# Principal Component Analysis and Face Recognition

# Singular Value Decomposition

Slide credit: Juan Carlos Niebles

# Singular Value Decomposition (SVD)

- There are several computer algorithms that can “factorize” a matrix, representing it as the product of some other matrices.
- The most useful of these is the Singular Value Decomposition.
- Represents any matrix A as a product of three matrices: $U\Sigma V^T$

# Singular Value Decomposition (SVD)

- Where U and V are rotation matrices, and Σ is a scaling matrix.

$$U\Sigma V^T = A$$

$$\underset{U}{\begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}$$

# Dimensions

- In general, if A is m x n, then U will be m x m, Σ will be m x n, and $V^T$ will be n x n.

$$\underset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

# Singular Value Decomposition (SVD)

- U and V are always rotation matrices.
  - Geometric rotation may not be an applicable concept, depending on the matrix.
  - So we call them “unitary” matrices – each column is a unit vector.
- Σ is a diagonal matrix
  - The number of nonzero entries = rank of A
  - The algorithm always sorts the entries high to low.

$$\underset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

# SVD Applications

$$\underset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

- Look at how the multiplication works out, left to right:
- Column 1 of **U** gets scaled by the first value from **Σ**.

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

- The resulting vector gets scaled by row 1 of $\mathbf{V}^T$ to produce a contribution to the columns of **A**

# SVD Applications

- We're building **A** as a linear combination of the columns of ***U***
- Using all columns of ***U***, we'll rebuild the original matrix perfectly
- But, in real-world data, often we can just use the first few columns of ***U*** and we'll get something close (e.g. the first ***Apartial***, above)

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$+ \underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

$$= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

# SVD Applications

- We can call those first few columns of ***U*** the *Principal Components* of the data.
- They show the major patterns that can be added to produce the columns of the original matrix.
- The rows of $\mathbf{V^T}$ show how the *principal components* are mixed to produce the columns of the matrix.

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$+ \underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

$$= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

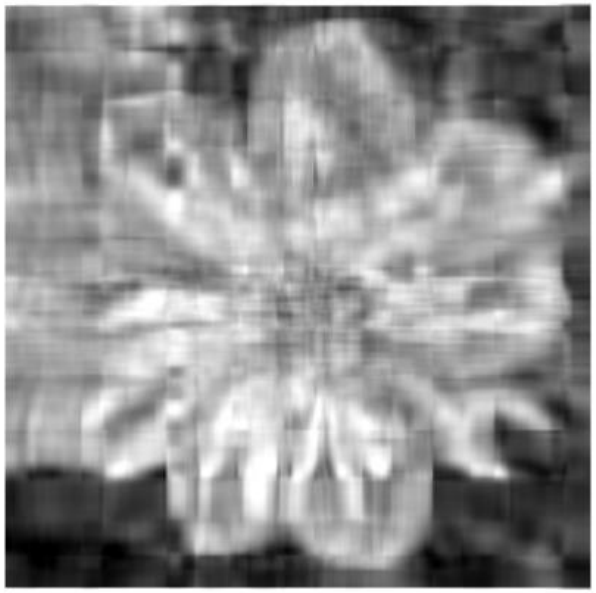
# SVD Applications

$$\underset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \underset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \underset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

We can look at **Σ** to see that the first column has a large effect

while the second column has a much smaller effect in this example

# SVD Applications



- For this image, using **only the first 10** of 300 principal components produces a recognizable reconstruction
- So, SVD can be used for image compression

# Principal Component Analysis

- Remember, columns of ***U*** are the *Principal Components* of the data: the major patterns that can be added to produce the columns of the original matrix.
- One use of this is to construct a matrix where each column is a separate data sample.
- Run SVD on that matrix, and look at the first few columns of ***U*** to see patterns that are common among the columns.
- This is called *Principal Component Analysis* (or PCA) of the data samples.

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \underset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

# Principal Component Analysis

- Often, raw data samples have a lot of redundancy and patterns.
- PCA can allow you to represent data samples as weights on the principal components, rather than using the original raw form of the data.
- This minimal representation makes machine learning and other algorithms much more efficient.

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \underset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

# Principal Component Analysis (PCA)

Slide credit: Juan Carlos Niebles

# Covariance

- Variance and Covariance are a measure of the “spread” of a set of points around their center of mass (mean).
- Variance – measure of the deviation from the mean for points in one dimension e.g. heights.
- Covariance as a measure of how much each of the dimensions vary from the mean with respect to each other.
- Covariance is measured between 2 dimensions to see if there is a relationship between the 2 dimensions e.g. number of hours studied & marks obtained.
- The covariance between one dimension and itself is the variance.

# Covariance

- So, if you had a 3-dimensional data set (x,y,z), then you could measure the covariance between the x and y dimensions, the y and z dimensions, and the x and z dimensions. Measuring the covariance between x and x , or y and y , or z and z would give you the variance of the x , y and z dimensions respectively.

$$\text{covariance}(X,Y) = \frac{\sum_{i=1}^{n} (\overline{X_i} - X)(\overline{Y_i} - Y)}{(n-1)}$$

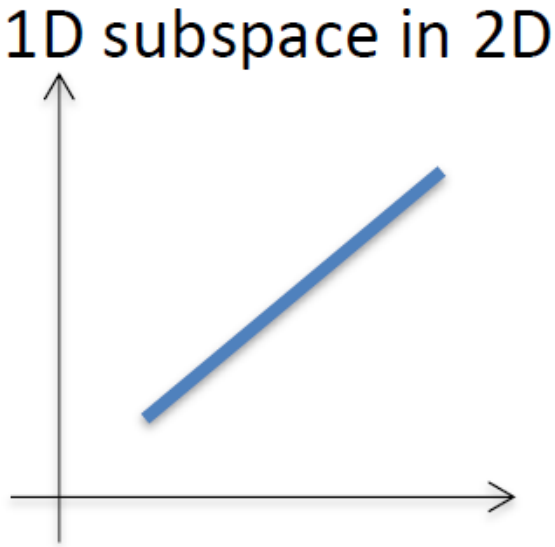
# Covariance Matrix

$$\mathrm{K}_{X_i X_j} = \operatorname{cov}[X_i, X_j] = \operatorname{E}[(X_i - \operatorname{E}[X_i])(X_j - \operatorname{E}[X_j])]$$

$$\mathrm{K}_{\mathbf{XX}} = \begin{bmatrix} \operatorname{E}[(X_1 - \operatorname{E}[X_1])(X_1 - \operatorname{E}[X_1])] & \operatorname{E}[(X_1 - \operatorname{E}[X_1])(X_2 - \operatorname{E}[X_2])] & \cdots & \operatorname{E}[(X_1 - \operatorname{E}[X_1])(X_n - \operatorname{E}[X_n])] \\ \operatorname{E}[(X_2 - \operatorname{E}[X_2])(X_1 - \operatorname{E}[X_1])] & \operatorname{E}[(X_2 - \operatorname{E}[X_2])(X_2 - \operatorname{E}[X_2])] & \cdots & \operatorname{E}[(X_2 - \operatorname{E}[X_2])(X_n - \operatorname{E}[X_n])] \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{E}[(X_n - \operatorname{E}[X_n])(X_1 - \operatorname{E}[X_1])] & \operatorname{E}[(X_n - \operatorname{E}[X_n])(X_2 - \operatorname{E}[X_2])] & \cdots & \operatorname{E}[(X_n - \operatorname{E}[X_n])(X_n - \operatorname{E}[X_n])] \end{bmatrix}$$

# Geometric interpretation of PCA

- Let's say we have a set of 2D data points x. But we see that all the points lie on a line in 2D.
- So, 2 dimensions are redundant to express the data. We can express all the points with just one dimension



# PCA: Principle Component Analysis

- Given a set of points, how do we know if they can be compressed like in the previous example?
  - The answer is to look into the correlation between the points.
  - The tool for doing this is called PCA.

# PCA: Principle Component Analysis

- Let us consider the data matrix, X.

$$X = \begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix}$$

- The sample mean is as follows

$$\mu = \frac{1}{n}\sum_i x_i = \frac{1}{n}\begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix}\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \frac{1}{n}X1$$

# PCA: Principle Component Analysis

- Center the data by subtracting the mean to each column of X.
- The centered data matrix is

$$X_c = \begin{bmatrix} | & & | \\ x_1 & \cdots & x_n \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ \mu & \cdots & \mu \\ | & & | \end{bmatrix}$$

$$= X - \mu 1^T = X - \frac{1}{n} X 1 1^T = X\left(I - \frac{1}{n} 1 1^T\right)$$

# PCA: Principle Component Analysis

The sample covariance matrix is

$$\Sigma = \frac{1}{n}\sum_i (x_i - \mu)(x_i - \mu)^T = \frac{1}{n}\sum_i x_i^c (x_i^c)^T$$

where $x_i^c$ is the $i^{th}$ column of $X_c$

This can be written as

$$\Sigma = \frac{1}{n}\begin{bmatrix} | & & | \\ x_1^c & \cdots & x_n^c \\ | & & | \end{bmatrix}\begin{bmatrix} - & x_1^c & - \\ & \vdots & \\ - & x_n^c & - \end{bmatrix} = \frac{1}{n} X_c X_c^T$$

# PCA: Principle Component Analysis

The matrix

$$X_c^T = \begin{bmatrix} - & x_1^c & - \\ & \vdots & \\ - & x_n^c & - \end{bmatrix}$$

is real (n x d). Assuming n>d it has SVD decomposition

$$X_c^T = \mathrm{M}\Pi\mathrm{N}^T$$

$$\mathrm{M}^T\mathrm{M} = I \qquad \mathrm{N}^T\mathrm{N} = I$$

and

$$\Sigma = \frac{1}{n} X_c X_c^T = \frac{1}{n} \mathrm{N}\Pi\mathrm{M}^T\mathrm{M}\Pi\mathrm{N}^T = \frac{1}{n} \mathrm{N}\Pi^2\mathrm{N}^T$$

# PCA: Principle Component Analysis

$$\Sigma = \mathrm{N}\left(\frac{1}{n}\Pi^2\right)\mathrm{N}^T$$

Note that N is (d x d) and orthonormal, and $\Pi^2$ is diagonal. This is just the eigenvalue decomposition of Σ

It follows that

- The eigenvectors of Σ are the columns of N
- The eigenvalues of Σ are

$$\lambda_i = \frac{1}{n}\pi_i^2$$

This gives an alternative algorithm for PCA

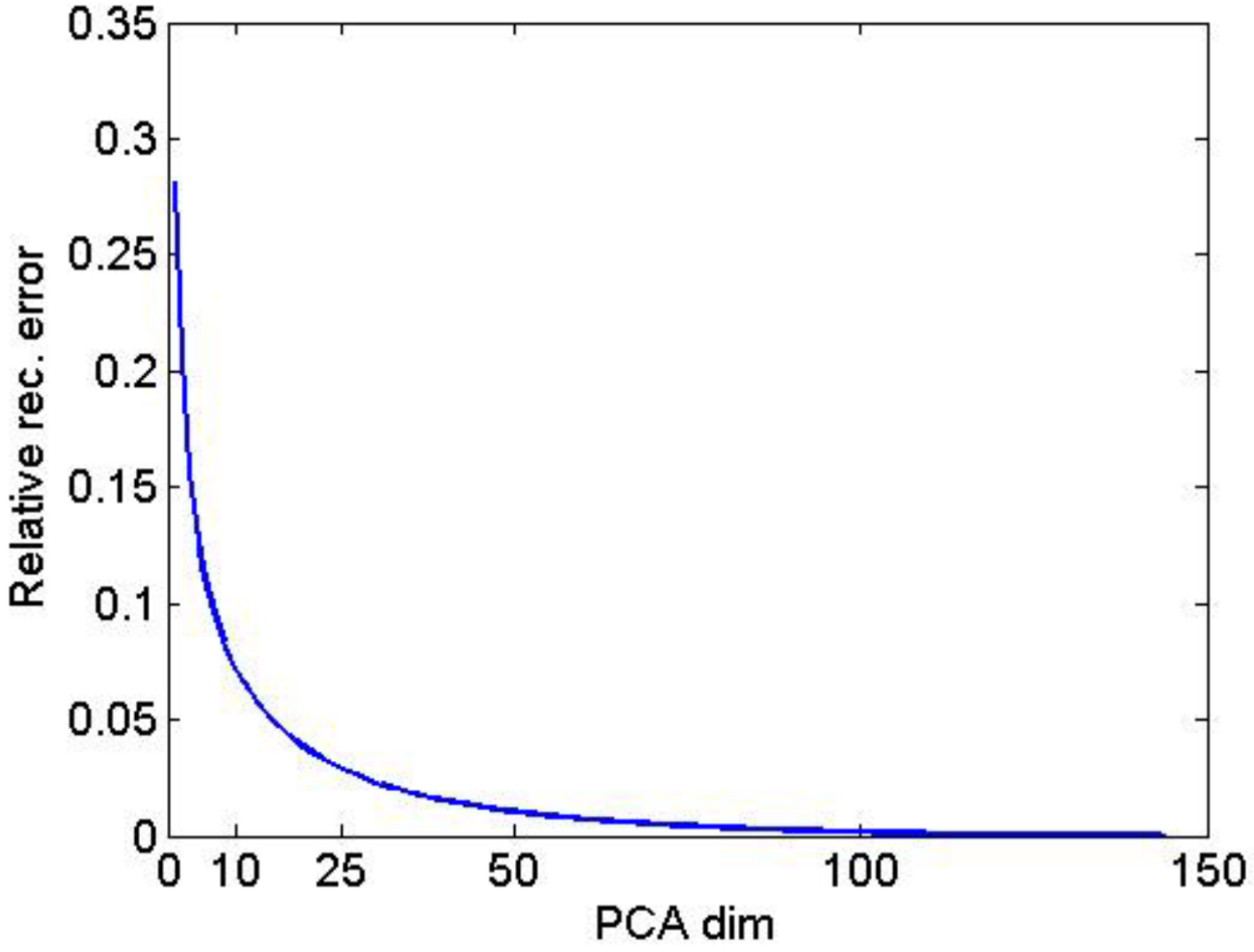
# Image Compression Using PCA

Slide credit: Juan Carlos Niebles

# Original Image

- Divide the original 372x492 image into patches:
  - Each patch is an instance that contains 12x12 pixels on a grid
- View each as a 144-D vector

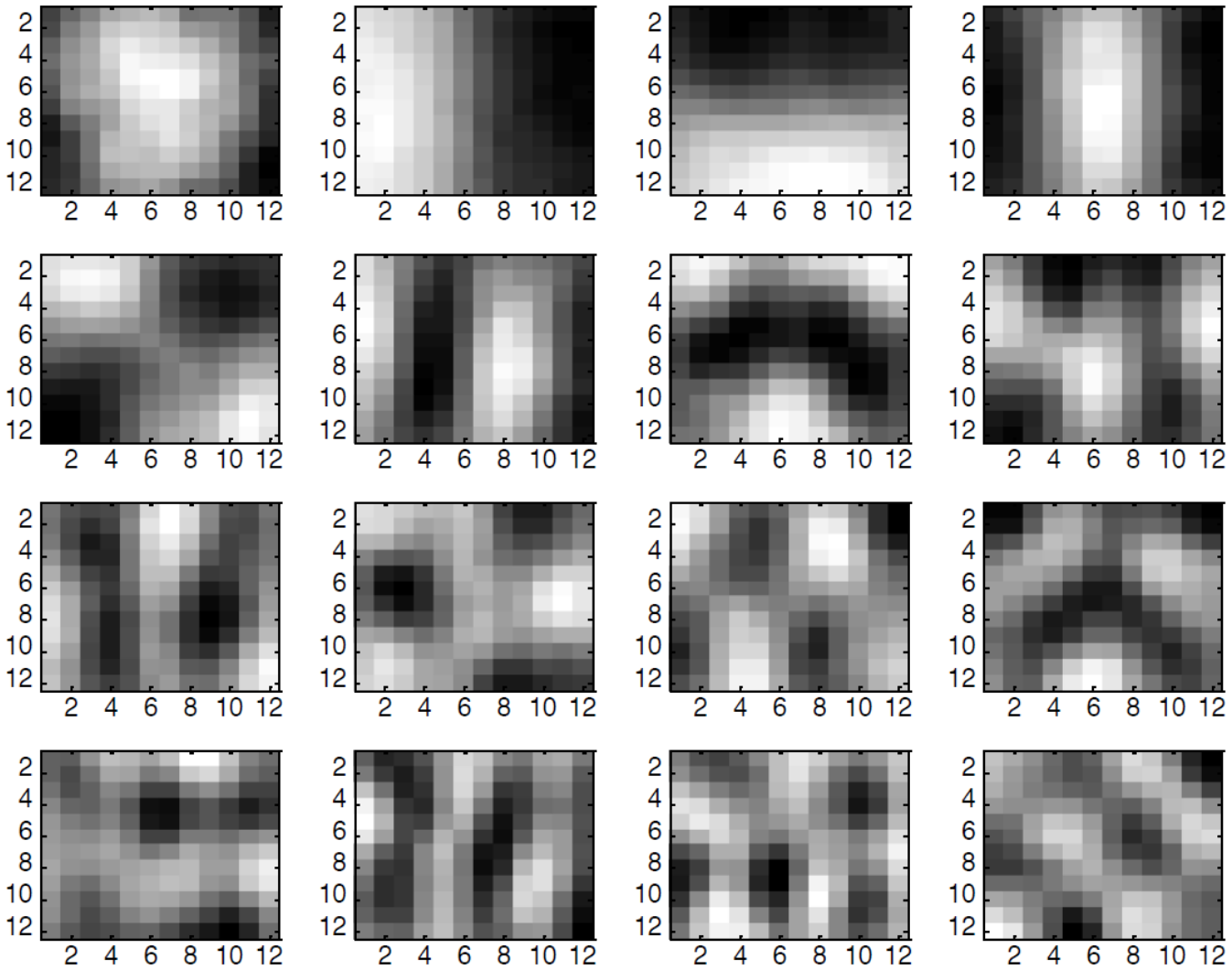
# $L_2$ error and PCA dimension



# PCA compression: 144D → 60D

# PCA compression: 144D → 16D

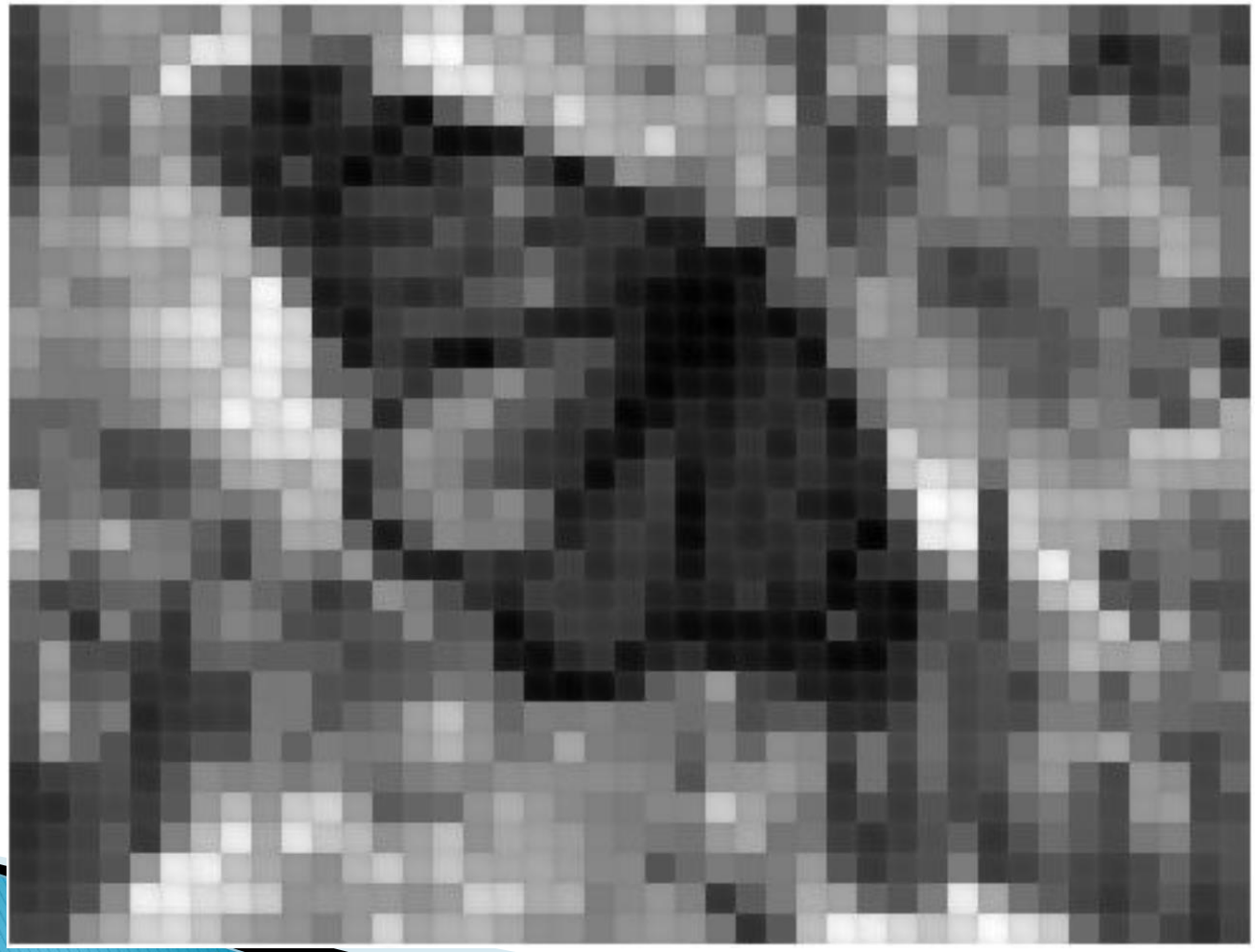
# 16 most important eigenvectors



# PCA compression: 144D → 6D

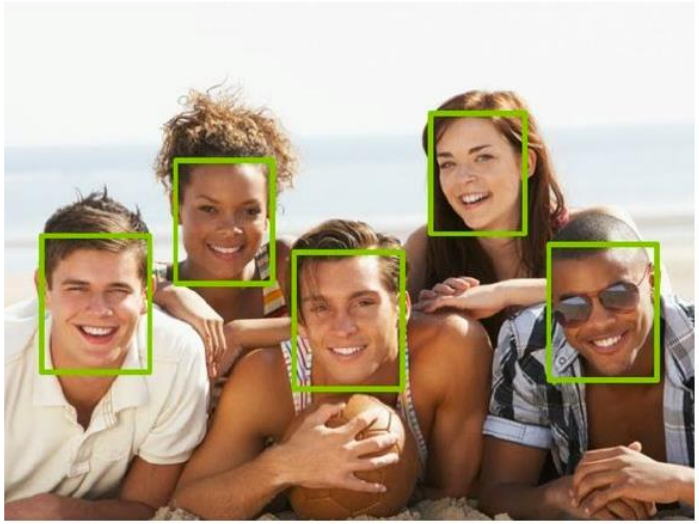
# PCA compression: 144D → 3D

# PCA compression: 144D → 1D

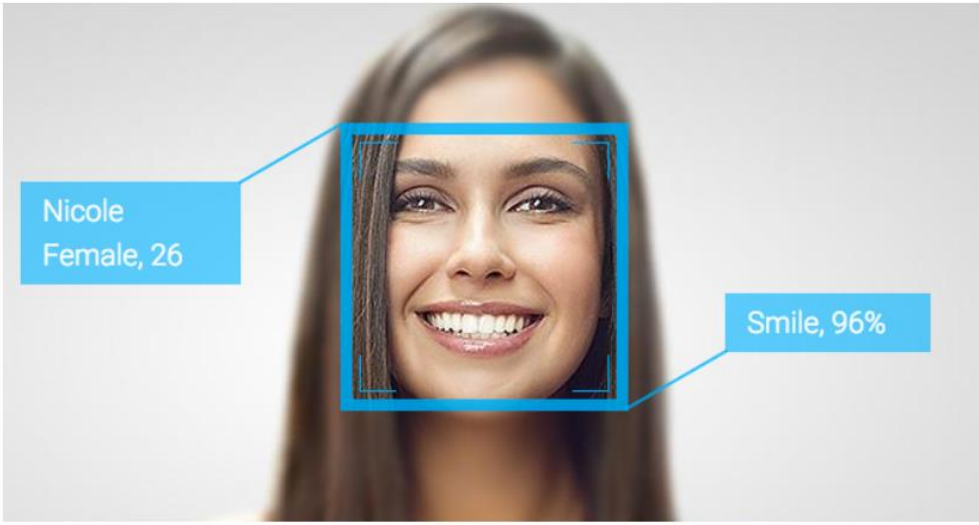


# Face Recognition

# Detection versus Recognition



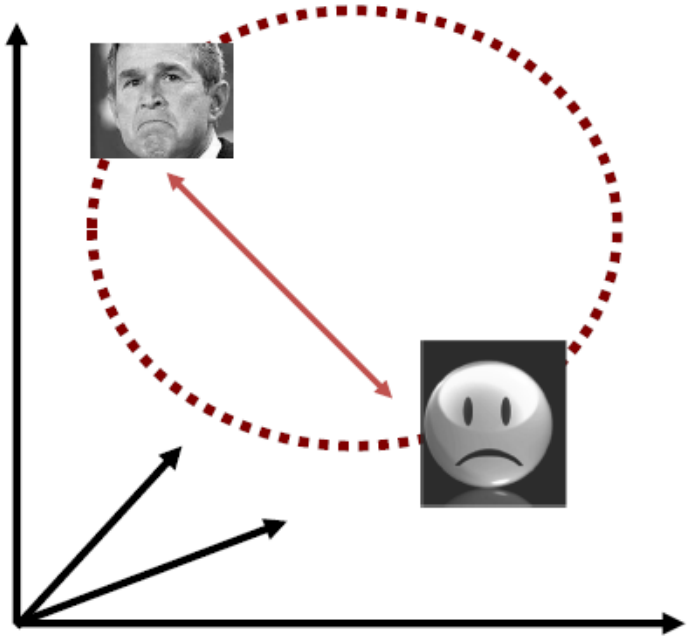
Detection finds the faces in images




Recognition recognizes WHO the person is

Slide credit: Juan Carlos Niebles

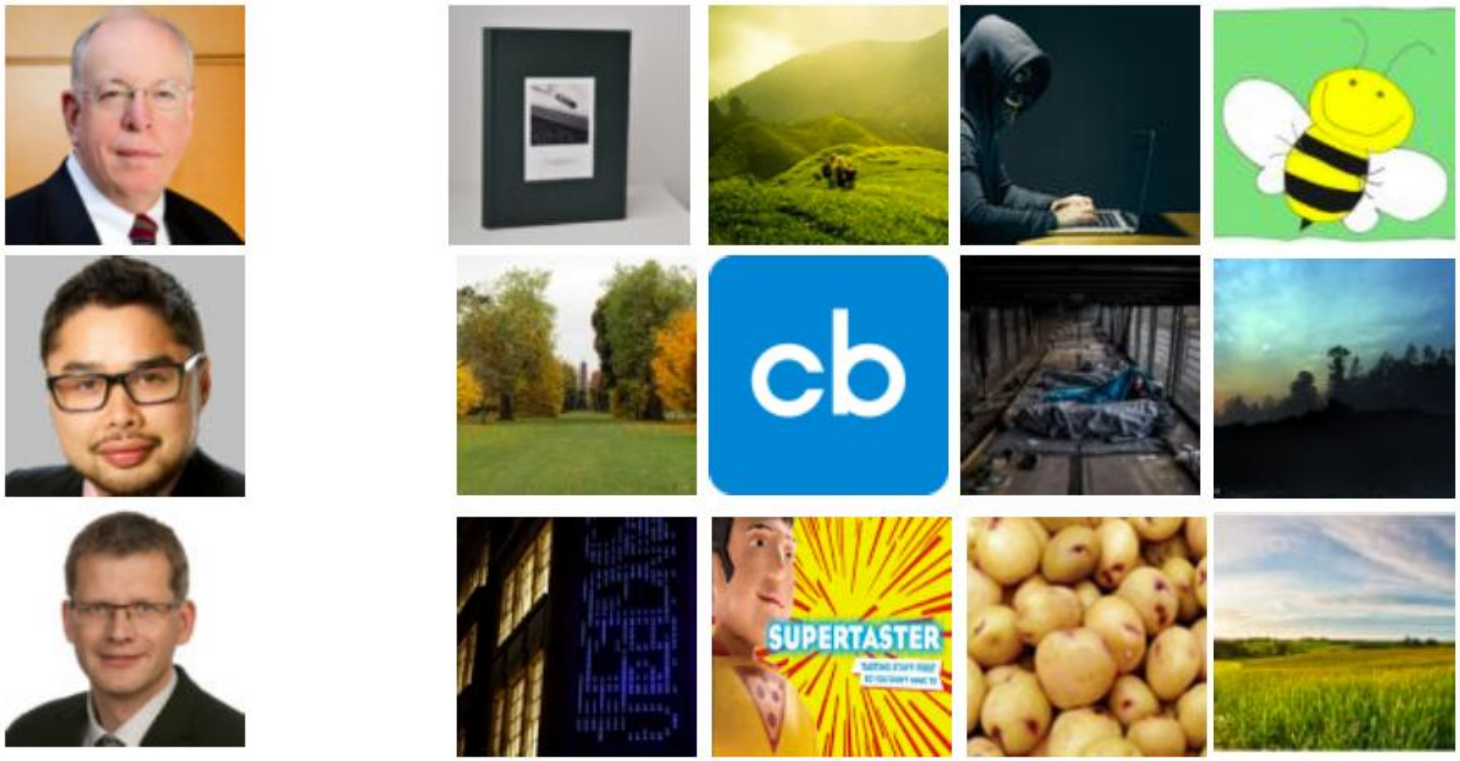
# Space of Faces



- An image is a point in a high dimensional space
  - If represented in grayscale intensity, an N x M image is a point in $R^{NM}$
  - E.g. 100x100 image = 10,000 dim

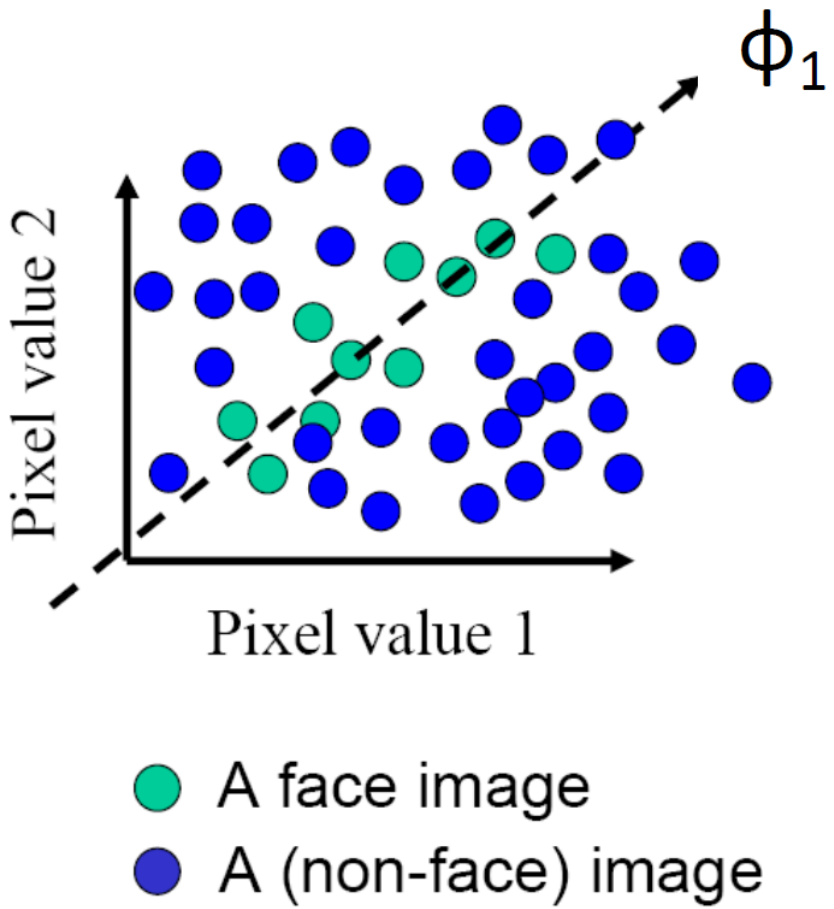
Slide credit: Chuck Dyer, Steve Seitz, Nishino

# 100x100 images can contain many things other than faces!




Slide credit: Juan Carlos Niebles

# Space of Faces




- An image is a point in a high dimensional space
  - If represented in grayscale intensity, an N x M image is a point in $R^{NM}$
  - E.g. 100x100 image = 10,000 dim
- However, relatively few high dimensional vectors correspond to valid face images
- We want to effectively model the subspace of face images

Slide credit: Chuck Dyer, Steve Seitz, Nishino

# How do we model the subspace of face images?

- Use PCA for estimating sub-space!!!! (eigenfaces)

# Eigenfaces

- Assume that most face images lie on a low-dimensional subspace determined by the first k (k<<d) directions of maximum variance.
- Use PCA to determine the vectors or “eigenfaces” that span that subspace.
- Represent all face images in the dataset as linear combinations of eigenfaces.

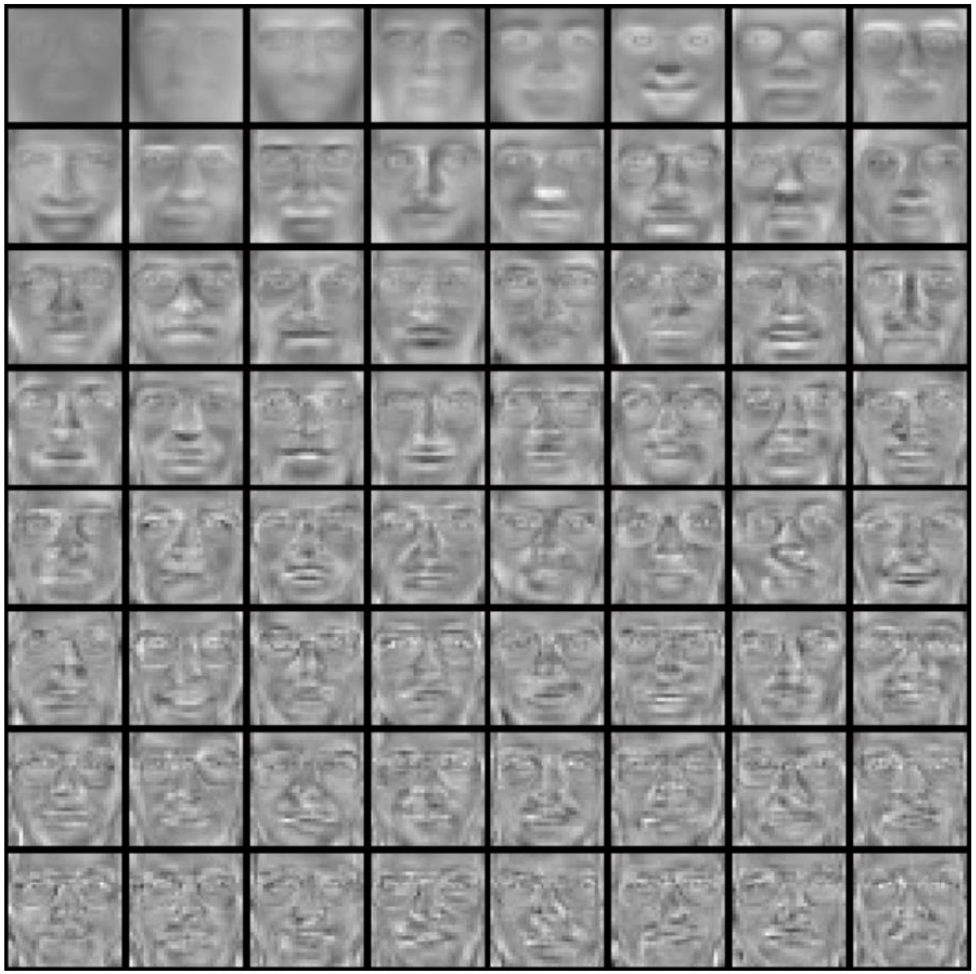
M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991

# Training images: $x_1,\ldots,x_N$

Slide credit: Juan Carlos Niebles

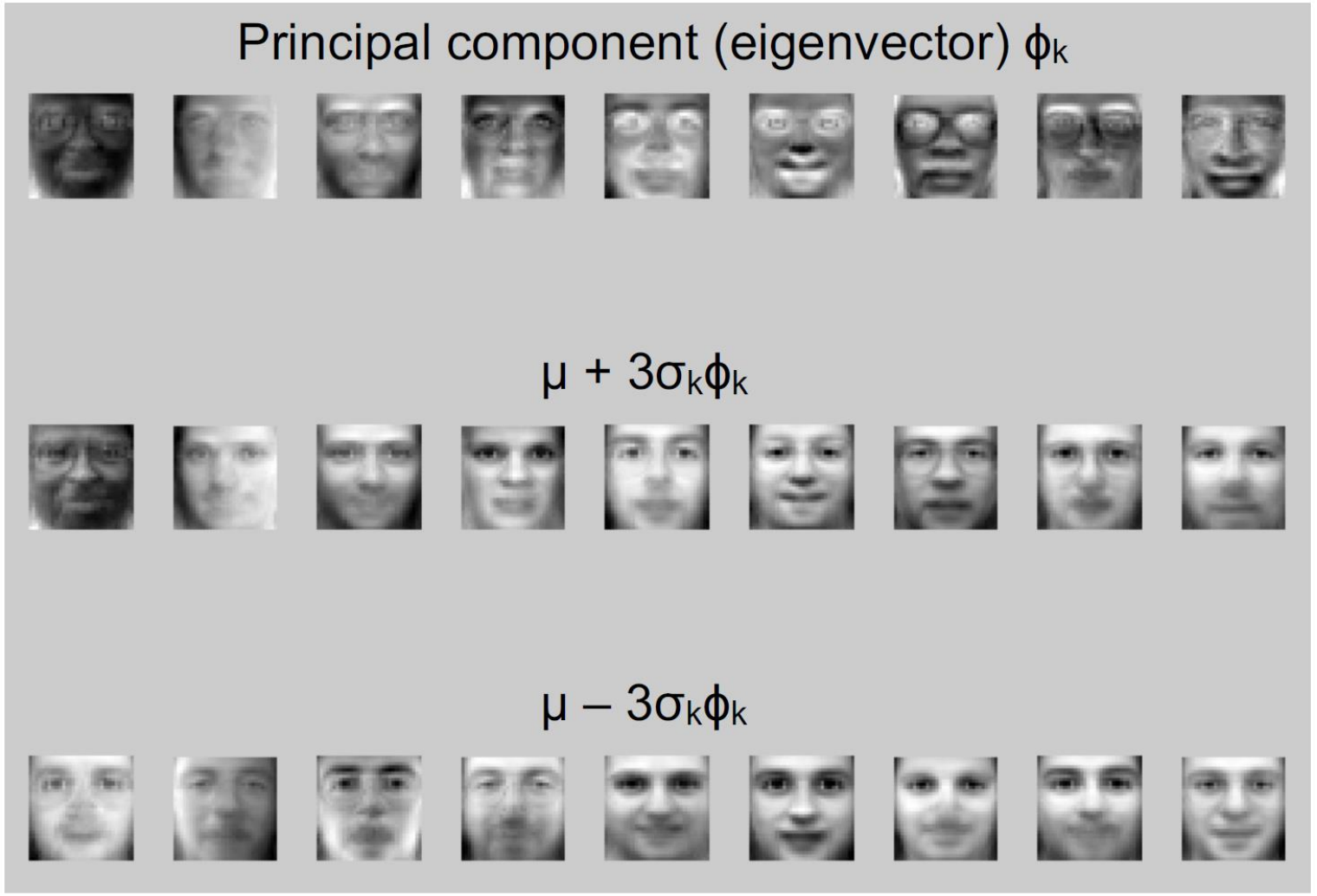
# Top eigenvectors: $\phi_1,\ldots,\phi_k$

Mean: μ



Slide credit: Juan Carlos Niebles

# Visualization of eigenfaces




Slide credit: Juan Carlos Niebles

# Eigenface algorithm

## Training

1. Align training images $x_1, x_2, \ldots, x_N$

Note that each image is formulated into a long vector!

2. Compute average face $\mu = \frac{1}{N}\sum x_i$

3. Compute the difference image (the centered data matrix)

$$X_c = \begin{bmatrix} | & & | \\ x_1 & \ldots & x_n \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ \mu & \ldots & \mu \\ | & & | \end{bmatrix}$$

$$= X - \mu 1^T = X - \frac{1}{n} X 1 1^T = X\left(I - \frac{1}{n} 1 1^T\right)$$

Slide credit: Juan Carlos Niebles

# Eigenface algorithm

4. Compute the covariance matrix

$$\Sigma = \frac{1}{n}\begin{bmatrix} | & & | \\ x_1^c & \dots & x_n^c \\ | & & | \end{bmatrix}\begin{bmatrix} - & x_1^c & - \\ & \vdots & \\ - & x_n^c & - \end{bmatrix} = \frac{1}{n} X_c X_c^T$$

5. Compute the eigenvectors of the covariance matrix Σ
6. Compute each training image $x_i$ 's projections as

$$x_i \rightarrow \left(x_i^c \cdot \phi_1, x_i^c \cdot \phi_2, \dots, x_i^c \cdot \phi_K\right) \equiv \left(a_1, a_2, \dots, a_K\right)$$

7. Visualize the estimated training face $x_i$

$$x_i \approx \mu + a_1\phi_1 + a_2\phi_2 + \dots + a_K\phi_K$$

Slide credit: Juan Carlos Niebles

# Eigenface algorithm


6. Compute each training image $x_i$ 's projections as

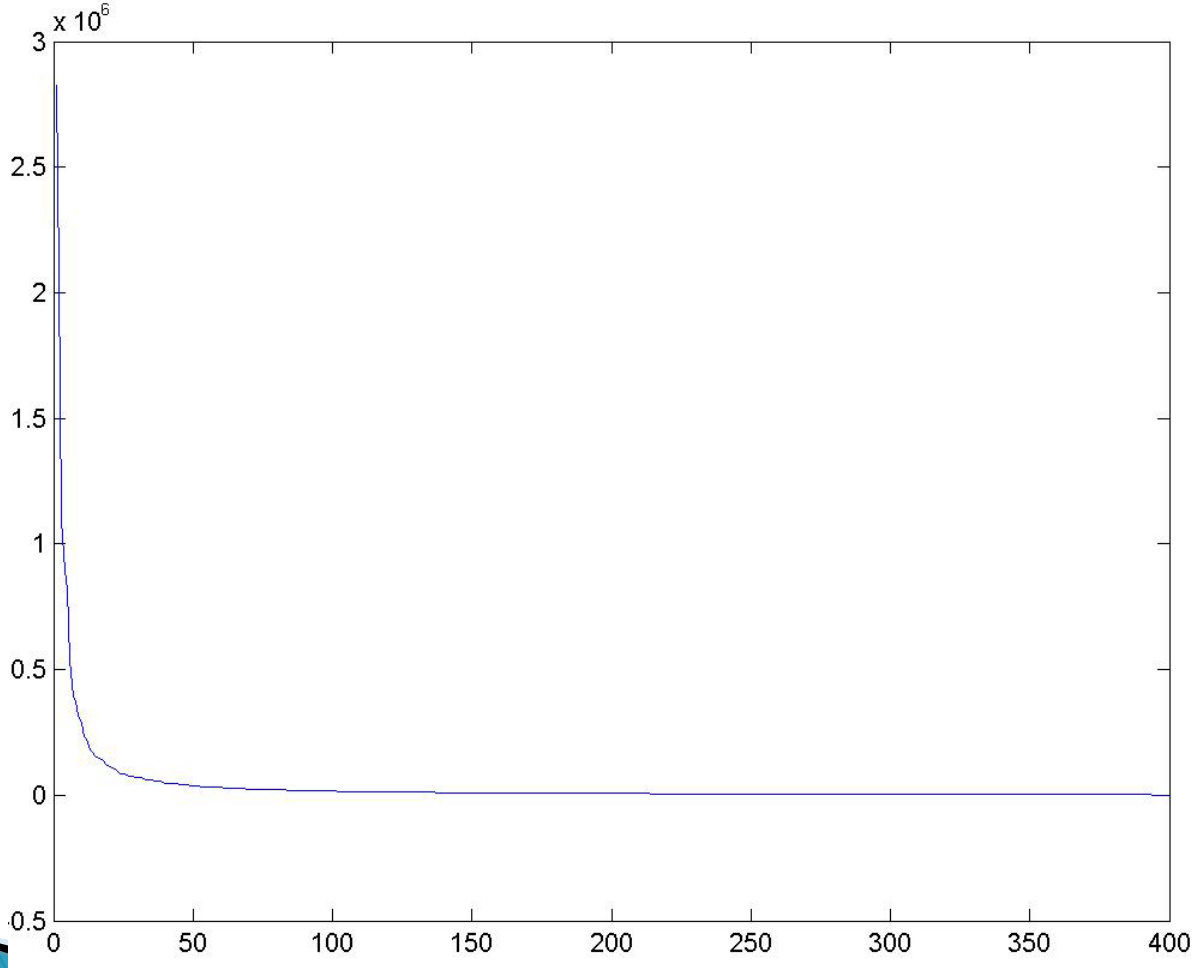
$$x_i \rightarrow \left(x_i^c \cdot \phi_1, x_i^c \cdot \phi_2, \ldots, x_i^c \cdot \phi_K\right) \equiv \left(a_1, a_2, \ldots, a_K\right)$$

7. Visualize the reconstructed training face $x_i$

$$x_i \approx \mu + a_1\phi_1 + a_2\phi_2 + \ldots + a_K\phi_K$$

Slide credit: Juan Carlos Niebles

# Eigenvalues (variance along eigenvectors)




Slide credit: Juan Carlos Niebles

# Reconstruction and Errors

K = 4

K = 200

K = 400

Slide credit: Juan Carlos Niebles

# Eigenface algorithm

Testing

1. Take query image t
2. Project into eigenface space and compute projection

$$t \rightarrow \left((t-\mu)\cdot\phi_1, (t-\mu)\cdot\phi_2, \ldots, (t-\mu)\cdot\phi_K\right) \equiv \left(w_1, w_2, \ldots, w_K\right)$$

3. Compare projection w with all N training projections
   - Simple comparison metric: Euclidean
   - Simple decision: K-Nearest Neighbor

   (note: this "K" refers to the k-NN algorithm, is different from the previous K's referring to the # of principal components)

Slide credit: Juan Carlos Niebles