FAST AND ACCURATE BINARY HALFTONE IMAGE RESOLUTION INCREASING BY DECISION-TREE LEARNING

Hae Yong Kim
Dept. Eng. Sistemas Eletrônicos, Univ. de São Paulo.
Av. Prof. Luciano Gualberto, trav. 3, 158; CEP 05508-900, São Paulo, SP, Brazil.
E-mail: hae@lps.usp.br

ABSTRACT

Digital halftone is the technique used to convert gray-scale images into binary ones, simulating gray shades by scattering appropriately black and white pixels. Sometimes, there arises the necessity of increasing the resolution of a halftone image. Some recent works have proposed a number of learning-based techniques to zoom binary images. However, they cannot consider a large neighborhood to decide the colors of the resolution-increased pixels, because their running time skyrockets with the growth of the window and the sample images sizes. The use of large window and samples are required to accurately zoom halftone images. This paper presents a new technique to zoom quickly and precisely images generated by any locally-decided halftone algorithm. It is based on the decision-tree learning and it is very fast, even using a large window or large samples. The zoomed images obtained by this technique are incredibly sharp and accurate.

1. INTRODUCTION

Most of ink-jet and laser printers used today cannot actually print gray scales. Instead, they are suited for printing only tiny black dots in paper (in this paper, color devices will not be considered.) Thus, any gray-scale image have to be first converted into a binary image by a halftone process before the printing actually takes place. A halftone technique simulates gray shades by scattering appropriately black and white pixels. That is, given a gray-scale image $G:\mathbb{Z}^2 \rightarrow [0,1]$, the halftone generates a binary image $B:\mathbb{Z}^2 \rightarrow \{0,1\}$ such that for any l and c:

$$\overline{B}(l,c) \cong G(l,c)\ ,$$

where $\overline{B}(l,c)$ is the average value of the image B in a neighborhood around the pixel (l, c).

There is an enormous variety of halftone techniques. The two most widely known are the error diffusion and the ordered dither [1]. Many other techniques have derived from them, for example, clustered-dot orthered dither, dot diffusion and space filling curves. Some of them are designed specifically for laser printers, for a laser printer is unable to print isolated dots or too finely interspersed black and white dots.

In a typical office environment, frequently arises the necessity of resolution conversion of a digital document. For example, a digital document originally intended to be impressed in a 300 dpi device often has to be printed in a 600 dpi printer. Many different spatial resolution conversion algorithms have so long been developed for gray-scale and color images. However, binary image resolution increasing algorithms have only recently been emerged [2, 3, 4]. They are all based on the machine learning, that is, the statistically optimal zooming scheme is constructed starting from sample images. Unfortunately, these algorithms cannot consider a large neighborhood to decide the colors of the resolution-increased pixels, because their running time and needed memory skyrocket with the growth of window and sample size. A small window is quite good to zoom impressed or handwritten characters, but it cannot zoom accurately halftone images. Our experimental data have shown that windows as large as 7×7 or 9×9 are needed to accurately zoom halftone images.

This paper improves the previous learning-based binary image resolution increasing algorithms so that even a halftone image can be accurately zoomed. The new technique is based on the decision-tree. Its training time is $O(wm \log m)$, where w is the window size and m the sample input-image size. Its application time is $O(n \log m)$, where n is the size of the image to-be-zoomed. This means that the performance of the new technique deteriorates only very slowly as the window and sample size increase. Unfortunately, the works [2, 3] present neither the processing time nor the complexity analysis. But we deem that, when large window and sample images are used, the new technique is thousands times faster than [2, 3] (both in training and in application), employing a conventional (non-parallel) computer. This estimate is based on the computational complexity analysis. Morever, the works [2, 3] do not assume any explicit inductive bias, while the decision-tree has a nice generalization scheme. The new technique is also thousands times faster than [4] in the training stage, though slightly slower in the application.

One remark: the new technique is unable to accurately zoom images generated by the error diffusion (or by any halftone algorithm where the output color is not locally decided.) Though, when it is applied to zoom a locally-decided halftone image using a large window, the resulting zoomed image presents an unexpected remarkably nice quality (see figures 2, 3 and 4.) Certainly, the new technique can zoom printed or handwritten characters as well.

This paper will be concerned only with the binary zooming by integer factors. Moreover, to simplify the notation, we will assume that the column and row zoom factors are equal.

2. WINDOWED ZOOM OPERATOR LEARNING

A binary image is a Boolean function $Q:\mathbb{Z}^2 \rightarrow \{0,1\}$. The support of an image Q is a finite subset of $\mathbb{Z}^2$ where the image is actually defined. Out of support, an image is supposed to be filled with a background color.

Let f be the zoom factor. A windowed zoom operator (WZ-operator) Ψ is an image transformation defined via a window:

$$W = \{W_1, \ldots, W_w\},\ W_i \in \mathbb{Z}^2,$$

and a characteristic function:

$$\psi: \{0,1\}^w \to \{0,1\}^{(f^2)}.$$

Each element W_i of the window is called a peephole and the function ψ is actually a set of f^2 Boolean functions:

$$\psi = \{\psi_1, \ldots, \psi_{f^2}\},\ \psi_i: \{0,1\}^w \to \{0,1\}.$$

The characteristic function ψ converts an input pixel p into f^2 output pixels y_i based on the content of the window W shifted to p, i.e. for $1 \le i \le f^2$,

$$y_i = \Psi(Q)(f\,p + d_i) = \psi_i(Q(W_1 + p), \ldots, Q(W_w + p)),$$

where $p \in \mathbb{Z}^2$ and d_i is a displacement vector associated with the i-th Boolean function. For example, in the figure 1, ψ converts the pixel p into pixels y_1, ..., y_4 based on the content of the 3×3 neighboring window.

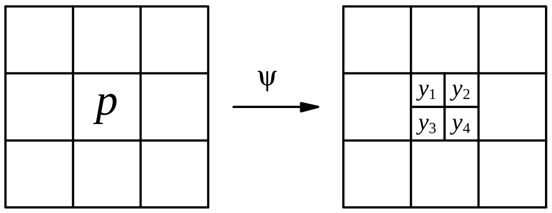


Fig. 1: WZ-operator with zoom factor f=2 (window 3×3).

In the WZ-operator learning stage, the sample input-image $A^{\mathbf{x}}$ and the corresponding output-image $A^{\mathbf{y}}$ are presented to a learning algorithm **A**. Based on these sample images, the learner **A** shall construct a WZ-operator $\hat{\Psi}$ such that, when $\hat{\Psi}$ is applied to another image $Q^{\mathbf{x}}$ (the image to-be-zoomed) the resulting zoomed-image $\hat{Q}^{\mathbf{y}} = \hat{\Psi}(Q^{\mathbf{x}})$ is expected to be similar to a supposedly unknown ideal output-image $Q^{\mathbf{y}}$.

More explicitly, let us denote the content in $A^{\mathbf{x}}$ of the window W shifted to p as $a_p^{\mathbf{x}}$ and call it the sample input-pattern at p:

$$a_p^{\mathbf{x}} = \left[A^{\mathbf{x}}(W_1 + p), A^{\mathbf{x}}(W_2 + p), \ldots, A^{\mathbf{x}}(W_w + p)\right].$$

Let us similarly define the pattern to-be-zoomed $q_p^{\mathbf{x}}$ from $Q^{\mathbf{x}}$. To each pattern $a_p^{\mathbf{x}}$, there are f^2 associated output-colors in the image $A^{\mathbf{y}}$, called the sample output-pattern at p and denoted as $a_p^{\mathbf{y}}$:

$$a_p^{\mathbf{y}} = \left[A^{\mathbf{y}}(f\,p + d_1), \ldots, A^{\mathbf{y}}(f\,p + d_{f^2})\right].$$

Let us similarly define the ideal output-pattern $q_p^{\mathbf{y}}$ and the processed output-pattern $\hat{q}_p^{\mathbf{y}}$ from the images $Q^{\mathbf{y}}$ and $\hat{Q}^{\mathbf{y}}$. A pair ($a_p^{\mathbf{x}}$, $a_p^{\mathbf{y}}$) is called a training example. Using the set of all training examples obtained scanning completely the sample images, the learning algorithm **A** has to construct f^2 Boolean characteristic functions $\hat{\psi}_1$, ..., $\hat{\psi}_{f^2}$ such that, for any p and $1 \le i \le f^2$, the processed color has to be the same as the ideal output color with high probability. That is, the following probability has to be high:

$$\Pr\left[\hat{q}_p^{\mathbf{y}}[i] = q_p^{\mathbf{y}}[i]\right] = \Pr\left[\hat{\psi}_i(q_p^{\mathbf{x}}) = Q^{\mathbf{y}}(f\,p + d_i)\right].$$

3. DECISION-TREE LEARNING

There are many learning techniques that can be used to achieve the goal stated above: neural networks, genetic algorithms, radial basis functions, etc. In fact, the works [2, 3] have proposed to use the statistical optimization and the work [4] the *k*-nearest neighbor learning to solve this problem. As stated in the introduction, their running time (either the training time or the application time) deteriorates quickly with the growth of the window or the sample size, and soon becomes impossible to be used. To be really useful, the technique has to be fast both in the training and in the application, even when window and sample size are large. Fortunately, the decision-tree learning fulfills these requirements. We briefly describe below the decision-tree construction algorithm. More details can be found in [5]. There is a well-known data structure, very similar to the decision-tree. It is the multidimensional binary tree, usually abbreviated as kd-tree, and used to find the nearest neighbor. The construction processes of decision-tree and kd-tree are quite similar. The differences lie mainly on the searching algorithms. Thus, the reader may consult also any work on the construction of the kd-tree [6, 7].

In the decision-tree generating process, the input-pattern space $\{0,1\}^w$ is split into two halves, and all sample input-patterns with black color in the splitting axis W_s will belong to one half-space and those with white color to another. The dimension of the half-spaces so obtained is one less than that of the original space, that is, $\{0,1\}^{w-1}$. To obtain an optimized tree, the algorithm must choose the splitting axis $s \in \{1, ..., w\}$ so that the resulting two half-spaces contain as equal as possible number of sample points (input-patterns). This choice maximizes the entropy of each splitting. If the difference of points is always zero or one, a perfectly balanced tree will be created. As the difference of input-sample quantities in two halves increases, more and more degenerated trees will be constructed. For each one of the two half-spaces obtained, the splitting process continues recursively, generating smaller and smaller spaces. In each splitting, an internal node is created and the splitting axis s is stored in it. This process stops when each space contains either only samples with the same output-pattern or only samples with the same input-pattern but with two or more different output-patterns. In the first case, a terminal node is created and the output-pattern is stored in it. A terminal node is also created in the second case, but the bitwise mode of the output-patterns is evaluated and stored.

Once the decision-tree has been constructed, its application is quite straightforward: given a pattern to-be-zoomed $q_p^{\mathbf{x}}$, the decision-tree is traversed from top to bottom, until a terminal node is reached. The information contained in this terminal node is then chosen as the zoomed output-pattern $\hat{q}_p^{\mathbf{y}}$.

The generalization (also known as inductive bias) is an important issue in any machine learning process. In the resolution-increasing, the generalization allows assigning a reasonable output-pattern to never-seen-before input-patterns, which improves WZ-operator's accuracy. The decision-tree learning has a good generalization policy, known as Occam's razor [5]: "Prefer the simplest hypothesis that fits the data." This inductive bias has already been applied to many different applications with good results.

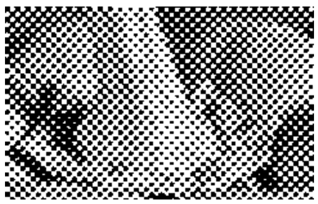
Let us analyze briefly the computational complexity of the decision-tree learning. The construction algorithms of the decision-tree and kd-tree are quite similar. Therefore, a decision-tree can be built in average time $O(wm \log m)$ like kd-tree, where m is the quantity of the training examples or, equivalently, the number of pixels in A^x [7]. If a decision-tree is built from m examples and if every terminal node contains only one sample, it is easy to see that the height of the decision-tree will be $\lceil \log_2 m \rceil + 1$ (where $\lceil \cdot \rceil$ stands for round up). Thus, if there can exist terminal nodes with more than one sample, the height will be at most $\lceil \log_2 m \rceil + 1$. Consequently, the searching of n query points in a decision-tree can be performed in time $O(n \log m)$, for the searching processes in a decision-tree and in a standard binary tree are identical (n is the number of pixels in the image to-be-zoomed Q^x). Note that the searching complexity does not depend at all on the dimension w of the input-pattern space. Finally, if every terminal node contains only one sample, the number of terminal nodes will be m and the number of internal nodes will be m-1. In practice, the total number of nodes uses to be much smaller than 2m-1, especially using a small window, for many terminal nodes contain much more than one sample. The quantity of informations stored in a terminal node is $O(f^2)$ and the quantity of informations stored in an internal node does not depend on any parameter. Considering f^2 as a constant (for it is always a small number) the total storage is $O(m)$. Certainly, a careful programming is required to assure that the program runs within the running time and space complexities stated above.

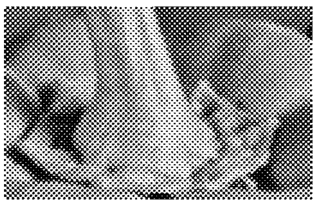
The analysis above shows that the performance of the construction and the searching algorithms, as well as the quantity of needed memory, deteriorate only very slowly as the window and sample size increase. Experimental results presented in the next section confirm that the technique is indeed very fast.

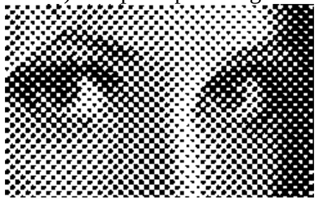
4. EXPERIMENTAL RESULTS

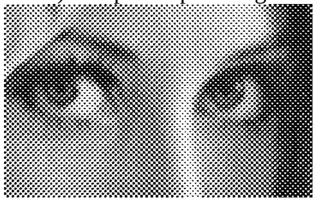
The proposed technique has been implemented and tested. Two completely independent 512×512 gray-scale images have been used for the training and the application (respectively the images *Peppers* and *Lenna*.) They have been converted into 300 and 600 dpi binary images (1050×1050 and 2100×2100 pixels) by a HP LaserJet driver for the Microsoft Windows. Most HP LaserJet drivers include 5 halftone techniques: *none, coarse, fine, line art* and *error diffusion*. Let us leave out the options *none* and *line art,* for the option *none* is a simple thresholding and the option *line art* is intended to imitate a handmade halftone. The default option is *coarse* and it is by far the best-looking option, when the resulting image is printed in a laser printer. This option seemingly implements the dot diffusion algorithm [1]. The option *fine* implements the ordered dither algorithm and the option *error diffusion* implements the Floyd-Steinberg's algorithm [1]. In addition, the proposed technique has also been tested with images generated by the clustered-dot ordered dither algorithm provided by the program "Image Alchemy."

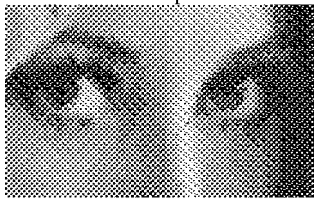
Using windows of different sizes, WZ-operators have been generated by the decision-tree learning. These operators have been applied to 300 dpi binary halftone images of "Lenna," generating resolution increased 600 dpi images.

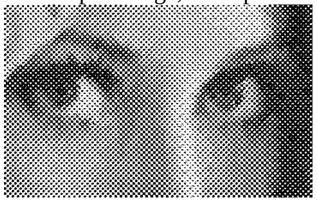

2a) Sample input-image.

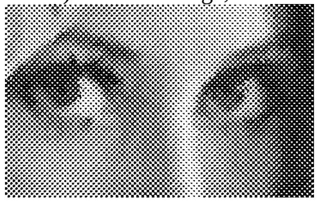

2b) Sample output-image.


2c) Image to-be-zoomed, 300 dpi.


2d) Ideal (unknown) output-image, 600 dpi.


2e) Zoomed-image, 4×4.


2f) Zoomed-image, 5×5.


2g) Zoomed-image, 7×7.

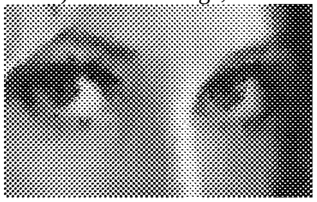

2h) Zoomed-image, 8×8.

Fig. 2: Resolution increasing of images generated by HP LaserJet driver, halftone option "coarse."

window	error	training	application	memory
4×4	7.54%	16s	4s	20 kBytes
5×5	2.64%	19s	5s	67 kBytes
7×7	1.81%	32s	5s	258 kBytes
8×8	1.71%	41s	5s	429 kBytes
9×9	1.77%	58s	5s	679 kBytes
4×4, 10-NN	7.53%	507s	3s	33 kBytes
5×5, 10-NN		3 days*	3s*	17 MBytes*

Tab. 1: Computer performance to obtain images in figure 2. *The last line presents estimated data.

The figures 2a-2b depict a portion of *coarse* halftone images used as the training images, the figure 2c is the image to-be-zoomed and the figure 2d is the ideal output image. The figures 2e-2h depict zoomed images, using different window sizes. The table 1 shows the accuracy and the computer performance of our technique. The error is the proportion of pixels in the zoomed-image that are different from the ideal output. The processing times were measured in a Pentium-300 computer. Note that, as expected by the complexity analysis, the application time is affected only slightly by the change of the window size. The needed memory decreases when the window shrinks because, as the window size diminishes, there are more and more repeated input samples. But, as we have analysed before, the memory needed is upper limited by $O(m)$, and thus even using a huge window the need of memory will not go beyond a tolerable limit.

The last two lines of the table 1 show the computer performance of the k-nearest neighbor learning (k=10) using the look-up-table, as proposed in [4]. As expected, its application is faster and the accuracy is slightly better than the new technique, but its training time is far longer than the decision-tree for w=16 and grows exponentially with the growth of w (the size of the window).

The figure 3 depicts the zooming of the *fine* halftone images and the figure 4 depicts the resolution increasing of *clustered-dot* halftone images.

The error rates of the proposed technique, applied to different halftone techniques, are presented in the table 2. As expected, the proposed technique did not work properly with the *error diffusion*, for this technique does not decide locally the colors of a zoomed output-pattern (note that the *error diffusion* error rates are always larger than 10%, no matter the window used.)

5. CONCLUSIONS

In this paper, we have presented a new technique that improves previous learning-based binary image resolution increasing schemes. It makes use of the decision-tree learning. This technique can use a window as large as 9×9 without significantly compromising the processing time. No previous technique was able to deal with so large a window, because their running times skyrocket as the window and sample size grow. This improvement made possible to accurately zoom even halftone images. The complexity analysis has shown that the running time of decision-tree learning deteriorates very slowly as the window and sample size grow. Moreover, the decision-tree learning presents a reliable and widely tested generalization scheme that increases the accuracy of the operator. The experimental data have shown that the decision-tree learning presents error rates similar to the k-nearest neighbor learning while the computer performance is greately improved. The experiments have also shown that indeed this technique can accurately zoom halftone images generated by any locally-dependent halftone algorithm.

6. ACKNOWLEDGEMENT

We would like to express our gratitude to CNPq for the partial financial support of this work.

7. REFERENCES

[1] D. E. Knuth, “Digital Halftones by Dot Diffusion,” *ACM T. on Graphics*, vol. 6, no. 4, pp. 245-273, 1987.

[2] R. P. Loce and E. R. Dougherty, *Enhancement and Restoration of Digital Documents: Statistical Design of Nonlinear Algorithms*, SPIE Press, 1997.

[3] R. P. Loce, E. R. Dougherty, R. E. Jodoin and M. S. Cianciosi, “Logically Efficient Spatial Resolution Conversion Using Paired Increasing Operators,” *Real-Time Imaging*, vol. 3, no.1, pp. 7-16, 1997.

[4] H. Y. Kim and P. S. L. M. Barreto, “Fast Binary Image Resolution Increasing by k-Nearest Neighbor Learning,” in *Proc. IEEE Int. Conf. on Image Proc.* (Vancouver, Canada), vol. 2, pp. 327-330 (TA9.06), 2000.

[5] T. M. Mitchell, *Machine Learning*, WCB/McGraw-Hill, 1997.

[6] J. H. Friedman, J. L. Bentley and R. A. Finkel, “ An Algorithm for Finding Best Matches in Logarithmic Expected Time,” *ACM T. Math. Software*, vol. 3, no. 3, pp. 209-226, 1977.

[7] F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, Springer-Verlag, 1985.

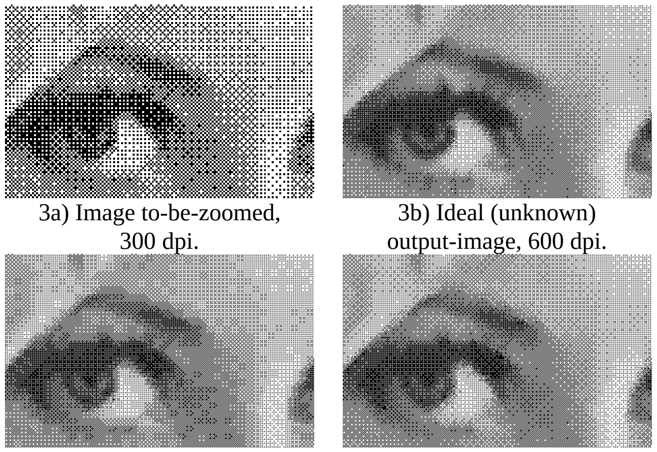

3a) Image to-be-zoomed, 300 dpi.
3b) Ideal (unknown) output-image, 600 dpi.
3c) Zoomed-image, 4×4.
3d) Zoomed-image, 7×7.

Fig. 3: Resolution increasing of images generated by HP LaserJet driver, halftone option “fine” (ordered dither.)

	4×4	5×5	7×7	9×9
coarse	7.54%	2.64%	1.81%	1.77%
fine	2.53%	1.98%	1.62%	1.63%
clustered-dot	4.85%	2.29%	1.84%	1.92%
error diffusion	12.90%	13.01%	14.50%	16.41%

Tab. 2: Error rates.

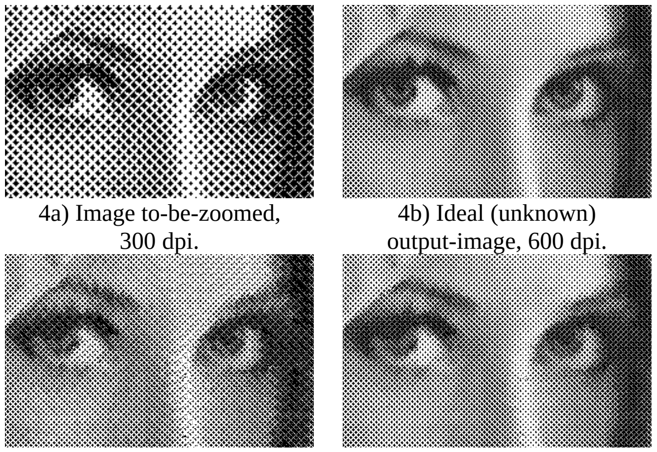

4a) Image to-be-zoomed, 300 dpi.
4b) Ideal (unknown) output-image, 600 dpi.
4c) Zoomed-image, 4×4.
4d) Zoomed-image, 7×7.

Fig. 4: Resolution increasing of images generated by “Image Alchemy,” option “clustered-dot halftone.”