GPCA: An Efficient Dimension Reduction Scheme for Image Compression and Retrieval

Jieping Ye
Dept. of Computer Science
University of Minnesota
jieping@cs.umn.edu

Ravi Janardan
Dept. of Computer Science
University of Minnesota
janardan@cs.umn.edu

Qi Li
Dept. of Computer Science
University of Delaware
qili@cis.udel.edu

ABSTRACT

Recent years have witnessed a dramatic increase in the quantity of image data collected, due to advances in fields such as medical imaging, reconnaissance, surveillance, astronomy, multimedia etc. With this increase has come the need to be able to store, transmit, and query large volumes of image data efficiently. A common operation on image databases is the retrieval of all images that are similar to a query image. For this, the images in the database are often represented as vectors in a high-dimensional space and a query is answered by retrieving all image vectors that are proximal to the query image in this space, under a suitable similarity metric. To overcome problems associated with high dimensionality, such as high storage and retrieval times, a dimension reduction step is usually applied to the vectors to concentrate relevant information in a small number of dimensions. Principal Component Analysis (PCA) is a well-known dimension reduction scheme. However, since it works with vectorized representations of images, PCA does not take into account the spatial locality of pixels in images. In this paper, a new dimension reduction scheme, called Generalized Principal Component Analysis (GPCA), is presented. This scheme works directly with images in their native state, as two-dimensional matrices, by projecting the images to a vector space that is the tensor product of two lower-dimensional vector spaces. Experiments on databases of face images show that, for the same amount of storage, GPCA is superior to PCA in terms of quality of the compressed images, query precision, and computational cost.

Categories and Subject Descriptors: H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms: Algorithms

Keywords: Dimension reduction, image compression, Principal Component Analysis, Singular Value Decomposition, tensor product, vector space

KDD'04, August 22–25, 2004, Seattle, Washington, USA.

1. INTRODUCTION

Recent years have witnessed an explosion in the the quantity and quality of image data (both still and video) generated by diverse applications, such as medical imaging, surveillance, reconnaissance, astronomy, multimedia, etc. Along with this increase has come the need to be able to store, transmit, and query large volumes of image data efficiently. While images are inherently 2D in nature, i.e., matrices of pixels, the mechanisms employed to operate on them often entail working with data in very high dimensions. The focus of this paper is on the design of a dimension reduction scheme for efficient image compression and retrieval, under limited storage.

A common type of query on image data is the retrieval of all images that are similar to a query image. Such a query is useful in content-based image retrieval and has received considerable attention in the database community [4, 5, 8]. To support such queries, the 2D images in the database are converted to a vector-based representation and a similarity query is answered by retrieving all vectors that are proximal to the query vector, under a suitably defined similarity metric. (The hope here is that proximity in the vector space is correlated to similarity in the image space.) Unfortunately, this image-to-vector transformation often results in very high-dimensional data. It is not uncommon for a 2D image of size 100×100 pixels to generate vectors whose dimensions run into several thousands. High dimensionality has a negative impact on virtually all aspects of image management, including image compression, storage, transmission, and retrieval.

A natural approach for achieving better performance is to reduce the dimensionality of the data. The idea is to apply a preprocessing step to the data so that most information is concentrated in a small number of dimensions. Besides reducing storage requirements and improving query performance, dimension reduction has the added benefit of often removing noise from the data, as such noise is usually concentrated in the excluded dimensions [2].

Principal Component Analysis (PCA) is a well-known scheme for dimension reduction [12]. This approach condenses most of the information in a dataset into a small number, p, of dimensions by projecting the data (subtracted by the mean) onto a p-dimensional axis system $\{\phi_i\}_{i=1}^{p}$ (p is usually pre-specified or determined by heuristics). The optimal axis system can be computed by the Singular Value Decomposition (SVD) [10]. The reduced dimensions are chosen in a way that captures essential features of the data with very little loss of information. PCA is popular because of its use

of multidimensional representations for the compressed format. Such representations allow database applications such as indexing to operate directly on the reduced representations without a first phase of reconstruction [3]. The work in [8, 13] adopts this approach for similarity-based image retrieval. However, traditional PCA is applicable only for data in vectorized representation. When raw data (say an image) is represented in matrix form, it has to be converted to a vector. A typical way to do this so-called "matrix-to-vector alignment" is to concatenate all the rows in the matrix together to get a single vector. There are two drawbacks to this matrix-to-vector alignment: First, spatial locality information may be lost, and, second, the alignment leads to higher time and space costs.

EXAMPLE 1.1. *Figure 1 shows an example of matrix-to-vector alignment. The 3 × 3 matrix on the left is aligned to get the single vector on the right. The numbers 1–9 denote corresponding positions in the matrix and vectorized representations. Positions 1 and 4 are neighbors in the matrix representation, while they are quite far away from each other in the vectorized representation. The same observations hold for positions 3 and 6, etc. Also, the size of the so-called covariance matrix in PCA is 9 × 9 in this case, as compared to the original 3 × 3 matrix.*

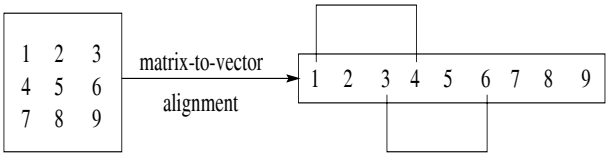

Figure 1: Matrix-to-vector alignment. 1–9 denote the positions in the matrix and vector formats.

Note that the SVD computation requires the whole data matrix to reside in main memory, which limits the applicability of PCA to relatively small image databases. Work has been done to address the high space complexity of PCA from the perspective of incremental SVD [14], where new data points are added to the database sequentially. But it has been found that most incremental algorithms tend to degrade in performance noticeably and frequent update is required [14]. Some recent work [1, 7, 9] applies random sampling to speed up the SVD computation. However, the effectiveness of these approaches is dependent on the spectral structure of the data matrix. There is little work in addressing the issue of spatial information loss due to the matrix-to-vector alignment in traditional PCA. The above two drawbacks are the motivation behind this paper.

1.1 Our results

In this paper, we propose *generalized PCA* (GPCA) for dimension reduction which aims to overcome the drawbacks in traditional PCA. As stated earlier, traditional PCA works on data in vectorized representation and it computes the p-dimensional axis system such that the projection of the data points (subtracted by the mean) onto the vector space spanned by this axis system has the largest variance among all p-dimensional axes systems. In contrast, the proposed GPCA algorithm deals with data in its native matrix representation and considers the projection onto a space, which is the tensor product of two vector spaces. More specifically, for given integers ℓ_1 and ℓ_2, GPCA computes the (ℓ_1, ℓ_2)-dimensional axis system $u_i \otimes v_j$, for $i = 1, \cdots, \ell_1$ and $j = 1, \cdots, \ell_2$, where $\otimes$ denotes the tensor product, such that the projection of the data points (subtracted by the mean) onto this axis system has the largest variance among all (ℓ_1, ℓ_2)-dimensional axes systems. We formulate GPCA as an optimization problem in Section 4. Unlike traditional PCA, there is no closed form solution to GPCA. We thus derive an iterative procedure for GPCA. Our empirical results show that the algorithm usually converges to a (local) maximum within two iterations. We perform extensive experiments on four well-known face datasets to evaluate the effectiveness of GPCA and compare it with traditional PCA. The four image datasets consist of gray scale images with sizes varying from 88×101 to 220×175. All images represent human faces in frontal view, roughly aligned with the image boundaries.

Our experiments show that:

- GPCA has distinctly lower computational cost than PCA.

Furthermore, given the same amount of space to store the transformation matrices and the compressed images:

- GPCA yields compressed images of significantly better visual quality than PCA;
- GPCA achieves significantly higher query precision than PCA;
- GPCA uses transformation matrices that are much smaller than PCA.

The rest of this paper is organized as follows. Section 2 illustrates the effect of dimension reduction on retrieval. Section 3 gives a brief introduction to traditional PCA. Section 4 introduces generalized PCA. Experimental results are presented in Section 5. Conclusions are offered in Section 6.

2. BACKGROUND

In this section, we present the background on the K-Nearest-Neighbor (K-NN) query, distance measures, the effect of dimension reduction on a K-NN query due to loss of information, and query precision (a common measure for the effectiveness of dimension reduction schemes).

2.1 K-Nearest-Neighbor query

One of the most common queries on a content-based retrieval system is of the form:

"Find the K most similar objects to a query object."

Such a request can be formulated as a K-Nearest-Neighbor query (K-NN query): Given a dataset S, a query object $q \in S$, and an integer $K \geq 1$, the K-NN query, selects a subset, $R(q, S)$, of K elements from S that have the smallest distance from q. That is,

(i) $R(q, S) \subset S$;

(ii) $|R(q, S)| = K$;

(iii) $\forall w \in R(q, S)$ and $\forall v \in S - R(q, S)$, $d(q, w) \leq d(q, v)$,

where $d(\cdot, \cdot)$ is a suitable distance measure (See Section 2.2).

EXAMPLE 2.1. *Consider a dataset, S, of six points A, B, ⋯, F in 2D, as in Figure 2. Under the Euclidean distance metric, a 2-NN query with point C returns B and D.*

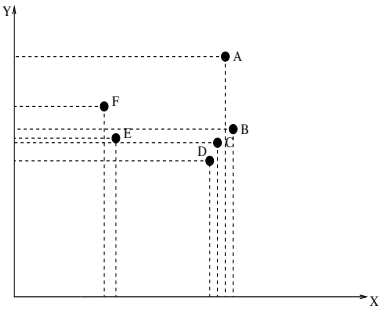

Figure 2: Illustrating a K-NN query on a 2D dataset.

2.2 Distance measures

An image is represented naturally as a matrix $X \in \mathbb{R}^{r \times c}$, where r is the number of rows and c is the number of columns. By a matrix-to-vector alignment, an image can be embedded in an N-dimensional space, $\mathbb{R}^N$, for $N = r \times c$. One way to measure the distance between two images is to compute the Euclidean distance between the corresponding vectors in $\mathbb{R}^N$. The Euclidean distance between two vectors is the 2-norm of the difference of the two vectors. The 2-norm of a vector is defined as follows:

DEFINITION 2.1. *The 2-norm of a vector $x \in \mathbb{R}^N$ is defined as $||x||_2 = \sqrt{x^T \cdot x} = \sqrt{\sum_{i=1}^{N} x_i^2}$, where x_i denotes the i-th coordinate of x.*

The distance between two vectors in a reduced k-dimensional space, $\mathbb{R}^k$, can be measured similarly.

The notion of distance between two vectors can be extended to matrices, where the *distance between two matrices* is the Frobenius norm of the difference of the two matrices. The Frobenius norm of a matrix is defined as follows:

DEFINITION 2.2. *The Frobenius norm of a matrix $M = (m_{ij}) \in \mathbb{R}^{r \times c}$ is defined as*

$$||M||_F = \sqrt{\sum_{i=1}^{r} \sum_{j=1}^{c} m_{ij}^2}.$$

2.3 Effect of dimension reduction on K-NN queries

Reducing data dimensionality may result in sufficient loss of information to affect the output of K-NN queries. Consider the example in Figure 2. If we retain only the X-coordinates of the points, then the 2-nearest neighbors of point C are A and D. Likewise, if we retain only the Y-coordinates of the points, then the 2-nearest neighbors of point C are B and E. However, the 2-nearest neighbors of point C in the original space are B and D. In either case, there is a loss of distance information from 2D to 1D that affects the output of the 2-NN query.

A common technique to minimize the loss of information is to apply a linear transformation on the data so that most of the information gets concentrated in a few dimensions. A well-known solution for this is the Principal Component Analysis (PCA), discussed in more detail in Section 3. Figure 3 shows the result of applying PCA on the six data points from Figure 2. First, a new set of coordinate axes $(X1, Y1)$ is computed. Then information about the input points is retained along one of the two axes, in this case $X1$. If we retain only the $X1$-coordinates of the points, then the 2-nearest neighbors of point C are B and D. Thus, the 2-NN information for C is preserved.

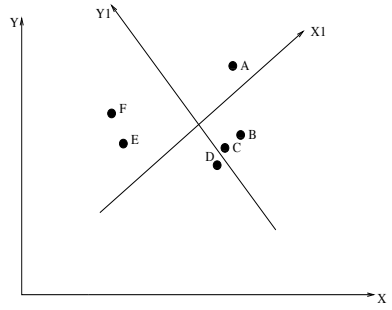

Figure 3: Change of axes from (X, Y) to $(X1, Y1)$.

2.4 Measure of effectiveness of dimension reduction algorithms

We can measure the effectiveness of a dimension reduction algorithm using the notion of query precision to compare the result of a query in the original N-dimensional space with the result in the reduced k-dimensional space. This approach was previously used for evaluating SVD-based dimension reduction algorithms in [14, 11].

Let S^N denote the set of data points (images) in the original N-dimensional space and let S^k denote the set of data points in the reduced k-dimensional space. Let $R(q, S^N)$ be the subset of points in S^N returned by a K-NN query with point q in S^N and let $R(q, S^k)$ be the subset of points in S^k returned by a K-NN query with the reduced-dimensional version of q in S^k. The *query precision* can be defined as follows:

$$\text{Precision}(q, N, k) = \frac{|R(q, S^N) \cap R(q, S^k)|}{|R(q, S^N)|}.$$

Note that for a K-NN query, with query point q, $|R(q, S^N)| = |R(q, S^k)| = K$. For the example in Figure 2, the precision of the 2-NN query with point C is 0.5 using either the X-dimension or Y-dimension.

Key notations used in the rest of this paper are listed in Table 1.

3. PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is one of the best-known methods for dimension reduction [12]. PCA was first applied to reconstruct human faces in [15], in the context of image compression. The Eigenface technique was developed in [16].

Assume that the n images in the dataset are originally represented in matrix form as $A_i \in \mathbb{R}^{r \times c}$, for $i = 1, \cdots, n$, where r and c are the number of rows and columns in the image, respectively. In vectorized representation, each image A_i is represented by a single vector a_i by concatenating

Notation	Description
n	number of points in the dataset
A_i	i-th image in matrix representation
a_i	i-th image in vectorized representation
r	number of rows in A_i
c	number of columns in A_i
N	dimension of a_i ($N = r * c$)
L	left reduction matrix by GPCA
R	right reduction matrix by GPCA
D_i	reduced representation of A_i by GPCA
d	number of rows and columns in D_i
G	transformation matrix by PCA
p	reduced dimension by PCA

Table 1: Notation

all the rows in A_i together in order (a so-called matrix-to-vector alignment). The dimension of the image vector a_i is thus $N = r \times c$. The advantage of using a vectorized representation is that the n images can be represented compactly by a single data matrix $A \in \mathbb{R}^{n \times N}$, where each row of A corresponds to a single image vector $a_i \in \mathbb{R}^N$. Later computations, including the covariance matrix defined below, come directly from the data matrix A.

DEFINITION 3.1. *The* covariance matrix Σ *of the dataset* $\{a_i\}_{i=1}^n \in \mathbb{R}^N$ *is* $\Sigma = Z^T Z$, *where the* i*-th row of the matrix* $Z \in \mathbb{R}^N$ *is* $a_i - m$ *and* $m = \frac{1}{n}\sum_{i=1}^n a_i$ *is the mean of the dataset.*

PCA determine the N eigenvectors of the $N \times N$ covariance matrix Σ, in which the (i, j)-th entry consists of the covariance between the dimensions i and j. The covariance matrix Σ can be diagonalized by computing the SVD of the matrix Z as $Z = UD\Phi^T$. By diagonalizing the covariance matrix as $\Sigma = ZZ^T = (UD\Phi^T)^T(UD\Phi^T) = \Phi\Lambda\Phi^T$, where $\Lambda = D^T D$, we can obtain the eigenvalues from Λ and the orthonormal eigenvectors from Φ. The eigenvalues in Λ are ordered in non-increasing order along the diagonal. Let $\{\phi\}_{i=1}^N$ be the set of eigenvectors corresponding to the columns in Φ.

The first step for performing the dimension reduction is to transform the data onto the new axis system $\{\phi_1, \cdots, \phi_N\}$. Hence, for a given data point $x = (x_1, \cdots, x_N)$, the coordinates in the new axis system are $(x \cdot \phi_1, \cdots, x \cdot \phi_N)$. In the actual dimension reduction step, down to dimension p, we retain the p eigenvectors corresponding to the largest p eigenvalues of Σ. These p eigenvectors are $\{\phi_1, \cdots, \phi_p\}$. When the data point x is projected onto the subspace spanned by the above p eigenvectors, the corresponding coordinates are $(x \cdot \phi_1, \cdots, x \cdot \phi_p)$.

DEFINITION 3.2. *Let* $S = \{a_1, \cdots, a_n\}$ *be a set of vectors in* $\mathbb{R}^N$. *Then the* variance *of* S *is defined as* $var(S) = \frac{1}{n-1}\sum_{i=1}^n ||a_i - m||_2^2$, *where* $m = \frac{1}{n}\sum_{i=1}^n a_i$ *is the mean of* S, *and* $||\cdot||_2$ *denotes the* 2*-norm of a vector as defined in Section 2.*

The intuition behind PCA is that it maximizes the variance of the projections of the data points (subtracted by the mean) onto the p-dimensional axis system $\{\phi_1, \cdots, \phi_p\}$, as stated below:

PROPOSITION 3.1. *Let* $G = [\phi_1, \cdots, \phi_p]$ *be the matrix consisting of the eigenvectors corresponding to the* p *largest eigenvalues from* Σ. *Then* G *solves the following maximization problem:*

$$G = \arg \max_{G \in \mathbb{R}^{N \times p} : G^T G = I_p} \frac{1}{n-1} \sum_{i=1}^n ||(a_i - m)G||_2^2, \quad (1)$$

where a_i *is the* i*-th row of data matrix* A, $m = \frac{1}{n}\sum_{i=1}^n a_i$ *is the mean, and* I_p *is the* $p \times p$ *identity matrix. That is, the projections of the data points (subtracted by the mean) onto the* p*-dimensional axis system* $\{\phi_1, \cdots, \phi_p\}$ *have the largest variance among all* p*-dimensional axes systems.*

4. GENERALIZED PCA

The key difference between PCA and the generalized PCA (GPCA) method that we propose in this paper is in the representation of image data. While PCA uses a vectorized representation of the 2D image matrix, GPCA works with a representation that is closer to the 2D matrix representation (as illustrated schematically in Figure 4) and attempts to preserve spatial locality of the pixels.

We will see later in this section that the matrix representation in GPCA leads to SVD computations on matrices with much smaller sizes. More specifically, GPCA involves the SVD computation on matrices with sizes $r \times r$ and $c \times c$, which are much smaller than the matrices in PCA (where the dimension is $n \times (r \times c)$). This reduces dramatically the time and space complexities of GPCA as compared to PCA. Furthermore, experimental results presented in Section 5 show superior performance of GPCA over PCA in terms of image compression and query precision. This is partly due to the fact that images are two-dimensional signals and there are spatial locality properties intrinsic to images that the representation used by GPCA seems to take advantage of.

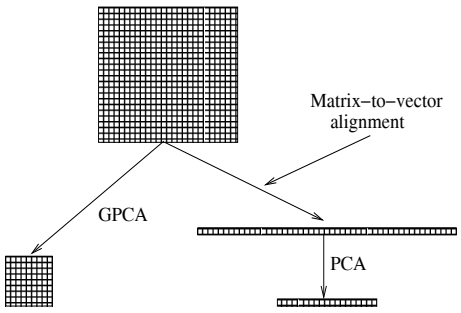

Figure 4: Schematic view of the key difference between GPCA and PCA. GPCA works on the original matrix representation of images directly, while PCA applies matrix-to-vector alignment first and works on the vectorized representation of images, which may lead to loss of spatial locality information.

4.1 The GPCA algorithm

In GPCA, we consider images as two dimensional signals and we consider the following (ℓ_1, ℓ_2)-dimensional axis system: $u_i \otimes v_j$, for $i = 1, \cdots, \ell_1$ and $j = 1, \cdots, \ell_2$, where $\otimes$ denotes the tensor product. (We show how to compute the vectors $u_i \in \mathbb{R}^{r \times 1}$ and $v_j \in \mathbb{R}^{c \times 1}$ later.) For a given

matrix $X \in \mathbb{R}^{r\times c}$, its projection onto the (i,j)-th coordinate, i.e., $u_i \otimes v_j$ is $u_i \cdot X \cdot v_j$. From Proposition 3.1, the p-dimensional axis system $\{\phi_1, \cdots, \phi_p\}$ in PCA is computed such that the projections of the data points (subtracted by the mean) onto this axis system have the maximum variance among all p-dimensional axes systems. Similarly, in GPCA, we compute an optimal (ℓ_1, ℓ_2)-dimensional axis system $u_i \otimes v_j$, for $i = 1, \cdots, \ell_1$ and $j = 1, \cdots, \ell_2$, such that the projections of the data points (subtracted by the mean) onto this axis system have the maximum variance among all (ℓ_1, ℓ_2)-dimensional axes systems. Unlike PCA, however, the projections of the data points onto the (ℓ_1, ℓ_2)-dimensional axis system in GPCA are matrices, instead of vectors. Similar to Definition 3.2, the mean and variance of a set of matrices can be defined as follows:

DEFINITION 4.1. *Let $S = \{X_1 \cdots, X_n\}$ be a set of matrices in $\mathbb{R}^{r\times c}$. Then the* variance *of S is defined as $var(S) = \frac{1}{n-1}\sum_{i=1}^{n} ||X_i - \bar{X}||_F^2$, where $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ is the mean of S, and $||\cdot||_F$ denotes the Frobenius norm of a matrix.*

EXAMPLE 4.1. *To illustrate this, let's consider the projection of X onto the $(2,2)$-dimensional axis system $u_i \otimes v_j$, for $i = 1, 2$ and $j = 1, 2$, where $X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 3 & 1 \\ 2 & 3 & 1 \end{pmatrix}$, $u_1 = v_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, and $u_2 = v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$. Then the projection of X onto $u_1 \otimes v_1$ is $u_1 \cdot X \cdot v_1 = (1,0,0)\begin{pmatrix} 1 & 2 & 3 \\ 4 & 3 & 1 \\ 2 & 3 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 1$. Similarly, the projections of X onto $u_1 \otimes v_2$, $u_2 \otimes v_1$ and $u_2 \otimes v_2$ are 2, 4, 3, respectively. Hence, the the projection of X onto the above $(2,2)$-dimensional axis system is $\begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$.*

A simple way to compute the projection above is to form two matrices $L = (u_1, u_2)$ and $R = (v_1, v_2)$. The projection can then be computed by $L^T X R \in \mathbb{R}^{2\times 2}$. Hence to compute the optimal axis system is equivalent to computing optimal matrices L and R with orthonormal columns.

Let $A_i \in \mathbb{R}^{r\times c}$, for $i = 1, \cdots, n$ be the n images in the dataset and $M = \frac{1}{n}\sum_{i=1}^{n} A_i$ be their mean. Let $\tilde{A}_i = A_i - M$, for all i. Then the variance of the projections of $\{\tilde{A}_i\}_{i=1}^n$ onto the (ℓ_1, ℓ_2)-dimensional axis system $u_i \otimes v_j$, for $i = 1, \cdots, \ell_1$ and $j = 1, \cdots, \ell_2$, can be computed as

$$\text{var}(L, R) = \frac{1}{n-1}\sum_{i=1}^{n} ||L^T \tilde{A}_i R||_F^2,$$

where $L = [u_1, \cdots, u_{\ell_1}] \in \mathbb{R}^{r\times \ell_1}$ and $R = [v_1, \cdots, v_{\ell_2}] \in \mathbb{R}^{c\times \ell_2}$.

Formulation of GPCA GPCA aims to compute two matrices $L \in \mathbb{R}^{r\times \ell_1}$ and $R \in \mathbb{R}^{c\times \ell_2}$ with orthonormal columns, such that the variance $\text{var}(L, R)$ is maximum.

Unfortunately, there is no closed form solution to the maximization problem [17]. The main observation, which leads to an iterative algorithm for GPCA, is stated in the following theorem:

THEOREM 4.1. *Let L, R be the matrices maximizing the variance $var(L, R) = \frac{1}{n-1}\sum_{i=1}^{n} ||L^T \tilde{A}_i R||_F^2$. Then*

- *For a given R, matrix L consists of the ℓ_1 eigenvectors of the matrix $M_L = \sum_{i=1}^{n} \tilde{A}_i R R^T \tilde{A}_i^T$ corresponding to the largest ℓ_1 eigenvalues.*
- *For a given L, matrix R consists of the ℓ_2 eigenvectors of the matrix $M_R = \sum_{i=1}^{n} \tilde{A}_i^T L L^T \tilde{A}_i$ corresponding to the largest ℓ_2 eigenvalues.*

PROOF. $\sum_{i=1}^{n} ||L^T \tilde{A}_i R||_F^2$ can be written as

$$\sum_{i=1}^{n} \text{trace}(L^T \tilde{A}_i R R^T \tilde{A}_i^T L) = \text{trace}\left(L^T M_L L\right),$$

where the trace of a matrix is the sum of the diagonal entries in the matrix. Hence, for fixed R, the maximum of $\sum_{i=1}^{n} ||L^T \tilde{A}_i R||_F^2$ is obtained, if and only if trace $\left(L^T M_L L\right)$ is also maximized.

Let the eigen-decomposition of M_L be $M_L = U\Lambda U^T$, where U has orthonormal columns, $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_r)$, and $\lambda_1 \geq \cdots \geq \lambda_r \geq 0$. It can be shown that trace $\left(L^T M_L L\right) \leq \sum_{i=1}^{\ell_1} \lambda_i$, and the maximum is obtained if $L \in \mathbb{R}^{r\times \ell_1}$ consists of the ℓ_1 eigenvectors of the matrix M_L corresponding to the largest ℓ_1 eigenvalues. (We omit the proof due to space limitation.)

Similarly, by the commutative property of the trace of matrices, that is, $\text{trace}(AB) = \text{trace}(BA)$, for any two matrices A and B, $\sum_{i=1}^{n} ||L^T \tilde{A}_i R||_F^2$ can also be written as

$$\sum_{i=1}^{n} \text{trace}(R^T \tilde{A}_i^T L L^T \tilde{A}_i R) = \text{trace}\left(R^T M_R R\right).$$

Hence, for fixed L, the maximum of $\sum_{i=1}^{n} ||L^T \tilde{A}_i R||_F^2$ is obtained, only if $R \in \mathbb{R}^{c\times \ell_2}$ consists of the ℓ_2 eigenvectors of the matrix M_R, corresponding to the largest ℓ_2 eigenvalues. □

The maximization of the variance $\text{var}(L, R)$ bears some resemblance to the one in [6], where the co-clustering of the gene expression data is considered. However, the two matrices L and R in GPCA have orthonormal columns, while they satisfy a specific discrete structure in [6].

REMARK 4.1. *In typical images, the number of rows and the number of columns are comparable. Therefore, when reducing the dimension using GPCA, we take $\ell_1 = \ell_2 = d$. This is only for simplicity in the exposition; our method can be extended easily to the more general case.*

Theorem 4.1 provides us an iterative procedure for computing L and R. More specifically, for a fixed L, we can compute R by computing the eigenvectors of the matrix M_R. With the computed R, we can then update L by computing the eigenvectors of the matrix M_L. The procedure can be repeated until the result converges (we prove convergence in Section 4.2). Theoretically, the solution from this iterative procedure is a local solution. The solution depends on the initial choice, L_0, for L. Experiments show that choosing $L_0 = (I_d, 0)^T$, where I_d is the identity matrix, produces excellent results. We use this initial L_0 in all the experiments. Given L and R, the projection of $\tilde{A}_i$ onto the axis system by L and R can be computed by $D_i = L^T \tilde{A}_i R$.

Conversely, given L, R and $\{D_i\}_{i=1}^n$, we can reconstruct $\{\tilde{A}_i\}_{i=1}^n$ as $\tilde{A}_i \approx LD_iR^T$, i.e., $A_i \approx LD_iR^T + M$, for $i = 1, \cdots, n$, where M is the mean. The reconstruction error for $\tilde{A}_i$ can be computed as $E_i = ||\tilde{A}_i - LD_iR^T||_F = ||\tilde{A}_i - LL^T\tilde{A}_iRR^T||_F$. We define the *root mean square error* (RMSE), to measure the average reconstruction error as follows:

$$\text{RMSE} \equiv \sqrt{\frac{1}{n}\sum_{i=1}^{n} ||\tilde{A}_i - LL^T\tilde{A}_iRR^T||_F^2}. \qquad (2)$$

We will show in the next section that the iterative procedure for computing L and R converges, as measured by the RMSE value. More specifically, we will show that the iterative procedure monotonically decreases the RMSE value, hence it converges, since the RMSE value is bounded from below by 0. A user-defined threshold η is used to check the convergence (more details are given in the next section). The pseudo-code for the GPCA algorithm is given in **Algorithm 1**.

Algorithm 1 GPCA($A_1, \cdots, A_n, d$)
Input: $A_1, \cdots, A_n, d$
Output: $L, R, D_1, \cdots, D_n$
begin
0. $M = \frac{1}{n}\sum_{j=1}^{n} A_j$ // Compute the mean
1. for j from 1 to n do begin
2. $\quad \tilde{A}_j = A_j - M$
3. end
4. $L_0 \leftarrow (I_d, 0)^T$
5. $i \leftarrow 0$
6. RMSE(i) $\leftarrow \infty$
7. Do
8. $\quad$ form the matrix $M_R = \sum_{j=1}^{n} \tilde{A}_j{}^T L_i L_i^T \tilde{A}_j$
9. $\quad$ compute the d eigenvectors $\{\phi_j^R\}_{j=1}^d$ of M_R corresponding to the largest d eigenvalues
10. $\quad i \leftarrow i + 1$
11. $\quad R_i \leftarrow [\phi_1^R, \cdots, \phi_d^R]$
12. $\quad$ form the matrix $M_L = \sum_{j=1}^{n} \tilde{A}_j R_i R_i^T \tilde{A}_j{}^T$
13. $\quad$ compute the d eigenvectors $\{\phi_j^L\}_{j=1}^d$ of M_L corresponding to the largest d eigenvalues
14. $\quad L_i \leftarrow [\phi_1^L, \cdots, \phi_d^L]$
15. $\quad$ RMSE(i) $\leftarrow \sqrt{\frac{1}{n}\sum_{j=1}^{n} ||\tilde{A}_j - L_iL_i^T\tilde{A}_jR_iR_i^T||_F^2}$
16 Until (RMSE($i-1$) − RMSE(i) $\leq \eta$)
17. $L \leftarrow L_i$
18. $R \leftarrow R_i$
19. for j from 1 to n do begin
20. $\quad D_j \leftarrow L^T\tilde{A}_jR$
21. end
22. return($L, R, D_1, \cdots, D_n$)
end

4.2 Convergence of the GPCA algorithm

In this section, we prove the convergence of the iterative procedure (Lines 7–16 in the GPCA algorithm). The result is stated in following theorem:

THEOREM 4.2. *The Do-Until Loop in the* **Algorithm 1** *(Lines 7–16) monotonically decreases the RMSE value as defined in (2), hence the GPCA algorithm converges.*

PROOF. Theorem 4.1 implies that the update of the matrices R and L in Lines 11 and 14 of **Algorithm 1** increases the variance $\frac{1}{n-1}\sum_{i=1}^{n} ||L^T\tilde{A}_iR||_F^2$, hence decreases $\sum_{i=1}^{n}(||\tilde{A}_i||_F^2 - ||L^T\tilde{A}_iR||_F^2)$.

It is easy to check by Linear Algebra that

$$\sum_{i=1}^{n} ||\tilde{A}_i - LL^T\tilde{A}_iRR^T||_F^2 = \sum_{i=1}^{n}(||\tilde{A}_i||_F^2 - ||L^T\tilde{A}_iR||_F^2).$$

It follows that the RMSE value decreases monotonically. The Do-Until Loop in **Algorithm 1** thus converges, since the RMSE value is bounded from below by 0. □

Theorem 4.2 shows that we can check the convergence of **Algorithm 1** using the RMSE value. More specifically, let RMSE(i) and RMSE($i-1$) be the RMSE values at the i-th and $(i-1)$-th iterations from **Algorithm 1**. Then, the convergence of the GPCA algorithm can be determined by checking whether RMSE($i-1$) − RMSE(i) $< \eta$, for some small threshold $\eta > 0$. In all our experiments, we choose $\eta = 0.05$.

4.3 A detailed example

To illustrate the GPCA algorithm, we generate three matrices with integer entries (for simplicity). The dimensions of the matrices are 3×3. That is, the number of points $n = 3$, the number of rows $r = 3$, and the number of columns $c = 3$. We also set $d = 2$. The three matrices $\{A_i\}_{i=1}^3$ are:

$$A_1 : \begin{pmatrix} 1 & 1 & 2 \\ 4 & 8 & 6 \\ 0 & 2 & 3 \end{pmatrix}, A_2 : \begin{pmatrix} 6 & 8 & 5 \\ 3 & 5 & 7 \\ 2 & 2 & 3 \end{pmatrix}, A_3 : \begin{pmatrix} 2 & 3 & 8 \\ 2 & 2 & 8 \\ 1 & 5 & 3 \end{pmatrix}.$$

By subtracting their mean, we obtain $\{\tilde{A}_i\}_{i=1}^3$ as:

$$\tilde{A}_1 : \begin{pmatrix} -2 & -3 & -3 \\ 1 & 3 & -1 \\ -1 & -1 & 0 \end{pmatrix}, \tilde{A}_2 : \begin{pmatrix} 3 & 4 & 0 \\ 0 & 0 & 0 \\ 1 & -1 & 0 \end{pmatrix}, \tilde{A}_3 : \begin{pmatrix} -1 & -1 & 3 \\ -1 & -3 & 1 \\ 0 & 2 & 0 \end{pmatrix}.$$

In Line 4 of the GPCA algorithm, we set $L_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$.

The matrix M_R in Line 8 can be computed as

$$M_R = \sum_{i=1}^{3} \tilde{A}_i^T L_0 L_0^T \tilde{A}_i = \begin{pmatrix} 16 & 25 & 1 \\ 25 & 44 & 0 \\ 1 & 0 & 20 \end{pmatrix}.$$

By computing the two eigenvectors of M_R corresponding to the largest two eigenvalues ($d = 2$) as in Line 9, we obtain

$$R_1 = \begin{pmatrix} -0.5058 & 0.0332 \\ -0.8626 & -0.0346 \\ -0.0131 & 0.9989 \end{pmatrix},$$

which corresponds to Line 11 of the GPCA algorithm. With the computed R_1, the matrix M_L in Line 12 can be computed as

$$M_L = \sum_{i=1}^{3} \tilde{A}_i R_1 R_1^T \tilde{A}_i = \begin{pmatrix} 57.4279 & -0.7428 & 0.6993 \\ -0.7428 & 21.2650 & -9.6046 \\ 0.6993 & -9.6046 & 4.9851 \end{pmatrix},$$

By computing the two eigenvectors of M_R corresponding to the largest two eigenvalues as in Line 13, we obtain

$$L_1 = \begin{pmatrix} -0.9995 & -0.0305 \\ 0.0253 & -0.9071 \\ -0.0179 & 0.4198 \end{pmatrix}.$$

With the computed R_1 and L_1, we can compute the root mean square error (RMSE) in Line 15 as

$$\text{RMSE}(1) = \sqrt{\frac{1}{3}\sum_{i=1}^{3} ||\tilde{A}_i - L_1 L_1^T \tilde{A}_i R_1 R_1^T||_F^2} = 1.2722.$$

The above procedure can be repeated and at the end of the second iteration (details omitted), we obtain

$$R_2 = \begin{pmatrix} -0.4904 & 0.0391 \\ -0.8714 & -0.0328 \\ -0.0094 & 0.9987 \end{pmatrix}, L_2 = \begin{pmatrix} -0.9996 & -0.0297 \\ 0.0257 & -0.9068 \\ -0.0151 & 0.4206 \end{pmatrix}.$$

and the new RMSE value can be computed by

$$\text{RMSE}(2) = \sqrt{\frac{1}{3}\sum_{i=1}^{3} ||\tilde{A}_i - L_2 L_2^T \tilde{A}_i R_2 R_2^T||_F^2} = 1.2696.$$

Since $\text{RMSE}(1) - \text{RMSE}(2) = 0.0026 < \eta = 0.05$, the algorithm exits the Do-Until Loop in Lines 7–16 after the second iteration. We obtain $L \leftarrow L_2$ and $R \leftarrow R_2$ in Lines 17 and 18 respectively. The projections of $\{\tilde{A}_i\}_{i=1}^3$ onto the $(2,2)$-dimensional axis system, which is the tensor product of the space spanned by the two columns of L and the other space spanned by the two columns of R, can be computed in Line 20 as

$$\begin{aligned} D_1 &= L^T \tilde{A}_1 R = \begin{pmatrix} -3.7219 & 2.9476 \\ 3.2720 & 1.0449 \end{pmatrix}, \\ D_2 &= L^T \tilde{A}_2 R = \begin{pmatrix} 4.9490 & 0.0127 \\ 0.3073 & 0.0307 \end{pmatrix}, \\ D_3 &= L^T \tilde{A}_3 R = \begin{pmatrix} -1.2272 & -2.9603 \\ -3.5794 & -1.0756 \end{pmatrix}, \end{aligned}$$

In summary, GPCA computes the optimal $L, R \in \mathbb{R}^{3\times 2}$ such that the original matrices $\{\tilde{A}_i\}_{i=1}^3 \in \mathbb{R}^{3\times 3}$ are transformed to $\{D_i\}_{i=1}^3 \in \mathbb{R}^{2\times 2}$.

4.4 Time complexity and memory requirement

4.4.1 Time complexity

We note that the most time-consuming steps of the GPCA algorithm are the formation of the matrices M_R and M_L (Lines 8 and 12 of **Algorithm 1**), the eigen-decompositions in Lines 9 and 13, and the computation of the reduced representation D_i in Line 20.

It takes $O(n \times d \times r \times (r+c))$ for computing M_R. The computation of the d eigenvectors on $M_R \in \mathbb{R}^{c\times c}$ takes $O(c^3)$. Similarly, it takes $O(n \times d \times c \times (r+c))$ for computing M_L and $O(r^3)$ for computing the d eigenvectors on $M_L \in \mathbb{R}^{r\times r}$ [10]. The computation of $D_i = (L^T(A_i R))$ using the given order is $O(r \times c \times d + r \times d^2) = O(r \times d \times (c+d))$. Hence, the total complexity of GPCA can be simplified as

$$O(I \times n \times (r+c)^2 \times d + n \times r \times d \times (c+d)),$$

where I is the number of iterations involved in the Do-Until Loop in Lines 7–16 of the GPCA algorithm and the time $O(c^3 + r^3)$ for computing the eigenvectors in Lines 9 and 13 of the GPCA algorithm is omitted, since it is usually negligible compared to the time for computing M_R and M_L.

Since $n \times r \times d \times (c+d) \leq n \times (r+c)^2 \times d$, the time complexity of GPCA is bounded above by $O((I+1) \times n \times (r+c)^2 \times d)$. Assuming $r \approx c \approx \sqrt{N}$, where $N = r \times c$, the time complexity of GPCA can be simplified as $O(InNd)$. Experiments in Section 5 show that the GPCA algorithm converges within two iterations, i.e., $I \leq 2$.

4.4.2 Memory requirement

The memory requirement of the GPCA algorithm can be analyzed through Lines 0, 8–9, 12–13 and 20 in **Algorithm 1**.

Lines 9, 13 and 20 in the algorithm only involves a single matrix with dimension $r \times c$. The most space-consuming steps are Lines 0, 8 and 12, where all A_i's are involved. The key observation is that the three matrices M, M_R and M_L in these three steps can be computed incrementally by reading A_i sequentially without losing any information. Hence the required memory for the GPCA algorithm can be as low as $O(r \times c)$.

The fact that GPCA computes the optimal solution without requiring all data points in the memory is a major advantage of GPCA over tradition PCA, especially for large databases. As mentioned in the Introduction, even though the memory requirement of PCA can be reduced using an incremental SVD algorithm or random sampling techniques, good performance is not guaranteed.

4.5 Storage

Both PCA and GPCA can be applied for image compression. The quality of compression is dependent on the space available for storing the transformation matrices and the reduced representations. In general, large storage leads to high quality of the compressed image and small storage leads to low quality. The experimental comparison of PCA and GPCA in the next section is based on the assumption that they both use the same amount of storage. Hence it is important to understand how to choose the reduced dimension for PCA and GPCA for a specific storage requirement.

The original dataset contains n images $\{A_i\}_{i=1}^n \in \mathbb{R}^{r\times c}$, hence it requires nN scalars to store the original n images, where $N = r \times c$.

GPCA compresses each of the n images of size $r \times c$ to size $d \times d$, while it needs to keep the two matrices $L \in \mathbb{R}^{r\times d}$ and $R \in \mathbb{R}^{c\times d}$ to reconstruct the original images. Hence it requires $(nd+r+c)d$ scalars in the reduced space by GPCA.

Next, consider PCA with p principal components. To store the p principal components, each in $\mathbb{R}^N$, it requires pN scalars. Each image is transformed to a reduced representation in $\mathbb{R}^p$, hence it requires np scalars to store the n reduced representations. In total, it requires $p(N+n)$ scalars in the reduced space by PCA.

In our experiments, we choose d for GPCA and p for PCA such that their storage requirements are (nearly) equal.

5. EXPERIMENTAL RESULTS

In this section, we experimentally evaluate the performance of the GPCA algorithm and compare it with PCA. Note that all the comparisons between GPCA and PCA are based on the assumption that they have the same storage requirement for the transformation matrices and the reduced

representations. All of our experiments are performed on a P4 1.80GHz Linux machine with 1GB memory. We divide this section into six parts. The four face image datasets used in the experiments are described in Section 5.1. Section 5.2 discusses the effect of the value of the reduced dimensionality, d, on GPCA. More specifically, we look at the effect of d on the query precision. The convergence property of GPCA is studied in Section 5.3. A detailed comparative study between GPCA and PCA is given in Section 5.4, where the comparison is based on the visual quality of the compressed images. Section 5.5 compares the effectiveness of GPCA and PCA in terms of query precision. In Section 5.6, we compare the efficiency of GPCA and PCA. Experiments show GPCA has distinctly smaller cost in time than traditional PCA.

For all the experiments, we use 10-fold cross validation to report the mean precision of a K-NN query with $K = 10$. In 10-fold cross validation, the entire dataset is split into ten subsets. We use all the data points in each of the subsets to query the set consisting of the union of the other nine subsets. The final precision reported is the mean precision of all queries.

5.1 Datasets

The following are the four face datasets in our study; all are publicly available.

- PIX[1] contains 300 face images of 30 persons. The image size of PIX image is 512×512. We subsample the images down to a size of $100 \times 100 = 10000$.

- Olivetti Research Laboratory (ORL) face dataset[2] is a well-known dataset for face recognition. It contains the face images of 40 persons, for a total of 400 images. The image size is 92×112. The face images are perfectly centralized. We use the whole image as an instance (i.e., the dimension of an instance is $92 \times 112 = 10304$).

- AR[3] is a large face image dataset. The instance of each face may contain large areas of occlusion, due to sun glasses and scarves. The existence of occlusion dramatically increases the within-class variances of AR face image data. We use a subset of AR. This subset contains 1638 face images of 126 persons. Its image size is 768×576. We first crop the image from row 100 to 500, and column 200 to 550, and then subsample the cropped images down to a size of $101 \times 88 = 8888$.

- PIE is a subset of the CMU–PIE face image dataset[4]. PIE contains 6615 face instances of 63 persons. More specifically, each person has $21 \times 5 = 105$ instance images taken under 21 different lighting/illumination conditions and 5 different poses. The image size of PIE is 640×480. We pre-processed each image using a similar technique as above. The final dimension of each instance is $220 \times 175 = 38500$.

[1]http://peipa.essex.ac.uk/ipa/pix/faces/manchester/test-hard/
[2]http://www.uk.research.att.com/facedatabase.html
[3]http://rvl1.ecn.purdue.edu/∼aleix/aleix_face_DB.html
[4]http://www.ri.cmu.edu/projects/project_418.html

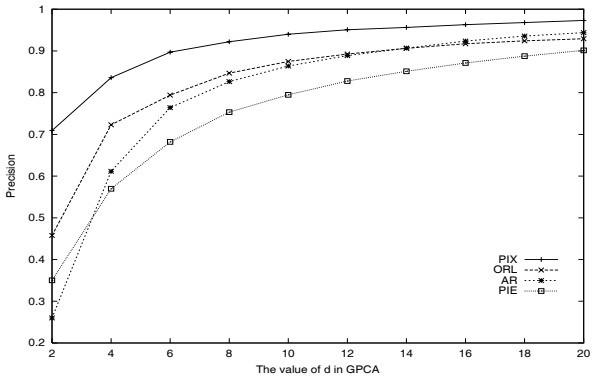

Figure 5: The effect of the value of d on GPCA, as measured by query precision

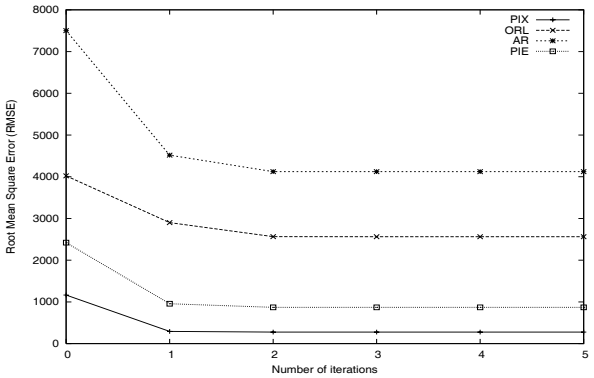

Figure 6: Convergence of GPCA

5.2 The effect of the value of d on GPCA

The value of d determines the dimensionality in the transformed space by GPCA. We did extensive experiments using different values of d on the four datasets. The results are summarized in Figure 5, where the x-axis denotes the value of d (between 2 and 20) and the y-axis denotes the query precision. As shown in Figure 5, the precision curves on all datasets have big jumps from $d = 2$ to 4 and 6.

As is clear from Figure 5, a larger value of d leads to a higher query precision, but a higher storage requirement. There is a clear trade-off between query precision and storage required. As discussed in last section, $(nd+r+c)d$ scalars are required in the reduced space by GPCA. In practice, we can determine the maximum value of d, for a given amount of available space.

5.3 Convergence of GPCA

We study the convergence of the GPCA algorithm in this experiment. The results for the four datasets: PIX, ORL, AR and PIE are shown in Figure 6, where the x-axis denotes the number of iterations, and the y-axis denotes the Root Mean Square Error (RMSE). We set $d = 20$ for the four datasets. Figure 6 shows that the RMSE drops dramatically during the first and second iterations and it stabilizes after two iterations.

5.4 Image quality under compression

Storage and transmission of large image data commonly apply image compression (coding) as a pre-processing step.

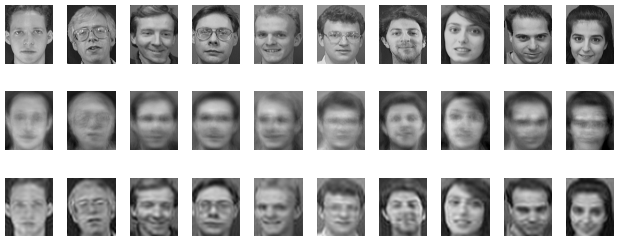

Figure 7: First row: raw images. Second row: images compressed by PCA. Third row: images compressed by GPCA.

Both PCA and GPCA can be applied for compression. We compare GPCA and PCA in terms of image compression, when using the same amount of storage.

We apply PCA and GPCA on the 400 images in the ORL dataset. We use $p = 15$ principal components in PCA and set $d = 20$ for GPCA correspondingly. The storage requirements for GPCA and PCA are, respectively, 164080 and 160560 scalars, and the transformation matrices have size 4080 and 154560, respectively. To illustrate the different performance of GPCA and PCA, we show the result of 10 images from 10 different persons in Figure 7. The 10 images in the first row are the raw images from the dataset. The 10 images in the second and third rows are the ones compressed by PCA ($p = 15$) and GPCA ($d = 20$) respectively. As can be seen, the images under GPCA are (visually) closer to the raw images.

As a second experiment, we apply PCA and GPCA on the 1638 images from the AR dataset, with $p = 62$ and $d = 20$. The storage requirements for GPCA and PCA are, respectively, 658980 and 652612 scalars, and the transformation matrices have size 3780 and 551056, respectively. We show the result on a single person (each person has 13 images) in Figure 8. The 13 images in the first row are the raw images from the dataset. The 13 images in the second row are the ones compressed by PCA ($d = 62$), and the 13 images in the third row are the ones compressed by GPCA ($d = 20$). Again, the quality of the GPCA images appears higher.

Figure 7 and Figure 8 show that GPCA yields visually higher quality images than PCA, when using the same amount of storage. This confirms our initial intuition that by working on the images in the matrix representation, GPCA is able to capture the spatial locality information intrinsic to the images much better than traditional PCA. The result in the next section further confirms this by showing the superior performance of GPCA over PCA in terms of query precision.

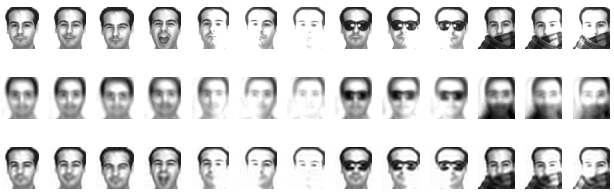

Figure 8: First row: raw images (same person). Second row: images compressed by PCA. Third row: images compressed by GPCA.

5.5 Effectiveness of GPCA measured by query precision

In this experiment, we evaluate the effectiveness of GPCA and compare it with PCA in terms of query precision as defined in Section 2. Figure 9 shows the precision curves of PCA and GPCA on three face image datasets: PIX, ORL, and AR. The x-axis denotes the value of d, and the y-axis denotes the query precision.

Note that the precision curve of PCA on the PIE dataset is not shown, since the corresponding data matrix is simply too large to reside in main memory. For all precision curves, the reduced dimension p in PCA is chosen such that both PCA and GPCA use the same amount of storage for the transformation matrices and the reduced representations. For example, on the AR dataset, $p = 3, 10, 22, 40, 62$ principal components are used in PCA, corresponding to $d = 4, 8, 12, 16, 20$ used in GPCA.

The main observations are:

- GPCA achieves noticeably higher precision than PCA on all datasets. This suggests that GPCA is able to capture spatial locality information better than PCA.
- The gap between the precision of GPCA and PCA decreases as the value of d increases. This is quite intuitive. A larger value of d leads to less information loss and higher precision for both PCA and GPCA.
- PCA is not applicable for the PIE dataset. The size of the PIE dataset is large, since both the number of data points and the dimension are large. PCA requires the whole data matrix in the main memory and is therefore not applicable in this case, while GPCA has smaller space requirements and is applicable for large datasets, like PIE.

The above experiments show that GPCA is a more effective dimension reduction scheme as it has significantly lower loss of information, i.e., high precision, when using the same amount of storage.

5.6 Efficiency of GPCA

In this experiment, we examine the efficiency of GPCA and compare with PCA. As shown in Figure 10, GPCA has distinctly less computational time than PCA. As the size of the dataset gets larger from PIX to ORL to AR, the speedup of GPCA over PCA increases. For the AR dataset, GPCA is almost two orders of magnitude faster compared to PCA.

Note that the CPU time of PCA on the PIE dataset is not shown, due to the same reason mentioned above.

6. CONCLUSIONS

A new dimension reduction algorithm, GPCA, is presented for face image databases. GPCA is a significant extension of PCA. The key difference between GPCA and PCA is that GPCA works on the matrix representation of images directly, while PCA uses a vector representation. GPCA has asymptotically minimum memory requirements, and lower time complexity than PCA, which is desirable for large face databases. GPCA also uses transformation matrices that are much smaller than PCA. This significantly reduces the space to store the transformation matrices and reduces the

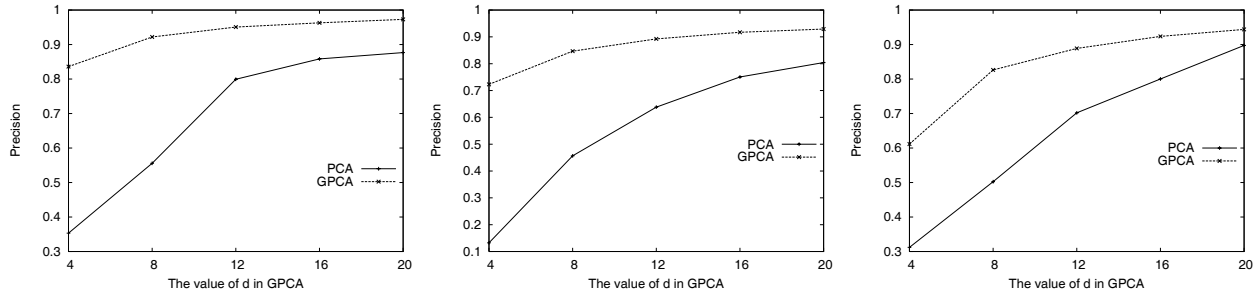

Figure 9: Comparison of query precision on PIX, ORL and AR image datasets (from left to right).

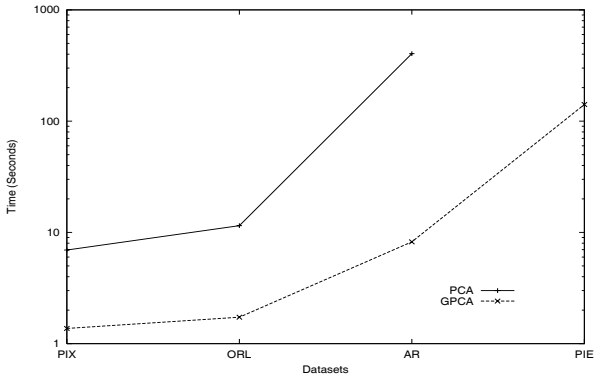

Figure 10: Comparison of running time (in log-scale) using the four datasets. Note that PCA is not applicable for the PIE dataset, due to its large size.

computational time in computing the reduced representation for a query image. Experiments show superior performance of GPCA over PCA, in terms of quality of compressed images and query precision, when using the same amount of storage.

Acknowledgement

Research of J. Ye and R. Janardan is sponsored, in part, by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAD 19-01-2-0014, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

7. REFERENCES

[1] D. Achlioptas and F. McSherry. Fast computation of low rank matrix approximations. In *ACM STOC Conference Proceedings*, pages 611–618, 2001.

[2] C. C. Aggarwal. On the effects of dimensionality reduction on high dimensional similarity search. In *ACM PODS Conference Proceedings*, Santa Barbara, California, USA, 2001.

[3] C. C. Aggarwal. Hierarchical subspace sampling: A unified framework for high dimensional data reduction, selectivity estimation, and nearest neighbor search. In *ACM SIGMOD Conference Proceedings*, Madison, Wisconsin, USA, 2002.

[4] P. Aigrain, H.J. Zhang, and D. Petkovic. Content-based representation and retrieval of visual media: A state-of-the-art review. *Multimedia Tools and Applications*, 3(3):179–202, 1996.

[5] V. Castelli, A. Thomasian, and C.S. Li. CSVD: Clustering and singular value decomposition for approximate similarity searches in high dimensional space. *IEEE TKDE*, 15(3):671–685, 2003.

[6] H. Cho, I.S. Dhillon, Y. Guan, and S. Sra. Minimum sum-squared residue co-clustering of gene expression data. In *SIAM Data Mining Conference proceedings*, pages 114–125, 2004.

[7] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. In *SODA Conference Proceedings*, pages 291–299, 1999.

[8] C. Faloutsos and K.I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD Conference Proceedings*, pages 163–174, San Jose, California, 1995.

[9] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. In *ACM FOCS Conference Proceedings*, pages 370–378, 1998.

[10] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996.

[11] H. Jin, B. C. Ooi, H. T. Shen, C. Yu, and A.Y. Zhou. An adaptive and efficient dimensionality reduction algorithm for high-dimensionality indexing. In *ICDE Conference Proceedings*, Bangalore, India, 2003.

[12] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[13] R. Ng and A. Sedighian. Evaluating multi-dimensional indexing structures for images transformed by principal component analysis. In *Proc. of the SPIE*, number 2670, pages 50–61, 1994.

[14] K. V. Ravi Kanth, D. Agrawal, A. E. Abbadi, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *ACM SIGMOD Conference Proceedings*, pages 166–176, 1998.

[15] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of Optical Society of America*, 4(3):519–524, 1987.

[16] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[17] J. Ye. Generalized low rank approximations of matrices. In *ICML Conference Proceedings*, 2004.