Taif University
College of Computers and Information Technology

Image Processing & Computer Vision

(AI:501837-3) - (DS:501838-3)

Programs:
Master of AI
Master of Data Science

Department:
Computer Science

Year:
Spring 2025

Lecture 7:

Advanced Topics in Image Processing:
Face Detection and Recognition

Prepared by: Maha A. Thafar

Lecture Outline

Lecture 7:

Part-1: Face Detection

- The Viola/Jones Face Detector.

Part-2: Face Recognition

- Basic Idea.
- PCA Concept.
- Eigenfaces Algorithm.
- Limitations

Face Detection vs. Face Recognition

- **Face Detection** is the process of detecting the presence of a face in an image or video.
- **Face detection** is a binary classification problem combined with a localization problem given a picture, decide whether it contains faces, and construct bounding boxes for the faces.
- **Face Recognition** takes the **faces detected from the localization phase** and attempts to **identify** whom the face belongs to.
- **Face Recognition** can thus be thought of as a method of ***person identification***, which we use heavily in **security** and **surveillance systems**.

Sliding Window Face Detection with Viola-Jones

P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features.* CVPR 2001.

P. Viola and M. Jones. *Robust real-time face detection.* IJCV 57(2), 2004.

Many Slides from Lana Lazebnik

Sliding Window Face Detection

Basic idea:

slide a window across image and evaluate a face model at every location

Challenges of face detection

- Sliding window detector must evaluate tens of thousands of location/scale combinations.
  - >> This is **slow** and **computationally expensive**.

Viola-Jones was designed to solve this limitation.

The Viola/Jones Face Detector

- The **Viola–Jones object detection framework** is a machine learning object detection framework proposed in 2001 by Paul Viola and Michael Jones.[1][2] It was motivated primarily by the problem of face detection, although it can be adapted to the detection of other object classes.
- A seminal approach to real-time object detection.
- **Training is slow, but detection is very fast.**

The Viola-Jones algorithm is based on four key ideas:

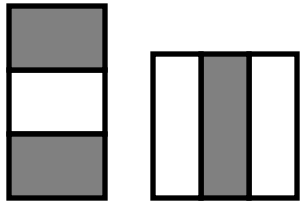
1. *HAAR-Like features*
2. *Integral images for accelerating feature computation*
3. *Adaboost learning for feature selection*
4. *Attentional Cascade of classifiers for fast non-face window rejection (i.e. false positive)*

https://www.baeldung.com/cs/viola-jones-algorithm#:~:text=What%20Is%20Viola%2DJones%20Algorithm,primarily%20conceived%20for%20face%20detection.

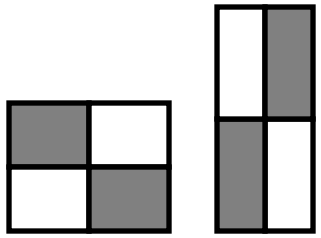
1. HAAR-Like Features

- Features that are fast to compute
- Differences of sums of intensity
- Computed at different positions and scales within sliding window

Two-rectangle features
(Edge Features)

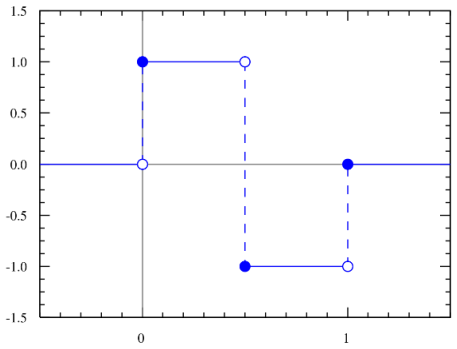


Three-rectangle features
(Line Features)



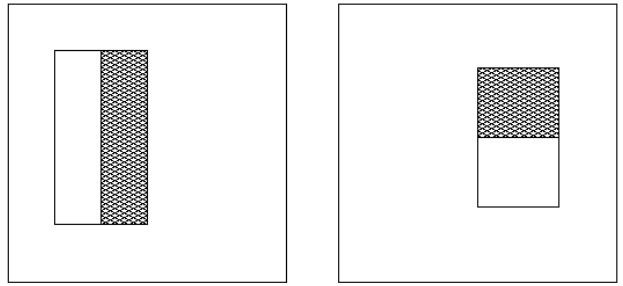
Four-rectangle
features, etc.

Haar wavelet

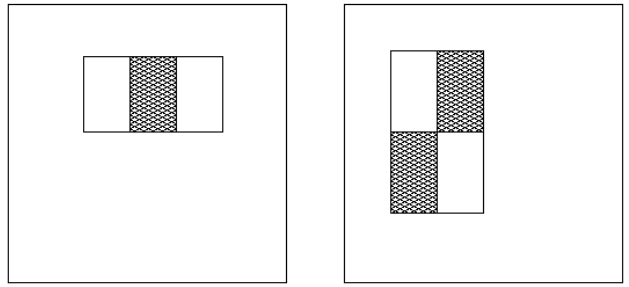


1. HAAR-Like Features

Image Features



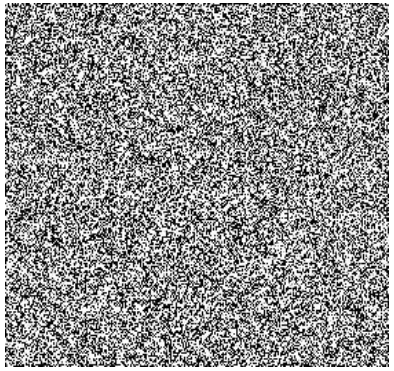
"Rectangle filters"



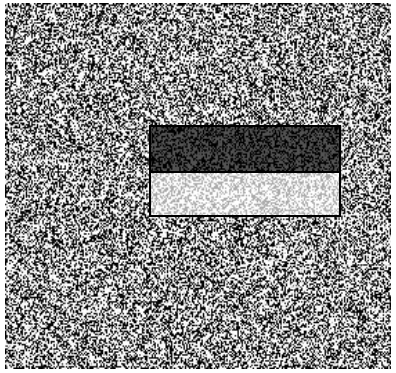
Value = ∑(pixels in white area) – ∑(pixels in black area)

1. HAAR-Like Features

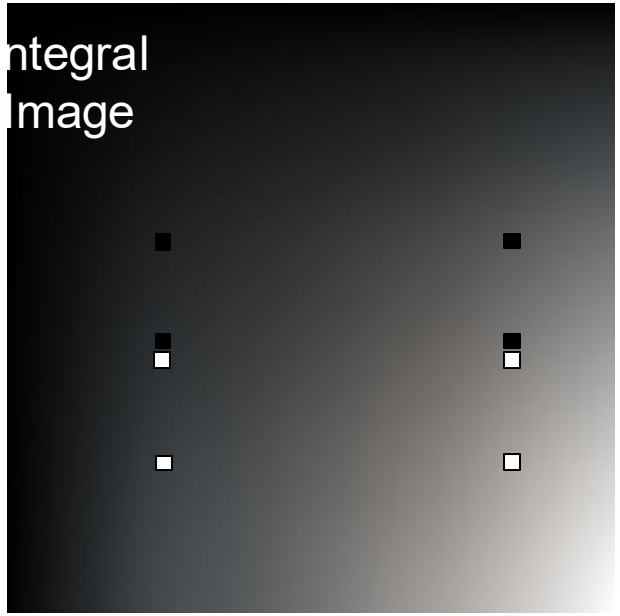
Source



Result



Computing a rectangle feature

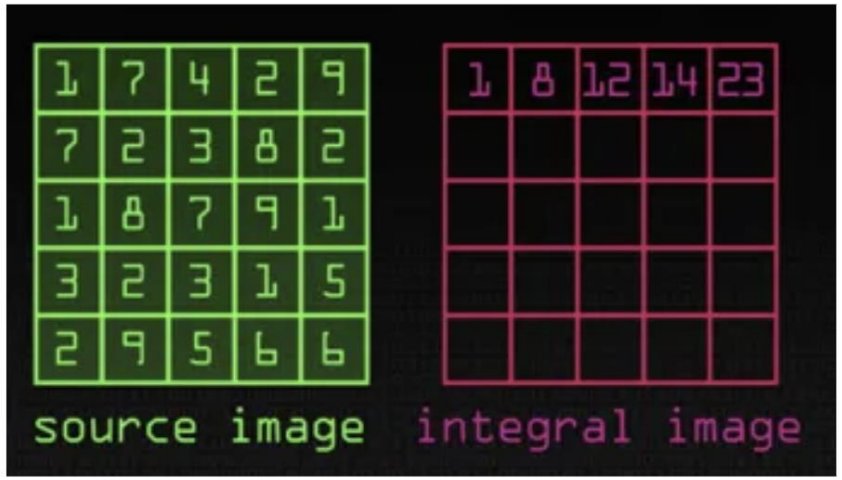


2. Integral images for fast feature computation

- To speed up features extraction process, an intermediate representation for the image called **integral image** is used.
- **The integral image is a data structure for efficiently computing the sum of pixel values in a rectangular image window.** Hence, the Haar-like features can be computed very quickly using the integral image representation.

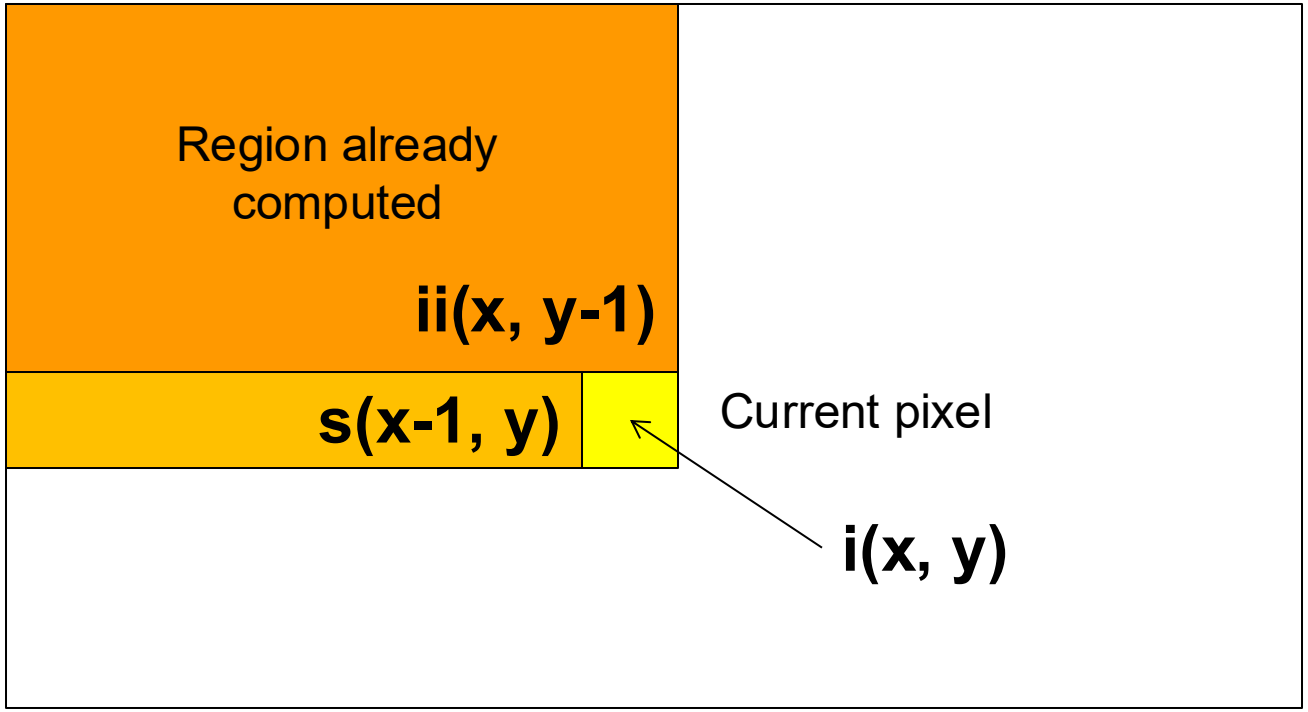
a) Given a grayscale image `I`, the integral image value `ii(x,y)` at the pixel `(x,y)` is the sum of all the pixels above and to the left of `(x,y)`, inclusive.
b) This can quickly be computed in one pass through the image.
c) 'Summed area table'

$i(x,y)$



Source: Detecting Faces (Viola Jones Algorithm) — Computerphile

Computing the Integral images



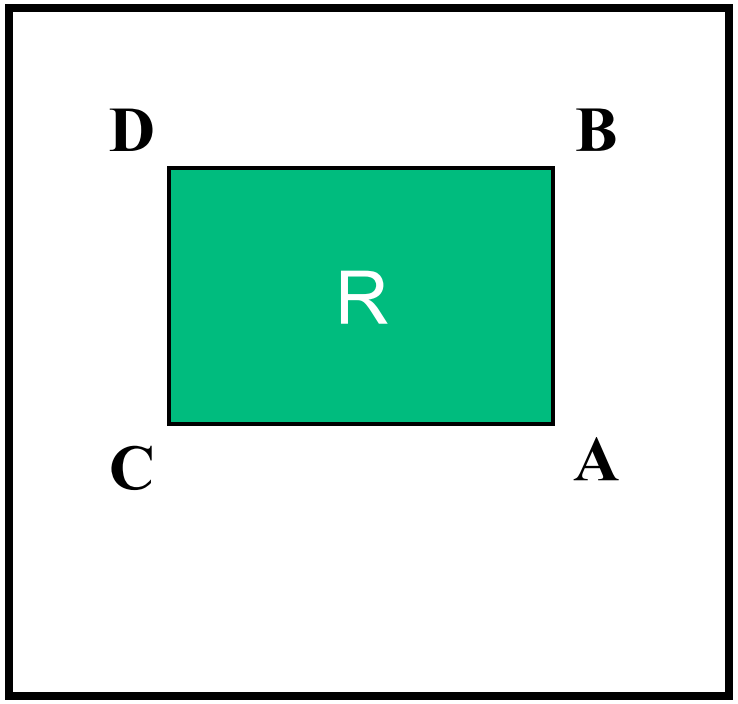
- Cumulative row sum: $s(x, y) = s(x–1, y) + i(x, y)$
- Integral image: $ii(x, y) = ii(x, y–1) + s(x, y)$

Python: ii = np.cumsum(i)

Computing sum with rectangle

- Let R be a desired rectangle.

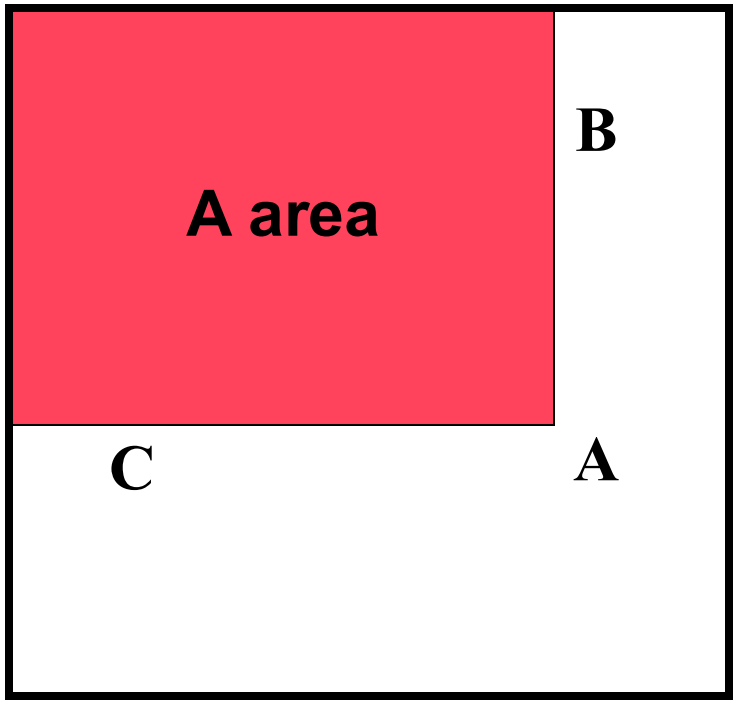
- A,B,C,D are the values of the integral image at the corners of R.



Computing sum with rectangle

- Let R be a desired rectangle.

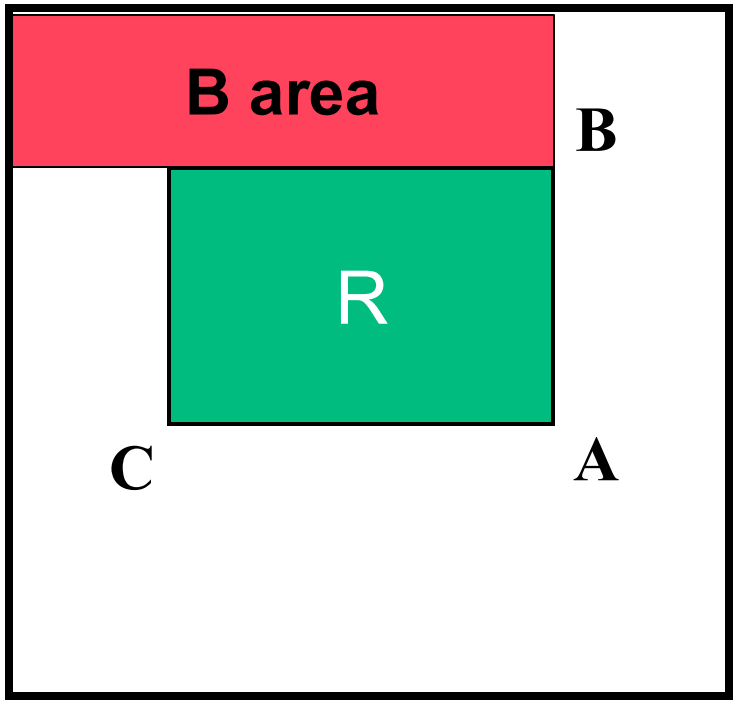
- A,B,C,D are the values of the integral image at the corners of R.



Computing sum with rectangle

•Let R be a desired rectangle.

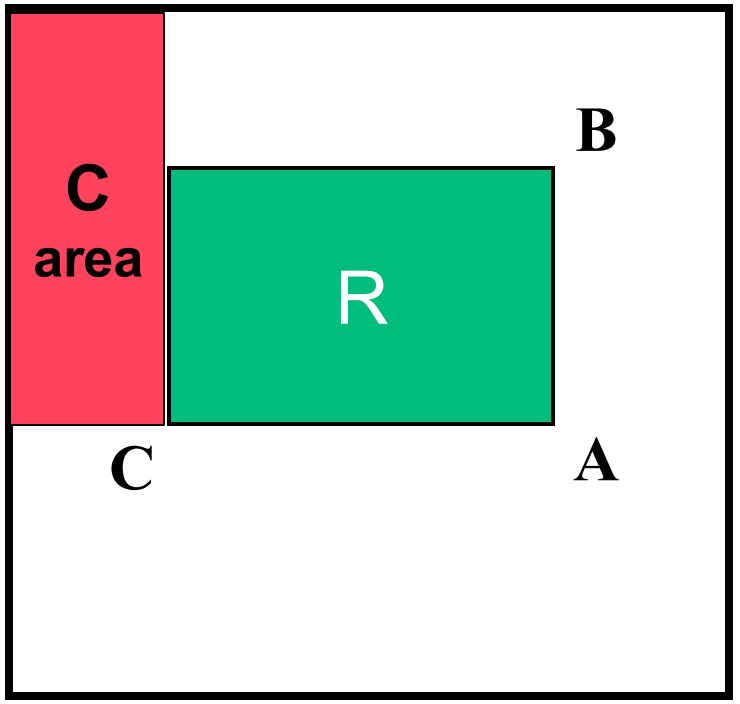
•A,B,C,D are the values of the integral image at the corners of R.



Computing sum with rectangle

•Let R be a desired rectangle.

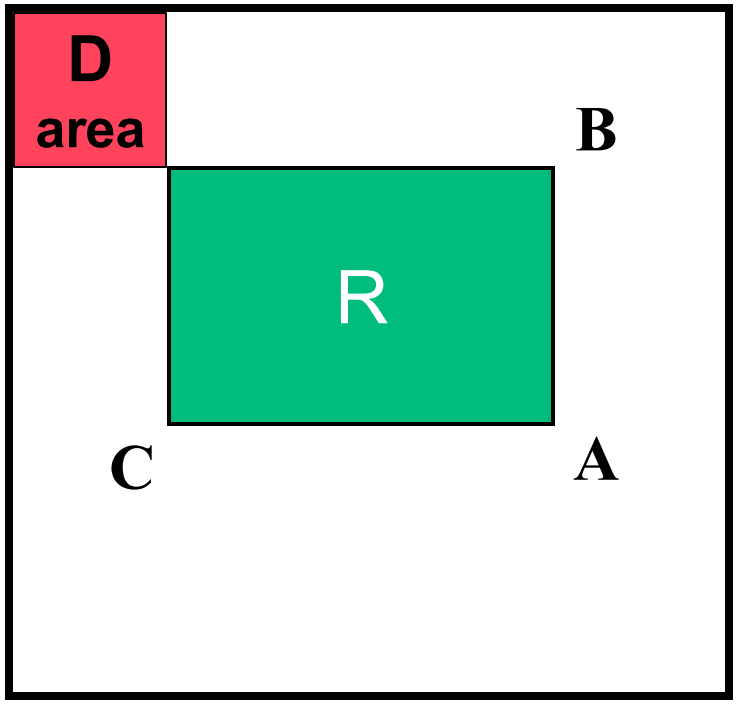
•A,B,C,D are the values of the integral image at the corners of R.



Computing sum with rectangle

- Let R be a desired rectangle.

- A,B,C,D are the values of the integral image at the corners of R.



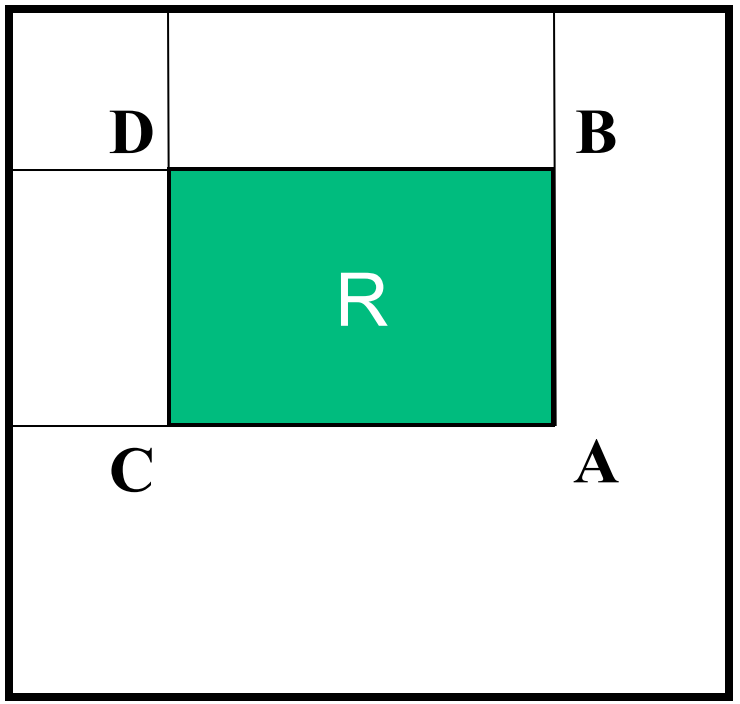
Computing sum with rectangle

•Let R be a desired rectangle.

•A,B,C,D are the values of the integral image at the corners of R.

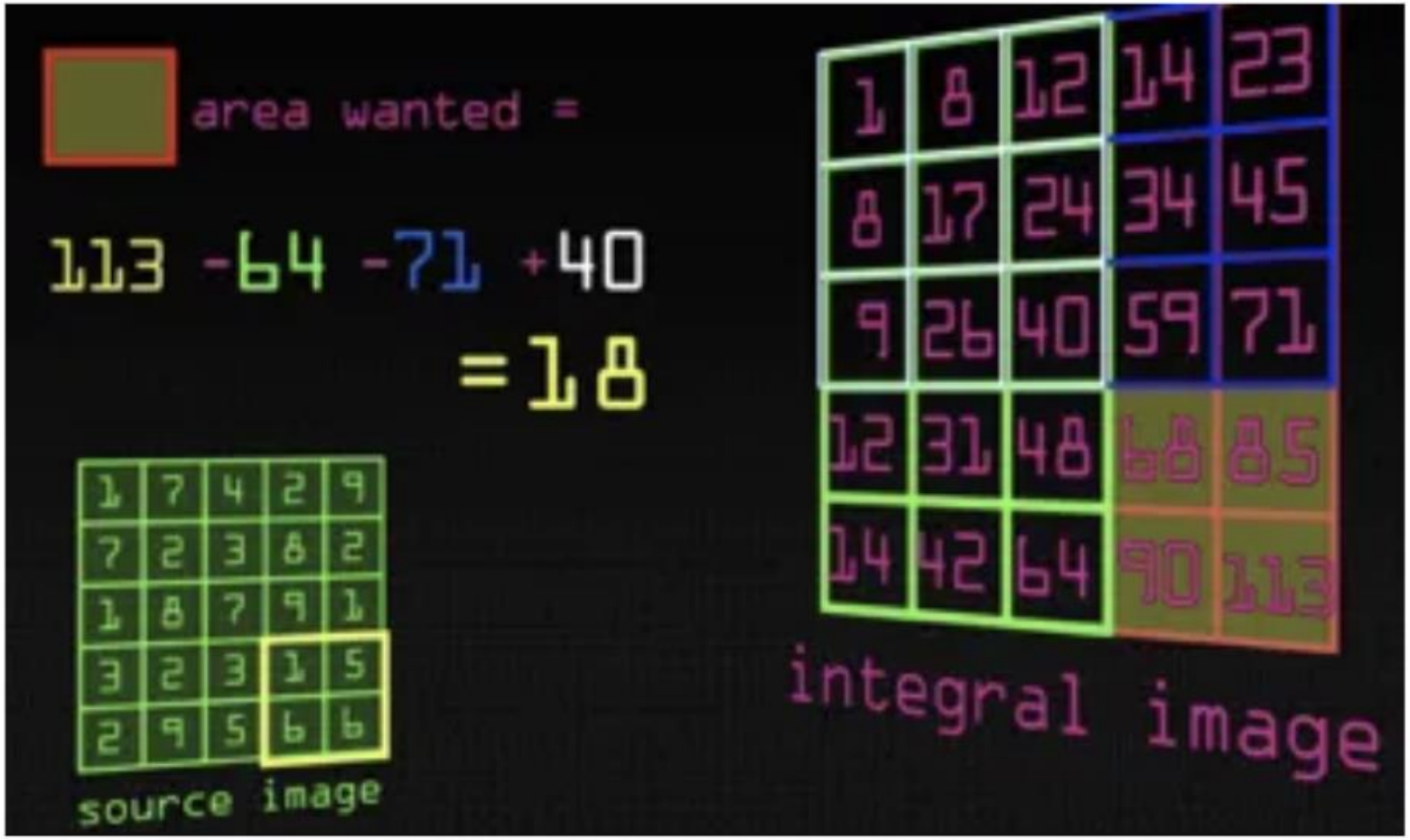
•The sum of original image values within the rectangle can be computed as:

$$sum = A - B - C + D$$



Only **3 additions** are required for any size of rectangle!

Computing the Integral images (Example)



Source: Detecting Faces (Viola Jones Algorithm) — Computerphile

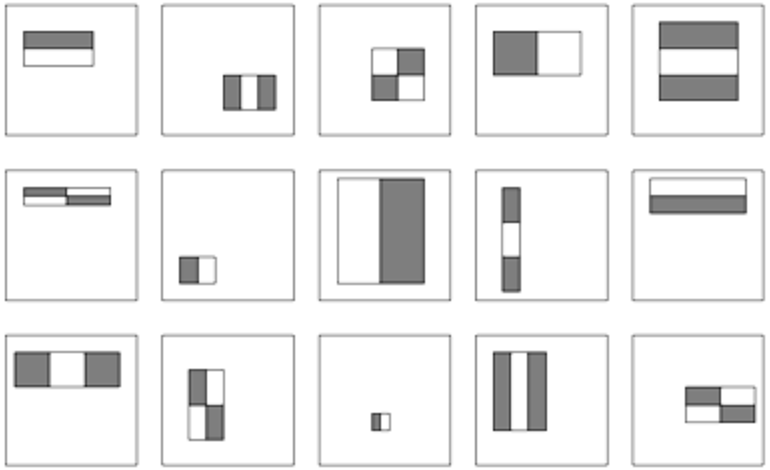
HAAR-Like Features

But these features are rubbish…! **Yes**, individually they are '**weak classifiers**'

But, what if we combine *thousands* of them…

How many features are there?

- For a 24x24 detection region, the number of possible rectangle features is ~160,000!
- At test time, it is impractical to evaluate the entire feature set.
- Can we learn a '**strong classifier**' using just a small subset of all possible features?

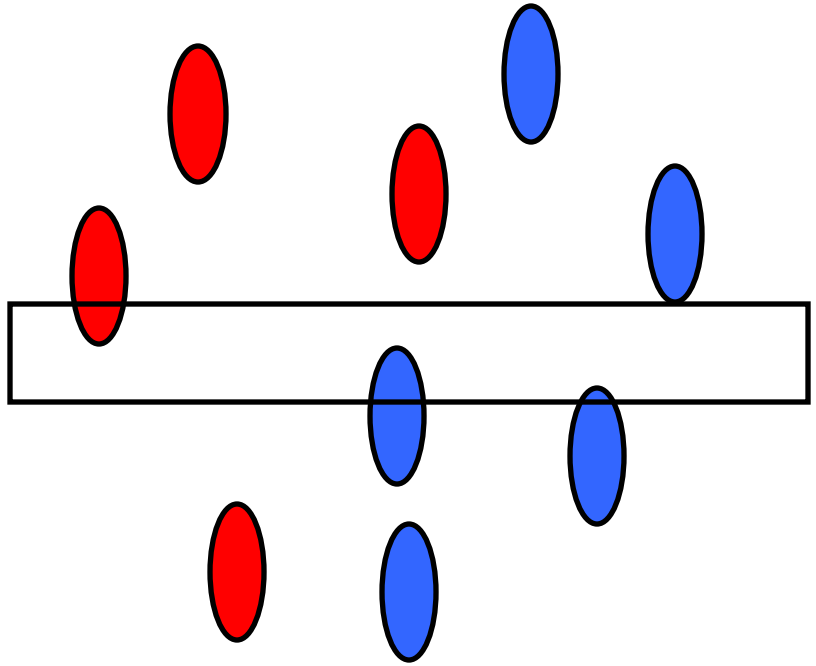


Viola-Jones algorithm uses AdaBoost to find the best features and to train a classifier. Each Haar-like feature represents a weak classifier. **The final classifier is given by a linear combination of weak classifiers. Larger weights are associated with better classifiers using the AdaBoost learning algorithm.**

3. Boosting for feature selection

Initially, weight each training example equally.

Weight = size of point



Slide credit: Paul Viola

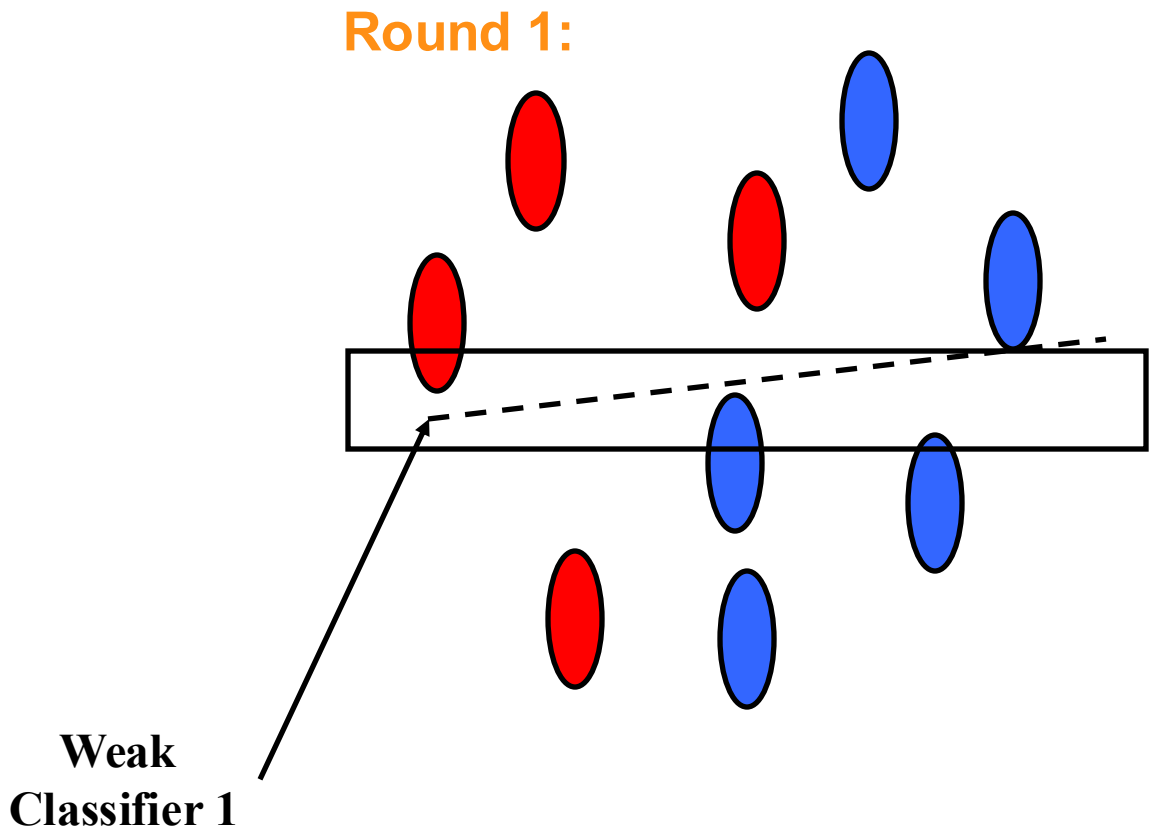
3. Boosting for feature selection

Weight = size of point

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.



Slide credit: Paul Viola

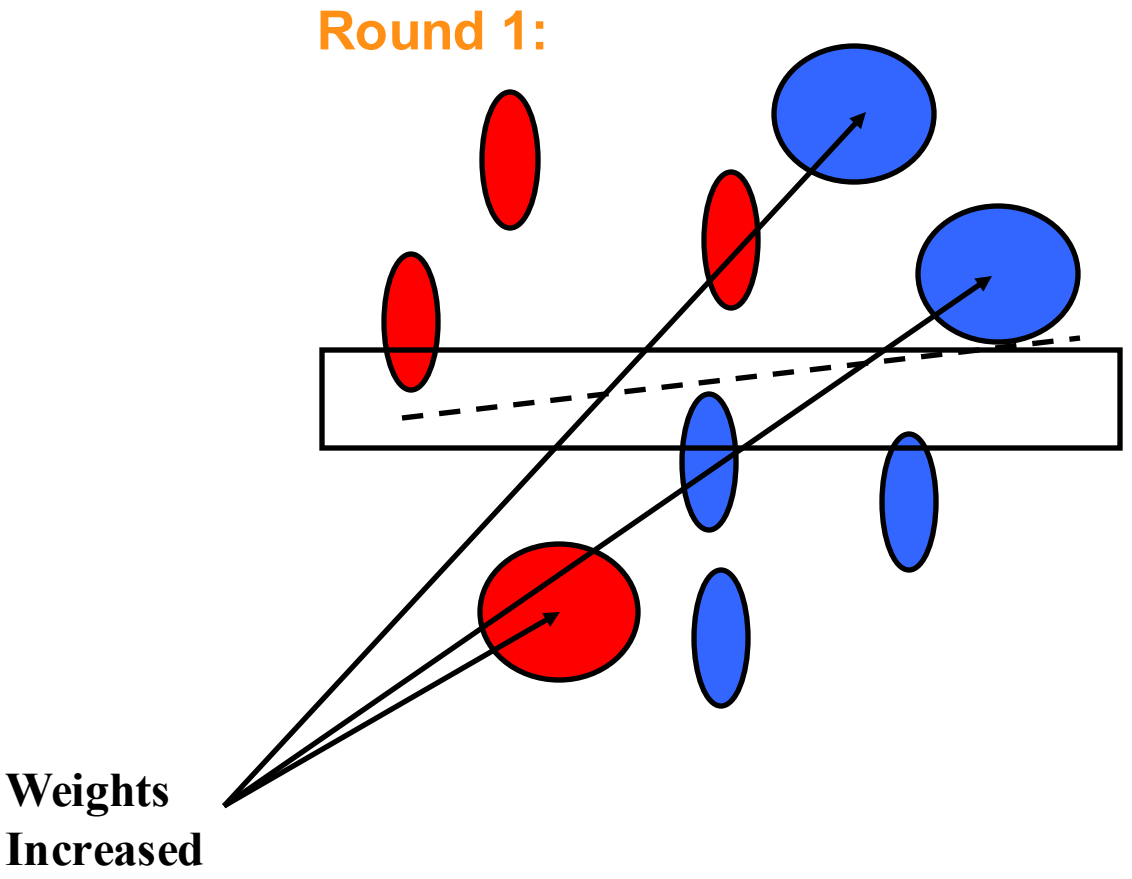
3. Boosting for feature selection

Weight = size of point

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.



Slide credit: Paul Viola

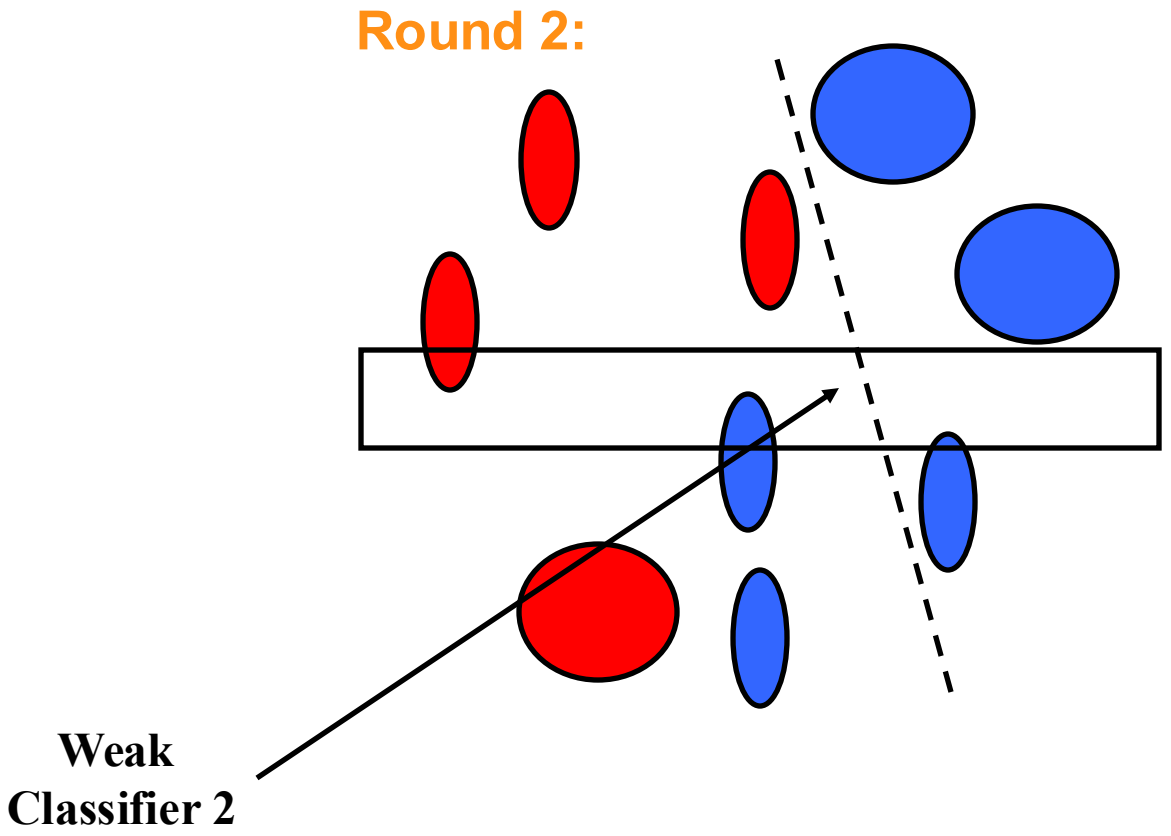
3. Boosting for feature selection

Weight = size of point

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.



Slide credit: Paul Viola

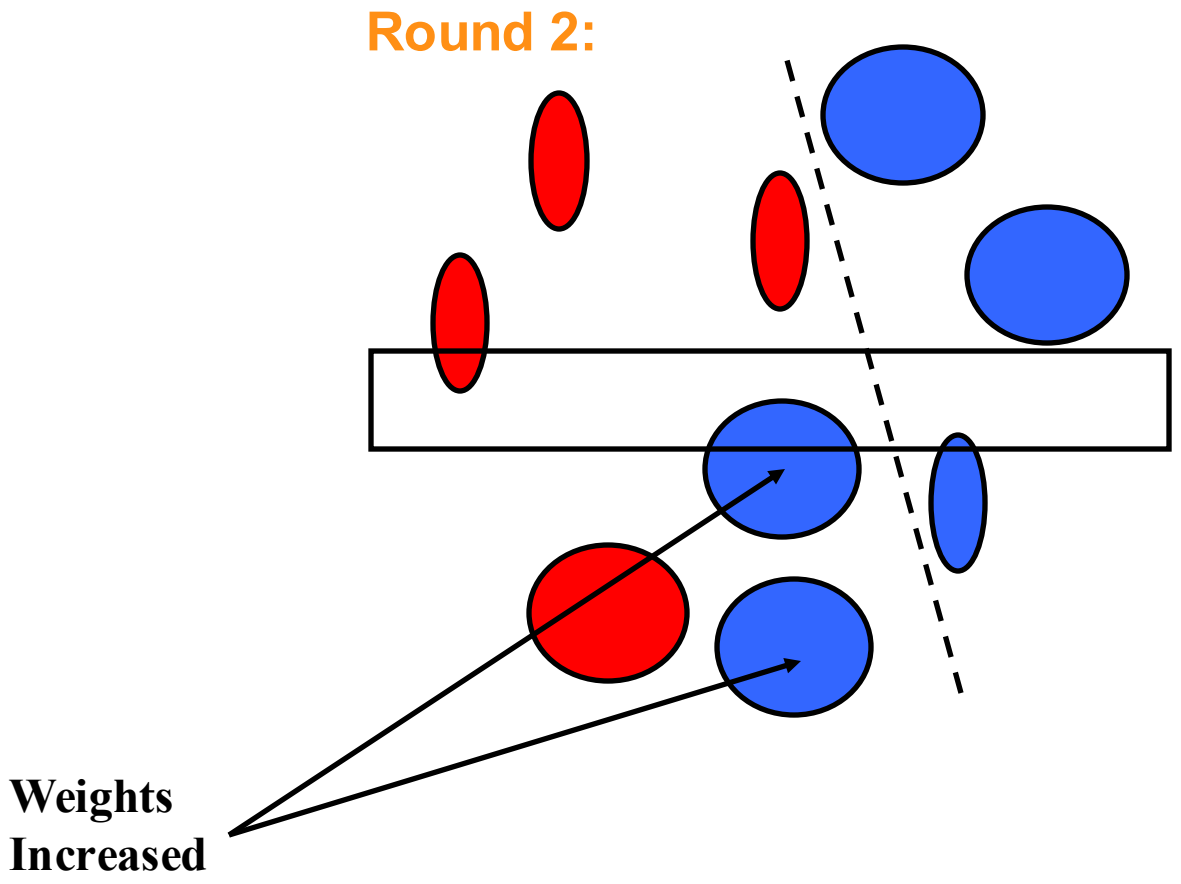
3. Boosting for feature selection

Weight = size of point

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.



Slide credit: Paul Viola

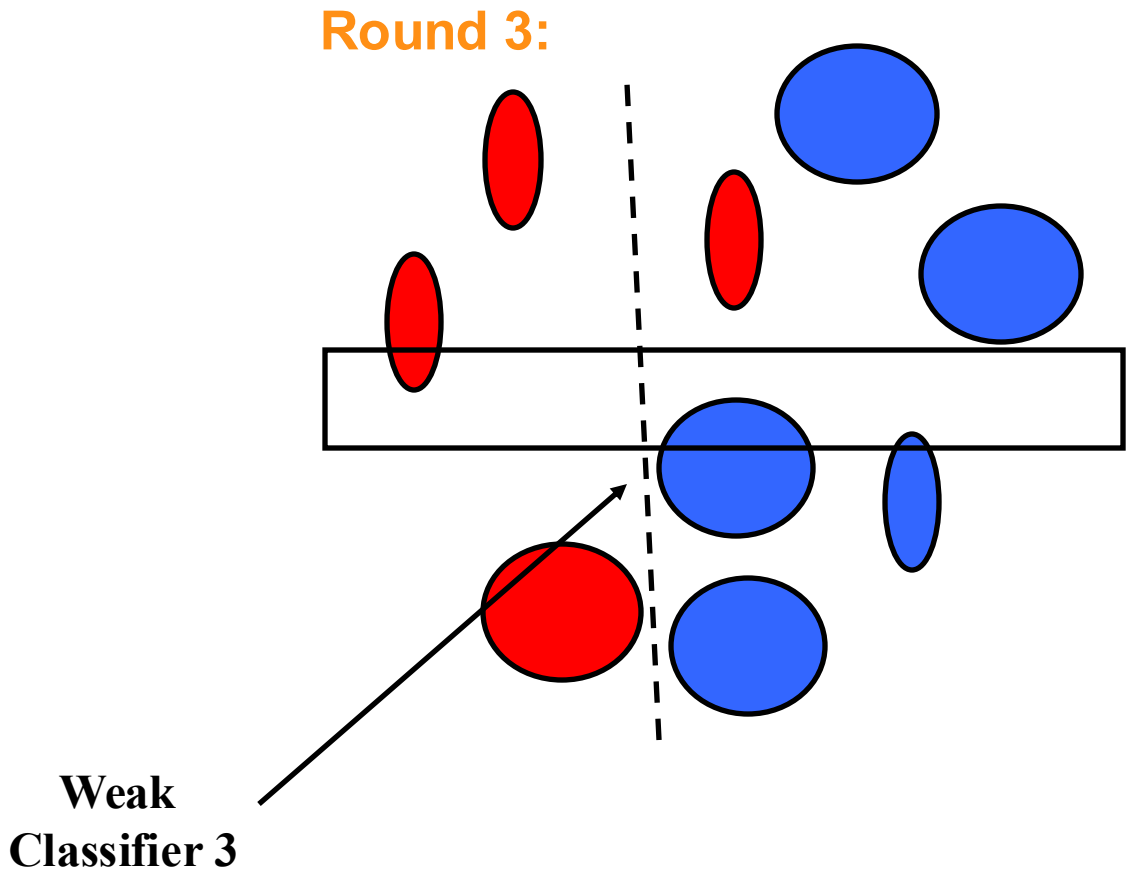
3. Boosting for feature selection

Weight = size of point

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

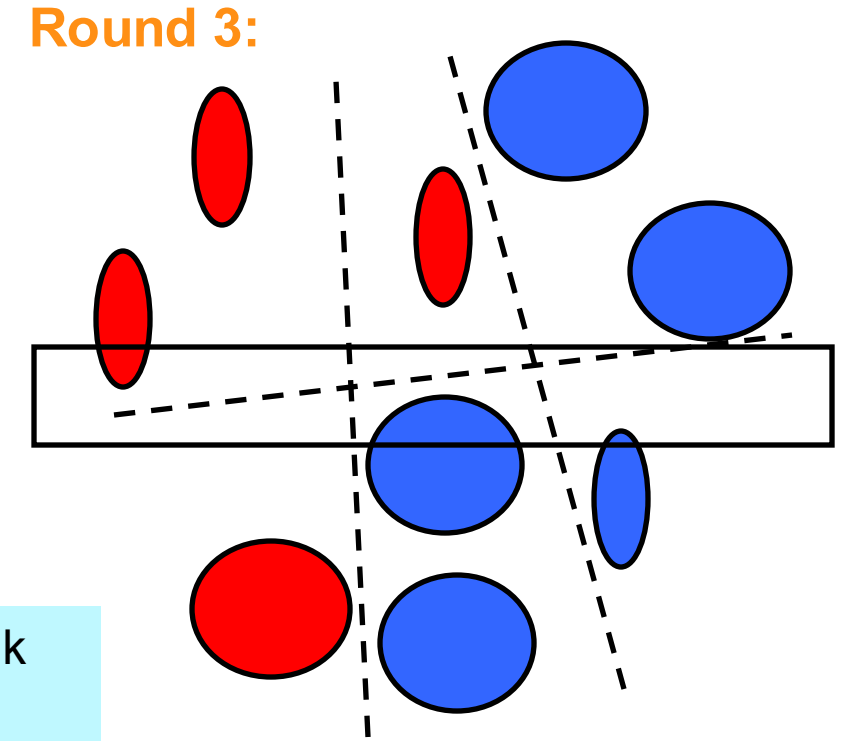


Slide credit: Paul Viola

3. Boosting for feature selection

Weight = size of point

- Compute final classifier as linear combination of all weak classifier.
- Weight of each classifier is directly proportional to its accuracy.

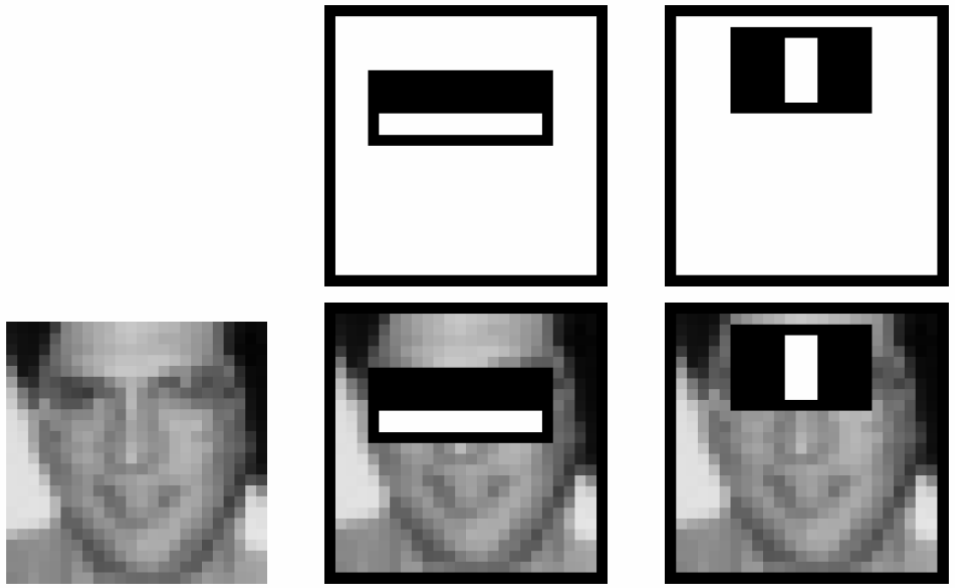


- ✓ Exact formulas for re-weighting and combining weak learners depend on the boosting scheme (e.g., **AdaBoost classifier**).
- ✓ Select discriminative features that work well together
- ✓ Choose week learner that minimize error on the weighted training set then reweight

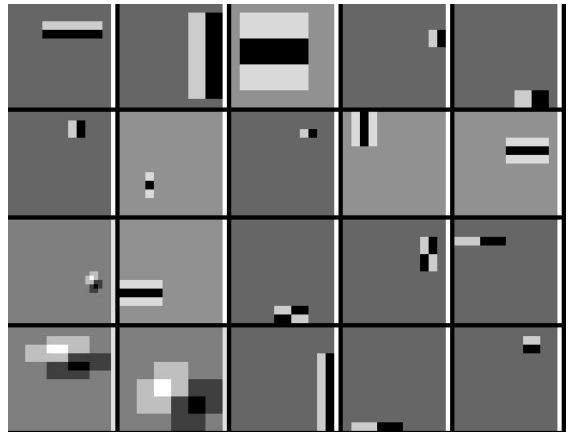
Y. Freund and R. Schapire, A short introduction to boosting, *Journal of Japanese Society for Artificial Intelligence*, 14(5):771-780, September, 1999.

3. Boosting for Face Detection

First two features selected by boosting:



Profile Features

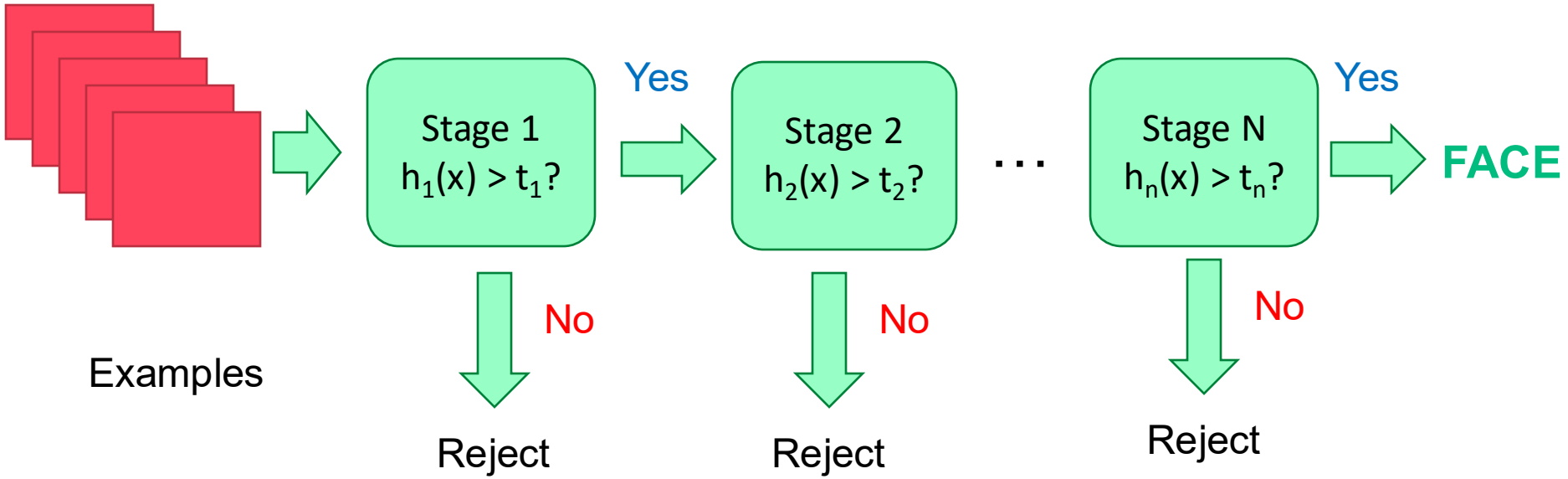


This feature combination can yield 100% recall and 50% false positive rate

Y. Freund and R. Schapire, A short introduction to boosting,
Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, September, 1999.

Slide credit: Paul Viola

4. Attention Cascade for Fast Detection



Fast classifiers early in cascade which reject many negative examples but detect almost all positive examples.

Slow classifiers later, but most examples don't get there.

4. Attention Cascade for Fast Detection

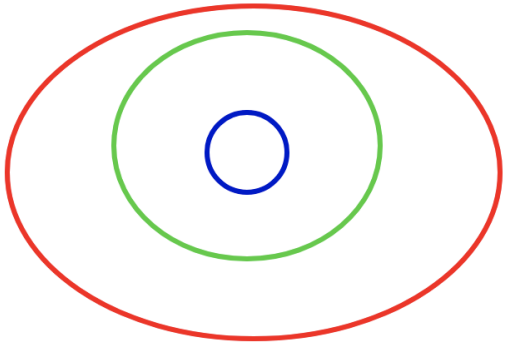
- Chain classifiers that are progressively more complex
- Minimize *false positive rates* at each stage, not absolute error

✓ A 1 feature classifier achieves 100% detection rate and about 50% false positive rate.

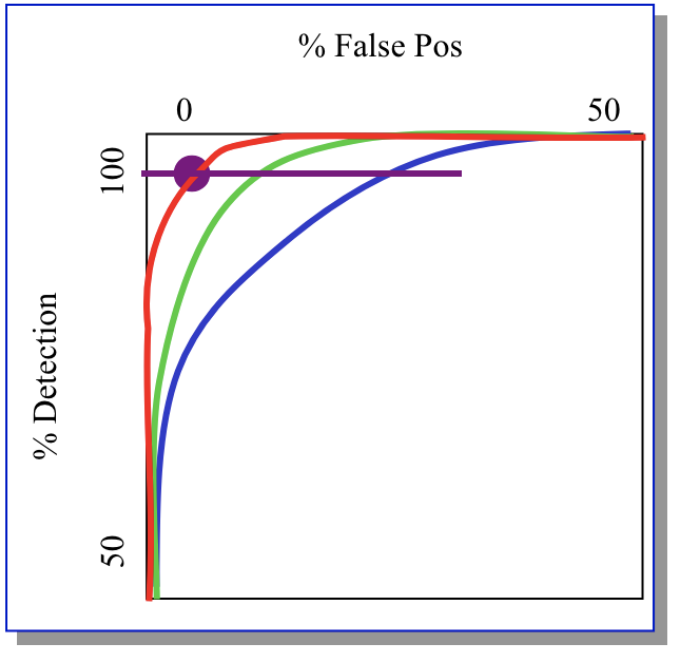
✓ A 5 feature classifier achieves 100% detection rate and 40% false positive rate (20% cumulative)

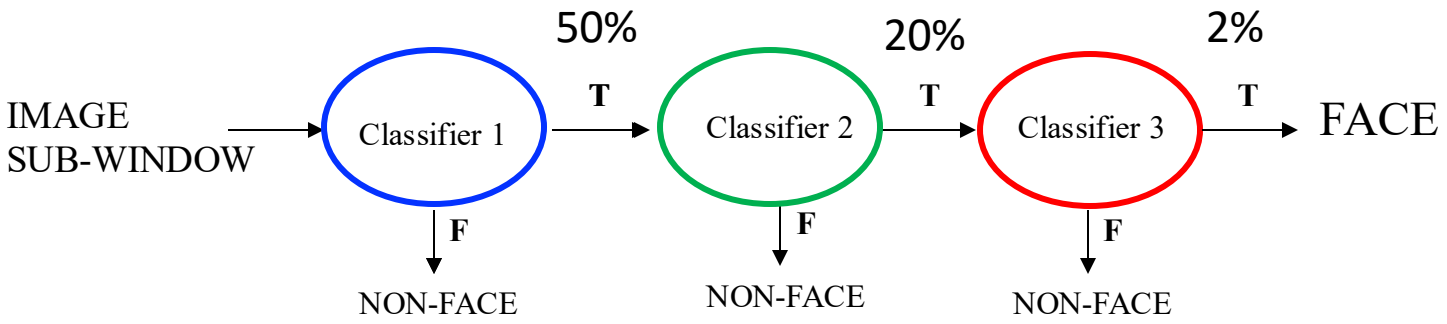
  ✓ using data from previous stage.

✓ A 20 feature classifier achieve 100% detection rate with 10% false positive rate (2% cumulative)



Receiver operating characteristic (ROC)





4. Attention Cascade for Fast Detection

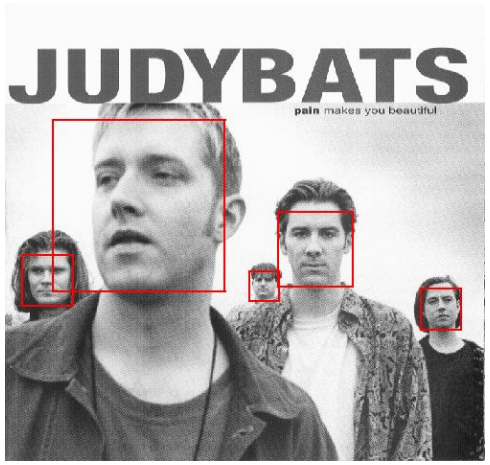
Training the cascade

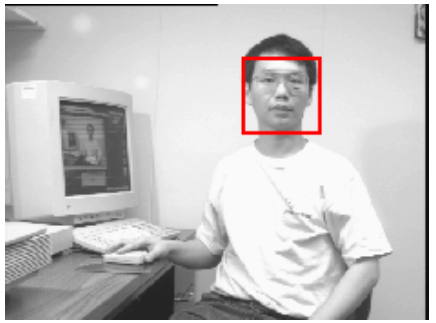
1) Set target detection and false positive rates for each stage
2) Keep adding features to the current stage until its target rates have been met
   a. Need to lower boosting threshold to maximize detection (as opposed to minimizing total classification error)
   b. Test on a *validation set*
3) If the overall false positive rate is not low enough, then add another stage
4) Use false positives from current stage as the negative training examples for the next stage

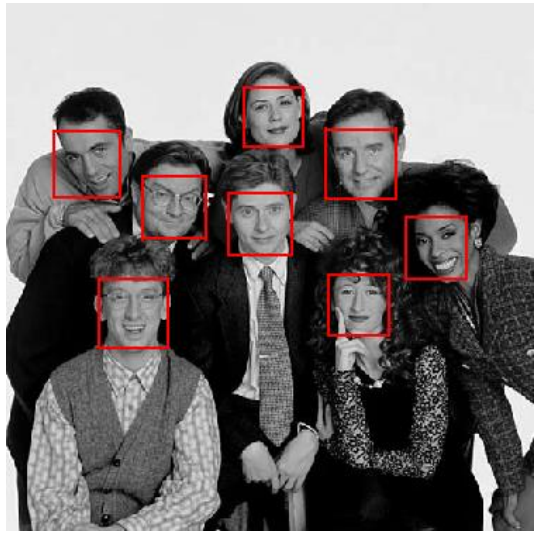
The Implemented System

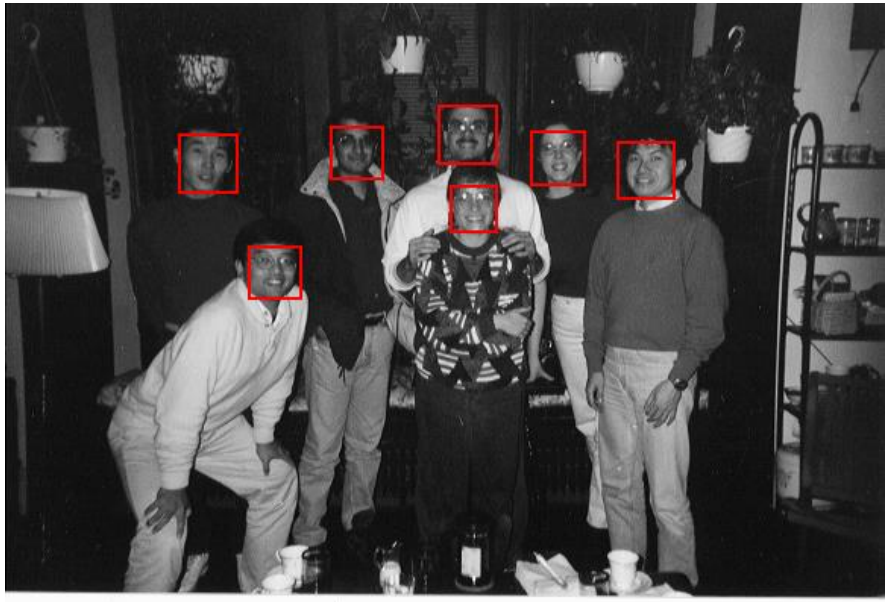
- **Training Data**
  - 5000 faces
    - All frontal, rescaled to 24x24 pixels
  - 300 million non-faces
    - 9500 non-face images
  - Faces are normalized
    - Scale, translation
- **Many variations**
  - Across individuals
  - Illumination
  - Pose

Output of Face Detector on Test Images

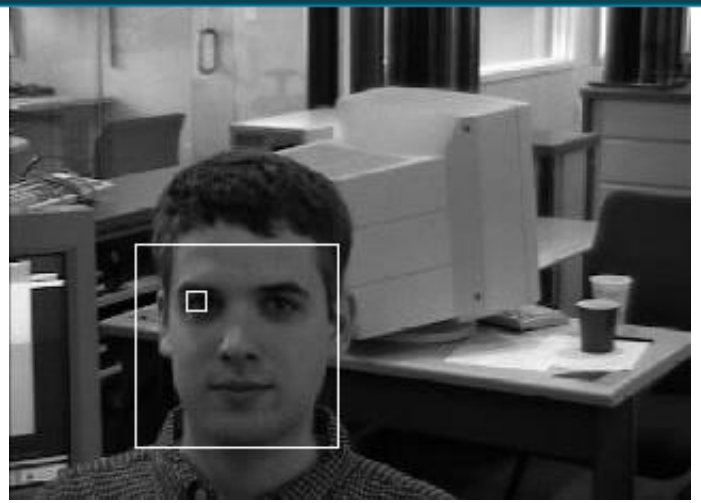




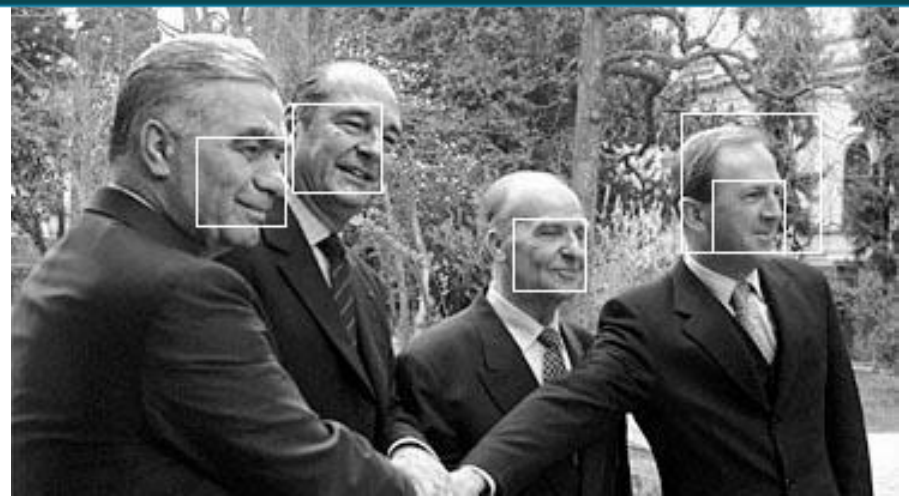




Other Detection Tasks

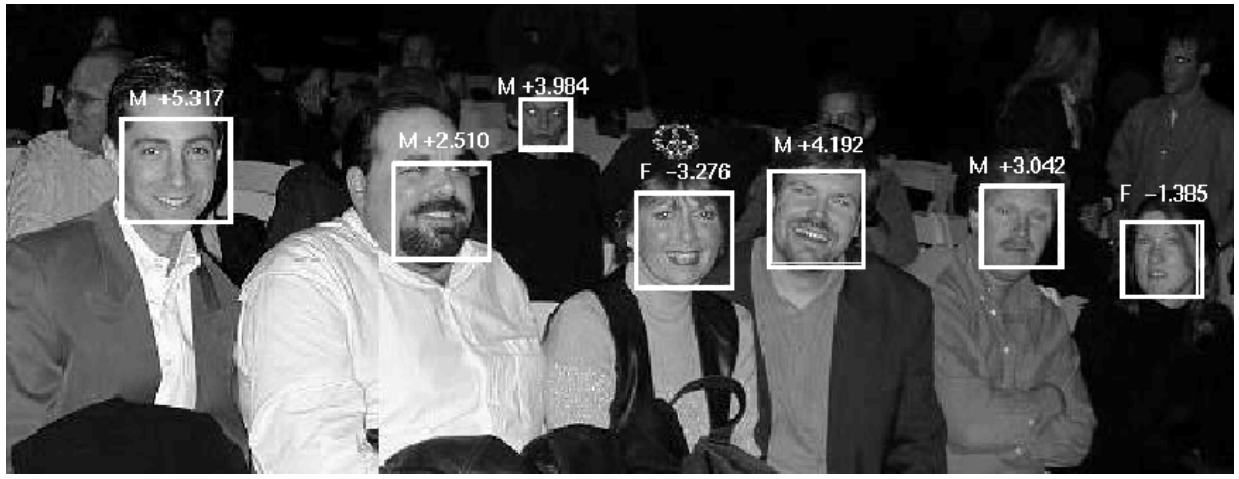


Facial Feature Localization



Profile Detection

Male vs. female



Face Recognition

- **Face Recognition** consists of a two-phase process:

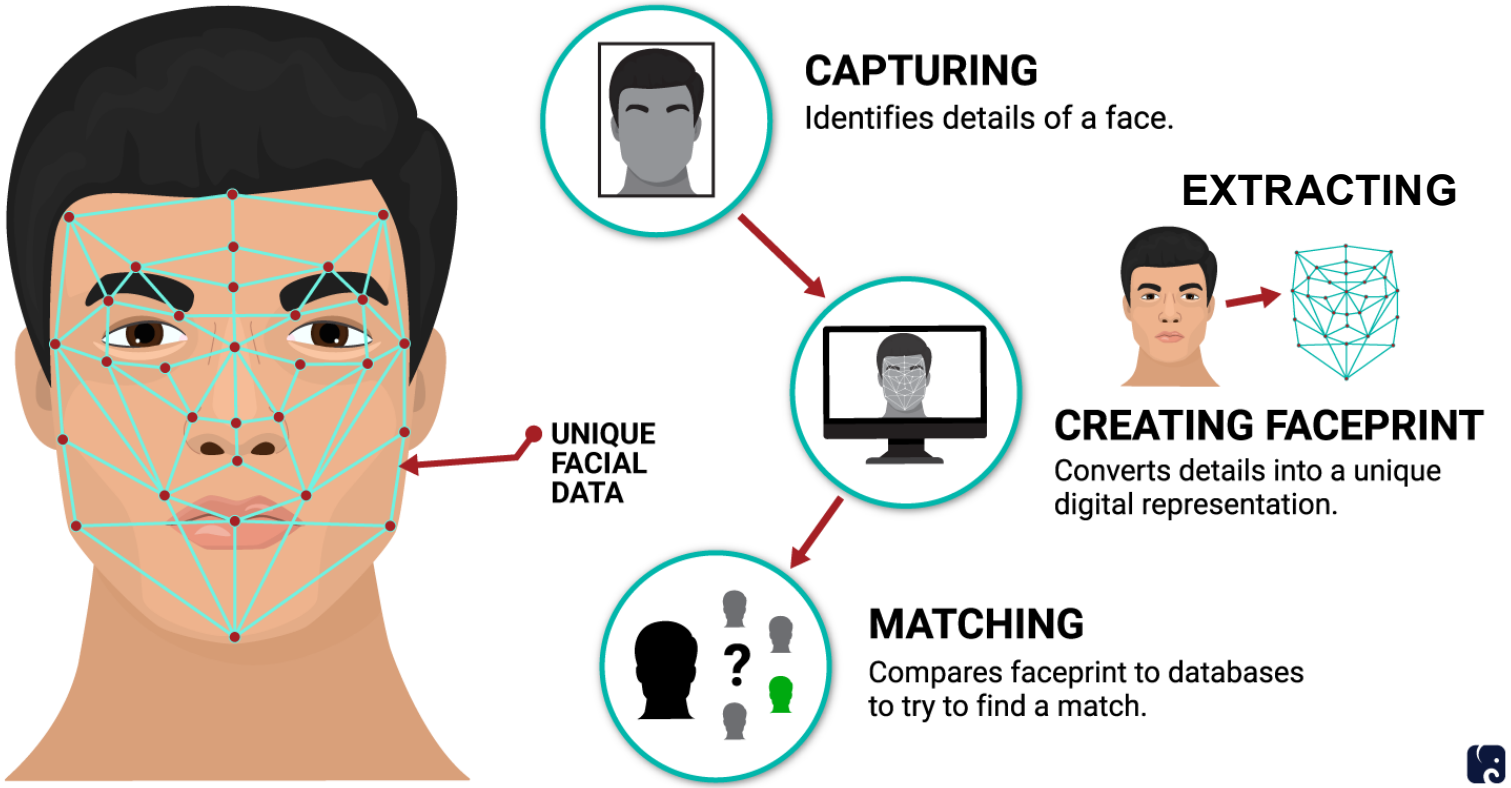
  Phase #1:

  Detect the presence of faces in an image or video stream using methods such as Viola Jones, Haar cascades, HOG + Linear SVM, DL, or any other algorithms that can localize faces.

  Phase #2:

  Take each of the faces detected during the **localization phase** and identify each of them — this is where we actually **assign a *name to a face***.

Face Recognition: Basic idea



Face recognition has become a popular area of research in computer vision and one of the most successful applications of image analysis and understanding.

https://stoplapdspying.org/facial-recognition-factsheet/

Face Recognition

1. Treat pixels as a vector

$\mathbf{x}$

2. Recognize face by nearest neighbor

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

$$\min_k \left\| \mathbf{y}_k^T - \mathbf{x} \right\|$$

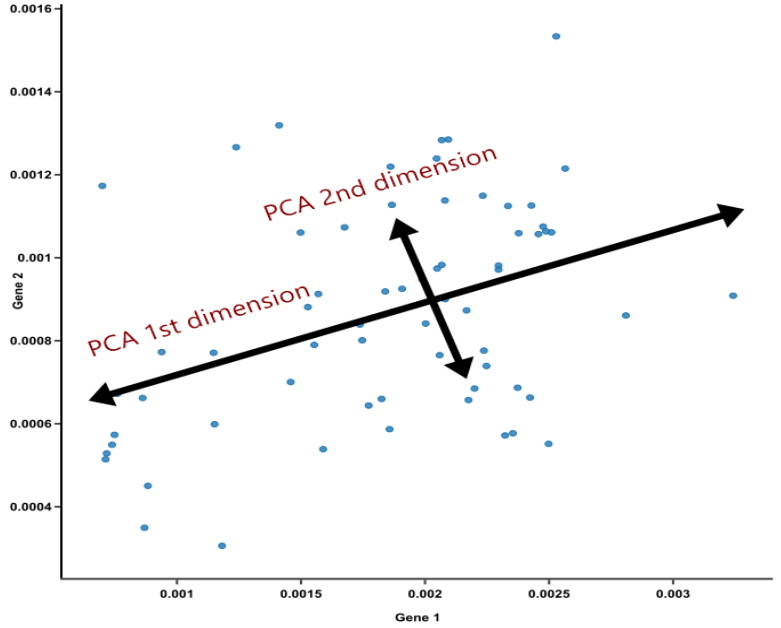
The space of face image

- When viewed as vectors of pixel values, face images are extremely high-dimensional
  - 100x100 image = 10,000 dimensions
  - Large memory and computational requirements
- We want to reduce dimensionality and effectively model the subspace of face images

Face Recognition

- In 1987, *Kirby* and *Sirovich* published their work "*A Low-Dimensional Procedure for the Characterization of Human Faces*". The proposed algorithm is called: ***Eigenfaces Algorithm.***
  - They demonstrated that a **standard linear algebra technique for dimensionality reduction**, **Principal Component Analysis (PCA)**, could be used **to identify a face using a a low-dimensional representation of face images** (i.e., feature vector smaller than 100-dim).
  - Furthermore, the "**principal components**" (i.e., the eigenvectors, or the "**eigenfaces**") could be used to actually ***reconstruct*** faces from the original dataset.
  - This implies that a face could be represented (and identified) as a ***linear combination*** **of the eigenfaces.**

So, first, let's take a look what is **PCA Algorithm** and how does it work?



Principal Component Analysis (PCA)

Pattern recognition in high-dimensional spaces.

- 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.**

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{bmatrix} \longrightarrow \text{dimensionality reduction} \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \cdots \\ z_K \end{bmatrix}$$

where K << N.

Principal Component Analysis (PCA)

- The goal of PCA is to reduce the dimensionality of the data while retaining as much information as possible in the original dataset.
- PCA computes a **linear transformation** that **maps** data from a high dimensional space to a lower dimensional sub-space KxN.

$$z_1 = u_{11}x_1 + u_{12}x_2 + \ldots + u_{1N}x_N$$
$$z_2 = u_{21}x_1 + u_{22}x_2 + \ldots + u_{2N}x_N$$
$$\ldots$$
$$z_K = u_{K1}x_1 + u_{K2}x_2 + \ldots + u_{KN}x_N$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{bmatrix}$$

- In short,

$$\mathbf{z} = \mathbf{W}\mathbf{x} \quad \text{where} \quad \mathbf{W} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1N} \\ u_{21} & u_{22} & \cdots & u_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ u_{K1} & u_{K2} & \cdots & u_{KN} \end{bmatrix}$$

Principal Component Analysis (PCA)

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

(1) Higher-dimensional space representation

$$\mathbf{x} = x_1\mathbf{v}_1 + x_2\mathbf{v}_2 + \cdots + x_N\mathbf{v}_N$$

$\mathbf{v}_1, \cdots, \mathbf{v}_N$ are the basis vectors of the N-dimensional space

(2) Lower-dimensional space representation

$$\hat{\mathbf{x}} = z_1\mathbf{u}_1 + z_2\mathbf{u}_2 + \cdots + z_K\mathbf{u}_K$$

$\mathbf{u}_1, \cdots, \mathbf{u}_K$ are the basis vectors of the K-dimensional space

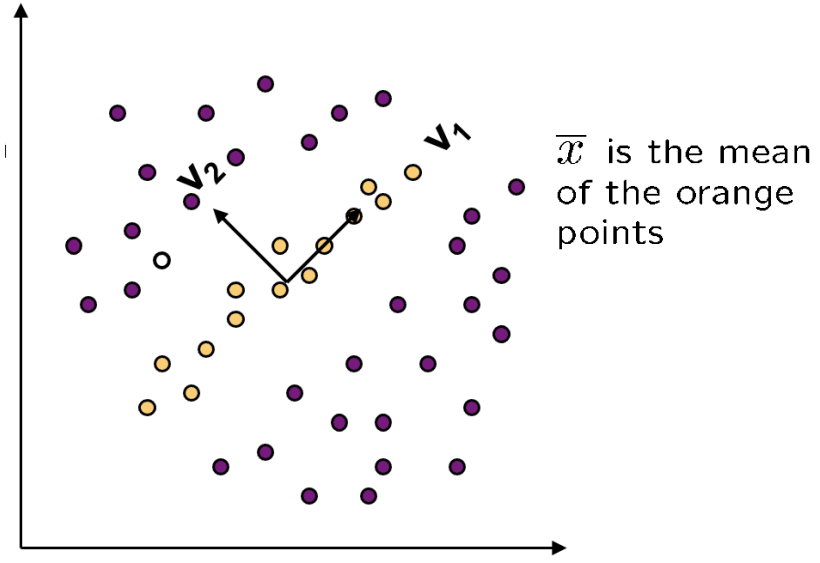
Note: If N=K, then $\mathbf{x} = \hat{\mathbf{x}}$

Principal Component Analysis (PCA) is a mathematical algorithm that helps to find patterns in data by identifying the most important features "**components**" that explain the variation in the data.

PCA: Dimensionality Reduction

- **Geometric interpretation**
  - PCA projects the data along the directions where the data varies the 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.

- **Dimensionality reduction**
  - We can represent the yellow 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



41

PCA: Dimensionality Reduction

- **Lower dimensionality basis**
  - Dimensionality reduction implies information loss!
  - **PCA preserves as much information** as possible by

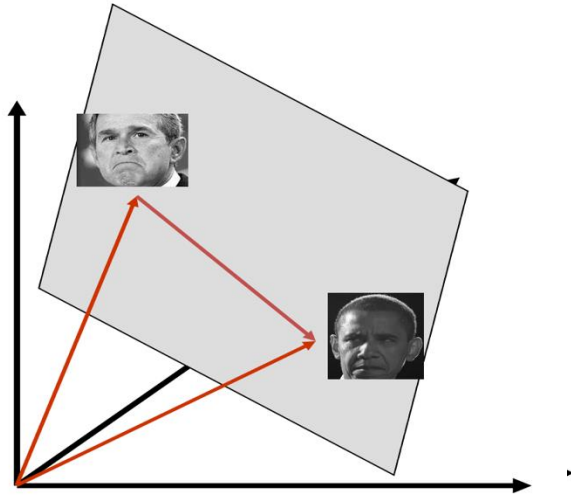
    Minimizing the error: $\|x - \hat{x}\|$

How should we determine the **best lower dimensional sub-space?**

The best low-dimensional space can be determined by the "**best**" **eigenvectors** of the **covariance matrix** of x (i.e., the eigenvectors corresponding to the "**largest**" **eigenvalues** – also called "***principal components***"

PCA in Face Recognition

- **An image is a point in a high dimensional space**
  - An N x M image is a point in R^{NM}
  - We can define vectors in this space.



Key IDEA: Images in the possible set are highly correlated, and the set of faces is a "subspace" of the set of images

➢ compress them to a low-dimensional subspace that captures key appearance characteristics. Suppose it is K dimensional.

✓ We can find the best subspace using **PCA**

✓ This is like fitting a "hyper-plane" to the set of faces

- spanned by vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_K}$

EIGENFACES:

["Eigenfaces for Recognition", Turk and Pentland, 1991]

Any face:

$$\mathbf{x} \approx \bar{\mathbf{x}} + a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_k\mathbf{v}_\mathbf{k}$$

- PCA can be used to identify the most important **facial features** that **distinguish one face from another** by analyzing a set of images of faces and identifying the most important features *(such as the distance between the eyes, the shape of the nose, etc).*
- Once these features have been identified, **PCA can be used to simplify the data by projecting the faces onto a lower-dimensional space of the most important features.** This can make it easier to compare faces and recognize them, since the irrelevant features have been removed.

Eigenfaces Algorithm

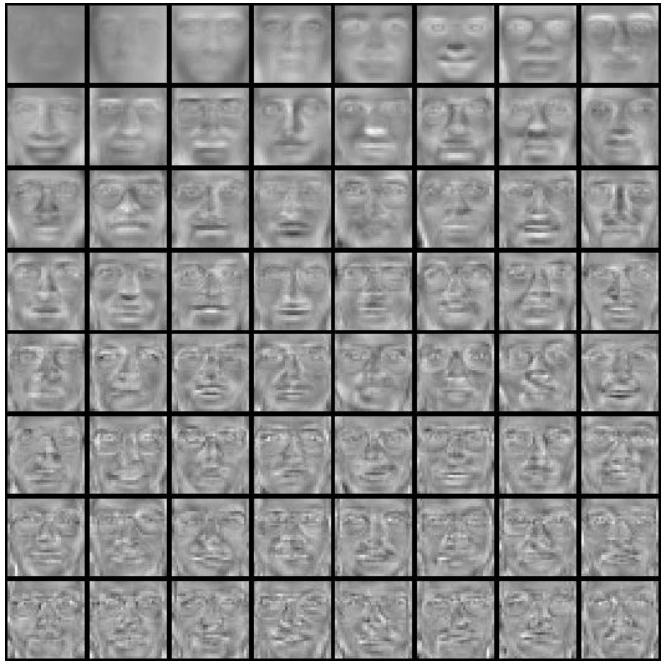
Eigenfaces look somewhat like generic faces.

Eigenfaces are the eigenvectors of the covariance matrix of the probability distribution of the vector space of human faces.

Mean: μ

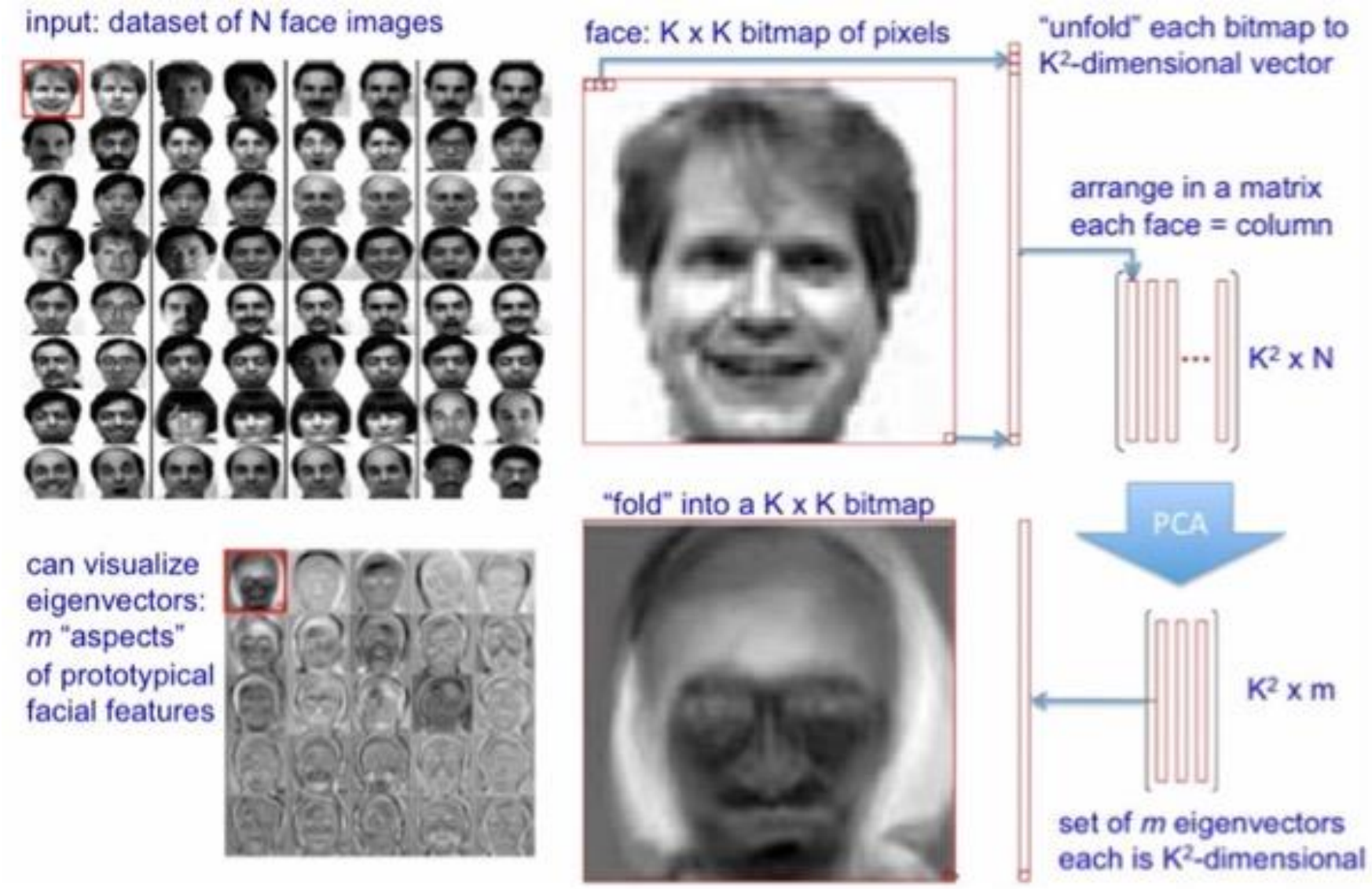
Training Images X1, …, Xn

Top eigenvectors: $u_1,\ldots u_k$



- Eigenfaces are the ‘standardized face ingredients’ derived from the statistical analysis of many pictures of human faces
- A human face may be considered to be a combination of these standardized faces.

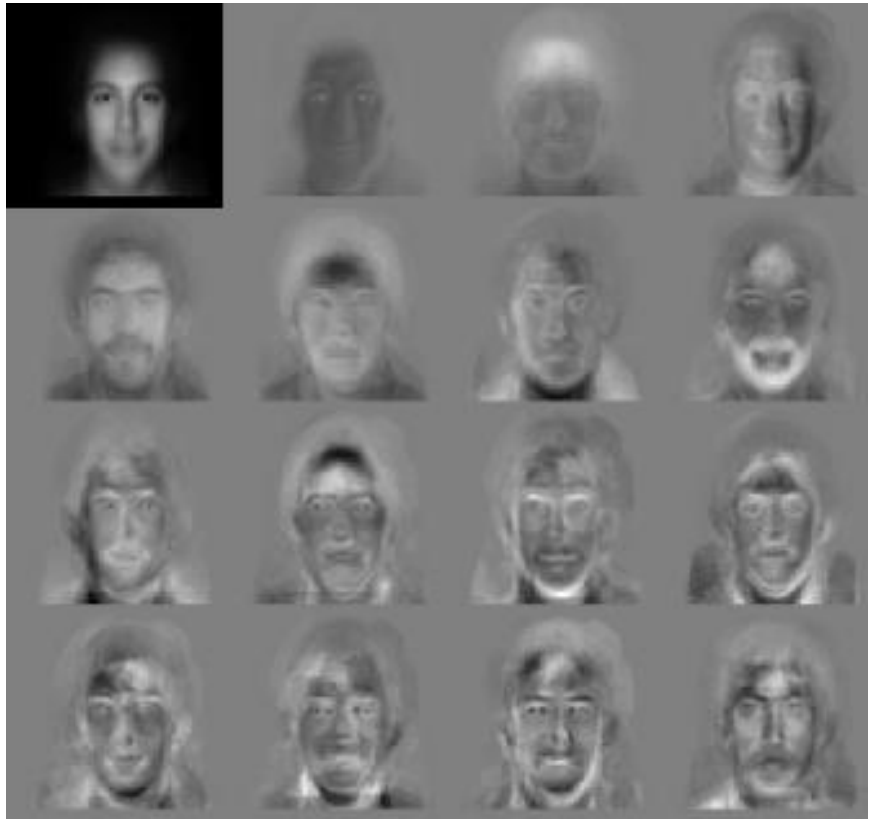
Eigenfaces Algorithm using PCA: An Overview



Eigenfaces Algorithm: Summary in words

Generating Eigenfaces:

1. Large set of images of human faces is taken.
2. The images are normalized to line up the eyes, mouths and other features.
3. The eigenvectors of the covariance matrix of the face image vectors are then extracted.
4. These eigenvectors are called **eigenfaces**.



Eigenfaces for Face Recognition:

- When properly weighted, eigenfaces can be summed together to create an approximate gray-scale rendering of a human face.
- A few eigenvector terms are needed to give a fair likeness of most people's faces.
- Eigenfaces provide a means of applying data compression to faces for identification purposes.

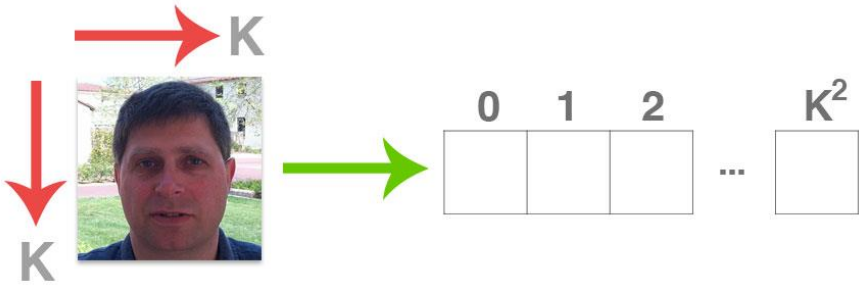
PCA extracts the **eigenvectors** of **A**

- Gives a set of vectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, ...
- Each one of these vectors is a direction in face space

Eigenfaces Algorithm – More details

- The first step in the Eigenfaces algorithm is to: **input a dataset of *N* face images.**
  - In order for face recognition to be successful and robust, we should ensure we have ***multiple*** **images per person** we want to recognize. Essentially, this first step is to construct our training set of images.
- Each **face is represented as a grayscale, *K x K* bitmap of pixels.**
- In order to apply the Eigenfaces algorithm, we need to **form a *single vector* from the image**. This is accomplished by **"flattening" each image into a K^2 -dimension vector.**

- After each image in the dataset has been flattened, we form a matrix of flattened images like M. Where *Z* is the number of images in our dataset. Our *entire dataset* is now contained in a *single* matrix *M*.
- Given this matrix ***M***, we are now ready to apply: the **Principal Component Analysis (PCA), the basis of the Eigenfaces algorithm.**



$$M = \begin{pmatrix} \square\square\square & \cdots & \square \\ \square\square\square & \cdots & \square \\ & \vdots & \\ \square\square\square & \cdots & \square \end{pmatrix}$$

$Z \times K^2$

Eigenfaces Algorithm - Steps

1. Compute the mean μ_i of each column in the matrix, giving us the average pixel intensity value for ***every*** *(x, y)* - coordinate in the image dataset.
2. Subtract the μ_i from each column c_i — this is called ***mean centering*** the data and is a required step when performing PCA.
3. Now that our matrix *M* has been mean centered, compute the ***covariance matrix.***
4. Perform an **eigenvalue decomposition** on the covariance matrix to get the **eigenvalues** λ_i and **eigenvectors** $|\lambda_i|$.
5. **Sort** $\mathbf{X_i}$ by x_i , largest to smallest.
6. Take the top *N* eigenvectors with the largest corresponding eigenvalue magnitude.
7. Transform the input data by projecting (i.e., taking the dot product) it onto the space created by the top *N* eigenvectors — these eigenvectors are called our ***eigenfaces***.

Eigenfaces Algorithm - Steps

8. Each row in the matrix is an eigenface with K^2 entries — *exactly like our original image*. **What does this mean**? Well, since each of these eigenface representations is actually a K^2 vector, we can reshape it into a *K x K* bitmap.

$$V = \begin{pmatrix} \square\square\square \cdots \square \\ \square\square\square \cdots \square \\ \vdots \\ \square\square\square \cdots \square \end{pmatrix}$$

$N \times K^2$

9. After applying an eigenvalue decomposition of the matrix *m*, we are left with a **matrix *v*, containing *n* rows (our eigenvectors) each of dimensionality *k-squared*.**

Eigenfaces Algorithm

- The image on the *left* is simply the average of *all* faces in our dataset, while the figures on the *right* show the most prominent deviations from the mean in our face dataset.
- This can be thought of as a *visualization* of the dimension in which people's faces vary the most.
- Lighter regions correspond to higher variation, where darker regions correspond to little to no variation.
- Here, we can see that our eigenface representation captures considerable variance in the eyes, hair, nose, lips, and cheek structure.

(LEFT) mean face image. *(RIGHT)* the top 16-eigenface representations

PCA & Eigenfaces Algorithm Summary

- PCA is a general dimensionality reduction technique
- Preserves most of variance with a much more compact representation
  - Lower storage requirements (eigenvectors + a few numbers per face)
  - Faster matching

What are the problems for face recognition?

Eigenfaces Algorithm

LIMITATIONS:

- ✗ Background changes cause problems.
- ✗ Light changes degrade performance.
  - Light normalization helps.
- ✗ Performance decreases quickly with changes to face size.
  - Scale input image to multiple sizes.
- ✗ Performance decreases with changes to face orientation.
- ✗ Not robust to misalignment.
- ✗ PCA assumes that the data follows a Gaussian distribution.

Practical Lesson

Syntax

```
detector = vision.CascadeObjectDetector
detector = vision.CascadeObjectDetector(model)
detector = vision.CascadeObjectDetector(XMLFILE)
detector = vision.CascadeObjectDetector(Name,Value)
```

Live Demo

Detect Faces in an Image Using the Frontal Face Classification Model

Create a face detector object.

```
faceDetector = vision.CascadeObjectDetector;
```

Read the input image.

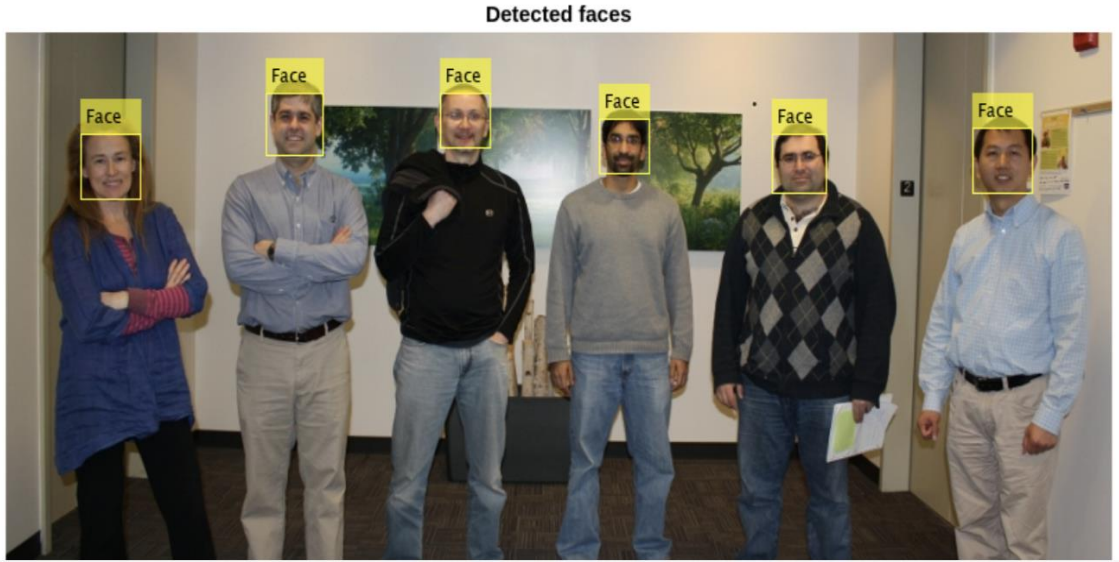
```
I = imread('visionteam.jpg');
```

Detect faces.

```
bboxes = faceDetector(I);
```

Annotate detected faces.

```
IFaces = insertObjectAnnotation(I,'rectangle',bboxes,'Face');
figure
imshow(IFaces)
title('Detected faces');
```



https://www.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html

References

- Computer Vision © James Hays, Brown University
- First Principles of Computer Vision, Shree Nayar, Computer Science Department, School of Engineering and Applied Sciences, Columbia University.
- Richard Szeliski, Computer Vision: Algorithms and Applications
- https://www.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html