COMBINING MULTIPLE CLASSIFIERS FOR PEN-BASED HANDWRITTEN DIGIT RECOGNITION

by
Fevzi Alimoğlu
B.S. in Computer Engineering, Boğaziçi University, 1994

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirement for the degree of Master of Science
in
Computer Engineering

Boğaziçi University
1996

COMBINING MULTIPLE CLASSIFIERS FOR PEN-BASED HANDWRITTEN DIGIT RECOGNITION

APPROVED BY:

Yrd. Doç. Dr. Ethem Alpaydın ....
(Thesis Supervisor)

Doç. Dr. Levent Akın ....

Doç. Dr. Yağmur Denizhan ....

DATE OF APPROVAL ....

ACKNOWLEDGEMENTS

I would like to thank to my thesis advisor Dr. Ethem Alpaydın for his invaluable support and understanding during my thesis work. His ideas affected my academic background since my undergraduate senior project.

I thank Dr. Levent Akın and Dr. Yağmur Denizhan, for participating my thesis committee and for their suggestions on my thesis report and defense.

I also thank to Nahit Emanet who provided me routines to get input data from the tablet. I would also like to express my sincere thanks to my roommates Ali Taylan Cemgil and Serhat Özdemir for their stimulating discussions and their useful suggestions.

I also thank Osman Burak Önal and Mustafa Akgül who helped me a lot during the painful process of writing the thesis in LaTeX.

Finally, my special thanks go to my family and to Sevgül Bektaş for their encouragement and moral support throughout all phases of this thesis.

ABSTRACT

Handwriting recognition has attracted enormous scientific interest because of its potential for improved man/machine interfaces. We have designed an on-line handwritten digit recognition system after the examination of different techniques based on statistical and neural pattern recognition approaches.

We collected a digit database from 44 people. We use two representations. The dynamic representation is based on constant length feature vectors of equally distanced points on the pen trajectory. The static representation converts the dynamic information to an image similar to images used in off-line recognition tasks.Then, we tested the well known statistical classification method k-nearest neighbor (k-NN) and neural multi-layer perceptron (MLP) and recurrent networks using both representations.

Classifiers trained with dynamic and static representations make misclassifications for different samples. We combine them first by forming a feature vector composed of dynamic and static representations. In order to achieve higher accuracy, we combine different classifiers using voting, mixture of experts, stacked generalization and cascading. The classifiers' selection criteria is based on the similarity among individual learners. We combine one MLP trained with dynamic data and one MLP trained with static data.

We provide the results for all methods and make comparisons in terms of generalization accuracy, memory requirement and learning time. We conclude that using two representations increase accuracy considerably without significantly increasing the memory requirements and learning time.

ÖZET

Gelişmiş insan/makine arabirimleri ile ilgili potansiyelinden dolayı, elyazısı tanıma cok büyük bilimsel ilgi çekmiştir. İstatistiksel ve yapay sinir ağı temelli yaklaşımlara dayalı olan birçok teknik inceledikten sonra, elle yazılmış rakamları tanıyan bir sistem tasarladık ve gerçekleştirdik.

Tanıma amaçlı olarak 44 kişiden bir rakam veritabanı topladık. Önişleme sonrasında iki gösterim kullanıyoruz. Dinamik gösterimde örnekleme noktaları sabit uzunlukta bir öznitelik vektörüne dönüştürülüyor. Bu noktalar kalem yolu uzerinde birbirlerinden eşit uzaklıktadır. Statik gösterimde ise dinamik bilgi bir görüntüye çevriliyor. Daha sonra çok bilinen istatistiksel sınıflandırm a metodu en yakın k-komşu (k-NN), yapay sinir ağı temelli çok katmanlı perseptron (MLP) ve özbağlı ağlarının bu gösterimler üzerindeki başarısını deniyoruz.

Dinamik ve statik gösterimleri öğrenen sınıflandırıcılar farklı örnekler üzerinde başarısız oluyor. İlk olarak hem dinamik hem de statik öznitelik vektorlerini birbirine ekleyerek yeni bir öznitelik vektörü elde ediyoruz. Daha yüksek başarım elde edebilmek için sınıflandırıcıları oylama, uzmanların karışımı, yığınlaştırılmış genelleştirme ve ardışık sınıflandırıcılar ile birleştiriyoruz. Sınıflandırıcı seçme kriteri sınıflandırıcının birbirine benzerlığıne bağlı. Birleştirme için dinamik verileri öğrenmiş bir MLP ve statik verileri öğrenmiş bir MLP seçiyoruz.

Tüm metodlar için sonuçları sunuyoruz ve aralarında genelleştirme doğruluğu, bellek ihtiyacı ve öğrenme zamanına göre karşılaştırmalar yapıyoruz. Sonuç olarak iki gösterimi kullanmak, bellek ihtiyacını ve öğrenme zamanını önemli ölçüde arttırmadan, doğruluğu önemli ölçüde yükseltiyor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

d_{ji}	Estimate of a learner
f	Combiner function for different learners
H_h	Hidden unit of a MLP
k	Number of neighbors in k-NN
m_i	Decision variable for k-NN
N	Number of resampling points or reslution of static image
O_j	The output of a MLP
P_c	Normalized output of a MLP
$P(C_i\|x)$	Posterior class probability
Q_{hk}	Recurrent weights between the hidden units of a MLP
r_i	Final estimate of combined learners
R_c	Desired output for a MLP
T_{ch}	Weights between hidden and output layers of a MLP
w	Effect of a neighbor in k-NN
x_j	Input feature vector
x_t	Horizontal coordinate of a sample point
y_i	Output vector
y_t	Vertical coordinate of a sample point
W_{hi}	Weights between input and hidden layers of a MLP
Z	Normalization factor in k-NN
β_j	Contributions of a classifier
δ	Scaling factor in normalization
ϕ	Sigmoid activation function
ψ	Combination model parameters

1. INTRODUCTION

Handwritten character recognition has attracted enormous scientific interest due to its evident practical utility. It is often needed to make bulk data entry. If data are already written, we use scanners and off-line Optical Character Recognition (OCR) systems. If these data are to be captured and processed as written, like in speech and handwriting, we use on-line systems. On-line handwriting is important where keyboards are difficult to use, e.g. when the writer is mobile.

The primary goal of this thesis is to construct a high-performance system that exploits the difference between two different representations of the same handwritten digit. One representation is dynamic, that is, the movement of the pen as the digit is written. The other representation is static, that is, the image generated as a result of the movement of the pen tip. We see that the two representations make mistakes for different samples and thus look for ways to combine them efficiently.

We create a digit database by collecting 250 samples from 44 writers. The samples written by 30 writers is used for training, cross-validation and writer dependent testing, and the digits written by the other 14 are used only for writer independent testing.

Our preprocessing converts raw data coming from the pressure sensitive tablet into fixed length feature vectors. Dynamic representation is similar in nature with the input signal. Static representation is based on the final image of the writing, similar to that used in off-line tasks.

After using two common classification methods k-Nearest Neighbor and Multi-layer Perceptrons (MLP) for both dynamic and static representations, and also recurrent neural networks for dynamic representation only, we focus on combining MLPs trained using samples of these two representations. We examine voting, stacked generalization, mixture of experts and cascading techiniques for combining the two MLPs. Finally, we define a similarity measure for classifiers and a general model applicable to most classifier combination methods.

A handwriting recognition system must contain a number of components (as shown in Figure 1.1):

1. The *Digitizer* captures the handwriting using a scanner or a tablet and stores in electronic format, ready to be processed by the rest of the system.

2. The *Preprocessor* eliminates digitizing errors and noise. Additionally, it reduces meaningless variability for the next stages. Typical steps are segmentation, size and slant normalization, resampling, downsampling, etc.

3. The *Feature Extractor* determines which properties of the preprocessed data are most meaningful and should be used in later stages. These features may be the velocity, acceleration, curvature data of the stylus derived from the handwriting input. The resulting feature vector is supposed to be the best possible representation of the input.

4. The *Recognizer* is the most important part of the system. It may use a syntactic or statistical approach. In the syntactic approach, a pattern is represented as a string, a tree or a graph of pattern primitives and their relations. The decision is made using a syntax analysis or a parsing process. In the statistical approach, a pattern is represented by a N-dimensional feature vector and the decision is made according to a similarity measure such as a distance metric or a discriminant function. Nowadays, the state-of-the-art techniques include artificial neural networks, multistage classification methods and multiexpert combination methods.

5. A *Postprocessor* such as dictionary checking is needed to make use of contextual information when we want to attain acceptable recognition accuracy especially in cursive writing.

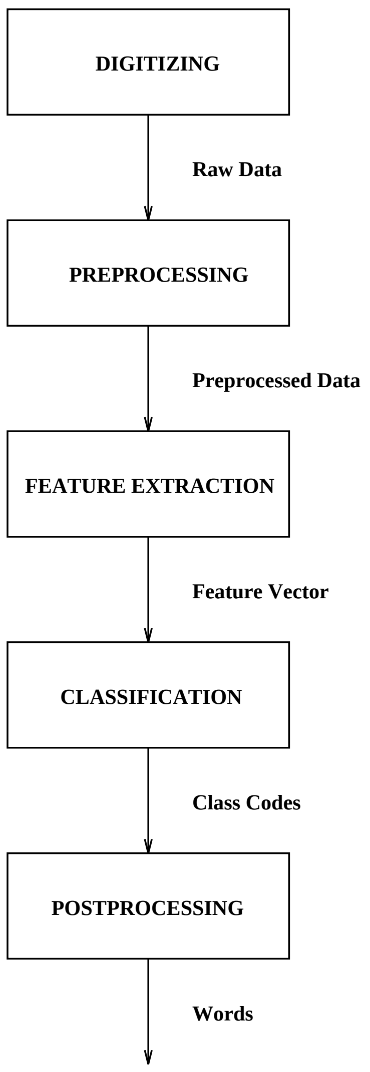

Figure 1.1: Stages of handwriting recognition.

2. HANDWRITING

Every written language has an alphabet of characters that are used to make communication possible. Differences between characters are usually more significant than differences between different samples of the same character.

Handwriting types can be divided into four groups:

1. *Discrete handwriting* which is also called handprint means that the user is constrained to write either within predefined boxes or with significant spaces between characters, i.e., segmentation is done by the user.
2. *Run-on mode* is similar to discrete, but slightly less constrained. Characters are allowed to touch each other. A single stroke cannot include more than one character. The candidate segmentation points are always at pen lifts.
3. *Cursive handwriting* is when almost all the letters forming a word are connected, except cases such as dots, umlauts, accents or crosses. Recognition can be performed at the word level or character level. Segmentation is a difficult task in this case. A dictionary checking step is employed at the last stage to get higher accuracy.
4. *Unconstrained handwriting* is the most common way of writing, a mixture of discrete and cursive modes.

2.1 On-line Character Recognition

In on-line or pen-based recognition, the input is a temporal sequence of the movements of the pen tip over a pressure sensitive tablet. Handwriting can be represented as a time-ordered sequence of coordinates of the pen tip on the tablet with varying speed and pressure at points. The ordering of the sequence of points and their positions give the necessary information for recognition.

Characters are entered from a touch terminal (tablet). The touch terminal is convenient for users who prefer a touch interface over a keyboard and mouse and it is a portable input device unlike conventional terminals and personal computers. In on-line mode, there is immediate feedback from the recognizer to the writer, which potentially enables the user to adjust his or her writing to the system.

A comprehensive review of online handwritten digit recognition is given by Tappert et al. [1] who also discuss hardware requirements, basic handwriting properties and fundamental recognition problems. Preprocessing techniques for segmentation, noise reduction, normalization and classification techniques such as decision trees based on features and dynamic programming are presented, but statistical methods are mentioned only briefly.

In a study by Guyon et al. [2], Time Delay Neural Networks (TDNN) have been used for the recognition of handwritten characters. A set of 12,000 digits and uppercase letters from 250 writers was used for training and 2,500 characters from other writers made up the testing set. The classification success exceeded 96 percent.

Manke and Bodenhausen [3] use a Multi-State Time Delay Neural Network (MS-TDNN) for single character and cursive handwriting recognition. This structure is an extension of the TDNN with a non-linear time alignment structure (Dynamic Time Warping). They obtain up to 97.7 percent word recognition rate both on writer dependent and independent single character and writer dependent cursive handwriting tasks with different vocabulary tasks up to 20,000 words.

TDNN structure was later expanded to include segmentation together with recognition by Schenkel et al. [4]. They describe two approaches for segmentation. The first method uses a combination of heuristics to identify particular pen-lifts as tentative segmentation points. The second methods relies on the empirically trained recognition engine for both recognizing characters and identifying segmentation points.

Hidden Markov Models (HMM) are among the most popular methods used for online handwriting, especially cursive. Nathan et al. [5] use a left to right HMM for each character that models the dynamics of the written script. The output probabilities at each arc of the HMM are represented using Gaussian mixtures. Their experiments resulted in a significant decrease in error rate for the recognition of discrete characters compared with elastic matching techniques.

TDNN and HMM have been used together by Schenkel at al. [6] in a system for cursive script and hand-point writing. They use a TDNN to estimate a posteriori probabilities for characters in a word. A HMM segments the word in a way which optimizes the global word score, taking a dictionary into account. The system, trained on 20,000 words from 59 writers, using a 25,000 words dictionary, achieves 89 percent character and 80 percent word recognition rate for a writer independent test set.

Cho and Kim [7] use a structure composed of multiple neural networks based on the fuzzy integral. Object evidence is combined nonlinearly in the form of fuzzy membership function, with subjective evaluation of the worth of the individual neural networks, each having a different number of input vector sizes, with respect to the decision. Their results are superior to voting techniques for numeral, lowercase and uppercase character recognition tasks.

Hoffman et al. [8] developed a a model that assumes that handwritten charcters are formed from a limited number of primitive hand motions characterized by the $x - y$ oscillations in th vertical (y) and horizontal (x) directions. These features are assumed to control neural "motor" activities. Their initial 50 percent success on isolated cursive script characters is improved by using a cluster of smaller networks and adding heuristics up to 80 percent.

2.2 Off-line Character Recognition

Off-line recognition requires first digitizing then recognition. Recognition is performed after the writing is completed. A person writes on paper and the handwriting is captured via a scanner or a camera, and then stored in a binary or gray scale digitized form. Unlike the on-line case, off-line recognition may be performed days, months or even years later. Most work has been on machine printed characters, but efforts are also made in order to recognize handwriting.

This technique cannot use any dynamic writing information: the number of strokes, the order of strokes, and the direction of stroke creation cannot be extracted from the final image. This makes it difficult to extract and identify the strokes of each character. However it is possble to treat off-line data as dynamic data by inferring the dynamic information. Conversely, the dynamic data can be easily converted to static using scan conversion algorithms.

A recent overview for off-line handwriting recognition is given by Suen et al. [9]. A summary of performace for digit recgnizers for the zip code tasks is given by Srihari [10].

Lee and Pan [11] present a new approach to representation and recognition of handwritten numerals. They first transform two-dimensional (off-line) static images of digits into three dimentional spatio-temporal representation by identifying the tracing sequences based on a set of heuristic rules acting as transformation operators. Local feature points are extracted using a multiresolution critical-point segmentation method, at varying degrees of scale and coarseness. They use a globally competitive and locally cooperative neural network for recognition. They achieve about 97 percent recognition rate on the zip code numerals acquired by the U.S. Postal Service. However, the timing information is not recovered.

3. DATA COLLECTION AND ANALYSIS

The data collection procedure has an important impact on the handwriting recorded. We want to prevent unwanted influences that will cause “unnatural” inputs. The subjects must behave as they do in a real life situation, otherwise the samples taken will not reflect correctly the nature of handwriting and the success of the recognition algorithms will be poor.

3.1 Data Collection Procedure

We used a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus. The input and display areas are located in the same place. Attached to the serial port of an Intel 486 based PC, it allowed us to collect handwriting samples. This tablet sends x and y tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 miliseconds.

Our data collection and displaying software was developed in MS-DOS environment using the Borland C++ 3.1 compiler. We ignored the pressure level values and we did not use them. The tablet resolution is 4157 by 3118 pixels. We mapped this to standard VGA resolution which is 640 by 480.

Subjects were volunteers from the Department of Computer Engineering of Boğaziçi University staff members and students. They were told that the writing would be recorded by the computer for “later usage.”

We collected samples of ten handwritten digits from 44 writers. These writers were asked to write 250 digits in random order inside boxes of 500 by 500 tablet pixel resolution. Subject were monitored only during the first entry screens. Each screen

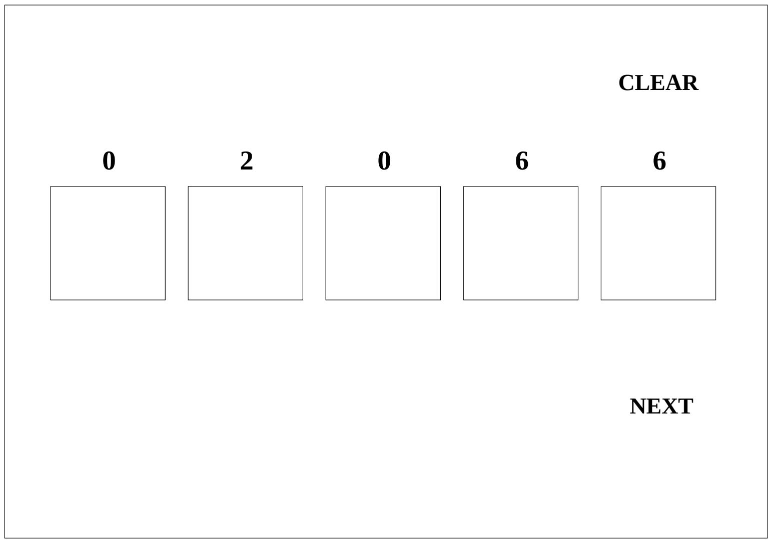

Figure 3.1: Tablet input screen.

contained five boxes with the digits to be written displayed above (Fig. 3.1). Subjects were told to write only inside these boxes. If they made a mistake or were unhappy with their writing, they were instructed to clear the content of a box by using an on-screen button. Progress was controlled using on-screen buttons. The first ten digits were ignored because most writers were not familiar with this type of input devices, but subjects were not aware of this.

3.2 Data Analysis

In our study the samples are digits of ten classes. But this does not imply that we have only ten shapes to distinguish. The digits were written by people having various styles of writing.

1. Every subject has one or more own prototype for each digit (Fig. 3.2).

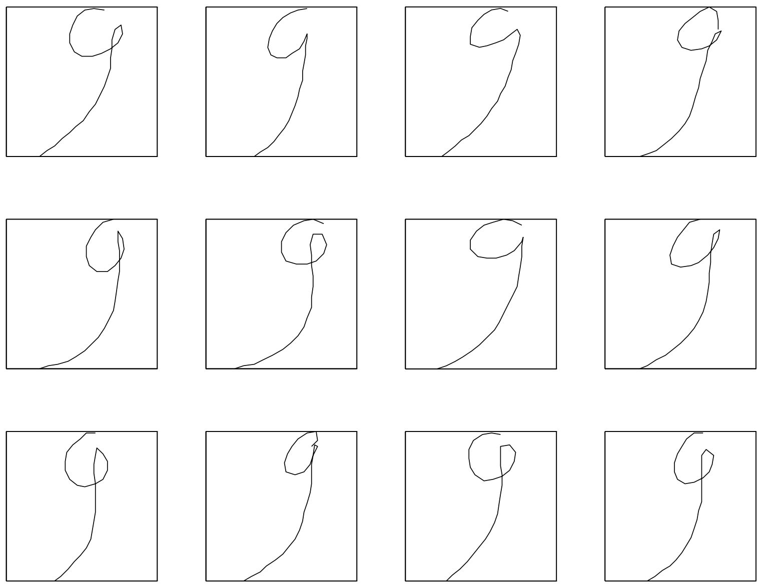

Figure 3.2: The digit "9" written by subject number 4.

2. Depending on various factors, the raw data is composed of vectors of different number of points.
3. Most of the subjects, but not all of them, wrote the digits about in the center of the boxes, however some of them wrote them bigger and some smaller.

These facts show us that we need preprocessing in order to convert data into a normalized form for improved recognition.

3.3 Data Inspection

The samples collected should be processed before using them for learning and recognition. We inspected our sample database using the Upview UNIPEN [12] viewer in SunOS 4.1.3_U1 environment. This inspection (authentication) allowed us to discard samples which were written in wrong places.

4. PREPROCESSING

The raw data captured from the tablet contains enormous variations in writing speed. Different writers write with different speed characteristics. Training a recognizer with such data leads to poor performance. The purpose of the preprocessing is to create an intermediate representation, incorporating to the extent possible, the a priori knowledge of the designer. This includes two complementary processes [13]:

1. Reducing variability which does not help discrimination between classes, therefore building up some invariance.
2. Enhancing variability which does help discrimination between classes.

Various preprocessing methods for on-line handwriting were discussed by Tappert et al. [1], Guerfali and Plamondon [14]. In our problem, the steps of our preprocessing, namely normalization, resampling (for dynamic representation), converting to bitmap, blurring and downsampling (for static representation), greatly reduce the meaningless variability.

Since digits were collected in discrete input boxes, we do not correct rotations. Additionally, irregular samples are not considered, as explained in the previous chapter.

We used MATLAB for data visualization and prototyping our preprocessing routines. Main programs were written in C and and compiled using the GNU C compiler on a DTK SPARC Classic+ machine with SunOS 4.1.3_U1 operating system.

4.1 Dynamic Representation

In our study, we used only x and y coordinate information. The stylus pressure level values were ignored since they do not give additional information about the shape of a digit. They are more appropriate for authentication tasks such as signature verification and writer identification.

4.1.1 Normalization

First we applied normalization to make our representation invariant to translations and scale distortions. The raw data that we capture from the tablet consists of integer values between 0 and 500 (tablet input box resolution). In order to normalize data, first we calculate the median (x_m, y_m) of a digit:

$$\begin{aligned} x_m &= \frac{x_{max} + x_{min}}{2} \\ y_m &= \frac{y_{max} + y_{min}}{2} \end{aligned} \tag{4.1}$$

The spreads on the two dimensions are calculated:

$$\begin{aligned} \delta_x &= \frac{x_{max} - x_{min}}{2} \\ \delta_y &= \frac{y_{max} - y_{min}}{2} \end{aligned} \tag{4.2}$$

and the bigger one is chosen as the scaling factor:

$$\delta = max(\delta_x, \delta_y) \tag{4.3}$$

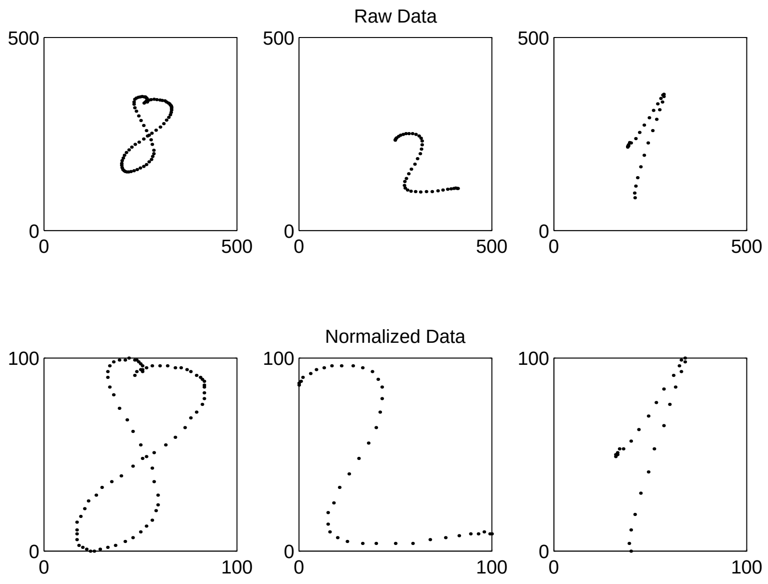

Figure 4.1: The normalization process.

The new coordinates are:

$$\begin{aligned} x_t &= 50 + 50\left(\frac{x_t - x_m}{\delta}\right) \\ y_t &= 50 + 50\left(\frac{y_t - y_m}{\delta}\right) \end{aligned} \tag{4.4}$$

such that the coordinate which has the maximum range varies between 0 and 100. Usually x stays in this range, since most characters are taller than they are wide. However, we observed the fact that the characters in which x has a greater range do exist and we decided to use the greatest δ value. (Fig. 4.1)

We could also have chosen using different scaling factors for the two axes:

$$\begin{aligned} x_t &= 50 + 50\left(\frac{x_t - x_m}{\delta_x}\right) \\ y_t &= 50 + 50\left(\frac{y_t - y_m}{\delta_y}\right) \end{aligned} \tag{4.5}$$

but this normalization would introduce distortions for very narrow digits such as “1” (Fig. 4.2).

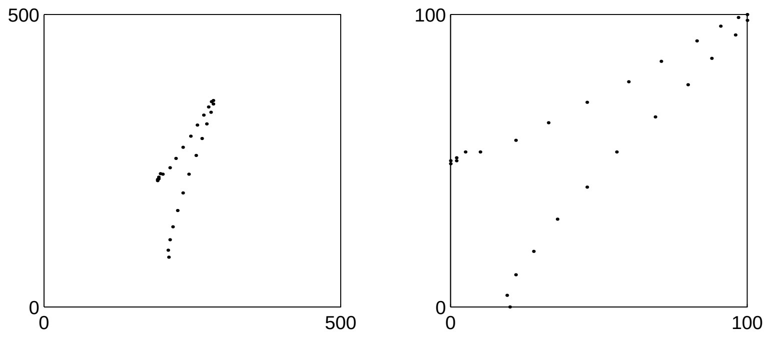

Figure 4.2: Normalization with different δ values causes distortion.

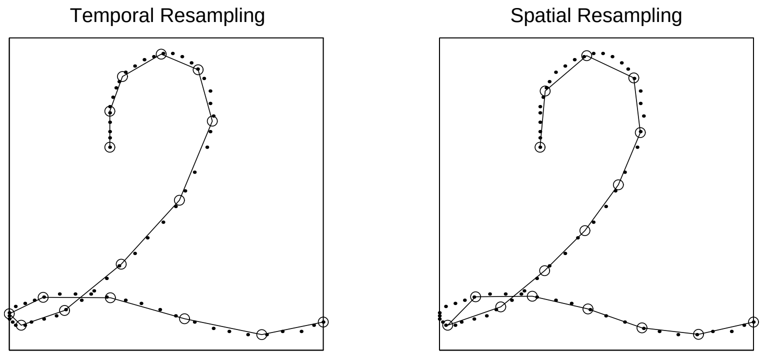

Figure 4.3: Temporal vs. spatial resampling to 16 points.

4.1.2 Resampling

In order to train and test our classifiers, we need to represent digits as constant length feature vectors. A commonly used technique leading to good results is resampling the (x_t, y_t) points. Temporal resampling (points regularly spaced in time) or spatial resampling (points regularly spaced in arc length) [2, 3, 5] can be used here (Fig. 4.3). Raw point data are already regularly spaced in time but the distance between them is variable. Previous research [15, 13] showed that spatial resampling to obtain a constant number of regularly spaced points on the trajectory yields much better performance, because it provides a better alignment between points.

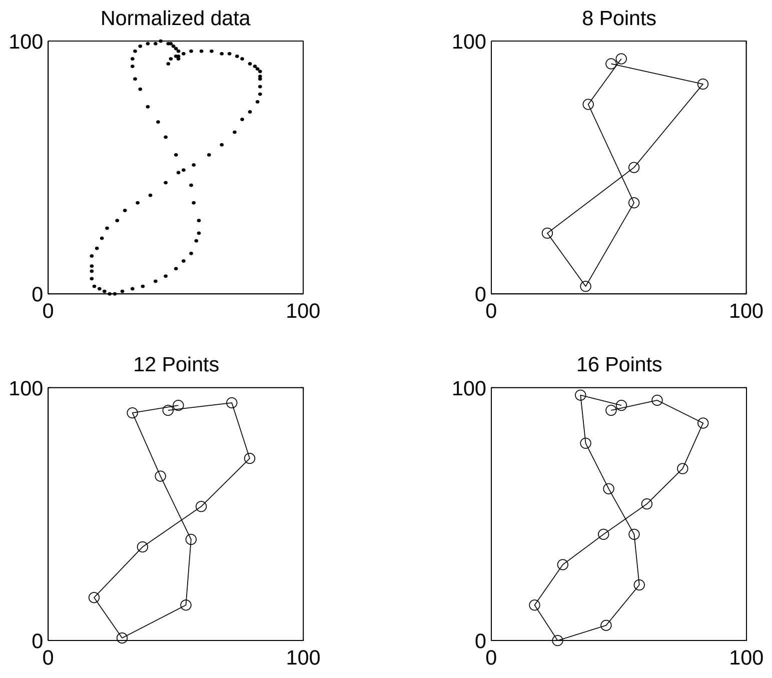

Figure 4.4: Spatial resampling to different number of points.

Our resampling algorithm uses simple linear interpolation between pairs of points. The resampled digits are represented as a sequence of N points (x_t, y_t), regularly spaced in arc length, as opposed to the input sequence, which is regularly spaced in time (Fig. 4.4).

For dynamic representation, the input vector is composed of ordered (x, y) pairs for each resampled point. So, the input vector size is $2N$, two times the number of points resampled. We considered spatial resampling to 8, 12 and 16 points in our experiments.

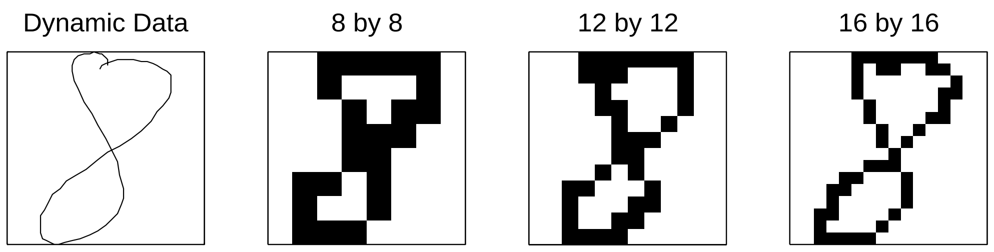

Figure 4.5: Converting to bitmaps of different resolutions.

4.2 Static Representation

The static representation differs from dynamic representation in the sense that it does not incorporate pen motion. Instead, it is based on solely the image of a character, as in off-line recognition tasks. First we convert dynamic data to bitmaps, then we blur it and perform downsampling on it.

4.2.1 Converting to Bitmaps

In order to obtain binary images of the characters, we used Bresenham's scan conversion algorithm [16] to create bitmaps from normalized on-line handwriting data. This algorithm is explained in Appendix A. We iterated through pairs of sequential point samples within each pen stroke.

Since every character in the database is normalized to a unit bounding box we use a square pixel array to store each image. The single parameter for this representation is the resolution (N by N) of the image. We considered three different image resolutions: 8 by 8, 12 by 12 and 16 by 16. Examples are shown in Fig. 4.5.

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

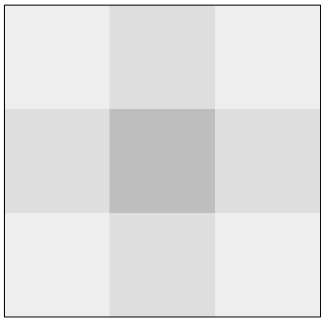

Figure 4.6: The 3 by 3 blurring kernel.

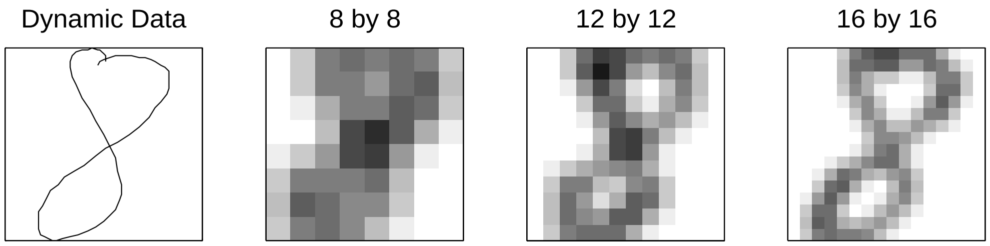

Figure 4.7: Blurred images.

4.2.2 Blurring

In preparing bitmaps for classification the spatial relationship was not preserved. Bitmaps which are usually similar can have very dissimilar feature vectors. Small variations in pen coordinates can produce different images because of quantization with hard thresholds. To solve this problem we smooth the bitmap so that a pixel's value is distributed across its neighbors. We accomplished this by convolving a unit volume blurring kernel (Fig. 4.6) with the image to produce a pixmap, in which pixels can take on arbitrary floating-point values. This kernel can be optimized for each bitmap resolution. A kernel of larger size may be appropriate for bitmaps of greater resolution. Examples are shown in Figure 4.7.

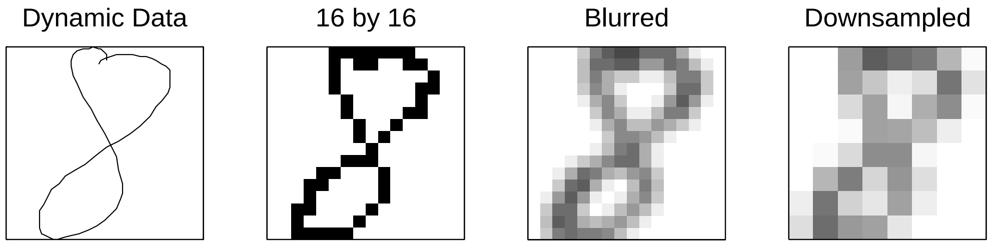

Figure 4.8: The process of obtaining static representation.

4.2.3 Downsampling

Downsampling reduces input dimensionality keeping extra information from a higher resolution blurred bitmap. We calculate the value of every new pixel by averaging the values of four correponding pixels of the larger image. So, the feature vector size is reduced by a factor of four. The output vector for static represenation is composed of N^2 input values for a N by N image (e.g. 8 by 8 bitmaps are obtained from 16 by 16 blurred bitmaps) (Fig. 4.8).

4.2.4 Summary

Figure 4.9 summarizes the final preprocessing method. For each sample, first we normalize the raw input. Then we apply spatial resampling to get the final dynamic representation. The feature vector size is two times the number of points resampled $(2N)$.

The static representation is obtained by converting the normalized data to bitmaps and then applying blurring and downsampling. Note that the dynamic data are converted to a bitmap having a higher resolution than the final image $(2N)^2$. The feature vector size is equal to the number of pixels in the final image (N^2).

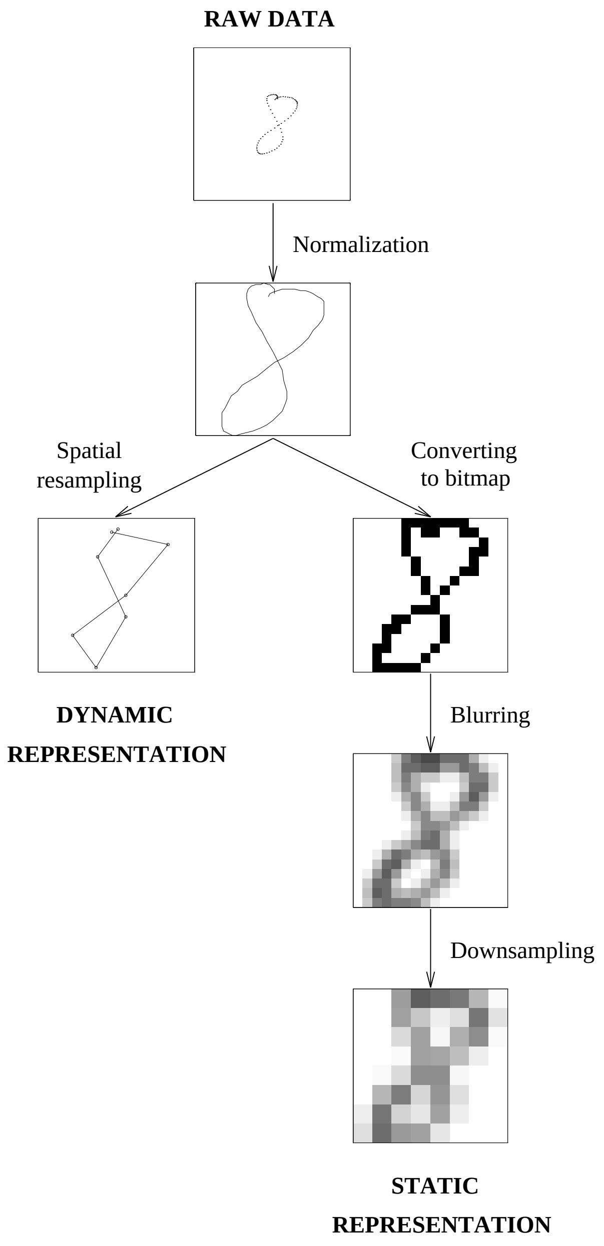

Figure 4.9: The stages of preprocessing.

5. SINGLE CLASSIFIERS

We use two approaches: In the time-delay version, previous inputs are delayed till the final one arrives and thus there is a conversion from time to space. This whole input frame is then given as one static pattern to the classifier. In this approach, we have used k-nearest neighbor and multi-layer perceptron. In the other approach, inputs are fed to a classifier as they arrive in time so the classifier should be able to cope with temporally changing data. In this approach, we have used a multi-layer perceptron with recurrent connections in the hidden layer.

5.1 Time-delay Classifiers

In the time-delay approach, the ordering in time is converted into an ordering in space and the whole input vector is given as input at once. This is a technique originally proposed for speech recognition where the input is also temporal.

5.1.1 K-Nearest Neighbor (k-NN)

k-NN is probably the most popular method used for classification [17]. This is due to its simplicity and high success despite this simplicity. Given a training set of labelled samples and an input pattern to be classified, we find the nearest k samples from the training set and assign the input to the class that is the most heavily represented among these k neighbors. For the case where we use only the nearest

Table 5.1: Weighting functions used by k-NN variants. x is the input and $x_{[j]}$ denotes its jth neighbor. $\|\cdot\|$ is the Euclidean norm.

Variant	w_j: Weighting function
Simple k-NN	1
Gaussian k-NN (Parzen windows)	$\exp\left[-\|x - x_{[j]}\|^2/2(\|x - x_{[k]}\|/3)^2\right]$
Fuzzy k-NN	$1/\|x - x_{[j]}\|^2$

neighbor, i.e. $k = 1$, we assign the class code of the closest character in the training set to the test character. To find the k closest, the most frequently used distance measure is Euclidean that assumes independent, numeric features and equal variances along dimensions. When x is the input and $x_{[j]}$ is its jth closest neighbor chosen from the training set, assuming equal prior class probabilities and to a normalization factor (Z), we compute the posterior class probabilities as:

$$P(C_i|x) = \frac{1}{Z}\sum_{j=1}^{k}\left(\frac{w_j}{\sum_{l=1}^{k} w_l}\right) m_i(x_{[j]}) \tag{5.1}$$

where w_j is the effect of the jth neighbor and $m_i(x)$ is 1 if $x \in C_i$ and 0 otherwise. In simple k-NN that is k-NN proper, each neighbor has equal effect. It has two variants, Gaussian and Fuzzy k-NN, where distant neighbors have less effect. Gaussian k-NN is also known as Parzen windows [17]. The weighting function of fuzzy k-NN [18] is faster to compute than a gaussian. The formulas used are given in Table 5.1.

5.1.2 Multi-Layer Perceptron (MLP)

In a multi-layer perceptron (MLP) [19] as shown in Figure 5.1, the output O_j is a weighted sum of the outputs of a number of nonlinear basis functions, named hidden units, H_h:

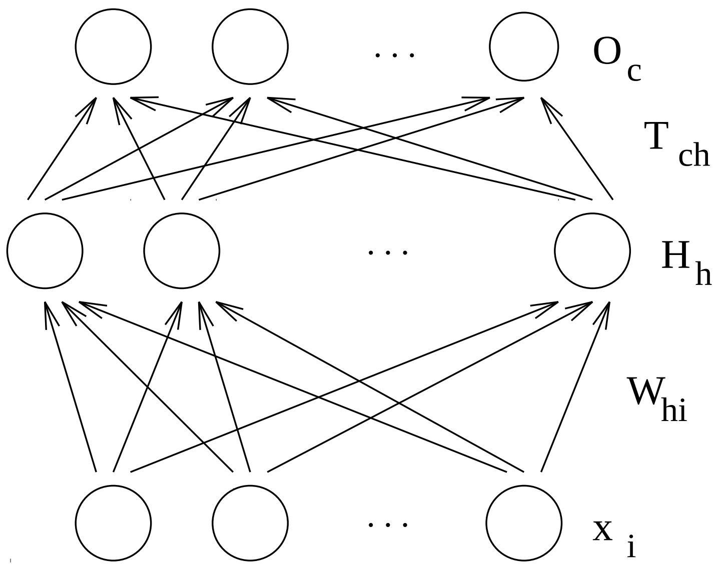

Figure 5.1: A multi-layer perceptron with one hidden layer.

$$\begin{aligned} O_c &= \sum_{h=1}^{H} T_{ch} H_h + T_{c0} \\ H_h &= \phi\left(\sum_{i=1}^{d} W_{hi} x_i + W_{h0}\right) \end{aligned} \tag{5.2}$$

where $\phi(\cdot)$ is for example the sigmoid:

$$\phi(a) = \frac{1}{1 + \exp(-a)} \tag{5.3}$$

We use the softmax function [20] to convert outputs to probabilities:

$$P_c = \frac{\exp O_c}{\sum_k \exp O_k} \tag{5.4}$$

thus guaranteeing that

$$P_c \geq 0 \text{ and } \sum_c P_c = 1$$

We find T and W values that minimize the cross entropy (directed Kullback-Leibler distance) using gradient-descent:

$$E(W) = -\sum_{c} R_c \log P_c \tag{5.5}$$

where R_i is the desired output that is 1 if $x \in C_i$ and 0 otherwise.

If we add a momentum factor α for speeding up convergence, our update formula for weights between output and hidden layers is:

$$\Delta W_{hc}\left(t\right) = \eta\left(R_c - P_c\right) H_h + \alpha \Delta W_{hc}\left(t-1\right) \tag{5.6}$$

After updating the weight vectors between the hidden and the output layers, the weight vectors between the hidden and input layers are calculated. If the activation function ϕ is a sigmoid, our weight update formula is:

$$\Delta W_{ih}(t) = \eta\left(\sum_{h}\left(R_c - P_c\right) T_{ch}\right) H_h(1 - H_h) x_i + \alpha \Delta W_{ih}(t-1) \tag{5.7}$$

Training by backpropagation requires many couples of forward-backward operations. We train the system by samples in the training set. A sample is randomly selected and forwarded to the system, then backward operation is applied to update the weight vectors. This is done for each sample until the training set is exhausted. This process is called an epoch. Generally many epochs are done until convergence. The W and T values that lead to highest success on the separate cross-validation set are chosen and kept as final.

5.2 Temporal Classification

Time-delay classifiers have the disadvantage that because the past inputs are delayed and then fed all together, the input dimensionality is larger. This causes also the classifier to be more complex and also may lead to the problem of curse of dimensionality. In the temporal approach, inputs are fed one after another as they arrive. Thus the classifier should now be able to detect temporal correlations and make decisions based on those.

For this case we used recurrent neural networks [21] [22]. We employed the backpropagation through time method and used a partially recurrent network where the hidden layer has fully recurrent connections (Fig. 5.2). This network differs from MLP proper in that Eq. (5.2) changes to

$$H_h(t) = \phi\left(\sum_{i=1}^{d} W_{hi}x_i + \sum_{k=1}^{H} Q_{hk}H_k(t-1) + W_{h0}\right) \tag{5.8}$$

where Q denote the weights of the recurrent connections between the hidden units and $H_h(0) = 0, \forall h$.

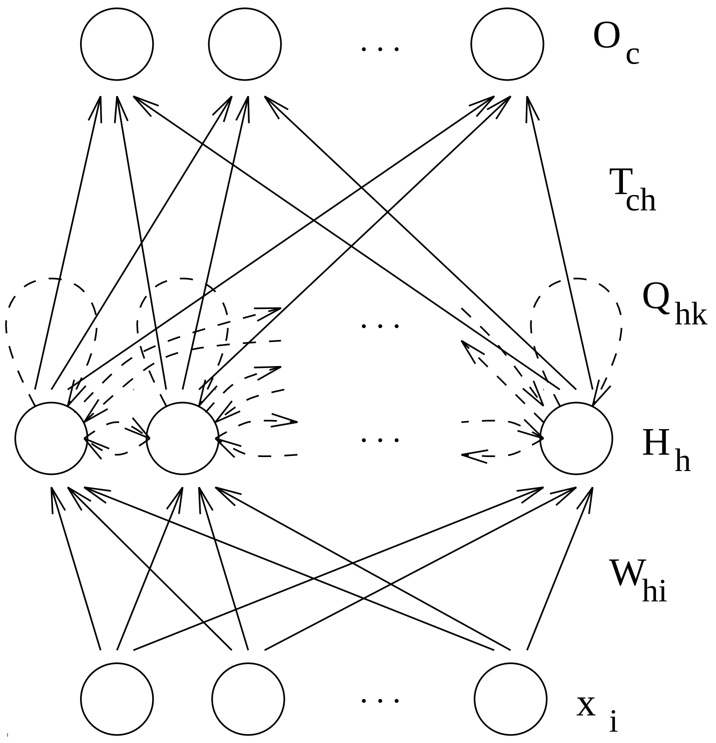

Figure 5.2: A recurrent neural network.

6. MULTIPLE CLASSIFIERS

The methods explained in the previous chapter are all single classifiers. Performance may be improved by tuning the preprocessing parameters and the classifier size. However, this tuning is a complex task and the classifier will still make errors for some regions of the problem space.

The idea of combining multiple classifiers is based on the idea that classifiers with different methodologies (structures) or different features can complement each other. Thus group decision may reduce the error and the system may give more accurate results.

Our goal is to combine different classifiers for which the error regions are as disjoint as possible. There are mainly two classifier combination schemes. In the first one, our classifiers use the same feature vectors. The difference between the classifiers is achived by using classifiers of different structures or of the same structure but trained with different learning parameters such as number of hidden layers and units for a MLP or number of nearest neighbor or distance metrics for k-NN.

In the second possibility, classifiers are trained with different feature vectors that can be multiple representations of a given input data [23] or data coming from different sources. It is possible to integrate physically different types of measurements and features by combining classifiers.

Some of the methods used for combining classifiers (experts) are:

1. *Voting* takes a linear, weighted sum of the output of the learners. Weights are nonnegative and sum up to one [24].
2. *Mixture of Experts* is similar to voting where weights are input dependent. There are a number of experts that partition the input space among themselves and a separate gating model is responsible from choosing the right expert for a given input [25] [26].

3. *Stacked Generalization* uses a separate learner that is also trained and is not restricted to be linear [27].

4. *Boosting* uses a serial approach where the next learner is trained to handle cases that cannot be learned by the previous learners [28].

5. *Cascading* uses a serial approach where the first learner is considered as a rule learner and the second an exception learner [29].

Let us say that we have m learners and n classes. We denote by d_{ji} the estimate of the learner j for class i. For now we ignore how they are calculated and instead we concentrate on the ways to combine them to find r_i the final estimate for class i, given input x.

$$r_i = f(d_{1i}, d_{2i}, \ldots, d_{mi} | \psi) \tag{6.1}$$

Assuming that r_i estimates the posterior probability of class i, and that the zero-one loss function is used [17], to minimize Bayesian risk, we choose the class with the highest probability:

$$c = \arg \max_i \, r_i \tag{6.2}$$

6.1 Similarity of Classifiers

Before we make decision on the classifiers to combine, we must define a measure that will help us to determine if a combination is useful or not, i.e., will improve accuracy.

We can view the class probabilities of classifiers as probability distributions.

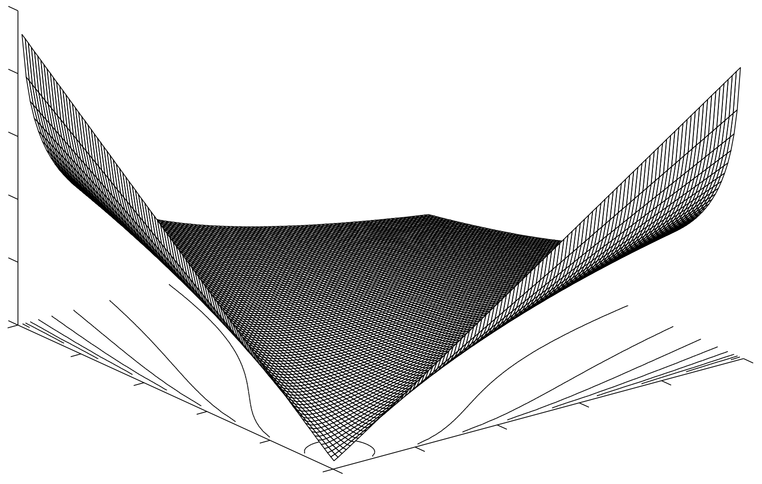

Figure 6.1: The Jeffreys-Kullback-Leibler distance (at minima when r_1 and r_2 are equal).

The distance between these distributions will reflect their similarity and utility of their combination. As a distance measure we can use the Kullback-Leibler distance, but since it is not symmetric it is not approriate for our case. Instead, we use the Jeffreys-Kullback-Leibler distance [30] which is a symmetric combination of the former (Fig. 6.1:

$$u(r_1, r_2) = -\frac{1}{Nm} \sum_{p=1}^{N} \sum_{i=1}^{m} r_{1i}^{p} \log\ r_{2i}^{p} + r_{2i}^{p} \log\ r_{1i}^{p} \tag{6.3}$$

where r_{1i}^{p} and r_{2i}^{p} are the outputs for probabilities for class i of classifiers r_1 and r_2 respectively for pattern p in the set of samples.

In case that we want to measure the similarity of more than two classifiers m, for each pattern, first we take the average of the class probabilities:

$$\overline{r}_i = \frac{1}{m} \sum_{j=1}^{m} r_{ji} \tag{6.4}$$

Then, the spread (or variance) of m classifiers is calculated as follows:

$$U(r_1, r_2, \ldots, r_m) = \frac{1}{m-1}\left(\sum_{k=1}^{m} u(\overline{r}_i, r_k)^2\right) \tag{6.5}$$

which if small shows that classifiers are similar.

Our goal is to find groups of classifiers which will have u and U values as close to zero as possible, in order to increase the overall success of the system.

6.2 Voting

Voting [24] (Fig. 6.2) is the simplest way to combine multiple classifiers. It corresponds to taking a linear combination of the learners. In Equation 6.1, if we denote by β_j the contribution of the jth classifier, $\psi = \bigcup_{j=1}^{m} \beta_j$ satisfying:

$$\forall j, \beta_j \geq 0 \; and \; \sum_{j=1}^{m} \beta_j = 1 \tag{6.6}$$

$$\begin{aligned} r_i &= f_V\left(d_{1i}(x), d_{2i}(x), \ldots, d_{mi}(x) | \beta_1, \beta_2, \ldots, \beta_m\right) && (6.7)\\ &= \sum_{j=1}^{m} \beta_j d_{ji}(x) && (6.8)\end{aligned}$$

If each voter (classifier) has equal weight $\beta_j = 1/m$ (i.e. we take the average), we call it *simple voting.*

$$r_i = \frac{1}{m}\sum_{j=1}^{m} d_{ji} \tag{6.9}$$

We sum the outputs of each classifier for each class and we select the class having the highest value as the final output of our system.

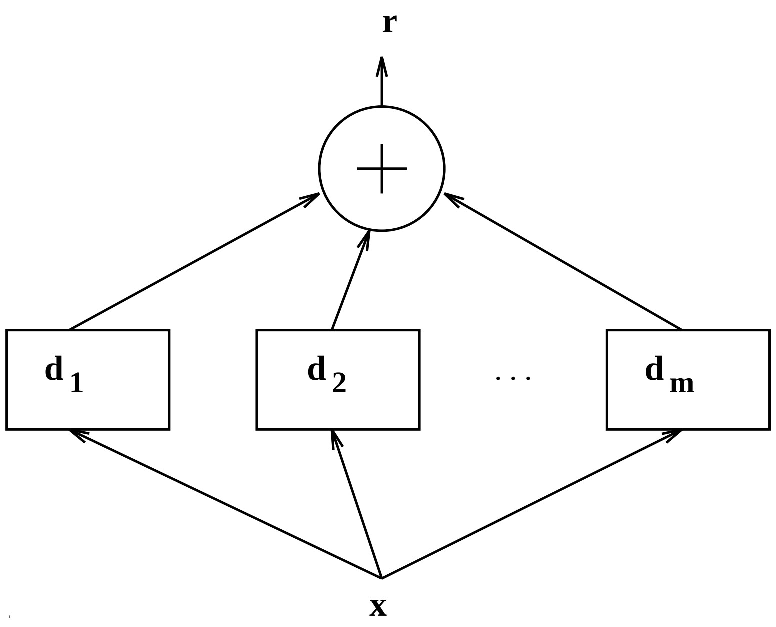

Figure 6.2: Voting.

Each classifier is separately trained to minimize:

$$E_j = -\sum_i y_i \log d_{ji}, \forall j. \tag{6.10}$$

where $y_i = 1$ if $x \in$ class i and 0 otherwise.

6.3 Mixture of Experts

Mixture of experts (ME) differs from the rest of the methods described in this chapter, in the sense that it is more rigid. All other methods are based on arbitrary elements and the learning rule for these elements is also arbitrary. Mixtures of experts [25] is composed of local experts which are linear perceptrons and there is a competitive gating mechanism that localizes the experts (Fig. 6.3). The gating network is also a linear perceptron whose output depends on the input. This approach has later been generalized to “hierarchies of experts” [31].

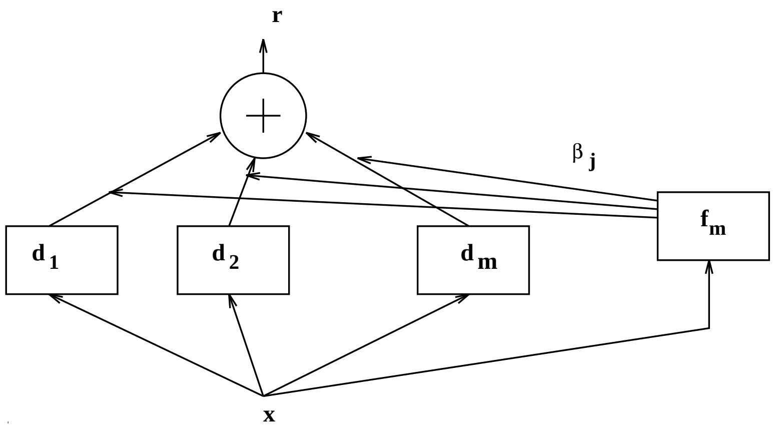

Figure 6.3: Mixture of experts.

$$\begin{aligned} r_i &= f_M\left(d_{1i}(x), d_{2i}(x), \ldots, d_{mi}(x)|\psi\right) && (6.11)\\ &= \sum_{j=1}^{m} \beta_j(x|\psi) d_{ji}(x) && (6.12) \end{aligned}$$

where $\beta_j(x|\psi)$ are the gating outputs.

Alpaydın and Jordan [26] describe and compare competitive and cooperative versions of ME for classification. In the competitive version, they use an error function that forces experts to compete. The gating values are the mixture proportions and the expert perceptron outputs are the means. They take Gaussian components with equal variances and the likelihood of the sample x is given as:

$$E = \log \sum_j g_j \exp\left[\sum_i y_i \log \ d_{ji}\right] \qquad (6.13)$$

This error equation forces experts to compete in order to increase the likelihood by preventing overlapping. For a given input, only one g_r is close to one and the rest are close to zero and its expert is responsible from the giving the system's output. We want to minimize the distance between the required output and the output probabilities of the expert r.

We can also use a cooperative scheme [26], in which the error function is the Kullback-Leibler distance between system's output r_i and the desired output y_i for all classes.

$$E = -\sum_i y_i \log\ r_i \tag{6.14}$$

6.4 Stacked Generalization

Stacked generalization (Fig. 6.4) is a technique proposed by Wolpert [27] that extends voting in that learners (called level 0 generalizers) are not necessarily combined linearly. The combination is made by a combiner system (called level 1 generalizer) that is also trained.

The classification of a pattern using stacked generalization is done as follows: The pattern is given to level 0 generalizers and propagated. Then, their output vectors are combined to construct the input vector for the level 1 generalizer. The output of the level 1 generalizer is the final output of the system.

$$r_i = f_S\left(d_{1i}(x), d_{2i}(x), \ldots, d_{mi}(x)|\psi\right) \tag{6.15}$$

The level 1 generalizer learns what the correct output is when level 0 generalizers give a certain output combination. Thus, level 1 generalizer needs to be trained on data unused in training the level 0 generalizers. The combiner f_S is trained to minimize:

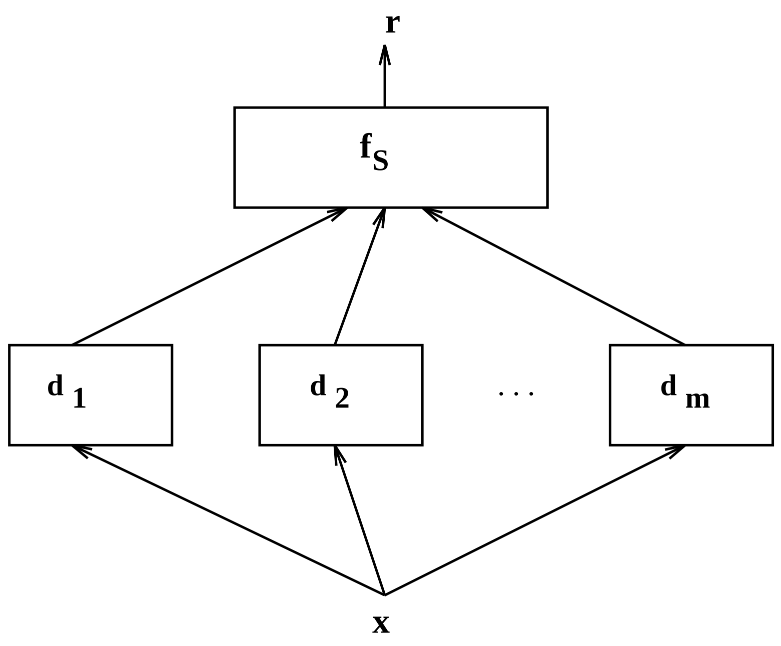

Figure 6.4: Stacked generalization.

$$E = -\sum_i y_i \log r_i \tag{6.16}$$

The training of the original stacked generalization algorithm, proposed by Wolpert uses leave-one-out method which is an expensive method.

6.5 Boosting

Boosting [28] (Fig. 6.5) is a technique for constructing a classifier which makes small error rates from classifiers which are doing just slightly better than 50 percent. Unlike previously described methods, learners are trained serially. After training the first learner, we train the second learner with the data on which the first fail making sure that the two learners complement each other. A third learner is trained with

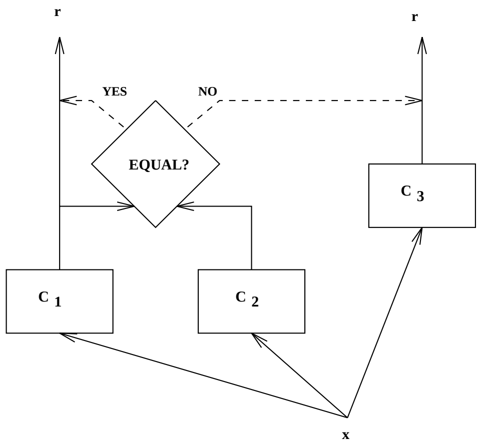

Figure 6.5: Boosting.

the data on which the first two learners disagree. In the recognition phase, the third learner is consulted only when the first two learners disagree. Boosting requires large training sets and is an expensive method.

$$r_i = f_B(c_1, c_2, c_3) \ \textit{where} \ c_j = \arg\max_i d_{ji}, j = 1, 2, 3 \tag{6.17}$$

$$f_B(c_1, c_2, c_3) = \begin{cases} c_1 & \text{if } c_1 = c_2 \\ c_3 & \text{otherwise} \end{cases} \tag{6.18}$$

6.6 Cascading

The main idea in cascading [29] is that a great percentage of the training samples can be explained by simple rule and there are a small number of exceptions. A *rule* is is supposed be valid for everywhere, but in practice *exceptions* exist, and we want to be able to succeed on them. Using two classifiers, one to learn the rule, C_r, and the other to learn the exceptions, C_x, seems to be good idea.

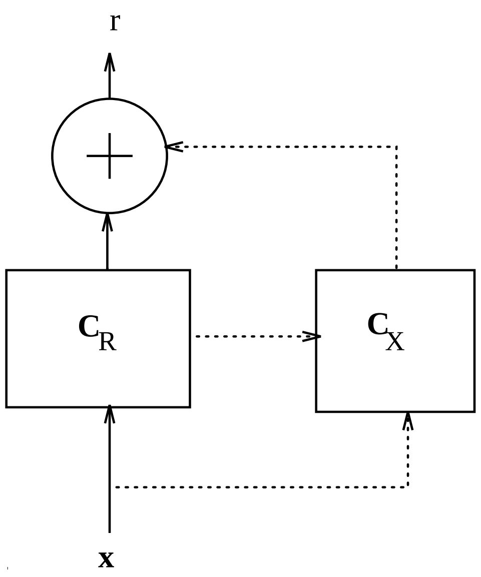

Figure 6.6: Cascading.

The major problem is to decide when a pattern is covered by the rule and thus should be learned by C_r, or is an exception and should be learned by C_x. We decide on this by looking at the highest class posterior and comparing it with a threshold θ. If it is greater than θ, we assume that it is covered by the rule, otherwise it is an exception. During recognition, we first check C_r. If the highest posterior exceeds θ, x is covered by the rule C_r, otherwise it is covered by the exception learner C_x (Fig. 6.6).

$$f_C(\{d_{Ri}\}_i, \{d_{Xi}\}_i = \begin{cases} c_R = \text{arg max}_i\ d_{Ri} & if d_{RC_R} > \theta \\ c_X = \text{arg max}_i\ d_{Xi} & \text{otherwise} \end{cases} \tag{6.19}$$

As a variant, instead of setting the system output as the output of the exception learner for an exception, we may perform voting over the two classifiers.

In order to train the system of cascaded classifiers, we should find the exceptional patterns that do not obey the rule. The rule learner is expected to cover a significant amount of the patterns. Here we can use leave-one-out if we have less samples, and k-fold cross-validations if we have a lot of them.

6.7 A General Formula for Networks with Gating

In addition the the model defined in Equation 6.1, we define a less general model for classifiers combined using an external combiner which determines the weight of each expert using a gating-like structure. All the classifier types previously described in this chapter (except stacked generalization and mixture of experts with non-linear perceptron combiner) are special cases of this model.

$$r_i = \sum_{j=1}^{m} \beta_j(x|\psi) d_{ji}(x) \tag{6.20}$$

In the case of simple voting, β_j are constant and equal to $1/m$. All experts have equal effect on the final output.

In mixture of experts, gating depends on the input and the weights ψ are also learned.

In boosting, we have only three experts (m=3). The gating values for each one of them depends on the outputs of the first two d_1 and d_2. If they agree on their decision, we can say that $\beta_1 = \beta_2 = 1/2$ and $\beta_3 = 0$. Otherwise, $\beta_1 = \beta_2 = 0$ and $\beta_3 = 1$.

Cascading is similar to boosting in that the gating decision are made using some sort of ifthen rules. We have two classifiers (m=2). If the first classifier is reliable, $\beta_1 = 1$ and $\beta_2 = 0$. Otherwise $\beta_1 = 0$ and $\beta_2 = 1$, or if voting is done, $\beta_1 = \beta_2 = 1/2$.

7. EXPERIMENTS

After preprocessing and elimination of nonsense input, we divided our database into two parts, one containing 30 writers and the other containing 14 writers. The first part is divided into three sets with 3,748 digits for training, 1,873 for cross-validation and 1,873 for writer-dependent testing. 3,498 examples from 14 writers are used to test performance on writers unseen during training.

All of our learning algorithms were implemented in C, compiled using the GNU C compiler and run on a Sun SPARC 20 workstation with Solaris 2.4 operating system.

In the tables and figures, we show the performance of the classifiers on the training set (TR), cross-validation set (CV), writer dependent test set (WD) and writer independent test set (WI). We also show the number of learning epochs (EP) and number of parameters (PS).

7.1 Single Classifiers

First we investigate the generalization ability and the cost of k-nearest neighbor (k-NN) and multi-layer perceptron (MLP) trained with dynamic and static data and recurrent networks trained with dynamic data. Our goal is to decide which type of classifiers is suitable to be combined. The factors that we consider are classification accuracy, number of parameters, learning and recognition time.

7.1.1 k-Nearest Neighbor (k-NN)

We examined the success rates using simple k-NN, Gaussian k-NN and Fuzzy k-NN versions for k values equal to 1, 3, 5, 7 and 9. The highest success rate for both representations was obtained using Gaussian k-NN. Here we present only the results for Gaussian k-NN which gives the best results in terms of accuracy. The rest of the results are in Appendix.

Learning takes one epoch. The number of parameters is Nd where N is the number of patterns in the training set and d is the dimensionality of input. For all k-NN variants the recognition phase is time consuming, because we have to calculate all distances between a sample to be classified and the training set samples. However, the learning phase virtually does not exist. In all k-NN experiments we use the Euclidean distance.

7.1.1.1 k-NN Results for Dynamic Representation

For a given representation, the success rate for both writer dependent and writer independent test sets changes very little for different k values. Results are in Tables 7.1, 7.2, 7.3 and Figure 7.1.

As the number of resampling points increase, the success rate for writer dependent test set tends to increase slowly and for writer independent test set it is almost the same.

Table 7.1: Results for Gaussian k-NN (dynamic, 8 points)

k	1	3	5	7	9
TR	100.00	100.00	100.00	100.00	100.00
CV	99.47	99.41	99.41	99.36	99.36
WD	99.84	99.79	99.73	99.84	99.79
WI	97.57	97.83	97.91	97.86	97.68
PS	59968	59968	59968	59968	59968

Table 7.2: Results for Gaussian k-NN (dynamic, 12 points)

k	1	3	5	7	9
TR	100.00	100.00	100.00	100.00	100.00
CV	99.41	99.41	99.36	99.36	99.41
WD	99.68	99.79	99.73	99.79	99.79
WI	97.94	97.97	98.06	98.00	97.94
PS	89952	89952	89952	89952	89952

Table 7.3: Results for Gaussian k-NN (dynamic, 16 points)

k	1	3	5	7	9
TR	100.00	100.00	100.00	100.00	100.00
CV	99.41	99.36	99.36	99.36	99.41
WD	99.79	99.73	99.73	99.79	99.79
WI	97.83	97.94	98.06	98.03	97.94
PS	119936	119936	119936	119936	119936

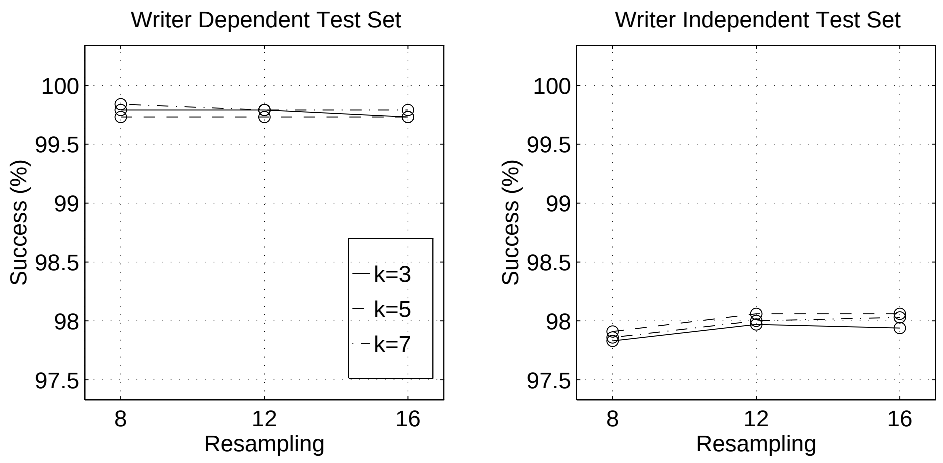

Figure 7.1: Gaussian k-NN performance versus resampling for different k values and dynamic representation.

7.1.1.2 k-NN Results for Static Representation

For 8 by 8 resolution, the success rate for writer independent test set slowly increases as k increases. For 12 by 12 and 16 by 16 resolutions, the success rate for writer independent test set slowly decreases as k increases. Results are in Tables 7.4, 7.5, 7.6 and Figure 7.2.

The best results are obtained for 8 by 8 resolution. As image resolution increases, the accuracy of k-NN decreases. This is because we use the Euclidean distance in which are input dimensions are assumed to be independent. We may obtain higher accuracy by using a distance metric which considers the dependency among input dimensions or we may use a larger blurring kernel in preprocessing to get thicker digits. Since the cost of k-NN is very high in terms of memory requirements and recognition time we do not do this.

Table 7.4: Results for Gaussian k-NN (static, 8 by 8 resolution)

k	1	3	5	7	9
TR	100.00	100.00	100.00	100.00	100.00
CV	98.24	98.51	98.40	98.24	98.13
WD	97.92	98.29	98.18	98.40	98.18
WI	95.65	96.17	96.28	96.46	96.48
PS	239872	239872	239872	239872	239872

Table 7.5: Results for Gaussian k-NN (static, 12 by 12 resolution)

k	1	3	5	7	9
TR	100.00	100.00	100.00	100.00	100.00
CV	97.60	97.65	97.54	97.28	97.22
WD	97.44	97.49	97.49	97.17	96.85
WI	94.48	94.48	94.43	94.31	94.11
PS	539712	539712	539712	539712	539712

Table 7.6: Results for Gaussian k-NN (static, 16 by 16 resolution)

k	1	3	5	7	9
TR	100.00	100.00	100.00	100.00	100.00
CV	96.42	96.42	96.16	95.78	95.41
WD	95.46	95.73	96.00	95.84	95.68
WI	91.34	91.82	91.45	91.45	91.08
PS	959488	959488	959488	959488	959488

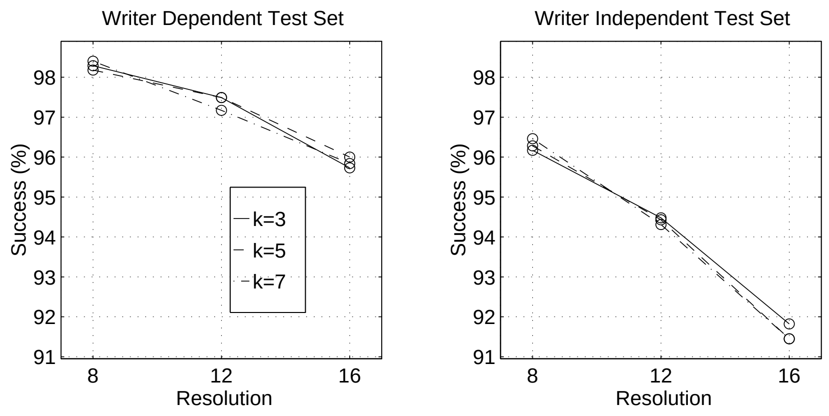

Figure 7.2: Gaussian k-NN performance versus resolution for different k values and static representation.

7.1.2 Multi-layer Perceptrons (MLP)

Each MLP structure is trained 10 times independently and we report averages and standard deviations of the learning epochs and success on training, cross-validation, writer dependent and independent test sets. The learning factor η is adaptive for improved convergence. It is decreased by multiplying with a factor less than one when cross-validation error does not decrease. Training stops when the learning factor becomes less than 0.001. A momentum factor α equal to 0.7 is used. We test the network after each epoch until the end of training, and report its performance on the cross-validation and test sets. The number of the hidden units are 10, 20, 30 and 40. Results are in Tables 7.7, 7.8, 7.9 and Figure 7.3.

We used multi-layer perceptrons with a single hidden layer. Such MLPs have $(d+1)h+(h+1)c$ parameters. Where d, is the dimension of the input, h, the number of hidden units, and c, the number of classes.

Table 7.7: Results for MLP (dynamic, 8 points)

HU	10	20	30	40
EP	15.20, 3.29	17.20, 3.55	16.60, 4.06	15.60, 4.35
TR	98.98, 0.16	99.21, 0.28	99.10, 0.13	99.33, 0.11
CV	98.33, 0.34	98.55, 0.18	98.60, 0.18	98.71, 0.14
WD	98.26, 0.31	98.72, 0.11	98.73, 0.18	98.75, 0.18
WI	95.26, 0.37	95.71, 0.18	95.79, 0.26	95.83, 0.29
PS	280	550	820	1090

Table 7.8: Results for MLP (dynamic, 12 points)

HU	10	20	30	40
EP	15.20, 3.05	15.90, 3.96	17.10, 4.31	14.90, 3.67
TR	99.11, 0.22	99.41, 0.10	99.51, 0.09	99.47, 0.11
CV	98.51, 0.16	98.81, 0.18	98.81, 0.19	98.82, 0.07
WD	98.47, 0.27	98.89, 0.18	98.94, 0.13	98.87, 0.13
WI	95.54, 0.30	95.99, 0.21	95.98, 0.24	95.93, 0.26
PS	360	710	1060	1410

7.1.2.1 MLP Results for Dynamic Representation

For writer dependent test set, accuracy tends to increase from 8 points to 12 points and decrease from 12 points to 16 points. This is caused by overfitting the training data. As expected, the number of hidden units used improves accuracy, but after 20 units this gain is not significant and it is achieved using a larger number of free parameters. Results are in Tables 7.7, 7.8, 7.9 and Figure 7.3.

Table 7.9: Results for MLP (dynamic, 16 points)

HU	10	20	30	40
EP	15.50, 3.21	12.90, 3.67	12.30, 3.16	14.70, 3.62
TR	98.96, 0.13	99.30, 0.17	99.33, 0.22	99.49, 0.11
CV	98.30, 0.29	98.83, 0.12	98.81, 0.11	98.92, 0.10
WD	98.33, 0.20	98.76, 0.14	98.79, 0.17	98.97, 0.15
WI	95.23, 0.39	96.00, 0.24	95.93, 0.24	96.17, 0.33
PS	440	870	1300	1730

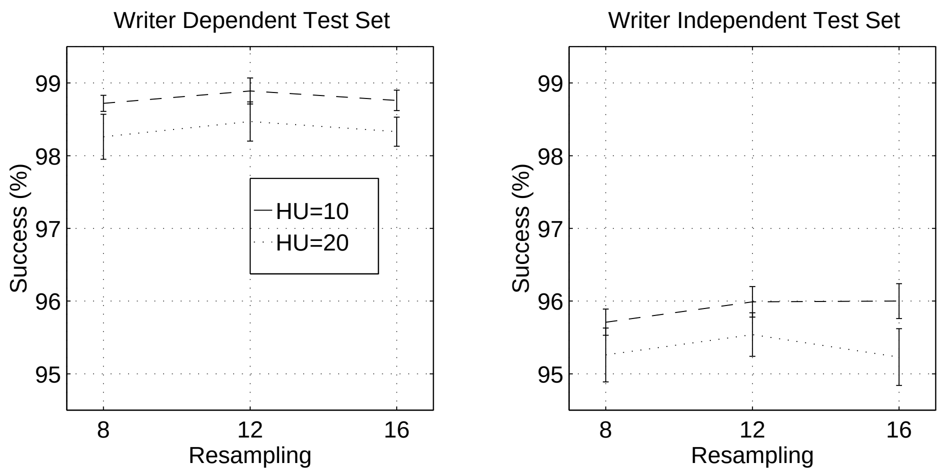

Figure 7.3: One hidden layer MLP performance versus resampling for different number of hidden units and dynamic representation.

Table 7.10: Results for MLP (static, 8 by 8 resolution)

HU	10	20	30	40
EP	12.10, 3.90	10.40, 2.63	12.60, 4.22	11.80, 2.97
TR	98.76, 0.36	99.02, 0.31	99.10, 0.32	99.14, 0.21
CV	96.24, 0.26	96.57, 0.21	96.68, 0.17	96.75, 0.28
WD	95.73, 0.37	96.16, 0.26	96.16, 0.31	96.29, 0.31
WI	94.25, 0.25	94.55, 0.22	94.53, 0.50	94.58, 0.34
PS	760	1510	2260	3010

Table 7.11: Results for MLP (static, 12 by 12 resolution)

HU	10	20	30	40
EP	8.60, 1.65	11.20, 3.01	8.80 , 1.81	10.00, 2.21
TR	99.61, 0.16	99.76, 0.15	99.74, 0.07	99.77, 0.07
CV	96.78, 0.20	97.12, 0.16	97.05, 0.12	97.03, 0.20
WD	96.34, 0.16	96.73, 0.20	96.64, 0.22	96.73, 0.23
WI	94.44, 0.31	94.81, 0.21	94.95, 0.25	95.03, 0.23
PS	1560	3110	4660	6210

7.1.2.2 MLP Results for Static Representation

For writer dependent test set, the best accuracy is obtained for 12 by 12 resolution. For 16 by 16 the success drops down to values below those given 8 by 8. This is caused by the 3 by 3 kernel which should be optimized, probably using a larger blurring kernel would increase the preformance. As expected, increasing the number of hidden units used improves accuracy, but after 20 units this gain is not significant and it is achieved using a larger number of free parameters. Results are in Tables 7.10, 7.11, 7.12 and Figure 7.4.

Table 7.12: Results for MLP (static, 16 by 16 resolution)

HU	10	20	30	40
EP	12.60, 4.72	41.40 , 11.36	9.20, 3.77	8.60, 4.20
TR	99.88, 0.08	95.22 , 1.67	99.88, 0.15	99.90, 0.09
CV	96.37, 0.19	96.90, 0.22	96.79, 0.16	96.92, 0.20
WD	96.18, 0.33	96.58, 0.36	96.71, 0.21	96.69, 0.14
WI	94.36, 0.39	94.75, 0.36	95.03, 0.27	94.93, 0.21
PS	2670	5330	7990	10650

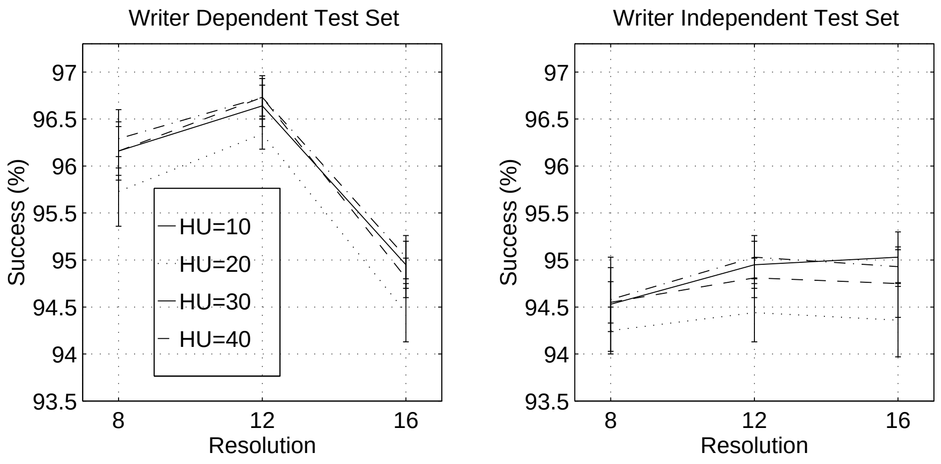

Figure 7.4: One hidden layer MLP performance versus resolution for different number of hidden units and static representation.

Table 7.13: Results for recurrent MLP with frame length equal to 2.

HU	4	8	12	16	20
EP	45.70, 12.11	41.40, 11.36	40.80, 11.78	35.30, 4.79	30.70, 4.37
TR	83.09, 3.22	95.22, 1.67	98.32, 0.32	99.90, 0.09	99.40, 0.23
CV	81.53, 3.21	93.47, 1.59	96.86, 0.57	96.92, 0.20	98.23, 0.34
WD	82.87, 3.31	94.22, 1.51	97.12, 0.58	96.69, 0.14	98.12, 0.46
WI	77.31, 3.72	90.03, 1.94	93.86, 0.79	94.93, 0.21	94.87, 0.89
PS	78	178	310	474	670

7.1.3 Recurrent Neural Network Results

Our recurrent networks with dynamic data of 8 points for different numbers of hidden units and a frame length of 2 (one resampled point per frame). We tested for 4, 8, 12, 16 and 20 hidden units. The accuracy on writer dependent and independent increases and the training time decreases as the number of hidden units increases.

Compared with MLPs recurrent networks have a longer training phase and classification time because of their recursive structure. Their accuracy is inferior. They are more suitable for cases where the temporal signal has a greater dimensionality and using MLPs is infeasible for example in speech processing.

7.2 Multiple Representations

We know that the error regions of the classifiers trained using dynamic and static representations are not the same. Some samples are correctly classified by both, some only by one of them and some by none of them. Table 7.14 shows the misclassi-

Table 7.14: Classifiers trained with different representations fail for different samples.

Set	Dynamic	Static	Both
Training	1.02	1.24	0.25
Cross-validation	1.67	3.76	0.46
Writer Dep. Test	1.74	4.27	0.70
Writer Indep. Test	4.74	5.75	1.70

Table 7.15: Jeffreys-Kullback-Leibler distances (average and standard deviations)

Set	Dyn. MLP, Dyn. Rec.	Dyn. MLP, Sta. MLP	Dyn. Rec., Sta. MLP
TR	0.040324, 0.012369	0.059724, 0.003548	0.068907, 0.008772
CV	0.056917, 0.016853	0.105380, 0.004592	0.121809, 0.011352
WD	0.050803, 0.012643	0.105533, 0.004386	0.119176, 0.008738
WI	0.076679, 0.017269	0.166287, 0.003946	0.173203, 0.015705

fication percentages for MLPs with ten hidden units trained with dynamic (8 points) and static (8 by 8) representation together with the percentage of samples recognized by none of them. If we find an efficient way of combining these classifiers we get much better results than both of them alone. We choose to use two classifiers: one for dynamic and the other for static data.

7.2.1 Multiple Classifiers

In order to choose the which types of classifiers should be used and combined, we measured the Jeffreys-Kullback-Leibler (Eq. 6.3) to determine the most appropriate type of representations and classifiers to combine (Table 6.3).

We select the two classifiers that will be combined using methods described in the previous chapter according to the results found in the previous section. k-NN methods give good results but they are very expensive in terms of memory requirements and recognition time. MLPs are good structures that are trained and give good results, worse than k-NN but use much less parameters and have shorter recognition time. Recurrent MLPs are also good classifiers but their generalization ability is lower than MLPs and they are applicable only to temporal dynamic represenatation.

After our decision on the type of classifiers, we decide on the size of the MLPs. Experiments done show us that the number of hidden units has not significant effect on the accuracy. So, we decide to use 10 hidden units which gives lower success than others but has much less parameters.

Finally, we select the representation parameters. Dynamic representation feature vector size increases linearly as the number of resampling points increase. However, for static representation this relation is quadratic. We choose 8 point resampled dynamic data and 8 by 8 resolution static images in our classifiers.

The MLPs trained with dynamic representation, have higher generalization ability than those trained with static. In the next sections we will call a MLP trained with dynamic data DMLP and a MLP trained with static images SMLP.

7.2.2 MLP Results for Concatenated Dynamic and Static Representations

As a first step toward combining the two representations, we chose the simplest option, concatenating the two features vectors to obtain a combined feature vector to train a MLP (DSMLP).

Thus, the input vector size is equal to 80 (2×8 = 16 for dynamic vectors and $8\times$

Table 7.16: Results for concatenated feature vectors (DSMLP)

	DMLP	SMLP	DSMLP 10 HU	DSMLP 20 HU
EP	15.20, 3.29	12.10, 3.90	13.50, 4.09	13.70, 2.67
TR	98.98, 0.16	98.76, 0.36	99.72, 0.12	99.86, 0.06
CV	98.33, 0.34	96.24, 0.26	98.75, 0.21	99.14, 0.10
WD	98.26, 0.31	95.73, 0.37	98.52, 0.17	98.67, 0.27
WI	95.26, 0.37	94.25, 0.25	96.40, 0.53	97.03, 0.41
PS	280	760	920	1810

8 = 64 for static image). We try for 10 and 20 hidden units. As it can be seen in Table 7.16, joint feature vector introduces more information and increases generalization power. Note that although the writer dependent test set difference between DMLP and DSMLP is very small, the writer independent test set is significantly better and increases as the number of hidden units is increased.

7.2.3 Voting

We trained DMLP and SMLP separately and we combined them by taking the arithmetic average of their outputs (simple voting).

The results obtained are better than DMLP, SMLP and DSMLP-10 (Table 7.17). Compared with DSMLP, voting is better and uses much less parameters. Note as in the concatenated case, that although the writer dependent test set difference between DMLP and DSMLP is very small, the writer independent test set is significantly better.

Table 7.17: Results for Voting

	DMLP 10 HU	SMLP 10 HU	Voting
EP	15.20, 3.29	12.10, 3.90	15.20+12.10
TR	98.98, 0.16	98.76, 0.36	99.55, 0.04
CV	98.33, 0.34	96.24, 0.26	98.94, 0.12
WD	98.26, 0.31	95.73, 0.37	98.41, 0.20
WI	95.26, 0.37	94.25, 0.25	97.09, 0.33
PS	280	760	280+760

Table 7.18: Results for Mixture of Experts with experts MLP and linear gating

	DMLP 10 HU	SMLP 10 HU	ME Coop	ME Comp
EP	15.20, 3.29	12.10, 3.90	17.30, 4.99	17.70, 6.02
TR	98.98, 0.16	98.76, 0.36	99.48, 0.28	99.43, 0.16
CV	98.33, 0.34	96.24, 0.26	97.64, 0.88	97.89, 0.75
WD	98.26, 0.31	95.73, 0.37	97.36, 0.97	97.74, 0.63
WI	95.26, 0.37	94.25, 0.25	95.29, 0.75	95.42, 0.74
PS	280	760	280+760+810	280+760+810

7.2.4 Mixture of Experts

We can view mixture of experts (ME) as another method of combining DMLP and SMLP. We consider them as experts and we modify weight update formulas accordingly. The gating network is also a MLP trained with a concatenated feature vector similar to that used to train DSMLP. Here the experts are MLPs. We tested cooperative and competitive versions of ME with gating network as linear or MLP.

In all cases the performance on the writer dependent test set of ME is lower than on DMLP but higher than on SMLP (Table 7.18). The writer independent test set success for ME is slightly higher than DMLP but with much more free parameters

Table 7.19: Results for Mixture of Experts with experts and gating MLP

	DMLP 10 HU	SMLP 10 HU	ME Coop	ME Comp
EP	15.20, 3.29	12.10, 3.90	15.20, 4.59	19.60, 7.63
TR	98.98, 0.16	98.76, 0.36	99.66, 0.18	99.65, 0.15
CV	98.33, 0.34	96.24, 0.26	97.93, 1.13	98.12, 0.58
WD	98.26, 0.31	95.73, 0.37	97.77, 1.11	97.94, 0.83
WI	95.26, 0.37	94.25, 0.25	95.75, 1.14	95.73, 0.54
PS	280	760	280+760+920	280+760+920

which means that we do not gain much.

The complexity of the gating network slightly increases the performance (Table 7.19). Additionally we may say that there is no significant difference between the results for cooperative and competitive schemes.

7.2.5 Stacked Generalization

It has previously been mentioned that in stacked generalization the combiner should be trained on a set different from the one on which the classifiers are trained. Our training algorithm uses 2-fold cross-validation as follows: We divide the training set into two parts: T_1 and T_2. First we train DMLP and SMLP (level 0 generalizers) on T_1. Then we see what output they produce on T_2. Probably, our classifiers will not be accurate on these patterns unseen during training. We save these responses in a separate set U. Then we train the combiner (level 1 generalizer) with U to learn the correct output when the level 0 generalizers respond in a certain way. The last step in out algorithm is training all level 0 generalizers (DMLP and SMLP) with the complete T.

As level 1 generalizers we use a simple linear perceptron and a MLP with 10

Table 7.20: Results for Stacked Generalization

	DMLP 10 HU	SMLP 10 HU	SG MLP 10HU	SG Linear
EP	15.20, 3.29	12.10, 3.90	129.80, 15.02	151.70, 17.30
TR	98.98, 0.16	98.76, 0.36	99.04, 0.13	98.82, 0.17
CV	98.33, 0.34	96.24, 0.26	98.98, 0.14	98.83, 0.21
WD	98.26, 0.31	95.73, 0.37	98.65, 0.25	98.62, 0.23
WI	95.26, 0.37	94.25, 0.25	96.73, 0.14	96.66, 0.36
PS	280	760	280+760+320	280+760+210

hidden units. Both give better results than individual classifers (Table 7.20). Combiner complexity has little influence on the system's performance.

7.2.6 Cascading

As decribed before, cascading is based on a rule-exception model and we have to decide which classifiers we should use. In our problem, the best recognizer is the dynamic one. However the static recognizer, although less accurate, is able to classify some samples that are misclassified by the dynamic classifier. So, we can set the rule learner as dynamic DMLP d_R and the exceptions learner as SMLP d_X.

Our learning algorithm uses 2-fold cross-validation. We divide the training set into two parts: T_1 and T_2. First, d_R is trained on T_1. All the patterns for which d_R is not certain, (i.e. the posterior of the right class is less than θ) are saved in a separate set U. Then we train d_R on T_2 and save the patterns from T_1 for which d_R is not certain in U. Then the exception learner d_X is trained with U and the rule s learner d_R is trained with the complete training set T.

The performance of cascading is similar to that of DMLP (Table 7.21). The rule-exception learner scheme does not increase the system's performance because we

Table 7.21: Results for Cascading

	DMLP 10 HU	SMLP 10 HU	Cas D-S
EP	15.20, 3.29	12.10, 3.90	115.00, 15.95
TR	98.98, 0.16	98.76, 0.36	99.16, 0.14
CV	98.33, 0.34	96.24, 0.26	98.12, 0.20
WD	98.26, 0.31	95.73, 0.37	98.00, 0.32
WI	95.26, 0.37	94.25, 0.25	95.37, 0.67
PS	280	760	280+760

need a large number of samples to train the exception learner which is a MLP. Using an exception classifier of a different structure may give better results.

7.2.7 Comparison

For the writer dependent case we see that stacked generalization and MLP trained using a concatenation of dynamic and static representations give the best results (Fig. 7.5). For stacked generalization, the complexity of the combiner has small effect on the performance. These methods capture the dependency between the two representations since both are used for training at the same time.

Voting and cascading, in which modules are trained separately, there is no way of capturing the relation between dynamic and static representations to yield higher accuracy. DMLP alone even is better than cascading. The performance of mixture of experts is the lowest.

For the writer independent case, the situation is different (Fig. 7.6). Here the best results are obtained using voting and MLP trained using a concatenation of dynamic and static representation. The two stacked generalization and mixture of experts variants give worse results. The success rate of cascading is higher than DMLP

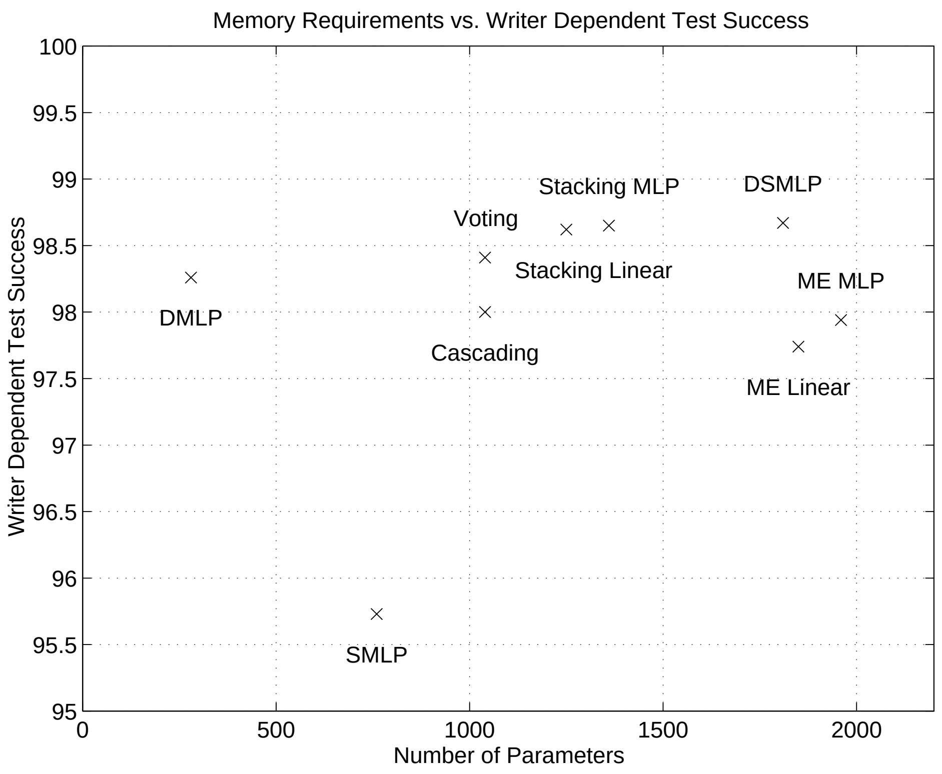

Figure 7.5: Writer dependent test set success versus number of parameters.

alone. Here, mixture of experts outperforms cascading.

The difference between the writer dependent and writer independent test cases can be explained using the way of combining the classifiers. In stacked generalization and DSMLP, our systems are trained using both representations. This way, the dependency between them is learned. However, DSMLP has greater generalization power than stacked generalization. In the training phase, stacked generalization memorizes the relation beteen the two represenatations for our first 30 writers, but for the unseen writers this is not always valid.

In the voting case, since the classifiers are trained independenly, no relation between the two classifiers in learned. For writer independent samples, the optimal combining is close to linear. This explains the difference between the success in the two testing cases. The situation for cascading is similar.

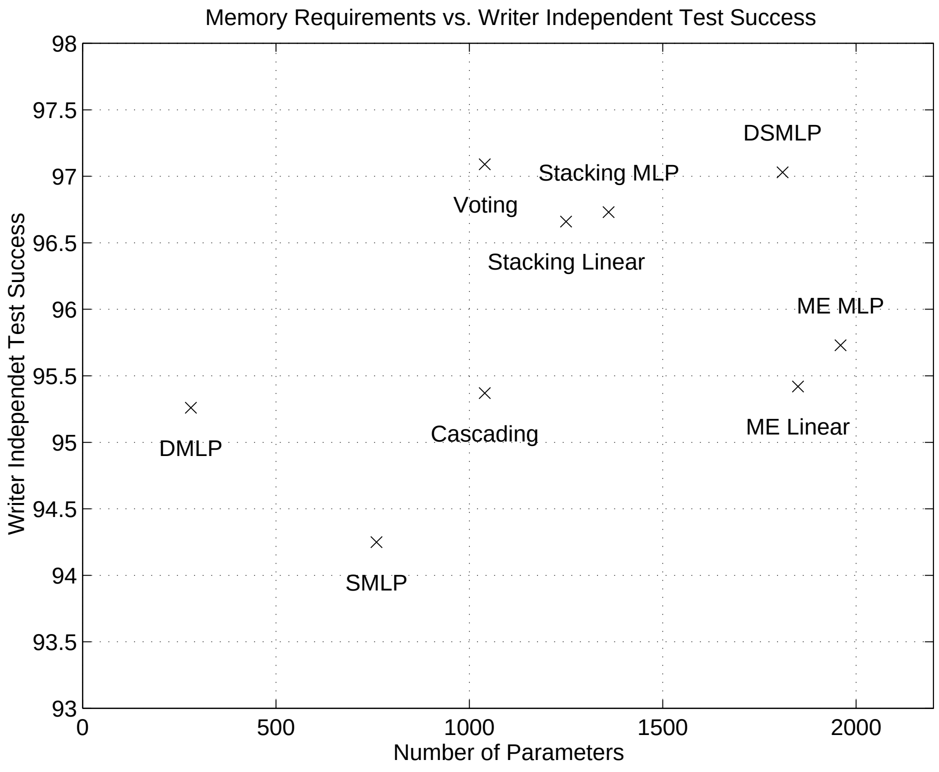

Figure 7.6: Writer independent test set success versus number of parameters.

8. CONCLUSION

In this thesis we have designed an on-line handwritten digit recognition system after the examination of different approaches. This system includes digitizing, preprocessing and recognition parts.

Digitizing is performed while the user writes on a pressure sensitive tablet using a cordless stylus. Digits are written into special boxes. The tablet samples and sends x, y values of the stylus using its serial interface to the computer that will process them. We developed a software that reads data and stores in a meaningful format.

In order to use in learning we collected 250 digit samples from each of 44 people. Raw data coming from tablet is preprocessed in various ways. First it is normalized. In the dynamic preprocessing, the sampling points are converted to a constant length feature vector of equally distanced points on the pen trajectory. The static preprocessing converts the dynamic information to an image similar to images used in off-line recognition tasks.

After preprocessing, we test the well known classification methods k-NN, MLP and recurrent networks for our representations with different parameters (resampling point number and image resolution). The classifiers selection criteria is based on a distance measure between the class probability outputs of the individual learners. Here we choose the most suitable size of input vectors and classifiers to combine: MLP with 10 hidden units trained with 8 resampled points and MLP with 10 hidden units but trained with 8 by 8 images. Their average writer independent success is 95.26 and 94.25 percent respectively. Classifiers trained with dynamic and static representations make misclassifications for different samples. Their error regions are different and combining them will increase the accuracy of the recognizer.

Training classfiers such as MLPs with a new feature vector made up by concatenating the dynamic and static feature vector increased the performance of our system. Using this method we obtained average writer independent success of 97.03 percent for 20 hidden units which is higher than both.

In order to achieve better accuracy, we combine different classifiers using voting, mixture of experts, stacked generalization and cascading.

We tried simple voting using the single classifiers determined above and we obtained 97.09 success for writer independent test set. This outperforms the previous method, especially in terms of the number of parameters used.

Stacked generalization gave slightly worse results (96.73 percent) than voting, probably because the optimal combination rule is very close to simple voting where the weights of the classifiers are equal. The complexity of the combiner also memorized the input output pairs, instead of generalizing.

Mixture of experts, in our new form of MLP experts and MLP or linear gating network gave the worst results for writer dependent test set (97.94 percent). In the writer independent case, its ranking is better and the accuracy is 95.75 percent. The cooperative and competitive schemes are similar, both on accuracy and memory requirements.

In cascading, we tried two possibilities: a multi-layer perceptron trained with dynamic data as rule learner and a multi-layer perceptron trained with static data as exception learner and vice versa. The first option outperformed the second since DMLP is a better generalizer.

After testing the above methods we conclude that using multiple classifiers and multiple represenatations (especially in our problem) improves performance significantly. For our database voting seems to be the best choice.

This work can be extended to learning not only digits but also letters and punctuation marks. Exploring the alternatives of using the two representations with different type of classifiers (k-NN, LVQ, etc.) and combining them in a hierarchical manner may increase performance considerably.

APPENDIX A

Here we explain Bresenham's scan converting Algorithm. Given two endpoints, we try to find the "in-between" points on a pixel grid. Bresenham's line algorithm determines subsequent points from the start poit by making a decision between the two next available points by determining which is closer to the ideal point.

The algorithm for the first octant (i.e. where the slope is between zero and one) can be summarized as follows:

```
Calculate absolute values of changes in $x$ and $y$ coordinates between endpoints
Calculate initial decision value
Calculate decision variable increments
          one for choosing point to right and one for choosing point to right-up
While endpoint not reached
      Plot the pixel at the start points
      Check the decision variable
      If the decision variable $P$ is positive then choose point to right and up
      If the decision variable $P$ is negative or zero then choose point to right
      The starting point and decision variable are determined again
```

APPENDIX B

Results for k-NN (dynamic, 8 points)

k	1	3	5	7	9
TR	100.00	99.73	99.65	99.57	99.52
CV	99.47	99.36	99.25	99.20	98.93
WD	99.84	99.68	99.73	99.68	99.47
WI	97.57	97.91	97.77	97.51	97.34

Results for Gaussian k-NN (dynamic, 8 points)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	99.41	99.41	99.36	99.36
WD	99.79	99.73	99.84	99.79
WI	97.83	97.91	97.86	97.68

Results for Fuzzy k-NN (dynamic, 8 points)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	99.31	99.31	99.31	99.25
WD	99.73	99.73	99.73	99.68
WI	97.97	97.77	97.60	97.54

Results for k-NN (dynamic, 12 points)

k	1	3	5	7	9
TR	100.00	99.73	99.68	99.60	99.52
CV	99.41	99.41	99.31	99.09	99.04
WD	99.68	99.79	99.73	99.68	99.57
WI	97.94	98.11	97.97	97.80	97.60

Results for Parzen Windows (dynamic, 12 points)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	99.41	99.36	99.36	99.41
WD	99.79	99.73	99.79	99.79
WI	97.97	98.06	98.00	97.94

Results for Fuzzy k-NN (dynamic, 12 points)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	99.41	99.36	99.36	99.25
WD	99.79	99.79	99.79	99.73
WI	98.14	98.00	97.83	97.88

Results for k-NN (dynamic, 16 points)

k	1	3	5	7	9
TR	100.00	99.73	99.65	99.57	99.55
CV	99.41	99.25	99.31	99.09	99.04
WD	99.79	99.73	99.68	99.73	99.68
WI	97.83	98.17	97.97	97.83	97.43

Results for Parzen Windows (dynamic, 16 points)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	99.36	99.36	99.36	99.41
WD	99.73	99.73	99.79	99.79
WI	97.94	98.06	98.03	97.94

Results for Fuzzy k-NN (dynamic, 16 points)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	99.36	99.36	99.36	99.25
WD	99.73	99.73	99.79	99.73
WI	98.17	97.91	97.88	97.88

Results for k-NN (static, 8 by 8 resolution)

k	1	3	5	7	9
TR	100.00	99.28	98.85	98.56	98.29
CV	98.29	98.56	98.02	97.60	97.33
WD	97.92	97.81	97.76	97.54	97.49
WI	95.65	95.74	96.00	95.94	95.60

Results for Parzen Windows (static, 8 by 8 resolution)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	98.51	98.40	98.24	98.13
WD	98.29	98.18	98.40	98.18
WI	96.17	96.28	96.46	96.48

Results for Fuzzy k-NN (static, 8 by 8 resolution)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	98.51	98.13	97.86	97.00
WD	98.08	98.02	97.92	97.81
WI	96.03	96.26	96.08	95.91

Results for k-NN (static, 12 by 12 resolution)

k	1	3	5	7	9
TR	100.00	98.75	97.84	97.33	96.91
CV	97.60	97.17	96.69	96.26	95.89
WD	97.44	96.90	96.53	96.00	95.46
WI	94.48	93.77	93.54	93.17	92.45

Results for Parzen Windows (static, 12 by 12 resolution)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	97.65	97.54	97.28	97.22
WD	97.49	97.49	97.17	96.85
WI	94.48	94.43	94.31	94.11

Results for Fuzzy k-NN (static, 12 by 12 resolution)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	97.33	97.01	96.90	96.58
WD	97.28	96.64	96.21	95.94
WI	94.17	94.00	93.45	93.14

Results for k-NN (static, 16 by 16 resolution)

k	1	3	5	7	9
TR	100.00	97.81	96.08	95.65	94.58
CV	96.42	94.98	93.86	93.59	93.06
WD	95.46	95.09	95.09	94.34	93.54
WI	91.34	90.65	90.05	89.77	89.08

Results for Parzen Windows (static, 16 by 16 resolution)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	96.42	96.16	95.78	95.41
WD	95.73	96.00	95.84	95.68
WI	91.82	91.45	91.45	91.08

Results for Fuzzy k-NN (static, 16 by 16 resolution)

k	3	5	7	9
TR	100.00	100.00	100.00	100.00
CV	95.62	95.03	94.39	94.02
WD	95.52	95.68	94.82	94.39
WI	91.25	90.85	90.54	90.31

REFERENCES

1. Tappert C. C., C. Y. Suen, and T. Wakahara, "The State of the Art in On-line Handwriting Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 8, pp. 787-808, August 1990.

2. Guyon, I., P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard, "Design of a Neural Network Classifier Character Recognizer for a Touch Terminal," *Pattern Recognition*, Vol. 24, No. 2, pp. 105-119, 1991.

3. Manke S., and U. Bodenhausen, "A Connectionist Recognizer for On-line Cursive Handwriting Recognition," in *International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, (Adelaide, South Australia), pp. 633-636, IEEE Signal Processing Society, April 1994.

4. Schenkel M., H. Weissman, I. Guyon, C. Nohl, and D. Henderson, "Recognition-based Segmentation of On-line Hand-printed Words," in S. J. Hanson, J. D. Cowan, and C. L. Giles (eds.), *Advances in Neural Information Processing Systems*, Vol. 5, (Denver, Colorado), pp. 723-730, Morgan Kaufman, 1993.

5. Nathan K. S., J. R. Bellegarda, D. Nahamoo, and E. J. Bellegarda, "On-line Handwriting Recognition Using Continuous Parameter Hidden Markov Models," in *International Conference on Acoustics, Speech, and Signal Processing*, Vol. 5, (Minneapolis, Minnesota), pp. 121-124, IEEE Signal Processing Society, April 1993.

6. Schenkel M., D. Henderson, and I. Guyon, "On-line Cursive Script Recognition Using Time Delay Neural Networks and Hidden Markov Models," in *International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, Adelaide, South Australia, pp. 637-640, IEEE Signal Processing Society, April 1994.

7. Cho, S., and J. H. Kim, "Combining Multiple Neural Networks by Fuzzy Integral for Robust Classification," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-25, No. 2, pp. 380-384, February 1995.

8. Hoffman, J., Skrzypek, and J. J. Vidal, "Cluster Network for Recognition of Handwritten, Cursive Script Characters," *Neural Networks*, Vol. 6, pp. 69-78, 1993.

9. Suen, C. Y., R. Legaut, C. Nadal, M. Cheriet, and L. Lam, "Building a New Generation of Handwriting Recognition Systems," *Pattern Recognition Letters*, Vol. 14, pp. 303-315, April 1993.

10. Srihari, S. N., "Recognition of Handwritten and Machine-printed Text for Postal Address Interpretation," *Pattern Recognition Letters*, Vol. 14, pp. 291-302, April 1993.

11. Lee, S., and J. C. Pan, "Unconstrained Handwritten Numeral Recognition Based on Radial Basis Competitive and Cooperative Networks with Spatio-temporal Feature Representation," *IEEE Transactions on Neural Networks*, Vol. 7, No. 2, pp. 455-474, March 1996.

12. Guyon, I., UNIPEN 1.0 Format Definition, ftp://ftp.cis.upenn.edu/pub/UNIPEN-pub/definition/unipen.def, 1994.

13. Kassel, R. H., "A Comparison of Approaches to On-line Handwritten Character Recognition," Ph.D. Dissertation, Massachussets Institute of Technology, 1995.

14. Guerfali, W., and R. Plamondon, "Normalizing and Restoring On-line Handwriting," *Pattern Recognition*, Vol. 26, No. 3, pp. 419-431, 1993.

15. Bellegarda, E. J., J. R. Bellegarda, D. Nahamoo, and K. S. Nathan, "A Fast Statistical Mixture Algorithm for On-line Handwriting Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 12, pp. 1227-1233, December 1994.

16. Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*, second ed., Reading, Massachussetts: Addison-Wesley Publishing Company, Inc., 1990.

17. Duda, R. O. and P. E. Hart, *Pattern Classification and Scene Analysis*, New York: John Wiley and Sons, Inc., 1973.

18. Keller, J. M., M. R. Gray and J. A. Givens, "A Fuzzy k-Nearest Neighbor Algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-15, No. 4, pp. 580-585, July/August 1985.

19. Rumelhart D. E., G. E. Hinton, and J. L. McClelland, "A Framework for PDP," in D. E. Rumelhart, J. L. McClelland (eds.) *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol. 1, Cambridge, MA, 1986.

20. Bridle, J. S., "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in F. Fogelman-Soulie, J. Herault (eds.), *Neurocomputing*, pp. 227-236, Berlin: Springer Verlag, 1990.

21. Werbos, P. J., "Backpropagation Through Time: What It Does and How to Do It," *Proceedings of the IEEE*, Vol. 78, No. 10, pp. 1550-1560, October 1990.

22. Williams, R. J., and D. Zipser, "A Learning Algorithm for Continually Fully Recurrent Neural Networks," *Neural Computation*, Vol. 1, pp. 270-280, 1989.

23. Alimoğlu, F., and E. Alpaydın, "Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwriting Recognition," in *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium*, Istanbul, 17-28 June 1996, pp. 115-122.

24. Hansen, L. K., and P. Salamon, "Neural Network Ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 10, pp. 993-1001, October 1990.

25. Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation*, Vol. 3, pp. 79-87, 1991.

26. Alpaydın, E., and M. I. Jordan, "Local Linear Perceptrons for Classification," *IEEE Transactions on Neural Networks*, Vol. 7, No. 3, pp. 788-792, May 1996.

27. Wolpert, D. H., "Stacked Generalization," Neural Networks, Vol. 5, pp. 241-259, 1992.

28. Drucker, H., R. Schapire, and P. Simard, "Improving Performance in Neural Networks Using a Boosting Algorithm," in S. J. Hanson, J. Cowan, L. Giles (eds.), *Advances in Neural Information Processing Systems*, 5, pp. 42-49, Morgan-Kaufmann, 1993.

29. Kaynak, C., “Methods of Combining Multiple Classifiers and Their Application to Handwritten Digit Recognition,” MS. Thesis, Boğaziçi University, 1995.

30. McLachlan, G., *Discriminant Analysis and Statistical Pattern Recognition*, Wiley, 1992.

31. Jordan, M. I., R. A. Jacobs, “Hierarchical Mixtures of Experts and the EM Algorithm,” *Neural Computation*, Vol. 6, pp. 181-214, 1994.