Fast image compression using matrix K-L transform

Daoqiang Zhang, Songcan Chen *

Department of Computer Science and Engineering, Nanjing University of Aeronautics & Astronautics, Nanjing 210016, P.R. China.

Abstract: A novel matrix K-L transform (MatKLT) is proposed as an extension to the conventional K-L transform (KLT) in this paper for fast image compression. Experimental results on 18 publicly available benchmark images show that the MatKLT is tens to hundreds times faster than standard KLT with comparable image compression quality.

Keywords: Karhunen-Loeve transform (KLT); Matrix KLT (MatKLT); Image compression; Fast algorithm

1. Introduction

Transform coding is one of the most important methods for lossy image compression. However, as an optimal linear dimension-reduction transform in the mean-squared error or reconstructed error sense, the Karhunen-Loeve transform (KLT) [1] or principal component analysis (PCA) [2] can hardly be used in image compression due to its slow speed in seeking the transform from the covariance matrix constructed by given training data. Assume we are given a set of *n*-dimensional training data $\{x_1, x_2, ..., x_L\}$, then for any *n*-dimensional vector x， the Karhunen-Loeve transform is defined as:

$$y = \Phi^T (x - m_x) \tag{1}$$

where $m_x = (1/L)\sum_{i=1}^{L} x_i$ is the mean vector of training data, and the column vector of Φ is the eigenvector corresponding to the covariance matrix $C_x = (1/L)\sum_{i=1}^{L}\left[(x_i - m_x)(x_i - m_x)^T\right]$.

The larger the scale or dimension of the covariance matrix is, the slower the speed of computing the eigenvectors and hence transform is, and then so is performing compressing or encoding transform. To mitigate this problem, two methods are usually adopted. The first one is to replace KLT with Discrete Cosine Transform (DCT) as used in the JPEG standard [1]. Although able to achieve much faster compression than KLT, DCT leads to relatively great degradation of compression quality at the same compression ratio compared to KLT. The second is to use the parallelism technique [3][4] such

* Corresponding author: Tel: +86-25-489-2805; Fax: +86-25-489-3777.
E-mail: dqzhang@nuaa.edu.cn (D.Q. Zhang), s.chen@nuaa.edu.cn (S.C. Chen)

as neural networks including associative memories [5] and the adaptive principal component extraction (APEX) [2]. Despite of being a powerful parallelism method, neural networks [5] must also seek a tradeoff between the compression speed and quality. In addition, in order to obtain enough accurate approximation to classical KLT, more training steps and thus more time are needed in realization of neural networks for the KLTs.

Apart from the above-mentioned methods, to reduce the scale of the covariance matrix of the original KLT is obviously a cheaper alternative. To the best of our knowledge no other researchers have made attempts in this direction. In order to introduce our idea, let us recall construction of the covariance matrix in KLT for image compression: First, partition an image to be compressed into a set of non-overlapping subimage blocks with specified size and then for each block, row by row concatenate it into a vector, finally collect all these concatenated vectors to construct a so-needed covariance matrix with which the eigenvalues and corresponding eigenvectors, hence so-needed transform, can be found. Obviously, the scale of the block determines efficiency of computation. The smaller the block is, the faster the computation for the covariance matrix, and hence also the faster getting the transform. However, smaller block limits, in turn, the improvement of the compression ratio so that makes the KLT inapplicable in low-bit-rate encoding cases. In order to improve this situation, we change the traditional constructing method directly from the concatenated vectors. Encouraged by the successes on previous results [6], in this paper, a matrix K-L (linear) transform technique is proposed as an extension to the KLT for fast image compression. Its main idea is to use directly a matrix-type rather than vector-type representation to construct the covariance matrix, called as *generalized covariance matrix* (GCM), so that its scale is smaller than that of the covariance matrix constructed by those concatenated vectors. As a consequence, the speed of computing the transform from the GCM is greatly promoted. Taking an image block concatenated to a n-dimension vector as an example, the scale of the KLT covariance matrix is $n \times n$, while that of the GCM is just $m \times m$, when it is constructed by the $m \times p$-dimensional matrix which is a rearrangement for that n-dimensional vector, where the m and p meet $m \times p = n$. Obviously the reduction ratio of both scales reaches p^2, which is an impressive result for large p. Such a reduction naturally greatly speeds up finding the transform. Experimental results on 18 publicly available benchmark images show that MatKLT is tens to hundreds times faster than standard KLT with comparable image compression quality.

The rest of this paper is organized as following: We first formulate the matrix K-L transform

(MatKLT) in Section 2. And in Section 3, image compression experiments based on MatKLT is introduced and results are given. At last, we draw our conclusions in Section 4.

2. Matrix K-L transform

A. Formulation of MatKLT

There are two main steps in our proposed MatKLT. The first is the 'reassemble or rearrange' step which rearranges the components of a concatenated subimage-block vector into a new matrix with dimensionality $m \times p$, which will directly use the subimage block matrix as a special case and thus be more flexible. For example, a vector $x=[1,2,3,3,5,2,7,0,2,1]^T$ can be rearranged into $A=[1,2,3,3,5,2,7,0,2,1]^T$ and $\begin{bmatrix} 1 & 3 & 5 & 7 & 2 \\ 2 & 3 & 2 & 0 & 1 \end{bmatrix}$, corresponding to p=1 and 5 respectively. And the second step is performing MatKLT on these matrices obtained in the first step. Below is a derivation of our MatKLT.

Suppose that we are given a matrix-type random variable $A = [a_{ij}]$ with dimension $m \times p$, and intend to find a linear transform Φ_d to compress A to B described by

$$B = \Phi_d^T (A - \overline{A}) \tag{2}$$

where $\overline{A} = E(A)$ is the mean of A, $E(.)$ is an mathematical *expectation*, and $\Phi_d = (\phi_1, \phi_2, \cdots, \phi_d) \in R^{m \times d}$ is a transform to be found. Equation (2) implements a dimensionality-reduction from A to B. Reversing the process for solving B, we can reconstruct A by $\hat{A} = \Phi_d B + \overline{A}$ by means of Eq. (2). Now the key to implementing compression is to find the transform Φ_d. We still adopt the reconstructed error criterion (REC) as used in the KLT and minimize it to seek such an optimal transform Φ_d. Define the RCE as follows:

$$\begin{aligned} REC(\Phi_d) &= E\left\{ \left\| A - \hat{A} \right\|^2 \right\} = E\left\{ \left\| A - \Phi_d B - \overline{A} \right\|^2 \right\} = E\left\{ \left\| A - \overline{A} - \Phi_d \Phi_d^T (A - \overline{A}) \right\|^2 \right\} \\ &= E\left\{ tr((A - \overline{A} - \Phi_d \Phi_d^T (A - \overline{A}))(A - \overline{A} - \Phi_d \Phi_d^T (A - \overline{A}))^T \right\} \\ &= E\left(\left\| A - \overline{A} \right\|^2 \right) - tr(\Phi_d^T E((A - \overline{A})(A - \overline{A})^T) \Phi_d) \end{aligned} \tag{3}$$

where $tr(.)$ denotes the trace of a matrix, and I the identity with dimension $d \times d$, and $\|A\| = \left(\sum_{i=1}^{m} \sum_{j=1}^{p} |a_{ij}|^2 \right)^{1/2}$ is the *Frobenius* norm. And the latter two lines of Eq. (3) are from the following two characters of trace and Frobenius norm: 1) $\|A\| = tr(AA^T)$; 2) $tr(AB) = tr(BA)$. Set

$R = E((A-\overline{A})(A-\overline{A})^T)$, i.e., a GCM for matrix-type random variable A. From Eq.(3), the minimization of REC with respect to Φ_d is equivalent to maximization of $J(\Phi_d)$ defined by

$$J(\Phi_d) = tr(\Phi_d^T R \Phi_d) \tag{4}$$

subject to the constraints that $\Phi_d^T \Phi_d = I$. Equations (3) and (4) give an identical optimization formulation to the KLT except that the KLT covariance matrix constructed by vector-type random variables is replaced with a new GCM by matrix-type ones. In fact, the following derivation of the transform is identical to that of the KLT. However for completeness of description, we still give the following reformulation.

To maximize $J(\Phi_d)$ subject to the constraints, we define a Lagrangian function as follows:

$$L(\Phi_d) = J(\Phi_d) - \sum_{j=1}^{d} \lambda_j (\phi_j^T \phi_j - 1) \tag{5}$$

where λ_js are Lagrange multipliers. Differentiating Eq.(5) with respect to ϕ_j s and setting the corresponding derivatives to 0, we have

$$R\phi_j = \lambda_j \phi_j \quad j=1, 2, \ldots, d \tag{6}$$

This is an eigenvalue-eigenvector system with respect to (λ_j, φ_j). Taking into account that R is symmetric and positive semi-definite, therefore its eigenvalues are non-negative, which leads Eq.(6) to a maximum value of $J(\Phi_d) = \sum_{j=1}^{d} \lambda_j$, equivalently, a minimum of $REC(\Phi_d) = \sum_{j=d+1}^{m} \lambda_j$. Here we take the d eigenvectors of R corresponding to the first d largest eigenvalues to construct the so-needed transform $\Phi_d = (\phi_1, \phi_2, \cdots, \phi_d)^T$, which achieves the REC minimization and at the same time retain maximally the original information of data. As a result, the original data is effectively compressed.

B. Un-correlation Analysis

As we have known, the KLT can remove correlation between the original data components to achieve a goal of compression. In fact, our MatKLT also possesses such a property as analyzed below. Rewriting B in row to $[(b_1)^T, (b_2)^T, \ldots, (b_d)^T]^T$, where $b_j = \phi_j^T (A - \overline{A})$ $j=1,2,\ldots,d$. Then for any b_i and b_j,

$$E(b_i b_j^T) = E[\phi_i^T (A - \overline{A})(A - \overline{A})^T \phi_j] = \phi_i^T R \phi_j = \lambda_j \phi_i^T \phi_j = \begin{cases} \lambda_j & i = j \\ 0 & i \neq j \end{cases} \quad (7)$$

In the above derivation, we use Eq.(4). Equation (7) tells us that any two different row vectors, b_i and b_j, of B are indeed uncorrelated, which is helpful to the subsequent compression. In particular, when the matrix A is concatenated into a vector, our MatKLT reduces to the ordinary KLT.

C. Real Reconstruction

In a real implementation, generally, we are only given a limited sample set of matrix data $\{A_i, i=1,2,\ldots,N\}$. Then the covariance matrix or GCM R and the mean $\overline{A}$ will respectively be estimated by the given sample dataset as follows:

$$R = \frac{1}{N}\sum_{i=1}^{N}(A_i - \overline{A})(A_i - \overline{A})^T \quad \text{and} \quad \overline{A} = \frac{1}{N}\sum_{i=1}^{N} A_i \quad (8)$$

Substituting them into Eq.(6) and then solving it, we can obtain a transform Φ_d for the given dataset. To implement decompression or reconstruction, we use the following equation

$$\hat{A} = \Phi_d B + \overline{A} \quad (9)$$

It implements a restoration for the original image with minimal errors.

Now let us discuss the effect of p on the compression speed. According to the definition of R, its scale becomes smaller due to the relation of $m \times p = n$ as p increases. Therefore, to obtain a high speedup for MatKLT, we require p to take a value as large as possible. On the other hand, as can be seen from Eq. (2), in order to achieve a high compression ratio, the value of p cannot be too large. So p controls the tradeoff between the speed and compression ratio or compression quality.

In the next section, we will give comparison experimental results on 18 publicly available benchmark images to illustrate characteristic of our MatKLT.

3. Image compression experiments

A. Configuration

In our experiments, we compared proposed MatKLT with classical KLT and DCT algorithms on 18 benchmark images. Specifically, we compare the compressed image quality evaluated by peak signal-to-noise ratio (PSNR), and the corresponding run-time of the algorithms. All the algorithms are executed on the same computer equipped with 1.7 G Intel Pentium 4 processor, 256 M memory, and 40G hard disk, and simulated using Matlab 6.5 by Mathworks Inc. It is important to note that no

optimizations regarding execution time are considered in all algorithms, and we are only concerned on the relative execution speed of KLT and MatKLT. On the other hand, we only compare the compressed image quality performances between DCT and the other two algorithms. We do not list the execution time of DCT because the program structure of DCT is much different from those of KLT and MatKLT and hence the comparison between them is of little sense.

The size of the images used in the experiments are 512x512, except for the images 'barbara' (720x580), 'boats' (720x576), 'camera' (256x256), 'columbia' (480x480) and 'goldhill' (720x576). For all images, we use block size of 16x16, except for the image 'barbara' (20x20). At last, we use d =16 in Eqs. (4) and (5) for KLT (p=1) for all images except for the image 'barbara' (d=20). To maintain the same compression ratio, d =8 (p=2) and d =4 (p=4) are used in Eqs (4) and (5) for MatKLT (for 'babara', d =10 (p=2) or d =5 (p=4)). Similarly, 16 components (20 for 'barbara') are kept in the transformed image matrix. Furthermore, no quantization steps are used in all algorithms and 8-bits are used to represent components of the compressed matix B in Eq. (2). Thus for all images the compression ratio is 16:1, except for the image 'barbara' (20:1).

B. Results

Table 1 shows the comparisons of compressed image quality (in PSNR) and execution times between KLT, MatKLT. Here we compute PSNR as follows:

$$PSNR(db) = 10\log_{10}\frac{255^2}{(1/mp)\sum_{i=1}^{m}\sum_{j=1}^{p}\left(a_{ij}-\hat{a}_{ij}\right)^2} \tag{10}$$

where a_{ij} and $\hat{a}_{ij}$ represent the pixel values of original image $A=[a_{ij}]_{m\times p}$ and reconstructed image $\hat{A}=[\hat{a}_{ij}]_{m\times p}$ respectively. From Table 1, it can be observed that KLT achieves the best reconstructed image quality, which proves its optimum in transform image coding. The reconstructed image using MatKLT (p=2) is a little worse than KLT, in which case only about 0.5 db degradation is incurred compared with KLT. Even if p=4, the degradation in MatKLT is still just about 1-2 db. As a comparison, the compressed image quality is very poor when using DCT. As shown in table 1, generally, there is about 5 db degradation in DCT compared to KLT, and for some images such as 'woman2', the degradation is even 11 db. Fig. 1 gives the reconstructed images of KLT, MatKLT and DCT on the image 'Lena'. From Fig. 1, there are no apparent visual differences between the reconstructed images using KLT and MatKLT, while the DCT reconstructed image is much degraded.

Table 1 Comparisons of the algorithm performances: PSNR and speed (in the bracket, unit: second)

Test image	KLT (p=1)	MatKLT (p=2)	MatKLT (p=4)	DCT
Baboon	22.89(3.13)	22.71(0.11)	22.43(0.02)	22.43
Barbara	24.51(3.28)	23.84(0.44)	23.32(0.03)	20.77
Boats	28.63(2.81)	27.95(0.11)	26.91(0.02)	23.26
Bridge	23.73(3.09)	23.40(0.11)	22.91(0.03)	21.13
Camera	24.37(2.47)	23.39(0.09)	22.83(0.03)	20.84
Columbia	29.38(2.49)	28.53(0.06)	26.73(0.02)	22.16
Couple	26.52(2.88)	26.07(0.08)	25.38(0.02)	23.16
Crowd	27.23(2.67)	26.64(0.08)	25.57(0.02)	21.45
Goldhill	29.33(3.02)	28.96(0.09)	28.20(0.03)	24.19
Lake	26.06(2.86)	25.47(0.08)	24.50(0.02)	21.18
Lax	23.26(3.25)	23.09(0.09)	22.87(0.02)	21.98
Lena	30.81(2.89)	30.39(0.09)	29.61(0.02)	25.17
Man	27.43(3.05)	27.12(0.08)	26.48(0.02)	23.43
Milkdrop	23.26(3.25)	23.09(0.09)	22.87(0.02)	21.98
Peppers	29.73(3.08)	28.86(0.09)	27.70(0.02)	22.66
Plane	27.48(2.61)	26.95(0.08)	25.67(0.02)	22.49
Woman1	27.68(3.05)	27.29(0.09)	26.63(0.02)	23.65

On the other hand, the execution speed of MatKLT (p=2, 4) is much improved compared to KLT (p=1). From the same table, the improvement of speed of MatKLT (p=2) over KLT (p=1) is about 30:1, and that of MatKLT (p=4) over KLT (p=1) is about 300:1 for most images. To achieve so great improvement in speed, the paid price is only 1-2 db degradation of the reconstructed image compared with that of KLT. Fig. 2 shows a comparison of performances of MatKLT under different values of p for 'Lena' image with block size 32x32. Seen from the figure, as p grows, the quality of reconstructed image degrades gradually, while the execution time is greatly reduced. However, in order to keep a balance among the speed, compression ratio and image quality, we confine ourselves to only p=2 and 4 in this paper.

4. Conclusions

We presented a novel matrix K-L transform and applied it into image compression. The experimental results show that the MatKLT method requires much less computation time than KLT at the price of a little degradation of compressed image quality. This method has the potentiality to be a faster method for image data reduction, especially for real-time and progressive decoding applications. The next research is to compare the execution time of MatKLT, KLT and DCT algorithms

implemented with optimizations, e.g adopting the C programming language instead of the Matlab used in this paper. It is worth to mention for the K-L transform or PCA implementation that although we employed the so-called batch methods [2] for the transform computation in our experiments; however, in practice, we can use the Hebbian-based neural networks to more effectively and adaptively implement the MatKLT algorithm, which will become one of our subjects in the next research.

Acknowledgements

We would like to thank the reviewers for their valuable suggestions for improving the presentation of this paper. Meanwhile we also thank National Science Foundations of China and of Jiangsu under Grant Nos. 60473035 and BK2002092, Jiangsu Natural Science Key Project (BK2004001), Jiangsu "QingLan" Project Foundation and the Returnee's Foundation of China Scholarship Council for partial supports respectively.

References

[1] R.C. Gonzalez, R.E. Woods, Digital Image Processing, 2nd Edition, Prentice-Hall, Jan. 2002.

[2] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice-Hall, Jul. 1998.

[3] S. Bhama, H. Singh, N.D. Phadte, "Parallelism for the faster implementation of the K-L transform for image compression", Pattern Recognition Letters, vol. 14, pp. 651-659, Aug. 1993.

[4] S. Costa, S. Fiori, "Image compression using principal component neural networks", Image and Vision Computing, vol. 19, pp. 649-668, Aug. 2001.

[5] C.C. Wang, C.R. Tsai, "Data compression by the recursive algorithm of exponential bidirectional associative memory", IEEE Trans. Syst. Man. Cybern., vol.28, no. 4,pp. 125-134, Apr. 1998.

[6] Jian Yang, Jingyu Yang, "From image vector to matrix: a straightforward image projection technique-IMPCA vs. PCA", Pattern Recognition, vol. 35, no. 9, pp. 1997-1999, Sep. 2002.

(a) (b) (c)

(d) (e)

Fig. 1 Compressed images at 0.5 bpp without quantization. (a) original 'Lena' image, (b) KLT result (PSNR=30.8), (c) MatKLT (p=2, PSNR=30.4), (d) MatKLT (p=4, PSNR=29.6), (e) DCT result (PSNR=25.2).

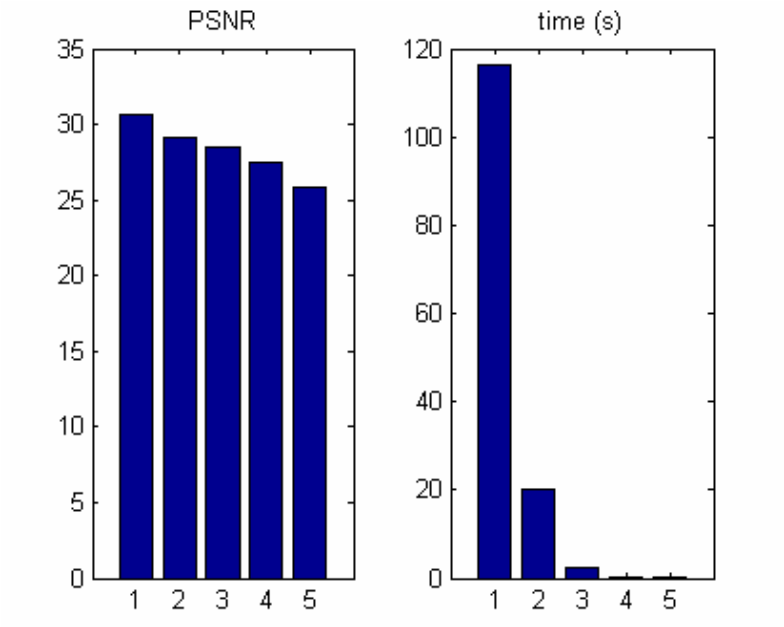

Fig. 2 Comparisons of performance of five methods: 1. KLT (i.e. MatKLT with p=1), 2. MatKLT (p=2), 3. MatKLT (p=4), 4. MatKLT (p=8), 5. MatKLT (p=16).