

Mixing Neural Networks and Contextual Analysis in a High-Speed Handwriting Recognizer

B. Lazzerini[†], L.M. Reyneri^{††}, F. Gregoretti^{††}, A. Mariani[†]

[†] Dipartimento di Ingegneria della Informazione - Università di Pisa

^{††} Dipartimento di Elettronica - Politecnico di Torino

Abstract

This article presents HACRE, a system for recognizing handwritten amounts on checks, which integrates neural network algorithms with context analysis techniques. In particular, HACRE consists of two major subsystems which are strictly interacting: the former is based on an ensemble of neural networks and carries out a kind of pre-recognition of the individual characters, which is by necessity only approximate; the latter is based on a context analysis module which carries out a lexical analysis of the recognized characters, based on a mutual correlation between the legal and courtesy amounts, which must match exactly. This analysis produces hypotheses which correct the errors made by the neural networks. The proposed system is implemented on an ad-hoc VLSI chip containing an array of dedicated processors tailored to the application, and tightly interconnected to a host personal computer.

1 INTRODUCTION

Handwriting recognition [1, 2, 3, 4, 5, 6] is a major issue in a wide range of application areas, including mailing address interpretation [2], document analysis, signature verification [9] and, in particular, bank check processing [2].

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 very high, therefore real-time constraints can 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 HACRE, a complete system for high-speed recognition of the amount on

banking checks. Section 2 gives an overview of the system and the hardware platform, while Sections 3 and 4 describe the image preprocessing and neural recognition steps, respectively. Sections 5 and 6 describe the clustering and the context-based modules, while Section 7 presents some performance results.

2 SYSTEM DESCRIPTION

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 amount 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 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, 4]. 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 cour-

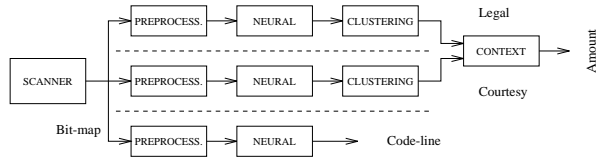


Figure 1: Block diagram of HACRE.

tesy 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 (see Table 1).

The proposed system integrates five subsystems which are in cascade, as shown in Fig. 1:

- 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 [6]. In correspondence to each character of a handwritten word, the *neural subsystem* of HACRE 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.

2.1 Hardware platform

The HACRE system was developed to run on a specific hardware platform (PAPRICA-3), purposely designed at the Politecnico di Torino [13], and tightly interfaced to a host personal computer. The former implements the image preprocessor and the *neural subsystem*, while the host computer implements the clustering and *context analysis subsystems*.

As shown in Fig. 2, the kernel of PAPRICA-3 is a linear array of 64 identical 1-bit Processing Elements (PEs) connected to an image memory via a bidirectional 64-bit wide bus. The image memory

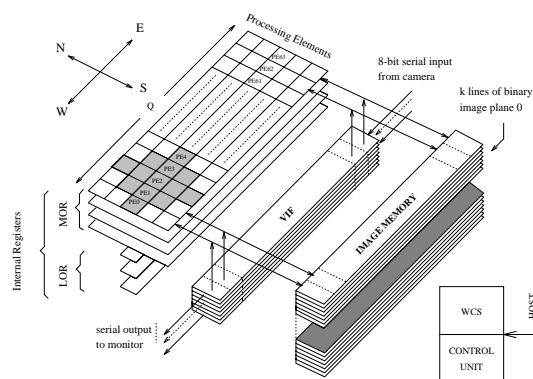


Figure 2: General architecture of the Processor Array.

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 (64 bits wide), and a single cycle is needed to load an entire line of data into the PE’s internal registers.

Data are 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 line 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.

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* [11]: the result of an operation depends, for each processor, on the value of pixels in a given neighborhood (5×5 , as sketched by the grey squares in Fig. 2). Data from E, EE, W and WW directions (where EE and WW denote pixels two bits apart in E and W directions) 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 \rightarrow 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 5×5 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 logical and algebraic operations which act on either Logical Registers or the central bit of Morphological Registers.

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, 64 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 has to be repeated many times, and instruction fetching from an external memory can lead to quite high overheads. Hence we chose to pre-load each block of instructions into an internal memory, named *Writable Control Store (WCS)*, (1K words \times 32 bits), and to fetch instructions from there. The performance of a fast cache can thus be obtained with a hit ratio close to 1, at a fraction of the cost and complexity.

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 pro-

cessor sends the contents of one of its registers and which provides a *Status Word* divided into two sub-fields. 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 global information may be accumulated and stored in the *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 HACRE to implement a neural network, by computing the degree of matching between an image and a set of templates (weight matrices, see Section 4).

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, and in the emulation of pyramidal (hierarchical) processing.

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

Some 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.

The HACRE system is composed of one PAPRICA-3 chip plus a fast static RAM providing a $128K \times 64$ 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.

3 IMAGE PREPROCESSOR

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

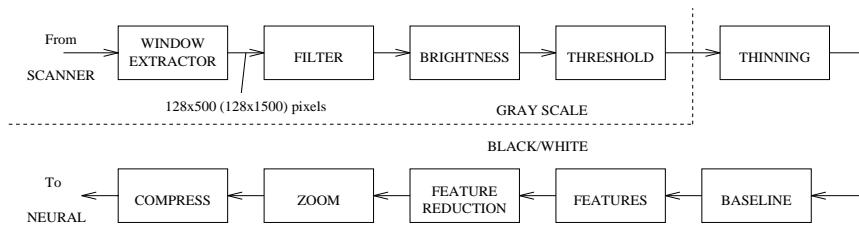


Figure 3: Block diagram of the image preprocessor.

1. The **WINDOW EXTRACTOR** acquires the input image from the **SCANNER**, at a resolution of approximately 200 dpi, gray scale, 16 levels. The scanner is an 876-pixel CCD line camera scanned mechanically over the image, from right to left (for practical reasons), at a speed of 2m/s (which is equivalent to about 700 characters/s). Image size is about 876×1500 pixels, although character recognition takes place in two smaller areas of known coordinates. These areas, 128×500 and 128×1500 pixels in size, respectively, are extracted from the scanned image by means of an ad-hoc CCD controller. Figure 4.a shows an example of a scanned image.

Imaging sensor and light source are both cheap, therefore illumination and detector sensitivity may vary significantly from pixel to pixel. Furthermore, reflectivity and paper color may also vary significantly among different areas of the image. Dark black and bright white pixels are read as 0 and F values, respectively, although in practice the contrast is much smaller, namely 2 to 6 levels, on average.

As described in Section 2.1, 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 3×3 pixel kernel. The result is shown in 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. The pixel by pixel *white levels* and *black levels* (one per each pixel of the CCD camera) are computed as the maximum (respectively, minimum) brightness intensity encountered at each pixel, while the image is scanned. White and black levels are periodically damped, to reduce the influence of very bright and very black spots, which are small in size. The result is shown in 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 levels. The result is shown in Fig. 4.d. From this point onwards, only the B/W image is processed. The **THRESHOLD** block also filters and partially removes the spot noise from the image, as shown in Fig. 4.e.
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 [11] which reduces the width of lines, while preserving stroke connectivity. Thinning is required to properly identify character features (see the **FEATURES** block).
6. The **BASELINE** block detects the *baseline* of the handwritten text, which is a horizontal stripe intersecting the text in a known position, as shown in 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.

The baseline is required to detect properly the area where the handwritten text is located, in order to get rid of unwanted texts: for instance, other preprinted messages such as the name of the bank (not present in the example shown in Fig. 4, but visible in Fig. 6).

The baseline is also used to reduce the height of the processed image, from 128 pixels (minimum size to guarantee catching the whole manuscript, independently of its position within a ± 5 mm tolerance boundary) to 64 pixels (to match the processor size).

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, this block detects the four

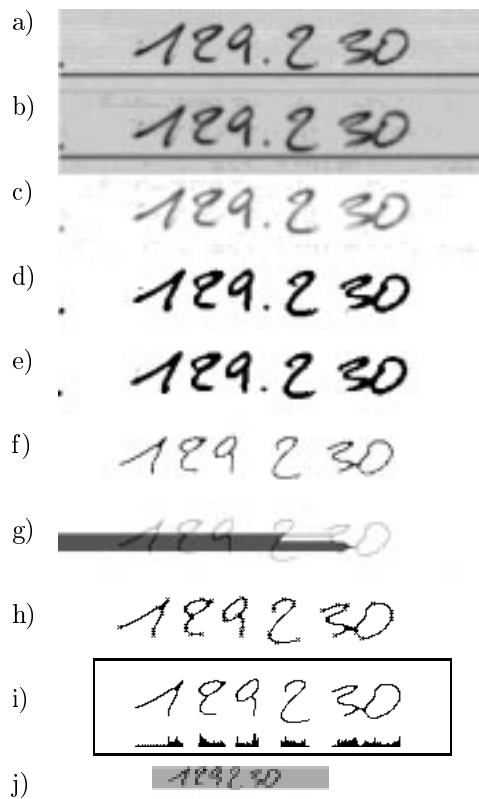


Figure 4: Preprocessing steps of handwritten images (an “easy” example): a) original image, 200 dpi, 16 gray levels; b) low-pass filtered image; c) compensated for brightness; d) thresholded image; e) spot noise removal; f) thinned, after 6 steps; g) finding baseline (at the left side of the image);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

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 4.

8. The FEATURE REDUCTION block reduces the number of features, by removing both redundant and useless features. Examples of redundant features can be seen in the vertical stroke of digit “9” shown in Fig. 4.h.
9. The ZOOM block evaluates the *vertical size* of the manuscript and scales it in size to approximately 25-30 pixels. Scaling the size facilitates the task of the neural recognizer, as the number of weights required for proper recognition is reduced. Vertical size is an approximate measurement of calligraphy size, and is

defined as the minimum height of a rectangle which contains about 90% of the strokes (for digits, but the percentage may vary for literals). The height is computed by means of the height histogram shown in Fig. 4.i.

We decided not to include 100% of the strokes, to prevent very long but localized vertical strokes from influencing the computation of the vertical size of the manuscript (which sometimes happens with digits “1”, “5” and “7”). For texts including alphanumeric (in particular, lower case letters), the percentage was slightly reduced to take into account the presence of ascending and descending strokes of letters such as “b”, “g”, “t”, etc.

10. The COMPRESS block reduces image size further approximately by a factor 2 by means of an ad-hoc topological transformation which does not preserve image shape, although it does preserve its connectivity, as shown in Fig. 4.j. This transformation depends on the features detected by the FEATURES block, as areas containing fewer features are compressed more than others containing a greater number of features.

At this point, after all the preprocessing steps, the B/W image is ready for the following neural recognition steps (see Section 4). The image was reduced both in size (down to 14×18 (=252) or 12×21 (=252) pixels for the courtesy and the legal amounts, respectively), in numbers of gray levels (2), and in stroke thickness (1 pixel), and noise was removed.

All these steps, if properly tuned, contribute to increasing the reliability of the neural recognizer. At the end each character fits into 256 bits, which are then reorganized as four adjacent processor memory words (4×64 bits) in order to optimize the performance of the neural detector.

Table 2 lists the execution times of all the individual blocks (running on the hardware described in Section 2).

4 NEURAL SUBSYSTEM

The second subsystem of HACRE is a hybrid neural network recognizer. Most handwriting recognizers [1]-[6] 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. The process of segmentation is useful in order to send separated characters to character recognition;

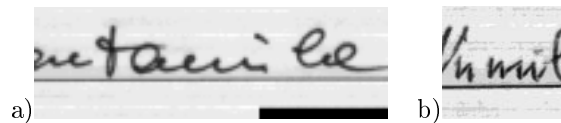


Figure 5: Problems arising in handwriting segmentation: a) difficulties in segmentation; b) ambiguities and pseudo-characters.

2. *character recognition*; each segmented character is recognized. In the literature there is an enormous amount of recognition algorithms [1, 2, 3, 4, 5, 6], based on different techniques. We have considered most of them and analyzed their applicability to our 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 4.3. At this stage of processing, context is not yet considered, as it is used by the *context analysis subsystem* described in Section 6.


Segmentation is quite a difficult task [12], 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”, “vvi”, etc.

It is obvious that an erroneous segmentation process sends the 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.

4.1 Pseudo-characters

An additional feature of HACRE is its ability to recognize a set of so-called *pseudo-characters*, which mostly coincide with traditional characters, except for some differences which were 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”s and “n”s 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.























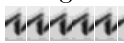
pseudo character	Amount	
	legal	courtesy
	a,o,O	0
	d	-
	i,u,m,n	-
	c,C	-
	e,l	-
	e,l	-
	m,n	-
	r	-
	t	-
	v,V	-
	a	-
	l,I	1
	o,O	0
	-	1
	-	2
	-	3
	-	4
	-	5
	-	6
	-	7
	-	8
	-	9

Table 1: A few pseudo-characters used by HACRE, for the legal and the courtesy amounts.

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...”¹ (with an appropriate coding of non-ASCII pseudo-characters).

As the correspondence between characters and pseudo-characters is not univocal, each word in the dictionary produces more than one equivalent using pseudo-characters. For example, the letter “a” can be translated into either “o”, or “ci”, “cl”.

The optimal set of pseudo-characters should be the one which minimizes the error rate of HACRE. In theory it should be found through optimization of the measured probability of correct recognition, but this would take too long to measure. We therefore decided to identify heuristically a sub-optimal set, by manually analyzing a large set of examples.

Table 1 lists the pseudo-characters used in the project. A new set of pseudo-characters is at present being identified, resulting from the latest results obtained.

4.2 Centering Detector

For all the reasons described above, we chose a slightly different approach for segmentation. We

¹the first five letters of the Italian equivalent of “One million”

decided to use an *integrated segmentation and recognition (ISR)* approach [6], in which character segmentation is tightly integrated with character recognition, and no preliminary segmentation is required. This approach has three main advantages:

1. there is no need to develop a specific segmentation algorithm. This significantly shortened the design phase of HACRE;
2. no segmentation error is introduced. In reality, the ISR approach performs a task (center localization) which partially resembles segmentation, except that it only detects pseudo-character centers, without separating them from their neighboring pseudo-characters. Therefore errors can still be introduced in center localization, but not in segmentation;
3. localization of pseudo-character centers can partially be implemented by means of a neural network, therefore the system can *self-learn* to detect centers from a set of appropriate *training examples (training set)*. This further reduces development time.

In detail, the CENTERING DETECTOR scans the preprocessed and compressed image from right to left (for mechanical reasons) and extracts a *sliding window* of a fixed size (either 14×18 or 12×21), namely one window for each preprocessed line (see Section 3). Note that windows without strokes are immediately skipped, as they contain no useful information. 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* which depend on several image characteristics, as described below. The centering functions contain local maxima, which roughly coincide with the location of pseudo-character centers, as shown in Fig. 6. Centering functions have more maxima than pseudo-character centers, but spurious centers are easily removed by the *clustering subsystem* described in Section 5.

The two centering functions $\mathcal{C}_n(x)$ and $\mathcal{C}_f(x)$ are computed by two independent blocks, respectively:

1. a *neural detector*: a three-layer WRBF+MLP network [5, 10], similar to the one shown in Fig. 7, with 252 inputs, 30 and 3 hidden units and 1 output unit, trained to detect pseudo-character centers. Training is supervised and the training set is made up of a large number of patterns obtained by manual classification, containing pseudo-characters which are either

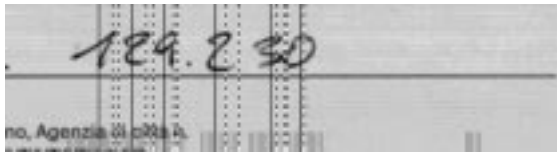


Figure 6: Center positions found by the two center detectors: neural (solid lines) and feature-based (dashed lines).

centered or not, each one associated with a target value given by:

$$t(x) = 1 - \frac{2|x - x_C|}{\bar{w}}, \quad \text{with } t(x) \geq 0 \quad (1)$$

where x_C is the correct center coordinate (still obtained by manual classification), while \bar{w} is the *average pseudo-character width*, in pixels, computed as a side result of manual classification of the training set;

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 function also depends on a weighted average of some geometrical characteristics of the pseudo-characters, such as: i) height of the pseudo-character at the center of the sliding window, ii) curvature of the pseudo-character in correspondence to the top-most feature, iii) distance between adjacent vertical strokes, etc.

Figure 6 shows the results of the two center detectors. Solid and dashed lines are drawn in correspondence of the peaks of the centering functions $\mathcal{C}_n(x)$ and $\mathcal{C}_f(x)$ of the neural and the feature-based detector, respectively. It is clear that each pseudo-character center is localized at more than one position, but the *clustering subsystem* easily filters and removes the redundant ones, as described in Section 5.

Note also that the pseudo-character’s “center” is not necessarily at its geometrical center, but at a given predefined position, unique to each pseudo-character, which improves the overall performance of the recognition system. This position is automatically learnt by the neural detector while, for the feature-based detector, some heuristics were used during development.

The CENTERING DETECTOR delivers the following values to the *context analysis subsystem*, for the k -th peak of either $\mathcal{C}_n(x)$ or $\mathcal{C}_f(x)$:

1. the window coordinate x_k of the peak;
2. the *centering confidence* $e_n(x_k) = \mathcal{C}_n(x_k)$ of the neural detector;
3. the *centering confidence* $e_f(x_k) = \mathcal{C}_f(x_k)$ of the feature-based detector.

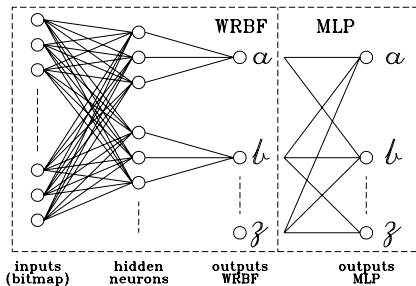


Figure 7: Three-layer neural network block for character recognition.

4.3 Character Recognizer

The CHARACTER RECOGNIZER recognizes each individual pseudo-character, using a hybrid approach, which mixes feature-based [7] and neural [8] 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. HACRE isolates all characters which have not been recognized using features and passes them to a neural network trained by an appropriate training set. This section describes only the neural recognizer, as it is the most critical block in the system.

The CHARACTER RECOGNIZER is “triggered” for each pseudo-character center detected by the CENTERING DETECTOR. As shown in Table 2, 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, on 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 [5] with 252 inputs, 100 hidden and 20 output units, one for each pseudo-character to be recognized (digits 0-9, plus a few delimiters, as shown in Table 1). For the legal amount, a two-layer WRBF network with 252 inputs, 230 hidden and 46 output units, 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 pseudo-characters, as shown in Table 1.

This network is initialized as described in [5], 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 [10], 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 [10].

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

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

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 .

The recognition confidences are then further processed for the *context analysis subsystem*, which receives the *weighted recognition confidence* $\hat{e}_r^j(x_k)$:

$$\hat{e}_r^j(x_k) = \sum_{n=1}^{N_M} P(c_j | c_{i(n),n}) \cdot e_r^{i(n)}(x_k) \quad (3)$$

where $P(c_j | c_{i,n})$ is the probability that the CHARACTER RECOGNIZER has in input pseudo-character c_j conditioned by the fact that in output it recognizes pseudo-character c_i in the n^{th} position of the sorted list $\mathcal{L}(x_k)$, while $i(n)$ is the index of the pseudo-character in the n -th position of the sorted list $\mathcal{L}(x_k)$, and N_M is the number of pseudo-characters considered.

The conditional probability $P(c_j | c_{i,n})$ is calculated, by applying Bayes’ theorem, as follows:

$$P(c_j | c_{i,n}) = \frac{P(c_{i,n} | c_j) \cdot P(c_j)}{\sum_{l=1}^{N_M} P(c_{i,n} | c_l) \cdot P(c_l)} \quad (4)$$

where

$P(c_{i,n} | c_j)$ is the probability that the CHARACTER RECOGNIZER recognizes pseudo-character c_i in the n^{th} position of the sorted list $\mathcal{L}(x_k)$, when there is pseudo-character c_j in input, while $P(c_j)$ is the probability that pseudo-character c_j is present in input. In practice, $P(c_{i,n} | c_j)$ is obtained by submitting all the pseudo-characters in the training set to the CHARACTER RECOGNIZER and verifying, for each pseudo-character c_j , the occurrence frequency of pseudo-character c_i in the n^{th} position of the sorted list $\mathcal{L}(x_k)$. $P(c_j)$ is the appearance frequency of pseudo-character c_j in the training set.

For each center x_k , the *neural subsystem* produces a tuple

$$\mathcal{T}(x_k) \triangleq \{x_k, e_n(x_k), e_f(x_k), \hat{\mathcal{L}}(x_k)\} \quad (5)$$

where e_n and e_f are the two centering confidences, while $\hat{\mathcal{L}}(x_k)$ is called *list of alternatives* and is defined as:

$$\hat{\mathcal{L}}(x_k) \triangleq ((c_1, \hat{e}_r^1(x_k)), \dots, (c_N, \hat{e}_r^N(x_k))) \quad (6)$$

The list of alternatives $\hat{\mathcal{L}}(x_k)$ contains as many elements as there are pseudo-characters. The list is then sorted according to decreasing values of the weighted recognition confidence \hat{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*.

5 CLUSTERING SUBSYSTEM

5.1 Characterization of the Centering Detector

In order to produce hypotheses about input based on the *neural subsystem* output, we characterized the behavior of the CENTERING DETECTOR.

Generally, more than one center is detected in correspondence to a specific pseudo-character. We call *cluster* the group of contiguous centers related to the same pseudo-character (see Figs. 6 and 8). The centers detected by the CENTERING DETECTOR can be classified as *body centers* (b_i in Fig. 8), which are close to a pseudo-character center, *tail centers* (t_i in Fig. 8), typically corresponding to ligatures or side terminal strokes of a pseudo-character, and *spurious centers* (s_i in Fig. 8), which do not correspond to a real pseudo-character, but typically to an intermediate position between two consecutive pseudo-characters. In general, there are more body than tail centers in a pseudo-character.

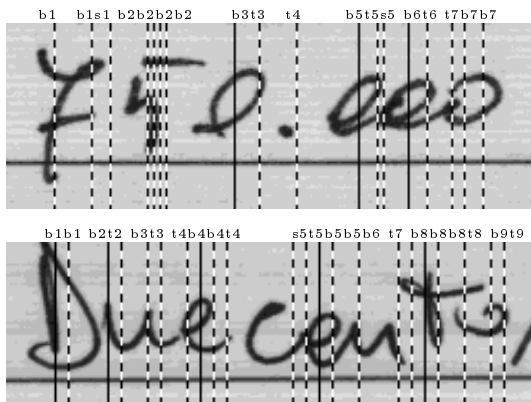


Figure 8: Body (b_i), tail (t_i) and spurious (s_i) centers. ‘i’ indicates the cluster the center is associated with. Character ‘c’ in the middle of the lower bitmap is a non revealed pseudo-character.

We calculated, for all pseudo-characters in the training set (see Table 1):

1. the statistical distribution of the center distances, in particular:
 - the average distance between two contiguous centers belonging to the same cluster (*average intra cluster distance*); and
 - the average distance between two contiguous centers belonging to two consecutive clusters (*average inter cluster distance*);
2. the average recognition confidences of the first two pseudo-characters in the sorted lists of alternatives associated with body centers.

The behavior of the CENTERING DETECTOR was as follows, for some typical cases:

- *Well detached pseudo-characters* (for instance, “7”, “5” and “0” in Fig. 8): the CENTERING DETECTOR detects some (typically 2 – 3) adjacent body and few (1 – 2) tail centers; when a pseudo-character is correctly recognized, it is present (in about 90% of cases) with a high recognition confidence as either the first or the second alternative in the sorted lists associated with all the body centers. The average inter cluster distance is about 1.4 to 1.5 times greater than the average intra cluster distance.
- *Close (non touching) pseudo-characters or touching pseudo-characters* (for instance, “000” in Fig. 8): the CENTERING DETECTOR usually detects spurious centers, too; the first two alternatives of the list associated with a spurious center are usually different from the first two alternatives of the sorted lists associated with the adjacent left and right centers (either spurious or not). The distance between two consecutive centers (either spurious or not) is approximately the same (generally, a few pixels). Spurious centers may also be due to ambiguities in centering detection; these ambiguities (and therefore the occurrence of spurious centers) are reduced by using pseudo-characters rather than characters.
- *Non revealed pseudo-character* (for instance, “c” in Fig. 8): the CENTERING DETECTOR does not localize the pseudo-character and often splits it into two parts and interprets each of them as being part of the adjacent left and right pseudo-characters, respectively; in this case (which is not the one shown in Fig. 8), groups of tail centers belonging to the same pseudo-character look as dense as groups of body centers. However, the inter cluster distance is a little greater (typically, 10 to 15 pixels) than the intra cluster distance.

- *Delimiters*: a delimiter is a pseudo-character (like “#”) written at the beginning and/or the end of an amount; the CENTERING DETECTOR recognizes a series of centers which cannot be correlated with each other. Typically, all the alternatives associated with these centers have low recognition confidences and differ from each other.

We used the collected statistics to interpret the CENTERING DETECTOR output: the detection of one of the characteristics listed above offers a clue to one of the previous situations.

Also, for clustering purposes, we calculated on the training set the average number of centers per cluster, and the average pseudo-character width $\bar{\omega}$.

Finally, to associate a confidence degree with our interpretation of the *neural subsystem* output, we measured for all legal and courtesy amounts in the training set, the distribution of a few stochastic processes such as the number of centers versus the number of pseudo-characters, and the recognized pseudo-character versus the input character.

5.2 Clustering

Clustering aims at grouping centers into clusters. By exploiting information gathered during the previous characterization phase (see Section 5.1), the clustering procedure produces a list of *hypotheses of clustering*, i.e., different groupings of the centers detected by the CENTERING DETECTOR.

We produce more than one hypothesis since it is not possible to select only one with a sufficiently high degree of confidence. All those hypotheses will be considered by the *context analysis subsystem*.

We first build three *basic centering functions* $f_n(x - x_k)$, $f_f(x - x_k)$ and $f_r(x - x_k)$, for each center x_k . The three functions are associated with, respectively, the centering confidences $e_n(x_k)$ and $e_f(x_k)$, and the weighted recognition confidence $\hat{e}_r^{j_{\max}}(x_k)$ of the pseudo-character $c_{j_{\max}}$ with the highest confidence in the list $\hat{\mathcal{L}}(x_k)$.

Basic functions are triangular functions of the horizontal coordinate x :

$$f_n(x - x_k) \triangleq \begin{cases} e_n(x_k) \cdot \left(1 - \frac{2|x - x_k|}{\bar{\omega}}\right), & \text{for } 2|x - x_k| \leq \bar{\omega} \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

while similar formulae apply to $f_f(x)$ and $f_r(x)$. Then we define the *global centering function*:

$$\mathcal{F}(x) \triangleq \sum_k f_n(x - x_k) + f_f(x - x_k) + f_r(x - x_k) \quad (8)$$

Partial sums of f_n and f_f are shown in Fig. 9. This function appears as a sequence of *peaks* (local maxima) corresponding to centers, and valleys (local

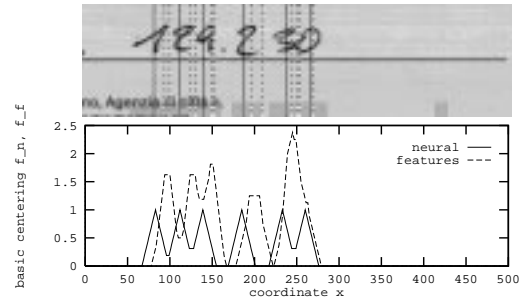


Figure 9: Sums of the two basic centering functions f_n and f_f , for $\bar{\omega} = 25$.

minima). Note that, due to how the global centering function is built, two or more centers next to each other (less than $\bar{\omega}/2$) either originate only one peak, or produce a few (typically 2) very close peaks which have approximately the same height. In the former case, the peak is likely to be close to the actual pseudo-character (geometrical) center, while in the latter case, only one peak is considered and the other(s) are ignored.

As shown in Fig. 9, there are peaks which can be classified either as *isolated* or *non isolated*. A peak is isolated if and only if: either its nearest peak is at a distance greater than or equal to $\bar{\omega}$, or it is much higher than the closest peak. We assume that isolated peaks identify as many clusters.

Typically, an isolated peak corresponds to a well detached pseudo-character. On the other hand, a group of non isolated peaks may correspond to one or more pseudo-characters, mainly depending on the distances between the peaks and the similarity of the pseudo-characters in the first alternatives of their lists $\hat{\mathcal{L}}(x_k)$.

The clustering process identifies a collection of *main peaks* which initially contains all and only the isolated peaks (we suppose that at least one main peak exists).

We then start building a cluster around each main peak, by selecting from detected centers. We decide whether or not to associate a center with the j -th cluster depending on i) its distance from the j -th main peak, ii) the similarity of the first two pseudo-characters of the center and of the main peak, and iii) other information collected during the characterization phase described in Section 5.1.

If a center might be associated with either of two consecutive clusters, we investigate the possible presence of a non revealed pseudo-character. The same is done when the distance d between two consecutive clusters is approximately $\frac{\bar{\omega}}{2}$.

At this point, not all centers have been associated with the clusters. Then we behave as follows: if d is greater than $\bar{\omega}$, we assume the presence of $d/\bar{\omega}$ clusters with a width approximately equal to $\bar{\omega}$, and we add a central peak (if any) of each such cluster

to the collection of the main peaks. If no center falls within a hypothesized cluster, there is assumed to be a non revealed pseudo-character at the input. A possible cluster specifically introduced to consider a non revealed pseudo-character is assumed to have a dummy main peak associated with an empty list of alternatives.

The presence of a non revealed pseudo-character will be taken into account by the *context analysis subsystem* to interpret the output of the *neural subsystem* correctly.

On the other hand, if the distance $d \ll \bar{\omega}$, it may be reasonable to merge two clusters into one, by removing the lower of the two peaks from the collection of main peaks. Note that whenever multiple hypotheses appear to be reasonable, all of them are taken into account, so that the *clustering subsystem* actually generates a set of *clustering hypotheses*.

Each hypothesis is the sequence of the main peaks of the clusters:

$$\mathcal{H}^p = (x_{k_1^p}, x_{k_2^p}, \dots, x_{k_{N_P}^p}) \quad (9)$$

where $x_{k_i^p}$ is the coordinate of the i -th main peak in the p -th hypothesis, while N_P is the number of clusters.

The clustering procedure also sorts hypotheses according to their *clustering confidence* (see formula (8)):

$$e(\mathcal{H}^p) \triangleq \sum_{i=1}^{N_P} \mathcal{F}(x_{k_i^p}) \quad (10)$$

6 CONTEXT ANALYSIS SUBSYSTEM

6.1 Hypothetical Courtesy and Legal Amounts

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* \mathcal{A}^m . These amounts are obtained by generating all the possible combinations of the pseudo-characters (digits and delimiters) associated with each main peak, and taking into account some trivial semantic rules (for instance, no delimiter can appear between two digits, etc.). Up to a maximum of two delimiters on both sides of a courtesy amount are supported.

The amounts are then sorted according to decreasing values of the *amount confidence*

$$e(\mathcal{A}^m, \mathcal{H}^p) \triangleq \frac{\sum_{i=1}^{N_P} \hat{e}_r^{j(m,i)}(x_{k_i^p})}{N_P} \quad (11)$$

where \hat{e}_r^j is the weighted recognition confidence, $j(m, i)$ indicates the index of the pseudo-character in the i -th position of the m -th amount, while p

and m are the indexes of the hypothesis of clustering and the hypothetical courtesy amount, respectively. N_P is the number of pseudo-characters of the p -th hypothesis of clustering.

Similar considerations apply to legal amounts too.

6.2 Guided Subdivision of the Legal String

The hypothetical courtesy amount is 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”²); furthermore, if the number of digits is greater than or equal to 7, the legal amount includes the token “milion” (for either “milione” or “miloni”³). Note that a characteristic token is a word in the original (not yet translated) dictionary.

The tokens subdivide the legal string into one, two, or three substrings, called *first level substrings*, each corresponding to a number of at most three digits. The number associated with the leftmost substring may have either one, two or three digits; all the other substrings, called *intermediate substrings*, are associated with numbers that contain exactly three digits. For example, if the courtesy amount is 123,456,789, the characteristic tokens and the first level substrings are ⁴:

centoventitreMILIONiquattrocentocinquantaseiMILAsettecentottantanove

A first level substring is then processed as follows.

- If the associated number has three digits, the token “cento”⁵ is looked for; for example, the number 456 corresponds to the string “quattroCENTOcinquantasei”. In this way each first level substring is subdivided into two substrings (called *second level substrings*), corresponding, respectively, to a number of one digit (hundreds) and a number of two digits (tens and units).
- If the associated number has two digits, the following cases may happen:
 - if the number is greater than or equal to 10 and less than or equal to 19, one of the tokens “dieci”, ..., “diciannove”⁶ is looked for;

²Italian for “thousand/s”

³Italian for “million/s”

⁴Italian for “123,456,789”

⁵Italian for “hundred”

⁶Italian for “ten”, ..., “nineteen”


- if the number is greater than or equal to 20, one of the following tokens is looked for: “vent”, “trent”, ..., “novant”⁷, and a substring (called *third level substring*), corresponding to a number of one digit is isolated; e.g., *CINQUANTasei*⁸.

- If the associated number has one digit, one of the tokens “uno”, ..., “nove”⁹ is looked for.

The intermediate first level substrings always correspond to numbers of three digits. However, if the most significant digit is 0, the string is processed as if there were two digits. Likewise, if the first two digits are 0, the string is processed as if there were one digit. Similarly, the second level substrings corresponding to numbers of two digits, the most significant of which is 0, are processed like strings corresponding to numbers of one digit.

6.3 Characteristic Tokens Search in the Legal Amount

The system tries to localize the position of the characteristic tokens in the legal amount (actually, the hypothetical legal amount). The recognition of a token is driven by the search for *key characters* which help us to guess the presence of the token (e.g., “q” for token “cinque”¹⁰, “t” for token “sette”¹¹, etc.). In practice, the system searches for *patterns*, i.e., sequences of pseudo-characters in the legal string which correspond to the translation of the tokens being searched for. As a consequence, one or more consecutive *key pseudo-characters* representing the key character are looked for. Recall that, as the translation of characters into pseudo-characters is not univocal, each characteristic token has more than one equivalent pattern made up of pseudo-characters.

Once a key character has been found, the system moves both to the right and to the left looking for appropriate sequences of pseudo-characters corresponding to the other characters of the token. For example, the system looks for the sequence of a few  at the left side of an “l” to match the token “mil” (see Fig. 5.b), or a sequence “se” at the left side of a “t” to match token “sette”.

On the other hand, the position of the currently searched token in the legal string is estimated as follows. Based on the number of recognized digits, but taking possible misrecognition of such digits into account, a range of starting positions (in the number of pseudo-characters) is established based

on the minimum and maximum lengths of all possible strings preceding the token being searched for.

Consider, for example, the pattern $\{c_a c_b c_c c_d\}$ and suppose it is identified by the key pseudo-character c_c in the k -th position. The system controls the pseudo-characters adjacent to c_c and computes a *token belief* as:

$$I_P \triangleq \hat{e}_r^a(x_{k-2}) + \hat{e}_r^b(x_{k-1}) + \hat{e}_r^c(x_k) + \hat{e}_r^d(x_{k+1}) + \dots \quad (12)$$

where $\hat{e}_r^a(x_{k-2}), \dots$ are the recognition confidences associated with the pseudo-characters c_a, \dots in position $k-2, \dots$.

The token belief is higher in proportion to the length of a corresponding correct sequence of pseudo-characters. The value of a token belief is minimum when very few or no pseudo-characters have been found that match the characters of the characteristic token.

The pattern with the highest belief is assumed to match the token being searched for.

6.4 Resolving ambiguity among digits

The digits which are difficult to distinguish are grouped together. For example, “1” might be mistaken for “7”, “3” for “8”, and so on. Therefore, if a digit belonging to one of such groups is recognized, a pattern coincident with any of the elements of the group is searched for in the appropriate position of the legal string. Note that the same digit can be included in different groups.

6.5 Final Amount Analysis

As previously described, the courtesy amount field is used to produce hypotheses which are then verified by analyzing the legal amount field. The detection of a token is a proof of the correctness of the part of the hypothesis that suggested the presence of the token. When all the tokens in the legal string (namely, in all third level substrings) have been detected, then 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 (see Table 3).

⁷ Italian for “twenty”, “thirty”, ..., “ninety”

⁸ Italian for “fiftysix”

⁹ Italian for “one”, ..., “nine”

¹⁰ Italian for “five”

¹¹ Italian for “seven”

<i>Image preprocessor</i>	HACRE processor		Pentium 90 MHz		Sparc 10
	worst case		morphol.	ad-hoc	morphol.
	$\mu\text{s}/\text{line}$	ms/check	$\mu\text{s}/\text{line}$	$\mu\text{s}/\text{line}$	$\mu\text{s}/\text{line}$
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	52,620	56,500

<i>Neural subsystem</i>	HACRE processor		Pentium 90 MHz	
	worst case		morphol.	ad-hoc
	ms/psd-char	ms/check	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	5.68	272.7	27,840	

<i>clustering subsystem</i>	Pentium 90 MHz			
	ms/psd-char		ms/check	
	courtesy	legal	courtesy	legal
<i>clustering subsystem</i>	4.0	1.3	28.0	36.0
<i>context analysis subsystem</i>	0.26		6.7	

PREPROCESSING + RECOGNIZER (HACRE) and CONTEXT ANALYSIS (Pentium 90MHz)		
	ms/psd-char	ms/check
Total	7.15	343

Table 2: Average execution times of the processing step of HACRE (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. Note that total times may be slightly shorter than the sum of individual times. This is due to the pipelined internal architecture of HACRE.

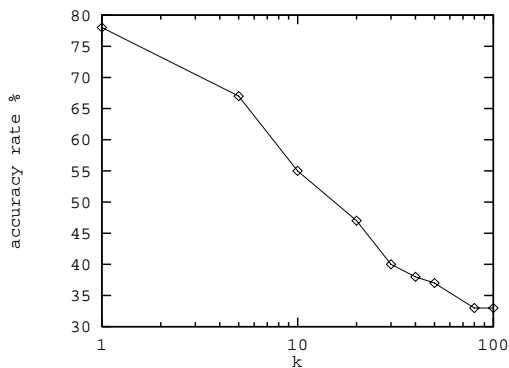


Figure 10: Accuracy rate versus the position k in which the correct courtesy amount is found.

7 PERFORMANCE EVALUATION

Table 3 lists the performance of the major processing blocks. Performance is given in terms of accuracy rates, for the CHARACTER RECOGNIZER and the *context analysis subsystem*, and in terms of RMS centering error, for the CENTERING DETECTOR.

Figure 10 shows the percentage of checks correctly recognized (in the first 4 positions) versus

the position of the correct amount in the list of hypothetical courtesy amounts.

Table 2 lists execution times of the various processing blocks; figures are given for a system with 64 PEs, running at 66 MHz. It is clear that the system can recognize up to 3 cheques per second.

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 implemented on the Pentium and their performance also listed in Table 2 for comparison.

Note that the clustering average execution time per pseudo-character is calculated as the ratio between the clustering average execution time of the amount and the average number of pseudo-characters of the amount. As the average numbers of pseudo-characters are 6.9 and 27.4, for the courtesy and the legal amounts, respectively, we obtain the data in Table 2.

Module	Parameter	court.	legal
CENTERING DETECTOR	RMS centering error (distance between detected center coordinate x_k and actual (geometrical) center of pseudo-character), in pels	2.5	3.0
CHARACTER RECOGNIZER	Average accuracy rate (correct, if the pseudo-character is in the list five positions in the list $\hat{\mathcal{L}}(x_k)$)	74%	52%
CHARACTER RECOGNIZER	Average accuracy rate (on all centers of the correct clusters)	88%	65%
<i>clustering subsystem</i>	Ratio between number of correct clusters and number of hypothesized clusters	87%	74%
<i>context analysis subsystem</i>	Average accuracy rate (correct, if the correct amount is in the first 4 positions of the list of final amounts)	53%	

Table 3: Performance of the HACRE system, measured on 80 checks.

Note that the performance of HACRE is two to three decades faster than that of Pentium and Sparc station, for almost all the programs considered.

8 CONCLUSION

This paper has described a high-speed handwriting recognizer for interpreting the amount on Italian banking checks. The system has an accuracy of 53% and can interpret up to 3 checks per second.

Acknowledgments

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

References

- [1] R.M. Bozinovic and S.N. Srihari, "Off-line cursive script word recognition", *IEEE Trans. on PAMI*, vol. 11, pp. 68-83, Jan. 1989.
- [2] E. Cohen, "Computational theory for interpreting handwritten text in constrained domains", *Artificial Intelligence*, Elsevier Science ed., New York, vol. 67, pp. 1-31, 1994.
- [3] M. Costa, E. Filippi and E. Pasero, "Combining Multi-Layer Perceptrons in Classification Problems", *Proc. of European Symposium on Artificial Neural Networks*, ESANN 94, Bruxelles, April 1994, pp. 49-54.

- [4] 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*, pp. 303-311, Sept. 1991.
- [5] L.M. Reyneri, "Weighted Radial Basis Functions for Improved Pattern Recognition and Signal Processing", *Neural Processing Letters*, May 1995, pp. 2-6.
- [6] M. Schenkel, H. Weissman et al., "Recognition-based Segmentation of On-line Hand-printed Words", *Advances in Neural Information Processing System 3*, Morgan Kaufmann ed., 1992, pp. 723-730.
- [7] 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.
- [8] C.J.C. Burges, J.I. Ben, Y. LeCun, J.S. Denker, C.R. Nohl, "Off-line Recognition of Handwritten Postal Words Using Neural Networks", in *Int'l Journal on Pattern Recognition and Artificial Intelligence*, vol. 7, 1993, pp. 689-703.
- [9] G. Pirlo, "Algorithms for Signature Verification", in *Fundamentals in Handwriting Recognition*, S. Impedovo ed., Springer-Verlag, Berlin, 1993, pp. 436-454.
- [10] S. Haykin, "Neural Networks: A Comprehensive Foundation", *Mc Millan College Publishing Company*, New York, 1994.
- [11] J. Serra, "Image Analysis and Mathematical Morphology", *Academic Press*, London, 1992.
- [12] J. Keeler, D.E. Rumelhart, "A Self-organizing Integrated Segmentation and Recognition Neural Net", in *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann, San Mateo (CA), 1992, pp. 496-503.
- [13] F. Gregoretti, R. Passerone, L.M. Reyneri, C. Sansoè, "The Implementation of the PAPRICA-3 Parallel Architecture", in *Proc. of 2nd Int'l Conference on Massively Parallel Computing Systems*, Ischia (I), May 1996, IEEE Computer Society ed., pp. 87-94.