Eigen Representations: Detecting faces in images

Class 6. 15 Sep 2011

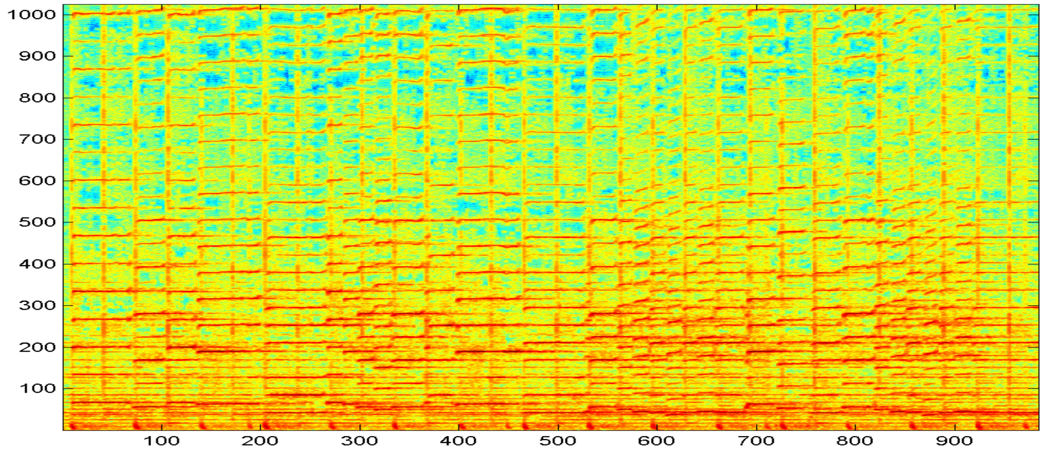
Instructor: Bhiksha Raj

Administrivia

- Project teams?
- Project proposals?

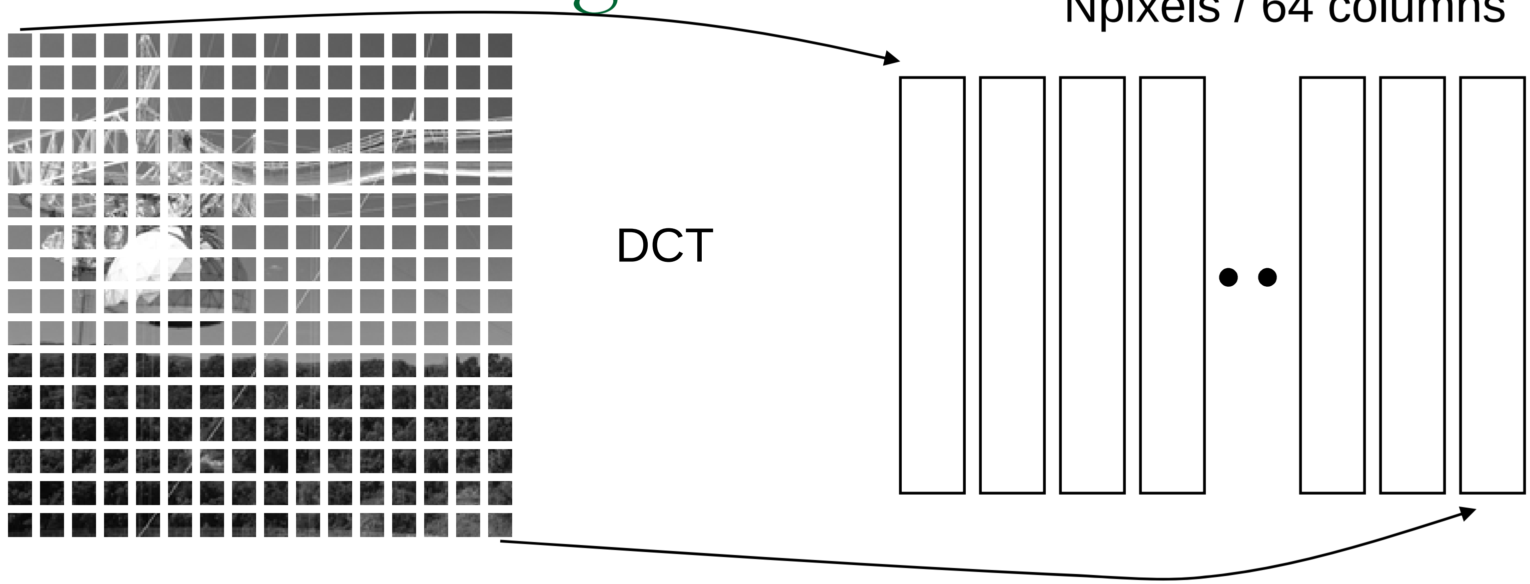
- TAs have updated timings and locations (on webpage)

Last Lecture: Representing Audio



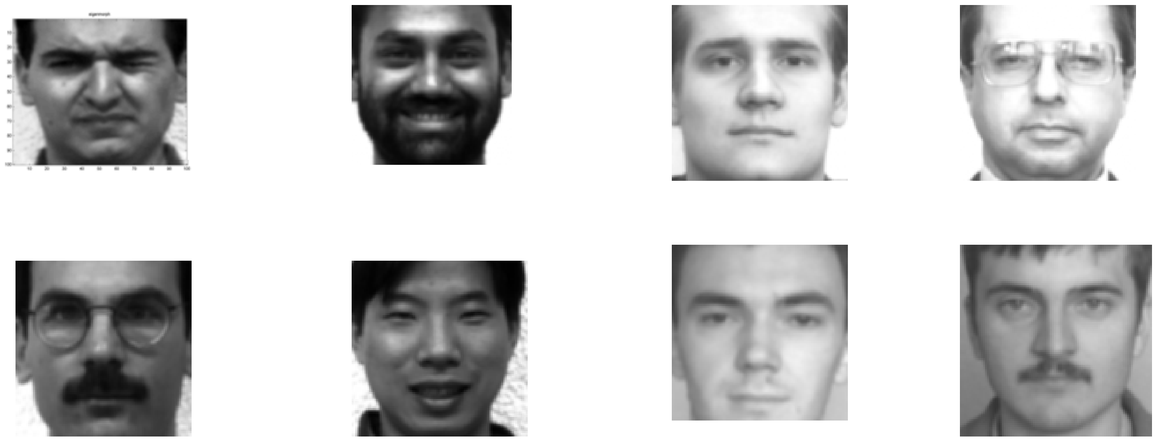
- Basic DFT
- Computing a Spectrogram
- Computing additional features from a spectrogram

What about images?



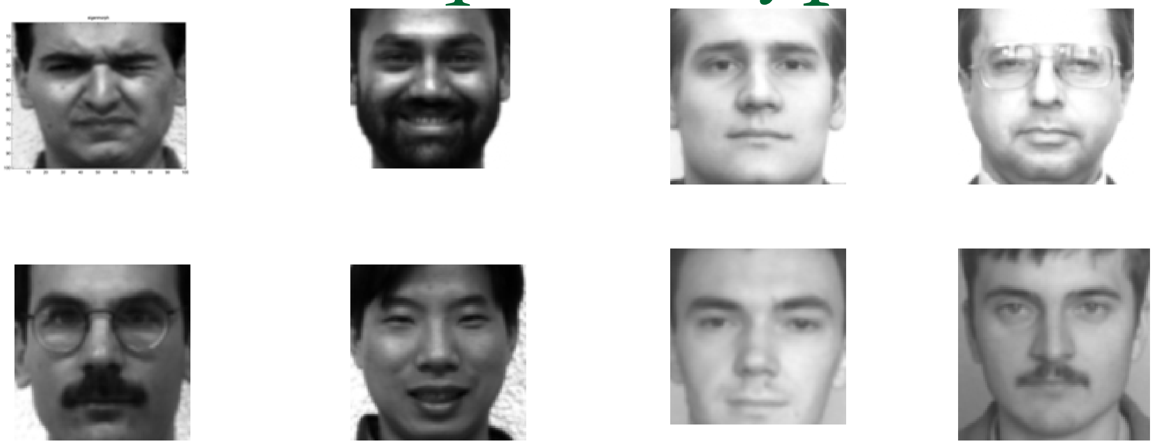
- DCT of small segments
  - 8x8
  - Each image becomes a matrix of DCT vectors
- DCT of the image
- Haar transform (checkerboard)
- ***Or data-driven representations..***

Returning to Eigen Computation



- A collection of faces
  - All normalized to 100x100 pixels
- What is common among all of them?
  - Do we have a common descriptor?

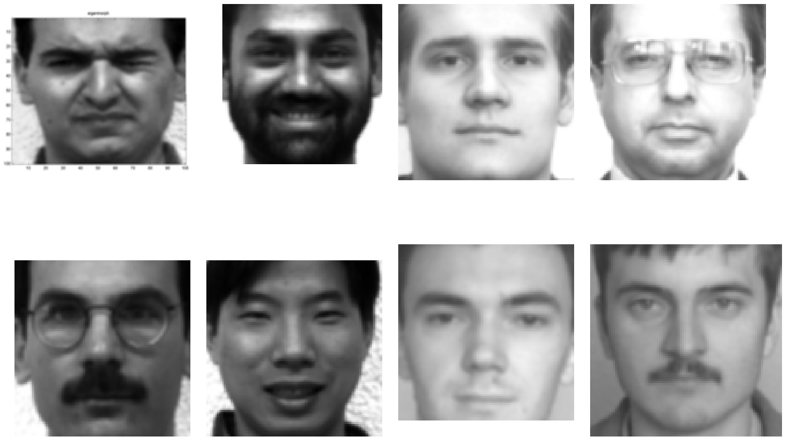
A least squares typical face

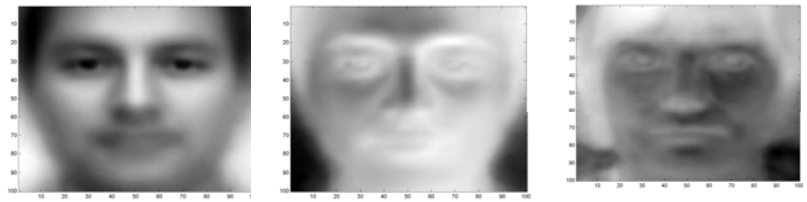


The typical face

- Can we do better than a blank screen to find the most common portion of faces?
  - The first checkerboard; the zeroth frequency component..
- Assumption: There is a "typical" face that captures most of what is common to all faces
  - Every face can be represented by a scaled version of a typical face
    - What is this face?
- Approximate **every** face f as $f = w_f V$
- Estimate V to minimize the squared error
  - How?
  - What is V?

A collection of least squares typical faces

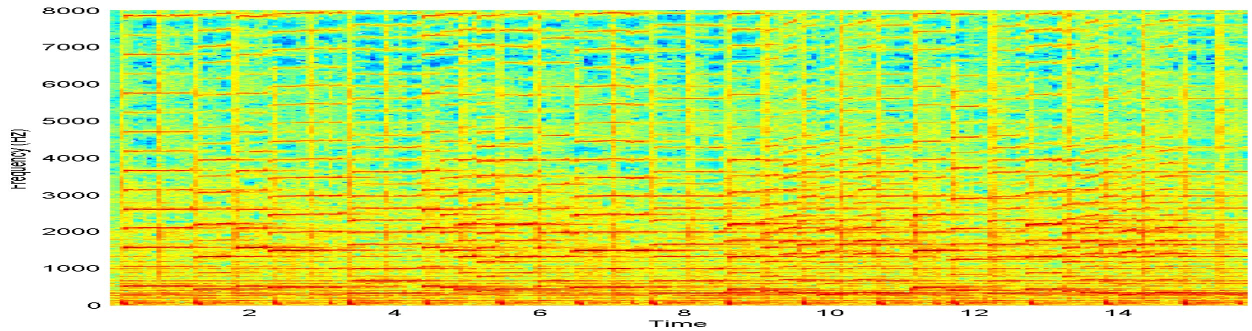




- Assumption: There are a set of K "typical" faces that captures most of all faces
- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + w_{f,3} V_3 + .. + w_{f,k} V_k$
  - V_2 is used to "correct" errors resulting from using only V_1
    - So the total energy in $w_{f,2}$ ($\Sigma\, w_{f,2}^2$) must be lesser than the total energy in $w_{f,1}$ ($\Sigma\, w_{f,1}^2$)
  - V_3 corrects errors remaining after correction with V_2
    - The total energy in $w_{f,3}$ must be lesser than that even in $w_{f,2}$
  - And so on..
  - $V = [V_1\ V_2\ V_3]$
- Estimate V to minimize the squared error
  - How?
  - What is V?

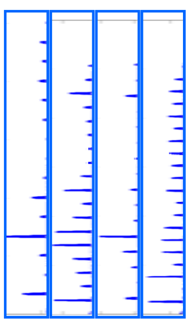
A recollection

M =



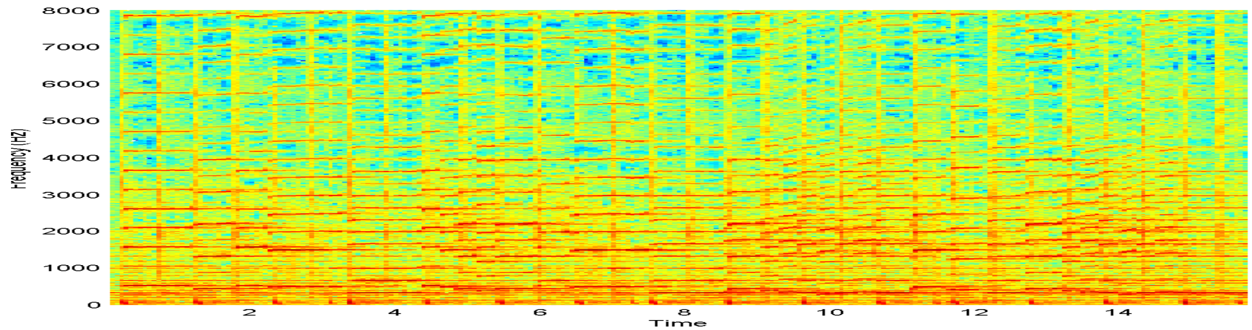
V=PINV(W)*M

W =

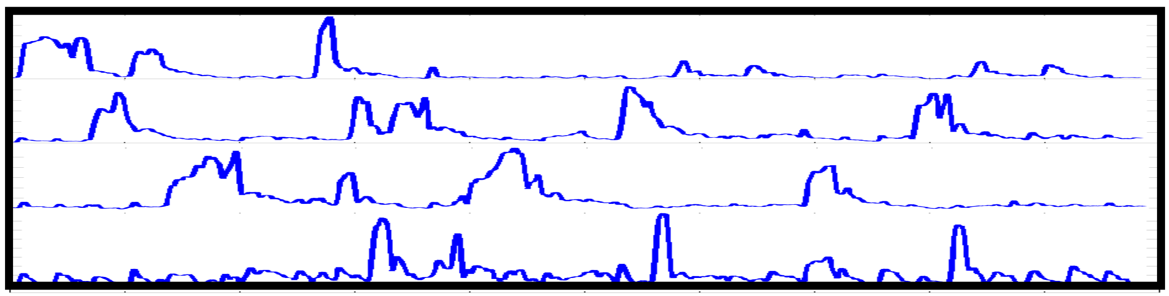


How about the other way?

M =



V =



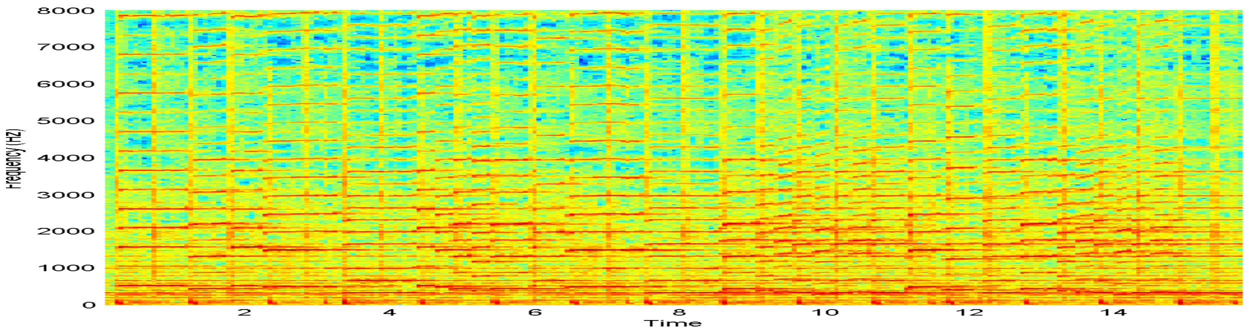
W = ?

U = ?

- W = M * Pinv(V)

How about the other way?

M =



V = ?

W = ?

U = ?

- W V \approx = M

Eigen Faces!

M = Data Matrix

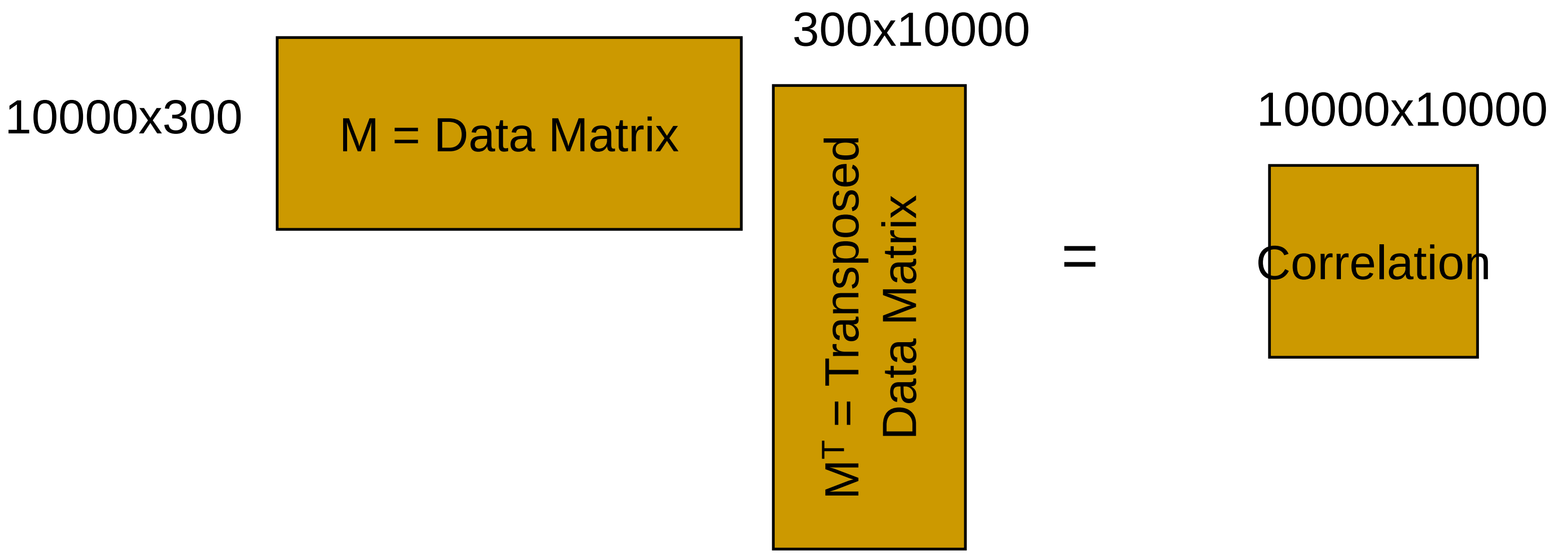
V

W

U = Approximation

- Here W, V and U are ALL unknown and must be determined
  - Such that the squared error between U and M is minimum
- Eigen analysis allows you to find W and V such that U = WV has the least squared error with respect to the original data M
- If the original data are a collection of faces, the columns of W represent the space of *eigen faces.*

Eigen faces



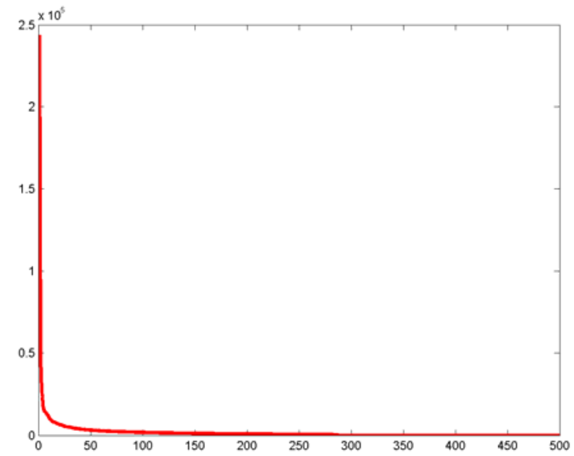
- Lay all faces side by side in vector form to form a matrix
  - In my example: 300 faces. So the matrix is 10000 x 300
- Multiply the matrix by its transpose
  - The correlation matrix is 10000x10000

Eigen faces

[U,S] = eig(correlation)

$$S = \begin{bmatrix} \lambda_1 & . & 0 & . & 0 \\ 0 & \lambda_2 & 0 & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & . & 0 & . & \lambda_{10000} \end{bmatrix}$$

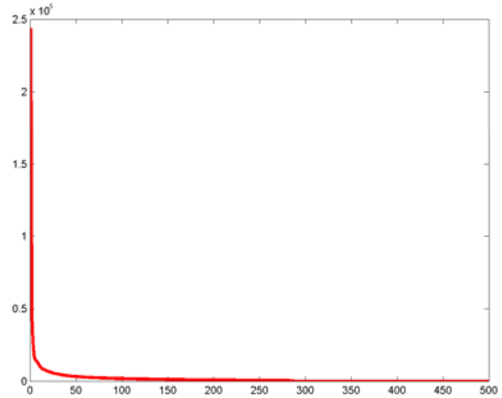
$$U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \bullet\bullet\bullet \end{bmatrix}$$



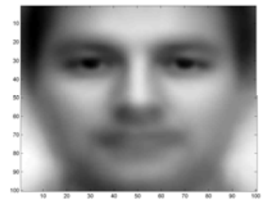
- Compute the eigen vectors
  - Only 300 of the 10000 eigen values are non-zero
    - Why?
- Retain eigen vectors with high eigen values (>0)
  - Could use a higher threshold

Eigen Faces

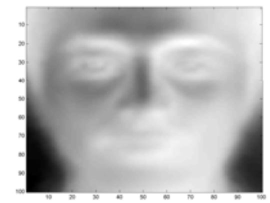
$$U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \bullet\bullet\bullet \end{bmatrix}$$



eigenface1



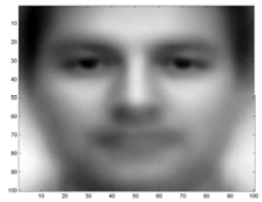
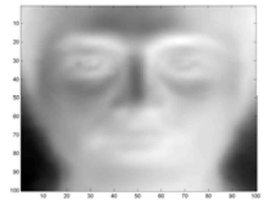
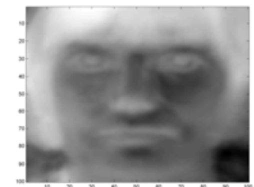
eigenface2

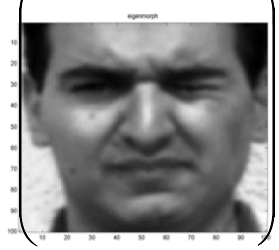


eigenface3

- The eigen vector with the highest eigen value is the first typical face
- The vector with the second highest eigen value is the second typical face.
- Etc.

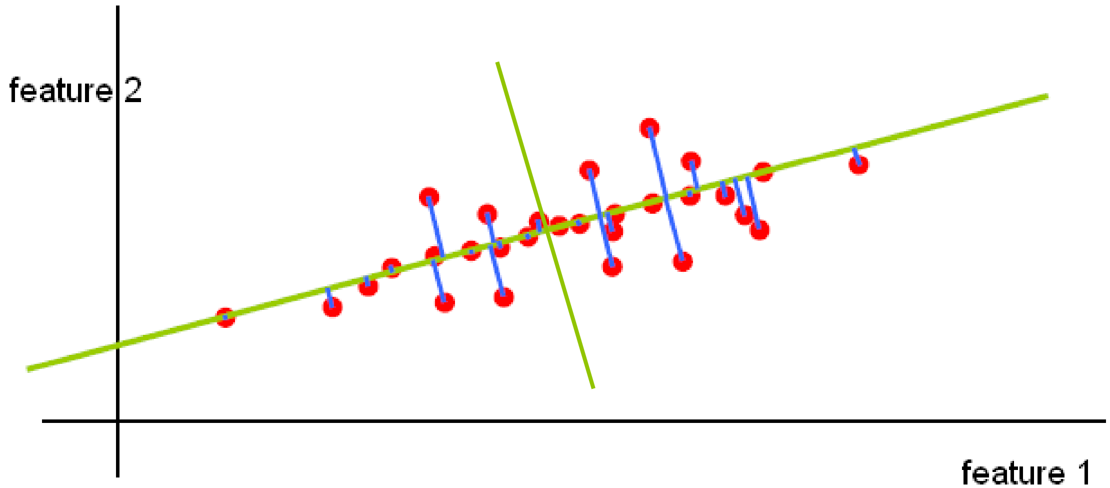
Representing a face

$=$ w_1  $+$ w_2  $+$ w_3  $\bullet\bullet\bullet$

Representation $\left(\right.$$\left.\right)$ $= [w_1\ w_2\ w_3\ \ldots.]^T$

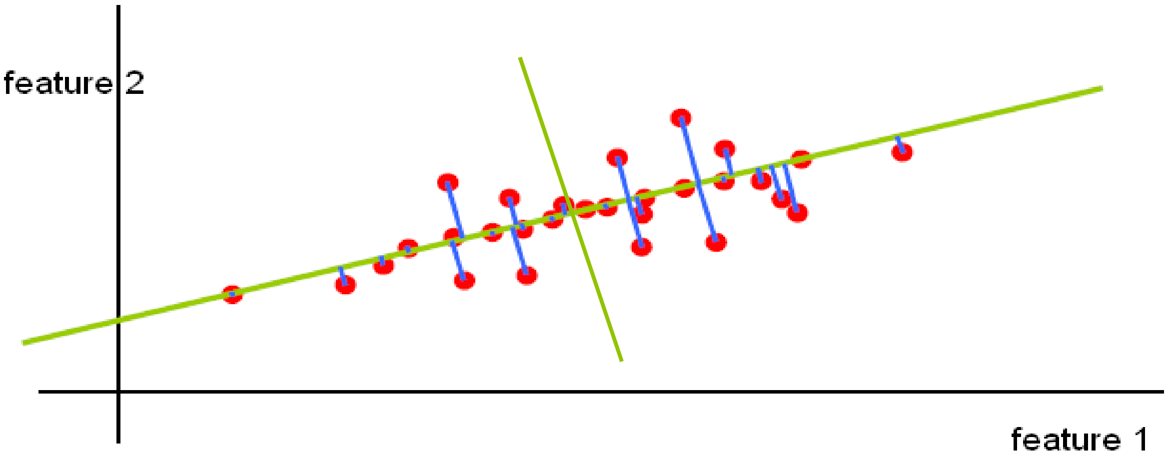
- The weights with which the eigen faces must be combined to compose the face are used to represent the face!

Principal Component Analysis



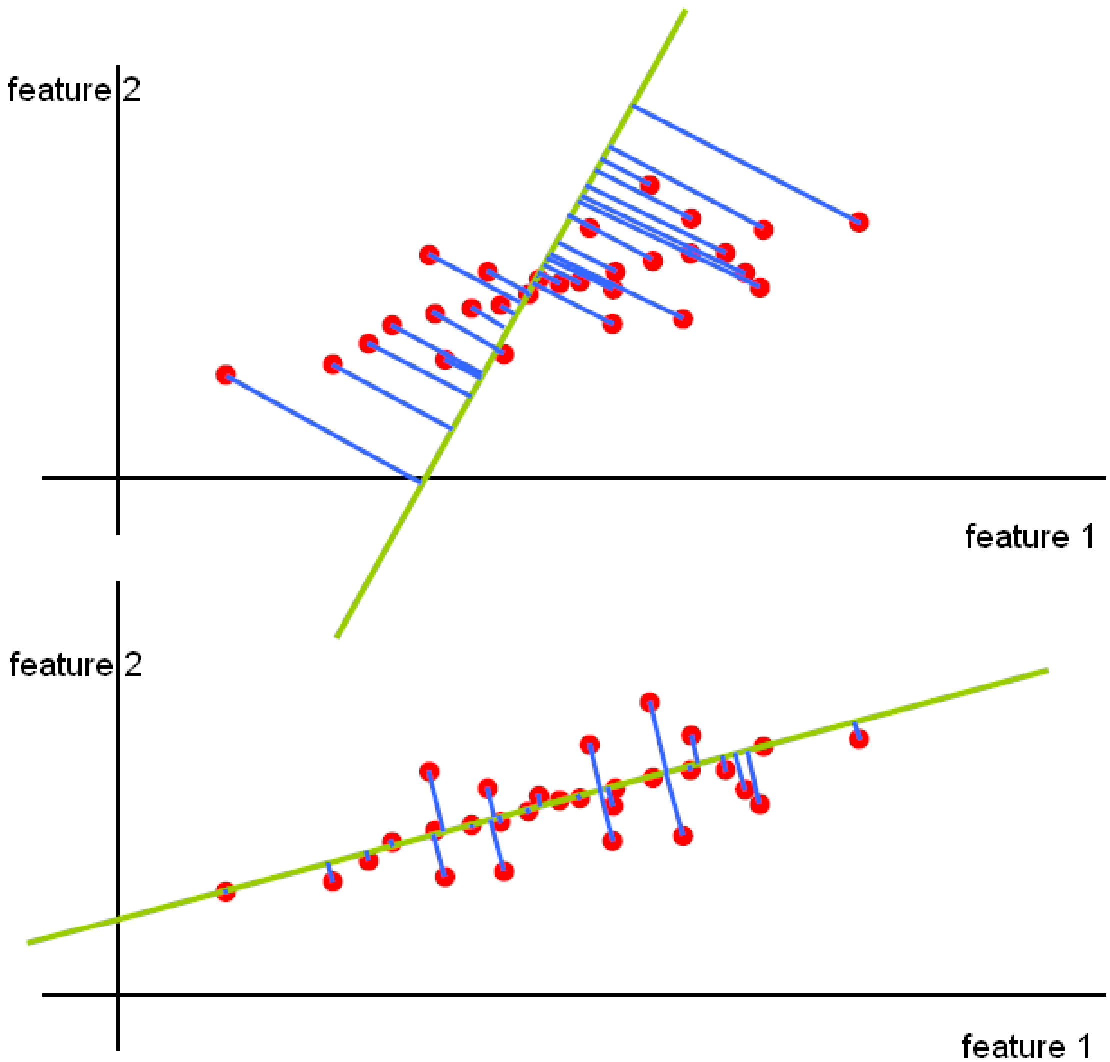
- Eigen analysis: Computing the “Principal” directions of a data
  - What do they mean
  - Why do we care

Principal Components == Eigen Vectors



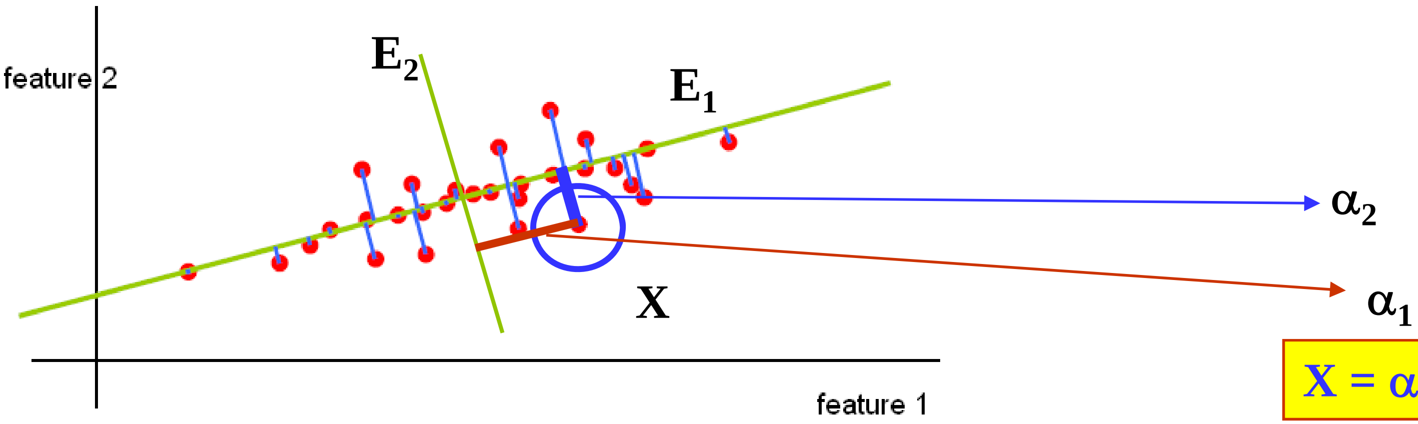
- Principal Component Analysis is the same as Eigen analysis
- The "Principal Components" are the Eigen Vectors

Principal Component Analysis



Which line through the mean leads to the smallest reconstruction error (sum of squared lengths of the blue lines) ?

Principal Components



$\mathbf{X = \alpha_1 E_1 + \alpha_2 E_2}$

- The first principal component is the *first Eigen* ("typical") vector
  - $X = \alpha_1(X)E_1$
  - The first Eigen face
  - For non-zero-mean data sets, the average of the data
- The second principal component is the second "typical" (or correction) vector
  - $X = \alpha_1(X)E_1 + \alpha_2(X)E_2$

SVD instead of Eigen

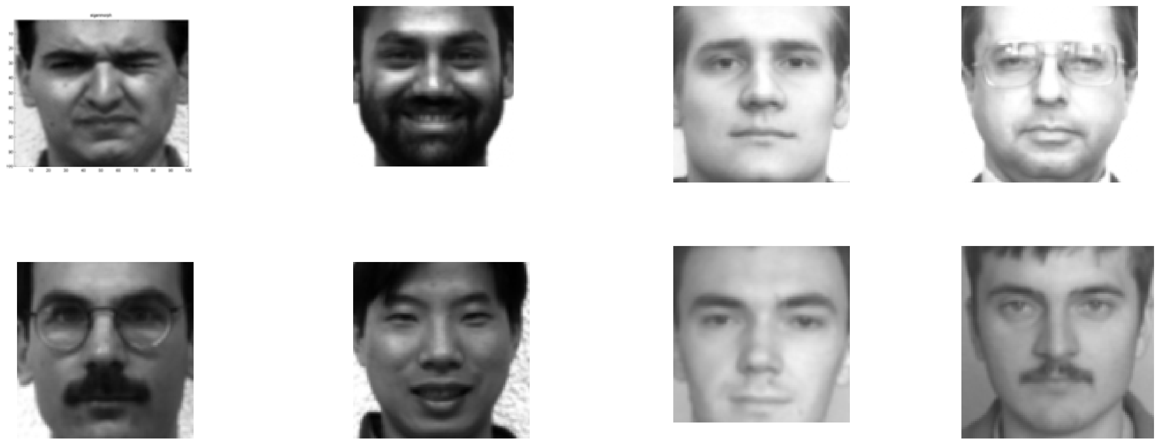
10000x300
M = Data Matrix = U=10000x300 S=300x300 V=300x300

$$U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \cdots \end{bmatrix}$$

- Do we need to compute a 10000 x 10000 correlation matrix and then perform Eigen analysis?
  - Will take a very long time on your laptop
- SVD
  - Only need to perform "Thin" SVD. Very fast
    - U = 10000 x 300
      - The columns of U are the eigen faces!
      - The Us corresponding to the "zero" eigen values are not computed
    - S = 300 x 300
    - V = 300 x 300

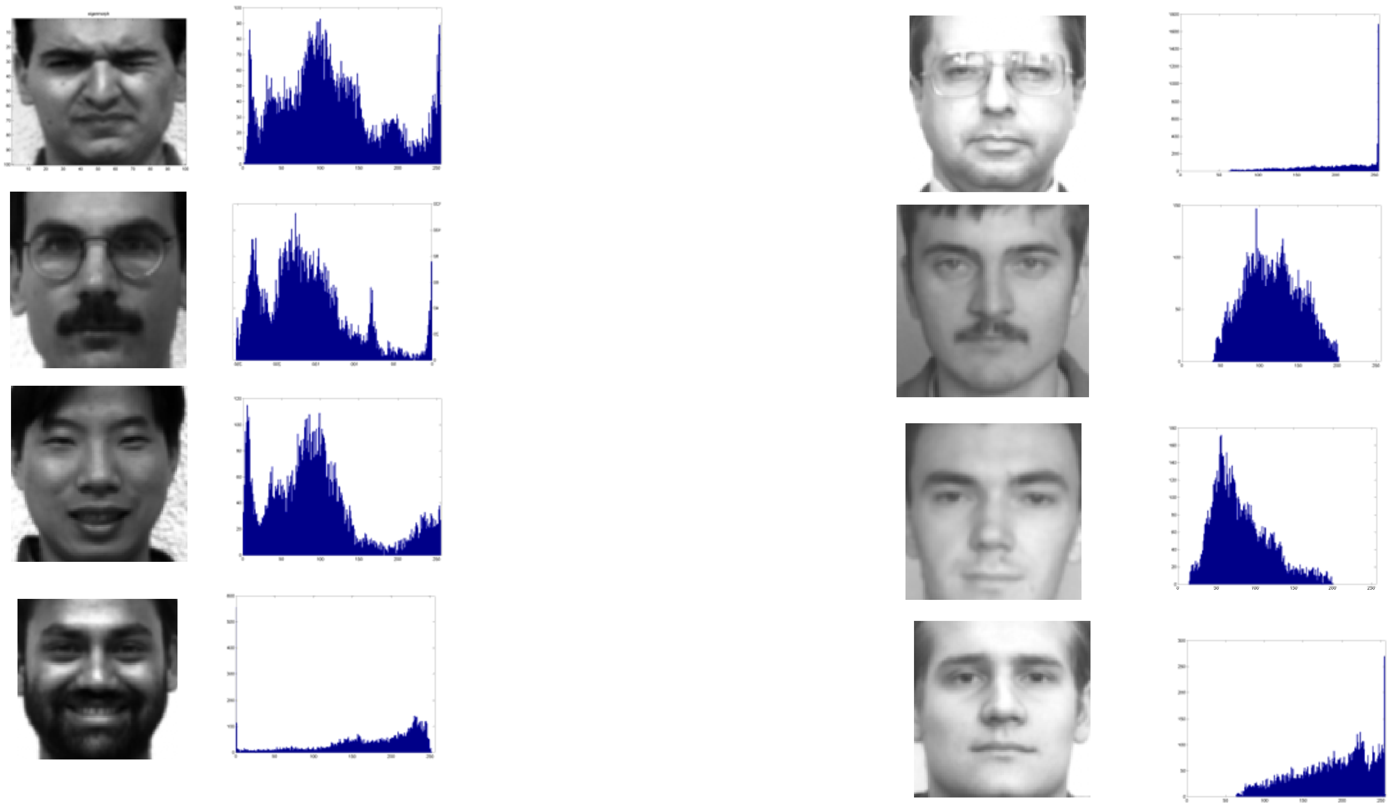
NORMALIZING OUT VARIATIONS

Images: Accounting for variations



- What are the obvious differences in the above images
- How can we capture these differences
  - Hint – image histograms..

Images -- Variations



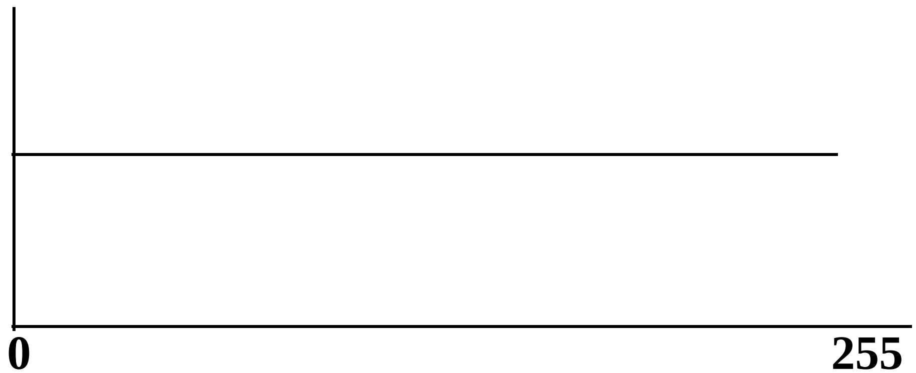
- Pixel histograms: what are the differences

Normalizing Image Characteristics

- Normalize the pictures
  - Eliminate lighting/contrast variations
  - All pictures must have “similar” lighting
    - How?
- Lighting and contrast are represented in the image histograms:

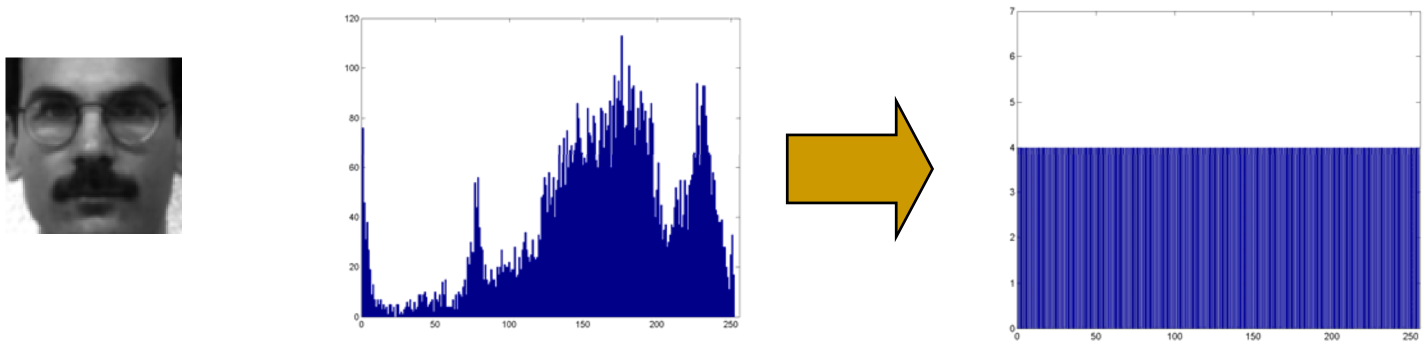
Histogram Equalization

- Normalize histograms of images
  - Maximize the contrast
    - Contrast is defined as the “flatness” of the histogram
    - For maximal contrast, every greyscale must happen as frequently as every other greyscale



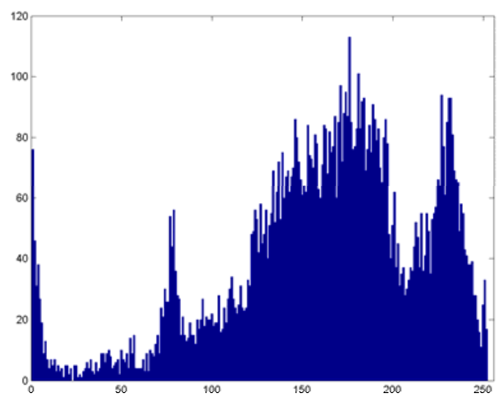
- Maximizing the contrast: Flattening the histogram
  - Doing it for every image ensures that every image has the same constrast
    - I.e. exactly the same histogram of pixel values
      - Which should be flat

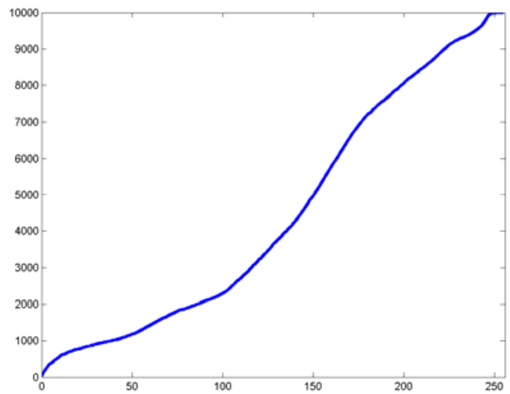
Histogram Equalization



- Modify pixel values such that histogram becomes "flat".
- For each pixel
  - New pixel value = f(old pixel value)
  - What is f()?
- Easy way to compute this function: map cumulative counts

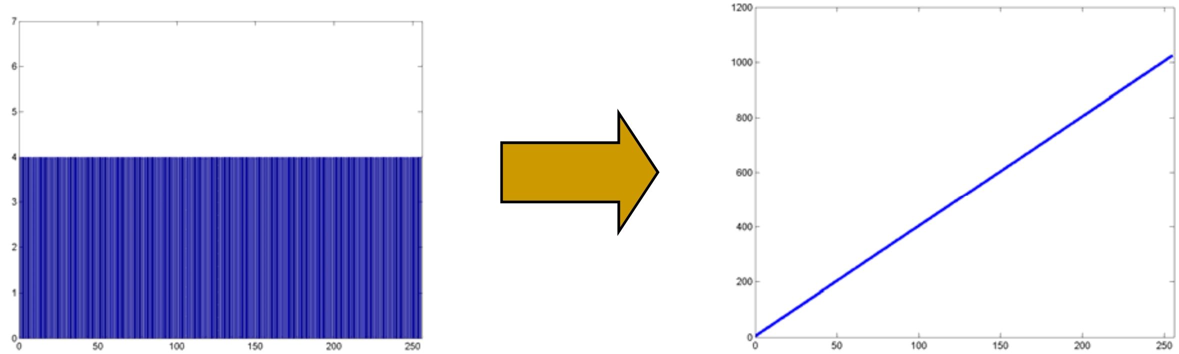
Cumulative Count Function



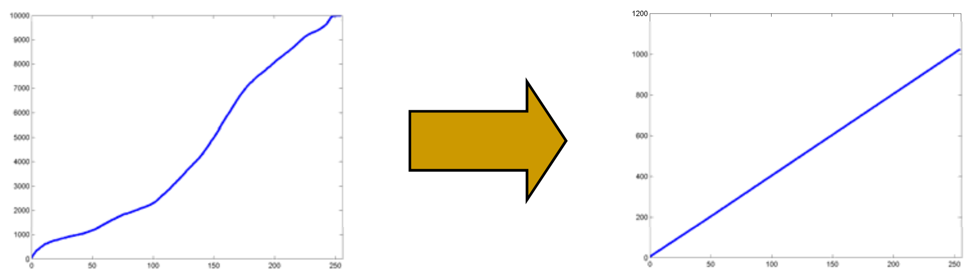


- The *histogram (count)* of a pixel value X is the number of pixels in the image that have value X
  - E.g. in the above image, the count of pixel value 180 is about 110
- The *cumulative count* at pixel value X is the total number of pixels that have values in the range 0 <= x <= X
  - CCF(X) = H(1) + H(2) + .. H(X)

Cumulative Count Function

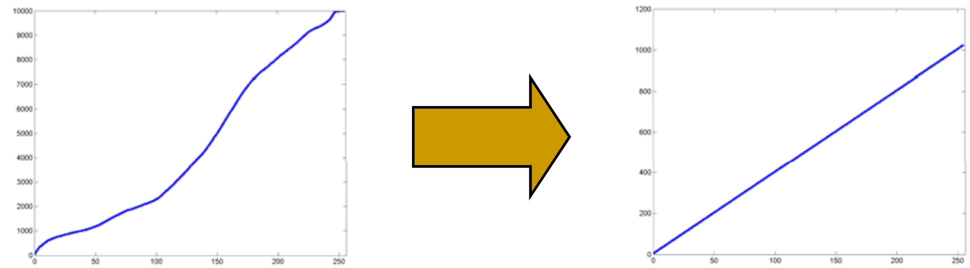


- The cumulative count function of a uniform histogram is a line



- We must modify the pixel values of the image so that its cumulative count is a line

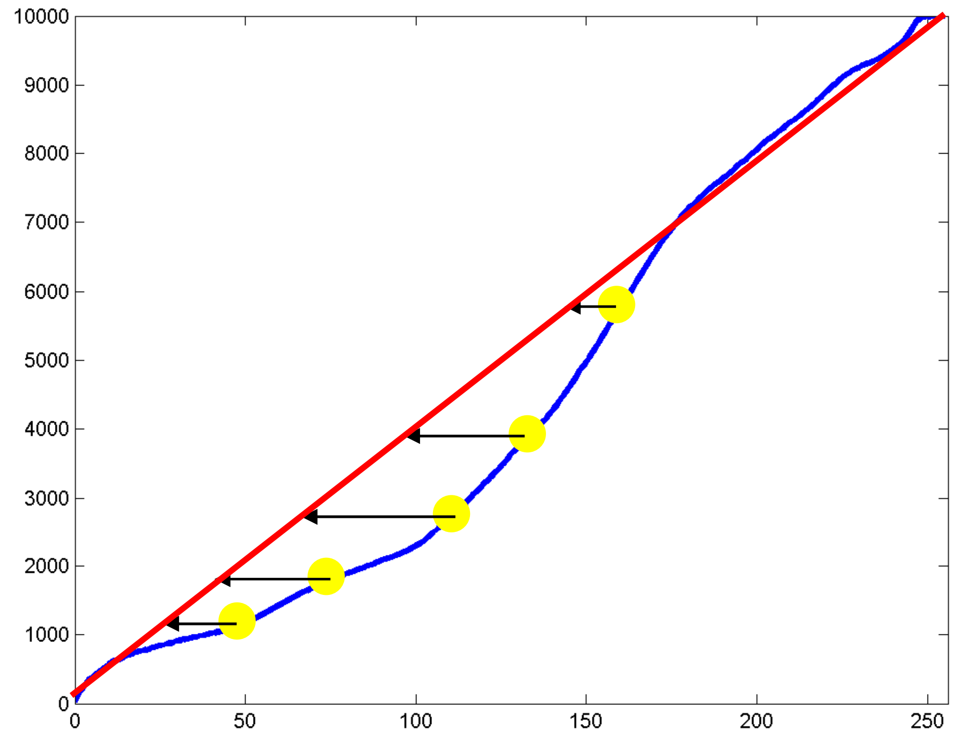
Mapping CCFs



Move x axis levels around until the plot to the left looks like the plot to the right

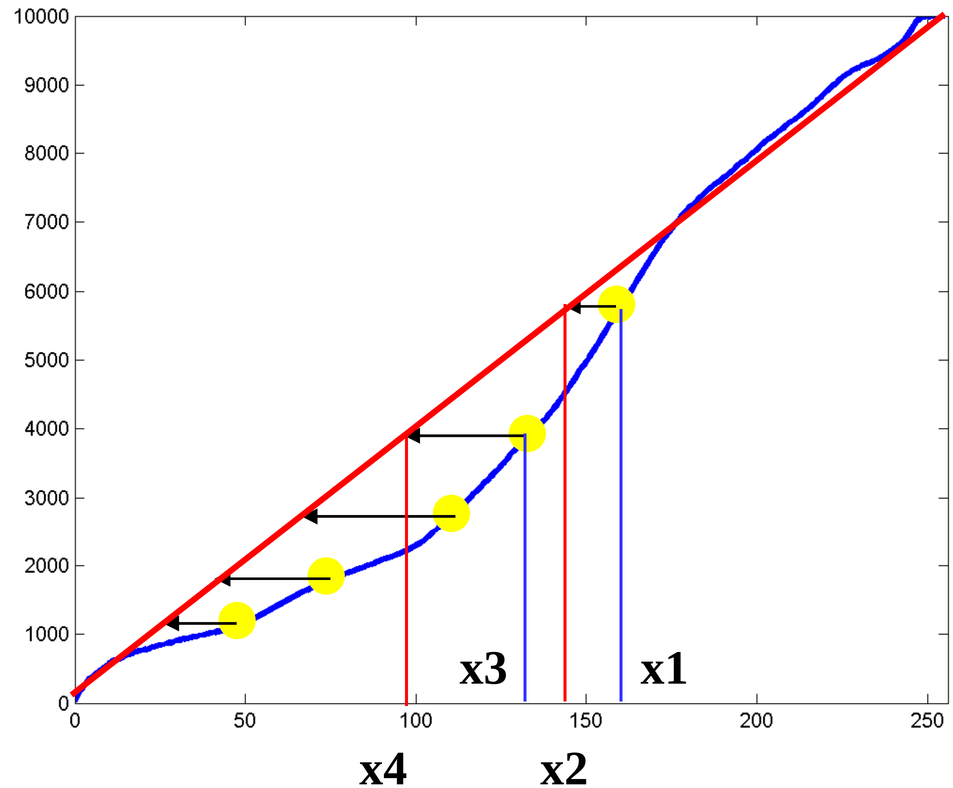
- CCF(f(x)) -> a*f(x)   [of a*(f(x)+1) if pixels can take value 0]
  - x = pixel value
  - f() is the function that converts the old pixel value to a new (normalized) pixel value
  - a = (total no. of pixels in image) / (total no. of pixel levels)
    - The no. of pixel levels is 256 in our examples
    - Total no. of pixels is 10000 in a 100x100 image

Mapping CCFs



- For each pixel value x:
  - Find the location on the red line that has the closet Y value to the observed CCF at x

Mapping CCFs



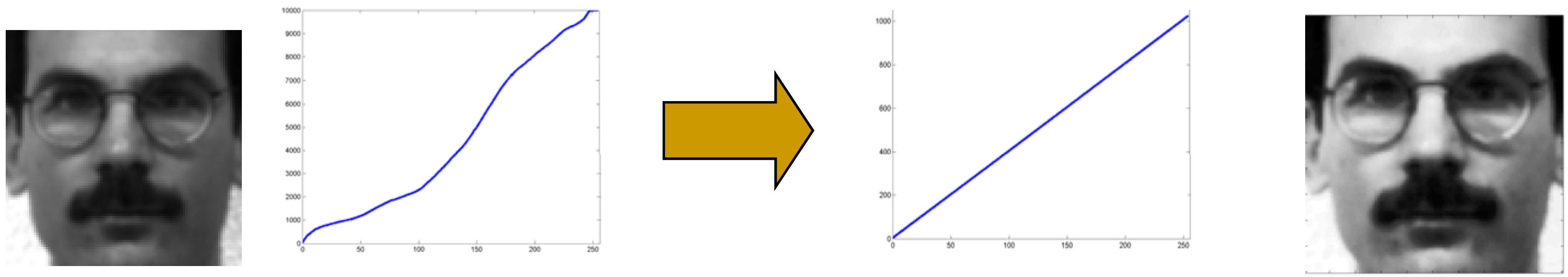
f(x1) = x2

f(x3) = x4

Etc.

- For each pixel value x:
  - Find the location on the red line that has the closet Y value to the observed CCF at x

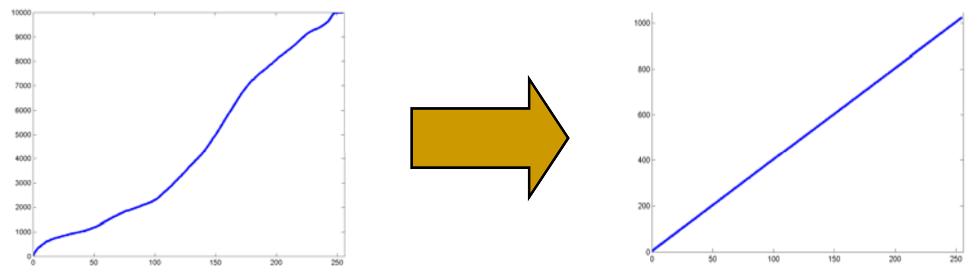
Mapping CCFs



Move x axis levels around until the plot to the left looks like the plot to the right

- For each pixel in the image to the left
  - The pixel has a value x
  - Find the CCF at that pixel value CCF(x)
  - Find x’ such that CCF(x’) in the function to the right equals CCF(x)
    - x’ such that CCF_flat(x’) = CCF(x)
  - Modify the pixel value to x’

Doing it Formulaically



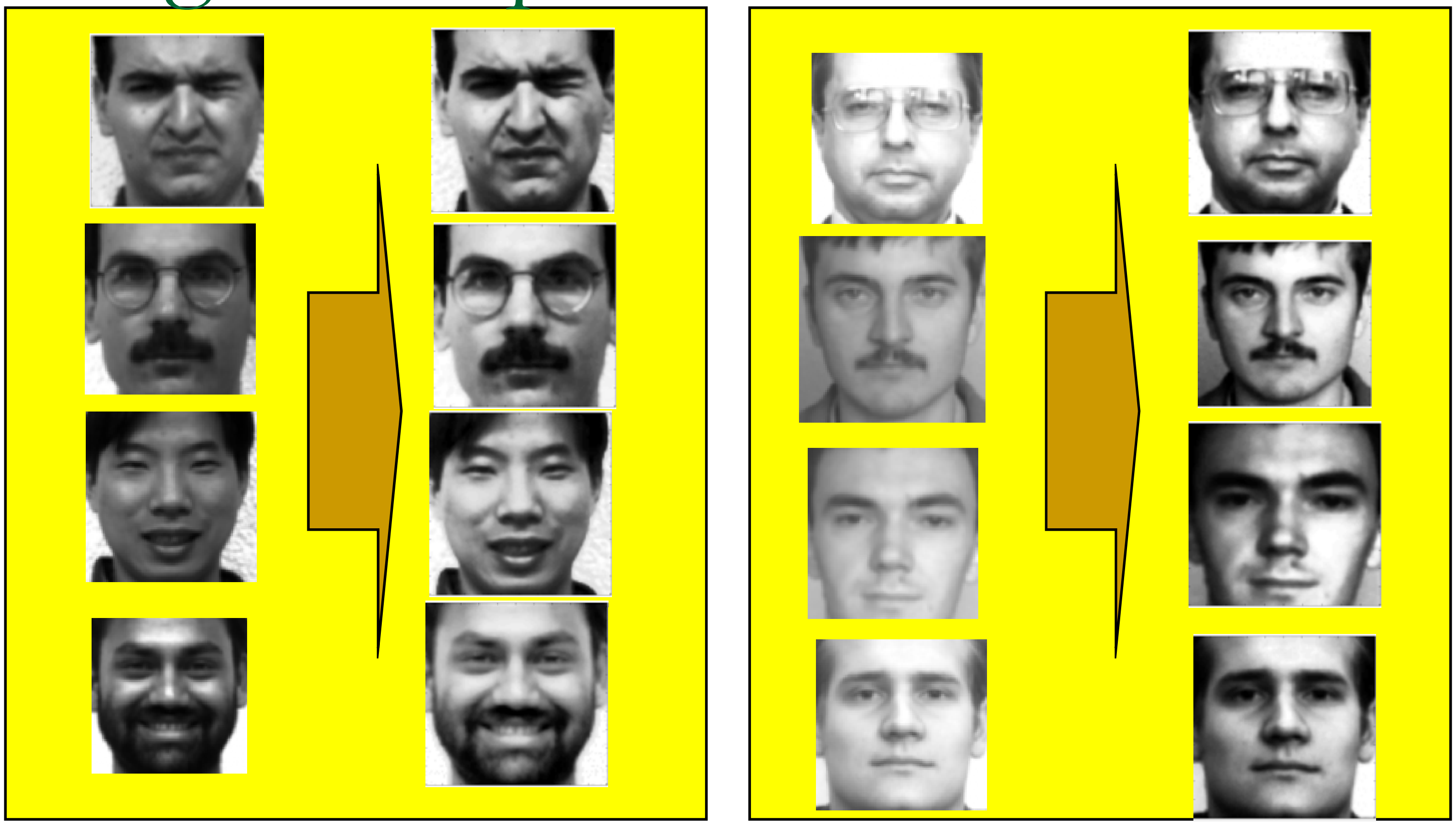
$$f(x) = round\left(\frac{CCF(x) - CCF_{\min}}{Npixels - CCF_{\min}} Max.pixel.value\right)$$

- CCF_{min} is the smallest non-zero value of CCF(x)
  - The value of the CCF at the smallest observed pixel value
- Npixels is the total no. of pixels in the image
  - 10000 for a 100x100 image
- Max.pixel.value is the highest pixel value
  - 255 for 8-bit pixel representations

Or even simpler

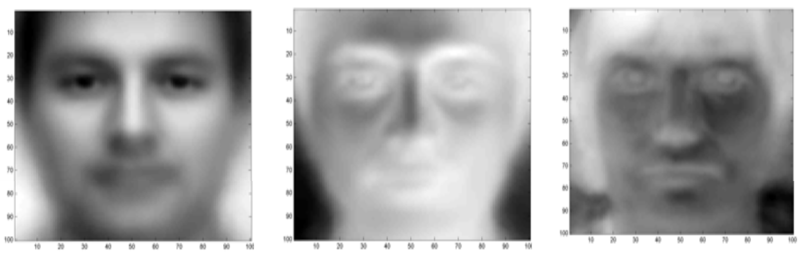
- Matlab:
  - `Newimage = histeq(oldimage)`

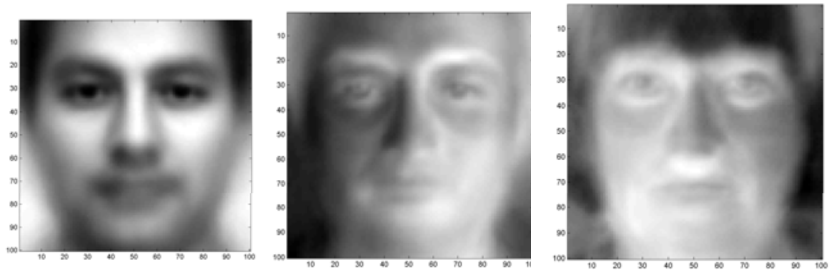
Histogram Equalization



- Left column: Original image
- Right column: Equalized image
- All images now have similar contrast levels

Eigenfaces after Equalization

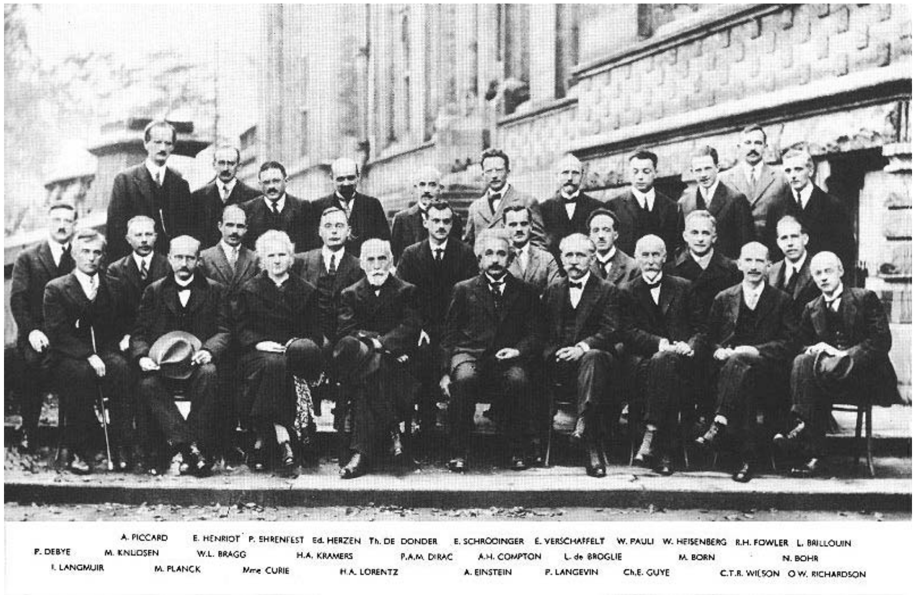




- Left panel : Without HEQ
- Right panel: With HEQ
  - Eigen faces are more face like..
    - Need not always be the case

Detecting Faces in Images

Detecting Faces in Images



- Finding face like patterns
  - How do we find if a picture has faces in it
  - Where are the faces?
- A simple solution:
  - Define a “typical face”
  - Find the “typical face” in the image

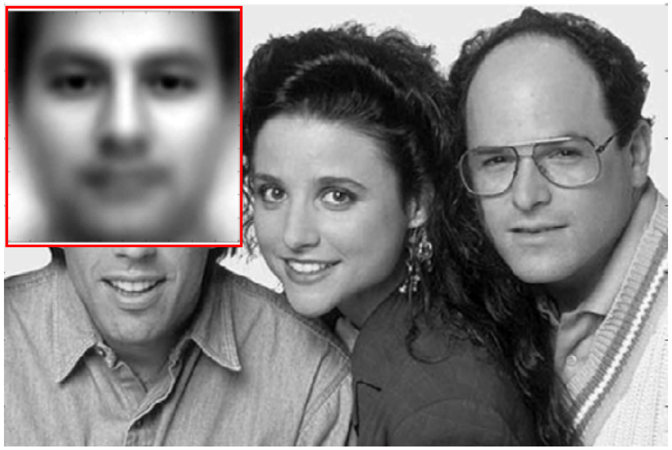
Finding faces in an image

- Picture is larger than the "typical face"
  - E.g. typical face is 100x100, picture is 600x800
- First convert to greyscale
  - R + G + B
  - Not very useful to work in color

Finding faces in an image

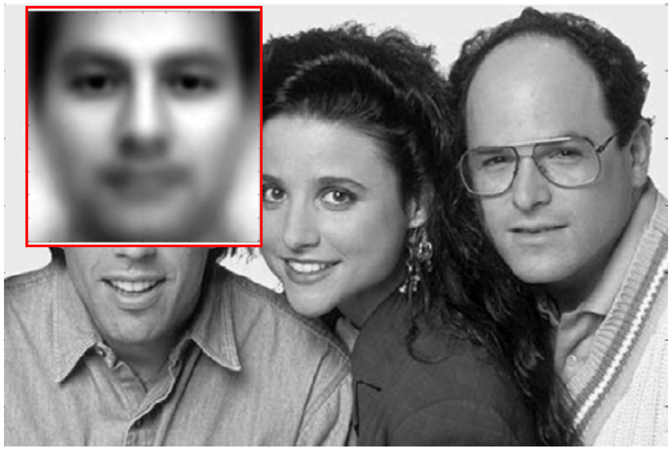
- Goal .. To find out if and where images that look like the “typical” face occur in the picture

Finding faces in an image



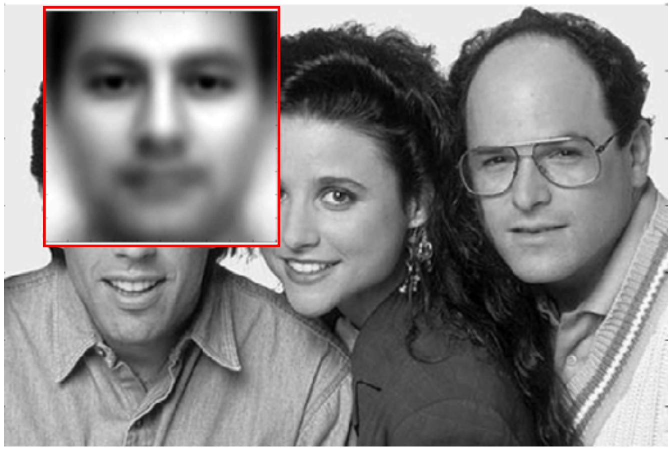
- Try to “match” the typical face to each location in the picture

Finding faces in an image



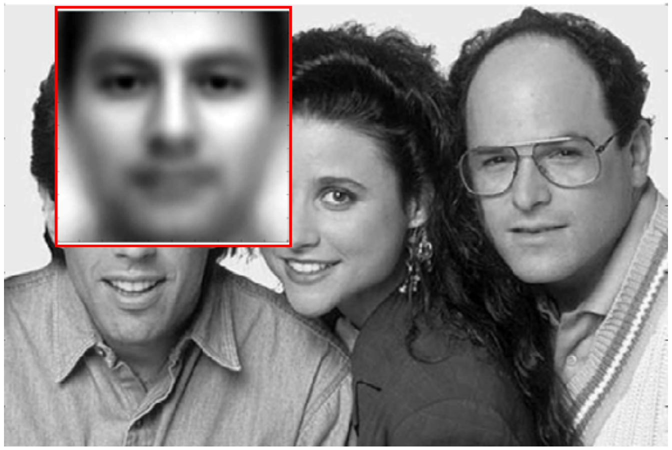
- Try to “match” the typical face to each location in the picture

Finding faces in an image



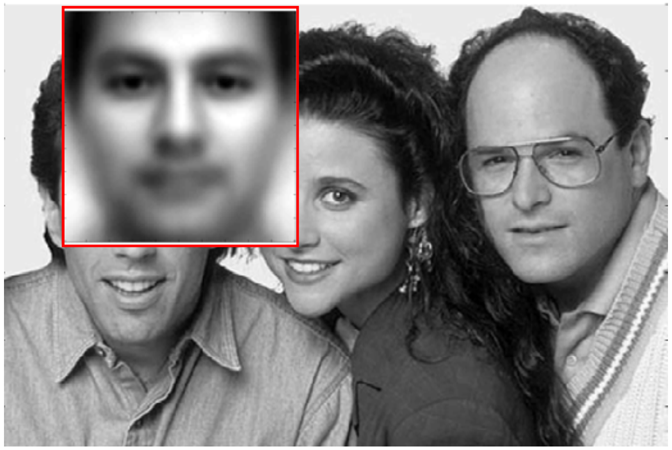
- Try to “match” the typical face to each location in the picture

Finding faces in an image



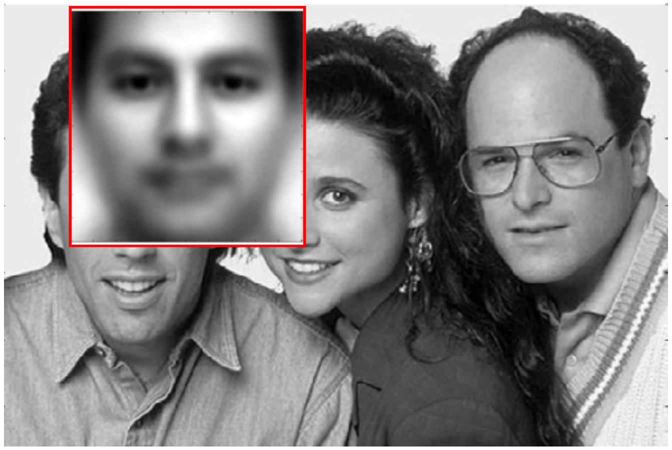
- Try to "match" the typical face to each location in the picture

Finding faces in an image



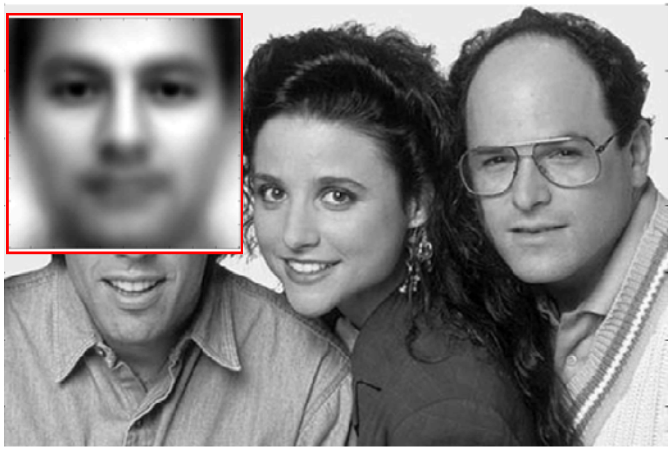
- Try to “match” the typical face to each location in the picture

Finding faces in an image



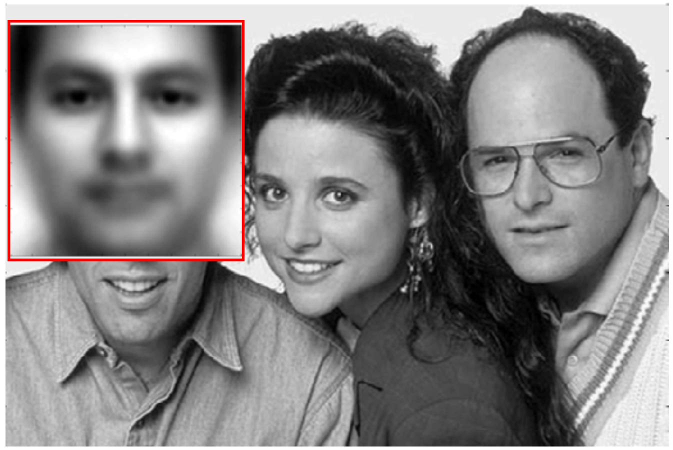
- Try to “match” the typical face to each location in the picture

Finding faces in an image



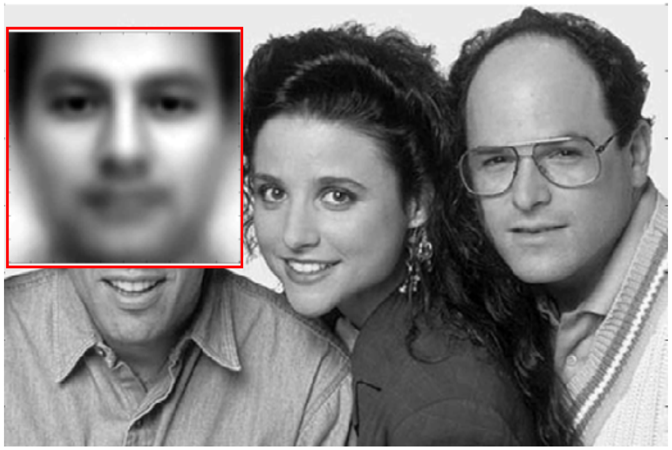
- Try to “match” the typical face to each location in the picture

Finding faces in an image



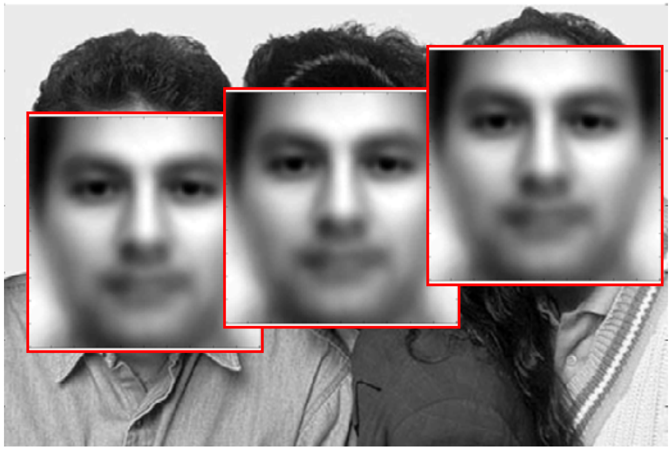
- Try to “match” the typical face to each location in the picture

Finding faces in an image



- Try to "match" the typical face to each location in the picture

Finding faces in an image

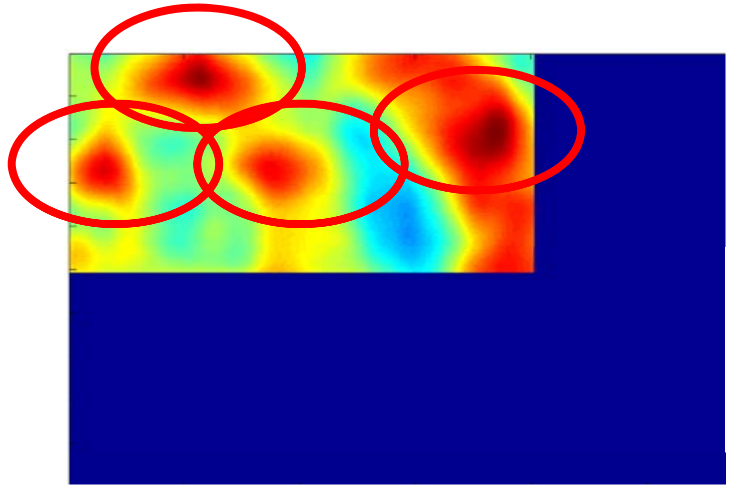


- Try to "match" the typical face to each location in the picture
- The "typical face" will explain some spots on the image much better than others
  - These are the spots at which we probably have a face!

How to "match"

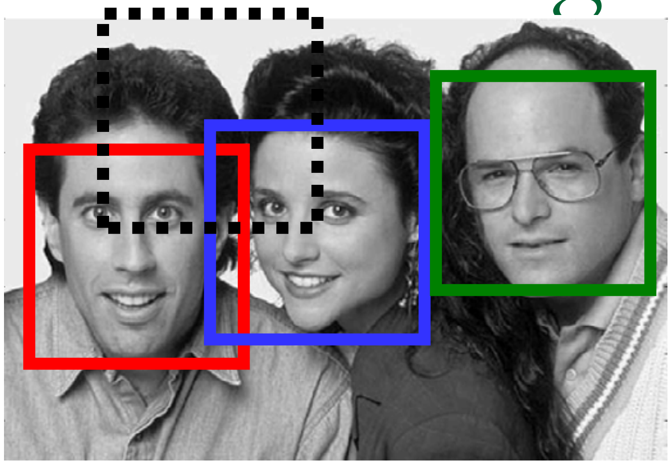
- What exactly is the "match"
  - What is the match "score"
- The DOT Product
  - Express the typical face as a vector
  - Express the region of the image being evaluated as a vector
    - But first histogram equalize the region
      - Just the section being evaluated, without considering the rest of the image
  - Compute the dot product of the typical face vector and the "region" vector

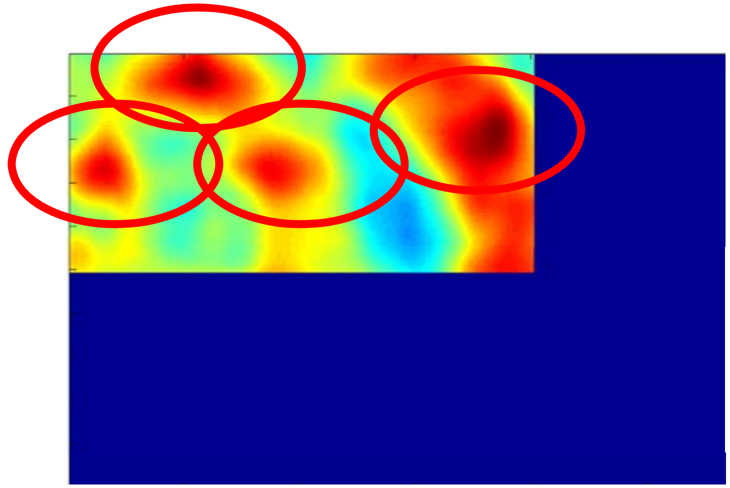
What do we get



- The right panel shows the dot product a various loctions
  - Redder is higher
    - The locations of peaks indicate locations of faces!

What do we get

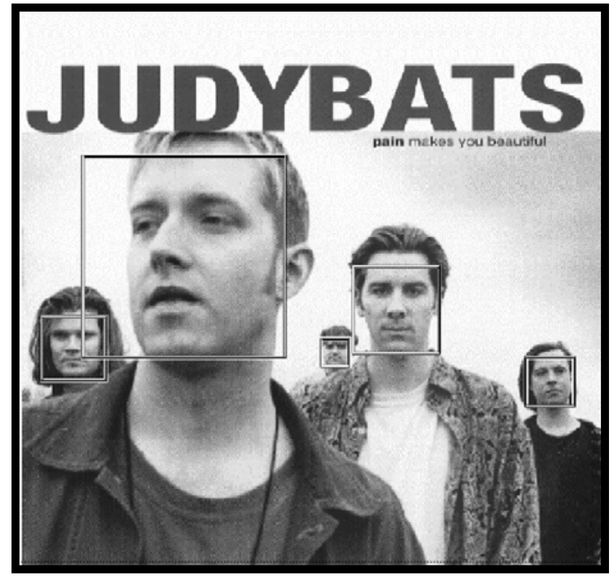




- The right panel shows the dot product a various loctions
  - Redder is higher
    - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
  - Likes George’s face most
    - He looks most like the typical face
- Also finds a face where there is none!
  - A false alarm

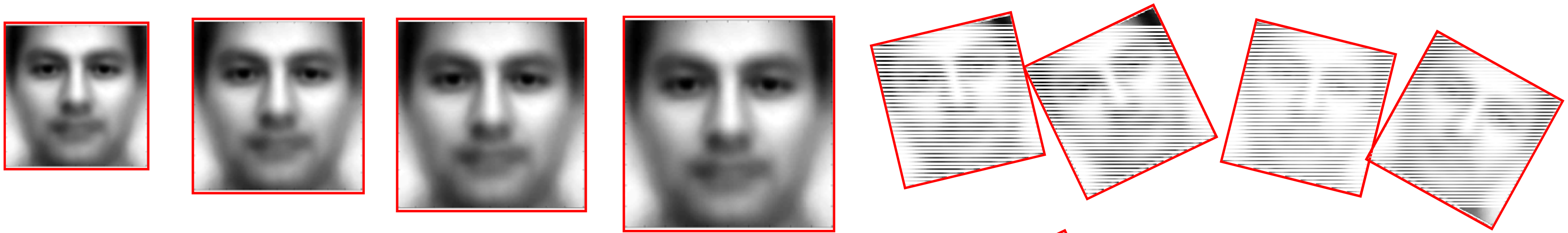
Scaling and Rotation Problems

- Scaling
  - Not all faces are the same size
  - Some people have bigger faces
  - The size of the face on the image changes with perspective
  - Our “typical face” only represents one of these sizes



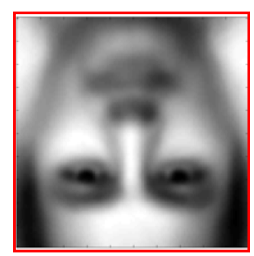
- Rotation
  - The head need not always be upright!
    - Our typical face image was upright

Solution



- Create many “typical faces”
  - One for each scaling factor
  - One for each rotation
    - How will we do this?
- Match them all

- Does this work
  - Kind of .. Not well enough at all
  - We need more sophisticated models



Face Detection: A Quick Historical Perspective

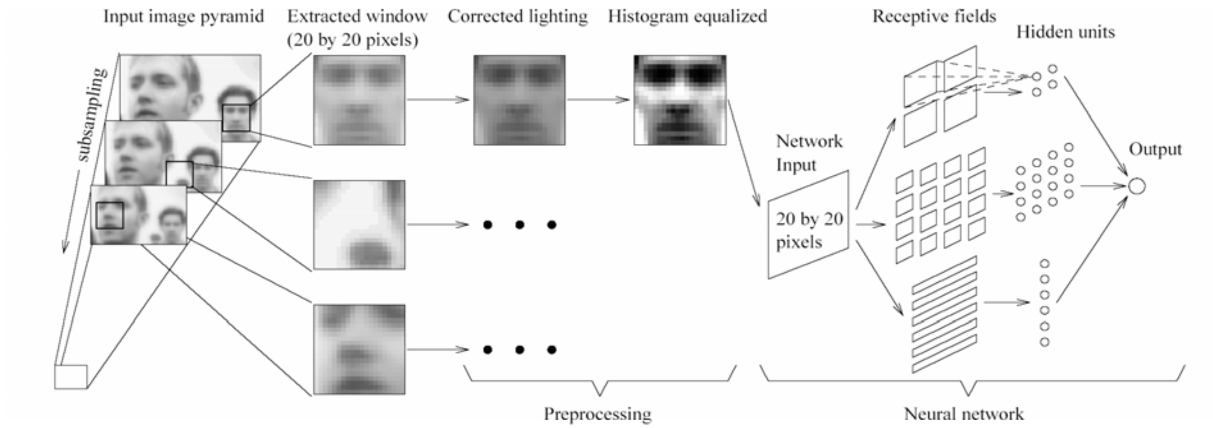


Figure 1: The basic algorithm used for face detection.

- Many more complex methods
  - Use edge detectors and search for face like patterns
  - Find "feature" detectors (noses, ears..) and employ them in complex neural networks..

- The Viola Jones method
  - Boosted cascaded classifiers

- But first, what is boosting

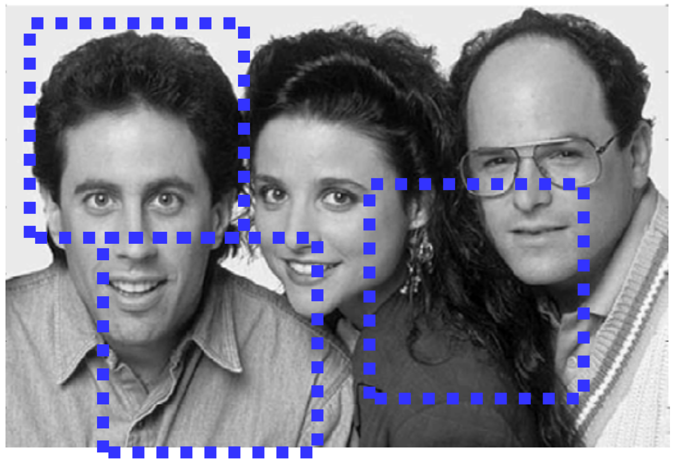
And even before that – what is classification?

- Given “features” describing an entity, determine the category it belongs to
  - Walks on two legs, has no hair. Is this
    - A Chimpanizee
    - A Human
  - Has long hair, is 5’4” tall, is this
    - A man
    - A woman
  - Matches “eye” pattern with score 0.5, “mouth pattern” with score 0.25, “nose” pattern with score 0.1. Are we looking at
    - A face
    - Not a face?

Classification

- Multi-class classification
  - Many possible categories
    - E.g. Sounds “AH, IY, UW, EY..”
    - E.g. Images “Tree, dog, house, person..”
- Binary classification
  - Only two categories
    - Man vs. Woman
    - Face vs. not a face..
- Face detection: Recast as binary face classification
  - For each little square of the image, determine if the square represents a face or not

Face Detection as Classification



For each square, run a classifier to find out if it is a face or not

- Faces can be many sizes
- They can happen anywhere in the image
- For each face size
  - For each location
    - Classify a rectangular region of the face size, at that location, as a face or not a face
- This is a series of ***binary*** classification problems