Paper published in Proceedings of
Fifth IEEE International Conference on Image Processing,
October 4-7, 1998, Chicago, Illinois, USA,
vol. 2, pp. 737 - 741, 1998

Automatic Design of Nonlinear Filters by Nearest Neighbor Learning

Hae Yong Kim and Flávio A. M. Cipparrone

Automatic Design of Nonlinear Filters by Nearest Neighbor Learning

Hae Yong Kim and Flávio A. M. Cipparrone
LPS, DEE, Escola Politécnica, Universidade de São Paulo.
Av. Prof. Luciano Gualberto, tr. 3, 158; CEP 05508-900, São Paulo, SP, Brazil.
E-mails: hae@lps.usp.br, cippafla@lps.usp.br.

Abstract

Nonlinear filters have been used not only in the noise elimination but in a wide variety of image processing applications. Traditionally the design of a digital filter is a manual task and the user accomplishes it based on his previous experiences. Unfortunately often this is not a trivial task. Thus some recent works try to overcome this difficulty constructing the filters automatically by computational learning, neural networks, genetic algorithms and statistical estimation. These works use typical input-output images of the application as the training samples. Many different kinds of filters can be easily constructed using this approach. This paper proposes the use of the nearest neighbor (NN) learning to the automatic filter construction. The kd-tree (k-dimensional binary tree) is used to accelerate the NN searching. A texture recognition application example is depicted.

1. Introduction

In recent years, there has been a growing interest in the nonlinear image processing. Nonlinear filters are used not only in the noise elimination but in a wide variety of applications such as texture recognition, shape recognition, segmentation, edge detection, skeleton obtaining, etc. Nevertheless, the construction of the desired filter, that is, the choice of the most adequate filter for a given application, is not a trivial task. To overcome this difficulty, many different techniques have been proposed, for example, the use of the artificial intelligence [1] and the fuzzy expert system [2].

A different methodology proposes to project a filter Ψ supposing that the probability distribution P, responsible for the generation of the input image $Q^{\mathbf{x}}$ and the output image $Q^{\mathbf{y}}$, is entirely known. For example, let $Q^{\mathbf{x}}$ be a noisy image and $Q^{\mathbf{y}}$ the corresponding clean image. If the statistical process P of the corruption of the image $Q^{\mathbf{y}}$ is known, a filter Ψ can be constructed to minimize the expectation of the difference between the clean image $Q^{\mathbf{y}}$ and the processed image $Q^{\mathbf{p}} = \Psi(Q^{\mathbf{x}})$. In this context, the classic image processing textbooks, as [3], expose linear techniques for the image restoration; the works [4, 5] design the stack filter that minimizes the mean absolute error; and the work [6] designs the morphological filter that minimizes the mean square error.

In practice, the distribution P is usually unknown. Thus, a more realistic and useful approach uses the training input-output images $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$, generated by the distribution P, instead of the distribution P itself. The filter Ψ is projected automatically from the sample images by a learning algorithm. This is the underlying idea of the recent works [7-9], in spite of the apparently different theories on which they are based.

When an image $Q^{\mathbf{x}}$ is processed by an automatically constructed filter Ψ, probably the most of the patterns found in $Q^{\mathbf{x}}$ do not appear in the training input image $A^{\mathbf{x}}$. Therefore, an important subject of the automatic filter designing is the strategy adopted to generalize the behavior of the filter to the untrained patterns. The aim of this strategy is to generalize heuristically the filter to the unknown domain points using function approximators. If the generalization is successful, the processed image $Q^{\mathbf{p}}=\Psi(Q^{\mathbf{x}})$ will be "similar" to the unknown ideal output image $Q^{\mathbf{y}}$, even using only a small amount of training samples. To achieve this aim, the literature uses, for example, genetic algorithms [7], neuro-fuzzy operators [8] or decompose the filter as a set of elementary morphological operators [9].

The present paper proposes the use of the NN learning [10], for it seems to be the most intuitive solution to the "heuristic generalization." The kd-tree [11, 12] is used to accelerate the NN searching. The processing time of this new approach is longer than the technique reported in [9]. Nevertheless, the new technique usually generates filters with better quality. In order to shorten the paper, we will be concerned only with the gray-scale filters.

2. Windowed filter learning problem

Let $(E, +)$ be an Abelian group (usually, $E=\mathbb{Z}^2$) and let K be an interval of integer numbers. A gray-scale image is usually defined as a function $Q:E\rightarrow K$, also denoted as $Q\in K^E$. Let $A^{\mathbf{x}}$, $A^{\mathbf{y}}$, $Q^{\mathbf{x}}$ and $Q^{\mathbf{y}}$ be respectively the *input*

sample, the *output sample*, the *image to be processed* and the *ideal output image*.

The goal of this paper is only the design of *windowed filters*, for the most of the filters used in practice are windowed. A windowed filter $\Psi:K^E \to K^E$ is defined via a *window* $\vec{W} = (W_1, ..., W_w)$, $W_i \in E$, and a *characteristic function* $\psi:K^W \to K$ as follows:

$$\Psi(Q)(p) = \psi(Q(W_1 + p),\ldots,Q(W_w + p)),\ Q \in K^E,\ p \in E\ .$$

The *support* of an image is a finite subset of E where the image is really defined. The size of the support is the number of pixels of the image. Out of its support, an image is considered to be filled with a background color. The colors of the pixels of $A^{\mathbf{x}}$, that belong to the window $\vec{W}$ translated to $p \in E$, define a point

$$a^{\mathbf{x}} = [A^{\mathbf{x}}(W_1 + p), \ldots, A^{\mathbf{x}}(W_w + p)]$$

in the space K^W. This point, whose coordinates are the gray-scales of the pixels within the window $\vec{W} + p$, is called a *pattern*. To each pattern $a^{\mathbf{x}}$ so obtained, there is an associated output value $a^{\mathbf{y}} = A^{\mathbf{y}}(p)$ in the space K. Let m be the size of the support of the images $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$. Let us denote the data obtained when all the pixels of $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$ are scanned as:

$$\vec{a} = (a_1,\ldots,a_m) = ((a_1^{\mathbf{x}},a_1^{\mathbf{y}}),\ldots,(a_m^{\mathbf{x}},a_m^{\mathbf{y}}))\ .$$

In the filter designing stage, a *learning algorithm* $\mathbf{A}:\left(\bigcup_{m\geq 1}(K^W \times K)^m\right) \to \left(K^W \to K\right)$ receives the sequence $\vec{a}$ and generates the characteristic function $\psi=\mathbf{A}(\vec{a})$. The function ψ and the window $\vec{W}$ together represent a windowed filter Ψ.

There exists a unknown joint *probability distribution* P in $K^W \times K$ that generates independently each element of the sequence $\vec{a}$. The same probability distribution generates independently each element of the sequence $\vec{q}$ that forms the images $Q^{\mathbf{x}}$ and $Q^{\mathbf{y}}$:

$$\vec{q} = (q_1,\ldots,q_n) = ((q_1^{\mathbf{x}},q_1^{\mathbf{y}}),\ldots,(q_n^{\mathbf{x}},q_n^{\mathbf{y}}))\ ,$$

where n is the number of pixels of $Q^{\mathbf{x}}$ and $Q^{\mathbf{y}}$.

In the filter application stage, the characteristic function ψ is applied to each input point $q_i^{\mathbf{x}}$, generating processed output values $q_i^{\mathbf{p}} = \psi(q_i^{\mathbf{x}})$. The sequence of all output values forms the *processed image* $Q^{\mathbf{p}}$.

The error of a filter can be measured using a real-valued *loss function* $l(q_i^{\mathbf{p}},q_i^{\mathbf{y}})$ that measures the difference between the processed and the ideal outputs (e.g., the square error $(q_i^{\mathbf{p}} - q_i^{\mathbf{y}})^2$ or the absolute error $|q_i^{\mathbf{p}} - q_i^{\mathbf{y}}|$). If the learning process was effective, the resulting mean error will be small.

3. Nearest neighbor learning

Although many different techniques (e.g., neural networks and genetic algorithms) can be used to solve the windowed filter learning problem, the *NN learning* seems to be the most natural solution.

In the original NN learning [10], given a point $q_i^{\mathbf{x}} \in K^W$, its processed value $q_i^{\mathbf{p}} = \psi(q_i^{\mathbf{x}}) = \mathbf{A}(\vec{a})(q_i^{\mathbf{x}})$ is the mode (the most frequent value) among the outputs of the nearest neighbors in the training sequence $\vec{a}$. Instead of the mode, the arithmetic mean was used to solve the "ties," for it minimizes the expected value of the square loss function.

To use NN learning in practice, a good algorithm for the NN searching problem is required. The NN searching problem can be stated as follows: "Given m training points $a_1^{\mathbf{x}}$, ..., $a_m^{\mathbf{x}}$ and n points to be processed $q_1^{\mathbf{x}}$, ..., $q_n^{\mathbf{x}}$ (all points in the space K^W) find, for each point to be processed $q_i^{\mathbf{x}}$, the set of the nearest training points." This problem has a trivial solution with the computational complexity $O(wmn)$. In this solution, the distances between each $q_i^{\mathbf{x}}$ and all the m training points are computed and hence the nearest training points are chosen. Let us refer to this as *brute-force* algorithm. Obviously, this algorithm is excessively slow.

There are faster algorithms for the NN searching and probably the most used and known is the *kd-tree*. The kd-tree is a generalization of the simple binary tree used for sorting and searching. The root of the tree represents the set of all training points. Each non-terminal node has two successor nodes that represent two subsets defined by partitioning the parent's training points. The terminal nodes represent mutually exclusive small subsets of the training points. For a thorough exposition, the reader is referred to [11, 12]. To use the kd-tree in the windowed filter learning problem, the output gray-scale $a_j^{\mathbf{y}}$ should be stored in the terminal node, as well as the corresponding pattern $a_j^{\mathbf{x}}$. In practice, to minimize the use of the memory, only the coordinates (line and column numbers) of the pattern $a_j^{\mathbf{x}}$ in $A^{\mathbf{x}}$ was stored in the terminal node (note that the coordinates of $a_j^{\mathbf{y}}$ in $A^{\mathbf{y}}$ are the same). This information, together with the images $A^{\mathbf{x}}$ and $A^{\mathbf{y}}$, allows to evaluate $a_j^{\mathbf{x}}$ and $a_j^{\mathbf{y}}$.

A kd-tree, constructed as above, can be viewed as a characteristic function ψ. Given a pattern $q_i^{\mathbf{x}}$, a searching in the kd-tree finds the nearest training patterns. The output value of the characteristic function $\psi(q_i^{\mathbf{x}})$ is then defined as the arithmetic mean of the output gray-scales of

the nearest patterns. According to [12], the computational complexity of the construction of a kd-tree is $O(wm \log m)$ and the n points searching in a kd-tree takes $O(nwm^{(1-1/w)})$. That is, kd-tree is fast in non excessively large dimension w.

4. Experimental results

The application of the proposed technique in a texture recognition problem is depicted in the figure 1. The figure 1a shows the map of the south region of Brazil with mean values in January of the daily maximum temperatures. The figure 1b, edited manually, corresponds to the checked texture. Using a H-shaped window with 7 elements (figure 2a) and figures 1a and 1b as the training samples, we obtained a nonlinear filter that recognizes the checked texture. Applying it in the figure 1c (temperatures in June), we obtained the figure 1d. Filtering it four times with 6×6 median filter, the figure 1e resulted. The median filter is widely used in the noise elimination and its definition can be found, for example, in [13]. The figure 1f was obtained merging the figures 1c and 1e. The construction of the kd-tree took 30 seconds and its application 11530 seconds in a computer with Pentium 100 MHz processor (table 1).

Note in the figure 2b that the checked texture is full of irregularities and a manual design of the checked texture recognition filter would have been a troublesome task. Applying exactly the same process in another map (figure 2c), the "checked texture" was again successfully recognized (figure 2d). This process can also be used to recognize other textures. Using the figures 1a and 2e as training samples, "dark hatched texture" of the figure 2c was recognized, resulting the figure 2f.

Size of the window	Brute-force	Kd-tree (training)	Kd-tree (application)
w=2	42843s	16s	138s
w=3	58622s	18s	141s
w=5	?	20s	2345s
w=7	?	30s	11530s

Table 1: The performance of the brute-force and the kd-tree, measured in a "Pentium-100," to process an 424×312 image using 422×312 sample images. The line w=7 exhibits the processing time to obtain the figure 1d.

5. Conclusions

In this paper, the windowed filter learning problem has been formalized and the nearest neighbor learning has been proposed as a solution to this problem. To accelerate the processing, the proposed technique has been implemented using kd-tree. As an application example, this technique has been used in the gray-scale texture recognition problem.

Acknowledgments

This work has been partially supported by FAPESP.

References

[1] M. Schmitt, "Mathematical morphology and artificial intelligence: an automatic programming system," *Signal Processing*, vol. 16, no. 4, pp. 389-401, 1989.

[2] H. Y. Kim, F. A. M. Cipparrone and M. T. C. Andrade, "Technique for constructing grey-scale morphological operators using fuzzy expert system," *Electron. Lett.*, vol. 33, no. 22, pp. 1859-1861, 1997.

[3] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Addison-Wesley, 1992.

[4] E. J. Coyle and J. H. Lin, "Stack Filters and the Mean Absolute Error Criterion," *IEEE Trans. Ac., Speech, Signal Proc.*, vol. 36, no. 8, pp. 1244-1254, Aug. 1988.

[5] W. L. Lee, K. C. Fan and Z. M Chen, "Design of optimal stack filter under MAE criterion," in *Proc. IEEE Int. Conf. Image Proc.* (Santa Barbara, USA), vol. 1, pp. 420-423, 1997.

[6] E. R. Dougherty, "Optimal mean-square N-observation digital morphological filters - optimal binary filters," *CVGIP: Image Understanding*, vol. 55, no. 1, pp. 36-54, Jan. 1992.

[7] R. Harvey and S. Marshall, "The use of genetic algorithms in morphological filter design," *Signal Processing: Image Comm.*, vol. 8, pp. 55-71, 1996.

[8] F. Russo, "Nonlinear filtering of noisy images using neuro-fuzzy operators," in *Proc. IEEE Int. Conf. Image Proc.* (Santa Barbara, USA), vol. 3, pp. 412-415, 1997.

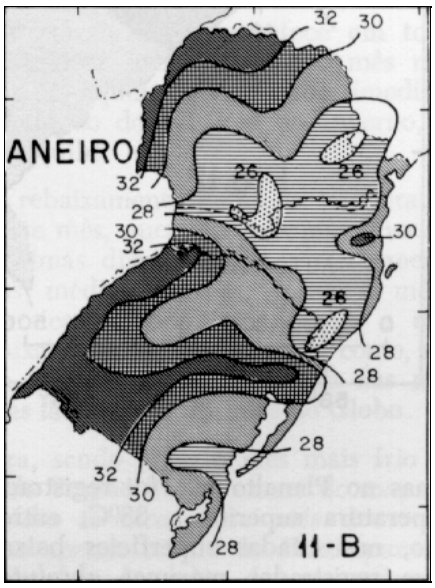
[9] H. Y. Kim, "Quick construction of efficient morphological operators by computational learning," *Electron. Lett.*, vol. 33, no. 4, pp. 286-287, 1997.

[10] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE T. Information Theory*, vol. IT-13, no. 1, pp. 21-27, 1967.

[11] 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, Sep. 1977.

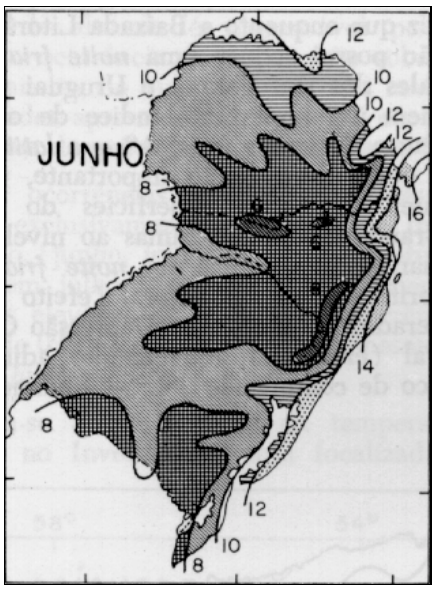
[12] F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, Springer-Verlag, New York, 1985.

[13] I. Pitas and A. N. Venetsanopoulos, *Nonlinear Digital Filters - Principles and Applications*, Kluwer, 1990.



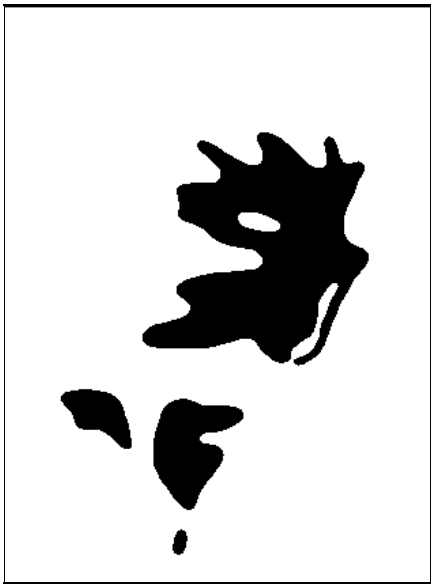
(1a) Input sample (A^x).
422×312 pixels, 1 byte per pixel.

(1b) Output sample (A^y) corresponding to the "checked texture" of 1a.

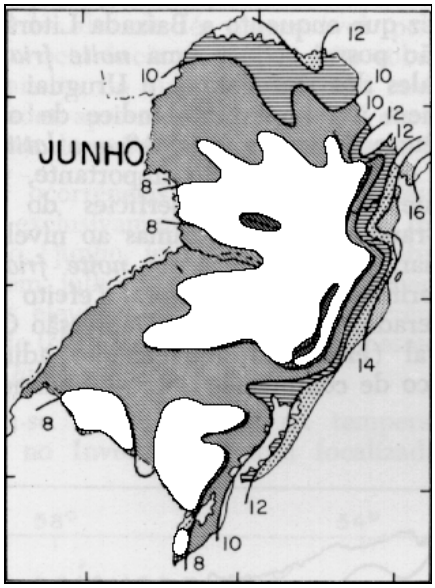


(1c) Image to be processed (Q^x).
424×312 pixels, 1 byte per pixel.

(1d) Processed image (Q^p) using the H-shaped window (figure 2a).

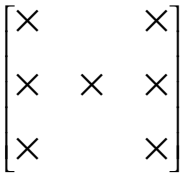


(1e) Image obtained applying 6×6 median filter four times in 1d.

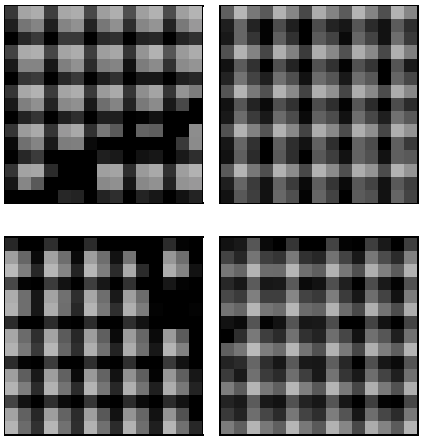


(1f) The result of the merging of the figures 1c and 1e.

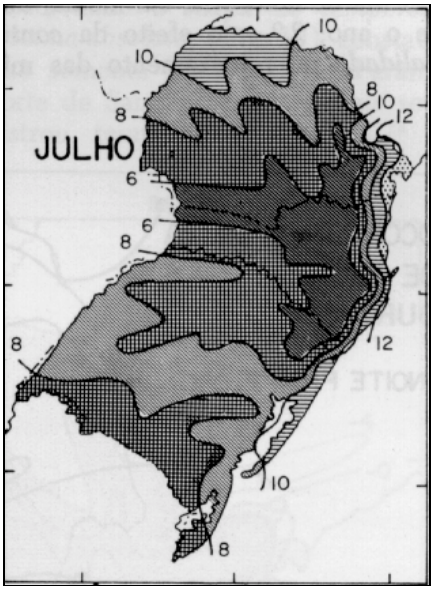
Figure 1: To construct a filter that recognizes the checked texture, the figure 1b was edited manually to correspond to the checked texture region of the figure 1a. Using the figures 1a and 1b as the training input-output samples, a windowed filter was automatically designed. Applying this filter in 1c, the figure 1d was generated. Filtering this image four times with the 6×6 median filter we obtained the figure 1e. Merging the original figure 1c with the figure 1e, the figure 1f resulted and it has the original "checked texture" painted in white.



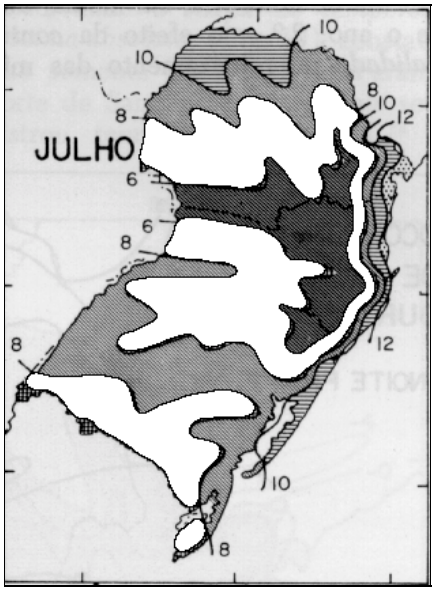
(2a) H-shaped window with 7 elements.



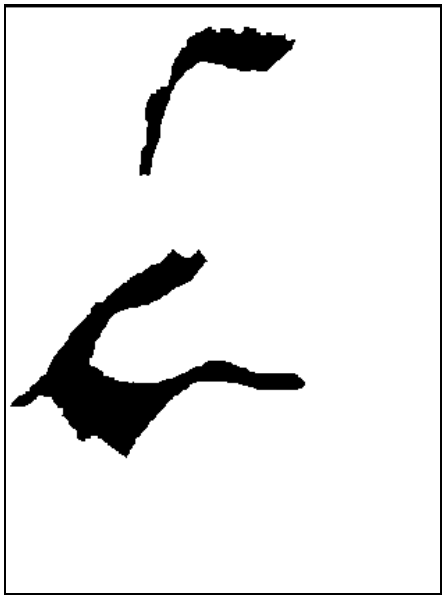
(2b) Zoomed views of the checked texture, extracted from the figure 1c.



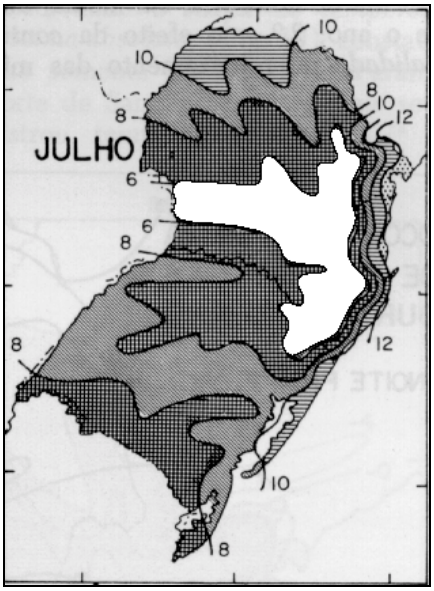
(2c) Map of the temperatures in July (image to be processed Q^x).



(2d) Image obtained applying in 2c the same process used to obtain 1f.



(2e) Output sample (A^y). "Dark hatched texture" of 1a.



(2f) Recognition of the "dark hatched texture" of the figure 2c.

Figure 2: The process described in the figure 1 to recognize the "checked texture" was applied in another figure (2c) and it successfully recognized the aimed texture (figure 2d). The same process was used to recognize the "dark hatched texture." Applying the filter constructed using 1a and 2e as training samples in the figure 2c, the figure 2f resulted.