Improvements in Handwritten Character Recognition: a Mixture of Experts

Gabriele Pieri
pieri@cli.di.unipi.it
gpieri@cs.vu.nl
Gabriele.Pieri@cwi.nl

Universitá degli Studi di Pisa
Vrije Universiteit (VU) Amsterdam
Centrum voor Wiskunde en Informatica (CWI) Amsterdam

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

July 5, 2000

Supervisors:
Han la Poutré, Sander Bohté (CWI)
Catholijn Jonker (VU)
Alessandro Sperduti (Universitá degli Studi di Pisa)

Aknowledgments:

I would like to thank first of all Sander Bohté for following me during all my period in CWI, for the patience, the help, and for believing in the success of this work. Thanks to all my supervisors in CWI, VU and Universitá di Pisa, and to all the people in the research group Software Engeneering 4 at CWI. Finally thanks to everybody else who helped me.

Thank you!

Gabriele Pieri

ABSTRACT

Hand-written character recognition is a field of research with possibly important developments, and still open to researches. Several types of recognition methods over different types of on-line or off-line classification exist. A relatively new type of Neural Network, Temporal NN, is used for classification and compared with other classical handwritten character recognition methods. I study also a classification via multiple level algorithms in a hierarchical manner, that is the combination of the above-mentioned method with other already studied methods to achieve a better recognition rate.

First task was implementation of a Graphical User Interface to record characters (digits & letters), and the Preprocessing and Processing phase. The latter is the core of the recognition and has been realized in several different forms to performs and fit different types of classification. Classification is made on the base of a two level hierarchy of classifiers.

In the first phase, experiments were made to test single algorithms for classification and comparison to existing classifiers. Subsequently tests on the full two level hierarchy were performed, trying to find combinations which yield the best performances.

Compared to some results in literature the hierarchical mixture of experts seems to improve performance and to be a promising technique. An issue that is to be taken into account is, that this system does not make use of any rejection rate as widely used in literature. This could be a future development at a very low cost.

Table of Contents

1 Introduction **7**

2 Previous research **10**
1 Statistical approach . . . 11
2 K-Nearest Neighbour . . . 11
3 Retrieving temporal patterns . . . 11
4 Backpropagation implementation . . . 12
5 Hierarchical classifiers . . . 13
6 Temporal encoding . . . 14
7 On-line example . . . 14
8 Why hierarchical classification ? . . . 15

3 Implemented System **16**
1 Overview . . . 16
2 Implementation of GUI . . . 16
3 Pre-processing . . . 17
4 Processing . . . 19
4.1 Circular Gaussian Receptive Field Processing . . . 21
4.2 Multiple Oriented Gaussian Receptive Fields Processing . . . 22
4.3 Local Velocity Based Processing . . . 23
4.4 Spike Processing . . . 24
5 Lower Level Classification . . . 26
5.1 K-Nearest Neighbour . . . 27
5.2 Classical Error-Backpropagation . . . 27
5.3 Crossvalidation . . . 27
6 Higher Level Classification . . . 28

4 Experimental Results **30**
1 Overview . . . 30
2 Single Classifier . . . 30
2.1 Choice of parameters in KNN and semi-linear function . . . 30

2.2 KNN, BP and KNN vs. BP experiments 31
2.3 Experiments on restricted sets 32
3 Multiple classifiers 34
3.1 Decreasing number of classifiers used 35

5 Temporal pattern **37**
1 Overview 37
2 Introduction to Spiking Neurons Networks 37
3 Derivation of an Error Backpropagation for Networks of Spiking Neurons 38
4 Experiments 38
4.1 Experiment on Spiking Neurons Network only 39
4.2 Experiment with the addition of the SNN classifier to the hierarchy 39
4.3 Experiments with the substitution of a classifier in the original hierarchy 40

6 Conclusion **44**

References **46**

List of Figures

3.1 Graphical User Interface 17
3.2 Example of a recorded character 18
3.3 Example of character before Pre-processing 19
3.4 Modification after Pre-processing 19
3.5 Effect of Pre-processing on recorded characters: edge's problem 20
3.6 Example of Circular Receptive Fields 21
3.7 Example of a multiple elliptical receptive field 22
3.8 Example of Local Velocity Processing 24
3.9 Example of Spike Processing 26
3.10 Hierarchical Multilevel Classification 29

4.1 Choice of the parameter for semi-linear normalization function 32

5.1 Spiking Neurons Network 38

List of Tables

3.1 Example of an input to the higher level classifier (i.e. Error Backpropagation). Columns C_1 to C_4 represent inputs to the Neural Network, A is the desired output 28

4.1 Average results of crossvalidation on 304 digits set (divided into 4 sets) to find out the best K value for K-Nearest Neighbour algorithm. Between 3 and 5, the latter value was chosen due to the possibility of increasing the number of compared examples (see last experiments at the end of this chapter). 31
4.2 First results on K-Nearest Neighbour algorithm over different types of processing 31
4.3 For KNN the comparison is made only with Circular Processing. Thus differences are even more relevant 33
4.4 Results on a restricted number of examples (4) from 9 classes 33
4.5 Results on a restricted number (15) of examples from 17 different classes 33
4.6 Results of the hierarchical classifier compared to each single classifier used. Four KNN classifiers are used in the lower level, one BP classifier in the top level. The set is the original one with 304 digits (9 classes). Results on single classifiers differs from the ones in Table 4.2 because crossvalidation was implemented in a different manner, in order to operate on smaller sets, thus faster, even if with a decrease in the performance. 34
4.7 Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Velocity Processing one. 35
4.8 Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Multiple Oriented Processing one. 35
4.9 Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Spike Processing one. 36
4.10 Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Circular Processing one. 36

5.1 Results of Spiking Neurons Network Error Backpropagation over the 304 digits set . . . 39

5.2 The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier. For example: the *4* in the cell indexed (2,3) means that there were 4 characters actually *twos* recognized as *threes*. 40
5.3 The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier. 40
5.4 The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier. 41
5.5 The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier. 41
5.6 The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier. 42
5.7 Results of the hierarchical classifier compared to each single classifier used. Four KNN classifiers and the SNN Backpropagation classifier are used in the lower level, one BP classifier in the top level. The set is the original one with 304 digits (9 classes). 42
5.8 Comparison of performance, on a single set (Set number 2) from crossvalidation, between hierarchical classifier with and without SNN Error Backpropagation in the lower level 42
5.9 Comparison of performance, on a single set (Set number 3) from crossvalidation, between hierarchical classifier with and without SNN Error Backpropagation in the lower level 42
5.10 Results of different hierarchical classifiers compared to each single classifier used. Three KNN classifiers and the SNN Backpropagation classifier (here in place of the KNN with Velocity Processing that was the one with the worse performance) are used in the lower level, one BP classifier in the top level. The set is the original one with 304 digits (9 classes). In the last two rows is made a comparison with previous results: hierarchical classifier with 5 different classifiers in lower level (see Table 5.7) and hierarchical classifier with the 4 original KNN classifiers (see Table 4.6) 43

Chapter 1
Introduction

Machine recognition of handwritten characters continues to be a topic of intense interest among many researchers, primarily due to the potential commercial applications in such diverse fields as document recognition, check processing, forms processing, address recognition etc.

The need for new techniques arises from the fact that even a marginal increase in recognition accuracy of individual characters can have a significant impact on the overall recognition of character strings such as words, postal codes, ZIP codes, courtesy amounts in checks, street number recognition etc.

Accurate, automatic, writer-independent recognition of unconstrained handwriting remains elusive. Writer-independent means that a system must be able to recognize handwriting from different writers without need of changing any of its part. Unconstrained means that the writers are not forced to change their natural writing styles, that could be cursive, discrete or a mix of both.

One way around this problem is to constrain writing style, for example, by limiting users of such systems to write standard characters strictly discretely, or, more severely, to use special character sets designed to maximize recognizability. However, many people's own natural style of writing contains a mix of cursive and discrete writing, with letters not strictly all separated or all connected within words. To allow users to write naturally and be recognized accurately, it is therefore frequently necessary for them to go through a training process in which they provide samples of their own writing so that the recognizer can adapt to their styles. Such writer-dependent recognition normally makes only a fraction of the errors that the same system would make running in a writer-independent mode.

Recognition techniques have been based on syntactic, structural, statistical or neural networks methodologies.

Neural networks have proven to be a promising technique for bitmap pattern recognition. The prototypical use of neural networks is in *structural pattern recognition.* In such a task, a collection of features—visual, semantic, or otherwise—is presented to a network and the network must categorize the input feature pattern as belonging to one or more classes. For example, a network might be trained to classify animal species based on a set of attributes describing living creatures such as "has tail", "lives in water", or "is carnivorous"; or a network could be trained to recognize visual patterns over a two-dimensional pixel array as a letter in $\{A, B, \ldots, Z\}$. In such tasks, the network is presented with all relevant information simultaneously.

In contrast, *temporal pattern recognition* involves processing of patterns that evolve over time.

The appropriate response at a particular point in time depends not only on the current input, but potentially all previous inputs. Following the assumption made by Mozer in [20] one can assume that time is quantized into discrete steps, a sensible assumption because many time series of interests are intrinsically discrete, and continuous series can be sampled at a fixed interval.

The basic problems associated with neural networks may be summarized as follows:

- The need to use a large number of representative samples is of crucial importance in training the neural network. This often results in a fairly slow learning rate. Also convergence of the learning algorithm is often not easy to achieve.
- The neural network itself becomes excessively large, if the individual pixels of the character image are used as input. Even at a low spatial resolution of 16×16 for a character image, one has to deal with 256 input nodes. When this is combined with the nodes distributed across the hidden layers, one is faced with the awesome task of training a very large network
- The amount of training a neural network receives is crucial in achieving high recognition accuracy with test characters. It is possible to over-train or under-train a neural network,resulting in poor performance in real world recognition applications

In this work of thesis the chosen approach follows two main directives:

- Improving the performance of already existing classifiers with a relatively novel methodology.
- Exploit temporal pattern of recorded writings to improve performance.

Due to the above-mentioned problems with neural networks, and to the research proposals, the approaches that this thesis follows are:

1 – Implementation of utility program to *record characters* without any burden of pixel mapping and image gray-level managing or processing.

2 – Exploitation of *time aspects* to improve performance, balancing with the intrinsic problems that a realistic model of temporal processing (i.e. the decoding of temporal codes) has.

3 – Use of a *mixture of experts* (i.e. classifiers), in different hierarchical levels to improve performance trying to keep the computationally costs low.

With the aim of trying a different approach, and with the need of having a time component in the recorded data, the shortest and best way was to implement an interface to record characters on from scratch. Due to the main aim of the research, the exploitation of temporal pattern, instead of putting the stress on the graphical structure of the recorded characters, the stress has been put on the easier task of retrieving spatiotemporal coordinates.

The second point is linked with the use of a particular novel approach to neural networks. This approach, which takes into account the time component, is a spiking neurons network (SNN). There exists some evidence during the last years which indicates that many biological neural systems use the timing of single action potential (or "spikes") as a way of coding information. Hence the spike-time paradigm for neuronal information processing has been receiving increased attention (e.g. with the derivation of an error-backpropagation algorithm for a network of spiking neurons [2]). It follows that applying SNN to a task like character recognition, where the temporal pattern can be retrieved as a main feature to be exploited, is straightforward because of the temporal nature of Spiking Neurons Networks.

Another reason for trying Spiking Neurons Networks as an approach, is to try a somehow different approach that is to exploit time as an additional dimension and not as a burden, like often reported in literature [4, 22, 28].

The last point, a mixture of different experts (i.e. classifiers), is the main point of this research. The use of different experts is to grant a decrease in the error rate with regard to the single classifiers, while the use of simple (in term of computational complexity) classifiers and a low number of samples allows to the whole system to be computationally acceptable.

The system is based on hierarchical levels of classifiers. The higher level has only one *final* classifier which is the one who gives the answer to the user. The previous levels (actually only two levels are implemented) serve as an input to the next.

In this work various solutions for this hierarchical structure are analyzed. Many modifications are tried to find the better performance among all various combination.

The organization of this thesis is as follows. In chapter 2 there is an outline on the literature related to character recognition and several different approaches, that were useful for the implementation of the subsequent system. In chapter 3 one can find details about the implementation of the system. Chapter 4 describes some first experimental results without the use of the SNN approach. The following chapter 5 introduces temporal patterns in more detail, the way SNN is used, some experimental results, and a comparison to previous ones. Finally conclusions and a summary are given in chapter 6.

Chapter 2
Previous research

Several successes have been reported in applying Artificial Neural Networks (ANN) to hand-written character recognition either *off-line* or *on-line*.

The difference in these two approaches lies in the situation where they are required to work. When the system is supposed to give a result in real-time, just after the user has finished writing, this is called on-line recognition. In on-line recognition data are collected on an electronic tablet that traces the movement of the pen, thereby preserving temporal information.

The other situation happens when the system has the possibility of receiving all the data relative to the writing. Using this approach the system has the time of processing without need of giving a real-time answer to the user. The advantage is also that an off-line system is not required to run on small, therefore less powerful, machines. This type of system usually is allowed to run on workstations or anyway powerful machines. Notice that typically the temporal information is not preserved in off-line systems.

It seems that near-human performance is within reach, at least in cases where digits are accurately segmented [18]. In the literature we can find many different approaches some of which are explained more in detail in sections 1 through 7. Starting from the naive k-Nearest Neighbor (KNN) classifier to Radial Basis Function (RBF) networks, Backpropagation (BP) networks, and the newest approach: Temporal Networks.

Starting from the work of Lee [14] where he makes a comparison between three different approaches (KNN, RBF and BP), they were shown to have equivalent classification accuracy in a large hand-written digit-recognition task. These classifiers differ substantially in memory usage, training time, and classification time. Such implementation characteristics, not the error rate, dictate the feasibility of these classifiers.

In the next sections some of the above-mentioned approaches to character-recognition are listed that seemed to be of some interest or useful to understand the problems typically encountered. In section 1 some statistical approaches to character recognition are explored, while in following section 2 classifiers using K-Nearest Neighbour algorithm are discussed. Section 3 shows some approaches that are used to retrieve temporal patterns from recorded characters, and how are they used for classification. In section 4 a brief overview on the use of Error Backpropagation in Artificial Neural Networks in the literature for the purpose of character recognition is given. Section 5 focuses on our main idea, the hierarchical combination of classifiers, and how it has been studied by other authors. Section

6 reports an experiment by Buonomano et al. [4] on a possible exploitation of the time component, and temporal encoding. Later in section 7 an on-line example of handwritten character recognition is reported, which summarizes some of the approaches previously reported. Finally section 8 explains why a hierarchical mixture of classifiers is the approach we took.

1. Statistical approach

There are some statistical approaches based on feature extraction. Due to the fact that statistical selection of characters into classes require short time, in [27] a system is presented which is based on Fuzzy statistical measures to identify relevant features for a rule-base generation. The system is designed to extract general rules, based on characters' features[1], to classify other characters. Rules are fuzzy, because every character possess in some measure every feature.

This process, of extracting fuzzy rules automatically, can be speeded up, if from the possible number of feature combinations only the most relevant or meaningful combinations for each character are extracted and analyzed.

Often the time complexity component is critical, as for instance in on-line recognition, thus the aforementioned statistical approach can give good enough results as in [22] (see section 7 later).

2. K-Nearest Neighbour

A quite obvious and simple (compared to more complex algorithms or Neural Networks) way to classify digits is using the KNN algorithm or variations of it. This technique performs a classification of an input pattern X by directly comparing it with a large number of reference patterns (that are called *prototypes*) $Y_i : i = 1 \ldots M$. The classification value is determined by the K "nearest" patterns in a certain metric. That is, a majority voting is performed between the above-mentioned K "nearest" patterns, to assign the tested character to the most "voted" class.

Another technique is the k-means clustering [17], which is a modification of k-nearest neighbour. In [25] this is used in a slightly modified way, with a binary clustering technique for the generation of large prototypes sets. The algorithm is employed in a learning phase on each class separately. The resulting prototypes are average vectors of attributes training patterns and therefore have real components even when the training patterns are binary. However, the k-means algorithm can be slightly modified by replacing the mean values by a majority rule, and it results in a modified k-nearest neighbour algorithm.

This kind of approach is suitable, and thus fast, when the number of training examples is small and may contribute complementary information. On the other hand when the number of prototypes is large, this method requires a large memory space and long computing time.

Yan [34] tries to overcome this with the use of Neural Networks in combination with the above-mentioned approach. His goal is handwritten digit recognition. Yan suggests to generate prototypes (using either the k-means clustering or other clustering methods), then these prototypes are optimized using a multi-layer perceptron to increase their classification power. Each hidden node of the perceptron corresponds to a prototype and characters to be tested are passed through as inputs. Finally the trained perceptron is mapped back to a nearest neighbour classifier for the final classification.

A comparison is made between the recognition rates with and without optimization (perceptron) of the prototypes. The performance increases on average around 4%, and the best recognition rate is obtained with 500 prototypes used, 96.1% on test set, but the training time is more than 25 hours and recognition time more than two minutes (each character).

3. Retrieving temporal patterns

A very important feature, that represents a very relevant characteristic in our implemented system, is the temporal pattern feature and its use to improve classification accuracy.

[1] Example of these features are characteristics in the trace of the character like: *vertical line, O-like curve, A-like curve* ...

Using an approach as in [13] one could transform a two-dimensional (2-D) spatial representation into a three-dimensional (3-D) spatiotemporal representation by identifying the tracing sequence based on a set of heuristic rules acting as transformation operators. In that way one retrieves the temporal pattern from an off-line environment.

In [13], a new neural network architecture is proposed, called radial-basis competitive and co-operative network (RCCN). The network is composed of three layers: input layer of n linear units, hidden layer of one or more nonlinear (or semilinear) units, and output layer of m units, constituting an multi-layer network between an n-dimensional input space and an m-dimensional output space, that can be a reduction from a larger number of features.

The network is said to be "globally competitive and locally co-operative". Globally competitive because among existing units in the hidden layer only one, in response to a training example, would contribute to the network output via winner-take-all competition. Locally co-operative because neighbouring units, in the network, co-operate with one another to enhance the capability of the network during mapping or classification. The co-operation is implemented with an interpolation among the formed clusters (i.e. m centers of the output space).

In [13] the authors make also a comparison between RCCN and other conventional competitive learning paradigms like ART [6], Counterpropagation Network [11] and classical radial basis function networks which exploit Gaussian activation functions as the transfer functions of hidden units [8, 19, 29]. They report as result of their technique, on digits recognition, a maximal performance of 96.5% of digits recognized. Without rejection of uncertain digits their result decreases to a maximal of 95.3%.

4. BACKPROPAGATION IMPLEMENTATION

With the purpose of optical character recognition (OCR) of printed documents, Avi-Itzhak et al. in [1] propose an approach with a BP Neural Network. The interesting part is the pre-processing and normalization phase of the characters before these are fed to the neural network.

Before it is fed to a neural network, the digitized image they recorded is pre-processed and normalized.

The pre-processing and normalization procedure serves several purposes; it reduces noise sensitivity and makes the system invariant to point size, image contrast and position displacement [2].

The two main steps of pre-processing are centering and scaling [3]. The image is centered by positioning the centroid of the image to the center of a fixed size frame.

For the multi-size and multi-font case, an additional scaling process must be applied to the images. The image is enlarged or reduced with a gain factor, that is calculated from characteristics of the image, so to have images with a constant value of a certain parameter[4] (called in [1] "radial moment").

The next step is to convert the two-dimensional images into vectors. The conversion is achieved by concatenating the rows of the two-dimensional pixel array.

Additionally, each vector is normalized to unit power (i.e. normalized dividing each component by vector's norm). The normalization reduces sensitivity to varying scanner gains (image-to-background contrast) as well as different toner darkness (shades of ink). This unit-norm vector is then fed into the neural network.

Hasegawa et al. in [10] show another example of error Backpropagation network, used in conjunction with a Restricted Coulomb Energy (RCE) network [30], to implement a so-called *hyper-spherical surface interactive interconnection* (HSII) learning method for category learning.

In this method the aim is the recognition of printed digits on medical X-ray films in hospitals.

The RCE is known as a powerful algorithm for category learning. The only potential problem is that the number of hidden units is apt to excessively increase if training examples posses noise. This is because weights between the input and hidden layers are extracted from the training examples

[2] In my described system the first two problems (point size and image contrast) do not occur because the recorded characters are not gray scale pixmap

[3] They start performing a thresholding due to the fact that they are operating over gray scale images

[4] Thus this scaling process can be seen as a normalization of the character to ease the real classification process

themselves which is probably not optimal.

To solve this problem, the authors propose a learning algorithm, the above-mentioned HSII. The network optimizes the number of hidden units and adjusts the connection weights between the input and hidden units. The optimization of the number of hidden units is carried out as in the RCE. The weights and biases are iteratively adjusted by error Backpropagation.

They operate on N-dimensional pattern vectors which have to be categorized into classes. Every input vector is normalized to satisfy condition that all data are distributed on a hyper-spherical surface of known radius. The HSII has two learning modes: (1) improving weights and biases by error Backpropagation and (2) adding a new hidden unit. One of this two is chosen each time based on the values of the back-propagated error signals.

Results on a set of 1200 elements yielded a recognition rate of 99.4% with a rejection rate of 0.6%. On the other hand RCE yielded 97.8% with a rejection rate of 2.2%.

5. Hierarchical classifiers

In this section we will see some architectures which make use of multiple classifiers designed in a hierarchical manner. Some of these characteristics will be used in our implemented system, whose main feature is exactly the multiple classifiers hierarchy.

Cao et al. in [5], in contrast to a supervised algorithm like Backpropagation, propose a hierarchical neural network architecture for the recognition of handwritten numerals.

The recognition system uses two neural networks that can be trained separately to perform the basic recognition task. The first network which functions as a feature extractor generates principal components (i.e. features) of the numeral image using Oja's rule [26]. The second neural network which functions as a classifier assigns the output of the first network to one of the clusters (numeral sub-classes). The second network is essentially an unsupervised clustering network that uses a weighted distance measure to derive the clusters. The considered distance is between cluster centers and the above-mentioned principal components extracted by the first network. The recognition system they describe can be considered as a neural implementation of a statistical classifier.

The authors propose the use of conditional probability functions to guide the clustering process. Assuming Gaussian distribution for the principal features, the conditional probability of a feature vector belonging to a given sub-cluster is evaluated. If this probability is low with respect to each of existing clusters, then a new cluster is initiated. After all the training samples have been used, merger operations are applied to reduce the number of clusters.

The decomposition of the global network into two independent networks facilitates rapid and efficient training of the individual network. The tests show that the proposed system outperforms a Backpropagation Neural Network, but only with low rejection rates.

Another example of the use of two different classifiers to recognize hand-written digits is the ZIP codes recognition system in [28]. One of the classifier is a time delayed neural network (TDNN) [12, 32] used to classify scaled digit-feature. The other classifier extracts the structure of each digit and performs a prototype matching.

The authors tried this approach of combining these two methods in order to produce a system as writer independent and reliable as possible. The first one is a powerful but more computationally intensive pixel oriented neural classifier, the other a fast and rather reliable method that analyses the structure of the digits. Both classifiers use a pre-processing step to remove some of the digit's variance.

As enunciated also in [31], in a TDNN each hidden unit has a receptive field that is limited by a time delay; that is, hidden units are connected to a limited temporal window within which they can detect temporal features. Since hidden units apply the same set of synaptic weights at different times, they produce similar responses to similar input patterns that are shifted in time.

The second classifier in [28] uses structural information and operates in a multi-stage process: a preliminary phase to construct a structural graph representation then the prototype matching to extract a feature vector and then for those digits which are close to more than one class a classification

stage.

For this latter step they used a neural classifier, specifically a Cascade-Correlation network [9], since these networks are able to adapt their architecture to the difficulty of the problem.

Having two classifiers, the problem of combining their (eventually) conflicting decisions arises. There are two main alternatives, parallel and sequential. For the parallel combination, both classifiers are run and their results are merged by some kind of voting mechanism. For the sequential combination, the simplest and thus lesser time consuming classifier C_0 is run first. If it recognizes the digit with high confidence we don't have to run the other, if it does not recognizes, we run the other classifier and results are merged.

The disadvantage of this method is of course that any misclassification done by C_0 cannot be overruled by the other, therefore we must ensure that C_0 yields very low error rates.

The TDNN used by Seni in [31], is used for cursive word recognition. In such a way to recognize letter by letter, and performing a string matching at the end in order to validate the output interpretation strings produced by the recognizer.

6. Temporal encoding

As an example of a different approach to the time feature, Buonomano et al. in [4] use temporal encoding of a spatial pattern. The authors try to demonstrate a manner by which the nervous system may generate temporal codes and show that temporal encoding of a spatial pattern has the interesting and computationally beneficial feature of exhibiting position invariance. That is, temporal encoding can be used to create position-invariant codes. The position invariance is obtained through the computation of latency histograms which represent the number of activated units (units can be considered as spread on a grid retina-like) at each latency interval. Thus the position of the activated units does not affect the latency histogram.

Conventional neural networks do not exhibit position invariance because information is stored in the spatial pattern of synaptic strengths. In [4] they develop an alternative hypothesis based on temporal coding. They show that local inhibition may underlie the temporal encoding of spatial images.

In [4] their network, units from the input layer provide excitatory input to the topologically equivalent unit in the next layer and inhibitory connections to the neighbouring units. During each time step, the sum of excitation and inhibition is computed. If the sum reaches threshold, a spike occurs. The latency is defined as the time step at which threshold was reached.

The second component of the model consisted of a recognition network, which is necessary in order to determine whether the temporal codes generated could actually be used for position invariant pattern recognition. For this purpose, it is necessary to decode the temporal code.

The next step was to determine whether once the latency codes are mapped spatially, they can be used for digit recognition. For this purpose the output of the previous network is used in a conventional Backpropagation network. Since each orientation sublayer generates a temporal code (i.e. latency), and only 10 different latencies are considered, the input to the backpropagation network consists of a single vector of length 40 (4 direction detectors for each latency). The Backpropagation network contains 16 hidden units and 10 output units (one for each numeral).

The average performance reported for a small set of digits is $93.4 \pm 0.16\%$.

7. On-line example

Nathan et al. in [22] developed a real-time on-line unconstrained handwriting word recognition system using statistical methods.

"Unconstrained" implies that the user may write in any style (e.g. printed, cursive or in any combination of styles). They focus on word recognition and on the writer independent task (i.e. the recognition has to be independent from the user who writes).

In on-line recognition data are collected on an electronic tablet that trails the movement of the pen. Considering the on-line nature of the system much effort was spent in designing a system that could run in real time on a small PC platform with limited memory. To achieve this the system is based on

a statistical method as hidden Markov model (HMM) [23, 21, 7], and due to their implementation it is suitable to be used with a small amount of memory.

The system consists of a pre-processing and feature extraction stage followed by a double matching stage: a fast match (FM) to generate a short list of potential candidate strings and a detailed match (DM), computationally more expensive to reorder each word in the short list, both matching are based on the above-mentioned hidden Markov models.

The only use they make of the time feature is to re-sample (and normalize) spatially the temporally equi-spaced points in their pre-processing step. Every point is mapped spatially from the incoming temporally equi-spaced point, in order to remove what they call "inconsistencies due to velocity". In that way they move back from a three-dimensional space to a two-dimensional space, retrieving only Cartesian coordinates of the points. Thus they remove the time component from their system before the processing step. This is due also to the fact that for a fast recognition they try to have as less free variables as possible.

An error rate of 18.9% is achieved for a writer-independent 21000 word vocabulary task. With an average recognition time per word range from 0.4 sec. to 0.48 sec.

8. Why hierarchical classification ?

Multiple classifier techniques could lead to improvements in performance. Furthermore a strategy with only neural networks as classifiers could be too computationally expensive.

Following these reasons, a mixed approach to the problem of handwriting recognition seems to lead to excellent results and many possible future developments. With a multiple classifier hierarchy there exists the possibility of changing the design of such hierarchy, in order to find out the best way of setting up the system. This in order to obtain a good performance and a less expensive (from a computationally point of view) design.

Chapter 3
Implemented System

1. Overview

In this chapter the implemented system will be explained, starting from the implementation of the Graphical User Interface in section 2, which serves as an utility program to record characters that will be fed to the system to be classified. Section 3 explains the pre-processing stage, where scaling and resizing on the recorded characters are performed. Following section 4 presents the feature extraction (processing) phase over the characters which have been recorded through the interface and subsequently pre-processed. Several types of processing are presented with the aim of extracting different features.

Finally sections 5 and 6 show how the hierarchical classifier works and how the single classifications are performed.

2. Implementation of GUI

For the purpose of recording and checking hand-written characters we developed a graphical user interface. With this (see Figure 3.1) interface the user has the possibility of tracing characters on the left part and pressing one of the buttons on the right to associate a classification to that character (buttons marked with digits and letters), then the new character can be recorded on the blank board by pressing on the *Reset* button. To start recording characters the user has to press the *Start* button.

In principle the interface is able to record 36 characters (a ... z $\cup$ 0 ... 9) considering only lower-case letters, that means to store the traced characters with their relative classifications. The size of the recording tablet in the left side of the GUI is arbitrary, and not relevant to the final classification, due to the following pre-processing step. Thus the size of the original recorded character is not relevant.

The characters are recorded as a sequence of temporal-spaced points. They are represented on the tablet as a sequence of colored dots, following the path traced by the user (see Figure 3.1). Other buttons are available to view characters after Pre-processing, characters just recorded, to discard wrong characters, and some buttons used to move inside the various vectors of recorded characters.

At the end, that occurs when the user presses the *Stop* button, the recorded characters are saved in an "Java-Object" file. The format of the Object (i.e. Character) is a matrix containing Cartesian coordinates and related time-of-passage in every row and rows represents each point of the recorded character.

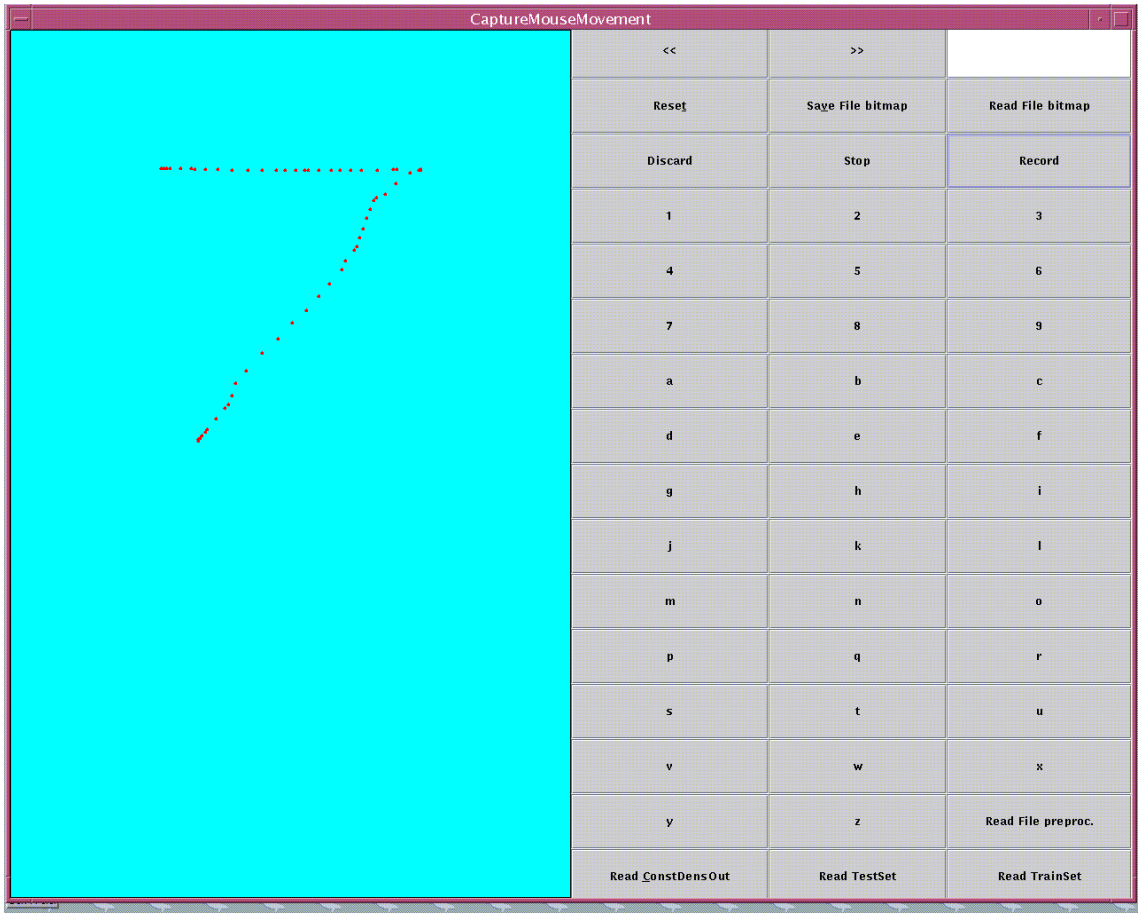

Figure 3.1: GUI: On the left the board to trace characters, on the right the key-pad to perform many possible actions

$$\left\langle \begin{array}{ccc} XCoordinate_1 & YCoordinate_1 & DELAY_1 \\ XCoordinate_2 & YCoordinate_2 & DELAY_2 \\ \vdots & \vdots & \vdots \\ XCoordinate_{NoP} & YCoordinate_{NoP} & DELAY_{NoP} \end{array} \right\rangle$$

Where NoP represents the total number of points recorded for a given character.

In Figure 3.2 is represented what the recorded character looks like.

The delay in the matrix is the time difference between every pair of successive recorded points. The "Java-Object" file format allows an easier access and use of data in the subsequent phase, the pre-processing step.

3. Pre-processing

Reading data from the above mentioned "Java-Object" file, this second step performs a scaling and centering of the character in order to reduce the variances among characters. The size of the final scaled grid is fixed in 150×150 pixels. The modified character are written again into a new "Java-Object" file, ready to be used by every kind of processing that is the following phase. First the characters are shifted to the top-left corner of the grid considering the minimum among all *X-Coordinates* and among all *Y-Coordinates* of the matrix.

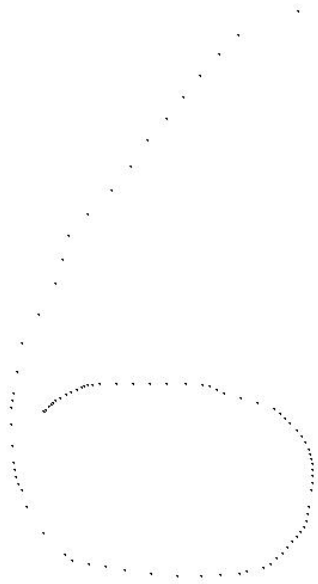

Figure 3.2: The character are recorded as a sequence of points on the sensible grid.

$$\begin{aligned} X' &= X\text{-}Coordinate - Min\text{-}X \\ Y' &= Y\text{-}Coordinate - Min\text{-}Y \end{aligned} \tag{3.3.1}$$

Then the characters are normalized from their original size into the 150 × 150 grid, using this transformation:

$$\begin{aligned} X'' &= \frac{X' \cdot SensibleGridSize}{Max\text{-}X - Min\text{-}X} \\ Y'' &= \frac{Y' \cdot SensibleGridSize}{Max\text{-}Y - Min\text{-}Y} \end{aligned} \tag{3.3.2}$$

Where the sensible grid size as said above is 150.

Opposite to [22] my pre-processing does *not* want to remove "inconsistencies" [1] due to velocity, because velocity can be considered as a very typical feature in characters hand-writing. Considering on-line recognition this is an issue that has to be taken into account to differentiate this from the off-line task.

As suggested by [28] there could be an improvement in the system by performing more sophisticated preprocessing, such as *slant correction* by estimating the principal axis of the digit and shearing it, so that his axis is vertical after the transformation.

In [5] the slant correction is performed estimating the slant of the digit by the direction chain codes of its contour as defined in equation 3.3.3.

[1] In [22] they talk about inconsistencies because their goal is to work on equi-spaced points in space. Thus the velocity features are not interesting for them.

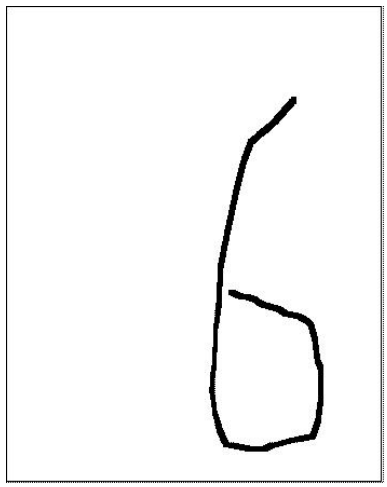

Figure 3.3: Before Pre-processing (The characters are not recorded in this continuous-line style, but as a series of points, as a dotted-line, see Figure 3.2).

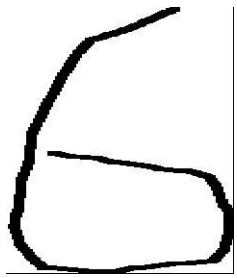

Figure 3.4: After Pre-processing, the character is re-sized and centered into a *150×150* grid.

$$\theta = \arctan\left(\frac{n_1 + n_2 + n_3}{n_1 - n_3}\right), \tag{3.3.3}$$

where n_i is the number of chain code elements detected at each different angle of $i * \frac{\pi}{4}, i = 1, 2, 3$.

4. PROCESSING

This stage represents the heart of the whole system. This is where the digits are encoded in order to perform the classification in the subsequent phase.

The pre-processed data being read in this stage are the output result of the previous phase in the above-mentioned "Java-Object" format.

In order to encode the information for every digit, a 13×13 grid of equi-spaced points (i.e. Neurons with their centers), is spread over the 150×150 sensible grid, starting from outside the edge of the grid, the actual distance between centers is set to 15 pixels. The reason of starting outside of the pre-processed grid is that after pre-processing the edges represent a very important zone of the grid (see Figure 3.5). Thus placing receptive fields outside of the grid (close to the edge) gives a right importance to those peculiar bordering points. These Neurons (i.e. points) are influenced by the nearby points through the distance between these and the *receptive fields* through which the system

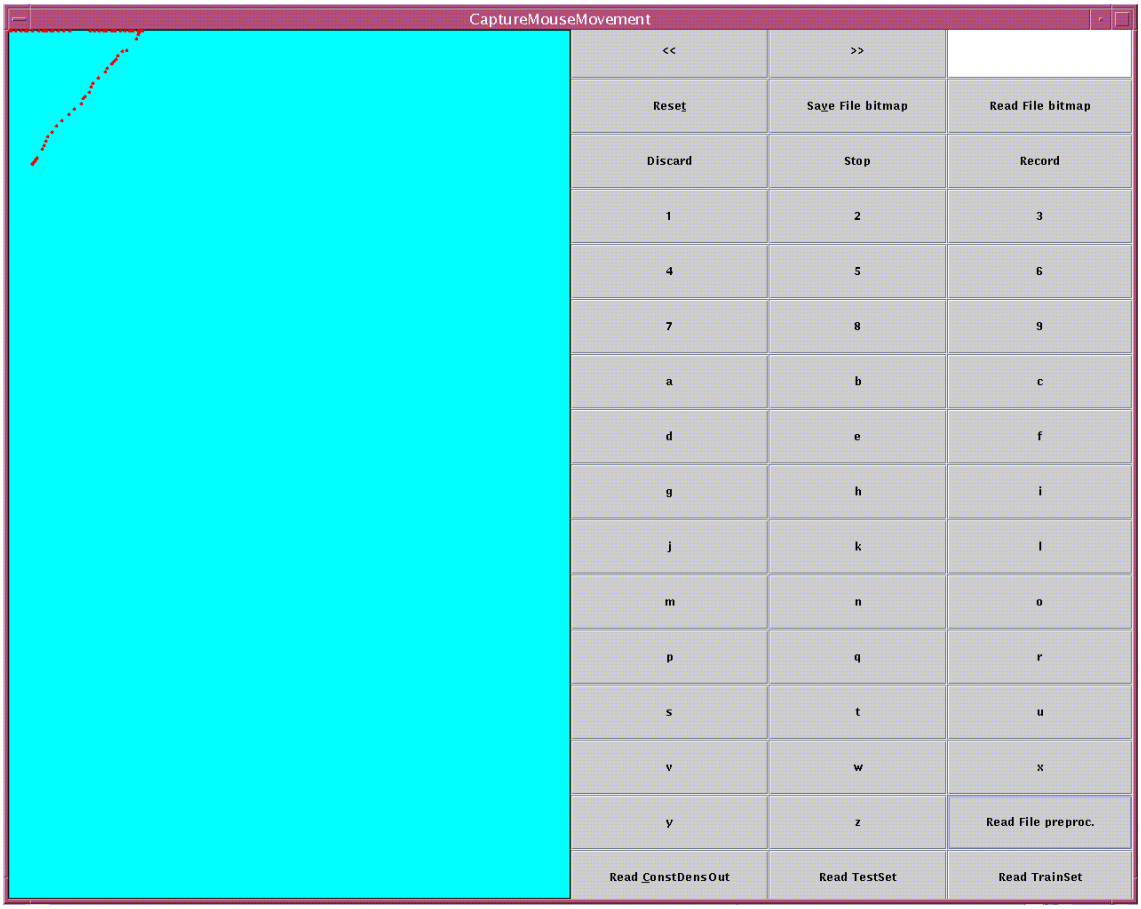

Figure 3.5: The same *seven* as in Figure 3.1 after Pre-processing. The horizontal part of the tracing is almost on the edge of the grid, therefore is important not to lose those points.

encodes characters' features.

A receptive field is a function of the distance between a point and the center of the same receptive field. This function can be different and can take into account other features like velocity, direction of the point, etc ...

In order to obtain different encoding for different features of the digits, there exists different processing:

- Circular Gaussian Receptive Field Processing
- Multiple Oriented Gaussian Receptive Fields Processing
- Local Velocity Based Processing
- Spike-Processing

In the following with few obvious exceptions all the references will be to a single character to simplify the reading, but in the real implementation all this processes are repeated for all the characters in the set.

4.1 Circular Gaussian Receptive Field Processing

Reading data performs a processing based on the distance between every receptive-field's center and all the recorded points.

In Figure 3.6 is shown how Circular receptive Fields look like on the sensible grid.

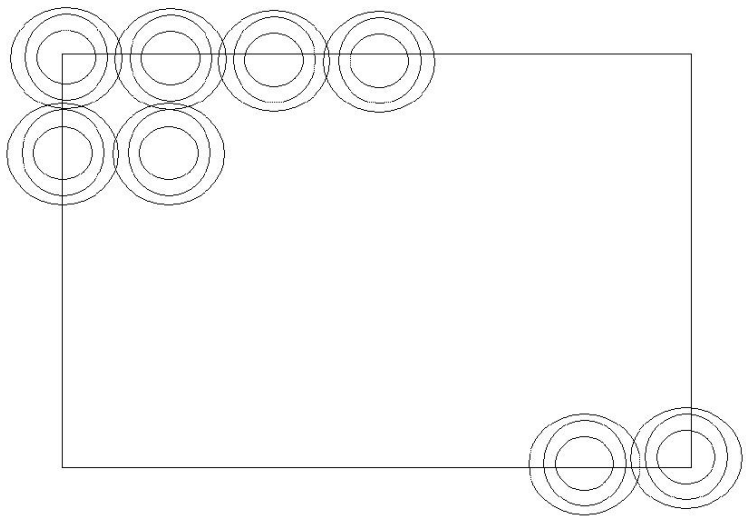

Figure 3.6: Circular Receptive Fields used for the first type of Processing on the sensible grid.

The system uses a Gaussian function of the above mentioned distance:

$$\mathcal{F}(Center_j, Point_i) = e^{-\frac{(distance(Center_j, Point_i))^2}{\sigma^2}} \tag{4.3.1}$$

Where $Point_i$ represents the coordinates x and y of the *i-th* recorded point of the character and $Center_j$ represents the coordinates x and y of the referring *j-th* neuron's center. The actual value of the variance σ^2 is 1.

This 4.3.1 is used in the following formula to find out an actual value for every neuron:

$$\forall \text{ Neuron } j : \mathcal{N}(Center_j) = \sum_{i=1}^{NumOfPoints} \mathcal{F}(Center_j, Point_i) \tag{4.3.2}$$

This value is then normalized in a range that is supposed to be suitable for the Spiking Neurons Network proposed in [2], (but it has been used also by every other classifier) that is an integer value in: $[0 \ldots 9] \cup [-1]$. A -1 value means that the actual value for that neuron was too small (i.e. under a prefixed threshold) and so is set to a *non-firing* value. The other values arise using a logarithmic scale over the resulting output.

Thus finally the activation (firing time) of the *j-th* neuron (for a given character) is given by:

$$\forall \text{ Neuron } j : Act(j) = \begin{cases} -1 & \text{if } \mathcal{N}(Center_j) < \text{THRESHOLD} \\ 0 & \text{if } \mathcal{N}(Center_j) > 0 \\ -\frac{\log \mathcal{N}(Center_j) \cdot 10}{\text{THRESHOLD}} & \text{otherwise} \end{cases} \tag{4.3.3}$$

Thus the resulting *169 (=13·13)* values for each character are written into a text file, which has size: *number of characters* lines and *169+4* columns (i.e. input values), that is the number of Neurons plus the classification value plus 3 control symbols.

This file (the processing-output) will be used as input to feed the various classifiers

4.2 Multiple Oriented Gaussian Receptive Fields Processing
In order to obtain a more accurate classification and to extract different features, this second type of processing has been implemented to retrieve the local direction feature from the characters. The main modifications to the previous processing are:

- 2 different σ values to obtain elliptical (oriented) receptive fields
- 4 different receptive fields for each Neuron (i.e. Center)

Both points accomplish the direction-feature detection. While the first point is thought to increase the importance of points along a certain direction, the second is to achieve different values for different directions

Figure 3.7 shows how a multiple elliptical receptive field looks like.

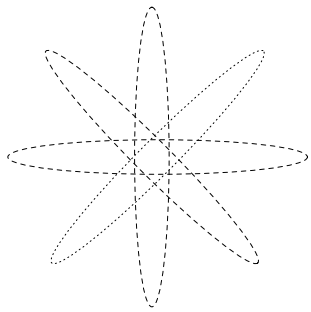

Figure 3.7: A multiple elliptical receptive field. Every receptive field like that substitute every circular recptive field (see Figure 3.6) in Multiple Oriented Gaussian RF Processing.

The Gaussian function changes in this way:

$$\mathcal{F}(C_j, P_i) = e^{-(\frac{C_j^x - P_i^x}{\sigma_x})^2} \cdot e^{-(\frac{C_j^y - P_i^y}{\sigma_y})^2} \tag{4.3.4}$$

Where C_j^x, C_j^y, P_i^x and P_i^y are the x and y coordinates of the center, and the recorded point respectively [2].

In order to obtain different receptive fields orientation, we decided, instead of computing different functions for every different orientation, to re-compute all the points and centers coordinate using an easy rotation matrix:

$$\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{4.3.5}$$

So modifying coordinates x and y with this 4.3.5 we obtain new coordinates $P_{x'}$, $P_{y'}$, $C_{x'}$ and $C_{y'}$ for every point and center.

$$\begin{pmatrix} P_{x'} & P_{y'} \end{pmatrix} = \begin{pmatrix} P_x & P_y \end{pmatrix} \times \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{4.3.6}$$

[2] Actual values used for σ_x and σ_y are 1 and 4 respectively

The Gaussian function from 4.3.4 (that to facilitate legibility is written for a given Center j and a given Point i) changes consequently in this way:

$$\mathcal{F}(C,P) = e^{-\left(\frac{C_{x'}-P_{x'}}{\sigma_x}\right)^2} e^{-\left(\frac{C_{y'}-P_{y'}}{\sigma_y}\right)^2} \tag{4.3.7}$$

To have different orientations we varied the angle α, actually 4 different orientations are used to distinguish the main direction axis. The used angles are: 0, $\frac{1}{4}\pi$, $\frac{1}{2}\pi$ and $\frac{3}{4}\pi$.

The same summation and normalization as in 4.3.2 and in 4.3.3 were used to obtain the final activation values. In this way the size of the output text file is 4 times larger than the previous because of the 4 different angles (i.e. directions) that lead to 4 different receptive fields. Thus now the file has *(169·4)+4* columns (i.e. input values).

4.3 Local Velocity Based Processing

As the quite-original name suggests, this type of processing is thought to be suitable for the extraction of the local velocity feature, that has typical peculiarities, which can be exploited if used in conjunction with other different characteristics.

In the final formulation this processing works with the same receptive fields as the previous processing, thus we still have 4 different oriented-receptive fields which are set to perceive different features in respect to previous processing and the number of outputs is also 4 times larger than the original one. Again the points are recomputed with the rotation matrix 4.3.5. The difference lies in the function 4.3.4 that now takes into account the local velocity value for every point, which is calculated in this way:

$$\text{Velocity}(i) = \frac{\text{distance}(i, i+1)}{\text{delay}(i+1)} \tag{4.3.8}$$

Where distance$(i, i+1)$ is the distance between the *i-th* point and the *(i+1)-th* point, of the recorded character, and delay$(i+1)$ is the time-delay in recording between the same pair of consecutive points, which is stored in the "Java-Object" file as the third component of the matrix.

Changing Velocity(i) into S_i and distance$(i, i+1)$ into D_i, the new formula is the following:

$$\mathcal{F}(C_j, P_i, S_i, D_i) = \textit{sigmoidal}(S_i, D_i) \cdot e^{-(D_{(C_j,P_i)})} \tag{4.3.9}$$

Where $\textit{sigmoidal}(S_i, D_i)$ is the following sigmoidal function, which is weighted (multiplied to a fixed factor W) to increase the relevance of the local velocity component in the function. Again the $D_{(C_j,P_i)}$ is the distance between the Center of neuron j and point i:

$$\textit{sigmoidal}(S_i, D_i) = \frac{W}{1 + e^{(K\cdot((S_i\cdot D_i)+V_0))}} \tag{4.3.10}$$

Where the values of constants K, V_0 and W were chosen to be $-1, -2$ and 10^6 respectively. In this equation 4.3.10, in the exponential, S_i is multiplied by D_i to give more relevance to the velocity component. This is because the Gaussian equation 4.3.9 is used in a summation of points, but the number of points is not fixed, thus a character with more points recorded, but for instance with slower velocity, could yield the same value as another one with less points, but recorded at higher velocity.

In Figure 3.8 is shown an example of a recorded character (a *six*) with stressed the points where the local velocity of the writing was higher and where it was lower. This higher and lower values of the local velocity are reflected into higher and lower value of the function.

Hence equation 4.3.2 for the summation changes consequently:

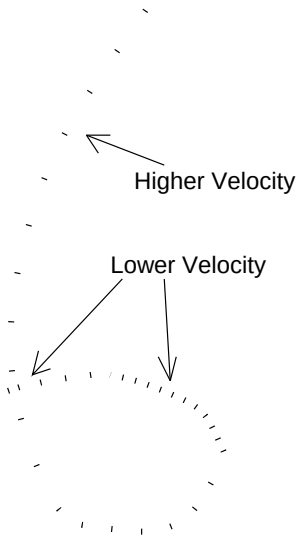

Figure 3.8: The recorded character has a different density of points depending on the local velocity. Where the points are more dense it means a lower velocity in the writing, the opposite where the points are more sparse

$$\mathcal{N}(C_j) = \sum_{i=1}^{NumOfPoints} \mathcal{F}(C_j, P_i, S_i, D_i) \tag{4.3.11}$$

While the normalization formula 4.3.3 is calculated in a different way, first we calculate a normalized logarithmic value considering the maximum (MAX) and the minimum (MIN) value among all $\mathcal{N}(Center_j)$:

$$\mathcal{L}(j) = \frac{\log \mathcal{N}(C_j) - \log \text{MIN}}{\log \text{MAX} - \log \text{MIN}} \tag{4.3.12}$$

Thus using this equation the normalization formula for the activation (firing time) of every neuron follows:

$$\forall \text{ Neuron } j : Act(j) = \begin{cases} -1 & \text{if } \log \mathcal{N}(C_j) = \log \text{MIN} \\ \text{INRG} - (\text{INRG} \cdot \mathcal{L}(j)) & \text{otherwise} \end{cases} \tag{4.3.13}$$

Where INRG is some pre-set upper limit for the input-range of the activation, thus the new values range for the output file is: $[0 \ldots \text{INRG}] \cup [-1]$.

4.4 Spike Processing

The latter kind of processing is performed in order to yield a suitable input for a Network of Spiking Neurons, which is supposed to improve performance of the classification system.

Hence the aim is to extract a temporal pattern from the recorded character. In order to obtain this, the matrix in section 2 containing characters' values has been modified to get the total delay (time) since the beginning of the drawing instead of the delay between every pair of successive recorded points. Here is the new matrix:

$$\left\langle \begin{array}{ccc} XCoordinate_1 & YCoordinate_1 & DELAY_1 \\ XCoordinate_2 & YCoordinate_2 & DELAY_1 + DELAY_2 \\ \vdots & \vdots & \vdots \\ XCoordinate_{NoP} & YCoordinate_{NoP} & \sum_{i=1}^{NoP} DELAY_i \end{array} \right\rangle$$

where NoP is the total number of points recorded for that character.

Thus, now the third column of the matrix represent the total (not partial) delay. Now using such a modified time, the concept of function $\mathcal{F}$ and the way its results are treated compared to the one used in the previous sections, changes drastically. The function changes to take into account time and distance.

Thus we have a modulation of the $\mathcal{F}$ as in 4.3.1, but this modulation is based on a temporal function:

$$\mathcal{F}(C_j, P_i, P_i^{\text{time}}, \text{TIME}) = \textit{timefun}(P_i^{\text{time}}, \text{TIME}) \cdot e^{-\frac{-(distance(C_j, P_i))^2}{\sigma^2}} \tag{4.3.14}$$

Where P_i^{time} represents the time-delay (total as above-mentioned) of the *i-th* point of the character, TIME is a time-counter that represents the current time. While *timefun* is a temporal function that is a representation of the temporal processing used in [2]. They use this for a Spiking Neurons Network while here it is used for a processing phase. The points of the character that are passed through the time function are only values for whom $P_i^{\text{time}} < \text{TIME}$.

$$\textit{timefun}(\text{CharT}, \text{CurT}) = \frac{(\text{CurT} - \text{CharT}) \cdot e^{(1 - \frac{\text{CurT}-\text{CharT}}{\tau})}}{\tau} \tag{4.3.15}$$

Where CurT is TIME and CharT is P_i^{time}.

The biggest change in the processing compared to previous processing is in the next step. In fact using the same methodology as in [2], we find out the time, starting from 0 at which the function $\mathcal{F}$ in 4.3.14 crosses a given threshold (actually fixed in e^{-450}). The system proceeds in small time-steps computing the function 4.3.14 (actual time-step is set to 10ms) until it crosses the threshold. That time-step is the final output, thus now there is not any summation to retrieve the actual value for every neuron:

$$\mathcal{N}(C_j) = \min \left\{ \text{TIME} : \mathcal{F}(C_j, P_i, P_i^{\text{time}}, \text{TIME}) > \text{THRESHOLD} \right\} \tag{4.3.16}$$

In the next Figure 3.9 an example is reported to show how the Spike Processing encode the information.

The next step, the normalization, is implemented using the same idea as in 4.3.13, but there are slight differences:

$$\forall \text{ Neuron } j : Act(j) = \begin{cases} -1 & \text{if } \mathcal{N}(C_j) = \text{MAX} \\ \text{INRG} \cdot \frac{\mathcal{N}(C_j) - \text{MIN}}{\text{MAX} - \text{MIN}} & \text{otherwise} \end{cases} \tag{4.3.17}$$

Where MIN and MAX are the minimum and maximum values respectively among all $\mathcal{N}(C_j)$ and INRG is the prefixed upper limit for the input-range of the activation.

Another solution, to avoid problems due to high values that are not taken enough in consideration by the SNN, is to shear the activation values in two parts using a mathematical modulo operation:

$$Act(j) = Act(j) \bmod (\text{INRG}/2) \tag{4.3.18}$$

This possible variation changes the range of the output file to $[0 \ldots \frac{\text{INRG}}{2}] \cup [-1]$, and can add new information to the temporal pattern encoding.

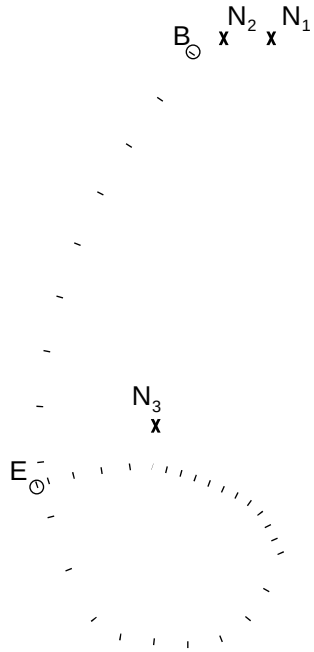

Figure 3.9: The point marked with B represents the beginning of the writing, while the point marked with E is the last point of the writing. The crosses marked as N_1, N_2, N_3 represent three centers of three different receptive fields. In the example the Neuron marked as N_2 will be a Neuron with a very low value of the function because it is very close to the beginning of the writing. Thus the function 4.3.14 will cross the threshold earlier the for Neuron N_3. This latter Neuron on the opposite will be on of the neuron with the higher value (i.e. time) because it is very close to the end of the writing.

5. Lower Level Classification

The classification of characters, as a mixture of experts, consists in a two-level classification stage. The lower level give multiple classifications to each considered character (one classification from every different classifier). The higher level performs a final classification using the lower level classifications as an input.

Multiple choices were avalaible for the role of basis low-level experts. The different classifiers used for this stage are:

- K-Nearest Neighbour (KNN)
- Classical Error Backpropagation (BP)

After some tests we decided to focus attention on the first type because Classical BP did not look like improving performance compared to KNN algorithm. A crossvalidation, using the K-Nearest Neighbour algorithm, was performed to estimate the variance of the results. As written in chapter 5, the other kind of classifier used is a Spiking Neurons Network (SNN) with Error Backpropagation.

The results of the processing stage are in format of a text-file with every character represented by a line in the file. Every line is composed by a character that is the actual classification of itself followed by the values of each neuron, whose number depend on the kind of processing considered.

5.1 K-Nearest Neighbour

The first step of this stage is to normalize the input values in the range $0 \ldots 1$, to achieve this the values are recomputed with the subsequent semi-linear function:

$$Act(i,j) = \begin{cases} 0 & \text{if } Act(i,j) = -1 \\ \left(-Act(i,j) \cdot \frac{0.6}{9}\right) + 1 & \text{otherwise} \end{cases} \tag{5.3.1}$$

Where $Act(i,j)$ represents the *j-th* activation (i.e. Neuron input value) of the *i-th* character (i.e. line of the file) among all recorded characters. The value 9 in the equation arises from the main experiments performed in the following chapters over a set of digits in the range $[1 \ldots 9]$, while 0.6 is the result of testing (see Chapter 4 Section 2.1). With these values normalized we can think to our matrix as a set of vectors (i.e. every line of the file) in a $[0 \ldots 1]^N$ space, where N is the number of input values for every character (i.e. different receptive fields) (e.g. 169, $169 \cdot 4$). In such a space it is possible to compute a metrical distance between points, the chosen distance is a pairwise difference computation:

$$dist(i,j) = \sum_{k=1}^{N} (Act(i,k) - Act(j,k))^2 \tag{5.3.2}$$

Thus is possible to look at *dist* as to a matrix. This matrix is normalized again in order to have a *norm* of 1 for every line of the matrix.

$$dist(i,j) := \frac{dist(i,j)}{\sqrt{\sum_{k=1}^{N} (dist(i,k))^2}} \tag{5.3.3}$$

This distance between characters is then used to find out the closer *neighbours* from every character.

Every row of the matrix represents a character and to give a classification to every character the K lowest values in the row (i.e. closest characters) are considered. The K column indices, which represent the *closest* characters, are considered. A poll is made among this K characters (identified by column indices) whose classification is known.

The most voted classification among the K characters is assigned as a final classification to the considered character.

5.2 Classical Error-Backpropagation

Another system that has been taken into consideration is classical Error Backpropagation (BP). The algorithm used is the BP implemented into *Matlab* program [33] with the Neural Network tool-box. The parameters were set as follows:

The network used is a feed-forward backpropagation network, the training function is *trainrp* (i.e. updating weights according to resilient backpropagation algorithm (see *Matlab* help), in general the network is used with different numbers of hidden layers and different numbers of units in every hidden layer. See chapter 4 for details. The number of epochs is fixed, unless differently specified, to 5000.

Backpropagation was abandoned as an independent classifier after testing revealed that its performance is slightly worse than K-Nearest Neighbour algorithm (see again chapter 4). It is used as a top level classifier (see section 6) to give a final decision but only among results of other previous systems.

5.3 Crossvalidation

The aim of this stage is to obtain a variance estimation of the results. This particular stage is not strictly subsequent to the K-Nearest Neighbour algorithm classification, because it is an extension of such algorithm.

In fact this part uses as a classifier the above-mentioned KNN, but is important to notice that the idea of crossvalidation is not linked in any way with the KNN. Thus it can be used to give an estimation of the variance of every kind of data set, as we show in section 6.

The input data (i.e. processing stage output) is divided into a number of different sets (actually we used 4 or 5 different sets), say n. Each set contains almost the same number of examples from every class, apart from this the choice of the examples at the beginning is made randomly.

Thus now we have n sets.

Definition:

- Test Set is the considered set (at the actual turn)
- Known Set the set of all the remaining examples (i.e. the union of all the other sets)

Repeating the next step n times, one for every set, we perform a KNN-algorithm-based classification on each set in the following way.

The system considers one by one all the examples in the Test Set and gives the classification of this example considering the K-Nearest Neighbours to this example among the Known Set. Thus at the end we have a classification for all the characters in this Test Set.

Repeating this step n times we have classification for all the characters. We have also a multiple performance measures, because we have n performance values over the different Test Sets, thus we accomplished a crossvalidation over the unique original set.

The crossvalidation program writes also a file that contains a so called *confusion matrix*. The confusion matrix is a representation of the classification performed by the classifier. The rows represent the actual classification (e.g. first row with index 1 represents all characters that are really ones). The columns represent the classification given by the classifier. Thus every cell (i, j) of the matrix contains the number of character that are actually *i's* and are recognized as *j's* by the classifier.

Follows that a perfect classification results in a diagonal matrix (i.e. values different from zero only on the main diagonal).

This matrix is also useful to understand what type of errors are more frequent by every classifier and every different processing. Which characters are "confused" with each other more frequently.

6. Higher Level Classification

The different lower level classifications are combined in this phase to be input of a *final* top level classifier.

The top level classifier used is the above-mentioned classic Error Backpropagation (5.2). In this case the number of inputs is the number of different classifiers used in the lower level (actually we used 4 or 5 different classifiers). The format of the input is now a matrix with every row representing a different character, and with every column representing a classification given by a lower level classifier to that character, plus a symbol for the actual classification (see Table 3.1).

C_1	C_2	C_3	C_4	A
1	1	1	4	1
2	7	2	7	2
8	3	3	3	3
4	4	4	4	4
⋮	⋮	⋮	⋮	⋮

Table 3.1: Example of an input to the higher level classifier (i.e. Error Backpropagation). Columns C_1 to C_4 represent inputs to the Neural Network, A is the desired output

Since we have only few inputs the Backpropagation used was a very simple (thus fast) one, without

any hidden layer. The low number of inputs and the lack of hidden units allow the system to be train-able in a very short time.

The output of this network is the final classification attributed by the system to each character.

Figure 3.10 shows the stage of hierarchical multilevel classification.

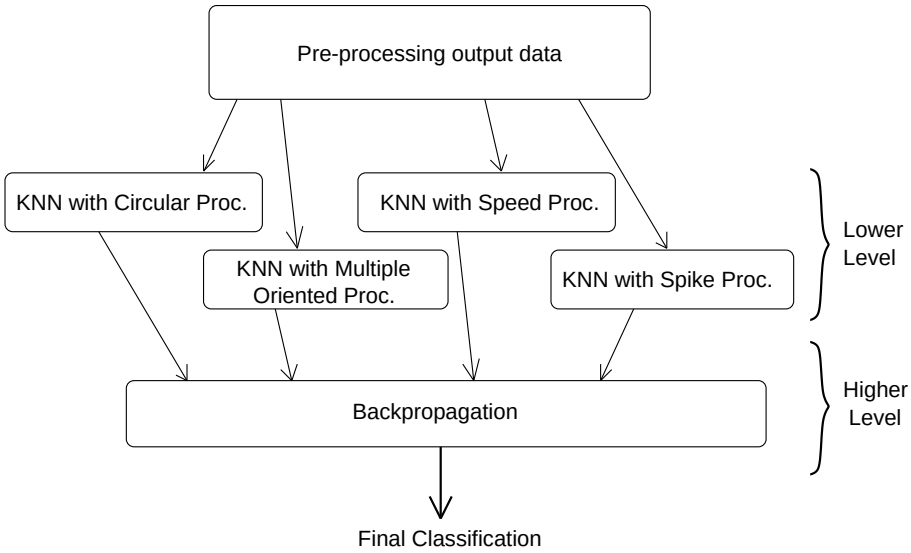

Figure 3.10: The figure shows how the hierachical classification works. In the lower level Classifiers use output data from Pre-processing to perform classification. The higher level (actually Backpropagation) takes the lower level classifications as input an yield as output the final classification.

To get these results at this level, with the Backpropagation algorithm we performed the above-mentioned crossvalidation. The use of crossvalidation gives also, as in the previous level, an estimation of the variance of the results. The differences with the above described are that:

- The Known Sets become the Training Sets for the Backpropagation Network
- The Test Sets are used as the Test Sets for the Backpropagation Network

Thus we still get multiple performance measures, over the different test sets. And in this way the system is able to test all the characters in the original set, without the need of a separate and too large training set.

It is very important to notice that in this research no *rejection rate* was used. Thus all the characters receive a classification. No possibility of doubting is left to the classifiers. In fact with a rejection rate, the classifiers when identify more than one class close (i.e. under a fixed threshold) to the one tested, decide to reject the character. This behaviour is fully understandable, and it is due to the fact that quite often it is less expensive to reject a character and either leave an human to check it or ask the user to rewrite it, than to accept a wrong classification.

This possibility is left for future development, for a further increase of the performance.

Chapter 4
Experimental Results

1. Overview

In this chapter some experimental results are presented. Section 2 presents preliminary results for the design of the architecture and then results on each single classifier. The following section 3, presents results of testing over the combined hierarchical classifier, and compares this with different single classifiers.

2. Single Classifier

In the first stage tests were performed on the single classifiers (i.e. using only one hierarchical level). To evaluate the performance of the system 304 digits were recorded. The recorded numeral are in the range $[1 \ldots 9]$[1].

2.1 Choice of parameters in KNN and semi-linear function

Some tests were performed to decide the K parameter in the K-Nearest Neighbours algorithm, and the parameter in the equation 5.3.1 of normalization of the input data (in the text this parameter is already fixed to 0.6).

The best parameter K depends on the size of the considered set. For my experiments, which have 33 or 34 examples per class, we found *5* as the best parameter (see Table 4.1). This is due also to the fact that during crossvalidation the actual number of examples compared to every single test example is not the total number of remaining examples. In fact the actual number is around $\frac{3}{4}$ of the total examples (if the cross validation is made over 4 examples) as explained in previous chapter (see section 5.3).

The choice of the parameter of the function 5.3.1 (in chapter 3) was made from a set of 6 different values ranging from 0.3 to 0.8. This parameter is a weight of the difference between a non firing time (i.e. an input value of -1) and the latest firing time (i.e. an input equal to the maximal input range). It gives also a weight of the differences among firing times (due to the nature of the equation this weight is complementary to the first).

Figure 4.1 shows the results of tests varying this parameter among different values of K.

The figure shows that the higher performances are yield by parameter set to *0.6*. The reason why

[1] 0 has been excluded because in the subsequent stage with the addition of letters it would too difficult (at least at this phase of developing) to distinguish among 0 (number zero) and *o* (letter)

crossvalidation		
K	# of digits recognized	Percentage
3	296	97.4%
5	296	97.4%
7	295	97.0%
9	294	96.7%
11	291	95.7%
13	290	95.4%

Table 4.1: Average results of crossvalidation on 304 digits set (divided into 4 sets) to find out the best K value for K-Nearest Neighbour algorithm. Between 3 and 5, the latter value was chosen due to the possibility of increasing the number of compared examples (see last experiments at the end of this chapter).

the higher values are obtained with a value of K set to 3 is that to have faster results this tests were performed over a smaller set, therefore performance decreases for higher values of K.

2.2 KNN, BP and KNN vs. BP experiments

The first tests were made on K-Nearest Neighbour algorithm varying the type of processing as written in the previous chapter (see section 5) The original set was divided into 4 different sets (each one with around 76 examples) and crossvalidation was performed.

Results are shown in Table 4.2.

KNN algorithm over 304 digits		
Processing Type	Performance	Standard Deviation
Circular Processing	96.8%	$\pm$2.4
Multiple Oriented Processing	99.2%	$\pm$0.7
Speed Processing	97.0%	$\pm$1.5
Spike Processing	95.1%	$\pm$2.2

Table 4.2: First results on K-Nearest Neighbour algorithm over different types of processing

As also noted in Chapter 3 Section 4, results on spike processing and circular processing are slightly worse also for a logical reason. The number of inputs in these two types is limited to *169*, while the other two types have a number of inputs 4 times larger. This results in a faster processing for the first and last type, even if performance is slightly worse.

Following these results, error Backpropagation was tested to verify the performance on the same set of digits.

As written in a previous section, Backpropagation was tested using *mat-lab* algorithm. The tests were performed varying different parameters, except the number of layers that was always set to 4 (i.e. 2 hidden, input and output layer). In every tested case KNN performed better than BP (see Table 4.3). The parameters used for the Backpropagation were the results over other tests on a restricted number of examples to find out the better parameters.

Given these results, we chose to use as a first level classifier the faster (for a low number of examples) and more reliable classifier: K-Nearest Neighbour. While keeping Backpropagation as a possibility for the higher level, due also to the fact that this algorithm on a low number of inputs can be very fast and reliable. And in the higher level these two characteristics are very significant.

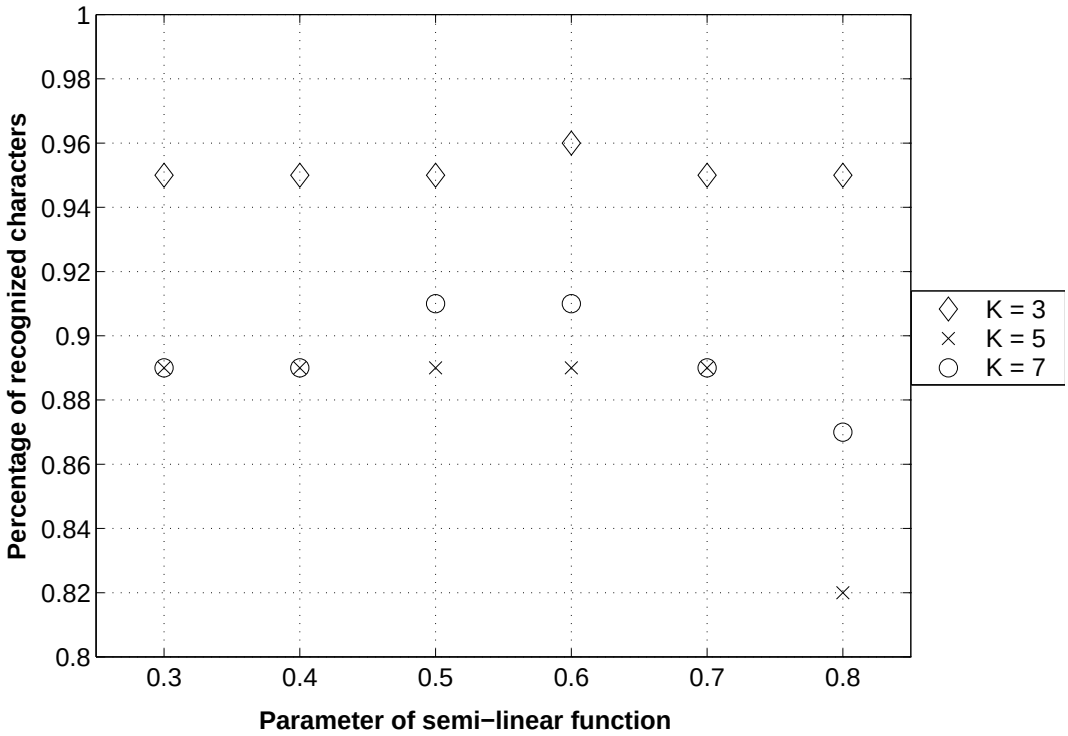

Figure 4.1: A plot of the experiments to choose the parameter for the semi-linear normalization function. On the X axis are the different values tested. On the Y axis the performance of the relatives values with different values of K. The were made on a small set of digits, thus the better performance are obtained with a small value of K.

2.3 Experiments on restricted sets

The following step was to test KNN on different sets to see how performance changes. First we tried on a very small set.

Reasons to experiment with a small number of example arises from the fact that these characters have to be recorded by an user. Thus a small number of examples avoids bothering the user with a long recording time. Another reason is the speed of processing and classification, which decreases for the smaller number of examples.

Extracting (with an implemented software) n examples from each class other tests were performed with KNN. In this situation when the number of examples per class was too small the parameter K was modified, otherwise performance would naturally drop.

In the subsequent Table 4.4 a test over 36 (i.e. *9* classes with *4* digits each) examples is reported.

From these results one can see that performances, with a very low number of examples, decreases in every kind of processing used.

A main reason, for this decrease, is that in the way crossvalidation works (see previous chapter), the actual size of the sets on which the recognition is made (considering a division in 4 sets), is 27 digits ($= 36 \cdot \frac{3}{4}$). That is: the tested digit is compared to only 3 others digits from each class.

To test the system more intensively other characters were recorded, now including alphabetical characters. Letters from a to f were recorded, for a total of 102 (17 for each class).

The same test as in the previous experiment was performed. In this test the number of examples taken from each classes is bigger: 17 examples.

Backpropagation vs K-Nearest Neighbour				
# of Training Epochs	# of Hidden Units per Layer	Performance	KNN Performance	Difference
1000	35–35	96.4%	96.8%	−0.4%
1000	75–35	96.7%	96.8%	−0.1%
1500	75–35	94.7%	96.8%	−2.1%

Table 4.3: For KNN the comparison is made only with Circular Processing. Thus differences are even more relevant

KNN algorithm over 36 randomly chosen digits			
Processing Type	Performance	Same Processing on 304 digits	Difference
Circular Processing	72.2%	96.8%	−24.6%
Multiple Oriented Processing	91.7%	99.2%	−7.5%
Velocity Processing	83.3%	97.0%	−13.7%
Spike Processing	61.1%	95.1%	−34.1%

Table 4.4: Results on a restricted number of examples (4) from 9 classes

With this set we also tried another experiment. A mixture of two types of processing: speed and multiple oriented. This result was achieved simply copying the two processing outputs one after the other. The number of inputs for the crossvalidation was doubled (*169·4·2=1352*). But the evidence of the results says that the ratio between the computational complexity and the performance is not high enough to justify the use of such processing.

The results are as follows from Table 4.5.

KNN algorithm over 255 randomly chosen characters (digits and alphabetical)					
Processing Type	Performance	Same Processing on 304 digits	Difference	Same Processing on 36 digits	Difference
Circular Processing	91.0%	96.8%	+5.8%	72.2%	−18.8%
Multiple Oriented Processing	96.2%	99.2%	+3.0%	91.7%	−4.5%
Velocity Processing	93.7%	97.0%	+3.3%	83.3%	−10.4%
Spike Processing	90.6%	95.1%	+4.5%	61.1%	−29.5%
Velocity + Multiple Oriented	97.1%	N.A.	N.A.	N.A.	N.A.

Table 4.5: Results on a restricted number (15) of examples from 17 different classes

These last results tell that Spike Processing decreases less in performance than Circular Processing when working on a more difficult task, but having more examples available for comparison. In general the conclusion is that all kind of processing are working better on a larger number of examples than on a smaller, but Spike Processing has the lowest decrease when the number of examples decreases.. Multiple Oriented Processing is quite stable (maximal gap between performances: 7.5%).

It is important to notice that the crossvalidation works as a multiple test on smaller sets (compared to the original set), until now this small sets size is equal to $1 + \left(\frac{n-1}{n}\right) \cdot S$, if S is the size of the whole set and n the number of crossvalidation sets used. The plus one is given by the example that the

classifier is actually classifying [2].

3. MULTIPLE CLASSIFIERS

The next step was to test how multiple classifiers were performing.

To have a benchmark with the previous results, the same set of 304 digits was used. There is a difference in the use of the crossvalidation. In fact the crossvalidation continues to divide the original set into n smaller sets, but on the opposite of what used until now, the algorithm chooses one of these sets as the Known Set each turn and the remaining sets are combined to form the Test Set. In this way the set of characters compared with the one tested is smaller, thus the algorithm results faster. Moreover the difficulty of the tests increases as the number of examples to compare to the tested one decreases.

Thus in this tests the size of the Known Sets is $1 + \left(\frac{1}{n}\right) \cdot S$.

The number of sets used this time is *5*. Thus every small set has a size of around 60 examples (while before was around 250).

As described in sections 5 and 6 in the previous chapter, a classification has been performed using the 4 main types of processing above described (i.e. Circular, Multiple Oriented, Speed and Spike). Thus the classifications given by the crossvalidation algorithm are used as input to an Error Backpropagation algorithm, that gives the final classification.

In the following Table 4.6 first are reported results of the crossvalidation for each single classifier with different processing then the *multiple level hierarchical classifier* results.

Single Classifiers Vs. Hierarchical Classifier		
Classifier Type	Performance	Standard Deviation
Circular Proc.	92.9%	±4.2%
Oriented Proc.	91.4%	±2.2%
Velocity Proc.	90.9%	±5.6%
Spike Proc.	94.7%	±3.3%
Hierarchical Class.	95.9%	±1.5%
Average Diff.	+3.2%	—

Table 4.6: Results of the hierarchical classifier compared to each single classifier used. Four KNN classifiers are used in the lower level, one BP classifier in the top level. The set is the original one with 304 digits (9 classes). Results on single classifiers differs from the ones in Table 4.2 because crossvalidation was implemented in a different manner, in order to operate on smaller sets, thus faster, even if with a decrease in the performance.

The interpretation of this result is that, with the multiple classifiers performance improves considerably. Another important result is the performance with the Spike Processing that with a very small Known Set improves radically compared to results of the other classifiers (i.e. it maintains the same performance as with larger Known set).

In this case, considering that the crossvalidation is changed, the actual size of the sets on which the recognition is made is around 60 (around 6 examples for each class). Comparing to example in Table 4.4, this is a double number of example, but still a small number, and results are definitely better than in the example with only 36 total characters.

[2] This means that if the original set has for instance 304 characters, crossvalidation divides this into n smaller sets, each of size $\frac{304}{n}$, then uses *n-1* of these sets as the Known Set and the remaining set as the Test Set (see previous chapter for further explanations)

3.1 Decreasing number of classifiers used

Finally some experiments were tested to see what happened to performance of the hierarchical classifier if one of the classifier in the lower level was not used as an input to the higher level. Following Tables (4.7,4.8,4.9 and 4.10) compare performances of the hierarchical classifier with and without one of the KNN algorithm classifiers.

Hierarchical Classifier w/o Velocity Processing Classifier			
Classifier Type	Performance	Standard Deviation	Diff.
Circular Proc.	92.9%	±4.2%	
Oriented Proc.	91.4%	±2.2%	
Spike Proc.	94.7%	±3.3%	
Hierarc. Class. w/o	95.3%	±2.5%	—
Hierarc. Class w/	95.9%	±1.5%	+0.6%
Velocity Proc.	90.9%	±5.6%	−4.4%

Table 4.7: Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Velocity Processing one.

Hierarchical Classifier w/o Oriented Processing Classifier			
Classifier Type	Performance	Standard Deviation	Diff.
Circular Proc.	92.9%	±4.2%	
Velocity Proc.	90.9%	±5.6%	
Spike Proc.	94.7%	±3.3%	
Hierarc. Class. w/o	96.6%	±1.4%	—
Hierarc. Class w/	95.9%	±1.5%	−0.7%
Oriented Proc.	91.4%	±2.2%	−5.2%

Table 4.8: Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Multiple Oriented Processing one.

Results from these tables let us see that in general (with the exception of the hierarchical classifier without Multiple Oriented Processing Classifier), performance with 3 classifiers is slightly worse than performance with 4 classifiers. Another important result is that, in general [3] the variance of the hierarchical classifier with only 3 lower level classifiers is bigger than the one with 4 lower level classifiers. Thus the fourth classifier improves performance in almost all the cases and gives stability to the combined hierarchical classifier.

[3] Again with the exception of the hierarchical classifier without Multiple Oriented Processing classifier, which variance is almost the same as the original hierarchical with 4 lower level classifiers

Hierarchical Classifier w/o Spike Processing Classifier			
Classifier Type	Performance	Standard Deviation	Diff.
Circular Proc.	92.9%	±4.2%	
Oriented Proc.	91.4%	±2.2%	
Velocity Proc.	90.9%	±5.6%	
Hierarc. Class. w/o	94.1%	±4.0%	—
Hierarc. Class w/	95.9%	±1.5%	+1.8%
Spike Proc.	94.7%	±3.3%	+0.6%

Table 4.9: Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Spike Processing one.

Hierarchical Classifier w/o Circular Processing Classifier			
Classifier Type	Performance	Standard Deviation	Diff.
Oriented Proc.	91.4%	±2.2%	
Velocity Proc.	90.9%	±5.6%	
Spike Proc.	94.7%	±3.3%	
Hierarc. Class. w/o	95.6%	±1.9%	—
Hierarc. Class w/	95.9%	±1.5%	+0.3%
Circular Proc.	92.9%	±4.2%	−2.7%

Table 4.10: Comparison between Hierarchical classifier with 3 lower level classifiers and the one with 4 lower level classifiers. The KNN algorithm classifier who was not used in this test was the Circular Processing one.

Chapter 5
Temporal pattern

1. Overview

Chapter 5 illustrates a way of learning temporal patterns to exploit the time component. Section 2 briefly introduces the concept of networks of spiking neurons. Section 3 presents an existing derivation of an Error-Backpropagation algorithm for networks of spiking neurons. Finally the last section shows how this classifier can be integrated to the already presented hierarchical system. Some experimental results on the single classifier and on the integration of the new classifier with the existing system are presented.

In addition to exploring the concept of combining experts for handwritten character recognition, we experimented with a classifier based on Spiking Neurons Networks, since such networks seem to be inherently suitable for learning temporal patterns, as are used in the spike processing part of our hierarchical classifier. Thus we tried to prove whether the use of this type of network for such task was increasing performance obtained in the previous chapter or not.

2. Introduction to Spiking Neurons Networks

A very different Neural Network approach that can really take into account the temporal component, in the writing of a character, is the novel spiking neurons network (SNN).

This type of network, defined in [16] has been shown to be theoretically more powerful than sigmoidal neural networks (e.g. by Maass [16, 15]).

Experimental evidence has accumulated during the last few years indicates that many biological neural systems use the timing of single action potentials (or "spikes") as a way of coding information [16]. In particular, they seem to describe better the actual output of a biological neuron, and allow investigations, on a theoretical level, of the possibilities of using time as a resource for computation and communication.

In the simplest (deterministic) model of a spiking neuron one assumes that a neuron v fires whenever its "potential" P_{v} reaches a certain threshold Θ_v. This potential P_{v} is the sum of so-called excitatory post-synaptic potentials ("EPSP") and inhibitory post-synaptic potentials ("IPSP"), which results from the firing of other neurons u that are connected through a "synapse" to neuron v. The firing of a "pre-synaptic" neuron u at time s contributes to the potential P_{v} at time t an amount that is modeled by the term $w_{u,v} \cdot \varepsilon_{u,v}(t-s)$ which consists of a "weight" $w_{u,v} \geq 0$ and a *response function* $\varepsilon_{u,v}(t-s)$, which models the impact in time of a PSP (either excitatory or inhibitory) on the potential

of the receiving neuron v.

The "weight" $w_{u,v} \geq 0$ in the term $w_{u,v} \cdot \varepsilon_{u,v}(t-s)$ reflects the "strength" of the synapse between neuron u and neuron v [16].

3. Derivation of an Error Backpropagation for Networks of Spiking Neurons

In [2] an error-backpropagation algorithm is derived for networks of spiking neurons.

In the derivation of the model they allow for each connection to have multiple delayed synapses with varying weights. This particular neuron model (see Figure 5.1) is referred to as the Multiple Delayed Spiking Neurons Network (MDSNN). Error-Backpropagation equations are derived for a feedforward network (see Figure 5.1A and 5.1B).

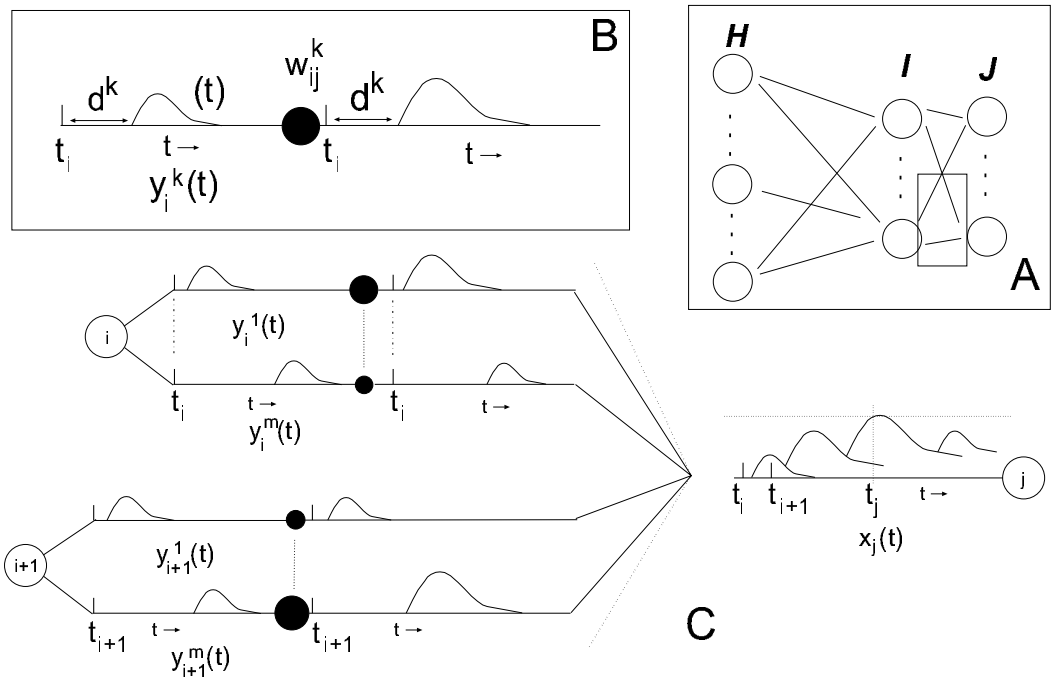

Figure 5.1: Network configuration and connection layout. A) A multi-layer feedforward network. In B) a single synaptic connection: the delayed pre-synaptic potential is multiplied by the synaptic efficacy to obtain the post-synaptic potential. C) Two small multi-synapse connections: for every connection, the post-synaptic potentials from all delayed synaptic terminals are summed to obtain the potential x_j. A spike at t_j is generated in neuron j when the input x_j crosses threshold Θ.

It is demonstrated that networks of spiking neurons (or "integrate-and-fire neurons") can perform complex non-linear classification in temporal code just as well as sigmoidal networks in firing rate code.

Given the ability to learn inherently temporal patterns, it seems very suitable to apply SNN to a task like on-line character recognition where temporal pattern is a main feature. Also in [24, 3] it is shown that an unsupervised model of SNN computing radial-basis functions (RBFs), allows the recognition (or clustering) of temporal encoded sequences even if they are distorted in various ways.

4. Experiments

In order to try to improve performance of the hierarchical classifier system presented in the previous chapter, we tried some experiments. The experiments mainly consisted in modifying the hierarchy

previously presented (see Figure 3.10). There were two main approaches:

- Adding the new classifier in the lower level as a new input for the higher level.
- Substitution of one of the lower level classifiers with the presented SNN-based Error Backpropagation.

Both experiments were tested on the same set of 304 digits (9 classes) as the experiments represented by Table 4.6. In this way the comparison was straightforward.

4.1 Experiment on Spiking Neurons Network only
Before the above-mentioned experiments, in Table 5.1 results of SNN Error Backpropagation are reported. The set is still the 304 digits set divided into the same 5 different smaller set for crossvalidation. The input data for this neural network was the original data after Spike Processing. The reason for this choice is that Spike Processing (as noted in chapter 3 section 4.4) was used in order to yield a suitable input for the above-mentioned network.

SNN Error Backpropagation over 304 digits		
Set #	Performance	Standard Deviation
1	73.6%	±32.7%
2	86.7%	±17.8%
3	89.6%	±12.5%
4	79.3%	±23.1%
5	90.1%	±13.7%
Total	83.8%	±7.3%

Table 5.1: Results of Spiking Neurons Network Error Backpropagation over the 304 digits set

From Table 5.1 we see that Spiking Neurons Network Error Backpropagation was hard to train, because it takes long time and the variance of the result is very high.

As noted in chapter 3 section 5, from the experiment we can obtain a confusion matrix to show how many characters from each class were recognized as characters from another class. Considering that there are 5 sets for the crossvalidation, we reported in the following Tables (5.2,5.3,5.4,5.5,5.6) the 5 different confusion matrices. The results stress that SNN Error Backpropagation has a high variance, results are varying in a range of 16% percentage.

Another important feature that comes out from these confusion matrices is that SNN Error Backpropagation is working very well on the recognition of some particular digits (e.g. *ones, fives, sixes, sevens, nines*), and this peculiarity can be exploited by a higher level classifier to improve performance when other classifiers fail to recognize or yield an uncertain classification for these digits.

4.2 Experiment with the addition of the SNN classifier to the hierarchy
The first experiment was with the addition of the new classifier as a *fifth* input to the classical Backpropagation in the higher level. Results are shown in Table 5.7[1].

Performance is not improving as expected, but considering the low performance of Spiking Neurons Network Error Backpropagation[2], and its instability results are quite good.

To show that with some tests a hierarchical classifier which exploit SNN Error Backpropagation can give very good results, we report some other results over individual sets in Tables 5.8, 5.9.

[1]Results on a crossvalidation over 4 sets out of 5. This is due to the fact the results of the SNN Error Backpropagation on set number 4 were corrupted and there was not enough time to run the network again.

[2]This low performance is due to lack of time, to the length (in time) of the SNN Error Backpropagation, and to some problems to give stability to the algorithm.

Confusion Matrix Set #1									
	1	2	3	4	5	6	7	8	9
1	28	0	0	0	0	0	0	0	0
2	0	9	4	0	4	11	0	0	0
3	0	1	4	0	2	21	0	0	0
4	0	0	0	22	0	5	0	0	1
5	0	0	0	0	28	0	0	0	0
6	0	0	0	2	0	26	0	0	0
7	0	0	0	0	2	0	25	0	0
8	0	5	0	0	2	5	1	14	0
9	0	0	0	0	0	0	0	0	28
Total				184		250		73.6%	

Table 5.2: The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier. For example: the *4* in the cell indexed (2,3) means that there were 4 characters actually *twos* recognized as *threes*.

Confusion Matrix Set #2									
	1	2	3	4	5	6	7	8	9
1	26	0	0	0	1	0	0	0	0
2	0	21	0	0	5	0	0	1	0
3	0	0	25	0	0	0	2	0	0
4	0	0	0	25	0	1	0	0	1
5	1	0	0	0	26	0	0	0	0
6	2	0	0	0	1	24	0	0	0
7	0	0	0	0	0	0	26	0	0
8	1	0	0	0	13	0	0	11	1
9	2	0	0	0	0	0	0	0	25
Total				209		241		86.7%	

Table 5.3: The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier.

Results in Tables 5.8, 5.9 show that on those sets where SNN Error Backpropagation worked quite good, performance of the hierarchical classifier is improving considerably, in other cases unfortunately SNN gave problems and results were not reliable. Results show that hierarchical classifier with the addition of SNN Error Backpropagation can reach very high peaks of performance (e.g. 98.8% with a very low variance on some sets). The high fluctuation of the results yielded by SNN Error Backpropagation does not allow to have a good average results. Moreover this fluctuation in data results in a higher variance with respect to the hierarchical classifier without SNN Error Backpropagation (e.g. $\pm 2.7\%$ against $\pm 1.5\%$). The main reason for this lies in the instable learning in the Spiking Neurons Network Error Backpropagation.

4.3 Experiments with the substitution of a classifier in the original hierarchy

The second part of the experiments was to test the performance of the hierarchical mixture of classifiers, while changing one of the KNN classifiers with the new SNN Error Backpropagation classifier. The experiment excluded the worst among the KNN classifiers, based on the performance with the

Confusion Matrix Set #3									
	1	2	3	4	5	6	7	8	9
1	26	0	0	1	0	0	0	0	0
2	0	23	0	0	0	0	0	4	0
3	0	0	26	0	0	1	0	0	0
4	0	0	0	27	0	0	0	0	0
5	0	0	0	0	27	0	0	0	0
6	0	0	0	2	0	24	0	1	0
7	0	2	0	0	0	0	24	0	0
8	0	0	0	3	0	0	0	23	0
9	0	2	0	0	0	1	0	8	16
Total				216		241		89.6%	

Table 5.4: The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier.

Confusion Matrix Set #4									
	1	2	3	4	5	6	7	8	9
1	16	0	0	11	0	0	0	0	0
2	0	7	15	0	0	0	1	4	0
3	0	2	25	0	0	0	0	0	0
4	0	0	0	25	0	2	0	0	0
5	0	1	0	0	25	0	0	1	0
6	0	0	0	4	1	21	0	1	0
7	0	0	0	0	2	0	24	0	0
8	0	0	0	4	0	0	0	22	0
9	0	0	0	1	0	0	0	0	26
Total				191		241		79.3%	

Table 5.5: The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier.

same set of data (see Table 4.6). The chosen classifier was the KNN based on Velocity Processing. Results are presented in Table 5.10.

Comparing the performance yielded by the hierarchical classifier with 5 inputs and hierarchical classifier with 4 inputs (but all KNN algorithms) it is obvious that it is more useful to add SNN Error Backpropagation as an additional classifier than substituting this latter one with one of the KNN algorithms. Furthermore computational costs do not increase considerably with the add of only one more classifier results (lower level results as higher level input).

Confusion Matrix Set #5									
	1	2	3	4	5	6	7	8	9
1	26	0	0	1	0	0	0	0	0
2	0	26	1	0	0	0	0	0	0
3	0	0	27	0	0	0	0	0	0
4	0	0	0	22	0	5	0	0	0
5	0	0	0	0	27	0	0	0	0
6	0	0	0	1	0	26	0	0	0
7	0	0	1	0	0	0	22	0	4
8	0	0	0	1	1	8	1	16	0
9	0	0	0	0	0	0	0	0	27
Total				219		243		90.1%	

Table 5.6: The rows represent the actual classification of the characters in this set, the columns the classification yielded by the SNN classifier.

Single Classifiers Vs. Hierarchical Classifier		
Classifier Type	Performance	Standard Deviation
Circular Proc.	92.9%	±4.2%
Oriented Proc.	91.4%	±2.2%
Velocity Proc.	90.9%	±5.6%
Spike Proc.	94.7%	±3.3%
SNN Error BP	83.8%	±7.3%
Hierarchical Class.	95.4%	±2.7%
Average Diff.	4.7%	—

Table 5.7: Results of the hierarchical classifier compared to each single classifier used. Four KNN classifiers and the SNN Backpropagation classifier are used in the lower level, one BP classifier in the top level. The set is the original one with 304 digits (9 classes).

Hierarchical Classifier on Set # 2 with and without SNN		
	Performance	Standard Deviation
Set 2 with SNN	98.8%	±0.4%
Set 2 without SNN	93.8%	±0.1%

Table 5.8: Comparison of performance, on a single set (Set number 2) from crossvalidation, between hierarchical classifier with and without SNN Error Backpropagation in the lower level

Hierarchical Classifier on Set # 3 with and without SNN		
	Performance	Standard Deviation
Set 3 with SNN	96.3%	±0.3%
Set 3 without SNN	95.7%	±0.7%

Table 5.9: Comparison of performance, on a single set (Set number 3) from crossvalidation, between hierarchical classifier with and without SNN Error Backpropagation in the lower level

Single Classifiers Vs. Hierarchical Classifiers		
Classifier Type	Performance	Standard Deviation
Circular Proc.	92.9%	±4.2%
Oriented Proc.	91.4%	±2.2%
SNN Error BP	83.8%	±7.3%
Spike Proc.	94.7%	±3.3%
Hierarchical Classifier (3 KNN and SNN)	92.6%	±3.2%
Hierarchical with 5 Classifiers	95.4%	±2.7%
Hierarchical with 4 Classifiers (all KNN)	95.9%	±1.5%

Table 5.10: Results of different hierarchical classifiers compared to each single classifier used. Three KNN classifiers and the SNN Backpropagation classifier (here in place of the KNN with Velocity Processing that was the one with the worse performance) are used in the lower level, one BP classifier in the top level. The set is the original one with 304 digits (9 classes). In the last two rows is made a comparison with previous results: hierarchical classifier with 5 different classifiers in lower level (see Table 5.7) and hierarchical classifier with the 4 original KNN classifiers (see Table 4.6)

Chapter 6
Conclusion

With the aim of improving the performance of handwritten characters classifiers, this work studies the possibility of a mixture of experts to classify characters. Another aim of the thesis was to exploit the temporal pattern as a positive feature in characters' writing.

First I showed the implemented system for recording, preprocessing and processing of the characters. Then these approaches and methodology were followed to classify these characters. And finally results of experiments using all different techniques were reported.

The results show that the mixture of experts, using K-Nearest Neighbour as an algorithm for the lower level and an Error Backpropagation for the higher level, is a promising technique. Results in chapter 4 show that the hierarchical classifier performance is at least 1.2% more efficient than each single classifier and more reliable (the variance is lower than in every other classifier).

The hierarchical classifier proved to be performing well even on a very small set of characters. This result is useful because the system becomes more user-oriented if the user does not have to train the network by recording hundreds of characters. Furthermore a system which performs well even on small number of examples, does not require a lot of resources to be implemented.

The fact that the lower level classifiers which work on data that has a larger size, but are small in number, allows the lower level to be fast. On the other hand the higher level uses an Error Backpropagation without any hidden layer. Thus even if it works on a large number of examples, the size of each data is small because it is just the output classification of the lower level classifiers, this fact allow the Error Backpropagation algorithm to be quite fast.

The fact that the system can quickly learn from a small number of examples written by a single user, lead us to conclude that the system is relatively *user independent.*

Comparing to other results in literature, one can see that, considering the fact that the system shown in this work does not use any kind of rejection rate for uncertain characters, results are quite good. Compared to some on-line systems [22] which are required to be fast and with a low memory usage, this system seems to perform well.

Other systems that use hierarchical classifiers [5, 28] (or variations of it in [31]) have similar or slightly better performances but always using a rejection rate. In [5] error rate is not going below 1.5% without a high rejection rate (more than 10%). On the other hand in [28] the performance is good even at low rejection rates, but the system, using a Time Delayed Neural Network and working on pixel-image of digits, is quite "heavy" and the training is made over a big number of examples

(around 120.000), while in my system the tests were performed at most over 304 examples (i.e. number of example 400 times smaller) .

In addition tests were performed to verify the performance of the hierarchical classifier when using a Spiking Neurons Network as a lower level classifier. An Error Backpropagation derived from SNN by Bohté et al. [2] was used to perform tests.

Due to lack of time and apparent instability in the algorithm, average results were not so good (i.e. were not improving as expected), but peak performance improved considerably with respect to the previous hierarchical classifier without SNN Error Backpropagation. The peak performance of the hierarchical classifier with SNN compared to the one without, is 5% better, and the performance is close to 99%. This result shows that hierarchical classifier using SNN Error Backpropagation could yield very high performance, but it requires a more careful study and testing.

Another peculiarity of SNN Error Backpropagation, that is shown by the confusion matrices presented in chapter 5, is the capability of recognizing some typical characters very accurately (actually tested on digits), that can be exploited either where other classifiers fail to recognize the same characters or other classifiers yield uncertain results.

With regard for memory footprints, the multiple hierarchical classifier has good results. Considering the case of the multiple oriented processing with $169 \cdot 4$ inputs, if the range of the input is limited to the range $[0 \dots 25] \cup [-1]$, that is 26 input values, the memory usage for a character with a definition of 150×150 pixel, is just *425 bytes*.

Another important remark is the obvious improvement that could be achieved by introducing a rejection rate, that as above-mentioned, in literature proved to be efficient and widely used. Furthermore in a "real-world" task is more convenient, and the computational cost of the introduction of such issue is almost null.

References

1. H. I. Avi-Itzhak, T. A. Diep, and H. Garland. High accuracy optical character recognition using neural networks with centroid dithering. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 17(2):218–224, 1995.
2. S. M. Bohté, J. N. Kok, and H. La Poutré. Spikeprop: Backpropagation for networks of spiking neurons. In M. Verleysen, editor, *Proceedings of the European Symposium on Artificial Neural Netwroks ESANN'2000*, pages 419–425, Bruges, 2000.
3. S. M. Bohté, J. N. Kok, and H. La Poutré. Unsupervised classification in a layered network of spiking neurons. In *Proceedings of IJCNN'2000*, page 211, 2000.
4. D. V. Buonomano and M. Merzenich. A neural network model of temporal code generation and position-invariant pattern recognition. *Neural Computation*, 11(1):103–116, 1999.
5. J. Cao, M. Ahmadi, and M. Shridhar. A hierarchical neural network architecture for handwritten numeral recognition. *Pattern Recognition*, 30(2):289–294, 1997.
6. G. Carpenter and S. Grossberg. Art2: Stable self-organization of pattern recognition codes for analof input patterns. *Appl. Opt.*, 26:4919–4930, 1987.
7. M. Chen and A. Kundu. A complement to variable duration hidden markov model in handwriting recognition. In *ICIP'94*, pages 174–178, 1994.
8. S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm fo radial basis function networks. *IEEE Trans. Neural Networks*, 2:302–309, 1991.
9. S. Fahlmann and C. Lebiere. The cascade correlation learning algorithm. Technical Report CMU-CS-90-100, Carnegie Mellon University, 1990.
10. A. Hasegawa, K. Shibata, K Itoh, Y. Ichioka, and K. Inamura. An adaptive neural network: Application to character recognition on x-ray films. *Neural Networks*, 9(1):121–127, 1996.
11. R. Hecht-Neilsen. Counterpropagation networks. *Appl. Opt.*, 26:4979–4984, 1987.
12. K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–43, 1990.
13. S. Lee and J. C. J. Pan. Unconstrained handwritten numeral recognition based on radial basis competitive and cooperative networks with spatio-temporal feature representation. *IEEE Transaction on Neural Networks*, 7(2):455–474, 1996.

14. Y. Lee. Handwritten digit recognition using k nearest-neighbor, radial-basis function, and backpropagation neural networks. *Neural Computation*, 3:440–449, 1991.

15. W. Maass. Fast sigmoidal networks via spiking networks. *Neural Computation*, 9:279–304, 1997.

16. W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

17. J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Math. Stat. and Prob.*, volume 1, 1967.

18. G. L. Martin and J. A. Pittman. Recognizing hand-printed letters and digits. In *Neural Information Processing Systems 2*, pages 405–414, 1990.

19. J. Moody and C. J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1:281–294, 1989.

20. M. C. Mozer. Neural net architectures for temporal sequence processing. In A. S. Weigend and N. A. Gershenfeld (Eds.), editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, volume XVII, pages 243–264, Redwood City, CA, 1993. Sante Fe Institute Studies in the Sciences of Complexity.

21. R. J. Nag, K. H. Wong, and F. Fallside. Script recognition using hidden markov models. In *ICASSP'86*, pages 2071–2074, 1986.

22. K. S. Nathan, H. S. M. Beigi, J. Subrahmonia, G. J. Clary, and H. Maruyama. Real-time on-line unconstrained handwriting recognition using statistical methods. In *ICASSP'95*, 1995.

23. K. S. Nathan, J. R. Bellegarda, D. Nahamoo, and E. J. Bellegarda. On-line handwriting recognition using continuos parameter hidden markov models. In *ICASSP'93*, volume 5, pages 121–124, 1993.

24. T. Natschläger and B. Ruf. Spatial and temporal pattern analysis via spiking neurons. *Network: Computation in Neural Systems*, 9(3):319–332, 1998.

25. M. Neschen and F. Nübel. Cognitus - fast and reliable recognition of handwritten forms based on vector quantisation. In *International Conference on High-Performance Computing and Networking*, Brussels, 1996.

26. E. Oja. Neural networks, principal components and subspaces. *Int. Journal of Neural Systems*, 1:61–68, 1988.

27. L. Peters, C. Laja, and A. Malaviya. A fuzzy statistical rule generation method for handwriting recognition. *Expert Systems*, 15(1), 1998.

28. M. Pfister, S. Behnke, and R. Rojas. Recognition of handwritten zip codes in a real—world non-standard-letter sorting system. *Applied Intelligence*, 12:95–115, 2000.

29. T. Poggio and F. Girosi. Networks for approximation and learning. In *Proc. IEEE*, pages 1481–1497, 1990.

30. D. L. Reilly, L. N. Cooper, and C. Erlbaum. A neural model for category learning. *Biological Cybernetics*, 45:35–41, 1982.

31. G. Seni. *Large vocabulary recognition of on-line handwritten cursive words.* PhD thesis, New York State University, 1995.

32. A. H. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE Trans. on Acoustics, Speech and Signal Processing*, 37:328–339, 1989.

33. The Math Works. Pro-matlab. the math works, inc. - version 5.3.0.10183 (r11).

34. H. Yan. Handwritten digit recognition using an optimized nearest neighbor classifier. *Pattern Recognition Letters*, 15:207–211, 1994.