Recognition

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

Normalized Correlation - Example

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

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

We will also have:

$$aq_1 + bq_2 + cq_3 = q_4$$

Geometric hashing uses a hash table to search for similar 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):

$$m_{2,0} + m_{0,2}$$
Face Recognition Using Eigenfaces

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

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 $\mathbb{R}^n$.
**Eigenfaces for Recognition**

**Basic Idea**
- Picture is in multidimensional space
  - Matrix representation of pictures
    - $256 \times 256$ 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
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 \( \Gamma_1, \Gamma_2, \Gamma_3, \ldots \) calculate the covariance matrix \( C \) by first subtracting the average image \( \Psi \). \( \Psi \) is calculated by summing up the \( M \) images and dividing by \( M \).

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

\[
C = \frac{1}{M} \sum_{i=1}^{M} \Phi_i \Phi_i^T = \Lambda \Lambda^T
\]

Where \( \Lambda = [\Phi_1, \Phi_2, \Phi_3, \Phi_M] \)

You can then calculate the eigenvectors for these matrices.

*But calculating with \( \Lambda \Lambda^T \) is solving a matrix that is \( N \times N \)!!! It's too much!* For example: for a picture 100 X100, \( N=10000 \).

---

**Initialization:** calculating Eigenfaces

**The Trick**

\( \Lambda^T \Lambda \) is an \( M \times M \) matrix. Where \( M \) is number of pictures.

By doing some manipulations we can use this.

If the eigenvectors are \( v_i \) then \( \Lambda^T \Lambda v_i = \lambda_i v_i \)

Multiplying both sides by \( \Lambda \) gives you \( \Lambda \Lambda^T (\Lambda v_i) = \lambda_i (\Lambda v_i) \)

\( \Lambda v_i \) is the set of eigenvectors for \( \Lambda \Lambda^T \)

So you can use \( \Lambda^T \Lambda \) to get \( v_i \), and you can use \( \Lambda v_i \) to get the eigenvectors for \( \Lambda \Lambda^T \) -- don't have to deal with \( N \times N \) matrices, just \( M \times M \).

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

*The Trick.*

Let's take 2 pictures, 3 pixels each:

\[ f = (f_1, f_2, f_3) \quad \text{and} \quad g = (g_1, g_2, g_3) \]

So, we will get:

\[ A = \begin{pmatrix} f_1 & f_2 & f_3 \\ g_1 & g_2 & g_3 \end{pmatrix} \]

\[ A' = \begin{pmatrix} f_1' & f_2' & f_3' \\ g_1' & g_2' & g_3' \end{pmatrix} \]

\[ A' A = \begin{pmatrix} f_1'^2 + f_2'^2 + f_3'^2 & f_1' g_1' + f_2' g_2' + f_3' g_3' \\ f_1' g_1' + f_2' g_2' + f_3' g_3' & g_1'^2 + g_2'^2 + g_3'^2 \end{pmatrix} \]

But:

\[ (A' A - \lambda I) \rightarrow \lambda \end{pmatrix} \rightarrow (A' \lambda A - \lambda I) \rightarrow (A' \lambda A - \lambda I) \]

So, we can solve only 2x2 matrix instead of 3x3 matrix, and then get the eigenvectors \( \mathbf{v}_i \) for \( A' A \) simply by the multiplication \( \mathbf{v}_i = \mathbf{A} \mathbf{v}_i \).
Get the New Image

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

\[ \omega_k = \Omega^T (\Gamma - \Psi) \text{ for } k=1,...,M \]

Vector \( \Omega = (\omega_1, \omega_2,...,\omega_M) \) represents input image in “face space”, where weight \( \omega_k \) – contribution of each eigenface.

Determine If the Image is a Face At All

1. The best case: on the subspace
2. Close enough
3. Too close – not a face

Original face

Projection

Recognition

\[ \text{Face class } (O_k) \text{ is the set of faces of one person} \]

To recognize we will find the minimum of \( e_k = ||O - O_k||^2 \), and if \( e_k \) is below some threshold \( \theta \), face belongs to person \( k \).

Otherwise the face is classified as “unknown”.

Face class example

recognized

unknown

Face space

recognized

Not a face, not used for training

Original face

Projection

Face class example

recognized

unknown

Face space

recognized

Not a face, not used for training
Learning

If the same unknown case is seen several times, calculate its characteristic weight pattern and incorporate it 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 a 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 a face.
Improving

1. Eliminating the Background
2. Scaling and Orientation
   • Pyramids
   • +/- 45° 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.)