Face Recognition

CS 663

Importance of face recognition

- The most common way for humans to recognize each other

- Study of the process of face recognition has applications in

  (1) security/surveillance/authentication,

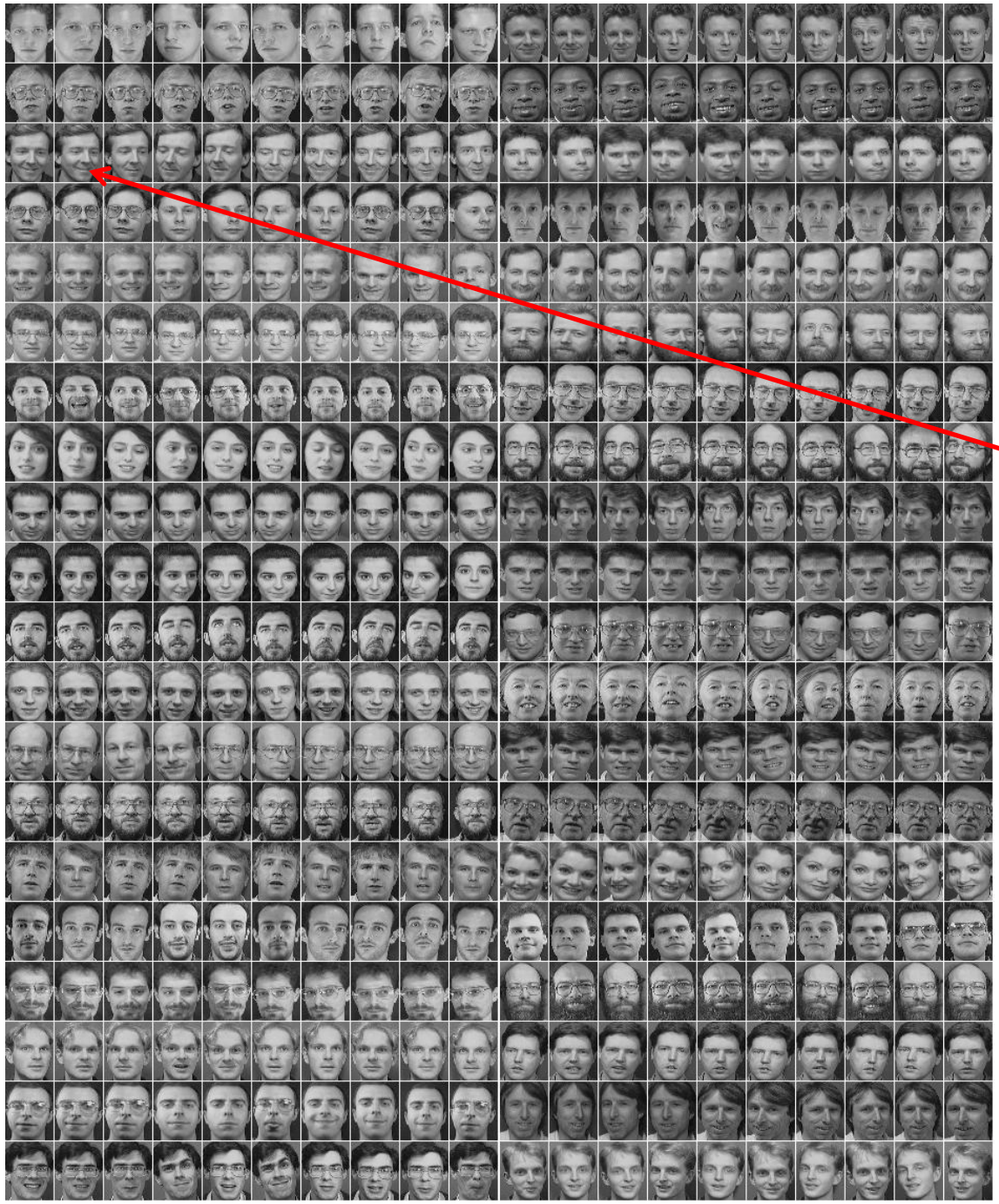
  (2) understanding of visual psychology,

  (3) automated tagging on Facebook ☺

Face recognition: problem statement

- Given a database of face images of people, and a new test image, answer the following question:

  “The test image contains the face of which individual from the database?”



A naïve method

- Compare the test image with each database image in terms of SSD (sum of squared differences).
- Choose the closest match (i.e., in terms of squared difference)!

$$\|I - J\|^2 = \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} (I_{ij} - J_{ij})^2$$

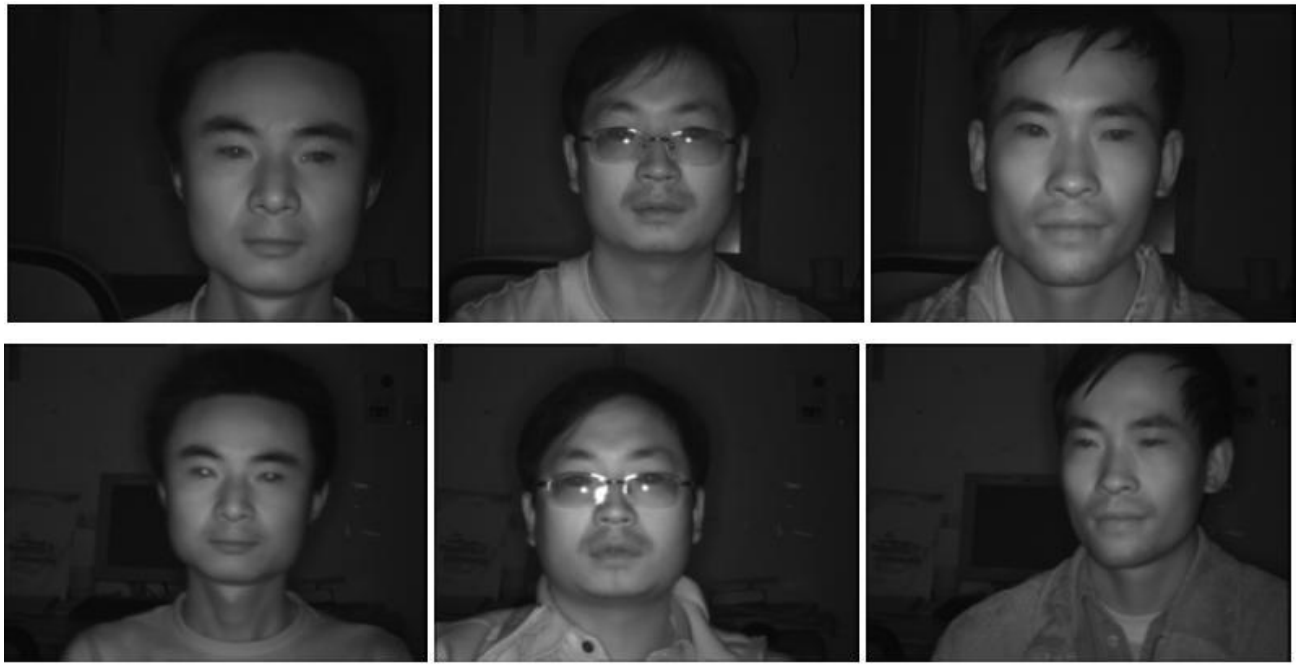
This method is fraught with problems!

Challenges in face recognition: detection

Where is (are) the face(s) in the picture?

http://iomniscient.com/newsletters/HTML/EN/image/FR.gif

Challenges in face recognition: pose variation



http://www4.comp.polyu.edu.hk/~biometrics/polyudb_face.files/image010.jpg

Images are 2D, face is a 3D object. There will be out of plane rotations (profile versus front), change in apparent size due to change in distance from camera.

Challenges in face recognition: illumination variation

http://www1.uwe.ac.uk/et/images/raw_v_Variation_2.jpg

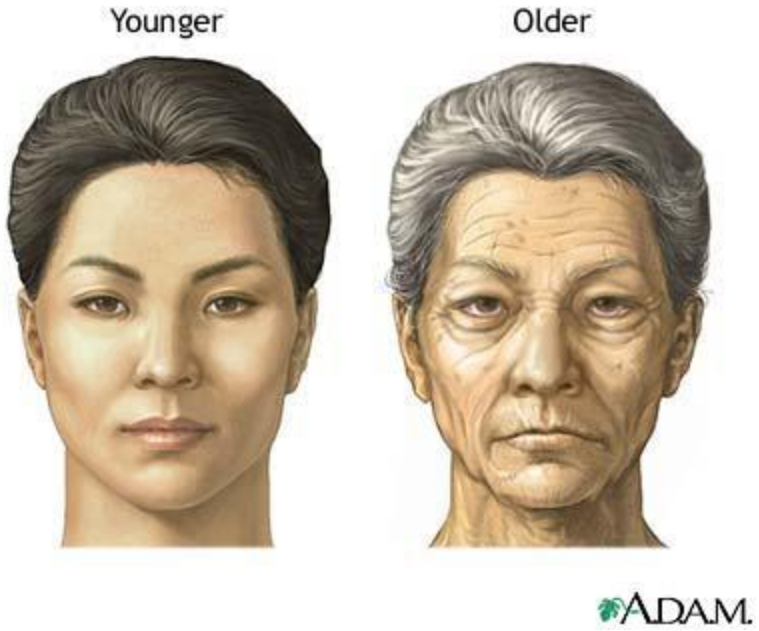
- Multiple light sources
- Change in direction of lighting sources
- Change in intensity/color/type of light source
- Shadows, specular reflections

Challenges in face recognition: expression variation

Varied expressions: smile, anger, frown, sadness, surprise, closed eyes, confusion, etc.

http://www.idealtest.org/userfiles/image/3d-facev1-fig.3.jpg

Challenges in face recognition: age variation



http://www.healthcentral.com//common/images/8/8691_13380_5.jpg

Challenges in face recognition: variation of facial accessories

http://gps-tsc.upc.es/GTAV/ResearchAreas/UPCFaceDatabase/Imatges/FaceOcclusionID1.jpg

Spectacles, beard and moustache, scarves, ear-rings, change of hair-styles, etc.

Challenges in face recognition: caricatures/paintings

http://www.embedded-vision.com/sites/default/files/news/ff_caricature4_f.jpg?1313157764

Challenges in face recognition: blur/noise/scanner artifacts

http://people.csail.mit.edu/celiu/FaceHallucination/soccer.jpg

And more!

- Even ignoring changes of pose, illumination, expression, etc., we tend to look different at different periods of time! Recognition still remains a challenge!

Face Recognition system: block diagram

(1) Collect database of face images of people. Record one or multiple images per person (called **"gallery image(s)"** of the person)

(2) Normalize the images: (manually) crop out the face from the overall image background, correct for pose variation or lighting changes

(3) Extract relevant features from the normalized face image (more on this later!)

(4) Label the features extracted from the image with that person's identity, and store in a database

TRAINING PHASE!

Face Recognition system: block diagram

(1) Collect an image of the person whose identity is to be determined* – called the **probe image**. In most cases the time gap between acquisition of probe and gallery images is significant (months/years)

(2) Normalize the probe image: (manually) crop out the face from the overall image background, correct for pose variation or lighting changes

(3) Extract the same features from the normalized face image (more on this later!) as from the gallery images

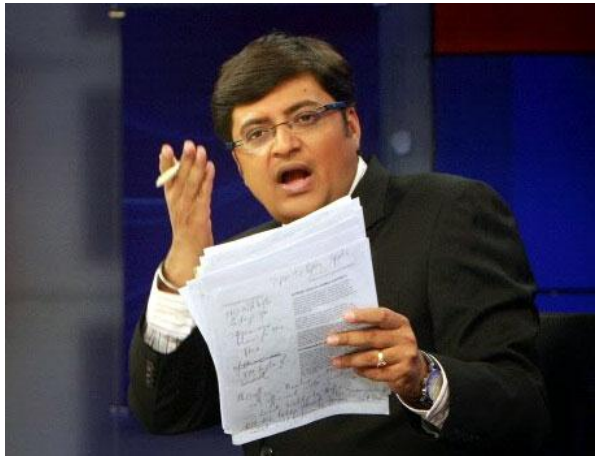
(4) Find the gallery image whose features most closely match (**nearest neighbor search**) those of the probe image. That tells you the identity.

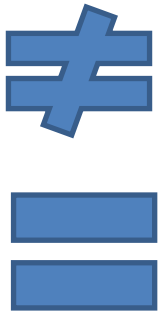
TESTING PHASE!

*For now, we assume that the person whose identity was to be determined, has images recorded in the gallery database

Problems related to face recognition

- **Face verification:** given two face images, determine whether they belong to the same individual (without concern for the individual's identity)..





Problems related to face recognition

- Ethnicity/gender identification from face images

- Is the given face image a photo or is it a painting/caricature?

What features to extract for face recognition? Many methods

- (1) Detect visible features: eyes, nose, mouth, high-points of the cheek, chin, eyebrows, etc. Not very robust!
- (2) **Statistical holistic approaches:** extract features using statistical method. These features may not necessarily have a physical interpretation (in terms of, say, eyes, nose, mouth, etc.)

Eigenfaces!

- We focus on the latter group of techniques in these lectures.
- One such technique for face recognition – called **Eigenfaces** – uses a statistical method called **Principal Components Analysis** (PCA).

Principal Components Analysis (PCA)

- Consider *N* vectors (or points) $\mathbf{x_i}$, each containing *d* elements, each represented as a **column vector**.
- We say: $\forall i, \mathbf{x_i} \in R^d$. *d* could be very large – like 250,000 or more.
- Our aim is to extract some *k* features from each $\mathbf{x_i}$, $k << d$.
- Effectively we are projecting the original vectors from a *d*-dimensional space to a *k*-dimensional space. This is called **dimensionality reduction**. PCA is **one method** of dimensionality reduction.

PCA

- How do we pick the '*k*' appropriate features?

- We look into a notion of "compressibility" – how much can the data be compressed, allowing for some small errors.

PCA: Algorithm

1. Compute the mean of the given points:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i, \mathbf{x}_i \in R^d, \bar{\mathbf{x}} \in R^d$$

2. Deduct the mean from each point:

$$\bar{\mathbf{x}}_\mathbf{i} = \mathbf{x}_\mathbf{i} - \bar{\mathbf{x}}$$

3. Compute the covariance matrix of these mean-deducted points:

$$\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N}\bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_\mathbf{i} - \bar{\mathbf{x}})(\mathbf{x}_\mathbf{i} - \bar{\mathbf{x}})^T, \textit{Note}: \mathbf{C} \in R^{d\times d}$$

$\textit{Note}: \mathbf{C}$ is a symmetric matrix, and it is positive-semidefinite

PCA: algorithm

4. Find the eigenvectors of **C**:

$$\mathbf{CV} = \mathbf{V\Lambda}, \mathbf{V} \in R^{d \times d}, \mathbf{\Lambda} \in R^{d \times d}$$

$\mathbf{V}$ – matrix of eigenvectors (each column is an eigenvector),

$\mathbf{\Lambda}$ – diagonal matrix of eigenvalues

Note: $\mathbf{V}$ is an orthonormal matrix (i.e. $\mathbf{VV}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$), as $\mathbf{C}$ is a covariance matrix and hence it is symmetric.

Note: $\mathbf{\Lambda}$ contains non-negative values on the diagonal (eigen-values)

5. Extract the *k* eigenvectors corresponding to the *k* largest eigenvalues. This is called the extracted **eigenspace:**

$$\hat{\mathbf{V}}_\mathbf{k} = \mathbf{V}(:,1:k)$$

There is an implicit assumption here that the first *k* indices indeed correspond to the *k* largest eigenvalues. If that is not true, you would need to pick the appropriate indices.

PCA: algorithm

6. Project each point onto the eigenspace, giving a vector of *k* **eigen-coefficients** for that point.

$$\boldsymbol{\alpha}_{ik} = \hat{\mathbf{V}}_{\mathbf{k}}^{\mathbf{T}} \bar{\mathbf{x}}_{\mathbf{i}}, \boldsymbol{\alpha}_{ik} \in R^k; \boldsymbol{\alpha}_i = \mathbf{V}^T \bar{x}_i, \boldsymbol{\alpha}_i \in R^d$$

$$\text{As } \mathbf{V} \text{ is orthonormal, we have}$$

$$\bar{\mathbf{x}}_{\mathbf{i}} = \mathbf{V}\boldsymbol{\alpha}_{\mathbf{i}} = \mathbf{V}(:,1)\boldsymbol{\alpha}_i(1) + \mathbf{V}(:,2)\boldsymbol{\alpha}_i(2) + \ldots + \mathbf{V}(:,d)\boldsymbol{\alpha}_i(d)$$

$$\approx \hat{\mathbf{V}}_{\mathbf{k}}\boldsymbol{\alpha}_{ik} = \hat{\mathbf{V}}_{\mathbf{k}}(:,1)\boldsymbol{\alpha}_{ik}(1) + \hat{\mathbf{V}}_{\mathbf{k}}(:,2)\boldsymbol{\alpha}_{ik}(2) + \ldots + \hat{\mathbf{V}}_{\mathbf{k}}(:,d)\boldsymbol{\alpha}_{ik}(k)$$

We are representing each face as a linear combination of the *k* eigenvectors corresponding to the *k* largest eigenvalues. The coefficients of the linear combination are the eigen-coefficients.

Note that $\boldsymbol{\alpha}_{\mathbf{ik}}$ is a vector of the eigencoefficients of the *i*-th sample point, and it has *k* elements. The j-th element of this vector is denoted as $\boldsymbol{\alpha}_{\mathbf{ik}}$ (j).

PCA and Face Recognition: Eigen-faces

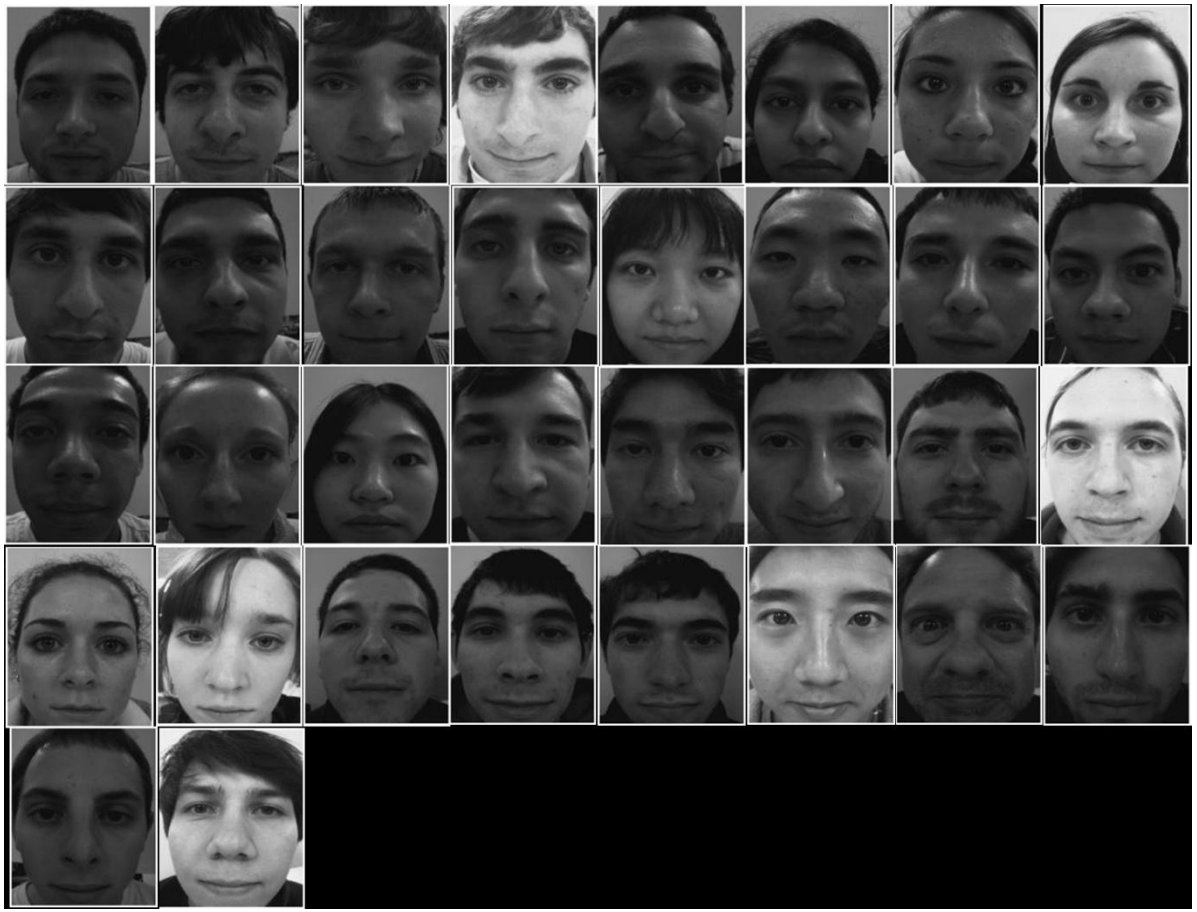
- Consider a database of cropped, frontal face images (which we will assume are aligned and under the same illumination). These are the **gallery images**.

- We will reshape each such image (a 2D array of size *H x W* after cropping) to form a column vector of $d = HW$ elements. Each image will be a vector $\mathbf{x}_i$, as per the notation on the previous two slides.

- And then carry out the six steps mentioned before.

- The eigenvectors that we get in this case are called **eigenfaces**. Each eigenvector has *d* elements. If you reshape those eigenvectors to form images of size *H x W*, those images **look like (filtered!) faces.**

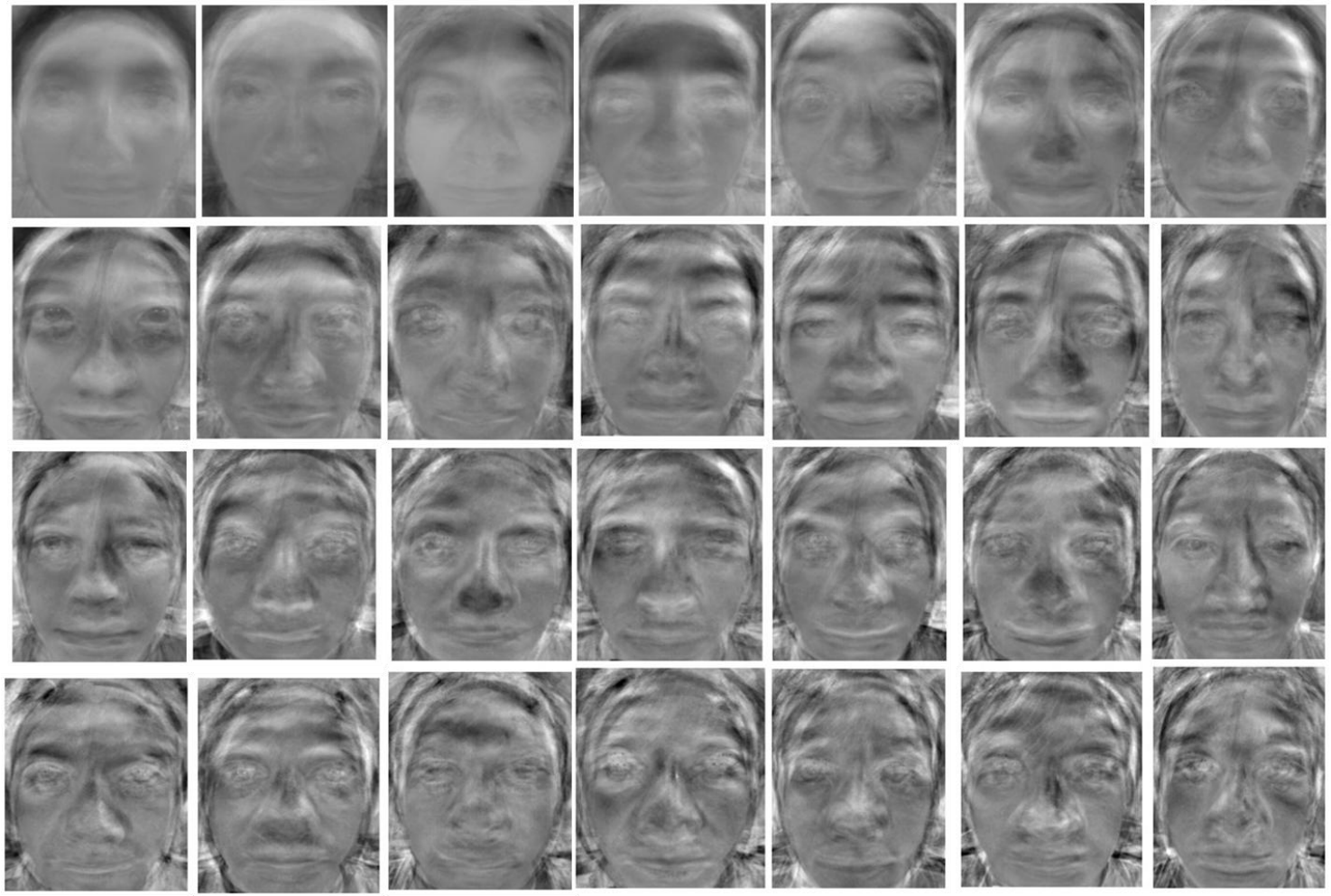
Example 1



A face database

http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

Top 25 Eigen-faces for this database!



http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

PCA and Face recognition: Eigenfaces

- For each gallery image, you compute the eigen-coefficients. You then store the eigen-coefficients and the identity of the person in a database.
- You also store $\hat{\mathbf{V}}_k, \bar{\mathbf{x}}$ in the database.
- During the testing phase, you are given a probe image (say) $\mathbf{z}_p$ in the form of a column vector of *HW* elements.
- You deduct the mean image from $\mathbf{z}_p$:

$$\bar{\mathbf{z}}_p = \mathbf{z}_p - \bar{\mathbf{x}}$$

PCA and Face recognition: Eigenfaces

- You then project the mean-deducted face image onto the eigen-space:

$$\boldsymbol{\alpha}_p = \hat{\mathbf{V}}_k^T \bar{\mathbf{z}}_p$$

Eigen-coefficients of the probe image $\mathbf{z}_p$.

- Now, compare $\boldsymbol{\alpha}_p$ with all the $\boldsymbol{\alpha}_{ik}$ (eigen-coefficients of the gallery images) in the database.

- Find the closest match in terms of the squared distance between the eigen-coefficients. That gives you the identity (see next slide).

PCA and Face recognition: Eigenfaces

$$j_p = \arg\min_l \left\| \boldsymbol{\alpha}_p - \boldsymbol{\alpha}_l \right\|_2^2$$

Eigen-coefficients of the probe image $\mathbf{z}_p$.

Eigen-coefficients of the l-th gallery image $\mathbf{x}_l$.

Note: other distance measures (different from sum of squared differences) may also be employed. One example is sum of absolute differences, given as follows: $\left\| \boldsymbol{\alpha}_p - \boldsymbol{\alpha}_l \right\|_1$

Another could be normalized dot product (and this distance measure should be maximized!): $\dfrac{\boldsymbol{\alpha}_p \bullet \boldsymbol{\alpha}_l}{\left\| \boldsymbol{\alpha}_p \right\|_2 \left\| \boldsymbol{\alpha}_l \right\|_2}$

PCA and Face recognition: eigenfaces

- The eigen-face images contain more and more high frequency information as the corresponding eigen-values decrease.
- Although PCA is a technique known for a long time, it's application in face recognition was pioneered by Turk and Pentland in a classic paper in 1991.

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

http://www.cs.ucsb.edu/~mturk/

http://web.media.mit.edu/~sandy/

PCA and Face recognition: eigenfaces

- We can regard the *k* eigenfaces as key **signatures**.
- We express each face image as a **linear combination** of these eigenfaces, i.e. the average face + (say) 3 times eigenface 1 + (say) 5 times eigenface 2 + (say) -1 times eigenface 3 and so on. (note: 3,5,-1 here are the eigen-coefficients, and some of them can be negative).

$$\bar{\mathbf{x}}_\mathbf{i} = \mathbf{V}\boldsymbol{\alpha}_\mathbf{i} = \mathbf{V}(:,1)\boldsymbol{\alpha}_\mathbf{i}(1) + \mathbf{V}(:,2)\boldsymbol{\alpha}_\mathbf{i}(2) + \ldots + \mathbf{V}(:,d)\boldsymbol{\alpha}_\mathbf{i}(d)$$

$$\approx \hat{\mathbf{V}}_\mathbf{k}\boldsymbol{\alpha}_\mathbf{ik} = \hat{\mathbf{V}}_\mathbf{k}(:,1)\boldsymbol{\alpha}_\mathbf{ik}(1) + \hat{\mathbf{V}}_\mathbf{k}(:,2)\boldsymbol{\alpha}_\mathbf{ik}(2) + \ldots + \hat{\mathbf{V}}_\mathbf{k}(:,d)\boldsymbol{\alpha}_\mathbf{ik}(k)$$

One word of caution: Eigen-faces

- The algorithm described earlier is computationally infeasible for eigen-faces, as it requires storage of a d x d Covariance matrix (d – the number of image pixels - could be more than 10,000). And the computation of the eigen-vectors of such a matrix is a $O(d^3)$ operation!

- We will study a modification to this that will bring down the computational cost drastically.

Eigen-faces: reducing computational complexity.

- Consider the covariance matrix:

$$\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N}\bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i-\bar{\mathbf{x}})(\mathbf{x}_i-\bar{\mathbf{x}})^T, \textit{Note}: \mathbf{C}\in R^{d\times d}$$

$$\textit{Note}: \mathbf{C} \text{ is a symmetric matrix, and it is positive-semidefinite}$$

- It will require too much memory if *d* is large, and computing its eigenvectors will be a horrendous task!

- Consider the case when *N* is much less than *d*. This is very common in face recognition applications. The number of training images is usually much smaller than the size of the image.

Eigen-faces: reducing computational complexity.

- In such a case, the rank of **C** is at the most *N-1*. So **C** will have at the most *N-1* non-zero eigen-values.

- We can write **C** in the following way:

$$\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N} \bar{\mathbf{x}}_i \bar{\mathbf{x}}_i^T \propto \mathbf{X}\mathbf{X}^T, \text{where}$$

$$\mathbf{X} = [\bar{\mathbf{x}}_1 \mid \bar{\mathbf{x}}_2 \mid \ldots \mid \bar{\mathbf{x}}_N] \in R^{d \times N}$$

Back to Eigen-faces: reducing computational complexity.

- Consider the matrix $\mathbf{X}^T\mathbf{X}$ (size N x N) instead of $\mathbf{X}\mathbf{X}^T$ (size d x d). Its eigenvectors are of the form:

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \lambda\mathbf{w}, \mathbf{w} \in R^N$$

$$\rightarrow \mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{w}) = \lambda(\mathbf{X}\mathbf{w}) \text{ [pre-multiplying by } \mathbf{X}]$$

Xw is an eigenvector of $\mathbf{C}=\mathbf{X}\mathbf{X}^T$! Computing all eigenvectors of **C** will now have a complexity of only $O(N^3)$ for computation of the eigenvectors of $\mathbf{X}^T\mathbf{X}$ + *O(N* x *dN)* for computation of **Xw** from each **w** = total of $O(N^3 + dN^2)$ which is much less than $O(d^3)$. Note that **C** has at most only *min(N-1,d)* eigenvectors corresponding to non-zero eigen-values (why?).

Eigenfaces: Algorithm (N << d case)

1. Compute the mean of the given points:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i, \mathbf{x}_i \in R^d, \bar{\mathbf{x}} \in R^d$$

2. Deduct the mean from each point:

$$\bar{\mathbf{x}}_\mathbf{i} = \mathbf{x}_\mathbf{i} - \bar{\mathbf{x}}$$

3. Compute the following matrix:

$$\mathbf{L} = \mathbf{X}^T\mathbf{X}, \mathbf{L} \in R^{N\times N}, \mathbf{X} = [\bar{\mathbf{x}}_\mathbf{1} \mid \bar{\mathbf{x}}_\mathbf{2} \mid \ldots \mid \bar{\mathbf{x}}_\mathbf{N}] \in R^{d\times N}$$

$$\textit{Note}: \mathbf{L} \text{ is a symmetric matrix, and it is positive-semidefinite}$$

Eigen-faces: Algorithm (*N* << *d* case)

4. Find the eigenvectors of **L**:
   $\mathbf{LW} = \mathbf{W\Gamma}, \mathbf{W} - eigenvectors, \mathbf{\Gamma} - eigenvalues,$

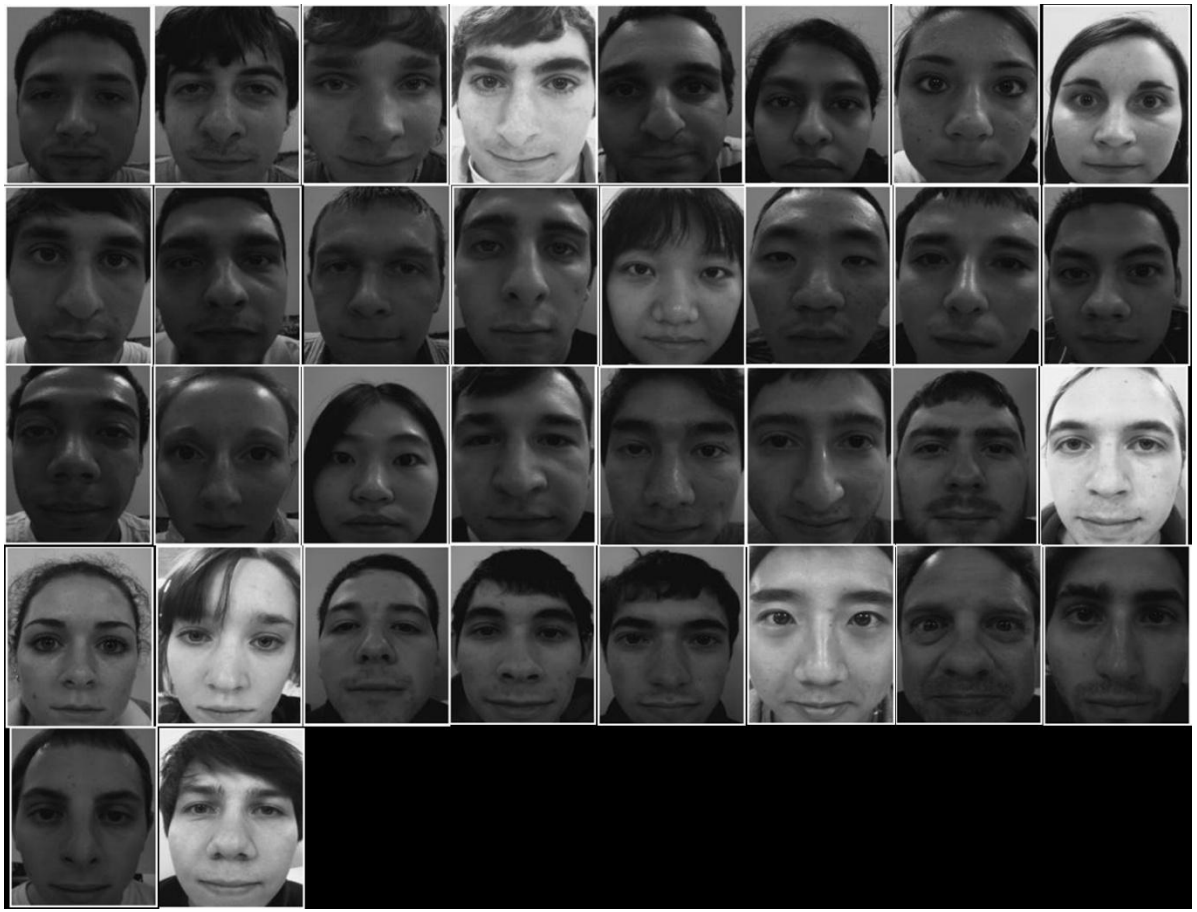
   $\mathbf{WW}^T = \mathbf{I}$
5. Obtain the eigenvectors of **C** from those of **L**:

   $\mathbf{V} = \mathbf{XW}, \mathbf{X} \in R^{d \times N}, \mathbf{W} \in R^{N \times N}, \mathbf{V} \in R^{N \times N}$

6. Unit-normalize the columns of **V**.
7. **C** will have at most only *N* eigenvectors corresponding to non-zero eigen-values*. Out of these you pick the top *k* (*k* < *N*) corresponding to the largest eigen-values.

* Actually this number is at most *N*-1 – this is due to the mean subtraction, else it would have been at most *N*.

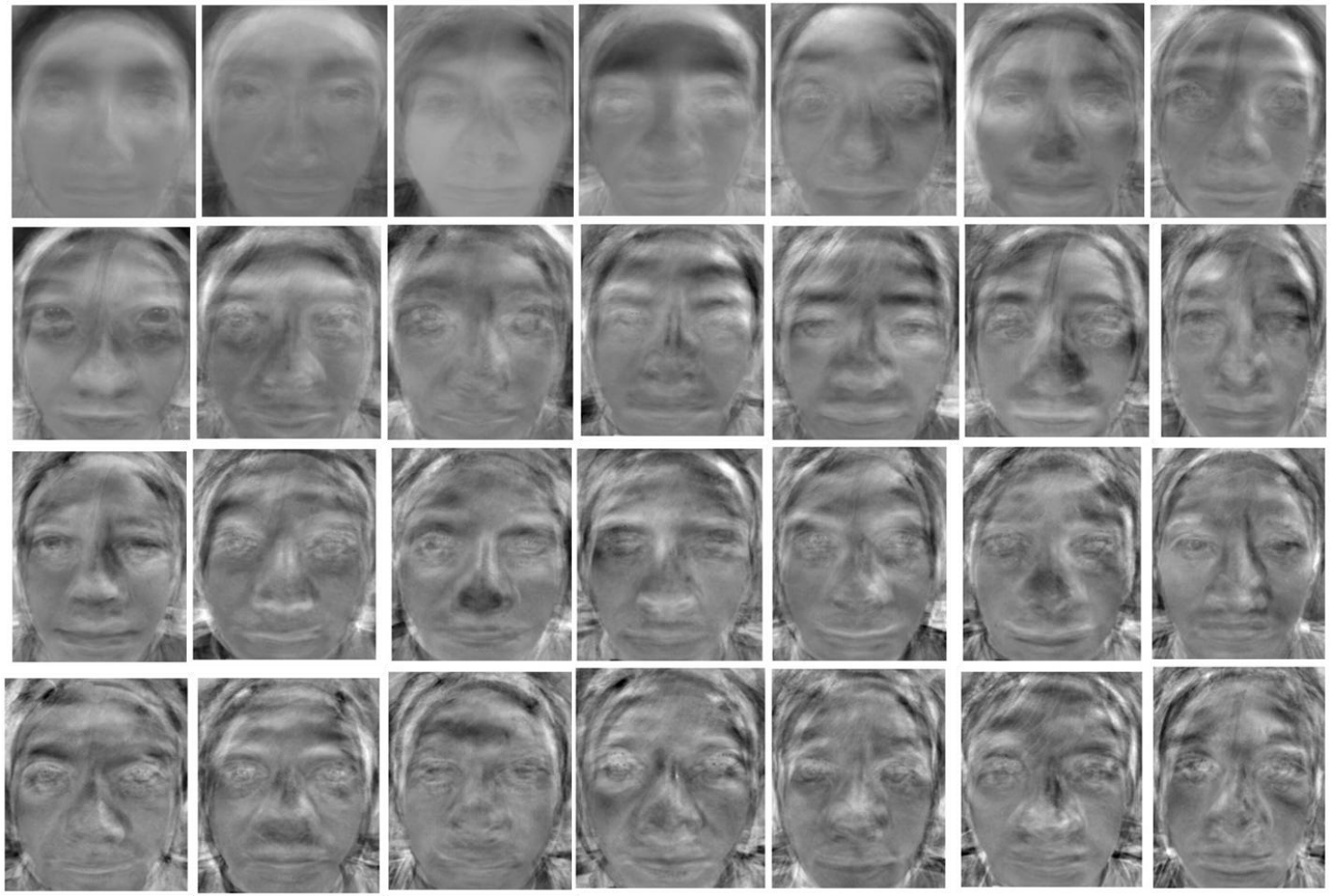
Example 1



A face database

http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

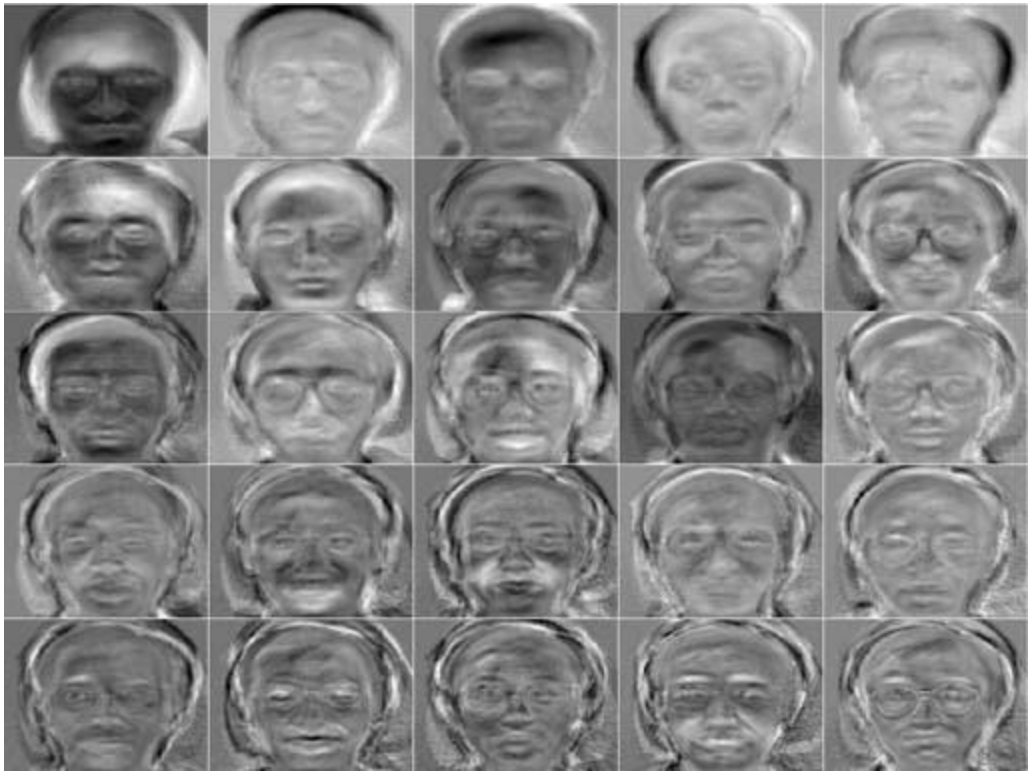
Top 25 Eigen-faces for this database!



http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

Example 2

The Yale Face database



Top 25 eigenfaces from the previous database

Reconstruction of a face image using the top 1,8,16,32,…,104 eigenfaces (i.e. *k* varied from 1 to 104 in steps of 8)

$$\mathbf{x_i} = \bar{\mathbf{x}} + \hat{\mathbf{V}}\boldsymbol{\alpha}_\mathbf{i} \approx \bar{\mathbf{x}} + \sum_{l=1}^{k} \hat{\mathbf{V}}(:,l)\boldsymbol{\alpha}_{\mathbf{ik}}(l)$$

What if both *N* and *d* are large?

- This can happen, for example, if you wanted to build an eigenspace for face images of all people in Mumbai.
- Divide people into coherent groups based on some visual attributes (eg: gender, age group etc) and build separate eigenspaces for each group.

PCA: A closer look

- PCA has many applications – apart from face/object recognition – in image processing/computer vision, statistics, econometrics, finance, agriculture, and you name it!

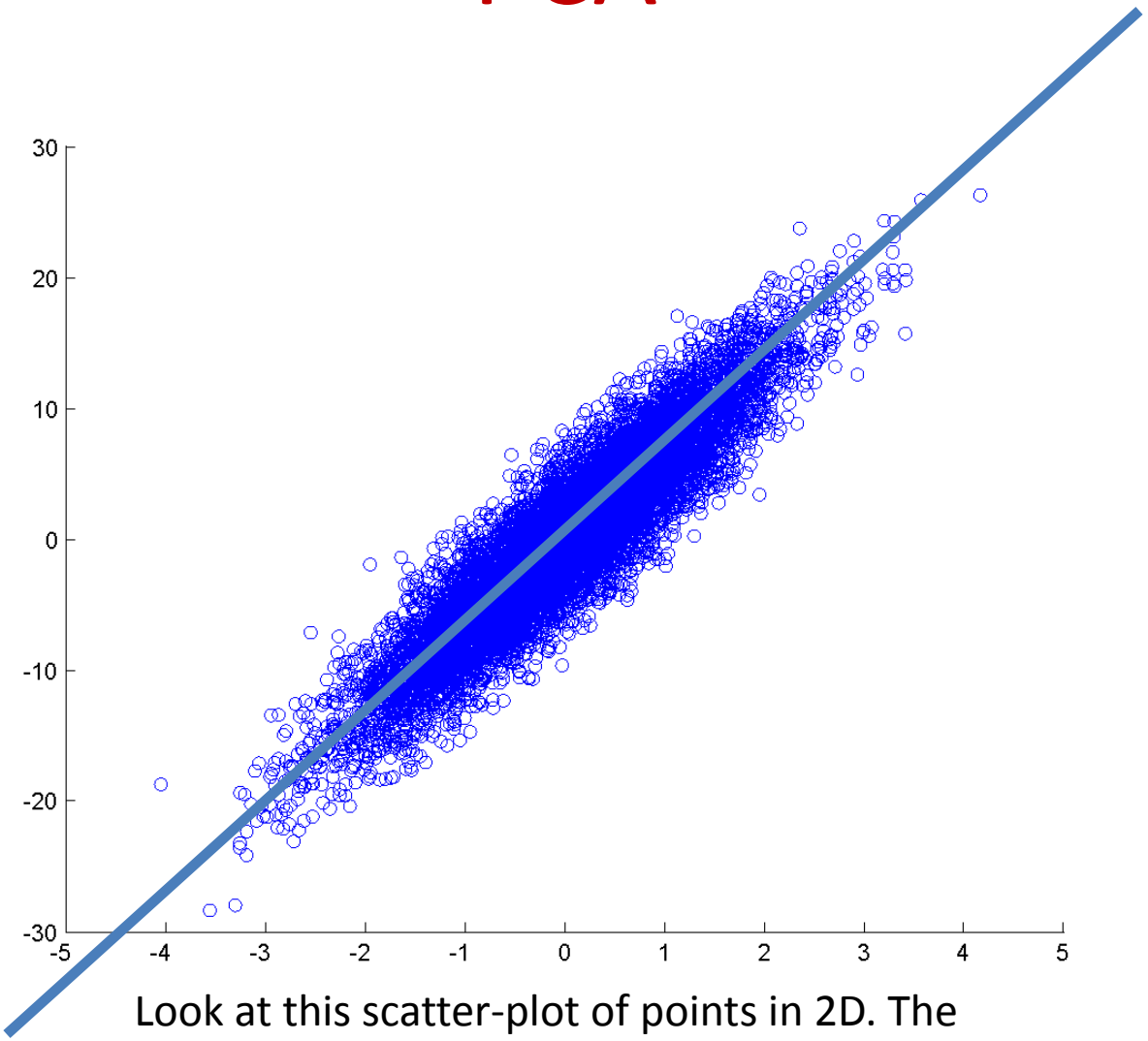
- Why PCA? What's special about PCA? See the next slides!

PCA: what does it do?

- It finds '*k*' perpendicular directions (all passing through the mean vector) such that the original data are approximated as **accurately** as possible when projected onto these '*k*' directions.

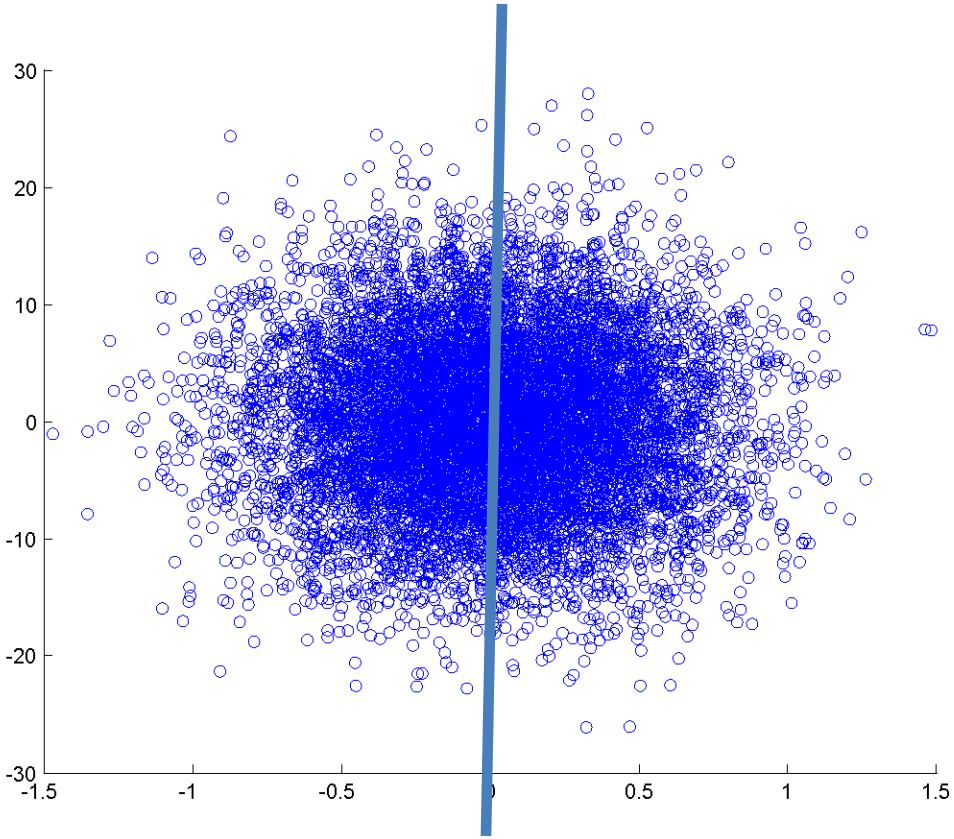
- We will see soon why these '*k*' directions are eigenvectors of the covariance matrix of the data!

PCA



Look at this scatter-plot of points in 2D. The points are highly spread out in the direction of the light blue line.

PCA



This is how the data would look if they were rotated in such a way that the major axis of the ellipse (the light blue line) now coincided with the Y axis. As the spread of the X coordinates is now relatively insignificant (**observe the axes!**), we can approximate the rotated data points by their projections onto the Y-axis (i.e. their Y coordinates alone!). This was not possible prior to rotation!

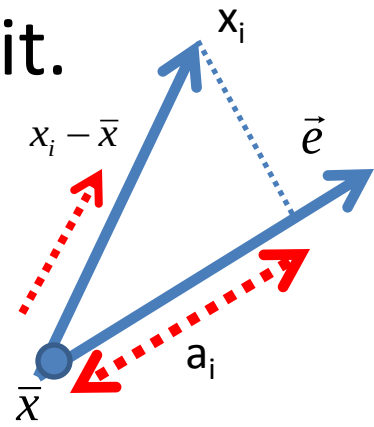
PCA

- As we could ignore the X-coordinates of the points post rotation and represent them just by the Y-coordinates, we have performed some sort of lossy data compression – or dimensionality reduction.

- The job of PCA is to perform such a rotation as shown on the previous two slides!

PCA

- **Aim of PCA:** Find the line $\vec{e}$ passing through the sample mean (i.e. $\bar{x}$), such that the projection of any mean-deducted point $x_i - \bar{x}$ onto $\vec{e}$, most accurately approximates it.



$$\text{Projection of } x_i - \bar{x} \text{ onto } \vec{e} \text{ is} = a_i\vec{e}, a_i \in R,$$

$$a_i = \vec{e}^T(x_i - \bar{x})$$

$$\text{Error of approximation} = \left\| (a_i\vec{e}) - (x_i - \bar{x}) \right\|^2$$

Note: Here $\vec{e}$ is a **unit vector.**

PCA

- Summing up over all points, we get:

$$\text{Sum total error of approximation} = J(\vec{\mathbf{e}}) = \sum_{i=1}^{N} \left\| (a_i \vec{\mathbf{e}}) - (\mathbf{x_i} - \bar{\mathbf{x}}) \right\|^2$$

$$= \sum_{i=1}^{N} \left\| (a_i \vec{\mathbf{e}}) \right\|^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}})$$

$$= \sum_{i=1}^{N} \left\| (a_i \vec{\mathbf{e}}) \right\|^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}})$$

$$= \sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}})$$

$$= \sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i^2, (\because a_i = \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}}))$$

$$= -\sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2$$

PCA

$$\text{Sum total error of approximation} = J(\vec{\mathbf{e}}) = -\sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2$$

$$= -\sum_{i=1}^{N} (\vec{\mathbf{e}}^t(\mathbf{x_i} - \bar{\mathbf{x}}))^2 + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2$$

$$= -\sum_{i=1}^{N} \vec{\mathbf{e}}^t(\mathbf{x_i} - \bar{\mathbf{x}})(\mathbf{x_i} - \bar{\mathbf{x}})^t \vec{\mathbf{e}} + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2$$

$$= -\vec{\mathbf{e}}^t \mathbf{S} \vec{\mathbf{e}} + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2 \quad (\text{where } \mathbf{S} = (N-1)\mathbf{C})$$

This term is proportional to the variance of the data points when projected onto the direction **e**.

PCA

$$J(\vec{\mathbf{e}}) = -\vec{\mathbf{e}}^{\mathrm{t}}\mathbf{S}\vec{\mathbf{e}} + \sum_{i=1}^{N} \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2$$

Independent of the direction e

Minimizing $J(\vec{\mathbf{e}})$ w.r.t. $\vec{\mathbf{e}}$ is equivalent to maximizing $\vec{\mathbf{e}}^{\mathrm{t}}\mathbf{S}\vec{\mathbf{e}}$ w.r.t. $\vec{\mathbf{e}}$.

We use the method of Lagrange multipliers to do so, while simultaneously imposing the constraint that $\vec{\mathbf{e}}^{t}\vec{\mathbf{e}} = 1$.

See appendix for details

So we have to take the derivative of the following modified function w.r.t. $\vec{\mathbf{e}}$ (and set it to 0)

$$\tilde{J}(\vec{e}) = \vec{\mathbf{e}}^{\mathrm{t}}\mathbf{S}\vec{\mathbf{e}} - \lambda(\vec{\mathbf{e}}^{t}\vec{\mathbf{e}} - 1)$$

Taking derivative of $\tilde{J}(\vec{\mathbf{e}})$ w.r.t. $\vec{\mathbf{e}}$ and setting it to 0, we get

$$\mathbf{S}\vec{\mathbf{e}} = \lambda\vec{\mathbf{e}},$$

so $\vec{\mathbf{e}}$ is an eigen - vector of $\mathbf{S}$.

As $\vec{\mathbf{e}}^{t}\mathbf{S}\vec{\mathbf{e}} = \lambda$ and we wish to maximize $\vec{\mathbf{e}}^{t}\mathbf{S}\vec{\mathbf{e}}$, we choose $\vec{\mathbf{e}}$ to be the eigen - vector corresponding to the maximum eigenvalue of $\mathbf{S}$.

PCA

- PCA thus projects the data onto that direction that minimizes the total squared difference between the data-points and their respective projections along that direction.

- This **equivalently** yields the direction along which the spread (or variance) will be maximum.

- Why? Note that the eigenvalue of a covariance matrix tells you the variance of the data when projected along that particular eigenvector:

$$\mathbf{S}\vec{\mathbf{e}} = \lambda\vec{\mathbf{e}} \rightarrow \vec{\mathbf{e}}^t\mathbf{S}\vec{\mathbf{e}} = \lambda$$

$$\vec{\mathbf{e}}^t\mathbf{S}\vec{\mathbf{e}} = \sum_{i=1}^{N}(\vec{\mathbf{e}}^t(\mathbf{x_i} - \bar{\mathbf{x}}))^2$$

This term is proportional to the variance of the data when projected along **e**.

PCA

- But for most applications (including face recognition), just a single direction is absolutely insufficient!

- We will need to project the data (from the high-dimensional, i.e. *d*-dimensional space) onto *k* (*k* << *d*) different mutually perpendicular directions.

- What is the criterion for deriving these directions?

- We seek those *k* directions for which the total reconstruction error of all the *N* images when projected on those directions is minimized.

PCA

- We seek those *k* directions for which the total reconstruction error of all the *N* images when projected on those directions is minimized.

$$J(\{\mathbf{e_j}\}_{j=1}^{k}) = \sum_{i=1}^{N} \left\| (\mathbf{x_i} - \bar{\mathbf{x}}) - \sum_{j=1}^{k} (\mathbf{e_j}^t (\mathbf{x_i} - \bar{\mathbf{x}}))\mathbf{e_j} \right\|_2^2$$

- One can prove that these *k* directions will be the eigenvectors of the **S** matrix (equivalently covariance matrix of the data) corresponding to the *k*-largest eigenvalues. These *k* directions form the eigen-space.

- If the eigenvalues of **S** are distinct, these *k* directions are defined uniquely (up to a sign factor)

PCA

- One can prove that these *k* directions will be the eigenvectors of the **S** matrix (equivalently covariance matrix of the data) corresponding to the *k*-largest eigenvalues. These *k* directions form the eigen-space.

- **Sketch of the proof:**

✓ Assume we have found $\mathbf{e}_1$ and are looking for $\mathbf{e}_2$ (where $\mathbf{e}_2$ is perpendicular to $\mathbf{e}_1$ and $\mathbf{e}_2$ has unit magnitude).

✓ Write out the objective function with the two constraints.

✓ Minimize it and do some algebra to see that $\mathbf{e}_2$ is the eigenvector of **S** with the second largest eigenvalue.

✓ Proceed similarly for other directions.

How to pick *k* in an actual application?

- Trial and error. Usually between 50 to 100 for images of size 200 x 200.
- Divide your training set into two parts – *A* and *B* (B is usually called the **validation set**). Pick the value of *k* that gives the best recognition rate on B when you train on A.
- Stick to that value of *k*.
- Note: a larger *k* implies a better reconstruction but it may even cause a *decrease* in the recognition accuracy!

How to pick *k* in an actual application?

- Note: a larger *k* implies a better reconstruction but it may even cause a *decrease* in the recognition accuracy!

- Why – because throwing out some of the eigenvectors may lead to filtering of the data, removing some unnecessary artifacts for example.

Some observations about PCA for face images

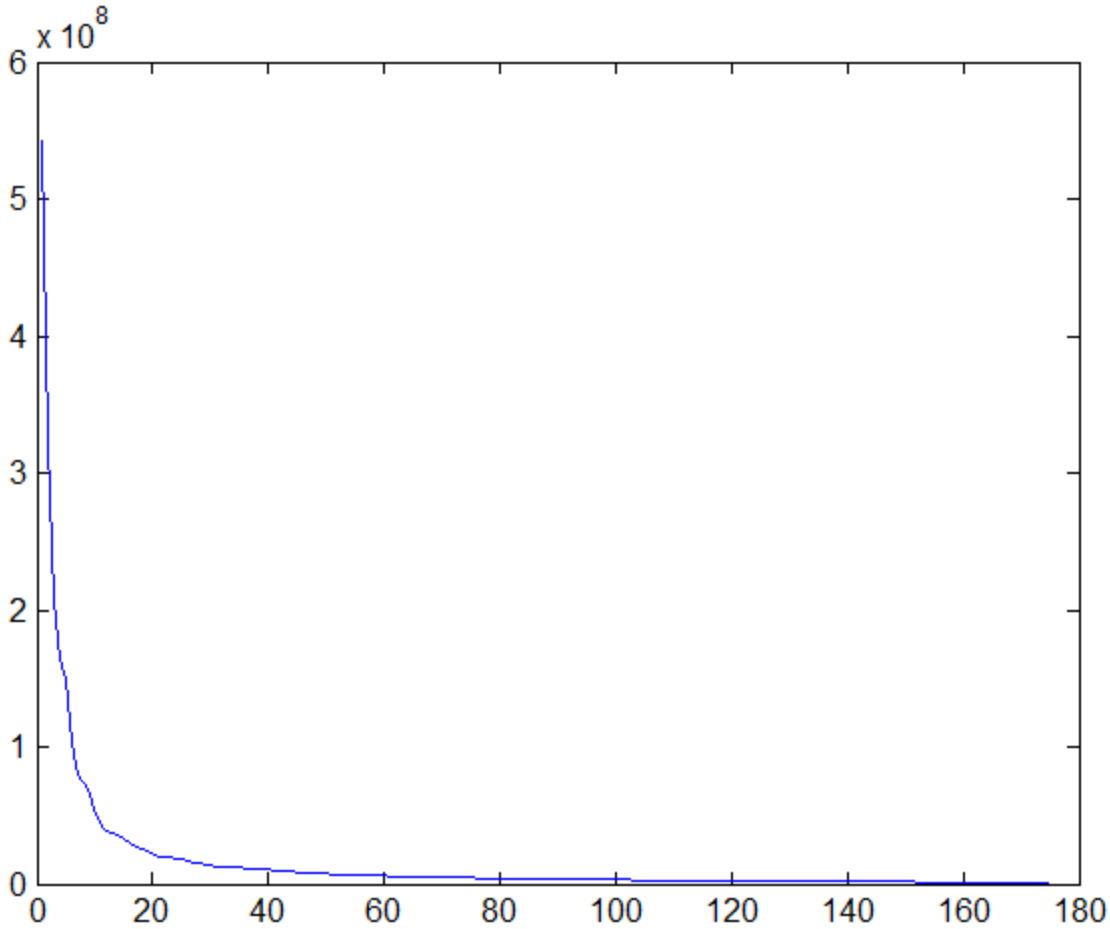
- The matrix **V** (with all columns) is orthonormal. Hence the squared error between any image and its approximation using just top *k* eigenvectors is given by:

$$\left\|\mathbf{x}_i - \tilde{\mathbf{x}}_i\right\|^2 = \left\|V(\boldsymbol{\alpha}_i - \tilde{\boldsymbol{\alpha}}_i)\right\|^2 (why?)$$

$$= \left\|\boldsymbol{\alpha}_i - \tilde{\boldsymbol{\alpha}}_i\right\|^2 (why?)$$

$$= \sum_{j=k+1}^{d} \boldsymbol{\alpha}_j^2 (why?)$$

This error is small ***on an average for a well-aligned group of face images*** – we will see why on the next slide.



The eigenvalues of the covariance matrix typically decay fast in value (if the faces were properly normalized). Note that the *j*-th eigenvalue is proportional to the variance of the j-th eigencoefficient, i.e.

$$\lambda_j = \mathbf{e}_\mathbf{j}^\mathbf{t}\mathbf{S}\mathbf{e}_\mathbf{j} = \sum_{i=1}^{N} \mathbf{e}_\mathbf{j}^\mathbf{t}(\mathbf{x}_\mathbf{i} - \overline{\mathbf{x}})(\mathbf{x}_\mathbf{i} - \overline{\mathbf{x}})^t\mathbf{e}_\mathbf{j} = (N-1)E(\alpha_{ij}^2)$$

What this means is that the data have low variance when projected along most of the eigenvectors, i.e. effectively the data are concentrated in a lower-dimensional subspace of the d-dimensional space.

Person-specific eigen-faces

- So far, we built one eigen-space for the whole database – consisting of multiple images each of multiple people.

- Alternative approach: Construct one eigen-space (i.e. a set of some *k* eigenvectors) for each of the *M* people. Assume we have multiple gallery images per person, possibly under different poses, illumination conditions and expressions.

Person-specific eigen-space

- Let the eigen-space for the *r*-th person be denoted as $\mathbf{V}_k^{(r)}$ and the corresponding mean image be denoted as $\bar{\mathbf{x}}^{(r)}$.
- The eigen-coefficients of a probe image onto the *r*-th eigen-space are given as:

$$\boldsymbol{\beta}_p^{(r)} = \mathbf{V}_k^{(r)T}(\mathbf{z}_p - \bar{\mathbf{x}}^{(r)})$$

- For every *r*, determine the following:

$$d(r) = \left\| \mathbf{z}_p - \bar{\mathbf{x}}_r - \hat{\mathbf{V}}_k^{(r)} \boldsymbol{\beta}_p^{(r)} \right\|^2$$

A measure of how well *'k'* eigen-vectors from the eigen-space for person *r*, are capable of reconstructing the probe image. Note that **k** ≤ number of gallery images for the r-th person (why?)

Person-specific eigen-space

- An alternative distance measure is:

$$d(r) = \left\| \bar{\beta}^{(r)} - \beta_p^{(r)} \right\|^2$$

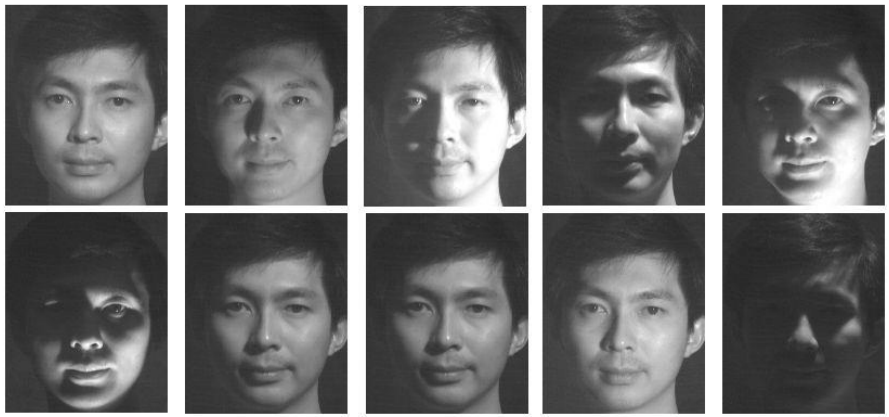
Average eigen-coefficient vector of all gallery images belonging to the *r*-th person

- The identity of the probe image is given by the eigen-space '*r*' for which *d(r)* was the minimum.

Face recognition under varied lighting

- The appearance variation over images of the same person under different illuminations is much greater than the variation over images of different people under the same lighting condition!

Illumination Variability



"The variations between the images of the same face due to illumination and viewing direction are almost always larger than image variations due to change in face identity."

-- Moses, Adini, Ullman, ECCV '94

Face recognition under varied lighting

- Given a database of images of *M* people each under *L* lighting conditions, create an eigenspace and **remove the top 3 principal components** for better face recognition!

- Why – we will understand the real reason in a computer vision class!

A word of clarification

- PCA ≠ magic. It has no in-built ability to perform pose normalization.
- For pose normalization, you can build pose-specific eigen-spaces or person-specific eigen-spaces using multiple images of a person under different poses.
- For (partial) illumination normalization, you can use the trick mentioned on the previous slide.
- However, in general pose normalization is a major problem in face recognition from 2D images.

Another word of clarification

- PCA at its core is a reconstruction algorithm.
- It is not a classification algorithm and is not designed to be one.
- But it showed very good results for face recognition and hence became popular in this community.
- There are other methods (eg: Linear Discriminant Analysis) which are designed for the purpose of good classification – on face images or other datasets.

PCA: Compression of a set of images

- Consider a database of *N* images that are "similar" (eg: all are face images, all are car images, etc.)
- Build an eigen-space from some subset of these images (could be all images, as well)
- We know that these images can often be reconstructed very well (i.e. with low error) using just a few eigenvectors.

PCA: Compression of a set of images

- Use this fact for image compression.
- Original data storage = d pixels x N images = Nd bytes (assume one byte per pixel intensity) = $8Nd$ bits.
- After PCA: Nk x number of bits to store each eigen-coefficient = $32Nk$ bits (remember $k << d$, example: $d \sim 250{,}000$ and $k \sim 100$).
- Plus storage of eigenvectors = $32dk$ bits (remember $k << M$ as well).
- Plus mean image = $8d$ bits.
- Total: $32(N+d)k + 8d$ bits

PCA: Compression of a set of images

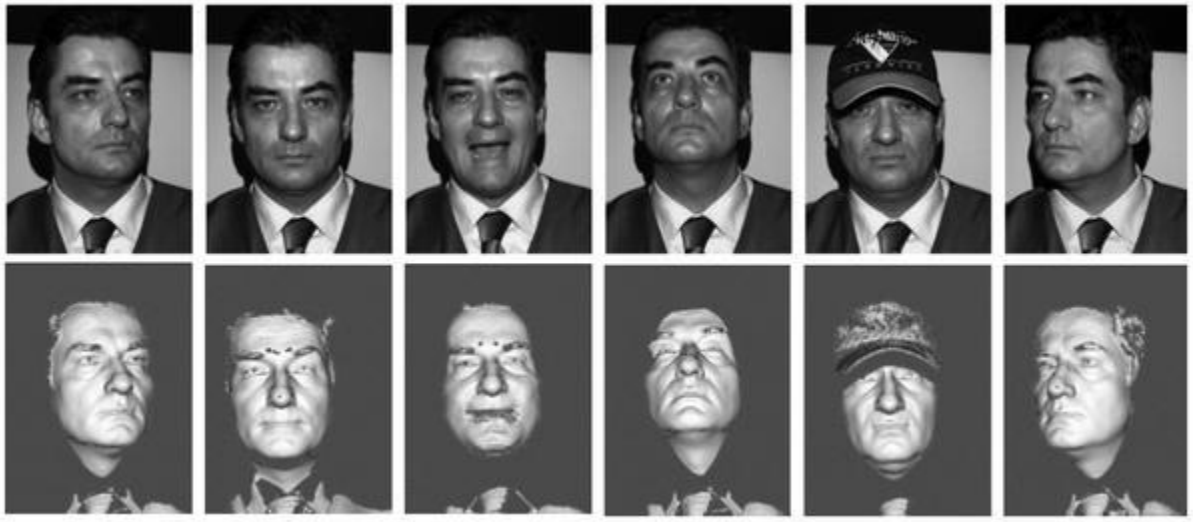
- Example: N= 5000, d = 250000, k = 100
- Original size/(size after PCA compression) ~ 12.2.
- Note: we allocated 32 bits for every element of the eigen-vector. This is actually very conservative and you can have further savings using several tricks.

PCA: Compression of a set of images

- This differs a lot from JPEG compression.
- JPEG uses discrete cosine transform (DCT) as the basis. PCA tunes the basis to the underlying training data! If you change the training data, the basis changes!
- The performance of the PCA compression algorithm will depend on how compactly the eigen-space can represent the data.
- We will study image compression using JPEG and other standards later on in the course.

Face Recognition: Other types of images

- From range data – called as 3D face recognition (http://www.wired.com/2008/12/german-security/)



Face Recognition: Other types of images

- From video!

Face recognition: from the FBI

- http://www.fbi.gov/about-us/cjis/fingerprints_biometrics/biometric-center-of-excellence/files/face-recognition.pdf

Conclusion

- We studied:

✓ Face recognition and related problem statements

✓ Eigenfaces algorithm – faster and slower versions

✓ Derivation of the algorithm

✓ Modifications for face detection/verification/person-specific eigenfaces

✓ Application for compression of images

References

- Section 3.8 of “Pattern Classification” by Duda and Hart
- http://en.wikipedia.org/wiki/Eigenface
- M. Turk and A. Pentland (1991). "Eigenfaces for recognition". *Journal of Cognitive Neuroscience*

Appendix: Covariance matrix

- In probability and statistics, the expected value of a scalar random variable x is called its mean:

$$\mu = E(x) = \int x p(x) dx \approx \frac{1}{N} \sum_{i=1}^{N} x_i$$

Sample values of the random variable

Probability density function of x

- The variance of a scalar random variable is the expected value of its squared deviation around the mean:

$$\sigma^2 = E((x-\mu)^2) = \int (x-\mu)^2 \, p(x) dx \approx \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2$$

Appendix: Covariance matrix

- The expected value of a vector random variable is called its mean vector:

$$\vec{\boldsymbol{\mu}} = E(\vec{\mathbf{x}}), \vec{\mathbf{x}} = (x(1), x(2), ..., x(d)), \vec{\boldsymbol{\mu}} = (\mu(1), \mu(2), ..., \mu(d))$$

$$\mu_k \approx \frac{1}{N} \sum_{i=1}^{N} x_{ki}, 1 \le k \le d$$

k-th sample value of $\mathbf{x}_i$

Appendix: Covariance matrix

- The variance is now replaced by a covariance matrix. Each entry contains the covariance between two elements of the vector:

$$\mathbf{C} = \begin{pmatrix} E((x(1)-\mu(1))^2) & E((x(1)-\mu(1))(x(2)-\mu(2))) & . & . & E((x(d)-\mu(d))(x(1)-\mu(1))) \\ E((x(2)-\mu(2))(x(1)-\mu(1))) & E((x(2)-\mu(2))^2) & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ E((x(1)-\mu(1))(x(d)-\mu(d))) & . & . & . & E((x(d)-\mu(d))^2) \end{pmatrix} \in R^{d\times d}$$

$$C_{kl} = E((x(k)-\mu(k))(x(l)-\mu(l))) \approx \frac{1}{N-1}\sum_{i=1}^{N}(x_i(k)-\mu(k))(x_i(l)-\mu(l)), 1 \le k \le d, 1 \le l \le d$$

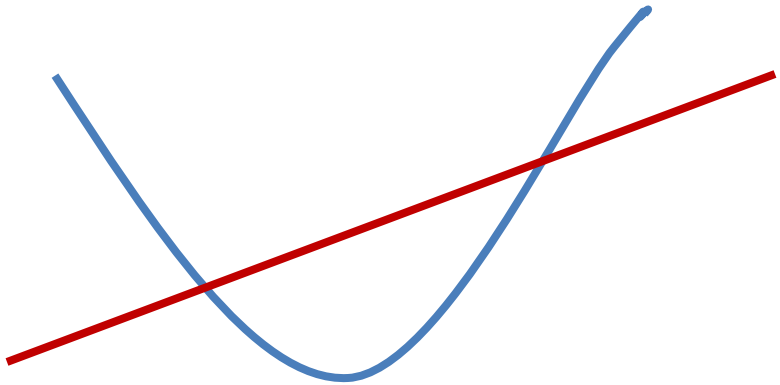
$$Note: \mu(k) = k-th \text{ element of vector } \mu$$

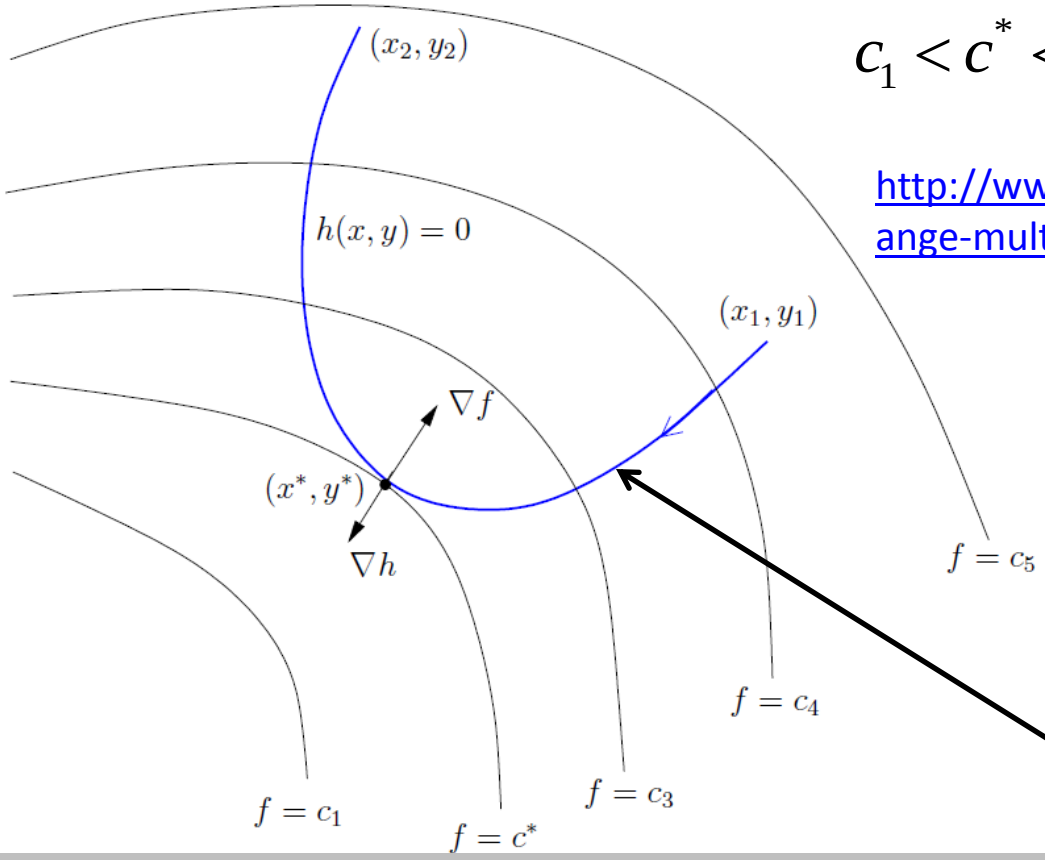
Appendix: A word about Lagrange Multipliers

- Consider you want to find the minimum or maximum of f(x,y).
- Normally, you take the derivative and set it to 0 and check the sign of the second derivative.
- Now you want to find the optimum subject to a constraint that h(x,y) = 0.

Appendix: A word about Lagrange Multipliers

- Physical analogy: if you drop a pebble into a parabola-shaped bowl, the pebble will fall to the bottom-most point.
- But if you placed a wooden plank into the bowl, the pebble will settle somewhere on the plank!





$$c_1 < c^* < c_2 < c_3 < c_4 < c_5$$

http://www.cs.iastate.edu/~cs577/handouts/lagrange-multiplier.pdf

Suppose we are walking from (x_1,y_1) to (x_2,y_2) along the constraint curve h(x,y) = 0. Initially, the tangent vector along h(x,y) = 0 has a component along –grad(f), and hence there is a decrease in the value of f(x,y) as we move along h(x,y) = 0. If the point (x*,y*) is a local minimum of f(x,y), a small motion along h(x,y) = 0 from (x*,y*) will have no component along grad(f), else it would cause an increase in the value of f. Hence the tangent to h(x,y) = 0 at (x*,y*) will be perpendicular to grad(f). As we move further, motion along h(x,y) = 0 will have a component along +grad(f) leading to an increase in the value of f(x,y). At the minimum point, we have grad(f) perpendicular to the tangent , i.e. grad(f) and grad(h) are collinear. Hence, there exists some value λ (the **Lagrange multiplier)** such that we have:

$$\nabla f(x^*, y^*) = \lambda \nabla h(x^*, y^*) \rightarrow \nabla(f(x^*, y^*) - \lambda h(x^*, y^*)) = 0$$

If (x',y') is the extremum of function $f(x,y)$, then a small perturbation to (x',y') will lead to no change in the value of f - in the limit when the magnitude of the perturbation is 0.

Hence we have $|\nabla \mathbf{f}(x', y') = 0|$.

But here we are constrained to move only along the curve $h(x,y) = 0$.

Let $\mathbf{s}$ be the tangent to this curve at a point (x', y').

The magnitude of the change in f due to infinitesimal movement along the curve is given by $\nabla \mathbf{f}(x', y') \bullet \mathbf{s}$.

If (x', y') is an extremum of f along this curve, then we must have

$\nabla \mathbf{f}(x', y') \bullet \mathbf{s} = 0$.

But the normal to the curve is $\nabla \mathbf{h}(x', y')$ and it is perpendicular to $\mathbf{s}$.

Hence we have $\nabla \mathbf{f}(x', y')$ is collinear with $\nabla \mathbf{h}(x', y')$.

$\therefore \nabla \mathbf{f}(x', y') = \lambda \nabla \mathbf{h}(x', y')$ for some value λ.

In our derivation for PCA, instead of $f(x,y)$ we have $\mathbf{e}^t\mathbf{S}\mathbf{e}$, and instead of $h(x, y) = 0$, we have $\mathbf{e}^t\mathbf{e} - 1 = 0$. Note that $\mathbf{e}$ is a vector in d dimensional space.