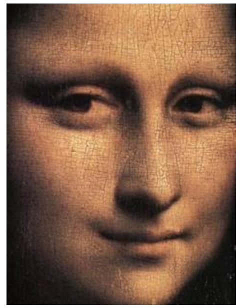
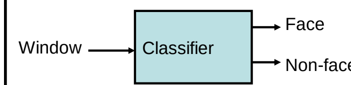
Face Detection and Recognition

Reading: Chapter 18.10
and, optionally,
“Face Recognition using
Eigenfaces” by M. Turk and
A. Pentland



Face Detection Problem

- Scan window over image
- Classify window as either:
  - Face
  - Non-face



Face Detection in most Consumer Cameras and Smartphones for Autofocus

The Viola-Jones Real-Time Face Detector

P. Viola and M. Jones, 2004

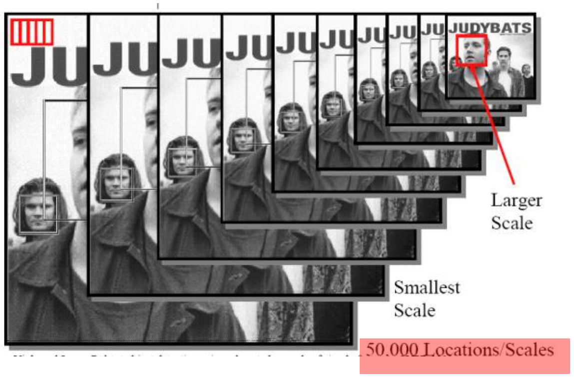
Challenges:

- Each image contains 10,000 – 50,000 locations and scales where a face may be
- Faces are rare: 0 - 50 per image
  - >1,000 times as many non-faces as faces
- Want a very small # of false positives: 10^{-6}

Use Machine Learning to Create a 2-Class Classifier

- Training Data (grayscale)
  - 5,000 faces (frontal)
  - 10^8 non-faces
- Faces are normalized
  - – Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose (rotation both in plane and out)

Use Classifier at *All* Locations and Scales



Building a Classifier

- Compute *lots* of very simple features
- Efficiently choose best features
- Each feature is used to define a “weak classifier”
- Combine weak classifiers into an **ensemble classifier** based on **boosting**
- Learn multiple ensemble classifiers and “chain” them together to improve classification accuracy

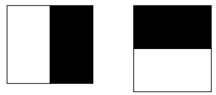
Computing Features

- At each position and scale, use a sub-image (“window”) of size 24 x 24
- Compute multiple candidate features for each window
- Want to rapidly compute these features

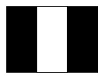
Features

- 4 feature types (similar to “Haar wavelets”):

Two-rectangle

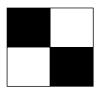


Three-rectangle

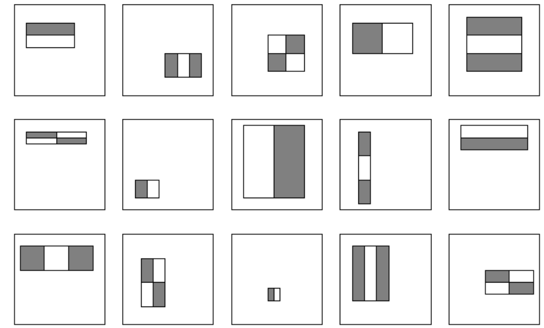


Value = ∑ (pixels in white area) - ∑ (pixels in black area)

Four-rectangle



Huge Number of Features



160,000 features for each window!

Computing Features Efficiently: **The Integral Image**

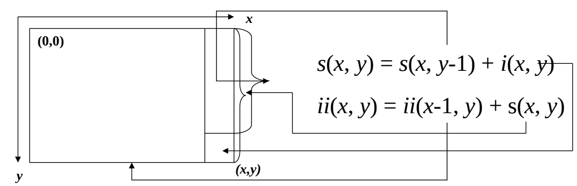
- aka “Summed Area Table”
- Intermediate representation of the image
  - Sum of all pixels above and to left of (x, y) in image i:

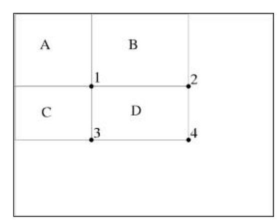
$$ii(x,y) = \sum_{x' \leq x,\, y' \leq y} i(x',y')$$

- Computed in one pass over the image:

$$ii(x, y) = i(x, y) + ii(x\text{-}1, y) + ii(x, y\text{-}1) + ii(x\text{-}1, y\text{-}1)$$

Using the Integral Image





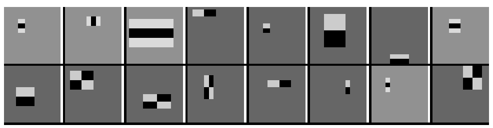
- With the integral image representation, we can compute the value of ***any*** rectangular sum in constant time
- For example, the integral sum in rectangle D is computed as:

$$ii(4) + ii(1) – ii(2) – ii(3)$$

Features as Weak Classifiers

- Given window x, feature detector f_t, and threshold θ_t, construct a weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$



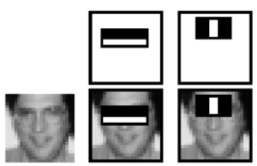
Boosting

- Boosting is a class of ensemble methods for sequentially producing multiple weak classifiers, where each classifier is dependent on the previous ones
- Make examples misclassified by previous classifier ***more important*** in the next classifier
- Combine weak classifiers linearly into combined classifier:

$$C(x) = \theta\left(\sum_i \overbrace{\alpha_i}^{\text{weight}} \overbrace{h_i(x)}^{\text{weak classifier}} + b\right)$$

Boosting

How to select the best features?



How to learn the classification function?

$$C(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots.$$

AdaBoost Algorithm

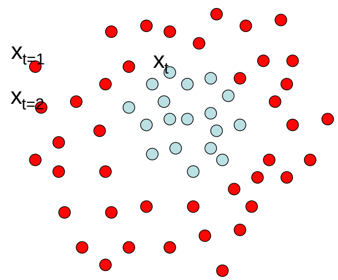
Given a set of training windows labelled +/-, initially give equal weight to each training example

Repeat T times

1. Select best weak classifier (min total weighted error on all examples)
2. Increase weights of examples misclassified by current weak classifier

- Each round greedily selects the best feature given all previously selected features
- Final classifier weights weak classifiers by their accuracy

AdaBoost Example



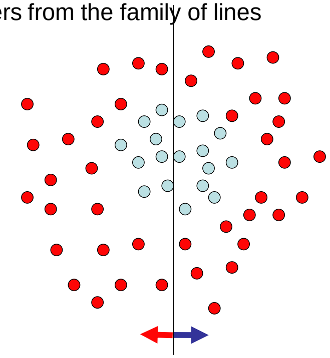
Each data point has a class label:

$$y_t = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$$

and a weight:
$w_t = 1$

Example

Weak learners from the family of lines

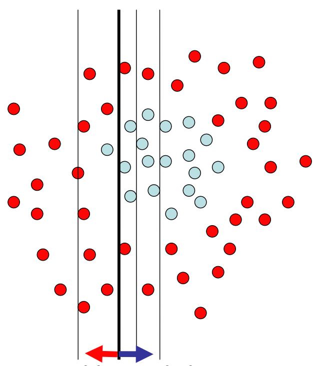


Each data point has a class label:

$$y_t = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$$

and a weight:
$w_t = 1$

Example



Each data point has a class label:

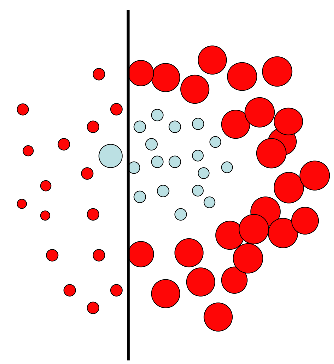
$$y_t = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$$

and a weight:
$w_t = 1$

This one is best

This is a '**weak classifier**': It performs slightly better than chance

Example



Each data point has a class label:

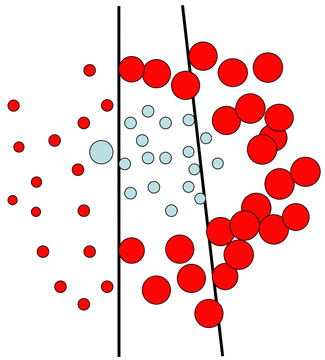
$$y_t = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$$

Update the weights:

$w_t \leftarrow w_t \exp\{-y_t h_t\}$

Re-weight misclassified examples

Example



Each data point has a class label:

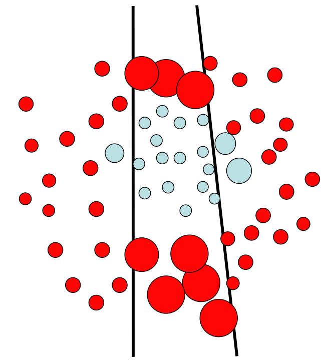
$y_t = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$

Update the weights:

$w_t \leftarrow w_t \exp\{-y_t h_t\}$

Select 2[nd] weak classifier

Example



Each data point has a class label:

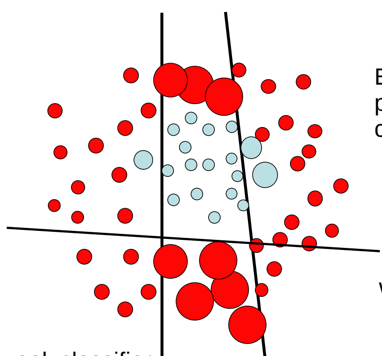
$y_t = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$

Update the weights:

$w_t \leftarrow w_t \exp\{-y_t h_t\}$

Re-weight misclassified examples

Example



Each data point has a class label:

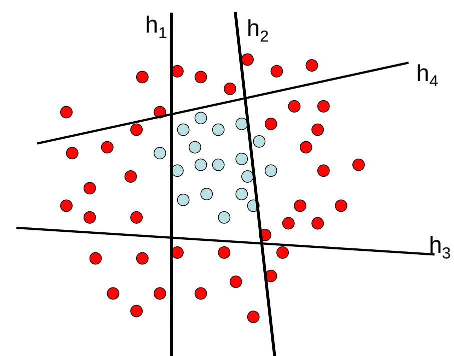
$y_t = \begin{cases} +1 & (\ ) \\ -1 & (\ ) \end{cases}$

Update the weights:

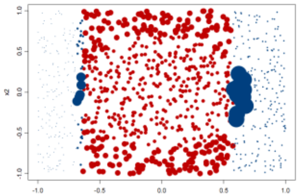
$w_t \leftarrow w_t \exp\{-y_t h_t\}$

Select 3[rd] weak classifier

Example

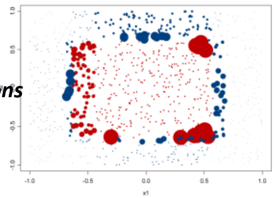


The final (non-linear) classifier is the combination of all the weak (linear) classifiers

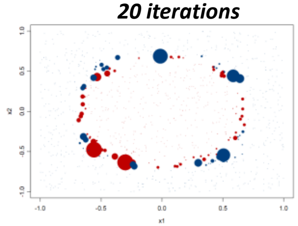


Classifications (colors) and weights (size) after *1 iteration* Of AdaBoost

3 iterations



20 iterations



from Elder, John. From Trees to Forests and Rule Sets - A Unified Overview of Ensemble Methods. 2007.

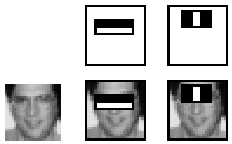
Slide by T. Holloway

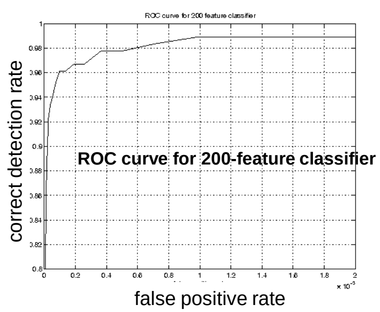
AdaBoost - Adaptive Boosting

- Learn a single simple classifier
- Classify the data
- Look at where it makes errors
- Re-weight the data so that the inputs where we made errors get higher weight in the learning process
- Now learn a 2nd simple classifier on the weighted data
- Combine the 1st and 2nd classifier and weight the data according to where they make errors
- Learn a 3rd classifier based on the weighted data
- … and so on until we learn *T* simple classifiers
- Final classifier is the weighted combination of all *T* classifiers

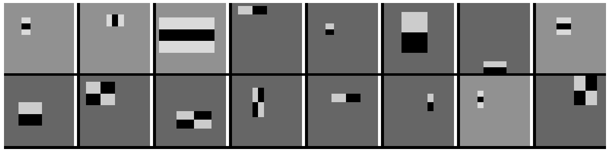
Example Classifier

- A classifier with *T*=200 rectangle features was learned using AdaBoost
- 95% correct detection on test set with 1 face in 14,084 false positives
- Not quite competitive

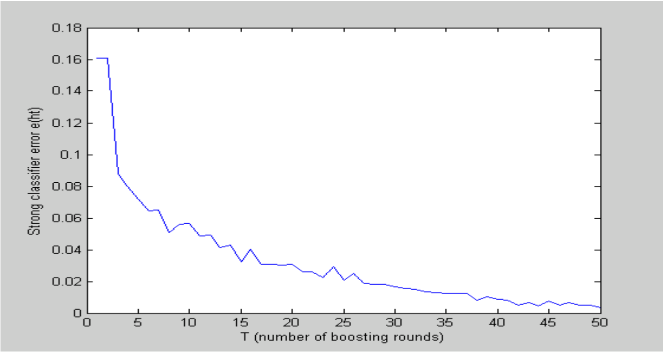




Learned Features



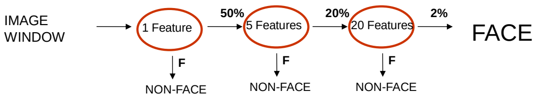
Classifier Error is Driven Down as *T* increases, but Cost Increases



Cascading

- Improve accuracy and speed by "cascading" multiple classifiers together
- Start with a simple (*T* small) classifier that rejects many of the negative windows while detecting almost all positive windows
- Positive results from the first classifier triggers the evaluation of a second (more complex; bigger *T*) classifier, and so on
  - Each classifier is trained with the false positives from the previous classifier
- A negative outcome at any point leads to the immediate rejection of the window

Cascaded Classifier



- A *T*=1 feature classifier achieves 100% detection rate and about 50% false positive rate
- A *T*=5 feature classifier achieves 100% detection rate and 40% false positive rate (20% cumulative)
  - using data from previous stage
- A *T*=20 feature classifier achieve 100% detection rate with 10% false positive rate (2% cumulative)

Results

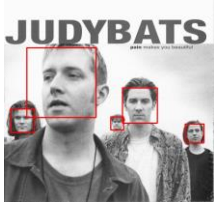
Structure of the Detector

- 38 layer cascade
- 6,060 features
- Training time: “weeks”

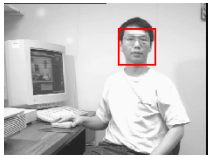
Layer number	1	2	3 to 4	5 to 38
Number of feautures	2	10	50	-
Detection rate	100%	100%	-	-
Rejection rate	50%	80%	-	-

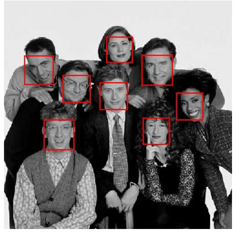
Results

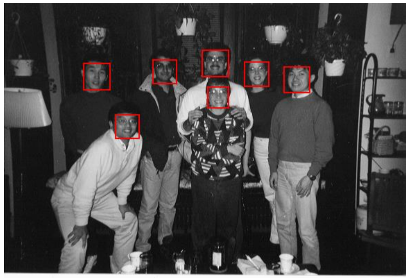
- Notice detection at multiple scales



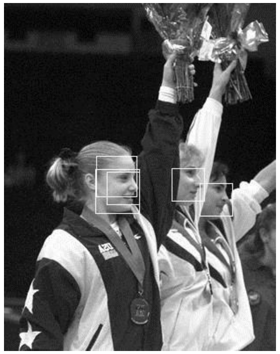
Results



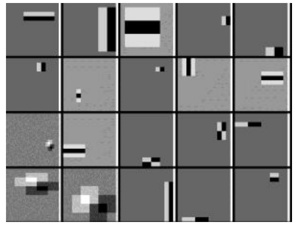




Profile Detection



Profile Features



Demos

- http://flashfacedetection.com/
- http://flashfacedetection.com/camdemo2.html

Face Recognition Problem

database

Query face

Face Verification Problem

- Face Verification (1:1 matching)

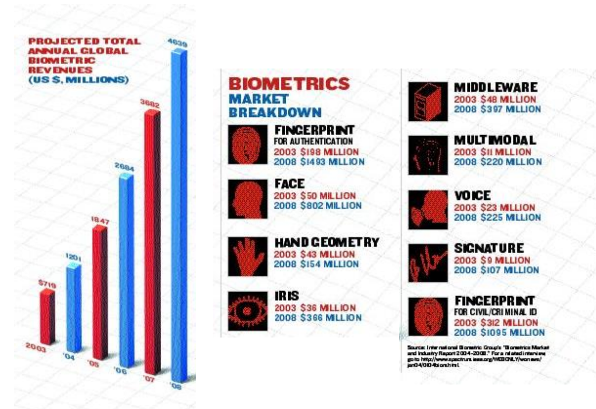
- Face Recognition (1:N matching)

Application: Access Control

www.viisage.com

www.visionics.com

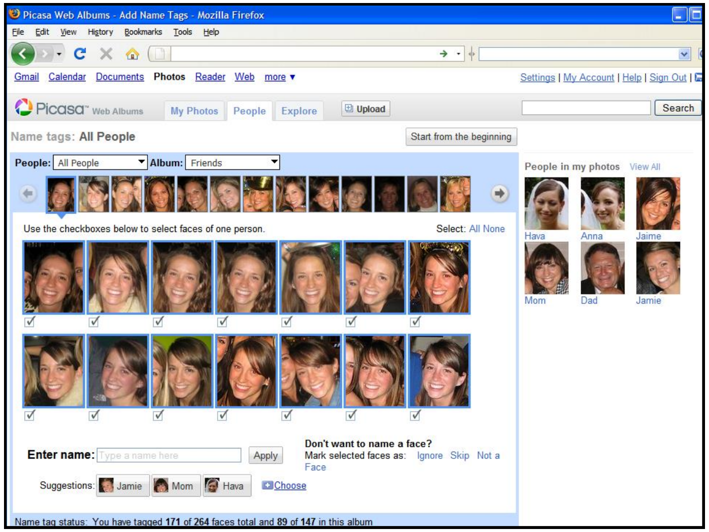
Biometric Authentication



Application: Video Surveillance

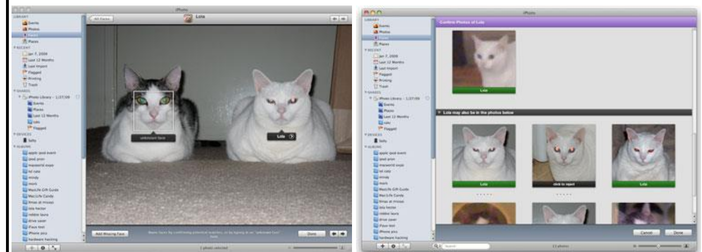
www.facesnap.de

Application: Autotagging Photos in Facebook, Flickr, Picasa, iPhoto, …



iPhoto 2009

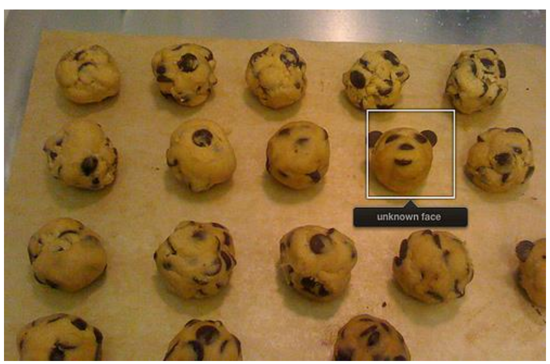
- Can be trained to recognize pets!



http://www.maclife.com/article/news/iphotos_faces_recognizes_cats

iPhoto 2009

- Things iPhoto thinks are faces



Why is Face Recognition Hard?

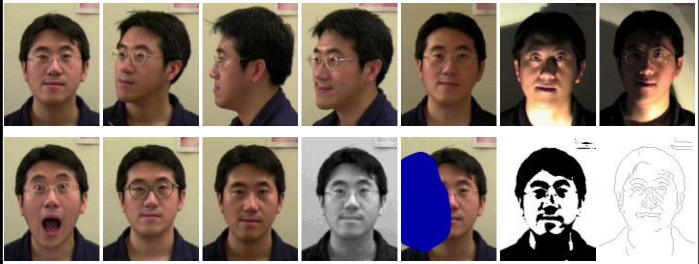
The many faces of Madonna

Recognition should be **Invariant** to

- Lighting variation
- Head pose variation
- Different expressions
- Beards, disguises
- Glasses, occlusion
- Aging, weight gain
- …

Intra-class Variability

- Faces with intra-subject variations in pose, illumination, expression, accessories, color, occlusions, and brightness



Inter-class Similarity

- Different people may have very similar appearance

www.marykateandashley.com

news.bbc.co.uk/hi/english/in_depth/americas/2000/us_elections

Twins

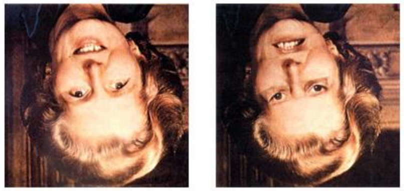
Father and son

Blurred Faces ***are*** Recognizable

Blurred Faces *are* Recognizable

Michael Jordan, Woody Allen, Goldie Hawn, Bill Clinton, Tom Hanks, Saddam Hussein, Elvis Presley, Jay Leno, Dustin Hoffman, Prince Charles, Cher, and Richard Nixon. The average recognition rate at this resolution is one-half.

Upside-Down Faces are Recognizable



The "Margaret Thatcher Illusion", by Peter Thompson

Context is Important

P. Sinha and T. Poggio, I think I know that face, *Nature* **384**, 1996, 404.

Face Recognition Architecture

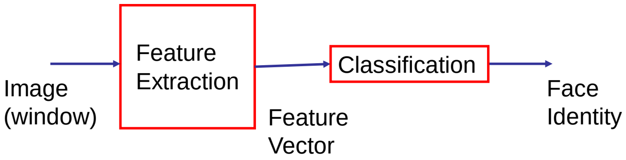
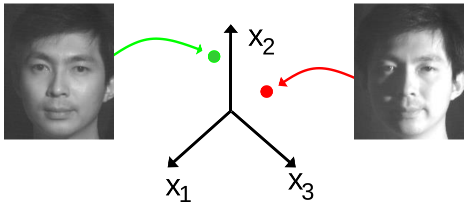


Image as a Feature Vector

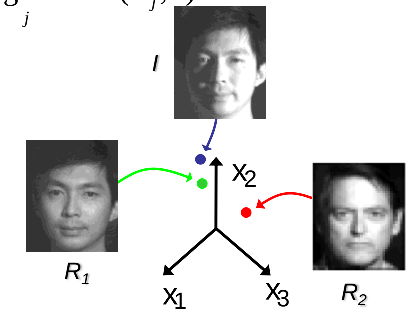


- Consider an n-pixel image to be a point in an n-dimensional "image space," $x \in \mathfrak{R}^n$
- Each pixel value is a coordinate of x
- Preprocess images so faces are cropped and roughly aligned (position, orientation, and scale)

Nearest Neighbor Classifier

$\{ R_j \}$ is a set of training images of frontal faces

$$ID = \arg\min_j dist(R_j, I)$$



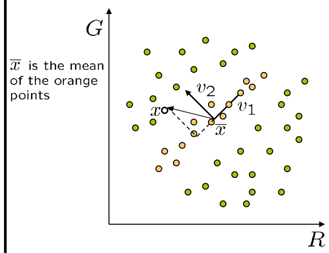
Key Idea

- Expensive to compute nearest neighbor when each image is big (n dimensional space)
- Not all images are very likely – especially when we know that every image contains a face. That is, **images of faces are highly correlated, so compress them into a low-dimensional, linear subspace** that retains the key appearance characteristics

Eigenfaces (Turk and Pentland, 1991)

- The set of face images is clustered in a "subspace" of the set of all images
- Find best subspace to reduce the dimensionality
- Transform all training images into the subspace
- Use nearest-neighbor classifier to label a test image

Linear Subspaces

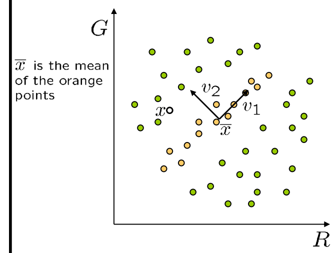


convert **x** into $\mathbf{v_1}$, $\mathbf{v_2}$ coordinates:

$$\mathbf{x} \rightarrow ((\mathbf{x} - \overline{x}) \cdot \mathbf{v_1}, (\mathbf{x} - \overline{x}) \cdot \mathbf{v_2})$$

- What does the $\mathbf{v_2}$ coordinate measure?
  - Distance to line defined by $\mathbf{v_1}$
- What does the $\mathbf{v_1}$ coordinate measure?
  - Position along the line

Dimensionality Reduction



- We can represent the orange points with *only* their $\mathbf{v_1}$ coordinates (since $\mathbf{v_2}$ coordinates are all essentially 0)
- This makes it much cheaper to store and compare points
- A bigger deal for higher dimensional problems

Principal Component Analysis (PCA)

- Problems arise when performing recognition in a high-dimensional space ("**curse of dimensionality**")
- Significant improvements can be achieved by first mapping the data into a ***lower-dimensional sub*****space**

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_N \end{bmatrix} --> reduce\ dimensionality --> y = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_K \end{bmatrix} \quad (K << N)$$

- **The goal of PCA is to reduce the dimensionality of the data while retaining the important variations present in the original data**

Principal Component Analysis (PCA)

- Dimensionality reduction implies information loss
- How to determine the best lower dimensional subspace?
- Maximize information content in the compressed data by **finding a set of *k* orthogonal vectors that account for as much of the data's variance as possible**
  - Best dimension = direction in *n*-D with max variance
  - 2^{nd} best dimension = direction orthogonal to first and max variance

Principal Component Analysis (PCA)

- The best low-dimensional space can be determined by the "best" eigenvectors of the covariance matrix of the data, i.e., the eigenvectors corresponding to the largest eigenvalues – also called "**principal components**"
- Can be efficiently computed using Singular Value Decomposition (SVD)

Algorithm

- Each input image, X_i , is an *n*D column vector of all pixel values (in raster order)
- Compute "average face image" from all *M* training images of all people:

$$A = \frac{1}{M}\sum_{i=1}^{M} X_i$$

- Normalize each training image, X_i, by subtracting the average face:

$$Y_i = X_i - A$$

Algorithm

- Stack all training images together

n x *M* matrix

$$\mathbf{Y} = [Y_1 Y_2 \ldots Y_M]$$

- Compute *n* x *n* Covariance Matrix

$$\mathbf{C} = \mathbf{Y}\mathbf{Y}^{\mathrm{T}} = \frac{1}{M}\sum_i Y_i Y_i^{\mathrm{T}}$$

Algorithm

- Compute eigenvalues and eigenvectors of **C** by solving

$$\lambda_i u_i = \mathbf{C} u_i$$

where the eigenvalues are

$$\lambda_1 > \lambda_2 > \ldots > \lambda_n$$

and the corresponding eigenvectors are

$u_1, u_2, \ldots, u_n$

Algorithm

- Each u_i is an n x 1 eigenvector called an "eigenface" (to be cute!)
- Each u_i is a direction/coordinate in "face space"

$$Y_i = w_1 u_1 + w_2 u_2 + \ldots + w_n u_n$$

$$X_i = \sum_{i=1}^{n} w_i u_i + A$$

- Image is exactly reconstructed by a linear combination of *all* eigenvectors

Algorithm

- Reduce dimensionality by using only the **best *k*** << n eigenvectors (i.e., the ones corresponding to the largest k eigenvalues

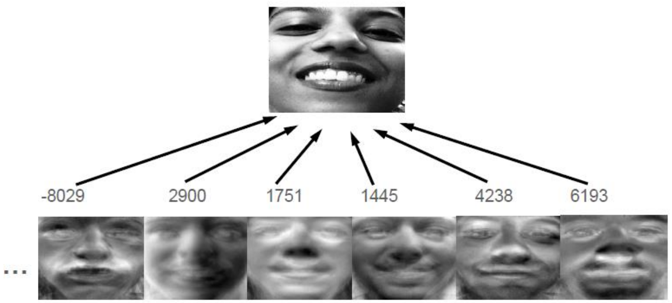
$$X_i \approx \sum_{i=1}^{k} w_i u_i + A$$

- Each image X_i is approximated by a set of k "weights" [w_{i1} , w_{i2}, …, w_{ik}] = $\boldsymbol{W}_i$ where

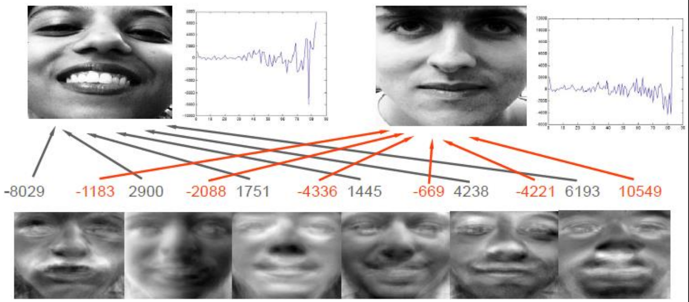
$$w_{ij} = u_j^{\mathrm{T}} (X_i - A)$$

Eigenface Representation

Each face image is represented by a weighted combination of a small number of "component" or "basis" faces



Eigenface Representation



Using Eigenfaces

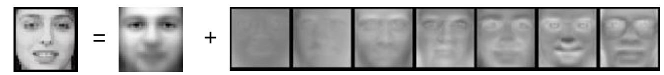
- **Reconstruction** of an image of a face from a set of weights
- **Recognition** of a person from a new face image

Face Image Reconstruction

- Face **X** in "face space" coordinates:

$$\mathbf{x} \rightarrow [\mathbf{u}_1^{\mathrm{T}}(\mathbf{x}-\mu), \ldots, \mathbf{u}_k^{\mathrm{T}}(\mathbf{x}-\mu)]$$
$$= \quad w_1, \ldots, w_k$$

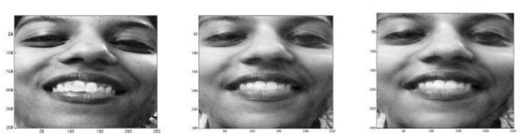
- Reconstruction:



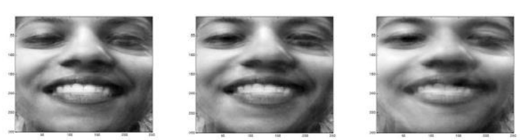
$$\hat{\mathbf{X}} \quad = \quad A \quad + \quad w_1 u_1 + w_2 u_2 + w_3 u_3 + w_4 u_4 + \ldots$$

Reconstruction

The more eigenfaces you use, the better the reconstruction, but even a small number gives good quality for matching



82 70 50



30 20 10

Eigenfaces Recognition Algorithm

Modeling (Training Phase)

1. Given a collection of n labeled training images
2. Compute mean image, A
3. Compute k eigenvectors, u_1 , …, u_k, of covariance matrix corresponding to k largest eigenvalues
4. Project each training image, X_i, to a point in k-dimensional "face space:"

$$\text{for } j = 1, \ldots, k \text{ compute } w_{ij} = u_j^{\mathrm{T}}(X_i - A)$$

X_i projects to $\boldsymbol{W}_i = [w_{i1}, w_{i2}, \ldots, w_{ik}]$

Eigenfaces Algorithm

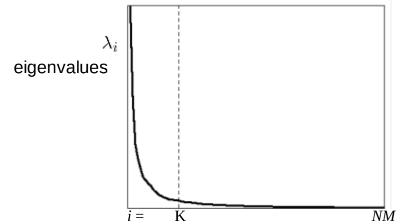
Recognition (Testing Phase)

1. Given a test image, *G*, project it into face space

$$\text{for } j = 1, \ldots, k \text{ compute } w_j = u_j^{\mathrm{T}}(G - A)$$

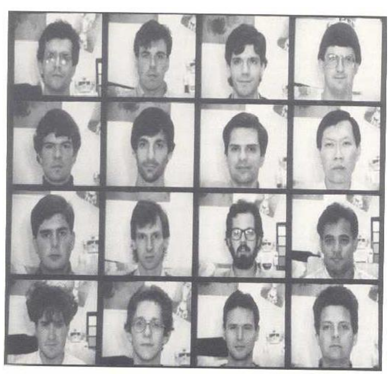
2. Classify it as the class (person) that is closest to it (as long as its distance to the closest person is "close enough")

Choosing the Dimension *K*



- How many eigenfaces to use?
- Look at the decay of the eigenvalues
  - the eigenvalue tells you the amount of variance "in the direction" of that eigenface
  - ignore eigenfaces with low variance

Example: Training Images

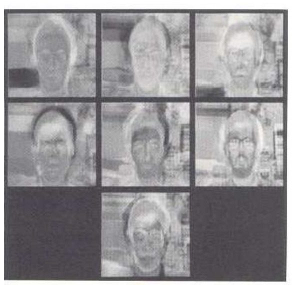


Note: Faces must be approximately registered (translation, rotation, size, pose)

[Turk & Pentland, 2001]

Eigenfaces

Average Image, *A*



7 eigenface images

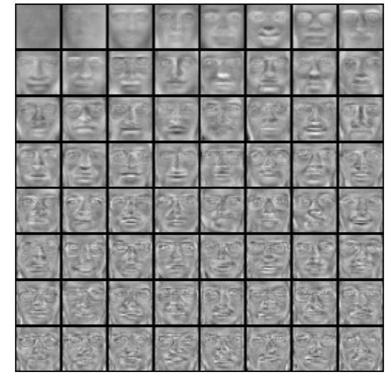
Example

Training images

Example

Average: A

Top eigenvectors: $u_1, \ldots u_k$

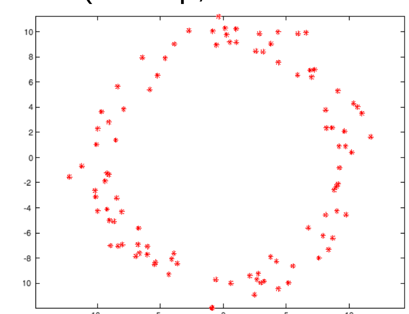


Experimental Results

- Training set: 7,562 images of approximately 3,000 people
- k = 20 eigenfaces computed from a sample of 128 images
- Test set accuracy on 200 faces was 95%

Limitations

- PCA assumes that the data has a Gaussian distribution (mean μ, covariance matrix **C**)



The shape of this dataset is not well described by its principal components

Limitations

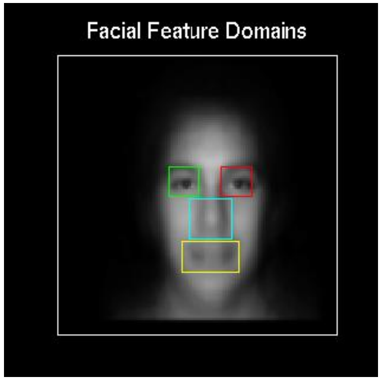
- Background (de-emphasize the outside of the face – e.g., by multiplying the input image by a 2D Gaussian window centered on the face)
- Lighting conditions (performance degrades with light changes)
- Scale (performance decreases quickly with changes to head size); possible solutions:
  - multi-scale eigenspaces
  - scale input image to multiple sizes
- Orientation (performance decreases but not as fast as with scale changes)
  - plane rotations can be handled
  - out-of-plane rotations are more difficult to handle

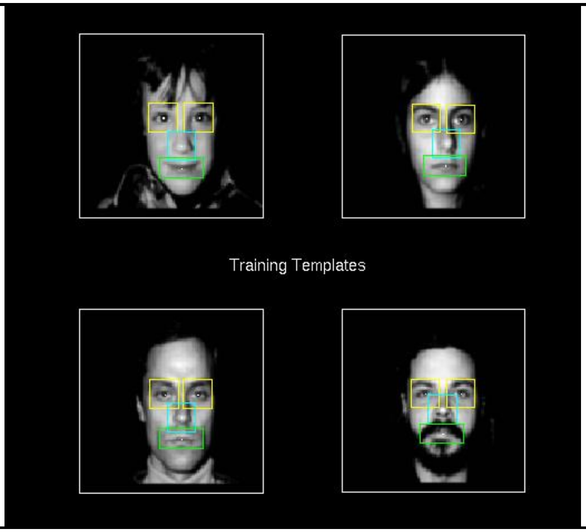
Limitations

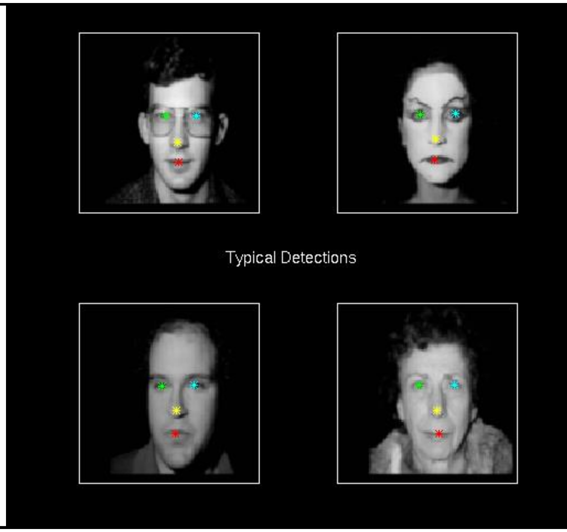
- Not robust to misalignment

Extension: Eigenfeatures

- Describe and encode a set of facial features: eigeneyes, eigennoses, eigenmouths
- Use for detecting facial features







Recognition using Eigenfeatures