Appearance Based Recognition

Dr. Gerhard Roth

Winter 2012

Recognition (Section 10.4)

- Given an image I, containing a single object and a database of images find the image in the database that is most similar to image I
- One possible way to recognize objects
  - Database has views of same object under different conditions
  - Input image is “close” to one of these database views
- Commonly used in face recognition systems
  - Database has number of faces (standard position)
  - Input image is a single face (standard position)

= ?

Assumptions in appearance recognition

1. Each image contains only a single object.
2. The objects are imaged by a fixed camera under weak perspective.
3. The images are normalized in size: that is the image frame is the minimum rectangle enclosing the largest appearance of this object.
4. The energy of the pixel values is normalized to one: $\sum_{i=1}^{N} \sum_{i=1}^{N} I(i,j)^2 = 1$
5. The object is completely visible and unconcluded in all images.

Comparing Images

- First transform 2D image into a 1D vector
  - A single N x N image X, becomes an N^2 dimensional vector
  - $x = [X_{11},X_{12},\ldots,X_{1N},X_{21},\ldots,X_{NN}]^T$
- Now given two images X_1 and X_2, and the two vectors x_1 and x_2 how do we compare them?
- One way, is to find distance between them according to some norm (usually L2)
  - $Dist(x_1, x_2) = \| x_1 - x_2 \|$ (just sum of squares of differences)
- If x1 and x2 are normalized ($\|x_1\| =1, \|x_2\|=1$)
  - Then x1 and x2 can be compared by correlation
  - This is the dot product $Dot(x_1, x_2) = x_1 \bullet x_2$
- If $Dist(x_1, x_2) = 0$ or $Dot(x_1, x_2)=1$ the x_1 and x_2 are identical

Euclidian Distance and Correlation

- In image processing language
  - Dist(x_1, x_2) is called Euclidian distance (L2 norm)
  - Dot(x_1, x_2) is called image correlation
- Given a single image y and a database of m images labeled x_1, x_2, …, x_m
- Want to find closest image in database to y?
  - Perform correlation and find largest result
  - Compute Euclidian distance and find smallest result
    - They both get the same answer (can be proven)
  - This will be the best match (see example program)
- How long does this take?
  - Time is proportional to m * N^2, where N^2 is number of pixels in the original image
  - Takes a long time since N^2 is large and so is m

Problems with this approach

- May need a very large database
  - Require a different image for different lighting conditions
  - Not much can be done about this issue in a simple way
    - Just hope that enough memory is available
- Often will require a lot of time
  - As in the previous slide to find the best match we need to do convolution against every image in the database
- Can not do much about size of database
  - This problem is intrinsic to the basic approach
- But we can decrease the execution time
  - By using the eigenspace or Principal Components Method
  - Sometimes called the PCA approach

Covariance of two variables

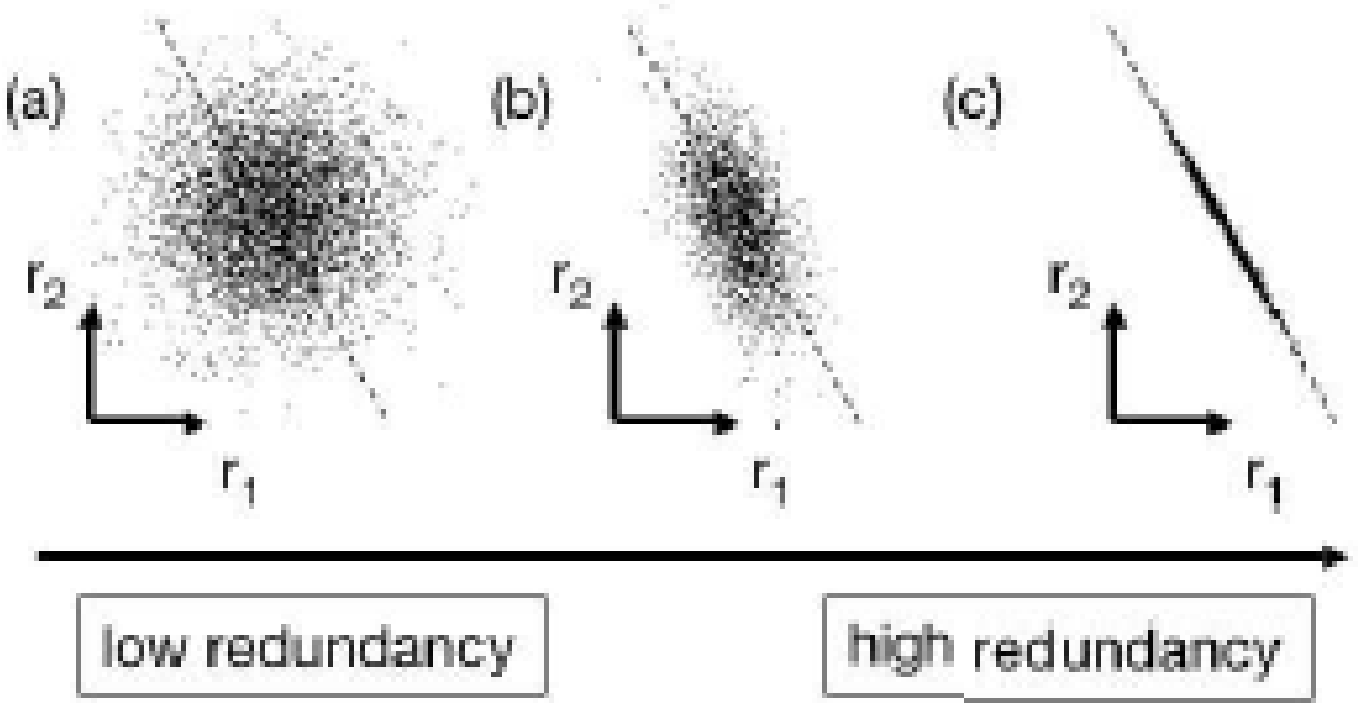
- Consider two variables x and y
- Covariance measures degree of linear relationship between x and y

$$\mathrm{cov}(x, y) = \frac{\sum_{i=1}^{i=n} (x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

- Now larger (smaller) cov(x,y) indicates a higher (lower) degree of linear relationship
  - Degree of linear relationship also called redundancy
- cov(x,y) can be plus or minus however always true that cov(x,y) = cov(y,x)
- This is a relationship between two variables

Covariance - Low and high redundancy

- Cov = 0 variables independent
- Cov > 0 increase together, Cov < 0 decrease together
- Cov high value – variables strongly dependent which means they have high redundancy



Covariance of a set of variables

- If we have a set of variables, say x,y, z there are a number of possible co-variances
- The associated co-variance matrix has three rows and three columns

$$C = \begin{bmatrix} \mathrm{cov}(x,x) & \mathrm{cov}(x,y) & \mathrm{cov}(x,z) \\ \mathrm{cov}(y,z) & \mathrm{cov}(y,y) & \mathrm{cov}(y,z) \\ \mathrm{cov}(z,x) & \mathrm{cov}(z,y) & \mathrm{cov}(z,z) \end{bmatrix}$$

- This matrix is real and symmetric
  - $C^T = C$
- In our case each image pixel is a variable!

Covariance matrix of the N^2 image

- First transform 2D image into a 1D vector
  - N x N image $\mathbf{x_i}$ becomes an N^2 dimensional vector
  - Make each image a column in an N^2 by M matrix
  - $\mathbf{x_i} = [X_{11},X_{12},\ldots,X_{1N},X_{21},\ldots,X_{NN}]^T$
- An image is a point in N^2 dimensional space
  - Each co-ordinate usually has a range of 0 to 255 (8 bits)
- Matrix X has m columns, one for each image
  - Matrix **X** has dimensions of N^2 by M, where there are M images
- Compute the co-variance matrix $Q = X X^T$
  - Q is of dimension N^2 by N^2
- Compute eigenvectors and eigenvalues
  - Covariance matrix is real and symmetric
  - Eigenvectors are orthonormal
  - There are N^2 of these eigenvectors

$$Q^T = Q$$

- What can we do with these eigenvectors?

Eigenvectors are basis of the N^2 space

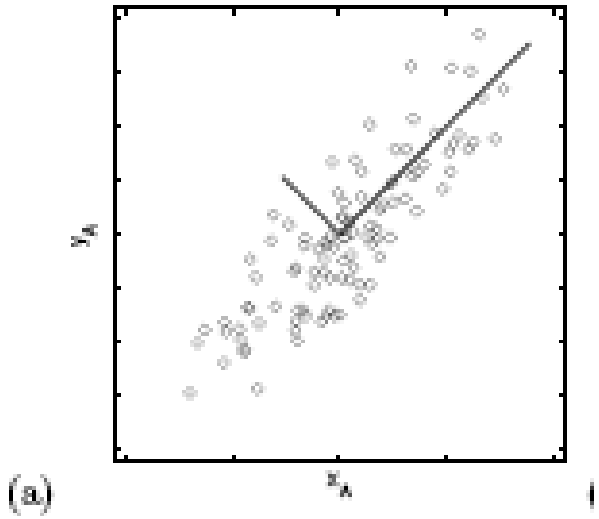
- An image is a point in N^2 dimensional space
  - Each co-ordinate usually has a range of 0 to 255 (8 bits)
- Eigenvectors of co-variance matrix are basis
  - Let $x_1, \ldots, x_{N^2}$ be the vectors for each of these M images
  - We can write each image $\mathbf{x_j}$ in terms of these eigenvectors
- Eigenvectors of this N^2 space are $e_1, \ldots, e_{N^2}$
- Then $x_j = \bar{x} + \sum_{i=1}^{i=N^2} g_{ji} e_i$ where $g_j = [g_1, g_2, \ldots, g_{N^2}]^T$ are the vector of components of x_j in this new basis called the eigenspace

Eigenspace – why use it?

- We are computing a new basis for the N^2 dimensional space of these images
  - This basis will have some advantages
- We will not need to use all N^2 co-ordinates in this new basis to represent an images
  - Instead we will use k basis vectors where $k << N^2$
- Why does this work and how is it done?
  - Works because of characteristics of eigenvectores
  - And also because of image redundancy
- Also called Principal Components Decomposition
  - We will use a set of pdf notes on PCA decomposition

•Principal components and values

- Eigenvectors of covariance matrix represent an orthogonal basis of this space
  - Also called the principal components
- Represent the directions of most variance
  - Eigenvalues of the eigenvectors is the amount of variance



PCA on the Co-Variance matrix

- Algorithm operates on co-variance matrix
  1. Select a normalized direction in the N^2 space in which the variance is maximized. Save this vector as p_1
  2. Find another direction in which the variance is maximized, but because of orthogonality we must restrict the search to directions perpendicular to all previous directions. Save this vector as pi.
  3. Repeat the process until we have all all N^2 vectors.
- Resulting set of vectors are called the principal components and are equal to the eigenvectors of the co-variance matrix
- Eigenvalues of these eigenvectors is the amount of variance for this eigenvector

Eigenspace

- Each image is a point in eigenspace
  - To compare two images means computing their distance in this new eigenspace basis (requires some dot products)
  - This gets same result as correlation and convolution
- Eigenspace is just a change of basis of the original N x N dimensional image space
  - How does this help us in any way?
- There is often redundancy in the images
  - Some pixels have a limited range of values
- Eigenspace or PCA finds this redundancy
  - The most redundant dimensions have largest eignvalues and vice-versa so we can ignore other dimensions

Principle components and values

- The eignenvalues of the co-variance matrix are the principal values and the eigenvectors are the principal components
- A large eigenvalue means that that this direction is more “principal” than others
- You can order the principal components (eigenvectors) by their principal values (eigenvalues)
- You can ignore eigenvectors that have a very small eigenvalue
  - They do not contribute anything important
  - This is an approximation but it does not cause a significant error when ignoring small eigenvalues

Efficient eigenvector computation

- An image is a point in N^2 dimensional space
  - Each co-ordinate usually has a range of 0 to 255 (8 bits)
  - Matrix **X** has M columns, one for each image and N^2 rows
  - Matrix **X** has dimensions of N^2 by M,
- If N^2 is 100 x 100 = 10,000 can not easily take eigenvectors of such a large matrix
  - One idea is to use smaller images
  - Another idea is to use “trick” for computing eigenvectors
- Eigenvectors of covariance matrix $X\ X^T$ can be computed from the eigenvectors of matrix $X^T\ X$
  - But $X^T\ X$ has dimension of M by M, not N^2 by N^2
- M images means there are only M-1 independent eigenvectors for the N^2 co-variance matrix
  - If $M << N^2$ this is a big speedup (number images << number pixels)

Efficient eigenvector computation

Define

$C = XX^T$ dimension N^2 by N^2 where N^2 is number pixels

$L = X^T X$ dimension M by M where M is #images

Let v be an eigenvector of L : $Lv = \lambda v$

Then Xv is eigenvector of C: $C(Xv) = \lambda(Xv)$

Proof:

$$\begin{aligned} C(Xv) &= XX^T(Xv) \\ &= X(X^T Xv) \\ &= X(Lv) \\ &= X\lambda v \\ &= \lambda(Xv) \end{aligned}$$

Eigenspace

- Find m non-zero eignenvalues of the matrix $X^T X$,

$$\lambda_1, \ldots, \lambda_M \qquad \lambda_1 \geq \lambda_2 \geq \ldots \lambda_M$$

- Each image $x_j \approx \bar{x} + \sum_{i=1}^{i=M} g_{ji} e_i$

- We can ignore remaining N^2 - M components
- Each image x_j is represented by a point g_j^T in M dimensional space where $M << N^2$
- Here M is number of images, N is the image size
  - So M is usually << N x N (typically 1000 < 100 x 100)
- N^2 is cost of correlation and convolution while same comparison in eigenspace now costs only O(M)
  - So eigenspace representation saves time in image comparison

Algorithm EIGENSPACE_LEARN

We consider the assumptions stated at the beginning of this section valid; moreover, we assume a fixed camera, fixed illumination conditions, and images of $N \times N$ pixels.

1. For each object o to be represented, $o = 1 \ldots O$:
   (a) place the object on the turntable;
   (b) acquire a set of n images by rotating the turntable by $\frac{360^\circ}{n}$ each time;
   (c) in all images, make sure to adjust the background so that the object can be easily segmented from the background;
   (d) segment the object from the background (see Exercise 10.9);
   (e) normalize the images in scale and energy as stated in the assumptions;
   (f) represent the normalized images as vectors, $\mathbf{x}_p^o$, where p is the rotation index, $p = 1, \ldots, n$.
2. Compute the average image vector, $\bar{\mathbf{x}}$, of the complete database $\left\{\mathbf{x}_1^1, \ldots \mathbf{x}_n^1, \mathbf{x}_1^2, \ldots \mathbf{x}_n^2, \ldots \mathbf{x}_n^O\right\}$.
3. Form the $N^2 \times N^2$ covariance matrix, $Q = XX^\top$, with $X = \left[\mathbf{x}_1^1 | \mathbf{x}_2^1 | \ldots \mathbf{x}_n^1 | \mathbf{x}_1^2 | \ldots \mathbf{x}_n^O\right]$.
4. Compute the eigenvalues of Q, keep the first k largest eigenvalues and the associated eigenvectors, $\mathbf{e}_1, \ldots \mathbf{e}_k$.
5. for each object, o:
   (a) compute the k-dimensional eigenspace points corresponding to the n images:
   $$\mathbf{g}_p^o = [\mathbf{e}_1 | \ldots \mathbf{e}_k] \, (\mathbf{x}_p^o - \bar{\mathbf{x}});$$
   (b) store the discrete eigenspace curve $\left\{\mathbf{g}_1^o, \ldots, \mathbf{g}_p^o, \ldots, \mathbf{g}_n^o\right\}$, as the representation of object o.

The output is a set of O discrete curves in the k-dimensional eigenspace, each representing a 3-D object.

☞ As detailed in the Appendix

Eigenspace Learn

- Focus on steps 2 to 5
  - This process is done once only
  - If the set of images changes it must be redone
- We can use fewer than M eigenvectors (say K)
- Choose K eigenvectors such that

$$\sum_{i=1}^{i=K} \lambda_i \geq 0.95 \sum_{i=1}^{i=M} \lambda_i$$

  - So that the error in the approximation is not too large
- The value of K depends on the redundancy of the images, more redundant the smaller K, the less redundant the closer K is to M
  - Sum of all M eigenvalues represents the variability of the entire data set
  - If K of them accounts for 95% of the variability then using only K eigenvectors as a basis is a very good approximation

Eigenspace Identify

Algorithm EIGENSPACE_IDENTIF

The input is a $N \times N$ image, I, of one of the objects in the database. The image I must satisfy the assumptions stated at the beginning of this section and acquired so that the object can be easily segmented from the background. We assume the same illumination conditions and camera position adopted in EIGENSPACE_LEARN.

1. Segment the object from the background.
2. Normalize I in scale and energy, and represent the normalized image as a vector, $\mathbf{i}$.
3. Compute the k-dimensional eigenspace point corresponding to $\mathbf{i}$:

$$\mathbf{g} = [\mathbf{e}_1 | \ldots \mathbf{e}_k] \, (\mathbf{i} - \bar{\mathbf{x}}),$$

   where $\bar{\mathbf{x}}$ is the average image vector of the whole database.
4. Find the eigenspace point, $\hat{\mathbf{g}}$, created by EIGENSPACE_LEARN, closest to $\mathbf{g}$.

The output is the object associated to the curve on which $\hat{\mathbf{g}}$ lies; that is, the identity of the object in I.

Eigenspace Identify

- Take the input image and perform a dot product with the compressed eigenbasis
  - Get a new vector of K dimensions relative to this basis
  - Compare it to the stored eigenvectors
- Closest vector in eigenspace is the best match
  - Best in the same sense as correlation or convolution
- But this depends on the value of K
  - As you decrease you get a more approximate results
  - At some point if K is too small you will get an inaccurate answer
- Only experimental verification can be used to tell us what value of K is good enough
  - Choose k so that all eigenvalues sum to 95% is agood guide
- Note that that the more redundancy (the more similar all the images) the smaller the value of K
  - This works out well for faces which have a lot of redundancy
  - Also closest point speed is exponential time in K, so any reduction helps

Eigenspace reconstruction

- q is number of eigenvectors used (our k)
- Image for each eigenvector is an eigenimage

q=1

q=2

q=4

q=8

q=16

q=32

q=64

q=100

Original Image

Eigenspace limitations

- Must be able to segment the object
  - Segmenting a face is easier than other objects
- Will have problems if lighting changes between learned images and images to be recognized (lighting can be dealt with)
- Will be problems if learned images taken at different scale, orientation or viewpoint than the image to be recognized (very hard to fix)
- Makes a difference only for redundant images, where some parts of the images are very similar to each other
  - If images are all very different you gain nothing by this change of basis involved in eigenspace
  - But for many search applications there is redundancy!

Eigenspace limitations

- For large number of images in the database M you need to compute eigenvectors of a very large matrix
- So if $k \approx N^2$ then no simple speedup
- This only speeds up closest image computation
  - Still need other indexing algorithms to solve the closest point problem efficiently (such as k-d trees, etc.)
- Eigenspace/PCA an important image processing tool
  - Idea widely used in face recognition and object recognition
  - Also can be used in appearance based recognition
- Can also be used for compression
  - Look at eigenspace reconstruction slide
  - Can represent each face by a small number of eigenvectors
  - Not identical to original image but often very close
- PCA is a dimensionality reduction method