ANALOGIC CNN ALGORITHMS FOR SOME IMAGE COMPRESSION AND RESTORATION TASKS

by

Péter L. Venetianer, Frank Werblin, Tamás Roska, and Leon O. Chua

Memorandum No. UCB/ERL M94/30

28 February 1994

COVER PAGE

ANALOGIC CNN ALGORITHMS FOR SOME IMAGE COMPRESSION AND RESTORATION TASKS

by

Péter L. Venetianer, Frank Werblin, Tamás Roska,
and Leon O. Chua

Memorandum No. UCB/ERL M94/30

28 February 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ANALOGIC CNN ALGORITHMS FOR SOME IMAGE COMPRESSION AND RESTORATION TASKS

by

Péter L. Venetianer, Frank Werblin, Tamás Roska,
and Leon O. Chua

Memorandum No. UCB/ERL M94/30

28 February 1994

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ANALOGIC CNN ALGORITHMS FOR SOME IMAGE COMPRESSION AND RESTORATION TASKS

Péter L. Venetianer[†], Frank Werblin[‡], Tamás Roska[†] and Leon O. Chua,

Electronic Research Laboratory, U.C. Berkeley

[†] Computer and Automation Institute, Hungarian Academy of Sciences
[‡] Molecular Cell Biology, U.C. Berkeley

February 28, 1994

1 INTRODUCTION

In this report we present analogic cellular neural network (CNN) algorithms for real-time solutions of several important image processing tasks; namely, skeletonization, scratch removal and image compression. Our main concern was to present solutions which can be easily realized on the CNN Universal Machines (CNNUM) [1, 2] manufactured with todays technologies, so we restricted ourselves to single-layer 3x3 templates.

Many image processing and pattern recognition applications are based on the skeletonization of the image. Several CNN skeletonizatio solutions have already been published, but all of them use complex multi-layer, sometimes even non-uniform architectures. In contrast, our solution uses only linear single-layer 3x3 templates. Our algorithm can be further generalized to skeletonize grey-scale images as well.

In copying machines the glass panel often gets scratched after extensive usage. This scratch is of course copied as well, resulting in a visually annoying copy. Thus it is very important to remove the scratches from the copied material, i.e. to fill them with values of the neighboring pixels so as to make them invisible. Our solution works both for grey-scale and color images.

The main idea of our scratch removal algorithm can also in other appllications. Here we will show how it can be applied to enlarge images without a significant loss of quality. This is equivalent to compressing and decompressing images.

In the next Section we describe the skeletonization algorithm in details. In Section 3 we present scratch removal algorithm and use it in Section 4 to enlarge images.

2 SKELETONIZATION OF BLACK-AND-WHITE AND GREY-SCALE IMAGES

The problem of skeletonization is a very important task. For example it is used with the solution of several image processing and pattern recognition problems, such as character recognition. The problem of skeletonization using CNN has already been attacked in several papers [3, 4, 5], but all of the solutions presented so far use complex multi-layer, sometimes even non-uniform CNN architectures. Using current VLSI technologies it is difficult to implement a CNNUM chip on which these solutions could be realized. For this reason it is very important to find a solution which is not just theoretically correct, but can be realized on a chip, too. Here we present two analogic CNN algorithms, based on the idea of [6], which meet this requirement. Finding the skeleton of black-and-white objects is solved using single-layer, linear, 3x3 templates, making it ideally suited for today's CNNUM architectures. The algorithm can be further generalized [6] to extract the skeleton of grey scale objects using single-layer, 3x3 templates with simple nonlinearities.

2.1 Defining the Skeleton of Objects

Defining the skeleton of objects is ambiguous, because there are several possible solutions for a given input image. Here we define the skeleton of an object as "a stick figure, with each picture cell connected to two neighbors, except for the ones at the end of the stick and the branch points where sticks are connected together" [7], thereby preserving the connectivity and the shape of the objects. In our algorithm 8-connectivity is used.

2.2 The Black-and-White Skeletonization Algorithm

2.2.1 How does the Algorithm Work?

The idea behind the solution of the skeletonization problem is as follows: in each step we peel off black pixels having three white and two black neighbors in appropriate position. This means, that the white neighbors are 4-connected with each other, thus being in a line or an L shape, while the 2 black neighbors are not connected to the white ones, but 8-connected to each other. To picture this refer to the template definitions of the next subsection, where +0.25-s and -0.25-s in the control templates correspond to the white and black neighbors, respectively. As the algorithm peels off pixels from the object cyclically from all directions, but no more peeling is done if the pattern is just one pixel wide, it is clear, that it preserves the general shape of the object. The connectivity preserving property can also be proved easily: the 3 white neighbors guarantee, that the pixel is on the edge of the object, so removing it doesn't change the shape of the object, e.g. by creating a hole; the 2 black pixels are connected with all three unchecked neighbors (0-s in the control template) and with each other, so they preserve the connectivity of the remaining parts even if the actual cell is deleted.

2.2.2 The Algorithm

The skeletonization algorithm uses 8 steps, peeling off pixels circularly from the object, i.e. the first step peels off north-western corners, the second step peels off northern pixels, etc. Such a step is a logic decision, deleting black pixels having 3 white and 2 black neighbors corresponding to the locations of +0.25-s and -0.25-s in the templates, respectively. Executing all 8 steps peel off one layer of pixels from the object, which means that this procedure should be executed several times to reach our goal, the 1 pixel wide lines. An example is shown in Fig. 1. The 8 templates of the algorithm are as follows:

$$A_1 = [3] \qquad B_1 = \begin{bmatrix} 0.25 & 0.25 & 0 \\ 0.25 & -0.25 & -0.25 \\ 0 & -0.25 & 0 \end{bmatrix} \qquad I_1 = -0.75 \tag{1}$$

$$A_2 = [3] \qquad B_2 = \begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0 & -0.25 & 0 \\ -0.25 & -0.25 & 0 \end{bmatrix} \qquad I_2 = -0.75 \tag{2}$$

$$A_3 = [3] \qquad B_3 = \begin{bmatrix} 0 & 0.25 & 0.25 \\ -0.25 & -0.25 & 0.25 \\ 0 & -0.25 & 0 \end{bmatrix} \qquad I_3 = -0.75 \tag{3}$$

$$A_4 = [3] \qquad B_4 = \begin{bmatrix} -0.25 & 0 & 0.25 \\ -0.25 & -0.25 & 0.25 \\ 0 & 0 & 0.25 \end{bmatrix} \qquad I_4 = -0.75 \tag{4}$$

$$A_5 = [3] \qquad B_5 = \begin{bmatrix} 0 & -0.25 & 0 \\ -0.25 & -0.25 & 0.25 \\ 0 & 0.25 & 0.25 \end{bmatrix} \qquad I_5 = -0.75 \tag{5}$$

$$A_6 = [3] \qquad B_6 = \begin{bmatrix} 0 & -0.25 & -0.25 \\ 0 & -0.25 & 0 \\ 0.25 & 0.25 & 0.25 \end{bmatrix} \qquad I_6 = -0.75 \tag{6}$$

$$A_7 = [3] \qquad B_7 = \begin{bmatrix} 0 & -0.25 & 0 \\ 0.25 & -0.25 & -0.25 \\ 0.25 & 0.25 & 0 \end{bmatrix} \qquad I_7 = -0.75 \tag{7}$$

$$A_8 = [3] \qquad B_8 = \begin{bmatrix} 0.25 & 0 & 0 \\ 0.25 & -0.25 & -0.25 \\ 0.25 & 0 & -0.25 \end{bmatrix} \qquad I_8 = -0.75 \tag{8}$$

Looking at the templates it is obvious that we could have used their mirrored counterparts, moreover, we could have combined these templates with their mirrored pairs, resulting in better quality output, but also in slower execution, because then one layer is peeled off in 16 steps instead of 8.

It is also important to be able to decide, when to terminate the process. This can most easily be done by storing the actual image before the cycle, and comparing it to the result of the cycle. If the two images are equal, then the process should terminate. This condition can be tested by applying the logic XOR function to the two images, and then checking whether there are any black pixels left on the screen. The capability of checking the existence of a black pixel is part of the definition of the universal machine. Note, that all 8 steps should be executed before the comparison, because the fact that 1 of the 8 steps couldn't peel off anything doesn't mean, that there are no pixels left to be peeled.

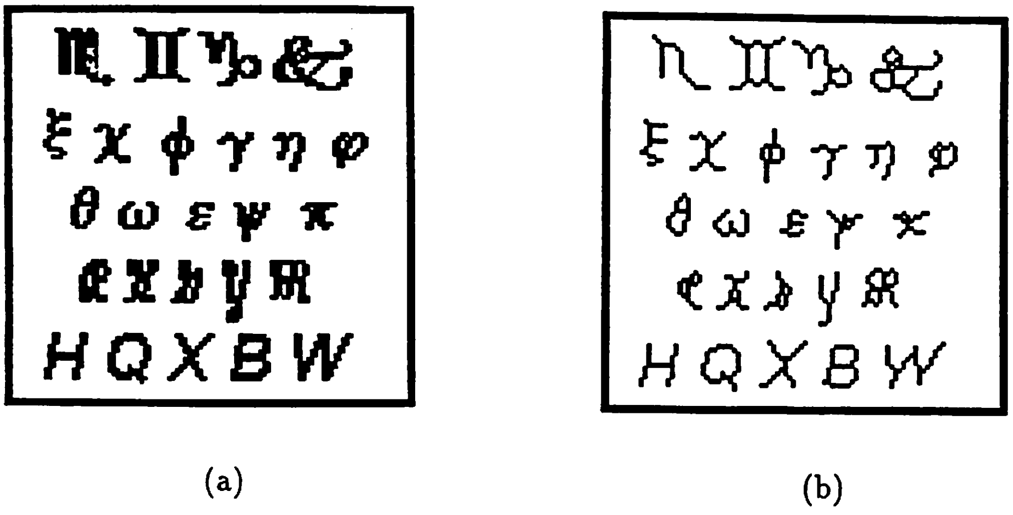


Figure 1: Skeletonizing a black-and-white image: (a) input, (b) output

2.2.3 Some remarks on the algorithm

The templates presented in the previous subsection were made supposing a continuous-time CNN architecture, but slightly modifying the template values, the algorithm can be implemented on a discrete-time (DTCNN) [8], as well.

As the algorithm peels off pixels layer by layer, the running time is proportional to the size (width) of the objects, but it is independent from the image size and the number of objects on the screen. We give estimates for the time requirements in τ time constants, the exact value of which falls into the 10–300ns range, depending on the CNNUM chip being used.

One cycle contains 8 peeling steps, each of them requiring 1τ on the DTCNN and $1-2\tau$ on the continuous time CNN (it is faster if the control template and current values can be increased).

This means, that one cycle takes $8-16\tau$, depending on the underlying architecture. In our example (see Fig. 1) 3 such cycles were needed.

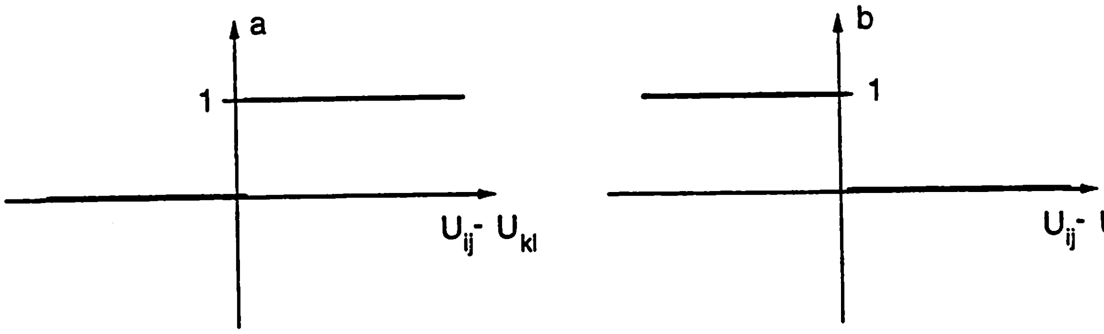
2.3 Grey-scale Skeletonization Algorithm

2.3.1 The Algorithm

Some objects have a fine structure, which would be obscured by converting it to a black-and-white image. If we want to skeletonize such an object, preserving the fine structure, we should do grey-scale skeletonization. Fortunately, our black-and-white algorithm can easily be adapted to grey-scale [6]. In the former case the condition of changing the value of a pixel was to have 3 white and 2 black pixels in an appropriate position; now we will be looking for 3 smaller and 2 greater neighbors. There changing the value of a pixel meant turning it into white; here it means taking the value of the greatest of its 3 smaller neighbors. Similarly to the black-and-white case, this algorithm also uses 8 steps to peel off a layer of pixels, but now one step contains more elementary operations. We will show only the templates of 1 of the 8 steps, the other seven can be easily determined.

The first elementary operation is selecting the cells to be modified using template (9) and its variants, which can be generated using the black-and-white algorithm as a reference, replacing the elements of the control template and the current value of templates (1)-(8) as when switching from template (1) to template (9):

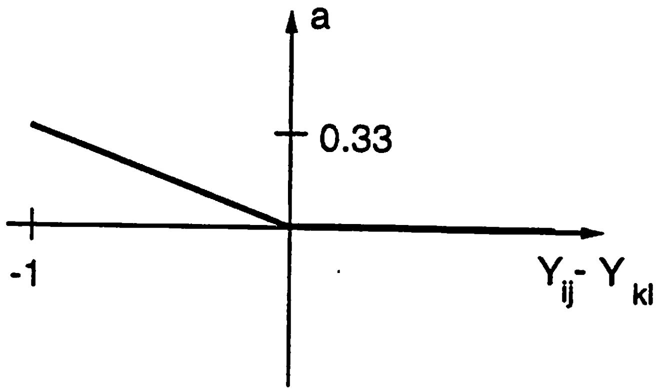
$$A = [1] \qquad \hat{B} = \begin{bmatrix} a & a & 0 \\ a & 0 & b \\ 0 & b & 0 \end{bmatrix} \qquad I = -4.5 \tag{9}$$



In this template, function a and b are sensitive to pixels smaller and larger than the actual one, respectively.

The result of this step is used as a fixed state mask in the second elementary step, where the selected pixels are replaced by the greatest of their 3 smaller neighbors. Here we again show only 1 of the 8 templates, the other 7 can be achieved by modifying template (10), having function a at the same position as in template (9).

$$\tilde{A} = \begin{bmatrix} a & a & 0 \\ a & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (10)$$



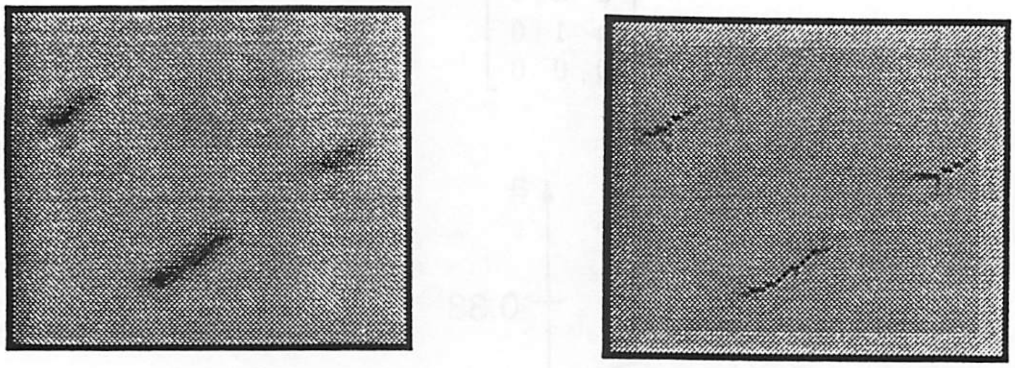
An example demonstrating the behavior of the algorithm is given in Fig. 2.

The condition of termination, just like in the black-and-white case. is the equivalence of the input and the output of a cycle, but as both are grey-scale, the logic functions cannot be applied to them. Instead, at the beginning of the cycle, a logic memory should be driven to white, and the result of each elementary selection step (template (9)) should be added to it with the logic OR function. This image accumulates the pixels where changes occured, so if it contains no black pixels at the end of a cycle, the process should terminate.

2.3.2 Some remarks on the algorithm

It follows from the way how the black-and-white algorithm was transformed to grey-scale, that the algorithm assumes that the objects are darker than the background. Also note, that not just the input, but the output is grey-scale, too. For this reason, it cannot be implemented on a DTCNN, which always results in a binary output.

As to the time demands of the algorithm: a step means first selecting the cells to be changed, then generating a fixed state mask from the selected cells and finally replacing these cells by the greatest of their smallest neighbors. There is an additional logic step to keep track of changes, that is to now if there is anything left to do. Altogether a step requires approximately 7τ, which means that a whole cycle is $\sim 56\tau$



(a) (b)

Figure 2: Skeletonizing a grey-scale image: (a) input, (b) output.

3 RESTORATION OF IMAGES WITH SCRATCHES

In copying machines the glass panel often gets scratched after extensive usage. This scratch is copied together with the material, resulting in the decrease of the quality of copying. Thus it is very important to remove the scratch from the copied material. To be able to do this we have to know where the scratches are, because otherwise we would modify the whole image, not just the corrupted part, what is not what we want. Fortunately, it is easy to locate the scratches, e.g. by copying a blank sheet and testing where we have something on the copy. So now we are ready to state the problem precisely:

> Given two images, one containing only the scratches, the other being the corrupted copied image, create a third image which is identical to the corrupted one where there are no scratches, and the scratches are filled with values close to the original ones.

3.1 The Algorithm

The algorithm described in this section can remove scratches of known locations from both *grey-scale* and color images. The same procedure applies to both cases, but with color images the algorithm should be run on all three color channels independently.

3.1.1 The concept of the algorithm

The basic idea is that in each step we deal only with those pixels which have the most known neighbors (pixels not being part of the scratch) and we fill in these pixels by the average of their known neighbors. According to our experiments, first dealing with pixels having 4 known neighbors for a few steps, and then switching to the ones with 3 known neigbors, filling in the whole scratched area this way, gives good results. Using the CNNUM one such step, e.g. filling pixels with 3 known neighbors, is done in several steps: we have to deal separately with the different combinations of known neighbors. This means 8 phases, in each phase knowing only the 3 Northern. North-Eastern, Eastern, etc. neighbors, respectively, thus filling the outermost layer of the scratch circularly.

3.1.2 The details of the algorithm

Each of the 8 phases contain 3 elementary steps:

- Select pixels of the scratch to be filled
- Fill the selected pixels
- Remove the selected and filled pixels from the scratch

The first two elementary steps both contain 8 different templates for the 8 phases, but here we will give only one of them, the others can easily be generated by rotating the non-center values.

Equations (11) and (12) show the template of selecting pixels having 3 and 4 neighbors. respectively. in the appropriate (north-western) direction:

$$A = [1] \qquad B = \begin{bmatrix} -0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad I = -1.5 \tag{11}$$

$$A = [1] \qquad B = \begin{bmatrix} -0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 \\ -0.5 & 0 & 0 \end{bmatrix} \qquad I = -2 \tag{12}$$

Equations (13) and (14) show the template of filling in pixels with the average of 3 and 4 neighbors, respectively. The output of the selection operation should be used as a fixed state mask to ensure that only the selected pixels will be modified.

$$A = [0] \qquad B = \begin{bmatrix} 0.34 & 0.33 & 0 \\ 0.33 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad I = 0 \tag{13}$$

$$A = [0] \qquad B = \begin{bmatrix} 0.25 & 0.25 & 0 \\ 0.25 & 0 & 0 \\ 0.25 & 0 & 0 \end{bmatrix} \qquad I = 0 \tag{14}$$

The third elementary step, deleting the selected pixels from the scratch, can be executed with the logic XOR function. If after this step the scratch totally dissapears, the process can terminate as the whole scratch is filled.

3.2 Limitations and capabilities

This algorithm doesn't rely on a database, from which we could reconstruct objects which are almost totally scratched out from the screen, but it does a kind of extrapolation based on the pixels surrounding the scratch. This means that we cannot expect to restore more information than what is coded in the non-scrathced cells, e.g. if a scratch totally covers the pupil of the eye, even the restored image won't contain the pupil.

The quality of the restored image depends on two factors: the width of the scratch and the image itself, because even a large portion of a homogeneous area can be restored reliably, while a highly textured area cannot.

The time demand of the algorithm depends on the size of the scratch. One step, i.e. filling from the 8 different directions, requires $24\tau s$. As such a cycle fills in a layer of pixels on both sides of the edge, the algorithm requires $\frac{width}{2} * 24\tau s$.

In Figure 3 an example containing the original, the scratched, and the restored image is shown, while in Figure 4 the steps of filling a blown up part of the image are presented, demonstrating the circular nature of the algorithm.

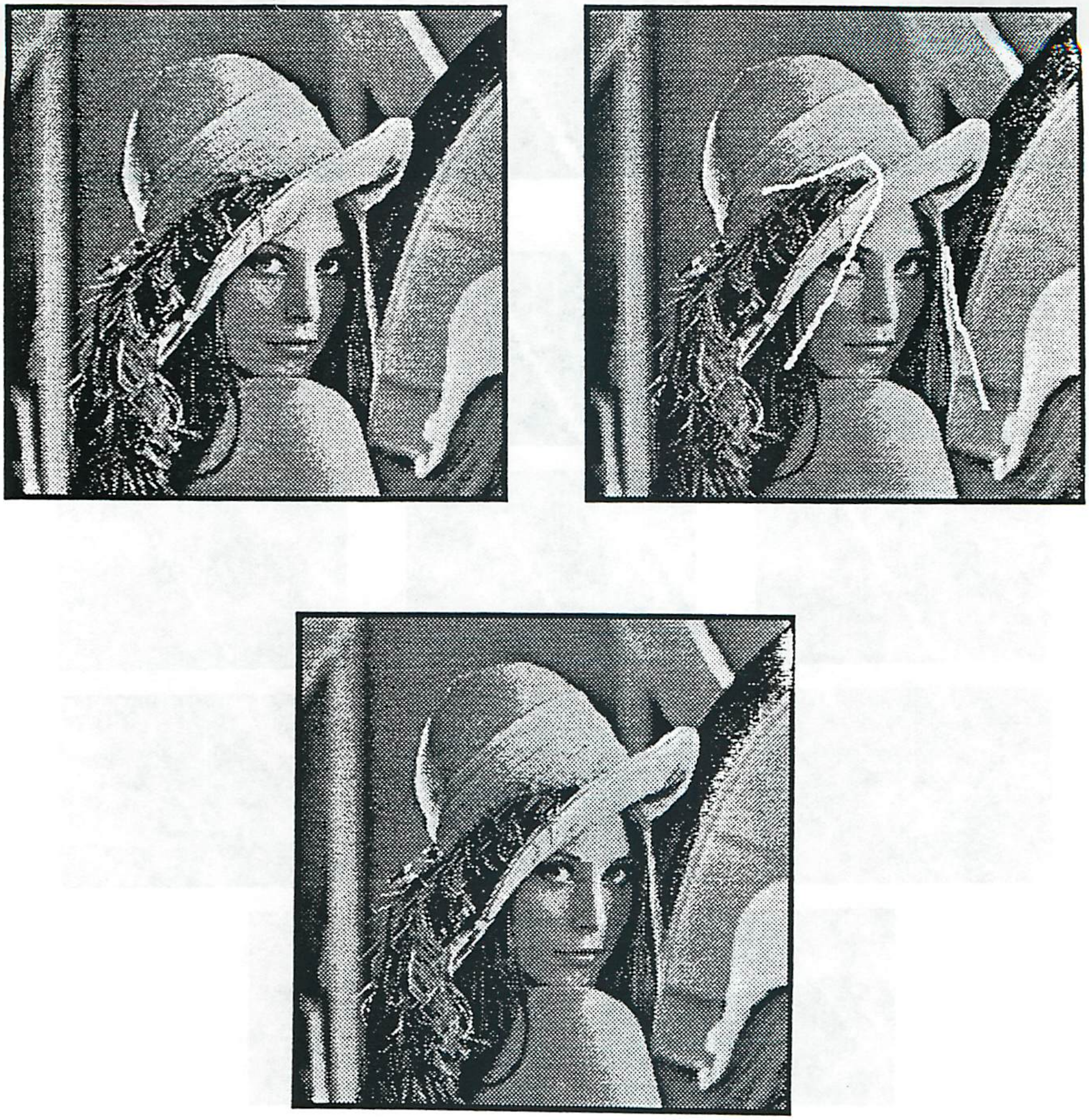


Figure 3: Removing scratches from Lena image: original, scratched and restored image

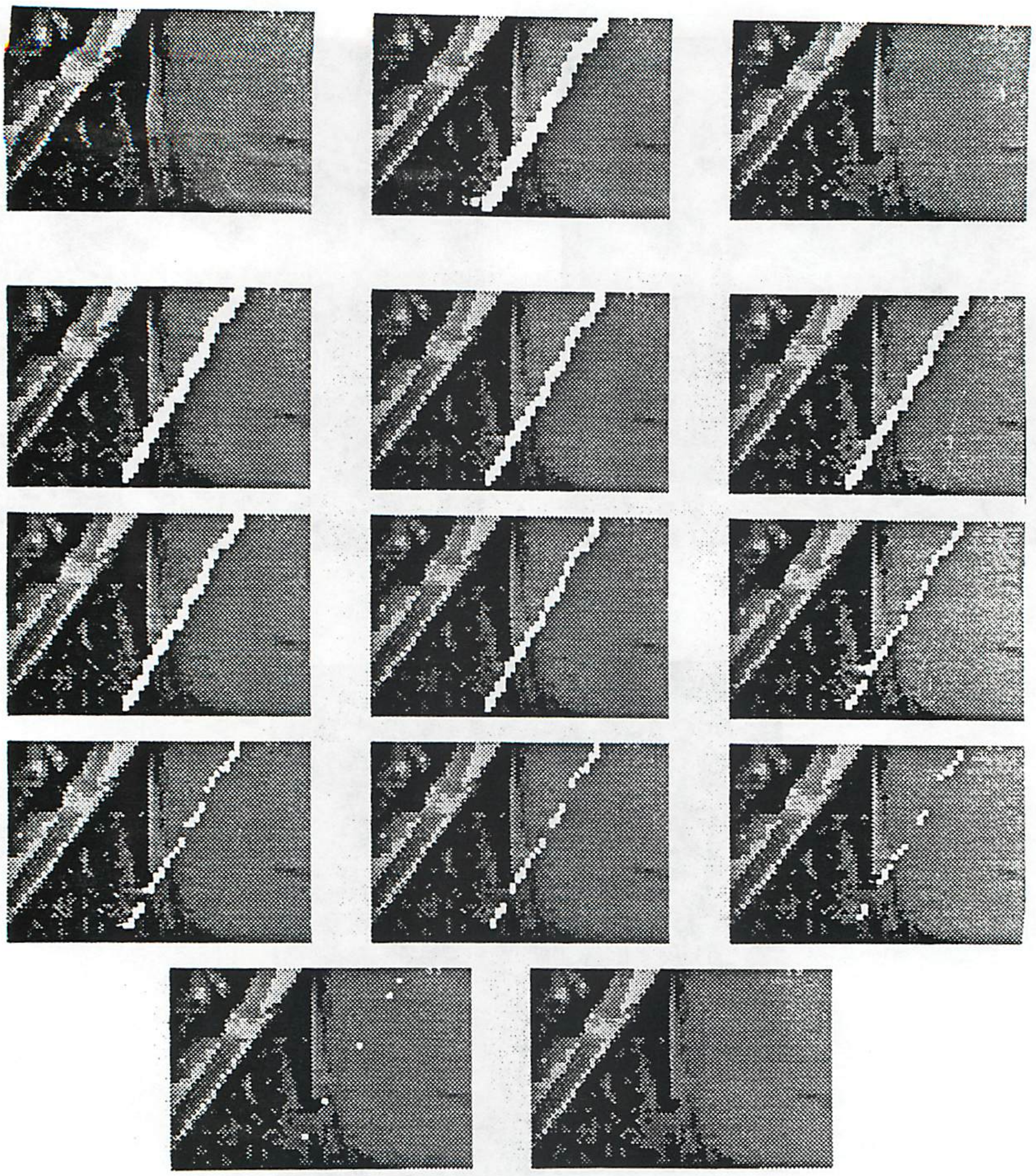


Figure 4: The steps of restoring images on a blown up part of the Lena image. The original. the scratched and the restored image in the first row, followed by the steps of restoring.

4 SMOOTHING ENLARGED IMAGES

The same basic idea, that is, filling in some locations by the average of their known neighbors can be used in some other places: we can increase the image size or the resolution of images using our algorithm. What we do is put the pixels of an image on every second pixel of a four times bigger one (doubling the number of both rows and columns) and fill in the remaining pixels by the average of the known ones. See Figure 5a, where the black squares represent the known pixels (every second pixel both horizontally and vertically), and the white ones have to be filled. First the squares with the *1* are filled by their 4 known neighbors (Figure 5b) using the template

$$A = [0] \qquad B = \begin{bmatrix} 0.25 & 0 & 0.25 \\ 0 & 0 & 0 \\ 0.25 & 0 & 0.25 \end{bmatrix} \qquad I = 0$$

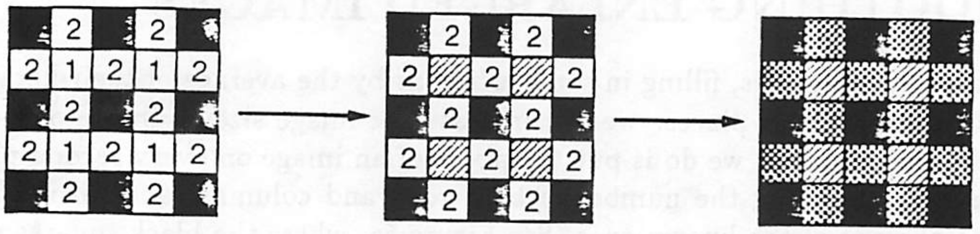
then the remaining pixels (with the *2* in them) are filled, also from the 4 known neighbors (Figure 5c) using the template

$$A = [0] \qquad B = \begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix} \qquad I = 0$$

This method can be used e.g. to enhance the quality when displaying images, for example in case of a TV set, not changing the number of transmitted pixels, we can increase the resolution of the displayed frames; the procedure can also be used for coding images: only $\frac{1}{4}$th of the pixels are transmitted or stored and finally the image can be restored with hardly worse quality. Figure 6a shows an original input image, enlarged with (Figure 6b) and without (Figure 6c) using our algorithm, then a part of these images are again enlarged two more times with (Figure 6d and f) and without (Figure 6e and g) using our algorithm. The difference is striking on the last pair (Figure 6f and g) but it is already big on the first pair (Figure 6b and c) especially at the shoulder of Lena.

5 CONCLUSIONS

In this paper we presented analogic CNN universal machine algorithms solving some important image processing tasks using only 3x3 single-layer templates and logic. These extremely fast procedures can serve as the core of complex image processing systems in a wide variety of applications.



(a) (b) (c)

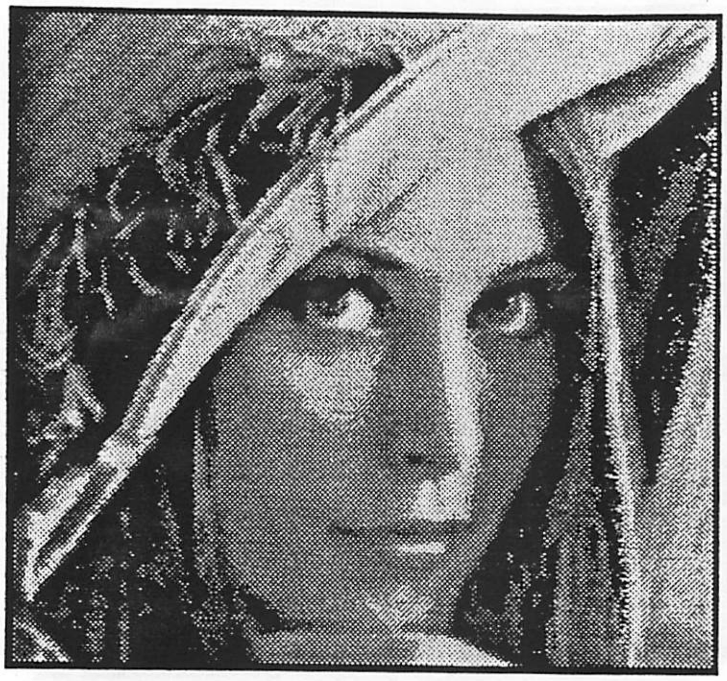
Figure 5: The steps of enlarging images. (a) Image with pixels at every second location. (b) Filling in the 1 pixels with their 4 known neighbors. (c) Filling in the 2 pixels with their 4 known neighbors.

(a)

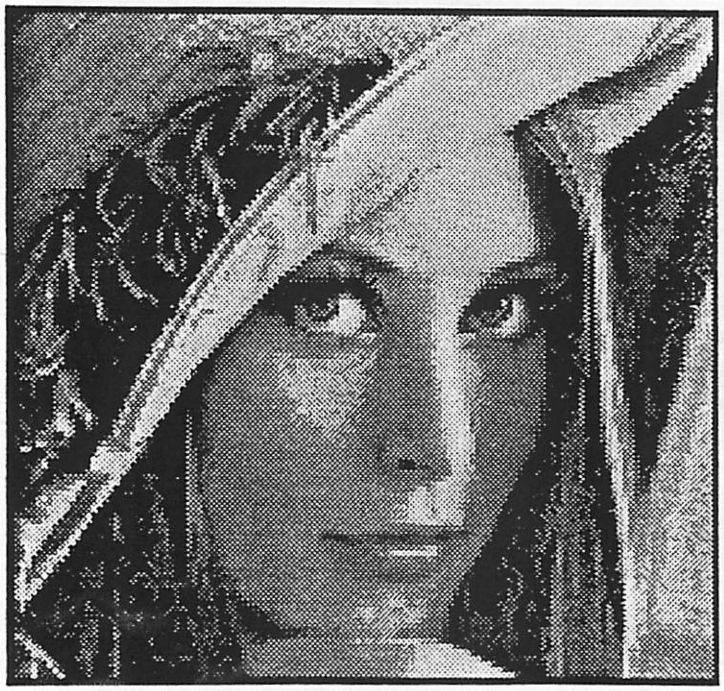
Figure 6: Enlarging images. (a) Original input image. (b,c) Enlarging (a) with and without using our algorithm. (d,e) Enlarging a part of (b,c) with and without using our algorithm. (f,g) Enlarging a part of (d,e) with and without using our algorithm.

(b)

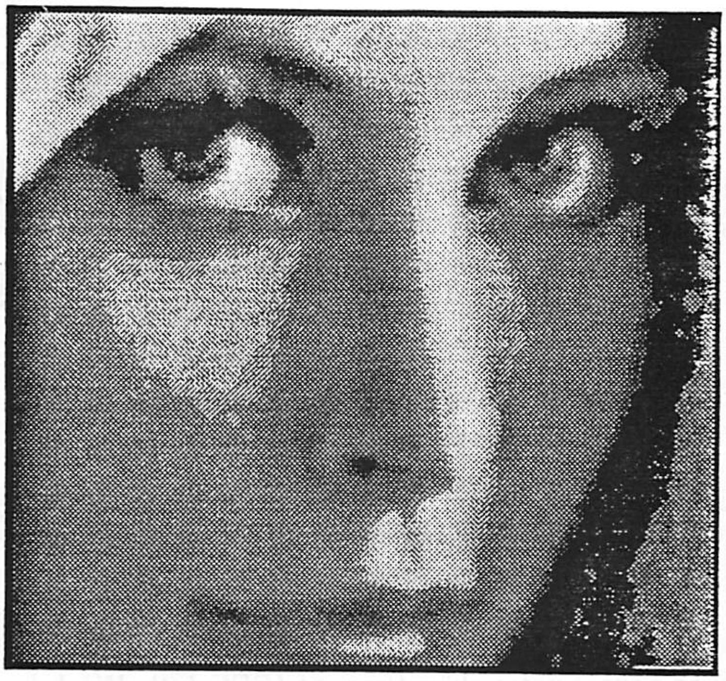
(c)

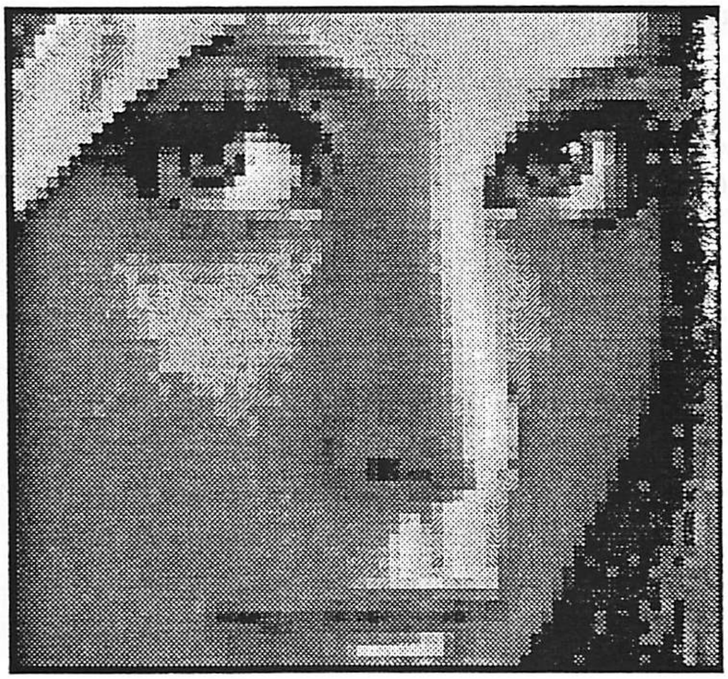


(d)



(e)


(f)


(g)

References

[1] Leon O. Chua and Lin Yang. "Cellular Neural Networks: Theory and Applications". *IEEE Trans. on Circuits and Systems*, 35:1257–1290, 1988.

[2] Tamás Roska and Leon O. Chua. "The CNN Universal Machine: Analogic Array Computer". *IEEE Trans. on Circuits and Systems-I*, 40:163–173, 1993.

[3] T. Matsumoto and al. "Several Image Processing Examples by CNN". *Proc. of IEEE Intl. Workshop on Cellular Neural Networks and Their Applications*, pages 100–111, 1990.

[4] Hubert Harrer and Joseph A. Nossek. "Skeletonization: a new application for discrete-time cellular neural networks using time-variant templates". *Proc. of IEEE Intl. Symposium on Circuits and Systems*, pages 2897–2900, 1992.

[5] D. Yu, C. Ho, X. Yu, and S Mori. "On the application of cellular automata to image thinning with cellular neural network". *Proc. of IEEE Intl. Workshop on Cellular Neural Networks and Their Applications*, pages 210–215, 1992.

[6] Kendall Preston, Jr. and Michael J.B. Duff. *"Modern cellular automata : theory and applications"*. New York : Plenum Press, 1984.

[7] Berthold K. P. Horn. *"Robot vision"*. New York : McGraw-Hill, 1986.

[8] Hubert Harrer and Joseph A. Nossek. "Discrete-time cellular neural networks". *International Journal of Circuit Theory and Applications*, 20(5):453–467, 1992.