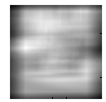
Recognition

- Correlation
- Features (geometric hashing)
- Moments
- Eigenfaces

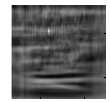
Normalized Correlation - Example

image

pattern

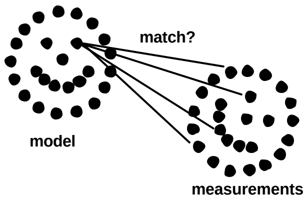


Correlation



Normalized Correlation

Correspondence Problem



• Solution for affine transformation(*): test matching of all triplets in the model and the data measurements.

• Problem: very high computational complexity.

• Solution: geometric hashing.

• (*) Affine = linear + translation.

The idea: if p_1, p_2, p_3, p_4 are points in the model which satisfy

$$ap_1 + bp_2 + cp_3 = p_4, a + b + c = 1$$

Then, if there's an affine model-data matching

$$p_1 \to q_1, p_2 \to q_2, p_3 \to q_3, p_4 \to q_4$$

We will also have: $aq_1 + bq_2 + cq_3 = q_4$

Geometric hashing uses a hash table to search for similar (a, b, c) triplets in the model and the data.

INVARIANTS

Quantities which do not change when the image is, for example, rotated. We will assume that images have been normalized by placing the center of mass at the origin.

Moments of set S: $m_{i,j} = \sum_{(x,y)\in S} x^i y^j$

Euclidean invariant (doesn't change under rotation):

$$m_{2,0} + m_{0,2}$$

Prepared by Michael Pechuk
Supervised by Daniel Keren

Face Recognition Using Eigenfaces

M.A. Turk and A.P. Pentland:

Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3 (1):71--86, 1991.

Introduction

- What?
  - Automatic Learning and Recognition
  - Real-time despite high complexity
- Why?
  - Security systems
  - Criminal identification and investigations.
  - Computer interface
- How?
  - PCA
  - “Face space”

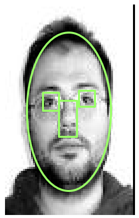
What is Face Recognition?

- You have a set of familiar faces. Given a new face, do you recognize it or not?



Background and Related Work

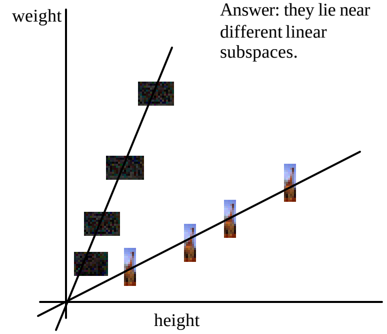
- Previous approaches
  - Key features: eyes, nose, mouth, head outline
  - Feature detection
  - Face model by position, size and relation of features
  - Geometric hashing



- Difficulties
  - High complexity
  - A lot of preprocessing (for example edge detection).

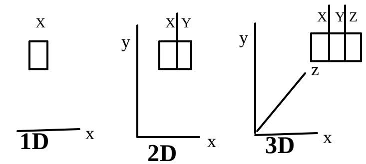
The idea:

How can we tell apart elephants and giraffes, based only on weight and height?



Answer: they lie near different linear subspaces.

Reminder: Linear Algebra



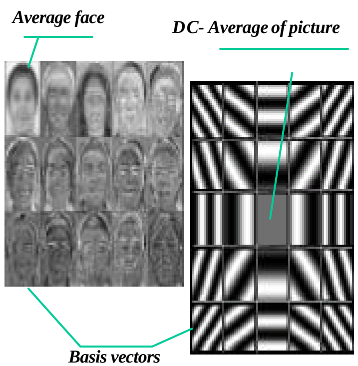
An $n \times n$ image is embedded in $\Re^{n^2}$.

Eigenfaces for Recognition

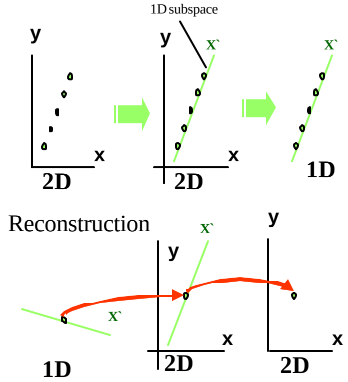
Basic Idea

- Picture is in multidimensional space
  - Matrix representation of pictures
    - 256x256 pixels = 65536 dimensions
- Face pictures are a subspace of picture space
  - "Face space"
    - Less dimensions
    - Images stored in series of weights
    - Eigenvectors – Eigenfaces
    - FFT – sinusoids, "face space" - eigenfaces

Eigenfaces vs FFT



Example of Dimension Decrease



Recognition process steps:

1. Initialization

Once

2. Get the new image and project it to face space
3. Determine if the image is a face at all
4. Recognition
5. Learning

Initialization: calculating Eigenfaces

PCA=Principal Component Analysis

Given a set of images $\{\Gamma_1, \Gamma_2, \Gamma_3 ...\}$ calculate the covariance matrix C, by first subtracting the average image Ψ. Ψ is calculated by summing up the M images and dividing by M.

This gives you a set of M Φ's. $(\Phi = \Gamma - \Psi)$

$$C = \frac{1}{M}\sum_{i=1}^{M}\Phi_i\Phi_i^t = AA^T$$

Where $A = [\Phi_1, \Phi_2, \Phi_3 .. \Phi_M]$

You can then calculate the eigenvectors for these matrices.

But calculating with AA^T is solving a matrix that is NxN !!! It`s too much! ***For example:*** *for a picture 100 X100 ,N=10000.*

Initialization: calculating Eigenfaces

The Trick

A^TA is an MxM matrix. Where M is number of pictures.

By doing some manipulations we can use this.

If the eigenvectors are v_i then $A^TAv_i = \lambda_i v_i$

Multiplying both sides by A gives you

$AA^T(Av)_i = \lambda_i(Av)_i$

Av_i is the set of eigenvectors for AA^T

So you can use A^TA to get v_i, and you can use Av_i to get the eigenvectors for AA^T -- don't have to deal with NxN matrices, just MxM.

For example: for *200 pictures 100 x100, we will deal with 200 x200 matrix instead of 10000 x10000 matrix!*

Initialization: calculating Eigenfaces

The Trick.

Lets take 2 pictures, 3 pixels each:

$$f = (f_1, f_2, f_3) \qquad g = (g_1, g_2, g_3)$$

So, we will get: $A = \begin{pmatrix} f_1 & g_1 \\ f_2 & g_2 \\ f_3 & g_3 \end{pmatrix}$ $\quad A^T = \begin{pmatrix} f_1 & f_2 & f_3 \\ g_1 & g_2 & g_3 \end{pmatrix}$

$$A^T A = \begin{pmatrix} f_1^2 + f_2^2 + f_3^2 & g_1 f_1 + g_2 f_2 + g_3 f_3 \\ g_1 f_1 + g_2 f_2 + g_3 f_3 & g_1^2 + g_2^2 + g_3^2 \end{pmatrix}$$

$$AA^T = \begin{pmatrix} f_1^2 + g_1^2 & f_1 f_2 + g_1 g_2 & f_1 f_3 + g_1 g_3 \\ f_1 f_2 + g_1 g_2 & f_2^2 + g_2^2 & f_2 f_3 + g_2 g_3 \\ f_1 f_3 + g_1 g_3 & f_2 f_3 + g_2 g_3 & f_3^2 + g_3^2 \end{pmatrix}$$

But:

$$(A^T A)v = lv \Rightarrow A(A^T A)v = l(Av) \Rightarrow (AA^T)(Av) = l(Av)$$

So, we can solve only 2x2 matrix instead of 3x3 matrix, and then get the eigenvectors $\mathbf{v_i}$ for AA^T simply by the multiplication $\mathbf{v_i} = Av_i$

Initialization: calculating Eigenfaces

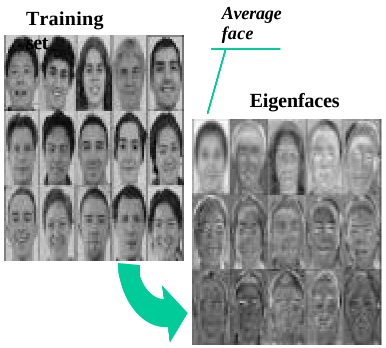
Eigenvector formula...

- To calculate the eigenvectors you must first calculate the eigenvalues, by using $|C-\lambda I| = 0$ and solving for λ.
- You then solve $Cv = \lambda v$

No more math!

- You can delete some of the eigenfaces, the ones with with the smallest associated eigenvalues.
- Once you have the eigenfaces, you should use them to project the training set into "***face space***".
- You store the images as a series of weights, which are the coefficients in face space, or simply the inner products with the eigenfaces (which are orthogonal).

Initialization: Eigenfaces example



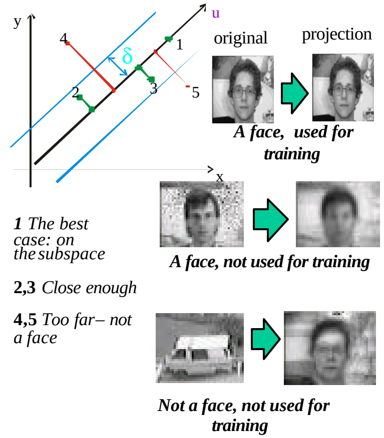
Get the New Image

A new face image (Γ) is projected into "***face space***" by a simple operation (projection):

$\omega_k = \upsilon_k^T(\Gamma - \Psi)$ *for k=1,…,M`*

Vector $\Omega^T = (\omega_1, \omega_2, .., \omega_{M'})$ represents input image in ***"face space",*** where weight ω_i – contribution of each eigenface

Determine If the Image is a Face at All



1 *The best case: on the subspace*

2,3 *Close enough*

4,5 *Too far– not a face*

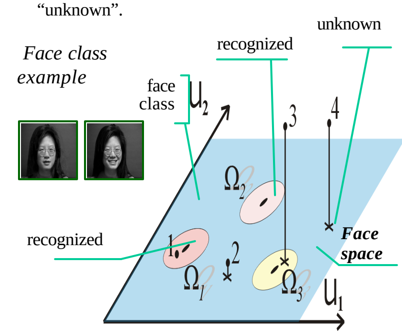
Recognition

Face class (O_k) is the set of faces of one person

To recognize we will find the minimum of $e_k = \|O - O_k\|^2$,

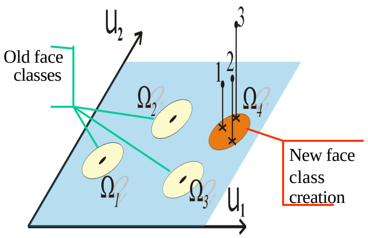
and if e_k is below some threshold θ, face belongs to person k,

Otherwise the face is classified as "unknown".



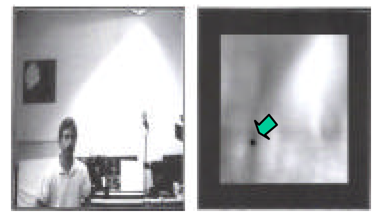
Learning

If same unknown case is seen several times, calculate its characteristic weight pattern and incorporate into the known faces – create a new ***face class*** (i.e., learn to recognize it)

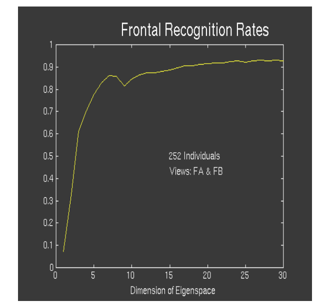


Using Eigenfaces to Detect Faces

"*Face map*" creation. The distance from sub image in a point to face space is used as a measure of "faceness".



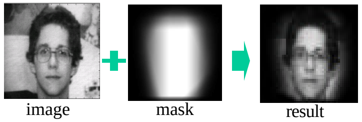
Face map. Low values (dark area) indicate the presence of face



Face space dimension impact on face recognition

Improving

1. Eliminating the Background



2. Scaling and Orientation
   - Pyramids
   - +/- 45^{o} rotation
3. Multiple Views

Summary

How does this method perform compared to other methods?

- Because the eigenvectors only need to be computed once and are easy to find, this is very fast compared to other methods.
- Other methods require a significant amount of preprocessing, where this does not. (i.e. calculating the edges within an image)
- Eigenfaces are accurate but have a hard time dealing with discrepancies between the training and testing sets in light, camera angle, and variable facial features. (i.e. smiling, mustaches, glasses, etc.)