Face

Announcements

- Survey (47 students participated in the survey)

  -(Lab-session-week-1, Lab-session-week-2). 9

  - (Lab-session-week-2, Lab-session-week-3). 38

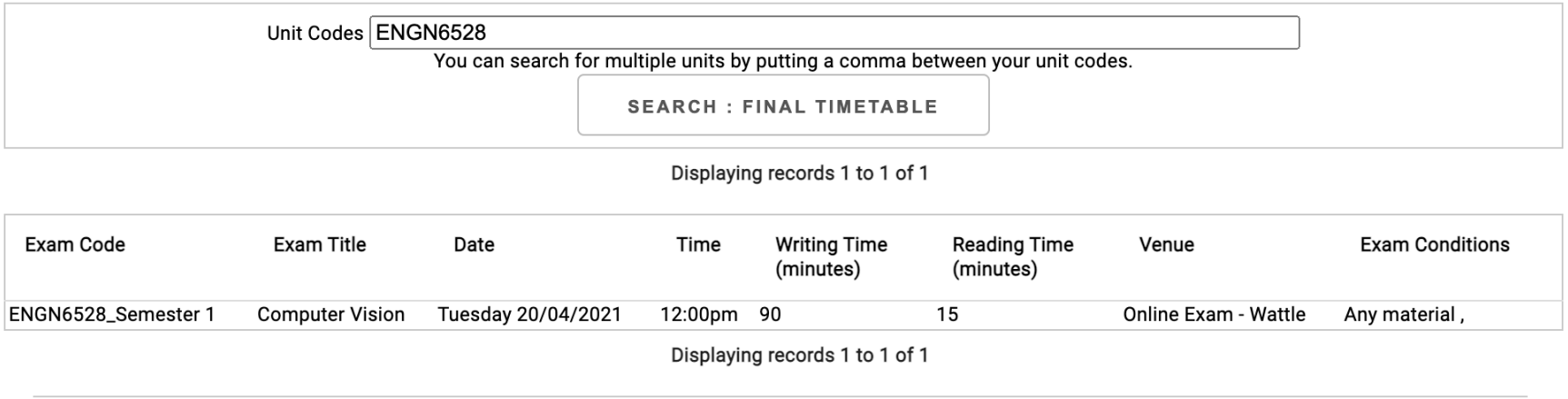
- Lab-Assignment-2 has been released.

  Due date: May 2nd, 23:59pm

- Tutorial on Lab assignment 2: Wed. 4:00pm

Announcement

- Mid-term Quiz



Unit Codes ENGN6528

You can search for multiple units by putting a comma between your unit codes.

SEARCH : FINAL TIMETABLE

Displaying records 1 to 1 of 1

Exam Code	Exam Title	Date	Time	Writing Time (minutes)	Reading Time (minutes)	Venue	Exam Conditions
ENGN6528_Semester 1	Computer Vision	Tuesday 20/04/2021	12:00pm	90	15	Online Exam - Wattle	Any material ,

Displaying records 1 to 1 of 1

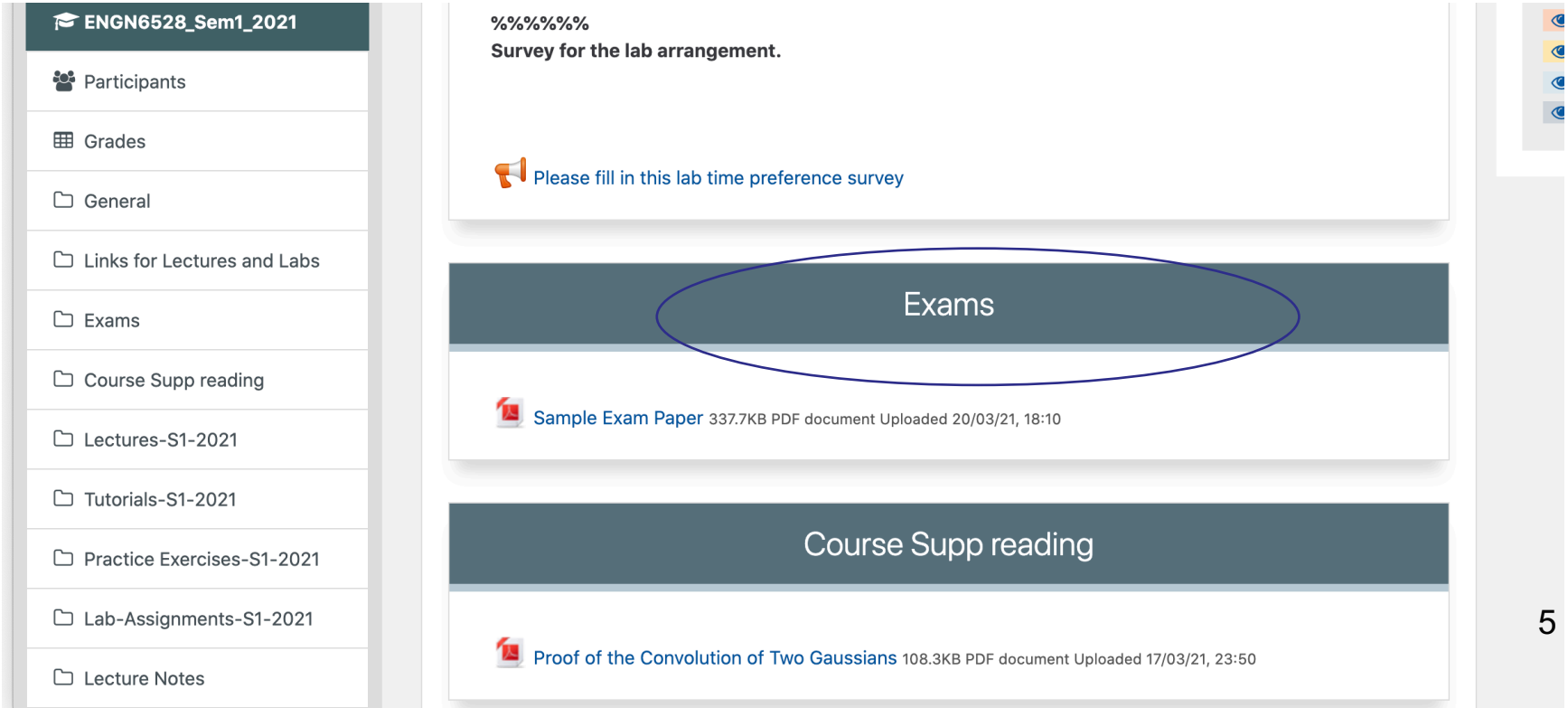
- 90 minutes in total including reading, downloading question paper, and uploading results.
- Questions are expected to be finished in one hour.
- Open book, download and upload papers through wattle. You need to work on questions by yourself.
- We are going to have multiple versions of the quiz papers. Students will be randomly assigned to different groups for exam.

Announcements

- Type of questions (see sampled exam paper)
  - Basic Calculation
  - Basic Concept
  - Algorithm design

Announcements

- We will provide instructions two weeks before the exam.



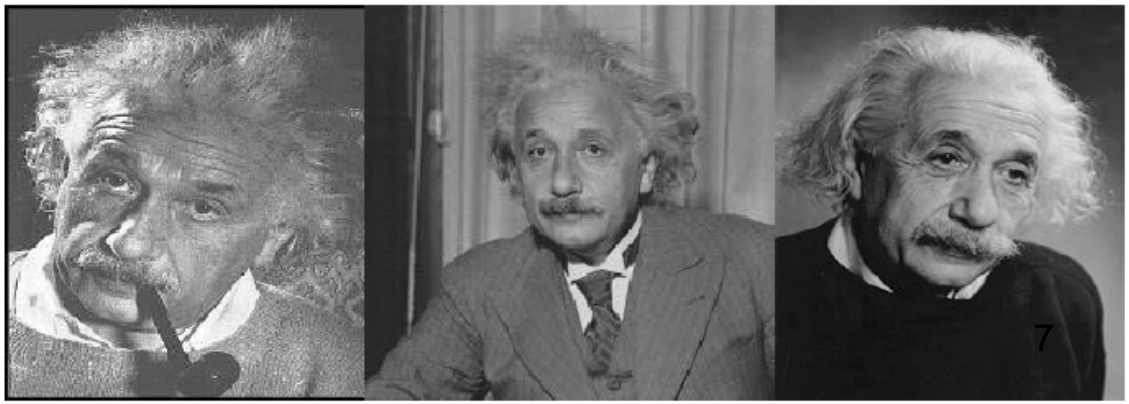
Review

- Face detection

Natural Face Recognition Ability

- Humans perform face recognition almost instantaneously
- Highly invariant to pose, scale, rotation, lighting changes
- Can handle partial occlusions and changes in age
- And we can do all this for faces of several thousand individuals

Who is this?



Today's lecture

- PCA and EigenFace representation [Turk & Pentland]
- Viola-Jones face detection: [Viola & Jones]

Face detection versus recognition

Detection

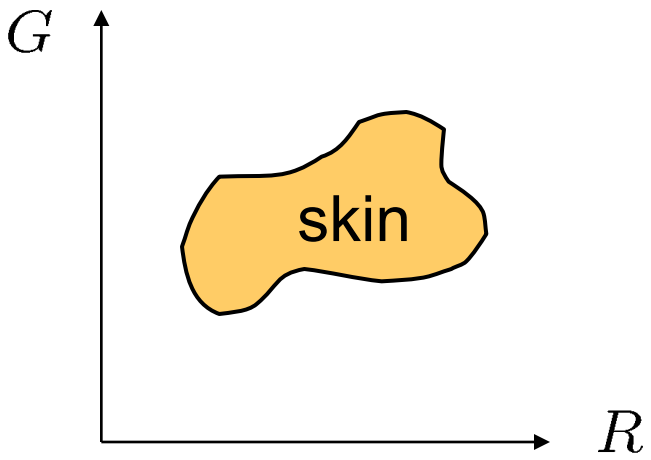
Recognition

“Sally”

Common face detection methods

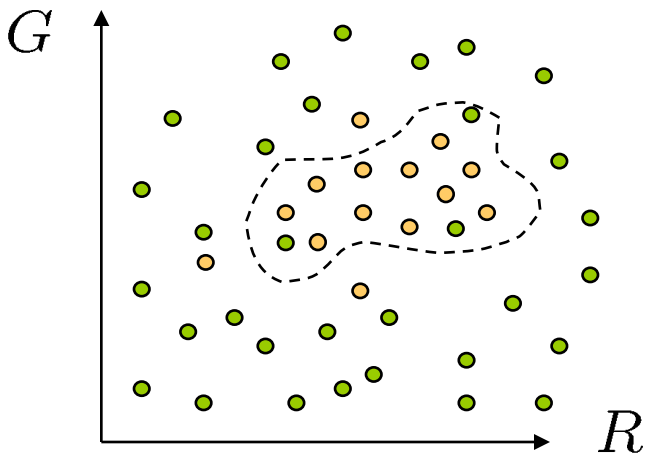
- Colour segmentation
- Template matching
- Eigen Face: Face space representation
- Viola-Jones Adaboost face detection(Part I)
- SIFT BoW detection
- Deep-Net detection
- …

Skin color detection



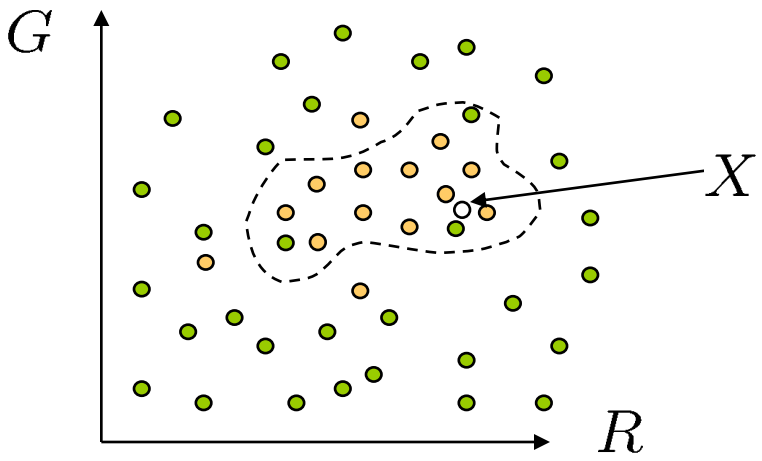
- Skin pixels have a distinctive range of colors
  - Corresponds to region(s) in RGB color space
- Skin classifier
  - A pixel X = (R,G,B) is skin if it is in the skin (color) region
  - How to find this region?

Skin colour detection



- **Learn** the skin region from examples
  - Manually label skin/non pixels in one or more “training images”
  - Plot the training data in RGB space
    - skin pixels shown in orange, non-skin pixels shown in green
    - some skin pixels may be outside the region, non-skin pixels inside.

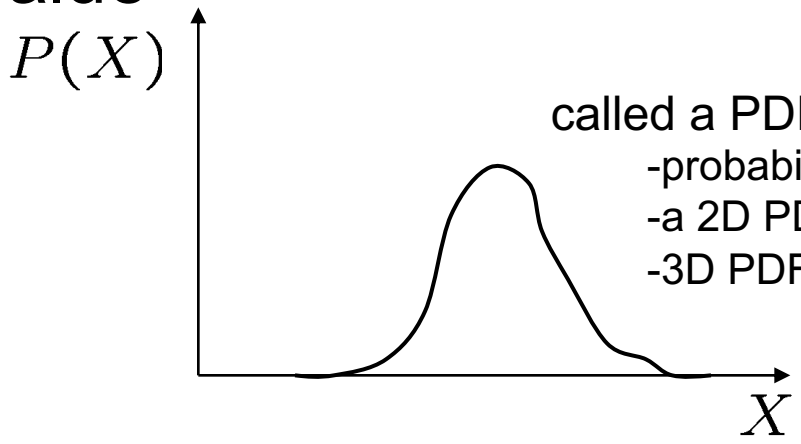
Skin classifier



- Given X = (R,G,B): how to determine if it is skin or not?
  - Nearest neighbor
    - find labeled pixel closest to X
  - Find plane/curve that separates the two classes
    - popular approach: Support Vector Machines (SVM)
  - Data modeling
    - fit a probability density/distribution model to each class

Probability

- X is a random variable
- P(X) is the probability that X achieves a certain value



called a PDF

-probability distribution/density function
-a 2D PDF is a surface
-3D PDF is a volume

$$0 \leq P(X) \leq 1$$

$$\int_{-\infty}^{\infty} P(X)dX = 1$$

continuous X

$$\sum P(X) = 1$$

discrete X

Bayes' Law

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- In the context of our problem:

what we can measure (**likelihood**)

domain knowledge (**prior**)

$$P(\text{skin}|R) = \frac{P(R|\text{skin})\ P(\text{skin})}{P(R)}$$

what we want (**posterior**)

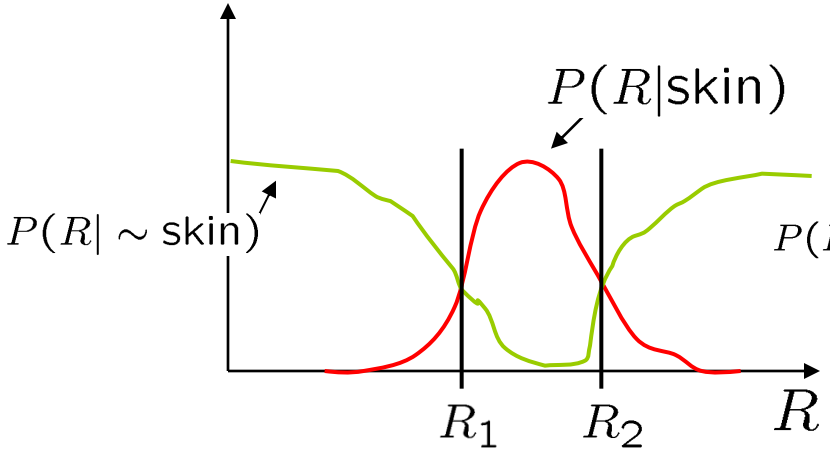
normalization term

$$P(R) = P(R|\text{skin})P(\text{skin}) + P(R|\sim\text{skin})P(\sim\text{skin})$$

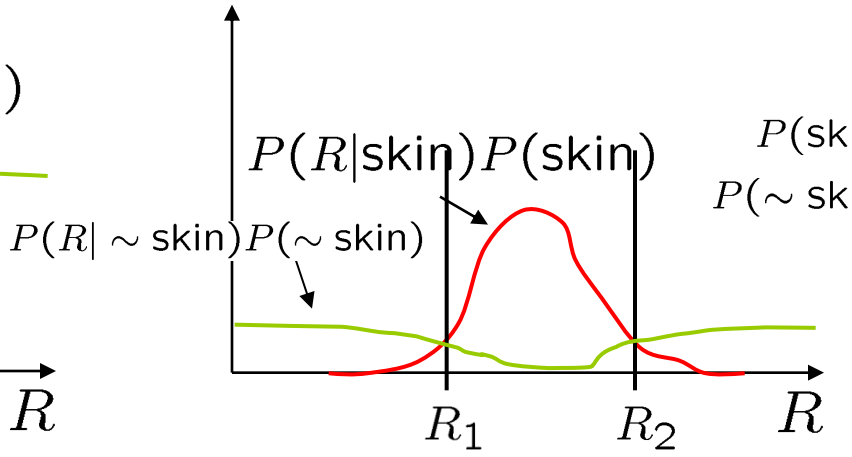
What can we use for the prior P(skin)?

- Domain knowledge:
  - P(skin) may be larger if we know the image contains a person
  - For a portrait, P(skin) may be higher for pixels in the center
- Learn the prior from the training set. How?
  - P(skin) is proportion of skin pixels in training set

Bayesian estimation



likelihood



posterior (unnormalized)

- Bayesian estimation
  - Goal is to choose the label (skin or ~skin) that maximizes the posterior ↔ minimizes probability of misclassification
    - this is called **Maximum A Posteriori (MAP) estimation**

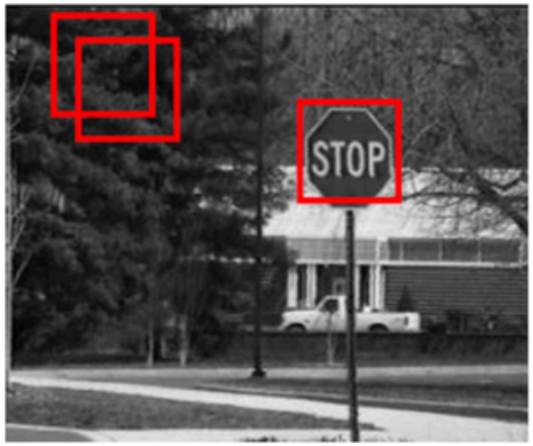
Template Matching

Objects can be represented by storing sample images or "templates"

Stop sign template

Hypotheses fromTemplate Matching

- Place the template at every location on the given image.
  - Compare the pixel values in the template with the pixel values in the underlying region of the image.
  - If a "good" match is found, announce that the object is present in the image.



- Possible measures are: SSD, SAD, Cross-correlation, Normalized Cross-correlation, max difference, etc.

Limitations of Template Matching

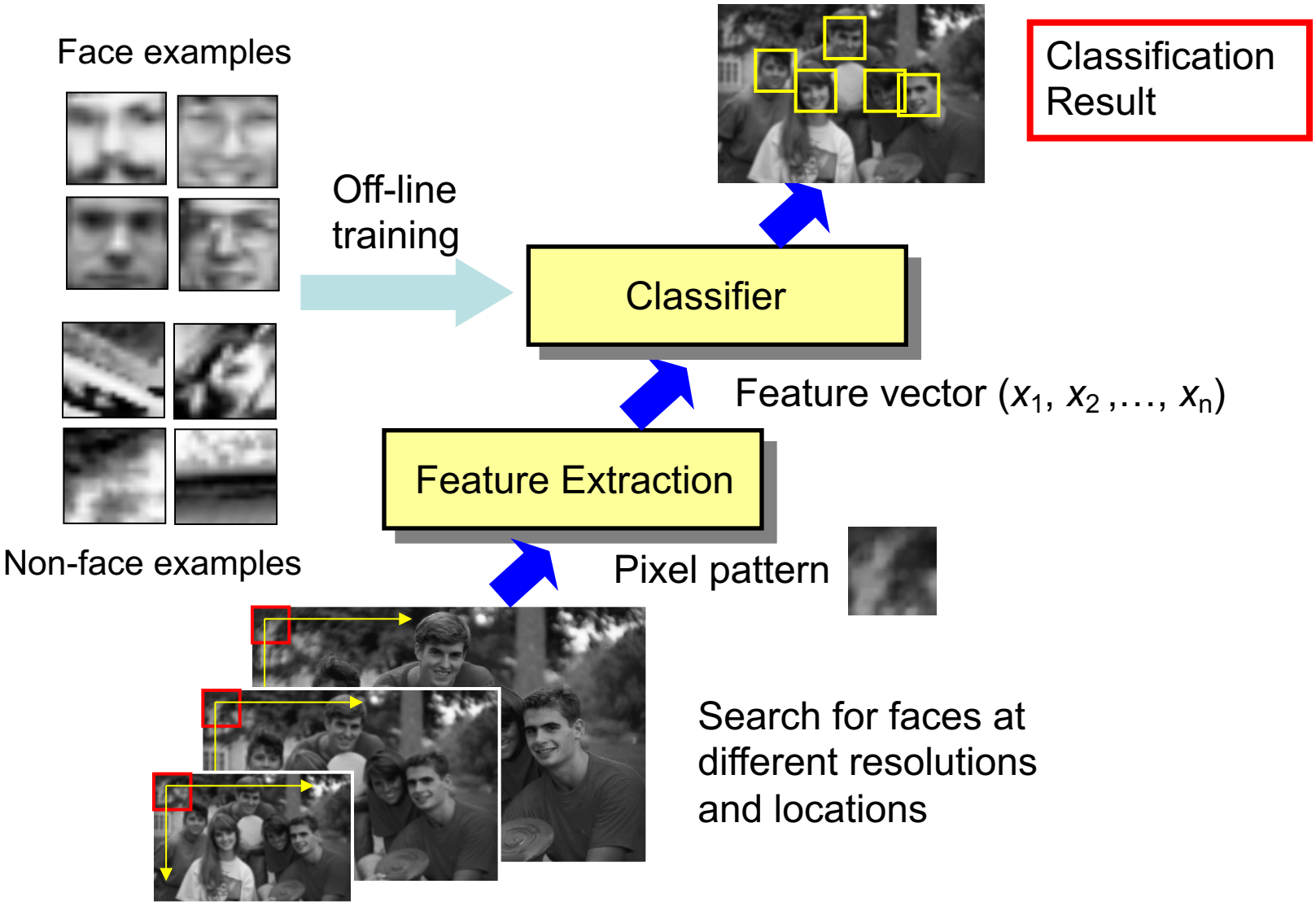
- If the object appears scaled, rotated, or skewed on the image, the match will not be good.

Solution:

- Search for the template and possible transformations of the template:

Not very efficient! (but doable …)

Face Detection – basic scheme



Face Detection – basic scheme

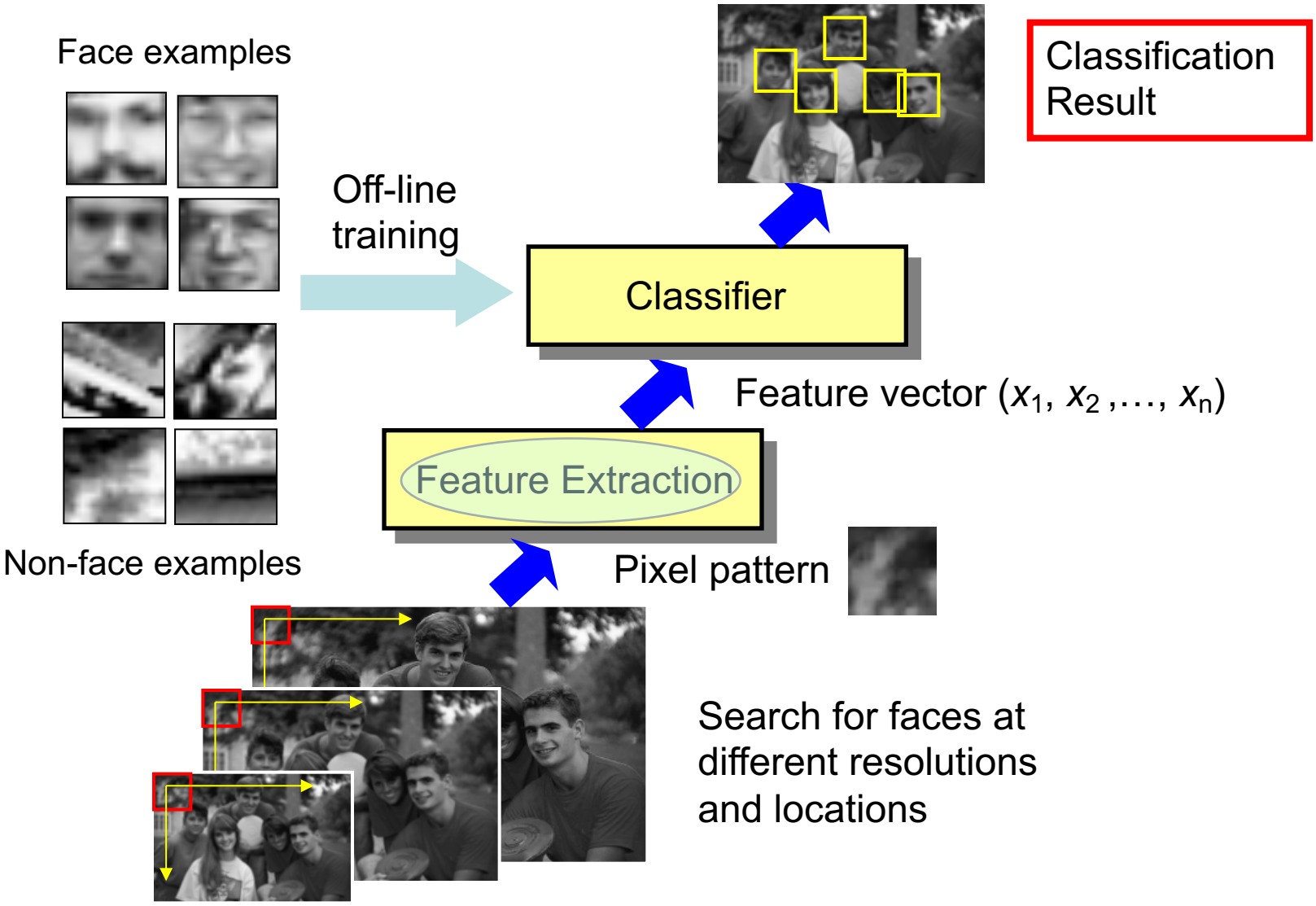
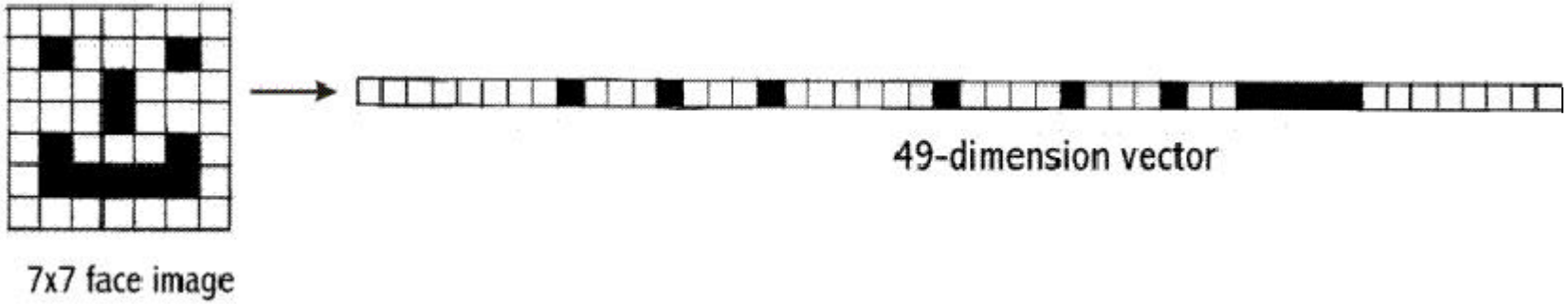


Image feature representation

- Images as a high dimensional vector



NN classifer

1. Treat pixels as a vector

$$\mathbf{x}$$

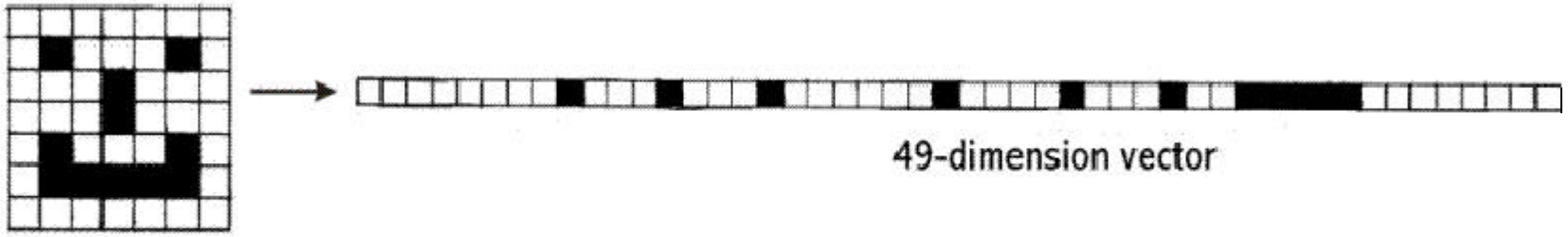
2. Recognize face by nearest neighbor

$$\mathbf{y}_1 \cdots \mathbf{y}_n$$

$$k = \underset{k}{\operatorname{argmin}} \left\| \mathbf{y}_k{}^T - \mathbf{x} \right\|$$

High Dimensional Correlated Data

- Images as a high dimensional vector



- A typical image used for image processing will be 512x512= 262144 dimension vector!
- (Registered) face images are highly correlated

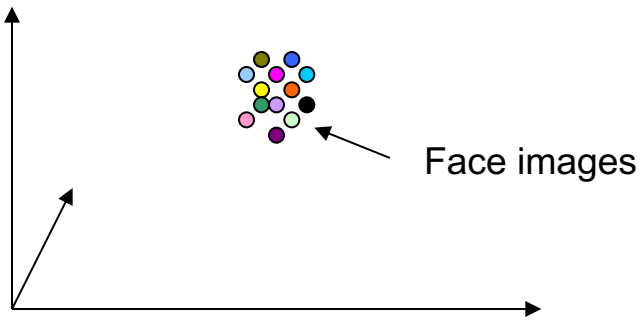


Image space (high dimensional space of all possible images)

The space of all face images

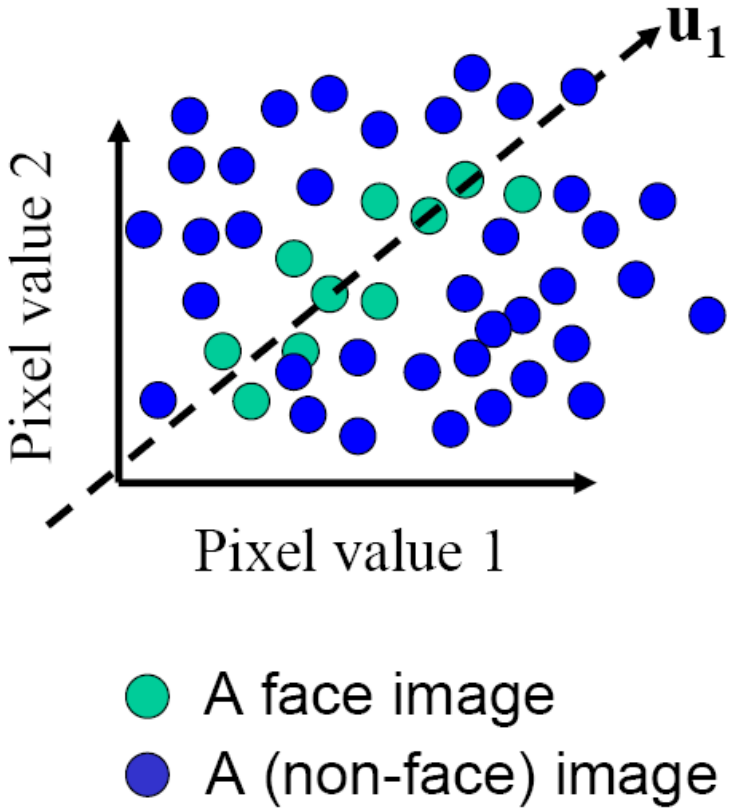
- When viewed as vectors of pixel values, face images are extremely high-dimensional
  - 512x512 image = 262,144 dimensions
  - Slow and lots of storage
- But very few 262,144-dimension long vectors are valid face images; real face vectors are sparse (i.e. sparse distribution) in the data space.
- We want to effectively model the subspace of face images.

Image feature representation:

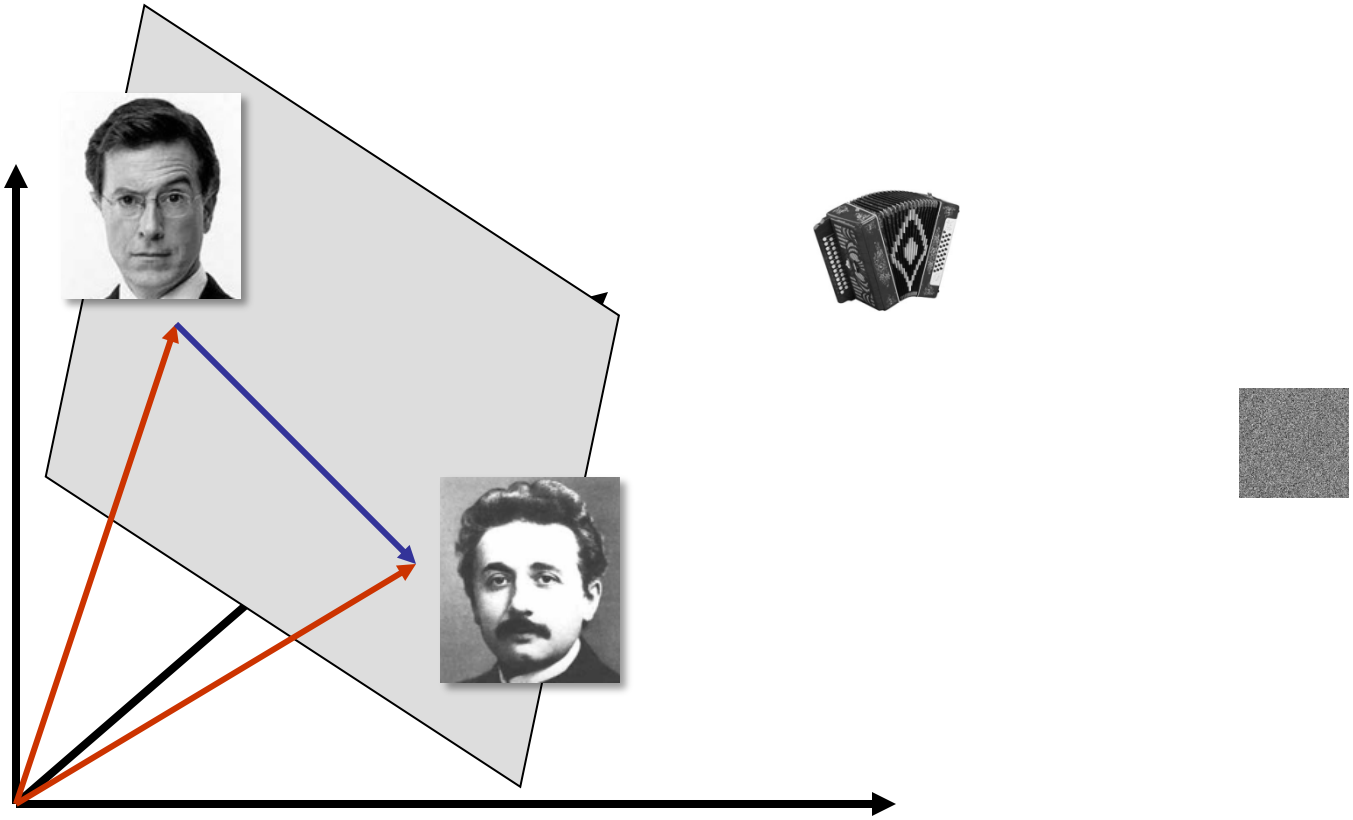
Dimensionality reduction

The space of all face images

- Eigenface idea:
  - Construct a low-dimensional linear subspace that best explains the variation in the set of face images



Dimensionality reduction



- The set of faces is a “subspace” of the set of all images
  - Suppose it is K dimensional
  - We can find the best subspace using PCA (principal component analysis)
  - This is like fitting a “hyper-plane” to the set of faces
    - spanned by vectors $\mathbf{v_1}$, $\mathbf{v_2}$, ..., $\mathbf{v_K}$
    - any face $\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_k\mathbf{v_k}$

Principal Component Analysis (PCA)

- General dimensionality reduction technique
- It preserves most of variance with a much more compact representation
  - Lower storage requirements (eigenvectors + a few numbers per face)
  - Faster for matching

Principal Component Analysis (PCA)

- Given: N data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ in R^d
- We want to find a new set of features that are linear combinations of original ones:

$$w = \mathbf{u}^T(\mathbf{x}_i - \boldsymbol{\mu})$$

(**μ**: mean of data points)

- Choose unit vector **u** in R^d that captures the most data variance

Principal Component Analysis

- Direction that maximizes the variance of the projected data:

$$\underset{\mathbf{u}}{\text{Maximize}} \quad \frac{1}{N}\sum_{i=1}^{N} \underbrace{\mathbf{u}^{\mathrm{T}}(\mathbf{x}_i-\mu)}_{\text{Projection of data point}}(\mathbf{u}^{\mathrm{T}}(\mathbf{x}_i-\mu))^{\mathrm{T}}$$

subject to $||\mathbf{u}||=1$

$$= \mathbf{u}^{\mathrm{T}}\underbrace{\left[1/N\sum_{i=1}^{N}(\mathbf{x}_i-\mu)(\mathbf{x}_i-\mu)^{\mathrm{T}}\right]}_{\text{Covariance matrix of data}}\mathbf{u}$$

$$= \mathbf{u}^{\mathrm{T}}\Sigma\mathbf{u}$$

→ **The direction that maximizes the variance is the eigenvector associated with the largest eigenvalue of Σ**

Another perspective

- Minimize approximation error

$$
\begin{aligned}
\|\vec{x_i} - (\vec{w} \cdot \vec{x_i})\vec{w}\|^2 &= \left(\vec{x_i} - (\vec{w} \cdot \vec{x_i})\vec{w}\right) \cdot \left(\vec{x_i} - (\vec{w} \cdot \vec{x_i})\vec{w}\right) \\
&= \vec{x_i} \cdot \vec{x_i} - \vec{x_i} \cdot (\vec{w} \cdot \vec{x_i})\vec{w} \\
&\quad -(\vec{w} \cdot \vec{x_i})\vec{w} \cdot \vec{x_i} + (\vec{w} \cdot \vec{x_i})\vec{w} \cdot (\vec{w} \cdot \vec{x_i})\vec{w} \\
&= \|\vec{x_i}\|^2 - 2(\vec{w} \cdot \vec{x_i})^2 + (\vec{w} \cdot \vec{x_i})^2 \vec{w} \cdot \vec{w} \\
&= \vec{x_i} \cdot \vec{x_i} - (\vec{w} \cdot \vec{x_i})^2
\end{aligned}
$$

Another perspective

- Minimize approximation error

$$MSE(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} ||\vec{x_i}||^2 - (\vec{w}\cdot\vec{x_i})^2$$

$$= \frac{1}{n}\left(\sum_{i=1}^{n} ||\vec{x_i}||^2 - \sum_{i=1}^{n} (\vec{w}\cdot\vec{x_i})^2\right)$$

Another perspective

- Minimize approximation error

$$MSE(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}||\vec{x_i}||^2 - (\vec{w}\cdot\vec{x_i})^2$$

$$= \frac{1}{n}\left(\sum_{i=1}^{n}||\vec{x_i}||^2 - \sum_{i=1}^{n}(\vec{w}\cdot\vec{x_i})^2\right)$$

Equivalent to Maximize: $\sum_{i=1}^{n}(\vec{w}\cdot\vec{x_i})^2$

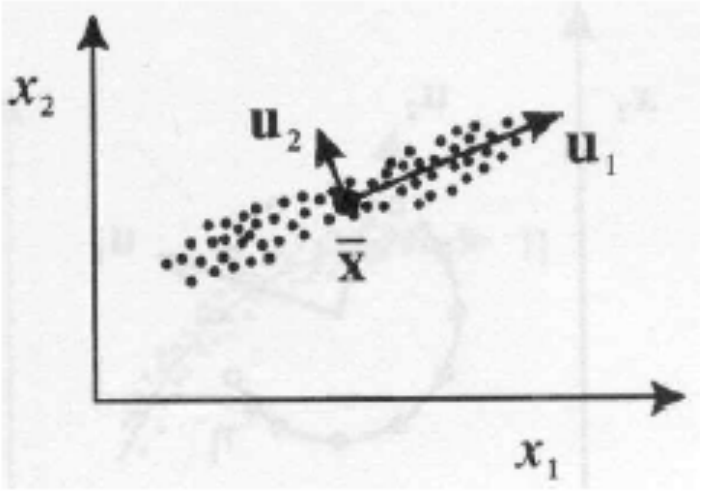
Another perspective

- Maximize variance

$$
\begin{aligned}
\sigma^2_{\vec{w}} &= \frac{1}{n}\sum_i \left(\vec{x_i}\cdot\vec{w}\right)^2 \\
&= \frac{1}{n}(\mathbf{xw})^T(\mathbf{xw}) \\
&= \frac{1}{n}\mathbf{w}^T\mathbf{x}^T\mathbf{xw} \\
&= \mathbf{w}^T\frac{\mathbf{x}^T\mathbf{x}}{n}\mathbf{w} \\
&= \mathbf{w}^T\mathbf{vw}
\end{aligned}
$$

Geometric interpretation

- PCA projects the data along the directions where the data varies most.
- These directions are determined by the eigenvectors of the covariance matrix corresponding to the largest eigenvalues.
- The magnitude of the eigenvalues corresponds to the variance of the data along the eigenvector directions.



PCA for dimension reduction

- Lower dimensionality basis
  - Approximate vectors by finding a basis in an appropriate lower dimensional space.

(1) Higher-dimensional space representation:

$$x = a_1 v_1 + a_2 v_2 + \cdots + a_N v_N$$

$v_1, v_2, ..., v_N$ is a basis of the N-dimensional space

(2) Lower-dimensional space representation:

$$\hat{x} = b_1 u_1 + b_2 u_2 + \cdots + b_K u_K$$

$u_1, u_2, ..., u_K$ is a basis of the K-dimensional space

- *Note:* if both bases have the same size ($N = K$), then $x = \hat{x}$)

PCA Algorithm

- Suppose x_1, x_2, ..., x_M are N x 1 vectors

Step 1: $\bar{x} = \frac{1}{M}\sum_{i=1}^{M} x_i$

Step 2: subtract the mean: $\Phi_i = x_i - \bar{x}$ (i.e., center at zero)

Step 3: form the matrix $A = [\Phi_1\ \Phi_2 \cdots \Phi_M]$ (N x M matrix), then compute:

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = \frac{1}{M} A A^T$$

(sample **covariance** matrix, N x N, characterizes the *scatter* of the data)

Step 4: compute the eigenvalues of C: $\lambda_1 > \lambda_2 > \cdots > \lambda_N$

Step 5: compute the eigenvectors of C: $u_1, u_2, \ldots, u_N$

PCA Algorithm

- Since C is symmetric, $u_1, u_2, \ldots, u_N$ form a basis, (i.e., any vector x or actually $(x - \bar{x})$, can be written as a linear combination of the eigenvectors):

$$x - \bar{x} = b_1 u_1 + b_2 u_2 + \cdots + b_N u_N = \sum_{i=1}^{N} b_i u_i \qquad b_i = \frac{(x - \bar{x}).u_i}{(u_i.u_i)}$$

Step 6: (**dimensionality reduction step**) keep only the terms corresponding to the K largest eigenvalues:

$$\hat{x} - \bar{x} = \sum_{i=1}^{K} b_i u_i \text{ where } K << N$$

- The representation of $\hat{x} - \bar{x}$ into the basis $u_1, u_2, \ldots, u_K$ is thus

$$\begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_K \end{bmatrix}$$

Symmetric eigendecomposition

eigenvalues/vectors of a symmetric matrix have important special properties

- all the eigenvalues are real
- the eigenvectors corresponding to different eigenvalues are orthogonal
- a symmetrix matrix is diagonalizable by an orthogonal similarity transformation:

$$Q^T A Q = \Lambda, \qquad Q^T Q = I$$

in the remainder of the lecture we assume that A is symmetric (and real)

PCA algorithm

- The linear transformation $R^N \rightarrow R^K$ that performs the dimensionality reduction is:

$$\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_K \end{bmatrix} = \begin{bmatrix} u_1^T \\ u_2^T \\ \dots \\ u_K^T \end{bmatrix} (x - \bar{x}) = U^T (x - \bar{x})$$

(i.e., simply computing coefficients of linear expansion)

Eigenfaces (PCA on face images)

1. Compute covariance matrix of face images

2. Compute the principal components ("eigenfaces")
   - K eigenvectors with top K largest eigenvalues

3. Represent all face images in the dataset as linear combinations of eigenfaces
   - Perform nearest neighbor on these coefficients

M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991

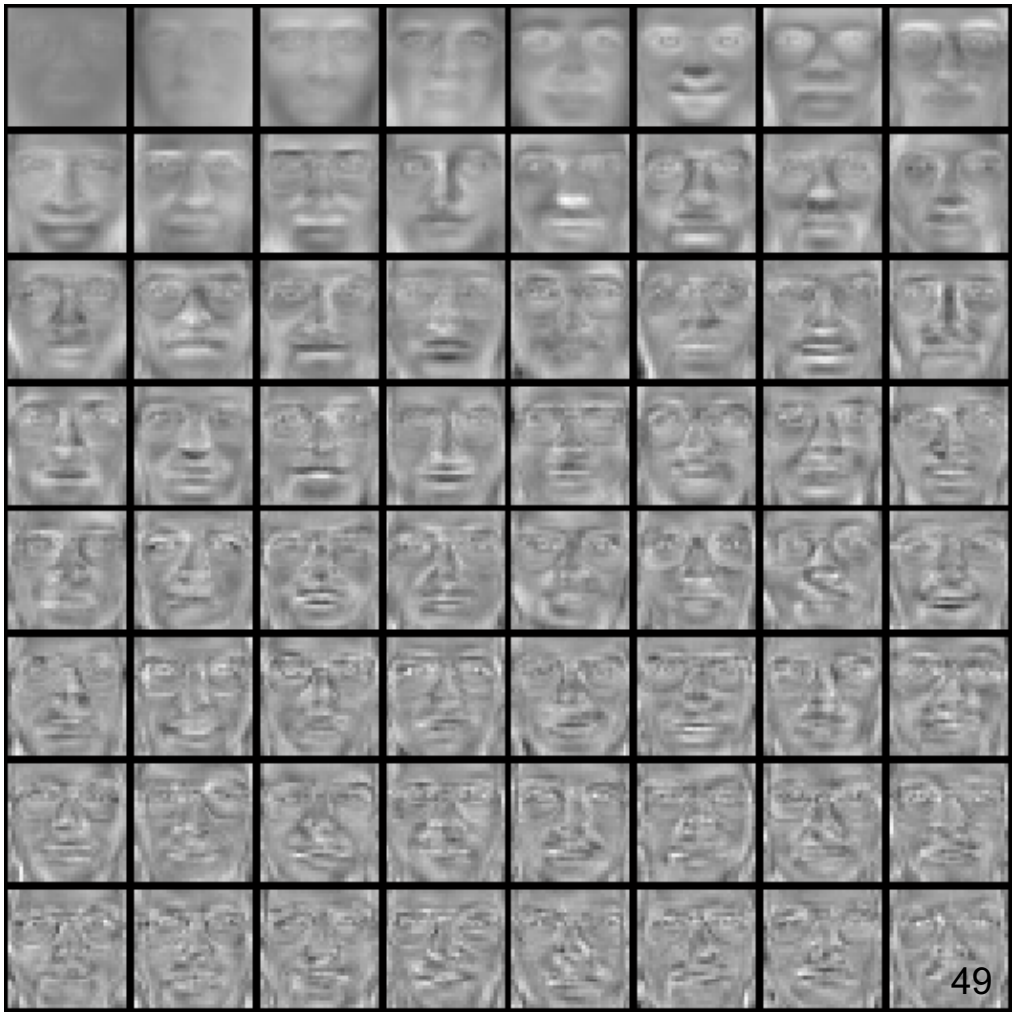
Eigenfaces example

- Training images
- $\mathbf{x}_1,\ldots,\mathbf{x}_N$

Eigenfaces example

Mean: μ

Top eigenvectors: $u_1,\ldots u_k$



Representation and reconstruction

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

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

$$= \quad w_1, \ldots, w_k$$

Representation and 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:

$\hat{x}$ = μ + $w_1u_1+w_2u_2+w_3u_3+w_4u_4+$ …

Reconstruction example

- The visualization of eigenvectors:

These are the top 4 eigenvectors from a training set of 400 images (ORL Face Database). They look like faces, hence called Eigenface.

Reconstruction

P = 4

P = 200

P = 400

After computing eigenfaces using 400 face images from ORL face database

How to choose K?

- Choose *K* using the following criterion:

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i} > Threshold \quad \text{(e.g., 0.9 or 0.95)}$$

- In this case, we say that we “preserve” 90% or 95% of the information (variance) in the data.

- If K=N, then we “preserve” 100% of the information in the data.

Eigenfaces

Computed using 400 face images from ORL face database

Recognition with eigenfaces

Process labeled training images

- Find mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
- Find k principal components (eigenvectors of $\boldsymbol{\Sigma}$) $\mathbf{u}_1,\ldots\mathbf{u}_k$
- Project each training image $\mathbf{x}_i$ onto subspace spanned by principal components:
  $(w_{i1},\ldots,w_{ik}) = (\mathbf{u}_1^T(\mathbf{x}_i - \boldsymbol{\mu}), \ldots, \mathbf{u}_k^T(\mathbf{x}_i - \boldsymbol{\mu}))$

Given novel image $\mathbf{x}$

- Project onto subspace:
  $(w_1,\ldots,w_k) = (\mathbf{u}_1^T(\mathbf{x} - \boldsymbol{\mu}), \ldots, \mathbf{u}_k^T(\mathbf{x} - \boldsymbol{\mu}))$
- Optional: check reconstruction error $\hat{\mathbf{x}} - \mathbf{x}$ to determine whether image is really a face
- Classify as closest training face in k-dimensional subspace

M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991

Recognition Using Eigenfaces

- Given an unknown face image Γ (centered and of the same size like the training faces) follow these steps:

Step 1: normalize Γ: $\Phi = \Gamma - \Psi$

Step 2: project on the eigenspace

$$\hat{\Phi} = \sum_{i=1}^{K} w_i u_i \quad (w_i = u_i^T \Phi) \qquad (where \| u_i \| = 1)$$

Step 3: represent Φ as: $\Omega = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_K \end{bmatrix}$

Step 4: find $e_r = \min_l \|\Omega - \Omega^l\|$ where $\| \Omega - \Omega^l \| = \sum_{i=1}^{K} (w_i - w_i^l)^2$

Step 5: if $e_r < T_r$, then Γ is recognized as face l from the training set.

The distance e_r is called *distance in face space (difs)*

Reconstruction from partial information

- Robust to partial face occlusion.

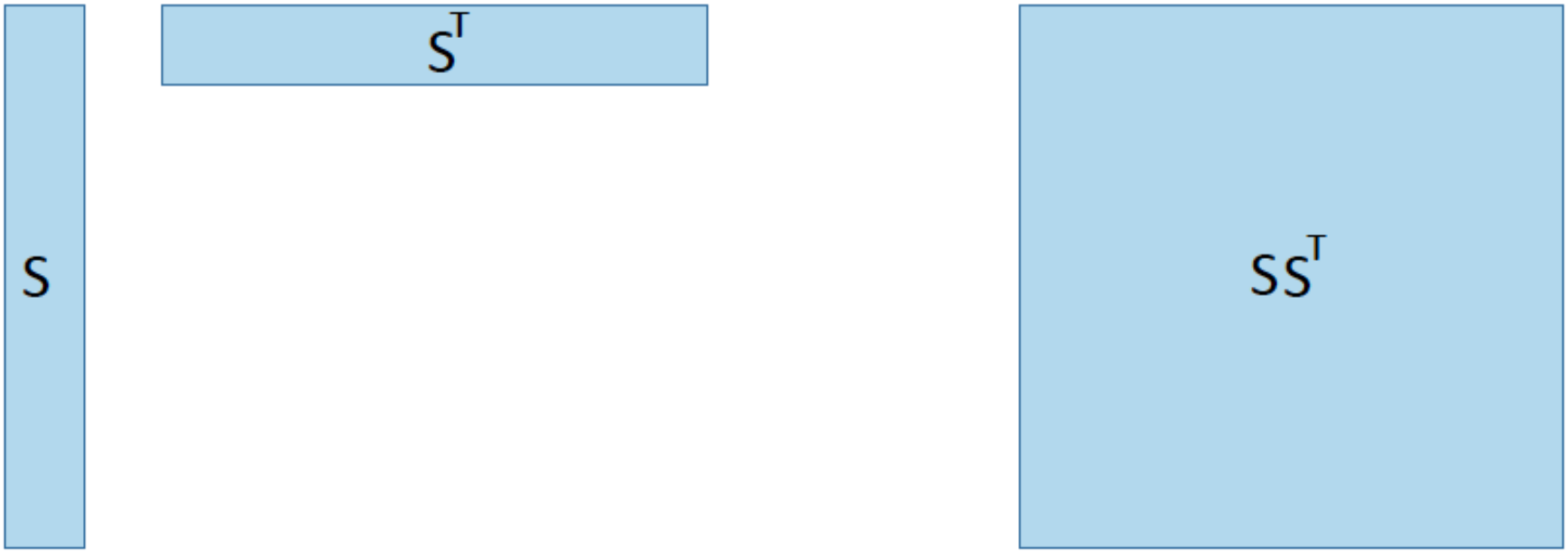
Input Reconstructed

Limitations

Global appearance method: not robust to misalignment, background variation

Issues?

- Image of size: 256 x 256, each image as a vector: 65526 x 1
- Training set size: M is around 1000
- High dimension of covariance matrix Σ: 65526 x 65526
- Eigen value decomposition is *intractable* for even small size image
- Solution?



S^TS

Eigenvalues of S^TS and SS^T are the same

Theorem 1.4. *Let* $\mathtt{S}$ *me an* $n \times k$ *matrix with* $k < n$. *Then the non-zero eigenvalues of* $\mathtt{S}\mathtt{S}^\mathsf{T}$ *are equal to the eigenvalues of* $\mathtt{S}^\mathsf{T}\mathtt{S}$, *and if* $\mathbf{v}$ *is an eigenvector of* $\mathtt{S}^\mathsf{T}\mathtt{S}$ *with eigenvalue* λ, *then* $\mathtt{S}\mathbf{v}$ *is an eigenvalue of* $\mathtt{S}\mathtt{S}^\mathsf{T}$ *with the same eigenvalue* λ. *The corresponding unit eigenvector is therefore* $\mathtt{S}\mathbf{v}/\|\mathtt{S}\mathbf{v}\|$.

Proof Suppose that $\mathtt{S}^\mathsf{T}\mathtt{S}\mathbf{v} = \lambda\mathbf{v}$. Multiplying by $\mathtt{S}$ on the left gives

$$\mathtt{S}(\mathtt{S}^\mathsf{T}\mathtt{S})\mathbf{v} = \mathtt{S}\lambda\mathbf{v}$$
$$(\mathtt{S}\mathtt{S}^\mathsf{T})(\mathtt{S}\mathbf{v}) = \lambda(\mathtt{S}\mathbf{v}) \ .$$

PCA dimension for high-dimensional ambient space.

(i) Find the centroid of the points $\mathbf{y}_i$, given by

$$\bar{\mathbf{y}} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{y}_i \ .$$

(ii) Form the matrix $\mathtt{S}$ whose columns are the vectors $\mathbf{y}_i - \bar{\mathbf{y}}$. This is a matrix of dimension $n \times k$.

(iii) Find the m largest eigenvalues of $\mathtt{S}^\mathsf{T}\mathtt{S}$ and corresponding eigenvalues $\mathbf{v}_i$.

(iv) Let $\mathtt{A}$ be the matrix whose columns are the vectors $\mathtt{S}\mathbf{v}_i/\|\mathtt{S}\mathbf{v}_i\|$. These are the eigenvectors of $\mathtt{S}\mathtt{S}^\mathsf{T}$.

(v) The best hyperplane of dimension m fit to the points $\{\mathbf{y}_i\}$ is the affine subspace of points of the form $\mathbf{x}' = \mathtt{A}\mathbf{x} + \bar{\mathbf{y}}$, for $\mathbf{x} \in \mathbb{R}^m$.

(vi) This can also be written as the points $\mathbf{x}'$ satisfying the equation

$$(\mathtt{A}^\perp)^\mathsf{T}(\mathbf{x}' - \bar{\mathbf{y}}) = 0$$

where $\mathtt{A}^\perp$ is the matrix consisting of the columns of $\mathtt{Y}$ corresponding to the $n - m$ **smallest** eigenvalues.

Table 1.2. PCA algorithm when ambient space is high-dimensional

Recall: PCA Algorithm

- Suppose x_1, x_2, ..., x_M are N x 1 vectors

Step 1: $\bar{x} = \frac{1}{M}\sum_{i=1}^{M} x_i$

Step 2: subtract the mean: $\Phi_i = x_i - \bar{x}$ (i.e., center at zero)

Step 3: form the matrix $A = [\Phi_1\ \Phi_2 \cdots \Phi_M]$ (NxM matrix), then compute:

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = \frac{1}{M} AA^T$$

(sample **covariance** matrix, NxN, characterizes the *scatter* of the data)

Step 4: compute the eigenvalues of C: $\lambda_1 > \lambda_2 > \cdots > \lambda_N$

Step 5: compute the eigenvectors of C: $u_1, u_2, \ldots, u_N$

Solution

- Follow Step1 and Step2 from former slide.
- Step3: Form the matrix $A = [\varphi_1, \varphi_2, \varphi_3, \varphi_4, \dots, \varphi_M]$, A is of size N x M, and compute $L = A^T A$ where L is of size M x M, and $L_{mn} = \varphi_m{}^T \varphi_n$
- Step 4: Compute eigen values of L.
- Step 5: Compute eigen vectors of L for the k largest eigen values, defined as q_i
- Step 6: Compute eigen vectors u_i of the original covariance matrix, namely the eigen faces as: $u_i = Aq_i$

Step 7: Normalize the eigen vector $u_i = \frac{u_i}{\|u_i\|}$

Another possible solution

PCA using Singular Value Decomposition.

(i) Find the centroid of the points $\mathbf{y}_i$, given by

$$\bar{\mathbf{y}} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{y}_i \ .$$

(ii) Form the matrix $\mathtt{S}$ whose columns are the vectors $\mathbf{y}_i - \bar{\mathbf{y}}$. This is a matrix of dimension $n \times k$.

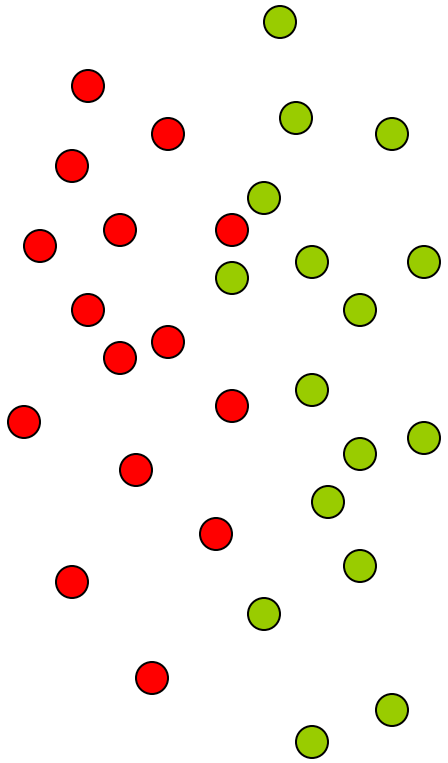
(iii) Find the SVD $\mathtt{S} = \mathtt{U}\mathtt{D}\mathtt{V}^{\mathsf{T}}$ and let $\mathtt{A}$ be the matrix consisting of the first m columns of $\mathtt{U}$.

(iv) The best hyperplane of dimension m fit to the points $\{\mathbf{y}_i\}$ is the affine subspace of points of the form $\mathbf{x}' = \mathtt{A}\mathbf{x} + \bar{\mathbf{y}}$, for $\mathbf{x} \in \mathbb{R}^m$.

Table 1.3. PCA algorithm using the Singular Value Decomposition.

Limitations

- The direction of maximum variance is not always good for classification



A more discriminative subspace: FLD

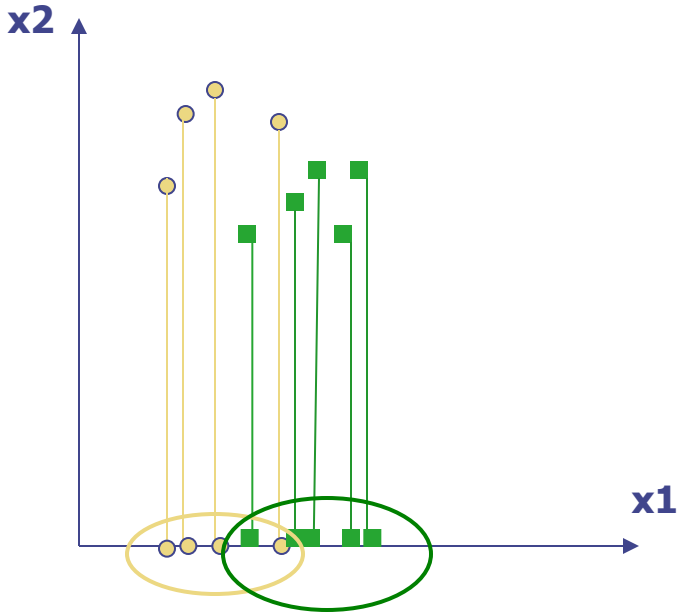
- Fisher Linear Discriminants → "Fisher Faces"

- PCA preserves maximum variance

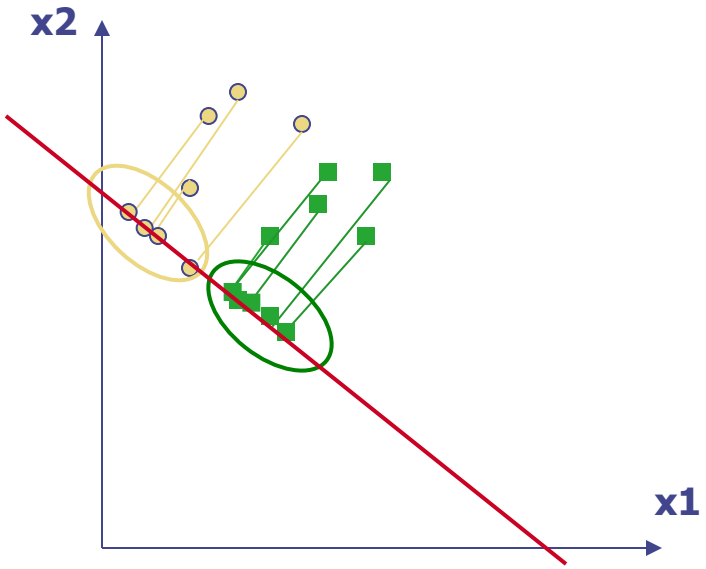
- FLD preserves discrimination
  - Find projection that maximizes scatter between classes and minimizes scatter within classes

Illustration of the Projection

- Using two classes as an example:

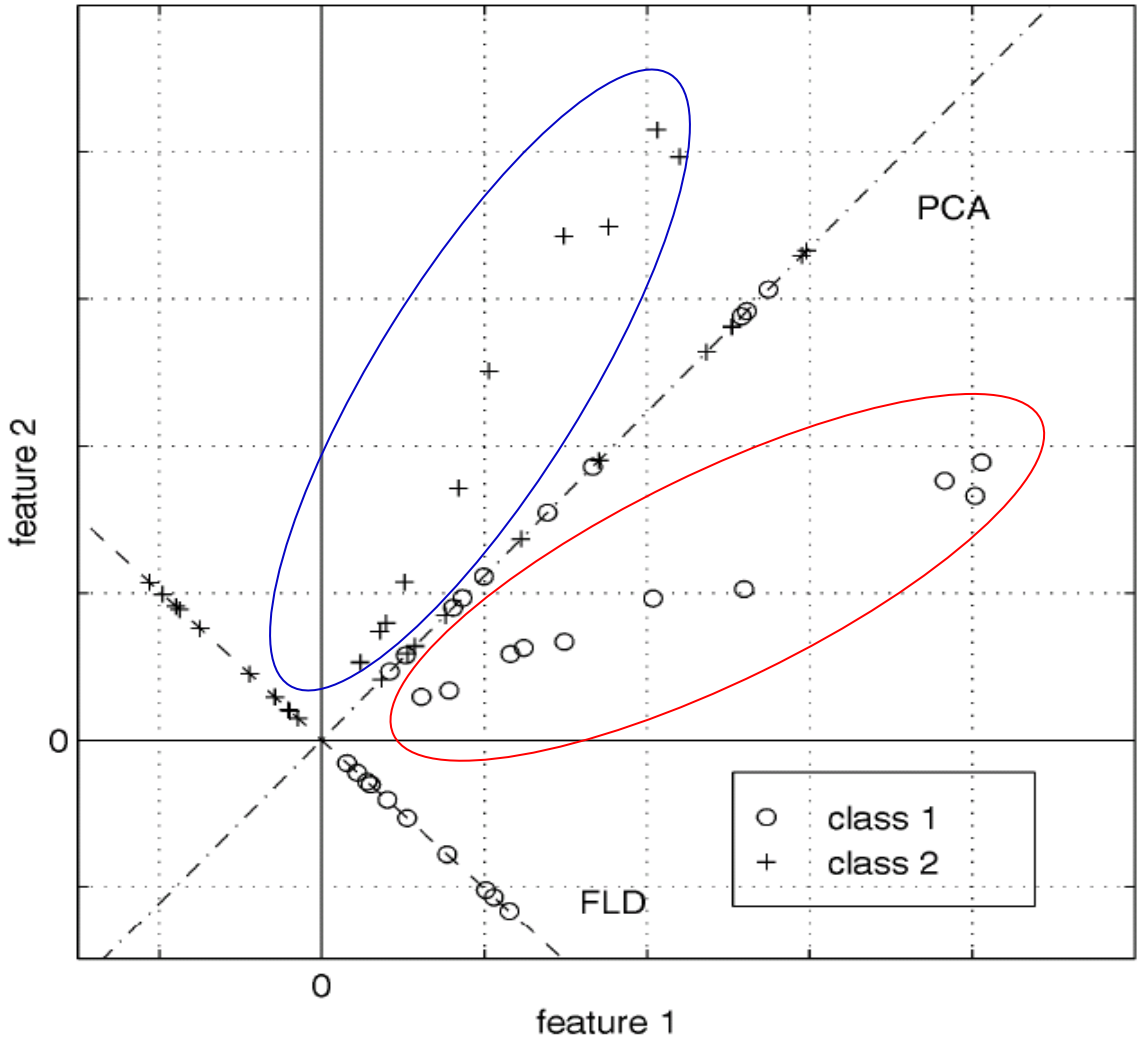


Poor Projection



Good

Comparing with PCA



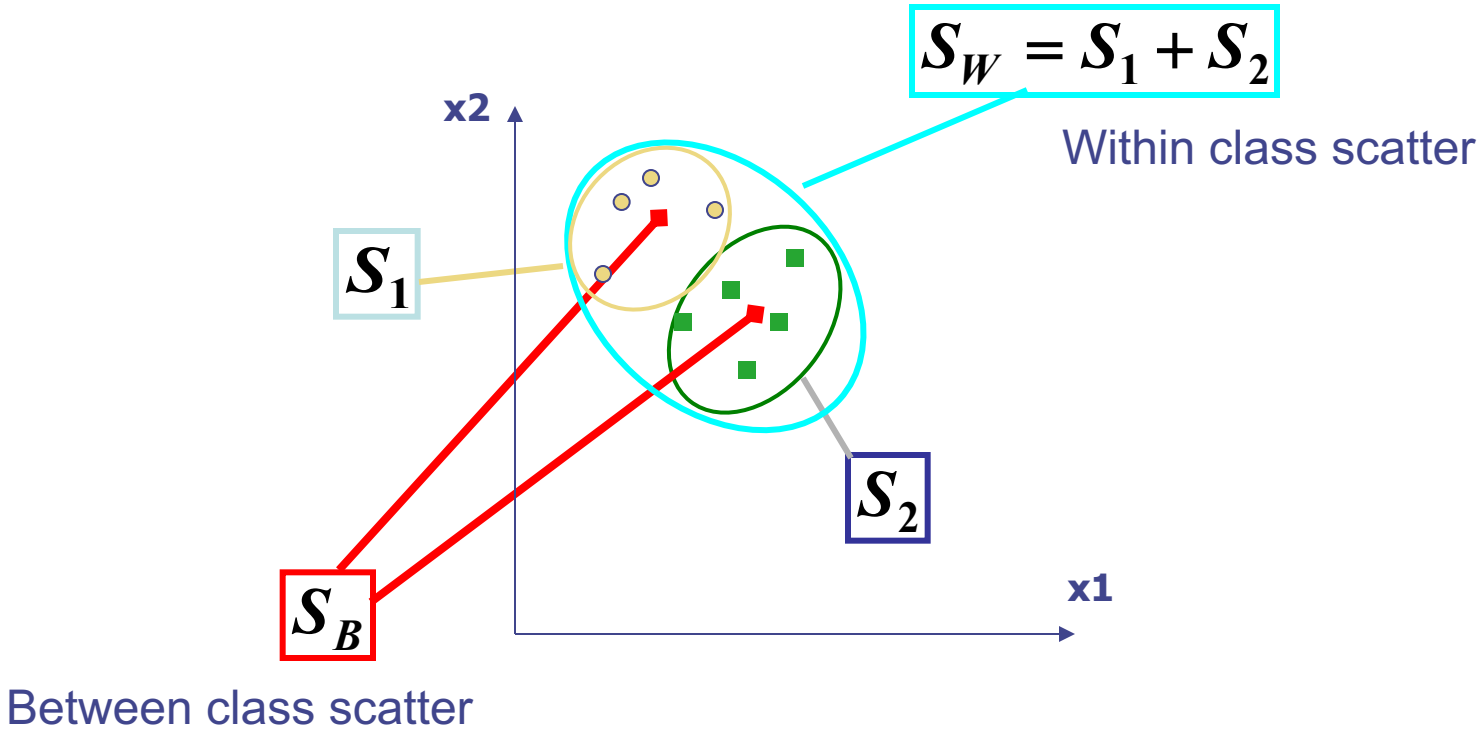
Variables

- N Sample images: $\{x_1, \cdots, x_N\}$
- c classes: $\{\chi_1, \cdots, \chi_c\}$
- Average of each class: $\mu_i = \frac{1}{N_i} \sum_{x_k \in \chi_i} x_k$
- Average of all data: $\mu = \frac{1}{N} \sum_{k=1}^{N} x_k$

Scatter Matrices

- Scatter of class i: $$S_i = \sum_{x_k \in \chi_i} (x_k - \mu_i)(x_k - \mu_i)^T$$
- Within class scatter: $$S_W = \sum_{i=1}^{c} S_i$$
- Between class scatter: $$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

Illustration



Mathematical Formulation

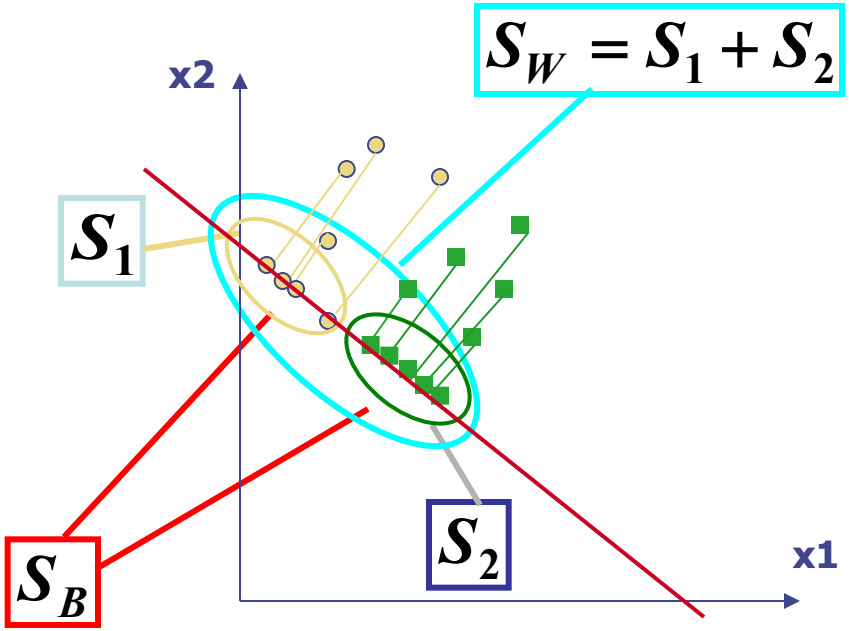
- After projection $y_k = W^T x_k$
  - Between class scatter $\tilde{S}_B = W^T S_B W$
  - Within class scatter $\tilde{S}_W = W^T S_W W$
- Objective

$$W_{opt} = \arg\max_{\mathrm{W}} \frac{\left|\tilde{S}_B\right|}{\left|\tilde{S}_W\right|} = \arg\max_{\mathrm{W}} \frac{\left|W^T S_B W\right|}{\left|W^T S_W W\right|}$$

- Solution: Generalized Eigenvectors

$$S_B w_i = \lambda_i S_W w_i \qquad i = 1, \ldots, m$$

Illustration



Recognition with FLD

- Similar to “eigenfaces”
- Compute within-class and between-class scatter matrices

$$S_i = \sum_{x_k \in \chi_i} (x_k - \mu_i)(x_k - \mu_i)^T \qquad S_W = \sum_{i=1}^{c} S_i \qquad S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

- Solve generalized eigenvector problem

$$W_{opt} = \arg\max_{\mathrm{w}} \frac{\left|W^T S_B W\right|}{\left|W^T S_W W\right|} \qquad S_B w_i = \lambda_i S_W w_i \qquad i = 1, \ldots, m$$

- Project to FLD subspace and classify by nearest neighbor

$$\hat{x} = {W_{opt}}^T x$$

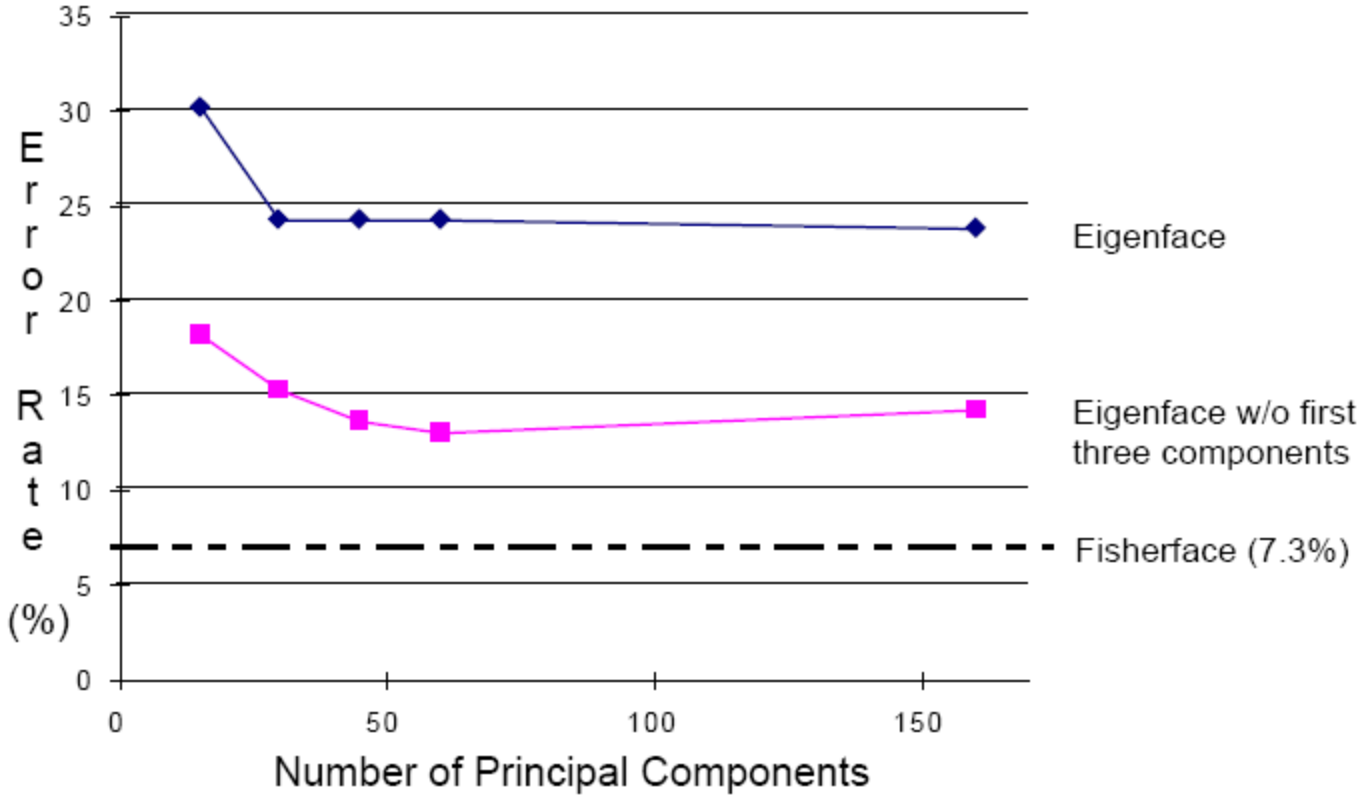
Results: Eigenface vs. Fisherface

- Input: 160 images of 16 people
- Train: 159 images
- Test: 1 image

- Variation in Facial Expression, Eyewear, and Lighting

Reference: Eigenfaces vs. Fisherfaces, Belheumer et al., PAMI 1997

Eigenfaces vs. Fisherfaces



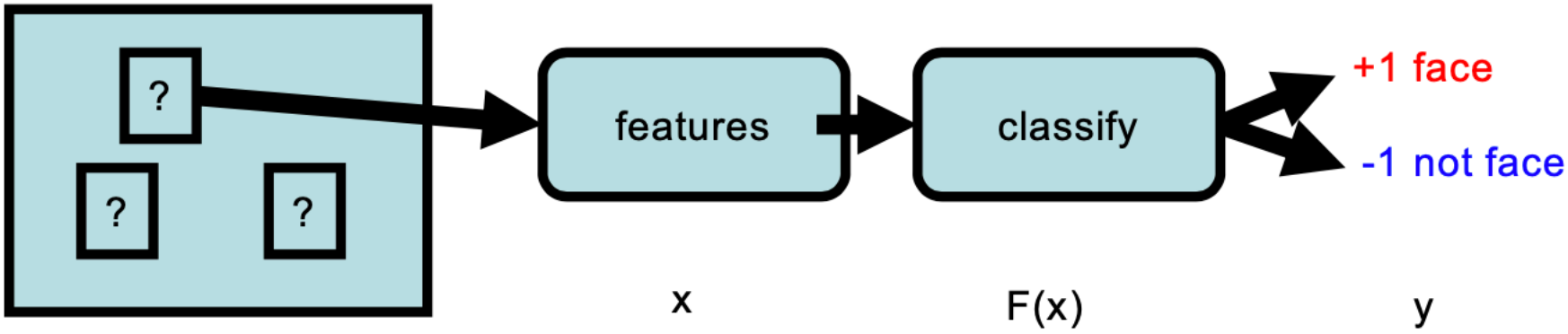
Reference: Eigenfaces vs. Fisherfaces, Belheumer et al., PAMI 1997

Readings

- Chapter 5.2.3 in the book:

Computer Vision: Algorithms and Applications

Face detection -- basic scheme



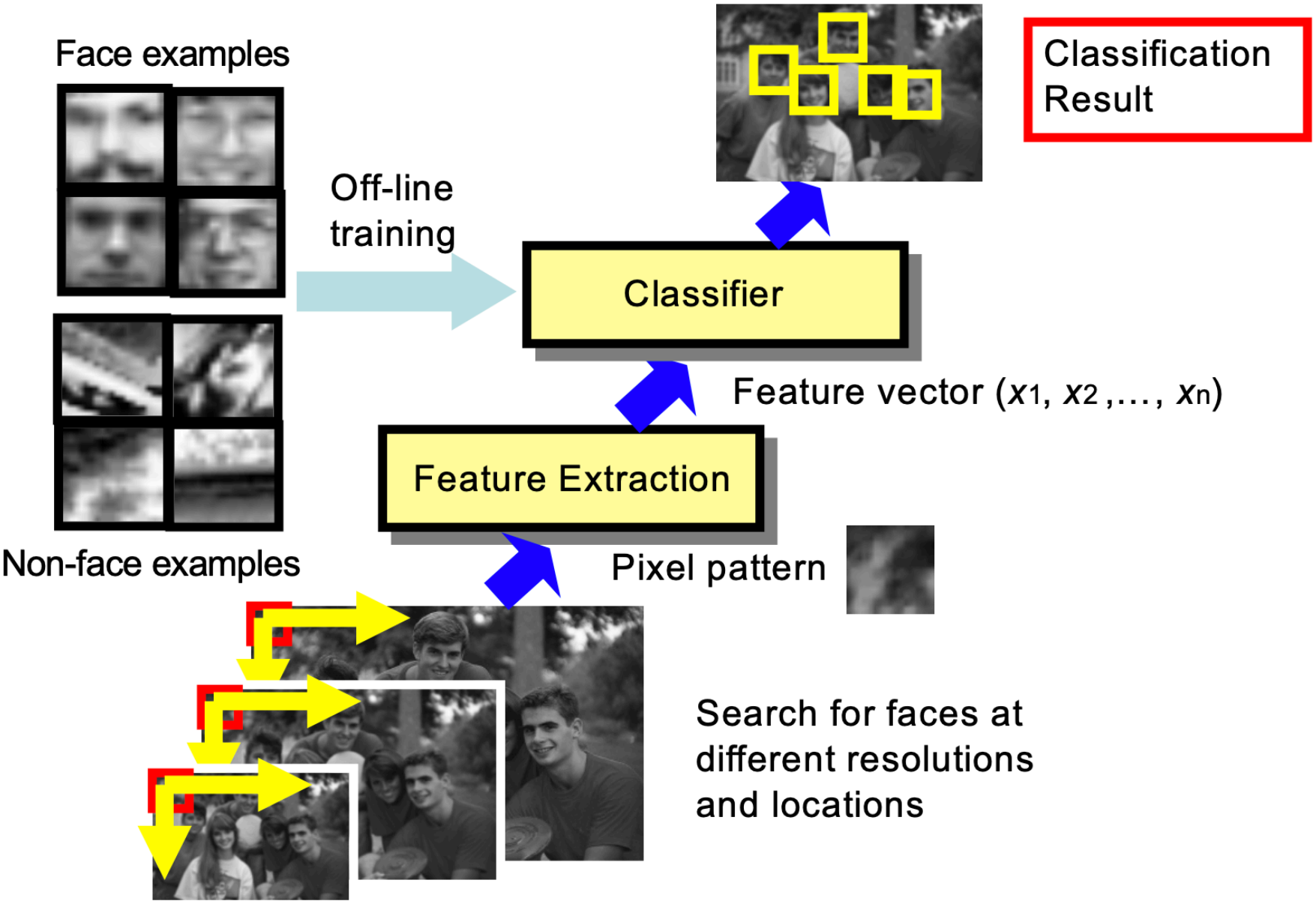
- We slide a window over the image
- Extract features for each window
- Classify each window into face/non-face

What is a face?

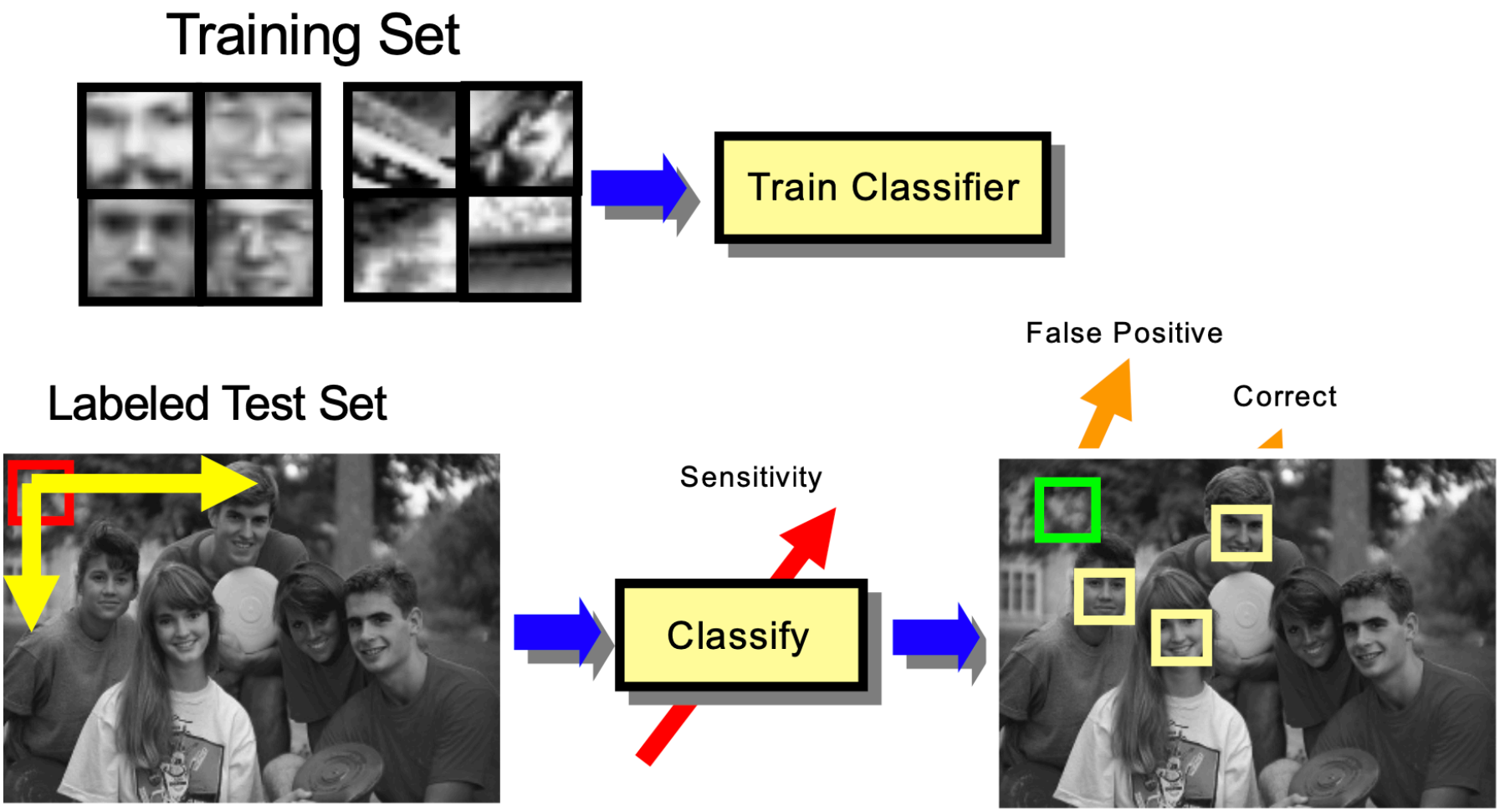
- Eyes are dark (eyebrows+shadows)
- Cheeks and forehead are bright.
- Nose is bright

Paul Viola, Michael Jones, Robust Real-time Object Detection, IJCV 04

Face detection – basic scheme



Training and Testing



Viola-Jones face detection

Paul A. Viola and Michael J. Jones

Intl. J. Computer Vision

57(2), 137–154, 2004

Some slides *(slides adapted from Bill Freeman, MIT 6.869, April 2005)*

Scan classifier over locs. & scales

Learn feature & classifier from data

- Training Data
- 5000 faces (frontal)
- 10^8 non faces
- Faces are normalized
  - Scale, translation
- Many variations
- Across individuals
- Illumination
- Pose (rotation both in plane and out)

Characteristics of the algorithm

– Feature set (…is huge about 16M features)
– Efficient feature selection using AdaBoost
– New image representation: Integral Image
– Cascaded Classifier for rapid detection

➢ Fastest known face detector for gray scale images (circa 2001, Marr Prize Winner).

Viola-Jones face detection

- 3 key ideas
  - Use of Haar features
  - Integral Image
  - Cascade classifier

- For more information, read the original paper.
- Also, lots of videos on the web.
  - https://www.youtube.com/watch?v=uEJ71VlUmMQ&vl=en-GB (Google: Viola-Jones Computerphile)

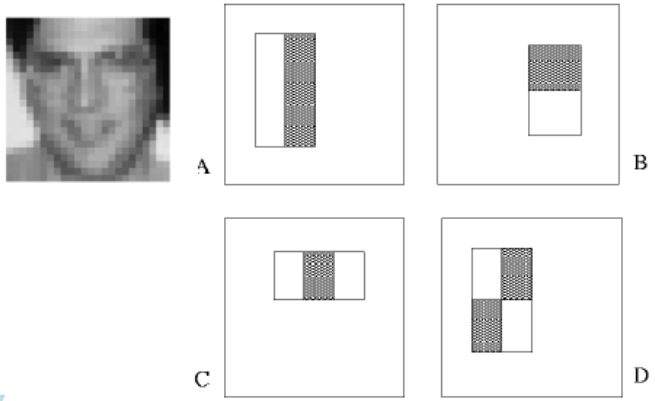
Haar features

Black = add
White = subtract.

Lots of kernels at different scales and shapes.

Image features

- “Rectangle filters”
- Differences between sums of pixels in adjacent rectangles



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

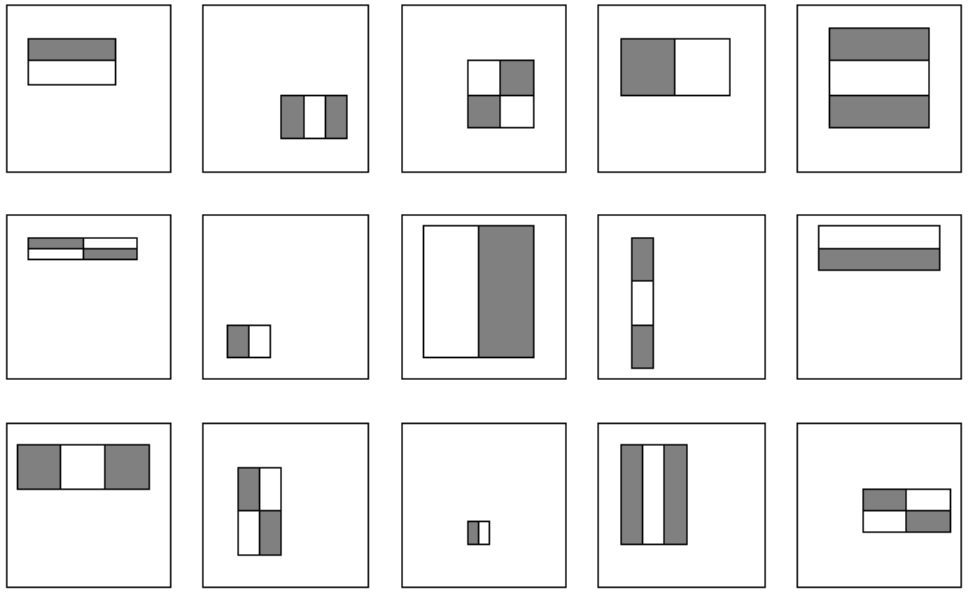
Threshold determined by training

What is a face?

- Eyes are dark (eyebrows+shadows)
- Cheeks and forehead are bright.
- Nose is bright

Paul Viola, Michael Jones, Robust Real-time Object Detection, IJCV 04

Huge library of Haar kernels



The integral image

3	2	1	5	4	7
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

Given an image …

3	2	1	5	4	7
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

What is the sum of all pixels in a given box?
(here equal to 36)

Construct the “integral image”

Integral image

- Efficient way to add up all pixels in a given region

3	2	1	5	4	7
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

Integral image

- Efficient way to add up all pixels in a given region

3	5	1	5	4	7
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

Integral image

- Efficient way to add up all pixels in a given region

3	5	1	5	4	7
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	5	4	7
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
9	11	15	16	16	22
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]
```

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
9	11	15	16	16	22
3	10	18	18	20	24
1	1	3	7	13	14
1	3	7	12	20	26
8	12	14	15	18	23

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]
```

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
9	11	15	16	16	22
3	10	18	18	20	24
1	1	3	7	13	14
1	3	7	12	20	26
8	12	14	15	18	23

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]

// Vertical pass
for i=1, ..., nrows
   for j=0, ..., ncols
      A[i][j] = A[i][j] + A[i-1][j]
```

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
9	11	15	16	16	22
3	10	18	18	20	24
1	1	3	7	13	14
1	3	7	12	20	26
8	12	14	15	18	23

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]

// Vertical pass
for i=1, ..., nrows
   for j=0, ..., ncols
      A[i][j] = A[i][j] + A[i-1][j]
```

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
12	11	15	16	16	22
3	10	18	18	20	24
1	1	3	7	13	14
1	3	7	12	20	26
8	12	14	15	18	23

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]

// Vertical pass
for i=1, ..., nrows
   for j=0, ..., ncols
      A[i][j] = A[i][j] + A[i-1][j]
```

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
12	11	15	16	16	22
3	10	18	18	20	24
1	1	3	7	13	14
1	3	7	12	20	26
8	12	14	15	18	23

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]

// Vertical pass
for i=1, ..., nrows
   for j=0, ..., ncols
      A[i][j] = A[i][j] + A[i-1][j]
```

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
12	16	15	16	16	22
3	10	18	18	20	24
1	1	3	7	13	14
1	3	7	12	20	26
8	12	14	15	18	23

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]

// Vertical pass
for i=1, ..., nrows
   for j=0, ..., ncols
      A[i][j] = A[i][j] + A[i-1][j]
```

Integral image

- Efficient way to add up all pixels in a given region

3	5	6	11	15	22
12	16	21	27	31	44
15	26	39	45	51	68
16	27	42	52	64	82
17	30	49	64	84	108
25	42	63	79	102	131

```
// Start with array A[0:nrows][0:ncols]

// Horizontal pass
for i=0, ..., nrows
   for j=1, ..., ncols
      A[i][j] = A[i][j] + A[i][j-1]

// Vertical pass
for i=1, ..., nrows
   for j=0, ..., ncols
      A[i][j] = A[i][j] + A[i-1][j]
```

Integral image

- Efficient way to add up all pixels in a given region

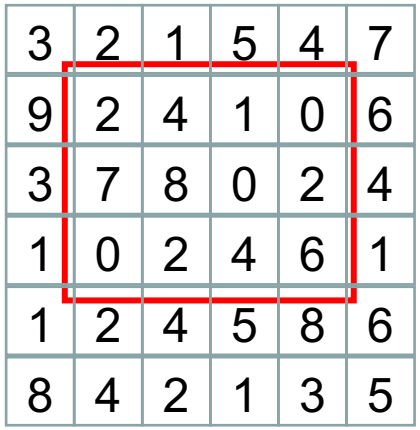
Each value A[i][j] in the integral image represents the sum of the pixels above and to the right.

3	5	6	11	15	22
12	16	21	27	31	44
15	26	39	45	51	68
16	27	42	52	64	82
17	30	49	64	84	108
25	42	63	79	102	131

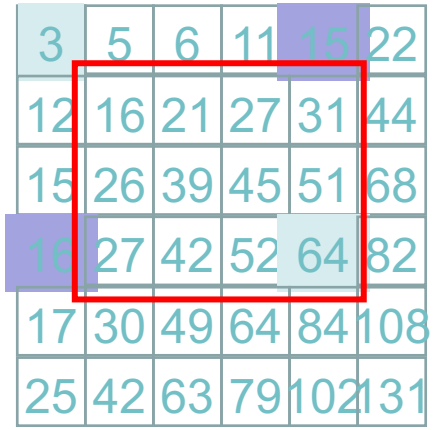
Eg 45 = Sum of pixel in the original image:

3	2	1	5	4	7
9	2	4	1	0	6
3	7	8	0	2	4
1	0	2	4	6	1
1	2	4	5	8	6
8	4	2	1	3	5

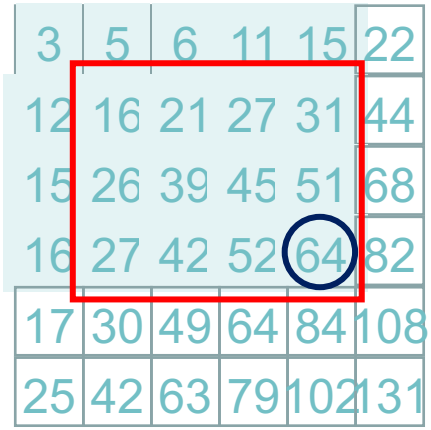
Finding the sum of all pixels in any box



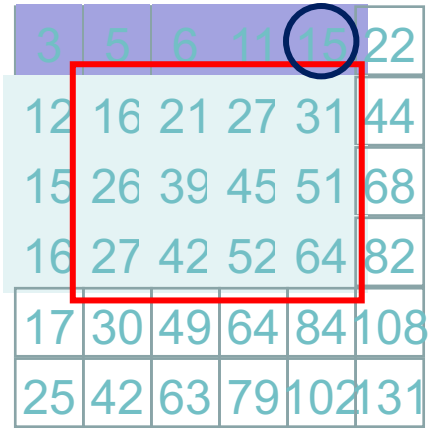
Sum = 36



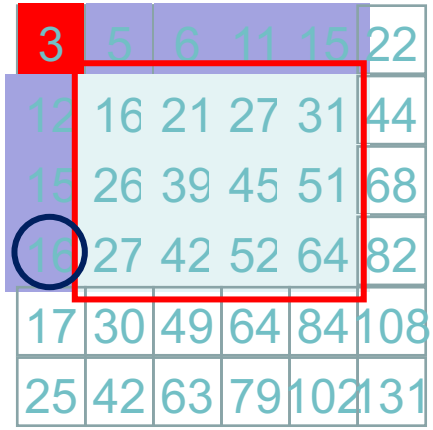
Sum = 36 = 64 + 3 – 15 - 16


3 5 6 11 15 22
12 16 21 27 31 44
15 26 39 45 51 68
16 27 42 52 64 82
17 30 49 64 84 108
25 42 63 79 102 131

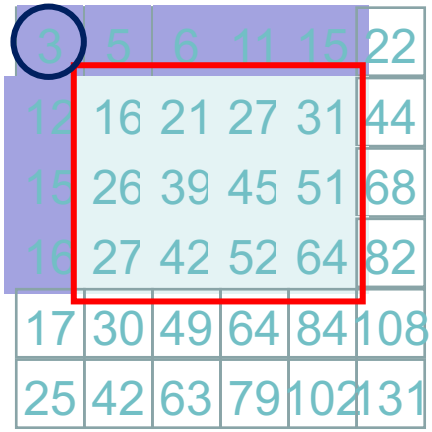
Sum = 36 = 64


3 5 6 11 15 22
12 16 21 27 31 44
15 26 39 45 51 68
16 27 42 52 64 82
17 30 49 64 84 108
25 42 63 79 102 131

Sum = 36 = 64 – 15



Sum = 36 = 64 - 15 -16



Sum = 36 = 64 - 15 -16 + 3

Summation of all pixels in any rectangle just requires summing (or subtracting) 4 values.

Readings

- M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991
- P. Viola and M. Jones, Rapid Object Detection using a Boosted Cascade of Simple Features, CVPR 2001