

# Computer vision: models, learning and inference

Chapter 13

Image preprocessing and feature  
extraction

# Preprocessing

- The goal of pre-processing is
  - to try to reduce unwanted variation in image due to lighting, scale, deformation etc.
  - to reduce data to a manageable size
- Give the subsequent model a chance
- Preprocessing definition: deterministic transformation of pixels  $p$  to create data vector  $x$
- Usually heuristics based on experience

# Structure

- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

# Normalization

- Fix first and second moments to standard values
- Remove contrast and constant additive luminance variations

Before



After

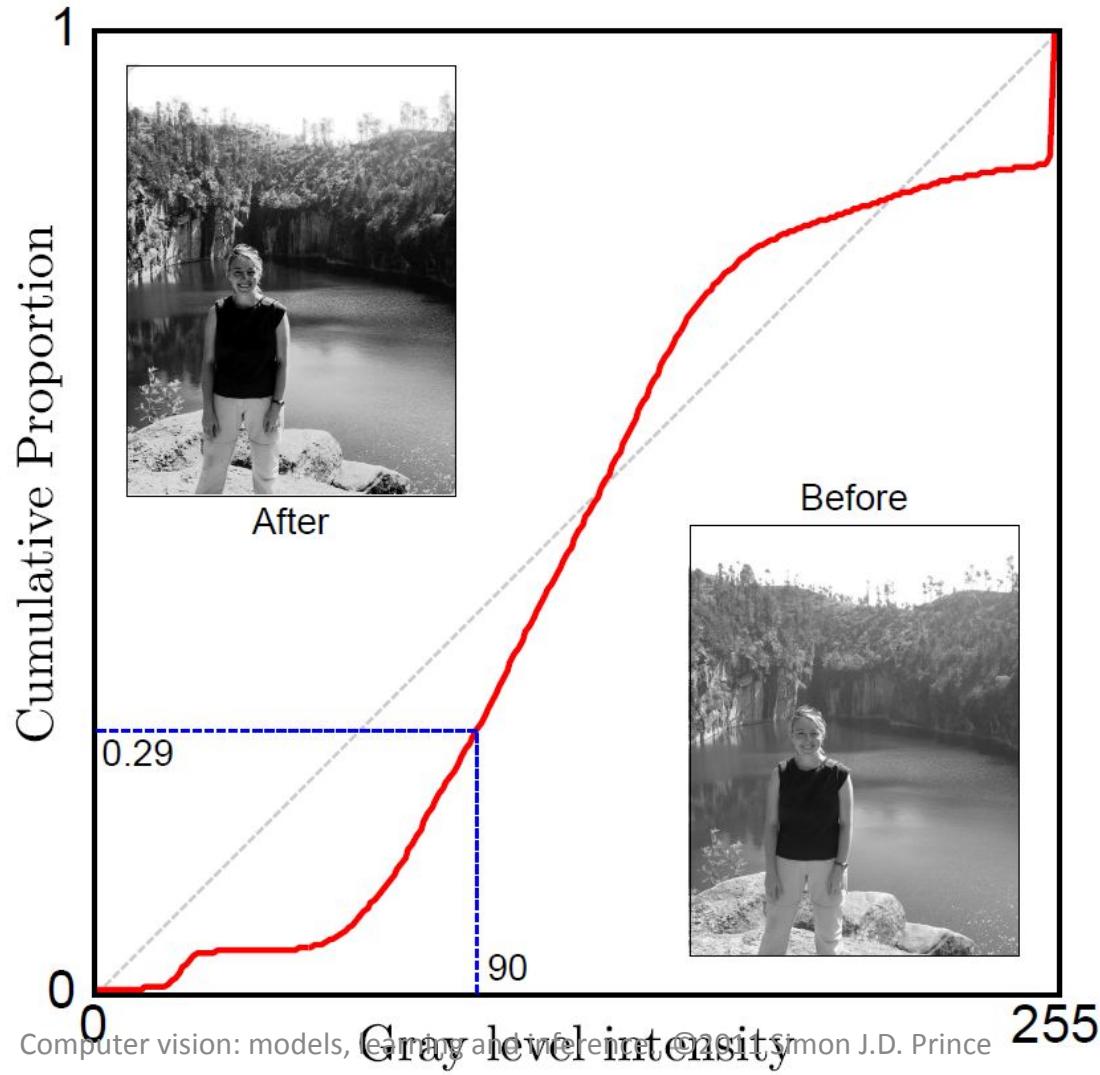
# Histogram Equalization

Make all of the moments the same by forcing  
the histogram of intensities to be the same



Before/ normalized/ Histogram Equalized

# Histogram Equalization



# Convolution

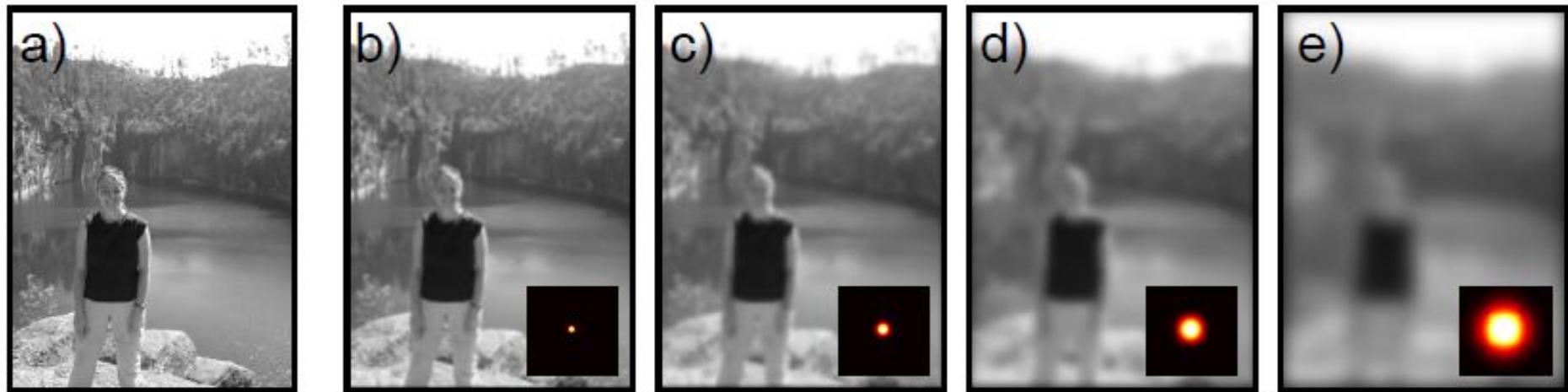
Takes pixel image  $\mathbf{P}$  and applies a filter  $\mathbf{F}$

$$x_{ij} = \sum_{m=-M}^M \sum_{n=-N}^N p_{i-m,j-n} f_{m,n}$$

Computes weighted sum of pixel values, where weights given by filter.

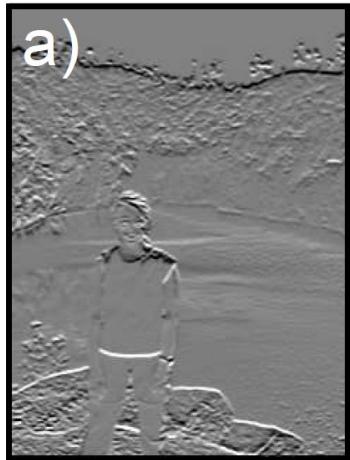
Easiest to see with a concrete example

# Blurring (convolve with Gaussian)

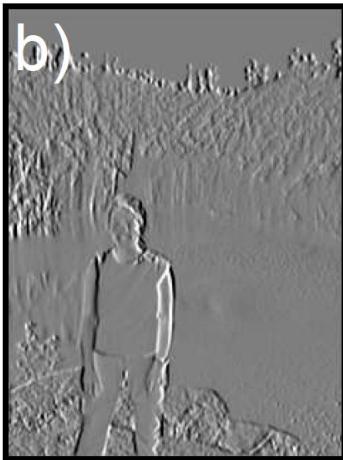


**Figure B.3** Image blurring. a) Original image. b) Result of convolving with a Gaussian filter (filter shown in bottom right of image). The image is slightly blurred. c-e) Convolving with a filter of increasing standard deviation causes the resulting image to be increasingly blurred.

# Gradient Filters



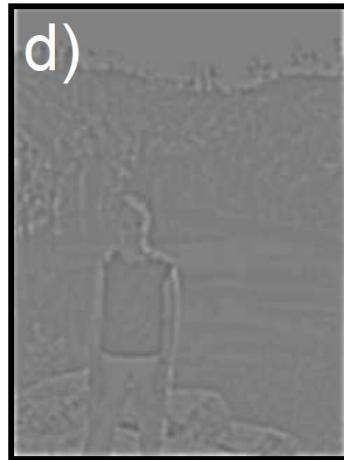
a) Prewitt (vertical)



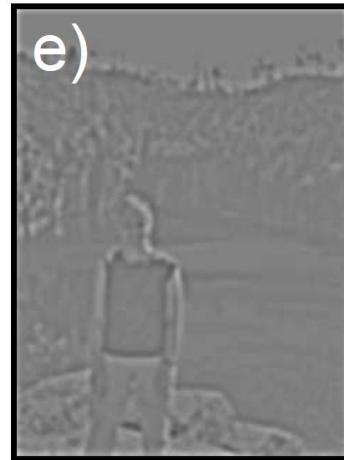
b) Prewitt (horizontal)



c) Laplacian



d) Laplacian of Gaussian

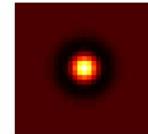
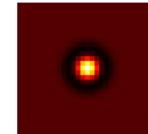


e) Difference of Gaussians

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

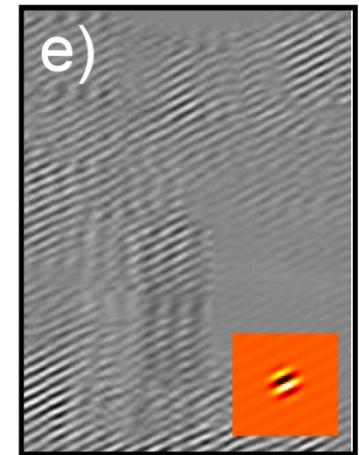
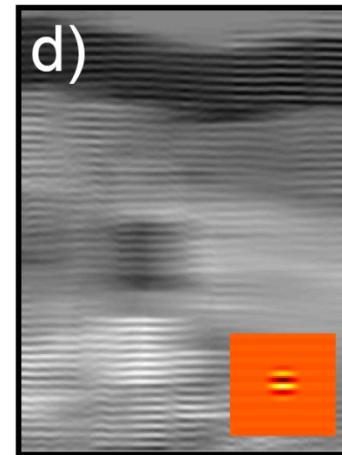
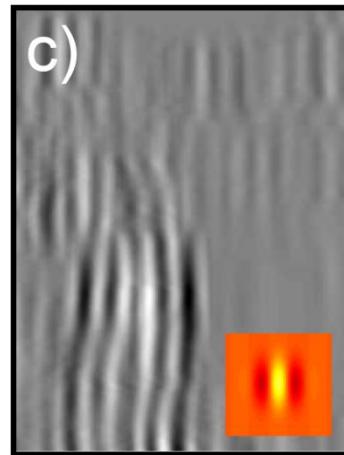
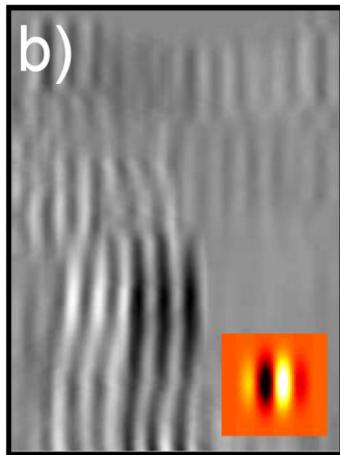
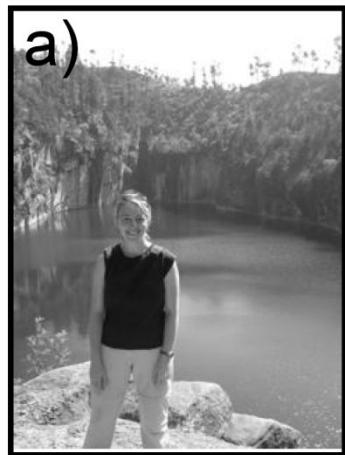
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



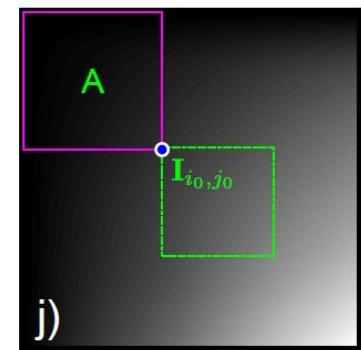
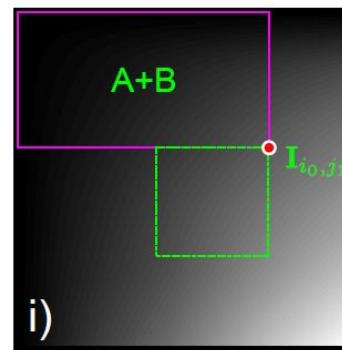
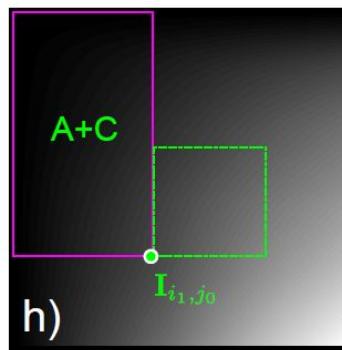
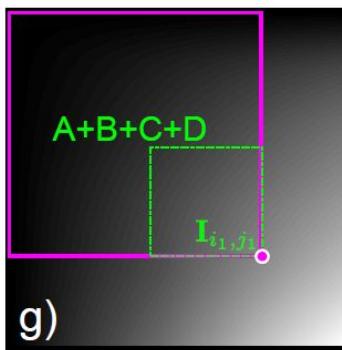
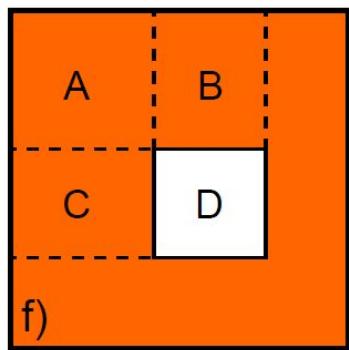
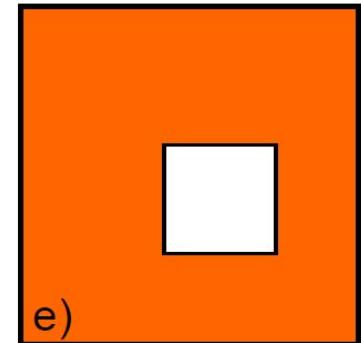
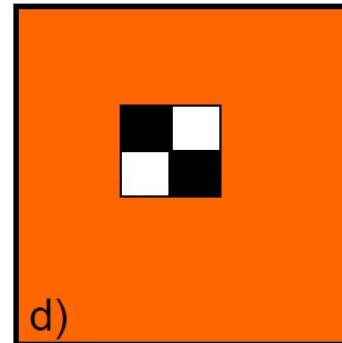
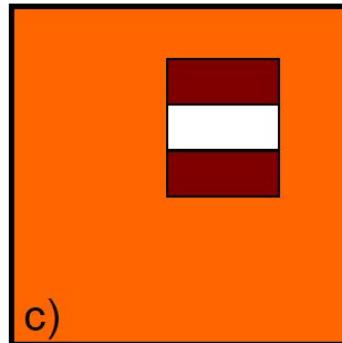
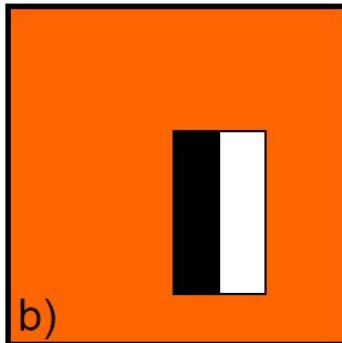
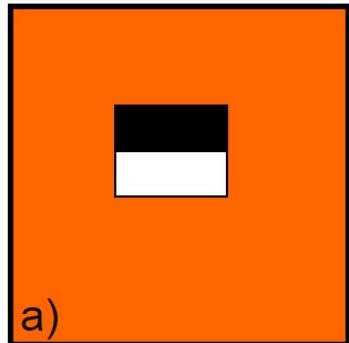
- Rule of thumb: big response when image matches filter

# Gabor Filters



$$f_{mn} = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{m^2 + n^2}{2\sigma^2}\right] \sin\left[\frac{2\pi(\cos[\omega]m + \sin[\omega]n)}{\lambda} + \phi\right]$$

# Haar Filters

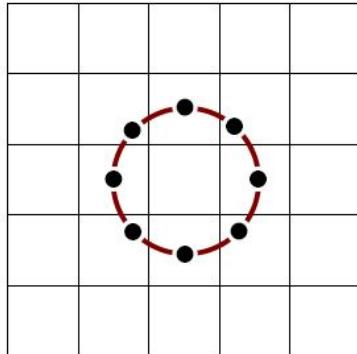


# Local binary patterns

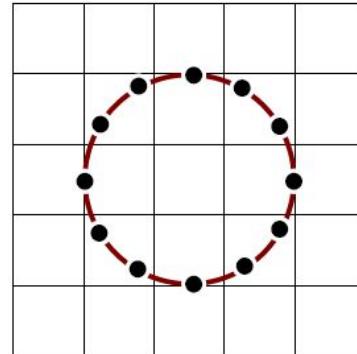
5	4	3
4	3	1
2	0	3

0	1	2	1
7	1	1	0
6	0	5	4

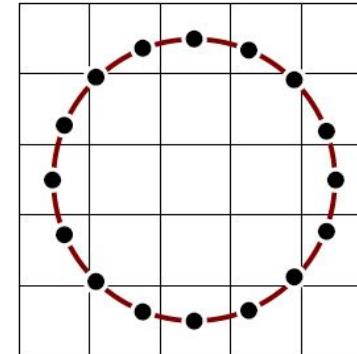
$$\text{LBP} = 10010111 = 151$$



$$(P = 8, R = 1.0)$$



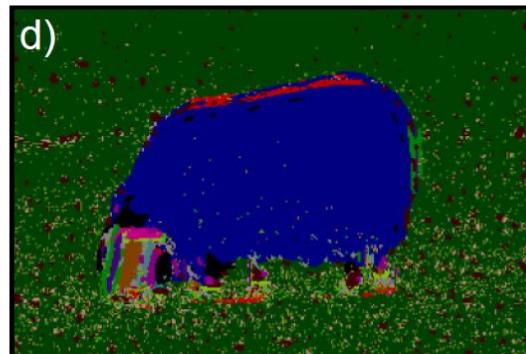
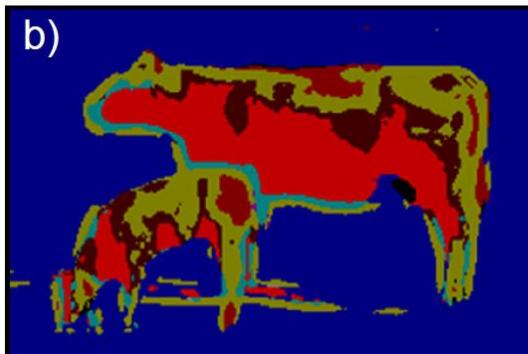
$$(P = 12, R = 1.5)$$



$$(P = 16, R = 2.0)$$

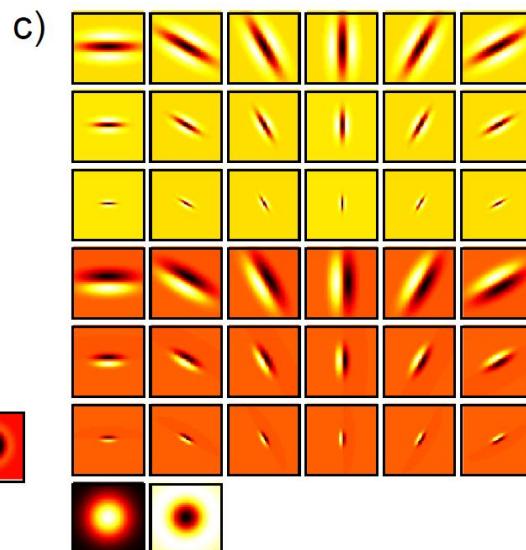
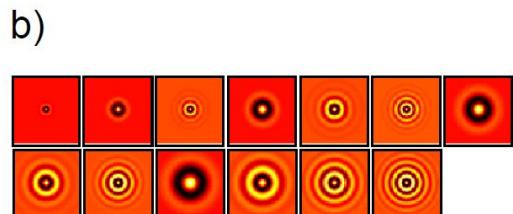
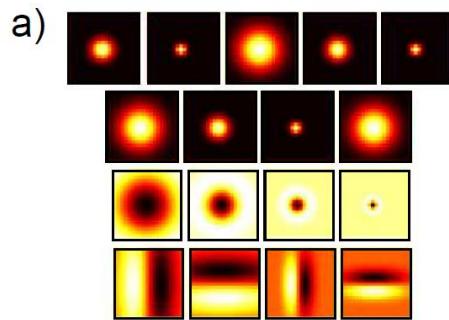
# Textons

- An attempt to characterize texture
- Replace each pixel with integer representing the texture ‘type’

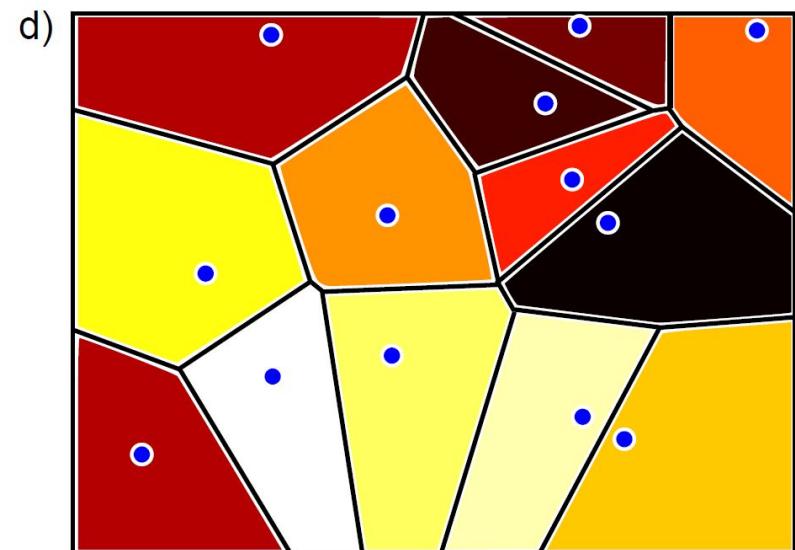


# Computing Textons

Take a bank of filters and apply  
to lots of images



Cluster in filter space

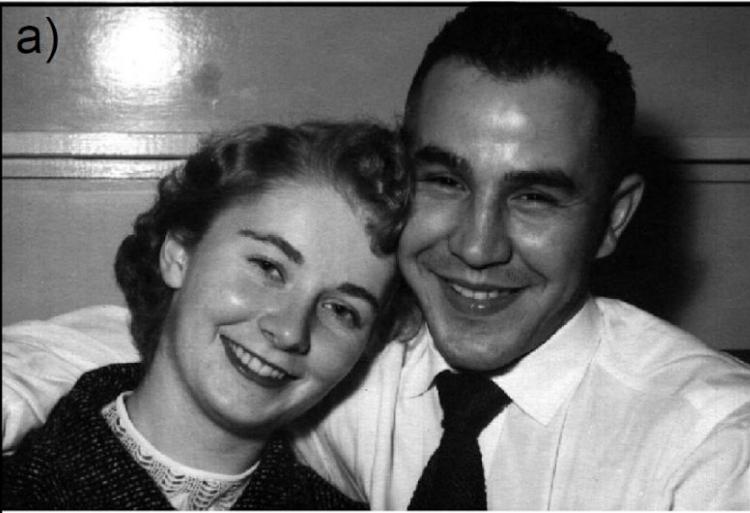


For new pixel, filter surrounding region with same bank,  
and assign to nearest cluster

# Structure

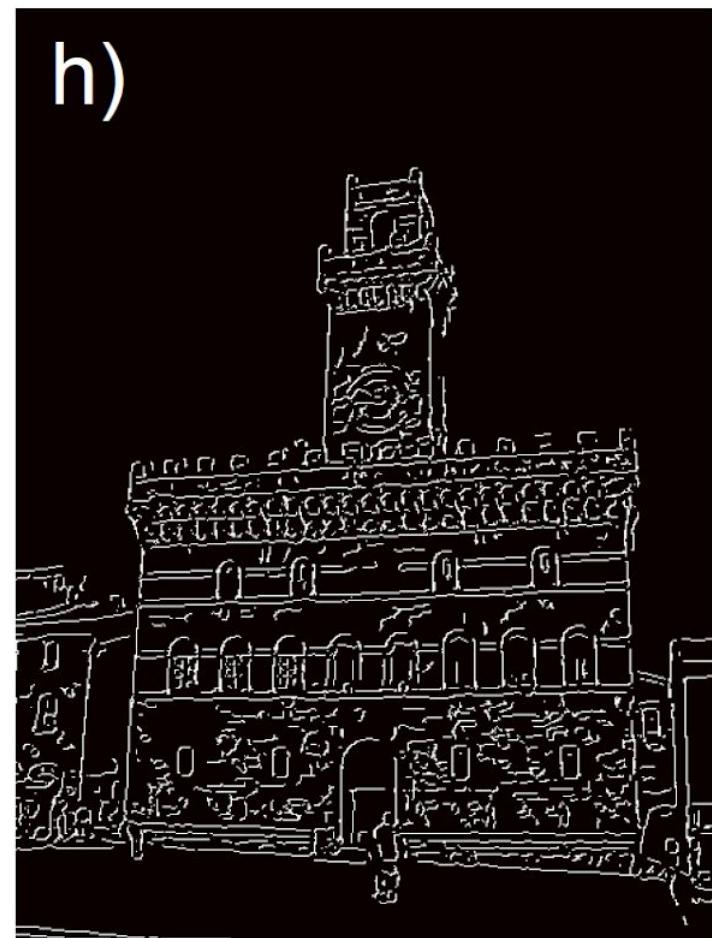
- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

# Edges

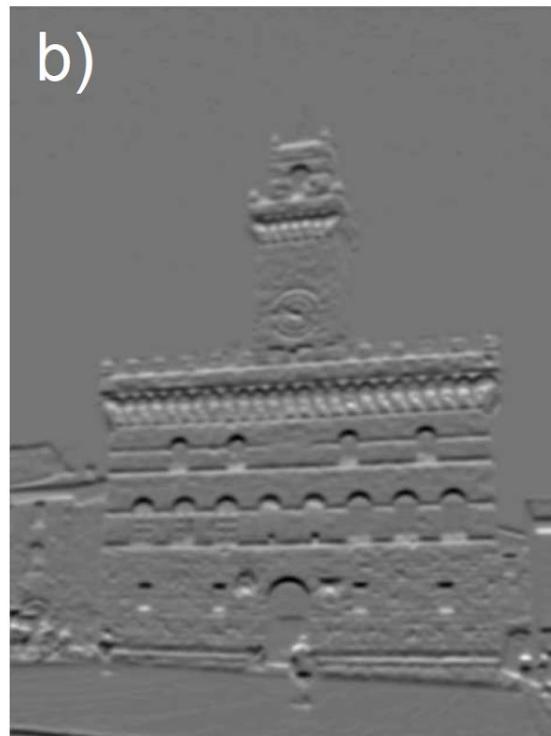


(from Elder and  
Goldberg 2000)

# Canny Edge Detector

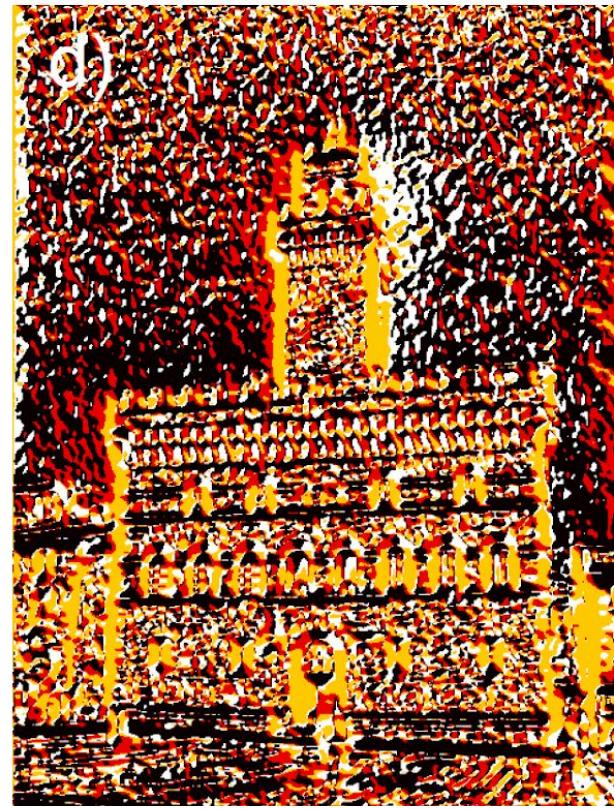


# Canny Edge Detector



Compute horizontal and vertical gradient images  $\mathbf{h}$  and  $\mathbf{v}$

# Canny Edge Detector

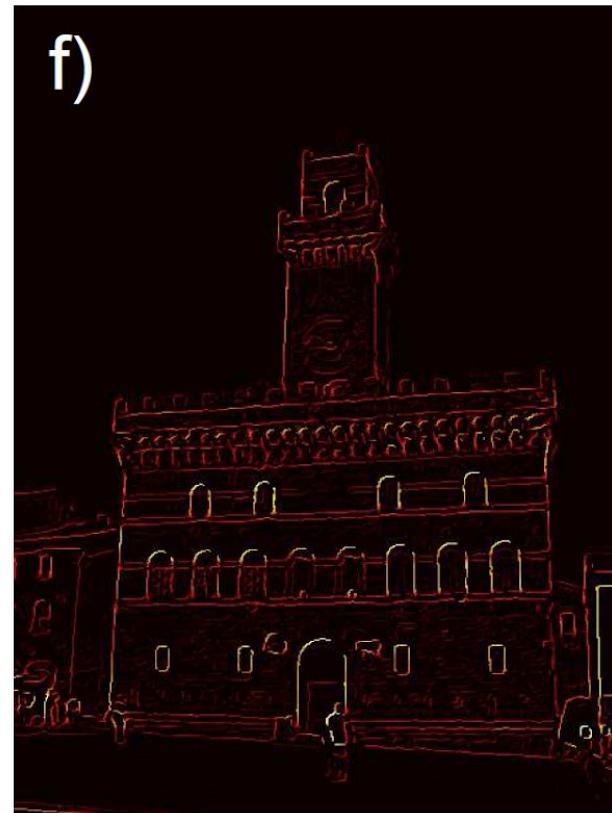
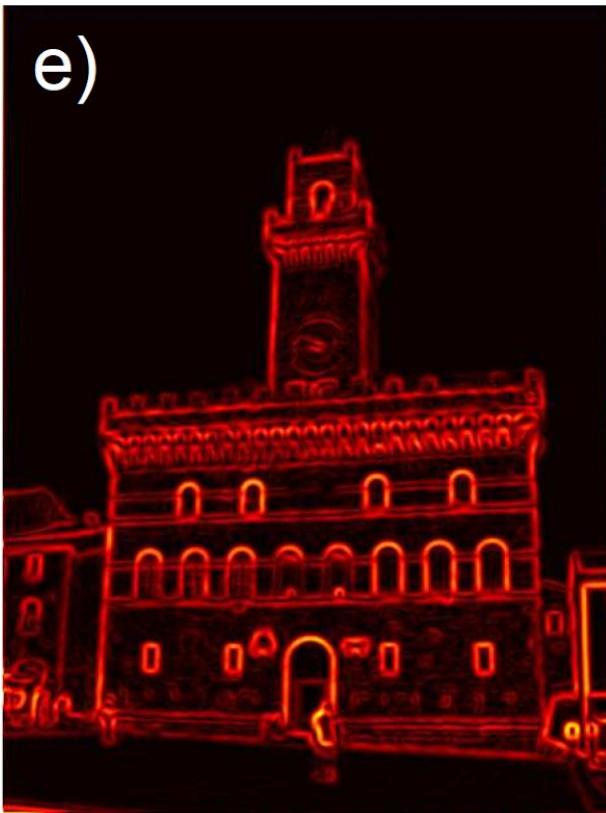


$$a_{ij} = \sqrt{h_{ij}^2 + v_{ij}^2}$$

$$\theta_{ij} = \arctan[v_{ij}/h_{ij}]$$

Quantize to 4 directions

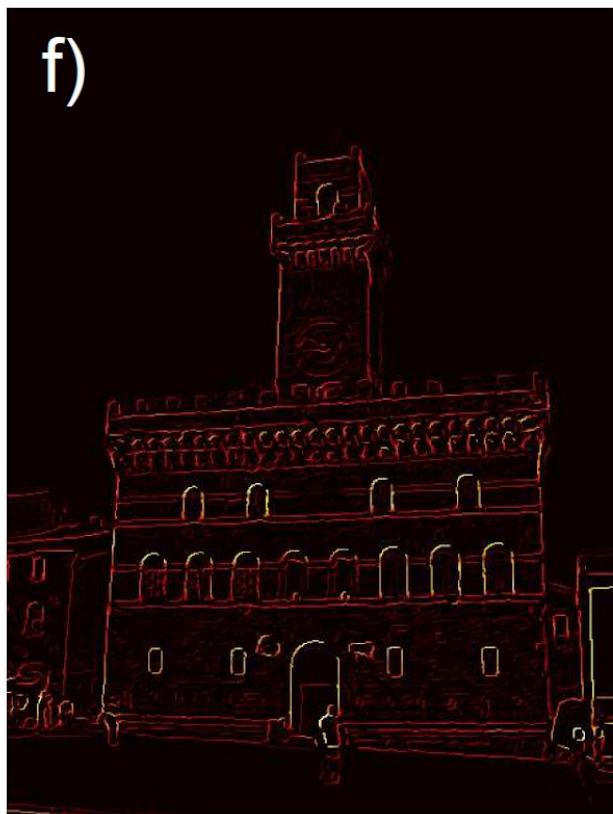
# Canny Edge Detector



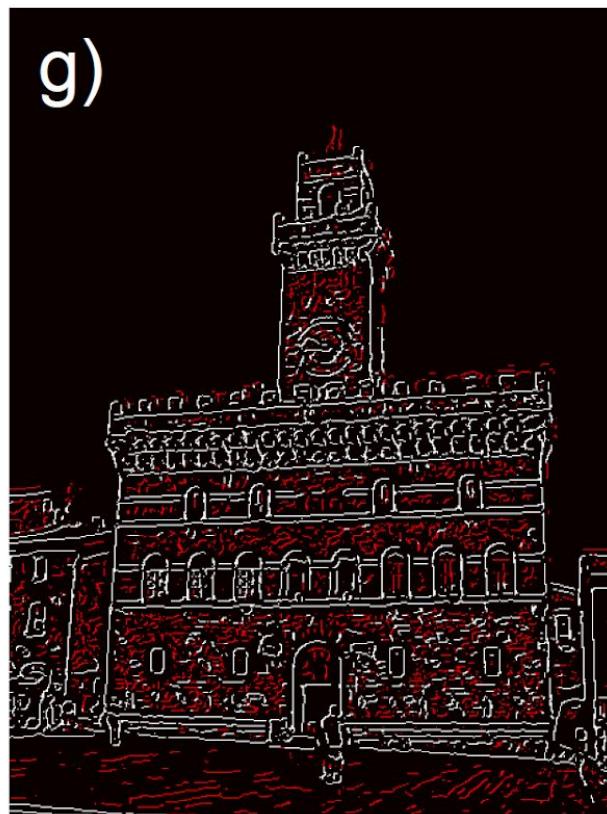
Non-maximal suppression

# Canny Edge Detector

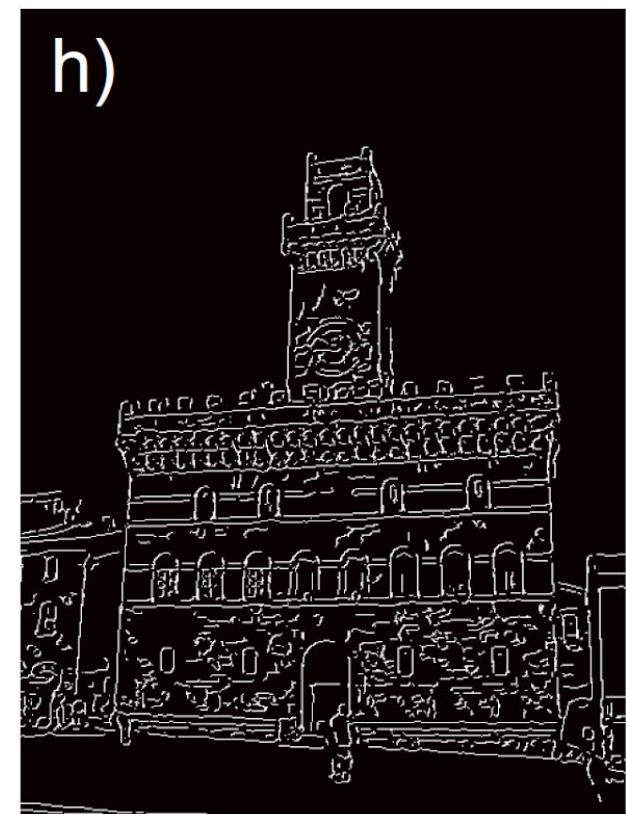
f)



g)

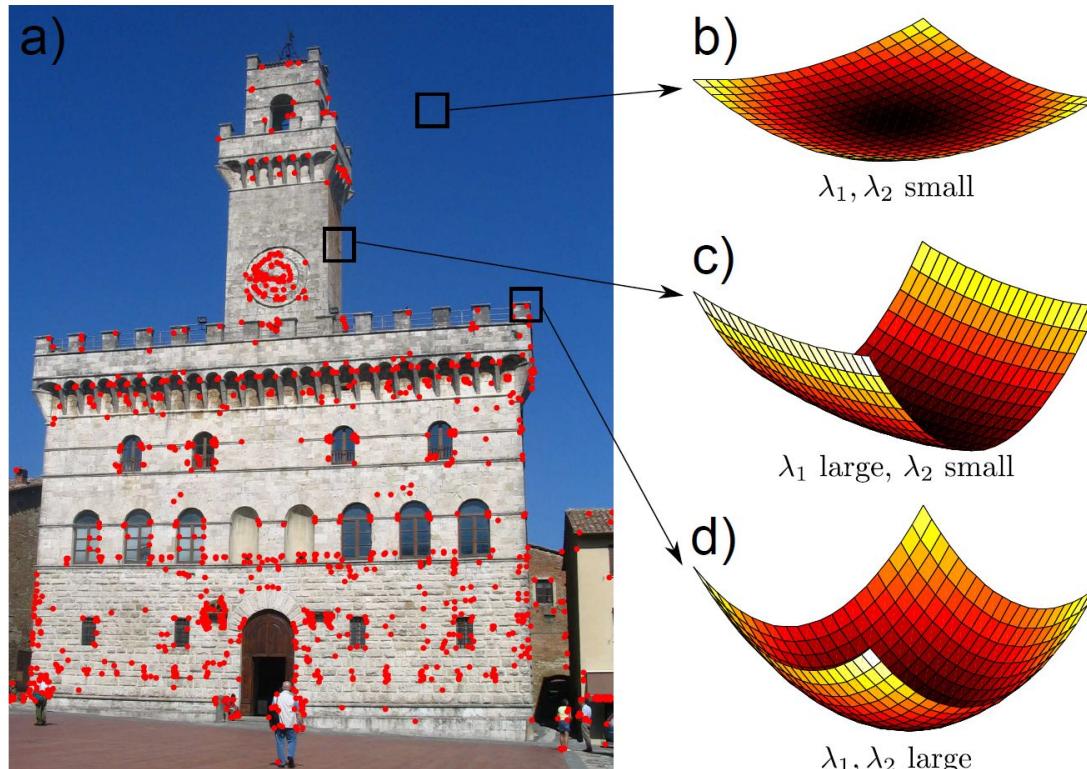


h)



## Hysteresis Thresholding

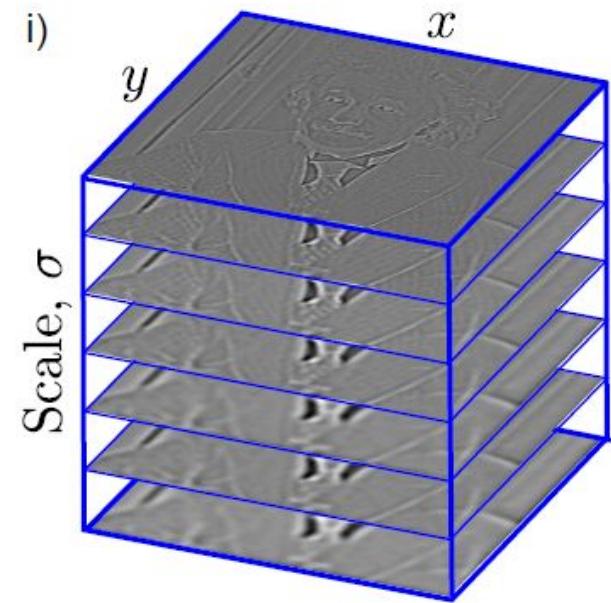
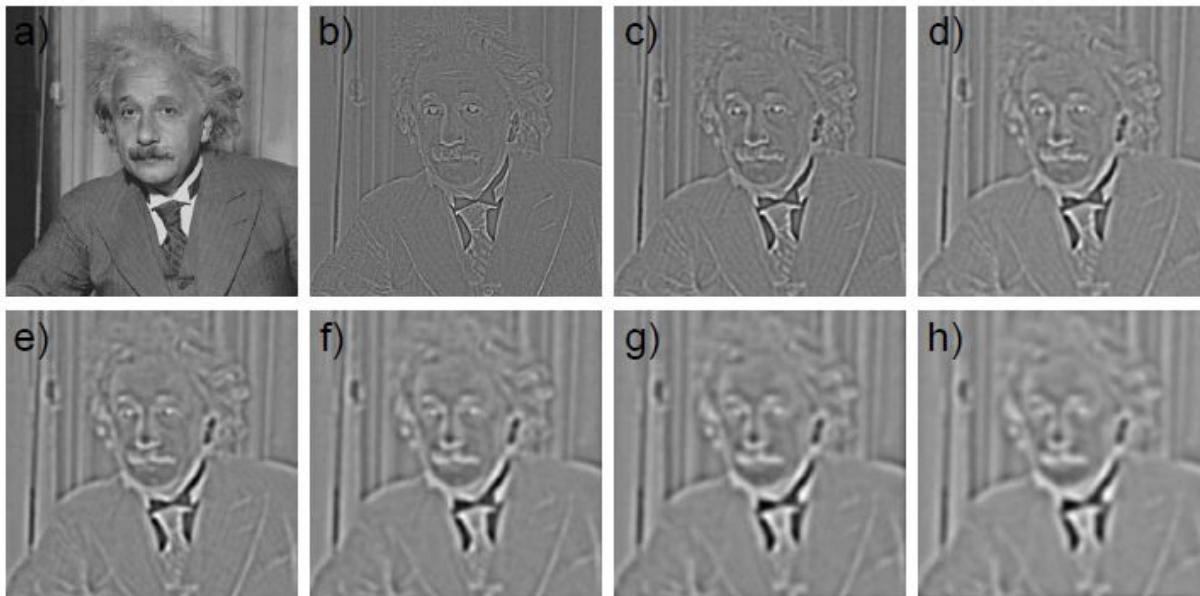
# Harris Corner Detector



Make decision based on  
image structure tensor

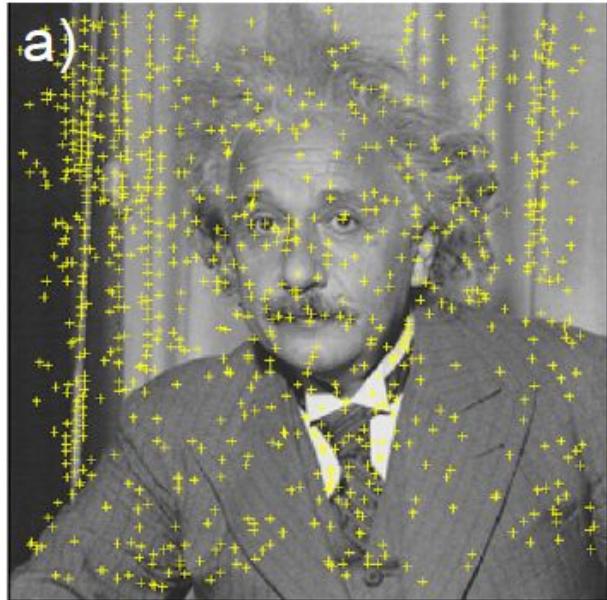
$$\mathbf{S}_{ij} = \sum_{m=i-D}^{i+D} \sum_{n=j-D}^{j+D} w_{mn} \begin{bmatrix} h_{ij}^2 & h_{ij}v_{ij} \\ h_{ij}v_{ij} & v_{ij}^2 \end{bmatrix}$$

# SIFT Detector

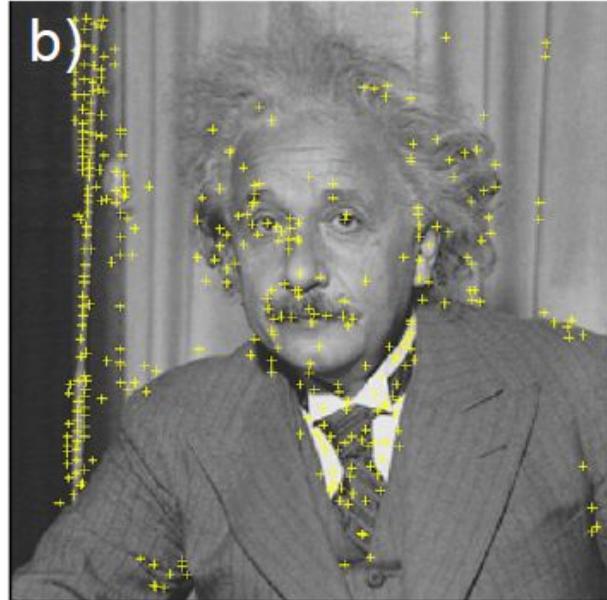


Filter with difference of Gaussian filters at increasing scales  
Build image stack (scale space)  
Find extrema in this 3D volume

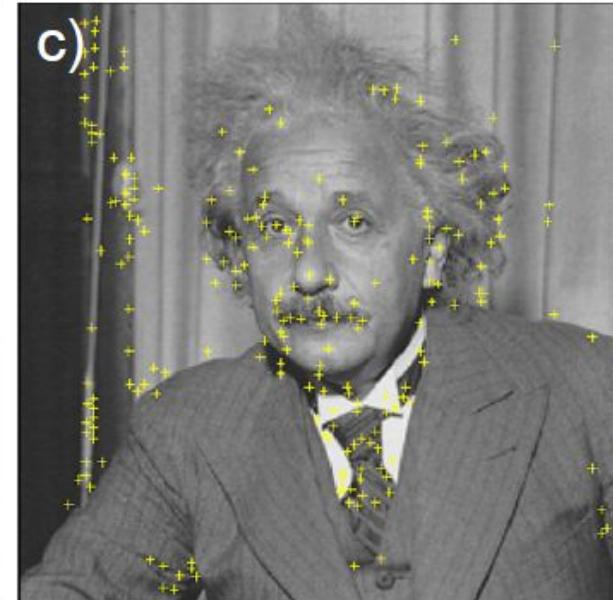
# SIFT Detector



a)



b)



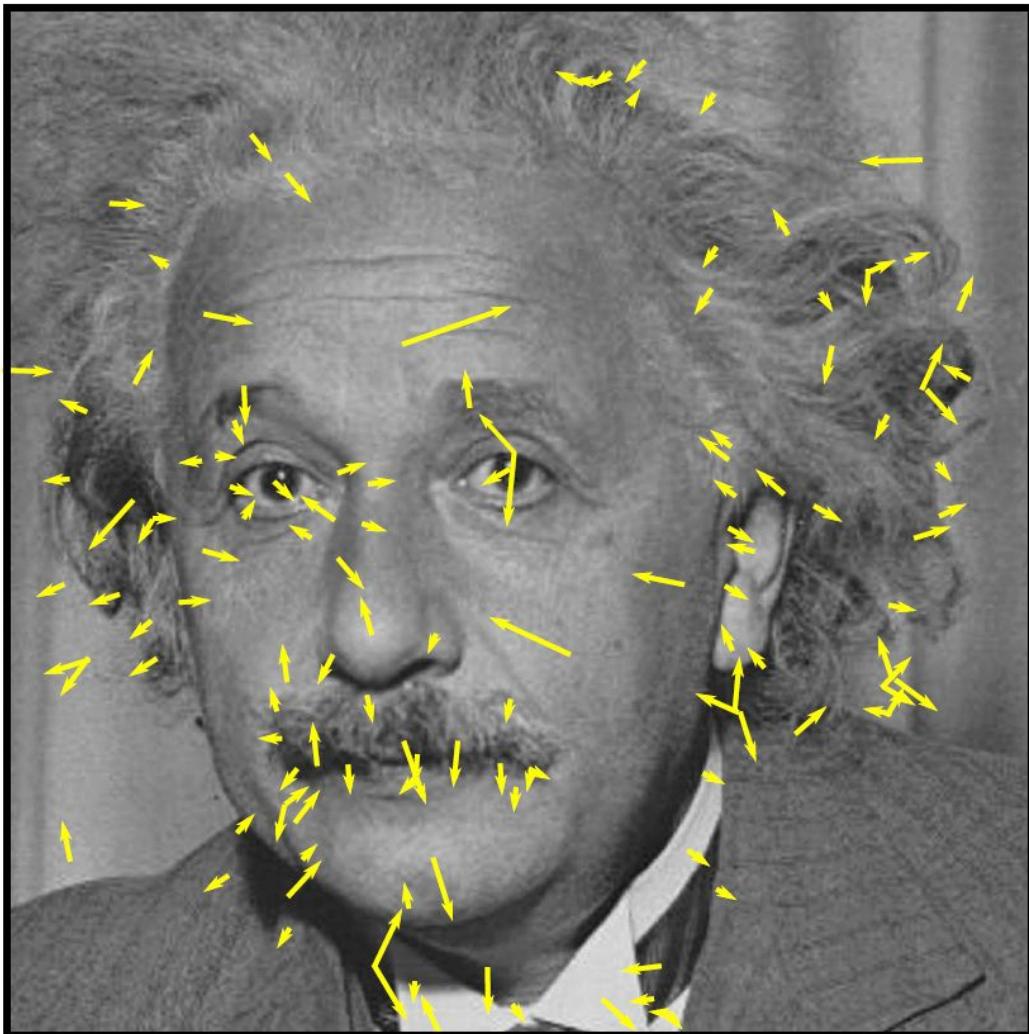
c)

Identified Corners

Remove those on  
edges

Remove those  
where contrast is  
low

# Assign Orientation



Orientation assigned by looking at intensity gradients in region around point

Form a histogram of these gradients by binning.

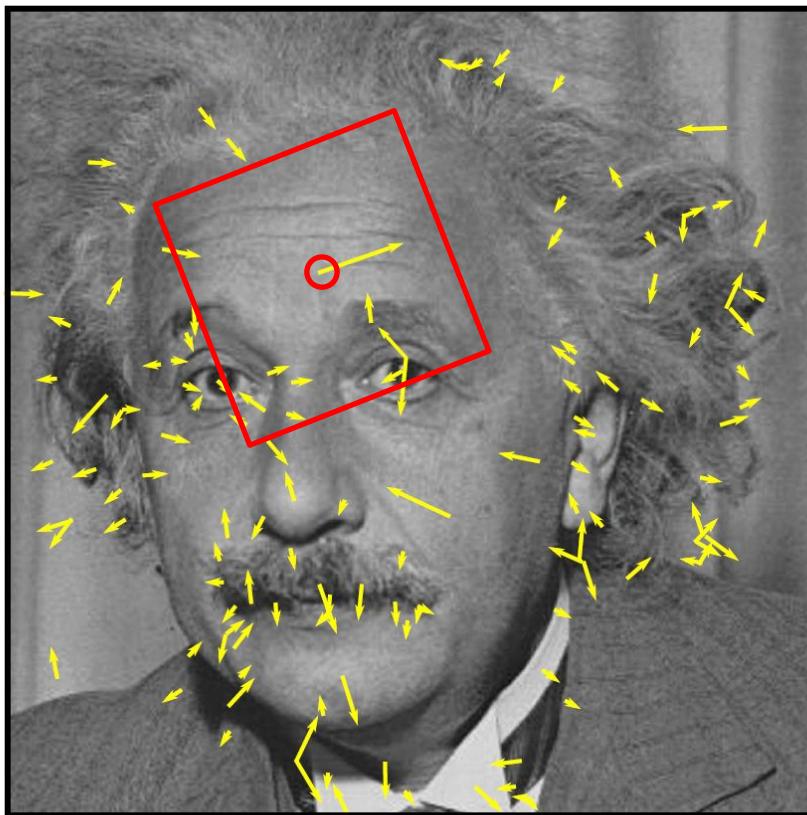
Set orientation to peak of histogram.

# Structure

- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

# Sift Descriptor

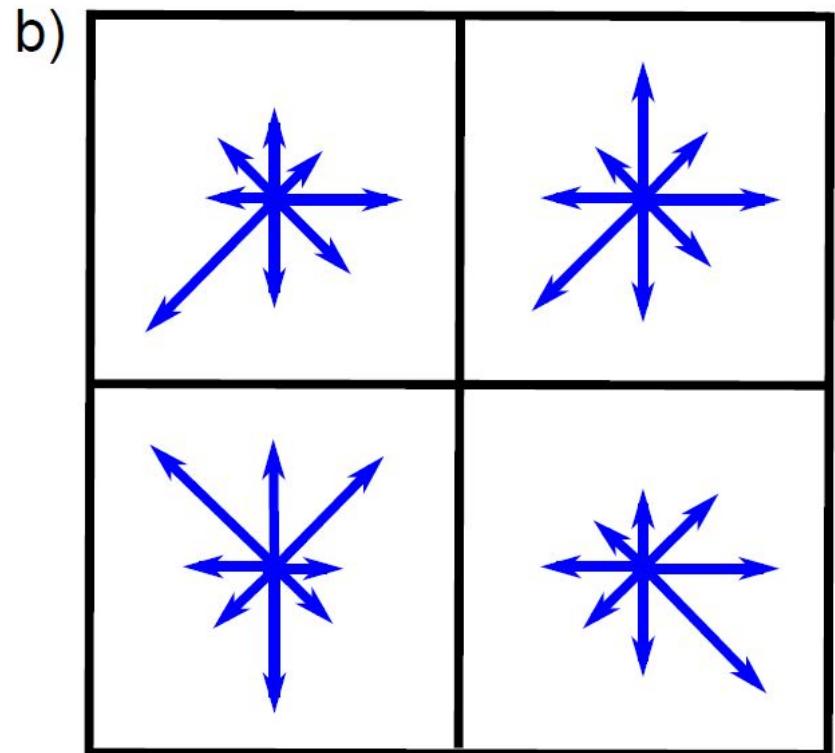
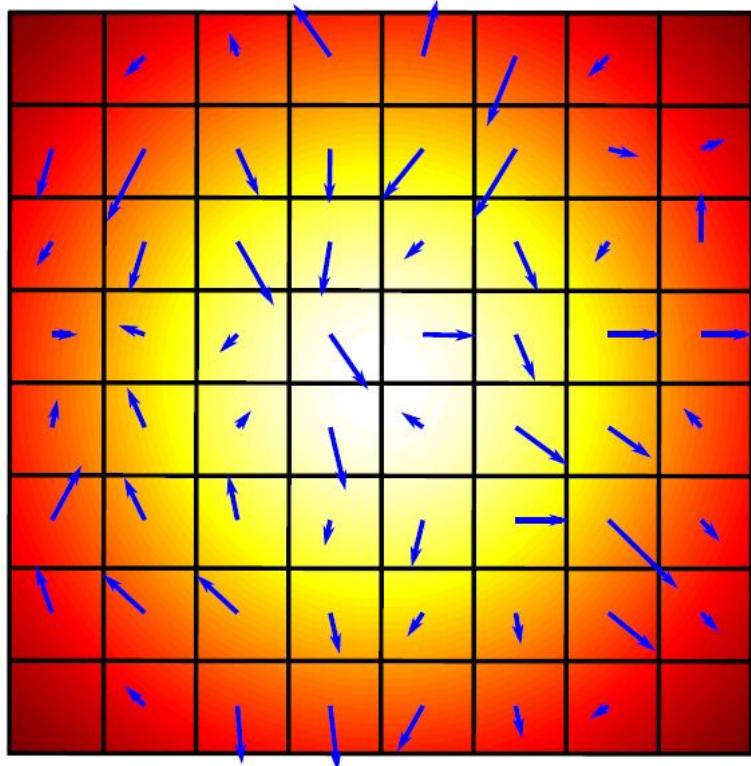
Goal: produce a vector that describes the region around the interest point.



All calculations are relative to the orientation and scale of the keypoint

Makes descriptor invariant to rotation and scale

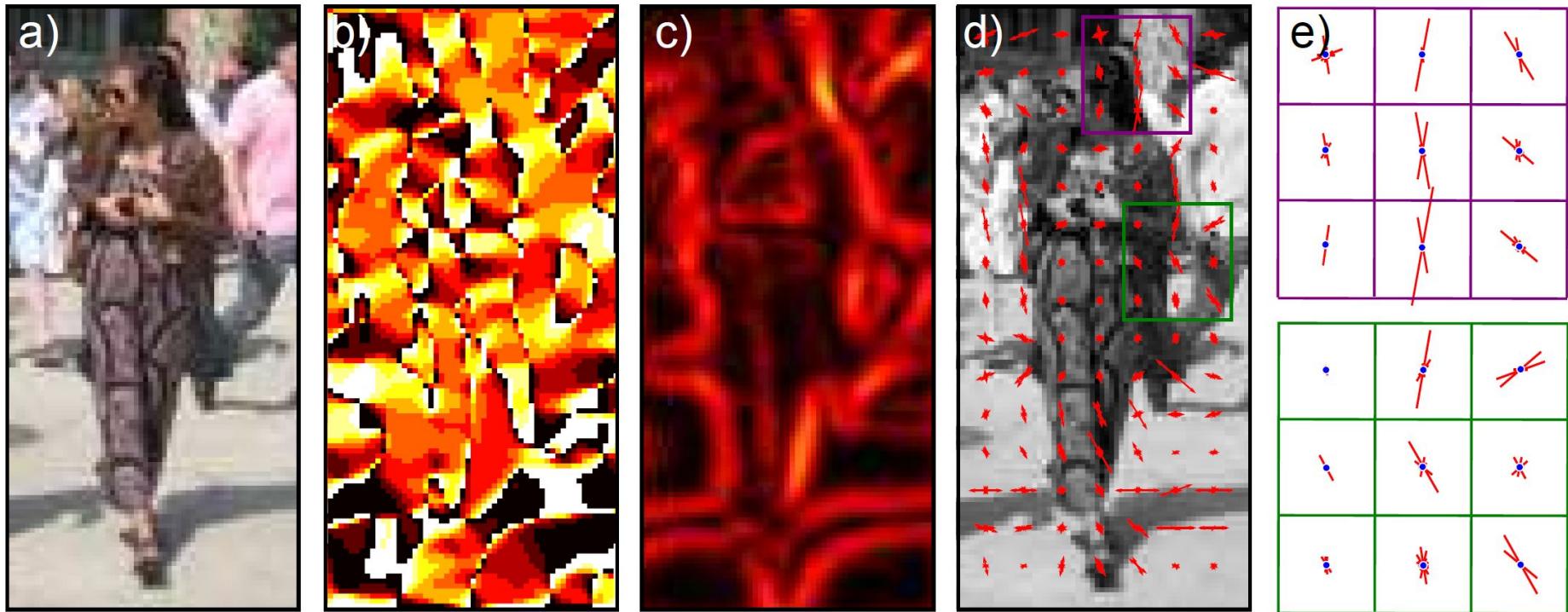
# Sift Descriptor



1. Compute image gradients

2. Pool into local histograms
3. Concatenate histograms
4. Normalize histograms

# HoG Descriptor

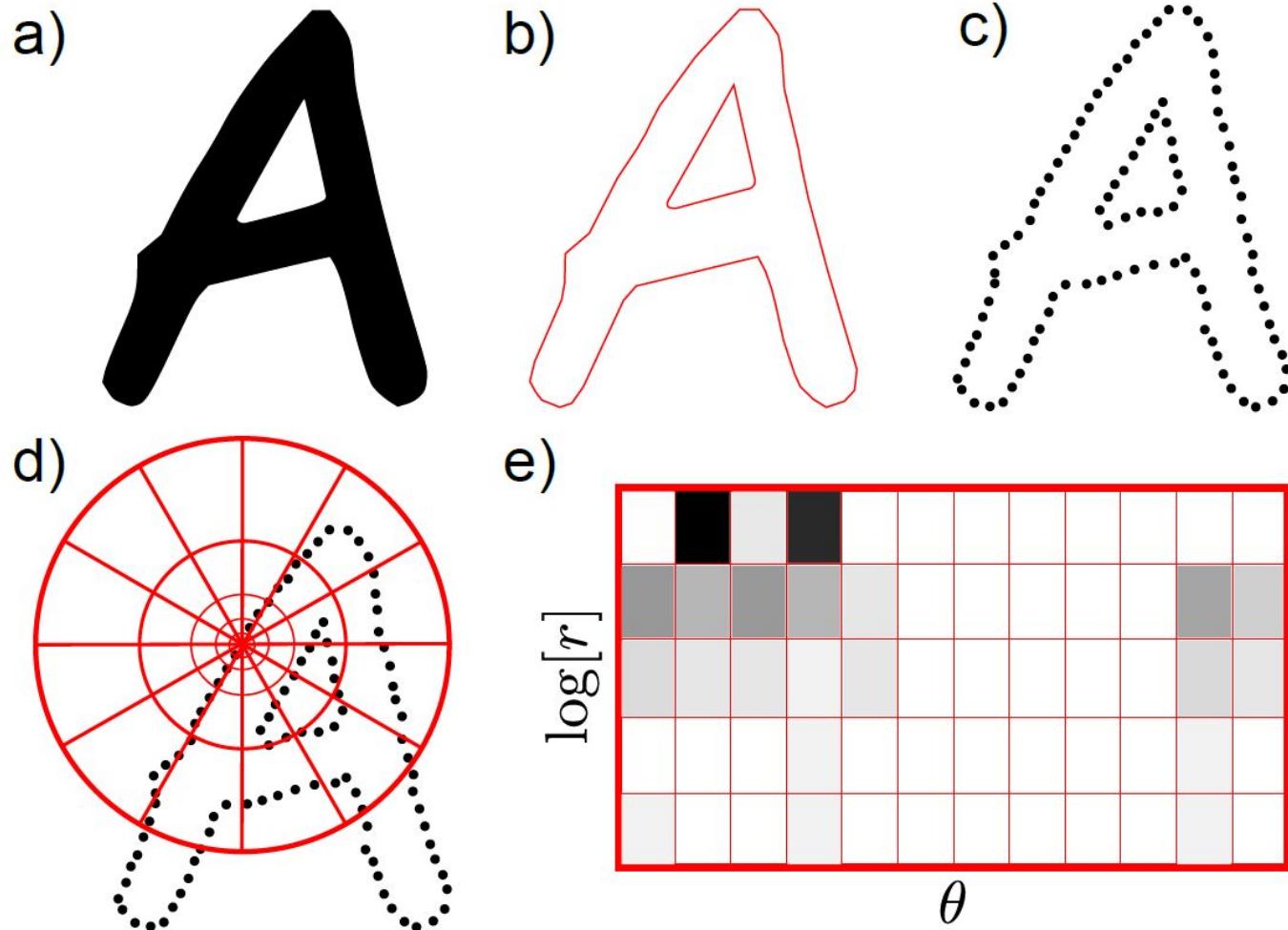


**Figure 13.17** HOG descriptor. a) Original image. b) Gradient orientation, quantized into 9 bins from 0 – 180°. c) Gradient magnitude. d) Cell descriptors are 9D orientation histograms that are computed within  $6 \times 6$  pixel regions. e) Block descriptors are computed by concatenating  $3 \times 3$  blocks of cell descriptors. The block descriptors are normalized. The final HOG descriptor consists of the concatenated block descriptors.

# Bag of words descriptor

- Compute visual features in image
- Compute descriptor around each
- Find closest match in library and assign index
- Compute histogram of these indices over the region
- Dictionary computed using K-means

# Shape context descriptor



# Structure

- Per-pixel transformations
- Edges, corners, and interest points
- Descriptors
- Dimensionality reduction

# Dimensionality Reduction

Dimensionality reduction attempt to find a low dimensional (or hidden) representation  $\mathbf{h}$  which can approximately explain the data  $\mathbf{x}$  so that

$$\mathbf{x} \approx f(\mathbf{h}, \theta)$$

where  $f[\bullet, \bullet]$  is a function that takes the hidden variable and a set of parameters  $\theta$ .

Typically, we choose the function family  $f[\bullet, \bullet]$  and then learn  $\mathbf{h}$  and  $\theta$  from training data

# Least Squares Criterion

$$\hat{\theta}, \hat{\mathbf{h}}_{1\dots I} = \operatorname{argmin}_{\theta, \mathbf{h}_1\dots I} \left[ \sum_{i=1}^I (\mathbf{x}_i - f[\mathbf{h}_i, \theta])^T (\mathbf{x}_i - f[\mathbf{h}_i, \theta]) \right]$$

Choose the parameters  $\theta$  and the hidden variables  $\mathbf{h}$  so that they minimize the least squares approximation error (a measure of how well they can reconstruct the data  $\mathbf{x}$ ).

# Simple Example

$$\mathbf{x}_i \approx \phi h_i + \boldsymbol{\mu}$$

Approximate each data example  $\mathbf{x}$  with a scalar value  $h$ .

Data is reconstructed by multiplying  $h$  by a parameter  $\phi$  and adding the mean vector  $\boldsymbol{\mu}$ .

... or even better, lets subtract  $\boldsymbol{\mu}$  from each data example to get mean-zero data

# Simple Example

$$\mathbf{x}_i \approx \phi h_i$$

Approximate each data example  $\mathbf{x}$  with a scalar value  $h$ .

Data is reconstructed by multiplying  $h$  by a factor  $\phi$ .

**Criterion:**

$$\hat{\phi}, \hat{h}_{1\dots I} = \underset{\phi, h_{1\dots I}}{\operatorname{argmin}} [E] = \underset{\phi, h_{1\dots I}}{\operatorname{argmin}} \left[ \sum_{i=1}^I (\mathbf{x}_i - \phi h_i)^T (\mathbf{x}_i - \phi h_i) \right]$$

# Criterion

$$\hat{\phi}, \hat{h}_{1\dots I} = \underset{\phi, h_{1\dots I}}{\operatorname{argmin}} [E] = \underset{\phi, h_{1\dots I}}{\operatorname{argmin}} \left[ \sum_{i=1}^I (\mathbf{x}_i - \phi h_i)^T (\mathbf{x}_i - \phi h_i) \right]$$

Problem: the problem is non-unique. If we multiply  $\mathbf{f}$  by any constant  $\alpha$  and divide each of the hidden variables  $h_{1\dots I}$  by the same constant we get the same cost. (i.e.  $(\mathbf{f}\alpha) (\mathbf{h}/\alpha) = \mathbf{f}\mathbf{h}$ )

Solution: We make the solution unique by constraining the length of  $\mathbf{f}$  to be 1 using a Lagrange multiplier.

# Criterion

Now we have the new cost function:

$$\begin{aligned} E &= \sum_{i=1}^I (\mathbf{x}_i - \boldsymbol{\phi} h_i)^T (\mathbf{x}_i - \boldsymbol{\phi} h_i) + \lambda(\boldsymbol{\phi}^T \boldsymbol{\phi} - 1) \\ &= \sum_{i=1}^I \mathbf{x}_i^T \mathbf{x}_i - 2h_i \boldsymbol{\phi}^T \mathbf{x}_i + h_i^2 + \lambda(\boldsymbol{\phi}^T \boldsymbol{\phi} - 1). \end{aligned}$$

To optimize this we take derivatives with respect to  $\boldsymbol{\phi}$  and  $h_i$ , equate the resulting expressions to zero and re-arrange.

# Solution

$$\hat{h}_i = \hat{\phi}^T \mathbf{x}_i$$

To compute the hidden value, take dot product with the vector  $\phi$

# Solution

$$\hat{h}_i = \hat{\phi}^T \mathbf{x}_i$$

To compute the hidden value, take dot product with the vector  $\hat{\phi}$

---

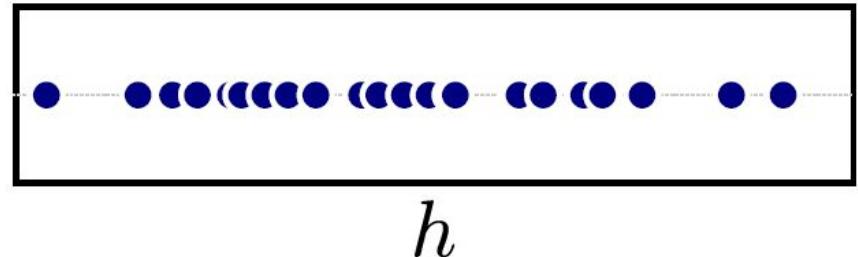
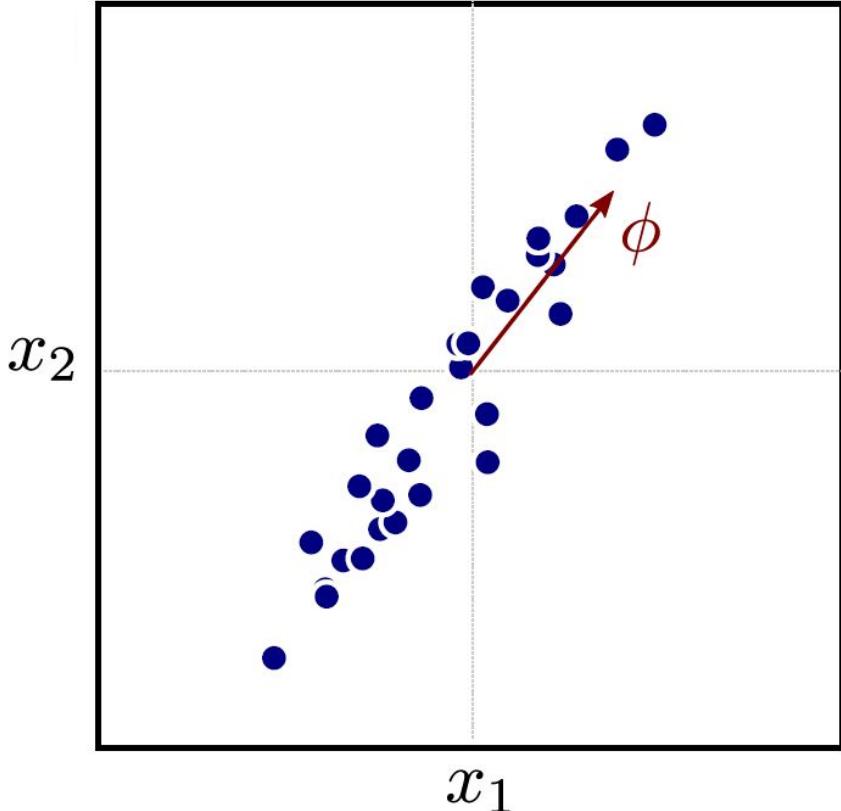
$$\sum_{i=1}^I \mathbf{x}_i \mathbf{x}_i^T \hat{\phi} = \lambda \hat{\phi}$$

or

$$\mathbf{X} \mathbf{X}^T \hat{\phi} = \lambda \hat{\phi} \quad \text{where} \quad \mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_I]$$

To compute the vector  $\hat{\phi}$ , compute the first eigenvector of the scatter matrix  $\mathbf{X} \mathbf{X}^T$ .

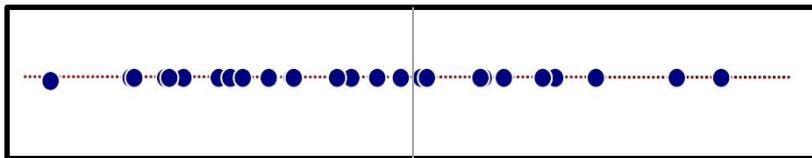
# Computing $h$



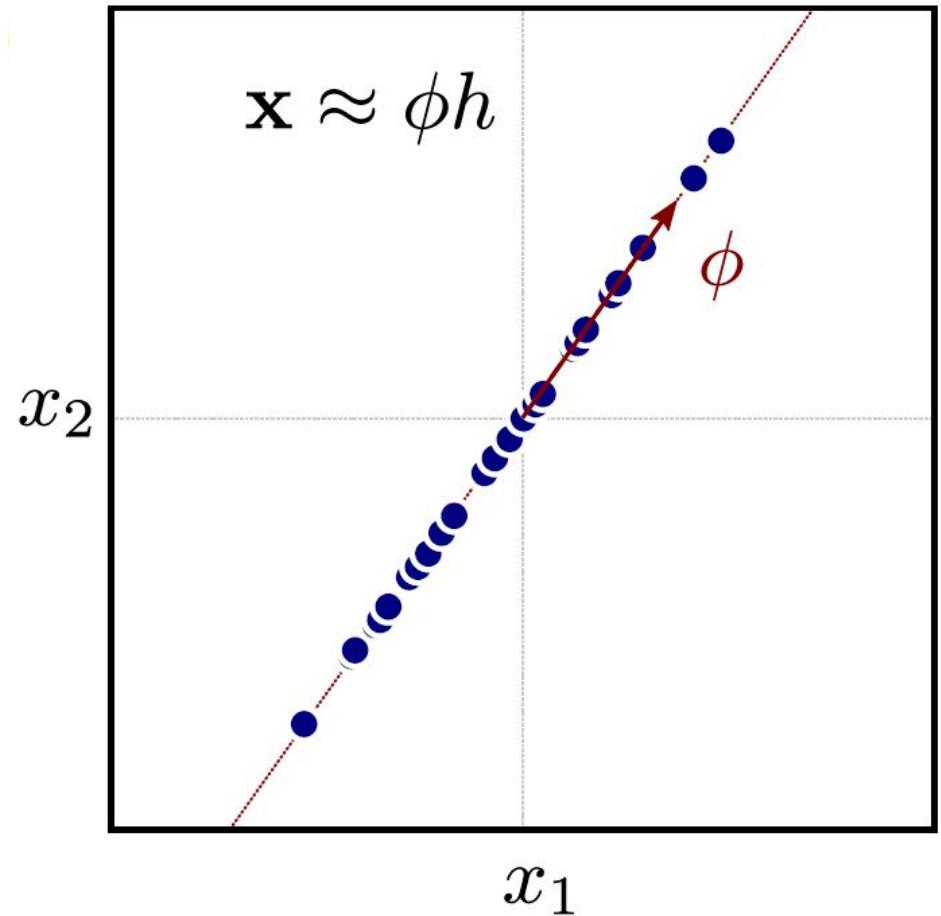
To compute the hidden value, take dot product with the vector  $\phi$

# Reconstruction

$$\mathbf{x}_i \approx \phi h_i$$



$h_i$



To reconstruct, multiply the hidden variable  $h$  by vector  $\varphi$ .

# Principal Components Analysis

Same idea, but now the hidden variable  $\mathbf{h}$  is multi-dimensional.  
Each component weights one column of matrix  $\mathbf{F}$  so that data is approximated as

$$\mathbf{x}_i \approx \Phi \mathbf{h}_i$$

This leads to cost function:

$$\hat{\Phi}, \mathbf{h}_1 \dots \mathbf{h}_I = \underset{\Phi, \mathbf{h}_1 \dots \mathbf{h}_I}{\operatorname{argmin}} [E] = \underset{\Phi, \mathbf{h}_1 \dots \mathbf{h}_I}{\operatorname{argmin}} \left[ \sum_{i=1}^I (\mathbf{x}_i - \Phi \mathbf{h}_i)^T (\mathbf{x}_i - \Phi \mathbf{h}_i) \right]$$

This has a non-unique optimum so we enforce the constraint that  $\mathbf{F}$  should be a (truncated) rotation matrix and  $\mathbf{F}^T \mathbf{F} = \mathbf{I}$

# PCA Solution

$$\mathbf{h}_i = \boldsymbol{\Phi}^T \mathbf{x}_i$$

To compute the hidden vector, take dot product with each column of  $\boldsymbol{\Phi}$ .

---

To compute the matrix  $\boldsymbol{\Phi}$ , compute the first  $D_h$  eigenvectors of the scatter matrix  $\mathbf{X}\mathbf{X}^T$ .

The basis functions in the columns of  $\boldsymbol{\Phi}$  are called **principal components** and the entries of  $\mathbf{h}$  are called **loadings**

# Dual PCA

**Problem:** PCA as described has a major drawback. We need to compute the eigenvectors of the scatter matrix

$$\mathbf{X}\mathbf{X}^T$$

But this has size  $D_x \times D_x$ . Visual data tends to be very high dimensional, so this may be extremely large.

**Solution:** Reparameterize the principal components as weighted sums of the data

$$\Phi = \mathbf{X}\Psi$$

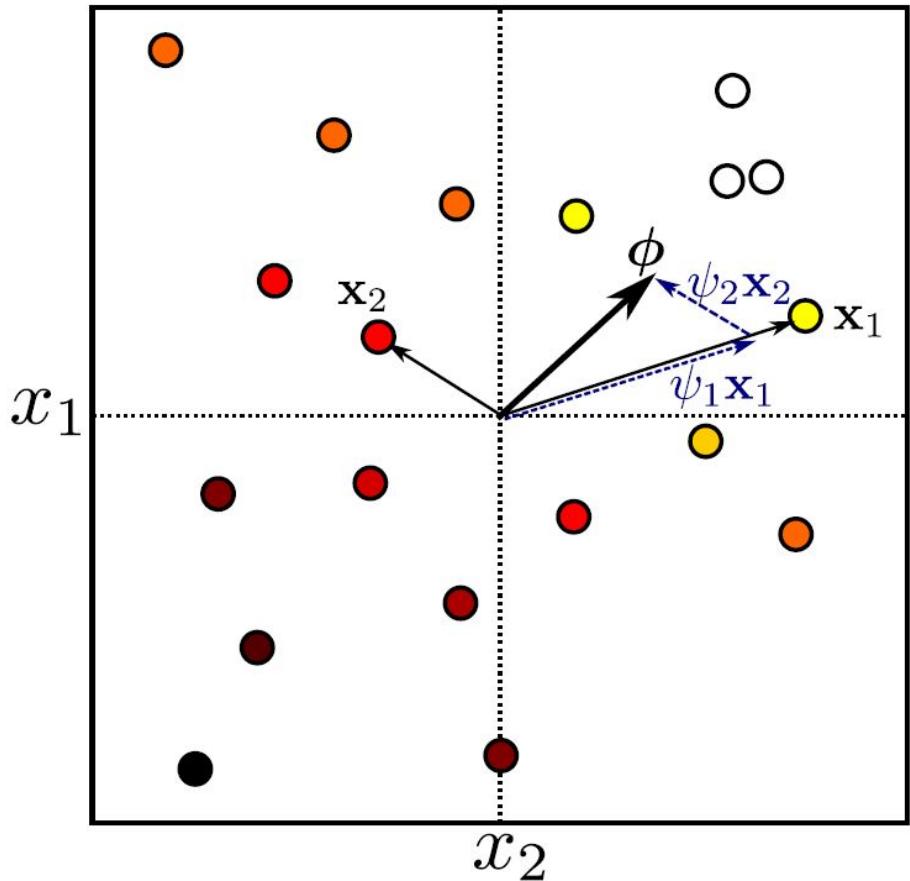
...and solve for the new variables  $\Psi$ .

# Geometric Interpretation

$$\Phi = X\Psi$$

Each column of  $\Phi$  can be described as a weighted sum of the original datapoints.

Weights given in the corresponding columns of the new variable  $\Psi$ .



# Motivation

**Solution:** Reparameterize the principal components as weighted sums of the data

$$\Phi = X\Psi$$

...and solve for the new variables  $\Psi$ .

**Why?** If the number of datapoints  $I$  is less than the number of observed dimension  $D_x$  then the  $\Psi$  will be smaller than  $\Phi$  and the resulting optimization becomes easier.

**Intuition:** we are not interested in principal components that are not in the subspace spanned by the data anyway.

# Cost functions

Principal components analysis

$$\hat{\Phi}, \hat{\mathbf{h}}_1 \dots I = \underset{\Phi, \mathbf{h}_1 \dots I}{\operatorname{argmin}} [E] = \underset{\Phi, \mathbf{h}_1 \dots I}{\operatorname{argmin}} \left[ \sum_{i=1}^I (\mathbf{x}_i - \Phi \mathbf{h}_i)^T (\mathbf{x}_i - \Phi \mathbf{h}_i) \right]$$

...subject to

$$\cancel{\Phi^T \Phi = \mathbf{I}}$$

---

Dual principal components analysis

$$E = \sum_{i=1}^I (\mathbf{x}_i - \mathbf{X} \Psi \mathbf{h}_i)^T (\mathbf{x}_i - \mathbf{X} \Psi \mathbf{h}_i)$$

...subject to  $\Phi^T \Phi = \mathbf{I}$  or  $\Psi^T \mathbf{X}^T \mathbf{X} \Psi = \mathbf{I}$ .

# Solution

$$\mathbf{h}_i = \boldsymbol{\Psi}^T \mathbf{X}^T \mathbf{x}_i = \boldsymbol{\Phi}^T \mathbf{x}_i$$

To compute the hidden vector, take dot product with each column of  $\boldsymbol{\Phi}=\boldsymbol{\Psi}\mathbf{X}$ .

# Solution

$$\mathbf{h}_i = \boldsymbol{\Psi}^T \mathbf{X}^T \mathbf{x}_i = \boldsymbol{\Phi}^T \mathbf{x}_i$$

To compute the hidden vector, take dot product with each column of  $\boldsymbol{\Phi} = \boldsymbol{\Psi} \mathbf{X}$ .

---

To compute the matrix  $\boldsymbol{\Psi}$ , compute the first  $D_h$  eigenvectors of the inner product matrix  $\mathbf{X}^T \mathbf{X}$ .

The inner product matrix has size  $I \times I$ .

If the number of examples  $I$  is less than the dimensionality of the data  $D_x$  then this is a smaller eigenproblem.

# K-Means algorithm

Approximate data with a set of means

$$\mathbf{x}_i \approx \boldsymbol{\mu}_{h_i}$$

Least squares criterion

$$\hat{\boldsymbol{\mu}}_{1\dots K}, \hat{h}_{1\dots I} = \operatorname{argmin}_{\boldsymbol{\mu}, h} \left[ \sum_{i=1}^I (\mathbf{x}_i - \boldsymbol{\mu}_{h_i})^T (\mathbf{x}_i - \boldsymbol{\mu}_{h_i}) \right]$$

Alternate minimization

$$\hat{h}_i = \operatorname{argmin}_{h_i} \left[ (\mathbf{x}_i - \boldsymbol{\mu}_{h_i})^T (\mathbf{x}_i - \boldsymbol{\mu}_{h_i}) \right]$$

$$\begin{aligned}\hat{\boldsymbol{\mu}}_k &= \operatorname{argmin}_{\boldsymbol{\mu}_k} \left[ \sum_{i=1}^I \left[ (\mathbf{x}_i - \boldsymbol{\mu}_{h_i})^T (\mathbf{x}_i - \boldsymbol{\mu}_{h_i}) \right] \right] \\ &= \frac{\sum_{i=1}^I \mathbf{x}_i \delta[h_i - k]}{\sum_{i=1}^I \delta[h_i - k]},\end{aligned}$$

