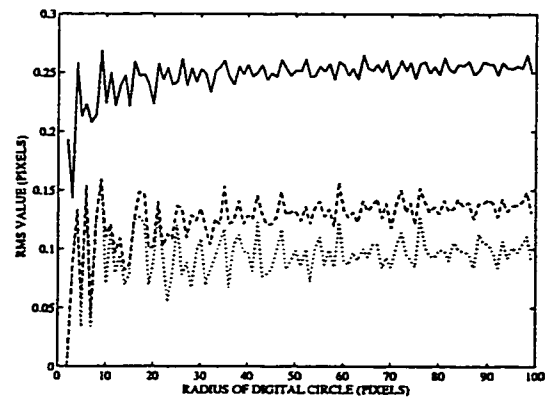
(a) Distance d_i from center to P_i



(b) Best $\alpha_{\pm 1}$ for $w = 3$



(c) Best $\alpha_{\pm 1}$ (solid), $\alpha_{\pm 2}$ (dotted) for $w = 5$



(d) RMS error: $w = 3$ (dashed); $w = 5$ (dotted)

Figure 28: Best Smoothing to Minimize $\overline{(R - d_i')^2}$.

The coordinates of the pixels for the first octant of the digital circles were generated using the simple procedure presented in Horn [63], with a small correction pointed out by Kulpa [78] (see also Doros [40]). The best smoothing parameters were obtained for integer radii values ranging from 2 pixels to 99 pixels, in steps of 1. The results are presented in Figure 28(b) for $w = 3$ and 28(c) for $w = 5$. For comparison, the values derived in section 5.3.1 from our model of a noisy horizontal edge are shown with dashed lines.

For $w = 3$, we see that for radii values larger than 20 pixels the best $\alpha_{\pm 1}$ oscillates around $\frac{1}{3}$, as derived from our model. Similarly, for $w = 5$, the best values of $\alpha_{\pm 1}$

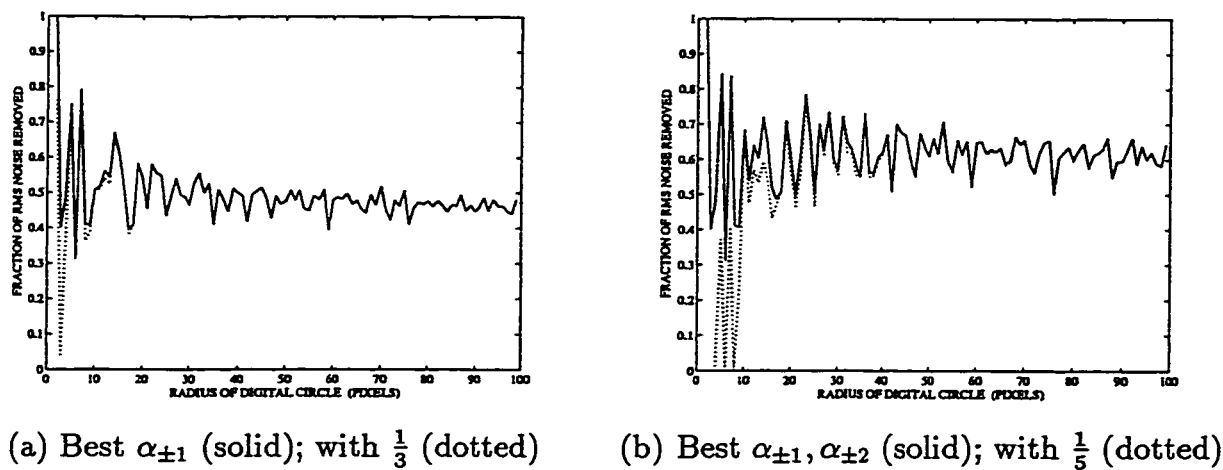


Figure 29: Fraction of RMS Noise Removed: (a) $w = 3$; (b) $w = 5$.

and $\alpha_{\pm 2}$ are close to the predicted value of 0.2. For small radii values however, the optimum $\alpha_{\pm 2}$ -values are much lower than this value and the optimum $\alpha_{\pm 1}$ -values are correspondingly higher. This is easily understood since a 5-pixel neighbourhood covers a relatively large portion of the circumference in these cases (as much as one eighth of the total circumference for a radius of 6 pixels, one fourth for a radius of 3 pixels). In fact, for radii values of 2, 3, 4, 6, and 8 pixels, it is best to use $\alpha_{\pm 2} = 0.0$ and smoothen using only the nearest neighbour.

Figure 28(d) compares the r.m.s. values of the errors on the radii without smoothing (solid line) to the best values possible after smoothing with window sizes of $w = 3$ (dashed line) and $w = 5$ (dotted line) respectively.

For each value of the radius, we have also compared the noise reduction achieved using the optimum parameters to that achieved with the constant values $\frac{1}{3}$ and $\frac{1}{5}$. The results are presented in Figure 29(a) and 29(b), for $w = 3$ and $w = 5$ respectively. For $w = 3$, the 2 curves are indistinguishable for $R > 18$ pixels, and they are very close for $R \geq 10$. For $w = 5$ and radii values 2, 3, 4, 6, and 8 pixels, smoothing with $\alpha_{\pm 1} = \alpha_{\pm 2} = 0.2$ is actually *worse* than no smoothing at all. But, for $R > 18$, the best curve and that obtained with these fixed values are very close.

Finally, for $R \geq 10$ and window sizes $w = 3$ and $w = 5$, Table 13 compares the mean noise reduction of 3 methods: the optimum method, corresponding to the

variable parameters of Figure 28(b) and Figure 28(c); the fixed parameter method derived from our model of section 5.3; and the best fixed parameter method obtained from numerical estimates. We see that the results are very close and that the method derived from our simple noise model compares very well with the numerically determined best fixed parameter method.

Window	Method	$\alpha_{\pm 1}$	$\alpha_{\pm 2}$	Mean noise reduction
w = 3	optimum	variable		0.484317
	model	$\frac{1}{3}$		0.482672
	best fixed	0.3329		0.482673
w = 5	optimum	variable	variable	0.6184
	model	$\frac{1}{5}$	$\frac{1}{5}$	0.6068
	best fixed	0.2148	0.1703	0.6117

Table 13: Mean Noise Reduction for $10 \leq R \leq 99$

5.4.2 Minimizing Error on Tangent Directions

In this section, we compare the direction of the tangent to a circle at a given point to the numerical estimate of that direction, obtained for digital circles. The situation is illustrated in Figure 30(a).

Since the *slope* of the tangent is infinite at pixel $(R, 0)$ of the digital circle, we will consider instead the *angle* which the tangent line makes with the x -axis. The radius from the center of the circle to pixel P_i makes an angle θ_i with the positive x -axis. Now consider the point where this radius intersects the continuous circle. Theoretically, the angle between the tangent to the circle at that point and the x -axis is $\frac{\pi}{2} + \theta_i$. On the other hand, the numerical estimate of this angle is given by $\frac{\pi}{2} + \varphi_i$, where φ_i is the angle between the horizontal axis and the perpendicular bisector of the segment from P_{i-1} to P_{i+1} (see Figure 30(a)). The difference between these angles, $(\varphi_i - \theta_i)$ is the error on the elevation of the tangent to the circle at the point of interest. Our goal is to minimize the r.m.s. value of $(\varphi'_i - \theta'_i)$, where the primes refer to the quantities after smoothing.

The values of φ_i and θ_i are readily computed in terms of the original coordinates of the digital circle as follows:

$$\theta_i = \tan^{-1} \left(\frac{y_i}{x_i} \right); \quad \text{and} \quad \varphi_i = \tan^{-1} \left(\frac{x_{i-1} - x_{i+1}}{y_{i+1} - y_{i-1}} \right). \quad (36)$$

The values of φ'_i and θ'_i are obtained similarly, in terms of the coordinates *after smoothing*.

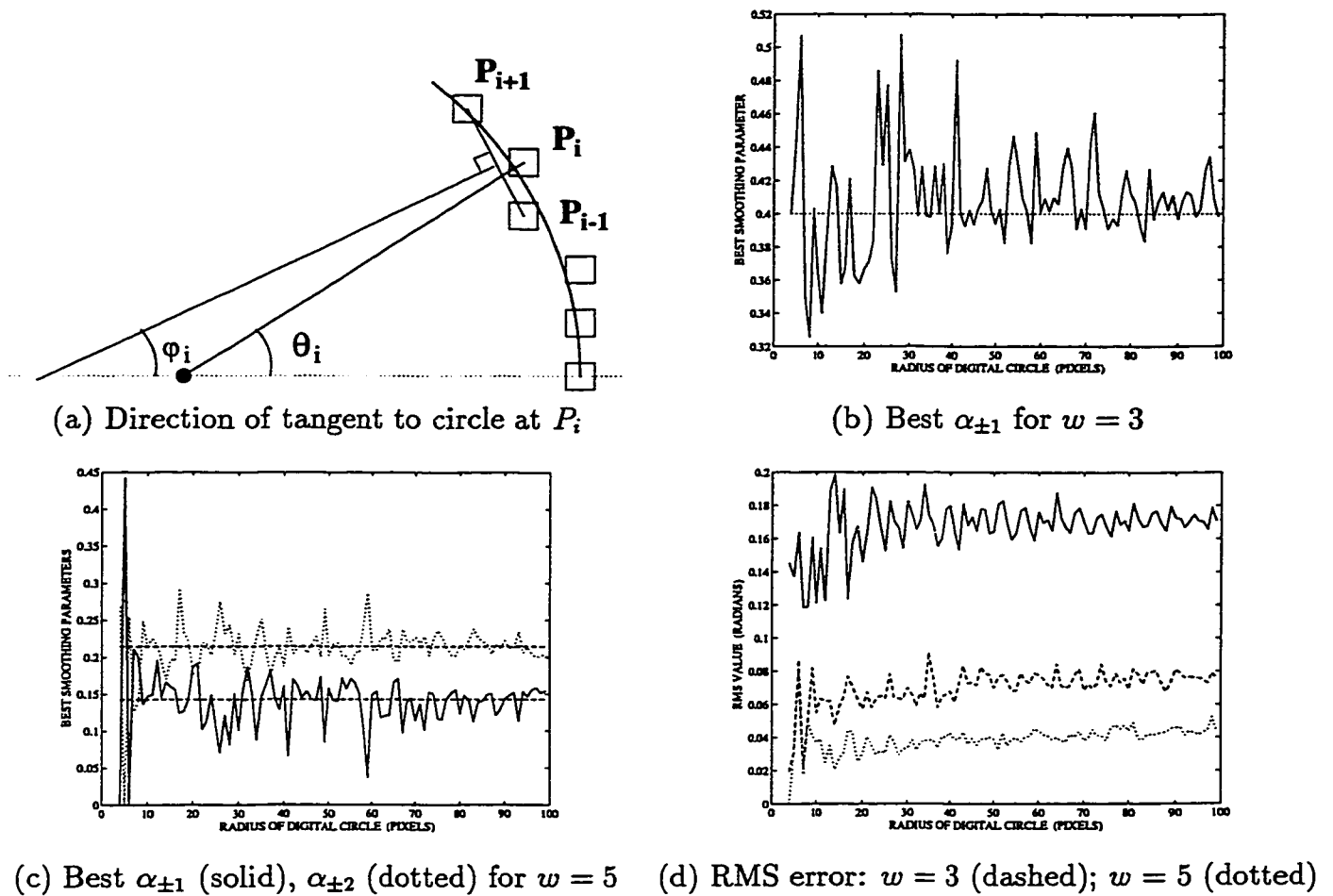


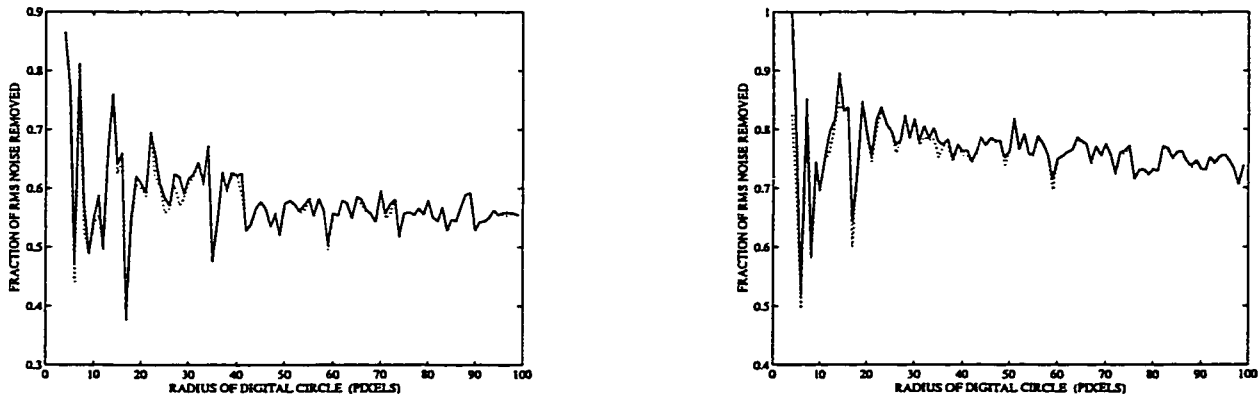
Figure 30: Best Smoothing to Minimize $\overline{(\varphi'_i - \theta'_i)^2}$.

Once again, the r.m.s. error for the entire circle can be computed by considering only the first octant; and the best smoothing parameters were obtained for integer

radii values ranging from 4 pixels to 99 pixels, in steps of 1. The results are presented respectively in Figures 30(b) for $w = 3$ and 30(c) for $w = 5$.

In section 5.3.2, for $w = 3$, the optimum value derived for $\alpha_{\pm 1}$ was $\frac{2}{5}$; for $w = 5$, the optimum values derived for $\alpha_{\pm 1}$ and $\alpha_{\pm 2}$ were $\frac{1}{7}$ and $\frac{3}{14}$ respectively. These values are shown with horizontal dashed lines in Figures 30(b) and 30(c). The best smoothing parameters vary with the values of R . However, when we compute their means for $4 \leq R \leq 99$, the results obtained are very close to the predicted values. Thus, for $w = 3$, $\overline{\alpha_{\pm 1}} = 0.408$ (compared to 0.4); for $w = 5$, $\overline{\alpha_{\pm 1}} = 0.1415$ (compared to 0.1429) and $\overline{\alpha_{\pm 2}} = 0.2129$ (compared to 0.2143).

Figure 30(d) compares the r.m.s. values of the errors on the elevation of the tangents to a circle without smoothing (solid line) to the best values possible after smoothing with window sizes $w = 3$ (dashed line) and $w = 5$ (dotted line).



(a) Using best $\alpha_{\pm 1}$ (solid); using $\frac{2}{5}$ (dotted) (b) Using best $\alpha_{\pm 1}, \alpha_{\pm 2}$ (solid); using $\frac{1}{7}, \frac{3}{14}$ (dotted)

Figure 31: Fraction of RMS Noise Removed: (a) $w = 3$; (b) $w = 5$.

The noise reduction produced by smoothing is equal to $1.0 - \frac{(\varphi'_i - \theta'_i)_{rms}}{(\varphi_i - \theta_i)_{rms}}$. For each value of the radius, we have compared the noise reduction achieved using the optimum parameters to that achieved with the constant values $\alpha_{\pm 1} = \frac{2}{5}$, for $w = 3$, and $\alpha_{\pm 1} = \frac{1}{7}$, $\alpha_{\pm 2} = \frac{3}{14}$, for $w = 5$. The results are presented in Figures 31(a) and 31(b) respectively.

As can be seen, the constant values predicted by our simple model yield noise

reduction results which are very close to optimum.

Finally, for $4 \leq R \leq 99$ and window sizes $w = 3$ and $w = 5$, Table 14 compares the mean noise reduction of the 3 methods as explained previously. The best smoothing parameters derived from our simple noise model and the numerically determined best fixed parameters are almost the same and their performance is nearly optimal.

Window	Method	$\alpha_{\pm 1}$	$\alpha_{\pm 2}$	Mean noise reduction
w = 3	optimum	variable		0.58084
	model	$\frac{2}{5}$		0.57474
	best fixed	0.4026		0.57478
w = 5	optimum	variable	variable	0.7658
	model	$\frac{1}{7}$	$\frac{3}{14}$	0.7572
	best fixed	0.1445	0.2088	0.7576

Table 14: Mean Noise Reduction for $4 \leq R \leq 99$

5.4.3 Minimizing Error on Deviation Angles

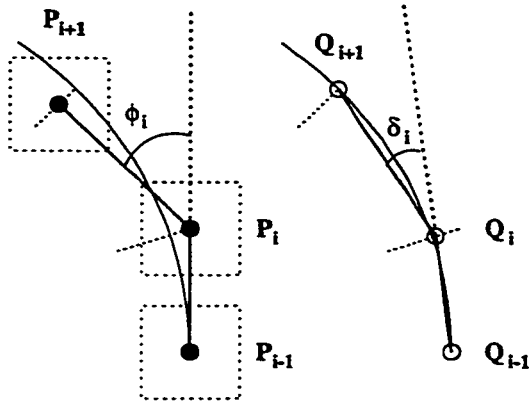
In this section, we compare the deviation angles along the circumference of a circle to the numerical estimates obtained for digital circles. The situation is illustrated in Figure 32(a).

Consider 3 consecutive pixels P_{i-1} , P_i and P_{i+1} on the circumference of a digital circle. The deviation angle at P_i is denoted by ϕ_i . Now the line segments from the center of the circle to these 3 pixels (partly represented by dashed lines in the figure) have elevation angles of θ_{i-1} , θ_i , and θ_{i+1} respectively. The intersections of these line segments with the circle are the true circle points Q_{i-1} , Q_i , and Q_{i+1} with these elevations. Connecting these points by line segments defines a deviation angle δ_i at Q_i , for which ϕ_i is a numerical estimate.

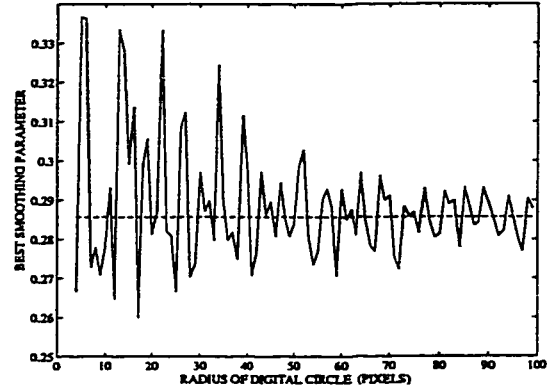
The difference between δ_i and ϕ_i is the error on the deviation angle at the point of interest. Our goal in this section is to find the optimum parameters which will minimize the r.m.s. value of this error, after smoothing.

In terms of the pixel coordinates (x_i, y_i) , the deviation angle ϕ_i is equal to

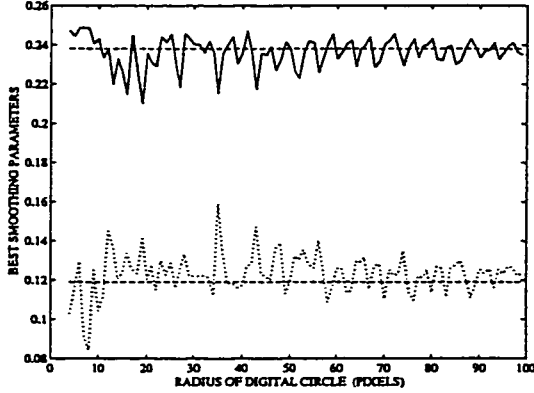
$$\phi_i = \tan^{-1} \left(\frac{(y_{i+1} - y_i)(x_i - x_{i-1}) - (x_{i+1} - x_i)(y_i - y_{i-1})}{(x_{i+1} - x_i)(x_i - x_{i-1}) + (y_{i+1} - y_i)(y_i - y_{i-1})} \right). \quad (37)$$



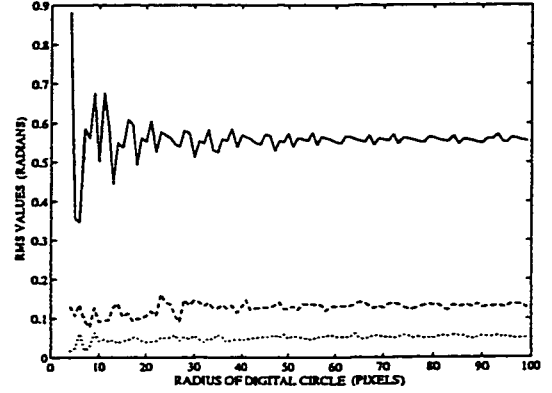
(a) Deviation angles ϕ_i and δ_i



(b) Best $\alpha_{\pm 1}$ for $w = 3$



(c) Best $\alpha_{\pm 1}$ (solid), $\alpha_{\pm 2}$ (dotted) for $w = 5$



(d) RMS error: $w = 3$ (dashed); $w = 5$ (dotted)

Figure 32: Best Smoothing to Minimize $\overline{(\delta'_i - \phi'_i)^2}$.

To compute δ_i , we first obtain the elevation angles as $\theta_i = \tan^{-1}(y_i/x_i)$ and then the coordinates of the circle points Q_i as

$$(\tilde{x}_i, \tilde{y}_i) = (R \cos \theta_i, R \sin \theta_i). \quad (38)$$

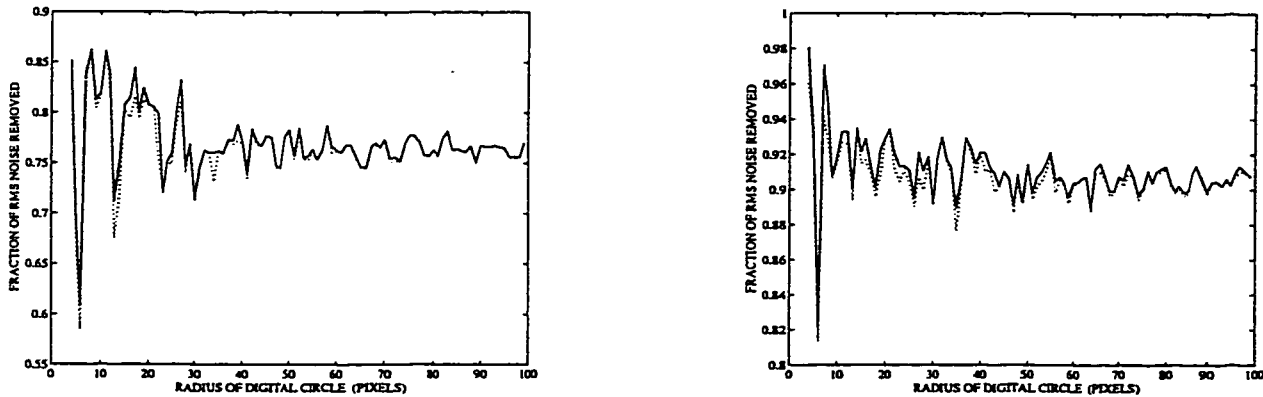
Finally, δ_i is computed as in Equation 37, using (\tilde{x}, \tilde{y}) instead of (x, y) .

The best smoothing parameters were obtained for integer radii ranging from 4 pixels to 99 pixels, in steps of 1. The results are presented in Figures 32(b) for $w = 3$ and 32(c) for $w = 5$.

In section 5.3.3, for $w = 3$, the optimum value derived for $\alpha_{\pm 1}$ was 0.2857; for $w = 5$, the optimum values derived for $\alpha_{\pm 1}$ and $\alpha_{\pm 2}$ were 0.2381 and 0.1189 respectively. These values are shown with dashed lines in Figures 32(b) and 32(c). Again, the best smoothing parameters vary with the values of R . However, when we compute their means for $4 \leq R \leq 99$, the results obtained are very close to the predicted values. Thus, for $w = 3$, $\overline{\alpha_{\pm 1}} = 0.2886$; for $w = 5$, $\overline{\alpha_{\pm 1}} = 0.2363$ and $\overline{\alpha_{\pm 2}} = 0.1232$.

Figure 32(d) compares the r.m.s. values of the errors on the deviation angles without smoothing (solid line) to the best values possible after smoothing with window sizes $w = 3$ (dashed line) and $w = 5$ (dotted line).

The noise reduction produced by smoothing is equal to $1.0 - \frac{(\delta'_i - \phi'_i)_{r.m.s.}}{(\delta_i - \phi_i)_{r.m.s.}}$. For each value of the radius, we have compared the noise reduction achieved using the optimum parameters to that achieved with the constant values $\alpha_{\pm 1} = 0.2857$, for $w = 3$, and $\alpha_{\pm 1} = 0.2381$, $\alpha_{\pm 2} = 0.1189$, for $w = 5$. The results are presented in Figures 33(a) and 33(b) respectively.



(a) Best $\alpha_{\pm 1}$ (solid); with 0.2857 (dotted) (b) Best $\alpha_{\pm 1}, \alpha_{\pm 2}$ (solid); with 0.2381, 0.1189 (dotted)

Figure 33: Fraction of RMS Noise Removed: (a) $w = 3$; (b) $w = 5$.

As can be seen, the constant values predicted by our simple model yield noise

reduction results which are very close to optimum.

For $4 \leq R \leq 99$ and window sizes $w = 3$ and $w = 5$, Table 15 compares the mean noise reduction of 3 methods as explained previously. The best smoothing parameters derived from our simple noise model and the numerically determined best fixed parameters are even closer than the 2 previous cases and their performance is very nearly optimal.

Window	Method	$\alpha_{\pm 1}$	$\alpha_{\pm 2}$	Mean noise reduction
w = 3	optimum	variable		0.770903
	model	0.2857		0.765716
	best fixed	0.2865		0.765732
w = 5	optimum	variable	variable	0.910833
	model	0.2381	0.1189	0.906895
	best fixed	0.2386	0.1190	0.906917

Table 15: Mean Noise Reduction for $4 \leq R \leq 99$

Finally, for $w = 5$, if one prefers to use $\alpha_{\pm 1} = \frac{2}{9}$ and $\alpha_{\pm 2} = \frac{1}{9}$ (computationally convenient for deviation angle measurements), the mean error reduction level is 0.8832. This is very good but not quite as effective as the optimum methods previously discussed. A comparison of the error reduction with the best possible solution, for every value of R , appears in Figure 34.

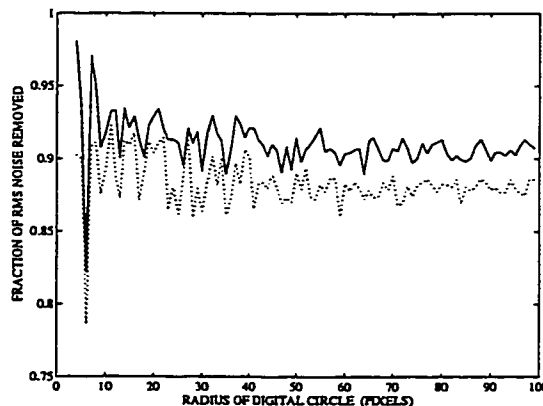


Figure 34: RMS Noise Removed for $w = 5$: best case (solid); $\frac{2}{9}, \frac{1}{9}$ (dotted)

5.5 Conclusion

This chapter has considered the problem of optimum smoothing of 2-D binary contours by local weighted averaging methods. These simple and effective methods are still widely used. Furthermore, there are many applications where smoothing is performed to improve the precision of *specific measurements* to be computed from the contour points. In such cases, the smoothing parameters should be chosen based on the nature of the computations intended, instead of relying on a single, general ‘optimality’ criterion. Thus our work was focused on optimum local weighted averaging methods tailored for specific computational goals. We have considered three such goals: obtaining reliable estimates of point positions, of slopes, and of deviation angles along the contours.

To study the problem, a simple model was defined to represent 1-pixel random noise along a straight horizontal border. Based on this simple model, an in-depth analytical investigation of the problem was carried out, from which precise answers were derived for the 3 chosen criteria.

Despite its simplicity, this model captures well the kind of perturbations which digitization noise causes in the numerical estimation of various quantities along 2-D binary contours *even with arbitrary curvature*. This was indeed verified, for window sizes of $w = 3$ and $w = 5$, by finding the best smoothing parameters, using equivalent criteria, for digital circles over a wide range of radii.

In this general case, the best smoothing parameters were found to vary according to the length of the radius. Thus, in order to take full advantage of these optimum filters, it would be necessary to compute local estimates of the radius of curvature for groups of consecutive pixels along the contour, and then apply the best parameters found for these radii. This would significantly reduce the efficiency of the smoothing operation. However, it is not really necessary to go to that extent since the performance of these varying-weight optimum filters can be very nearly approached by methods with a *fixed* set of parameters. The latter were derived by numerical computation, for a wide range of radii. And it turned out that their values were very close to those predicted using our simple digitization noise model.

These numerical computations with varying radius of curvature validate our proposed model and confer added confidence to the results obtained from it. Researchers requiring simple and effective local weighted averaging filters before making numerical estimates of specific quantities can thus rely on this model to derive optimum methods tailored to their particular needs.

Chapter 6

Curvature Feature Extraction

Our E4 numeral recognition system, with its 93.90% recognition rate and 1.60% substitution rate, performed relatively well (see Table 1 for a comparison with other individual systems on the CENPARMI database). To improve upon these results, with the same general approach, requires an understanding of its most important limits.

A careful analysis revealed that significant improvement could not be achieved simply by fine-tuning existing classification rules or adding new subsets of such rules. The system suffers from weaknesses in the preceding step, the structural feature extraction stage, which need to be addressed on their own. Some important curvature features are simply not extracted, or are diluted in noise, or are combined with other neighbouring features when they ought to stand on their own. Overcoming these difficulties was seen as key to the success of the new system.

In this chapter, we present the extensive work which addressed this aspect. The first section introduces the feature extraction scheme of the original E4 system. To determine whether this feature extractor was a sound starting point for improvement, a comparative study of several curvature feature extractors and/or ‘corner’ detectors was conducted; the second section of the current chapter discusses this study and its results. Based on this confirmation, the specific weaknesses of our initial feature extractor were investigated in detail and different avenues for solving its problems were examined; this work is presented in section 6.3. Finally, section 6.4 offers an

overview of the new curvature feature extractor which is the foundation of our new recognition system.

6.1 Feature Extraction in E4 System

In E4, preprocessing and edge extraction generate the following information:

- the number of rows and columns in the digit image;
- the area (black pixel count) of the body region;
- a list of the (row,column) coordinate pairs for the external contour;
- the area and minimum bounding box for each internal contour (hole) enclosed within the body region.

Hence only minimal information is kept on holes, allowing the system to assess their size and location without knowing their exact shape. On the other hand, full information is available for the external contours which hold most of the useful information for recognition.

6.1.1 Initial Processing of Holes

The number and position of holes will be used as a first indication to narrow down the search for the numeral's identity in the final classification stage. However, there are cases where the position of holes is awkward or their number is larger than normally expected. Three techniques are applied in succession to handle this situation. First, holes with an area of less than three pixels are discarded. Second, if there are more than three holes, only the largest three are kept. Finally, if there is more than one hole in the top or bottom part of the image, these holes are merged if they are nearby. Details can be found in Legault & Suen [94], pages 29-32.

These hole-merging operations help to identify approximately 1% more samples. They are particularly helpful for the digits '6', '8', and '9'. For these classes, respectively 2.5%, 2.8% and 1.5% of the samples are affected.

Figure 35 shows the result of the edge extraction process for an '8' which initially had five holes. However, pixel (8,22) was removed by the same preprocessing operation as described in section 4.5.2, which reduced the number of holes to 4. The smallest of them, the one delimited by edges 5 and 6, is then abandoned. Finally, the two bottom holes are merged together. This digit was properly recognized by E4 as an '8' with two (top and bottom) holes.

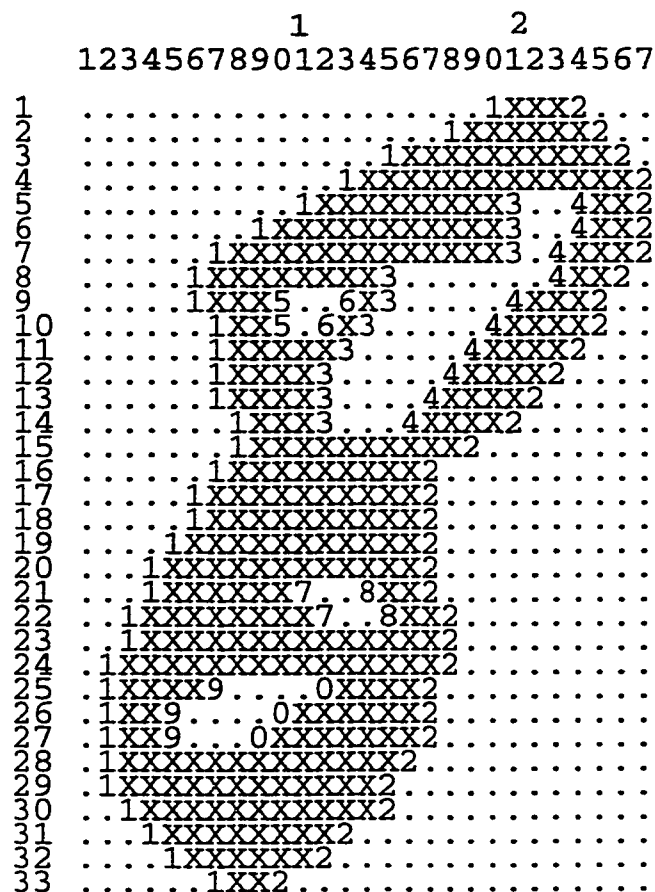


Figure 35: The Preprocessing of Multiple Holes

6.1.2 Structural Features From Outer Contours

In E4, as in our new recognition system, the aim of the feature extraction process is to capture the essential shape features along the external contours, i.e. to extract

those ‘significant’ concave and convex regions which will allow reliable identification. Before the features are extracted, a simple contour smoothing technique is applied as was described at the beginning of Chapter 5.

In Legault & Suen [95], a new method was presented for piecewise approximation of contours with parametric cubics and quartics. The first phase of this method is to partition the contour by selecting a small number of *reference points*. For the E4 recognition system, the extraction of structural features from the outer contours was essentially derived from this selection of reference points.

Let $[i, j]$ be the sequence of consecutive contour points starting at point i and ending at point j , and ϕ_j be the deviation angle, in radians, at point j . We define an *arc* as a maximal sequence of contour points $[k, k + L]$, with $L \geq 0$, satisfying the following two conditions:

1. $|\phi_k| \geq 0.005$
2. $\phi_j \times \phi_{j+1} \geq 0.005$, for $j \in [k, k + L - 1]$

By “maximal”, we mean that the sequence $[k, k + L]$ is not a proper subset of another longer arc. The length of the arc is L .

The first condition causes points where $\phi_j = 0$ to be skipped; hence a new arc begins only when the next non-zero value is encountered. The second condition ensures that arcs are sequences of points where the angle change has the same sign. Arcs with positive angle changes correspond to convex portions of the contour and arcs with negative angle changes to concave portions. The second condition also causes an arc to end when the contour becomes nearly straight again.

The algorithm examines the list of contour points and locates the arcs one by one. For each arc, the algorithm computes the cumulative deviation angle $\Phi^* = \sum_{j=k}^{k+L} \phi_j$; it also computes ϕ^* and j^* defined as $\phi^* = |\phi_{j^*}| = \max |\phi_j|$, for $j \in [k, k + L]$.

Rule #1, generating almost all the reference points, is defined as follows. A *reference point* is created within an arc when $|\Phi^*| \geq 0.5$. The contour point of index j^* is chosen as the reference point if $\phi^* \geq 0.75$. Otherwise, the middle point of the arc and its two neighbours on each side are examined; among these 5 points, the point j for which $|\phi_{j-1} + \phi_j + \phi_{j+1}|$ is maximum is taken as the reference point.

There are, in addition, three minor fine-tuning rules which can create or discard reference points. Rule #2 will discard reference point n , if its Φ^* -value is less than 0.8 *and* if reference points $(n - 1)$ and $(n + 1)$ are less than 13 points apart. Conversely, rule #3 will create an intermediate reference point between 2 consecutive reference points generated by rule #1, if they are far enough and cumulative deviation angles are large enough¹. Finally, there are cases where contours are so simple and smooth that the previous rules generate fewer than 2 reference points; then, rule #4 creates a first reference point at the contour point j such that $|\phi_{j-1} + \phi_j + \phi_{j+1}|$ is maximum; a second point is chosen half way further around the contour.

The overwhelming predominance of rule #1 can be grasped more precisely with the following facts: on average, rule #3 generates only 1 new reference point per 14 samples; in some experiments conducted with the ITRI database with an improved version of the E4 system, it was seen that *dropping rules #2, #3, and #4 altogether* only caused a small 0.28% decrease in the recognition rate and a small 0.04% increase in the substitution rate.

For each arc region, this information is recorded: the starting and final points of its associated arc, the value of Φ^* , and the location of the reference point itself, .

From Reference Points to Features

The next step is to determine *features* from reference points. Quite naturally, we consider the *region* corresponding to each arc as a structural (curvature) feature. The reference point itself is considered as the *focal point* of the feature. From now on, we will use the expression ‘focal point’ to designate this point in the feature region.

Figure 36 illustrates the feature regions which are extracted by the E4 recognition system². The first and last points of each feature (arc) region are identified by small empty squares and the internal points with empty circles, except for the focal point which is a black-filled circle.

We now explain how feature regions are categorized. Of course, if $\Phi^* < 0$, we have

¹ These intermediate reference points must undergo more testing before being finally accepted. For all the technical details, refer to pages 35-39 in Legault & Suen [94].

² The contours of numerals displayed in this and other figures are those obtained *after smoothing*.

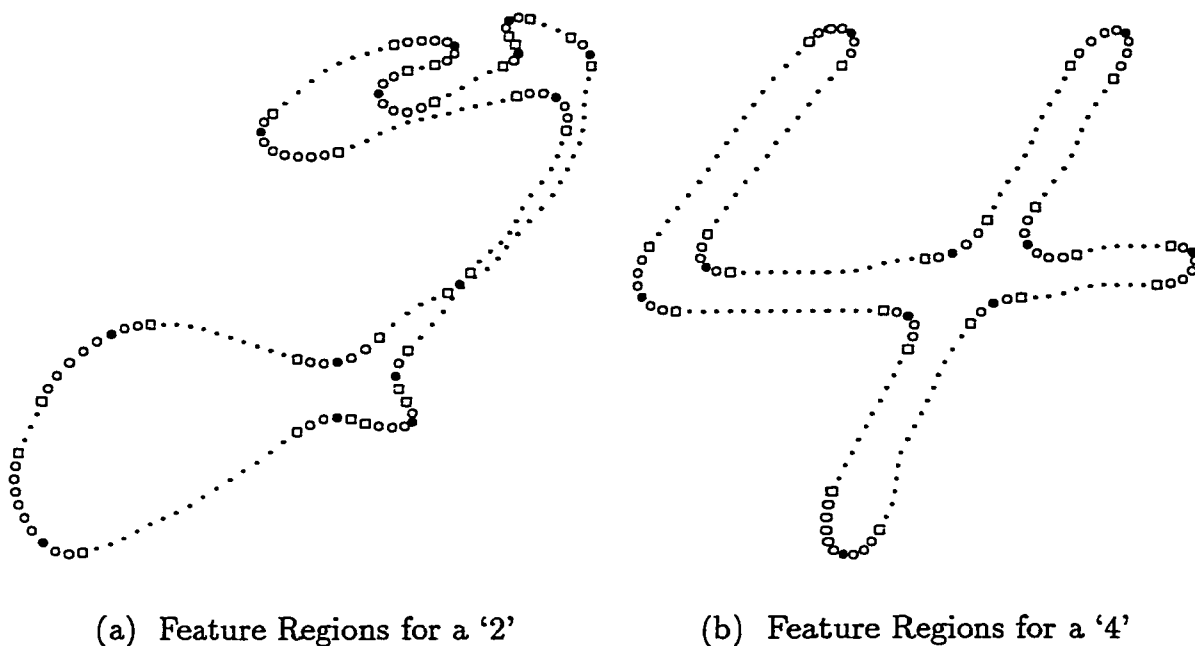


Figure 36: Feature Regions Extracted by E4 system

a concavity feature; if $0 < \Phi^* \leq 2.25$, we have a 'bend' feature; finally, if $\Phi^* > 2.25$ the feature region undergoes more testing to see if it qualifies as an endpoint.

First, two points are selected to obtain a width estimate. To choose these points, we proceed backward and forward from the focal point until we find a contour point where the deviation angle becomes negative, but no further than 5 contour points away from the focal point. Let Φ_E be the cumulative deviation angle between the 2 points found. We must have $\Phi_E > 2.45$. If so, the point obtained which is closest to the focal point is moved further, at the same distance as the other one. Then the width is computed as the Euclidian distance between these 2 points. This width divided by the maximum dimension of the image must be less than the decimal fraction given by the expression $0.18 + 0.15 \times (\Phi_E - 2.45)$. In rare cases, when no endpoints are detected with these rules, the algorithm verifies if pairs of nearby 'bend's could be combined to form an endpoint. The width is then allowed to be as much as 25% of the maximum dimension.

The expression $0.18 + 0.15 \times (\Phi_E - 2.45)$ represents one way to strike a trade-off

between cumulative deviation angle and relative width, as factors deciding whether a feature region qualifies as an endpoint or not. Conceptually, an endpoint region should be relatively narrow and have a large Φ_E . In the above expression, we allow the width of the region to be at most 18% of the maximum dimension of the image, *plus a tolerance* which increases in direct proportion to how much Φ_E exceeds 2.45.

Merging, Ordering and Attributes of Features

There is not always a one-to-one relationship between extracted feature regions and global shape features as humans would describe them. In fact, arc regions are associated with somewhat more local topological features. For example, the convex region at the bottom of the '6' in Figure 37 (a) is made up of three of our 'bend' curvature features.

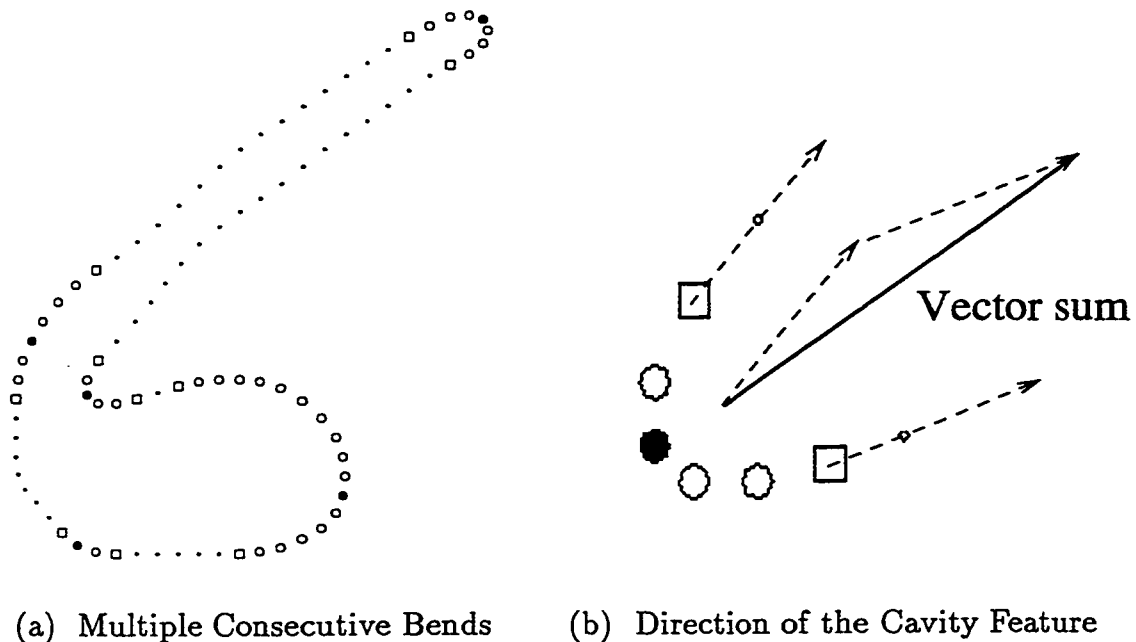


Figure 37: Features of a Numeral '6'

To address this difficulty, consecutive 'bend's are merged together to form a single new bend feature. The same approach is not taken for consecutive cavities, because

this more frequently leads to undesirable situations where distant and unrelated features are merged.

The features are re-ordered, if needed, so as to begin with the topmost endpoint. If there are no endpoints, we make sure that the list begins with a ‘bend’ feature.

For each feature region extracted, we record its type, its position, and the starting and final points of the arc or arcs associated with it. The general orientation or *direction* of the feature region is also computed. This is obtained as the vector sum of a unit vector pointing from the starting point of the feature region to the preceding contour point and a unit vector pointing from the final point of the feature region to the following contour point. The cavity feature region of the ‘6’ above is enlarged in Figure 37 (b) and the two unit vectors and their vector sum are displayed.

Brief Assessment

The feature regions extracted, with their associated focal points, are quite successful in capturing endpoints as well as cavities and ‘bend’s. They permitted the E4 system to obtain relatively good recognition results, for a single system. However, as mentioned in the introduction to this chapter, they could not allow to do much better. In fact, in some cases, the classification stage of E4 already had to compensate for some feature extractor shortcomings; for example, merged consecutive bends occasionally had to be ‘un-merged’ to test for missing endpoints or to correct for ill-positioned or mis-oriented bends.

This was somewhat to be expected since more emphasis was put on developing the classification stage of E4 than its feature extraction stage. The rules for extracting feature regions were drawn from previous work, developed with another purpose (see Legault & Suen [95]). Some modifications were made to the initial rules, but based only on limited experiments with a small set of 52 samples representing various writing styles for the 10 numerals.

To attain, with a single system, significantly higher reliability than E4 provided, it was necessary to bring much more care and sophistication into the extraction of curvature features from contours. The question arose whether or not the E4 feature extractor was a good starting point for re-design and refinement. The next section

answers this question by reporting on a comparative experiment with a number of well-known ‘corner detectors’ and other curvature feature extractor.

6.2 Comparative Study

In the literature, little is said about *how* features are extracted and there is generally no assessment of *how well* the selected features are located and extracted by the proposed method. Most publications tend to move onto the classification problem rather quickly. Furthermore, when the system is finally tested, the limitations of its feature selection and extraction are rarely mentioned although they could be largely responsible for the rejected or misclassified samples.

While the focus of our comparative study is on handwritten numerals, we believe our remarks and experimental results have broader applicability. In the main, they were published in Legault & Suen [93].

6.2.1 Problem Definition

Our goal is to detect significant concavities and convexities of contours. To be more precise, what is important here is *the identification of regions of consecutive pixels along the contours* which correspond as closely as possible to the global shape features identified by humans. For the ‘3’ of Figure 38, these regions can be roughly indicated by the sequences of circles, delimited by square boxes. For now, we are not concerned with the labelling of these regions as endpoints, smooth or sharp concavities etc.

Furthermore, we are seeking methods which generate their feature regions automatically, without human interaction; the location and relative size of these regions should be roughly invariant to rotation and scaling; the bottom of concavities and the tip of convexities, especially endpoints, should be located with some accuracy (see filled circles of Figure 38); and, finally, the methods should run very efficiently on a general purpose computer.

Except where indicated, the notation used will follow the definitions of section 5.1.

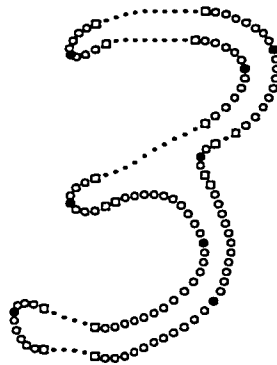


Figure 38: Global Shape Features of a '3'

6.2.2 Corners, Dominant Points and Curvature Features

The interest in corners (or dominant points) stems from the knowledge that much information about the shape of 2-D objects is concentrated at points of high curvature along their boundaries. Their detection is an attempt to capture the most striking points of a curve, as perceived by humans. While some corner-finding algorithms can be expected to perform poorly for the detection of long smooth curves along boundaries, they should be able to detect well endpoints and sharper turns. Moreover, some of the methods presented in this section are oriented towards curvature feature detection in general.

Difference of Slopes (DOS) Methods

In O’Gorman [122], methods which estimate the contour curvature at a point based on the difference between the slopes of 2 line segments which fit the data before and after that point are called DOS methods. In these methods, the angle θ_i , used for curvature estimation at point p_i , is the angle between vectors V_i^- and V_i^+ , where V_i^- joins $p_{i-m/2-s}$ to $p_{i-m/2}$ and V_i^+ stretches from $p_{i+m/2}$ to $p_{i+m/2+s}$. These vectors span an arc of length³ s , and they are separated by a gap with an arc length of m . The total length of the construction is $l = 2s + m$. Figure 39 illustrates the situation

³ Here we take the length of an arc as the number of v -vectors making up this arc.

for $s = 5$ and $m = 2$. Point p_i is indicated with a filled circle.

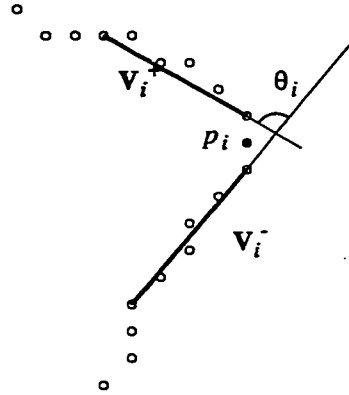


Figure 39: Illustration of DOS Methods

In Rosenfeld & Johnston [136], $m = 0$ and the curvature estimate is the cosine of the supplement of our θ . For a fixed s , the cosine estimates are computed at every point for vectors of decreasing lengths $s, s - 1, \dots$ as long as the obtained values increase. The maximum reached is taken as the curvature estimate and its corresponding arc length determines the *region of support*. Finally, points which are local maxima within (half) their region of support are kept as dominant points. To improve this method, Rosenfeld & Weszka [137], performed local averaging of the cosine values before determining the region of support and performing non-maxima suppression.

In Freeman & Davis [53], $m = 2 - s$.⁴ The ‘cornerity measure’ of each point is based on a local average of the θ -values and on the lengths of the relatively straight regions before and after the potential corner. This last factor appears important in human perception as well. Beus & Tiu [18] propose 2 improvements to the Freeman & Davis algorithm: first, they average the ‘cornerity’ measure at each point for several values of s ; second, they impose an upper bound on the lengths of the straight regions before and after the corner. With these changes, they claim that the algorithm performs

⁴ A negative value of m means that the arcs spanned by the 2 vectors V_i^- and V_i^+ overlap one another.

close to human expectations.

In O’Gorman [123], small positive values are used for m . Regions of the θ -plot which extend beyond the ‘zero-range’, defined as the curvature due to noise, correspond to corners and curves along the contour.

In Rutkowski & Rosenfeld [138], $m = 0$ but V_i^- and V_i^+ are defined differently: $V_i^- = \sum_{j=1}^s w_j v_{i-j+1}$ and $V_i^+ = \sum_{j=1}^s w_j v_{i+j}$. A corner is defined as a point where θ is a local maximum which exceeds a given threshold. In Cederberg [27], the v -vectors are replaced by their slopes in linear combinations and the measure of curvature at each point is taken as the difference between these combinations; furthermore, the w_j ’s decrease exponentially with j .

It has been pointed out that methods with fixed parameters, such as the s and m values above, are not appropriate when the features of interest exhibit different levels of detail. To overcome this problem, Teh & Chin [161] suggest that the key aspect is the determination of the region of support. To find the region of support for point p_i , they compute l_{ik} , which is the length of the chord from p_{i-k} to p_{i+k} , and d_{ik} the perpendicular distance from p_i to that chord. Starting with $k = 1$, k is incremented as long as l_{ik} increases *and* the absolute value of the ratio d_{ik}/l_{ik} increases as well. The final value of k determines the region of support for p_i . Using these regions, Teh and Chin obtain roughly the same set of dominant points with 3 very different measures of curvature.

Deguchi [37] proposes another solution. He uses a DOS method with $m = 0$ and several values of s . First, a value of s “large enough not to pick-up digitization or noise notches” is used. The curvature maxima and minima at this scale are the potential features of interest and their fine structure is determined by examining what happens to these peaks in the θ -plot as the value of s is lowered.

Methods Based on Arc-Chord Distances

There are several methods in this category which are often used to find polygonal approximations of planar curves. We briefly present 3 such methods here.

Ramer [129] adds points to a set of ‘break points’ until all contour points satisfy a certain colinearity test. More precisely, let us say that we have a set of k break points.

A chord is drawn from each break point to the next. Then, for each chord, distances are computed from each point of the associated arc to that chord. The point, among all contour points, for which this distance is largest *and* exceeds a certain threshold is added to the set of break points and the process is repeated until no more points can be found.

Ansari & Delp [8] use one of many versions of a split-and-merge algorithm. Starting with an arbitrary set of break points, the ‘splitting’ part of the algorithm is similar to Ramer’s method. The ‘merging’ part examines triplets of consecutive break points B_{i-1} , B_i and B_{i+1} and eliminates B_i if all distances from points on the associated arcs to the chord joining B_{i-1} to B_{i+1} are less than the given threshold.

In Phillips & Rosenfeld [127], an arc length s is chosen. For each boundary point p_i , the maximum distance $dmax_i$ from this point to any chord whose arc has length s and has the point p_i in its interior is obtained. A point is considered a ‘partition point’ if $dmax_i$ is a local maximum and it exceeds a given threshold.

Methods Based on Gaussian Smoothing

Asada & Brady [10] use a parametric representation of the contour relating the orientation of the tangent to the curve (i.e. the angle it makes with respect to a fixed direction) to the arc length along the curve. Then they examine the convolution of this contour representation with the first and second derivatives of a Gaussian filter, for different scale values (σ). Knowing the filtered responses of basic curvature changes (called ‘corner’, ‘smooth join’, ‘end’, ‘crank’, ‘bump’ and ‘dent’), it is possible to locate them by examining the movement of peaks and zero crossings across several scales.

Rattarangsi & Chin [130] use a parametric representation of the contour relating the x-y coordinates of contour points to the arc length along the curve. Curvature is computed using Gaussian-smoothed coordinate functions. Again the movement of local maxima and local minima of curvature across different scales (σ) is used to locate their ‘Γ’, ‘END’ and ‘STAIR’ models of curvature change.

Using a single value of σ “based on a priori knowledge about the scale of the boundary”, Ansari & Delp [8] follow the same approach to find the curvature maxima

and minima using Gaussian-smoothed coordinate functions. When they use these points as the starting set for their split-and-merge algorithm presented earlier, they call the resulting method “curvature-guided polynomial approximation”.

Comparison and Discussion

Early methods were often weak at detecting neighbouring corners and some more recent methods have tried to correct this problem. Other criticisms have been made of early methods. Some have argued that the need for the user to specify input parameters is a shortcoming; or that features may exist at different scales on the same contour and that a single set of parameters will not allow their appropriate detection.

Since corners (or dominant points) are *not* rigorously defined, the comparison of methods is not straightforward and the value of different algorithms is best judged by direct examination of their results against the specific requirements of particular applications. For example, the fact that features may exist at different scales on the same contour does not mean that the users are equally interested in all of them. Methods which have no input parameters and detect *all* corners at the finest level of detail [161] may not be appropriate for many applications because they can pick up too many noisy details; this is also the case for other recent methods which detect some corners or clusters of corners and then resolve them to their finest level of detail ([37] [130]).

For DOS methods, O’Gorman [122] has shown analytically, using certain assumptions, that the signal-to-noise ratio is better for small positive values of m (DOS⁺). For equal values of the signal-to-noise ratio, the same study shows that this DOS⁺ method yields narrower peaks than the Gaussian smoothing method, thus having better signal detectability. Some empirical comparisons are also presented to support these conclusions.

In recent years, other comparative studies have been published. While their conclusions are interesting, it should be kept in mind that the number of images used in these studies is generally *very small* (from 2 to 8 images) and that image resolution is much lower than is common with today’s technology. Teh & Chin [161] used 5 images

to compare their algorithm to 5 other algorithms: Rosenfeld-Johnston, Rosenfeld-Weszka, Freeman-Davis, Anderson-Bezdek [6] and Sankar-Sharma [143]. Based on the criteria they used, their algorithm did better, but the Rosenfeld-Johnston and Rosenfeld-Weszka algorithms also performed quite well. On some images, the Sankar-Sharma and Anderson-Bezdek methods could not locate many endpoints properly.

Rattarangsi & Chin [130] compared the same 6 algorithms to their scale-space algorithm using a digital ellipse and one of the 5 images of the Teh-Chin study. Their algorithm gives results which are comparable to Rosenfeld-Johnston. The performance of the Teh-Chin algorithm on the digital ellipse was especially poor, yielding a very large number of dominant points.

Liu & Srinath [102] used 8 test-images to compare 6 corner-detection schemes: Rosenfeld-Johnston, Rosenfeld-Weszka, Rutkowski-Rosenfeld, Medioni-Yasumoto [113], Beus-Tiu and Cheng-Hsu [30]. Their results show that the Medioni-Yasumoto and Cheng-Hsu methods are very sensitive to the smoothness of boundaries; furthermore, the Cheng-Hsu algorithm often cannot properly localize corners but instead it detects unwanted spurious corners. The Beus-Tiu method performed best on these images and the Rosenfeld-Johnston algorithm also did very well.

Ansari & Delp [8] show that Ramer's method and their own split-and-merge algorithm are sensitive to the scale and orientation of the image. Their curvature-guided polynomial approximation method is less sensitive to orientation but still sensitive to scale. More robustness is achievable if only "cardinal curvature points" are kept; these are the extreme curvature points when the boundary is smoothed by a Gaussian filter with a large standard deviation (σ).

Based on the above investigations, it appears that Rosenfeld-Johnston, Rosenfeld-Weszka, Beus-Tiu, and Teh-Chin achieved some of the best results. However, we note that the investigation reported in this chapter was conducted a few years ago; since then several new 'corner finding' methods have been published and a few new comparative studies. See, for example, Kadonaga & Abe [69] and Zhu & Chirlian [174].

6.2.3 Curvature Features in Recent OCR Literature

We now examine the methods of detecting curvature features found in the recent OCR literature and comment on their suitability for the solution of our problem as defined in section 6.2.1.

In Ahmed & Suen [3], the detection of certain relationships between edges of four basic types implies the presence of 2 kinds of “fundamental cavities”. In a horizontal scan of the binary image, north- and south-facing cavities are obtained and a vertical scan is needed to extract east- and west-facing cavities. However these cavities are not appropriate for our purposes: a south-facing cavity, for example, may have any number of east- or west-facing cavities on its left and right walls.

Mitchell & Gillies [114] and Kanungo & Haralick [71] extract features using the tools of mathematical morphology. Their definition of cavities as regions surrounded by strokes in at least 3 of the 4 cardinal directions, also does not suit our purposes for the following reasons. First, their features are not sequences of points along the contour, but 2-D regions of the image. This problem could be solved by using the intersection of these regions with the image contour. However, such cavity features are not invariant to rotation. In fact, for some orientations, cavities—no matter how large—will go unnoticed: this is the case for the four cavities created by the intersection of a horizontal stroke and a vertical one. For word recognition, Duderstadt et al. [42] use a number of features, including some “cavities” which are defined as sets of background pixels having exposure to the north, south, east or west sides of the bounding box⁵, but also to both the north and west sides (NW), the south and west sides (SW) etc... In this last case, the contour sections associated with the sets of background pixels may have no curvature at all.

In Ho et al. [62], global and local features are used for word recognition. Among the local features are letter shape features, including curves detected using Deguchi’s multi-scale curvature analysis.

D’Amato et al. [35], Commike & Hull [34] and Hull et al. [68] use a similar set of features of 8 types based on the amount of curvature present at any point. Thus an endpoint may be labelled as SPUR or STUB; a (less sharp) convexity as WEDGE,

⁵ These are identical to the cavities we just discussed.

CURL or ARC; and a concavity as NULL, INLET or BAY. The first step in the extraction of these features is the computation of a curvature estimate at each point which is analogous to the DOS methods; the angle of curvature at point p_i is given by

$$\theta_i = \sum_{j=i-s-m/2+1}^{i-m/2} c_j - \sum_{j=i+m/2+1}^{i+m/2+s} c_j.$$

Lee et al. [86] presents methods of recognizing handprinted and degraded machine-printed characters. For handprinted characters, they investigate two algorithms which detect concavities to supplement their compressed line adjacency graph (c-LAG) representation. One algorithm, which they call angle accumulation algorithm, might be of use to our problem. Using differential chain code values as a measure of local change in curvature, they define a concavity as “the longest sequence of perimeter points whose angular change between consecutive elements are all non-negative⁶ and their accumulated sum is greater than 1”. This basic definition is modified to accommodate a temporary angle change of -1 , if its preceding and succeeding non-zero angles are positive.

Finally, there is our own method as presented above in section 6.1.2.

6.2.4 Experiment and Results

Selected Methods and Implementation

Based on our appreciation of the suitability of the above-mentioned methods to achieve our stated goals⁷, we finally compared 7 schemes. We chose 4 corner and curvature detection algorithms: Rosenfeld-Johnston, Rosenfeld-Weszka, Beus-Tiu, Teh-Chin; as well as 3 algorithms from the recent OCR literature: D’Amato et al. Hull et al. Commike & Hull, Lee et al. Legault & Suen. The methods were implemented based on the information given in their papers⁸. Results provided in the literature were also reproduced to verify our implementations.

⁶ This definition incorporates within the concavity the straight portions of contour which may precede and follow the actual curved portion.

⁷ And because of time constraints!

⁸ In some cases, additional information was obtained through private communications.

It is obvious that each method selected could be modified and improved upon for the purposes of our application. However, our goal in this experiment is to compare these methods—as published, or with only slight modifications—for curvature feature detection.

For the Rosenfeld-Johnston and Rosenfeld-Weszka methods, the user-supplied parameter was set to $\max\{N/10, 4\}$, where N is the number of contour points of the blob under consideration. The lower bound of 4 was necessary for numerals composed of several (smaller) connected components. We made the additional requirement that a contour point must have a region of support extending at least 2 pixels on either side to be considered as a dominant point. These minor changes improved the results⁹.

The Teh-Chin algorithm was implemented in its 3 versions, using different “measures of significance”. The results obtained were almost identical. Thus we only retained the version using the k-cosine measure of significance for comparison purposes.

For the approach of Lee et al, we did *not* eliminate diagonal-facing concavities, since these do convey relevant information for handwritten numerals. To detect convexities, all differential chain code values were negated.

For the Beus-Tiu method, the number of corners to be extracted is normally an input parameter. Since this is not a reasonable request for our application, several trials were conducted to select the other input parameters so that all corners detected by the method could be kept provided their cornerity value exceeded a threshold value. Taking several factors into account, the best results were obtained as follows: the K_j -values were computed using $\ln(t_1)$ and $\ln(t_2)$ for $t_1, t_2 > 2$ and 1.0 otherwise; the maximum length of straight arms was set to $N/20$ (N being the number of contour points); values of s_1 and s_2 were set to 3 and 5 respectively for $N < 125$, to 8 and 12 respectively for $N \geq 425$, and, for $125 \leq N < 425$, we used $s_1 = \lfloor (N + 25)/50 \rfloor$ and $s_2 = s_1 + 3$. Finally, we used the cornerity value as such (and not its absolute value) and retained all local maxima *and minima* with absolute cornerity value larger than 2.0.

For the method of D’Amato et al. parameters were set as given in U.S. Patent

⁹ Tests were also performed using $N/15$ instead of $N/10$ but the latter gave better results.

4 628 532 (see Stone et al. [150]). We used $n = 4$ and $m = 2$. The following modification was applied: in the original scheme, region-length is reset to 1 upon $P1 \Rightarrow P2$ and $P2 \Rightarrow P3$ level transitions; since this corresponds to entering a more convex part of a convex region, we decided to keep the original starting point and continue incrementing the region length rather than resetting it to 1.

The Experiment

All methods were tested on 100 binary images of handwritten digits, selected from a subset of 5 000 unconstrained handwritten samples, taken from the 20 000-digit Montreal-Concordia database. This database was already mentioned in section 2.4.2 under the subsection heading ‘Transferring Expertise Across Databases’¹⁰. Ten samples were chosen for each numeral class, offering a variety of styles and sizes. Some have more than one connected component. They are shown in Figure 40.

For each sample, regions of the external contour corresponding to the global shape features to be detected were determined manually by two human volunteers (the author and his supervisor, Prof. C.Y. Suen). For each such region, they recorded its starting and ending points, as well as its ‘focal point’ (i.e. tip of an endpoint, or bottom of a concavity or convexity); they also indicated whether the region was an endpoint or a curved section.

The ‘fuzzy rules’ governing our selection can be summarized as follows: skip relatively straight portions of the contour, including some with wiggles; group together within a single region points where the direction change has constant sign. This last rule is applied somewhat loosely; for example, a region may include short sections with no direction change or even with *small* opposite direction change, if intuitively the resulting set of points seem to belong to the same curved region. One example was already given in Figure 38; others can be seen in Figure 41. In these figures, the selected regions are the sequences of circles, delimited by square boxes; the focal point of every region is identified by a filled circle.

The performance of each method is evaluated by comparing the feature regions

¹⁰ Note that in Legault & Suen [93], it is said erroneously that the 100 images used in this experiment were selected from the CENPARMI database.

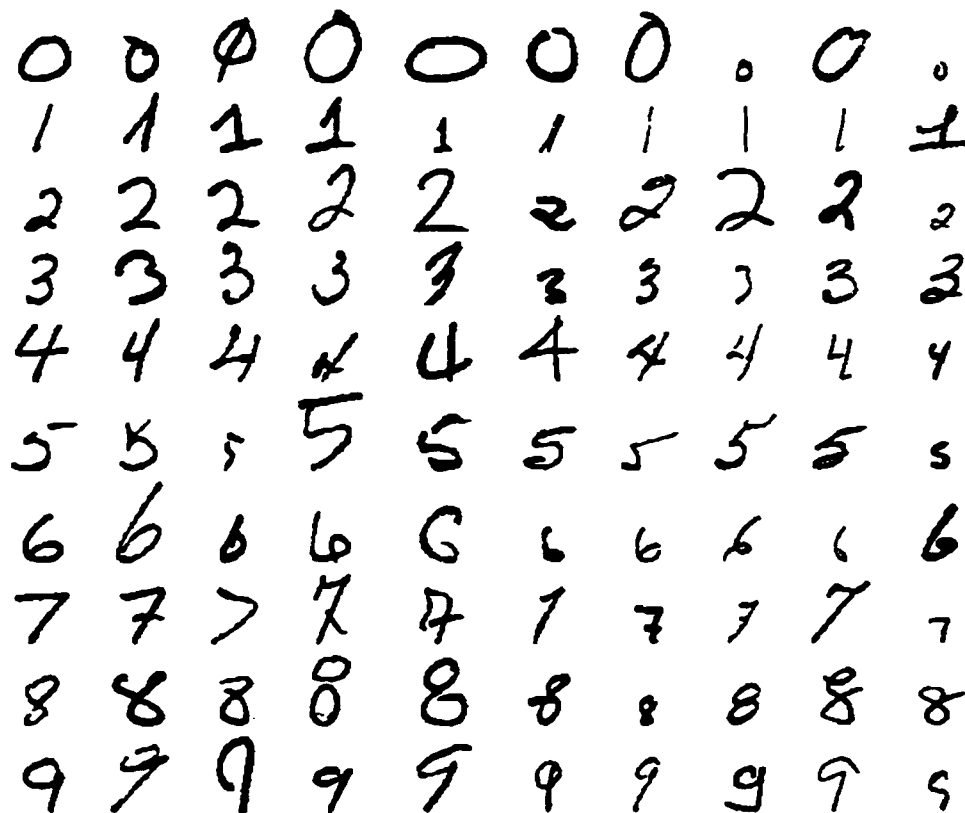


Figure 40: Numerals of Varying Styles and Sizes

that it detects to the feature regions established by the 2 human subjects. In order to have a quantitative basis for comparison of the 7 methods, a ‘measure of goodness’ (MG) was computed for each method and each sample. Let n_i be the number of regions detected by humans for sample i and R_i be the set of the indices of all contour points belonging to these regions. Furthermore, let S_i^k be the set of the indices of all contour points belonging to the regions selected for sample i by method k and m_i^k be the number of regions detected. We define $Q_i^k = R_i \cap S_i^k$ and $f = \min\{n_i/m_i^k, m_i^k/n_i\}$. The ‘measure of goodness’ of method k in determining the global feature regions of sample i is then defined as

$$MG_i^k = f \left(\frac{\sum_{j \in Q_i^k} w_j}{\sum_{j \in R_i} w_j} \right),$$

where w_j is the weight associated with each point whose index belongs to R_i . As

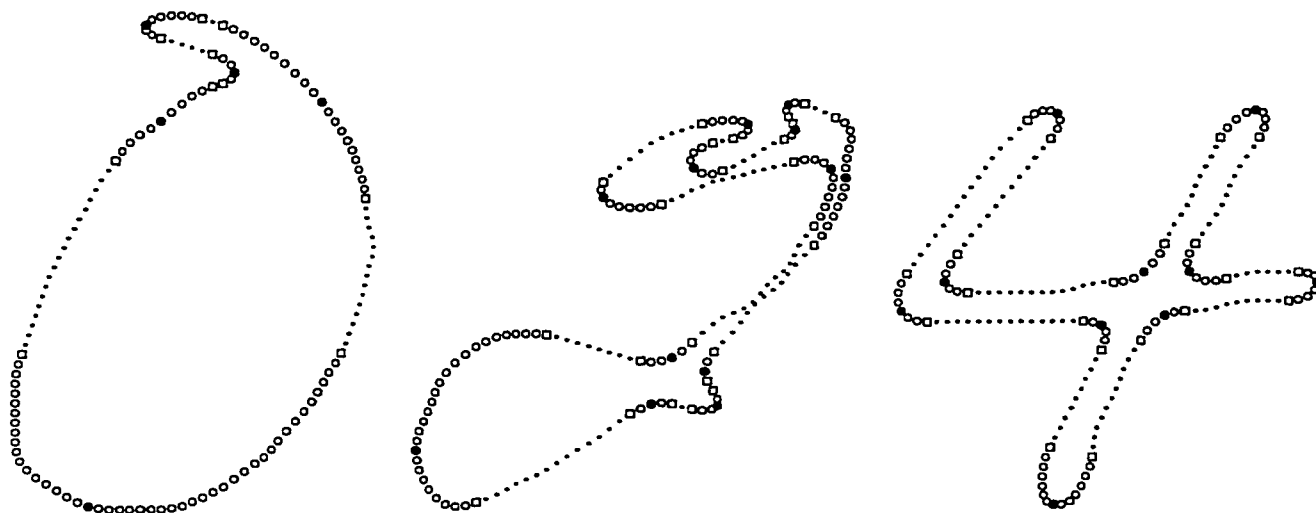


Figure 41: Other Examples of Feature Regions Selected Manually

defined, MG_i^k will impose a penalty to methods for each feature-region point not detected and also for finding a number of feature regions which is not the same as that established by humans. Ideally, for each sample, the weights w_j should depend on the relative importance of the individual feature regions for recognition purposes and on the importance of each point within its region. For simplicity, we have assigned a weight of 5 to all points in ‘endpoint regions’ and a weight of 2 to all other points in R_i . In general, these weights could be adjusted according to their significance in the intended application.

Since different numeral classes have different kinds of curvature features¹¹, some methods which do well for some classes might not do as well for others. Thus, for each method, we have computed the average ‘measure of goodness’ for each numeral class. The results are presented in Table 16.

Separate measures of goodness were also computed taking into account only the endpoint regions, as identified by humans. These are shown in Table 17.

¹¹ Numerals ‘1’, ‘4’ and ‘7’ tend to have only straight portions and sharp corners; other numerals have some long smooth curves.

Method	0	1	2	3	4	5	6	7	8	9	Overall
Beus-Tiu	0.46	0.84	0.68	0.70	0.86	0.73	0.56	0.82	0.64	0.62	0.69
D'Amato	0.41	0.77	0.71	0.77	0.82	0.77	0.63	0.71	0.63	0.68	0.69
Lee et al	0.46	0.57	0.67	0.68	0.73	0.64	0.72	0.71	0.67	0.55	0.64
Legault-Suen	0.39	0.84	0.73	0.74	0.82	0.72	0.56	0.80	0.61	0.64	0.68
Rosenfeld-Johnston	0.42	0.58	0.63	0.72	0.64	0.62	0.55	0.62	0.55	0.55	0.59
Rosenfeld-Weszka	0.44	0.75	0.76	0.80	0.73	0.71	0.59	0.70	0.60	0.63	0.67
Teh-Chin	0.14	0.21	0.20	0.22	0.22	0.21	0.16	0.23	0.22	0.17	0.20

Table 16: Average Measures of Goodness for the Methods Tested

Method	0	1	2	3	4	5	6	7	8	9	Overall
Beus-Tiu	0.89	0.86	0.76	0.77	0.87	0.85	0.78	0.80	0.81	0.80	0.82
D'Amato	0.87	0.86	0.90	0.85	0.87	0.85	0.86	0.82	0.77	0.84	0.85
Lee et al	0.82	0.70	0.83	0.71	0.70	0.78	0.90	0.69	0.76	0.78	0.77
Legault-Suen	0.89	0.89	0.93	0.90	0.91	0.92	0.87	0.88	0.91	0.92	0.90
Rosenfeld-Johnston	0.81	0.88	0.92	0.87	0.89	0.81	0.86	0.91	0.87	0.90	0.87
Rosenfeld-Weszka	0.93	0.85	0.92	0.98	0.96	0.87	0.88	0.97	0.99	0.93	0.93
Teh-Chin	0.41	0.36	0.36	0.34	0.32	0.35	0.32	0.32	0.35	0.31	0.34

Table 17: Average Measures of Goodness for Endpoint Regions Only

Analysis of Results

Among the methods for which results are presented in Table 16, the Beus-Tiu (BT), D'Amato (DA), Legault-Suen (LS) and Rosenfeld-Weszka (RW) algorithms give best results in terms of their overall MG -values. The Teh-Chin (TC) method performs very poorly. For the 4 best overall methods, the worst MG -values are obtained for classes 0, 6, and 8. The best MG -values are obtained for classes 1, 4, and 7 (for LS and BT), 1, 4, and 5 (for DA) and 1, 2, and 3 (for RW).

For endpoint detection, the RW, LS, and RJ methods achieved the best performances. Again, the TC algorithm scored very low compared to all other approaches tested. We note that the performance of LS is most uniform across numeral classes

ranging from 0.87 to 0.93; by comparison, the range of MG -values for RW is from 0.85 to 0.99. However, it should be noted that the overall statistics do not tell the entire story. They are based on our definition of MG which has its shortcomings (see next subsection). Thus it is important to examine the results of each method on individual samples as well. Such scrutiny can reveal the specific strengths and weaknesses of the methods studied. A few samples are now presented for the 4 best methods overall (BT, DA, LS, and RW), which also illustrate well their main weaknesses.

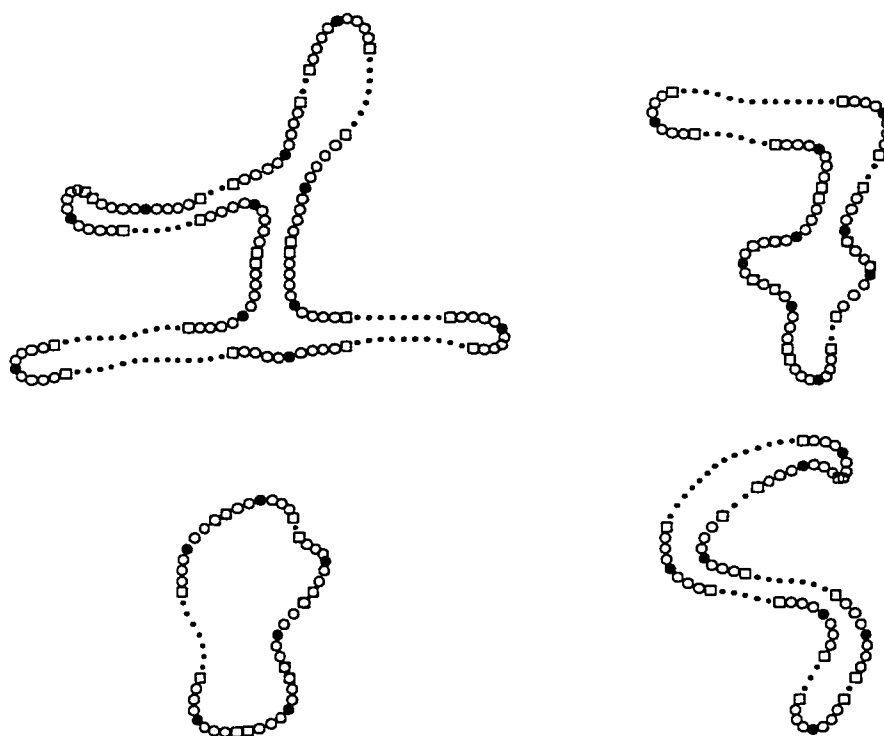


Figure 42: Weaknesses in Beus-Tiu (BT) Method

The typical weaknesses of the Beus-Tiu (BT) method are illustrated in Figure 42. For the '1', the focal point (filled circle) of the middle left endpoint is not well located; also, a useless feature was extracted in the middle of the flat base. For the '7', a concavity was missed at the bottom right of the digit (under the transversal bar). For the '8', two cavities were not extracted: the left cavity in the middle portion and a smaller top right cavity. Finally, for the '9', the focal point of the top right endpoint

is again poorly located.

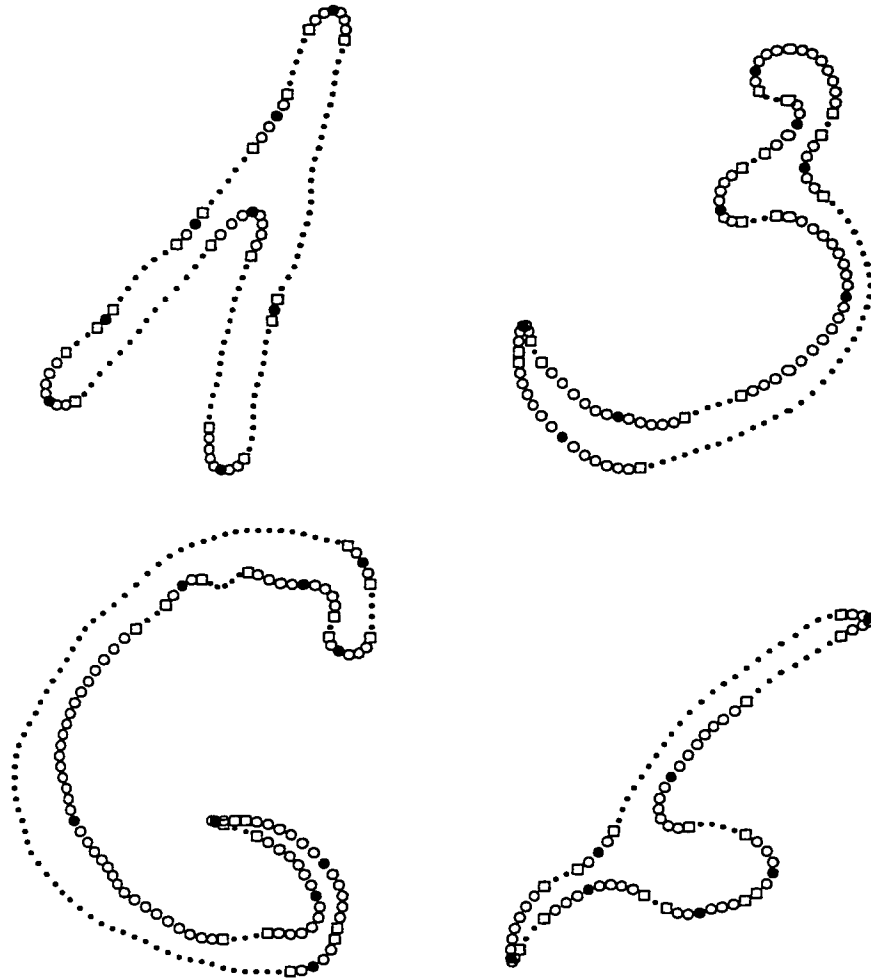


Figure 43: Weaknesses in D'Amato (DA) Method

The typical weaknesses of the D'Amato et al. (DA) method are illustrated in Figure 43. For the '1', three insignificant small cavities were extracted on the top side of the large diagonal stroke, ; another such irrelevant feature was extracted on the right side of the more vertical stroke. For the '3', a large convex region was undetected in the bottom right profile; also the top endpoint and the preceding 'bend' are combined into a single feature region. For the '6' on the left, the convex region which makes up all the left profile was completely missed! For the '6' on the right,

the focal point of the middle cavity in the right profile is ill-positioned; in addition, the same feature region extends too high into a straight portion of the contour.

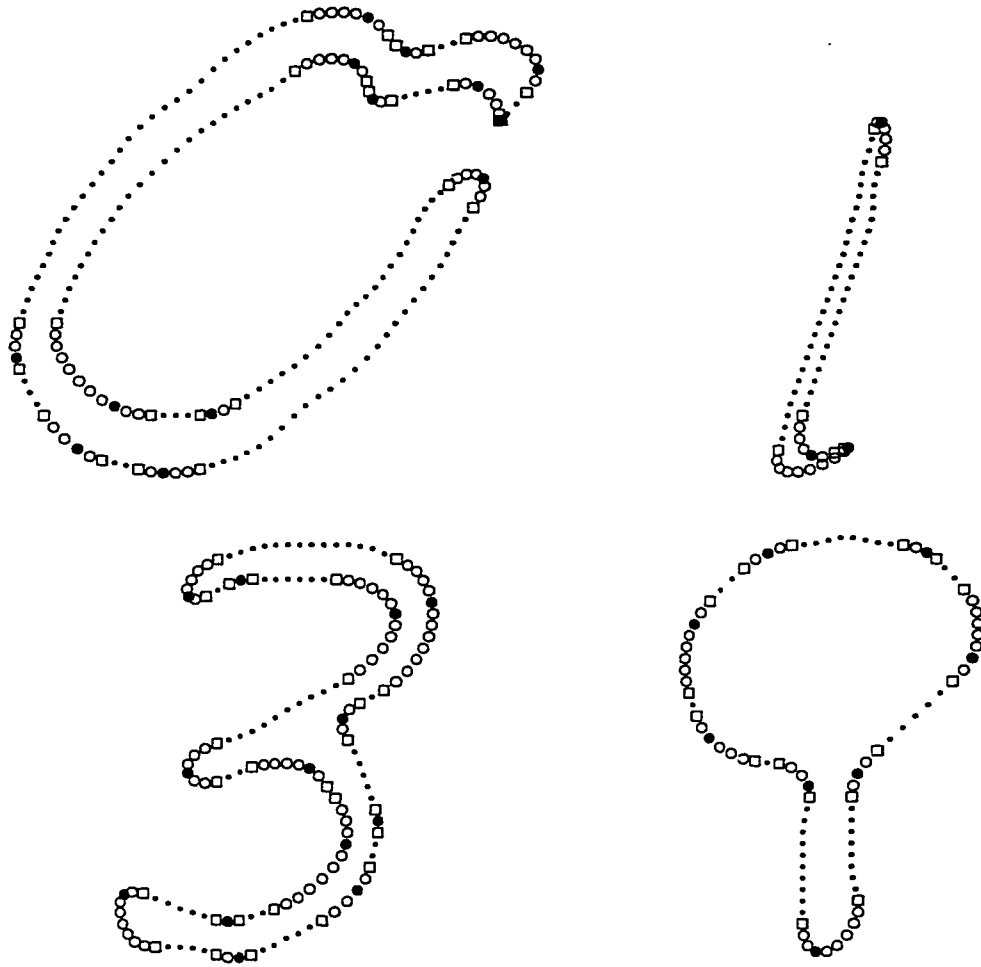


Figure 44: Weaknesses in Legault-Suen (LS) Method

The typical weaknesses of the Legault-Suen (LS) method are illustrated in Figure 44. For the '0', the curved stroke at the bottom is not captured as a single feature but as 2 consecutive cavities on the top side and 3 consecutive 'bend's on the bottom side. For the '1', the convex portion preceding the bottom endpoint is combined with the endpoint into a single feature. For the '3', again, the curved stroke in the bottom half is split into 3 cavities on one side and 3 'bend's on the other; also the location

of the focal point for the bottom endpoint could be improved. For the '9', the large convex region around the hole is again split into 5 smaller consecutive 'bend's.

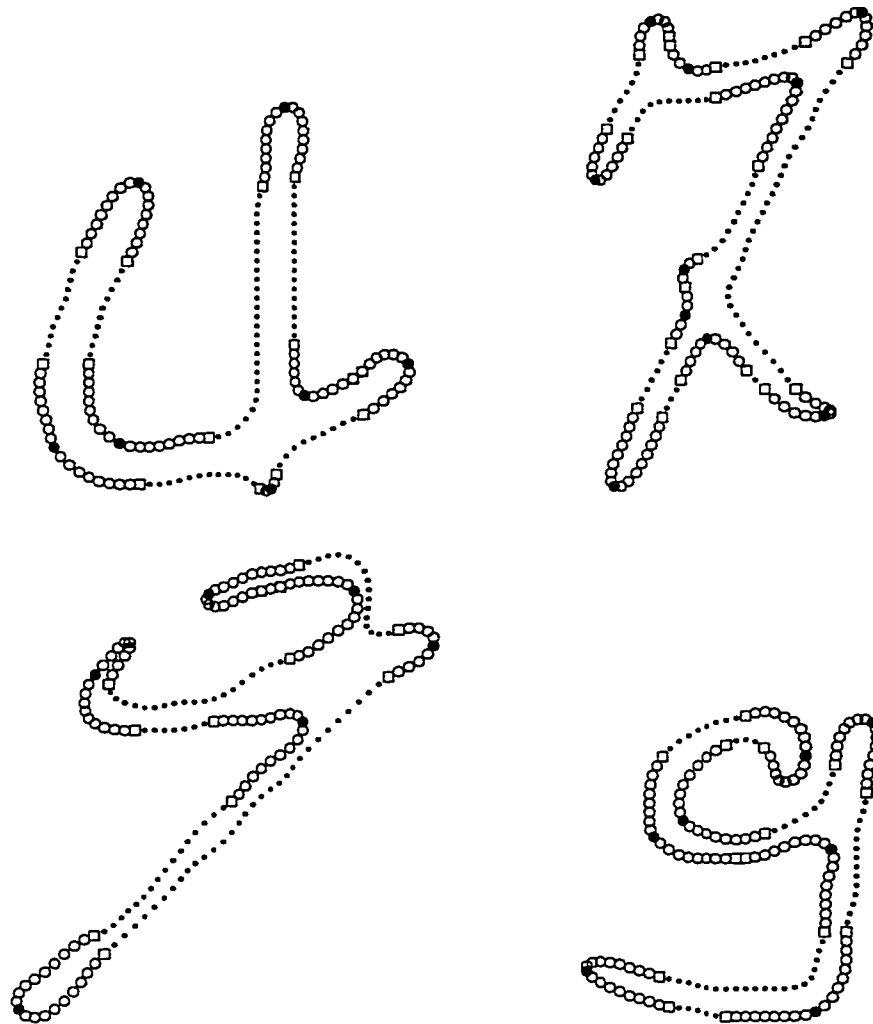


Figure 45: Weaknesses in Rosenfeld-Weszka (RW) Method

The typical weaknesses of the Rosenfeld-Weszka (RW) method are illustrated in Figure 45. For the '4', three cavities out of 4 were missed at the crossing of the horizontal and vertical strokes. For the '7', one cavity went undetected on the lower side in the top left portion of the image and 2 more cavities were missed in the middle again at the crossing of two strokes. For the '9' on the left, the 'bend' and

cavity features making up the wiggle in the top right portion of the image were both missed; furthermore, in the top left portion of the image, an endpoint region and the immediately following 'bend' region were captured as a single feature for which the focal point is strangely placed. For the '9' on the right, again the location of the focal point for the top curved endpoint is poorly chosen.

In summary, the LS method was found to be quite robust and reliable in the sense that it generally detects some points within each and every feature region. Only once is a feature region missed altogether. On the other hand, its major weakness is in the extraction of long smooth curves. In general, such curved portions will not be detected as a single global region; instead, as was seen in the examples, the global curvature feature may be split into several consecutive subsections of those curves.

On the other hand, the BT, DA, and RW methods are better able to extract these large curved regions as single features; and, on average, they capture a larger number of points within any detected feature regions. However this advantage is offset by the fact that they miss certain features completely. In addition, some features will have poorly localized focal points (for the BT and RW methods, this will happen mostly for endpoints whereas for the DA method it will occur mostly for cavities).

All four methods (BT, DA, LS, and RW) also suffer from other common problems, to varying extents. These include the detection of small, useless, feature regions and the poor localization of endpoints when strokes have curved endings.

The Rosenfeld-Johnston (RJ) and Lee et al. (LE) methods suffer from the same defects as the BT, DA, and RW methods, only to a larger extent. The definition of a concavity (convexity) in LE includes the straight portions of contour which precede and follow the actual curved portion. In some cases, these extensions may even cover the early part of following regions of opposite curvature.

The poor performance of TC is linked to its detection of a large number of tiny consecutive regions, thus preventing the method from extracting features at a higher, more global, level. Even endpoints are often split into several regions which account for the low *MG*-values of this method for endpoints as well.

Improving Comparative Studies

To conclude section 6.2, we will say a few words on how comparative studies of curvature feature extraction schemes could be improved. First we will return on our ‘measure of goodness’ and then we will comment on the problem of image sets for testing.

As stated above, our ‘measure of goodness’ statistics were far from telling the whole story. There are several weak spots in our definition of MG :

- There is no verification of *one-to-one relationship* between the regions detected by a method and the reference regions established by the two human observers; our definition rewards for extracting any point belonging to R_i , the set of *all* reference contour points and penalizes globally for detecting too many or too few regions.
- A uniform weight is given to all reference contour points in a feature region (5 in endpoint regions, 3 in ‘bend’ and cavity regions); the ‘heart’ of a feature region (i.e. points located at and around its focal point) should probably be given more weight than points further away.
- There is no penalty for detecting points lying outside of curved regions.

The first weakness mentioned above is indeed a major one. Because of this, we could NOT vary the parameters of the individual methods tested and simply retain the version which gave the highest MG -value. The actual feature regions extracted for the set of 100 test images had to be carefully examined for the various runs: sometimes higher MG -scores were obtained for qualitatively worse feature extraction performance! The results presented in Tables 16 and 17 are the best performances of the methods tested based on MG -values AND qualitative assesment.

One way of tackling the above weaknesses could be to replace MG -values for each numeral class by a number of different measurements which would be obtained, *across numeral classes, but separately for endpoint, concave, and ‘bend’ regions*. These measurements might include:

- the percentage of points in R_i detected;
- the number of points not in R_i which were also detected, as a percentage of points in R_i ;
- the fraction of the regions of interest (endpoints, concave, or ‘bend’ regions) which are completely missed by a method;
- the average number of regions of a method which overlap into a reference endpoint, concave, or ‘bend’ region;
- the average number of pixels (relative to digit size) separating the detected and the reference focal points.

A more general problem regarding the evaluation of curvature feature detectors is the lack of meaningful common benchmarks. Several studies, as mentioned previously, focus on a very small set of images which may have ‘historical value’ but which are generally unrelated to current real-life applications. In our work here, we have used a larger number of images (100) from one application: handwritten numeral recognition. It might be a good idea for researchers involved in concrete pattern recognition work in various areas to create databases of images typical of their field, clearly identifying the regions whose detection is key to successful classification in their applications. After experiments and exchanges, some common standard datasets might be recognized as references for testing new schemes.

6.3 Detailed Autopsy of E4 Feature Extractor

The preceding section has established that the E4 feature extraction scheme compares favorably with other corner detectors and curvature feature extractors. Yet, it has flaws which are partly responsible for the 1.6% error rate of the E4 recognition system on the CENPARMI database (for which it was developed). Furthermore, these weaknesses also account for a significant portion of the 5.10% and 5.03% error rates achieved respectively on the Concordia-Montreal and ITRI (Taiwan) databases (See section 2.4.2, under the heading *Transferring Expertise Across Databases*).

In view of our goal of developing a much improved recognition method for handwritten digits, these serious shortcomings must be addressed. The feature extraction process should be more reliable and more robust. By reliability, we mean the ability to extract all the relevant and needed features for correct classification, regardless of writing instruments, writing circumstances, image capture method and resolution, etc. By robustness, we mean the ability to discard, at the same time, all that is not relevant. In short and ideally, *always* getting what we need and *only* what we need.

This section carries out a detailed investigation of the shortcomings of the E4 feature extractor and points out avenues for improvement. To our knowledge, such analysis has rarely been carried out. In the literature, once recognition systems have been developed, there is generally no assessment of *how well* the selected features are located and extracted by the proposed methods, of the irrelevant features retained by these methods etc. However, in our experience, inadequate feature selection and extraction is largely responsible for rejections and misclassifications, in methods relying on explicit feature extraction.

There is yet another reason why such detailed analysis of feature extractor shortcomings should be of interest. The need to use larger training sets to ensure greater recognition reliability makes it necessary to consider the automation of the training process at one point or another. Some approaches, such as neural networks or functional classifiers [48] are naturally geared to self-learning; but developing syntactic or rule-based classifiers with automatic training is not obvious. Efforts towards this challenging goal have shown that the *noise effect* is critical and remains an open problem in structural learning (see Nishida & Mori [119]). We believe that the selection of reliable features and their robust and accurate extraction is the key to solve this problem.

6.3.1 Data Used for the Investigation

Three hundred and seventy (370) samples were used in the study of the weaknesses of the E4 feature extractor and to develop our new feature extractor (see next section). To properly test for reliability and robustness, these samples were extracted from 3 databases, collected in different geographical regions and digitized at different

resolutions.

This training data was composed of 5 training sets. The first 3 sets each contain 100 handwritten numerals, 10 per class, selected blindly from the CENPARMI database (166 PPI), the Concordia-Montreal database (200 PPI), and an ITRI-Taiwan database (400 PPI). See Appendix A. The 100-sample set from the CENPARMI database, called *train_us*, was composed of the samples from sets A and B, having indices given by $(40i + 20)$ for $i = 0, 1, 2, \dots, 49$. The 100-sample set from the Concordia-Montreal database, called *train_q5000*, was composed of samples extracted from the 5000-sample free-style training set, having indices given by $(50i + 25)$ for $i = 0, 1, 2, \dots, 99$. The 100-sample set from the ITRI-Taiwan database, called *train_tw*, was composed of samples extracted from sets TW-2 and TW-3, having indices given by $(20i + 10)$ for $i = 0, 1, 2, \dots, 49$.

The final 2 sets include a small number of ‘hand-picked’ samples from the CENPARMI (38 samples) and ITRI-Taiwan (32 samples) databases, for which it had been noticed in previous work that the feature extraction scheme was yielding particularly poor results. They are called *tough_us* and *tough_tw* respectively.

6.3.2 Problems with Endpoint Extraction

In this section, we perform a careful examination of several types of difficulties encountered by the E4 feature extractor in dealing with end-regions. Ideally, the strokes leading to end-regions have constant width, their edges are parallel, and the end-regions –whether we idealize them as perfectly circular or as composed of two right-angled corners– are expected to have an Φ^* -value of π . See Figure 46. In real-life data, of course, we encounter much greater variability and things end up being quite different...

- The first difficulty may arise when a stroke curves in the vicinity of an endpoint. We would expect to see, in addition to the end-region itself, two nearby feature regions –one ‘Cavity’ and one ‘Bend’– on opposite sides of the stroke. However, in some cases, there may not be a straight enough portion of the contour to separate the end- and the bend-regions and both are detected together as a

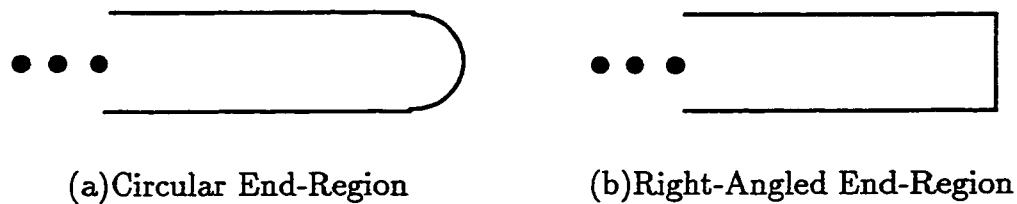


Figure 46: Idealized End-Regions

single arc. This is illustrated in Figure 47(a) where the bottom endpoint and the *following* bend-region are combined and in Figure 47(b)¹² where the bottom endpoint and the *preceding* bend-region are combined. In such cases, the value of Φ^* is much larger than π .

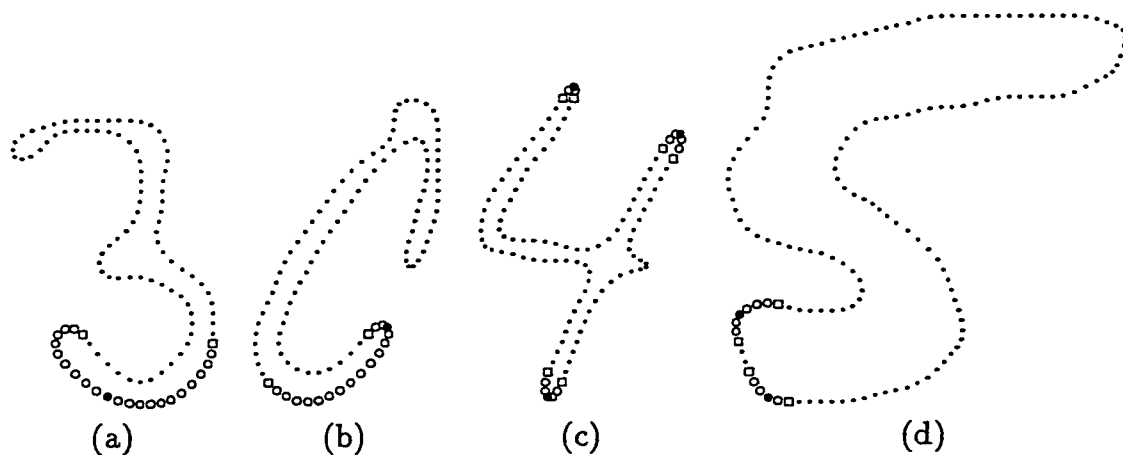


Figure 47: Defects in “E4” End-Region Extraction

For the sample shown in Figure 47(b), the value of ϕ^* was large enough to properly locate the focal point despite the poorly located feature region and the arc passed the width test to qualify as an endpoint. However, for the sample of Figure 47(a), this problem was compounded: 1) the feature region was poorly located; 2) because of this, the focal point (black circle) was ill-positioned within the feature region; 3) because of this, the arc failed the width test and did not qualify as an endpoint.

¹² For these images as well as those of Figures 47(d), 48(a) and 48(c), only the problematic region is shown to highlight the situation.

The solution to this first category of problems in the extraction of end-regions could be to detect and further examine arcs with ‘abnormally’ high values of Φ^* , trying to partition the arc into an end-region and a ‘Bend’.

- A second difficulty is that some regions which should be end-regions do not quite make the 2.45 threshold on Φ_E . Simply reducing the threshold is not a solution since this value already represents a compromise whereby some ‘true’ end-regions are missed and some ‘false’ ones are retained. An example of a missed endpoint is provided in Figure 47(c), where all the extracted endpoints are shown; the right extremity of the horizontal stroke has been missed. This problem arises when endpoints occur in the vicinity of cavities. In these cases, the smoothing operation performed on the contour tends to partially blend the 2 kinds of features, reducing the Φ^* -value for potential end-regions. A possible solution is to conditionally reduce the 2.45 threshold value, in the presence of nearby cavities.

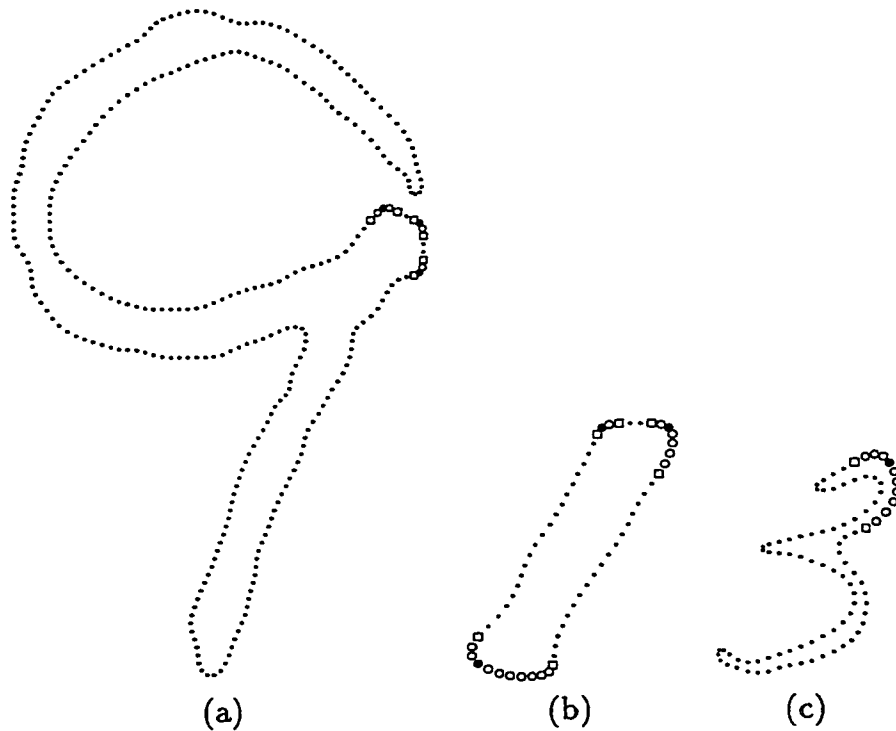


Figure 48: More Defects in “E4” End-region Extraction

- Occasionally, what humans perceive as an end-region may be split into several sub-regions. This third difficulty can be related to unusually wide strokes as in Figure 47(d) but not always, as in Figure 48(a). To remedy such situations, we must consider pairs and even triplets of nearby arcs and verify whether, taken together, they satisfy the end-region criteria.
- The fourth difficulty consists in endpoint regions not being detected as such because they miss the width criterion. The bottom region of the sample in Figure 48(b) is one such case¹³. The solution cannot simply be a relaxation of this criterion (i.e. permitting a larger fraction of the maximum dimension of the image as acceptable width) since it already allows certain regions to qualify as endpoint where it should not (see last difficulty).
- Figure 48(b) also exemplifies a fifth problem. We see that the focal point within the bottom region is incorrectly located. In Figure 47(a), a similar problem was caused by poor region extraction, but the same explanation cannot be invoked here. Rather it is the rule for choosing focal points within arcs which needs reviewing.
- A sixth difficulty is the detection of end-regions which would be better left as ‘Bend’s. One such ‘false’ end-region is highlighted in Figure 48(c). This problem also calls for the revision of the width measure and the width criterion.

The problems discussed above are the main ones encountered in the extraction of end-regions. At this point, one might wonder why we should insist on solving these problems from the contour representation of the numeral samples. Why not simply rely on skeletons for endpoint extraction? To answer briefly, let us first recall that for the great majority of samples, none of these problems exist. Also, our work indicates that it is possible to deal with most difficulties in an efficient and logical manner. Furthermore, skeletonization is a time-consuming process which has its own shortcomings, even for endpoint detection, such as the creation of spurious branches and endpoint erosion, especially for samples with wide strokes.

¹³ The feature extraction algorithm of “E4” initially found no endpoints at all for this sample. The second attempt was successful in merging the 2 top ‘Bend’s into an endpoint region.

6.3.3 Problems with Extraction of Cavities and Bends

We now briefly present some of the difficulties encountered by the E4 feature extractor in dealing with curvature regions other than endpoints. These problems are more frequent at higher resolutions (400 PPI), but they may also arise at lower resolutions.

- The first problem is exemplified in Figure 49 which displays the outcome of feature extraction for 3 samples. Here we see that significant and extended curvature regions can be left undetected: in Figure 49(a), most of the large convex region at the top of the ‘9’ is missed; in Figure 49(b), the bottom cavity on the left profile is totally missed; and in the largest piece of Figure 49(c), an important ‘Bend’ on the left profile and its associated ‘Cavity’ region on the right profile are both undetected.

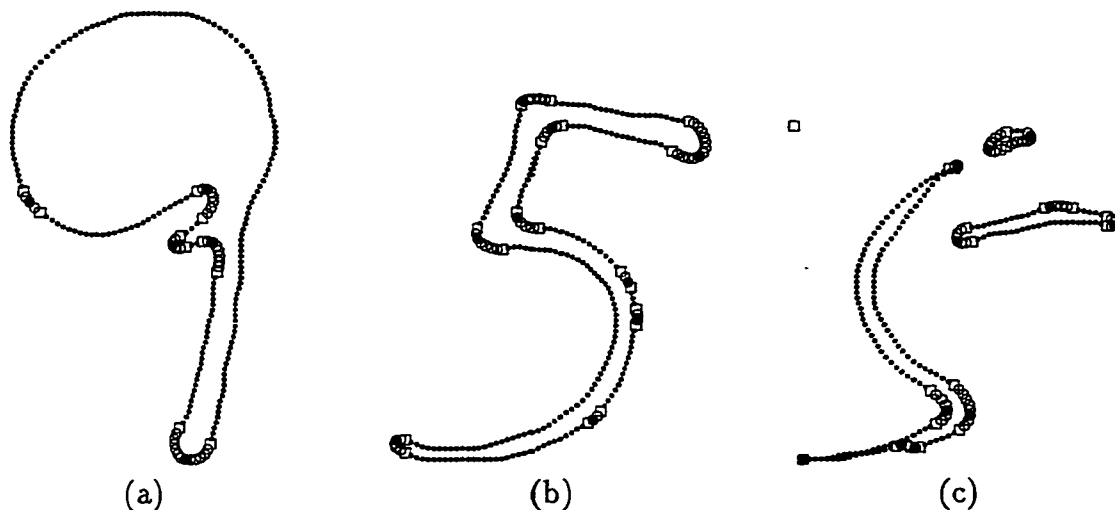


Figure 49: Large Curvature Regions Left Undetected

One avenue to solve this problem would be to keep track of the cumulative deviation angle within *inter-arc regions* (i.e. between the last point of an arc and the first point of the following arc) and to develop new rules to create additional features within inter-arc regions with large cumulative deviation angles.

- Figure 50(a) is an example of a second difficulty. Here the 2 cavities normally present and typical of the narrower middle section of an ‘8’ are not extracted.

These curvature regions are not as extensive as the ones illustrated previously and their cumulative deviation angles are not as important. However, when they go undetected, the size of inter-arc region which incorporates them is quite large, relative to the maximum dimension of the image. One possible solution could then be to relax the requirement $|\Phi^*| \geq 0.5$ (see Section 6.1.2) in the presence of ‘large’ inter-arc regions.

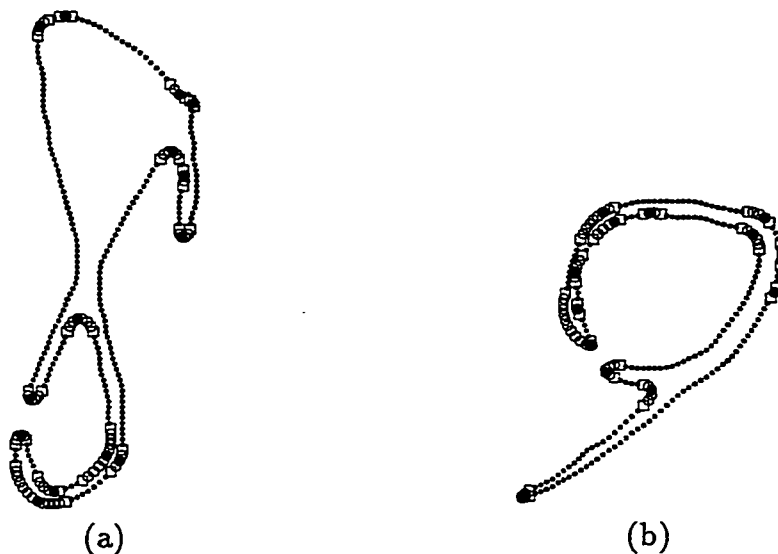


Figure 50: Missed or Fragmented Feature Regions

- The top half of the ‘9’ in Figure 50(b) illustrates another difficulty: the break-up of what humans would perceive as single large-scale feature regions into several fragments. Thus the top convex region is not detected as a single large ‘Bend’ feature but as 4 consecutive ‘Bend’ fragments; and the associated large top ‘Cavity’ is split into 5 much smaller cavities¹⁴.

The reason for this fragmentation is the second condition imposed (on the deviation angles) for consecutive contour points to belong to the same arc (see Section 6.1.2):

$$\phi_j \times \phi_{j+1} \geq 0.005, \text{ for } j \in [k, k + L - 1].$$

¹⁴ In addition, a significant portion of that large concave region is also missed.

Thus any very local straightening ($\phi_{j+1} = 0$) or wiggle in the opposite direction ($\phi_j \times \phi_{j+1} < 0.0$) causes the ‘accumulation’ of contour points within an arc region to stop. Due to the noisy nature of binary contours, this is likely to occur frequently in large concave or convex regions, splitting them into several arcs. In the worst cases, as in Figure 49, none of these fragments makes the 0.5 threshold, causing the loss of an entire global feature.

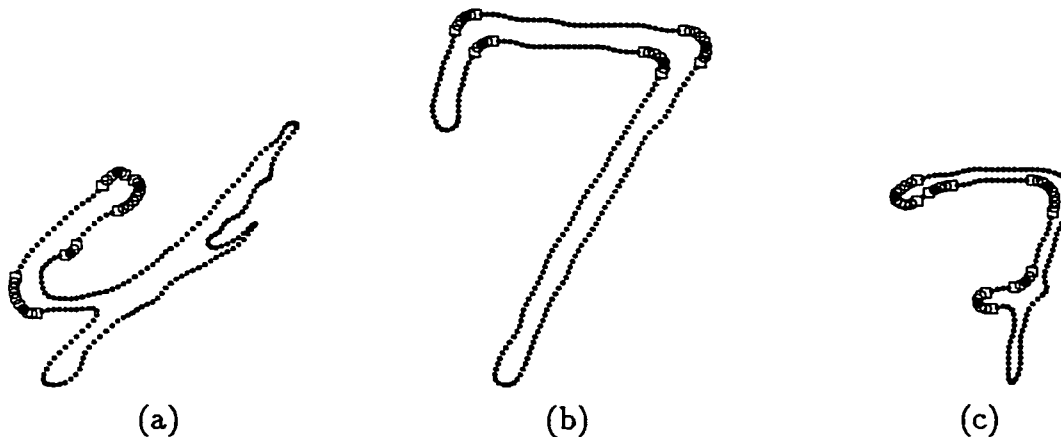


Figure 51: Merging Consecutive Bends or Cavities

To solve this fragmentation problem, rules must be established to determine when the extracted arcs should be merged into a larger curvature feature and how to accomplish the merge. Blindly merging all consecutive Bends or Cavities is not the answer. In the E4 feature extraction method, these issues were not really addressed. As a general rule, sequences of Bends were merged into a single Bend but the same was not performed for Cavities. For Bends this approach was suitable in most circumstances, but not all. Figure 51(a)¹⁵ provides an example of where such blind merging can be counter-productive: what normally constitutes the top left endpoint of a ‘4’ was instead detected as two distinct nearby Bends which were then merged with the preceding and following Bend regions, making up a huge and misleading feature region. Because of this blind merging, the E4 feature extractor had to keep track of the fine structure of merged Bends to untangle such situations. This is obviously not satisfactory

¹⁵ In Figure 51, only the features of interest are shown.

and a more sophisticated and sensible approach should be devised.

Figure 51(b) displays pairs of consecutive Bends and Cavities which should *not* be merged since their presence in these positions and with these directions is typical of this type of '7'. Figure 51(c) shows three consecutive cavities which again should not be merged; the first one, very close to the top left endpoint should probably be dropped while the other two should be kept as distinct features: the middle cavity is present in almost all '7's while the last one is caused by the crossing out of the '7', a much less frequent occurrence.

- Figures 52 (a) and (b)¹⁶ are examples of another difficulty: the extraction of small, parasitic, curvature features in the vicinity of an endpoint. In the first case, the useless small feature is a Cavity; in the second, a Bend. Since these features will in no way facilitate recognition (quite the contrary), we could probably make use of the proximity of the extracted endpoint to either drop the small feature or merge it with the endpoint region under certain conditions.

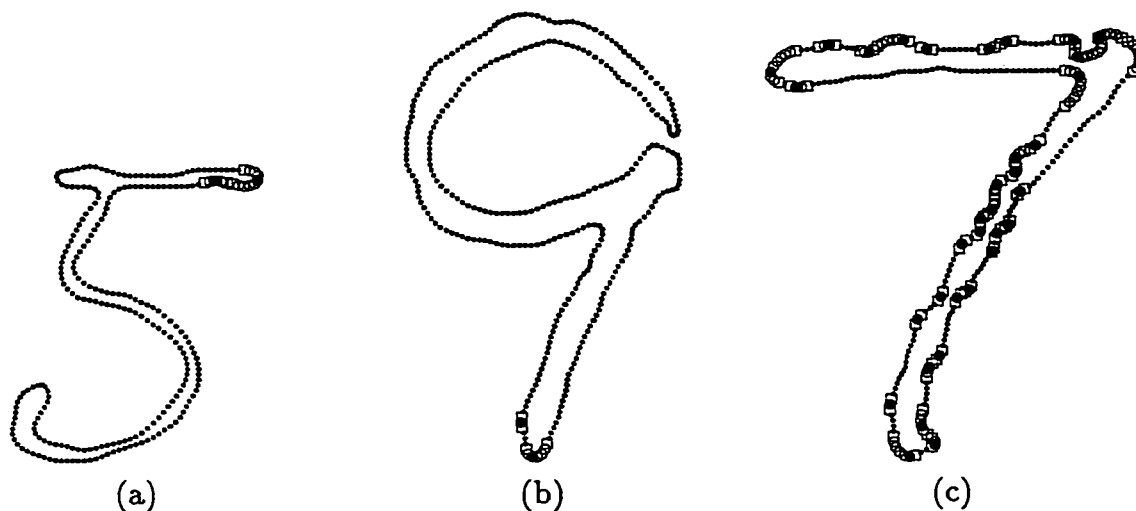


Figure 52: Problematic Small Features

- Figure 52(c) shows a sample with a very noisy border from which no less than 35 'significant' arc regions are extracted! This problem may have been caused by

¹⁶ Again, in these figures, only the features of interest are shown.

thinner ink and/or porous paper. While infrequent, it is not a unique situation. It might be possible to take advantage of the regularity of the ‘wiggles’ along the edges of the stroke to remove most of this noise.

6.4 The New Feature Extractor

In this section, we will present the different aspects of the new feature extractor. The basic approach remains the same as that of the E4 feature extractor, namely a bottom-up approach, where the processing begins at the pixel level by computing deviation angles along the contours of numerals. Some specific elements are also kept in the new scheme: contour smoothing, the same definition of an *arc* and of its Φ^* -attribute; the same thresholds for keeping an arc as a significant feature (0.5) and for submitting it to end-region testing (2.25).

However, the new feature extractor is much more complex and sophisticated¹⁷ at capturing the significant curvature features along the contours of numerals, at capturing them in their entirety rather than in fragments, and at getting rid of several small and meaningless features which the old system was extracting. Because of this complexity, we cannot give the full details of the system in this section. The final reference in this respect is the C program itself.

Of course, when we proceed upwards from the pixel level, we cannot claim that there necessarily exists a set of geometric, topological, or morphological relations which inherently define what will constitute a significant shape feature at a much higher level. It has long been known that corners or regions of high curvature are perceptually striking to human beings (Attneave [11]). But one cannot infer from this knowledge that any region which is small or not highly curved is necessarily without significance for any recognition task. In handwriting for example, numerals traced by some writers differ significantly from their idealized models. Thus a feature which normally should be striking at a certain position could be attenuated very much or even absent. To a human reader, the presence of even a small feature in that position can then be quite significant, given the high-level knowledge that something

¹⁷ The file which contains the code for this part of the system is 4446 lines long.

is expected there.

Because of this, even if we want to get rid of as much small meaningless features as possible, we must proceed with caution: it is preferable to keep some meaningless features initially rather than risk removing too many small features, including some significant ones. This is especially important given our goal of a highly reliable system. With this last goal, it is better to confirm in several ways the identity of a numeral, even through –at times– small but significant features. And if recognition fails because the extractor has retained useless features, the system can then try to filter them out and attempt classification once more.

We now provide insight into the various aspects of the new feature extractor. In very broad terms, we can say that the process consists, for each of the external contours making up the numeral, of the following steps:

- Computation of the deviation angles at each contour point;
- Extracting the features from the contour:
 - Extracting arc regions
 - Finding the focal point of an arc
 - Processing arcs to extract more global features:
 - * Merging nearby convex (Bend) arcs
 - * Testing for endpoints
 - * Splitting very large convex arcs into Bend-End or End-Bend feature pairs
 - * From arcs to Bend or Cavity features
- Conditionally merging consecutive cavities and bends
- Obtaining the direction of every feature

6.4.1 The Computation of Deviation Angles

In the E4 system, before calculating the deviation angles, the contour point coordinates were smoothed twice based on the formula

$$\mathbf{p}_i^{(k)} = \sum_{j=-1}^1 \frac{1}{3} \mathbf{p}_{i+j}^{(k-1)}, \quad k = 1, 2. \quad (39)$$

Equivalently, as described in the introduction to Chapter 5, we could have used a single pass of a triangular filter with window size $w = 5$ and weights equal to $(\frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{2}{9}, \frac{1}{9})$.

In our new recognition system, smoothing is not performed explicitly but rather implicitly in the computation of the deviation angles ϕ_i . Furthermore, varying amounts of smoothing are applied depending on image size/resolution, and the size of the image fragments under consideration. More precisely, for tiny blobs having fewer than 7 contour points, the ϕ_i 's are computed without any smoothing. Otherwise, the window size of the smoothing operation depends on the number of rows n_rows in the image: $w = 5$ is used for $n_rows < 45$; $w = 7$ is used for $45 \leq n_rows < 90$; and $w = 9$ is used for $n_rows \geq 90$. When the window size is determined to be 7 or 9, based on n_rows , it can still be reduced to $w = 5$ in the case of very small blobs¹⁸ to avoid oversmoothing.

We have decided not to use the optimum filters as determined in the preceding chapter. Instead, *triangular* filters are used for each window size (5, 7, and 9). The first reason is that we do not have the optimum filters for deviation angle measurements for window sizes 7 and 9. The second reason is that triangular filters allow special computer efficiency for the computation of deviation angles; we will discuss this below. Finally, we note that triangular filters, even though not optimal, offer excellent performance for removing noise from binary contours. In Section 5.4.3, it was shown that for digital circles of various radii the triangular filter using $w = 5$ with weights $(\frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{2}{9}, \frac{1}{9})$ provided a mean noise reduction level of 88.32% compared to 90.69% for the empirically determined best fixed-weights method, using $\alpha_{\pm 1} = 0.2386$ and $\alpha_{\pm 2} = 0.1190$.

¹⁸ The exact criterion used to determine 'very small blobs' is when the number of contour points is smaller than twice the window size (14 or 18).

We now explain how the deviation angles are obtained. At first, let us assume that contour smoothing is performed explicitly. Using triangular filters, the smoothed coordinates are first obtained as follows:

$$\mathbf{p}'_i = (x'_i, y'_i) = \sum_{j=-n}^n \frac{n+1-|j|}{(n+1)^2} (x_{i+j}, y_{i+j}) \quad (40)$$

The value of n is simply 2, 3, or 4 corresponding to window sizes 5, 7, or 9 respectively. Defining $dx'_i \equiv x'_i - x'_{i-1}$ and $dy'_i \equiv y'_i - y'_{i-1}$, the deviation angle ϕ_i is then computed as:

$$\phi_i = \tan^{-1} \left(\frac{dx'_i dy'_{i+1} - dx'_{i+1} dy'_i}{dx'_i dx'_{i+1} + dy'_i dy'_{i+1}} \right). \quad (41)$$

Reducing Computation Costs

The value of ϕ_i is needed for every contour point of every numeral image. Thus any significant savings in its computation is of special interest. From the last 2 equations, we first note that each of the 4 products involved will contain a factor $1/(n+1)^4$ which will then cancel out; thus we could simply drop the $1/(n+1)^2$ factor in computing Equation 40. But actually, it is not necessary at all to first derive the smoothed coordinates. The last equation indicates that only the values of the dx 's and the dy 's are needed to calculate the ϕ_i 's. In Appendix E, we show that for our triangular filters the (dx'_i, dy'_i) 's can be obtained recursively, at a very low and fixed cost *regardless of the window size used*:

$$(dx'_{i+1}, dy'_{i+1}) = (dx'_i, dy'_i) + \frac{1}{(n+1)^2} ((x_{i-n-1}, y_{i-n-1}) - 2(x_i, y_i) + (x_{i+n+1}, y_{i+n+1})). \quad (42)$$

For a window size of 5, this technique for computing the ϕ_i 's represents approximately a 50% reduction in computation cost compared to first getting the smoothed coordinates and then obtaining the (dx'_i, dy'_i) 's and ϕ_i 's. For window sizes of 7 and 9, the savings are even more important. The disadvantage is that we no longer have the smoothed coordinates for every contour point. But contour point coordinates or distances between contour points are needed only sporadically in feature extraction and classification. Thus our new system will explicitly compute smoothed coordinates only for those few contour points where it will be of interest.

6.4.2 The Extraction of Arc Regions

The basic definition of an *arc* adopted in the E4 system (see Section 6.1.2) is kept for the new system. The 2 conditions presented at the beginning of Section 6.1.2 remain valid with the very minor change of replacing the ' \geq ' signs with '>'. However, to correct defects identified previously, additional arcs are extracted from certain inter-arc regions; and consecutive arcs can also be merged into a single arc.

The extraction of arc regions from an external contour begins with the search for a starting point at which we can safely begin looking for the first arc¹⁹. This is the first point i at which $\phi_{i-1}\phi_i \leq 0.005$. From this safe starting point, the system searches for the first significant arc according to the old rule #1 (i.e. $|\Phi^*| \geq 0.5$). The degenerate situations where no safe starting point or no first arc can be extracted are handled as per rule #4 of the old system (again see beginning of Section 6.1.2).

Other significant arcs are extracted according to old rule #1. However, under certain conditions, each pair of newly extracted consecutive arcs meeting the $|\Phi^*| \geq 0.5$ criterion may undergo further testing; this testing may result in the creation of an additional significant arc in the inter-arc region of the pair. Let Φ_{inter}^* be the cumulative deviation angle of the inter-arc region²⁰, *point_gap* be the number of contour points within the inter-arc region, and *max_dim* be the maximum dimension of the image. The conditions for further testing are as follows:

```

if  $|\Phi_{inter}^*| \geq 0.9$ 
  GET_ARC_FROM_INTER_ARC
else if ( $|\Phi_{inter}^*| \geq 0.6$ ) and (point_gap < 0.55 max_dim)
  GET_ARC_FROM_INTER_ARC
else if point_gap  $\geq 0.55$  max_dim
{
  SCRUTINIZE_INTER_ARC
  if ("No new arc from SCRUTINIZE") and ( $|\Phi_{inter}^*| \geq 0.75$ )
    GET_ARC_FROM_INTER_ARC
}

```

¹⁹ We cannot assume that the first point generated by the contour extraction scheme is such a safe starting point, since it could happen to be right in the middle of a significant feature region.

²⁰ Φ_{inter}^* is the sum of the ϕ_i from the last point of the first significant arc to the first point of the following significant arc.

The function *GET_ARC_FROM_INTER_ARC* is responsible for extracting 22 additional arcs from 19 samples in file *train_us*, 52 additional arcs from 37 samples in file *train_q5000*, and 77 additional arcs from 47 samples in file *train_tw*. The more sophisticated *SCRUTINIZE_INTER_ARC* is called upon almost as frequently but is more selective at creating new arcs: with file *train_us*, it is called for 84 samples but creates an additional arc in only 14 cases; with file *train_q5000*, it is called for 59 samples but creates only 9 new arcs; finally, with file *train_tw*, the numbers are 99 and 11 respectively.

The Function GET_ARC_FROM_INTER_ARC

In general, this function is called when the absolute cumulative deviation angle Φ_{inter}^* of an inter-arc region is too large. The goal of the function is to determine where this curvature is mostly concentrated and to create an additional arc in that portion of the inter-arc region. More specifically, we have the following major steps:

- Determine the deviation angle *tolerance*
 - if $|\Phi_{inter}^*| < 0.9$

$$\text{tolerance} = 0.125 \times \Phi_{inter}^*$$
 - else if $\Phi_{inter}^* > 0$

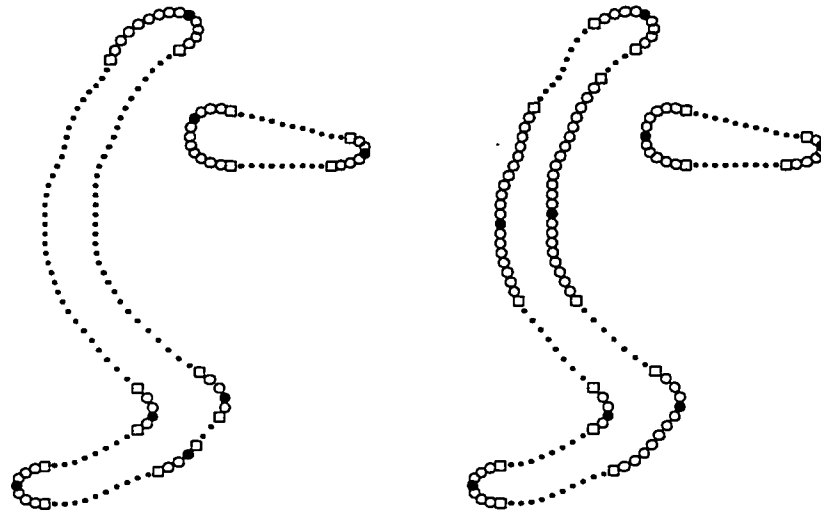
$$\text{tolerance} = \min(0.5, 0.2 \times \Phi_{inter}^*)$$
 - else

$$\text{tolerance} = \max(-0.5, 0.2 \times \Phi_{inter}^*)$$
- From both ends of the inter-arc region move towards the middle, skipping contour points as long as the cumulated deviation angle of skipped points does not exceed the tolerance. The function also uses a look-ahead approach to determine if the cumulated deviation angle has only temporarily exceeded the tolerance and will again fall below the tolerance at some further point. If this is the case, it keeps skipping contour points. We define *inner_point_gap* as the number of contour points within the inner portion of the inter-arc region so determined and Φ_{inner}^* as the cumulated deviation angle for the inner portion only; we also

define $CURV = \frac{|\Phi_{inter}^*|}{(point_gap/max_dim)}$. Finally we determine the focus (see Section 6.4.3) of the inner portion and compute Φ_{focal}^* the sum of the deviation angles at the focal point, at the preceding and at the following points.

- Exit without creating a new arc
 $if ((CURV + |\Phi_{inter}^*|) < 2.85) \text{ and } (\frac{inner_point_gap}{point_gap} > 0.75)$ OR
 $if ((CURV + |\Phi_{inter}^*|) < 2.50) \text{ and } (\frac{inner_point_gap}{point_gap} > 0.65)$
- Exit without creating a new arc
 $if (\Phi_{focal}^*/\Phi_{inner}^* < 0.125) \text{ and } (|\Phi_{inter}^*| < 0.8)$ OR
 $if (\Phi_{focal}^*/\Phi_{inner}^* < 0.25) \text{ and } (|\Phi_{inter}^*| < 1.0) \text{ and } (CURV < 3.0)$
- Otherwise, create a new arc spanning the inner portion of the inter-arc region determined above.

Figure 53 shows an example of features missed by the old feature extractor but now captured with the help of the GET_ARC_FROM_INTER_ARC function.



(a) E4 Feature Extraction (b) New Feature Extraction

Figure 53: New Bend and Cavity Captured by GET_ARC_FROM_INTER_ARC

The Function SCRUTINIZE_INTER_ARC

In general, this function is called when the absolute value of Φ_{inter}^* is less than 0.9 but the point length of the inter-arc region is larger than 55% of max_dim . Here a more sophisticated approach is applied to determine if an arc is present and, if so, its exact location. This method relies on the extraction of *subsequences* from the inter-arc region. A subsequence is a sequence of contour points with same-sign and non-zero deviation angles. More specifically, we have the following major steps:

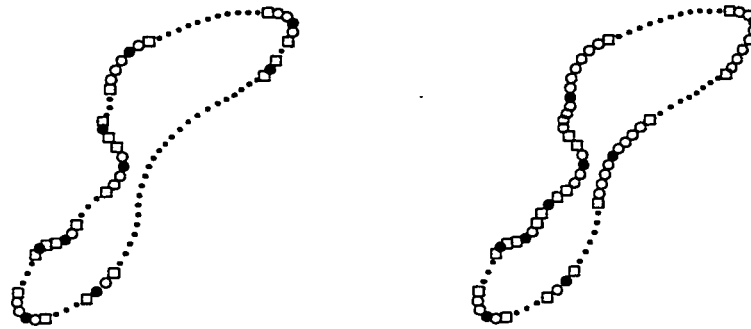
- Subsequences with absolute cumulative deviation angles $|\Phi_{sub}^*|$ not less than 0.25 are extracted from the inter-arc region.
- Consecutive subsequences with same-sign Φ_{sub}^* are merged together if their contour point separation is less than 10% of max_dim .
- Subsequences with maximum cumulative deviation angle Φ_{submax}^* and minimum cumulative deviation angle Φ_{submin}^* are obtained.
- If $\Phi_{submin}^* \leq -0.6$
 Create new arc in that subsequence
 else if $\Phi_{submax}^* \geq 0.6$
 Create new arc in that subsequence
 else if $\Phi_{submin}^* \leq -0.325$
 EXAMINE_FURTHER (might create a new arc or not)
 If *NoNewArcCreated* and ($\Phi_{submax}^* \geq 0.4$)
 EXAMINE_FURTHER (might create a new arc or not)
- EXAMINE_FURTHER:
 - If the subsequence is at the very end of the inter-arc region (no more than $0.075 \times max_dim$ contour points away from the preceding or following arc)
 AND if Φ_{sub}^* ²¹ is of sign opposite to the Φ^* of that arc
 Discard the subsequence

²¹ Within EXAMINE_FURTHER, Φ_{sub}^* stands for Φ_{submin}^* or for Φ_{submax}^* , whichever is being examined.

- If possible, expand the subsequence at both ends, incorporating contour points with $\phi_j = 0$ and following ones as long as no ϕ_j of sign opposite to that of Φ_{sub}^* is encountered. Let the cumulative deviation angle for the expanded subsequence be denoted Φ_{sub+}^*
- If $|\Phi_{sub+}^*| \geq 0.6$
Create new arc in expanded subsequence
- Otherwise, the method takes into account the point length of the *arms* on both sides of the subsequence to evaluate the significance of that sub-arc²².
 - * At each end of the subsequence, the *arm* is that portion of the contour in which EITHER there is no subset of consecutive points with absolute cumulative deviation angle larger than 0.10 and sign opposite to that of Φ_{sub}^* ; OR such a ‘wrong-sign’ subset may be included within the *arm* if it reaches no further than $10\% \times max_dim$ contour points away from the subsequence and is immediately countered by an at least equally important subset of contour points with cumulative deviation angle of the same sign as Φ_{sub}^* . Let arm_1 and arm_2 be respectively the point lengths of the arm preceding and the arm following the subsequence; and let $\Phi_{sub+arms}^*$ be the cumulative deviation angle of the contour portion spanned by the subsequence and its two arms.
 - * If $|\Phi_{sub+arms}^*| + \frac{arm_1+arm_2}{max_dim} \geq 0.75$
Create new arc in subsequence.

Figure 54 shows an example of a feature (middle right cavity of an ‘8’) missed by the old feature extractor but now captured with the help of the SCRUTINIZE_INTER_ARC function.

²² The importance of this factor was also taken into account in the ‘cornerity measure’ of Freeman & Davis [53].



(a) E4 Feature Extraction (b) New Feature Extraction

Figure 54: New Cavity Captured by SCRUTINIZE_INTER_ARC

6.4.3 Finding the Focal Point of an Arc

The method followed in the E4 recognition system to select a focal point for each arc region was presented in Section 6.1.2. In our new system, a different approach is used to correct problems discussed in Section 6.3.2.

First, a cumulative deviation angle threshold Φ_{thresh}^* is computed as a fraction of Φ^* ; the fraction is chosen as 0.33 if $\Phi^* \geq 2.25$ and 0.4 otherwise. Using s and f to denote the indices of the starting and final points of the arc, let k_s be the minimum k such that $|\sum_{j=s}^k \phi(j)| \geq \Phi_{thresh}^*$; furthermore, let k_f be the maximum k such that $|\sum_{j=k}^f \phi(j)| \geq \Phi_{thresh}^*$. The focal point is chosen as the middle of the sequence $[k_s, k_f]$. This criterion provides better focal point location for the sample of Figure 48(b) and many others. In particular, in the case of composite end-regions as in Figure 47(d), it selects intuitively correct focal points. Illustrations of the various improvements provided by the new feature extractor are given in Section 6.4.11.

6.4.4 From Arcs to More Global Features

In our new system, consecutive arcs with same-sign Φ^* may be merged under certain circumstances; thus an arc is not stored as a new entry in the feature list unless it appears that the associated feature region has been completely reconstructed and does not extend any further. Note that a merge count \mathcal{C} keeps track of the number of

original arcs making up a merged arc²³. The processing of significant arcs to obtain *features* considers various situations involving triplets of consecutive significant arcs labeled arc_1 , arc_2 , and arc_3 .

In general, the most recently extracted arc is in the arc_3 role i.e. the last of the considered triplet. When all tests have been applied to the triplet and depending on the results, the arcs trickle down one position in the triplet, with a newly extracted arc becoming the new arc_3 . In our implementation, when an arc is in the arc_1 role it is already established that it is *not* an endpoint feature and possible merges with *preceding* arcs have already been investigated. Of course, we cannot be certain that the very first arc extracted could not have combined with preceding arcs; verifying this possibility is delayed until the feature extraction process is completed. In our implementation also, endpoint testing is applied to an arc when it is in the arc_2 role.

We now describe the major steps involved in testing every triplet of arcs. To this end, we introduce the following notation. Let Φ_1^* , Φ_2^* and Φ_3^* be the cumulative deviation angles of arc_1 , arc_2 , and arc_3 respectively; and let C_1 , C_2 , and C_3 be their respective merge counts. Furthermore, let gap_{1-2} be the point distance between the last point of arc_1 and the first point of arc_2 and let $\Phi_{inter1-2}^*$ be the cumulative deviation angle between these points; and finally, let gap_{2-3} and $\Phi_{inter2-3}^*$ be the equivalent quantities defined between arc_2 and arc_3 .

- If $(\Phi_2^* > 2.25)$ and $(C_2 \leq 3)$
CHECK_END (See Section 6.4.5)
If successful, EXIT
- If arc_1 and arc_2 clearly do not belong to the same feature $(\Phi_1^* \times \Phi_2^* < 0)$, store arc_1 in the feature list as a feature on its own.
- If arc_2 and arc_3 and/or arc_1 are nearby Bend's, their possible merging is investigated. One objective of this step is to reconstruct wide endpoint regions which might have been extracted as 2 or even 3 nearby Bend's. The first attempt is at merging arc_2 and arc_3 ; this is done if both of them together are very likely

²³ The merge count C is set to 1 when an arc is initially extracted.

to constitute an endpoint region *and* if arc_3 on its own does not appear to be an endpoint region. If the 2 arcs are merged, the resulting arc_2 is tested for an endpoint. If the endpoint testing fails or if arc_2 and arc_3 were not merged initially, an attempt is made at merging arc_1 and arc_2 ; if this merge is successful, the resulting arc_2 will also be tested for an endpoint. More precisely, we have:

- If $(\Phi_2^* + \Phi_3^* + \Phi_{inter2-3}^* > 2.55)$ and $(gap_{2-3} \leq 0.10 \times max_dim)$ and $(\Phi_{inter2-3}^* > -0.5)$ and $(\Phi_3^* < 2.35)$
 - Merge arc_2 and arc_3 into arc_2 and extract a new arc_3 ;
 - If $C_2 \leq 3$
 - CHECK_END
 - If successful, EXIT
- If arc_2 and arc_3 were not merged OR if they were merged but CHECK_END failed to produce an endpoint, consider merging arc_1 and arc_2
 - * Avoid merging them if either arc_1 is a small Bend immediately preceded by a small ‘counter-balancing’ cavity or if arc_2 is a small Bend immediately followed by a small ‘counter-balancing’ cavity; such a ‘wobble-pair’ is marked and most likely filtered out together at a later stage. In this case, EXIT
 - * If $(gap_{1-2} \leq 0.10 \times max_dim)$ and $(\Phi_{inter1-2}^* > -0.5)$
 - Merge arc_1 and arc_2 into a new arc_2 and mark arc_1 as now vacant;
 - If $(\Phi_2^* \geq 2.55)$ and $(C_2 \leq 3)$
 - CHECK_END
 - If successful, EXIT
- If arc_1 has not been marked vacant, it could not be combined with other arcs, therefore store it into the feature list as a feature on its own.

When all arcs have been extracted and all arc triplets have been processed in succession as outlined above, the system examines the last inter-arc region which lies between the final contour point of the last feature and the initial contour point of the first feature. If conditions are met, functions GET_ARC_FROM_INTER_ARC and

SCRUTINIZE_INTER_ARC might create a final arc to be processed in that inter-arc region.

Finally, a function called SKIP_B_C_NEAR_E is called to detect and skip small bends or cavities which are very close to endpoints. This affects 6% of samples in file *train_us*, 24% of samples in file *train_tw*, and 13% of samples in file *train_q5000*.

Omitting the technical details, the criteria used are as follows:

- the feature near the endpoint must be small enough;
- the distance between the small feature and the endpoint must be small enough;
- the Φ^* -value for the small feature must be low;
- the first feature preceding that feature pair must be far enough;
- the first feature following that feature pair must be far enough;
- the combined Φ^* -value of the endpoint and small nearby feature must be within

acceptable bounds.

In some cases, the small skipped feature is taken into account in the recomputation of the endpoint's direction.

6.4.5 From Arc to Endpoint (Function CHECK_END)

In general, this function is called to test arc_2 on its own when its cumulative deviation angle Φ_2^* is larger than 2.25^{24} and its merge count C_2 does not exceed 3. We now outline the major steps involved:

- If $\Phi_2^* < 2.55$ and ϕ -values are positive at the contour points immediately preceding and/or immediately following the arc_2 region
 Extend arc_2 to include one or both of these points
- If $\Phi_2^* < 2.35$
 EXIT

²⁴ A more restrictive requirement of $\Phi_2^* > 2.55$ is occasionally imposed before calling CHECK_END when successive Bends have been merged.

- If $\Phi_2^* < 3.9^{25}$
 - If $((C_2 = 1) \text{ and } (\Phi_2^* > \pi)) \text{ OR } ((C_2 > 1) \text{ and } (\Phi_2^* > 3.9))$
 - * If focal point of arc_2 is asymmetrically located within the arc, shorten arc_2 at one extremity to prevent possible failure of the VERIFY_WIDTH test; if endpoint testing fails nevertheless, the original extent of the arc and Φ_2^* -value will be restored.
 - If the contour point length of arc_2 is larger than $0.55 \times max_dim$
EXIT
 - VERIFY_WIDTH (See Section 6.4.6)
 - If width test was successful
 - If $\Phi_2^* \geq 2.55$
 - Store arc_1 in the feature list
 - Store arc_2 as an endpoint in the feature list
 - Otherwise
 - test for CONDITIONAL_ENDPOINT (see Section 6.4.7)
 - If successful
 - Store arc_1 in the feature list
 - Store arc_2 as an endpoint in the feature list
- Otherwise
 - If $(\Phi_2^* > 5.25) \text{ and } (arc_2 \text{ spans more than half the external contour being processed})$
EXIT
 - Check if arc_2 is an End-Bend or Bend-End composite arc (see Section 6.4.8)

In concluding, we note that as with the E4 system, the new feature extractor tests an arc region for endpoint detection when $\Phi^* > 2.25$. However, the final requirement

²⁵ Occasionally, larger values are tolerated when resulting from purposely merged Bends, as the system is then clearly not dealing with an End-Bend composite region extracted as a simple arc.

$\Phi_2^* > 2.55$ is more stringent than that of the earlier method; only when a nearby cavity is present is this requirement relaxed to $\Phi_2^* > 2.35$. Also, it appears that arcs with Φ_2^* values anywhere between 2.35 and 3.90 may qualify as endpoints. The extreme values of this very broad interval were chosen experimentally but they are not arbitrary; significantly, their median value is 3.125, which is very close to π .

6.4.6 Endpoint Width Criterion

The problem of finding two contour points to estimate the width across the potential endpoint stroke is treated differently than in the E4 approach. Instead of using a *fixed* value of 5 for the maximum displacement before and after the focal point of the arc, a fraction of SW , the estimated stroke width (see Section 4.4.1), is used. More exactly, let $D_E = \max(4, 0.8 \times SW)$. The new method proceeds as follows:

- If the first and/or the last point of arc_2 are less than D_E contour points away from the focal point of arc_2 , they are moved further until either that point distance is reached or a contour point with negative ϕ_j is encountered
- The contour point extremity of arc_2 which is closest to the focal point is moved further away until both extremities of arc_2 are equally distant from the focal point
- The relocated contour point extremities of arc_2 are smoothed using a triangular filter with window size $w = 5$ and weights equal to $(\frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{2}{9}, \frac{1}{9})$
- The *width* estimate is taken as the Euclidian distance between the smoothed points, *plus 1* to account for pixel width. The endpoint width criterion is considered passed successfully if either

$$width \leq 1.90 \times SW \quad \text{or}$$

$$(width \leq 0.175 \times max_dim) \text{ and } (width/SW < 2.9 + 0.375 (\Phi_2^* - 2.35))$$

Note that the fraction of max_dim used in the very last test (0.175) is more restrictive than what was used in the original E4 method ($0.18 + 0.15 (\Phi^* - 2.45)$ and even 0.25 in some cases!).

6.4.7 Conditional Endpoint Testing

The `CONDITIONAL.ENDPOINT` function is called when arc_2 has successfully passed the width criterion and $2.35 \leq \Phi_2^* < 2.55$. The objective is to verify the presence of a nearby cavity which could account for the somewhat lower Φ_2^* -value. The function checks if arc_1 is a cavity and, if so, if gap_{1-2} and $\Phi_{inter1-2}^*$ satisfy one of these 2 conditions:

Either $gap_{1-2} < 0.1 \times max_dim$

or $(gap_{1-2} < 0.2 \times max_dim)$ and $(\Phi_{inter1-2}^* < -0.2)$

If not, it verifies if arc_3 is a cavity with gap_{2-3} and $\Phi_{inter2-3}^*$ satisfying similar conditions.

Notwithstanding the presence of a nearby cavity, the feature extractor will avoid creating a conditional endpoint if a closer Bend-arc is found on the other side of arc_2 which appears to combine well with arc_2 to create an even better endpoint.

6.4.8 Testing for End-Bend Composite Arc

We now summarize the approach developed to solve problems illustrated in Figures 47 (a) and (b) where a single extracted arc incorporates 2 features, namely an endpoint and an immediately adjacent Bend. All arcs with $\Phi_2^* \geq 3.90$ are tested for this possibility, except if such large Φ_2^* was the result of the feature extractor itself merging nearby arcs to reconstruct an endpoint region. In what follows, let s and f be respectively the starting and final contour points of the tested arc_2 .

- If $\Phi_2^* > 5.25$ and $[s, f]$ encompasses more than half the contour being processed
EXIT
- Let $\Phi_{bend}^* = \Phi_2^* - \pi$ be an estimate of the cumulative deviation angle for the portion of arc_2 which should *not* be included within the endpoint region (if there is one). As was done in Section 6.4.3 to find the focal point of an arc, the system proceeds to accumulate this angle from both extremities, s and f , of arc_2 . More precisely, let k_s be the minimum k such that $\sum_{j=s}^k \phi(j) \geq \Phi_{bend}^*$; furthermore, let k_f be the maximum k such that $\sum_{j=k}^f \phi(j) \geq \Phi_{bend}^*$. The longest of the 2

sequences $[s, k_s]$ or $[k_f, f]$ will become the Bend and the rest of arc_2 will be tested as a potential endpoint.

- If the longest sequence is $[k_f, f]$ (see for example Figure 55 (a))
 - Φ_{bend}^* is re-computed to $\sum_{j=k_f}^f \phi(j)$; and the potential end region is the sequence $[s, k_f - 1]$ with associated $\Phi_{end}^* = \Phi_2^* - \Phi_{bend}^*$
 - If $\Phi_{end}^* < 2.55$, move k_f to the next contour point and re-adjust Φ_{end}^* and Φ_{bend}^* accordingly
 - If contour point length $(k_f - 1 - s) > 0.55 \times max_dim$, EXIT
 - Find focal point of $[s, k_f - 1]$
 - VERIFY_WIDTH
 - If width test was successful
 - Store arc_1 in the feature list
 - Store $[s, k_f - 1]$ in the feature list as an endpoint
 - Put leftover bend $[k_f, f]$ in arc_1 .
 - Otherwise, keep $[s, f]$ as a large Bend (in arc_2).
- Otherwise (i.e. the longest sequence is $[s, k_s]$ as in Figure 55 (b)). The approach is the same as was just presented for $[k_f, f]$, *mutatis mutandis*, the potential endpoint region now being the sequence $[k_s + 1, f]$. There is only a minor difference when the width test is successful. The steps would then be:
 - If width test was successful
 - * If arc_1 is a cavity
 - Store arc_1 and $[s, k_s]$ as cavity and bend in the feature list
 - * Otherwise
 - Verify if the bends in arc_1 and $[s, k_s]$ can be merged
 - Consequently, store a single or 2 separate Bends in feature list
 - * Store $[k_s, f]$ in the feature list as an endpoint
 - Otherwise, keep $[s, f]$ as a large Bend (in arc_2).

Figures 55 (a) and (b) provide enlargements of the bottom region of the samples in Figures 47 (a) and (b) respectively. They illustrate at what point the splitting of the composite arc into 2 arcs is attempted for potential endpoint testing.

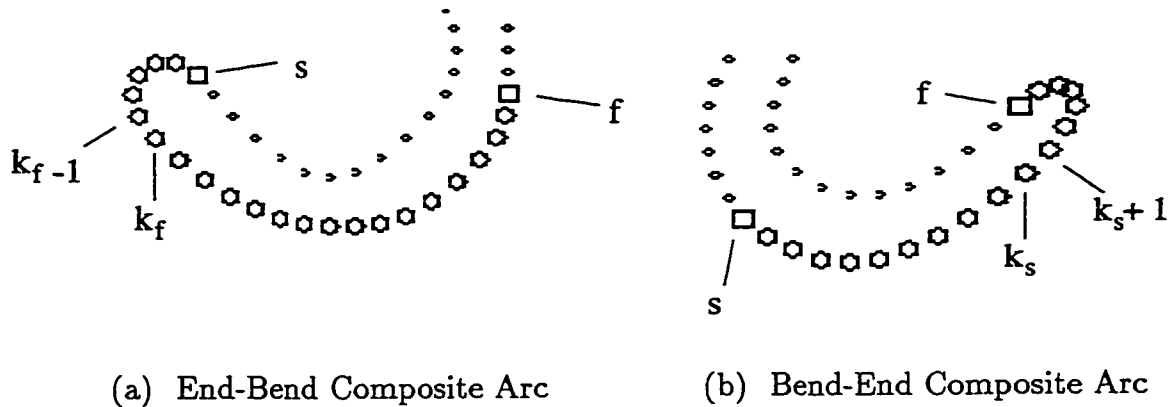


Figure 55: Processing Composite Arcs

6.4.9 Merging Consecutive Features

After the feature regions have been extracted from a given contour based on the techniques presented since the beginning of Section 6.4, the system will go over all sequences of cavity features and all sequences of bend features to determine whether some or all of them should be merged together. In our new system, this work is performed by the functions `CHECK_MERGE_CAVITIES` and `CHECK_MERGE_BENDS`. The general approach of these functions is to merge the consecutive features of the same type *unless they are deemed important enough to stand on their own*. We now summarize the major points of this processing, stressing the qualitative aspects and illustrating typical situations. For technical details, the reader is referred to the program.

The function `CHECK_MERGE_CAVITIES`

For each sequence of cavities, mostly the features are merged unless the following exclusion tests are positive:

- Avoid merging 2-cavity sequences when the Euclidian distance between focal points is relatively large ($> 0.25 \times n_rows$) and that portion of the contour is relatively straight; this occurs most frequently for 4's, 5's, and 7's. See Figure 56 (a).
- If the first or last cavity in a sequence has a small bend very close to it and if it is estimated that they together form a 'wiggle-pair', avoid merging this cavity with others in the sequence. The 2 features in a B-C or C-B wiggle-pair must have a very short inter-feature region and $|\Phi^*| < 2.0$; the point length of both features must also be very small; this last requirement can be relaxed provided some co-linearity requirements are met. The wiggle-pair is marked as such and generally filtered out at a later stage. See Figures 56 (a) and (b).
- Merge consecutive cavities when the point length of their inter-feature region is smaller than $\max(2, 0.05 \times max_dim)$.
- Avoid merging with others a cavity for which the cumulated deviation angle in a small neighbourhood around its focal point is less than -1.75 ; such a cavity is generally significant on its own. See Figure 56 (c).
- Avoid merging with others a cavity for which the cumulated deviation angle in a small neighbourhood around its focal point is less than -1.10 , $\Phi^* < -\pi/2$, and curvature is relatively high. See Figure 56 (d).
- Avoid merging with others a cavity with $\Phi^* < -0.90$ which is far enough from its two neighbouring features (one on each side) when one such neighbour is *not* a cavity. See Figure 56 (e).
- Avoid merging with the preceding (or following) cavity, a cavity for which the cumulated deviation angle in a small neighbourhood around its focal point is less than -1.0 and the distance to that preceding (or following) cavity is large enough. See Figure 56 (f).

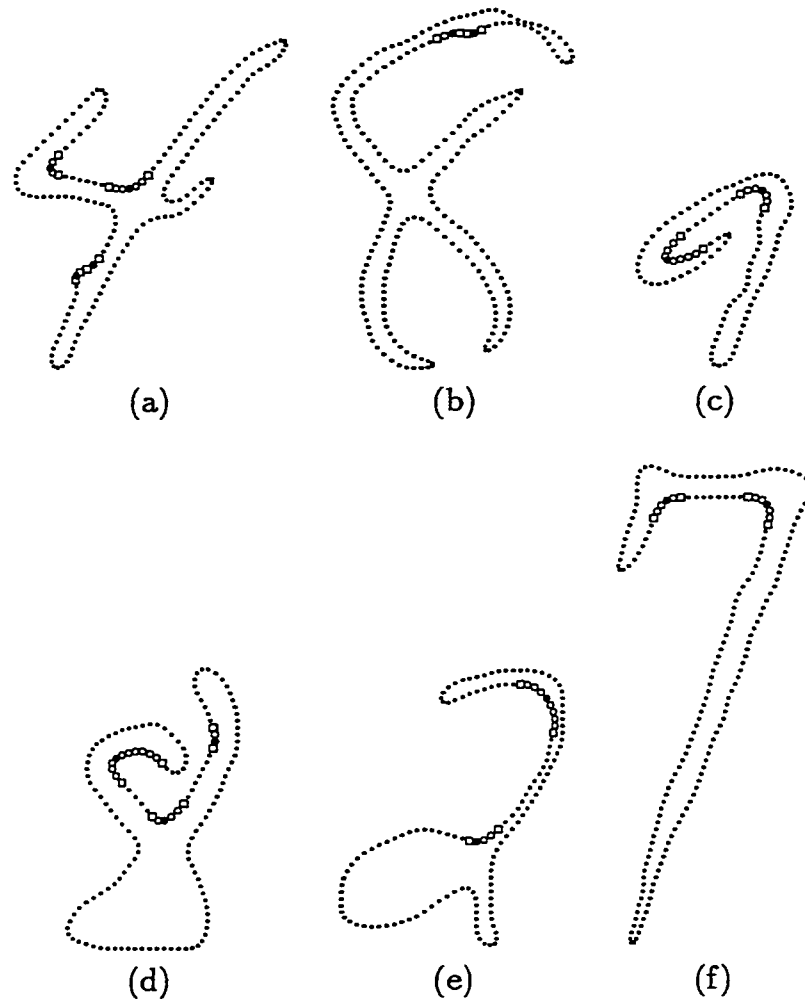


Figure 56: Cavities Not Merged With Neighbouring Cavities

The function `CHECK_MERGE_BENDS`

The approach to determine if consecutive bends should be merged is generally the same as that just presented for consecutive cavities; some of the exclusion tests are exactly equivalent to tests mentioned above while others are specific. For each sequence of bends, mostly the features are merged unless the following exclusion tests are positive:

- If the first or last bend in a sequence has a small cavity very close to it and if it is estimated that they together form a ‘wiggle-pair’, avoid merging this bend

with others in the sequence. The wiggle-pair is marked as such and generally filtered out at a later stage.

- Avoid merging with others a bend for which the cumulated deviation angle in a small neighbourhood around its focal point is more than 1.75; such a bend is generally significant on its own. See Figure 57 (a).

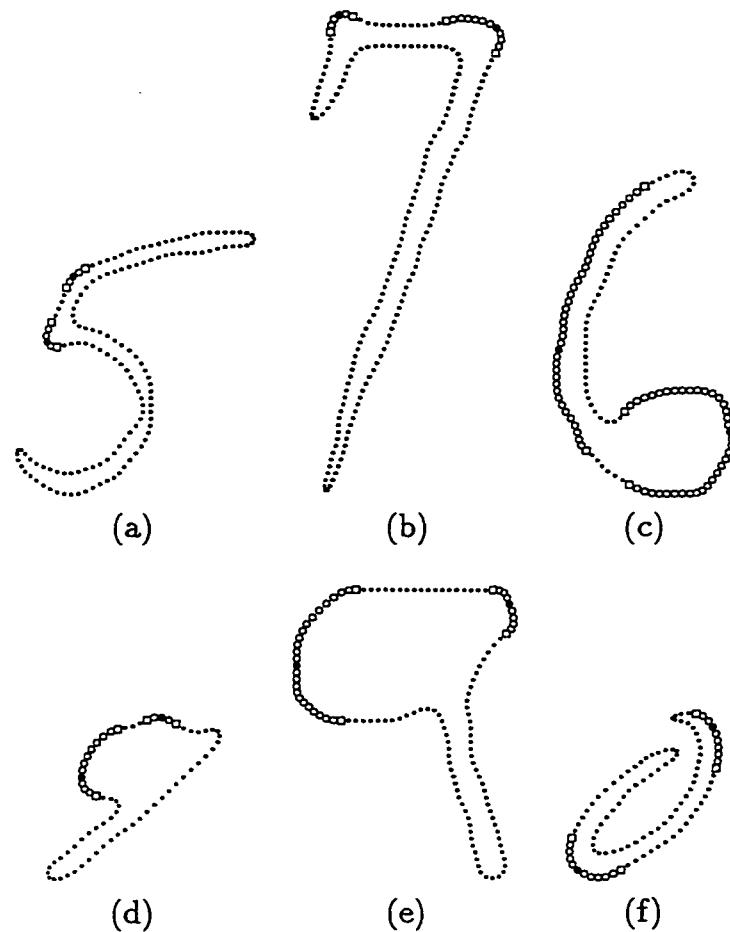


Figure 57: Bends Not Merged With Neighbouring Bends

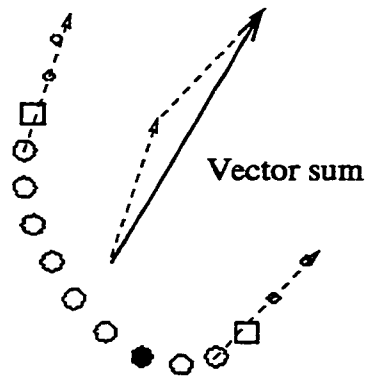
- Avoid merging with others a bend for which the cumulated deviation angle in a small neighbourhood around its focal point is more than 1.10, $\Phi^* > \pi/2$, and curvature is relatively high. See Figure 57 (b).

- Avoid merging a large and quite curved bend region with preceding bends of much smaller average curvature. This situation occurs mostly for 6's. See Figure 57 (c).
- Avoid merging a relatively small bend to a larger bend neighbour if the smaller bend has a much closer endpoint or cavity neighbour on the other side and the cumulated deviation angle between the bend regions $\Phi_{inter}^* < 0.10$. See Figure 57 (d).
- Avoid merging together very distant bends with small Φ_{inter}^* -values. See Figure 57 (e).
- Avoid merging with the preceding (or following) bend, a bend for which the cumulated deviation angle in a small neighbourhood around its focal point is larger than 1.0, and the distance to that preceding (or following) bend is large enough, and the inter-feature region between these bends is not too small and not too curved. See Figure 57 (f).

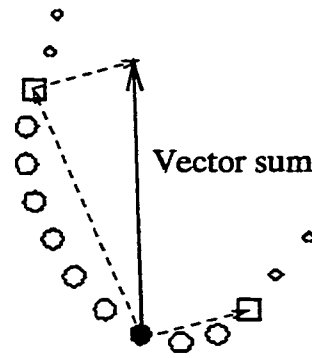
6.4.10 Computing Feature Directions

In Figure 37 (b), we illustrated how the *direction* of a feature was computed in the E4 feature extractor. Assuming that a feature region spans the contour point sequence $[s, f]$, the direction was obtained as the sum of a unit vector pointing from point s to point $s - 1$ and a unit vector pointing from point f to point $f + 1$. In general, this approach gives good results but occasionally, for endpoint regions with curved tips in particular, it is too sensitive to local direction. Experiments were carried out with 4 different approaches to direction measurement in order to find a more high-level method.

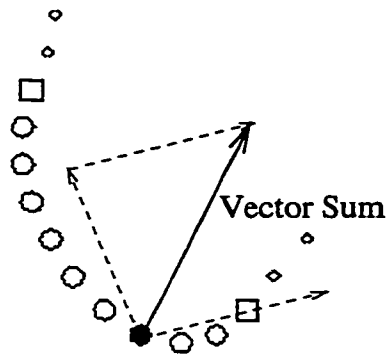
The first method still evaluates the overall direction of a feature based on local tangent directions at the extremities of the feature region. The unit vectors, instead of being based on *consecutive* contour points as in the E4 system, are directed from $s + 1$ to $s - 1$ and from $f - 1$ to $f + 1$ respectively. This is illustrated in Figure 58 (a).



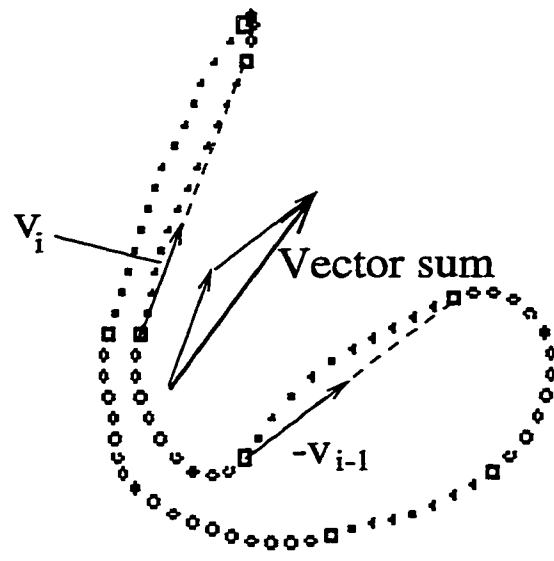
(a) Method #1



(b) Method #2



(c) Method #3



(d) Method #4

Figure 58: Four Different Direction Measurements

The second method is similar to that developed in Malowany [106]. It uses 3 special points of the feature region itself, namely s , f , and the focal point, to estimate the overall orientation of the feature. It proceeds by adding 2 vectors, the first connecting the focal point to the starting point s of the feature region and the second connecting the focal point to the final point f .²⁶ This is illustrated in Figure 58 (b). The third method uses the same approach but instead of adding the 2 vectors themselves, it adds two normalized vectors pointing in the same directions. This is illustrated in Figure 58 (c).

Finally, the fourth method is based on the location of a feature region with respect to its two neighbouring features. For feature region i , let \mathcal{V}_i be a unit vector directed from the final point of that region f_i to the starting point of the following feature s_{i+1} . We define the direction of feature i as being that of $(\mathcal{V}_i - \mathcal{V}_{i-1})$, where \mathcal{V}_{i-1} is a unit vector directed from f_{i-1} to s_i . This is illustrated in Figure 58 (d).

Note that methods #1 and #4 rely on contour point coordinates *after smoothing* using a triangular filter with window size $w = 5$ and weights equal to $(\frac{1}{9}, \frac{2}{9}, \frac{3}{9}, \frac{2}{9}, \frac{1}{9})$; methods #2 and #3 use the original unsmoothed contour point coordinates.

The 4 methods were compared using the 100 samples of file *train.us* as a reference. Based on a careful examination of their results, their ranking, from best to worse, was as follows: #4 (the best), #1, #3, and #2 (the worst). In general, the measurements of methods #1 and #4 were very comparable. Occasionally, they both yielded awkward computed values; but these occurred for very large ($|\Phi^*| > \pi$) bends or cavities in small and smooth zero's. In such cases, the identity of the numeral is rather easily inferred and the direction of the feature is unimportant. When a significant difference (say more than 10°) was found between the results of methods #1 and #4, the features were generally *curved endpoints*; method #1 is sensitive to such curved endpoints and gives the local direction whereas method #4 is more global and thus more immune to such local behavior. This is an advantage of method #4. For the top right endpoint of 4's, the range of direction values obtained (-105° to -130°) was narrower than for method #1 (-113° to -147°). For the top left endpoint of 7's and the bottom endpoint of 9's, a narrower range of direction values was also observed

²⁶ Note however that our focal point is defined quite differently than Malowany's 'apex'.

for method #4 compared to method #1.

Method #4 has other advantages. It requires 50% less computation than the other 3 methods since only the coordinates of the \mathcal{V}_i -vectors are required; this amounts to deriving one vector per feature, compared to all other methods which require 2 vectors per feature. In fact, the x- and y-coordinates of a given unit vector \mathcal{V}_i are stored as fields in the feature data structure. When large inter-feature regions are encountered (such as in 1's or 7's), they could be very convenient to assess the straightness of these contour portions (and their orientation, of course) and help in the classification process.

6.4.11 Assessment of New Feature Extractor

In this section, we will make an evaluation of our new feature extractor. This will be carried out in 3 parts. Firstly, we will consider all the problems identified in Section 6.3 entitled "Detailed Autopsy of E4 Feature Extractor" and discuss to what extent the new approach has solved these problems. Secondly, the results produced by the new feature extractor on the 100 samples of the comparative study in Section 6.2 will be presented. Thirdly, final comments will be presented and some shortcomings of the new approach will be reviewed.

Return on Problems Identified in 'Autopsy' of E4

Figure 59 shows the full results of the new feature extractor on the 4 samples of Figure 47 and the 3 samples of Figure 48. In Figures 59 (a) and (b), we see that the problems of bend-regions being combined with the endpoint itself have been resolved: the composite arcs were correctly partitioned and the focal points of endpoints are well located. In Figure 59 (c), the small middle right endpoint which was missed by E4 is now nicely captured. The endpoint at the bottom of the '5' in Figure 59 (d) is also detected by the new system. We note that in general the features obtained correspond well to perceived global curvature regions. Also, endpoint regions with extremely different stroke widths are extracted with success.

In Figure 48 (a), what we perceive as an endpoint was extracted as 3 very close

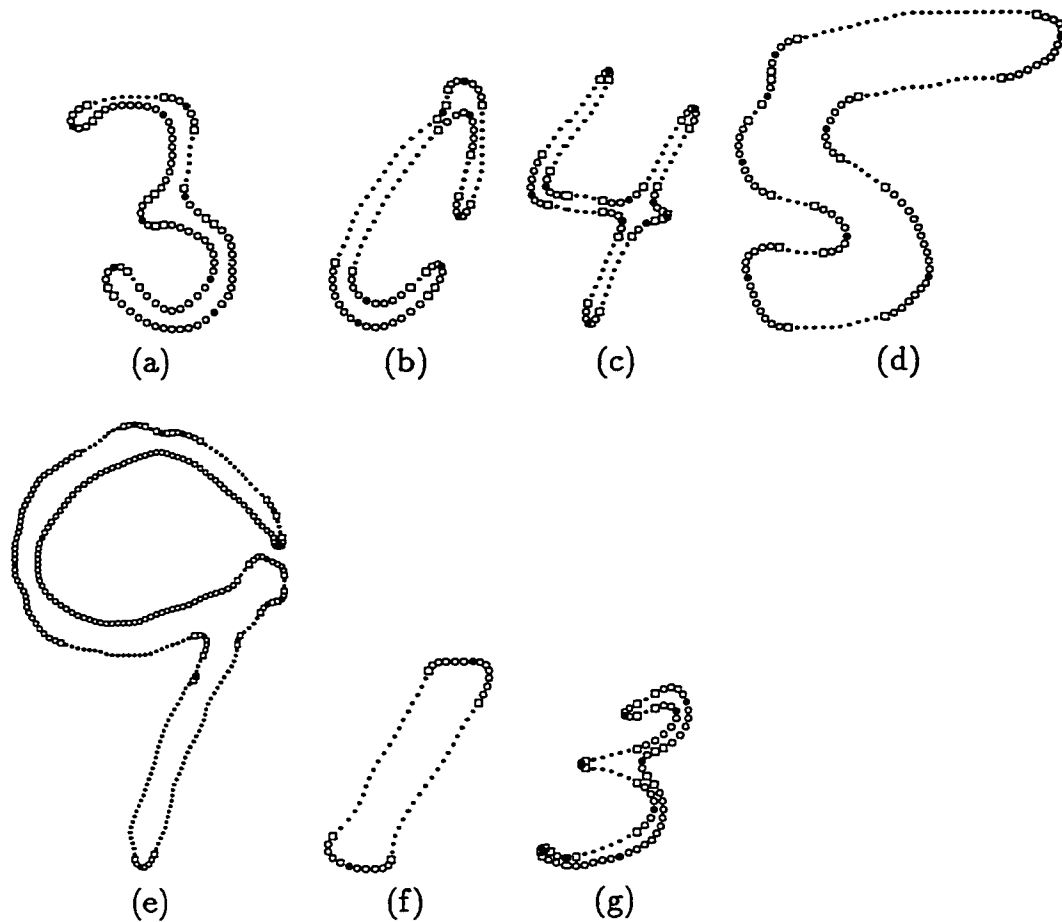


Figure 59: Samples Which Had End-Region Defects

bends by the E4 feature extractor. The new results are shown in Figure 59 (e). The endpoint is still not captured because only 2 of the 3 small bends are merged together; what prevented the merge of the 3 bends was the presence of a very close small cavity which was considered to make up a 'wiggle-pair' with the first of the 3 bends... However we note that the small parasitic bend at the bottom of the same '9' (See Figure 52) has been discarded by the new system.

In Figure 59 (f), the wide endpoints are correctly obtained and their focal points well located. In Figure 59 (g), the top right bend region is now extracted as such, not as an *endpoint* as was done by the E4 approach. These improvements are due to the new width measurement and criterion for endpoints and to a better method to

locate focal points.

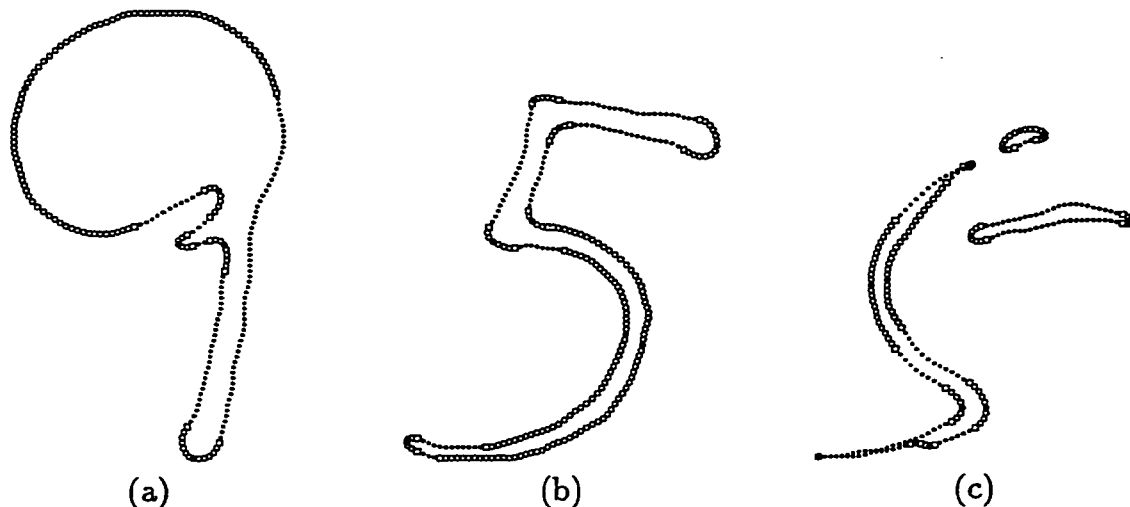


Figure 60: Large Curvature Regions Previously Undetected

The E4 feature extractor could leave large curvature regions completely undetected. Examples were given in Figure 49. The results obtained with the new feature extractor on the same image samples are presented in Figure 60. The improvements are very striking: not only are those large regions now captured but they are extracted globally as a single feature. Thus the entire convex region in the bottom right profile of Figure 60 (b) is now captured as a single Bend whereas before 3 tiny bends extracted by E4 still did not cover much of that region. Note also that the ‘wiggle-pair’ at the bottom of Figure 60 (c) has indeed been marked as such.

Now compare the results of our new feature extractor in Figures 61 (a) and (b) to what was obtained before in the E4 system as shown in Figures 50 (a) and (b). For the ‘8’, we note that the 2 middle cavities previously missed are now well extracted; in addition, several pairs of very close consecutive bends or cavities in Figure 50 (a) have now been merged. The top portion of the ‘9’, which was previously captured as a sequence of 5 tiny cavities and 5 small bends (also counting the last bend ‘buried’ in a composite B-E arc in the middle left part of the image), is now extracted as a single global cavity and a single global bend.

Concerning the merging of consecutive cavities and bends, the situations illustrated in Figure 51 are all handled correctly by the new feature extractor. The 2

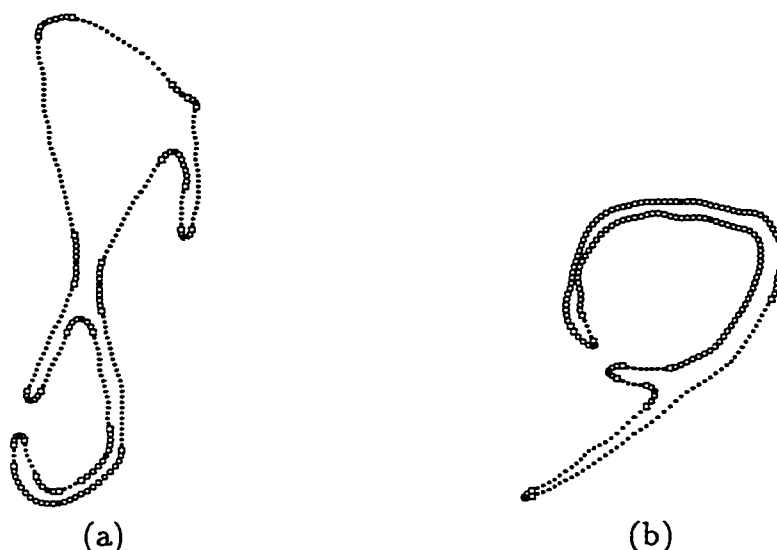


Figure 61: Previously Missed or Fragmented Feature Regions

nearby bends at the top left of the '4' in Figure 51 (a) are now combined to form an endpoint; the cavities and bends highlighted in Figure 51 (b) are still extracted and *not* merged together; finally, in Figure 51 (c), the cavity very close to the top left endpoint of the '7' has been discarded while the other 2 cavities have been preserved as distinct features.

Finally, the small parasitic features near endpoints of Figure 52 (a) and (b) have been removed in the new feature extraction method.

Comparing With Other Feature Extractors

In Section 6.2.4, seven curvature feature extraction methods were compared quantitatively and qualitatively based on their results on 100 digit samples. A 'measure of goodness' MG was computed for each method and each sample and average values of MG were provided (per class and overall) for feature regions of every type and for endpoints only. See Tables 16 and 17. Equivalent scores are provided for our new feature extractor in Table 18.

The overall MG -value for all classes and all feature types is now 0.77, which represents an 11.6% improvement over the previous best scores (0.69) for the Beus-Tiu and D'Amato et al. methods. On a class by class basis, our new curvature feature

New Method	0	1	2	3	4	5	6	7	8	9	Overall
All features	0.56	0.84	0.79	0.87	0.83	0.86	0.66	0.77	0.71	0.79	0.77
Endpoints only	0.86	0.87	0.87	0.87	0.88	0.94	0.70	0.83	0.81	0.86	0.85

Table 18: Average Measures of Goodness for the New Feature Extractor

extractor obtains best MG -values for all classes, except for 1's, 4's, and 7's. In the first 2 cases, our new method is a close second best; and in the last case, it comes in third place.

The overall MG -value for all classes but considering only endpoint regions is now 0.85. This represents a drop from the 0.90 score of the old E4 method; consequently, our new feature extractor only ranks fourth among the 8 methods tested. However, as previously explained, our definition of the 'measure of goodness' has some flaws and its values do not tell the entire story. Careful visual examination of the 100 samples of the test set reveals that endpoint regions are clearly missed in only 1 sample²⁷ and that the focal points for all these endpoint regions are generally very well selected. The lower score can be explained by three factors. First, there is the fact that our new method tends to extract narrower endpoint regions; thus the heart of these regions is almost always captured, while points at the extremities of the regions may be missed. The definition of MG assigns equal weight to all these points which penalizes our new method unduly. Of course, for classification purposes, this is irrelevant as long as these regions are detected and detected as *endpoints*.

The second and third factors leading to an apparently reduced performance in endpoint detection was revealed by considering the numeral class '6' for which the MG -drop is most severe (from 0.87 to 0.70). Close scrutiny revealed that samples #61, #65, #66, and #68 achieved very low MG -values (respectively 0.36, 0.39, 0.46, 0.48). For 3 of these samples, we are dealing with curved endpoints; and the human subjects extracted the "entire" curved region, while our new method trimmed such regions or separated them into a 'Bend-End' or 'End-Bend' pair of features (see Section 6.4.8).

²⁷ A tiny zero with an unusually smooth contour.

Again the focal points for these regions were very well located (better than done by the Rosenfeld-Johnston and Rosenfeld-Weszka methods which obtained higher MG -values for endpoint extraction). For sample #65 and a few others, several feature regions (not just endpoints) as well as their focal points appear 'shifted' from their desired location which reduces the MG -value considerably. This 'shift' is caused by the fact that our new method performs more sophisticated preprocessing than was done by E4, which was used to produce the reference contour points for human subjects and all other methods. As a result of this extra preprocessing, additional pixels are deleted from the original contours, causing the 'shift' when the newly extracted feature regions are mapped back onto the contour extracted by the E4 method. Again this unduly causes a penalty for our new method.

Method	MG-All	MG-Ends	MG-BC	f
Beus-Tiu	0.69	0.82	0.61	0.80
D'Amato	0.69	0.85	0.60	0.87
Legault-Suen	0.68	0.90	0.60	0.83
Rosenfeld-Weszka	0.67	0.93	0.53	0.80
New Method	0.77	0.85	0.69	0.87

Table 19: Comparing Overall Parameters For Best Five Methods

To investigate further the comparison between the best methods, new results were obtained. In Table 19, the overall MG -values are presented for all features, for endpoints only, and for bends and cavities (MG-BC in third column); and the fourth column shows the average f -value. The latter fraction was used in the computation of MG , as a penalty for methods detecting too many or too few features; an f -value of 1 would indicate that the method always extracts the same number of feature regions as the human subjects did. We see that in all areas except MG-Ends, which was already discussed, our new feature extractor scores best.

For the 100 samples in the test set and each method, we have also obtained the count of samples for which the method extracted more regions than humans, the count of samples for which the method extracted fewer regions than humans, and

the count of samples for which the number of regions extracted were the same. The results are presented in Table 20. The interpretation of these figures is as follows: for the Beus-Tiu method, for example, for 71 samples the method detected an average of 2.6 feature regions in excess of the number of regions identified by humans; for 8 samples, the method detected an average of 1.2 feature regions less than the number of regions identified by humans; for 21 samples, the numbers were the same.

Method	Count_More	Count_Fewer	Count_Same
Beus-Tiu	71 (+2.6)	8 (-1.2)	21
D'Amato	34 (+1.6)	32 (-1.4)	34
Legault-Suen	69 (+2.2)	5 (-1.0)	26
Rosenfeld-Weszka	42 (+2.3)	39 (-2.1)	19
New Method	25 (+1.8)	33 (-1.6)	42

Table 20: Sample Counts Comparing Regions Detected by Methods vs Humans

We see that our new feature extractor is the one which most frequently extracts the same number of feature regions as the human subjects did: this was the case for 42 samples out of 100. From the figures of this table, we can easily compute that on average, the Beus-Tiu method extracted 1.75 features too many per sample; the Damato et al. method extracted only 0.10 feature too many per sample; the E4 (Legault-Suen) method extracted 1.47 features too many per sample; the Rosenfeld-Weszka method extracted only 0.15 feature too many per sample; and finally, our new method extracted 0.08 feature too few per sample. Thus, *on average*, 3 methods seem to be able to extract the right number of regions: D'Amato et al. Rosenfeld-Weszka, and our new method. The higher MG-All value of our new method would then indicate that the contour point set it extracts for these regions matches more closely the contour point set extracted by humans.

We have further examined the results of our new method and of the method of D'Amato et al. (DA)²⁸ on the 100 samples, based on the 3 categories of Table 20.

²⁸ Based on the information of Tables 19 and 20, the comparison was conducted with the DA method because it appears to be the best, except for our own.

- The DA method extracts the same number of regions as expert human subjects did for 34 samples; but individual scrutiny reveals that for 14 of these 34 samples the regions are *not* all the same. This means that some regions identified by humans are missed or merged with neighbouring regions, while extra regions are extracted or large regions split into smaller ‘partial’ regions. In 5 of these 14 samples, a significant Bend was missed; in 2 cases, nearby ‘End-Bend’ pairs of features (in curved endpoints) were merged. In contrast, our new method extracts the same number of features as expert human subjects did for 42 samples and only for 6 of these samples are the feature regions not all the same. For these samples, no significant feature for recognition is missed.
- The DA method extracts fewer regions than human subjects did for 32 samples. In 11 cases, a large portion of a convex region is missed; in 6 cases, nearby ‘End-Bend’ pairs of features (in curved endpoints) are merged; in 1 case, an important cavity is missed. Our new method extracts fewer regions than humans did for 33 samples. In general, this is because of the merging of consecutive convex or concave regions into larger regions, which will not impact negatively on the classification stage; however, in 1 case, already mentioned, both endpoints of a tiny open zero are missed.
- The DA method extracts more regions than human subjects did for 34 samples. This is because large convex or concave regions are often split into several ‘partial’ sub-regions and because several useless small cavities (small bends to a much lesser extent) are detected. Our new method extracts more regions than humans did for 25 samples for similar reasons. However, there are fewer useless small cavities extracted than for the DA method. Some extra features detected are also already marked as wiggle-pairs which will facilitate their filtering at a later stage. Finally, in a few samples, the detection of additional regions is even an improvement over what human subjects perceived: occasionally, our new method captures composite arcs not detected by humans and splits them into an End-Bend pair of features; also, a very weak middle peninsula of a ‘3’

and an equally weak left middle cavity²⁹ of an '8' are extracted by our method while human subjects did not see them as significant enough to be detected.

To conclude this comparison, Table 21 presents a visual display of the relative distribution of specific weaknesses amongst the best 5 methods. The terms 'Incomplete features' refer to a feature of relatively large size for which a sub-region with significant curvature was left undetected. The terms 'Merged C-C or B-B' refer to pairs of consecutive features considered distinct by human subjects but merged as a single feature by the given method. The information was prepared by comparing the feature regions extracted by the 2 human subjects to those extracted by each method. Frequencies of occurrence of the problems were obtained and, for each 'defect', the maximum frequency was associated with 10 asterisks; other assignments were scaled proportionately. Thus, the relative frequency of one problem compared to another cannot be inferred from this table. For example, 10 asterisks were attributed to the Rosenfeld-Weszka method for 86 'undetected features' and the same number of asterisks was given to the D'Amato et al. method for 15 'incomplete features'.

PROBLEMS	METHODS				
	BT	DA	LS (E4)	RW	New
Undetected features	***	*****	*	*****	*
Incomplete features	*	*****	*****	*****	*
Misplaced focus/region	*****	*****	**	*****	*
Merged End-Bend	*	*****	*****	*****	
Extra features	*****	*****	*****	*****	*****
Fragmented features	*****	**	*****	***	*
Merged C-C or B-B		****		*	*****

Table 21: Relative Weaknesses of Best Five Methods

It is also important to realize that the weaknesses identified in comparison to perception by (2) human subjects are not all equally serious for our purposes. The top-down ordering of the problems in the table is that of decreasing negative impact

²⁹ In both cases, important features for classification.

on recognition, for methods of the type we have developed. The first problem, ‘Undetected features’ is definitely the most critical. The detection of ‘Extra features’ is a nuisance for tightly-defined shape models; but they may be the price to pay for guaranteeing that no significant feature is missed (and they can later be filtered out). To our knowledge, the last two ‘defects’ in the table have no adverse effect on recognition. Concerning the merging of consecutive cavities and bends, we must remember that our new method avoids such merges in situations such as those shown in Figures 56 and 57. It is in less clear-cut cases that those merges take place. Where the LS method was the worst, our new method is now the best at capturing large convex or concave regions globally. It may even have become ‘too good’...

Final Comments and Shortcomings of New Feature Extractor

To our knowledge, the new curvature feature extractor we have developed is definitely among the best currently available. As shown, it corrects all the major defects of the Legault-Suen method and compares very favorably to other feature extractors. It is *reliable* in that it almost never fails to extract a feature which is pertinent for classification. And it is *robust* since it only extracts a small number of relatively insignificant features (many of them are even marked as part of wobble-pairs, later to be discarded). As can be seen in many figures showing the outcome of the feature extractor, our new method generally produces a very nice coupling between ‘Bend-Cavity’ or ‘Cavity-Bend’ feature pairs which complement each other on the left and right profiles of several numerals; and it locates the focal point of a feature region in a manner which is intuitively very satisfying. It is also worth mentioning that for our new method, contrary to most other methods to which it was compared, there was no parameter adjustment to yield the best possible results on the 100 samples of the common test set. Furthermore, the reliability and robustness of our new feature extractor hold for a wide range of data resolution (between 166 and 400 PPI); to the very least, other methods would require parameter adjustment³⁰ to maintain their performance level over a similar range of resolutions. Finally, despite its higher complexity and sophistication, our new feature extractor remains very efficient.

³⁰ For examples of such adjustments, see the beginning of Section 6.2.4.

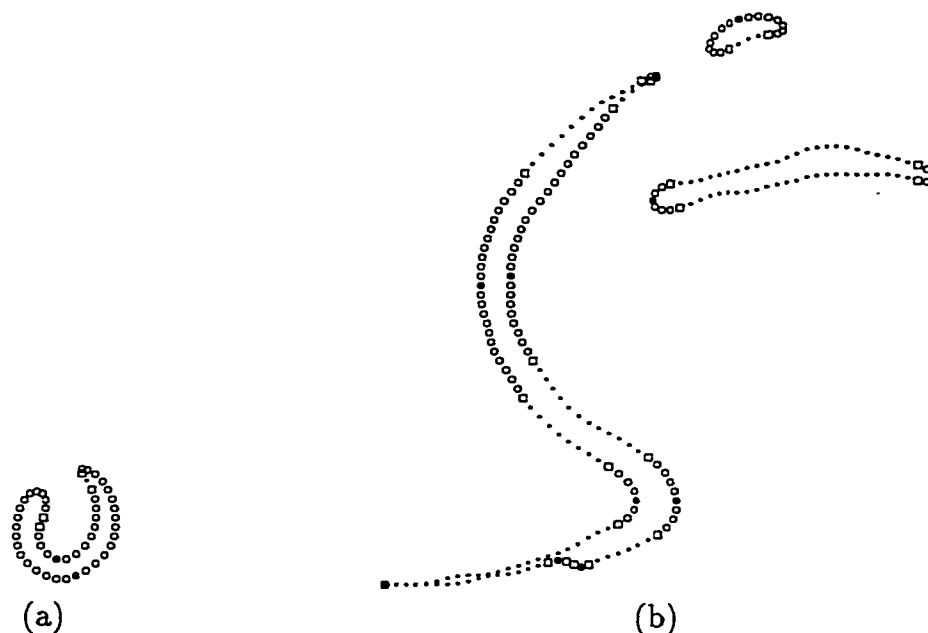


Figure 62: Leftover Shortcomings of New Feature Extractor

We can identify three remaining weaknesses of our new feature extractor. The first one was already mentioned: it is the case of the tiny smooth zero for which both endpoints were *missed*; this sample is shown in Figure 62 (a). In our later work with hundreds of images, we encountered one more sample for which an endpoint was left undetected. The second weakness concerns the features extracted from very small blobs. See for example the smallest blob in Figure 62 (b); only a Bend feature is extracted. This difficulty will be addressed later by developing special feature extraction procedures for small blobs. Finally, the problem of the very noisy contour of Figure 52 (c) is not resolved at this point. Getting rid of all tiny features at this early stage could be counter-productive as this might occasionally discard a small but meaningful feature for the classification process. This problem is thus best addressed at a later stage in the recognition process (when the first classification attempt fails). Nevertheless, of the 34 extracted features for this sample, a dozen are already marked as belonging to wiggle-pairs.

Chapter 7

Development of Classification Rules

The approach governing the production of our new classifier was outlined in Section 3.2. As discussed, it is not our goal to fully develop the classification stage of the recognition system; instead, we have chosen to demonstrate that the foundations we have built indeed allow to meet the high recognition and very high reliability performance targets that we have set. In this chapter, we explain how this approach was implemented.

The E4 recognition system had a tree classifier and every single classification rule was hard-coded in the source program¹. Our new classifier also has a tree-like core, but its key building blocks are repeated calls to the same function, `TRY_RULE_SET`, which verifies if specific subsets of rules are satisfied. These rules are not hard-coded into the classification program; instead, they reside in separate *rule files* where they are encoded according to a syntax designed specifically for this purpose.

Section 7.1 summarizes the various steps involved in developing classification rules for each numeral class. Essentially, our classification system was tailored to recognize the CENPARMI database. But the CEDAR database was also used in a secondary way which will be explained. Section 7.2 outlines the syntax which was developed to codify classification rules in a compact and convenient format. Section 7.3 presents

¹ The E4 classification program was written in Fortran.

the main functionalities of a Development Tool which was created to assist in the ‘painstaking’ task of handcrafting the needed classification rules. Section 7.4 summarizes the actual effort which went into the development of our partial classifier. Finally, Section 7.5 offers some concrete examples of classification rules for the numeral class ‘2’.

7.1 Steps Involved in Training for Each Class

As explained in the introduction to Chapter 4, the input data to our recognition system consist of files of *run-length encoded* binary images of single numerals. These files are simply referred to as *data files*. Considering one or several data files of training samples for a particular numeral class, here is a list of major steps which are performed in the training process:

- The samples of CENPARMI training sets A and B belonging to the class of interest are examined one by one; those which either do not really belong to the given class or whose presence would impact negatively on the training process are simply removed from the training material.
- Most classification rules require the value of a certain feature attribute to fall within acceptable bounds; in order to find discriminating attributes and to obtain the acceptable bounds, we must compile statistics for equivalent features on several samples. A first step in this direction is to define and mark on each sample a so-called *starting feature*; in general, this starting feature will not have the same index in the feature array of each sample, but it will be topologically equivalent.
- The set of training samples for the class of interest must then be partitioned into several subsets, each subset being collected in a distinct *model file*²; samples can be regrouped in a model file because they have a common specific shape for their entire contour, or because they share a particular shape variant for a certain

² The specific format of a *model file* is presented in Section 7.3.

portion of the contour of the considered numeral. Odd-shaped samples, samples with spurious holes, and samples with wiggly contours are also regrouped into 3 model files for each numeral class; this offers the basis for the investigation and solution of these problems when enough samples have been gathered.

- Specific sets of classification rules must be developed for model files and each set of classification rules is stored in a separate *rule file*. This is definitely the step which is the most time-consuming.
- The overall classification process must be adjusted by adding code to the program file "classify.c" which governs the ordering of rule files to be fired. As much as possible, we try to rank the traversal of rule files (and of individual rules within a rule file) to favor the efficiency of the classification stage.
- When all rule files have been developed for the class of interest, the partial recognition system is applied to all 4,000 samples of the CENPARMI training sets (A and B). Of course, samples which belong to other classes should be rejected; if misclassifications are found, the appropriate rule files are corrected and/or refined to avoid any error.
- The partial recognition system is finally applied to 13,954 samples of the CEDAR database. This is done primarily to test the reliability of the new rule files developed for the class of interest. Thus, if misclassifications are found, the appropriate rule files are corrected and/or refined to avoid these errors. An additional goal is to *relax* the classification rules developed for the CENPARMI database alone to improve the recognition ability of the system: several CEDAR samples are initially rejected even if their shape is very similar to those of CENPARMI recognized samples; accomodating them is often just a matter of broadening acceptable intervals for a few attribute values. Such adjustment is carried out as long as it does not cause any misclassification. There is *no attempt* to develop new rule files corresponding to specific shape models found in the CEDAR database but not significantly present in the CENPARMI database.

The above steps will be made clearer with concrete examples in Section 7.5, after presenting the syntax used to encode classification rules and the special tool developed to help in creating these rules.

7.2 Syntax for Classification Rules

7.2.1 Operator-Rules

As mentioned above, the majority of classification rules require the value of a certain feature attribute to be inside an acceptable range. The general form of this type of rule is as follows:

$$A\{op\} V_1 [V_2]$$

where A is an evaluated attribute represented by a string of 1 to 4 letters; $\{op\}$ is one of the 7 possible operators: $=$, \sim , $<$, $>$, $\&$, $|$, and $@$; V_1 (and V_2 , for the 2 operators $<$ and $>$) are values of a type compatible with the evaluated attribute. The values can be integers, real values, or single characters. The operator \sim tests for inequality (\neq) and the two operators $\&$ and $|$ are defined as follows:

$$A\& V_1 V_2 \quad \text{stands for} \quad V_1 < A < V_2$$

and

$$A| V_1 V_2 \quad \text{stands for} \quad (A > V_1) \text{ OR } (A < V_2)$$

The operator $@$ will be defined shortly. Depending on the specific attribute tested, not all operators may be allowed. Also, when the values must be of type integer or single character type, this is indicated explicitly in the rule by writing $V_1.i$ or $V_1.c$ respectively. Finally we note that, upon evaluation, all such 'op'-rules simply result in a true or false value.

7.2.2 Other Type of Rules

Some rules are not 'operator'-rules. They may affect the value of certain parameters of the classifier. For instance,

- Some rules set or modify elements of the array *PF*, a small array of pointers to features which are moved around the feature list of an unknown numeral.
- Some rules modify the elements of another small array called *stored_value* which is used as temporary storage to run all kinds of arithmetic and/or comparison tests. Note that values are originally stored as elements of this array with the @ operator mentioned previously.
- Some rules may set the value of the integer parameters *tested_hole_index* and *tested_piece_index*, compute the bounding box of a feature or feature sequence of the same type, or mark certain features as visited (or unmark them), etc...

By default, all these rules simply return a true value.

7.2.3 A Few Examples

Based on the general explanations given above, Table 22 presents a few examples of classification rules encoded with our syntax.

#	RULE	EXPLANATION
1	F2t= E	Is feature pointed to by PF[2] of type E (endpoint)?
2	I3=2+ 1	Set PF[3] to feature following that pointed to by PF[2]
3	F3d 2.7 -1.3	Is direction of feature pointed to by PF[4] larger than 2.7 or smaller than -1.3?
4	F3B1	Compute bounding box of same-type feature sequence of which feature pointed to by PF[3] is apart
5	Bt< 1.5	Is top row of bounding box smaller than 1.5?
6	Bb@ 0	Assign bottom-most row of bounding box to <i>stored_value[0]</i>
7	C0/4> 0.99	Is ratio <i>stored_value[0]/stored_value[4]</i> larger than 0.99?
8	M3	Mark feature pointed to by PF[3] as visited

Table 22: Examples of Syntax Rules

If rules #4 to #7 are consecutive rules in a particular rule file, their meaning is as follows. First, the feature pointed to by PF[3] may be preceded and/or followed

by one or many features of the same type. Rule #4 considers that entire sequence of same-type features and computes its bounding box, the associated coordinates of the extreme contour points responsible for that bounding box, and the cumulative deviation angle for that feature sequence; rule #5 verifies that the feature sequence reaches the first row of the image; and, since *stored_value*[4] is always the total number of rows for the current sample image, rule #7 verifies if the feature sequence also reaches the last row of the image.

For a complete presentation and discussion of the syntax, please refer to Appendix F.

7.2.4 Syntax, Classification, and Rule Generation

For a given numeral class, there are of course several model files. Model files have associated rule files, 'describing' a particular contour shape or small portion of a contour shape; but there is not necessarily a one-to-one relationship between model and rule files. For each numeral class, a file named '*d.rules_list*' (where *d* stands for a digit from 0 to 9) contains the list of the names of all rule files created for the recognition of that class. Initially before any sample is processed, the recognition system will, in turn, read each of the 10 files containing these lists and will also read every single rule contained in the rule files named therein.

Within a rule file, the rules are encoded, *one per line*, using the syntax previously outlined. During classification, by default, conjunction (AND) of rules is assumed. Thus, when a particular group of rules is being processed by the function TRY_RULE_SET, every consecutive ordinary rule must be satisfied; as soon as one such rule fails, processing of that rule file terminates without success and other shape variants (or portion thereof) encoded in other rule files can begin to be tested. Of course, there are several situations where we also need to consider the disjunction (OR) of individual (or small groups of) classification rules. Towards this end, an OR-rule has been defined in our syntax (see Appendix F).

There are several advantages to using the syntax we have created and storing the classification rules separate from the 'classification engine' program.

- The rules so expressed are much more compact than the corresponding *if*-statements would be if written in the source code.
- The separation of the rule files from the source code offers much more flexibility: rules can be re-organized, modified or added in a rule file without requiring the classification program to be recompiled every time.
- All rule files being processed by the same function, `TRY_RULE_SET`, it is a simple matter to track down the traversal of rule files as classification proceeds and to accumulate relevant statistics. For example, a specific label can be associated with each rule file; when the rule file is traversed successfully, this label is concatenated to the overall *classification label*, which will in the end provide the recognition path for a sample within our classifier. Such recognition paths are very useful in pinpointing which rule file to modify in order to accommodate rejected samples or to reject misclassified samples. Sample counts can also easily be updated indicating how many samples of the target class and how many samples of the other classes have traversed each rule file successfully. Such statistics could help assessing the efficiency of the classifier design.
- The firing of every single rule is controlled by the same function, `APPLY_RULE`. When we want to relax some rules to accommodate certain rejected samples, this makes it possible to pinpoint precisely which rule failed, causing the rejection.
- A parser was written to verify the syntactic correctness of a rule before adding it to a rule file. Since rule files can also be edited manually with any text editor, the parser is also called to verify every rule as it is read in by the system, before recognition begins.

Finally, we note that the same syntax is very useful for the development tool described in the next section, With that tool it is possible to formulate queries to visualize, from a data (or model) file, only those samples which satisfy certain conditions; and the conditions can simply be written with our syntax. More importantly, in the development and testing of new classification rules, statistics need to be collected on the range of values taken by certain feature attributes. Our syntax and its

associated parser can easily be adjusted to meet this objective. When processing a model file with our development tool, we can for example enter the incomplete rule 'F3d' (without an operator and a range of values); and the tool will process the samples belonging to the model file, obtaining for each the direction (d) of the specific feature pointed to by PF[3]; upon completion, the range of values for the direction of this feature will be displayed. If the rule seems valuable, it can then be entered automatically in the associated rule file with the range of values just found.

7.3 Tool for Rule Development

A very useful interface was constructed to facilitate database inspection and assist in the development of classification rules. This development tool allows the user to:

- Visualize the result of our feature extractor on any sample of a database which is run-length encoded and list the values of various attributes for each of the extracted features;
 - the samples of a file can be viewed consecutively;
 - the sample of a specific rank within a file can be viewed directly;
 - it is also possible to view only those samples which satisfy certain conditions (for example, view only the samples with 1 or more holes; or view only the samples with exactly 3 endpoints; etc.)
 - the tool can also be used to demonstrate the specific results of certain feature extraction steps such as the re-extraction of features for small blobs or the reconnection of broken pieces making up one numeral.
- Mark a starting feature on each sample; simple conditions defining the starting feature are written and the tool automatically performs the task; the user can view the result for each sample and accept or correct it.
- Separate the samples of a given class into a number of subsets (called *model files*) which will provide the material for the development of specific groups of

classification rules; the user can assign the sample to an already existing subset (by choosing the subset from a list) or can create a new subset.

- Process model files to produce corresponding *rule files*; in this mode, the tool provides statistics about the range of values taken by specific attributes of an equivalent feature on each sample; pointers to features of interest can be moved forward or backward on all samples of the model file and new statistics obtained for another feature.

The format of a run-length coded (RLC) file was already explained. Model files, on the other hand, are files of pointers to specific samples in RLC files, also giving the index of the starting feature for each sample. Their precise format is shown in Table 23.

<i>RLC file_name₁</i>	<i>n₁</i>
<i>rank₁</i>	<i>starting_feature₁</i>
<i>rank₂</i>	<i>starting_feature₂</i>
<i>⋮</i>	<i>⋮</i>
<i>rank_{n₁}</i>	<i>starting_feature_{n₁}</i>
<i>RLC file_name₂</i>	<i>n₂</i>
<i>rank₁</i>	<i>starting_feature₁</i>
<i>rank₂</i>	<i>starting_feature₂</i>
<i>⋮</i>	<i>⋮</i>
<i>rank_{n₂}</i>	<i>starting_feature_{n₂}</i>
<i>RLC file_name₃</i>	<i>n₃</i>
<i>etc</i>	<i>etc</i>

Table 23: Format of a Model File

On the first line, after the name of a first base (RLC) file, n_1 indicates the number of samples to be drawn from that file; the following lines in the model file (one per sample) provide the ranks of the samples within the specified RLC file and the index of their starting feature. Then, if needed, the name of another base file is given and the number of samples to be drawn from it etc. A model file can draw samples from any number of base files.



Figure 63: Database Inspection and Rule Development Interface

Note that in the visual inspection mode, our interface can process either base (RLC) or model files and display the result of the feature extractor on individual samples. The general appearance of our development tool is shown in Figure 63 on the preceding page. Appendix G describes it more fully and explains its various functionalities.

7.4 Overview of Classifier Development

Our aim is to demonstrate that it is possible to develop a single recognition method which achieves a high recognition rate and very high reliability, based on the general avenues outlined in Section 3.2 and in particular based on the robust curvature feature extractor that was developed for this thesis. As seen before, *reliability* is defined as:

$$Reliability = \frac{Recognition}{Recognition + Substitution}. \quad (43)$$

Imposing a threshold of α on reliability, we have arrived at a ceiling on the substitution rate:

$$Substitution \leq \frac{(1 - \alpha) \times Recognition}{\alpha}. \quad (44)$$

In particular, if we set a target recognition rate of 90%, we have the following requirements: for a reliability of 99%, the substitution rate must not exceed $0.\overline{90}\%$; for a reliability of 99.5%, the substitution rate must not exceed 0.452%; and for a reliability of 99.9%, the substitution rate must not exceed $0.09\overline{009}$.

To achieve a substitution rate of no more than 0.5% on a test set of 2000 samples, a numeral recognition system must not make more than 10 errors. Thus, on average, the classifier's rules for recognizing any particular digit class is not allowed more than 1 mistake. The performance matrix of the E4 system³ on the CENPARMI test set is presented in Table 24. The recognition rate is 94.65% and the substitution rate is 1.55%. To verify the feasibility of our goal, it was first decided to focus on the recognition of numerals of class '2'; as can be seen from Table 24, this part of the E4 recognizer was responsible for 7 (i.e. 22.6%) of the 31 errors of the system. If this

³ In its most refined version, after additional training with the ITRI database (see Section 3.1.1).

in/out	0	1	2	3	4	5	6	7	8	9	Rejected
0	188	0	1	0	0	0	1	0	2	0	8
1	0	196	1	0	0	0	0	0	0	0	3
2	0	0	192	1	0	0	0	0	0	0	7
3	0	0	1	191	0	0	0	0	1	0	7
4	0	0	0	0	186	2	0	0	0	2	10
5	0	0	0	2	0	190	0	0	0	2	6
6	0	1	0	0	0	1	192	0	0	0	6
7	0	0	4	0	1	0	0	185	0	1	9
8	0	0	0	0	0	1	0	0	190	0	9
9	0	0	0	0	4	1	0	1	0	183	11
Errors	0	1	7	3	5	5	1	1	3	5	

Table 24: Performance Matrix of E4 System

number could be brought down to 0 or 1, we would already have a good indication that our goal is achievable for a single recognition system. We also note from these results that 5 samples are mistakenly classified as '4's; and the same number are misrecognized as '5's and as '9's. The most frequent errors are '7's misclassified as '2's (4 samples) and of '9's misclassified as '4's (4 samples also).

Of course, we cannot base ourselves only on the E4 performance matrix to decide around which classes a partial recognizer should be developed to demonstrate the validity of our approach. In the literature, it is generally considered that the most confusing pairs in numeral recognition are the $4 \Leftrightarrow 9$ and the $0 \Leftrightarrow 6$ pairs. The E4 system seems particularly strong in dealing with the latter confusing pair, since only one '0' is mistakenly classified as a '6' and there is not one instance of the reverse error. Could our new system maintain this strength?

Unfortunately, several researchers provide only overall recognition statistics with no performance matrix. We focus here on relevant information provided by some authors. In Mai & Suen [105], for a test set of 8485 samples, the most frequent misclassifications are '2's recognized as '3's (18 cases) and '3's recognized as '2's (13 cases). In Lam & Suen [82], for the standard CENPARMI test set of 2000 samples, the most frequent error is '4's recognized as '9's (4 cases); there are also 3 instances

of each of the following misclassifications: $0 \rightarrow 6$, $2 \rightarrow 3$, $9 \rightarrow 5$, $9 \rightarrow 8$; on the other hand, the class causing the greatest number of confusions is '9', since a total of 8 samples are mistakenly recognized as '9'. In Abuhaiba & Ahmed [1], 6.7% of '2's are misclassified (mostly as '8's) and 6.5% of '6's are misclassified (mostly as '0's). In Heutte et al. [59], the most frequent misclassifications reported are of '4's recognized as '9's (1.55% of all '4's) and of '3's recognized as '2's (1.01% of all '3's); the two classes causing the greatest number of confusions are '2' and '9', responsible for respectively 15.6% and 15.5% of all errors. Finally, in Bailey & Srinath [14], it is said that the most confusing pair is $4 \Leftrightarrow 9$ (11 errors out of 660 samples, without a breakdown of the $4 \rightarrow 9$ and $9 \rightarrow 4$ errors).

Based on the above, it was decided to focus on 4 classes for the development of our partial recognizer: '0', '2', '6', and '9'.

7.5 Illustration of Approach for '2's

In this section, the process of developing classification rules for class '2' will be reviewed. This will provide concrete illustration of the steps outlined in Section 7.1 and of the use of the interface and special syntax we have created.

7.5.1 Assignment of Starting Feature

The starting feature for a numeral class is defined with a few simple rules. As the feature list of an unknown numeral is examined, those few rules are applied in turn to every feature until they are fired with success; this in turn triggers the entire classification attempt for that class. Thus a starting feature must be one which is present on every sample of the class of interest. Furthermore, it should be defined rather loosely; otherwise, some aspect of its definition might not be fulfilled for some members of the class.

The files *a-2.rlc* and *b-2.rlc* contain the 400 training samples of the CENPARMI database for class '2'. Initially, they were processed with the following definition of a starting feature: "The starting feature must be a cavity, preceded by an endpoint



(a) Starting Feature (b) Discarded: a-2.rlc (#34) (c) Discarded: b-2.rlc (#128)

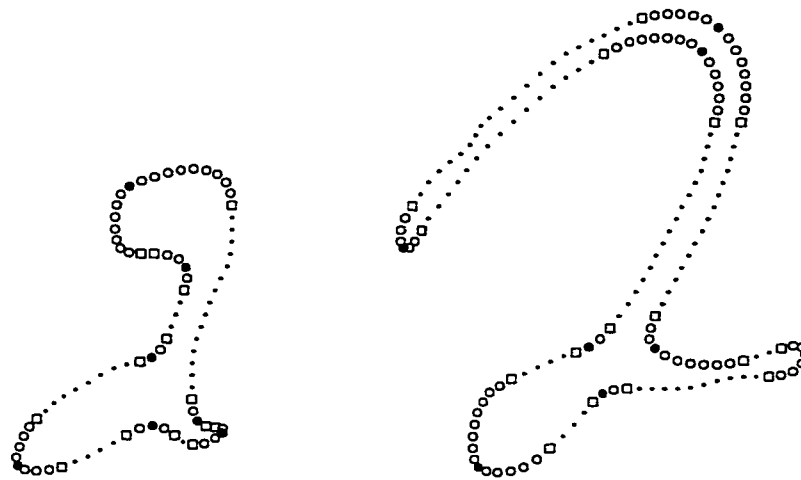
Figure 64: Starting Feature and Discarded Samples for Class ‘2’

feature whose focal point is located in the top half of the image”. The feature that we are targeting in this manner is the one highlighted in Figure 64 (a). Using our syntax, and assuming that the feature pointers $PF[1]$ and $PF[0]$ point to the feature of interest and to the preceding features respectively, this translated to:

```
F1t= C
F0t= E
F0xo@ 0
C0/4 < 0.5
```

After visualizing the result of this definition on several samples, using the *1ST FEATURE Test & Show* option of our development tool, this definition appeared promising. Thus these rules were stored in a special rule file called *start.rules.2*, and then read back by the tool and applied systematically using the *1ST FEATURE Mark & Verify* option. The results were somewhat surprising. In the sample by sample validation process, 2 samples were discarded because they are not really ‘2’s. They are shown in Figures 64 (b) and (c): sample #34 of file *a-2.rlc* is actually composed of two digits, a ‘1’ and a ‘2’, connected together; and sample #128 of file *b-2.rlc* is probably an incomplete ‘2’ (bottom portion only). Of the remaining 398 training samples, 11 had no top endpoint at all and 5 others had a top endpoint in the *bottom* half of the image. Figure 65 (a) is an example of the first situation: that ‘2’ has a top hole and the feature preceding the top cavity is a *bend* encircling that hole. Figure

65 (b) is an example of the second situation.



(a) No top endpoint (b) Top endpoint in bottom half of image

Figure 65: Problems with Starting Feature Definition

The definition of the starting feature for ‘2’s was therefore relaxed to the following 2 rules:

$$F1t = C$$

$$F0t \sim C$$

In words: “The starting feature is a cavity, preceded by a non-cavity feature”. With the above rules, individual validation revealed that the automatic assignment of the starting feature was successful in 99% of all cases; manual editing of the starting feature was only necessary in 4 of the 398 samples.

7.5.2 Clustering: Creation of Model Files

The clustering operation is performed with the help of the development interface using the *CLUSTERING Proceed* menu option. It consists of visualizing the training samples one by one and assigning them to a subset⁴ based on shared characteristics such as

⁴ A model file is created for each subset.

- The number and position of hole(s);
- A specific shape in a portion of the contour, represented by a similar feature sequence;
- The presence of certain anomalies: spurious holes, wiggle-pairs of features (i.e. small and nearby bends and cavities), odd and/or confusing shapes for the class of interest.

For the numeral class ‘2’, a total of 47 model files were created. Note that a sample may belong to more than one model file. Finally, subsets which are formed at this early stage may later be combined together if it is found that common rule files can accommodate them; and subsets with too few members may be abandoned, the information on these particular shapes being too skimpy to develop adequate classification rules.

Models Based on Presence and Position of Holes

Name of model file	Samples	Name of model file	Samples
2.bthole_2	200	2.top_and_btholes_2	9
2.straight_bottom_2	73	2.tophole_topB_2	9
2.open_bthole_2	27	2.high_bottomleft_2	2

Table 25: Models Based on Number and Position of Holes

If all digits were handwritten neatly and in accordance with Canadian shape standards, all samples of numeral classes ‘1’, ‘3’, ‘4’, ‘5’, and ‘7’ would be written without any hole; all samples of ‘0’ would be written with a large hole; all samples of ‘2’, and ‘6’ would be written with a bottom hole; all samples of ‘9’ would be written with a top hole; and all samples of ‘8’ would be written with both a top hole and a bottom hole. However, human writing styles are the result of various influences and subjects write with more or less care depending on circumstances; moreover, writing instruments as well as data capture and processing constitute additional factors which can cause

departure from standards. A brief analysis of the number of holes present in the 4000 training samples of the CENPARMI database can be found in Appendix H.

Six model files were created in relation to the presence and location of holes for the numeral class '2' (mostly, a bottom hole is present), and also in relation to the shape of the numeral in its bottom and bottom left portion. Their names are listed in Table 25 along with the number of samples regrouped in them. A typical numeral representative of the contents of each of these model files is displayed in Figure 66.



Figure 66: Example of Numerals for Model Files Based on Holes

Models Based on Shape of Bottom Right Portion of '2'

Because of the diversity of writing styles, there are several possible feature sequences (in terms of feature types, positions, directions, etc.) for the bottom right portion of the numeral '2'. A large number of model files were created to regroup samples with similar shapes in that part. For samples with a bottom hole, Table 26 presents the list of the names of these model files and the number of samples recorded in each. Table 27 presents similar information for samples without a bottom hole.

It is apparent from these tables that there are a few predominant writing styles corresponding to model files containing a relatively large number of samples. Developing classification rules for these few writing styles allows to rapidly build a core classifier with a good starting recognition rate. However there is also a large number of less frequent writing styles. Developing classification rules for each of these additional models represents the same amount of work but the associated increase

Name of model file	Samples	Name of model file	Samples
2.bh_CEC_2	108	2.bh_CdownE_2	3
2.bh_CBEC_2	24	2.bh_B_2	2
2.bh_CdownEC_2	22	2.bh_CB_2	2
2.bh_EC_2	13	2.bh_CECC_2	2
2.bh_CBEC_2	10	2.bh_CEBC_2	1
2.bh_CBC_2	6	2.bh_CdownEBC_2	1
2.bh_noprolong_2	4	2.bh_ChighEC_2	1
2.bh_BC_2	3	2.bh_notop_right_BorE_2	1

Table 26: Models for Bottom-Right Portion of '2's with a Bottom Hole

Name of model file	Samples	Name of model file	Samples
2.no_hole_EorB_E_2	67	2.no_hole_short_stem_2	3
2.no_hole_EorB_C_E_2	30	2.no_hole_EorB_B_ECC_2	2
2.no_hole_EhighE_2	12	2.no_hole_EorB_CBEC_2	2
2.no_hole_EorB_CBEC_2	6	2.no_hole_E_B_B_2	1
2.no_hole_EorB_B_E_2	5	2.no_hole_EorB_CBECBC_2	1
2.no_hole_EorB_CEBC_2	3	2.no_hole_EorB_CE_2	1
2.no_hole_EorB_E_CC_2	3		

Table 27: Models for Bottom-Right Portion of '2's Without a Bottom Hole

in recognition rate gets smaller and smaller. This is a concrete illustration of the 'diminishing returns' phenomenon mentioned in our introduction to Chapter 3.

Figure 67 shows one example of the typical shape (or feature sequence) of interest for the first 6 model files of Table 26. In Figure 67 (a) the bottom-right ending of the numeral '2' is a straight stem; in Figure 67 (b), after creating the bottom hole, the pen stroke typically first goes downward and then curves upward again.; in Figure 67 (c), the stem points markedly downward; in Figure 67 (d), the bottom portion of the numeral is quite flat and there is no cavity feature in that region; in Figure 67 (e), the writing style is similar to that of Figure 67 (b), but the ending stroke of the '2' is much longer and the final upward curve creates an extra cavity too far from the other (more standard) cavity to be merged with it.

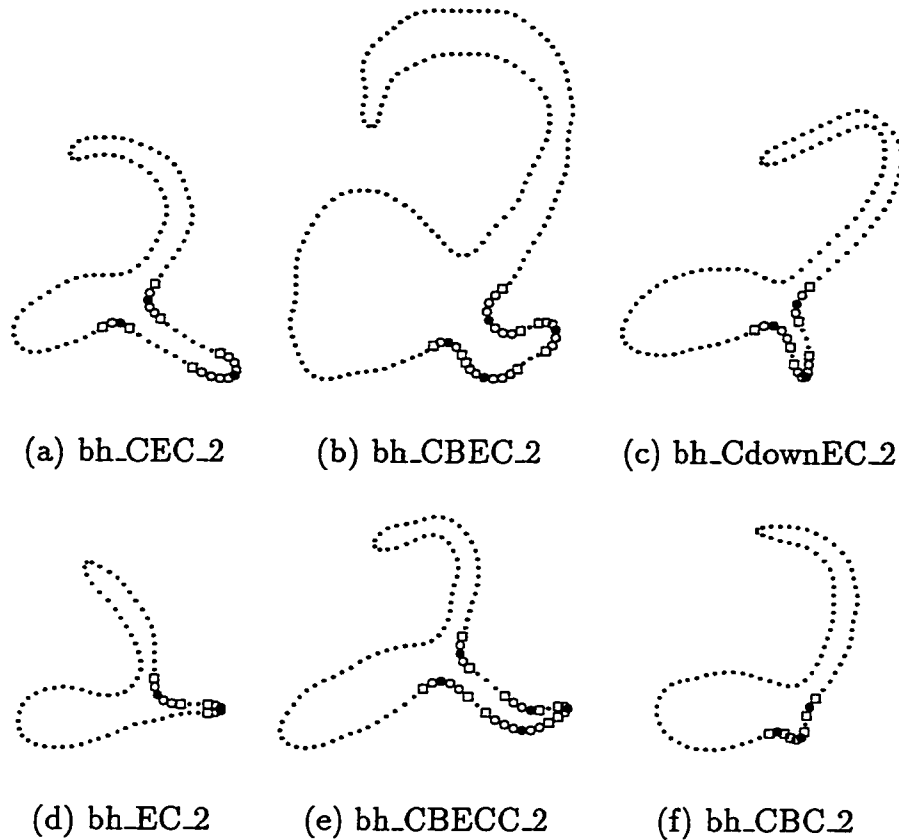


Figure 67: Some Bottom-Right Feature Sequences for ‘2’s With Bottom Holes

Figure 68 shows one example of the typical shape (or feature sequence) of interest for the first 6 model files of Table 27. While the styles illustrated in Figures 68 (a), (b), and (d) are similar to others encountered in ‘2’s with bottom holes, the writing styles displayed in Figures 68 (c), (e), and (f) are specific to samples written purposely without a bottom hole.

Models Based on Shape of Top Portion of ‘2’

Following the contour of a ‘2’ in counter-clockwise fashion, the next features encountered after the bottom right (stem) portion are much more stable: after the quite standard cavity, a convex⁵ region normally makes up the top right part of the right

⁵ Generally, this is a large and smooth *bend* feature; but it can be written more sharply and be extracted as an *endpoint*.

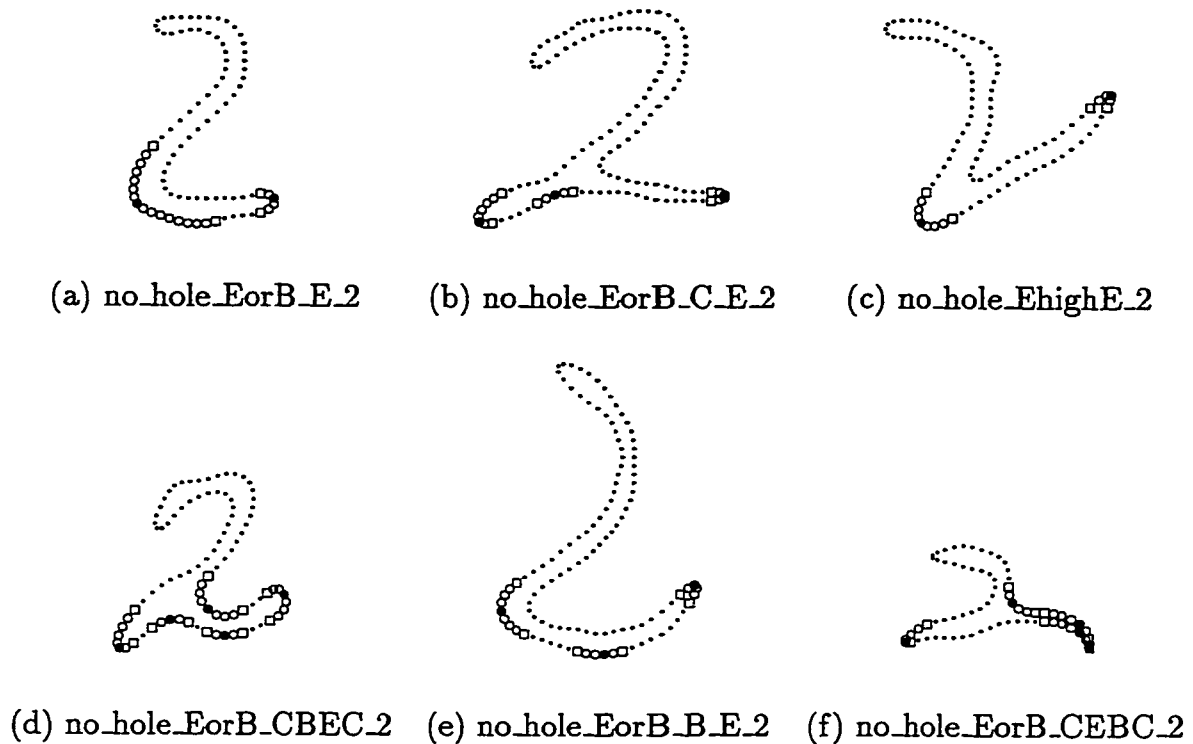


Figure 68: Some Bottom-Right Feature Sequences for ‘2’s Without Bottom Holes

profile. But after that point, in the ‘top portion’ of the numeral, there are once again variations in style. Table 28 lists the names of the model files created to regroup samples based on the different shapes for that part of the contour. The so-called *simple_top* model represents the overwhelming majority of cases. In fact, all 12 images in Figures 67 and 68 are of this type.

Figure 69 presents examples of less common shapes for the top portion of the handwritten numeral ‘2’. Despite their lower frequency, they are not at all accidental artifacts; clearly, they are manifestations of some individuals’ writing style. Another aspect of the ‘diminishing returns’ is apparent here: not only is the gain in recognition rate lower and lower as the frequency of the model decreases, but these less frequent shapes often correspond to more complex feature sequences. Thus developing rule files for these rarer variations can even mean *more work!*

Name of model file	Samples	Name of model file	Samples
2.simple_top_2	337	2.topB_2	2
2.topC_BorE_E_2	25	2.topCB_2	1
2.curly_start_2	4	2.topEBE_2	1
2.topC_EorB_CE_2	2		

Table 28: Models for Top Portion of '2's

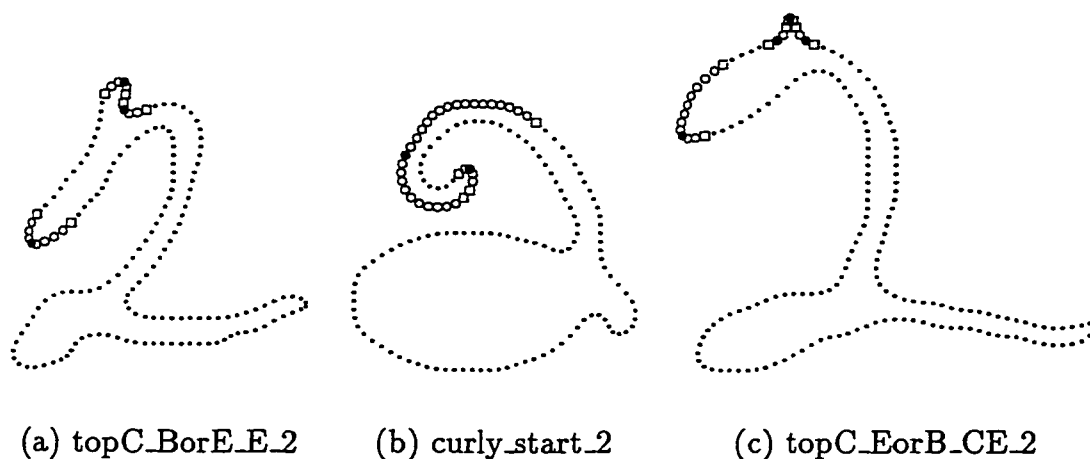


Figure 69: Some Less Common Top Feature Sequences for '2's

Models Based on Shape Anomalies

During the individual screening of training samples in the *CLUSTERING Proceed* mode of the development interface, samples whose shape was considered quite rare or peculiar were put in the model file *2.odd_samples*. The 'odd' shape can be the result of several factors including writing style, sloppiness, processing such as binarization, etc. At this stage, no decision is made about whether the classifier should make accommodations for these shapes or avoid doing so. The 5 samples which were put in this category are shown in Figure 70.

Another anomaly which was noticed for several classes is the presence of spurious holes, generally within the breadth of the pen stroke. Of course, such holes are tiny;



Figure 70: Samples of '2' With Unusual or Peculiar Shapes

but discarding *all* tiny holes is not necessarily an appropriate solution since expected and significant holes are sometimes tiny too (especially at low resolutions such as 166 or 200 PPI). To investigate the problem and find a better solution, it was decided to regroup samples of a given class carrying such spurious holes into a model file. For '2's, this model file is called *2.spurious_holes* and contains 5 samples. Examples of a spurious hole and of an equally tiny but this time significant hole are given in Figures 71 (a) and (b) respectively; in both cases the hole has an area of 2 pixels only. Note that in the case of the first of these images the preprocessing is able to get rid of the wide horizontal 1-pixel thick artifact and thus extracts a decent contour for that '2'.

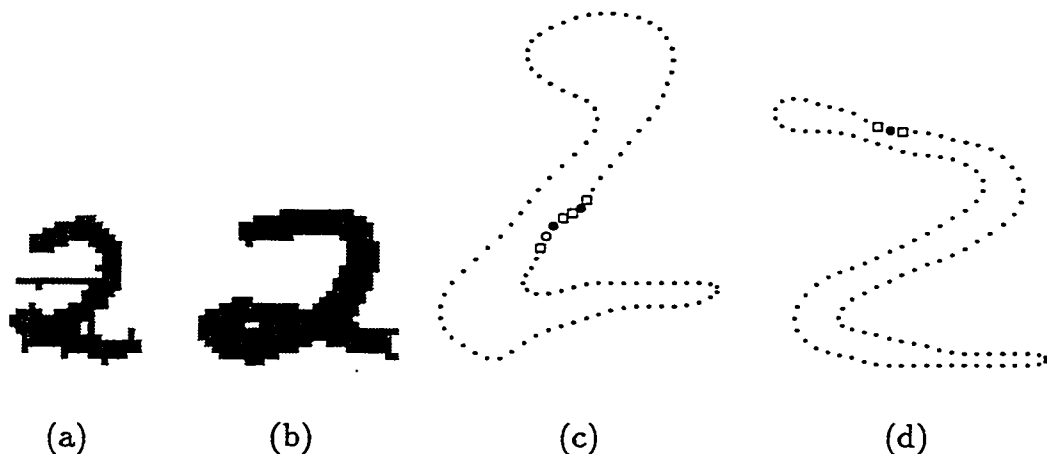


Figure 71: Samples With Tiny Holes or Spurious Features

Finally, 20 samples for which the feature list included ‘wiggle-pairs’ or small artifacts which are not clearly style-related and hence cannot justify the creation of distinct model files were regrouped in a model file called *2.wigglyContours*. The same is to be done when samples of other classes are clustered. The intent is to create a specific database to investigate the problem of filtering out these wiggles and small artifacts after a first classification attempt fails. If the filtering is successful at removing only the unwanted features, a second recognition attempt would then succeed. Figure 71 (c) illustrates a ‘wiggle-pair’, already marked as such by the feature extractor; the small cavity in the top portion of the sample in Figure 71 (d) shows an example of a spurious feature which should also be filtered out at a later stage.

7.5.3 Rule Generation: Creation of Rule Files

A rule file may be specifically developed for a model file, but not necessarily since this might produce duplicate rules in several rule files. For example, the 2 cavities highlighted in Figure 67 (a) are also present in Figures 67 (b), (c), (e), and (f). Hence it is better to create a specific rule file for the testing of these two cavities (individually and in their inter-relation). Similarly, there is a bottom-right endpoint in almost all models shown in Figures 67 and 68, and it would be counter-productive to replicate in as many rule files the tests to be performed on that endpoint feature. Thus several model files may be put to contribution to develop a rule file associated with one or many features which they have in common; and several rule files may be required for the recognition of some shape variant embodied in one model file.

Playing Safe

In developing the classification rules, we have applied the philosophy outlined in Section 3.2. For instance, the samples shown in Figure 72 were voluntarily left out of the process: the samples in Figure 72 (a), (b), and (c) are deemed potentially confused with a crossed-out ‘7’, a strange ‘3’⁶, and a European-style ‘1’ respectively;

⁶ Preprocessing removes the horizontal 1-pixel thick line in the middle right portion.

the other 2 samples are too unique in their style⁷.

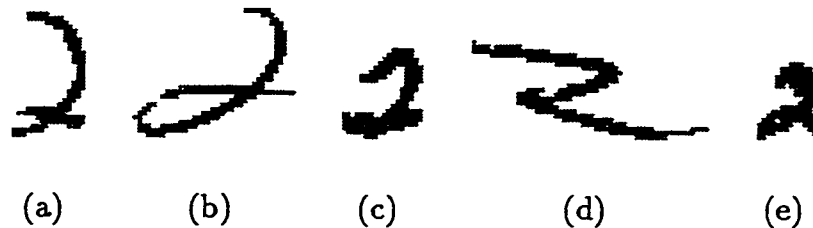


Figure 72: Samples Left Out of Rule Generation Process

Furthermore, great care was taken to define *safe boundaries* between classes and to ensure that samples would be rejected rather than risking an error. Some of the implications are illustrated with various samples⁸ in Figure 73. The first 2 images indicate the need to define a safe boundary between ‘2’s with a bottom hole (Figure 73 (a)) and ‘0’s with a hole (Figure 73 (b)); this boundary could be set using the maximum height reached by the hole, relative to the number of rows. The sample of Figure 73 (c) is a member of the file *a-2.rlc*; not only must it not be recognized as a ‘2’, but it should also be excluded from the class of ‘7’s if we were to develop rules for that class. The last 2 samples exemplify the danger of $2 \Leftrightarrow 7$ misclassifications: Figure 73 (d) is a ‘2’ which our classifier recognizes as such; Figure 73 (e), on the other hand, is a ‘7’ which should preferably not be incorporated in either of these classes.

In general, a rule file carries out exhaustive testing on the specific feature or feature sequence which it is meant to capture. We can summarize the results of our efforts as follows: sixty three (63) rule files were created for the classification of ‘2’s; these files contain anywhere from 1 to 41 rules, for a total of 613 rules. This work is quite time-consuming and a detailed presentation of these rules is of course not relevant. But we will discuss the classification approach and highlight some relevant aspects in

⁷ After preprocessing, the 1-pixel hole in the last image is filled; and there is only one large bend feature at the top.

⁸ Some of these samples come from the CEDAR database; see Section 7.5.5.

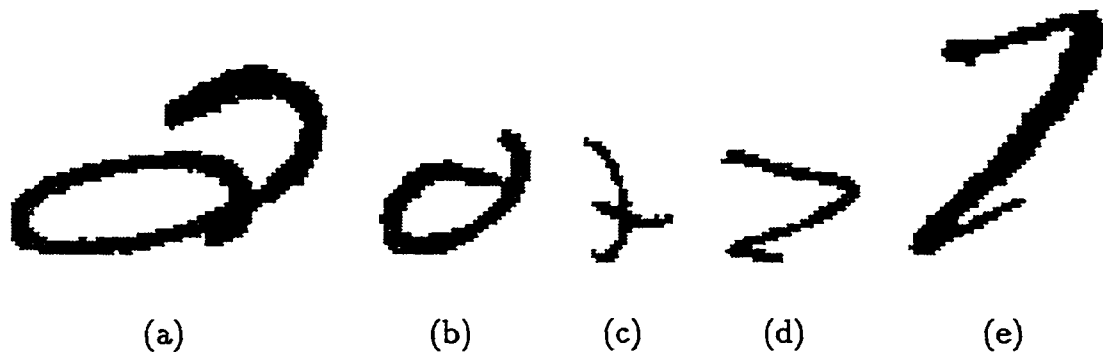


Figure 73: Defining Safe Boundaries for Recognition

the next subsections.

Processing an Unknown Sample for Membership in Class '2'

There are samples with a bottom hole, with a top hole, with both bottom and top holes, and without any hole. The position ratio⁹ is used to determine where a hole is located. For 2's, a hole with *position_ratio* > 1.24 was initially acceptable as a candidate bottom hole; and a hole with *position_ratio* < 0.35 was initially acceptable as a candidate top hole. The number and position of holes initially orient the classification process.

After this initial categorization, the approach is to follow the external contour of the numeral, applying various rule files to consecutive portions of the feature list until the contour 'tracing' is completed. As an example, consider the attempts to recognize a '2' with a bottom hole. After the starting-feature rules (see Section 7.5.1) have been fired successfully for a cavity in the feature list of an unknown sample, the sequence of events is as follows:

- the initial cavity (or cavity sequence) undergoes various tests;
- the hole and the bottom portion of the contour encircling the hole undergo various tests;

⁹ The number of rows above the hole divided by the number of rows below the hole.

- the bottom-right portion of the contour is tested for the presence of any of the acceptable shape variants already discussed;
- the top-right portion of the contour is tested;
- the top portion of the contour is tested for the presence of any of the acceptable shape variants already discussed;

When rule files are fired successfully and certain ‘milestones’ are reached around the contour, features in the feature list are marked as having been traversed successfully¹⁰. When all rule files required for a certain model have been fired with success, the classifier makes sure that all features in the feature list have been marked as a result; if there is any unmarked feature, recognition as a member of the corresponding class fails.

The approach of allowing several variants for specific parts of a numeral, of marking these parts when traversed with success, and of requiring that all parts be marked before recognition succeeds was already used by recognition expert E1, as discussed in Suen et al. [157], for the branches of their skeletonized patterns (see also Nadal & Suen [117]). Here we have applied these notions to the *contours* of numerals, and with more refinement concerning the number and definition of the allowed variants.

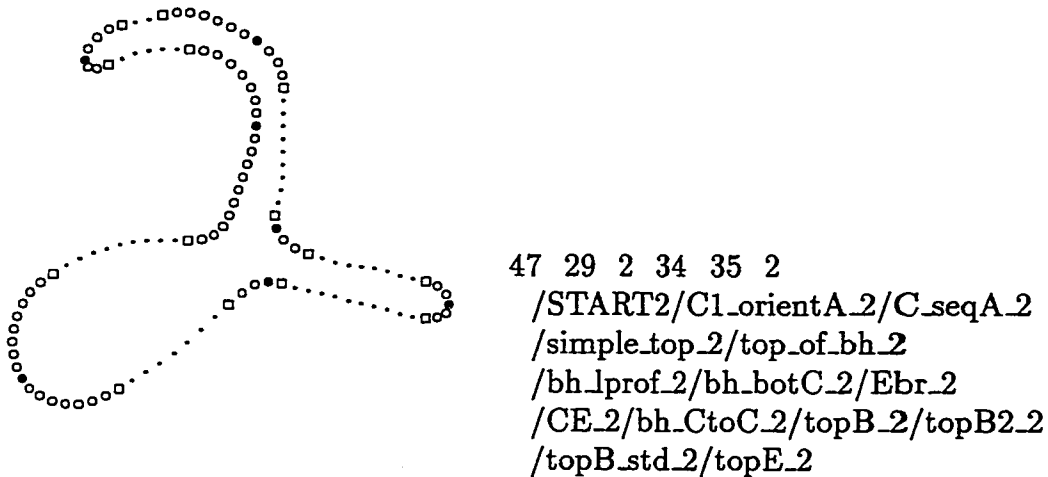
Examples of Specific Rule Files

When the new recognition program is applied to a run-length-coded file of individual digit images, it produces an output file (with extension *.ids*) indicating the classification results, using 1 line per sample. The typical output line begins with 6 integers, followed by the *classification label* produced by the classifier. The first 5 integers are identical to those of the *.rlc* input file¹¹: a file number; a sample number; the numeral’s identity, the number of rows and the number of columns in the image. The sixth integer is the classifier’s decision, in the range 0-10, with 10 indicating rejection.

¹⁰ If classification as a member of that class fails at some ulterior point, features will be *unmarked* before recognition as a member of another class is attempted.

¹¹ See the introduction to Chapter 4.

The image of sample #53 in file *a-2.rlc* is displayed in Figure 74 (a) and the corresponding 53rd line in output file *a-2.ids* is shown in Figure 74 (b), spread onto several lines.

(a) Sample #53 in *a-2.rlc*

(b) Output line for that sample

Figure 74: Classifier Output for a Specific Sample

Each name between slashes is a partial classification label indicating that the rules in the rule file of the same name have been fired successfully. The initial *START2* refers to the starting feature rules discussed in Section 7.5.1. Thus, after that point, we know that *PF[1]* points to the large west-pointing cavity and *PF[0]* points to the preceding endpoint. We now examine the specific contents of some of the rule files involved:

- *C1_orientA* has only 2 rules:

1. F1B1
2. Bov= b

After ‘computing’ the bounding box of the (possible) cavity sequence including

PF[1], this requires that its vertical orientation be from top to bottom.

- *C_seqA* contains the following rules:

1. F0xo@ 0
2. C0/4 < 0.61
3. F1B2
4. Bt@ 0
5. C0/4 < 0.375
6. F0B1
7. Bl@ 0
8. F1B1
9. Bl@ 1
10. C0 < 1
11. Br@ 1
12. C1-0@ 1
13. C1/4 > 0.20
14. F1B3
15. Bt@ 0
16. S0- 1.0

The first 2 rules require that $d_1 < 0.61 \times n_rows$ (See Figure 75 (a)). Rules 3-5 require that $d_2 < 0.375 \times n_rows$. Rules 6-13 require that $d_3 > 0.20 \times n_rows$. The last 3 rules ‘compute’ the bounding box of the contour portion extending from the first point of the top endpoint region to the first point of the bottom left bend region and store the topmost row (minus 1.0) in the array element *stored_value*[0]. This bounding box is referred to later in rule file *bh_lprof_2*.

- *simple_top_2* contains only the following rule:

1. C0/6 < 1.45

This requires that $stored_value[0] < 1.45 \times SW$, where SW is the estimated stroke-width.

- *top_of_bh_2* contains only the following 2 rules:

1. Ht@ 9

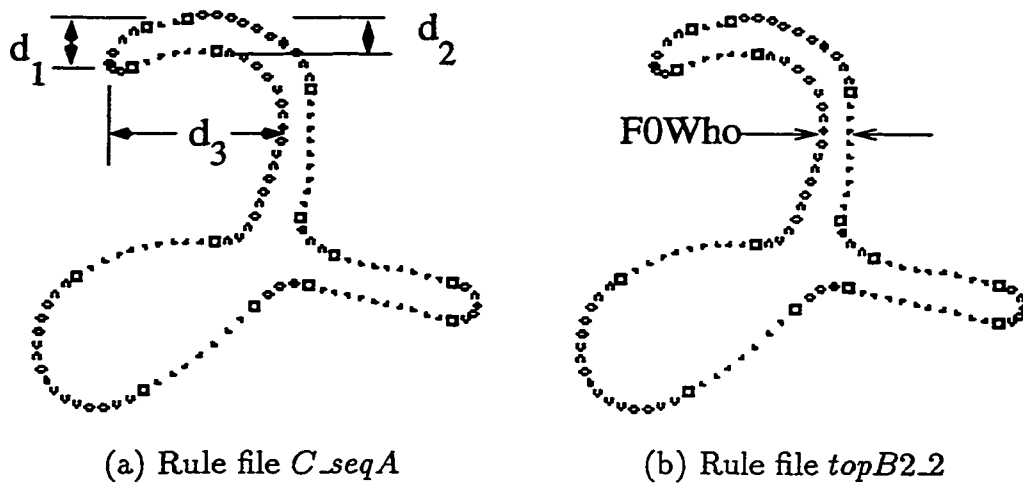


Figure 75: Some of the Distances Involved in Rules

2. $C9/4 > 0.4$

which require that the top row of the hole be located in the bottom 60% of the image.

• *bh_lprof_2* contains the following rules:

1. Br@ 0
2. C0-7@ 0
3. Hr@ 8
4. C0-8> -1.1
5. Bb@ 0
6. C0/4> 0.4
7. I0=1+ 0
8. I1+ C
9. F1H2
10. C0/6< 1.5
11. C2/4> 0.7
12. M01-

Rules 1-4 require that the bottom hole lie almost entirely to the left of the

rightmost column reached by the above cavity (sequence). Rules 5-6 require that the lowest point of the bounding box computed at the end of rule file *C_seqA* be in the bottom 60% of the image. Rule 7 advances PF[0] to PF[1] and rule 8 advances PF[1] to the next non-cavity feature. Rule 9 is actually a call to a fairly extensive function which follows the non-cavity feature sequence, starting with PF[1], around the outside contour of the bottom hole; this function computes 3 estimates of the body region thickness above, to the left, and below the hole. An average of these 3 thicknesses is saved in *stored_value*[0]; the leftmost column encountered around the hole is saved in *stored_value*[1]; and the lowest position reached under the hole is saved in *stored_value*[2]. PF[1] is advanced to the cavity following the non-cavity features encircling the hole (hence to the bottom cavity in this case). Rule 10 ensures that *stored_value*[0] $< 1.45 \times SW$. Rule 11 ensures that *stored_value*[2] reaches into the bottom 30% of the image. Finally, rule 12 marks all features from PF[0] to PF[1], excluding the latter.

- *bh_botC_2* contains the following rules:

1. F1t= C
2. F1xo@ 0
3. C0/4> 0.49
4. F1yl@ 0
5. C0-7@ 0
6. C0/8> 1.0
7. I2=1+ 1
8. O 2 0
9. F2t= C
10. I2+ 1

Rule 1 verifies that PF[1] is indeed a cavity. Rules 2-3 ensures that the focal point of that cavity is located in the bottom 51% of the image. Rules 4-6 verify that last contour point of PF[1] cavity region is to the right of rightmost column reached by the bottom hole. Rule 7 makes PF[2] point the feature following PF[1] cavity (hence the bottom right endpoint in the case of Figure 75). Rule 8 states that we must satisfy the next 2 rules (9-10) or none at all... Rules 8-10

simply amount to saying that "if PF[2] points to a cavity feature, advance PF[2] one step further".

- Finally, we jump ahead and consider one last rule file, *topB2_2*, which contains the following rules:
 1. O 2 4
 2. F0Who@ 2
 3. C2/6 < 1.4
 4. F0xo@ 2
 5. C2/4 < 0.33
 6. F0Whl@ 2
 7. C2/6 < 1.4

Rule 1 requires that either rules 2-3 or rules 4-7 be satisfied. Rule 2 stores into *stored_value*[2] the horizontal width across the body region taken at the focal point of PF[0]. This is illustrated in Figure 75 (b). Rule 3 requires that $\text{stored_value}[2] < 1.4 \times SW$. If this condition is not successful, rules 4-5 ensure that this focal point is in the top third of the image and rules 6-7 apply the same criterion to the horizontal width across the body region taken at the last point of the feature region PF[0].

7.5.4 Results With CENPARMI Training Samples

After developing the classification rules for '2's as explained above, the partial recognition system was applied to all 6000 samples of the CENPARMI training and testing sets. The results are shown in Table 29. The recognition rates on sets A, B, and T are respectively 91%, 94%, and 78.5%. There are no errors at all. Given that misclassification as '2's was the most important source of errors in the E4 system, this is already a notable achievement. In addition, the majority of the 30 samples of '2' from sets A and B which are rejected have small spurious holes and/or tiny noise features along their boundaries. Thus, another classification pass, after proper filtering of these artifacts, would likely provide a non-negligible increase in the recognition rate. This would be most welcome, since the 78.5% value for the test set is not satisfactory.

in/out	Train A		Train B		Test T	
	2	Rejected	2	Rejected	2	Rejected
0	0	200	0	200	0	200
1	0	200	0	200	0	200
2	182	18	188	12	157	43
3	0	200	0	200	0	200
4	0	200	0	200	0	200
5	0	200	0	200	0	200
6	0	200	0	200	0	200
7	0	200	0	200	0	200
8	0	200	0	200	0	200
9	0	200	0	200	0	200
Errors	0		0		0	

Table 29: Partial Performance Matrix on CENPARMI Sets A, B, T

However there is another avenue to improve this figure. Indeed the number of samples in the CENPARMI database is relatively small. As a consequence, the number of digits in several model files is quite low; and since the recognition rules are tightly tailored to the available data, it is hypothesized that the ranges of acceptable values for some feature attributes are probably too narrow. This causes unnecessary rejections in the test set. One way of verifying and correcting this is to apply our partial recognizer to other data and to relax some rules to accommodate rejected samples which should belong to the same models. Occasionally, relaxing rules might cause substitutions; this would reveal weaknesses of our classifier which are presently unknown and which could then be corrected.

7.5.5 Relaxing Rules With CEDAR Training Data

The relaxation of classification rules is carried out using 13,954 samples of the CEDAR database regrouped in 7 sets of 2,000 samples¹², labeled *cedar1*, *cedar2*, etc... Each set includes 10 files, one per numeral class, each containing 200 run-length-coded binary

¹² Except for the last set which has 1,954 samples.

images¹³. These files are named *cedar*'*i*'-'*d*'.*rlc*, with '*i*' ∈ {1 .. 7} and '*d*' ∈ {0 .. 9}.

Relaxing Rules With Samples of File *cedar1-2.rlc*

We begin with file *cedar1-2.rlc*. Initially, when the partial recognizer is applied to these 200 '2's, 104 are properly recognized and 96 are rejected. Thus the recognition rate is only 52%... The process of relaxing classification rules is carried out as follows:

- The output file produced by the recognizer, *cedar1-2.ids*, is processed by a small utility program which creates a model file containing only the rejected samples and other model files of misclassified samples, if any. In addition to their regular content, these model files also contain, for each sample, the *classification label* produced by the recognizer.
- Using these model files as input, rejected samples can be viewed individually with our development interface. As usual, their image is displayed in the drawing area; but in addition to their feature list, their classification label is also written to the scrolled text area. Taking into account the shape of the image, the sequence in which rule files are known to be fired in the classification program and the classification label of the rejected sample, it is easy to pinpoint which rule file should have accommodated this sample but failed to do so. The function *APPLY_RULE* can be edited to report the exact rule which failed when that particular sample and that rule file was processed; furthermore, the value of the tested attribute at that point can also be displayed, indicating in what way the range of acceptable values should be modified.
- A misclassified sample, if any, is processed in a similar fashion. Visual inspection reveals that the shape is indeed similar to that of the 'wrong' class in certain portions of the contour, justifying the successful firing of a number of rule files for that wrong class. But it also allows the user to see where the shape of the misclassified sample is notably different from the class it was assigned to; this pinpoints a specific rule file in which existing loopholes have just been revealed and must be corrected.

¹³ The last file *cedar7-9.rlc* contains only 154 '9's.

After processing the 96 rejected samples of file *cedar1-2.ids* in the manner just described, 149 samples are correctly recognized (74.5%) and 51 are still rejected.

Applying Variable Amounts of Smoothing

The number of rejections remains relatively large and observation reveals that almost all rejected samples have small noise features along their contour. The problem is much more important than for the CENPARMI database. Up until that point in the development process, all contours had been smoothed with a triangular filter of width $w = 5$ before the extraction of features. Noting that the CEDAR database resolution was 300 PPI¹⁴, it was decided to investigate whether applying more smoothing to samples of 'larger size' (in terms of their number of rows, which increases in direct proportion to data resolution) could help solve the problem. This extra smoothing would have 2 advantages: firstly, to eliminate many small noise features, thus favoring more recognition during the first pass; and secondly, to attenuate other noise features not eliminated which would facilitate their removal by later feature filtering.

File	Uniform smoothing $w = 5$		Variable smoothing $w = 5, 7, \text{ or } 9$	
	Recognized	Rejected	Recognized	Rejected
a-2.rlc	183	17	184	16
b-2.rlc	188	12	188	12
cedar1-2.rlc	149	51	164	36
cedar2-2.rlc	125	75	148	52
cedar3-2.rlc	119	81	134	66
cedar4-2.rlc	121	79	152	48
cedar5-2.rlc	126	74	143	57
cedar6-2.rlc	126	74	147	53
cedar7-2.rlc	136	64	143	57

Table 30: Recognition of '2's Depending on Smoothing

Several experiments were carried out, with different numbers of categories and different thresholds for these categories. The best results were obtained for the variable

¹⁴ Much higher than the 166 PPI of the CENPARMI database.

amounts of smoothing already presented in Section 6.4.1. Table 30 shows the improvement in recognition rate obtained for the 1400 '2's of the *cedar1* through *cedar7* datasets. For comparison, the results are also presented for CENPARMI training sets A and B. For these, only one more sample is recognized by applying variable amounts of smoothing. For the CEDAR '2's however, the gain is much more significant: on average, the recognition rate increases from 64.4% to 73.6%.



Figure 76: A '7' Misclassified as a '2'

Finally, we note that during the relaxation process with samples from file *cedar1-2.rlc* and the experiments with variable amounts of smoothing, 2 misclassified samples were found. Both are '7's; the first one was already shown in Figure 73 (e) and the second one is displayed in Figure 76. Both errors were corrected by developing tighter rules concerning the lower portion of '2's without a bottom downward-pointing cavity feature.

Relaxing Rules With Samples of File *cedar2-2.rlc*

A second file of 2's from the CEDAR database, *cedar2-2.rlc*, was processed to relax classification rules. The results are shown in Table 31; note that all samples of all other classes were rejected correctly. The number of correctly recognized samples from the file *cedar2-2.rlc* increased from 148 to 162, representing a gain in recognition rate of 7%. The increments in number of samples recognized are much lower for the other files of '2's. The same was observed when relaxation of rules was first investigated with file *cedar1-2.rlc*. This suggests that it was mostly different classification rules

which needed relaxation in those 2 cases; thus fairly large training sets are required to cover all rules which might need amendment. Clearly, further improvement could be achieved with file *cedar3-2.rlc*; also, for other classes, it was decided to always attempt relaxation next with the file having the lowest count of recognized samples, instead of processing them consecutively as was done here.

File	Recognized	Increment	Rejected
a-2.rlc	185	+1	15
b-2.rlc	188	0	12
cedar1-2.rlc	166	+2	34
cedar2-2.rlc	162	+14	38
cedar3-2.rlc	135	+1	65
cedar4-2.rlc	154	+2	46
cedar5-2.rlc	145	+2	55
cedar6-2.rlc	151	+4	49
cedar7-2.rlc	146	+3	54

Table 31: Recognized Samples After Relaxing With *cedar2-2.rlc*

Finally, it is interesting to note what has been gained so far concerning the recognition of CENPARMI testing set T. At this stage, 168 samples of '2' are correctly classified and the other 32 are rejected. All 1800 samples of other classes are properly rejected. Thus the recognition rate for '2's has increased from 78.5% to 84%.

Chapter 8

Overall System & Results

The general design of our recognition system was presented at the end of Chapter 3. The present chapter will deal with a number of issues related to this design which have not yet been discussed. Results will also be presented. For technical information concerning program size and execution time, see Appendix I.

After preprocessing (including contour extraction) and feature extraction, numerals composed of several connected components undergo further processing aimed at reconnecting the pieces together and/or discarding tiny pieces; this is explained in Section 8.1. Then tiny, spurious holes are filtered out; this is presented in Section 8.2. After this point, classification is attempted only if the sample consists of a single connected component. The special 2-component classification module at the center of Figure 3 was not implemented since it is meant only for certain styles of '5' (mostly), '4', and '1' and our partial classifier does not deal with these classes. The first recognition pass is discussed in Section 8.3.

If the sample is not recognized, there may be two more attempts at classification. First, various filtering operations are performed on its feature list; if any modification results from this, another recognition pass is applied. Second, for samples with at least one hole, if the filtering operations did not modify the numeral's feature list or if it still has not recognized after the second pass, another recognition pass may be tried after dropping a small hole. Sections 8.4 and 8.5 examine these aspects.

Finally, Section Section 8.7 presents the results achieved by our system.

8.1 Processing Multi-Component Samples

The CENPARMI database contains only single-component samples. But since we have also used other databases to relax classification rules and to improve their reliability, it was necessary to deal with multi-component numerals (i.e. digits broken in several pieces). For example, 3.43% of the 13 954 training samples used from the CEDAR database are broken into 2 or more pieces; and the 5000-sample free-style training set of the Concordia-Montreal database (the so-called *e-set*) has 4.82% of its samples which are broken in 2 or more pieces. For the 4000-sample training set of the ITRI-Taiwan database (sets *tw1*, *tw2*, *tw3*, and *tw4*), this percentage increases to 10.20%; and for the larger 24 427-sample training set of the ITRI-Taiwan database (sets *t20* to *t24*), it reaches 13.33%. See Appendix J for a detailed breakdown of these figures depending on the numeral class and the actual number of broken pieces.

Solutions to the problem of broken digits were developed based on a collection of 255 samples extracted from the CEDAR (300 PPI) and the Concordia-Montreal (200 PPI) databases. The primary goal is to reconnect the disjoint pieces together to recover a single-component image. But there are also tiny fragments which are simply discarded; in addition, the specific weakness of the feature extractor in relation to small blobs is addressed at this point, because it is important for proper reconnection. The general approach followed is to modify the binary image and then to submit the amended digit to the feature extractor once again.

8.1.1 Deleting Tiny Pieces

When the number of pixels making up a small piece is no more than the stroke width, the tiny piece is deleted: its pixels are re-written as background pixels and the number of pieces is decreased by 1.

The 2 images of Figure 77 (a) show examples of such tiny blobs which are removed. The image on the left has 3 pieces; two of them are important for recognition purposes¹

¹ Note that for 2-component images, the design of our recognition system prescribes a classification attempt first; only when this fails to recognize certain styles of '5', '4' (as here), or '1', will the reconnection of broken pieces be attempted.

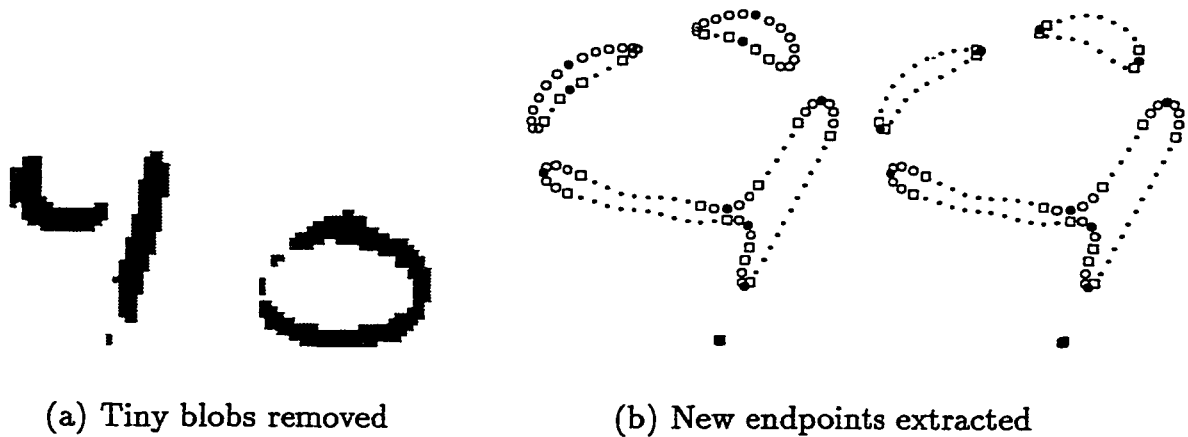


Figure 77: Handling Small Blobs

and the third is a tiny insignificant fragment which is removed. For the second image of Figure 77 (a), removal of the 2 tiny blobs is not as positive as in the preceding case, since they appear to be part of the original stroke of the writer; their erasure causes the zero to be opened more widely than was initially the case. Removal of tiny blobs affected 22 of the 255 samples in this experiment. Not removing them before the reconnection attempts of section 8.1.3 was also tried. In many cases, it made no real difference; in some cases, it improved the final result but in others it created noise spikes on the outer contour. In the final evaluation, there seemed to be no net advantage in keeping them.

8.1.2 Extracting New Endpoints from Small Pieces

When it is not due to the writing style, the presence of more than one piece in a digit image is generally the result of short breaks along the pen stroke, caused by binarization thresholds (converting fainter regions into background) or writing accidents. Thus we would expect most pieces to be elongated, with endpoints at both extremities. In its simplest version, reconnection could then be based on detecting pairs of nearby endpoints ‘aimed’ at each other.

At the end of Chapter 6, it was said that the extraction of features from small

blobs is at times problematic with our new feature extractor. Another example is given in the first image of Figure 77 (b). This digit is broken into 4 pieces and for the 2 fragments at the top, no endpoints were extracted, only a very large bend (encompassing both extremities of the fragment and the convex region between them) and a small cavity. Not detecting the extremities of the broken pieces could hinder proper reconnection. In order to solve this problem, the system examines all but the largest piece to ensure that each of them has at least 2 endpoints. Any blob with fewer than 2 endpoints is processed as follows:

- The feature list for the blob is dropped;
- A first endpoint region is determined: if the number of contour points is less than 12, the region is limited to a single point where the value of ϕ_j is maximum; for blobs with 12 or more contour points, a 3-point region is chosen, where $(\phi_{j-1} + \phi_j + \phi_{j+1})$ is maximum.
- A second endpoint is chosen, half way further around the contour.

The fact that other possibly significant features on the smaller pieces are dropped is not a problem since the immediate goal is the reconnection of the disjoint pieces; if this operation succeeds, contour and feature extraction will be performed again on the modified binary image. The last image of Figure 77 (b) shows the result of this endpoint extraction on the 2 top fragments. Among the 255 samples of this experiment, 26 had new endpoint extraction performed on them.

8.1.3 Reconnecting the Pieces Together

If the number of disjoint blobs in the image is denoted by n_pieces , the number of connections to make in order to obtain a single-component image is $(n_pieces - 1)$. In our system, the reconstruction of broken digits is carried out in two main stages. In the first stage, the best locations for the connections are found and as each site is found, it is tested to verify if such a connection appears safe and sound; if so, the appropriate information is recorded and the next connection is sought and tested; if not, the reconnection attempts cease. No reconstruction of the image takes place

during this first stage. In the second stage, the system verifies that the area of the pieces which would still be disconnected (if any) is relatively unimportant. If this is indeed the situation, the connections are actually performed on the binary image itself; otherwise, the reconnection process is considered to have failed. We now explain the first phase in more technical details:

- The method starts with the largest piece and considers each of its non-cavity features in turn². For each non-cavity feature of the largest piece, focal-point distances are measured to each non-cavity feature of all the other pieces. The shortest and second-shortest distances are recorded, with indications concerning what pieces and what features on these pieces would be connected. The following tests are then applied to the potential connection:
 - The straight-line path joining the two focal points must be free from object pixels; this is to avoid connections across an already existing stroke. If the path is clear, and one of the two features involved is a large enough Bend region (with a number of points exceeding 40% of the maximum dimension of the image), the focal point of the Bend might be relocated so that it is closer to the other focal point to which it could be connected. If on the other hand the path is not clear and the distance is less than ($3 \times \textit{stroke_width}$) and the pixel area of the yet unconnected piece represents more than 15% of the total blob area of the numeral, an attempt is made to find other points in the same feature regions which would be even closer; when such points are found, another ‘clear path’ verification is made.
 - If the inter-focal point distance is less than $2.15 \times \textit{stroke_width}$, the connection is immediately considered safe and sound. Otherwise, two more tests are carried out. The first one verifies that the distance separating the two focal points is less than 1.26 times the largest dimension of the piece to be connected and that one of the two features involved points in the direction of the other (with a 0.8 radian tolerance). If this is satisfied, the connection is also considered safe and sound. Otherwise, the second test

² Connections are considered possible only between E-E and E-B feature pairs.

performs the same verifications but this time with the features involved in the second-best distance.

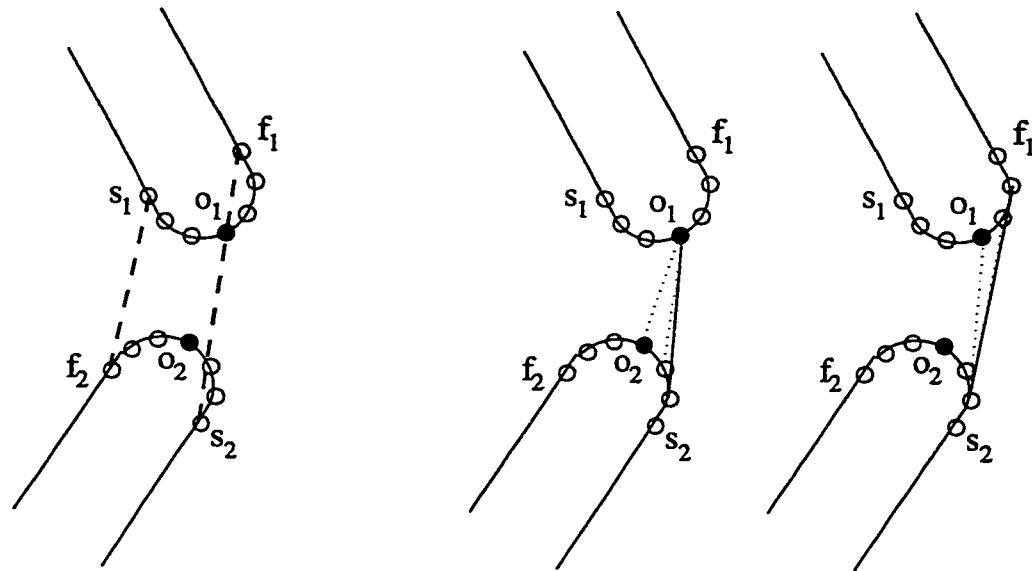
If the potential connection satisfies the above tests, the pieces involved are marked as connected (even though they are not really, at this point) and the method proceeds to the next step; otherwise, the search for feature pairs to connect ends.

- In the second step, each non-cavity feature of the 2 'connected' pieces is considered in turn; focal-point distances are computed to each non-cavity feature of all but these two pieces. Again the shortest and second-shortest distances are recorded, with indications concerning what pieces and what features on these pieces would be connected. The tests just described are also applied to this second potential connection (and perhaps to its 'second-best' substitute) and, if successful, the method marks this third piece as connected and proceeds to the next step.
- In the third step, the same approach is applied using focal-point distances from each non-cavity feature of the 3 'connected' pieces to each non-cavity feature of all but these 3 pieces...
- The first phase ends as soon as a safe and sound connection can no longer be found or when $(n_pieces - 1)$ such connections have been found.

When the first phase is completed, the sum of the pixel areas of the pieces with safe and sound connections between them is computed. If this sum exceeds 92% of the total blob area of the image, reconstruction of the binary image will be undertaken as follows:

- Small and left out pieces (if any) are erased by converting each of their object pixel to the background value.
- The gaps between feature pairs to be reconnected must be filled with object pixels to construct a single-component image. This is done by finding 2 proper line segments to delimit the region to be filled and then by setting each pixel

in this region to the object value. The situation is illustrated in Figure 78 (a) where it can be seen that simply joining the starting and final contour points of the feature regions does not in general provide acceptable segments for our purpose. A nice technique was developed which takes into account the stroke width of the numeral:



(a) Incorrect solution

(b) Delimiting region to fill on 'right' side

Figure 78: Delimiting Region to be Filled

- To find the first point on the 'right' side, the segment joining the focal points o_1 and o_2 is first used as a reference; the end of that segment is moved from o_2 to the preceding contour points (towards s_2) as long as the perpendicular distance from o_2 to the new segment increases and no more than $(stroke_width/2)$ times. This results in the solid line segment in the left illustration of Figure 78 (b).
- To set the second point on the 'right' side, the other end is moved to contour points following o_1 (towards f_1) as long as the perpendicular distance from o_1 to the new segment increases and, again, no more than

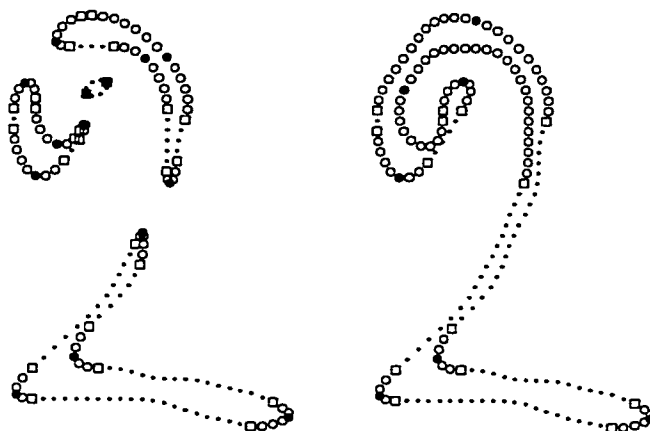
(*stroke_width/2*) times. This finally results in the solid line segment illustrated in the right portion of Figure 78 (b).

- A similar approach is applied on the ‘left’ side to obtain another appropriate delimiting segment.
- All pixels between the 2 segments are changed to object pixels.

When broken digit reconnection succeeds, the reconstructed binary image is processed again from the beginning: preprocessing, edge and contour extraction as well as feature extraction are carried out once more.

8.1.4 Evaluation

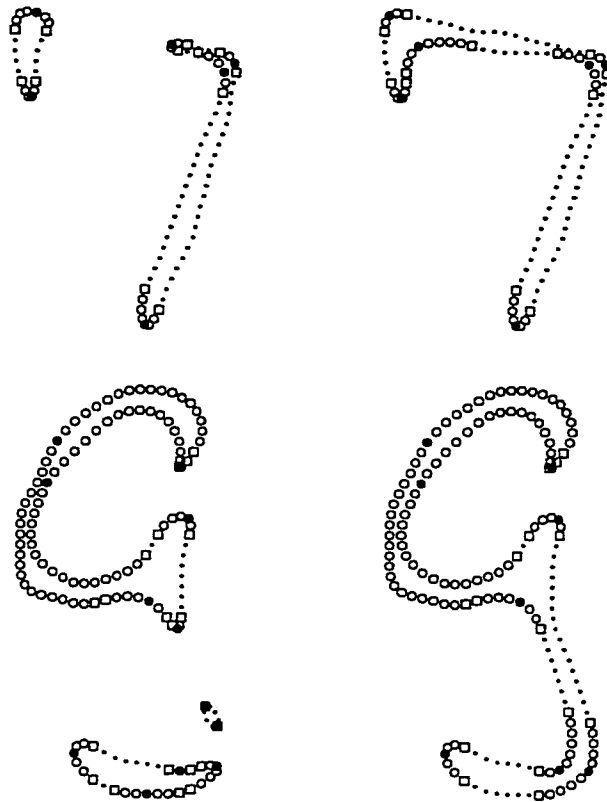
The broken digit reconnection scheme produces very good results. In general, most pieces are reconnected together through End-End connections. We begin by showing a few examples of the numerals where these predominant reconnections work very well.



(a) Broken numeral (b) Reconnected numeral

Figure 79: Reconnecting Broken Pieces: E-E Connections (Case 1)

In Figure 79, we have a ‘2’ broken in 4 pieces which is very nicely reconnected. In Figure 80, we have a ‘7’ broken in 2 pieces which are fairly distant, and a ‘9’ broken in

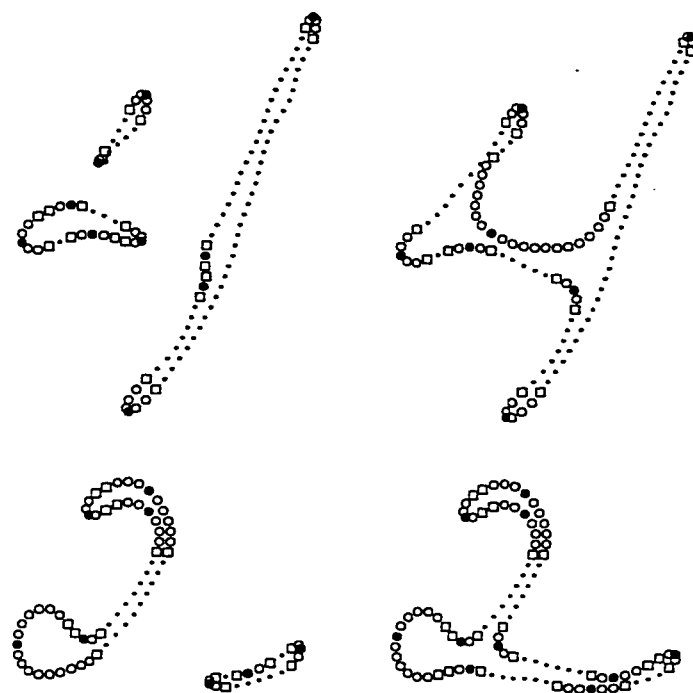


(a) Broken numerals (b) Reconnected numerals

Figure 80: Reconnecting Broken Pieces: E-E Connections (Cases 2 & 3)

3 pieces. Again, the reconstruction is successful. Given that the gap separating pieces in Figure 80 (and also further, in the bottom sample of Figure 81) is quite large, it might be thought that such reconnection could create problems in other situations. Here we have operated under the assumption that all fragments belong to a single numeral; in this context, the only factor to be considered is whether the reconnection makes sense or not and larger gaps can indeed be bridged safely. Of course, in the case of numeral strings or cursive script, when fragments can belong to neighbouring digits or characters, one must be much more cautious. Our reconnection scheme could still be very useful but the distance conditions (and possibly the direction conditions as well) should be tightened in an important way.

Even though they are less frequent, End-Bend reconnections between broken



(a) Broken numerals (b) Reconnected numerals

Figure 81: Reconnecting Broken Pieces: E-B Connections (Cases 1 & 2)

pieces also play an important role and are very successful as well. Two cases are shown in Figure 81. In the top case, the numeral is broken in 3 pieces and both reconnections involve a pair of End-Bend features. In this case, both bends are relatively small regions and its focal point was not displaced before reconnection. In the bottom case, we see that the focal point of the bend feature encircling the hole is located on the left side; because this bend is relatively large, its focal point was relocated before the reconnection took place: it was moved completely on the other side much closer to the broken piece's left endpoint. It is at this location that the reconnection takes place in quite a nice way. Thus broken digits with holes can also be reconstructed correctly.

Finally, Figure 82 provides two examples where reconnection is not attempted because the distance between broken pieces is too large (relative to the maximum dimension of the piece to be reconnected). However, the distant piece's area is less

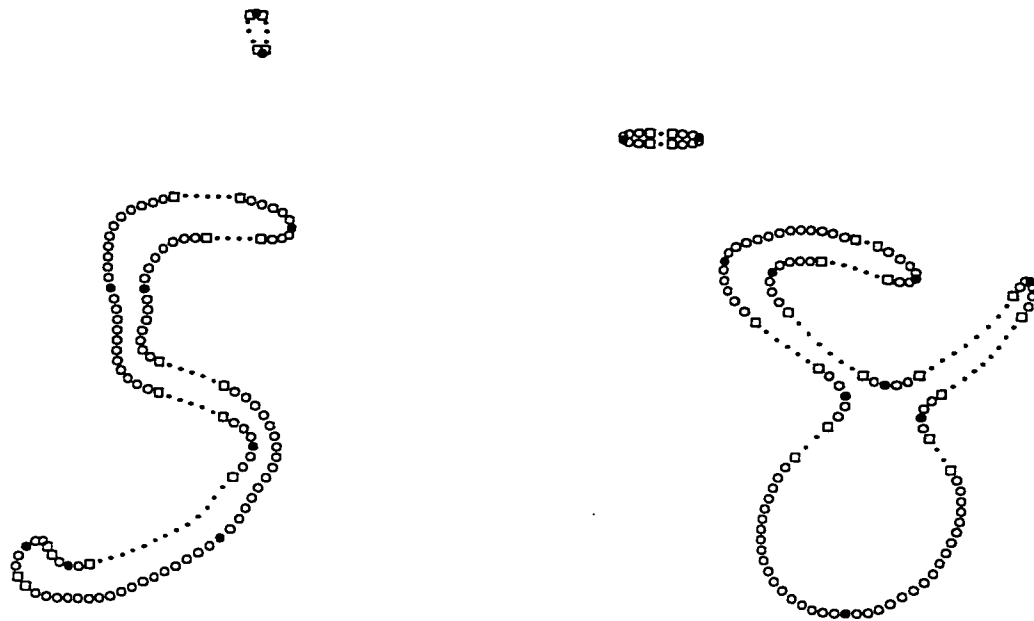


Figure 82: Reconnecting Broken Pieces: Too Distant Pieces

than 8% of both pieces' combined area and it is simply dropped; the dimensions of the image are re-adjusted to those of the largest piece.

Overall, for the 255 samples composing the material of this experiment, the handling of the broken pieces is completely satisfactory in 235 (or 92%) of the cases: either the reconnection is complete or small pieces (making up less than 8% of the total area) are dropped rather than making awkward connections. In all these cases, correct classification is expected to result from this processing. The fact that the stroke width is taken into account in reconstructing the missing links of the binary image is a nice characteristic of our method which produces nice-looking images and avoids creating spurious curvature features at the connections.

There are 8 samples where the area of the reconnected pieces (or of the largest piece alone, if no reconnection is deemed safe and sound) is less than 92% of the total blob area. This leads to rejection without any classification attempt. In the remaining 12 cases, reconnection is performed but can result in awkward-looking shapes; for 3 of these samples, the problem stems from the poor binary image the system begins with; for 3 more, recognition of 2-piece digits would most likely succeed before reconnection

attempts; finally, for the last 6 samples, poor or wrong reconnection yields unusual shapes which most likely would also be rejected and not lead to errors.

8.2 Filtering Spurious Holes

The presence and position of holes is an important global feature which initially guides our classification method. However, small spurious holes can appear in binary images and prevent this initial procedure from operating properly. This can be the result of local breaks within the stroke width; but it can also be due to strokes coming very close to each other or to other factors. Some spurious holes are shown in Figure 83. Note that the binary images shown are not the original images but those obtained after preprocessing, which may already have filled 1-pixel holes and opened up other small holes.

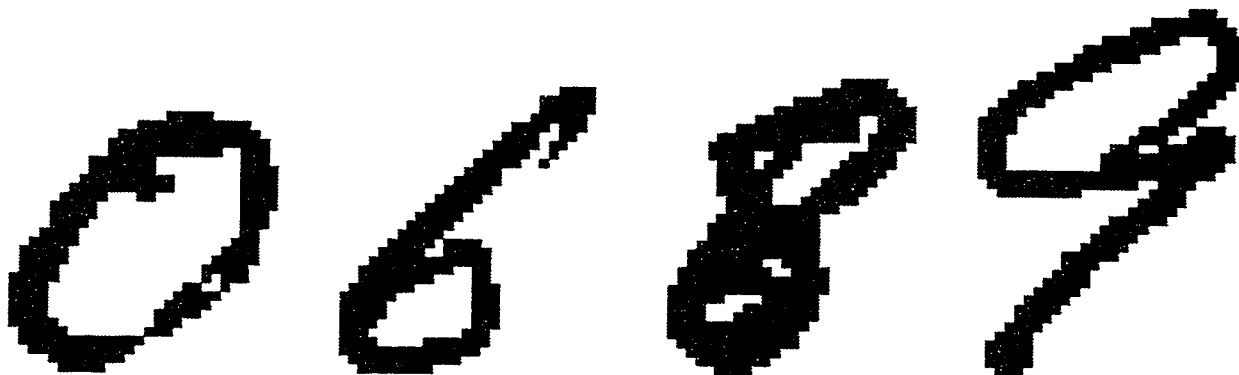


Figure 83: Spurious Holes

The solution cannot simply be to drop all tiny holes because a number of very small holes can still be significant, especially at low resolutions such as 166 PPI (CENPARMI database) and 200 PPI (Concordia-Montreal database). The procedure dealing with this problem in our system operates as follows. First of all, spurious hole filtering is performed only for samples with more than 2 holes, and for samples with 2 holes when the vertical distance between the bottom of the higher hole and the top of the lower hole is smaller than 20% of n_{rows} ; the last condition tries to

avoid getting rid of one of the two typical holes of an '8', when one of them is very small. For samples satisfying these requirements, holes with a pixel area smaller than $1.25 \times \textit{stroke_width}$ are dropped³. But we always make sure to keep at least one hole, even if this last hole might also pass the preceding test.

The guiding notion here is to keep potentially significant holes, even if tiny, because this increases chances of recognition with high reliability. Of course, there is the possibility that such small holes may be insignificant and oddly-located and prevent recognition. Another procedure will assess this possibility, but only *after the first recognition pass*. See Section 8.5.

For CENPARMI training sets A and B, 48 tiny holes are initially filtered out on 45 samples (1.2%) before the first recognition pass. This includes one hole for each of the first 2 images of Figure 83 and two holes for each of the last two images.

8.3 First Recognition Pass

The classifier was developed for single-component digits⁴ and for 4 classes. The classifier begins by setting pointers to the first feature in the feature list and to the preceding feature; then the number of holes is considered:

- For samples without holes, the unknown numeral is tested for classes '2', '9', '0', and '6' in that order⁵. If the *starting feature* rules for class '2' are immediately satisfied with this setting of the feature pointers, the classification process for that class will continue, following the contour, portion by portion. As was seen in Section 7.5, different rule files will be fired, associated with various possible subshapes for each portion. For efficiency reasons, the subshapes are tried in order of decreasing frequency. If classification succeeds immediately for '2', the recognition process ends. Otherwise, the 2 pointers are restored to point again to the first feature in the feature list and to the preceding one and classification as

³ The binary image is not 'corrected'; these holes are simply dropped from their linked list.

⁴ All multi-component digits, even those with 2 components, are first processed for reconnection; the vast majority become single-component digits.

⁵ Which is the order of their decreasing frequency, as samples without holes, in the CENPARMI training sets.

'9' is attempted with this initial configuration. If this succeeds, the recognition process ends. Otherwise, the same is attempted for class '0', and then, if not successful, for class '6'. If these trials fail, the 2 pointers are moved one step ahead in the feature list and the second feature is tested for starting feature, in turn for each of these classes; then the third feature, the fourth, etc. Thus we see that classification will be more efficient if the starting feature selected for each numeral class is always among the first ones in the unknown sample's feature list. When all features in the feature list have been tried out as starting feature for these 4 classes without success, another similar round is performed, this time testing only for rather rare 2's with an opened bottom loop.

- For samples with one hole, the classifier proceeds as follows. Because of its high frequency in this category and because of its simple shapes, all attempts at recognizing class '0' are carried out first. If this fails, the classifier tries to recognize classes '6', '9', and '2' (in that order) using feature pointers as explained above, and trying each feature of the feature list as starting feature for these classes. However, there is one difference: the position of the hole is taken into account. Thus, the numeral with one hole will not be tested for class '6' at all, unless the hole satisfies the following conditions: vertically, the hole must cover less than 66 % of the number of rows; its bottom row must be located in the lower 40% of the image; and the center of the hole must be in the lower half of the image. In the same manner, the variants of '2' with a bottom hole will be tried on the unknown sample only if the position ratio of its hole is larger than 1.24; and the variants of '2' with a top hole will be tried only if the position ratio of its hole is smaller than 0.35. If the hole conditions are satisfied, after a complete unsuccessful round of trying each feature in the feature list as starting feature for classes '6', '9', and '2', another similar round is performed, but this time testing only for rare 2's with a top hole *and* an opened bottom loop.
- After the filtering of spurious holes explained in the preceding section, the numbers of samples with 2 (or more) holes in CENPARMI training sets A and B



Figure 84: Typical Samples With 2 Holes, After Spurious Hole Filtering

are much lower than those recorded in Table 40 of Appendix H. For class '0', of the 10 samples which originally had two holes, only 2 remain; for class '2', 9 samples remain out of the original 12; for class '6', of the 18 samples with two or three holes, 3 samples remain with two holes and 1 with three holes; finally, for class '9', of the original 13 samples, 2 remain with two holes. Typical examples are shown in Figure 84. From the CENPARMI training material in these 4 classes, it is clear that the presence of two holes is mostly due to handwriting style for classes '2' and '6'. For classes '0' and '9', this occurs less frequently and is rather the result of writing accidents⁶. Thus no attempt was made to develop classification rules for zeroes or nines with two holes. Only 2's and 6's were taken into account in this category. If both holes satisfy their respective position ratio conditions, the classifier first tries to recognize a '2' with both a top and a bottom hole by combining the possibilities explored separately in the preceding situation (for digits with 1 hole). If this fails, more severe requirements are imposed on the 2 holes at the bottom of the '6': the *top* of their common bounding box must be in the bottom 40% of the image and the middle must be in the bottom 30%; when this is the case, classification proceeds as in the case with a single bottom hole.

⁶ Of course, slashed zeroes, and even slashed nines, are present in significant numbers in other databases and specific classification rules would then have to be developed.

8.4 Filtering the Feature List

The results of the first recognition pass are presented further in Section 8.7. When the identity of the numeral is still unknown at the end of this first pass, it could be because the feature list includes small noise features obviously not accounted for in any of the shape variants for which rule files were developed. Three filtering operations are performed trying to purge the feature list from these nuisances. If any modification results from these operations, a second recognition pass will be attempted.

8.4.1 Filtering Wiggle-Pairs of Features

The first filtering consists of removing all the features already marked as wiggle-pairs by the feature extractor (if any). See Section 6.4.4 for their definition and Figure 56 for some examples. The cumulative deviation angles associated with the removed features and their preceding inter-arc regions are summed up. If the total exceeds 1.0 and the feature before the wiggle pair is a Bend feature, this last feature incorporates that contour region and its cumulative deviation angle and focal point location are amended correspondingly; otherwise, the sum is simply added to the inter-arc value of the feature following the wiggle-pair.

8.4.2 Filtering Small Convex/Concave Features in Large Concave/Convex Environment

Figures 85 (a) and (b) show small cavity or bend features, 'lost' in a large environment of feature(s) of opposite sign curvature⁷. A filtering operation was developed to remove these features. Special care was taken not to affect the typical middle bends and middle cavities of '3's which are also immersed in opposite sign curvature regions. In order to be filtered out, a small Bend or Cavity feature must meet the following conditions:

⁷ In this figure, only the features of interest and their neighbouring features are shown to highlight the situation discussed.

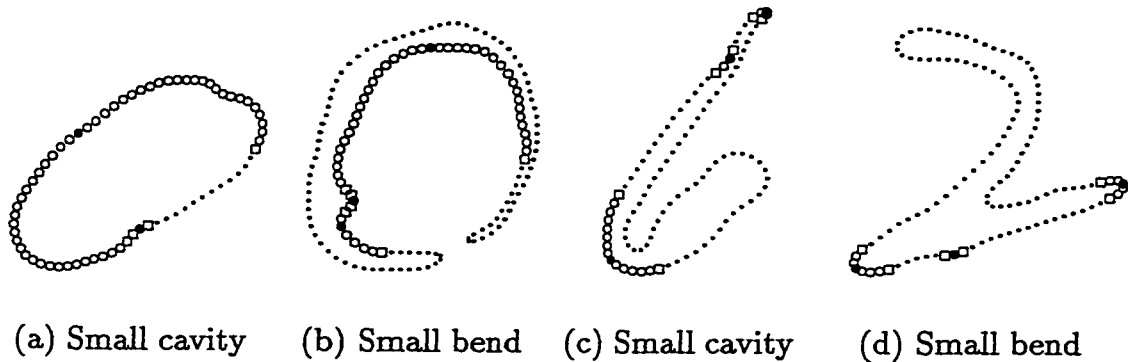


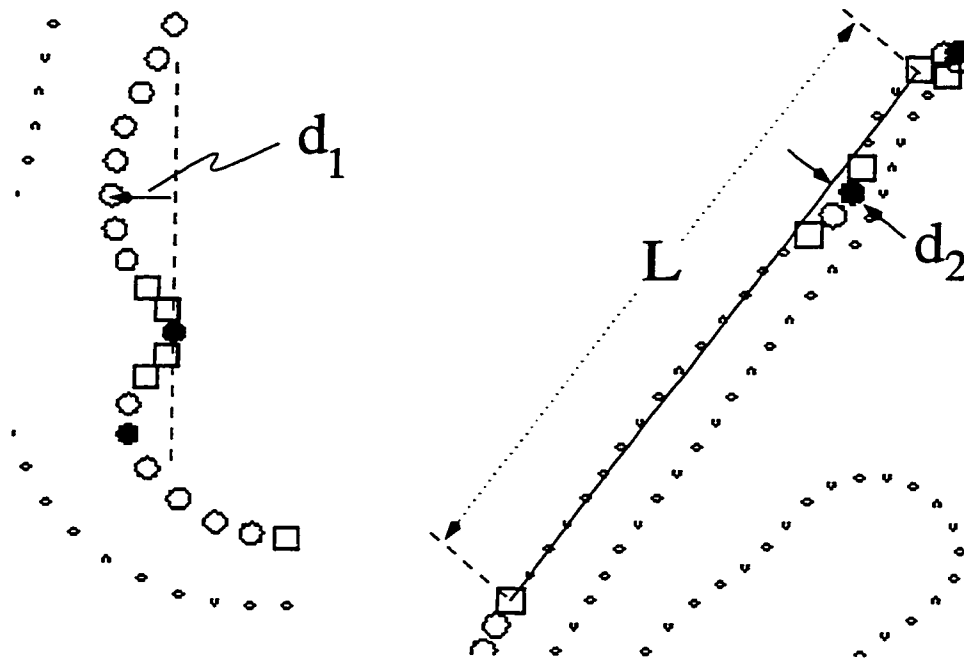
Figure 85: Filtering Out Small Isolated Features

- The feature must be small enough: the absolute value of the cumulative deviation angle for its associated region must be less than 1.6; and the region size must not exceed 10% of max_dim ⁸.
- The feature must have a nearby large sequence of features of opposite sign curvature: The gap separating the feature of interest and that sequence must not be longer than 7.5% of max_dim ; the sequence must cover a contour length larger than 50% of max_dim and its absolute cumulative deviation angle (including possibly that of the feature on the other side of the feature of interest, if it is of the same type as the sequence) must exceed 3.0.
- The feature on the other side of the feature of interest must either be of the same type as the feature sequence or sufficiently far away (a distance exceeding 20% of max_dim).
- The depth of the perturbation caused by the feature of interest in its environment of opposite sign curvature must be relatively small. The situation is illustrated in Figure 86 (a) which displays a magnified view of the vicinity of the bend feature of Figure 85 (b). Through the focus of the feature potentially to be filtered out, a straight line segment is drawn perpendicular to the feature's

⁸ For small Bends, an extra pixel is tolerated.

direction. From the start of the following feature region, distances are computed from every second point to that line segment, as long as these distances do not decrease. The same is done on the other side of the feature of interest. The largest distance is kept and is denoted d_1 in the figure. For cavity features, we must have $d_1 < 7.5\% \times \text{max_dim}$; for bend features, the condition is less stringent: $d_1 < 1 + 10\% \times \text{max_dim}$.

If the feature satisfies the above conditions, it is removed from the feature list; the cumulative deviation angles of its region and preceding inter-arc region are added to the inter-arc value for the following feature.



(a) Second filtering operation

(b) Third filtering operation

Figure 86: Measuring Relative Perturbation Caused by Small Feature

8.4.3 Filtering Less Significant Isolated Features

The third filtering operation removes small cavity and bend features such as those shown in Figures 85 (c) and (d). To be filtered out, the feature of interest must satisfy the following conditions:

- The number of contour points in the feature region must not exceed $1 + 10\% \times max_dim$.
- The portion of contour from the end of the preceding feature to the beginning of the following feature must be more than twice the size of the feature to be removed.
- The elevation of the focal point of the feature of interest above the straight line joining the last point of the preceding feature to the first point of the following feature must be small. The situation is illustrated in Figure 86 (b) which displays a magnified view of the vicinity of the cavity feature of Figure 85 (c). The elevation, denoted d_2 in that figure, must satisfy either $1 + |d_2| < 7.5\% \times max_dim$ or $|d_2| < 0.12 \times L$.

If the feature satisfies the above conditions, it is removed from the feature list; the cumulative deviation angles of its region and preceding inter-arc region are added to the inter-arc value for the following feature.

8.5 Dropping Smallest Hole

Although the filtering of tiny spurious holes discussed in Section 8.2 purges images of most of the unwanted holes, there are still a few cases where such a problem persists and prevents correct classification. Thus after the first recognition pass and, possibly, after a second recognition pass if any feature filtering was successful, a hole filtering operation is performed if the digit is still not recognized. This operation removes the smallest hole from the sample, provided its area is less than 10% of the blob's pixel area and, if there is more than 1 hole, provided also that the smallest hole's area is less than 33% of the largest hole's area.

8.6 Aiming for Higher Reliability

After developing the needed classification rules and the above feature filtering and hole filtering procedures, our recognizer was applied to the CENPARMI A and B sets, and to the *cedar1* through *cedar7* training sets. Very few erroneous classifications were found and changes were brought to the relevant rules to avoid these wrong results. To reinforce the reliability of our classifier even more, it was decided to run it on other databases: thus we used the 5 000 free-format and 5 000 box-format training numerals from the Concordia-Montreal database (sets *e* and *g* respectively); also the 4 000 samples (sets *tw1* to *tw4*) and the 24 467 samples (sets *t20* to *t24*) of both ITRI-Taiwan databases.

The recognition system was applied to these 38 467 samples with a single purpose in mind: find misrecognized samples and bring appropriate changes or additions to the classification rules in order to avoid as many of these errors as possible. Not a single rule was developed to recognize rejected samples. All together, 32 erroneous classifications were found (10 as '0's; 13 as '2's; and 9 as '6's). On average, this amounts to slightly over 10 errors per class for which the classifier was developed. Extrapolating a similar performance for other classes results in 107 errors for 38 467 samples, an error rate of approximately 0.28%. The reliability of our system is indeed quite good on data of various origins. And it is hoped that by correcting most of the 32 errors found, it will be improved even further. We now discuss the misclassifications and the remedies found for them.

The samples which are incorrectly classified as '0' are shown in Figure 87. The sample of Figure 87 (a) was in a file of '4's. Originally, the feature extractor finds 3 bends and 1 cavity on its outer contour; after feature filtering the cavity was discarded, thus causing the substitution in the second recognition pass. The 4 samples in Figure 87 (b) were in files of '6'; the first one revealed a weakness in the classification rules for '6's without a hole; the other 3 images share a common shape; even though the hole is quite high, the top stem should be cause enough to reject the sample. The causes of the 8 \rightarrow 0 substitutions shown in Figure 87 (c) are twofold: the bulkiness of these three '8's in their middle section and the hole filtering operation of Section

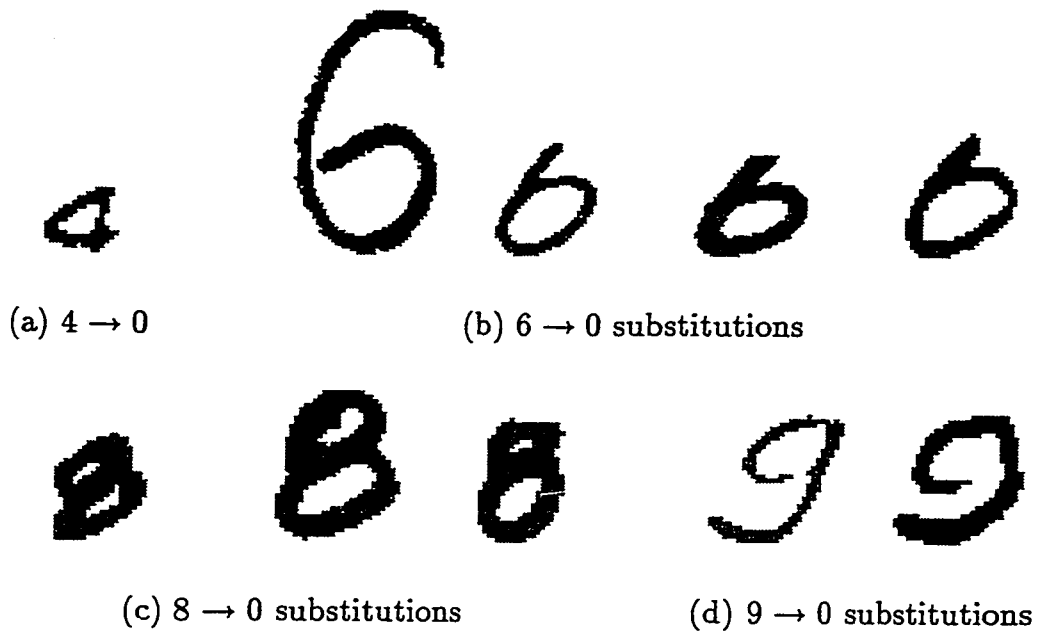


Figure 87: Samples Incorrectly Classified as '0'

8.5. In the first case, both cavities are extracted but one is later filtered out; in the second case, only one cavity is extracted and it is also filtered out; in the last case, no cavities at all are initially extracted. Later, when the top hole is also filtered out, the substitutions occur. It is possible to avoid these substitutions, given the very low position of the top of the bottom hole. The substitutions of Figure 87 (b) also revealed clear shortcomings of the classification rules for '0' which were corrected.

The samples which are incorrectly classified as '2' are shown in Figure 88. The four 1 → 2 substitutions of Figure 88 (a) were corrected based on the presence of a very low cavity feature on the left profile of these numerals; this can also occur on some '2's but when it does, their top-down stroke is much more slanted than for these '1's. Rules were also modified or added to reject the samples of Figure 88 (b); the second case revealed flaws in the rules that had been developed to recognize '2's with an open loop at the bottom. The six 7 → 2 errors of Figure 88 (c) could also be corrected to rejections. This was done by taking into account the characteristics of the stroke in the middle right portion of the image (5 cases out of 6). For the third sample, the straightness of its right profile can also be used to force a rejection.

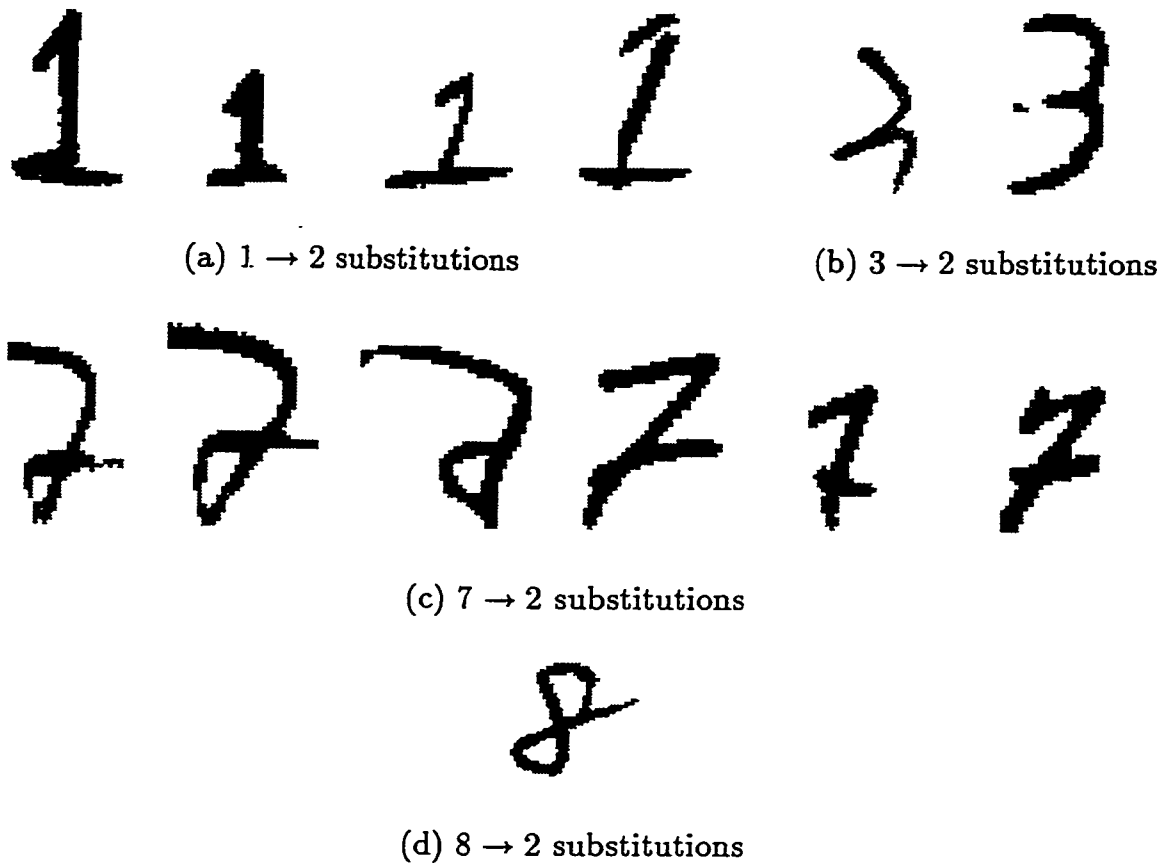


Figure 88: Samples Incorrectly Classified as '2'

Finally, the 8 → 2 substitution of Figure 88 (d) can be prevented: for '2's with a top and a bottom hole, the top hole does not reach this low and the vertical gap between holes is much wider.

Finally, the samples which are incorrectly classified as '6' are shown in Figure 89. The two cases of Figure 89 (a) revealed loopholes in the classification rules for '6' and were easily corrected. The 8 → 6 substitution of Figure 89 (b) was due to the filled hole in the top portion of the '8' and the filtering of the small cavity on the left profile when the first recognition pass failed; the problem can be solved because of the unusual thickness (relative to the *stroke_width* and to *n_cols*) at the level of the middle right cavity. The last six errors are 4 → 6 substitutions. The first one can be avoided because of the unusual height of the hole. Four of the five other

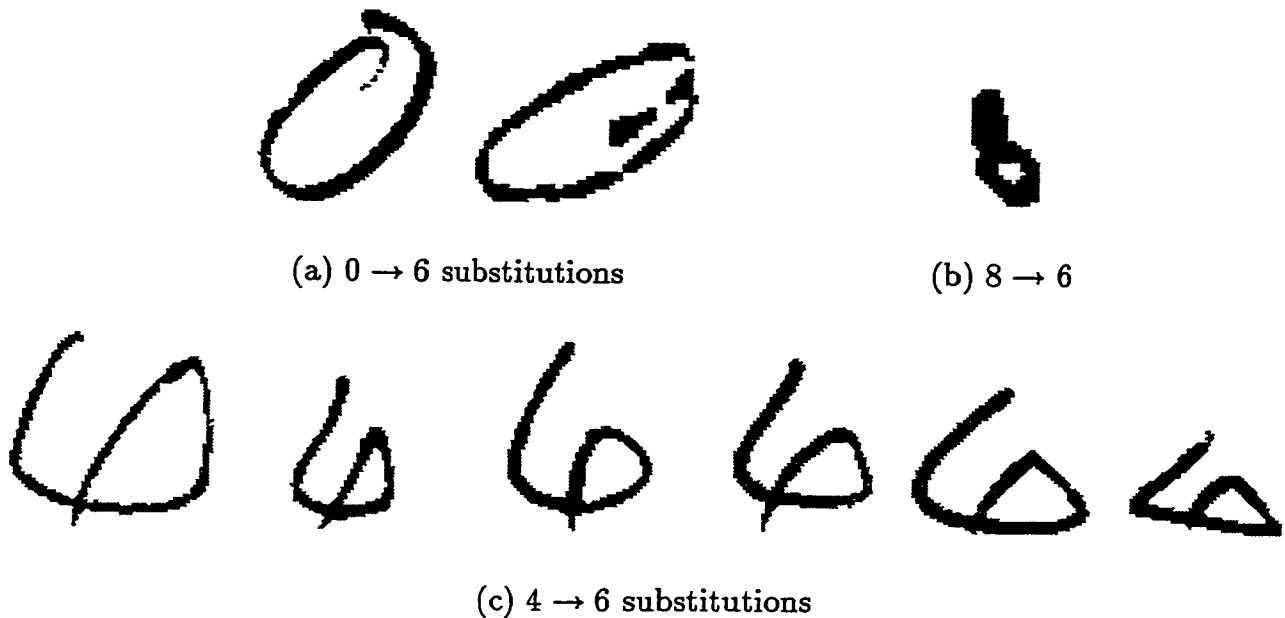


Figure 89: Samples Incorrectly Classified as '6'

misclassifications could be avoided by detecting some characteristics related to the triangular shape of the hole.

Of the 32 substitutions found, 30 were modified to rejections. The only errors remaining are the sample of Figure 87 (a) and the third sample from the left in Figure 89 (c). In general, it was possible to modify or add rules to reject these samples without reducing the recognition rate for true '0's, '2's, and '6's,

We conclude this section with two remarks. First, it is noteworthy that the fact that we have developed only a *partial* classifier tends to increase the error rate: several classes then have no other choice but error or rejection. In particular, some $8 \rightarrow 0$ and $7 \rightarrow 2$ errors are samples whose shapes are quite 'normal' and would have been classified properly in the first recognition pass; it was only after feature filtering that the misclassification occurred. The second remark pertains to our scheme to reconnect (or filter out pieces from) broken numerals. It should be noted that none of the substitutions observed was due to wrong or poor reconnection. Thus our method seems quite reliable.

8.7 Results

Table 32 presents the performance matrix of our recognizer when applied to the 4 000 training samples of the CENPARMI database. Results are presented after the first pass and after the final (second or possibly third) pass. There are of course no substitutions. On average, there is a gain of 3.4% in the recognition rate between the first and the final passes.

in/out	After first pass			After final pass		
	0	2	6	0	2	6
0	379	0	0	393	0	0
1	0	0	0	0	0	0
2	0	374	0	0	381	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	367	0	0	387
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
Rec.	94.75%	93.50%	91.75%	98.25%	95.25%	96.75%

Table 32: Partial Performance Matrix on Combined CENPARMI Sets A and B

in/out	0	2	6
0	183	0	0
1	0	0	0
2	0	173	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	182
7	0	0	0
8	0	0	0
9	0	0	0
Rec.	91.5%	86.5%	91.0%

Table 33: Partial Performance Matrix on CENPARMI Test Set T

Table 33 presents the partial performance matrix for the 2000-sample CENPARMI

test set. The average recognition rate is 89.7% and there is not a single error. This clearly demonstrates the possibility of developing a single-approach recognition system with a good recognition rate and a very high reliability. Of course, the fact that we get no error on this test set does not really mean that the reliability of our system is 100%. The CENPARMI test set is rather small and we need to apply our system to a larger database before drawing firm conclusion about its reliability.

In order to obtain a better assesment, it is possible to apply the system to other data on which it was not trained at all. Results will be provided for test sets from the CEDAR, the Concordia-Montreal and both ITRI-Taiwan databases.

Class in	Total	0	2	6
0	355	319	0	0
1	289	0	0	0
2	224	0	189	0
3	208	0	0	0
4	183	0	0	0
5	117	0	0	0
6	245	0	0	211
7	221	0	0	0
8	191	0	0	0
9	180	0	0	0
Rec. Rate		89.9%	84.4%	86.1%

Table 34: Partial Performance Matrix on CEDAR goodbs Test Set

First, we consider the CEDAR testing material. The *goodbs* set consists of 2 213 well segmented numerals and the *bs* set is a superset of 2 711 samples whose machine-segmented numerals are not always very nice. The sets are unbalanced, so the number of samples in each class will also be provided. Table 34 shows the performance on the well-segmented samples. As expected the recognition rates are lower than that for the CENPARMI data, since there was no real training on this database: it was used only to relax classification rules for shape models of the CENPARMI database. Still the average recognition rate of 86.8% is really quite good. Of course the CEDAR numerals are also extracted from U.S. postal codes. What is more impressive however is that once again, no errors are present.

Class in	Total	0	2	6
0	434	363	0	0
1	345	1	0	0
2	296	1	221	0
3	260	0	0	0
4	234	0	0	0
5	193	0	0	0
6	281	0	0	227
7	241	0	0	0
8	216	0	0	0
9	211	0	0	0
Rec. Rate		83.6%	74.7%	80.8%

Table 35: Partial Performance Matrix on CEDAR *bs* Test Set

Table 35 shows the results on the *bs* test set including several poorly segmented numerals. Of course, the average recognition rate drops to 79.7%. But despite the lower quality of the additional material (compared to the *goodbs* set), 44 more '0's, 32 more '2's, and 16 more '6's are recognized; most probably, the feature filtering operations are able to remove some of the smaller problems caused by poor segmentation. There are however 2 substitutions. When compared to the 811 samples correctly recognized, this yields a 0.25% error rate⁹

The 5000-sample free-format testing set of the Concordia-Montreal database is called the *f*-set. The partial performance matrix of our system on these digits is presented in Table 36. The average recognition rate is 83.9% and there are 9 errors: one '8' and one '9' are misclassified as '0'; three '1's, one '3' and one '7' are misclassified as '2'; two '5's are misclassified as '6'.

Next we examine the results on the ITRI-Taiwan testing sets. This database contains some samples in writing styles which are quite different from those common in the U.S. Table 37 provides results for the combined 3000-sample testing sets *tw5*, *tw6*, and *tw7*. The average recognition rate is now 74.6% and there are 5 errors: one '6' is misclassified as a '0'; two '3's are misclassified as '2's; and two '4's are misclassified as '6's. Relating the 5 errors to the 671 correctly recognized samples

⁹ Dividing the number of misclassified samples by the number of correctly recognized samples overestimates the error rate.

in/out	0	2	6
0	436	0	0
1	0	3	0
2	0	403	0
3	0	1	0
4	0	0	0
5	0	0	2
6	0	0	420
7	0	1	0
8	1	0	0
9	1	0	0
Rec.	87.2%	80.6%	84.0%

Table 36: Partial Performance Matrix on Concordia-Montreal f -Set

yields an estimated error rate of 0.75%.

in/out	0	2	6
0	237	0	0
1	0	0	0
2	0	212	0
3	0	2	0
4	0	0	2
5	0	0	0
6	1	0	222
7	0	0	0
8	0	0	0
9	0	0	0
Rec.	79.0%	70.7%	74.0%

Table 37: Partial Performance Matrix on ITRI-Taiwan tw5-tw7 Sets

We recall that, when first applied to the same 3 000 testing samples, our previous numeral recognizer E4 achieved recognition and substitution rates of 79.83% and 5.03% respectively. This last figure is very much higher than what is shown possible by our partial recognizer. The majority voting combination of experts E1, E2, and E4 (see Section 2.1.1) initially yielded a recognition rate of 74.41% and a substitution rate of 1.60%. This is still higher than our estimated 0.75% rate.

Finally, Table 38 presents our results for 22 024 samples of the t_{25} to t_{29} ITRI-Taiwan testing sets. The average recognition rate is 72.2% and there are 18 errors: two '4's, two '6's and one '8' are misclassified as '0'; two '3's and two '8's are misclassified as '2'; and eight '4's and one '8' are misclassified as '6'. Dividing these errors by the number of correctly recognized samples yields an estimated substitution rate of 0.37%.

Class in	Total	0	2	6
0	2049	1626	0	0
1	2723	0	0	0
2	2741	0	1830	0
3	2473	0	2	0
4	2114	2	0	8
5	1996	0	0	0
6	1964	2	0	1380
7	2186	0	0	0
8	1855	1	2	1
9	1923	0	0	0
Rec. Rate		79.4%	66.8%	70.3%

Table 38: Partial Performance Matrix on ITRI-Taiwan t_{25} - t_{29} Sets

All the results of this section clearly demonstrate that it is possible to achieve high reliability with a single-approach system based on structural features extracted from the contours of numerals. In the final stage of our work, classification rules were developed strictly for shape models with sufficient frequency in the CENPARMI database. These rules were relaxed *for the same shape models* by using some CEDAR training digits. But no attempt was made to develop rules for new shape models found in that database, nor for the TAIWAN data which offers different shapes in significant numbers. When these classification rules are applied to databases other than CENPARMI's, there will of course be a drop in recognition rate; what we wanted to avoid was an important increase in substitution rate. In relation to this, we recall results communicated by Strathy and reported at the end of Chapter 2. When trained on the CEDAR *br*-set, a neural network approach scored recognition and substitution rates of 98.46% and 1.54% respectively of the *goodbs* testing set; but the same system had a 9.68% error rate on 5 000 samples of the ITRI database. Even for a combination

of 3 neural nets trained on the same material, the corresponding forced-recognition error rates were as low as 0.77% on the *goodbs* data but reached 7.14% on the ITRI data.

We can probably obtain a good estimate of the reliability of our classifier on properly segmented digits by considering its combined performance across several data sets. Taking into account the CENPARMI test set T , the CEDAR *goodbs* test set, the free-format Concordia-Montreal test set f , and the ITRI-Taiwan $t25$ to $t29$ tests sets, our partial recognizer correctly recognized 7 352 samples and misclassified 27 others. The ratio of these numbers gives an estimated error rate of 0.37%.

Chapter 9

Conclusion

This thesis began with an in-depth account of recent advances in unconstrained handwritten numeral recognition, conveying the diversity and sophistication of the methods themselves and the approaches to combine them. In a detailed discussion comparing human and computer performance, it was argued that human recognition remains more reliable. Evidences included an experiment on confusing samples conducted by this author and colleagues; the high rejection levels required by even the best methods to achieve very low error rates; visual inspection of samples misclassified by computers; and the very serious drops in reliability which are observed when recognition methods are applied to databases other than those they were trained on.

Our stated goal was to make a contribution towards matching human performance in terms of very high reliability. In recent years, the combination of several recognition methods has been very successful in this area. But we wanted to know if the limits of single methods, particularly structural model-based methods, could be overcome to deliver much more reliable classification on their own. Towards this end, we revisited all stages of the recognition process typical of such approaches: preprocessing, feature extraction, and classification. More specifically, based on previous work, we hypothesized that overcoming weaknesses in the feature extraction stage was key to achieving our goal. Hence, much of our research focused on this problem to reach higher levels of robustness and reliability. Compared to a previous effort at developing a recognition system of the same type (E4), our general approach to the development

of classification rules was also modified strategically. For the selected classes, much care was taken to explicitly identify the (global or partial) shape variants we wanted the system to recognize and to tightly model each of these shape variants in a more refined and exhaustive manner. A special syntax and a development interface were designed to assist in the intricate and time-consuming task of creating the required classification rules.

Results from a partial classifier built on these foundations were provided at the end of the preceding chapter. We believe that they strongly validate our approach. First, the results on the CENPARMI test set indicate the feasibility of creating a single-method numeral recognition system with a high recognition rate (around 90%) and a very low substitution rate. In fact, there were no errors at all for the classes of interest. The great reliability of the classification rules which encode the ‘learned’ shape variants of the CENPARMI data was also demonstrated by applying the system to other databases from North America, as well as from Taiwan where writing styles can be markedly different. All together, averaging over 4 different test sets which include a total of 31 237 samples, the estimated combined substitution rate was found to be 0.37%. If work was carried out to increase the recognition rates on some of these databases, it is likely that the substitution rate would drop even lower.

We now sum up what we feel are original contributions of this research.

9.1 Original Contributions

Overall, the most important contribution is probably the demonstration that very high reliability can be attained together with a high recognition rate, not only by combination of recognition methods, but also by single-method systems. This was accomplished using rule-based classification and structural features, approaches for which interest has dropped significantly in the past few years. The limits of these approaches in dealing with numeral recognition have been pushed further.

Key requirements to reach our goal in the areas of robust, reliable feature extraction and of sophisticated, tightly-crafted classification rules were fulfilled with methods and algorithms which were described in this thesis and are original to our

system. We now go over each stage of the recognition process, listing more specific contributions resulting from our efforts:

- The edge and contour extraction procedures (Chapter 4) used in our method are not new; as indicated they are based on the edge-type concept of Ahmed & Suen [4]. However the notion of *parent edge* which is used to easily unravel the relative nesting of blobs and holes in the image is a novelty. The efficient combination¹ of edge and contour extraction with preprocessing operations to correct several image defects and with many global measurements is also specific to our implementation. In addition, to our knowledge, the methods to estimate the stroke width and to compute blob and hole areas as edges are being chained are also new.
- The detailed analytical investigation of optimal local weighted averaging methods to smooth contours and curves (Chapter 5), based on a simple noise model, is entirely original. It offers insights into the best sets of parameters in view of specific measurements to be made on the smoothed image, and allows to assess the relative performance of other sets of parameters for the same tasks. Experiments were carried out to verify the applicability of the findings to the general situation.
- The new and sophisticated approach to the extraction of curvature features from image contours (Chapter 6) is an important contribution. The resulting feature extractor can be used in any pattern recognition task where such features are deemed useful. We believe it is one of the best around. In addition, we point out the detailed comparative study with other curvature feature extractors, based on a relatively large number of images.
- The tool for rule development based on our feature extractor and its associated syntax (Chapter 7) are also potentially useful by-products of our work. This tool allows the user to visualize the extracted features on any image, and to query data files for images meeting certain conditions; it also provides good support

¹ In a single pass over the binary image.

for the subjective clustering and the trial of potential classification rules etc. It could be of help to develop a rule-based classifier for any application where our feature extractor is adequate.

- The method to reconnect numerals in broken fragments (Chapter 8), which makes use of the information obtained by our feature extractor and reconstructs the missing links based on the stroke width estimate, is also new. It is quite successful and could be used in other applications where initial data capture and processing produce fragmented images. For example, it could be useful in repairing the ‘damage’ resulting from the removal of printed ‘baselines’ on checks and other forms processed by computer.
- Finally, since execution time was also a concern, several implementation tricks, buried in the programming code, are also little original contributions in their own right.

9.2 Future Work

Avenues for future research along the lines of the work presented in this thesis include further development of the classification stage and efforts to automate this aspect.

As a first step, it might be interesting to complete the classifier for other classes based on the CENPARMI training sets. Once this is achieved, similar work could be performed for other databases. Of course, the classification rules can be developed incrementally, based only on the rejected samples and those for which the system is in error; for the latter, the goal is to devise rules to force their rejection without reducing the recognition rate of the ‘erroneous’ class significantly; for the former, the goal is to cluster them according to the new shape variants they offer and to develop new rule files for their description, inserting their use at the proper junction in the classification process. This could be done first for other databases of North American handwriting styles and later for more different databases. In this last case, one could investigate more deeply the problem of fairly close shapes for different classes from samples of different origins, similar to the 4 → 6 substitution examples of Figure 89

(c). To what extent is it possible to prevent these substitutions without affecting the recognition rate adversely. It appears that our attempts of Section 8.6, while they allowed us to solve part of the problem for the training material, did not prevent the re-occurrence of the same kind of substitution for the Taiwan test sets (see Tables 37 and 38).

The automation of part (or all) of the process to generate classification rules is not an easy task. The reason is that our system makes use of a very broad range of information and there is not necessarily a general pattern governing at what point this or that piece of information is used. The human expertise of the developer himself has been the guiding factor. Of course, simple global attempts could be made but the high reliability of the classification rules would most likely be lost and that was the whole purpose of our work! Nevertheless, some avenues are worth investigating. These include:

- Some automation of the clustering procedure: once starting features are marked, it could be possible to specify (in some standard notation to be developed) particular shape variants and their position on the contour; the development interface would then scan all training samples and automatically regroup together in a model file those meeting the specification. The result might still have to be validated by visual inspection.
- Some automation of the rule generation procedure: for each of the shape variants, composed of a small number of consecutive features, it could also be possible to specify a sequence of potential rules related to these features individually² and to relationships between them as well. The model file would then be processed according to these 'instructions' and the corresponding rule file would be generated automatically. Validation by visual inspection of the rule file and possibly by partial recognition experiments could still be required.

Another interesting question is whether the output of our feature extractor could be used as input to train a completely different kind of classifier such as a neural

² For this the syntax developed for classification rules would be a sufficient tool, simply by giving the (initial) feature attribute part of each rule.

network. The difficulty lies in the fact that such input, describing the sequence of curvature features around the contour of a numeral, is, by its very essence, *not* of fixed length. One avenue to explore is to train several such networks, using one for each specific feature sequence. Thus these networks would have different sizes for their input layers. One would have to address the cyclic nature of the feature sequences and determine whether their cyclic permutations are to be considered as the same sequence or not. Such a classifier would only make use of the information initially available at the input layer, contrary to our present method where several additional measurements are computed as classification proceeds. Because of the lower diversity of information used, it is likely that the resulting neural network would not be as reliable; of course, it is also possible that the sophisticated and time-consuming learning process of neural networks would result in a classifier making the most of the more limited, yet still quite rich, information of the original feature list, and equalling the reliability performance...

Finally, another interesting problem would be to investigate the combination of the method proposed in this thesis with other methods. With a fully-developed classification stage, the high reliability of our approach could make it a good candidate as a front-end recognizer and as a validating unit for the proposed cuts of a numeral string segmenter. It could also be given a 'strong vote' in the combination scheme. It is also possible to use our approach with an incomplete classifier to reinforce existing combinations. For example if most errors in some existing combination of methods are from substitutions into a few specific classes, classification rules could be developed to high levels of recognition and reliability only for these few classes and the partial recognizer be given a 'strong vote' for the cases in which it has expertise.

References

- [1] I. Abuhaiba and P. Ahmed. A fuzzy graph theoretic approach to recognize the totally unconstrained handwritten numerals. *Pattern Recognition*, 26(9):1335–1350, 1993.
- [2] A. K. Agrawala and A. V. Kullarni. A sequential approach to the extraction of shape features. *Comp. Graphics and Image Proc.*, 6:538–557, 1977.
- [3] P. Ahmed and C. Y. Suen. Computer recognition of totally unconstrained handwritten zip codes. *International Journal of Pattern Recognition and Artificial Intelligence*, 1(1):1–15, 1987.
- [4] P. Ahmed and C. Y. Suen. Edge classification and extraction of shape features. In *Proc. 7th Int. Conf. on Pattern Recognition*, pages 593–596, Montreal, 1984.
- [5] F. Ali and T. Pavlidis. Syntactic recognition of handwritten numerals. *IEEE Trans. Syst. Man and Cybernet.*, 7:537–541, 1977.
- [6] I. M. Anderson and J. C. Bezdek. Curvature and tangential deflection of discrete arcs: A theory based on the commutator of scatter matrix pairs and its application to vertex detection in planar shape data. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-6:27–40, 1984.
- [7] V. Anh, J. Y. Shi, and H. T. Tsui. Scaling theorems for zero crossings of bandlimited signals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18:309–320, 1996.

- [8] N. Ansari and E. J. Delp. On detecting dominant points. *Pattern Recognition*, 24(5):441–451, 1991.
- [9] N. Ansari and K.-W. Huang. Non-parametric dominant point detection. *Pattern Recognition*, 24(9):849–862, 1991.
- [10] H. Asada and M. Brady. The curvature primal sketch. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8(1):2–14, 1986.
- [11] F. Attneave. Some informational aspects of visual perception. *Psychol. Review*, 61(3):183–193, 1954.
- [12] J. Babaud, A. P. Witkin, M. Baudin, and R. O. Duda. Uniqueness of the gaussian kernel for scale-space filtering. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8(1):26–33, 1986.
- [13] N. I. Badler and C. Dane. The medial axis of a coarse binary image using boundary smoothing. In *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, pages 286–291, 1979.
- [14] R. Bailey and M. Srinath. Orthogonal moment features for use with parametric and non-parametric classifiers. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18(4):389–399, 1996.
- [15] J. A. Bangham, P. D. Ling, and R. Harvey. Scale-space from nonlinear filters. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18:520–528, 1996.
- [16] G. Baptista and K. M. Kulkarni. A high accuracy algorithm for recognition of handwritten numerals. *Pattern Recognition*, 21(4):287–291, 1988.
- [17] M. Beun. A flexible method for automatic reading of handwritten numerals. *Philips Tech. Rev.*, 33:89–101,130–137, 1973.
- [18] H. L. Beus and S. S. H. Tiu. An improved corner detection algorithm based on chain-coded plane curves. *Pattern Recognition*, 20(3):291–296, 1987.

- [19] B. Blesser. Multistage digital filtering utilizing several criteria. *U.S. Patent 4 375 081*, February 1983.
- [20] L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, V. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of classifier methods: A case study in handwritten digit recognition. In *Proc. 12th Int. Conf. on Pattern Recognition*, pages 77–82, Jerusalem, October 1994.
- [21] M. Brady, J. Ponce, A. Yuille, and H. Asada. Describing surfaces. *Comp. Vision, Graphics and Image Proc.*, 32:1–28, 1985.
- [22] R. M. Brown, T. H. Fay, and C. L. Walker. Handprinted symbol recognition system. *Pattern Recognition*, 21(2):91–118, 1988.
- [23] J. F. Canny. A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.
- [24] J. Cao, M. Ahmadi, and M. Shridhar. A hierarchical neural network architecture for handwritten numeral recognition. *Pattern Recognition*, 30(2):289–294, 1997.
- [25] J. Cao, M. Ahmadi, and M. Shridhar. Recognition of handwritten numerals with multiple feature and multistage classifier. *Pattern Recognition*, 28(2):153–160, 1995.
- [26] D. W. Capson. An improved algorithm for the sequential extraction of boundaries from a raster scan. *Comp. Vision, Graphics and Image Proc.*, 28:109–125, 1984.
- [27] R. Cederberg. An iterative algorithm for angle detection on digital curves. In *Proc. 4th Int. Conf. on Pattern Recognition*, pages 576–578, Kyoto, Japan, November 1978.
- [28] M.-H. Chen and R. Chin. Partial smoothing splines for noisy boundaries with corners. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-15(11):1208–1216, 1993.

- [29] M. H. Chen and P. F. Yan. A multiscale approach based upon morphological filtering. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-11:694–700, 1989.
- [30] F. Cheng and W. Hsu. Parallel algorithm for corner finding on digital curves. *Pattern Recognition Letters*, 8:47–53, 1988.
- [31] Z. Chi, J. Wu, and H. Yan. Handwritten numeral recognition using self-organizing maps and fuzzy rules. *Pattern Recognition*, 28(1):59–66, 1995.
- [32] S.-B. Cho. Neural-network classifiers for recognizing totally unconstrained handwritten numerals. *IEEE Trans. on Neural Networks*, 8(1):43–53, 1997.
- [33] E. Cohen, J. J. Hull, and S. N. Srihari. Understanding handwritten text in a structured environment: Determining zip codes from addresses. *Int. Journal of PR and AI*, 5(1,2):221–264, 1991.
- [34] A. Y. Commike and J. J. Hull. Rule learning for syntactic pattern recognition. In *Proc. 4th U.S. Postal Service Advanced Technology Conf.*, pages 621–633, November 1990.
- [35] D. D’Amato, L. Pintsov, H. Koay, D. Stone, J. Tan, K. Tuttle, and D. Buck. High speed pattern recognition system for alphanumeric handprinted characters. In *Proc. IEEE Comp. Society Conf. on Pattern Recog. and Image Proc.*, pages 165–170, Las Vegas, Nevada, June 1982.
- [36] W. de Waard. An optimized distance method for character recognition. *Pattern Recognition Letters*, 16:499–506, 1995.
- [37] K. Deguchi. Multi-scale curvatures for contour feature extraction. In *Proc. 9th Int. Conf. on Pattern Recognition*, pages 1113–1115, Rome, November 1988.
- [38] J. D. Dessimoz. Curve smoothing for improved feature extraction from digitized pictures. *Signal Processing*, 1:205–210, 1979.

- [39] A. R. Dill, M. D. Levine, and P. B. Noble. Multiple resolution skeletons. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-9(4):495–503, 1987.
- [40] M. Doros. Algorithms for generation of discrete circles, rings, and disks. *Comp. Graphics and Image Proc.*, 10:366–371, 1979.
- [41] H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *Int. Journal of PR and AI*, 7:705–720, 1993.
- [42] R. A. Duderstadt, M. J. Cykana, A. V. Monfared, and D. H. Gibbs. Isolated word recognition for postal address processing. In *Proc. 4th U.S. Postal Service Advanced Technology Conf.*, pages 233–245, November 1990.
- [43] B. Duerr, W. Haettich, H. Tropf, and G. Winkler. A combination of statistical and syntactical pattern recognition applied to classification of unconstrained handwritten numerals. *Pattern Recognition*, 12:189–199, 1980.
- [44] M. J. Eccles, M. P. C. McQueen, and D. Rosen. Analysis of the digitized boundaries of planar objects. *Pattern Recognition*, 9:31–42, 1977.
- [45] T. J. Ellis, D. Proffitt, D. Rosen, and W. Rutkowski. Measurements of the lengths of digitized curved lines. *Comp. Vision, Graphics and Image Proc.*, 10:333–347, 1979.
- [46] J. Franke. Experiments on the cenparmi data set with different structured classifiers. In *Proc. 5th U.S. Postal Service Advanced Technology Conf.*, pages A167–A181, Nov. 30 – Dec. 2 1992.
- [47] J. Franke. *Experiments with the KH30F2 Statistical Data Base Using Different Structured Classifiers and Testing with the BS Data Set from SUNY (USPS)*. Technical Report, AEG Daimler-Benz, Ulm, Germany, February 1992.
- [48] J. Franke. On the functional classifier. In *Proc. 1st Int. Conf. on Document Analysis and Recognition*, pages 481–489, St-Malo, France, September 1991.

- [49] J. Franke, L. Lam, R. Legault, C. Nadal, and C. Y. Suen. Experiments with the cenparmi database combining different classification approaches. In *Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition*, pages 305–311, Buffalo, N.Y., May 1993.
- [50] H. Freeman. Computer processing of line drawing images. *Computer Surveys*, 6:57–97, 1974.
- [51] H. Freeman. On the digital computer classification of geometric line patterns. In *Proc. Nat. Electronics Conf.*, Volume 18, pages 312–324, 1962.
- [52] H. Freeman. On the encoding of arbitrary geometric configurations. *IRE Trans. Electron. Comput.*, EC-10:260–268, 1961.
- [53] H. Freeman and L. S. Davis. A corner finding algorithm for chain-coded curves. *IEEE Transactions on Computers*, C-26:297–303, 1977.
- [54] P. D. Gader, D. Hepp, B. Forester, T. Peurach, and B. T. Mitchell. Pipelined systems for recognition of handwritten digits in USPS zip codes. In *Proc. 4th U.S. Postal Service Advanced Technology Conf.*, pages 539–548, November 1990.
- [55] P. D. Gader and M. A. Khabu. Automatic feature generation for handwritten digit recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18(12):1256–1261, 1996.
- [56] G. Gallus and P. W. Neurath. Improved computer chromosome analysis incorporating preprocessing and boundary analysis. *Physics in Medecine and Biology*, 15(3):435–445, 1970.
- [57] J. Geist, R. Wilkinson, S. Janet, P. Grother, B. Hammond, N. Larsen, R. Klear, M. Matsko, C. Burges, R. Creecy, J. Hull, T. Vogl, and C. Wilson. *The Second Census Optical Character Recognition Systems Conference, Tech. Report # NIST 5452*. Technical Report, Nat. Inst. of Standards and Technology, Gaithersburg, MD, May 1994.

- [58] M. Gilloux. Research into the new generation of character and mailing address recognition systems at the french post office research center. In *Proc. 2nd IPTP Conf. on Postal Processing Systems in the 21st Century and Character Recognition*, pages 25–36, Tokyo, Japan, January 1992.
- [59] L. Heutte, J. Moreau, B. Plessis, J. Plagnaud, and Y. Lecourtier. Handwritten numeral recognition based on multiple feature extractors. In *Proc. 2nd Int. Conf. on Document Analysis and Recognition*, pages 167–170, Tsukuba Science City, Japan, October 1993.
- [60] G. E. Hinton, P. Dayan, and M. Revow. Modeling the manifolds of images of handwritten digits. *IEEE Trans. on Neural Networks*, 8(1):65–74, 1997.
- [61] T. K. Ho, J. J. Hull, and S. N. Srihari. Decision combination in multiple classifier systems. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-16(1):66–75, 1994.
- [62] T. K. Ho, J. J. Hull, and S. N. Srihari. A word shape analysis approach to recognition of degraded word images. In *Proc. 4th U.S. Postal Service Advanced Technology Conf.*, pages 217–231, November 1990.
- [63] B. K. P. Horn. Circle generators for display devices. *Comp. Graphics and Image Proc.*, 5:280–288, 1976.
- [64] B. K. P. Horn and E. J. Weldon Jr. Filtering closed curves. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8(5):665–668, 1986.
- [65] J. S. Huang and K. Chuang. Heuristic approach to handwritten numeral recognition. *Pattern Recognition*, 19(1):15–19, 1986.
- [66] Y. S. Huang and C. Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-17(1):90–94, 1995.
- [67] Y. S. Huang and C. Y. Suen. An optimal method of combining multiple classifiers for unconstrained handwritten numeral recognition. In *Proc. 3rd Int.*

- Workshop on Frontiers in Handwriting Recognition*, pages 11–20, Buffalo, N.Y., May 1993.
- [68] J. J. Hull, A. Commike, and T.-K. Ho. Multiple algorithms for handwritten character recognition. In *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, pages 117–124, Concordia University, Montreal, April 1990.
- [69] T. Kadonaga and K. Abe. Comparison of methods for detecting corner points from digital curves. In *Proc. IAPR Int. Workshop on Graphics Recognition*, pages 3–12, Penn State Scanticon, August 1995.
- [70] P. Kammenos. Performances of polar coding for visual localisation of planar objects. In *Proc. 8th Int. Symposium on Industrial Robots*, pages 143–149, Stuttgart, Germany, 1978.
- [71] T. Kanungo and R. M. Haralick. Character recognition using mathematical morphology. In *Proc. 4th U.S. Postal Service Advanced Technology Conf.*, pages 973–986, November 1990.
- [72] T. Kasvand and N. Otsu. Regularization of digitized plane curves for shape analysis and recognition. In *Proc. SPIE Conf. 'Architecture and Algorithms for Digital Image Processing*, pages 44–52, San Diego, August 1983.
- [73] K. Kimura and M. Shridhar. Handwritten numeral recognition based on multiple algorithms. *Pattern Recognition*, 24(10):969–983, 1991.
- [74] J. J. Koenderink. The structure of images. *Biological Cybernetics*, 50:363–370, 1984.
- [75] A. Krzyzak, W. Dai, and C. Y. Suen. Unconstrained handwritten character classification using modified backpropagation model. In *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, pages 155–166, Concordia University, Montreal, April 1990.

- [76] C. L. Kuan and S. N. Srihari. A stroke-based approach to handwritten numeral recognition. In *Proc. 3rd U.S. Postal Service Advanced Technology Conf.*, pages 1033–1041, 1988.
- [77] Z. Kulpa. Area and perimeter measurement of blobs in discrete binary pictures. *Comp. Graphics and Image Proc.*, 6:434–451, 1977.
- [78] Z. Kulpa. A note on the paper by B.K.P. Horn: Circle generators for display devices. *Comp. Graphics and Image Proc.*, 9:102–103, 1979.
- [79] Z. Kulpa. On the properties of discrete circles, rings, and disks. *Comp. Graphics and Image Proc.*, 10:348–365, 1979.
- [80] L. Lam and C. Y. Suen. Increasing experts for majority vote in ocr: Theoretical considerations and strategies. In *Proc. 4th Int. Workshop on Frontiers in Handwriting Recognition*, pages 245–254, Taipei, Taiwan, December 1994.
- [81] L. Lam and C. Y. Suen. Optimal combinations of pattern classifiers. *Pattern Recognition Letters*, 16:945–954, 1995.
- [82] L. Lam and C. Y. Suen. Structural classification and relaxation matching of totally unconstrained handwritten zip-code numbers. *Pattern Recognition*, 21(1):19–31, 1988.
- [83] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, and H. S. Baird. Constrained neural network for unconstrained handwritten digit recognition. In *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, pages 145–154, Concordia University, Montreal, April 1990.
- [84] Y. Le Cun, L. Jackel, B. Boser, J. S. Denker, H. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 41–46, 1989.

- [85] D. Lee and T. Pavlidis. One-dimensional regularization with discontinuities. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-10(6):822–829, 1988.
- [86] D. S. Lee, S. W. Lam, and S. N. Srihari. A structural approach to recognize hand-printed and degraded machine printed characters. In *Pre-Proc. IAPR Workshop on Syntactic & Structural Pattern Recognition*, pages 256–272, Murray Hill, New Jersey, June 1990.
- [87] D. S. Lee and S. N. Srihari. Handprinted digit recognition: A comparison of algorithms. In *Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition*, pages 153–162, Buffalo, N.Y., May 1993.
- [88] D. S. Lee and S. N. Srihari. A theory of classifier combination: the neural network approach. In *Proc. 3rd Int. Conf. on Document Analysis and Recognition*, pages 42–45, Montreal, Canada, August 1995.
- [89] S. Lee, C.-H. Kim, H. Ma, and Y. Y. Tang. Multiresolution recognition of unconstrained handwritten numerals with wavelet transform and multilayer cluster neural network. *Pattern Recognition*, 29(12):1953–1961, 1996.
- [90] S. Lee and Y. Kim. A new type of recurrent neural network for handwritten character recognition. In *Proc. 3rd Int. Conf. on Document Analysis and Recognition*, pages 38–41, Montreal, Canada, August 1995.
- [91] S. Lee, Y. Kim, and M. Kim. Multilayer cluster neural network for off-line recognition of totally unconstrained handwritten numerals. In *Proc. 4th Int. Workshop on Frontiers in Handwriting Recognition*, pages 115–124, Taipei, Taiwan, December 1994.
- [92] Y. Lee. Handwritten digit recognition using k-nearest-neighbor radial basis function and backpropagation neural networks. *Neural Computation*, 3:440–449, 1991.

- [93] R. Legault and C. Y. Suen. A comparison of methods of extracting curvature features. In *Proc. 11th Int. Conf. on Pattern Recognition*, pages 134–138, The Hague, The Netherlands, Aug.–Sept. 1992.
- [94] R. Legault and C. Y. Suen. *A Contour-Based Recognition System for Totally Unconstrained Handwritten Numerals*. Technical Report, Concordia University, Montreal, 1989.
- [95] R. Legault and C. Y. Suen. Contour tracing and parametric approximations for digitized patterns. In A. Krzyzak, T. Kasvand, and C. Y. Suen, editors, *Computer Vision and Shape Recognition*, pages 225–240, World Scientific Publishing Co., Singapore, 1989.
- [96] R. Legault and C. Y. Suen. Optimal local weighted averaging methods in contour smoothing. Accepted for publication *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1997.
- [97] R. Legault and C. Y. Suen. *Smoothing of 2-D Binary Contours: A Survey and New Results*. Technical Report, Concordia University, Montreal, 1995.
- [98] R. Legault, C. Y. Suen, and C. Nadal. Classification of confusing handwritten numerals by human subjects. In *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, pages 181–193, Concordia University, Montreal, April 1990.
- [99] R. Legault, C. Y. Suen, and C. Nadal. Difficult cases in handwritten numerals recognition. In H. S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 235–249, Springer-Verlag, Heidelberg, Germany, 1992.
- [100] X. Li and T. Chen. Optimal \mathcal{L}_1 approximation of the gaussian kernel with application to scale-space construction. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-17(10):1015–1019, 1995.
- [101] T. Lindeberg. Scale-space for discrete signals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-12(3):234–254, 1990.

- [102] H.-C. Liu and M. D. Srinath. Corner detection from chain code. *Pattern Recognition*, 23(1/2):51–68, 1990.
- [103] D. G. Lowe. Organization of smooth image curves at multiple scales. *Int. Journal on Computer Vision*, 3:119–130, 1989.
- [104] A. K. Mackworth and F. Mokhtarian. The renormalized curvature scale space and the evolution properties of planar curves. In *Proc. Comp. Vision and Pattern Recognition*, pages 318–326, Ann Arbor, Michigan, June 1988.
- [105] T. Mai and C. Y. Suen. A generalized knowledge-based system for the recognition of unconstrained handwritten numerals. *IEEE Trans. Syst. Man and Cybernet.*, 20(4):835–848, 1990.
- [106] S. E. Malowany. *A Graphical Rule-Based System for Recognizing On-Line Handwritten Digits*. Master's Thesis, Dept. of Computer Science, Concordia University, 1993.
- [107] E. Mandler and J. Schürmann. Combining the classification results of independent classifiers based on the dempster/shafer theory of evidence. In E. Gelsema and L. Kanal, editors, *From Pixel to Features III*, pages 381–393, Elsevier Science Publishers B.V. (North-Holland), 1988.
- [108] P. Maragos. Pattern spectrum and multiscale shape representation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-11(1):701–716, 1989.
- [109] G. Martin and J. Pittman. Recognizing handprinted letters and digits. In D. Touretzsky, editor, *Neural Information Processing Systems 2*, pages 405–414, Morgan Kaufmann, San Mateo, California, 1990.
- [110] T. Matsui, T. Tsutsumida, and S. Srihari. Combination of stroke/background structure and contour-direction features in handprinted alphanumeric recognition. In *Proc. 4th Int. Workshop on Frontiers in Handwriting Recognition*, pages 87–96, Taipei, Taiwan, December 1994.

- [111] T. Matsui, I. Yamashita, T. Wakahara, and M. Yoshimuro. State of the art of handwritten numeral recognition in japan (the results of the 1st iptp character recognition competition). In *Proc. First European Conf. dedicated to Postal Technologies*, pages 3–10, Nantes, France, 1993.
- [112] J. W. McKee and J. K. Aggarwal. Computer recognition of partial views of curved objects. *IEEE Transactions on Computers*, C-26:790–800, 1977.
- [113] G. Medioni and Y. Yasumoto. Corner detection and curve representation using cubic B-splines. *Comp. Vision, Graphics and Image Proc.*, 39:267–278, 1987.
- [114] B. T. Mitchell and A. M. Gillies. A model-based computer vision system for recognizing handwritten zip codes. *Machine Vision and Applications*, 2:231–243, 1989.
- [115] F. Mokhtarian and A. Mackworth. Scale-based description and recognition of planar curves and two-dimensional shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8(1):34–43, 1986.
- [116] S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE, Special issue on Optical Character Recognition*, 80(7):1029–1058, July 1992.
- [117] C. Nadal and C. Y. Suen. *Recognition of Totally Unconstrained Handwritten Digits by Decomposition and Vectorisation*. Technical Report, Concordia University, Montreal, 1988.
- [118] H. Nishida. Toward automatic construction of structural models for unconstrained handwritten characters. In M. L. Simmer, C. G. Leedham, and A. J. W. M. Thomassen, editors, *Handwriting and Drawing Research: Basic and Applied Issues*, pages 359–372, IOS Press, 1996.
- [119] H. Nishida and S. Mori. An approach to automatic construction of structural models for character recognition. In *Proc. 1st Int. Conf. on Document Analysis and Recognition*, pages 231–241, St-Malo, France, September 1991.

- [120] H. Nishida and S. Mori. Structural analysis and description of curves by quasi-topological features and singular points. In H. S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured Document Image Analysis*, pages 139–187, Springer-Verlag, Heidelberg, Germany, 1992.
- [121] T. Noumi, T. Matsui, I. Yamashita, T. Wakahara, and T. Tsutsumida. Results of second iptp character recognition competition and studies on multi-expert handwritten numeral recognition. In *Proc. 4th Int. Workshop on Frontiers in Handwriting Recognition*, pages 338–346, Taipei, Taiwan, December 1994.
- [122] L. O’Gorman. An analysis of feature detectability from curvature estimation. In *Proc. Comp. Vision and Pattern Recognition*, pages 235–240, Ann Arbor, Michigan, June 1988.
- [123] L. O’Gorman. Curvilinear feature detection from curvature estimation. In *Proc. 9th Int. Conf. on Pattern Recognition*, pages 1116–1119, Rome, November 1988.
- [124] J. Oliensis. Local reproducible smoothing without shrinkage. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(3):307–312, 1993.
- [125] E. J. Pauwels, L. J. V. Gool, P. Fiddelaers, and T. Moons. An extended class of scale-invariant and recursive scale space filters. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-17(7):691–701, 1995.
- [126] S.-C. Pei and J.-H. Horng. Fitting digital curve using circular arcs. *Pattern Recognition*, 28(1):107–116, 1995.
- [127] T.-Y. Phillips and A. Rosenfeld. A method of curve partitioning using arc-chord distance. *Pattern Recognition Letters*, 5:285–288, 1987.
- [128] T. Poggio, H. Voorhees, and A. Yuille. *A Regularized Solution to Edge Detection*. Technical Report, M.I.T. AI Laboratory, AI Memo 776, 1985.
- [129] U. Ramer. An iterative procedure for the polygonal approximation of plane closed curves. *Comput. Graphics and Image Proc.*, 1:244–256, 1972.

- [130] A. Rattarangsi and R. T. Chin. Scale-based detection of corners of planar curves. In *Proc. 10th Int. Conf. on Pattern Recognition*, pages 923–930, Atlantic City, NJ, June 1990.
- [131] C. M. Reinsch. Smoothing by spline functions. *Numerische Mathematik*, 10:177–183, 1967.
- [132] M. Revow, C. Williams, and G. Hinton. Using generative models for handwritten digit recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18(6):592–606, 1996.
- [133] M. Revow, C. Williams, and G. Hinton. Using mixtures of deformable models to capture variations in handprinted digits. In *Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition*, pages 142–152, Buffalo, N.Y., May 1993.
- [134] D. Rosen. On the areas and boundaries of quantized objects. *Comp. Graphics and Image Proc.*, 13:94–98, 1980.
- [135] A. Rosenfeld. *Picture Processing by Computer*. Academic Press, New York, 1969.
- [136] A. Rosenfeld and E. Johnston. Angle detection on digital curves. *IEEE Transactions on Computers*, C-22:875–878, 1973.
- [137] A. Rosenfeld and J. Weszka. An improved method of angle detection on digital curves. *IEEE Transactions on Computers*, C-24:940–941, 1975.
- [138] W. S. Rutkowski and A. Rosenfeld. *A Comparison of Corner-Detection Techniques for Chain-Coded Curves*. Technical Report, Computer Science Center, University of Maryland, 1978.
- [139] M. Sabourin, A. Mitiche, D. Thomas, and G. Nagy. Classifier combination for hand-printed digit recognition. In *Proc. 2nd Int. Conf. on Document Analysis and Recognition*, pages 163–166, Tsukuba Science City, Japan, October 1993.

- [140] P. Saint-Marc, J. S. Chen, and G. Medioni. Adaptive smoothing: A general tool for early vision. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-13(6):514–529, 1991.
- [141] T. Sakai, M. Nagao, and H. Matsushima. Extraction of invariant picture substructures by computer. *Comp. Graphics and Image Proc.*, 1(1):81–96, 1972.
- [142] P. V. Sankar and E. V. Krishnamurthy. On the compactness of subsets of digital images. *Comp. Graphics and Image Proc.*, 8:136–143, 1978.
- [143] P. V. Sankar and C. V. Sharma. A parallel procedure for the detection of dominant points on a digital curve. *Comp. Graphics and Image Proc.*, 7:403–412, 1978.
- [144] I. J. Schoenberg. Spline functions and the problem of graduation. *Proc. Nat. Acad. Sci.*, 52:947–950, 1964.
- [145] B. Shabaray and D. J. Anderson. Optimal estimation of contour properties by cross-validated regularization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-11(6):600–610, 1989.
- [146] B. Shabaray and D. J. Anderson. Optimal smoothing of digitized contours. In *Proc. Comp. Vision and Pattern Recognition*, pages 210–218, Miami Beach, Florida, June 1986.
- [147] M. Shridhar and A. Badreldin. Recognition of isolated and simply connected handwritten numerals. *Pattern Recognition*, 19(1):1–12, 1986.
- [148] P. Simard, Y. LeCun, and J. Denker. Memory-based character recognition using a transformation invariant metric. In *Proc. 12th Int. Conf. on Pattern Recognition*, pages 262–267, Jerusalem, October 1994.
- [149] S. Smith, M. Bourgojn, K. Sims, and H. Voorhees. Handwritten character classification using nearest neighbor in large databases. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-16(9):915–919, 1994.

- [150] D. H. Stone, L. A. Pintsov, and D. P. D'Amato. Alphanumeric handprint recognition. *U.S. Patent 4 628 532*, December 1986.
- [151] N. Strathy and C. Y. Suen. A new system for reading handwritten zip codes. In *Proc. 3rd Int. Conf. on Document Analysis and Recognition*, pages 74–77, Montreal, Canada, August 1995.
- [152] L. Stringa. Efficient classification of totally unconstrained handwritten numerals with a trainable multilayer network. *Pattern Recognition Letters*, 10:273–280, 1989.
- [153] L. Stringa. A new set of constraint-free character recognition grammars. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-12(12):1210–1217, 1990.
- [154] C. Y. Suen. Distinctive features in the automatic recognition of handprinted characters. *Signal Processing*, 4:193–207, 1982.
- [155] C. Y. Suen, M. Berthod, and S. Mori. Automatic recognition of handprinted characters. the state of the art. *Proceedings of the IEEE*, 68(4):469–483, 1980.
- [156] C. Y. Suen, R. Legault, C. Nadal, M. Cheriet, and L. Lam. Building a new generation of character recognition systems. *Pattern Recognition Letters*, 14:303–315, 1993.
- [157] C. Y. Suen, C. Nadal, R. Legault, T. A. Mai, and L. Lam. Computer recognition of unconstrained handwritten numerals. *Proceedings of the IEEE, Special issue on Optical Character Recognition*, 80(7):1162–1180, July 1992.
- [158] C. Y. Suen, C. Nadal, T. A. Mai, R. Legault, and L. Lam. Recognition of handwritten numerals based on the concept of multiple experts. In *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, pages 131–144, CENPARMI, Concordia University, Montreal, April 1990.

- [159] C. C. Tappert. *Speed, Accuracy, Flexibility Trade-Offs in On-Line Character Recognition*. Technical Report, IBM Research Report RC13228 (#59158), T. J. Watson Research Center, Yorktown Heights, 1987.
- [160] A. E. Taylor and W. R. Mann. *Advanced Calculus*. John Wiley & Sons, 1983.
- [161] C.-H. Teh and R. T. Chin. On the detection of dominant points on digital curves. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-11(8):859–872, 1989.
- [162] M. H. Thien and H. Bunke. Handwritten numeral recognition by perturbation method. In *Proc. 4th Int. Workshop on Frontiers in Handwriting Recognition*, pages 97–106, Taipei, Taiwan, December 1994.
- [163] D.-M. Tsai and M.-F. Chen. Curve fitting approach for tangent angle and curvature measurements. *Pattern Recognition*, 27(5):699–711, 1994.
- [164] M. D. Wheeler and K. Ikeuchi. Iterative smoothed residuals: A low-pass filter for smoothing with controlled shrinkage. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-18(3):334–337, 1996.
- [165] R. Wilkinson, J. Geist, S. Janet, P. Grother, C. Burges, R. Creecy, B. Hammond, J. Hull, N. Larsen, T. Vogl, and C. Wilson. *The First Census Optical Character Recognition Systems Conference, Tech. Report # NIST 4912*. Technical Report, Nat. Inst. of Standards and Technology, Gaithersburg, MD, August 1992.
- [166] A. Witkin. Scale-space filtering. In *Proc. Int. Joint Conf. on AI*, pages 1019–1022, 1983.
- [167] M. Worring and A. Smeulders. Digitized circular arcs: Characterization and parameter estimation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-17(6):587–598, 1995.
- [168] L. Wu and Z. Xie. Scaling theorems for zero-crossings. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-12(1):46–54, 1990.

- [169] D. M. Wuescher and K. L. Boyer. Robust contour decomposition using a constant curvature criterion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-13(1):41–51, 1991.
- [170] L. Xu, A. Krzyzak, and C. Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Trans. Systems, Man, and Cybernetics*, 22(3):418–435, 1992.
- [171] H. Yan. Design and implementation of optimized nearest neighbor classifiers for handwritten digit recognition. In *Proc. 2nd Int. Conf. on Document Analysis and Recognition*, pages 10–13, Tsukuba Science City, Japan, October 1993.
- [172] H. Yan. Handwritten digit recognition using an optimized nearest neighbor classifier. *Pattern Recognition Letters*, 15:207–211, 1994.
- [173] A. L. Yuille and T. A. Poggio. Scaling theorems for zero crossings. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8(1):15–25, 1986.
- [174] P. Zhu and P. M. Chirlian. On critical point detection of digital shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-17(8):737–748, 1995.

Appendix A

About Some Numeral Databases

A.1 CEDAR Database

This database is available on CEDAR CDROM 1 and consists of 21 179 binary images of digits extracted from U.S. ZIP codes. The resolution is 300 PPI. Training data is in a directory called *br* and includes a total of 18 468 samples; balanced training sets, of 200 samples per class, are labeled *cedar1*, *cedar2*, ... , *cedar7*. Testing data (2 711 samples) is in the *bs* directory; a subset of 2 213 well segmented samples were also extracted from the *bs* directory and are under the *goodbs* directory.

A.2 CENPARMI Database

One of the first databases used for comparative purposes is the CENPARMI database, composed of approximately 17 000 isolated numerals from an estimated 3 400 writers. The data was collected from dead letter envelopes by the U.S. Postal Service at various locations in the U.S. The samples were digitized in 16 grey levels on a 64x224 grid of 0.153 mm square elements, giving a resolution of 166 PPI.

Each digitized 5-digit zipcode was iteratively enhanced, binarized and segmented, trying to obtain no more than 5 body regions per zipcode ¹. All resulting binary images of individual digits (in a single piece) were run-length encoded.

¹Some images were also manually segmented.

Because of data imbalance (for example, there are only 697 nine's for 3 595 one's), 2 training sets A and B and a testing set T, each consisting of 2 000 samples (200 of each class), were constituted. For a detailed description of this database, please refer to Suen et al. [157].

A.3 Concordia-Montreal Database

This database contains 20 000 run-length encoded binary images of individual digits. It was collected by CENPARMI at Concordia University and in the Montreal area. Each of 500 subjects was asked to write the 0...9 sequence 8 times, 4 sequences being written in pre-formed boxes and the other 4 in free style on lined paper. Only the second and third sequences of each kind were retained. The data was digitized at a resolution of 200 PPI. Two sets of 5 000 samples, one for training purposes and the other for testing, were prepared for the free- and the box-format data; they are known as the *e* and *f* sets, for free-format data, and the *g* and *h* sets, for box-format. Training and testing samples are not from the same writers.

A.4 ITRI-Taiwan Databases

Lastly, we have used two ITRI-Taiwan databases. The smaller database contains 6 997 run-length encoded binary images of individual digits. This data was collected and prepared by the ITRI Corporation in Taiwan. The numerals were written in pre-formed boxes, digitized in 256 grey levels at a resolution of 400 PPI, and then binarized. Many samples are written in styles not generally encountered in North America. The digits are separated into 7 sets of 1 000 samples (100 per numeral class, except for the last set containing only 97 nine's), labelled TW-1, TW-2, ... TW-7. The first 4 sets are training material and the last 3 are testing sets. The larger database is separated into 10 sets, labeled *t*₂₀ to *t*₂₉. The first 5 sets, *t*₂₀ to *t*₂₄, are training material and contain 24 427 samples; for a breakdown of the number of digits in every class, see Appendix J. The last 5 sets are testing material and they contain 22 024 samples.

Appendix B

Derivation of ϕ_{rms} For Noisy Horizontal Border

The root mean square of the deviation angle, ϕ_{rms} , is computed by considering the 8 possible configurations of 3 consecutive pixels generated by our simple model. Taking into account the probability of each configuration and its associated deviation angle, ϕ_{rms} is easily obtained.

The situations and values of interest are shown in Table 39. Using these values, we obtain an expression for the mean square deviation angle:

$$\overline{\phi_i^2} = \sum_{i=1}^8 P_i \phi_i^2 = [p(1-p)^2 + p^2(1-p)] \cdot \left[\frac{\pi^2}{4} + \frac{2\pi^2}{16} \right] \quad (45)$$

Factoring out $p(1-p)$ in the first bracket above we have:

$$p(1-p)^2 + p^2(1-p) = p(1-p)[1-p+p] = p(1-p). \quad (46)$$

Substituting this last result in Equation 45 yields:

$$\overline{\phi_i^2} = p(1-p) \cdot \frac{3\pi^2}{8} \quad (47)$$

Finally, ϕ_{rms} is obtained by taking the square root:

$$\phi_{rms} = \frac{\pi}{2} \sqrt{\frac{3p(1-p)}{2}}. \quad (48)$$

APPENDIX B. DERIVATION OF ϕ_{RMS} FOR NOISY HORIZONTAL BORDER301

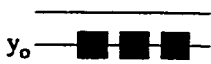
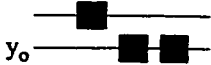






Index i	Configuration C_i	Probability P_i	Deviation angle squared ϕ_i^2
1		$(1 - p)^3$	0
2		$p(1 - p)^2$	$\frac{\pi^2}{16}$
3		$p(1 - p)^2$	$\frac{\pi^2}{4}$
4		$p(1 - p)^2$	$\frac{\pi^2}{16}$
5		$p^2(1 - p)$	$\frac{\pi^2}{16}$
6		$p^2(1 - p)$	$\frac{\pi^2}{4}$
7		$p^2(1 - p)$	$\frac{\pi^2}{16}$
8		p^3	0

Table 39: Possible Values of ϕ_i^2 for Unsmoothed Border

Appendix C

Best Parameters to Minimize d'_{rms}

We must now minimize $\sum_{j=-n}^n \alpha_j^2$, subject to the constraint $\alpha_0 + 2 \sum_{j=1}^n \alpha_j - 1 = 0$. The method of Lagrange multipliers consists of building an expression which incorporates the constraint and minimizing that expression with respect to the α_j 's, for $j = 0, 1, \dots, n$. More precisely, we must minimize

$$\sum_{j=-n}^n \alpha_j^2 + \lambda (\alpha_0 + 2 \sum_{j=1}^n \alpha_j - 1)$$

By using the symmetry constraint $\alpha_{-j} = \alpha_j$ for $j = 1, 2, \dots, n$, this is equivalent to minimizing:

$$\alpha_0^2 + 2 \sum_{j=1}^n \alpha_j^2 + \lambda (\alpha_0 + 2 \sum_{j=1}^n \alpha_j - 1)$$

Taking partial derivatives with respect to α_0 and each of the α_j 's, we obtain:

$$\begin{aligned} \frac{\delta}{\delta \alpha_0} : \quad 2\alpha_0 + \lambda &= 0 \\ \frac{\delta}{\delta \alpha_j} : \quad 4\alpha_j + 2\lambda &= 0 \end{aligned} \tag{49}$$

From the first equation, we derive $\lambda = -2\alpha_0$. Substituting this result into the second equation yields for $j = 1, 2, \dots, n$

$$\alpha_j = \frac{-2\lambda}{4} = \frac{4\alpha_0}{4} = \alpha_0. \tag{50}$$

Replacing each α_j by α_0 in the normalization condition, we obtain:

$$\alpha_0 + 2 \sum_{j=1}^n \alpha_j - 1 = \alpha_0 + 2n\alpha_0 - 1 = 0. \quad (51)$$

Hence

$$\alpha_j = \frac{1}{2n+1}, \text{ for } j = 0, 1, \dots, n. \quad (52)$$

Appendix D

Minimizing $m'_r m_s$

Using the forward difference formula $m'_i = y'_{i+1} - y'_i$, we have:

$$\begin{aligned}
 m_i'^2 &= (y'_{i+1} - y'_i)^2 \\
 &= \left(\sum_{j=-n+1}^{n+1} \alpha_{j-1} y_{i+j} - \sum_{j=-n}^n \alpha_j y_{i+j} \right)^2 \\
 &= (\alpha_n y_{i+1+n} - \alpha_{-n} y_{i-n})^2 + 2 \sum_{j=-n+1}^n \alpha_n (\alpha_{j-1} - \alpha_j) y_{i+1+n} y_{i+j} \\
 &\quad - 2 \sum_{j=-n+1}^n \alpha_{-n} (\alpha_{j-1} - \alpha_j) y_{i-n} y_{i+j} + \left(\sum_{j=-n+1}^n (\alpha_{j-1} - \alpha_j) y_{i+j} \right)^2 \quad (53)
 \end{aligned}$$

Expanding the last squared summation and taking the mean, we obtain the following expression:

$$\begin{aligned}
 \overline{m_i'^2} &= \alpha_n^2 \overline{y_{i+1+n}^2} - 2\alpha_{-n} \alpha_n \overline{y_{i+1+n} y_{i-n}} + \alpha_{-n}^2 \overline{y_{i-n}^2} \\
 &\quad + 2 \sum_{j=-n+1}^n \alpha_n (\alpha_{j-1} - \alpha_j) \overline{y_{i+1+n} y_{i+j}} - 2 \sum_{j=-n+1}^n \alpha_{-n} (\alpha_{j-1} - \alpha_j) \overline{y_{i-n} y_{i+j}} \\
 &\quad + \sum_{j=-n+1}^n (\alpha_{j-1} - \alpha_j)^2 \overline{y_{i+j}^2} + 2 \sum_{\substack{j < k \\ j=-n+1 \\ k=-n+1}}^n (\alpha_{j-1} - \alpha_j) (\alpha_{k-1} - \alpha_k) \overline{y_{i+j} y_{i+k}} \quad (54)
 \end{aligned}$$

For any $s, t \in \mathcal{Z}$, $\overline{y_s y_t} = (y_o + p)^2$ and $\overline{y_s^2} = (y_o + p)^2 + p(1-p)$. Substituting these results, Equation 54 can be further simplified by making use of Equation 16.

We obtain:

$$\overline{m_i'^2} = 2p(1-p) \left[\sum_{j=-n}^n \alpha_j^2 + \sum_{j=-n+1}^n \alpha_{j-1} \alpha_j \right]. \quad (55)$$

Using Equation 16 again, some algebraic manipulation leads to an expression for $\overline{m_i'^2}$ involving only the n independent parameters $\alpha_1, \alpha_2, \dots, \alpha_n$:

$$\begin{aligned} \overline{m_i'^2} = 2p(1-p) & \left[1 - 2\alpha_1 + 4(\alpha_1 - 1) \sum_{j=1}^n \alpha_j \right. \\ & \left. + 2 \sum_{j=1}^n \alpha_j^2 + 4 \left(\sum_{j=1}^n \alpha_j \right)^2 - 2 \sum_{j=1}^{n-1} \alpha_j \alpha_{j+1} \right]. \end{aligned}$$

Appendix E

Obtaining (dx'_i, dy'_i) Recursively

Using triangular smoothing filters of any window size $w = 2n + 1$, we have:

$$(x'_i, y'_i) = \sum_{j=-n}^n \frac{n+1-|j|}{(n+1)^2} (x_{i+j}, y_{i+j}) \quad (56)$$

and for the smoothed coordinates of the preceding contour point we have:

$$(x'_{i-1}, y'_{i-1}) = \sum_{j=-n-1}^{n-1} \frac{n+1-|j+1|}{(n+1)^2} (x_{i+j}, y_{i+j}) \quad (57)$$

Subtracting Equation 57 from Equation 56 yields:

$$(dx'_i, dy'_i) = \frac{1}{(n+1)^2} \left((x_{i-n-1}, y_{i-n-1}) + \sum_{j=-n}^{n-1} (|j+1| - |j|)(x_{i+j}, y_{i+j}) + (x_{i+n}, y_{i+n}) \right) \quad (58)$$

Now the value of $(|j+1| - |j|)$ is simply 1 for $j \geq 0$, and -1 otherwise. Thus we have:

$$(dx'_i, dy'_i) = \frac{1}{(n+1)^2} \left(\sum_{j=0}^n (x_{i+j}, y_{i+j}) - \sum_{j=-n-1}^{-1} (x_{i+j}, y_{i+j}) \right) \quad (59)$$

For the following contour point, the same difference yields:

$$(dx'_{i+1}, dy'_{i+1}) = \frac{1}{(n+1)^2} \left(\sum_{j=1}^{n+1} (x_{i+j}, y_{i+j}) - \sum_{j=-n}^0 (x_{i+j}, y_{i+j}) \right) \quad (60)$$

Subtracting Equation 59 from Equation 60, most terms in the sums cancel out leaving:

$$(dx'_{i+1}, dy'_{i+1}) - (dx'_i, dy'_i) = \frac{1}{(n+1)^2} ((x_{i-n-1}, y_{i-n-1}) - 2(x_i, y_i) + (x_{i+n+1}, y_{i+n+1})) \quad (61)$$

By simply moving (dx'_i, dy'_i) to the right side of the preceding equation, we obtain the recurrence relation of Equation 42 in Section 6.4.1.

We note that the very first pair (dx'_i, dy'_i) can easily be computed in terms of the original unsmoothed coordinates using Equation 59. Then all following pairs can be 'updated' with the recurrence relation, at a low and fixed cost involving the unsmoothed coordinates of only 3 contour points, independently of the value of n (hence w) from which we started.

Appendix F

Syntax For Classification Rules

The detailed syntax used for the classification rules is discussed in this appendix. The order of presentation is lexicographic, based on the initial capital letter of the rules.

F.1 Bounding Box Rules

The rules presented in this section do not cause the computation of bounding box parameters for a feature or a feature sequence; they test these parameters *after* they have been evaluated (by rule 5 in section F.3).

1. $B\{l | t | r | b | v | h | a\}\{op\} V_1 [V_2]$

The vertical bars inside the braces indicate disjunction i.e. only one of the 7 lowercase letters must be present; their meaning is as follows:

- Bl : leftmost column of bounding box;
- Bt : topmost row of bounding box;
- Br : rightmost column of bounding box;
- Bb : bottommost row of bounding box;
- Bv : vertical span of bounding box relative to number of rows in image;
- Bh : horizontal span of bounding box relative to number of columns in image;
- Ba : cumulative deviation angle for feature (or feature sequence);

The meanings of *op*, and of V_1 and $[V_2]$ are already explained in Section 7.2.1.

2. Boh= {l | r}

When traveling along a feature (or feature sequence), the orientation horizontally is from right to left (l) or from left to right (r)

Bov= {t | b}

When traveling along a feature (or feature sequence), the orientation vertically is from bottom to top (t) or from top to bottom (b).

3. B{l | r}x{op} V₁ [V₂]

- Blx : row(x) coordinate of leftmost contour point of bounding box;
- Brx : row(x) coordinate of rightmost contour point of bounding box;

B{t | b}y{op} V₁ [V₂]

- Bty : column(y) coordinate of topmost contour point of bounding box;
- Bby : column(y) coordinate of bottommost contour point of bounding box;

F.2 Comparison Rules

The following rules are evaluated using some of the 10 elements of the array *stored_value*.

1. C_{i₁}{+ | - | * | /}i₂{op - {=, ~}} V₁ [V₂]

Here i_1 and i_2 are integers in the range [0..9] and {op - {=, ~}} represents one of the operators defined in {op} with the exception of the equality and inequality operators. For example, C0/1< 0.75 means $\frac{\text{stored_value}[0]}{\text{stored_value}[1]} < 0.75$; and C1+2@ 3 means $\text{stored_value}[3] \leftarrow \text{stored_value}[1] + \text{stored_value}[2]$.

2. C_{i₁}=i₂

This tests the equality of 2 elements of the array *stored_value*. In practice, this is only allowed to compare the types of 2 features.

3. C_{i₁}<i₂

After firing this rule, *stored_value*[i₁] will contain the least of the 2 stored values (of index i_1 and i_2).

C_{i₁}>i₂

After firing this rule, *stored_value*[i_1] will contain the maximum of the 2 stored values (of index i_1 and i_2).

4. $Ci_1a\{\text{op} - \{=, \sim\}\} V_1 [V_2]$

Here ' i_1a ' stands for the absolute value of *stored_value*[i_1]. Again the equality and inequality operators are not allowed.

5. $Ci_1\{\text{op} - \{=, \sim, @\}\} V_1 [V_2]$

Here the equality, inequality and @ operators are not allowed.

F.3 Feature Rules

1. $Fi_1\{a | d | m | t | i\}\{\text{op}\} V_1 [V_2]$

Here i_1 is any integer in the range [0..5] and the meaning of the lowercase letters in the other pair of braces is as follows:

- a : cumulative deviation angle of feature region pointed to by PF[i_1];
- d : direction of feature region pointed to by PF[i_1];
- m : merge-count¹ of feature region pointed to by PF[i_1];
- t : type (C, B, or E) of feature region pointed to by PF[i_1];
- i : cumulative deviation angle in the inter-arc region preceding PF[i_1];

2. $Fi_1\{x | y | W\}\{f | l | o\}\{\text{op}\} V_1 [V_2]$

The meaning of the lowercase letters in the first set of braces is as follows:

- x : row position;
- y : column position;
- W : Width of body region measured perpendicularly to contour boundary;

and the meaning of the lowercase letters in the second set of braces is as follows:

- f : first contour point of feature region pointed to by PF[i_1];
- l : last contour point of feature region pointed to by PF[i_1];
- o : focal point of feature region pointed to by PF[i_1];

¹This is the number of initial Bends or Cavities which were merged together to form a single Bend or Cavity feature region.

For example $F3yf@ 1$ means to assign to $stored_value[1]$ the column coordinate of the first point of the feature region pointed to by $PF[3]$; and $F2Wo@ 3$ means to assign to $stored_value[3]$ the width of the body region measured perpendicularly to the contour boundary at the focal point of the feature region pointed to by $PF[2]$. The measurement $F2Wo$ is illustrated in Figure 90 (a).

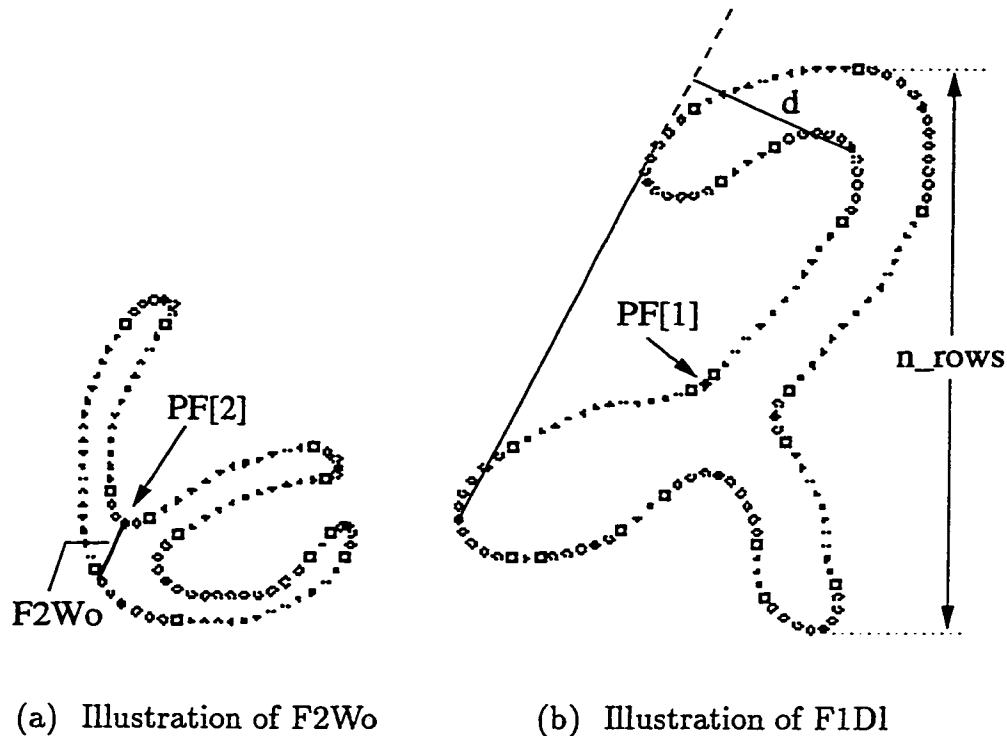


Figure 90: Examples of Width and Depth Feature Rule Measurements

3. $F_{i_1}W\{h | v\}\{f | l | o\}\{op\} V_1 [V_2]$

This rule is similar to the W -case of the preceding rule, except that the width of the body region is not measured perpendicularly to the contour boundary but horizontally (h) or vertically (v). All the other parameters have the same interpretation.

4. $F_{i_1}Dl\{op\} V_1 [V_2]$

Here $F_{i_1}Dl$ means the relative depth of the left(west)-oriented same-type feature sequence of which the feature pointed to by $PF[i_1]$ is apart; and 'relative depth'

means the actual depth divided by the number of rows in the image. Consider the rule $F1D| > 0.2$ in the context of Figure 90 (b); this would then mean that the ratio $\frac{d}{n_rows}$ must be larger than 0.2.

5. $F_{i_1}B\{0 | 1 | 2 | 3 | 4 | 5\}$

This rule causes the computation of bounding box parameters associated with the feature pointed to by $PF[i_1]$ (and possibly neighbouring features also). If the feature pointed to by $PF[i_1]$ is preceded and/or followed by other features of the same type, let STFS stand for the complete same-type feature sequence to which $PF[i_1]$ belongs. The meaning of the numbers inside the pair of braces is then as follows:

- 0 : bounding box of feature region pointed to by $PF[i_1]$;
- 1 : bounding box of STFS;
- 2 : bounding box of region from last point of feature before STFS to first point of feature after STFS;
- 3 : bounding box of region from first point of feature before STFS to first point of feature after STFS;
- 4 : bounding box of region from last point of feature before STFS to last point of feature after STFS;
- 5 : bounding box of region from first point of feature before STFS to last point of feature after STFS;

The bounding box results are stored in variables which can then be tested by the rules of Section F.1

6. $F_{i_1}H\{2 | 6\}$

This rule was created for 2's and 6's with a bottom hole ; it verifies that the feature sequence of endpoints and Bends starting at $PF[i_1]$ actually encircles the hole.

7. $F_{i_1}v = \{0 | 1\}$

Has the feature pointed to by $PF[i_1]$ been visited yet ($= 1$) or not ($= 0$)?

8. $F_{i_1}u\{x | y\}\{op\} V_1 [V_2]$

Here $F_{i_1}u_x$ is the x-coordinate of the unit vector directed from the last point of the feature pointed to by $PF[i_1]$ to the first point of the following feature; and $F_{i_1}u_y$ is the y-coordinate of the same unit vector (see Section 6.4.10).

9. $F_{i_1}E\{0 | 1\}\{op\} V_1 [V_2]$

This function computes a measure of the relative elevation at the middle contour point between:

- the focal point of the feature pointed to by $PF[i_1]$ and the focal point of the following feature ($F_{i_1}E_o$); or
- the last point of feature pointed to by $PF[i_1]$ and the first point of the following feature region ($F_{i_1}E_l$);

The construction is similar to the one illustrated in Figure 86 (b), except that the line segment connects different pairs of points and the elevation is measured half-way between the extremities.

F.4 Global Feature Rules

1. $G\{P | H | S | R | C | O\}3\{op\} V_1 [V_2]$

The measurements referred to by these rules are the following:

- GP : the number of disjoint Pieces composing the image;
- GH : the number of Holes in the image;
- GS : the computed Stroke width (see Section 4.4.1);
- GR : the number of Rows (*n_rows*) in the image;
- GC : the number of Columns (*n_cols*) in the image;
- GO : the number of empty columns at the left of the image²;

$GR(n_rows)$, $GC(n_cols)$, $GS(stroke_width)$, and $GO(on_left_empty)$ are permanently kept in elements 4, 5, 6, and 7 of the *stored_value* array. As several measurements need to be considered relative to these parameters, this really speeds up calculations (with Compare-rules for example).

F.5 Hole Rules

1. $H\{0 \mid 1 \mid 2 \mid 3\}$

This rule simply sets the parameter *tested_hole_index* to one of the values inside the pair of braces.

2. $H\{b \mid l \mid t \mid r \mid p \mid h\}\{op\} V_1 [V_2]$

The meaning of the attributes covered by this rule is:

- H_a : area of hole (in pixels);
- H_b : bottommost row of hole;
- H_l : leftmost column of hole;
- H_t : topmost row of hole;
- H_r : rightmost column of hole;
- H_p : position ratio³ of hole;
- H_h : height ratio of hole⁴;

In all cases, the hole targeted by the above rules is the one designated by *tested_hole_index*.

F.6 Index Rules

The index rules are used to set, advance, or backup the pointers to features (PF) in the feature list.

1. $I_{i_1}\{+ \mid -\} V_{1.i}$

$V_{1.i}$ is an integer value indicating the number of features by which $PF[i_1]$ must be moved forward (+) or backward, in relation to its current position.

2. $I_{i_1}\{+ \mid -\}\{= \mid \sim\} \{B \mid C \mid E\}$

A couple of examples will easily clarify the meaning of rules following this syntax. Thus $I2+= E$ means ‘Move $PF[2]$ forward (+) to next endpoint’; and

³The number of rows above the hole divided by the number of rows below the hole.

⁴The height of the hole divided by *n_rows*.

$I1\sim C$ means 'Move PF[1] backward (-) to first feature which is not (\sim) a cavity'.

3. $Ii_1=i_2\{+ | -\} V_1.i$

This rule sets PF[i_1] relative to PF[i_2]. Thus $I2=4+ 1$ means 'Set PF[2] so that it points to the feature following the one pointed to by PF[4]'. Note that $V_1.i$ can be 0.

4. $Ii_1\{C | B | E\}\{+ | -\}$

If PF[i_1] is of the specified type, move it to the next (+) or to the preceding (-) feature. For example, $I3C-$ means 'If the feature pointed to by PF[3] is a cavity, set PF[3] to the preceding feature'.

Note that in all cases, moving a PF-pointer *forward* requires that the feature landed on must not be marked as already visited; no such condition is imposed on backward moves.

F.7 Marking Rules

1. Mi_1

Mark feature pointed to by PF[i_1] as already visited.

2. $Mi_1i_2\{+ | -\}$

Mark as already visited all features from the one pointed to by PF[i_1] to the one pointed to by PF[i_2]; inclusion or exclusion of PF[i_2] in the marking is indicated by + or - respectively.

3. U

Un-mark all features in the feature list. This is done before attempting a new recognition path.

F.8 Or Rules

1. O $V_1.i$ $V_2.i$

The Or-rule is followed immediately by 2 groups of rules, the first containing $V_1.i$ rules and the second containing $V_2.i$ rules. The Or-rule is considered as having succeeded if either one of these 2 groups of rules is satisfied. For example, the rule 'O 2 3' must be followed by at least 5 rules (2+3); and either the next 2 rules or the 3 rules after that must be satisfied for the Or-rule to be successful. Note that the last integer value, $V_2.i$, can be zero and that a group of 0 rules is always satisfied by default. This can be useful to conditionally fire certain rules. For example, the 3 rules

O 2 0

F2t~ C

I2+ 1

will move PF[2] to the next feature, if it is not pointing to a cavity⁵. Note also that Or-rules can be nested; if either of the 2 groups of rules specified by an Or-rule contains nested Or-rules, the number of lines specifying the length of the first two groups must include those of the nested Or-rules.

F.9 Piece Rules

1. P i_1

Here i_1 is an integer in the range $[0..(n_pieces - 1)]$. This rule simply sets the parameter *tested_piece_index* to the value specified. The next set of rules can then take action on the so-specified piece.

2. P{s | l | t | r | b | a | p | f | c | e}{op} V_1 [V_2]

The meaning of the attributes covered by these rules is as follows, always keeping in mind that the piece targeted is the one of index specified by *tested_piece_index*:

⁵If PF[2] initially points to a cavity, the second rule fails and thus the first group of rules fails; but the second group of 0 rule succeeds by default and the Or-rule is still considered successful. Thus the rules after these three can be fired.

- Ps : spanned area of targeted piece;
- Pl : leftmost column of targeted piece;
- Pt : topmost row of targeted piece;
- Pr : rightmost column of targeted piece;
- Pb : bottommost row of targeted piece;
- Pa : true area of targeted piece;
- Pp : number of contour points of targeted piece;
- Pf : number of features of targeted piece;
- Pc : number of cavities of targeted piece;
- Pe : number of endpoints of targeted piece;

F.10 Storage Modify Rules

The following rule can be used to modify the elements of the *stored_value* array with simple arithmetic functions.

1. $S_{i_1}\{+ | - | * | /\}$ $V_{i_1}.f$

For example, 'S2+ 4.0' means to increment the value of *stored_value*[2] by 4.0; and 'S1* 2.5' means to multiply the value of *stored_value*[1] by 2.5, storing the result back in *stored_value*[1].

Appendix G

Interface for Rule Generation

The development tool to assist in database investigation and classification rule generation was built using UIM/X (version 2.9), a comprehensive, second-generation, Graphic User Interface (GUI) Builder produced by Visual Edge Software Ltd. The interface is a program consisting of 11 source code files (written in 'C') plus a makefile. The makefile and 9 of the 11 source code files are generated automatically by UIM/X, based on the various components selected in the interface design and the properties defined for those components; the total size of these 10 files is 103 K. The other 2 source code files, *rule_parse_and_apply.c* (99 K) and *rule_dev_interface.c* (128 K) were written by the author. The former file contains the syntax parser and the rule firing engine; this file is also used as a component of the numeral recognition system. The other file contains all the callback routines which define the behaviour of the interface i.e. what actions are triggered when the various buttons are clicked on etc. Of course, in addition to the above, all the source code developed for the various stages of our new recognition system (preprocessing and contour extraction, feature extraction, special feature extraction for small blobs, reconnection of digits in broken pieces, feature filtering) is also used by the interface. After compiling and linking, the resulting executable file is of size 2,4 M.

We begin with an overall description of the interface from top to bottom, after which we will provide more details for every aspect. See Figure 63.

1. The top portion of the interface contains 2 *textfields* and 5 *buttons* which are

used for file opening and handling.

2. Under the horizontal line separator, there are 4 *option menus* corresponding to the 4 modes of operation of the interface: simple visualization of samples (VIEW); first feature marking (1ST FEATURE); clustering training samples into model files (CLUSTERING); gathering statistics for a given model file and writing rules into the corresponding rule file (STATISTICS).
3. Under the 4 option menus, we have the CONDITION(S) textfield and the PARSE button; the textfield allows to enter statements using our special syntax for the VIEW or STATISTICS mode; and the button triggers the parsing of the written statements (for syntax verification).
4. The PROCESS_PIECES *toggle-button*, when activated, allows to see the results of special feature extraction steps for processing small blobs or reconnecting samples broken into several disjoint pieces.
5. The (black) *drawing area* is where the result of feature extraction is displayed in specific colors associated to each feature type.
6. To the right of the drawing area is a large *scrolled text* window where the values of several attributes for each feature are listed; this is also used to display various messages to the user of the interface.
7. Under the drawing area, are the NEXT IMAGE and PREVIOUS IMAGE buttons, whose role is obvious.
8. Under the scrolled text area, is the LIST_FEATURES toggle-button, the FILTERING toggle-button and a LIST AGAIN button.
9. The END_SESSION button performs several maintenance operations before terminating the interface program.

G.1 File Handling

The RLC BASE FILE textfield allows the user to enter the full name (including path) of a file which is part of a database (in our work, such files have the extension `.rlc`). For example,

```
/home/guest/ocr/isolated_num/abt/a/a-2.rlc
```

The associated buttons OPEN BASE and RESET BASE are used to initially open the file and then to rewind it to the beginning, after some processing has been done. The MODEL FILE textfield allows the user to enter the full name of a model file. When a model file is processed, the name of the base file from which samples are currently drawn is written automatically into the RLC BASE FILE textfield. There are 3 buttons for model file handling. The OPEN MODEL and RESET MODEL buttons play the same role as their counterpart for RLC files; the CANCEL MODEL button closes the model file and associated base files and clears other areas of the interface.

G.2 The Option Menus

G.2.1 The VIEW Menu

The VIEW menu offers 3 options:

- VIEW All: Allows to visualize the result of feature extraction on consecutive samples in either a base (RLC) or a model file; it is possible to move forward or backward in the file with the NEXT IMAGE and PREVIOUS IMAGE buttons. When the LIST FEATURES toggle-button (located under the large scrolled text area) is on, the values of several parameters are listed:
 - The number of rows, the number of columns, and the estimated stroke-width.
 - For each connected component of the image, the number of contour points and, for each feature extracted from that contour, a list of attributes: indices of initial, focal, and last contour points associated with that feature,

feature type, cumulated deviation angle for that feature and for the inter-arc region between the preceding and current feature, direction of the feature, merge-count of the feature; finally, the total number of endpoints, cavities and bends for that blob.

- For each hole, its pixel area and the coordinates of its minimum bounding box.

When the LIST FEATURES toggle-button is off, only the rank of the sample currently displayed in the (black) drawing area is written to the scrolled text area.

- VIEW Conditional: Allows to visualize the result of feature extraction only for those samples which satisfy certain conditions, entered in the CONDITION(S) textfield using the syntax detailed in Appendix F. Several conditions can be written, separated by semi-colons. Again, it is possible to move forward or backward in the file (either a base file or a model file) with the NEXT IMAGE and PREVIOUS IMAGE¹ buttons; the interface processes samples consecutively, skipping those who do not satisfy the conditions and displaying only the next (or previous) sample which does. The PARSE button must be clicked and the rules must be parsed with success before the NEXT IMAGE and PREVIOUS IMAGE buttons will perform their expected work.
- VIEW Given_rank: Allows to visualize the result of feature extraction for a particular sample by specifying its rank within the file². Then this option is selected, a small dialog box pops up which provides a textfield for entering the rank of the desired sample, and an OK button to trigger the search and display the sample of interest. This small dialog box remains active (for visualizing other specific samples) until one of the other options (VIEW All or VIEW Conditional) is selected again.

¹With the VIEW Conditional option, a maximum of 10 preceding images are kept at all times; the PREVIOUS IMAGE button cannot be used to back off beyond that point.

²This is useful for investigating why certain samples have been rejected or erroneously classified.

In the VIEW mode of operation, the PROCESS_PIECES toggle-button, located just above the drawing area, can be activated. This causes the PREVIOUS IMAGE button to disappear and 2 new buttons to appear, labeled SMALL PIECES and CONNECT PIECES. When a sample has more than one connected component, the initial display in the drawing area shows the result of the feature extraction stage (as described in Section 6.4) on each connected component. It is then possible to click on the SMALL PIECES and CONNECT PIECES buttons to perform other feature extraction functions and visualize their results. The SMALL PIECES button triggers the erasing of very small blobs and the extraction of improved endpoint regions on other small pieces and displays the new results. The CONNECT PIECES button launches functions that attempt to reconnect the disjoint pieces together and also displays the new result.

In the VIEW mode of operation, the FILTERING toggle-button, located at the bottom under the scrolled text area, can also be activated. This causes the PREVIOUS IMAGE button to disappear and one new button to appear, labeled FILTER FEATURES, just under the toggle-button. When a sample and its extracted features are displayed in the drawing area, clicking on the FILTER FEATURES button applies all the feature filtering operations to the feature list. As a result, a new display is shown with the corresponding (possibly amended) list of feature attributes in the scrolled text area.

G.2.2 The 1ST FEATURE Menu

The 1ST FEATURE option menu is used to assign a starting feature to every sample of a base (RLC) file or to discard the sample. This menu offers 5 options:

- 1ST FEATURE Not Active: This option is the initial setting and must be returned to before other option menus can be used.
- 1ST FEATURE Test & Show: This option is used to experiment with various possible conditions for defining the starting feature. The tests are written in the CONDITION(S) textfield and then parsed by clicking on the PARSE button;

then the NEXT IMAGE and PREVIOUS IMAGE buttons can be used to view the result. When the interface user is satisfied, the next option can be selected.

- 1ST FEATURE Store Rules: This option allows one to copy what is written in the CONDITION(S) textfield into a special rule file called *stored_rules.d* (where 'd' is a digit from 0 to 9); once these rules are stored, they can be used to assign a starting feature to samples of any RLC file of the same digit class. This is done with the next two options. Note that all 10 *stored_rules.d* files are kept in a subdirectory called *stored_rules*.
- 1ST FEATURE Read Rules: After writing the name of a base file in the RLC BASE FILE textfield, the user can select this option. As a result, the rules defining the starting feature for that class are automatically read from the appropriate *stored_rules.d* file, written into the CONDITION(S) textfield, and PARSEd. Then the next menu option can be used.
- 1ST FEATURE Mark & Verify: This option is used to create the starting *model file* associated with the specified base file³. When the Mark & Verify option is selected, the PREVIOUS IMAGE button disappears and 2 new buttons appear under the NEXT IMAGE button; they are labeled EDIT 1ST and DISCARD IMAGE. With this option, the use of the 3 buttons is as follows:
 - NEXT IMAGE: If a sample was already displayed in the drawing area with its proposed starting feature highlighted, clicking the NEXT IMAGE button indicates acceptance of this sample and its starting feature as displayed; thus the rank of this sample and the index of its starting feature are appended as a new line in the model file under construction. Then the next sample is processed and displayed: the first feature in its feature list which satisfies the rules in the CONDITION(S) textfield is highlighted as proposed starting feature.

³A starting model file must be created for any base file to be used as training material for the classifier.

- EDIT 1ST: If the feature highlighted as starting feature in the drawing and scrolled text areas is not appropriate, or if no feature at all satisfied the starting rules, the user can manually assign a starting feature by clicking on the EDIT 1ST button. This causes a small dialog box to appear and the user can enter the index⁴ of the appropriate feature in a textfield and then click the OK button of that dialog box; the image is displayed again with the new starting feature highlighted; corrections can be made, as many times as necessary. When satisfied, the user clicks the NEXT IMAGE button to record the information and move on to the next sample. Note that the EDIT 1ST button disappears the first time it is clicked on and never comes back; instead the small dialog box remains visible thereafter and can be used to make corrections on any sample if needed.
- DISCARD IMAGE: If the sample displayed should really *not* be a member of the class of interest, the user clicks on this button. The sample is then not included in the model file under construction; instead it is stored in a special (model) file of discarded samples in a separate subdirectory (called *discarded*). In that model file, the index of its starting feature is set to -1.

We end with an example to clarify matters so far. Assume that we started with a run-length encoded file containing 200 samples of numeral class ‘2’, from training set A of the CENPARMI database. The name of this file is *a-2.rlc*. After processing this file, using the 1ST FEATURE Mark & Verify menu option, a model file was created called *a-2.start*; this file begins with the following 4 lines:

```
/home/guest/ocr/isolated_num/abt/a/a-2.rlc 199
1 0
2 0
3 1
```

The first line indicates the name of the (single) base file from which this model file is constructed; the number 199 indicates that only one sample was discarded in the Mark & Verify operation. The following lines indicate the ranks of the samples in the

⁴The LIST_FEATURES toggle-button should be activated during the Mark & Verify operation to facilitate the entry of the valid index.

file *a-2.rlc* and the index of their starting feature. Note that 0 is the index of the first feature in the feature list.

G.2.3 The CLUSTERING Menu

The CLUSTERING option menu is used in conjunction with a model file containing only valid (i.e. ≥ 0) starting feature indices. This menu offers 2 options:

- **CLUSTERING Not Active:** This is the initial setting and must be returned to before using other option menus or even before clustering another model file.
- **CLUSTERING Proceed:** This option is used to assist in the clustering of the starting model file. The interface will
 - display the sample image with its starting feature highlighted;
 - display the list of existing model files;
 - allow the user either to indicate in which of the existing model files the current sample should be added or to indicate the name of a new model to be initialized with the current sample.
 - update all files automatically as the clustering proceeds.

When the CLUSTERING Proceed option is selected, a small dialog box appears which asks the user to indicate/confirm the name of the directory where the models should be created or already are. After this step is cleared, the list of names of existing model files is displayed in the scrolled text area, each name preceded by a numerical index; and another dialog box is popped up which allows the user to enter that index to assign the sample to an existing model or to enter the name of a new model to create.

G.2.4 The STATISTICS Menu

The STATISTICS option menu is used in conjunction with a model file to help develop and write the specific classification rules for that model. A model file regroups samples

which have a similar shape for their entire contour or for a certain portion of the contour. This menu also offers 2 options:

- **STATISTICS Not Active:** This is the initial setting and must be returned to before using other option menus.
- **STATISTICS Proceed:** This option is used to obtain statistics concerning the values of a specific attribute of a given feature on each sample belonging to that model. When the **STATISTICS Proceed** option is selected, a small dialog box appears which asks the user to indicate/confirm the name of the directory where the rule files should be stored. After this step is cleared, another dialog box is popped up which allows the user to type classification rules in a textfield (following our syntax) and have these rules parsed and then appended to the specified rule file. Two new buttons labeled **SAVE NEW F1** and **GET STATISTICS** also appear, respectively above and under the **PARSE** button.

The procedure is as follows: the user enters one or more rules in the **CONDITION(S)** textfield, ending with an 'incomplete' rule i.e. a rule which specifies only a feature attribute (for example *F3d* or *C1/4*) without a logical operator and value(s). These rules are parsed (**PARSE** button) and the **GET STATISTICS** button is clicked. All samples of the model file are then automatically processed and the values of the specified attribute are computed and stored in an array. The range of values taken is displayed as a result in the scrolled text area. The rules may be copied directly from the **CONDITION(S)** textfield to the **RULE** textfield of the smaller dialog box and then stored.

Some rules simply cause feature pointers to move along the feature list, including the starting feature pointer. If the work needs to be interrupted before the classification rules for a particular model are completely developed, it is possible to overwrite the model file with these updated starting features. This is the purpose of the **SAVE NEW F1** button.

G.3 The Drawing Area

The drawing area has a black background. Its fixed dimensions on the screen are 300 points X 300 points. To display the samples, each original (unsmoothed) image is scaled to fit completely within the drawing area, using as much of the area as possible. Thus different scales are applied to different samples. The images are displayed with the extracted feature regions highlighted in different colors. The body of the numeral is *grey*; the contour points of an endpoint region are shown in *cyan*; the contour points of a cavity region are shown in *yellow*; the contour points of a bend region are shown in *green*. Within each feature region, the focal point is displayed in *red*. The starting feature is highlighted by displaying the numeral 1 (black on white background) in the vicinity of its focal point.

Appendix H

Holes in CENPARMI data

For the CENPARMI training data, Table 40 gives 4 counts for each numeral class, corresponding to the number of samples of that class having 0, 1, 2, and more than 2 holes respectively. Note that there are 400 training samples for each class.

Class	Number of holes			
	0	1	2	>2
0	56	334	10	0
1	397	3	0	0
2	174	214	12	0
3	352	45	3	0
4	391	9	0	0
5	364	31	5	0
6	39	343	17	1
7	389	11	0	0
8	10	173	203	14
9	133	254	11	2

Table 40: Sample counts depending on number of holes

There are a number of instances where the above statistics are somewhat surprising. In particular, consider the following classes:

- ‘1’ : 3 samples (0.75%) have 1 hole;

- '5' : 5 samples (1.25%) have 2 holes;
- '6' : 17 samples (4.25%) have 2 holes and 1 sample has more than 2 holes;
- '7' : 11 samples (2.75%) have 1 hole;
- '8' : 10 samples (2.50%) have no hole at all and 14 (3.50%) have more than 2 holes;
- '9' : 11 samples (2.75%) have 2 holes and 2 even have more than 2 holes;

Figure 91 gives examples of samples with an unusual number of holes. In some cases, the anomaly is caused by the writing style of the person. In other cases, tiny spurious holes within the stroke width are responsible for the problem; these holes are due to writing accidents and/or binarization thresholds.

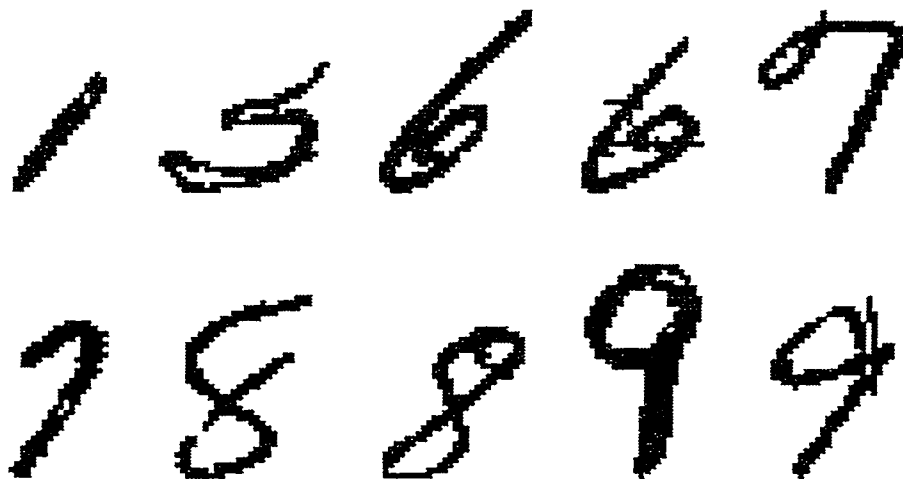


Figure 91: Examples of unexpected number of holes

Appendix I

General Implementation Information

Our handwritten numeral recognition system is a program which consists of 10 source code files written in 'C', and 123 rule files. Table 41 lists the program files and their respective sizes.

	Program file name	File size (Bytes)
1.	new_recog_main.c	11 929
2.	init_finish.c	8 491
3.	get_edges.c	73 001
4.	get_objects.c	18 865
5.	get_features.c	181 270
6.	display_message.c	1 691
7.	many_pieces.c	41 164
8.	classify.c	69 321
9.	rule_parse_and_apply.c	101 122
10.	feature_filtering.c	23 439
	TOTAL	530 293

Table 41: Program files in system

The first file is the program's *main* function, including calls to key functions in

the various other files. The second file contains initialization and finalization code performed at the very beginning and at the very end of program execution as well as linked list maintenance which is executed after processing each sample. Files #3 and #4 are the implementation of the preprocessing of Chapter 4. File #5 is the feature extractor presented in Section 6.4. The sixth file defines a `DISPLAY_MESSAGE` function which is used in several of the other files, mostly for error messages to the user, in case of problems; the messages can be displayed either to the `stderr` stream or to the scrolled text area of the development interface. File #7 implements the handling of sample images composed of several connected components as discussed in Section 8.1. File #8 contains the code to read all the rule files when the program is launched and then to orchestrate the firing of the various rule files in the classification stage. The syntax parser and rule application code introduced in Section 7.2.4 are found in file #9. Finally, the feature filtering operations of Sections 8.4 and 8.5 are implemented in file #10.

The size of the executable file for the complete recognition system is 385 024 bytes. Simple tests were performed with the CENPARMI training set *A* (2 000 samples) and with the CEDAR training set *cedar1* (2 000 samples) to obtain estimates of execution time. On a SPARC10 workstation, the total processing time was 8.0 ms per sample for the first set (166 PPI) and 15.1 ms per sample for the second set (300 PPI). The corresponding processing rates were 126 samples/sec. and 66 samples/sec. respectively. On a SPARC Ultra workstation, processing rates are doubled. From these figures, it is clear that our system is quite efficient. The classification stage alone takes approximately 23% of total processing time for the 166 PPI images and 15% for the 300 PPI images.

Table 42 presents, for each class of the partial classifier that was developed, the number of rule files and the number of individual rules and the total byte size of these rules.

Class	Number of rule files	Number of rules	Total size (bytes)
0	20	329	1 758
2	67	693	5 621
6	36	920	7 443
TOTAL	123	1 942	14 822

Table 42: Numbers of rule files and rules

Appendix J

Number of Pieces in Numerals

This appendix provides, for each numeral class, the number of samples which have 1, 2, 3, 4, and more than 4 connected components. The sample counts are given for the 13 954 training samples used from the CEDAR database, for the 5000-sample free-style training set of the Concordia-Montreal database, and for both training sets of the ITRI-Taiwan database (one with 4 000 samples and one with 244 27 samples).

Class	Samples with number of pieces					Total
	= 1	= 2	= 3	= 4	> 4	
0	1384	13	2	0	1	1400
1	1396	4	0	0	0	1400
2	1378	20	2	0	0	1400
3	1377	23	0	0	0	1400
4	1367	32	0	1	0	1400
5	1063	330	5	1	1	1400
6	1393	7	0	0	0	1400
7	1389	10	1	0	0	1400
8	1388	12	0	0	0	1400
9	1341	13	0	0	0	1354
Total	13476	464	10	2	2	13954
Percent	96.57	3.33	0.07	0.01	0.01	100.00

Table 43: CEDAR database: Statistics on number of pieces

Class	Samples with number of pieces					Total
	= 1	= 2	= 3	= 4	> 4	
0	472	21	7	0	0	500
1	494	5	1	0	0	500
2	488	8	3	1	0	500
3	485	15	0	0	0	500
4	467	30	3	0	0	500
5	416	80	3	1	0	500
6	493	7	0	0	0	500
7	487	13	0	0	0	500
8	485	13	2	0	0	500
9	472	23	4	1	0	500
Total	4759	215	23	3	0	5000
Percent	95.18	4.30	0.46	0.06	0.00	100.00

Table 44: Concordia-Montreal database: Statistics on number of pieces

Class	Samples with number of pieces					Total
	= 1	= 2	= 3	= 4	> 4	
0	364	26	8	1	1	400
1	390	8	1	1	0	400
2	368	27	4	1	0	400
3	352	40	6	2	0	400
4	354	39	4	2	1	400
5	314	75	9	1	1	400
6	364	30	5	1	0	400
7	358	31	8	1	2	400
8	355	39	3	3	0	400
9	373	19	5	1	2	400
Total	3592	334	53	14	7	4000
Percent	89.80	8.35	1.33	0.35	0.20	100.00

Table 45: Taiwan tw1-tw4 database: Statistics on number of pieces

Class	Samples with number of pieces					Total
	= 1	= 2	= 3	= 4	> 4	
0	2102	308	101	31	12	2554
1	2871	78	1	0	0	2950
2	2636	245	29	8	0	2918
3	2333	353	69	12	2	2769
4	1887	222	33	12	0	2154
5	1572	561	79	15	2	2229
6	1829	209	68	12	2	2120
7	2152	234	38	9	3	2436
8	1811	199	54	9	6	2079
9	1978	169	59	8	4	2218
Total	21171	2578	531	116	31	24427
Percent	86.67	10.55	2.17	0.47	0.13	100.00

Table 46: Taiwan t20-t24 database: Statistics on number of pieces