Machine Learning for Signal Processing

Eigenfaces and Eigenrepresentations

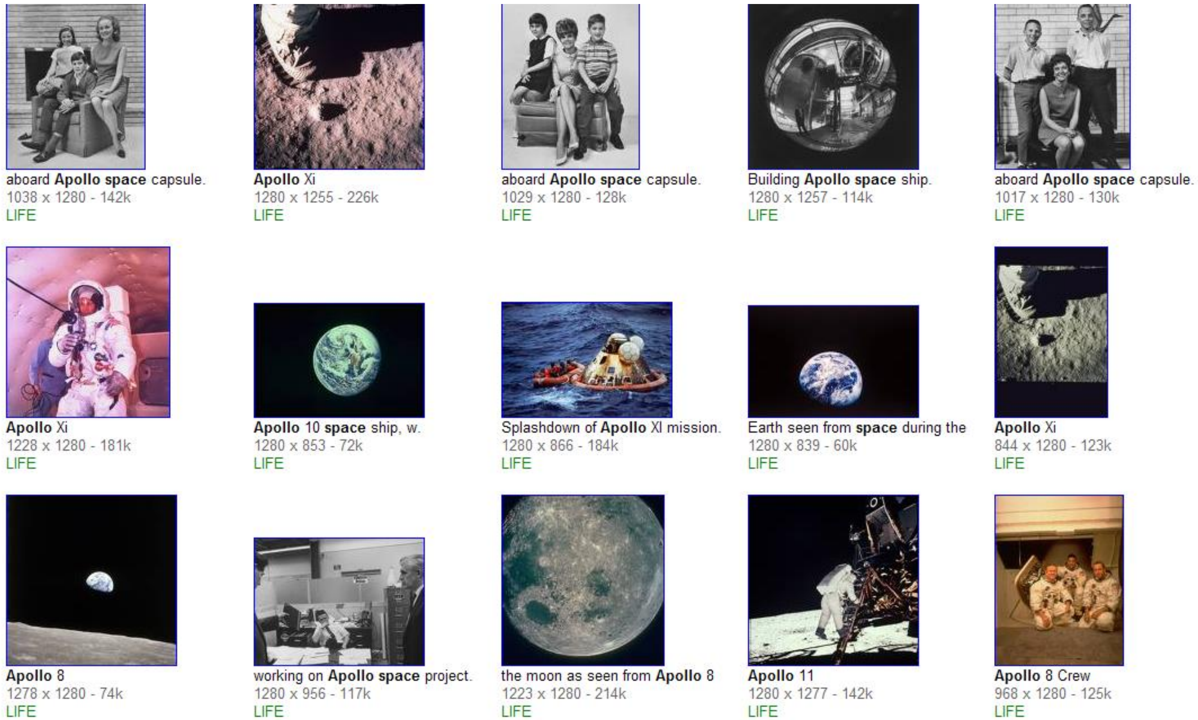
Class 6. 17 Sep 2013

Instructor: Bhiksha Raj

Administrivia

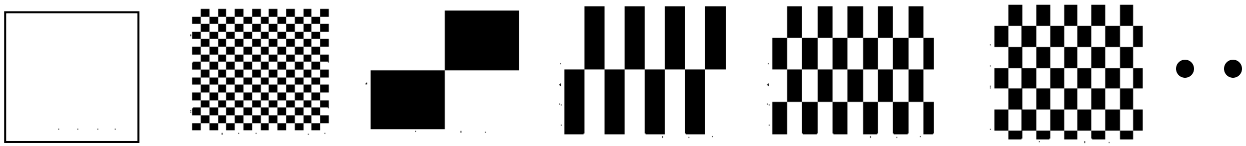
- Project teams?
  - By the end of the month..
- Project proposals?
  - Please send proposals to TAs, and cc me
- Reminder: Assignment 1 due in 9 days

Recall: Representing images



- The most common element in the image: background
  - Or rather large regions of relatively featureless shading
  - Uniform sequences of numbers

Adding more bases



B_1 B_2 B_3 B_4 B_5 B_6

- Checkerboards with different variations

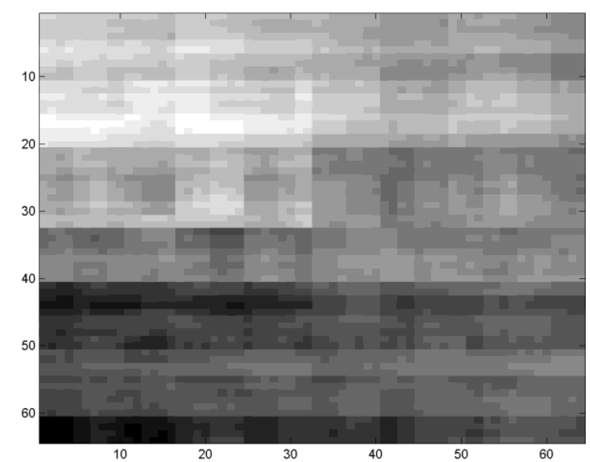
$$\mathrm{Image} \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$$

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \cdot \\ \cdot \end{bmatrix} \qquad B = [B_1 \;\; B_2 \;\; B_3]$$

$$BW \approx \mathrm{Image}$$

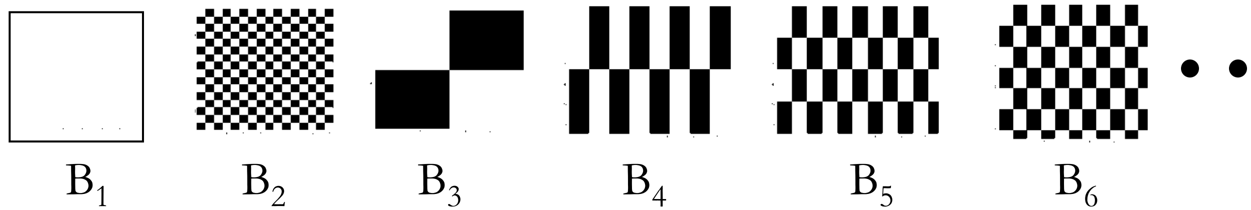
$$W = pinv(B)\,\mathrm{Image}$$

$$PROJECTION = BW$$



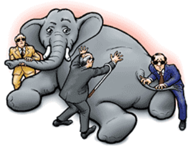
Getting closer at 625 bases!

"Bases"



$$image \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots$$

- "Bases" are the "standard" units such that all instances can be expressed a weighted combinations of these units
- Ideal requirements: Bases must be orthogonal
- Checkerboards are one choice of bases
  - Orthogonal
  - But not "smooth"
- Other choices of bases: Complex exponentials, Wavelets, etc..



Data specific bases?

- **Issue**: **All the bases we have considered so far are *data agnostic***
  - Checkerboards, Complex exponentials, Wavelets..
  - We use the same bases regardless of the data we analyze
    - Image of face vs. Image of a forest
    - Segment of speech vs. Seismic rumble
- How about data specific bases
  - Bases that consider the underlying data
    - E.g. is there something better than checkerboards to describe faces
    - Something better than complex exponentials to describe music?

The Energy Compaction Property

- **Define "better"?**
- The description

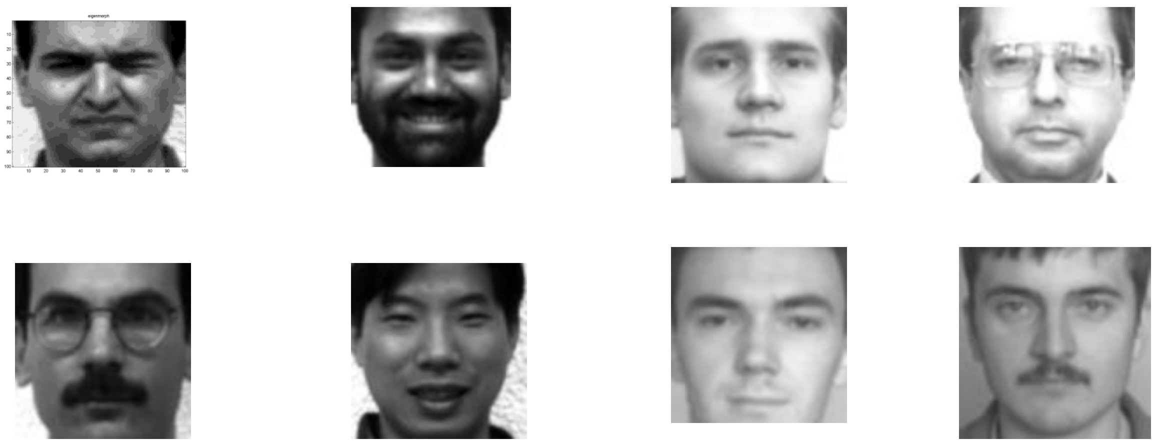
$$X = w_1 B_1 + w_2 B_2 + w_3 B_3 + \ldots + w_N B_N$$

- The ideal:

$$\hat{X} \approx w_1 B_1 + w_2 B_2 \qquad Error = \left\| X - \hat{X} \right\|^2$$

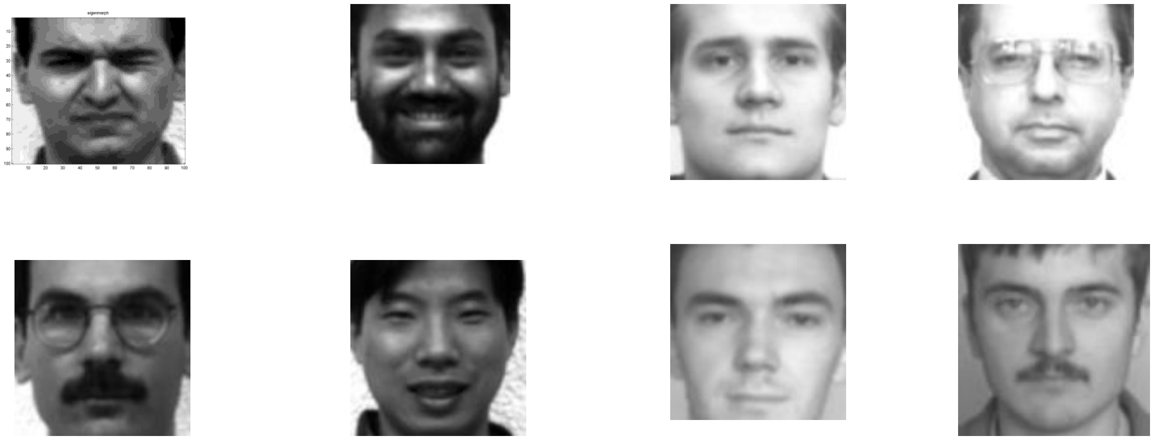
  – If the description is terminated at any point, we should still get most of the information about the data
    - Error should be small

Data-specific description of faces

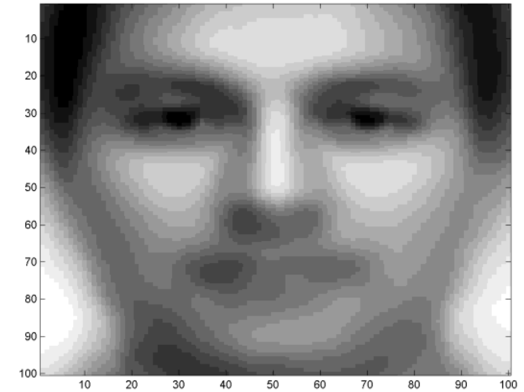


- A collection of images
  - All normalized to 100x100 pixels
- What is common among all of them?
  - Do we have a common descriptor?

A typical face

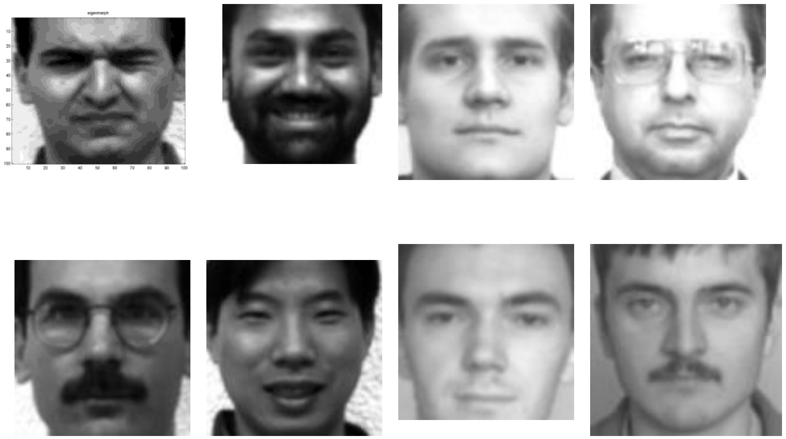


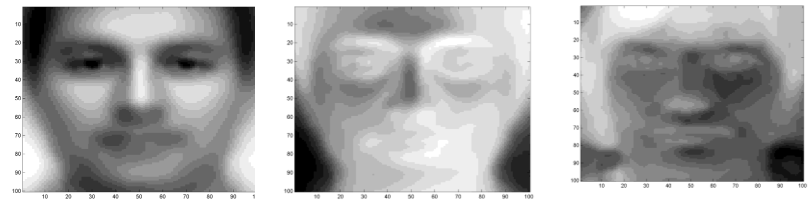
The typical face



- **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
  - We will denote this face as V
- Approximate **every** face f as $\mathrm{f} = \mathrm{w_f}\ \mathrm{V}$
- 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 on average

$$\left\| f - (w_{f,1}V_{f,1} + w_{f,2}V_{f,2}) \right\|^2 < \left\| f - w_{f,1}V_{f,1} \right\|^2$$

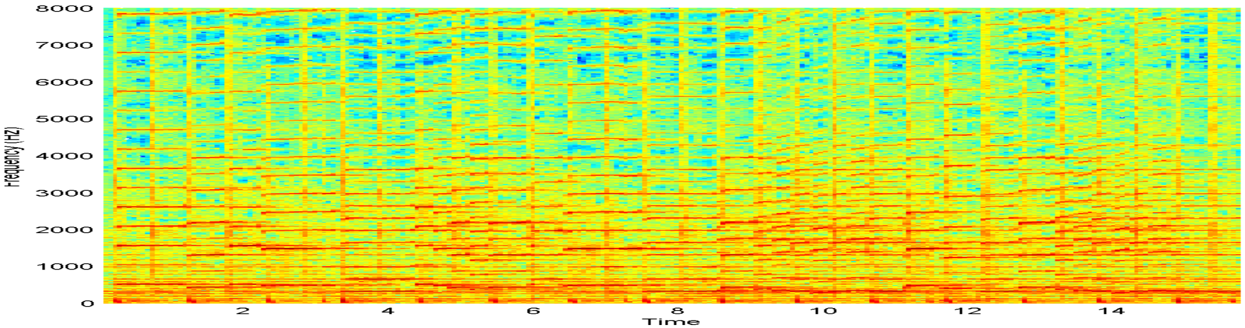
  - V_3 corrects errors remaining after correction with V_2

$$\left\| f - (w_{f,1}V_{f,1} + w_{f,2}V_{f,2} + w_{f,3}V_{f,3}) \right\|^2 < \left\| f - (w_{f,1}V_{f,1} + w_{f,2}V_{f,2}) \right\|^2$$

  - And so on..
  - **$V = [V_1\ V_2\ V_3]$**
- Estimate V to minimize the squared error
  - ***How? What is V?***

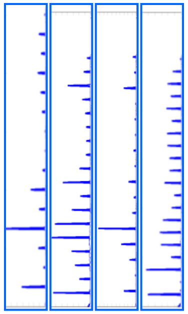
A recollection

M =



S=Pinv(N) M

N =



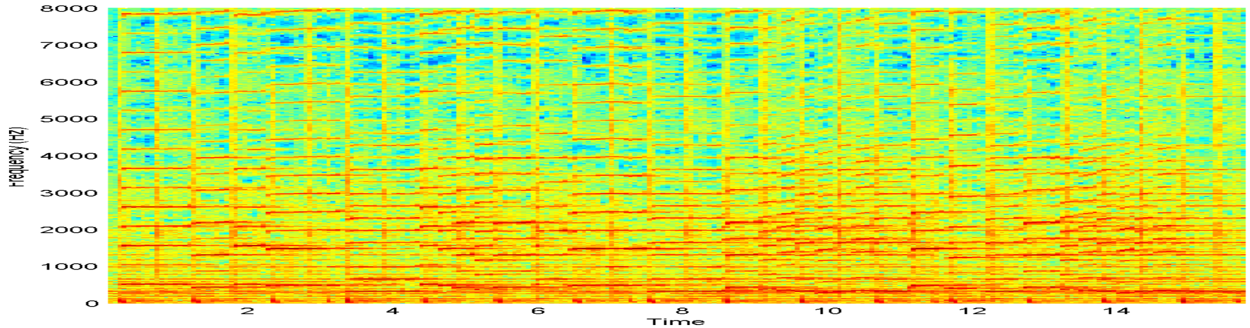
$$U = NS \approx M$$

$$S = pinv(N)M$$

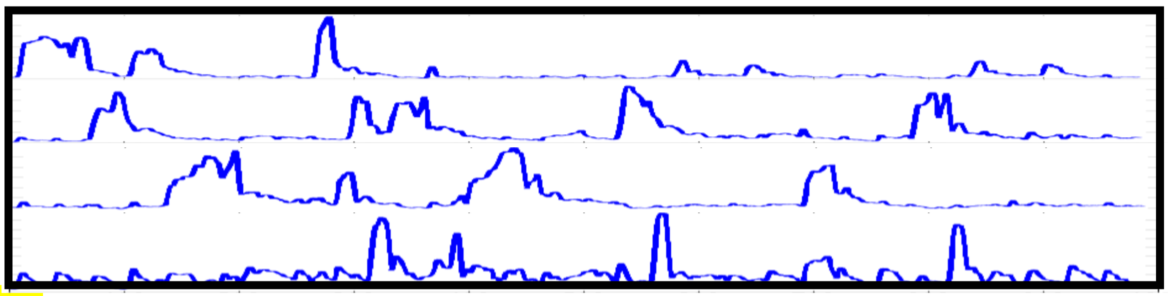
- Finding the best explanation of music M in terms of notes N
- Also finds the *score* S of M in terms of N

How about the other way?

M =



S =



N = M Pinv(S)

N = ?

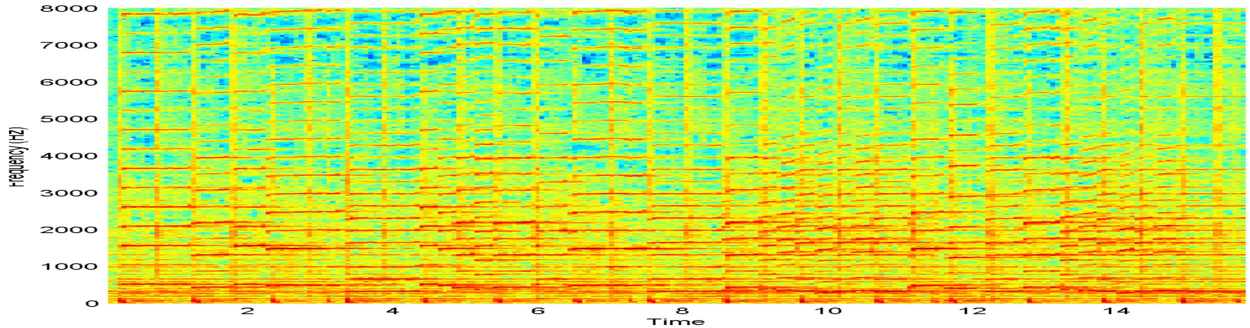
U = ?

$$U = NS \approx M$$
$$N = M \; pinv(S)$$

- Finding the *notes* N given music M and score S
- Also finds best explanation of M in terms of S

Finding Everything

M =



S = ?

N = ?

U = ?

$$U = NS \approx M$$

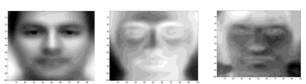
- Find the four notes and their score that generate the closest approximation to M

The same problem

M = 

W

Typical faces

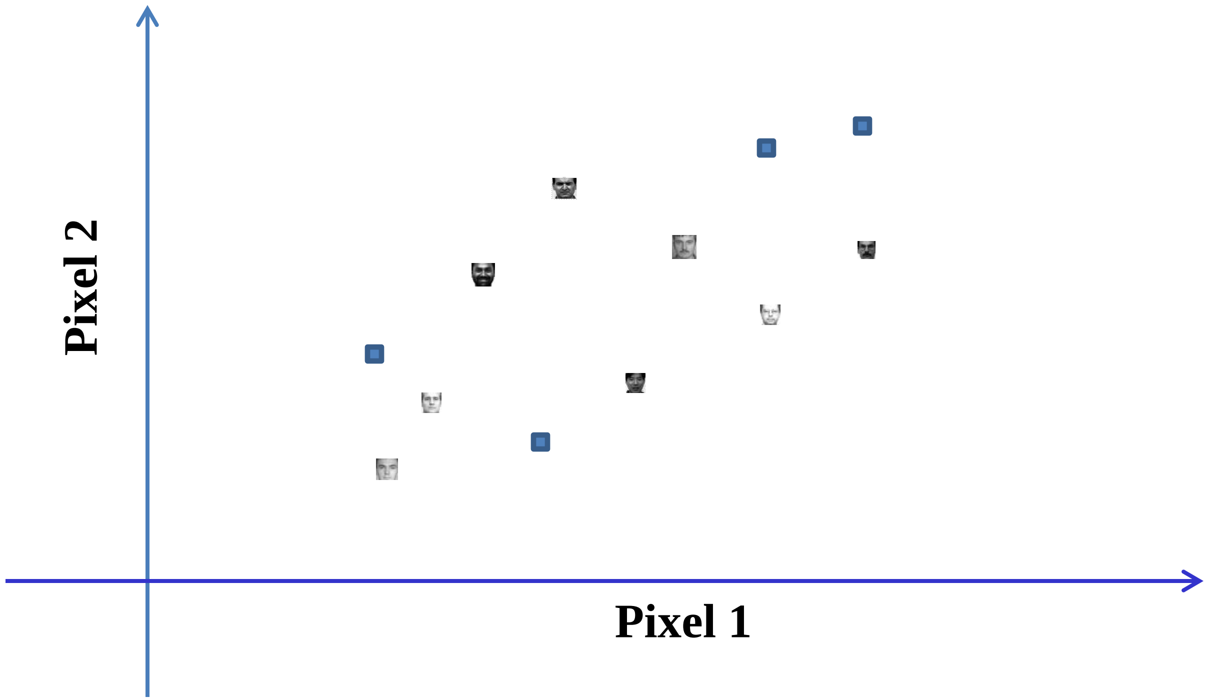


V

U = Approximation

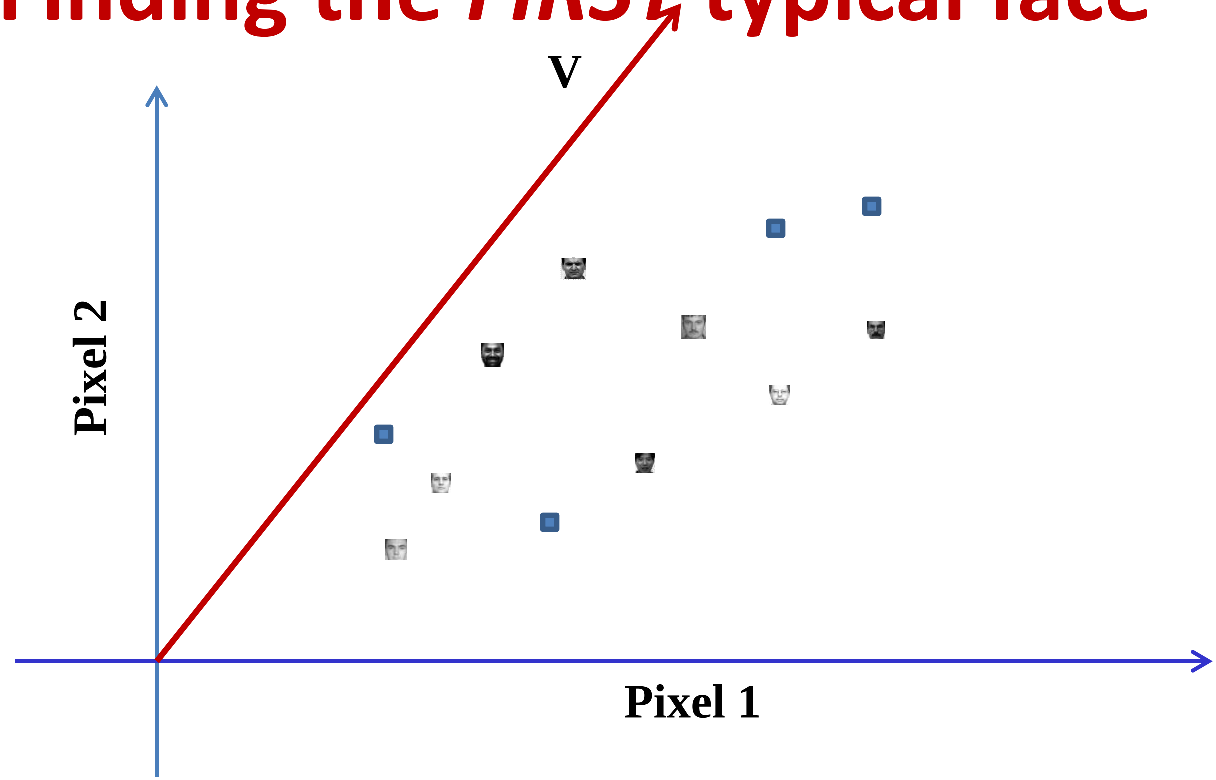
- Here W, V and U are ALL unknown and must be determined
  - Such that the squared error between U and M is minimum

Abstracting the problem: Finding the *FIRST* typical face



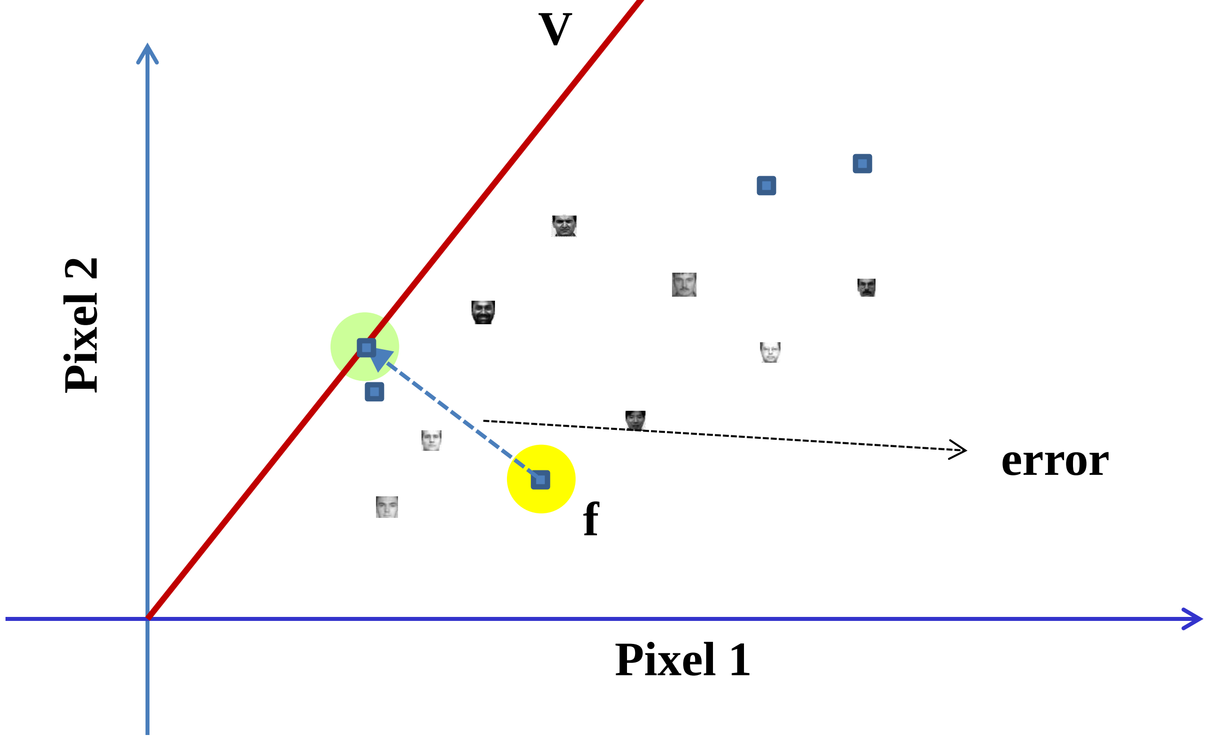
- Each "point" represents a face in "pixel space"

Abstracting the problem:
Finding the *FIRST* typical face



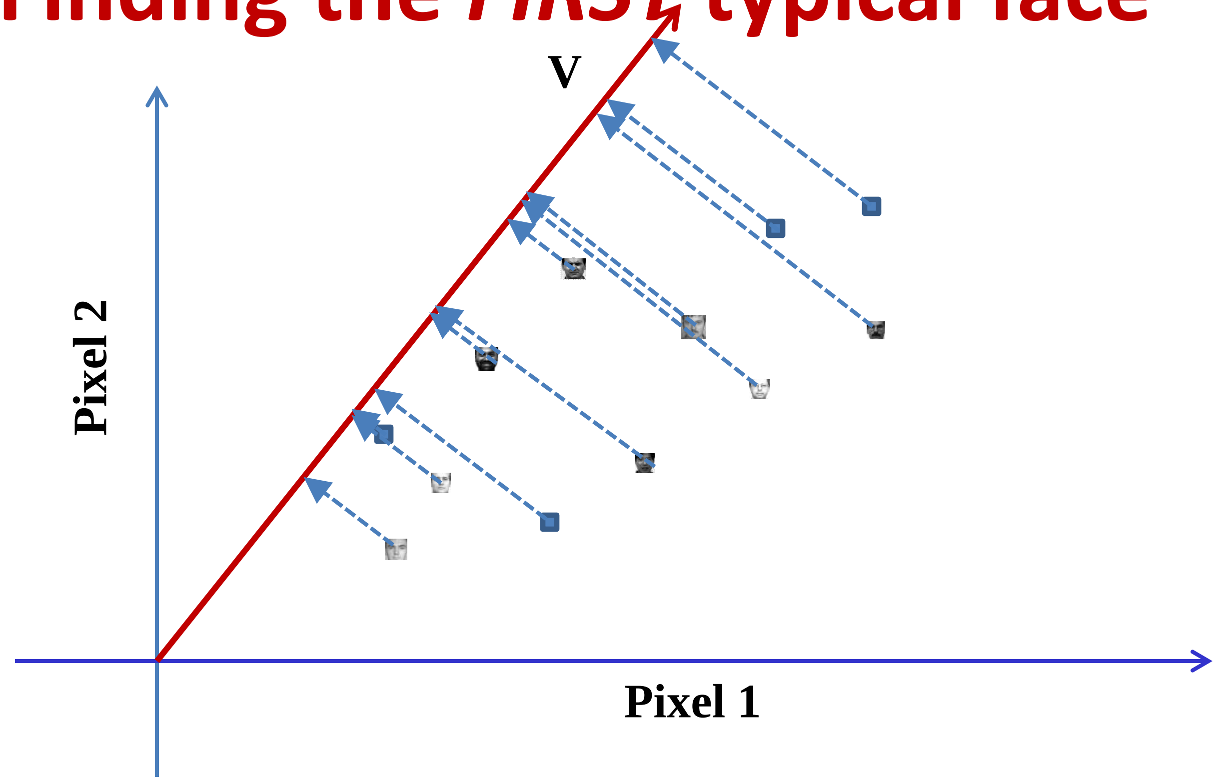
- Each “point” represents a face in “pixel space”
- Any “typical face” V is a vector in this space

Abstracting the problem:
Finding the *FIRST* typical face



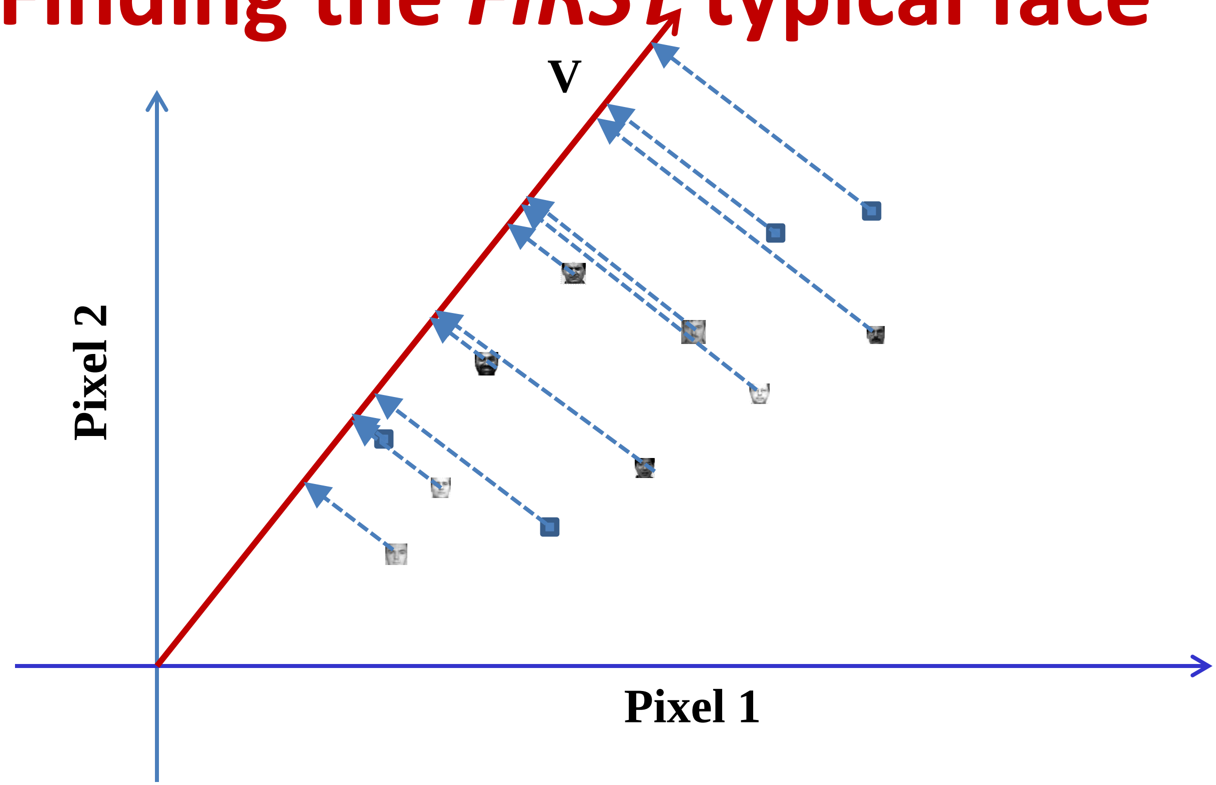
- Each "point" represents a face in "pixel space"
- The "typical face" V is a vector in this space
- The ***approximation*** $\mathrm{W}_{\mathrm{f},}\mathrm{V}$ for any face f is the *projection* of f onto V
- The distance between f and its projection $\mathrm{W}_{\mathrm{f}}\mathrm{V}$ is the *projection error* for f

Abstracting the problem: Finding the *FIRST* typical face



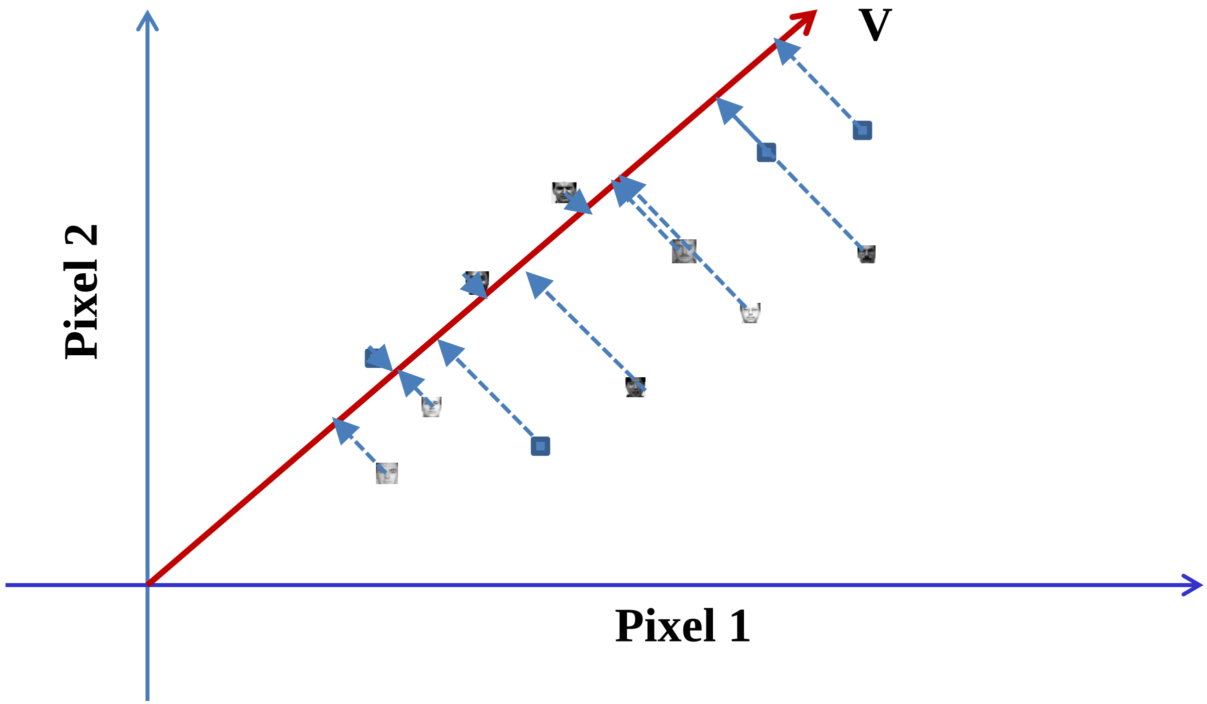
- *Every* face in our data will suffer error when approximated by its projection on V
- The total squared length of all error lines is the *total squared projection error*

Abstracting the problem:
Finding the *FIRST* typical face



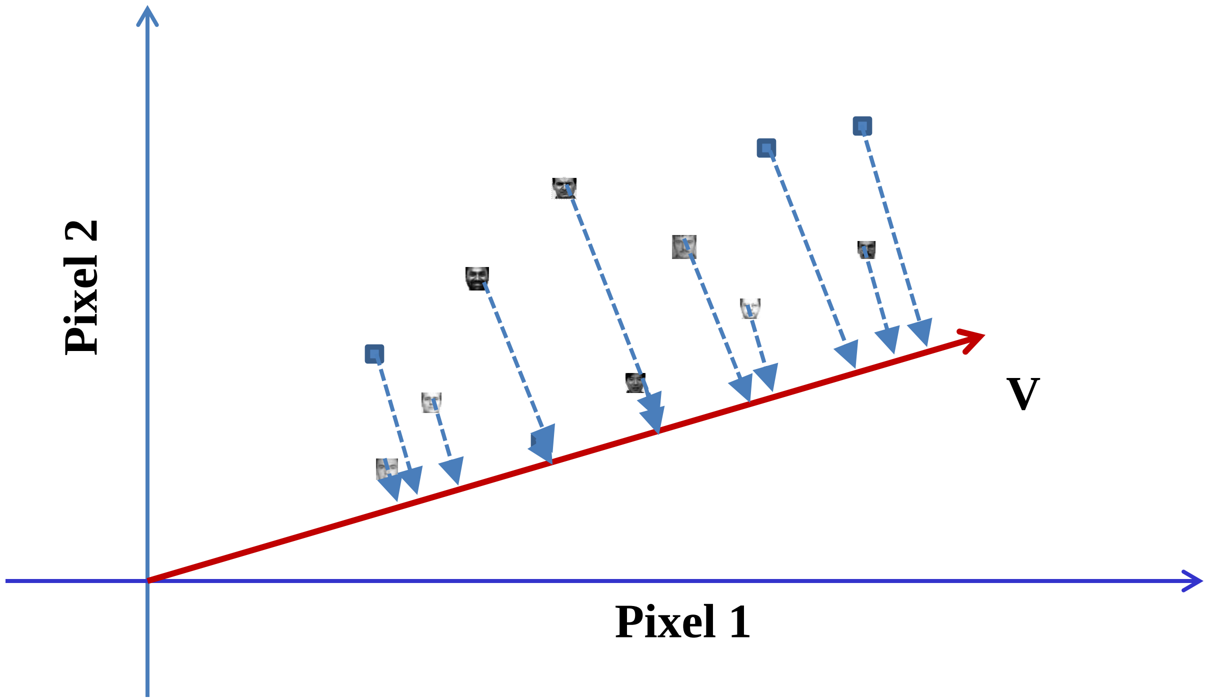
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem: Finding the *FIRST* typical face



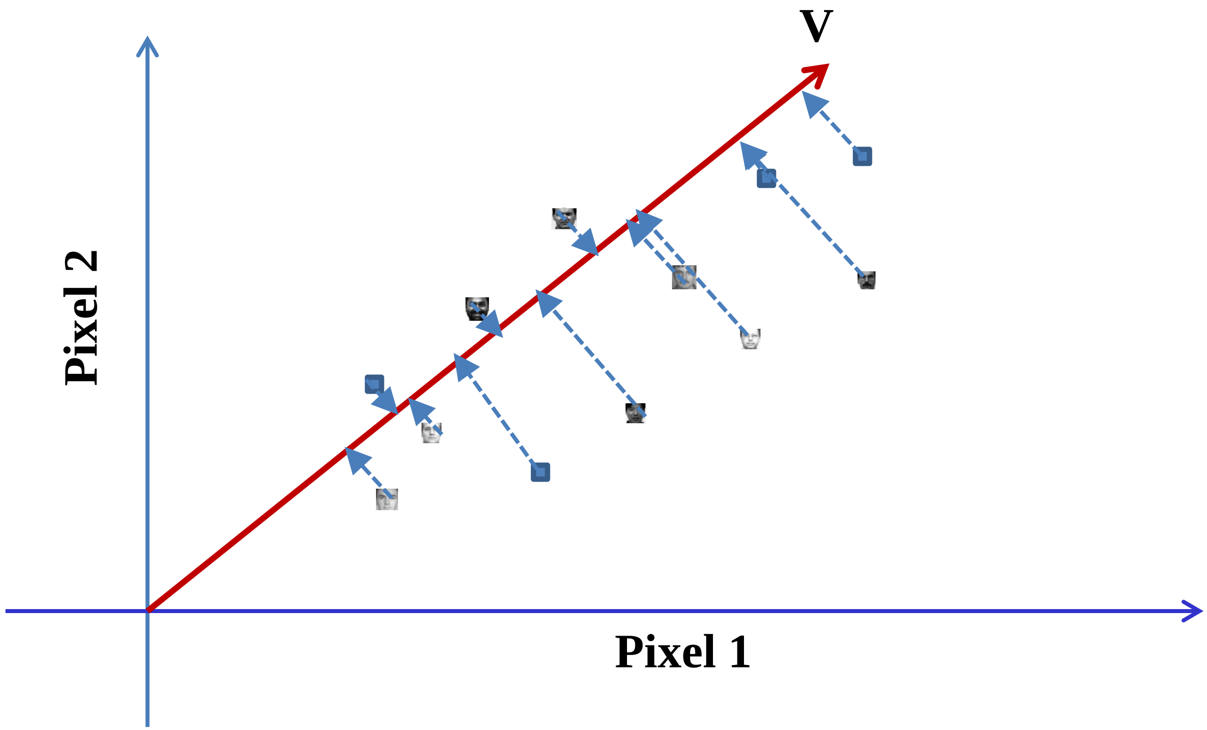
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem:
Finding the *FIRST* typical face



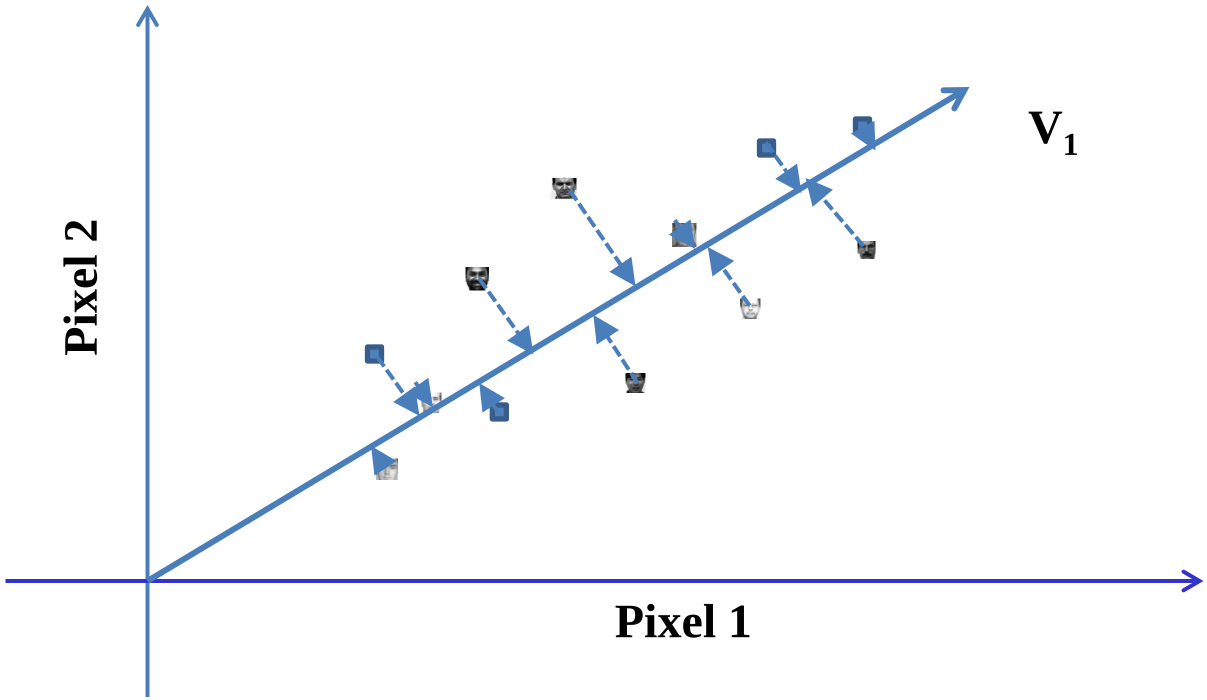
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem: Finding the *FIRST* typical face



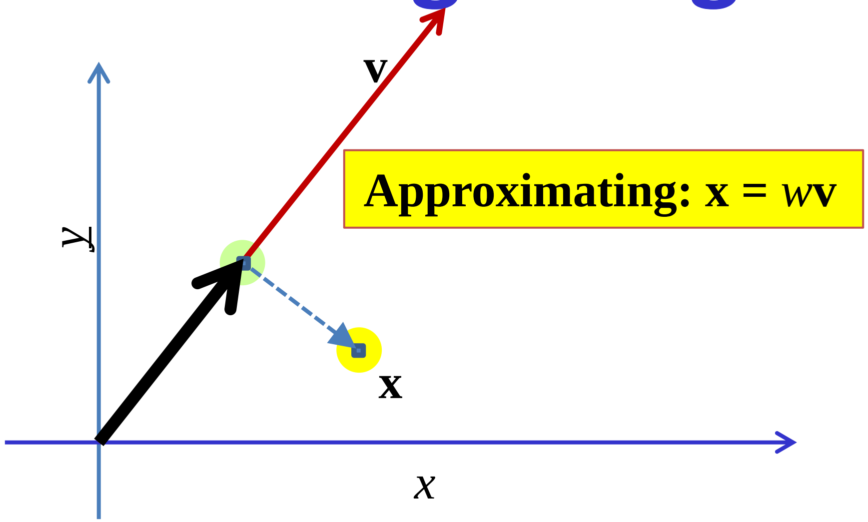
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!

Abstracting the problem:
Finding the *FIRST* typical face



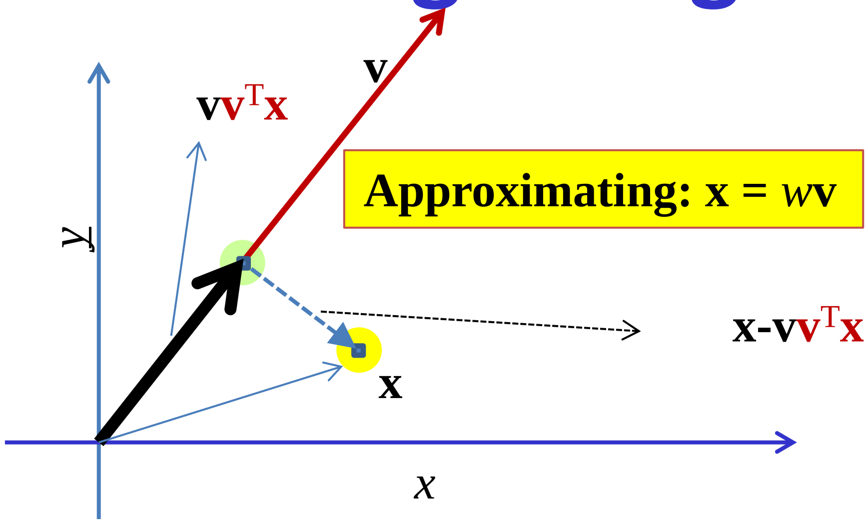
- The problem of finding the first typical face V_1:
  Find the V for which the total projection error is minimum!
- This "minimum squared error" V is our "best" first typical face
- **It is also the first *Eigen face***

Formalizing the Problem: Error from approximating a single vector



- Consider: approximating $\mathbf{x} = w\mathbf{v}$
  - E.g **x** is a face, and "**v**" is the "typical face"
- Finding an approximation $w\mathbf{v}$ which is closest to **x**
  - In a Euclidean sense
  - Basically projecting **x** onto **v**

Formalizing the Problem: Error from approximating a single vector



- Projection of a vector **x** on to a vector **v**

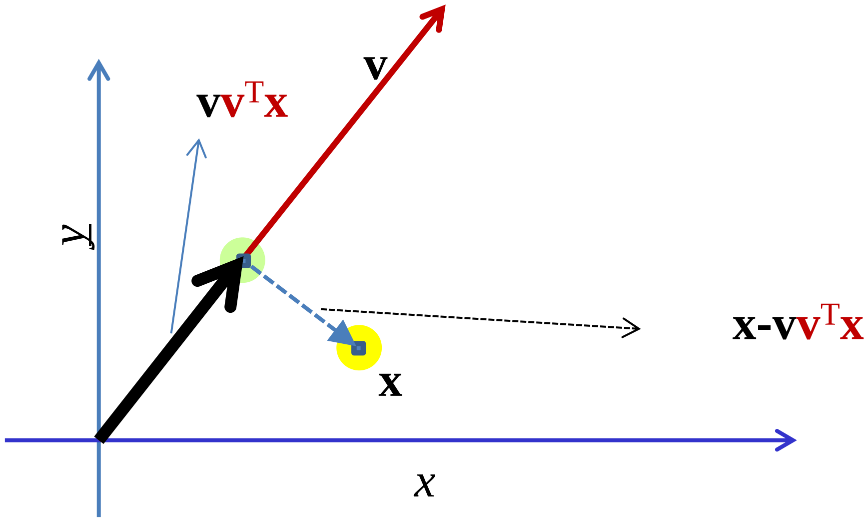
$$\hat{\mathbf{x}} = \mathbf{v}\frac{\mathbf{v}^T\mathbf{x}}{|\mathbf{v}|}$$

- Assuming **v** is of unit length: $\hat{\mathbf{x}} = \mathbf{v}\mathbf{v}^T\mathbf{x}$

$$error = \mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x}$$

$$\text{squared error} = \left\|\mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x}\right\|^2$$

Error from approximating a single vector

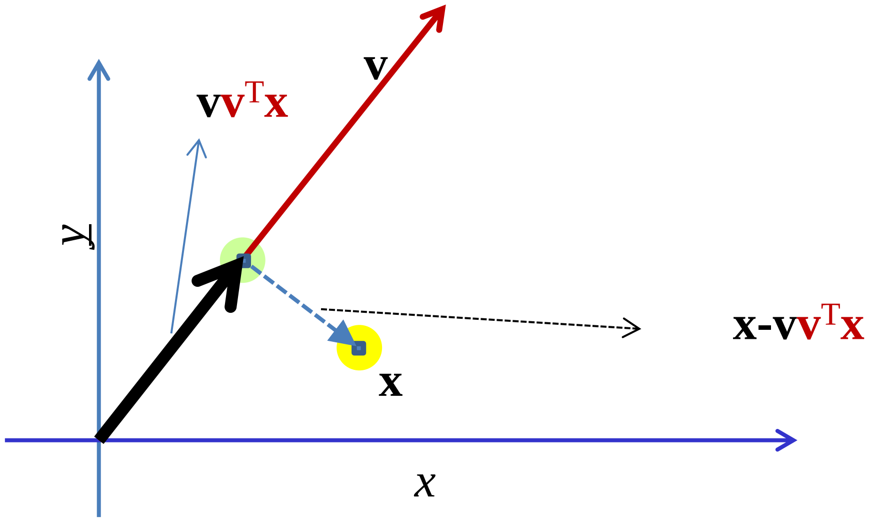


- Minimum squared approximation error from approximating $\mathbf{x}$ as it as $w\mathbf{v}$

$$e(\mathbf{x}) = \left\| \mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x} \right\|^2$$

- Optimal value of w: $w = \mathbf{v}^T\mathbf{x}$

Error from approximating a single vector

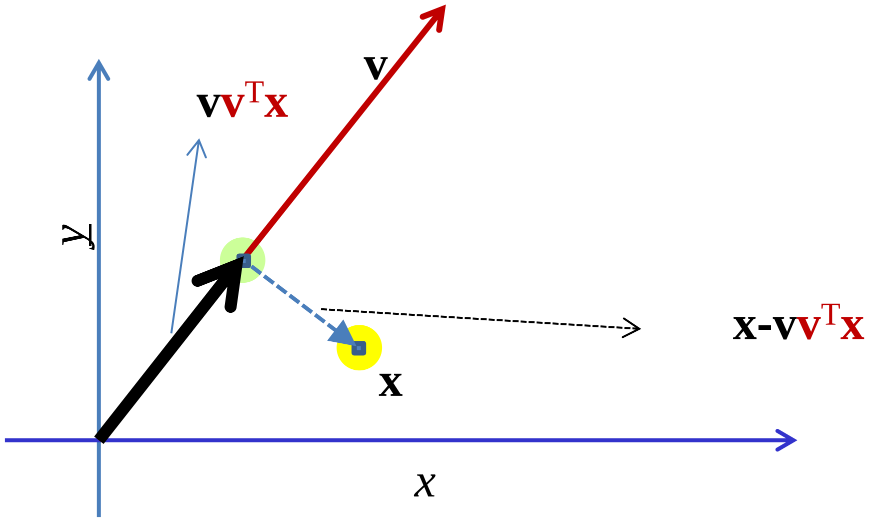


- Error from projecting a vector **x** on to a vector onto a unit vector **v** $e(\mathbf{x}) = \left\| \mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x} \right\|^2$

$$e(\mathbf{x}) = \left(\mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x}\right)^T\left(\mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x}\right) = \left(\mathbf{x}^T - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\right)\left(\mathbf{x} - \mathbf{v}\mathbf{v}^T\mathbf{x}\right)$$

$$= \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} + \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{v}\mathbf{v}^T\mathbf{x}$$

Error from approximating a single vector

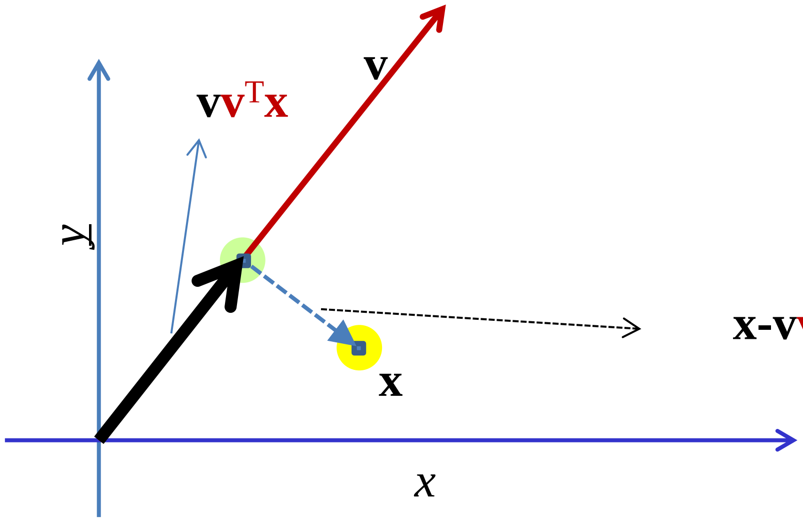


- Error from projecting a vector **x** on to a vector onto a unit vector **v** $e(\mathbf{x}) = \left\| \mathbf{x} - \mathbf{w}^T \mathbf{x} \right\|^2$

$$e(\mathbf{x}) = \left(\mathbf{x} - \mathbf{w}^T\mathbf{x}\right)^T \left(\mathbf{x} - \mathbf{w}^T\mathbf{x}\right) = \left(\mathbf{x}^T - \mathbf{x}^T\mathbf{w}^T\right)\left(\mathbf{x} - \mathbf{w}^T\mathbf{x}\right)$$

$$= \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} + \mathbf{x}^T\mathbf{v}\underbrace{\mathbf{v}^T\mathbf{v}}_{=1}\mathbf{v}^T\mathbf{x}$$

Error from approximating a single vector



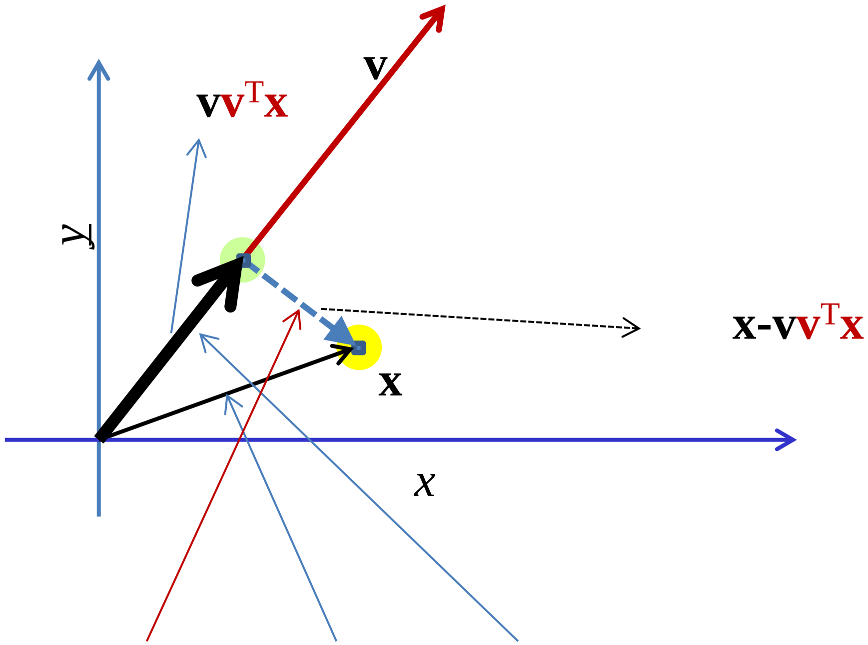
- Error from projecting a vector $\mathbf{x}$ on to a vector onto a unit vector $\mathbf{v}$ $\quad e(\mathbf{x}) = \left\| \mathbf{x} - \mathbf{w}^T\mathbf{x} \right\|^2$

$$e(\mathbf{x}) = (\mathbf{x} - \mathbf{w}^T\mathbf{x})^T(\mathbf{x} - \mathbf{w}^T\mathbf{x}) = (\mathbf{x}^T - \mathbf{x}^T\mathbf{w}^T)(\mathbf{x} - \mathbf{w}^T\mathbf{x})$$

$$= \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x} + \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x}$$

$$\boxed{e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x}}$$

Error from approximating a single vector

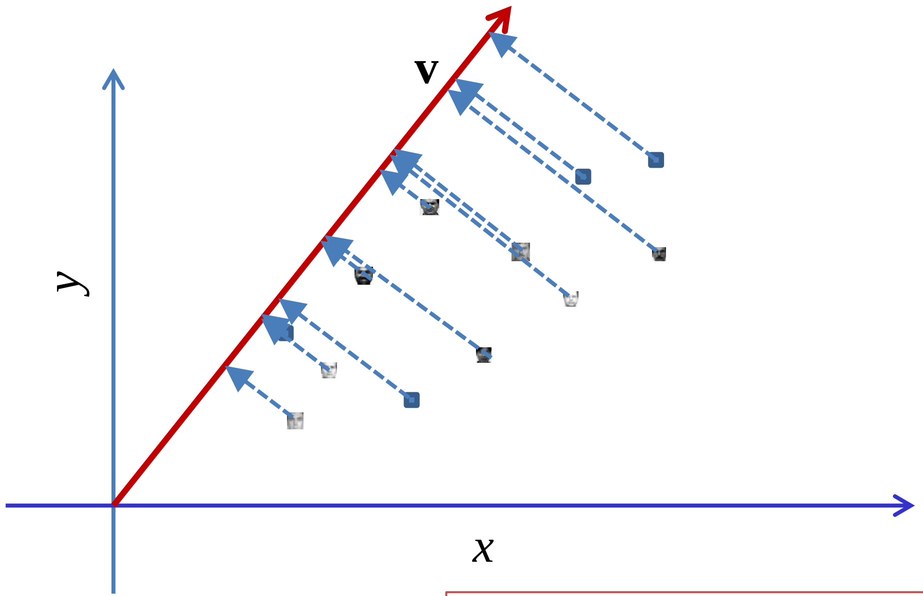


$$\mathbf{e}(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \underbrace{\mathbf{x}^T\mathbf{v}}.\underbrace{\mathbf{v}^T\mathbf{x}}$$

Length of projection

This is the very familiar pythogoras' theorem!!

Error for *many* vectors

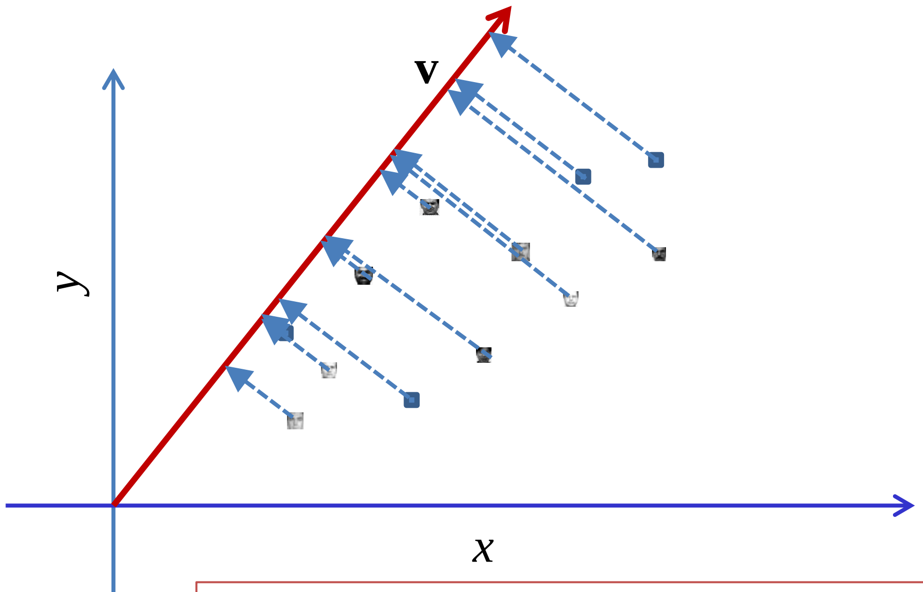


- Error for one vector: $\boxed{e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x}}$
- Error for many vectors

$$E = \sum_i e(\mathbf{x}_i) = \sum_i \left(\mathbf{x}_i^T\mathbf{x}_i - \mathbf{x}_i^T\mathbf{v}\mathbf{v}^T\mathbf{x}_i\right) = \sum_i \mathbf{x}_i^T\mathbf{x}_i - \sum_i \mathbf{x}_i^T\mathbf{v}\mathbf{v}^T\mathbf{x}_i$$

- **Goal: Estimate v to minimize this error!**

Error for *many* vectors



- Total error: $\boxed{E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{v}\mathbf{v}^T \mathbf{x}_i}$
- Add constraint: $\mathbf{v}^T\mathbf{v} = 1$
- Constrained objective to minimize:

$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{v}\mathbf{v}^T \mathbf{x}_i + \lambda\left(\mathbf{v}^T\mathbf{v} - 1\right)$$

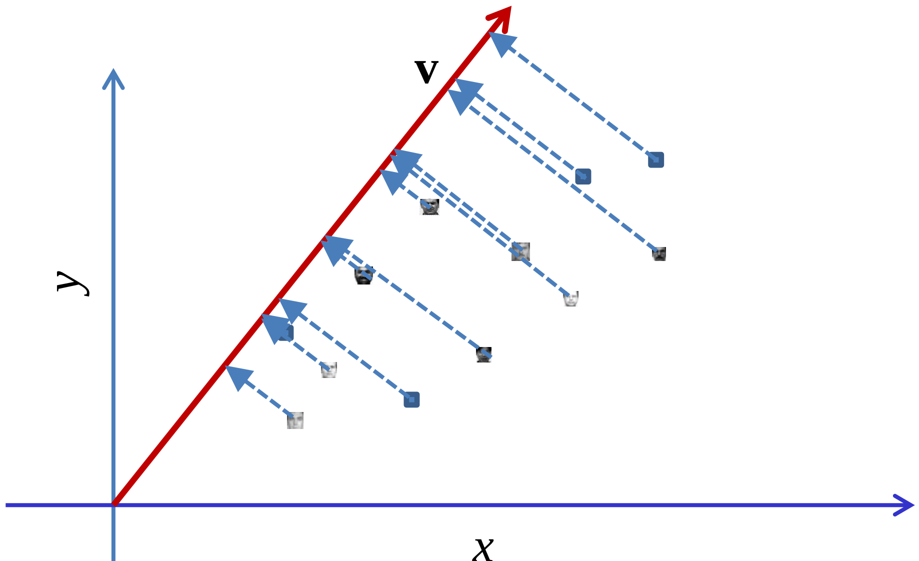
Two Matrix Identities

- Derivative w.r.t $\mathbf{v}$

$$\frac{d\mathbf{v}^T\mathbf{v}}{d\mathbf{v}} = 2\mathbf{v}$$

$$\frac{d\mathbf{x}^T\mathbf{v}\mathbf{v}^T\mathbf{x}}{d\mathbf{v}} = 2\mathbf{x}\mathbf{x}^T\mathbf{v}$$

Minimizing error



$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{v}\mathbf{v}^T \mathbf{x}_i + \lambda\left(\mathbf{v}^T \mathbf{v} - 1\right)$$

- Differentiating w.r.t **v** and equating to 0

$$-2\sum_i \mathbf{x}_i \mathbf{x}_i^T \mathbf{v} + 2\lambda \mathbf{v} = 0$$

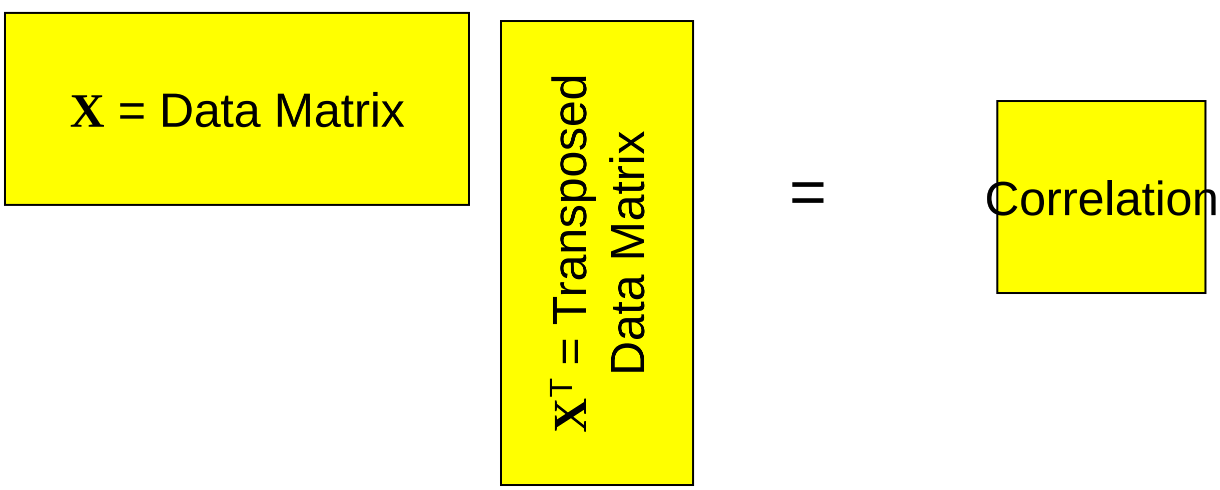
$$\left(\sum_i \mathbf{x}_i \mathbf{x}_i^T\right)\mathbf{v} = \lambda \mathbf{v}$$

The correlation matrix

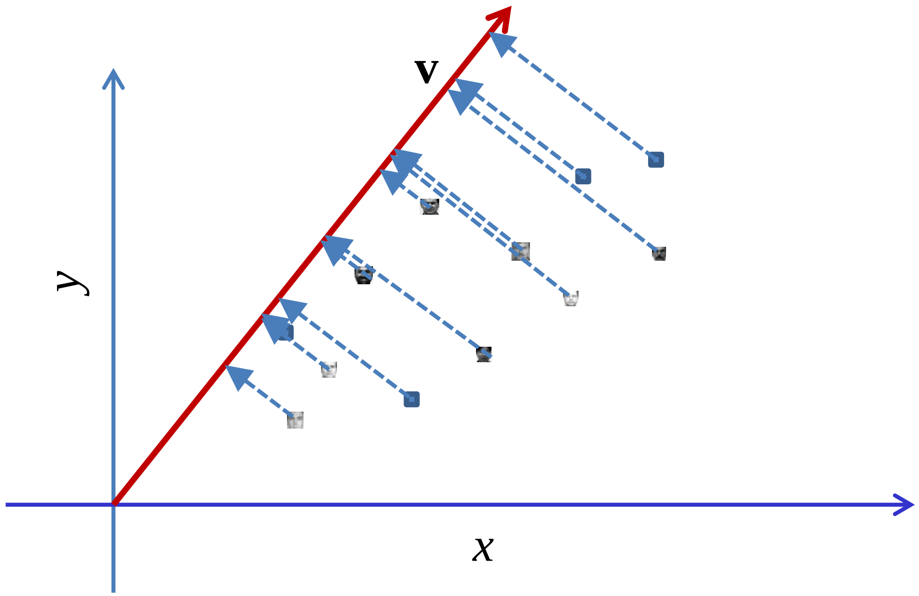
$$\left(\sum_i \mathbf{x}_i \mathbf{x}_i^T\right)\mathbf{v} = \lambda \mathbf{v}$$

- The encircled term is the *correlation matrix*

$$\mathbf{X} = \left[\mathbf{x}_1\ \mathbf{x}_2\ \ldots \mathbf{x}_N\right] \qquad \sum_i \mathbf{x}_i \mathbf{x}_i^T = \mathbf{X}\mathbf{X}^T = \mathbf{R}$$



The best “basis”



- The minimum-error basis is found by solving

$$\mathbf{Rv} = \lambda \mathbf{v}$$

- $\mathbf{v}$ is an Eigen vector of the correlation matrix $\mathbf{R}$
  - λ is the corresponding Eigen value

What about the total error?

$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{w}^T \mathbf{x}_i$$

- $\mathbf{x}^{\mathrm{T}}\mathbf{v} = \mathbf{v}^{\mathrm{T}}\mathbf{x}$ (inner product)

$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{v}^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{v} \qquad = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \left(\sum_i \mathbf{x}_i \mathbf{x}_i^T \right) \mathbf{v}$$

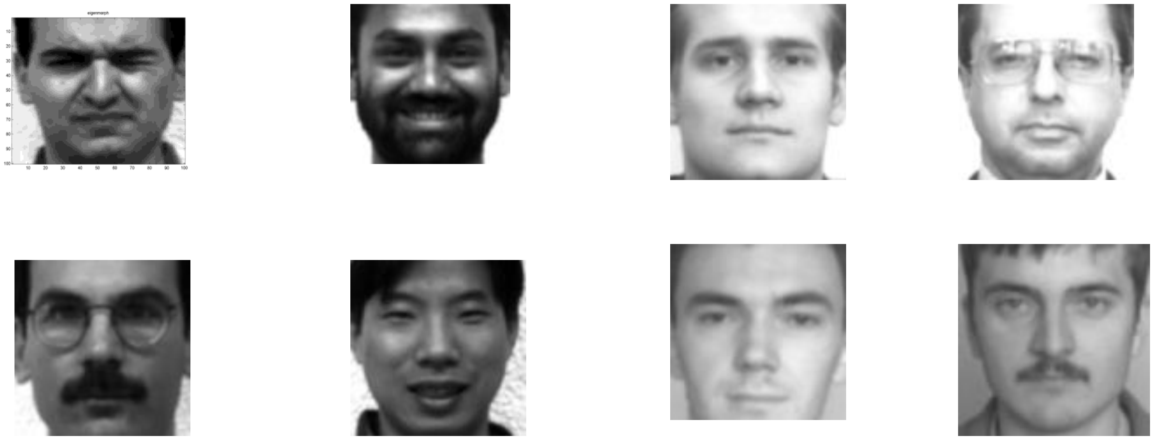
$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \mathbf{R} \mathbf{v} = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \mathbf{v}^T \lambda \mathbf{v} = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \lambda \mathbf{v}^T \mathbf{v}$$

$$\boxed{E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \lambda}$$

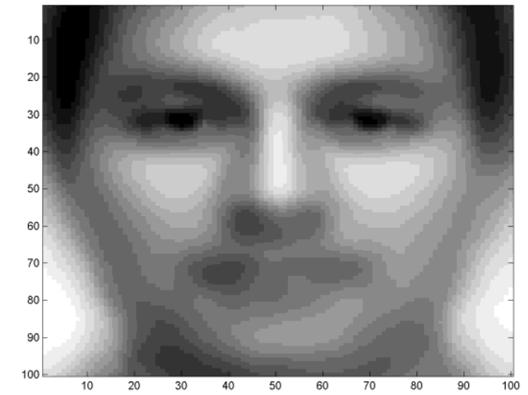
Minimizing the error

- The total error is $$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \lambda$$
- We already know that the optimal basis is an Eigen vector
- The total error depends on the *negative* of the corresponding Eigen value
- To *minimize* error, we must *maximize* λ
- i.e. Select the Eigen vector with the largest Eigen value

The typical face

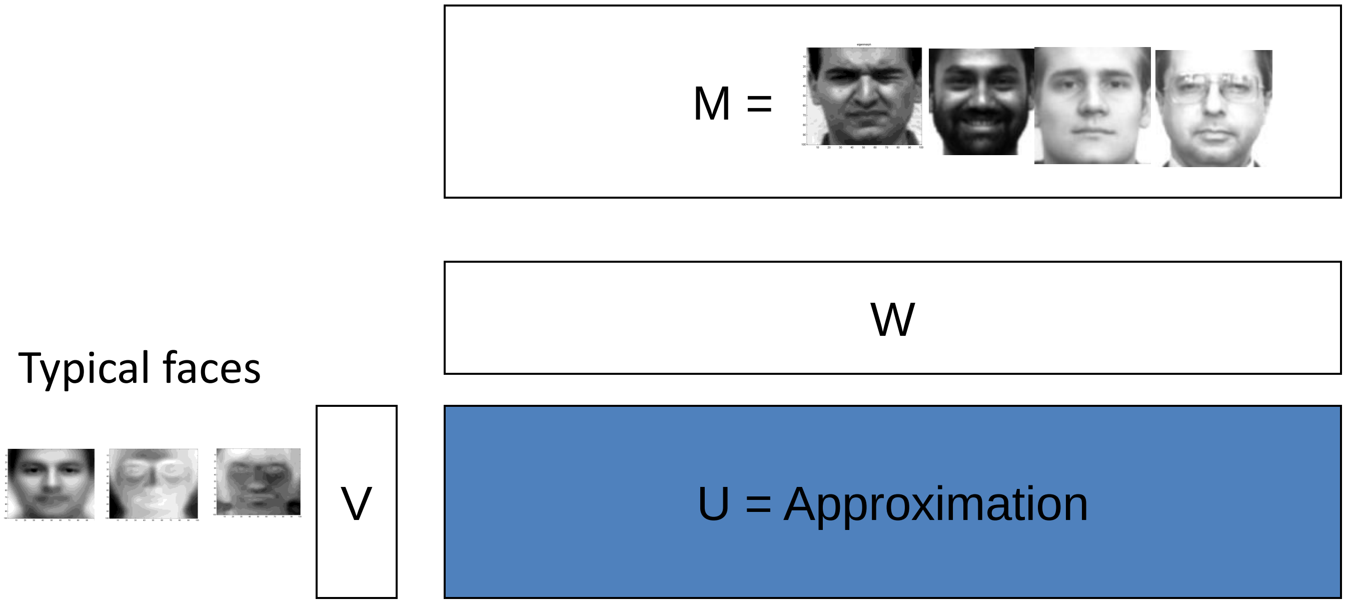


The typical face



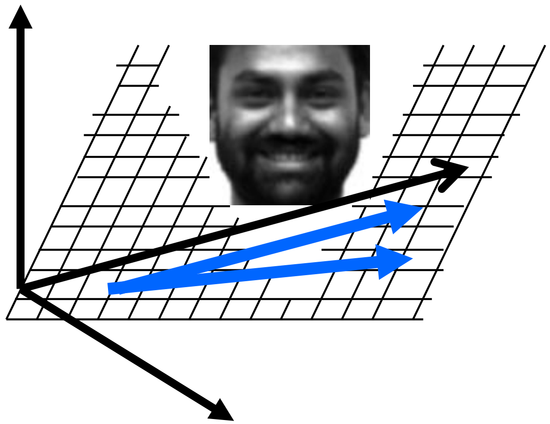
- Compute the correlation matrix for your data
  - Arrange them in matrix **X** and compute **R** = **XX**^{T}
- Compute the *principal* Eigen vector of R
  - The Eigen vector with the largest Eigen value
- This is the typical face

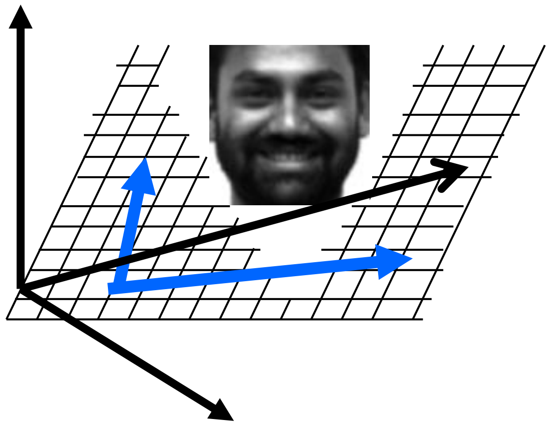
With many typical faces



- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + \ldots + w_{f,k} V_k$
- Here W, V and U are ALL unknown and must be determined
  - Such that the squared error between U and M is minimum

With multiple bases



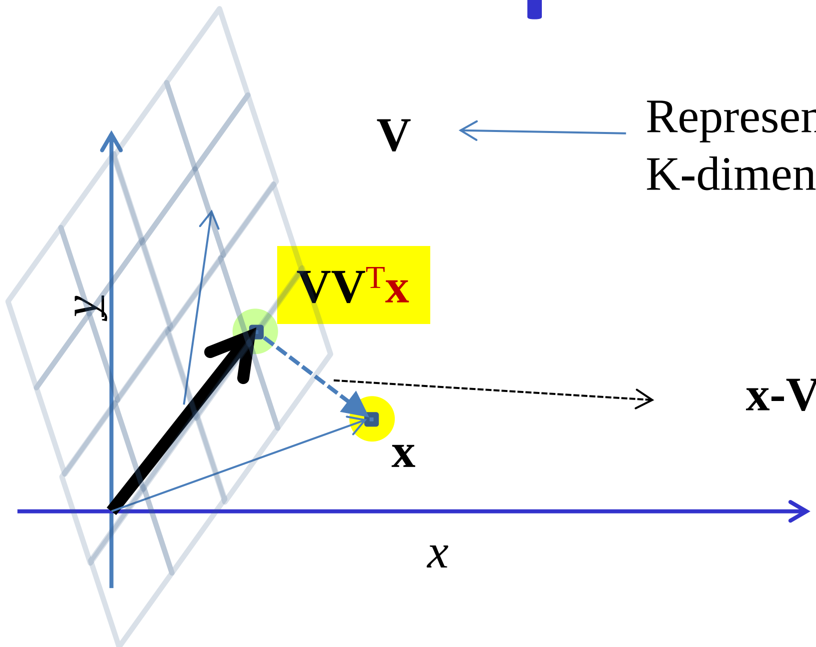


- **Assumption: all bases v_1 v_2 v_3.. are unit length**
- **Assumption: all bases are orthogonal to one another: $v_i^T v_j = 0$ if i != j**
  - We are trying to find the optimal K-dimensional subspace to project the data
  - Any set of vectors in this subspace will define the subspace
  - Constraining them to be orthogonal does not change this

- I.e. if $\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2\ \mathbf{v}_3\ \ldots]$, $\mathbf{V}^T\mathbf{V} = \mathbf{I}$
  - $\mathrm{Pinv}(\mathbf{V}) = \mathbf{V}^T$

- Projection matrix for $\mathbf{V}$ = $\mathbf{V}\mathrm{Pinv}(\mathbf{V}) = \mathbf{V}\mathbf{V}^T$

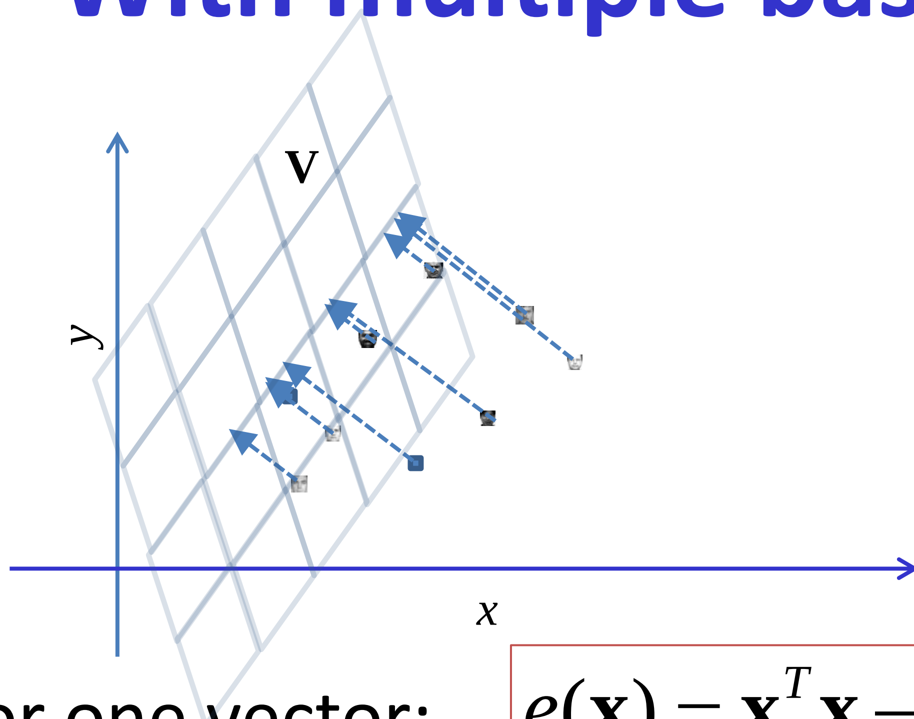
With multiple bases



- Optimal projection for a vector $\hat{\mathbf{x}} = \mathbf{V}\mathbf{V}^T\mathbf{x}$
- Error vector = $\mathbf{x} - \hat{\mathbf{x}} = \mathbf{x} - \mathbf{V}\mathbf{V}^T\mathbf{x}$

- Error length = $e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}$

With multiple bases



- Error for one vector: $\boxed{e(\mathbf{x}) = \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{V}\mathbf{V}^T\mathbf{x}}$
- Error for many vectors

$$E = \sum_i \mathbf{x}_i^T\mathbf{x}_i - \sum_i \mathbf{x}_i^T\mathbf{V}\mathbf{V}^T\mathbf{x}_i$$

- **Goal: Estimate $\mathbf{V}$ to minimize this error!**

Minimizing error

- With regularization $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{I}$, objective to minimize

$$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_i \mathbf{x}_i^T \mathbf{V}\mathbf{V}^T \mathbf{x}_i + trace\left(\Lambda\left(\mathbf{V}^T\mathbf{V} - \mathbf{I}\right)\right)$$

  – Note: now Λ is a diagonal matrix
  – The regularization simply ensures that $\mathbf{v}^{\mathrm{T}}\mathbf{v} = 1$ for every basis

- Differentiating w.r.t $\mathbf{V}$ and equating to 0

$$-2\left(\sum_i \mathbf{x}_i \mathbf{x}_i^T\right)\mathbf{V} + 2\Lambda\mathbf{V} = 0$$

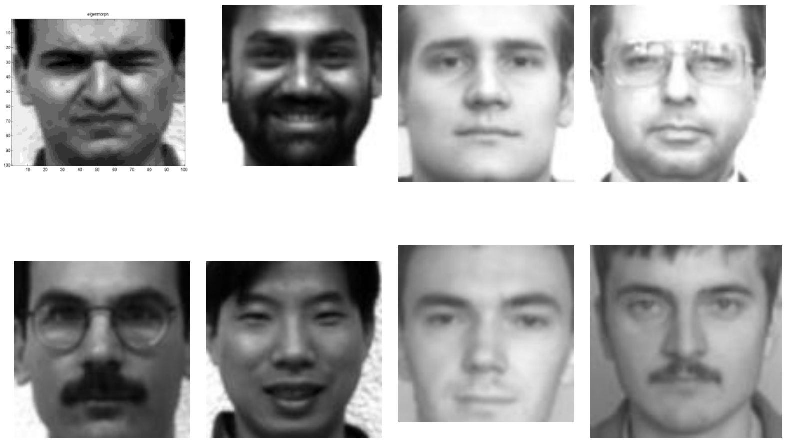
$$\mathbf{R}\mathbf{V} = \Lambda\mathbf{V}$$

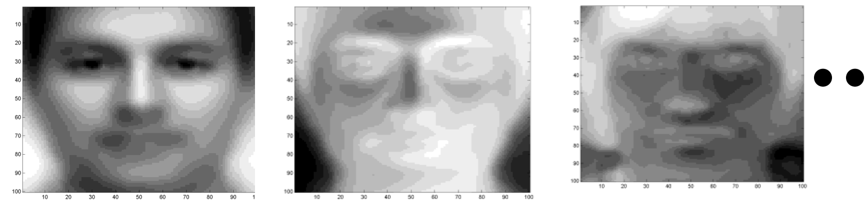
Finding the optimal K bases

$$\mathbf{RV} = \Lambda\mathbf{V}$$

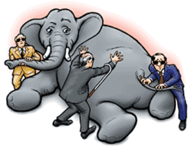
- Compute the Eigendecompsition of the correlation matrix
- Select *K* Eigen vectors
- But which *K*?
- Total error = $$E = \sum_i \mathbf{x}_i^T \mathbf{x}_i - \sum_{j=1}^{K} \lambda_j$$
- Select *K* eigen vectors corresponding to the *K* largest Eigen values

Eigen Faces!

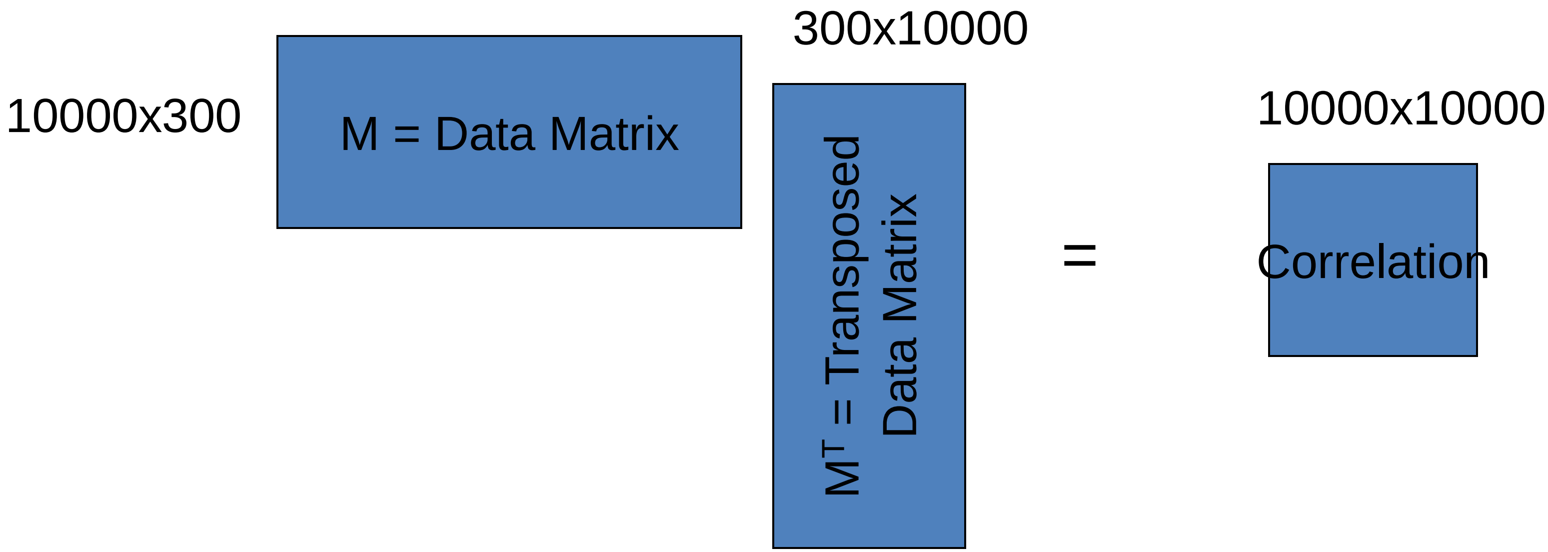




- Arrange your input data into a matrix $\mathbf{X}$
- Compute the correlation $\mathbf{R} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$
- Solve the Eigen decomposition: $\mathbf{R}\mathbf{V} = \Lambda\mathbf{V}$
- The Eigen vectors corresponding to the K largest eigen values are our optimal bases
- We will refer to these as *eigen faces.*



How many Eigen faces



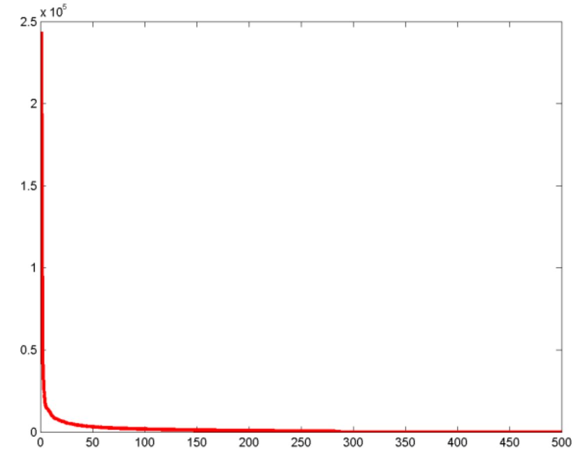
- How to choose "K" (number of Eigen faces)
- Lay all faces side by side in vector form to form a matrix
  - In my example: 300 faces. So the matrix is 10000 x 300
- Multiply the matrix by its transpose
  - The correlation matrix is 10000x10000

Eigen faces

[U,S] = eig(correlation)

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

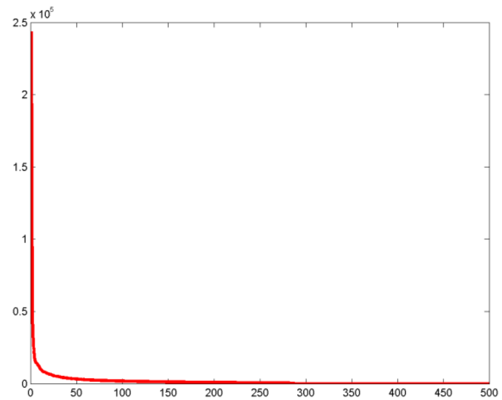
$$U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \cdots \end{bmatrix}$$



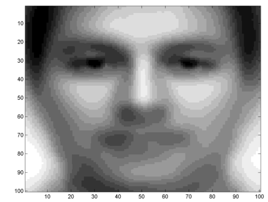
- Compute the eigen vectors
  - Only 300 of the 10000 eigen values are non-zero
    - Why?
- Retain eigen vectors with high eigen values (>0)
  - Could use a higher threshold

Eigen Faces

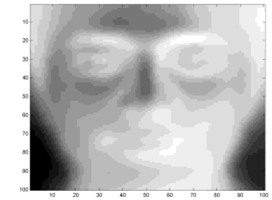
$$U = \begin{bmatrix} \text{eigenface1} & \text{eigenface2} & \cdots \end{bmatrix}$$

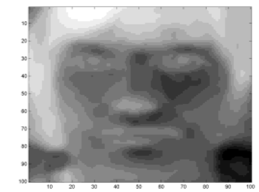


eigenface1



eigenface2

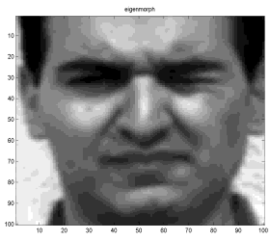


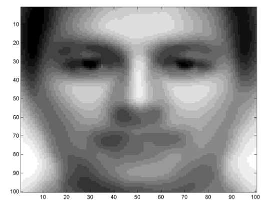
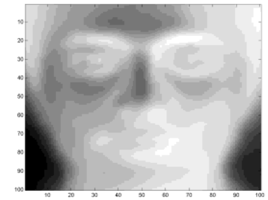
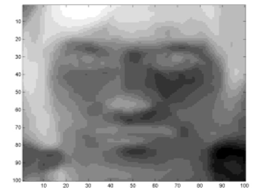


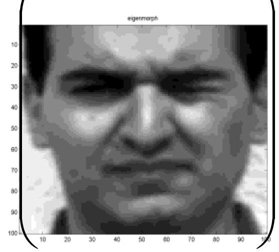
eigenface3

- 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  …

Representation 

= $[w_1 \; w_2 \; w_3 \; \ldots.]^T$

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

Energy Compaction Example

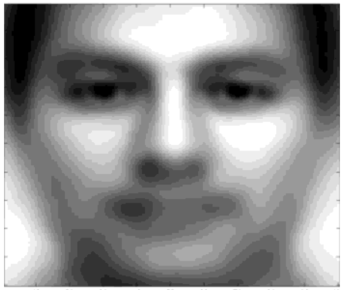
- One outcome of the "energy compaction principle": the approximations are recognizable

- Approximating a face with one basis:

$$f = w_1 \mathbf{v}_1$$

Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable

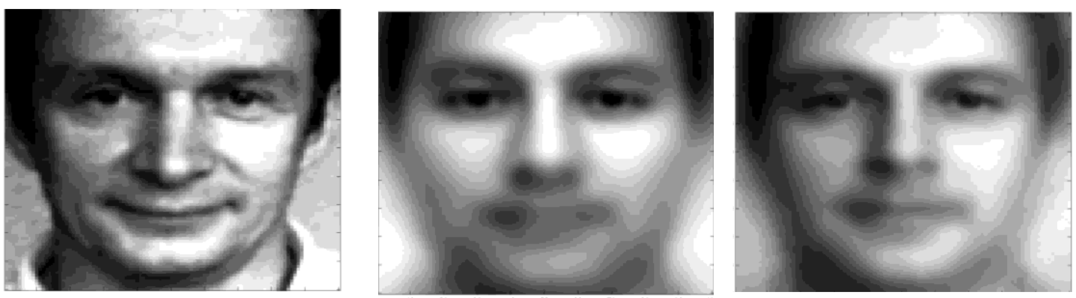



- Approximating a face with one Eigenface:

$$f = w_1 \mathbf{v}_1$$

Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable

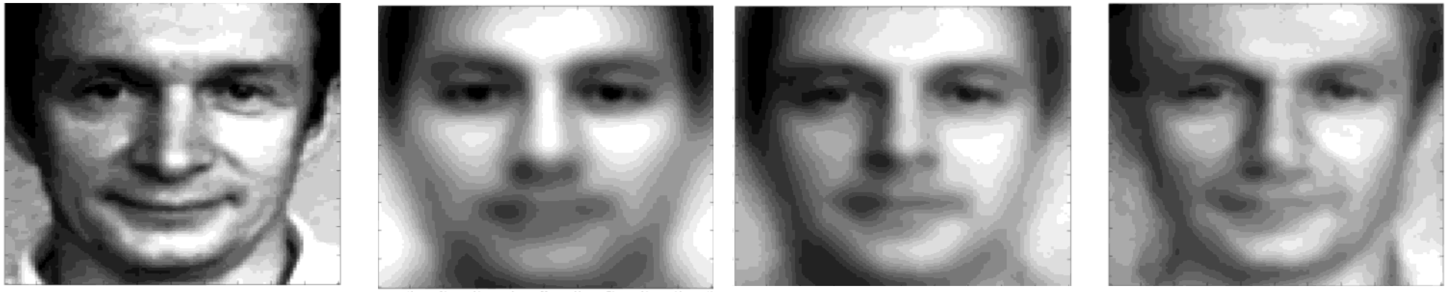


- Approximating a face with 10 eigenfaces:

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots w_{10}\mathbf{v}_{10}$$

Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable



- Approximating a face with 30 eigenfaces:

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30}$$

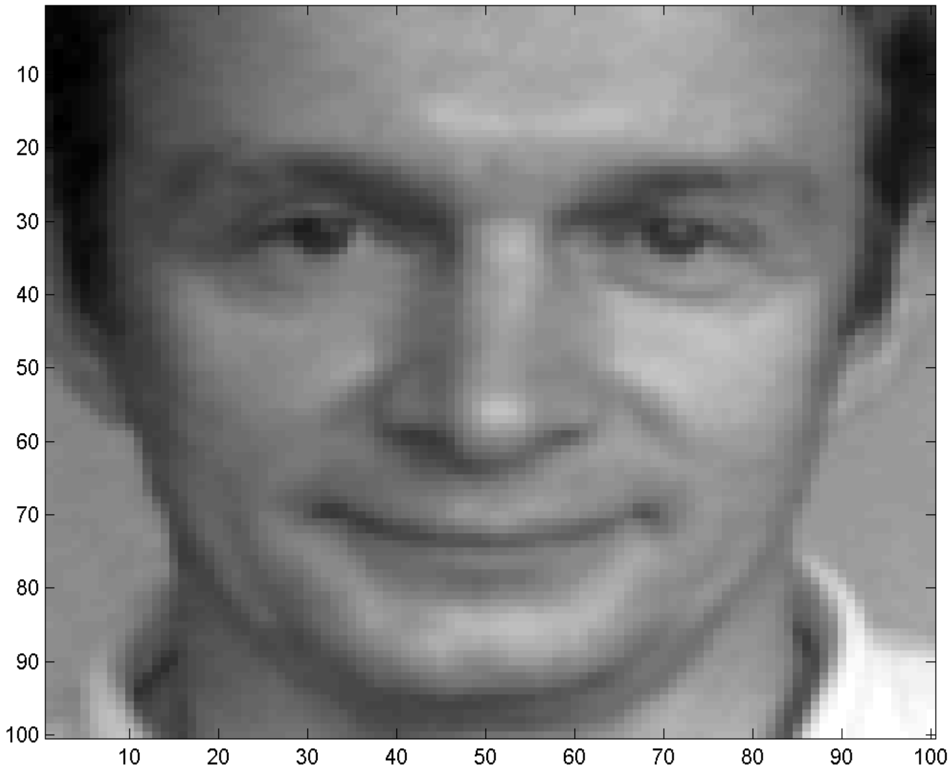
Energy Compaction Example

- One outcome of the "energy compaction principle": the approximations are recognizable

- Approximating a face with 60 eigenfaces:

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30} + \ldots + w_{60}\mathbf{v}_{60}$$

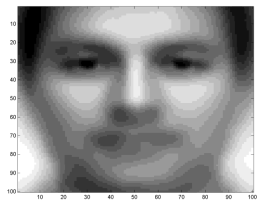
How did I do this?



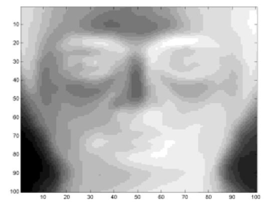
- Hint: only changing weights assigned to Eigen faces..

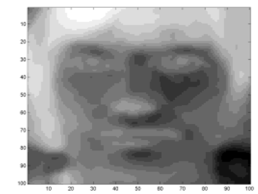
Class specificity

eigenface1



eigenface2





eigenface3

- The Eigenimages (bases) are very specific to the class of data they are trained on
  - Faces here
- They will not be useful for other classes

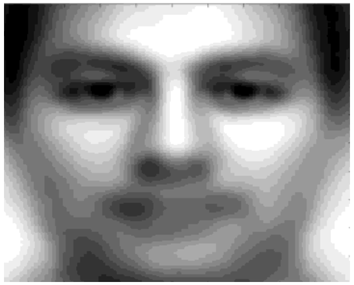
Class specificity

- Eigen bases are class specific

- Composing a fishbowl from Eigenfaces

Class specificity

- Eigen bases are class specific

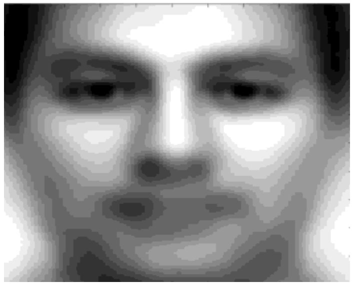


- Composing a fishbowl from Eigenfaces
- With 1 basis

$$f = w_1 \mathbf{v}_1$$

Class specificity

- Eigen bases are class specific

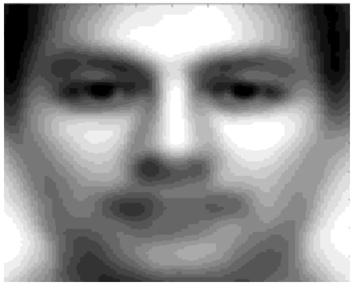


- Composing a fishbowl from Eigenfaces
- With 10 bases

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + ... + w_{10}\mathbf{v}_{10}$$

Class specificity

- Eigen bases are class specific

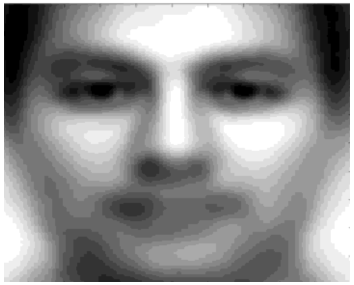
   

- Composing a fishbowl from Eigenfaces
- With 30 bases

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30}$$

Class specificity

- Eigen bases are class specific

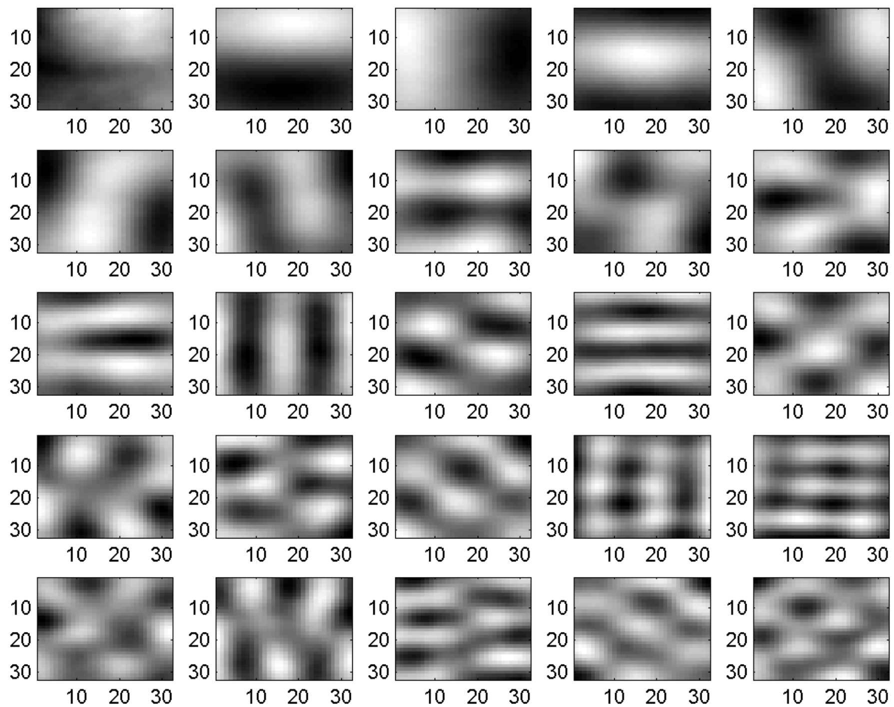




- Composing a fishbowl from Eigenfaces
- With 100 bases

$$f = w_1\mathbf{v}_1 + w_2\mathbf{v}_2 + \ldots + w_{10}\mathbf{v}_{10} + \ldots + w_{30}\mathbf{v}_{30} + \ldots + w_{100}\mathbf{v}_{100}$$

Universal bases

- Universal bases..



- End up looking a lot like *discrete cosine transforms!!!!*
- *DCTs are the best "universal" bases*
  - *If you don't know what your data are, use the DCT*

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

Using SVD to compute Eigenbases

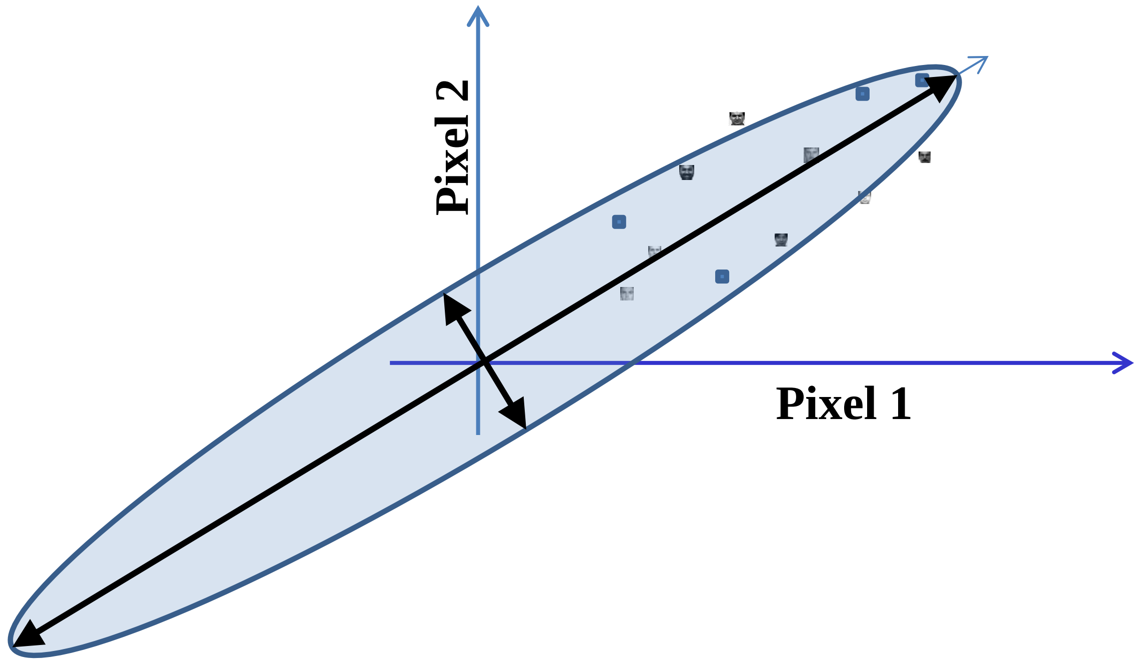
[U, S, V] = SVD(X)

- U will have the Eigenvectors
- Thin SVD for 100 bases:

[U,S,V] = svds(X, 100)

- Much more efficient

Eigenvectors and scatter



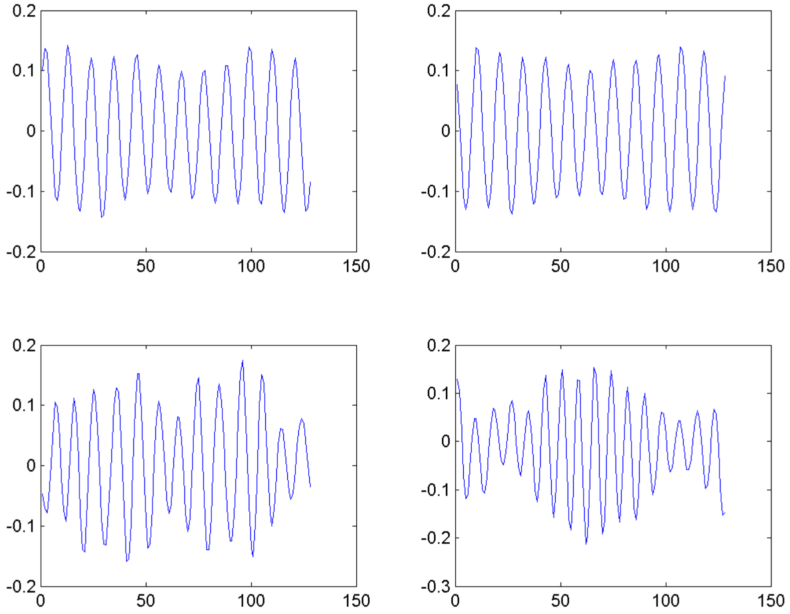
- Turns out: Eigenvectors represent the major and minor axes of an ellipse centered at the origin which encloses the data most compactly

- The SVD of data matrix X uncovers these vectors

What about sound?

- Finding Eigen bases for speech signals:

- Look like DFT/DCT
- Or wavelets



- DFTs are pretty good most of the time

Eigen Analysis

- Can often find surprising features in your data
- Trends, relationships, more
- Commonly used in recommender systems

- An interesting example..

Eigen Analysis

Figure1. Experiment setup @Wean Hall mechanical space. Pipe with arrow indicates a 10" diameter hot water pipe carrying pressurized hot water flow, on which piezoelectric sensors are installed every 10 ft. A National instruments data acquisition system is used to acquire and store the data for later processing.

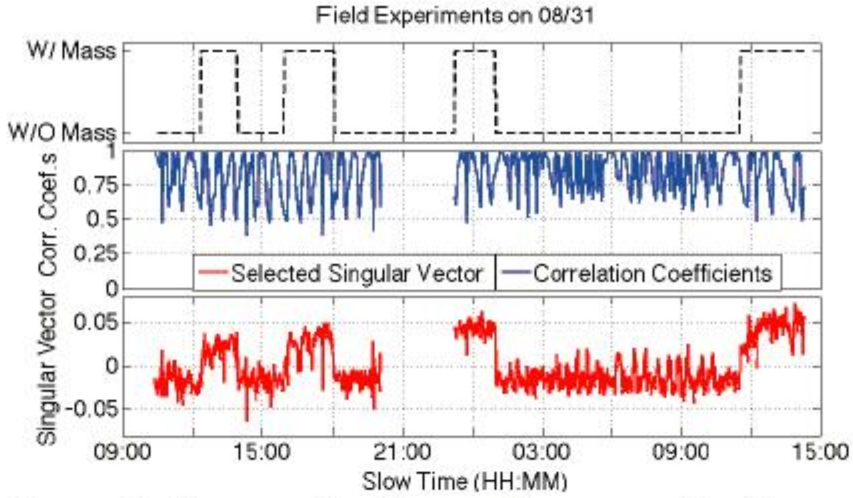


Figure 2. Damage detection results compared with conventional methods. **Top**: Ground truth of whether the pipe is damaged or not. **Middle**: Conventional method only captures temperature variations, and shows no indication of the presence of damage. **Bottom**: The SVD method clearly picks up the steps where damage are introduced and removed.

- Cheng Liu's research on pipes..
- SVD automatically separates useful and uninformative features

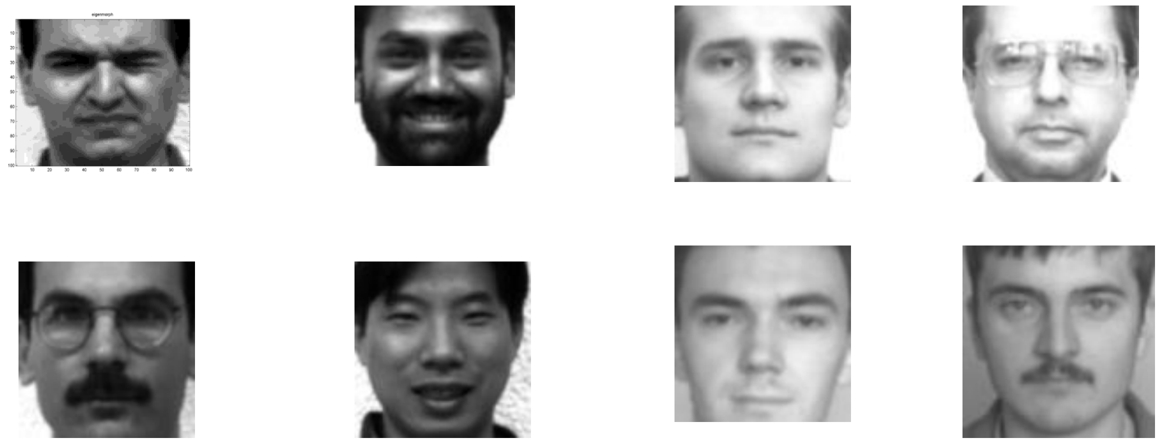
Eigen Analysis

- But for all of this, we need to “preprocess” data

- Eliminate unnecessary aspects
  – E.g. noise, other externally caused variations..

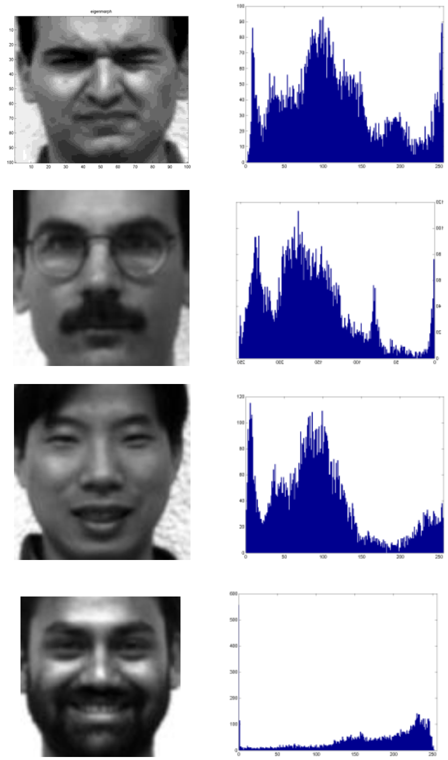
NORMALIZING OUT VARIATIONS

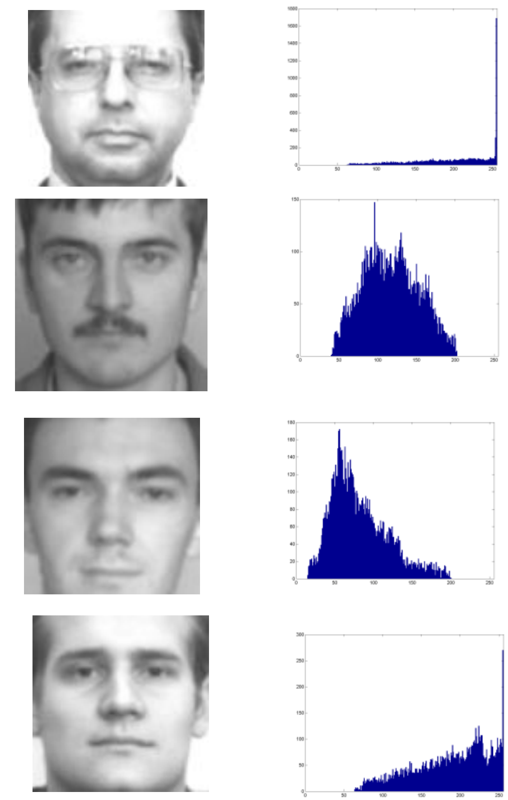
Images: Accounting for variations



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

Images -- Variations





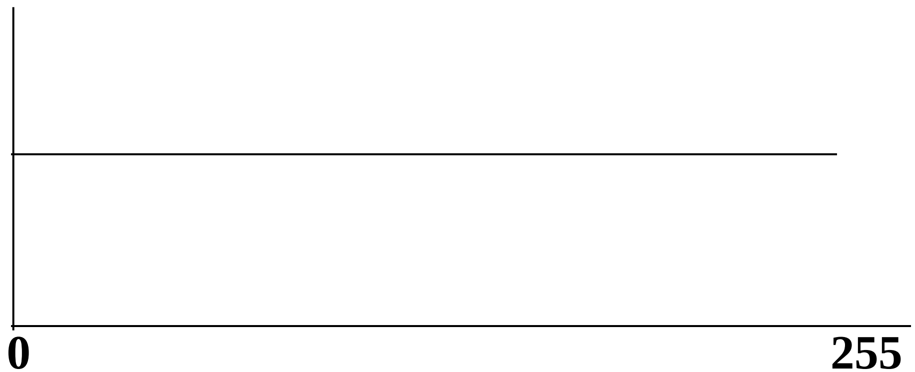
- Pixel histograms: what are the differences

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:

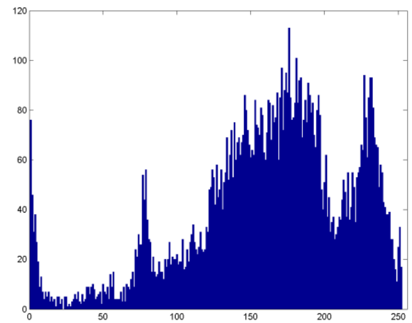
Histogram Equalization

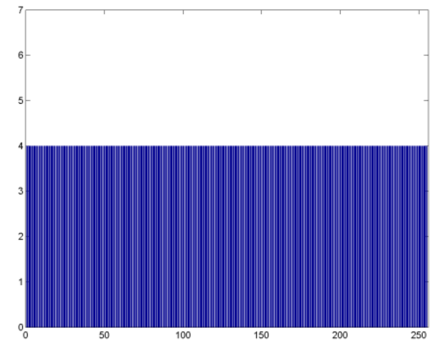
- Normalize histograms of images
  - Maximize the contrast
    - Contrast is defined as the "flatness" of the histogram
    - For maximal contrast, every greyscale must happen as frequently as every other greyscale



- 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

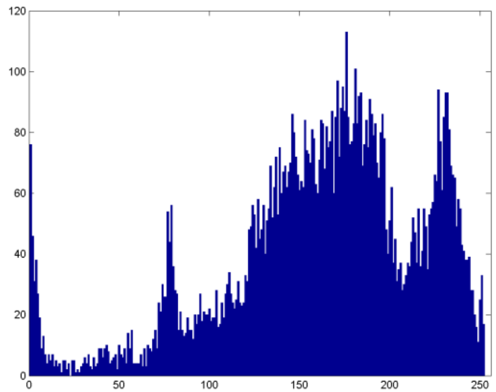
Histogram Equalization

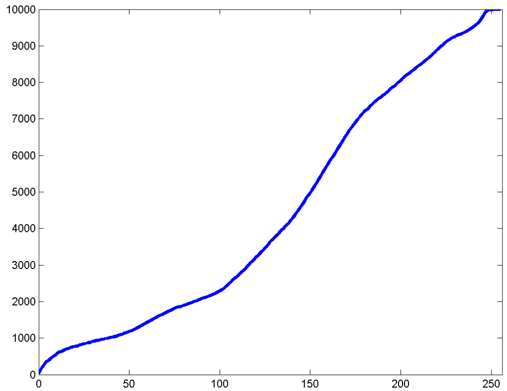




- Modify pixel values such that histogram becomes “flat”.
- For each pixel
  - New pixel value = f(old pixel value)
  - What is f()?
- Easy way to compute this function: map cumulative counts

Cumulative Count Function

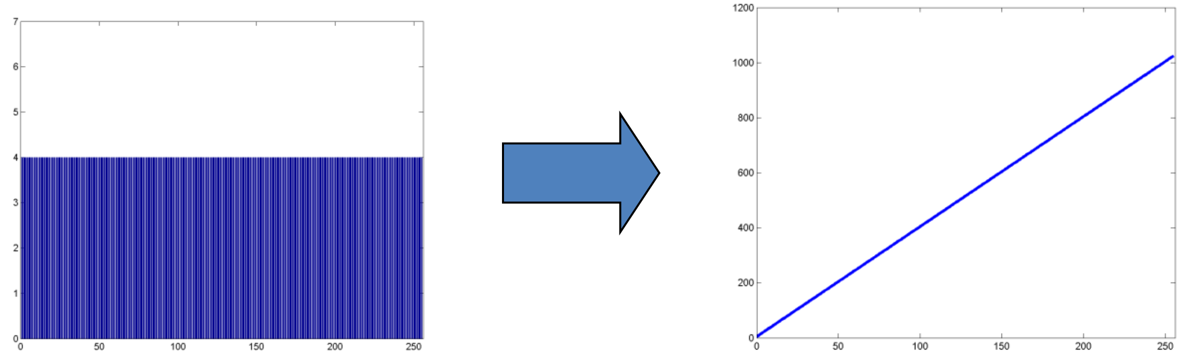




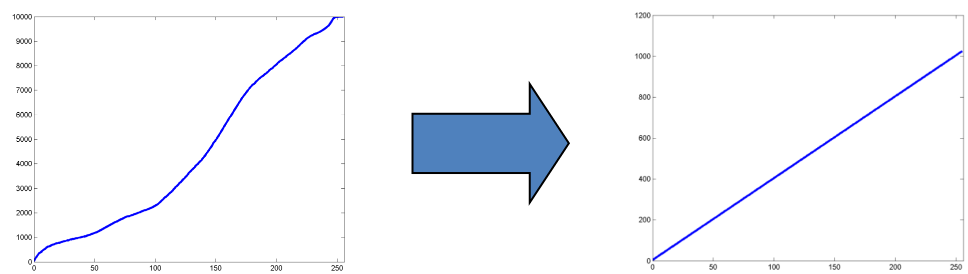
- The *histogram (count)* of a pixel value X is the number of pixels in the image that have value X
  - E.g. in the above image, the count of pixel value 180 is about 110

- 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

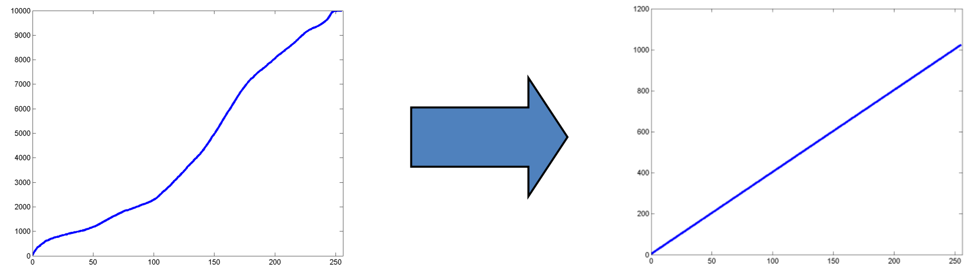


- The cumulative count function of a uniform histogram is a line



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

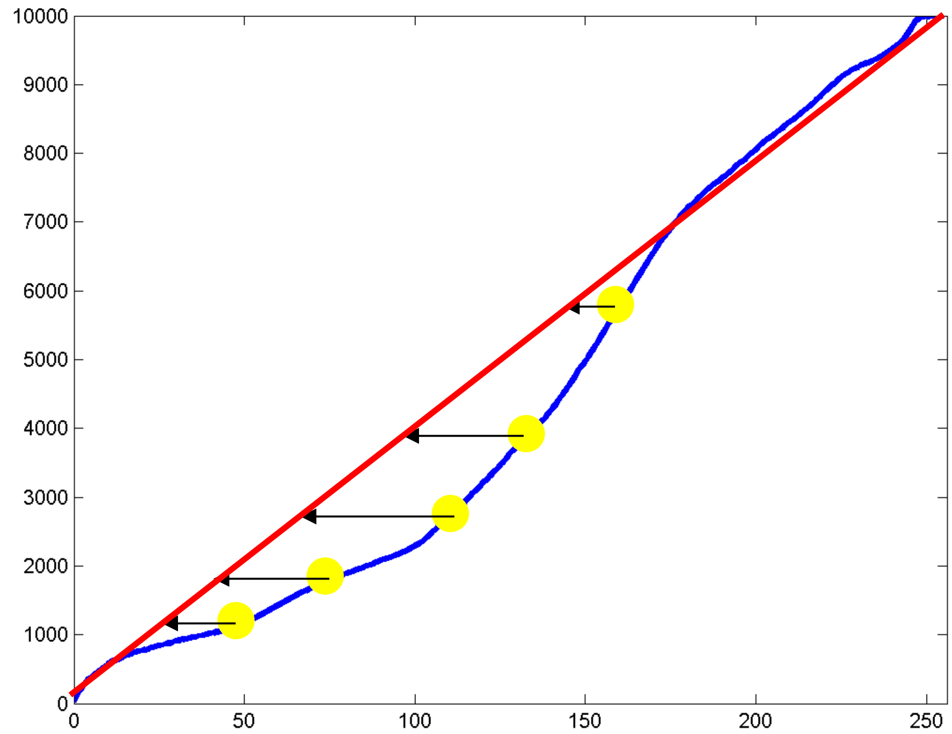
Mapping CCFs



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

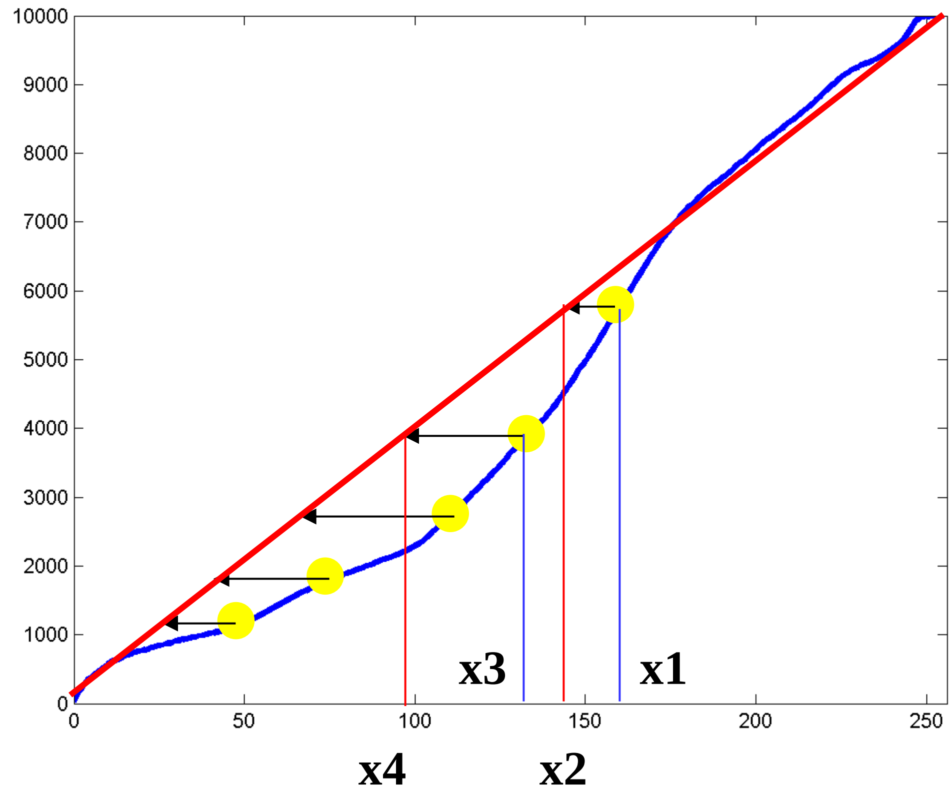
- CCF(f(x)) -> a*f(x) [or a*(f(x)+1) if pixels can take value 0]
  - x = pixel value
  - f() is the function that converts the old pixel value to a new (normalized) pixel value
  - a = (total no. of pixels in image) / (total no. of pixel levels)
    - The no. of pixel levels is 256 in our examples
 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



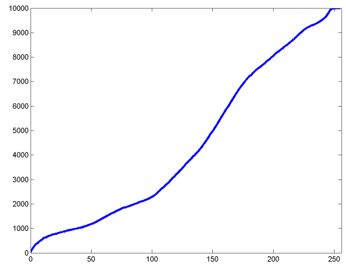
f(x1) = x2

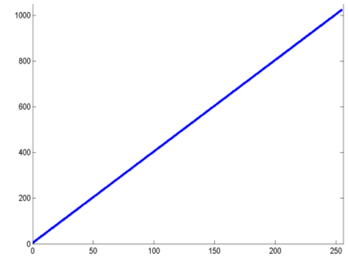
f(x3) = x4

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

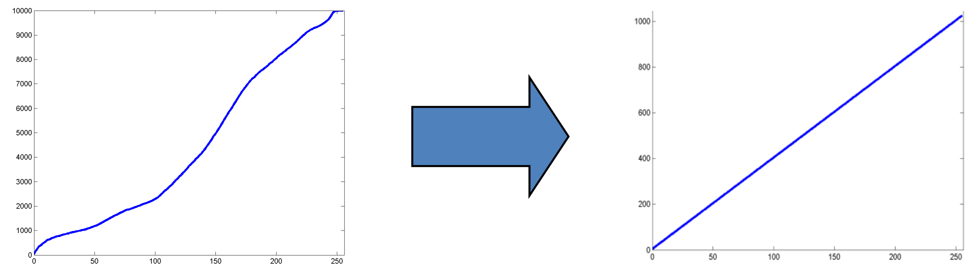




Move x axis levels around until the plot to the left looks like the plot to the right

- 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



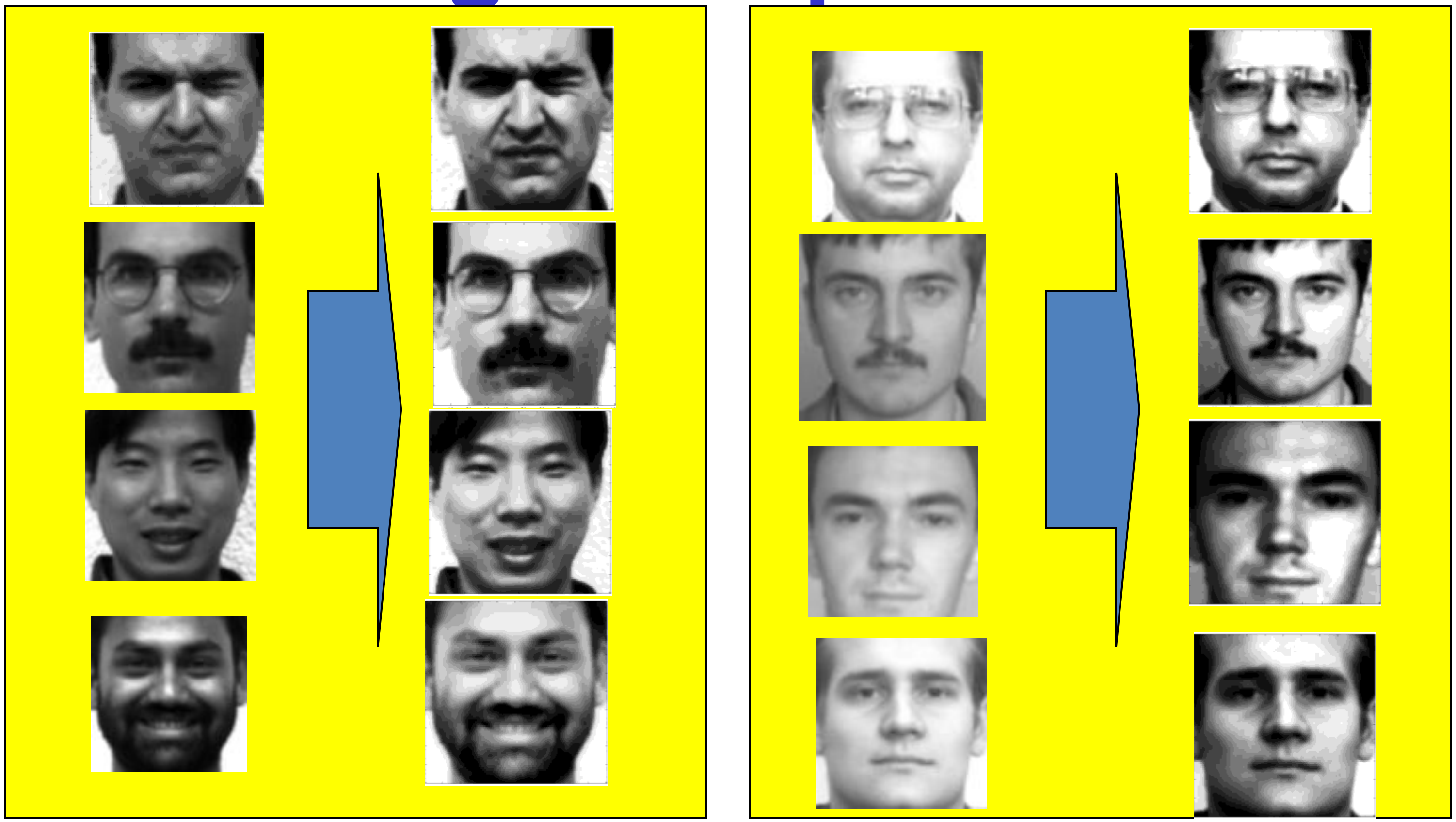
$$f(x) = round\left(\frac{CCF(x) - CCF_{\min}}{Npixels - CCF_{\min}} Max.pixel.value\right)$$

- 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

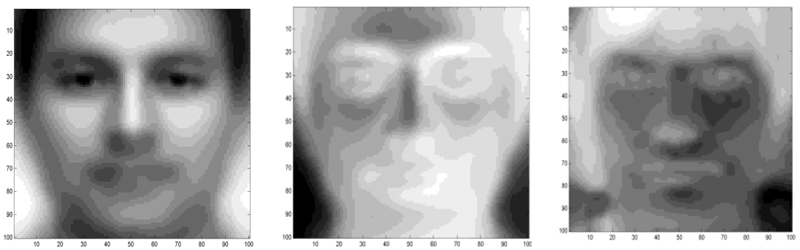
- Matlab:
  - `Newimage = histeq(oldimage)`

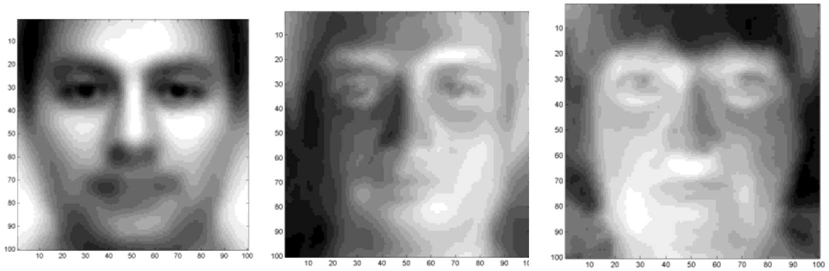
Histogram Equalization



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

Eigenfaces after Equalization

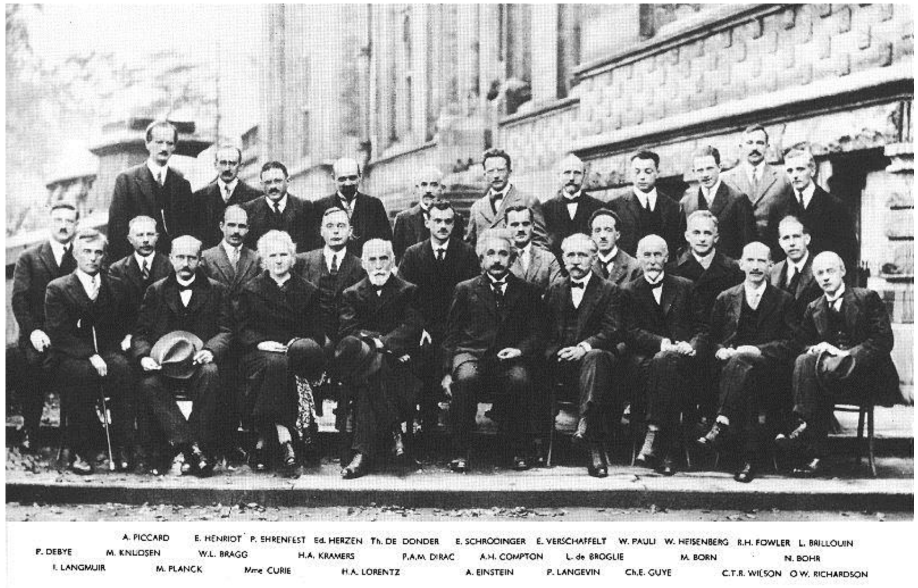




- Left panel : Without HEQ
- Right panel: With HEQ
  - Eigen faces are more face like..
    - Need not always be the case

Detecting Faces in Images

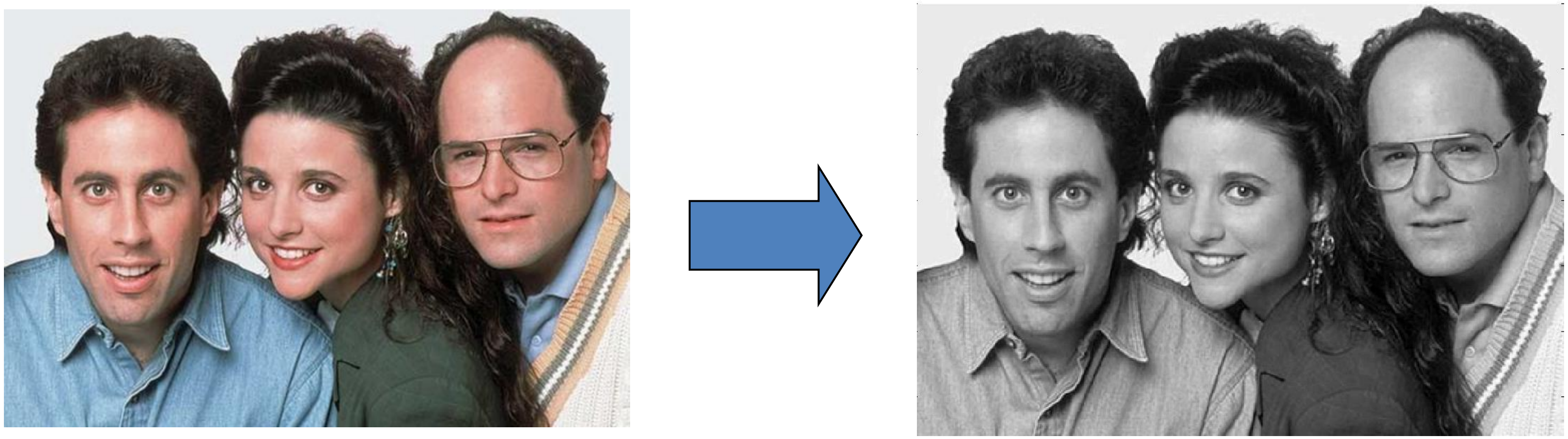
Detecting Faces in Images



- Finding face like patterns
  - How do we find if a picture has faces in it
  - Where are the faces?

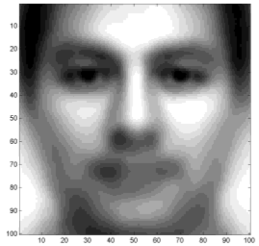
- A simple solution:
  - Define a "typical face"
  - Find the "typical face" in the image

Finding faces in an image



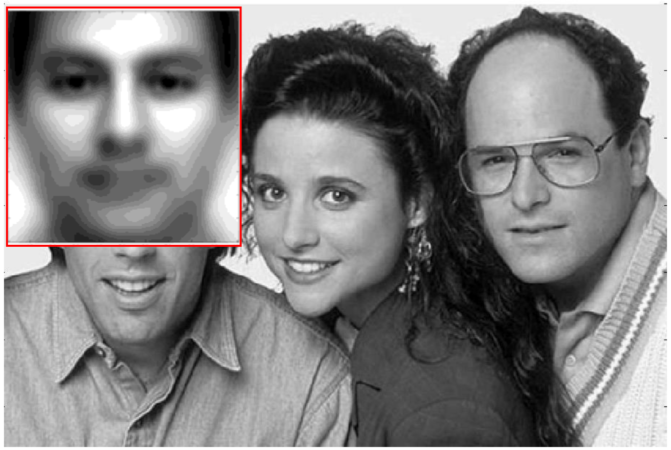
- Picture is larger than the “typical face”
  - E.g. typical face is 100x100, picture is 600x800
- First convert to greyscale
  - R + G + B
  - Not very useful to work in color

Finding faces in an image



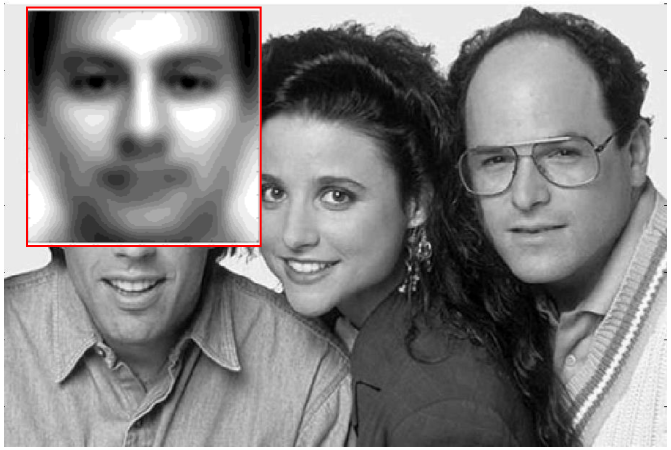
- Goal .. To find out if and where images that look like the “typical” face occur in the picture

Finding faces in an image



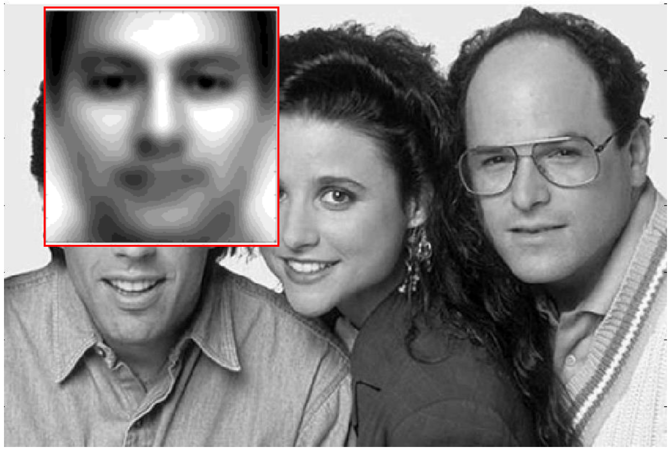
- Try to “match” the typical face to each location in the picture

Finding faces in an image



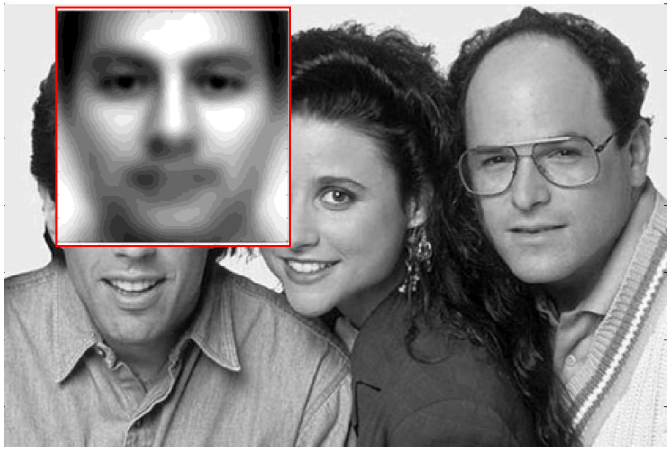
- Try to “match” the typical face to each location in the picture

Finding faces in an image



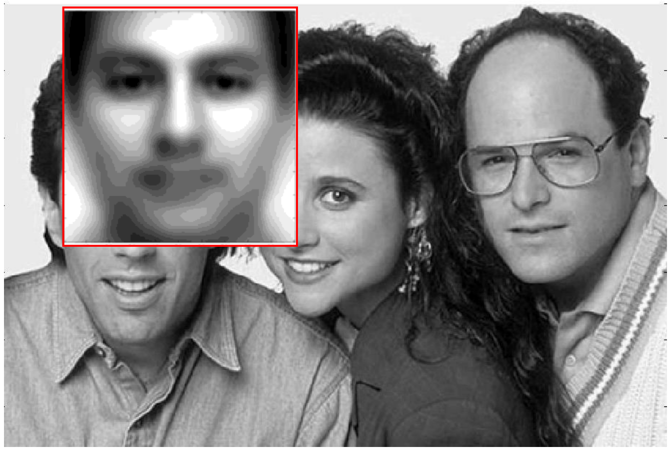
- Try to "match" the typical face to each location in the picture

Finding faces in an image



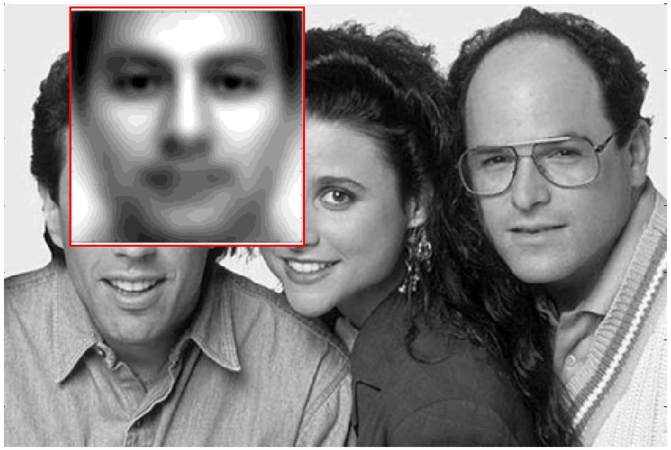
- Try to "match" the typical face to each location in the picture

Finding faces in an image



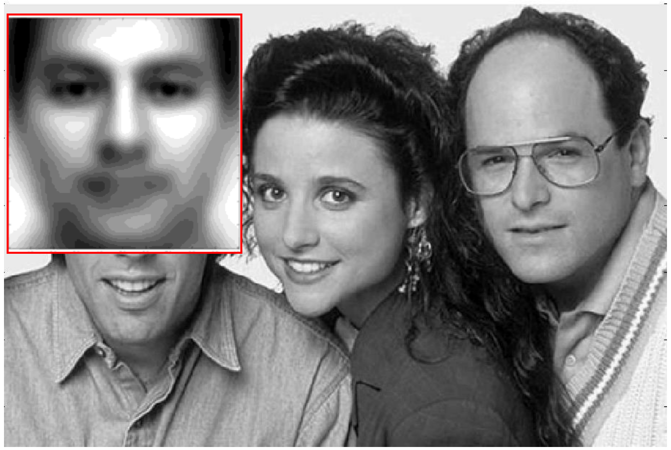
- Try to “match” the typical face to each location in the picture

Finding faces in an image



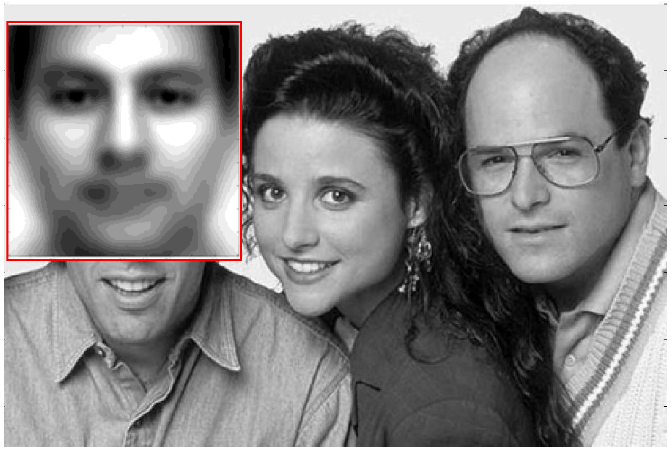
- Try to “match” the typical face to each location in the picture

Finding faces in an image



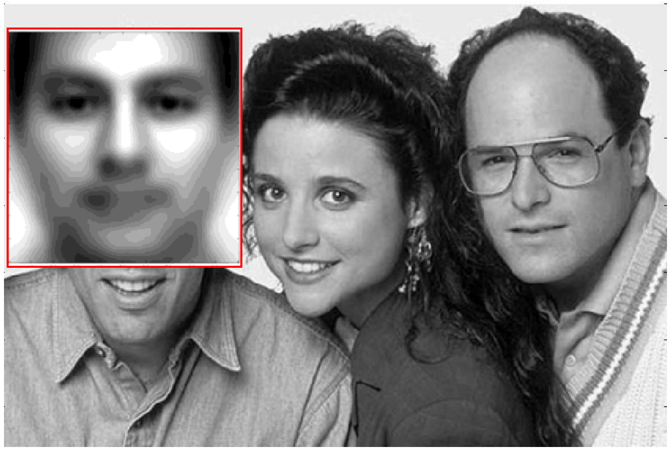
- Try to "match" the typical face to each location in the picture

Finding faces in an image



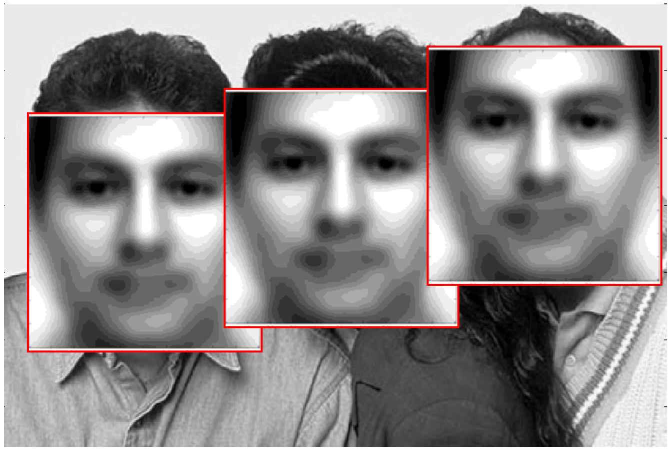
- Try to “match” the typical face to each location in the picture

Finding faces in an image



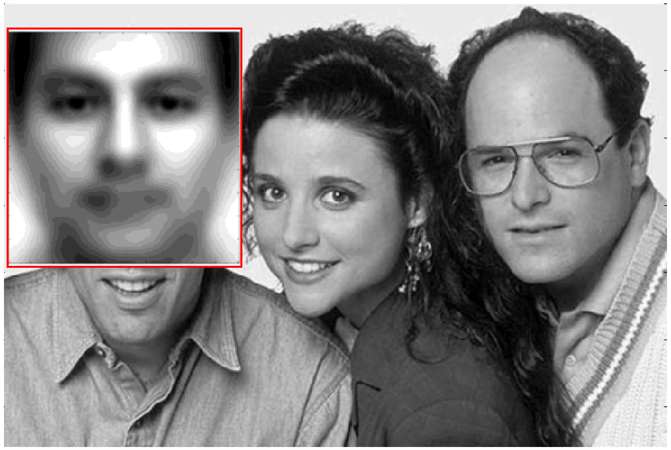
- Try to “match” the typical face to each location in the picture

Finding faces in an image



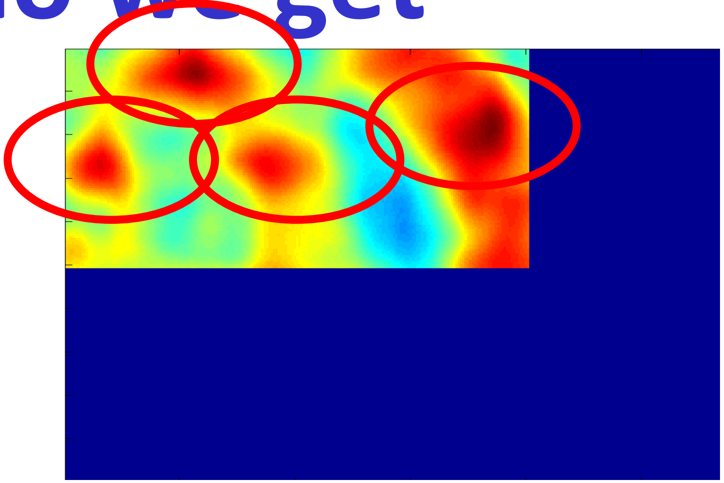
- Try to “match” the typical face to each location in the picture
- The “typical face” will explain some spots on the image much better than others
  - These are the spots at which we probably have a face!

How to "match"



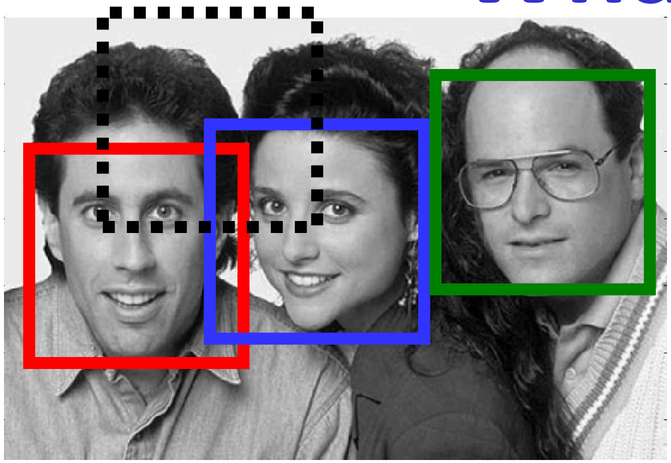
- What exactly is the "match"
  - What is the match "score"
- The DOT Product
  - Express the typical face as a vector
  - Express the region of the image being evaluated as a vector
    - But first histogram equalize the region
      - Just the section being evaluated, without considering the rest of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

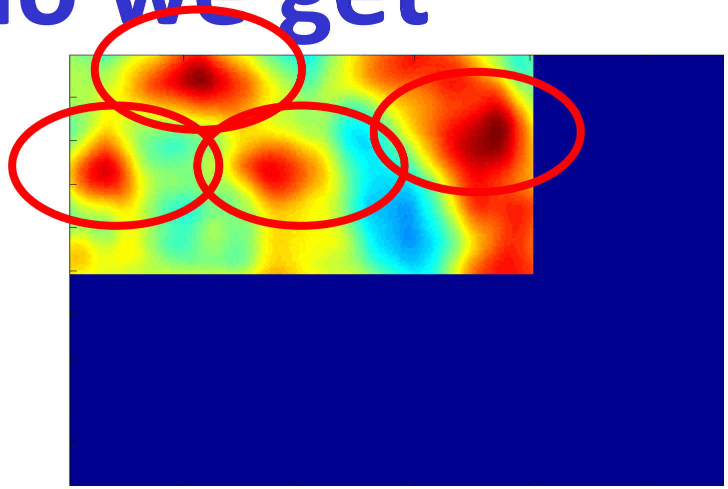
What do we get



- 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

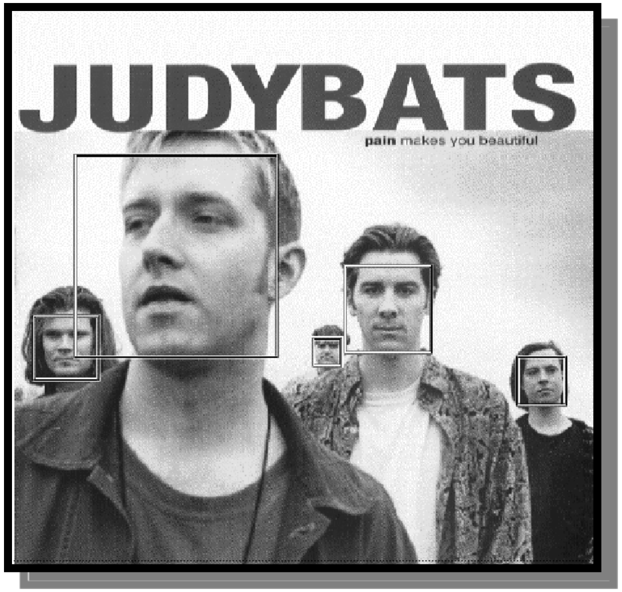




- The right panel shows the dot product a various loctions
  - Redder is higher
    - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
  - Likes George's face most
    - He looks most like the typical face
- Also finds a face where there is none!
  - A false alarm

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

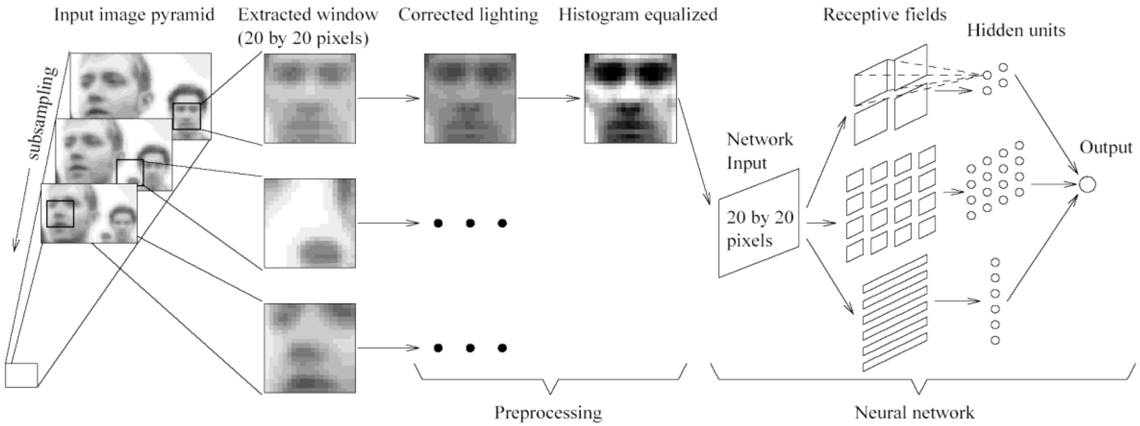


Figure 1: The basic algorithm used for face detection.

- 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
- Other classifiers

- later in the program..