Improving the Speed of Kernel PCA on Large Scale Datasets

Tat-Jun Chin and David Suter
Institute for Vision Systems Engineering,
Monash University, Victoria, Australia.
{ tat.chin | d.suter }@eng.monash.edu.au

Abstract

This paper concerns making large scale Kernel Principal Component Analysis (KPCA) feasible on regular hardware. The KPCA has been proven a useful non-linear feature extractor in several computer vision applications. The standard computation method for KPCA, however, scales badly with the problem size, thus limiting the potential of the technique for large scale data. We propose a novel method to alleviate this problem. The essence of our solution lies in partitioning the data and greedily filtering each partition for a sparse representation. Incremental KPCA is then utilized to merge each partition to arrive at the overall KPCA. We also provide experimental results which demonstrate the effectiveness of the approach.

1. Introduction

KPCA has been used as an effective non-linear feature extraction method in the field of computer vision. Some of the applications that benefitted from the non-linearity induced by the kernelization of standard PCA include face recognition, image denoising, acquisition of multiple view descriptors and human motion modeling. There is also the possibility of a variety of other applications within surveillance that can be tackled by non-linear feature extraction. Therefore, it is worthwhile investigating KPCA for processing large scale datasets, since video surveillance systems usually produce massive amounts of data.

However, the standard KPCA computation method via eigendecompositions involves a time complexity of $O(n^3)$, with n being the number of training vectors, thus implementing KPCA on large datasets is prohibitively expensive. In this paper, we propose a novel solution to alleviate this problem. We take the "divide-and-conquer" approach which differs from several methods proposed previously to tackle the problem. The basis of our solution lies in partitioning the large dataset that is to be processed into many smaller and more manageable chunks. Through a process called Greedy KPCA [5], we "filter" the individual data chunks for representative vectors. The filtered data chunks are then merged through an incremental KPCA process to arrive at the overall KPCA. We show through experiments that our solution is faster than previous approaches and is capable of maintaining close approximation to the ground truth given by the standard KPCA computation method.

We also demonstrate two practical computer vision applications using the proposed method, namely image denoising and super-resolution of face images. The second application is potentially useful in visual surveillance, since contemporary CCTV systems capture in low-resolution and faces recorded might not be informative enough for recognition. This can be rectified through super-resolution. As mentioned previously, the applications of KPCA to video surveillance problems is not limited to just this.

2. Related Work

Several methods have been proposed previously to solve the computational complexity problem of KPCA. A class of these methods is the sampling based approaches [9, 10, 1, 5]. For example, Williams et al. [10] proposed the Nyström method. This involves a prior random sampling of the training data to create a reduced-rank kernel matrix which is then used to recover the eigenspectrum of the original kernel matrix by using the Nyström theory. It is pointed out in [10] that the major difference between their work and the sparse greedy matrix approximation method [9] is how the sampling is performed. The form of the eventual kernel matrix approximation is actually identical. The Greedy KPCA (GKPCA) which was proposed in [5] involves greedily filtering the training data for a small but representative subset of which the linear span closely resembles the linear span of the original data. Standard PCA is then performed on the subspace (of the kernel induced feature space) defined by the small subset. One stage of our proposed method includes using GKPCA, so the next section is devoted to de-

Proceedings of the IEEE International Conference
on Video and Signal Based Surveillance (AVSS'06)
0-7695-2688-8/06 $20.00 © 2006 IEEE

scribing it in more detail.

A non-sampling based method is the Kronecker factorization approach [11] proposed by Wu et al. The Kronecker factorization approximates a positive definite matrix by the Kronecker product of two smaller positive definite matrices. Eigendecompositions of the smaller matrices are computationally cheaper, and the eigenspectrum of the original matrix can be reconstructed from the smaller eigenspectrums. However, there is a prior cost of performing the Kronecker factorization, and two methods were proposed in [11] to implement it. It is shown in [11] that their method is computationally faster than the Nyström method.

We summarize the contributions of this paper here: We argue that partitioning the training data for multiple GKPCA filtering is faster than performing GKPCA on the whole dataset. We also propose a method to compute the overall KPCA by incrementally merging the individual GKPCA results. We demonstrate that our method presents a substantial improvement in processing speed, while maintaining close approximation to standard KPCA results. In addition, the KPCA expansion from our method is automatically *sparser* due to the prior GKPCA filtering. This differs from some of the methods outlined here which attempt to approximate the full eigenspectrum, hence requiring the KPCA result to be expanded from all training data.

3. Greedy Kernel PCA

We have training data $a = [x_1 \cdots x_n] \in \mathbb{R}^{m \times n}$ residing in input space $\mathbb{R}^m$ which is implicitly mapped to a feature space $\mathcal{F}$ to become $A = \phi(a)$, where the non-linear map $\phi(\cdot)$ is induced by a kernel function $k(\cdot,\cdot)$ [4]. GKPCA aims to greedily "filter" a to produce a subset $\dot{a}$ such that

$$span(\dot{A}) \approx span(A) \ , \tag{1}$$

where $\dot{A} = \phi(\dot{a})$. Let $J = \{j_1, \cdots, j_s\}$ be the set of indices of vectors in $\dot{a}$. Then $J \subset \{1, \cdots, n\}$ with $s < n$ and $\dot{a} = \{x_j : j \in J\}$. The intuition behind GKPCA is that if $span(\dot{A})$ is highly similar to $span(A)$ and at the same time $s \ll n$, processing $\dot{A}$ for KPCA is significantly faster than processing A with the results being effectively similar.

To see this, we first orthogonalize $\dot{A}$ to obtain an orthonormal basis $\dot{A}\beta$. An easy way to perform this is via Kernel SVD [4]. The data A is projected onto $\dot{A}\beta$ to obtain

$$z = \beta^T \dot{A}^T A \ , \tag{2}$$

where $z \in \mathbb{R}^{s \times n}$. The kernel function $k(\cdot,\cdot)$ is utilized to compute $\dot{A}^T A$. If we define $\tilde{A}$ to be the approximation of A using the reduced subset $\dot{A}$, then

$$A \approx \tilde{A} = \dot{A}\beta z \ . \tag{3}$$

PCA is then performed on the columns of z. However, instead of factorizing the Gram matrix $G = z^T z$ of size $n \times n$, we factorize the scaled covariance matrix $C = zz^T$ of size $s \times s$ which is computationally cheaper. Note that z has to be centered beforehand by subtracting the mean $\mu = \sum_{i=1}^{n} z_i$, where z_i is the i-th column of z. Let the SVD of C be $C = U\Sigma V^T$. The *approximate* kernel principal components of A are then

$$\dot{A}\alpha, \quad \text{where } \alpha := \beta U. \tag{4}$$

Similarly, the *approximate* mean of A is $\dot{A}\nu$, where $\nu := \beta\mu$. Note that compared to standard KPCA which involves factorizing the overall kernel matrix, the kernel principal components are expanded from a sparser dataset $\dot{a}$. This is the objective for which the GKPCA was proposed [5].

A measure of how well $\dot{A}\beta z$ approximates A given a reduced set $\dot{a}$ is the mean square error defined by

$$\epsilon_{MS}(A|J) = \frac{1}{n} \sum_{i=1}^{n} \left\| \phi(x_i) - \dot{A}\beta z_i \right\|^2 \ , \tag{5}$$

Note that J is sufficient to specify $\dot{a}$ and $\dot{A}\beta$. For a fixed cardinality s of J, finding the optimal subset $\dot{a}^*$ of a can then be casted as the optimization problem of finding the corresponding J^*:

$$J^* = \underset{J \subset \{1,\cdots,n\}}{\arg\min} \ \epsilon_{MS}(A|J) \ . \tag{6}$$

Several practical algorithms were introduced to implement the optimization, with one having the lowest time complexity of $O(ns^2)$. The reader is referred to [5] for details.

4. Large Scale Kernel PCA

We are dealing with large datasets typified by $a = [x_1 \cdots x_n] \in \mathbb{R}^{m \times n}$, with $n > 5000$. Performing KPCA by directly factorizing the kernel matrix of a using *standard* hardware would be prohibitive, since the computational complexity involved is $O(n^3)$. Greedily filtering a for a lesser subset *may* allow us to deal with a smaller factorization problem, but searching for an optimal subset $\dot{a}$ can be time-consuming, especially when n and s are very large, since the complexity of GKPCA is $O(ns^2)$. Furthermore, there is no guarantee that, to a satisfactory approximation accuracy, a small enough subset $\dot{a}$ always exists such that computational effort for matrix factorization is greatly reduced. We propose a method to alleviate this problem.

We first establish the meaning of commonly used symbols. Given a matrix M, the symbol $M_{a:b,c:d}$ defines the submatrix that contains the elements of M within the intersection of the a-th row till the b-row and the c-th column till the d-th column. If the row or column specifiers are omitted, e.g. $M_{:,c:d}$, take all available rows or columns.

Proceedings of the IEEE International Conference
on Video and Signal Based Surveillance (AVSS'06)
0-7695-2688-8/06 $20.00 © 2006 IEEE

4.1. Partitioning the Training Data

The proposed solution begins by partitioning a into c smaller blocks a_k, $k = \{1, \cdots, c\}$, with the definition $a = [a_1 \cdots a_c]$. Let d_k denote the size of each block, hence $n = \sum_{k=1}^{c} d_k$. GKPCA is then evaluated on each block, yielding a total of c subsets $\dot{a}_k$ for c blocks a_k. The number of vectors s_k in each subset can be set to a constant value, or be allowed to vary in order to achieve a pre-determined approximation accuracy. Hence, each data block $A_k = \phi(a_k)$ in $\mathcal{F}$ can be approximated by the corresponding reduced subset as $\tilde{A}_k = \dot{A}_k \beta_k z_k$, with $\dot{A}_k = \phi(\dot{a}_k)$. Matrix β_k accounts for orthogonalization of $\dot{A}_k$ and z_k contains the projection components of A_k onto $\dot{A}\beta_k$. Let μ_k be the mean of z_k, then $\nu_k = \beta_k \mu_k$ spans the mean of $\tilde{A}_k$ using $\dot{A}_k$.

The next step involves computing the KPCA of a by incrementally performing PCA using each approximation data block $\tilde{A}_k$ as incremental data. We define the desired overall KPCA model to be the structure

$$\Omega^* = \{A^*, d^*, \nu^*, \alpha^*, \Sigma^*, V^*\} \ . \tag{7}$$

$A^* \in \mathcal{F}$ is the library of s^* expansion vectors, while d^* indicates the *effective* number of data on which the KPCA was performed. Coefficients ν^* and α^* respectively span the data mean $A^*\nu^*$ and kernel principal components $A^*\alpha^*$. Matrices Σ^* and V^* are respectively the singular values and right singular vectors obtained from the factorization process for KPCA using SVD.

The structure Ω^* is initialized by performing PCA on the first block $\tilde{A}_1$ using z_1. Let z_1 be factorized as $z_1 = U_1 \Sigma_1 (V_1)^T$ (z_1 is centered with μ_1 prior to this). Then $\alpha_1 = \beta_1 U_1$ spans the approximate kernel principal components of a_1 (refer to §3 for details). The parameters of Ω^* are initialized using the PCA results of $\tilde{A}_1$: $A^* \leftarrow \dot{A}_1$, $d^* \leftarrow d_1$, $\nu^* \leftarrow \nu_1$, $\alpha^* \leftarrow \alpha_1$, $\Sigma^* \leftarrow \Sigma_1$, and $V^* \leftarrow V_1$.

4.2. Incremental Kernel PCA

We wish to update Ω^* using $\tilde{A}_2$. Computing the new number of data $d^*_{(new)}$ can be easily done as $d^*_{(new)} = d^* + d_2$. Computing $\nu^*_{(new)}$ for the new mean is also trivial, with

$$\nu^*_{(new)} = \frac{1}{d^* + d_2} \begin{bmatrix} d^*\nu^* \\ d_2\nu_2 \end{bmatrix} \ , \tag{8}$$

after which we can define $A^*_{(new)}$ as the concatenation of A^* and $\dot{A}_2$, i.e. $A^*_{(new)} = [\ A^* \ \ \dot{A}_2\ ]$. For reasons that will be explained later, we define the intermediate matrix

$$\tilde{E} := \begin{bmatrix} \dot{A}_2\beta_2(z_2 - \mu_2) & \sqrt{\frac{d^* \cdot d_2}{d^* + d_2}}(A^*\nu^* - \dot{A}_2\nu_2) \end{bmatrix} \ . \tag{9}$$

The left submatrix on the right-hand-side is merely the mean-adjusted version of $\tilde{A}_2$. Matrix $\tilde{E}$ can be written in the form $A^*_{(new)}\gamma$, with

$$\gamma := \begin{bmatrix} \begin{bmatrix} \mathbf{0}_{s^*, d_2} \\ \beta_1(z_2 - \mu_2) \end{bmatrix} & \sqrt{\frac{d^* \cdot d_2}{d^* + d_2}} \begin{bmatrix} \nu^* \\ -\nu_2 \end{bmatrix} \end{bmatrix} \ . \tag{10}$$

The symbol $\mathbf{0}_{r,c}$ indicates an $r \times c$ matrix of zeroes. Our goal is to find the PCA of $[\ \tilde{A}^* \ \ \tilde{A}_2\ ]$, where $\tilde{A}^*$ is the reconstruction of feature space vectors using the model Ω^*:

$$\tilde{A}^* = A^*(\alpha^*\Sigma^*(V^*)^T + \nu^*) \ . \tag{11}$$

To this end, it was proven in [8] that the *scatter* matrix corresponding to $[\ \tilde{A}^* \ \ \tilde{A}_2\ ]$ is equal to

$$[\ A^*\alpha^*\Sigma^*(V^*)^T \ \ \tilde{E}\ ][\ A^*\alpha^*\Sigma^*(V^*)^T \ \ \tilde{E}\ ]^T \ . \tag{12}$$

This implies that to perform PCA on $[\ \tilde{A}^* \ \ \tilde{A}_2\ ]$, it is sufficient to perform an SVD on $[\ A^*\alpha^*\Sigma^*(V^*)^T \ \tilde{E}\ ]$. Hence, we can apply incremental Kernel SVD procedures [3] using $\tilde{E}$ as increment data to update Ω^*.

The derivations in [3] demonstrate how Kernel SVD can be computed incrementally without explicitly evaluating the mapping ϕ (i.e. the kernel trick). It begins by identifying the factorization of $[\ A^*\alpha^*\Sigma^*(V^*)^T \ \tilde{E}\ ]$ as:

$$\begin{bmatrix} A^*\alpha^* & J \end{bmatrix} \begin{bmatrix} \Sigma^* & L \\ \mathbf{0}_{d_2+1,\, s^*} & K \end{bmatrix} \begin{bmatrix} V^* & \mathbf{0}_{d^*,\, d_2+1} \\ \mathbf{0}_{d_2+1,\, s^*} & \mathbf{I}_{d_2+1} \end{bmatrix}^T \ . \tag{13}$$

$\mathbf{I}_r$ represents an $r \times r$ identity matrix. Matrix L is the projection of $\tilde{E}$ onto the kernel principal components of Ω^*:

$$L = (\alpha^*)^T (A^*)^T A^*_{(new)}\gamma \ . \tag{14}$$

The matrix $(A^*)^T A^*_{(new)}$ is realizable using the kernel function since it contains dot products only. To define J and K, first define H as the orthogonal component from $span(A^*\alpha^*)$ to $\tilde{E}$, i.e. $H = \tilde{E} - A^*\alpha^*L = A^*_{(new)}\eta$, with

$$\eta := \begin{bmatrix} \gamma_{1:s^*,:} - \alpha^*L \\ \gamma_{(s^*+1):(s^*+s_2),:} \end{bmatrix} \ . \tag{15}$$

H is related to J and K as $H = JK$, with J being an orthonormal basis for the subspace $span(H)$, while K is simply the projection of H onto J. To obtain J, we can apply Kernel SVD on H with the kernel matrix to be factorized as

$$M_H = \eta^T (A^*_{(new)})^T A^*_{(new)}\eta \ . \tag{16}$$

This is computable using the kernel function. Hence, J has the form $J = A^*_{(new)}\kappa$, with κ containing the coefficients for spanning the orthonormal basis J. K can be computed using the kernel function as $K = \kappa^T (A^*_{(new)})^T A^*_{(new)}\eta$.

Proceedings of the IEEE International Conference
on Video and Signal Based Surveillance (AVSS'06)
0-7695-2688-8/06 $20.00 © 2006 IEEE

The middle matrix of Eq. (13) is constructed and factorized using the SVD as $U'\Sigma'(V')^T$ which is then substituted back into Eq. (13). The matrix $[\,A^*\alpha^*\Sigma^*(V^*)^T\ \tilde{E}\,]$ on which SVD is sought then assumes the form

$$[\,A^*\alpha^*\Sigma^*(V^*)^T\ \ \tilde{E}\,] = U''\Sigma''(V'')^T\ . \qquad (17)$$

This is updated PCA of $[\,\tilde{A}^*\ \tilde{A}_2\,]$. Singular values Σ'' is simply equal to Σ', while right singular vectors V'' is the result of the multiplication between the right-most matrix of Eq. (13) with V'. The left singular vectors U'' is

$$U'' = \left[\ A^*\alpha^* \quad A^*_{(new)}\kappa\ \right] U' = A^*_{(new)}\alpha^*_{(new)}\ , \quad (18)$$

with $\alpha^*_{(new)} := \left[\ \begin{matrix}\alpha^*\\ \mathbf{0}_{s_2,s^*}\end{matrix}\ \right] U'_{1:s^*,:} + \kappa U'_{(s^*+1):(s^*+s_2),:}\ .$

The other parameters of the updated KPCA model are simply $\Sigma^*_{(new)} = \Sigma''$ and $V^*_{(new)} = V''$. Finally, set $\Omega^* \leftarrow \Omega^*_{(new)}$, where $\Omega^*_{(new)}$ is the structure that contains all the parameters with the subscript (new). Following the processes described above, the KPCA model Ω^* is then subsequently updated using $\tilde{A}_3$ and so on.

Note that $\tilde{A}_1$ could have been *deflated* by retaining $r < s_1$ kernel principal components before being used to initialize Ω^*. In this case, the size of matrices α^*, Σ^* and V^* would differ, and the size of other affected matrices in the derivations above should be changed accordingly. In general, deflation can be performed on Ω^* after each update. This is useful if the variance of the processed data is concentrated on a few kernel principal components only.

5. Comparison of Computational Complexity

Following our method, the data a is partitioned into c blocks a_k with $k = \{1, \ldots, c\}$ on which separate GKPCA's are performed. Each block is of size d_k. Assume that to *satisfactory* approximation accuracies, $s_k < d_k$ vectors are greedily filtered from each block a_k. The individual computational effort required would scale as $O(d_k s_k^2)$. When many partitions were created, i.e. if $d_k, s_k \ll n, s$, the cumulative cost of the individual GKPCA's would be significantly less than performing an overall GKPCA on a to obtain $s = \sum_{k=1}^{c} s_k$ representative vectors.

For the incremental KPCA process, a series of small SVD's are to be carried out. The most computationally expensive step is the SVD of the middle matrix of Eq. (13) and the orthogonalization of matrix H (using Kernel SVD). During incremental KPCA, if no deflations are to be made, the largest SVD problem among these small SVD's is of size $s \times s$. This is as costly as performing a direct KPCA on a by using the aggregated reduced set $[\,\dot{a}_1, \ldots, \dot{a}_c\,]$ which contains s vectors— there is no free lunch. In fact, the resultant kernel principal components are effectively the same

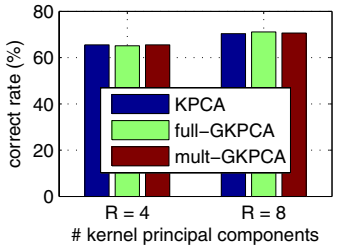

Figure 1. Recognition rates of 3 KPCA methods for ABALONE data.

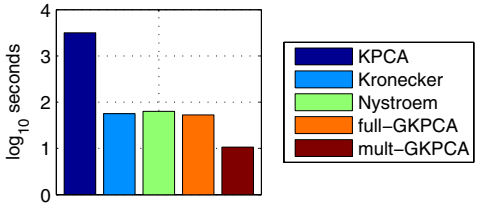

Figure 2. Training time of five KPCA methods on the ABALONE training set (3000 vectors). Times for the Kronecker and Nyström methods are approximate (taken from [11]).

(*nonetheless*, partitioning the data and performing multiple GKPCA's allowed us to obtain the reduced set in significantly shorter time than applying GKPCA on a directly).

If we are willing to deflate the KPCA model during updates by retaining only a subset of available kernel principal components, the cumulative computational effort for the small SVD problems can be substantially smaller than factorizing an $s \times s$ matrix. It could be advantageous to deflate if the data distribution is concentrated on a few kernel principal components only, and the resultant KPCA model is effectively similar to the ground truth.

6. Experimental Results

We first examine the performance of the proposed method using the ABALONE dataset from the UCI Repository [2]. This dataset contains 4177 training instances, each of which has 8 attributes of abalone measurement and one label indicating age (1–29 years). Following [11], we assign vectors with ages 1–10 into Class 1 and ages 11-29 into Class 2. A binary classifier is built based on this assignment. We designate the first-3000 vectors as the training set and the remaining as the testing set. KPCA is applied as a non-linear feature extractor on the training set using the Gaussian kernel of width 2. Here, 3 methods are

Proceedings of the IEEE International Conference
on Video and Signal Based Surveillance (AVSS'06)
0-7695-2688-8/06 $20.00 © 2006 IEEE

used— standard KPCA, full-GKPCA (GKPCA on the full training set) and the proposed method which we abbreviate to mult-GKPCA. For mult-GKPCA, we randomly partition the training set into 10 blocks. GKPCA is performed individually on these partitions to achieve mean-squared-error (mse) of 0.001 in approximating the original data block. During incremental KPCA, the intermediate KPCA models are deflated such that only 99.99% of the data variance is retained. For full-GKPCA, filtering is conducted until the number of filtered vectors is the same as the total number of filtered vectors from mult-GKPCA. For all 3 methods, both training and testing sets are then projected onto the kernel principal components obtained. Using the projection components, the nearest-neighbour rule is followed to assign the label of the testing set. Figures 1 and 2 respectively illustrate the classification rate and overall training duration of all 3 methods (and 2 other competing algorithms [9, 11]). All 3 methods return very similar classification rates (refer to [11] for the performance of the Kronecker and Nyström[1] methods). This proves that, for the ABALONE dataset, the proposed KPCA method can closely approximate the ground truth KPCA. Secondly, our method presents a substantial improvement in training speed over other methods. Also, our method provides a much sparser KPCA expansion (only 266 vectors out of 3000 were filtered out).

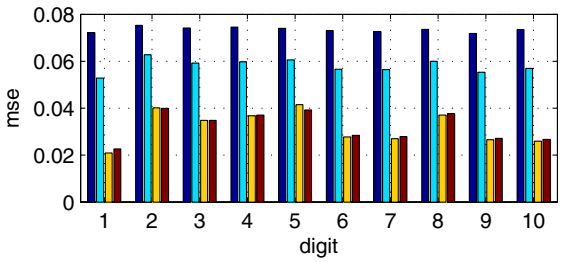

Figure 4. Comparison of average mse of different methods. For each digit, column 1 is PCA (R=128), column 2 is PCA (R=64), column 3 is full-GKPCA (R=256) and column 4 is mult-GKPCA (R=256).

The second experiment involves denoising of handwritten digit images from the USPS dataset. This dataset contains 7291 training and 2007 testing vectors of length 256 dimensions (16×16 pixels). We aim to learn an image model for the handwritten digits by applying (kernel) PCA on the training set. Images from the testing set are corrupted with normal noise of 0.25 s.d. before being denoised by centering, projecting and reconstructing it using the mean and (kernel) principal components. We compare 3 methods for the task: PCA, full-GKPCA and mult-GKPCA. For the kernel methods, we use the Gaussian kernel of width 4. For mult-GKPCA, we use the intrinsic 10-digit partition of the 7291 vectors. During filtering, a stopping criterion of 0.1 mse is employed. During incremental KPCA, a maximum of 600 kernel principal components are retained. For full-GKPCA, filtering is conducted until the number of filtered vectors is the same as the total number of filtered vectors from mult-GKPCA. For PCA, due to the sheer size of the training set, the incremental method [8] is used. We use the mse between an original test image and its denoised version as a fidelity measure. Figure 3 shows some sample denoising results which indicate that the kernel methods perform generally better than PCA. Figure 4 depicts the average mse's of the 3 methods which show that mult-GKPCA performs almost equally as well as full-GKPCA. However, the total training duration for full-GKPCA is 19447.56s, while mult-GKPCA requires *only* 858.40s! Also, for these methods, the expansion size is only 3623 (out of 7291) vectors.

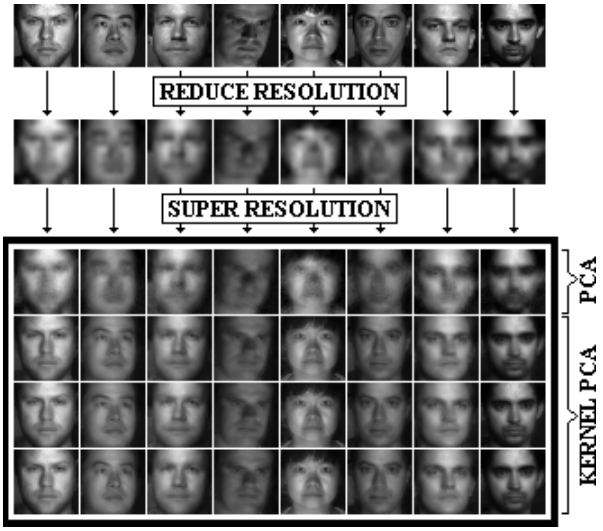

Figure 5. Yale face image super-resolution. Rows 1 and 2 respectively show the original image and low-resolution image. Rows 3 to 6 respectively show super-resolution results of PCA, KPCA, full-GKPCA and mult-GKPCA.

The third experiment involves performing super-resolution on face images from Yale Face Database B [6]. This dataset contains 5760 images of 10 subjects in different poses and lighting conditions. We crop and resize the face images to 36×36 pixels. The dataset is partitioned into 2 disjoint subsets: a training set with 5000 images and a testing set with 760 images. We aim to learn a model for the

[1] In [11], the Nyström method was forced to have the same execution time as the Kronecker method (hence the result in Figure 2), but this caused it to produce inferior classification rates. To achieve similar rates, the training time has to be prolonged for better KPCA approximation.

Proceedings of the IEEE International Conference
on Video and Signal Based Surveillance (AVSS'06)
0-7695-2688-8/06 $20.00 © 2006 IEEE

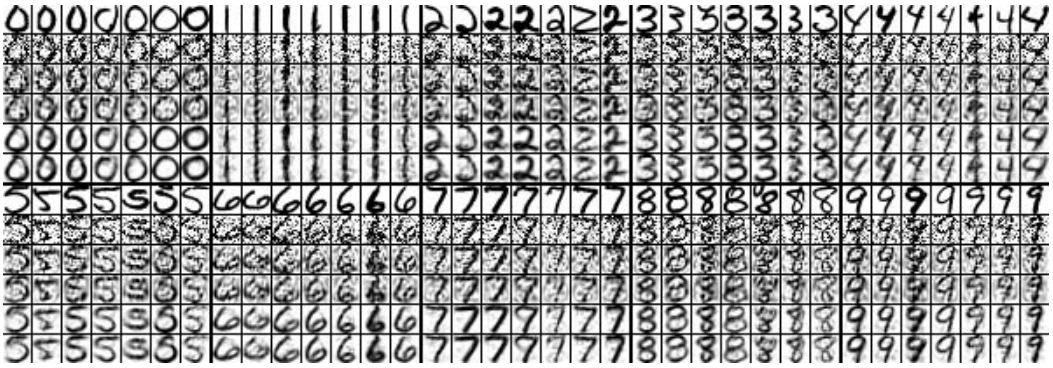

Figure 3. USPS denoising. For each digit block, row 1 is the original test images, row 2 the corrupted versions, row 3 is denoised using PCA (R=128), row 4 is denoised using PCA (R=64), row 5 is denoised using full-GKPCA (R=256) and row 6 is denoised using mult-GKPCA (R=256).

face image distribution by performing (kernel) PCA on the training set. Images from the testing set are blurred (through resizing and interpolation) before being enhanced by centering, projecting and reconstructing it using the mean and (kernel) principal components. We compare 4 methods for the task: PCA, KPCA, full-GKPCA and mult-GKPCA. For the kernel methods, we use the Gaussian kernel of width 1000. For mult-GKPCA, we use the intrinsic 10-subject partition of the 5000 vectors. A maximum of 250 vectors are filtered from each block. During incremental KPCA, a maximum of 1000 kernel principal components are retained. As in the USPS experiment, we use the mse between an original test image and its enhanced version as an objective score. PCA achieved the lowest average mse of 238.11, while KPCA, full-GKPCA and mult-GKPCA respectively achieved 286.29, 334.82 and 350.42. However, as shown in Figure 5, enhancement by the kernel methods are visually much better than PCA. Such results were obtained by [7] as well (refer to the paper for an explanation). What is crucial here is that mult-GKPCA performed as well as KPCA and full-GKPCA. In terms of training speed however, KPCA and full-GKPCA respectively needed 6132.08s and 4541.93s, while mult-GKPCA needed 1224.80s only.

7. Conclusion

In this paper, we proposed a novel method for evaluating the KPCA on large scale datasets. The basis of our solution lies in partitioning a large dataset into many smaller data blocks. We proposed a method to merge the individual GKPCA-processed data blocks to obtain the overall KPCA. Through experiments, we showed that our method presents a substantial improvement in training speed over previous approaches, while maintaining close approximation to the ground truth KPCA. A major factor contributing to the reduction of the training duration is that we perform GKPCA on multiple small data blocks rather than the overall dataset. Finally, we demonstrated 2 practical computer vision applications using the proposed method.

References

[1] D. Achlioptas, F. McSherry, and B. Schölkopf. Sampling techniques for kernel methods. In *Advances in NIPS*, 2001.

[2] C. Blake, E. Keogh, and C. Merz. UCI repository of machine learning databases, Department of Computer Science, University of California, Irvine.

[3] T.-J. Chin, K. Schindler, and D. Suter. Incremental kernel SVD for face recognition with image sets. In *FG*, 2006.

[4] N. Cristianini and J. Shawe-Taylor. *Kernel methods for pattern analysis*. Cambridge Uni. Press, 2004.

[5] V. Franc. *Optimization algorithms for kernel methods*. PhD thesis, Centre for Machine Perception, Czech Technical University, 2005.

[6] A. Georghiades, P. Belhumeur, and D. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE PAMI*, 23(6):643–660, 2001.

[7] K. I. Kim, M. O. Franz, and B. Schölkopf. Iterative kernel principal component analysis for image modeling. *IEEE PAMI*, 27(9):1351–1366, 2005.

[8] J. Lim, D. Ross, R.-S. Lin, and M.-H. Yang. Incremental learning for visual tracking. In *Advances in NIPS*, pages 793–800, 2004.

[9] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *ICML*, 2000.

[10] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in NIPS*, 2001.

[11] G. Wu, Z. Zhang, and E. Chang. Kronecker factorization for speeding up kernel machines. In *SIAM International Conference on Data Mining (SDM)*, 2005.

Proceedings of the IEEE International Conference
on Video and Signal Based Surveillance (AVSS'06)
0-7695-2688-8/06 $20.00 © 2006 IEEE