Multi-resolution Classification Trees in OCR Design

MARCEL BRUN, JUNIOR BARRERA, NINA S. T. HIRATA,
NESTOR W. TREPODE, DANIEL DANTAS AND ROUTO TERADA

Departamento de Ciência da Computação
Instituto de Matemática e Estatística - Universidade de São Paulo
Rua do Matão 1010, CEP 05508-900, São Paulo, SP
<mbrun,jb,nina,walter,ddantas,rt>@ime.usp.br

Abstract. This paper recalls the idea of classification trees in OCR (Optical Character Recognition) systems and proposes a technique for the automatic design of these classification trees. The design of both the classification trees and of the classification operators are based on training from sample pairs of observed-ideal images, allowing the development of customized OCRs.
Keywords: Classification tree, OCR, morphological multi-classifier.

1 Introduction

The aim of an optical character recognition (OCR) system is to generate electronic documents (ASCII files) corresponding to printed materials such as books, office documents, memoranda, etc. In general, this process involves image analysis procedures. Recognition of characters is one of the major tasks involved in the process of an OCR. In character recognition, one usually faces two fundamental problems: to segment the characters (letters and symbols) [5, 6] and to classify each of the characters.

Several approaches for character recognition have been proposed in the last decades [7]. There are several commercial OCRs in the market. In order to improve the accuracy of the overall result, the usual approaches incorporate from image processing and pattern recognition techniques to spell checking procedures. While the generality of some OCRs limits their accuracy, it is possible to project a customized OCR for a specific font or document set and achieve higher accuracy. Specific OCRs may incorporate training and filtering that are particularly suited for the document type to be processed. In this case, in place of generality, we seek accuracy.

In this paper we consider the use of morphological operators for character recognition, extending a previous result on the use of classification trees in OCR [4]. In the referred work, a method based on a classification tree was introduced. Instead of designing one classifier for classifying all objects in a single classification step, the classification process is viewed as a multi-step procedure, that is, objects are classified into subclasses and then each subclass is further classified into more subclasses. However, a difficulty part of the proposed technique is the design of the classification tree. Here we discuss an automatic technique for designing classification trees.

Following this introduction, in Section 2 we recall some basic concepts on mathematical morphology for binary images, its extension to the classification of shapes, and design of morphological multi-classifiers. In Section 3, we recall classification trees in the context of OCRs. In Section 4, we introduce a technique for the automatic design of classification trees. In Section 5, we describe the main components involved in the proposed OCR. In Section 6, we present some application results. Finally, in Section 7, we present the conclusions of this work and discuss further researches in this field.

2 Morphological operators for shape classification

An image is typically composed by several objects, where each object may correspond to more than one component in the image. The purpose of classification is to assign a unique classification code to each object in the image, according to the class it belongs to. This may be accomplished by assigning the correct classification code to all pixels in each object.

Let $E = Z^2$ be the integer plane and $\mathcal{P}(E)$ be the power set of E. A binary image on E may be modeled by a binary function $f : E \to \{0, 1\}$, or equivalently, by a subset $S \subseteq E$, by letting $x \in S \iff f(x) = 1$, for all $x \in E$.

A mapping $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$ is *translation invariant* (t.i.) if and only if

$$x \in \Psi(S)_h \iff x \in \Psi(S_h), \forall x, h \in E, \forall S \subseteq E, \quad (1)$$

where S_h denotes the translation of set S by vector h.

Let $W \subseteq E$ be a finite subset, called *window*. A mapping $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$ is *locally defined within* W (l.d.) if and only if

$$x \in \Psi(S) \Longleftrightarrow x \in \Psi(S \cap W_x), \forall x \in E, \forall S \subseteq E. \quad (2)$$

A mapping $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$ that is both i.t. and l.d. is called a *binary* W*-operator* and can be characterized by a binary function $\psi : \mathcal{P}(W) \to \{0, 1\}$, as follows:

$$x \in \Psi(S) \Longleftrightarrow \psi(S_{-x} \cap W) = 1, \forall x \in E\,. \quad (3)$$

Binary W-operators can be used, by adjusting the window size, to recognize shapes of objects that belong to one particular class. Hence, if there are n different classes, the classification could be performed by a family ψ_j of n binary operators, one for the recognition of objects in each of the classes. The drawback of this approach is that one needs to design an operator for each class. We also need to deal with the question on how to deal with objects that are recognized by more than one operator.

A natural extension of binary classifiers are the multi-classifiers [4]. Let $\{0, 1, \ldots, n\}^E$ denote the set of all mappings from E to $\{0, 1, \ldots, n\}$ (corresponding, in our context, to the set of gray-level images defined on E, with $n + 1$ gray levels) and let us consider an operator of the form $\Psi : \mathcal{P}(E) \to \{0, 1, \ldots, n\}^E$, characterized by a multi-level function $\psi : \mathcal{P}(W) \to \{0, 1, \ldots, n\}$. Here, an operator assigns one classification label in $\{0, 1, \ldots, n\}$ to each of the shapes in W. In this case, the input is a binary image and the output is a gray level classified image. Each pixel in the output image has a value between 0 and n, the classification label. The label 0 is reserved for the background, and n indicates the number of classes considered.

The advantage of using multi-classifiers is in the fact that instead of designing n operators, one for each class, one needs to design only one operator.

2.1 Representation of multi-classifiers

Let ψ be a mapping from $\mathcal{P}(W)$ to $\{0, 1, \ldots, n\}$. Define the *kernel of* ψ *at level* i as being the set $K_i(\psi)$ given by, for any $\mathbf{x} \in \mathcal{P}(W)$,

$$\mathbf{x} \in K_i(\psi) \Longleftrightarrow \psi(\mathbf{x}) \geq i. \quad (4)$$

Note that $K_n(\psi) \subseteq K_{n-1}(\psi) \subseteq \cdots \subseteq K_1(\psi) \subseteq K_0(\psi)$.

The mapping ψ has the following sup-decomposition [3]:

$$\psi(\mathbf{x}) = max\{i : \mathbf{x} \in K_i(\psi)\}. \quad (5)$$

The *basis of* ψ *at level* i, $B_i(\psi)$, is the set of maximal intervals of $K_i(\psi)$. In terms of basis, the sup-decomposition can be written [3] as

$$\psi(\mathbf{x}) = max\{i : \mathbf{x} \in [A, B], [A, B] \in B_i(\psi)\}. \quad (6)$$

2.2 Design of multi-classifiers

Here we describe a procedure for designing multi-classifiers from training data. In our context, a *training data* is a set of observed-ideal pairs of images, illustrating the desired classification. The observed images are binary images containing the objects to be classified and the ideal images are the respective images with the correct classification label.

Shapes $\mathbf{x} \subseteq W$ and their respective classification labels y are collected from the training data. Then, they are used to estimate $P(y = i|\mathbf{x})$, $i = 0, 1, 2, \ldots, n$. For each shape $\mathbf{x}$ observed in the training data, the most frequently assigned label defines the class c to which $\mathbf{x}$ belongs, that is, we set $c(\mathbf{x}) = i$ by choosing i such that $P(y = i|\mathbf{x}) = max\{P(y = j|\mathbf{x}), j = 0, 1, 2, \ldots, n\}$.

The next step consists in designing an operator ψ such that $\psi(\mathbf{x}) = c(\mathbf{x})$, for each observed shape $\mathbf{x}$. The procedure consists in designing each of the n basis, corresponding to the maximal intervals of the kernel (eq. 4), from level 1 to n. The bases $B_j(\psi)$ associated to the kernels can be computed by an incremental procedure [4, 1]. In this procedure, minimization of the representation and generalization of the classification (i.e., attribution of classification to the shapes not observed in the training data) are accomplished. Minimization of representation is given by the basis representation of the kernels and generalization depends on how the algorithm deals with the non-observed shapes. In this work we use the ISI (incremental splitting of intervals) [2] algorithm.

The design of multi-classifiers may be computationally inefficient if the number of classes and the window size are large. Another problem is the estimation precision of the conditional probabilities. Recently, classification trees [4] were proposed as an approach to overcome these difficulties.

3 Classification Trees

A *classification tree* breaks the original classification task in a non-single number of classification steps. In other words, instead of assigning the correct classification in a single classification step, the objects are initially classified into subfamilies. Then, in a second step of classification, each of the subfamilies are classified into another number of subfamilies, and this process is

repeated for each subfamily until a full classification is performed. This idea is shown through a diagram in Figure 1.

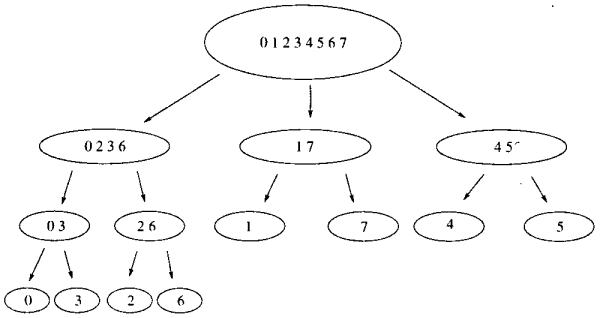

Figure 1: Example of a classification tree.

In the example of Figure 1, the classification process consists of three steps. The whole set of classes ($\{0,1,2,3,4,5,6,7\}$) is first split into three subfamilies, namely, the sets $\{0,2,3,6\}$, $\{1,7\}$ and $\{4,5\}$. The set $\{0,2,3,6\}$ is then split into two subsets, $\{0,3\}$ and $\{2,6\}$, while the other two are split into unitary sets, and so on.

Each non-leaf node V of a classification tree contains the following information:

- a subfamily $\mathcal{F}(V)$ of classes.
- a list of child nodes $child(V)$. The subfamilies assigned to the child nodes form a disjoint partition of $\mathcal{F}(V)$.
- an operator (multi-classifier) Ψ_V that classifies the objects contained in classes of $\mathcal{F}(V)$ into the respective subfamilies of its child nodes.

The leaf nodes contain objects from a single class. Thus, there is no operator assigned to the leaf nodes.

Once such a structure is given, we need to design operators that are able to classify objects according to the tree. For instance, the operator assigned to the root node must be able to separate the family of all possible objects ($\{0,1,2,3,4,5,6,7\}$) into three subsets $\{0,2,3,6\}$, $\{1,7\}$ and $\{4,5\}$. The operator assigned to the first child node must be able to separate objects in $\{0,2,3,6\}$ into two subfamilies, namely $\{0,3\}$ and $\{2,6\}$, and so on.

Given an image and a classification tree whose set of objects in the root node is the set of all objects present in the image, in order to classify those objects, one must traverse the entire tree (except for the leaf nodes) applying the operators. The operator of the root node is applied on the original image. An image is created for each child node, containing only those objects classified as being of the respective subfamily. Then, the operator of each of the child nodes is applied on the respective images, and so on, until a leaf node is reached. The classification result is obtained from all final images.

The basic idea behind this technique is that classification into intermediary classes of objects should be easier because the operator does not need to perform complex discrimination of shapes.

4 Design of Classification Trees

To design a classification tree, a tree structure is usually fixed and then an operator for each non-leaf node is designed using the training procedure described in Section 2.2. Window size is empirically tested for each node, until an acceptable result is found. If a character receives more than a certain percentage of classification as being of a subfamily, then it is kept in the training images of the respective child node. Due to this, a character may be in images of more than one of its children. If a character is kept in a subfamily where it should not be, the operator assigned to split that subfamily tends to eliminate it because it is trained to recognize only those characters in the respective subfamily.

A difficulty part on designing classification trees is to specify the family of classes assigned to each node. In designing the classification trees, a natural choice is to keep the objects that have similar shapes in a same subfamily.

We propose a new technique that automatically designs a classification tree structure. The goal is to use small windows at the first levels of the tree and generate intermediary groups that may be easily discriminated.

4.1 Classification matrix

Let $\mathcal{F}$ be a family of object classes. We would like to find a relatively small window W that is enough to discriminate objects in different classes of $\mathcal{F}$. In order to analyze the discriminatory power of a given window W, we introduce the *classification matrix*.

To build a classification matrix, for a given window W and family $\mathcal{F}$, a multi-classifier based on W is designed from images containing objects of the classes in $\mathcal{F}$, according to the description in Section 2.2. This operator is applied on the same images, in order to compute $T(i,j)$, i.e., how many pixels of a character of class i are classified as being of a class j. The sum of a row i in T indicates the total number of pixels corresponding to objects of class i.

Figure 2 shows a classification matrix for the family $\{A, B, C, D, E, F\}$. In this matrix we can see that

T(i,j)	A	B	C	D	E	F
A	210	0	0	0	0	3
B	0	239	0	43	0	0
C	0	0	172	0	0	0
D	0	0	0	381	0	0
E	0	0	0	0	118	8
F	2	0	0	0	28	271

Figure 2: Example of a classification matrix.

210 pixels of character A are correctly classified while 3 of them are incorrectly classified as being of character F. Two pixels of character F are classified as being of character A and 28 pixels as of character E, over 271 correct classifications. On the other hand, no part of characters A, E and F has been classified as B, C or D, and no part of these letters as A, E or F. Thus, we can say that this classifier is able to separate the family $\{A, E, F\}$ from the family $\{B, C, D\}$. Moreover, we can see that letter C is never misclassified, and no other letter is misclassified as letter C. Therefore, we can partition the family $\{A, B, C, D, E\}$ in three subsets, $S_1 = \{A, E, F\}$, $S_2 = \{B, D\}$ and $S_3 = \{C\}$.

4.2 Similarity graph

The discriminatory power of a multi-classifier can be analyzed through its classification matrix. To analyze the similarity between objects of distinct classes, we define a non-directed graph G, where the vertices are the classes (characters), and there exists an edge between two vertices i and j if and only if $T(i, j) > 0$ or $T(j, i) > 0$. This graph, induced by the classification matrix, is the *similarity graph*, where similar characters are connected by an edge. In this graph, three cases may occur:

1. The graph is completely disconnected, i.e., $T(i, j) = 0, \forall i, j,\ i \neq j$. This is the ideal case, i.e., the window is enough to discriminate all the characters.

2. The graph is totally connected, i.e., $T(i, j) > 0, \forall i, j,\ i \neq j$. This indicates that the characters contain similar shapes relative to the window W and they may not be discriminated; window W is too small. The greater are the values of $T(i, j)$ and $T(j, i)$, the larger is the similarity between the characters i and j.

3. The graph contains nontrivial connected subgraphs. In this case, the operator does not classify all objects correctly. It is possible, however, to group the characters into subfamilies in such a way that no part of a character in a subfamily receives the classification of a character in another subfamily.

Figure 3 shows the graph obtained from the matrix T in Figure 2. We can see that there is three connected subgraphs with vertex sets $V_1 = \{AFE\}$, $V_2 = \{BD\}$ and $V_3 = \{C\}$, respectively. This means

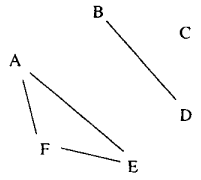

Figure 3: Connected subgraphs.

that an operator based on W would be able to discriminate the elements in V_1 from the elements in V_2 or V_3. However, it may not be able to clearly discriminate characters A, F and E, because there is, for instance, some "confusion" between the shapes present in A and F. Our conclusion here is that the operator based on W could be used to split the set $\{A, B, C, D, E, F\}$ in the three sub-sets.

In general, case (3) results in a family $\{G_1, G_2, \ldots, G_p\}$ of disjoint subgraphs, meaning that no part of an object in a class of G_i has been classified as being of a class in another subgraph G_j. In this case, the window is able to separate elements of different subgraphs. Hence, if we define the sets $V_1, V_2, \ldots, V_p$ as the subfamilies under a node, a W-operator would be able to discriminate the elements in G into the p subfamilies.

In practice, even if the misclassification $T(i, j)$ and $T(j, i)$ are nonzero, these values may be very small, compared to the number of pixels corresponding to the characters i and j. In other words, if only a small part of a character is classified incorrectly, it is still possible to assign the right classification to it.

Let $N(i)$ and $N(j)$ denote the number of pixels of all occurrences of characters i and j, respectively, and define

$$N(i, j) = \min\{N(i), N(j)\}\,. \tag{7}$$

Graph G is created from a slightly modified classification matrix, which we denote T', obtained by taking into account a threshold that depends on $N(i, j)$ and

a percentage parameter θ, $0 \leq \theta \leq 100$:

$$T'(i,j) = \begin{cases} T(i,j), & i \neq j \text{ and } T(i,j) > \frac{\theta \cdot N(i,j)}{100} \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

The resulting graph will have an edge between two classes only if the misclassification between them is significant, i.e., if the number of misclassified pixels between them is a considerable percentage (defined by θ) of the number of pixels belonging to those classes. This may cause some loss in the quality of the classifier, but it tries to prevent the use of large windows, which are not desirable because a large amount of training data is needed to reach a good precision. The loss in the classification quality can be overdue by the gain in precision.

4.3 Design procedure

To define a classification tree structure, initially a root node is created and a graph G is built from the classification matrix T', obtained from the whole family of classes $\{0, 1, \ldots, n\}$, using the first window in a sequence of increasing windows. In our experiments we have used the sequence 3×3, 5×3, 5×5, 7×5, ... (Figure 4).

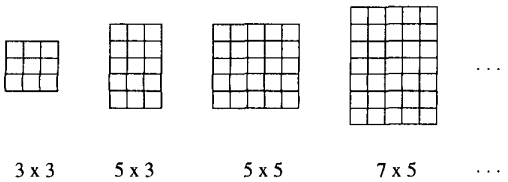

Figure 4: Windows sequence.

The design process incrementally builds a classification tree, according to the rules described next. These rules depend on the case that occurs in G (listed at the beginning of the section). If case (1) occurs, the process ends for that node. If case (2) occurs, that means that the window is too small, and the process is repeated for that node, using the next window in the sequence of windows. Depending on the shape of the characters, it may happen that even very large windows can not separate characters into disjoint sets. On the other hand, it is not desirable to allow the window to increase too much, because that will imply very poor precision for the trained operator. To avoid this problem, each time the window is increased in this case, the parameter θ is also increased by a given factor. This may imply that the discrimination quality could be compromised. However, since this will happen to a branch of the tree, it may be insignificant relative to the overall classification result. Finally, if case (3) occurs, then the family of classes in that node is split into subfamilies corresponding to the disjoint subgraphs. For each subfamily it is created a child node, and the process ends for that node. The window to be used for the child nodes is the next one in the sequence and the parameter θ is not changed.

The process is iteratively repeated while there are nodes to be processed. At the end of this process, a tree structure with a window and a family of classes assigned to each node is defined. Once the structure is defined, the next step consists in training a multi-classifier for each node. As we have seen before, each multi-classifier is trained to separate the characters in the family of classes assigned to it into subfamilies corresponding to its child nodes.

Figure 5 shows a complete classification tree, except for the leaf nodes, designed for a character recognition problem discussed in Section 6.1. Figure 6 enhances a branch of this tree corresponding to the subfamily composed by characters "mr1IliLY". A 7×7 window is enough to discriminate characters "m" and "r" from the others. A larger window, 9×9, is enough to recognize characters "L" and "Y", while a 11×9 window recognizes characters "T" and "i". The last class, contains three very similar characters ("1", "I" and "l") and the window needed to be increased to 13×11. Usually, for a window of this size, the amount of training data required for good precision is very large. In this case, however, this problem is attenuated because the variety of shapes to be analyzed is limited.

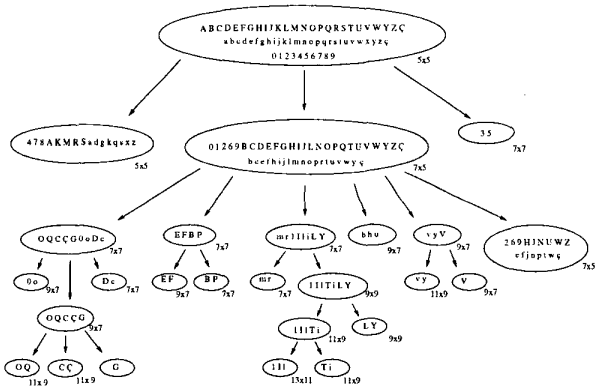

Figure 5: An example of classification tree.

5 OCR Implementation

The two main parts of the designed OCR consists of a training and an application component. The training

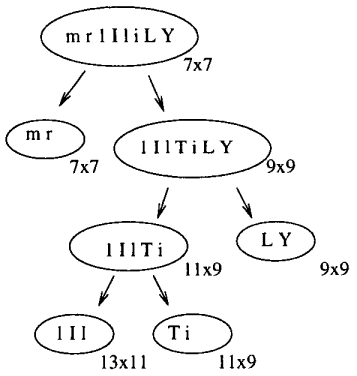

Figure 6: A subtree of the classification tree.

component allows the user to train a classification tree from samples of observed-ideal images. The application component allows the user to apply a classification tree.

Images need to be adequately prepared for the utilization of both components. The characters need to be segmented [6] and labeled [4] according to their natural order of occurrence in the image, in order to generate the correct text after the classification. The preparation process may involve some tasks such as filtering, enhancement, conversion of images from gray level to binary, character normalization and down sampling, not necessarily in this order. Depending on the images involved, the filtering protocol may vary from case to case. The resulting image must be a gray level image representing a binary labeled image.

Two procedures that aims to reduce the variety of shapes are *edge noise filter by stamp* and *anchoring* [1]. The anchoring process needs information on the labeling of characters. Figure 7 shows a possible sequence of steps in the preprocessing stage.

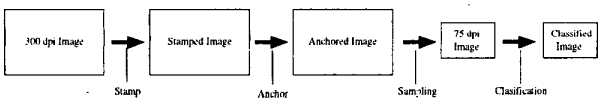

Figure 7: Preprocessing stages for classification.

Once the pre-processing stage is applied on the images, one can either design a classification tree (the tree structure and the classification operators) by using the training component of the OCR, or apply a given classification tree on those images using the application component and generate the respective RTF files.

The classification is based on a previously designed classification tree. The trained classifier attributes a value to each point in the image. Thus, a character in the image may be labeled with different values and, therefore, we need to decide which class label will be assigned to each character. This is done by voting: for each character, we assign the most frequent value assigned by the designed multi-classifier to its points.

Once the characters in the image are classified, the next step of the OCR is to create a text document corresponding to the text contained in the image. In this part of the process, the labeled image is important to establish the order in which the characters must appear in the text. For the cases in which the most frequent label is not clear, the respective character may be marked by a special symbol that means "unrecognized". The unrecognized characters may be later solved by spell checking procedures or by other recognition techniques.

An OCR should also recognize structures such as words, paragraphs, titles. OCRs may incorporate another functionalities such as text segmentation, automatic text layout recognition, and others. In such cases, there should be also a new component that will integrate all kinds of information (character, layout, pages, etc).

6 Experimental Results

The new methodology to design classification trees was tested for two groups of images: a set of images obtained from laser printed materials corrupted with edge noise, and another set of images obtained from documents printed by a dot matrix printer.

6.1 Laser printer digitized images

The source images are composed of 63 different characters, the letters in lower and upper cases, and the numbers from 0 to 9. Each image is composed of 630 characters, containing exactly 10 of each character. These images were obtained by scanning a laser printed text at 300 dpi. Because of the good quality of these images, we added 25% of salt and pepper noise on the external edges of the characters, in order to simulate a degradation that may usually occur in scanned text images. The external edges of an image can be obtained by subtracting the image from its dilatation by a 3×3 box structuring element B, i.e., $Edge = \delta_B(Image) - Image$. Figure 8 shows the effect of this noise addition.

Two stamp operators were designed from two images. These operators were applied on the images, one after the other, and then the anchoring process followed by a down-sampling by a factor of 4 were applied

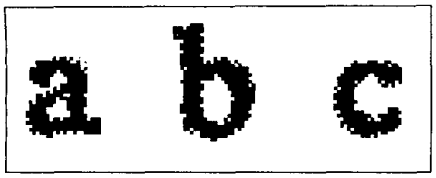

Figure 8: Characters with salt noise.

on them. Eight of these images were used for the generation of the classification tree, using $\theta = 1.5\%$ and threshold factor of 2, both experimentally determined. The obtained classification tree is the one depicted in Figure 5.

Figure 9 shows the result of the operator used to discriminate the letters "E" and "F" at the node labeled EF in the classification tree (Figure 5). The same region in the original image is also shown. The original image contains other characters than "E" and "F", but they are not being considered when processing this node. Almost all points in each of the characters are correctly classified.

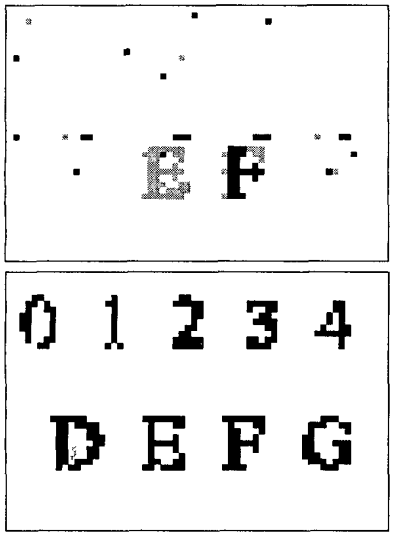

Figure 9: Top: classification result for a node. Bottom: Same region in the original image.

The designed classification tree was tested on ten different test images of the same type. Table 1 shows the result of these tests, including also tests with some commercial OCRs. Compared to these commercial OCRs, our performance was significantly better. In each image, from a total of 630 characters, only a small number of them (between 7 and 13) were misclassified.

OCR	Accuracy
Textbridge	87.89%
Cuneiform'99	90.63%
Ours	98.33%

Table 1: Accuracy on the first group of images.

6.2 Dot matrix printer digitized images

Dot matrix printers generates poor quality images, because the letters are roughly formed by points. The documents we have used are printed in a special paper with a yellow background pattern increasing the difficulty of the recognition.

A top-hat transformation, with a 7×7 diamond structuring element was applied to eliminate the background pattern. The images were binarized by thresholding with an small threshold value. The binary images were then filtered by a closing-opening operator with a 5-point cross structuring element. These images have been processed by two stamp operators, followed by anchoring and down-sampling by a factor of 4. A sample of a scanned gray-scale image and the respective filtered, binarized and shrank images are shown in Fig. 10.

Four images with a total of 5040 characters in them were used to obtain the stamp operators and to design a classification tree. Parameters θ (threshold) of 3% and threshold factor of 3 were used. The obtained classification tree was tested on two different images with a total of 2505 characters. The result is shown in Table 2 together with results of a commercial OCR. The performance of the commercial OCR on the original gray-scale image is very poor. Thus, we also tested it on the images filtered by a threshold followed by two median filters and also by our filter. In this OCR, an option "Dot matrix" was set on to process the images. The best result of the commercial OCR, which is due

OCR	Tested image	Accuracy
Cuneiform'99	Original gray-level	20.19%
Cuneiform'99	Our filter	77.83%
Cuneiform'99	Threshold + medians	85.97%
Ours	Our filter + stamping	88.58%

Table 2: Accuracy on the second group of images.

to an external filtering, is statistically close to our result. However, our result can be improved by using more training data, because only four training images have been used in this experiment.

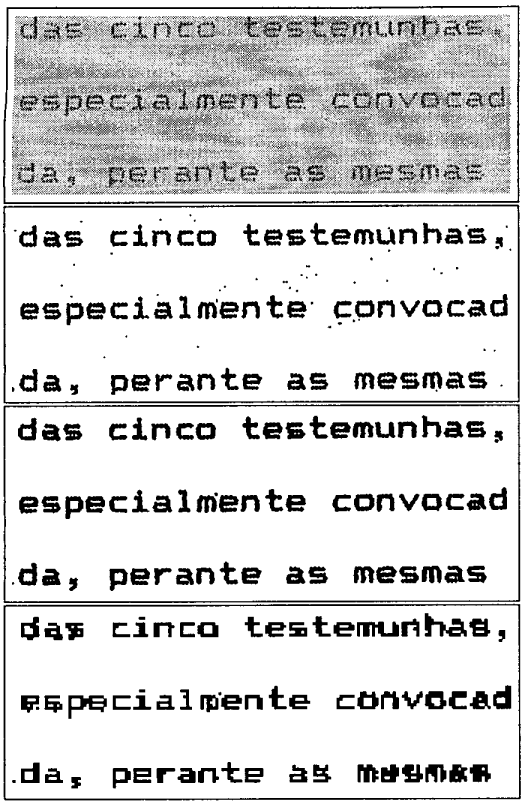

Figure 10: From top to bottom: original, binarized, filtered and shrank images (scale modified).

7 Conclusions

In this paper, we have presented a technique for automatic design of classification trees, based on concepts of classification matrix and similarity graph. This method is efficient because it does not analyze all possible combinations of subfamilies for a given family of classes. These subfamilies are automatically generated by analyzing the similarity graph, obtained from the classification matrix that, essentially, tells how well a given window W discriminates the objects in different classes.

A characteristic of our approach is that it always tries to find the smallest window that is enough to separate the set of objects into disjoint subsets. This allows the same object to be analyzed at different resolutions. Basically, at the initial classification steps, a gross separation of characters can be done by a relatively small window. At the deepest steps of the classification process, the window required to separate the characters is larger, however, the variety of objects that need to be recognized is usually very small, making classification statistically easier.

The concepts introduced here have been applied in the context of OCR design, however they may be applied to any classification problem. The results we have obtained for two sets of images are better than of some commercial OCRs. Since our approach is based on training, it is possible to design customized classifiers for particular classification problems.

8 Acknowledgments

M. B. is supported by FAPESP under grant 98/15586-9. N. W. T. is supported by FAPESP under grants 99/11147-3 and 00/10684-4.

References

[1] J. Barrera, M. Brun, R. Terada, and E. R. Dougherty. Boosting OCR classifier by optimal edge noise filtering. *ISMM 2000, International Symposium on Mathematical Morphology and its Applications to Image and Signal Processing V*, 2000.

[2] J. Barrera, E. R. Dougherty, and N. S. Tomita. Automatic Programming of Binary Morphological Machines by Design of Statistically Optimal Operators in the Context of Computational Learning Theory. *Electronic Imaging*, 6(1):54–67, January 1997.

[3] J. Barrera, R. Terada, R. Hirata Jr, and N. S. T. Hirata. Automatic Programming of Morphological Machines by PAC Learning. *Fundamenta Informaticae*, 41(1-2):229–258, January 2000.

[4] J. Barrera, R. Terada, R. A. Lotufo, N. S. T. Hirata, R. Hirata Jr., and F. A. Zampirolli. An OCR based on Mathematical Morphology. In *Nonlinear Image Processing IX*, volume 3304 of *Proceedings of SPIE*, pages 197–208, San Jose, CA, January 1998.

[5] R. G. Casey and E. Lecolinet. A Survey of Methods and Strategies in Character Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, July 1996.

[6] Y. Lu. Machine Printed Character Segmentation – An Overview. *Pattern Recognition*, 28(1):67–80, 1995.

[7] S. N. Srihari, J. J. Hull, and B. Suny. *Encyclopedia of Artificial Intelligence*, volume 2, chapter Character Recognition, pages 138–150. John Wiley and Sons, Inc., 1992.