IMAVIS1802

ELSEVIER

Image and Vision Computing 00 (2001) 000–000

image AND vision COMPUTING

www.elsevier.com/locate/imavis

Image compression using principal component neural networks

S. Costa, S. Fiori*

Neural Networks and Adaptive System Research Group, University of Perugia-DIE/UNIPG, Via Ischia 131, Marcelli di Numana, I-60026 Perugia, Italy

Received 6 April 2000; accepted 18 December 2000

Abstract

Principal component analysis (PCA) is a well-known statistical processing technique that allows to study the correlations among the components of multivariate data and to reduce redundancy by projecting the data over a proper basis. The PCA may be performed both in a batch method and in a recursive fashion; the latter method has been proven to be very effective in presence of high dimension data, as in image compression. The aim of this paper is to present a comparison of principal component neural networks for still image compression and coding. We first recall basic concepts related to neural PCA, then we recall from the scientific literature a number of principal component networks, and present comparisons about the structures, the learning algorithms and the required computational efforts, along with a discussion of the advantages and drawbacks related to each technique. The conclusion of our wide comparison among eight principal component networks is that the cascade recursive least-squares algorithm by Ci-chocki, Kasprzak and Skarbek exhibits the best numerical and structural properties.

Keywords: Image compression; Principal component; Neural network

1. Introduction

Data reduction techniques aim at providing an efficient representation of the data; we consider the research stream which focuses on the compression procedure consisting of mapping the higher dimensional input space into a lower dimensional representation space by means of a linear transformation, as in the Karhunen–Loéve Transform (KLT). The classical approach for evaluating the KLT requires the computation of the input data covariance matrix and then the application of a numerical procedure to extract the eigenvalues and the corresponding eigenvectors; compression is obtained by the use of the only eigenvectors associated with the most significant eigenvalues as a new basis. When large data sets are handled, this approach is not practicable because the dimensions of the covariance matrix become too large to be manipulated. In addition, the whole set of eigenvectors has to be evaluated even though only some of them are used.

In order to overcome these problems, neural-network-based approaches were proposed. Neural principal component analysis (PCA) is a second-order adaptive statistical data processing technique introduced by Oja [23–25] which helps to remove the second-order correlation among given random processes. In fact, consider the stationary multivariate random process $\mathbf{x}(t) \in \mathbb{R}^p$ and suppose its covariance matrix $\mathbf{\Phi} \stackrel{\text{def}}{=} E[(\mathbf{x} - E[\mathbf{x}])(\mathbf{x} - E[\mathbf{x}]^{\mathrm{T}}]$ exists bounded. If $\mathbf{\Phi}$ is not diagonal, then the components of $\mathbf{x}(t)$ are statistically correlated. This second-order redundancy may be partially (or completely) removed by computing a linear operator $\mathbf{F}$ such that the new random signal defined by $\mathbf{y}(t) \stackrel{\text{def}}{=} \mathbf{F}^{\mathrm{T}}(\mathbf{x}(t) - E[\mathbf{x}]) \in \mathbb{R}^m$ has uncorrelated components, with $m \leq p$ arbitrarily selected [7,12,24,25,29,31]. The operator $\mathbf{F}$ is known to be the matrix formed by the eigenvectors of $\mathbf{\Phi}$ corresponding to its largest eigenvalues [7,12]. The elements of $\mathbf{y}(t)$ are termed principal components of $\mathbf{x}(t)$; their importance is proportional to the corresponding eigenvalues $\sigma_i^2 \stackrel{\text{def}}{=} E[y_i^2]$ which are supposed to be arranged in descending order ($\sigma_i^2 \geq \sigma_{i+1}^2$).

The data-stream $\mathbf{y}(t)$ represents a compressed version of data-stream $\mathbf{x}(t)$; after the reduced-size data-stream has been processed (i.e. stored, retrieved, transmitted), it needs to be recovered, i.e. brought back to its original size. However, the principal component-based data reduction technique is not loss-less, thus only an approximation $\hat{\mathbf{x}}(t)$ of the original data-stream may be recovered. As $\mathbf{F}$ is an orthonormal operator, an approximation of $\mathbf{x}(t)$ is given by $\hat{\mathbf{x}}(t) = \mathbf{F}\mathbf{y}(t) + E[\mathbf{x}]$; it minimizes the reconstruction error $E[\|\mathbf{x} - \hat{\mathbf{x}}\|^2]$ which equals $\sum_{k=m+1}^{p} \sigma_k^2$.

In 1982, Oja [23] proposed the use of a simple neural unit to extract the first principal component from the input. Since this pioneering work, several new learning algorithms have been proposed for extending the one-unit neural system to a

* Corresponding author.
E-mail address: sfr@unipg.it (S. Fiori).

0262-8856/01/$ - see front matter
PII: S0262-8856(01)00042-7

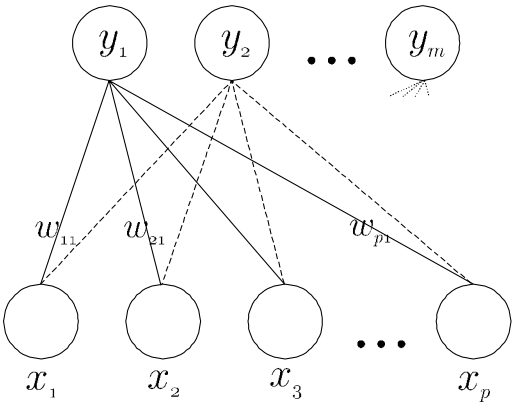

Fig. 1. Single-layered feed-forward neural network for GHA.

complete neural network for the extraction of more than one principal component.

The aim of this paper is to present a comparison of principal component neural networks applied to still image compression, which is indeed a natural field of application of PCA [13], as numerical simulations show that a little amount of principal components is required in order to reconstruct a good quality image after compression. Moreover, as any component is intrinsically endowed with a measure of significance given by its power (or variance), it is really straightforward to perform efficient components coding by optimal bit allocation, which provides an extra compression.[1]

The paper is organized as follows: in Section 2 we present a description of considered algorithms and associated neural structures. Some details about implementation are given in Section 3. Section 4 is devoted to assess results with numerical and structural comparisons; compression performances are discussed along with efficient bit allocation, network's generalization capability, which provides an useful tool for performing image compression without network's relearning, and colored image compression issues. Concluding remarks and final comments are given in Section 5.

2. Principal component networks

The classical contribution in the field of principal component networks has been given by Sanger [29], who used an online version of the well-known Gram–Schmidt orthogonalization algorithm, termed generalized Hebbian learning (GHA). Also, Rubner and Tavan [28] and later Kung and Diamantaras [7,21] introduced a linear neural network endowed with lateral inhibitory connections and an additional set of learning equations for achieving output de-correlation, termed adaptive principal-component extractor (APEX).

Over recent years, several authors tried to give different rules for generalizing classical ones. It is worth citing the recursive least-square approach (RLS-PCA) by Bannour and Azimi-Sadjadi [3]; the class of ψ-APEX learning rules, introduced as a generalization of APEX rule by Fiori et al. [9–11]; the successive application of modified Hebbian learning (SAMH) by Abbas and Fahmi [1], who introduced the concept of sequential extraction of principal components from previously deflated data; the cascade recursive least-squares approach (CRLS) by Cichocki et al. [5], that combines the advantages of both SAMH and RLS-PCA. Also worth mentioning are the non-linear extensions to PCA by Oja, Karhunen and coworkers [14,17–19,26], the extensions of PCA for performing independent component analysis (ICA) by Karhunen, Oja and coworkers [18,20,30], the recently developed extensions to classical PCA to its complex-valued counterpart by DeCastro et al. [6] and the non-linear complex-valued counterpart for performing blind separation of circularly distributed signals by Fiori [8]. A PCA-related argument is principal subspace extraction, which is the computation of the subspace of the input space spanned by the principal eigenvectors [15,16,22,24].

The algorithms considered within this paper are the GHA [29], the APEX [7], the ones belonging to ψ-APEX class [11], the SAMH [1], RLS-PCA [3], and CRLS [5]. The reasons for which we chose to present a comparison of these algorithms, among the cited ones, is that we wish to compare low-complexity second-order data reduction techniques only, and that the known algorithms for performing independent component analysis have not yet proven to give reliable results in large-scale data processing. Also, we refer to the interesting paper by Puga and Alves [27] who tested the classical variance-based bit allocation scheme with ICA concluding that, in comparison, PCA produces better distortion-ratio profiles as well as better visual results.

The considered algorithms base upon Oja's principal component neuron described by $y(t) = \mathbf{w}^{\mathrm{T}}(t)\mathbf{x}(t)$, where $\mathbf{x}(t) \in \mathbb{R}^p$ represents the stationary zero-mean multivariate random process whose first principal component is looked for, $\mathbf{w}(t) \in \mathbb{R}^p$ is the neuron's weight vector, and $y(t) \in \mathbb{R}$ is the neuron's output signal. Oja's learning rule [23] reads:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta[y(t)\mathbf{x}(t) - y^2(t)\mathbf{w}(t)], \tag{1}$$

where η is a small learning step-size and t denotes discrete time. This expression clearly reveals the presence of the Hebbian term $+\mathbf{x}(t)y(t)$ and of a stabilizing term, thus it is also referred to as stabilized (or modified) Hebbian learning equation.

2.1. The feed-forward network and GHA rule

The GHA proposed by Sanger in Ref. [29] is one among

[1] In practice the values of the principal components by themselves are not sufficient for the reconstruction of the original signals; in fact, the compression operator and any other information concerning the compression procedure (as the mean value and the bit allocation scheme) must be part of the compressed data. This slightly reduces the achievable compression rate; this problem will not be considered within the present paper.

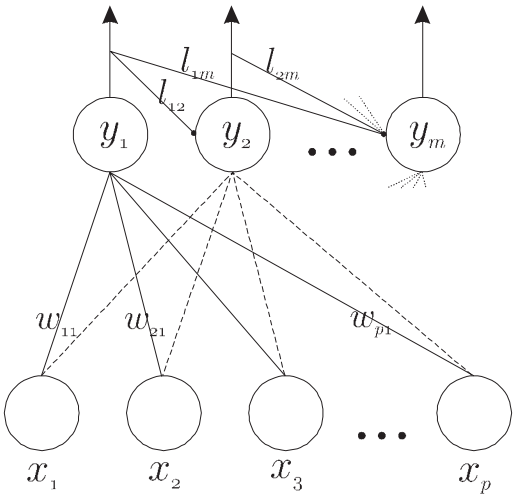

Fig. 2. Laterally connected network for APEX.

the best-known learning algorithms that allow a linear neural network to extract a selected number of principal components. It applies to a single-layered feed-forward neural network (see Fig. 1) described by:

$$\mathbf{y}(t) = \mathbf{W}^{\mathrm{T}}(t)\mathbf{x}(t), \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^p$ represents the input vector, $y \in \mathbb{R}^m$ denotes network's output vector and $\mathbf{W} \in \mathbb{R}^{p\times m}$ is the weight-matrix connecting inputs to outputs; here we assume $m \leq p$ arbitrarily selected. The GHA learning rule can be written as:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \eta(\mathbf{x}(t)\mathbf{y}^{\mathrm{T}}(t) - \mathbf{W}(t)\mathrm{LT}[\mathbf{y}(t)\mathbf{y}^{\mathrm{T}}(t)]), \tag{3}$$

where η is a small positive learning step-size, and the operator LT[ ] returns the lower-triangular part of the matrix contained within.

This rule implies parallel extraction of all m principal components. It can be viewed as a modification of Oja's rule (1) by rewriting Eq. (3) in local mode (for each entry w_{ij}):

$$w_{ij}(t+1) = w_{ij}(t) + n[y_i(t)\underbrace{(x_j(t) - \sum_{k<1} w_{kj}(t)y_k(t))}_{x'_j} - y_1^2(t)w_{ij}(t)], \tag{4}$$

where the modification consists in the introduction of x'_j, which takes the place of x_j. In this way it is possible to use the same algorithm in sequential mode also, which means sequentially extracting one component at a time.

The GHA rule has two flaws: there is no general criterion to choose the value of learning rate, which was originally chosen in a heuristic way using a trial-and-error approach, and the applied Gram–Schmidt orthogonalization procedure is ineffective with principal components related to the smallest eigenvalues [5].

2.2. The laterally-connected architecture and APEX learning rule

Kung and Dimantaras presented in Ref. [7] a learning rule referred to as APEX that applies to a neural network with lateral connections, shown in Fig. 2, having the hierarchical structure proposed by Rubner and Tavan in Ref. [28]. The aim of lateral connections in $\mathbf{l}_i = [l_{i1}, l_{i2}, ..., l_{im}]^{\mathrm{T}}$, also termed "anti-Hebbian connections" or "orthogonalization connections", is to orthogonalize the synaptic weights extracted by the network. The network structure is described by the following relationships:

$$\tilde{\mathbf{y}}_i(t) = \tilde{\mathbf{W}}_i^{\mathrm{T}}\mathbf{x}(t), \tag{5}$$

$$y_i(t) = \mathbf{w}_i^{\mathrm{T}}(t)\mathbf{x}(t) - \mathbf{l}_i^{\mathrm{T}}(t)\tilde{\mathbf{y}}_i(t), \tag{6}$$

where y_i is the output of the ith neuron (note the sequential structure), $\mathbf{w}_i$ is the weight-vector connected to the output y_i, $\mathbf{l}_i$ is the lateral-connections vector connected to the previous $i-1$ outputs and $\tilde{\mathbf{y}}_i$ is just the vector of previous outputs $(1/(i-1))$. Thus $\tilde{\mathbf{W}}_i$ is the weight-matrix connecting the inputs to the first $i-1$ outputs and does not include $\mathbf{w}_i$; only $\mathbf{w}_i$ and $\mathbf{l}_i$ are trained when the ith neuron is interested by learning, while $\tilde{\mathbf{W}}_i$ remains constant.

The tth iteration of the learning algorithm is described by:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(y_i(t)\mathbf{x}(t) - y_i^2(t)\mathbf{w}_i(t)), \tag{7}$$

$$\mathbf{l}_i(t+1) = \mathbf{l}_i(t) + \eta(y_i(t)\tilde{\mathbf{y}}_i(t) - y_1^2(t)\mathbf{l}_i(t)). \tag{8}$$

Kung and Dimantaras [7] also described a parallel version for APEX. As opposed to the sequential one, the parallel version can extract all the m principal components at the same time. A way to build up the parallel version from the sequential one is to recast the learning equation in matrix form; for the input–output relationship we get:

$$\mathbf{y}(t) = \mathbf{W}^{\mathbf{T}}(t)\mathbf{x}(t) + \mathbf{L}^{\mathbf{T}}(t)\mathbf{y}(t), \tag{9}$$

and for the learning algorithm:

$$\mathbf{W}(t+1) = \eta\mathbf{X}(t)\tilde{\mathbf{Y}}(t) + \mathbf{W}(t)[\mathbf{I}_m - \eta\tilde{\mathbf{Y}}^2(t)], \tag{10}$$

$$\mathbf{L}(t+1) = -\eta\mathrm{SUT}[\mathbf{Y}(t)\tilde{\mathbf{Y}}(t)] + \mathbf{L}(t)[\mathbf{I}_m - \eta\tilde{\mathbf{Y}}^2(t)]. \tag{11}$$

It is important to remark that in Eq. (9), the quantities $\mathbf{x}(t)$, $\mathbf{y}(t)$, $\mathbf{W}(t)$ and $\mathbf{L}(t)$ must be evaluated at the same time. In Eqs. (10) and (11) $\mathbf{X}$ is a $p\times m$ matrix, $\mathbf{Y}$ and $\tilde{\mathbf{Y}}$ are $m\times m$ matrices defined by:

$$\mathbf{X} \stackrel{\mathrm{def}}{=} \underbrace{[\mathbf{x}\,\mathbf{x}...\mathbf{x}]}_{m}, \quad Y \stackrel{\mathrm{def}}{=} \underbrace{[\mathbf{y}\,\mathbf{y}...\mathbf{y}]}_{m}, \quad \tilde{\mathbf{Y}} \stackrel{\mathrm{def}}{=} \mathrm{diag}(y_1, y_2, ..., y_m)$$

and operator SUT[ ] returns the strictly upper-triangular part of the matrix contained within.

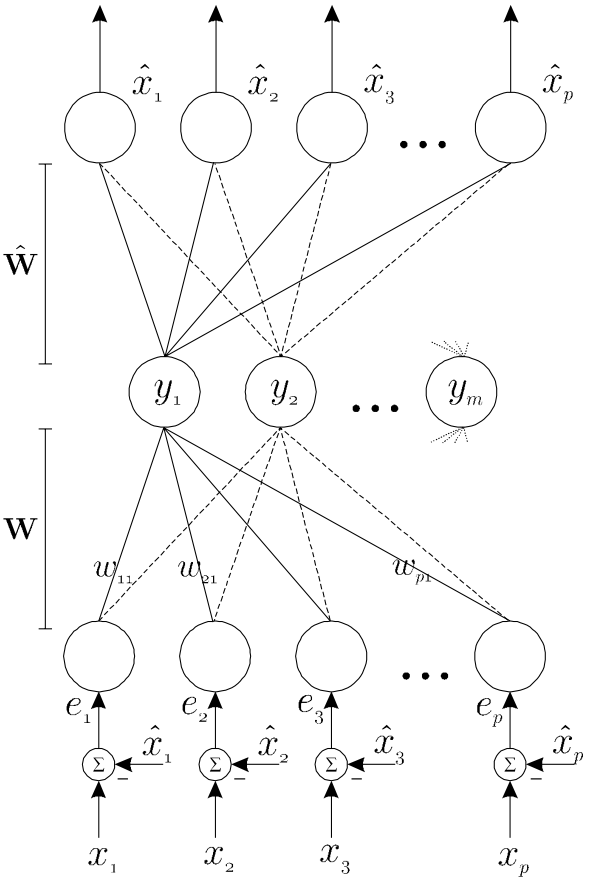

Fig. 3. Autoencoder network with the "deflation" mechanism for the SAMH.

2.3. The ψ-APEX learning rule

A PCA transform is such that the transformed scalar random processes in $\mathbf{z}(t) = \mathbf{W}^{\mathrm{T}}(t)\mathbf{x}(t)$ are characterized by maximum variances; furthermore, any PCA vector must be orthogonal with respect to each other and possess unitary norm so that $\mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{I}_p$.

In contrast to the heuristic derivation of Kung–Diamantaras, in Ref. [11] a learning algorithm based on criterion optimization was presented; in fact, the above targets can be thought of as separate objectives to be attained by means of same Rubner–Tavan's laterally connected neural network, and by the stochastic gradient optimization of the criterion:

$$N(\mathbf{W},\mathbf{L}) \stackrel{\mathrm{def}}{=} \mathrm{TR}[E[\mathbf{y}\mathbf{y}^{\mathrm{T}}]] + \mathrm{TR}[\mathbf{W}^{\mathrm{T}}\mathbf{W} - \mathbf{I}_m)\tilde{\mathbf{\Lambda}}] + \mathrm{TR}[(\mathbf{L}^{\mathrm{T}}\mathbf{L})\tilde{\mathbf{\Psi}}]. \quad (12)$$

Here $\tilde{\mathbf{\Lambda}}$ is a diagonal matrix of Lagrange multipliers whose optimal expression can be found by solving $\mathbf{W}^{\mathbf{T}}(\partial N)/(\partial \mathbf{W}) = 0$ for $\tilde{\mathbf{\Lambda}}$, and operator TR[ ] returns the trace of the matrix contained within.

In Ref. [11] we proposed to adapt the direct-connection weight-matrix $\mathbf{W}$, in order to only maximize the transformed data variance measured by Eq. (12), by means of the following learning rule:

$$\mathbf{W}(t+1) = \eta\mathbf{X}(t)\tilde{\mathbf{Y}}(t) + \mathbf{W}(t)[\mathbf{I}_m - \eta\tilde{\mathbf{Y}}(t)\tilde{\mathbf{Z}}(t)], \quad (13)$$

$$\tilde{\mathbf{Z}} \stackrel{\mathrm{def}}{=} \mathrm{diag}(z_1, z_2, \ldots, z_m), \quad (14)$$

and to adapt the lateral-connection weight-matrix $\mathbf{L}$, in order to only make the components of the response vector $\mathbf{y}$ uncorrelated, i.e. to minimize their correlation measured by the same criterion (12), according to the following rule:

$$\mathbf{L}(t+1) = -\eta\rho\mathrm{SUT}[\mathbf{Y}(t)\tilde{\mathbf{Y}}(t)] + \mathbf{L}(t)[\mathbf{I}_m - \eta\tilde{\mathbf{\Psi}}(t)], \quad (15)$$

$$\tilde{\mathbf{\Psi}} \stackrel{\mathrm{def}}{=} \mathrm{diag}(\psi_1, \psi_2, \ldots \psi_m), \quad (16)$$

where the ψ_k are arbitrary functions that at least guarantee the stability of the network, ρ is a magnifying factor, and η is a positive learning step-size.

A learning rule with structures (13)–(15) with a generic ψ function is termed ψ-APEX. It is worth noticing that the y^2-APEX (i.e. $\psi(y) = y^2$) is very similar to, but is not the same algorithm as, the original APEX; in Refs. [9–11] we also discussed the choices $\psi_k(y_k) = 0$ (0-APEX) and $\psi_k(y_k) = |y_k|$ ($|y|$ − APEX).

In this case too, it is easy to change the parallel version into the sequential one. To this aim let us define the following network quantities:

$$\tilde{\mathbf{y}}_i(t) = \tilde{\mathbf{W}}_i^{\mathrm{T}}\mathbf{x}(t), \quad (17)$$

$$z_i(t) = w_i^{\mathrm{T}}(t)\mathbf{x}(t), \text{ and } y_i(t) = z_i(t) = \mathbf{l}_i^{\mathrm{T}}(t)\tilde{\mathbf{y}}_i(t). \quad (18)$$

Then weights updating equations read:

$$\mathbf{w}_i(t+1) = \eta y_i(t)\mathbf{x}(t) + \mathbf{w}_i(t)(1 - \eta y_i(t)z_i(t)), \quad (19)$$

$$\mathbf{l}_i(t+1) = \eta\rho y_i(t)\tilde{\mathbf{y}}_i(t) + \mathbf{l}_i(t)(1 - \eta\psi_i(t)). \quad (20)$$

The involved constant parameters have the same meaning as in the parallel version.

The ψ-APEX as well as APEX suffer from the problem of learning step-size selection: it has to be chosen heuristically or on the basis of the prior knowledge of eigenvalues, which is rather impractical. APEX also comes with a step-size learning theory, which easily results in an unstable algorithm.

2.4. The autoencoder network and SAMH rule

Abbas and Fahmy proposed in Ref. [1] an algorithm based on Oja's rule, the SAMH (successive application of modified Hebbian learning rule); the employed neural network architecture is shown in Fig. 3; in neural networks literature it is normally referred to as *autoencoder*.

The structure is strictly sequential: first the 1st neuron with output y_1 is trained with weight vector $\mathbf{w}_1 = [w_{11}, w_{21}, \ldots, w_{p1}]^{\mathrm{T}}$ until convergence, then the second neuron is trained until convergence, and so on. Every time a neuron starts learning, it computes a new input data-sequence by subtracting the reconstructed data-obtained by using the previously computed eigenvectors-from the

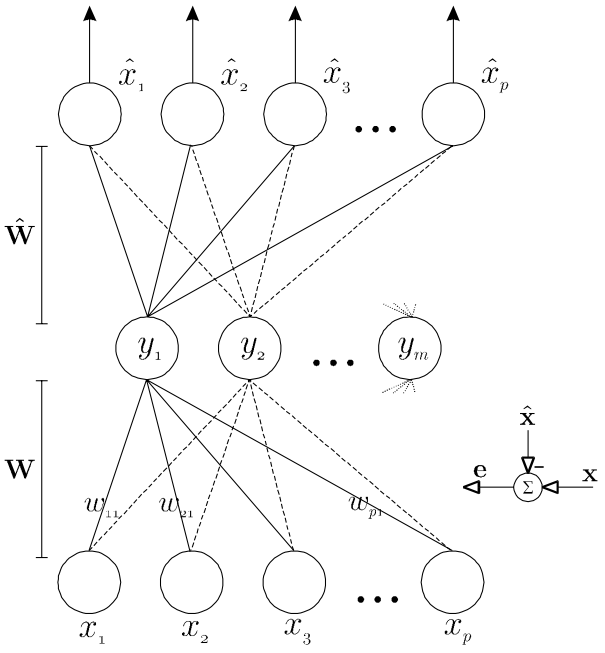

Fig. 4. Network with the "deflation" mechanism used in the RLS-PCA.

original data-sequence. This operation is called by some authors "deflation" and allows to achieve the orthogonality among the eigenvectors.

It is worth noticing that in the two-layer structure of the autoencoder, only the first layer gets trained, as the weight matrix $\hat{\mathbf{W}} \in \mathbb{R}^{m\times p}$ equals $\mathbf{W}^{\mathrm{T}}$. By defining the new input vector $\mathbf{e}_i(t) \in \mathbb{R}^p$ as the result of deflation:

$$\mathbf{e}_i(t) \stackrel{\mathrm{def}}{=} \mathbf{x}(t) - \sum_{k=1}^{i-1} y_k(t)\mathbf{w}_k, \quad i \geq 2, \tag{21}$$

$$e_1(t) \stackrel{\mathrm{def}}{=} \mathbf{x}(t), \tag{22}$$

the input–output relationship can be written as:

$$y_i(t) = \mathbf{w}_i^{\mathrm{T}}(t)\mathbf{e}_i(t), \tag{23}$$

and the learning equation reads:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta(y_i(t)\mathbf{e}_i(t) - y_i^2(t)\mathbf{w}_i(t)). \tag{24}$$

It is important to remark that Abbas–Fahmi's learning algorithm consists in a sequential utilization of Oja's rule for each neuron; thus the ith neuron sees the input signal deflated from the previous $i-1$ principal components, so that the ith principal component becomes the first principal component of the deflated signal. However, as in the GHA the Gram–Schmidt orthogonalization procedure is employed, which makes the reliable extraction of all components difficult.

2.5. The autoencoder network and RLS-PCA rule

Bannour and Azimi-Sadjadi introduced in Ref. [3] a PCA algorithm based on the RLS approach. It uses a sequential structure too, so only one neuron works at each step.

The neural network's structure is shown in Fig. 4. It is similar to the architecture utilized for SAMH (see Fig. 3), except that now the new input sequence $\mathbf{e}_i(t)$, defined above, is not used for the "deflation" but only in the updating operations. It can be better seen in the following equations that describe network's learning:

$$y_i(t) = \mathbf{w}_i^{\mathrm{T}}(t)\mathbf{x}(t), \tag{25}$$

$$K_i(t) = \frac{P_i(t)y_i(t)}{[1 + y_1^2(t)P_i(t)]}, \tag{26}$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + K_i(t)[\mathbf{e}_i(t) - y_i(t)\mathbf{w}_i(t)], \tag{27}$$

$$P_i(t+1) = [1 - K_i(t)y_i(t)]P_i(t), \tag{28}$$

where Kalman gain K_i plays the role of the learning step-size η and P_i comes from the inverse of the covariance of the ith neuron's output. The value of $P_i(0)$, for each neuron, is set, according to Ref. [3], to 0.5.

2.6. The cascade neural network and CRLS rule

In Ref. [5] Cichocki et al. proposed a neural approach to PCA that follows the SAMH and RLS-PCA ones, combining their advantages. The CRLS relies on an inherently sequential neural structure (Fig. 5), which is formally identical to the SAMH one (Fig. 3). In fact, while in SAMH the error signal is obtained by subtracting the data partially

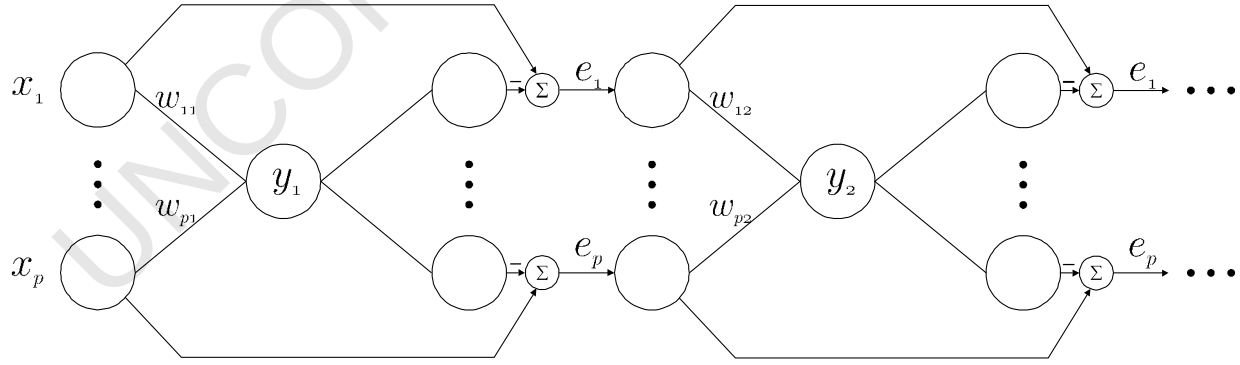

Fig. 5. Cascade neural network with the "deflation" mechanism for the CRLS.

```
for i = 1 to Num.Outputs
        set w_i(0)
        set η, P_i(0), ϑ_i(0) (if required)
        for d = 1 to MaxNum.Epochs
            CNT= 0;
            for t = 1 to Num.Patterns
                y_i(t) = input-output relationship
                w_i(t + 1) = weight update equation
                Update lateral connections (if any)
                Update K_i, P_i, ϑ_i (if required)
                if ||Δw_i(t)|| < ε then
                    CNT=CNT+1
                end if
            end for
            if CNT=Num.Patterns then
                Exit and jump to
                the next neuron
            end if
        end for
        Perform operations for computing
        the "deflation" (if used)
end for
```

Fig. 6. Code structure for the implementation of the learning algorithms.

reconstructed by means of previously extracted eigenvectors from the original data-sequence, in the CRLS the subtraction is between the previous error and the data reconstructed using only the last generated eigenvector (for the first neuron the error signal equals the original data stream).

It is clear how this "deflation" procedure is analogous to the one described about the SAMH algorithm. The input–output relationship (23) holds for the neurons in the CRLS network; also, two successive neurons in the cascade link by the equations:

$$\mathbf{e}_i \stackrel{\text{def}}{=} \mathbf{e}_{i-1}(t) - y_{i-1}\mathbf{w}_{i-1}, \tag{29}$$

$$\mathbf{e}_1(t) \stackrel{\text{def}}{=} \mathbf{x}(t). \tag{30}$$

The learning equations for the ith neuron read:

$$\vartheta(t) = \vartheta(t-1) + y_i^2(t), \tag{31}$$

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \frac{y_i(t)}{\vartheta(t)}[\mathbf{e}_i(t) - \mathbf{w}_i(t)y_i(t)], \tag{32}$$

where $1/\vartheta$ plays the role of η, and $\vartheta(0)$ sets to the variance of all the error sequences every time a new neuron starts learning, according to Ref. [5].

3. Some implementation details

In order to perform a fair comparison among the mentioned algorithms, we decided to utilize for all the algorithms the *sequential* structure that, being natural for the SAMH, RLS-PCA and CRLS, is not natural for the other algorithms. However for the APEX, and thus for the ψ-APEX algorithms, a sequential version exists (see Sections 2.2 and 2.3), even if their distinctive feature is the possibility of a parallel extraction of a predefined number of principal components. For the GHA, Sanger in Ref. [29] does not cite this possibility explicitly; however, the extension to the sequential case is quite straightforward (see, e.g. Bannour and Azimi-Sadjadi [3] and Cichocki, Kasprzak and Skarbek [5]). This decision followed from testing both the versions of these algorithms and observing that the parallel version is less efficient: in fact, even if each neuron starts to lean immediately, in practice it begins correct convergence

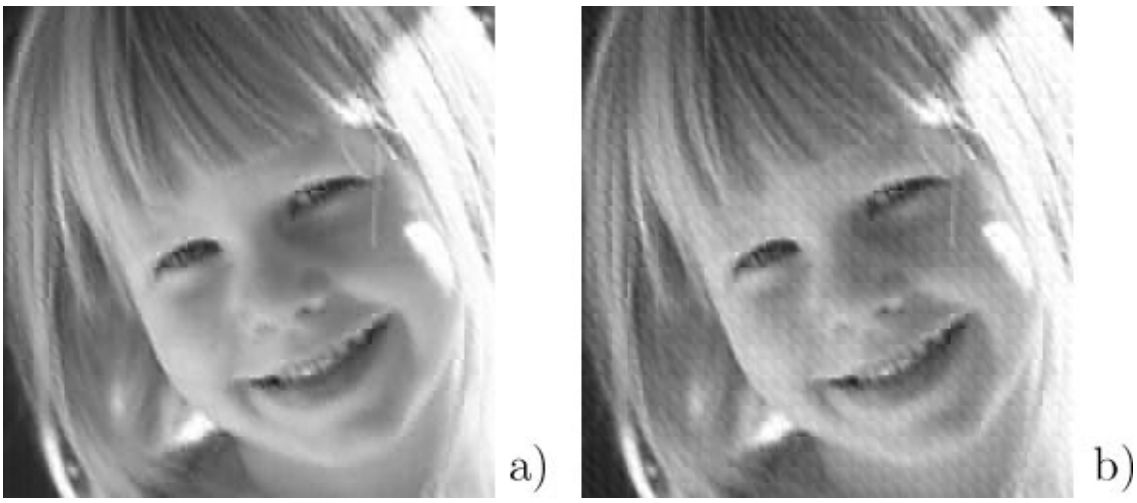

Fig. 7. Compression results, with and without variable quantization (8 outputs and a maximum of 40 epochs per component): (a) without variable quantization (SNR = 28.39 dB, CR = 1 bpp); and (b) with variable quantization (SNR = 27.09 dB, CR = 0.625 bpp).

only when the previous neurons have converged first; this leads to wasting much CPU time.

In order to implement the algorithms and to make the neurons' learning sequential, we employed the code structure shown in Fig. 6. Note that it implies that only when *all* the $\|\Delta \mathbf{w}_i(t)\|$, computed for each pattern of any whole epoch, satisfy the termination condition, the corresponding neuron stops learning, while in Ref. [5] the stability of only one $\mathbf{w}_i(t)$ is sufficient to terminate the learning phase of ith neuron and to make successive neuron learning. This is because when some patterns are similar it is highly probable that a single $\mathbf{w}_i(t)$ keeps almost constant, which does not imply convergence.

For implementing the algorithms we used a neural network with 64 inputs and a variable number of outputs. The number of inputs comes from the 8×8 blocks of the image that are used as training samples, with the image being scanned from left to right, from top to bottom (the sample blocks do not overlap). Also, as iterations measure, we define an *epoch* as a complete block-by-block scanning of the whole image. Note that in this way the number of patterns associated to each epoch depends on the image size; for instance, an 8×8 block-by-block scanning of a 256×256 image corresponds to an epoch of 1024 learning patterns, while an epoch for a 512×512 image corresponds to 4096 patterns; as a consequence, the number of epochs necessary for an algorithm to converge also depends upon the size of the image that the algorithm should compress.

We used images with 256 gray-levels (8 bits) and normalized the input values from the discrete range 0–255 to the interval [0, 1]; then we computed and subtracted the mean value of each subimage, storing it for further decompression operations.

In the compression operations we may introduce a variable quantization allowing to assign a higher number of bits to the first components and a lower number to the last ones, depending on the values of their variances.

As performance index, decompression signal-to-noise-ratio (SNR) is employed. It is defined as:

$$SNR = 10 \log_{10} \frac{\sum_{i=1}^{N_{\text{row}}} \sum_{j=1}^{N_{\text{col}}} I_{ij}^2}{\sum_{i=1}^{N_{\text{row}}} \sum_{j=1}^{N_{\text{col}}} (I_{ij} - \hat{I}_{ij})^2} (dB), \tag{33}$$

where I is the original image, $\hat{I}$ is the reconstructed one and $N_{\text{row}} \times N_{\text{col}}$ is the support size. About the interpretation of numerical results, it is worth noting that values greater than 25–30 dB of SNR for an image are such that the reconstructed versions may be visually indistinguishable one from another; also, when about 64 principal components are used to represent a 64-pixel block, clearly the SNR reaches very high values (corresponding to very low compression rates), provided that the weight-matrix is orthogonal, regardless of the adherence of weight-vectors to true eigenvectors.

4. Experimental results

In order to assess the performances of the considered algorithms, in this section simulation results are presented and discussed. In Section 4.1 some preliminary issues are presented about the effect of compressed image coding by variable quantization; Section 4.2 is devoted to numerical performance assessment on different categories of pictures, while in Section 4.3 the computational complexity of the considered algorithms is evaluated; Section 4.4 presents a discussion on the generalization ability of the networks which allows compressing images by means of the network 'computed' over a prototype image; in Section 4.5 the use of an algorithm allowing for principal subspace estimation and the compression of colored images by PCA algorithms are briefly addressed.


a)


b)

Fig. 8. Reference images: (a) "Child" (256 × 256); and (b) "Lenna" (512 × 512).

Table 1
Comparison of the algorithm performances: number of epochs necessary for the first 8 neurons to converge (maximum limit fixed to 40), image: "Lenna". SNR ratio achieved upon reconstruction operated on the basis of the first 8 components

Algorithm	No. of PCs								SNR (dB)
	1	2	3	4	5	6	7	8	
GHA	40	40	40	40	40	40	40	40	25.82
APEX	40	40	40	40	40	40	40	40	5.70
0-APEX	40	40	40	40	40	40	40	40	25.87
$\|y\|$-APEX	40	40	40	40	40	40	40	40	25.78
y^2-APEX	40	40	40	40	40	40	40	40	25.89
SAMH	40	40	40	40	40	40	40	40	25.91
RLS-PCA	2	11	13	15	19	29	40	40	25.94
CRLS	2	11	13	15	19	29	40	40	25.92

4.1. Compressed image coding by variable quantization

In order to perform variable quantization based upon Sangers' paper [29] we compute the log of the variances of the principal components and linearly normalize the results between 4 and 8 as we observed that associating a number of bits ranging from a maximum of 8 for the first principal component to a minimum of 4 for the last one allows keeping almost non-degraded the SNR with respect to the constant quantization case (i.e 8 bits for each component). An index which in general measures the achieved compression ratio (CR) can be written as:

$$CR = \frac{\sum_{i=1}^{m} b_i}{64} \text{ bits per pixel (bpp)}, \tag{34}$$

where b_i is the number of bits associated to the ith component. CR represents the average number of bits required for coding each pixel. For a meaningful comparison, it is worth noting that the uncompressed images require 8 bits per pixel to be represented, thus they have CR = 8 bpp. This is thus the reference value: smaller values of CR denote better compression rates.

An example is displayed in Fig. 7 where the result of compression of the image "child" (see Fig. 8a for the original image) with and without variable quantization is illustrated. As we can note, there is only about 1 dB of degradation in the case of variable quantization at the expense of a light windowing effect.

We mention that recently a merged compression/quantization procedure has been proposed [2], which adapts the quantization algorithm to the image and makes this operation better suited to available data.

4.2. Numerical performances evaluation

To evaluate the performances of the PCA algorithms, we consider both the convergence speed, measured as the number of epochs necessary for each neuron to extract the corresponding component, and the quantitative difference between the original and reconstructed images, showing the SNR values obtained by running the different algorithms.

About the kinds of image we used for testing the cited principal component networks, we oriented our simulations to: (1) natural scenes, (2) textures, (3) artifacts (printed text), and (4) high-frequency images. For each category we repeated the same simulations in order to be able to form an opinion about the best performing algorithm for each class. Finally, the obtained results, for each algorithm, have been averaged over the four categories.

Unless otherwise indicated, the maximum number of epochs per component equals 40, the learning rate (if any) is 0.01, and the threshold ϵ was set to 2×10^{-4}, which ensures the maximum attainable value of the SNR.

Fig. 9. Image sequence obtained with CRLS algorithm: (a) 1PC, SNR = 18.09 dB, CR = 0.125 bpp; (b) 2PCs, SNR = 20.48 dB, CR = 0.25 bpp; (c) 3PCs, SNR = 21.89 dB, CR = 0.375 bpp; (d) 4PCs, SNR = 23.01 dB, CR = 0.05 bpp; (e) 5PCs, SNR = 24.12 dB, CR = 0.625 bpp; (f) 6PCs, SNR = 24.79 dB, CR = 0.75 bpp; (g) 7PCs, SNR = 25.32 dB, CR = 0.875 bpp; and (h) 8PCs, SNR = 25.94 dB, CR = 1 ***bpp***.


a)
b)
c)
d)
e)
f)
g)
h)

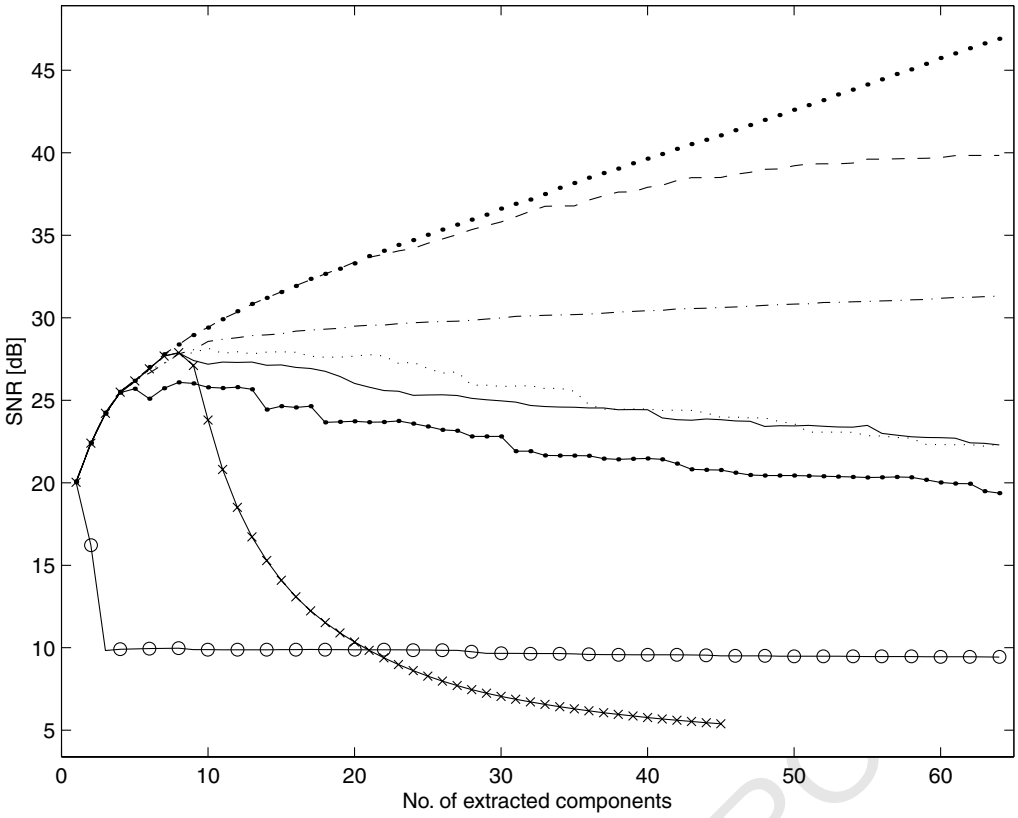

Fig. 10. SNR value versus the number of extracted components (image: "Child"). Legend: CRLS = point, RLS = dashed, GHA = dot-dashed, y^2-APEX = dotted, 0-APEX = solid, $|y|$-APEX = solid-point, SAMH = solid-×-mark, APEX = solid-circle.

4.2.1. Performances on natural scenes

For these tests the 512×512 "Lenna" image and 256×256 "Child" image were used; they are shown in Fig. 8.

In the first simulation, we set the number of outputs to $m = 8$; the results are shown in Table 1. The best algorithms in this test resulted to be the CRLS and RLS-PCA, which achieve a higher SNR value with a smaller number of epochs. The other algorithms are not able to converge quickly (i.e. within the 40 epochs allowed for each neuron).

A visual example of the quality of the reconstructed images, obtainable as the number of extracted principal component increases, is reported in Fig. 9; these results have been obtained by running the CRLS network.

Results show that the algorithms are practically equivalent when a few principal components are extracted. Thus, a more revealing comparison is the extraction of all the 64 components. In Fig. 10 we can see the results for the image "Child", and in Fig. 11 for "Lenna". The RLS-PCA and CRLS exhibit better performances.

4.2.2. Performances on textures

Fig. 13 reports the performances of the eight considered PCA algorithms on the texture shown in Fig. 12. Also, Fig. 15 reports the performances of the PCA algorithms on the texture shown in Fig. 14.

For both textured images the CRLS, RLS-PCA and GHA exhibit a good behavior, the algorithms belonging to the ψ-APEX class perform rather similarly among them, while old APEX and SAMH show a greater difficulty to extract the principal components.

4.2.3. Performances on a printed text

Fig. 17 reports the performances of the PCA algorithms on the printed text image shown in Fig. 16.

In this case the GHA algorithm performs the best with 49 dB of achieved SNR, the RLS algorithm follows with 47.5 dB, and the CRLS with 46.8 dB; also, 0-APEX shows a good performance with 40.5 dB. It is worth noticing that in spite of the gaps among the SNR values achieved by the algorithms, at these very high signal-to-noise ratios, visually the reconstructed images look similar, thus no practical differences exist among the behavior of the four cited best performing algorithms. In this case only the last four principal components make the difference among the networks.

4.2.4. Performances on high-frequency images

An interesting experiment concerns high-frequency image compression by the principal component networks. The image used for test is a picture of a polyester slab taken from the Carnegie-Mellon University database, depicted in Fig. 18.

In order to confirm it has very high frequency components, in Fig. 19 the amplitude spectrum of the image was

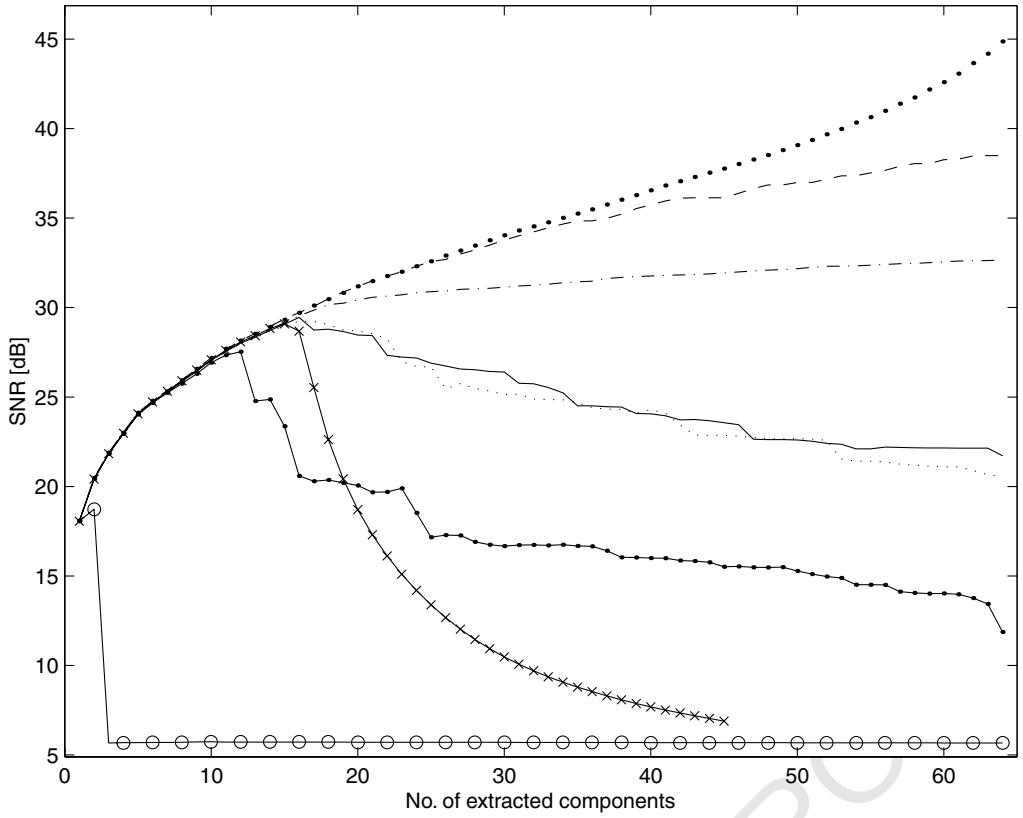

Fig. 11. SNR value versus the number of extracted components (image: "Lenna"). Legend: CRLS = point, RLS = dashed, GHA = dot-dashed, y^2-APEX = dotted, 0-APEX = solid, $|y|$-APEX = solid-point, SAMH = solid-×-mark, APEX = solid-circle.

reported as obtained by applying the bidimensional discrete Fourier transform (DFT). In the figure the normalized frequencies ω_x and ω_y range in $[-\pi, +\pi]$, where the central value 0 corresponds to low frequency and edge values $\pm\pi$ correspond to high frequency. When compared to the spectrum of natural image "Lenna", the "Polyester" clearly exhibits a spreader spectrum which has strong components in the boundary of spectra support.

In Fig. 20, the performances of the PCA algorithms on the high-frequency image "Polyester" are illustrated. As the characteristics of the high-frequency image are very different from those pertaining to, e.g. natural scenes, it is intuitive to expect different numerical performances. Particularly it seems that the algorithms all behave poorly, as also confirmed by results reported in Table 2; the low values of achieved SNR's with eight components extracted is particularly worth noting.

4.2.5. Average numerical performances

The results of numerical simulations shown in the preceding sections, related to four categories of images, reveal that the performances of the principal component algorithms considered in the present comparison depend upon the characteristics of the images to be compressed; also, they allow us forming an opinion about the best performing one for each category.

In this section we aim at presenting a global result about numerical performances, by showing the SNR curves attained by each algorithm averaged over the four categories. With reference to the previous simulations, we chosen the following images to represent the four categories: "Child", "Metal", "Text" and "Polyester". Fig. 21 shows the obtained results.

The CRLS algorithm and the RLS-PCA definitely exhibit the best numerical performances in our simulations, granting relatively higher values of SNR ratios with respect to the other algorithms, which correspond to absolutely excellent results. It should however be remarked that when only few principal components are extracted, the cited algorithms perform rather similarly, while the difference among their behavior clearly emerges when several components are looked upon.

4.3. Computational complexity evaluation

The computational complexity of the considered algorithms is evaluated by measuring the number of floating point operations (flops) required during code running. The results have been reported in Table 3 and are referred to the image "Child" and $m = 4$ extracted components. As the epochs for each component (for each algorithm) have been set to 10 for a fair comparison, a total of 40 learning epochs were run. It is worth noting that having kept constant m and the number of

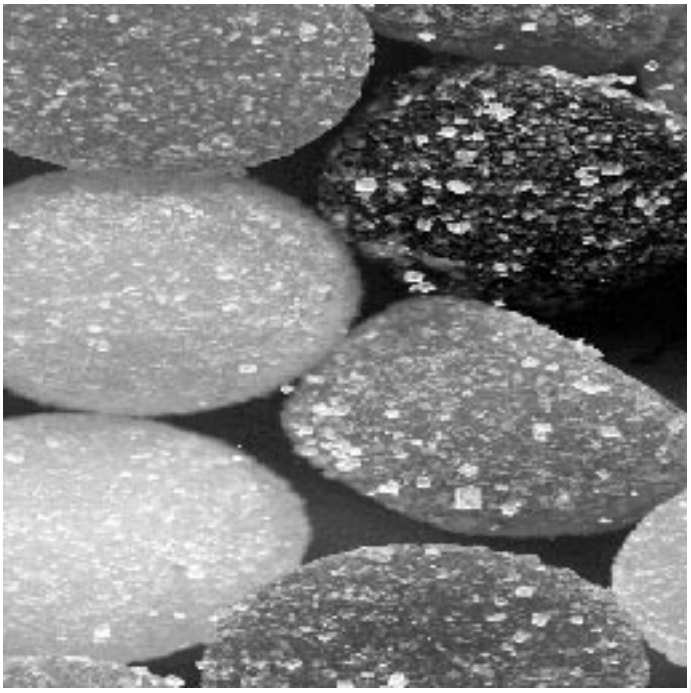

Fig. 12. Sample of a texture (image "Charms").

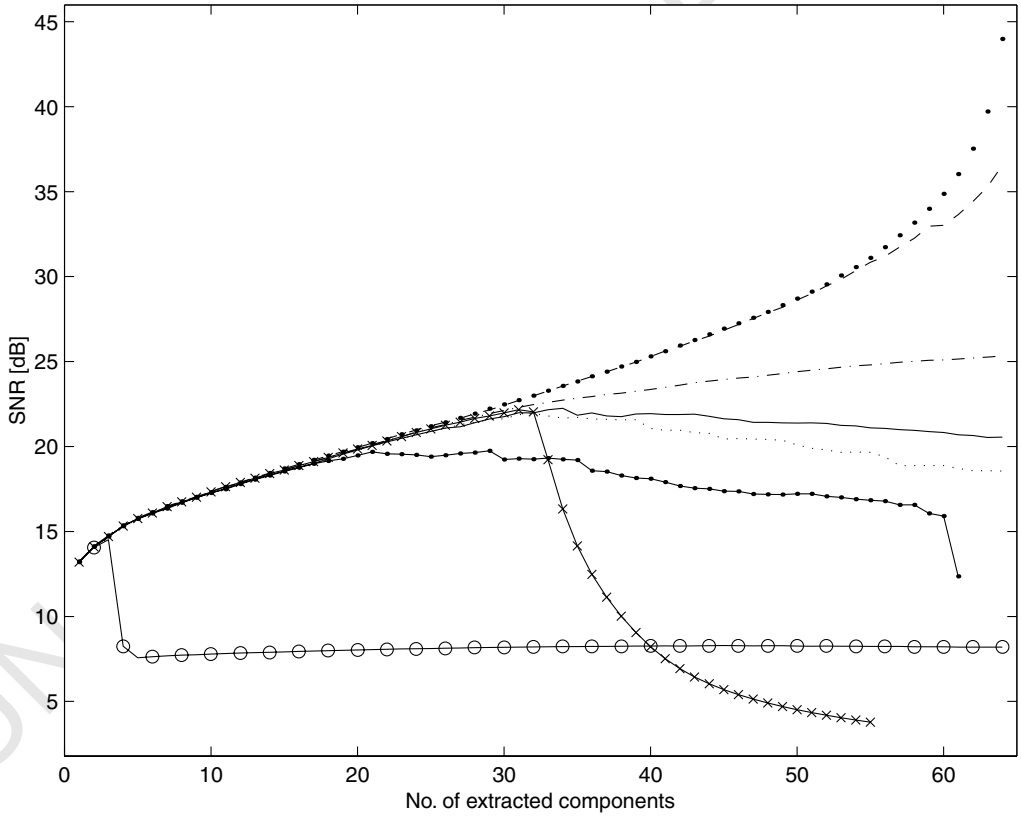

Fig. 13. SNR value versus the number of extracted components (image: "Charms"). Legend: CRLS = point, RLS = dashed, GHA = dot-dashed, y^2-APEX = dotted, 0-APEX = solid, $|y|$-APEX = solid-point, SAMH = solid-× -mark, APEX = solid-circle.

Fig. 14. Sample of a texture (image "Metal").

ring to the code structure of Fig. 6, as the total number of times that the operations explained in the most inner cycle are executed, that is No. of outputs × No. of epochs × No. of patterns. Table 3 also shows the computation times[2] required for the algorithms to run on a common platform (450 MHz clock, 64 Mb RAM).

As confirmed by simulation results, the SAMH, RLS-PCA and CRLS have a lighter structure with respect to the other algorithms; this can be explained by recognizing that in each cycle the GHA, APEX and ψ-APEX use the matrix $\tilde{\mathbf{W}}$ (whose size is $64 \times (i-1)$ for the ith neuron), therefore as the index of the currently extracted component grows, the computational complexity grows, too. The advantage of the CRLS with respect to SAMH and RLS-PCA is given by the use of the "deflation" procedure, i.e. in the CRLS the current neuron's weight-vector $\mathbf{w}_i$ only is used. About computation times, it is worth noting that the reported numbers account for several concurring effects as pure computation and memory allocation/de-allocation.

Another discriminating element in the comparison is the storage requirement of each algorithm, i.e. the required

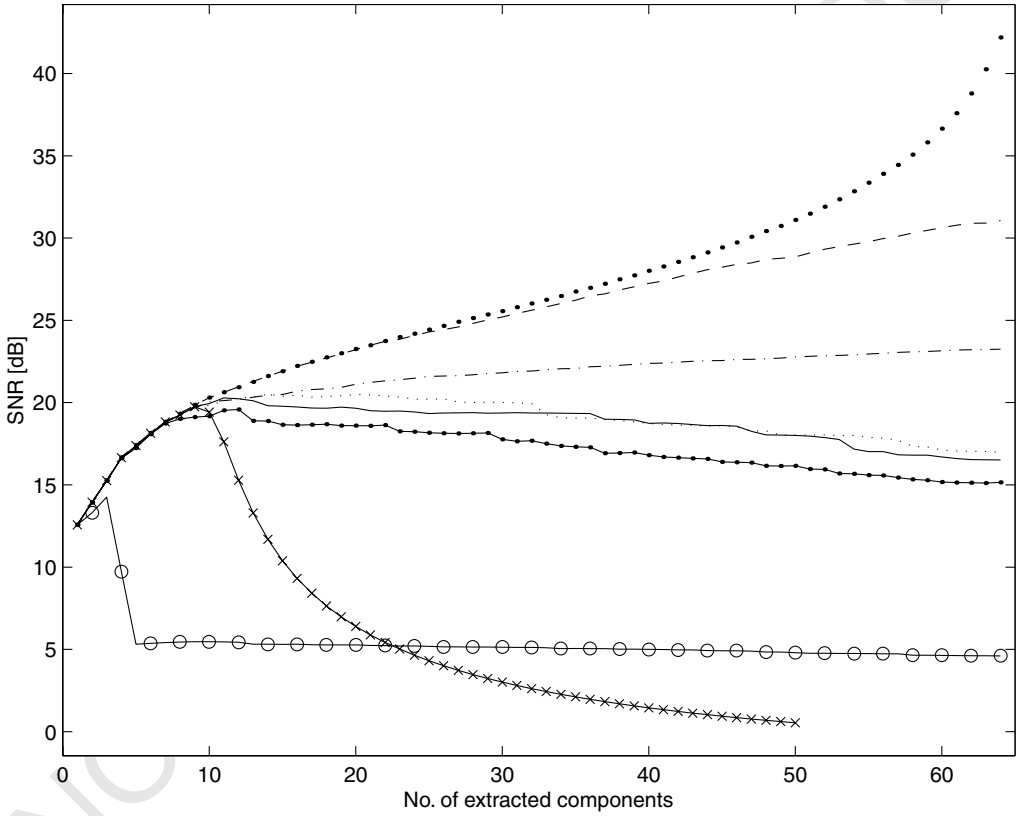

Fig. 15. SNR value versus the number of extracted components (image: "Metal"). Legend: CRLS = point, RLS = dashed, GHA = dot-dashed, y^2-APEX = dotted, 0-APEX = solid, $|y|$-APEX = solid-point, SAMH = solid-×-mark, APEX = solid-circle.

iterations, the reported results are actually independent of the images' size. The table shows the total number of required flops, and the average number of flops per iteration; here the number of iterations is defined, refer-

[2] They refer to an interpreted programming language, and were found to have about two times the magnitude order of the corresponding versions implemented with a compiled programming language.

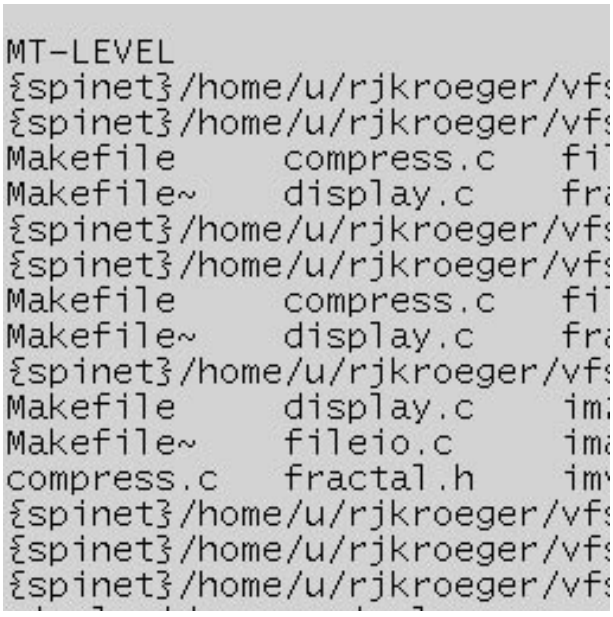

Fig. 16. Sample of a printed text (image "Text").

memory capacity for storing the floating point variables involved in the computer implementation of the neural topologies along with the learning procedures. Estimates of the memory requirements are given in Table 4. Once again, the CRLS algorithm exhibits the best features.

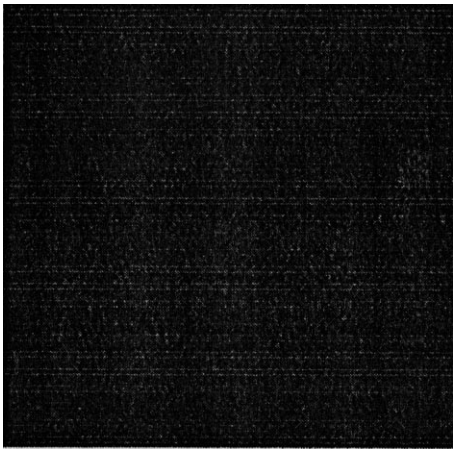

Fig. 18. A real-world image (picture of a polyester slab) with high-frequency components (image: "Polyester").

4.4. On generalization ability

An important feature of neural networks is the generalization ability, which is generally referred to as the property of a network to react correctly to input patterns never seen before, i.e. to patterns that do not belong to the training set. In fact, generalization is a good measure of a learnt network's suitability to solve a problem, because it reveals the degree of adherence, attained by the network, to the

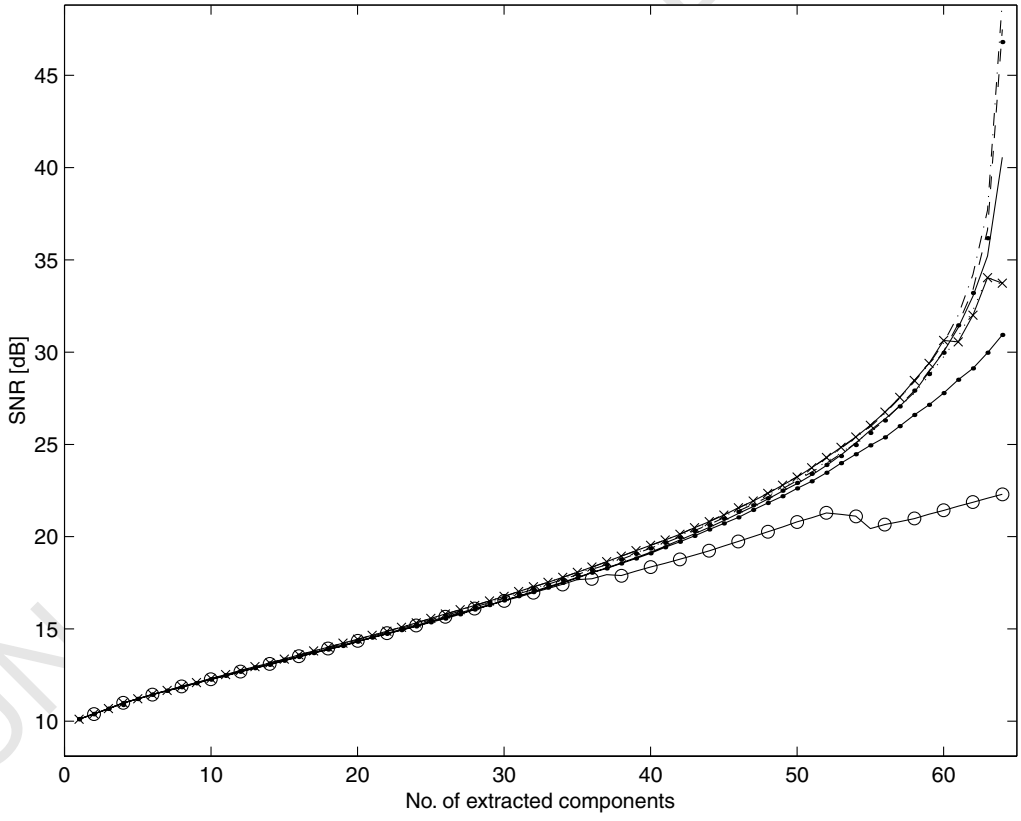

Fig. 17. SNR value versus the number of extracted components (image: "Text"). Legend: CRLS = point, RLS = dashed, GHA = dot-dashed, y^2-APEX = dotted, 0-APEX = solid, $|y|$-APEX = solid-point, SAMH = solid-× -mark, APEX = solid-circle.

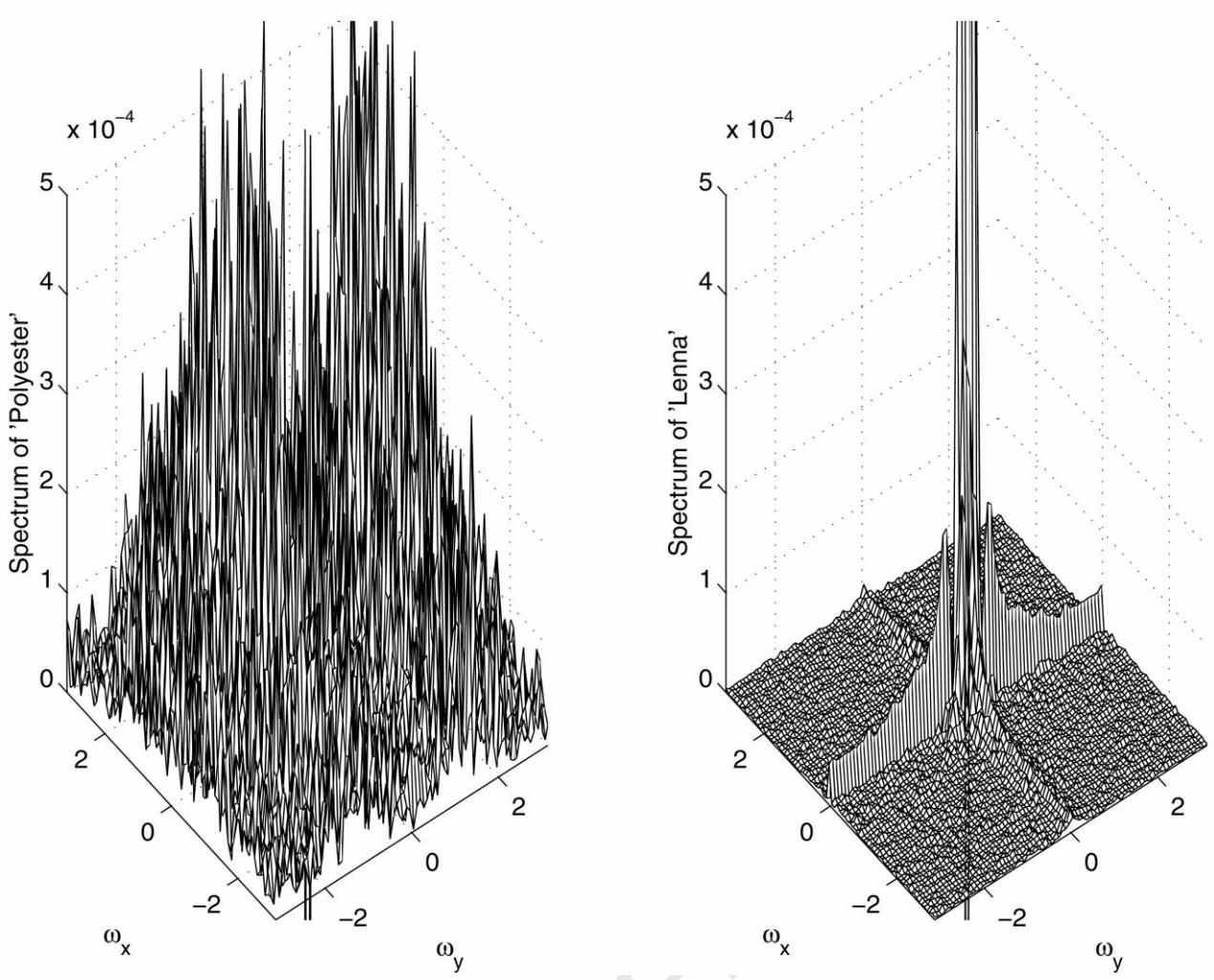

Fig. 19. The two-dimensional spectra of "Polyester" image and of "Lenna" image for comparison, obtained by the use of bidimensional DFT.

physical phenomena underlying the data [4,32]. Generalization is also closely related to the concept of interpolation by a learnt model, built up from samples of the input space, which allows to treat properly the data situated in a part of the input space where there were learning data.

In the context of the first experiments on image compression by principal component neural networks, it has been observed experimentally that different images belonging to the same category (e.g. natural scenes) give raise to PCA networks with very similar set of weights [29]. This suggests that the linear PCA model is strongly plausible and representative of some categories of pictures, thus the generalization ability would hold for them, naturally providing a fast compression technique. In fact, by exploiting this property, we can think to train a network on a "prototype" image and to use the obtained network to compress a picture of the same kind.

As demonstrated by the examples in Fig. 22, the results are excellent. The advantage is the high speed of this operation, i.e. the compression phase is very fast with respect to network re-learning. As an interesting result, the SNR of the image "Child" is higher than the SNR of image "Lenna". This phenomenon may be explained by observing that the image "Lenna" is more complicated and exhibits much more details than "Child" does, thus the network trained on the first one contains much more information about natural scenes, and is therefore a good candidate to represent a prototype for that class.

Another interesting example is shown in Fig. 23, where the results of compression/decompression are illustrated for two environmental images and an artificial one (a printed text) coming from the Waterloo University repository; these results have been obtained by the CRLS principal component network trained on the "Lenna" image. The visual result for the first two pictures is good; however, the reported SNR values reveal that the quality of reconstructed images degrades with respect to the original ones; about the printed text, the learnt network clearly does not work properly on it, confirming that the generalization performance of a learnt principal component network depends on the category that the learning data belong to.

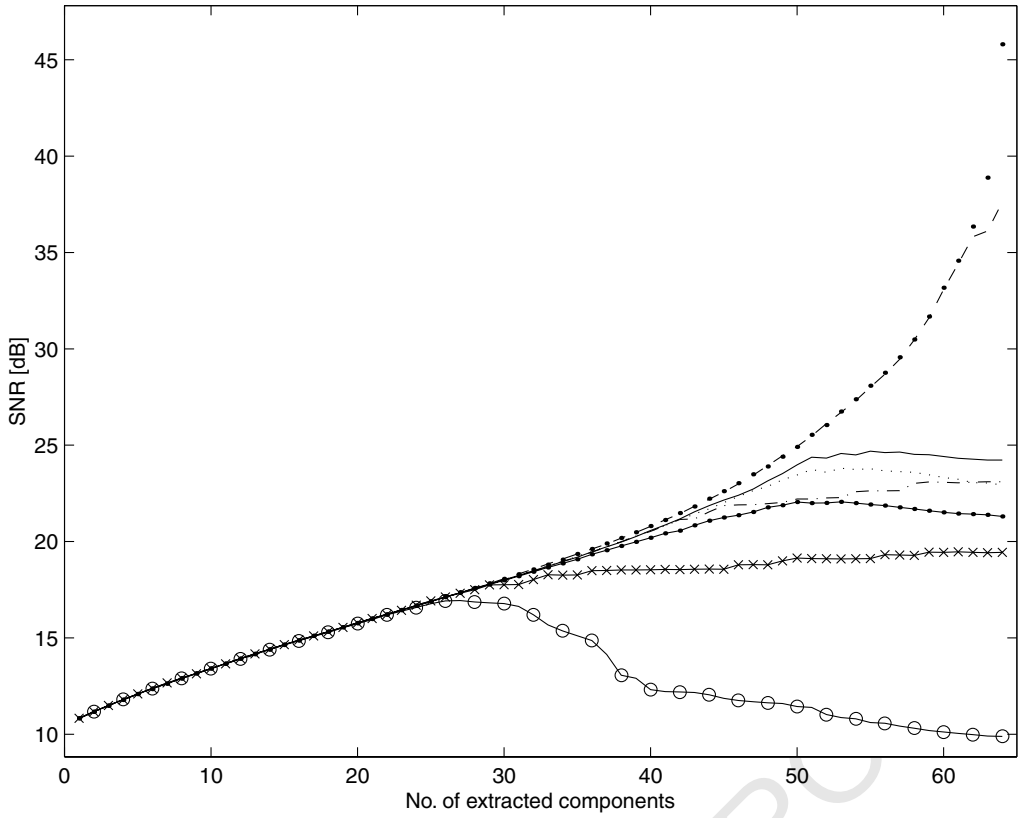

Fig. 20. SNR value versus the number of extracted components (image: "Polyester"). Legend: CRLS = point, RLS = dashed, GHA = dot-dashed, y^2-APEX = dotted, 0-APEX = solid, $|y|$-APEX = solid-point, SAMH = solid-× -mark, APEX = solid-circle.

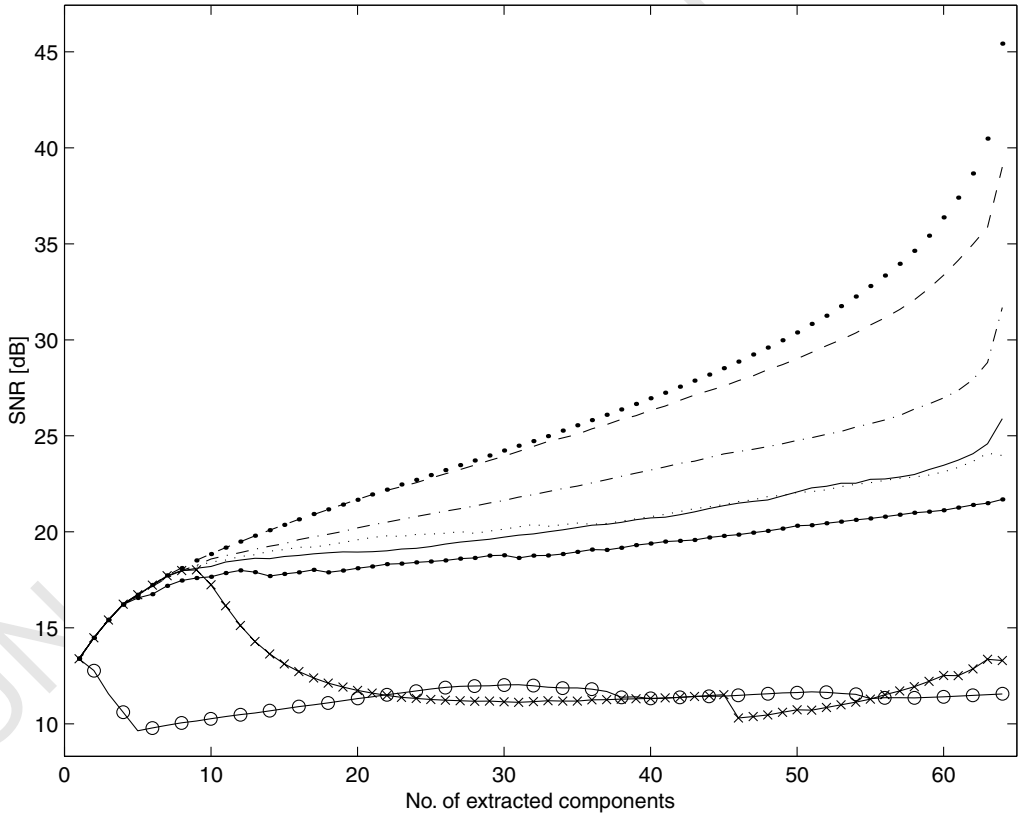

Fig. 21. Average SNR value versus the number of extracted components (four selected images). Legend: CRLS = point, RLS = dashed, GHA = dot-dashed, y^2-APEX = dotted, 0-APEX = solid, $|y|$-APEX = solid-point, SAMH = solid-× -mark, APEX = solid-circle.

Fig. 22. Example of the generalization property (algorithm: CRLS, 8 outputs and a maximum of 40 epochs per component): (a) image "Lenna" (SNR = 25.94 dB); (b) image "Child" (SNR = 28.36 dB); (c) image "Lenna" compressed with the network computed in (b) (SNR = 25.07 dB); and (d) image "Child" compressed with the network computed in (a) (SNR = 27.77 dB).

Table 2
Comparison of the algorithm performances: number of epochs necessary to the first 8 neurons to converge (maximum limit fixed to 40), image: "Polyester". SNR ratio achieved upon reconstruction operated on the basis of the first 8 components

Algorithm	No. of PCs								SNR (dB)
	1	2	3	4	5	6	7	8	
GHA	40	40	40	40	40	40	40	40	12.9163
APEX	40	40	40	40	40	40	40	40	12.9045
0-APEX	40	40	40	40	40	40	40	40	12.9154
$\|y\|$-APEX	40	40	40	40	40	40	40	40	12.9147
y^2-APEX	40	40	40	40	40	40	40	40	12.9093
SAMH	40	40	40	40	40	40	40	40	12.9112
RLS-PCA	24	21	22	25	30	29	24	25	12.8948
CRLS	23	33	28	28	30	25	28	25	12.8921

Table 3
Complexity comparison: total flops and average flops per iteration, and computation times on a 256×256 image, 4 outputs. Platform: 450 MHz/96 Mb machine

Algorithm	Flops	Ave. flops/iter.	Time (for 40 epochs) (s)
GHA	43,950,914	1073	15.49
APEX	29,706,240	725	16.86
0-APEX	29,748,004	726	20.32
$\|y\|$-APEX	29,746,927	726	21.20
y^2-APEX	29,790,633	727	20.77
SAMH	20,226,882	494	11.97
RLS-PCA	20,513,602	501	15.76
CRLS	17,686,854	432	13.19

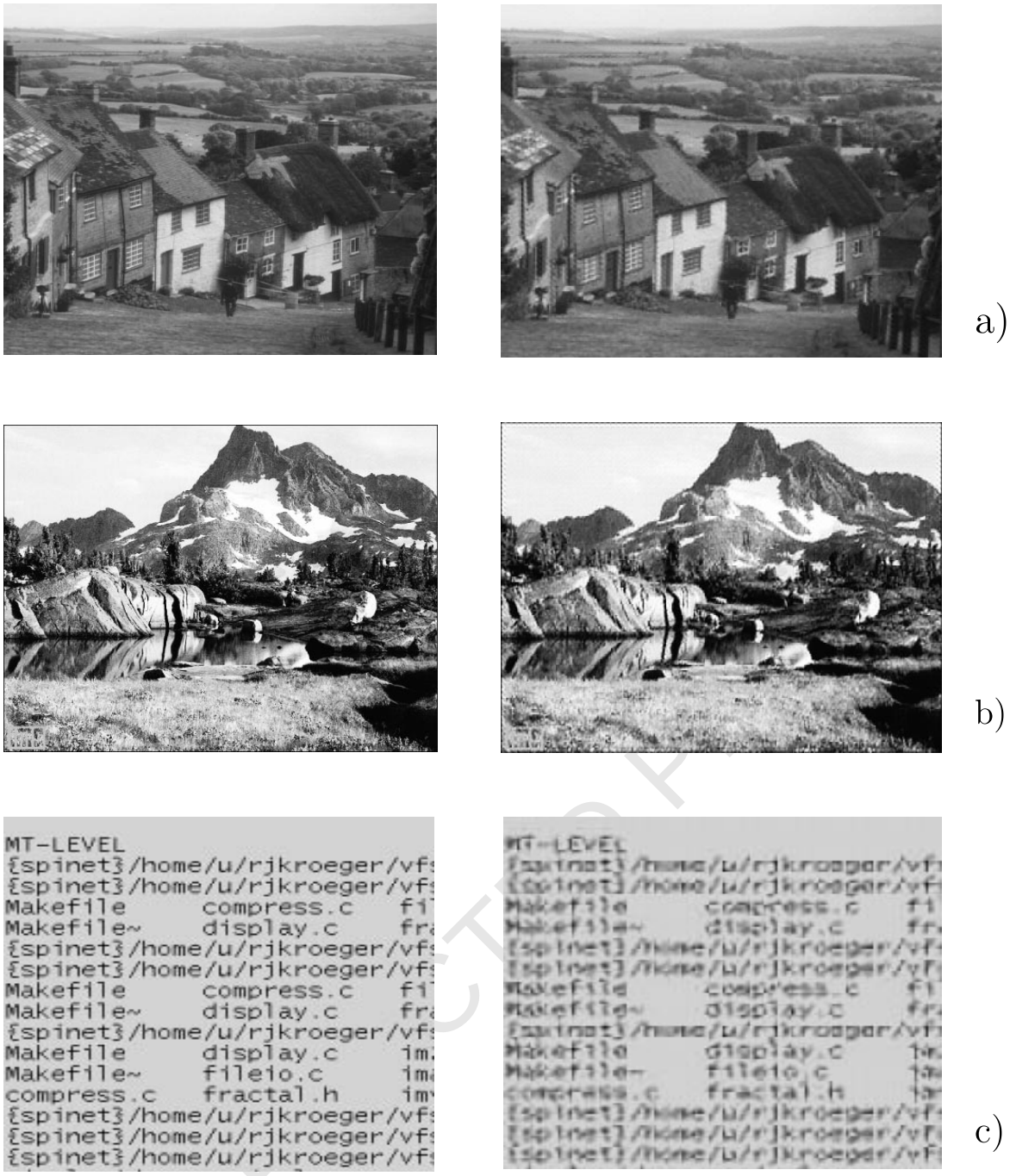

Fig. 23. Example of generalization (algorithm: CRLS, 8 outputs and a maximum of 40 epochs per component, trained on "Lenna" image): (a) original/reconstructed image "Goldhill" (SNR = 23.22 dB); (b) original/reconstructed image "Mountain" (SNR = 14.57 dB); (c) original/reconstructed image "Text" (SNR = 11.31 dB).

4.5. Further experimental results

Computer simulations on image compression have been performed by implementing Oja's principal subspace rule [24] too. This algorithm is not able to extract directly the selected eigenvectors of data covariance matrix, but only a linear combination of them, which is a rotation; thus it extracts vectors spanning the principal subspace of input space. Moreover, it has been observed experimentally (see for instance Ref. [31]) that such algorithm exhibits a kind of *whitening* effect, that is, the eigenvector rotation makes the extracted components have similar variance. As a consequence, the components are not ordered at the network's outputs, thus the coding scheme cannot be employed; furthermore, the extracted components all contain a fraction of information about the compressed image and none of them can be neglected.

As another interesting matter, in order to extend the

Table 4
Comparison of the algorithms' storage requirements as functions of networks' input and output sizes (a memory unit corresponds to the memory capacity required for storing a floating point number)

Algorithm	Memory units
GHA	$2p + (p + 1)(m + 1)$
APEX	$4p + 2 + (p + 1)(m - 1) + (3/2)m(m - 1)$
ψ-APEX	$4p + 3 + (p + 3)(m - 1) + (3/2)m(m - 1)$
SAMH	$2p + (2p + 1)(m - 1)$
RLS-PCA	$3p + 4 + (2p + 1)(m - 1)$
CRLS	$3p + 3$

considered algorithms to colored image compression, it is necessary to take into account that colors are coded and subdivided in gray-level subimages, which can be compressed and decompressed separately. Perhaps, the most known coding technique is the RGB one; it is an additive model in that any color is obtained by summing to black different degrees of red, green and blue lights. A subtractive model is the CMYK one, which starting from white generates the different colors by subtracting tones of cyan, magenta, yellow and black. Other well-known models are the HSB (hue, saturation, brightness), HLS, YIQ and so forth. In our experiments we employed the RGB and HSB coding subdivision techniques and compared the obtainable compression results. Best results have been obtained by operating on RGB subimages; likely, this may be explained by observing that the R, G, B subimages have relatively smooth shapes (that are to be preferred when using PCA techniques, cf. experiments on high-frequency images) in opposition to subchannel H, which is responsible for the worst results connected to the use of HSB coding.

5. Conclusions

The aim of this paper was to present recent developments in image compression by principal component neural networks. The main results can be summarized as follows:

- among the considered algorithms the best results about image quality compression/decompression have been obtained with the CRLS algorithm;
- extracting up to eight principal components, the considered algorithms behave similarly about the quality of compression and bit-per-pixel rate; however, the SAMH, RLS-PCA and CRLS exhibit the lowest computational complexity;
- by exploiting principal components ordering, automatically provided by the sequential extraction, compressed image coding can be performed by optimal bit allocation; the variable quantization implies small SNR degradation at the expense of a light windowing effect;
- by exploiting the network's generalization ability fast image compression can be performed by training a network on prototype natural images corresponding to the category that the image to be compressed belong to, in order to obtain the best performance.

Acknowledgements

This research was partially supported by the Italian MURST: contact author S.F.

References

[1] H.M. Abbas, M.M. Fahmy, Neural model for Karhunen–Loéve transform with application to adaptive image compression, IEE Proc. I, Commun. Speech Vis. 140 (2) (1994) 135–143.
[2] M. Breazu, B.J. Beggs, G. Toderean, I.P. Mihu, Merging the transform step and the quantization step for Karhunen–Loève transform based image compression, in Proceedings of International Joint Conference on Neural Networks, vol. 5, 2000, pp. 483–488.
[3] S. Bannour, M.R. Azimi-Sadjadi, Principal component extraction using recursive least squares learning, IEEE Trans. Neural Networks 6 (2) (1995) 457–469.
[4] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Oxford, 1995.
[5] A. Cichocki, W. Kasprzak, W. Skarbek, Adaptive learning algorithm for principal component analysis with partial data, Proc. Cybernetics Syst. 2 (1996) 1014–1019.
[6] M.C.F. De Castro, F.C.C. De Castro, J.N. Amaral, P.R.G. Franco, A complex valued Hebbian learning algorithm, in Proceedings of the International Joint Conference on Neural Networks (IEEE-IJCNN), 1998, pp. 1235–1238.
[7] K.I. Diamantaras, S.-Y. Kung, Principal Component Neural Networks: Theory and Applications, Wiley, New York, 1996.
[8] S. Fiori, Blind separation of circularly distributed source signals by the neural extended APEX algorithm, Neurocomputing 34 (1–4) (2000) 239–252.
[9] S. Fiori, F. Piazza, A. General, Class of ψ-APEX PCA neural algorithms, IEEE Trans. Circuits Syst. I 47 (9) (2000) 1394–1398.
[10] S. Fiori, A. Uncini, A unified approach to laterally-connected neural nets, in Proceedings of the IX European Signal Processing Conference (EUSIPCO), vol. 1, 1998, pp. 379–382.
[11] S. Fiori, An experimental comparison of three PCA neural networks, Neural Process. Lett. 11 (3) (2000) 209–218.
[12] S. Haykin, Neural Networks, MacMillan, New York, 1994.
[13] A.K. Jain, Fundamentals of Digital Image Processing, Prentice-Hall, Englewood Cliffs, NJ, 1989.
[14] J. Karhunen, Optimization criteria and nonlinear PCA neural networks, in Proceedings of the International Joint Conference on Neural Networks (IEEE-IJCNN), 1994, pp. 1241–1246.
[15] J. Karhunen, Stability of Oja's subspace rule, Neural Computation 6 (1994) 739–747.
[16] J. Karhunen, J. Joutsensalo, Learning of non-linear principal component subspace, in Proceedings of the International Joint Conference on Neural Networks (IEEE-IJCNN), 1993, pp. 2409–2412.
[17] J. Karhunen, J. Joutsensalo, Nonlinear generalizations of principal component learning algorithms, in Proceedings of the International Joint Conference on Neural Networks (IJCNN), 1993, pp. 2599–2602.
[18] J. Karhunen, J. Joutsensalo, Representation and separation of signals using nonlinear PCA type learning, Neural Networks 7 (1) (1994) 113–127.
[19] J. Karhunen, J. Joutsensalo, Generalizations of PCA, optimization problems, and neural networks, Neural Networks 8 (4) (1995) 549–562.
[20] J. Karhunen, L. Wang, R. Vigario, Nonlinear PCA type learning for

source separation and independent component analysis, in Proceedings of the IEEE International Conference on Neural Network (ICNN), 1995, pp. 995–1000.
[21] S.Y. Kung, K.I. Diamantaras, J.S. Taur, Adaptive principal component extraction (APEX) and applications, IEEE Trans. Signal Processing 42 (5) (1994) 1202–1217.
[22] G. Mathew, V. Reddy, Orthogonal eigensubspace estimation using neural networks, IEEE Trans. Signal Processing 42 (1994) 1803–1811.
[23] E. Oja, A simplified neuron model as a principal component analyzer, J. Math. Biol. 15 (1982) 267–273.
[24] E. Oja, Neural networks, principal components, and subspaces, Int. J. Neural System 1 (1989) 61–68.
[25] E. Oja, Principal components, minor components, and linear neural networks, Neural Networks 5 (1992) 927–935.
[26] E. Oja, Beyond PCA: statistical expansions by nonlinear neural networks, Proceedings of the International Conference on Artificial Neural Networks (ICANN), vol. 2, Springer, Berlin, 1994, pp. 1049–1054.
[27] A.T. Puga, A.P. Alves, An experiment on comparing PCA and ICA in classical transform image coding, in Proceedings of the Independent Component Analysis (ICA'98), 1998, pp. 105–108.
[28] J. Rubner, P. Tavan, A self-organizing network for principal-component analysis, Europhys. Lett. 10 (7) (1989) 693–698.
[29] T.D. Sanger, Optimal unsupervised learning in a single-layer linear feed-forward neural network, Neural Networks 2 (1989) 459–473.
[30] L. Wang, J. Karhunen, E. Oja, A bigradient optimization approach for robust PCA, MCA and source separation, in Proceedings of the International Joint Conference on Neural Networks (IEEE-IJCNN), 1995, pp. 1684–1689.
[31] L. Xu, Least mean square error reconstruction principle for self-organizing neural-nets, Neural Networks 6 (1993) 627–648.
[32] S.M. Zurada, Introduction to Neural Artificial Systems, West Publishing Company, 1992.