# A 32-neurons Slice for High Speed Handwriting Recognition

L.M. Reyneri†, F. Gregoretti†, B. Lazzerini††, R. Passerone†
† Dipartimento di Elettronica - Politecnico di Torino
†† Dipartimento di Ingegneria della Informazione - Università di Pisa

## Abstract

This article presents PAPRICA-3, a high-speed 32-neurons slice for real-time neural processing of images. The system is a programmable parallel processor array with the instruction set tailored to the emulation of neural networks and to image processing. Dedicated hardware features allow simultaneous image acquisition, image processing and neural network emulation. The 32-neurons slice is cascadable to match the required size. The system is under fabrication and has been simulated at a clock frequency of 100 MHz (typical). Estimated performance is up to 130 MCPS for perceptrons. The system has been tested on the recognition of handwritten check amounts, by integrating neural network algorithms with context analysis techniques.

## 1 Introduction

[1] Handwriting recognition [1, 2, 4] is a major issue in a wide range of application areas, including mailing address interpretation, document analysis, signature verification [3] and, in particular, bank check processing.

Besides the classical problems encountered in reading machine-printed text, such as word and character segmentation and recognition, the domain of handwritten text recognition has to deal with other problems such as the apparent similarity of some characters with each other, the unlimited variety of writing styles and habits of different writers, and also the high variability of character shapes issued by the same writer over time. Furthermore, the relatively low quality of the text image, the unavoidable presence of background noise and various kinds of distortions (for instance, poorly written, degraded, or overlapping characters) can make the recognition process even more difficult.

The amount of computations required for a reliable recognition of handwritten text is therefore very high, and real-time constraints can only be satisfied either by using very powerful and expensive processors, or by developing ad-hoc processors tailored to the specific task.

We have decided to develop a dedicated architecture, to cope with tight cost constraints and size requirements, and to tailor the recognition algorithms to the architecture capabilities.

In this paper we present PAPRICA-3, a massively parallel image processor with the instruction set dedicated to the emulation of several types of neural networks (such as perceptrons, self-organizing maps, cellular networks). The main difference with respect to other hardware neural network implementations resides in its complete programmability, therefore the system can also accomplish other tasks such as image processing, image acquisition, image compression, symbolic computation, etc.

The proposed VLSI device has been designed to be used in a minimum-size configuration consisting of 1 chip, 1 external RAM, plus some glue logic to interface with a host processor. The device has been inserted into a complete system for high-speed recognition of the amount on banking checks, which mixes image processing algorithms, neural networks and a context analysis subsystem.

The VLSI device has been designed in parallel with the development of the software, so that the hardware and software designs have in-

[1] This work has been partially supported by the EEC MEPI initiative DIM-103 *Handwritten Character Recognition for Banking Documents.*

fluenced each other very much. That resulted into an efficient though flexible implementation for a wide class of applications.

## 2 Hardware Description

The VLSI chip of PAPRICA-3 has been designed bearing in mind the following constraints/requirements:

- cascadability: the chip implements a slice of 32 neurons (namely, all that could fit in 100 $mm^2$, but additional neurons can be cascaded with nothing but one RAM chip per each additional PAPRICA-3 chip,
- one Processing Element per each neuron, to have the highest efficiency,
- as simple as possible processors, namely 1-bit processing elements; operations with higher resolution can be computed by means of a bit-serial approach; this provided the best complexity/flexibility/performance compromise among all the architectures which were analyzed,
- provisions for both image processing, mathematical morphology [6], neural network emulations, image acquisition, etc.
- easy interface with a host processor, to improve the overall functionality of the system,
- the highest degree of programmability, although keeping the hardware complexity at a low level; this has been achieved by letting the host processor (instead of PAPRICA-3) perform a number of operations which normally occur at a lower rate,
- both "local" and "global" instructions; the former are used to implement a sort of neighborhood, while the latter are available to compute global operations, such as summations, maxima, minima, winner-takes-all, etc.
- provisions for simple handling of external look-up tables (either external RAM or ROM devices),
- a set of external "status registers" where PAPRICA-3 can accumulate neuron outputs which can be read (or written into) in parallel by the host processor,
- a set of direct binary I/Os (6+6, in the prototype) channels by means of which PAPRICA-3 can either interrupt the host or activate stepper motors, CCD cameras, etc.,

The only (undesirable) limitation which has been added is that a maximum of 256 neurons per aggregate are available (although the chip can sequentially emulate several aggregates, with minor limitations). This limitation applies only to perceptrons, as limitations for self-organizing maps are much less stringent.

### 2.1 Internal Architecture

PAPRICA-3 is the latest version of a previous hardware designed at the Politecnico di Torino. As shown in fig. 1, the kernel of PAPRICA-3 is a linear array of 32 identical 1-bit Processing Elements (**PE**s) connected to an image memory via a bidirectional 32-bit wide bus.

The image memory is organized into addressable words whose length matches that of the processor array. Each word contains data relative to one binary pixel plane (also called *layer*) of one line of an image (32 bits wide), and a single cycle is needed to load an entire line of data into the PE's internal registers.

Data can be transferred into the internal registers of each PE, processed and explicitly stored back into memory according to a RISC-like processing paradigm.

When executing a program, the correspondence between line number, pixel plane of a given image, and absolute word address is computed in hardware by means of data structures, named *Image Descriptors*, stored in the *Control Unit*. An Image Descriptor is a memory pointer consisting of three parts: a base address and two line counters which can be reset or increased by a specified amount under program control. Two additional counters can be used to modify program flow; instructions are provided to preset, increase and test them.

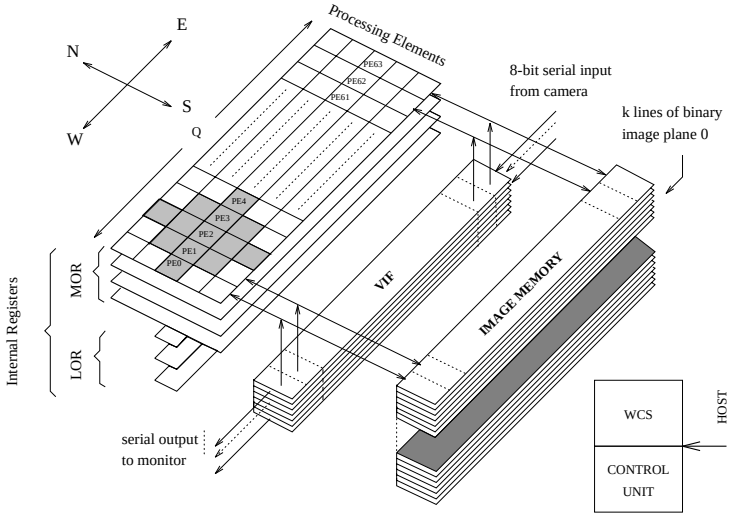


Figure 1: General architecture of the Processor Array.

## 2.2 Instruction set

Each PE is composed of a Register File and a 1-bit Execution Unit, and processes one pixel of each line. The core of the instruction set is based on *morphological operators* [6]: the result of an operation depends, for each processor, on the value of pixels in a given neighborhood ($5 \times 5$, as sketched by the grey squares in fig. 1). Data from E, EE, W and WW directions (where EE and WW denote pixels two bits apart in E and W directions, respectively) may be obtained by direct connection with neighboring PEs, while all other directions correspond to data of previous lines (N, NE, NW, NN) or of subsequent lines (S, SE, SW, SS).

To obtain the outlined neighborhood, a number of internal registers (16 per each PE, at present), called *Morphological Registers* (**MOR**), have a structure which is more complex than that of a simple memory cell, and are actually composed of five 1-bit cells with a S→N shift register connection. When a load operation from memory is performed, all data are shifted northwards by one position and the south-most position is taken by the new line from memory. In this way, data from a $5 \times 5$ neighborhood are available inside the array for each PE, at the expense of a two-line latency. A second set of registers (48 per each PE, at present), called *Logical Registers* (**LOR**), is only 1-bit wide.

The instruction set includes also *logical* and *algebraic* operations (AND, OR, NOT, EXOR, etc.), which act on either Logical Registers or the central bit of Morphological Registers. These operations can also be used to match input patterns against predefined templates, or to compute algebraic operations such as sums, differences, multiplications, etc. As PEs are 1-bit computing elements, all algebraic operations have to be computed using a bit-serial approach.

An important characteristic of the system is the integration of a serial-to-parallel I/O device, called *Video InterFace (VIF)*, which can be connected to a linear CCD array for direct image input (and optionally to a monitor, for direct image output). The interface is composed of two 8-bit, 32 stages shift registers which serially and asynchronously load/store a new line of the input/output image during the processing of the previous/next line. Two instructions activate the bidirectional transfer between the PE's internal registers and the VIF, ensuring also proper synchronization with the CCD and the monitor.

Image processing algorithms consist of sequences of low level steps, such as filters, convolutions, etc., to be performed line by line over the whole image. This means that the same block of instructions often has to be repeated many times, and instruction fetching from an external memory can lead to quite high over-

heads. Hence we chose to pre-load each block of instructions into an internal memory, named *Writable Control Store* (**WCS**, 1K words × 32 bits), and to fetch instructions from there. The performance of a fast cache with a hit ratio close to 1 can thus be obtained, at a fraction of the cost and complexity.

### 2.3 Inter-processor communication

Two inter-processor communication mechanisms are also available to exchange information among PEs which are not directly connected. The first one is a *Status Evaluation Network* to which each processor sends the 1-bit content of one of its registers and which provides a *Status Word* divided into two subfields. The first one is composed of two global flags, named *SET* and *RESET*, which are true when the contents of the specified registers are all '1's or all '0's, respectively. The second one is the *COUNT* field which is set equal to the number of processing elements in which the content of the specified register is '1'.

This inter-processor communication mechanism can be used to compute global functions such as maxima, minima (e.g., for emulation of fuzzy systems), logical OR and AND of boolean neurons, neighborhood communications, neuron summations in perceptrons, external look-up tables, winner-takes-all, seed propagation algorithms, etc.

This global information may also be accumulated and stored in an external *Status Register File* and used for further processing, or to conditionally modify program flow. Status Registers can also be read by the host processor. For instance, Status Registers have been used in the example of section 3 to implement a neural network, by computing the degree of matching between an image and a set of templates (weight and center matrices).

The second communication mechanism is an *Inter-processor Communication Network*, which allows global and multiple communications among clusters of PEs. The topology of the communication network may be varied at run-time: each PE controls a switch that enables or disables the connection with one of its adjacent processors. The PEs may thus be dynamically grouped into clusters, and each PE can broadcast a register value to the whole cluster with a single instruction. This feature can be very useful in algorithms involving *seed-propagation* techniques, in the emulation of pyramidal (hierarchical) processing and for cellular neural networks or for local communication (short range neighborhood).

A Host Interface allows the host processor to access the WCS, and a few internal configuration registers. The access is through a conventional 32-bit data bus with associated address and control lines.

Some additional control and status bits are used to exchange information with the host processor: these include a START input line and a RUNNING output line, plus other six input and six output lines called *Host Communication Channels* (**HCC**). HCC input lines can be tested during program execution to modify program flow, while HCC output lines can be used as flags to signal certain conditions to the host processor (for instance, interrupts).

The example described in section 3 uses a system composed of two PAPRICA-3 chips plus two fast static RAMs providing a 128K×32 bits Image Memory, 256 Status Registers, a host processor, and an interface with the CCD imager. The system operates with a clock frequency of up to 100MHz, providing 6.4 Gpixops/s peak throughput. The system is internally pipelined with a complex pipeline structure which provides an average performance of 3.5 Gpixops/s.

## 3 An Application Example

The aim of this work was to recognize real-world checks, where handwriting is assumed to be unboxed and usually unsegmented, so that characters in a word may touch or even overlap. Fortunately, the amount on Italian checks consists of only one word, thus no phrase segmentation is required. The amout is also written twice: the *legal amount* (namely, the literal one), and the *courtesy amount* (namely, the numerical one).

The two fields are placed in well-known areas of the check, and an approximate localization of these two areas can be obtained from the information contained in the *code-line* printed

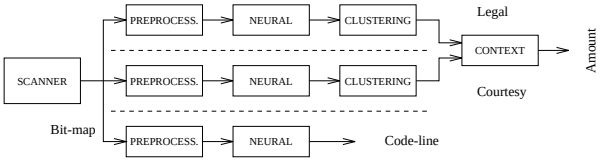


Figure 2: Block diagram of the handwriting recognizer.

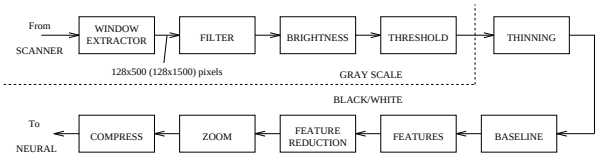


Figure 3: Block diagram of the image preprocessor.

at the bottom of the check (assuming that each bank has its own check layout, as is usually the case).

Experimental results have shown that an acceptable recognition performance for a given application can only be achieved by using contextual information (namely, redundancy) [2]. For instance, recognition errors can be detected and corrected by means of a dictionary. In the specific application, redundancy is present both in the exact correspondence that exists between the legal and the courtesy amounts, and in the very limited size of the dictionary to be used (the combinations of about 30 words, for Italian checks); also, it is interesting to note that only 15 out of the 26 letters of the English alphabet are used in such a dictionary.

The proposed system integrates five subsystems which are in cascade, as shown in fig. 2:

- a mechanical and optical *scanner*, to acquire a bit-map image of the check;
- an *image preprocessor* for preliminary filtering, scaling, and thresholding of the image;
- a *neural subsystem*, based on an ensemble of neural networks, which detect character centers and provide hypotheses of recognition for each detected character;
- a *clustering subsystem* which improves the performance of the centering detector of the *neural subsystem*;
- a *context analysis subsystem* based on a lexical and syntactic analyzer.

Legal and courtesy amounts are preprocessed and recognized independently (at the character level) and then the two streams of information are sent to the common *context analysis subsystem*, which exploits all the mutual redundancy.

In practice, also the *code-line field* located at the bottom of the checks is scanned and processed, to obtain general information about the bank and, indirectly, about check layout. Recognition of this field is easy, as it is printed on a light background and its graphical quality is usually very high. It is therefore not discussed here.

The *neural* and *clustering subsystems* carry out a pre-recognition of the individual characters, based on an *integrated segmentation and recognition* technique. In correspondence to each character of a handwritten word, the *neural subsystem* produces a list of "candidate" characters, instead of just one as in other recognizers.

The *context analysis subsystem* combines the candidate characters and, guided by the mutual redundancy present in the legal and courtesy amounts, produces hypotheses about the amount so as to correct errors made by the *neural subsystem*. The error rate of the *neural subsystem* alone is high, but this is strongly reduced by the *context analysis subsystem*. At the end, the system generates a list of possible amounts, which are sorted according to a decreasing recognition confidence.

The *image preprocessor* and the *neural subsystem* are executed on one or more PAPRICA-3 chips (two, in the prototype), while the *clustering subsystem* and the *context analysis subsystem* are executed by an external host processor (a Pentium, in the prototype), which can implement these types of algorithms more efficiently.

### 3.1 Image preprocessing

The first preprocessing subsystem is the image preprocessor shown in fig. 3, which consists of the blocks described below.

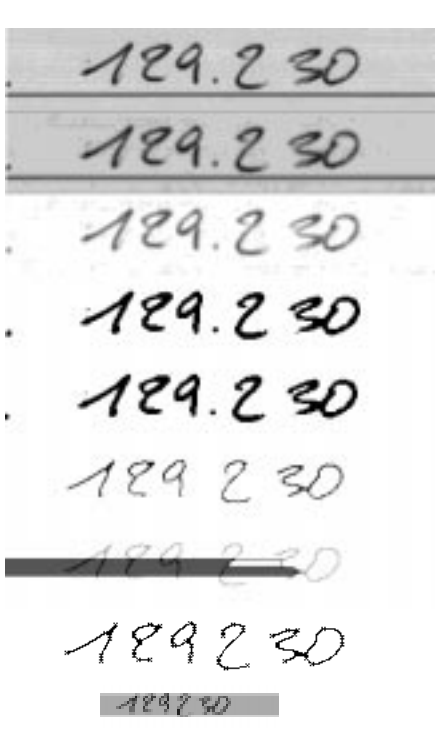

Figure 4: Preprocessing steps of handwritten images (an "easy" example): a) original image, 200 dpi, 16 gray levels; b) low-pass filtered image; c) compensated for brightness; d) thresholded image; e) spot noise removal; f) thinned, after 6 steps; g) finding baseline (at the left side of the image); h) features detection (features are tagged by small crosses); i) compressed.

1. The `WINDOW EXTRACTOR` acquires the input image from the `SCANNER`, at a resolution of approximately 200 dpi, 16 gray levels. The scanner is an 876-pixel CCD line camera scanned mechanically over the image, from right to left (due to practical reasons), at a speed of 2m/s (which is equivalent to about 700 characters/s). Image size is about $876 \times 1500$ pixels, although character recognition takes place in two smaller areas of known coordinates (see fig. 4.a).

   Image acquisition is performed by the VIF, in parallel with processing, and a whole image line is acquired in just one clock cycle.

2. The `FILTER` block computes a simple low-pass filter with a $3 \times 3$ pixel kernel (see fig. 4.b).

3. The `BRIGHTNESS` block compensates for the non-uniform detector sensitivity and paper color. A pixel-wise adaptive algorithm shifts the white level to a pre-defined value (see fig. 4.c).

4. The `THRESHOLD` block converts the gray-scale image, after compensation, into a B/W image; conversion takes place by comparing the brightness of each pixel with an adaptive threshold which is a function of both the white and the black levell (see fig. 4.d). From this point onwards, only the B/W image is processed.

5. The `THINNING` block reduces the width of all the strokes to 1 pixel, as shown in fig. 4.f. Thinning is a morphological operator [6] which reduces the width of lines, while preserving stroke connectivity.

6. The `BASELINE` block detects the *baseline* of the handwritten text, which is a horizontal stripe intersecting the text in a known position (see fig. 4.g, left side). Unlike the other steps of preprocessing, the baseline cannot be detected only by means of *local* algorithms (namely, algorithms with a limited neighborhood), as it is a *global* parameter of the entire image. It is computed throughout the scan of the whole image, therefore is available only at the end of the image.

7. The `FEATURES` block detects and extracts from the image a set of 12 *stroke features*, which are helpful for further character recognition. As shown in fig. 4.h (crosses), this block detects the four *left*, *right*, *top* and *bottom concavities*, and the *terminal strokes* in the eight main directions.

   Features are helpful both for center localization and for character recognition, as their types and positions identify almost univocally the character to be recognized. Features are not used alone, but together with the neural recognizer to improve recognition reliability, as described in section 3.2.

8. The `FEATURE REDUCTION` (see fig. 4.h), the `ZOOM` and the `COMPRESS` (see fig. 4.i) blocks reduce, respectively, the number of features (by removing both redundant and useless ones), the *vertical size* of the manuscript (to approximately 25-30 pixels), and the overall size of the manuscript (by a linear factor of 2), by means of ad-hoc topological transformations which do not preserve image shape, although they do preserve its connectivity. These transformations depend on the features detected by the `FEATURES` block, as areas containing fewer features are compressed more than others containing more features.

After all the preprocessing steps, the B/W image is ready for the following neural recognition steps (see section 3.2). The image was reduced both in size (down to $14 \times 18$ (=252) or $12 \times 21$ (=252) pixels for the courtesy and the legal amounts, respectively), in number of gray levels (2), and in stroke thickness (1 pixel), and noise was removed.

Each character fits into 256 bits, which are then reorganized as eight adjacent processor memory words ($8 \times 32$ bits), in order to optimize the performance of the neural detector.

Table 1 lists execution times of individual blocks.

## 3.2 Neural subsystem

The second subsystem is a hybrid neural network recognizer. Most handwriting recognizers [1]-[2] require two consecutive steps, namely:

1. *character segmentation*; each word is first segmented into individual characters. Each character should contain all and only the strokes of the desired character, while strokes of adjacent characters should be removed completely.

2. *character recognition*; each segmented character is recognized. In the literature there is an enormous amount of recognition algorithms [1, 2, 4], based on different techniques. We have considered most of them and analyzed their applicability to the chosen hardware platform, with the constraint of high-speed real-time processing.

   In the end we chose a hybrid approach, which mixes feature-based and neural recognition, as described in detail in section 3.2.3. At this stage of processing, context is not yet considered, as it is later used by the *context analysis subsystem*.

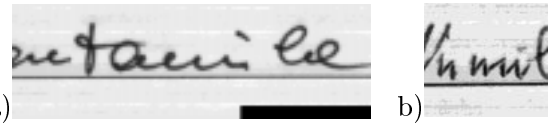


Figure 5: Problems arising in handwriting segmentation: a) difficulties in segmentation; b) ambiguities and pseudo-characters.

### 3.2.1 Centering detector

Segmentation is quite a difficult task, especially for handwritten texts, as there is no clear separation between consecutive characters (see examples in fig. 5). Furthermore the same sequence of strokes can often be interpreted in different ways. For instance, the strokes shown in fig. 5.b can either be interpreted (also by human readers) as "n mi", "nnn", "nnu", "n ini", "vvv", "vwi", etc.

It is obvious that an erroneous segmentation process sends wrong data to the character recognition step. Therefore it may add a number of unwanted artifacts, and it also increases the recognition error rate, as characters not properly segmented are mostly misinterpreted.

For all the reasons described above, we chose a slightly different approach for segmentation. We decided to use an *integrated segmentation and recognition* (**ISR**) approach, in which character segmentation is tightly integrated with character recognition, and no preliminary segmentation is required.

In detail, the `CENTERING DETECTOR` scans the preprocessed and compressed image from right to left (for mechanical reasons) and extracts a *sliding window* of fixed size (either $14 \times 18$ or $12 \times 21$), namely one window for each preprocessed line (see section 3.1). Note that windows without strokes are immediately skipped,

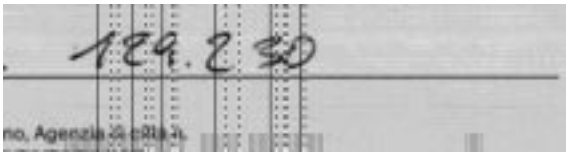

Figure 6: Center positions found by the two center detectors: neural (solid lines) and feature-based (dashed lines).

as they contain no useful information. Thanks to the instruction set available, lines can be skipped in as low as one clock cycle. Each window is associated with its *window coordinate* $x$, which is the distance in pixels of the window's geometrical center from the right hand side of the check.

The `CENTERING DETECTOR` then computes, for each new window coordinate, two *centering functions* $\mathcal{C}_n(x)$ and $\mathcal{C}_f(x)$, which are computed by two independent blocks, respectively:

1. a *neural detector*: a three-layer WRBF + MLP network [4, 5], similar to the one shown in fig. 7, with 252 inputs, 30 and 3 hidden units and 1 output unit (for a total of 7590 centers and weights), trained to detect character centers. Training is supervised and the training set is made up of a large number of patterns obtained by manual classification, containing characters which are either centered or not, each one associated with a target value which is function of its "centering".

2. a *feature-based detector*: this computes the centering function according to the relative position, type and quantity of the features detected by the `FEATURES` block.

The centering functions contain local maxima, which roughly coincide with the location of character centers, as shown in fig. 6. Centering functions have more maxima than character centers, but spurious centers are easily removed by the *clustering subsystem* (see section 3.3).

### 3.2.2 Pseudo-characters

During the design of the system, we have been faced with the opportunity to recognize a set of so-called *pseudo-characters*, which mostly coincide with traditional characters, except for some differences which have been explicitly introduced to improve recognition performance.

The idea behind the concept of pseudo-characters results from the analysis of the problems arising in the identification of character centers. For instance, the ambiguous trace in fig. 5.b can be recognized more easily and with less ambiguity, if it is recognized as a sequence of six pseudo-characters .

Most "m" and "n" consist of the sequence of three (respectively, two) pseudo-characters  . It is obvious that, since characters are not recognized in their original forms, words must be translated according to the chosen set of pseudo-characters. This means that the dictionary used by the *context analysis subsystem* has to be preprocessed. For the example in fig. 5.b, the dictionary will contain the word "U  l...", instead of "Un mil..."[2] (with an appropriate coding of non-ASCII pseudo-characters).

Furthermore all "similar" characters (such as lower case "a"s and "o"s) are identified by the same pseudo-characters.

### 3.2.3 Character recognizer

The `CHARACTER RECOGNIZER` recognizes each individual pseudo-character, using a hybrid approach, which mixes feature-based [3] and neural [1] recognizers.

First of all, features extracted by the `FEATURES` block are used to identify all easy-to-recognize characters. For instance, most "0", "6", "9" digits (but not only these) are written well enough that a straightforward and fast analysis of the main features and strokes is sufficient to recognize those characters with a high accuracy.

Other characters are more difficult to recognize using only features; for instance, digits "4", "7" and some types of "1" can be recognized more easily using neural techniques. All characters which have not been recognized using features are isolated and passed to a neural network trained by an appropriate training set. This section describes only the neural recognizer, as it is the most interesting block in

[2] the first five letters of the Italian equivalent of "one million"

the system.

The CHARACTER RECOGNIZER is "triggered" for each pseudo-character center detected by the CENTERING DETECTOR. As shown in table 1, the CHARACTER RECOGNIZER is the slowest piece of code, due to the large number of synaptic weights involved. Fortunately it is run at a relatively low rate, namely every 15 lines, in the average, therefore its effects on computing time are limited.

As shown in fig. 7 the neural recognizer consists of the cascade of two neural networks:

1. For the courtesy amount, a two-layer WRBF network [4] with 252 inputs, 100 hidden and 20 output units (for a total of 27.400 centers and weights), one for each pseudo-character to be recognized (digits 0-9, plus a few delimiters). For the legal amount, a two-layer WRBF network with 252 inputs, 230 hidden and 46 output units (for a total of 63.340 centers and weights), associated with 15 upper and lower case letters (a, c, d, e, i, l, m, n, o, q, r, s, t, u, v, as these are all and the only characters required to write any number in Italian), and a set of additional pseudo-characters.

   This network is initialized as described in [4], without any further training. The training set used to initialize the network was obtained by manually classifying a number of examples.

2. A one layer MLP [5], with 20 inputs and 20 outputs (or 46 and 46, for the legal amount), which is trained to improve the quality of the output of the neural recognizer. The network was trained using an adaptive delta rule [5].

For each coordinate $x_k$, the output of the CHARACTER RECOGNIZER is a list

$$\mathcal{L}(x_k) = \{(c_1, e_r^1(x_k)), (c_2, e_r^2(x_k)), \ldots, (c_j, e_r^j(x_k)), \ldots\} \quad (1)$$

of *recognition confidences* $e_r^j(x_k)$ associated with each pseudo-character $c_j$. The list is then sorted according to decreasing values of $e_r^j$.

As said before, the CENTERING DETECTOR detects more centers than there are pseudo-characters. This is not a bug, as it helps the CHARACTER RECOGNIZER to improve its performance, since the neural network is allowed to "see" the same pseudo-character more than once, from different "points of view". The additional centers detected are then easily filtered by the *clustering subsystem*.

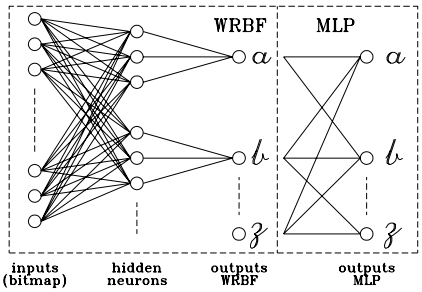


Figure 7: Three-layer neural network block for character recognition.

## 3.3 Clustering subsystem

The *clustering subsystem* aims to group the centers detected by the CENTERING DETECTOR into as many *clusters* as there are pseudo-characters in the courtesy (or legal) amount.

To this aim, some statistics are calculated on the training set, including the average *intra cluster distance*, the average *inter cluster distance*, the average number of centers per cluster, and the average pseudo-character width.

First, we build three triangular *basic centering functions*, for each center $x_k$. The three functions are associated with, respectively, the centering confidences $\mathcal{C}_n(x_k)$ and $\mathcal{C}_f(x_k)$, and the highest recognition confidence of the pseudo-characters in the list $\mathcal{L}(x_k)$. Then we define a *global centering function* which sums the three basic functions and appears as a sequence of *peaks* (local maxima) corresponding to centers, and *valleys* (local minima).

Using the collected statistics, the peaks are grouped into clusters. Each cluster is represented by its highest central peak, named *main peak*.

Whenever multiple hypotheses appear reasonable, all of them are taken into account, so that the *clustering subsystem* actually generates a set of *hypotheses of clustering*.

| | PAPRICA-3 | | Pentium 90 MHz | | Sparc 10 |
|---|---|---|---|---|---|
| | worst case | | morphol. | ad-hoc | morphol. |
| *Image preprocessor* | $\mu$s/line | ms/check | $\mu$s/line | $\mu$s/line | $\mu$s/line |
| WINDOW EXTRACTOR + FILTER | 1.38 | 2.76 | 1,700 | 705 | 1,370 |
| BRIGHTNESS | 2.95 | 5.90 | 1,970 | 320 | 1,660 |
| THRESHOLD | 0.91 | 1.82 | 830 | 255 | 570 |
| THINNING | 8.34 | 16.7 | 7,390 | - | 7,850 |
| BASELINE | 4.24 | 8.28 | 4,320 | - | 3,890 |
| FEATURES | 3.05 | 6.10 | 9,490 | - | 10,430 |
| ZOOM | 2.24 | 4.48 | 820 | 160 | 760 |
| COMPRESS† | 30.8 | 61.6 | 21,350 | - | 24,950 |
| OTHER (VARIOUS) | 3.25 | 6.50 | 3,330 | - | 5,020 |
| TOTAL PREPROCESSING | 57.2 | 114 | 51,200 | - | 56,500 |

| | PAPRICA-3 | | Pentium 90 MHz | |
|---|---|---|---|---|
| | worst case | | morphol. | ad-hoc |
| *Neural subsystem* | ms/psd-char | ms/check | ms/psd-char | ms/check |
| CENTERING (FEATURES) | 0.40 | 19.2 | 1,440 | - |
| RECOGNIZER (FEATURES)†† | 3.60 | 172.8 | 12,960 | - |
| RECOGNIZER (NEURAL)†† | 1.68 | 80.7 | - | 13,440 |
| TOTAL RECOGNIZER | 5,680 | 272.7 | 27,840 | |

Table 1: Average execution times of the various processing steps (with 64 PEs at 66 MHz), while processing the courtesy amount. † COMPRESS acts on an image zoomed by an average factor 3.2, therefore processing times are scaled accordingly. †† CHARACTER RECOGNIZER acts a few times per each pseudo-character, namely once every 15 lines on average.

### 3.4 Context analysis subsystem

For each hypothesis of clustering of the courtesy amount, starting from the one with the highest confidence, we produce a list of *hypothetical courtesy amounts*. These amounts are obtained by generating all the possible combinations of the pseudo-characters associated with each main peak.

The amounts are then sorted according to decreasing values of the *amount confidence*, which is calculated in terms of the recognition confidences of the component pseudo-characters.

Similar considerations also apply to legal amounts.

Each hypothetical courtesy amount is then used to divide the *legal amount* into smaller strings which are easier to recognize. Based on the number of digits in the courtesy amount, the presence of *characteristic tokens* can be assumed in the legal amount. For example, if the number of digits is greater than or equal to 4, the legal amount includes the token "mil" (for either "mila" or "mille"[3]).

The tokens subdivide the legal string into substrings, which are in their turn divided into lower-level substrings, and so on.

In practice, the system searches for *patterns*, i.e., sequences of pseudo-characters in the legal string which correspond to the translation of the searched tokens.

As the translation of characters into pseudo-characters is not univocal, each characteristic token may have more than one equivalent pattern. A *token belief* is computed in terms of the recognition confidences associated with the pseudo-characters of the patterns. The pattern with the highest belief is assumed to match the searched token.

When all the tokens in the legal string have been detected, the amount of the check is identified.

The identified amount is associated with an *amount confidence*, which is defined as the average of the token beliefs of its component tokens. Actually, as several hypotheses are considered, more than one check amount is identified. These amounts are sorted according to descending amount confidences.

The performance of the system is therefore evaluated as the percentage of amounts correctly recognized in the first $k$ ($1 \leq k \leq 4$) positions.

[3] Italian for "thousand"

| Neural Network | internal weights | | external weights | |
|---|---|---|---|---|
| | MCPS | MCUPS | MCPS | MCUPS |
| MLP, $32 \times n$ inputs, 1 bit/input, 8 bits/weight | 130 | 60 | 90 | 50 |
| MLP, $32 \times n$ inputs, 4 bit/input, 16 bits/weight, adaptive learning rate | 33 | 19 | 30 | 16 |
| Kohonen, $1 \times 30$ neurons, 8 inputs, 8 bits/input, $1 \times 5$ neighborhood | 110 | 60 | | |
| Kohonen, $30 \times 30$ neurons, 8 inputs, 8 bits/input, $5 \times 5$ neighborhood | | | 90 | 49 |

Table 2: Performance of the PAPRICA-3 system, in single-chip configuration (namely, 32 PEs), running at 100 MHz, with either internal weights (max. 60 bits/neuron) or external weights (no size limitation).

## 4 Performance Evaluation

Table 1 lists the execution times of the various processing blocks for the example presented in section 3; figures are given for a system with 64 PEs (namely, 2 chips), running at 66 MHz. All the programs were also tested on both a Pentium at 90 MHz and a Sparc Station 10, using the same algorithms based on mathematical morphology, which are well suited to the specific problems of bitmap processing and character recognition. Some programs (`FILTER`, `BRIGHTNESS`, `THRESHOLD`, `ZOOM`, `CENTERING DETECTOR`, `CHARACTER RECOGNIZER`) could be implemented more efficiently on a sequential computer using more traditional methods (ad-hoc programs). These were also implemented on the Pentium and their performance listed in table 1 for comparison.

The performance of PAPRICA-3 is 100 to 1,1000 times faster than that of Pentium and Sparc Station, for almost all the programs considered.

Table 2 lists the execution times of a single-chip PAPRICA-3 system running at 100 MHz, for other well-known neural algorithms such as Perceptrons (MLP) and Kohonen maps [5]. As all mathematical operations are implemented in a bit-serial fashion, system performance depend heavily on input and weight resolution. Furthermore, the best performance can be obtained when the number of either neurons or inputs match the number of PEs.

## References


[1] R.M. Bozinovic and S.N. Srihari, "Off-line cursive script word recognition", *IEEE Trans. on PAMI*, vol. 11, Jan. 1989, pp. 68-83.

[2] L. Evett, C. Wells, F. Keenan, T. Rose and R. Whitrow, "Using linguistic information to aid handwriting recognition", *Proc. Int. Workshop on Frontiers in Handwriting Recognition*, Sept. 1991, pp. 303-311.

[3] H.L. Teulings, "Invariant handwriting features useful in cursive-script recognition", in *Fundamentals in Handwriting Recognition*, S. Impedovo ed., Springer-Verlag, Berlin, 1993, pp. 179-198.

[4] L.M. Reyneri, " Weighted Radial Basis Functions for Improved Pattern Recognition and Signal Processing", *Neural Processing Letters*, May 1995, pp. 2-6.

[5] S. Haykin, "Neural Networks: A Comprehensive Foundation", *Mc Millan College Publishing Company*, New York, 1994.

[6] J. Serra, "Image Analysis and Mathematical Morphology", *Academic Press*, London, 1992.