Annales Univ. Sci. Budapest., Sect. Comp. 17 (1998) 269-276

AN OCR PROGRAM APPLYING CLASSICAL METHODS AND NEURAL NETWORKS

J. Kömüves, L. Csink, Zs. Csereszny e,
L. Körösi and M. Puskely
(Budapest, Hungary)

Abstract. In this paper we present an attempt to combine classical methods of character recognition with neural networks. There are a lot of OCR (optical character recognition) algorithms, but they differ in their effectiveness regarding different letters. We use a neural network to learn the effectiveness of each method depending on the given letter. The principle is similar to that of blackboard systems, but it is realised with a Kohonen network. We will give an overview of the character recognition problems we faced, describe our program and give test results.

1. Introduction

Character recognition systems contribute to the process of storing information electronically, but they would also give help to the blind reading a paper. They are cheaper than typing text in. Plenty of image processing techniques exist ([3], [4], [5]) which were developed for classification of objects. Optical character recognition is a special case of them because printed characters have rather unique features ([10], [11]). Just two examples from Hungary.

Recognita [5] is a Hungarian software which was developed using heuristic algorithms. It has great success, mainly in Europe. It is based on analysing the contour of characters. It also has a skeletonisation pre-processing method. These result in font independent recognition. Having extracted the features, a tree based classification algorithm decides what the examined character is.

Another Hungarian software on the market is *OCulaR*. It operates with multilayer backpropogation neural networks.

In the framework of a project at Kandó Kálmán Polytechnic we have tried to combine the classical methods of OCR with neural networks and examine their effectiveness. The task was to write a program which is able to recognize printed text, for example, in a block, and convert it into an ASCII file. The input of the program is a graphic file made by scanner. The output is an ASCII file. We made a simple user interface so that the program would be easy to use, but the tests were made without it.

2. Description of the program

The program consists of a few main parts:

- image acquisition,
- pre-processing,
- image segmentation (separating the spots of different letters),
- recognition of letters, building strings up from the recognized letters,
- checking the found words against a knowledge base, and making modifications if an error is found.

We developed our system under UNIX on Sparc 10 and Sparc Classic computers, using SunOS 3.5 release.

2.1. Image acquisition

Printed information were transferred to a graphic file. We used a scanner of 256 grey levels. A 6×9 square inch printed area resulted a file in the range of 0.5-5 megabytes depending on resolution.

2.2. Pre-processing and segmentation

With the greatest care, the pages may be orientated slightly rotated when being put in the scanner. It will be more difficult to find a row or a printed string, and the extracted features on the orientation as well. Therefore the image must be rotated back. We developed a simple method for measuring the initial, wrong orientation of the image. It is converted to a bi-level one and two "big" characters of the first row are found. There are two kinds of letters by their size. "Big" characters are those the top of which is further away from the baseline than the others (e.g. A...Z, b, d, f, h, k, l, t). The top pixels are connected by a line. Ideally this line is horizontal, but mostly it is rotated by

a few degrees (Fig.1.). After measuring the slant of this line the whole image can be rotated back by the given angle.

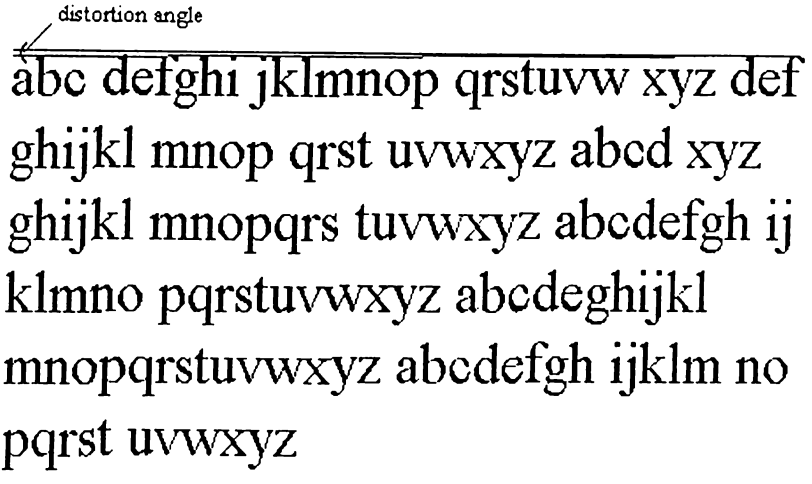

Fig.1. The rotated scanning

The rotation causes noise in the image. There are two solutions of this problem. One is rotating the original grey level image back, and converting it into bi-level again. Another solution is filtering (see [1], [2], [12]). We tried median filtering, but since every scanning is a time consuming process, it was omitted.

To identify the characters, we have to find them first. The search starts at the top of the image, and the rows are found first. In general empty row scans separate two adjacent rows. The height of a row is known in advance and it seems to be too great, the rows can be separated by the minima of the numbers of pixels in the line scans (Fig.2.).

Sometimes noisy black pixels can be found between two rows in principally empty scan lines. This may make the program "believe" that there is another row. To avoid detecting false rows a minimal row height is defined.

After having found the rows statistics are made on the space sizes between characters in each one. There will be a "jump" in the ordered vector of space sizes between the space size of letters and words. If the space is less than the space at the jump, it will separate the characters, otherwise it will separate two words. This allows us not to use pregiven parameters, the program will be more flexible.

The characters are normalized before feature extraction. They are placed into a 40 × 40 matrix. The features are examined in this matrix. Above the enclosing rectangle of the character, accents are searched for. The existence and the number of them is one of the features.

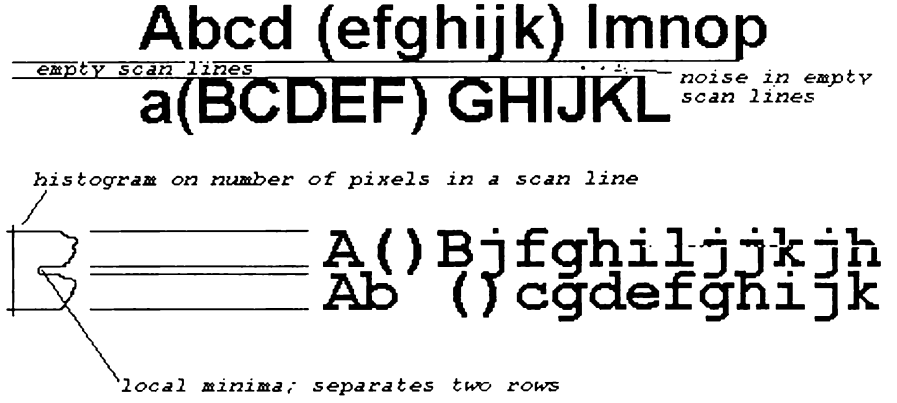

Fig.2. Separating two rows

2.3. Recognition

Our program applies four, quite well-known, well tested algorithms.

The first one is to scan the letters horizontally, and the number of changes from background to foreground are stored. Then scan is done vertically (see Fig.3.).

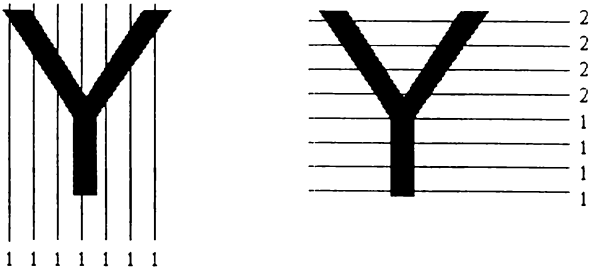

Fig.3. The slicing method

The second method determines the number of "bubbles" in the character. "8" and "B" have two, "o", "A" and "d" have one bubble. "I" and "m" have none. The number of bubbles is never used as a feature alone. But it means some additional information.

The third method is contour following. The outer shape of a letter is important information. Our program performs segmentation of the outer contour and stores the endpoints of the segments. The segments are approximated with straight lines (see Fig.4.).

Fig.4. Contour following

The last method is based on the idea that not only the outer shape but also the placement of the spot of the character renders important information. In case of a letter "I" more than the half of the black pixels are at the top of the spot. Dividing the matrix into not two, but many small parts, the proportion of the black pixels in a cell will be a relevant feature. We divided the matrix of the letter into 25 cells and the proportion of the black and white pixels were determined in each cell.

2.4. The neural network

Our aim is to detect different "k" (or any) letters belonging to the same class, not making difference between k, **k** or *k*.

After the features have been extracted, they will be evaluated. In general, the features will compose a vector and the learnt patterns are compared to the actual vector. Comparison of the feature by their Euclidian (or some similar) distance is too simple: it is difficult to decide which feature should be considered more important than the other in determining the letter.

Blackboard systems [9] are an effective solution. Their principle is that different algorithms result in different solutions to a problem. In most cases each of them has different knowledge about the given problem. An "inspector" program should see their results and should give different score to the algorithms depending on their effectiveness. Each piece of information is used

corresponding to its importance. It can be achieved with the score system, but not exclusively with it. The parts of the system can be replaced or new parts can be added to it, making the system more general and flexible.

A blackboard system needs rather complicated programming. Instead of this we used a neural network to learn the effectiveness of the algorithm. The input of the network is the feature vector and the output is the letter. After many iterations it learns the letters and attaches proper importance to the features provided by different sources.

A Kohonen network was used for this purpose, but it is not a self-organizing one ([7], [8]). There are minor differences between ours and the usual one. We applied supervised learning to increase the speed. Instead of computing the squares of the differences between weights and new vectors simple differences were taken into account. The input vector is composed of binary values of the features. Gray code also was tried but the effectiveness of the program did not improve.

2.5. Knowledge

The final step in the recognition is applying a database which contains all the words that are to be recognized. Sometimes, if characters are not recognized with sufficient certainty, the words will still be able to be identified because of the possibility of seeking in a database using wildchard characters. The search is very simple, we use the UNIX command *grep*. This database can be modified at any time. The program is also able to learn unknown characters. This needs ability of the interaction with the user. A simple user interface was written in *Xlib* to communicate.

3. Summary, conclusions

Our experiences show that the program learns the characters very fast. Figure 5 shows the result of recognition after only two learning cycles. Initially the network is "empty", does not know any letters. During the teaching the program scans the sample file (containing Times Roman letters) and asks the meaning of the unknown spots. Then it updates the weights of the neural network three times. After two such iterations (six updates) the program is tested with the sample file. The result is shown in Fig.5. It is easy to notice that the network mixes capital and small letters if they have the same shape. The reason of this is that the feature vector is very large and the information

about the size of letters does not carry enough weight. Increasing the number of elements in the feature vector about size this problem can be eliminated.

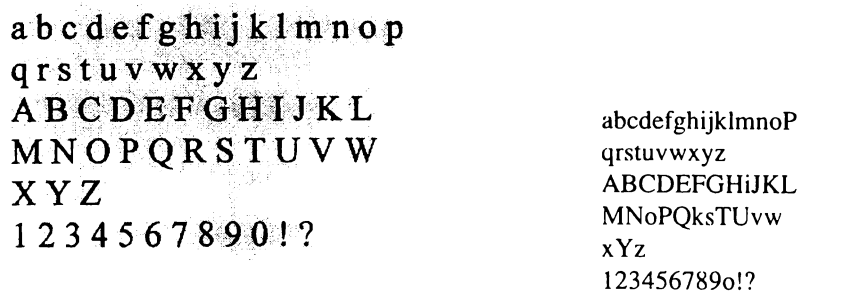

Fig.5. Results after a few learning iterations

Otherwise, only the letter "R" is misrecognized. After many learning cycles the neural network learnt the letters. The test with other character sets e.g. Courier or Serif) shows that the recognition is quite independent of the type of the fonts. After five iterations the program classifies the characters by a 10 percent probability of error.

This is the ideal case. The recognition is worse in case of a text scanned from a newspaper. The use of the word database is great help in the case of mixing similar characters. ("G" and "C", "E" and "F" are difficult to recognize properly.)

A possible direction of developing the program is improving the services. A line detection part would be desirable since even books of classical literature contain straight lines or simple drawings.

References

[1] **Álló G., Foglein J., Hegedüs Gy.Cs. and Szabó J.,** *Bevezetés a számítógépes képfeldolgozásba,* BME Mérnöktovábbképző Intézet, Budapest, 1993.

[2] **Parker J.R.,** *Practical computer vision using C,* Wiley & Sons, 1993.

[3] **Masuda I. et al.,** Approach to smart document reader system, *Proc. CVPR'85, San Francisco, 1985,* 600-686.

[4] **Shimotsuji S.,** A robust drawing recognition system based on contour shape analysis, *Proc. CVPR'85, San Francisco, 1985,* 717-719.

[5] **Marosi I. and Kovács E.,** Trends, applications, devices in optical character recognition, *Proc. of Conference on Intelligent Systems, Veszprém, Hungary, 1991.*

[6] **Csink L.,** Technical document recognition algorithms, *Proc. of Conference on Intelligent Systems, Veszprém, Hungary, 1991.*

[7] **Müller B. and Reinhardt J.,** *Neural networks. An introduction,* Springer, 1990.

[8] **Beale R. and Jackson T.,** *Neural computing. An introduction,* Institute of Physics Publishing Ltd., Bristol, 1994.

[9] **Corkill D.,** Blackboard systems, AI Expert, 1991.

[10] **Downto A.C., Tregidgo R.W.S. and Kabir E.,** Recognition and verification of handwritten and handprinted British postal addresses, *International Journal of Pattern Recognition and Artificial Intelligence,* **5** (1-2) (1991).

[11] **Shridhar M. and Badreldin A.,** A tree classification algorithm for handwritten character recognition, *Proc. 7th ICPR, Montreal, 1984, vol.1,* 615-618.

[12] **Niemann H.,** *Pattern analysis and understanding,* Springer, 1991.

J. Kőműves, L. Csink, Zs. Cseresznye,
L. Kőrösi and M. Puskely
Institute of Mathematics and Computing
Kandó Kálmán Polytechnic
H-1431 Budapest, Pf. 112.
Hungary
jozsi@miws.ipan.sztaki.hu
csink@novserv.obuda.kando.hu