ENGN6528

Week-6A (Face/Human Detection)

Eigenfaces, Turk and Pentland, 1991

Review: Eigenfaces (PCA on face images)

1. Compute covariance matrix of face images

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

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

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

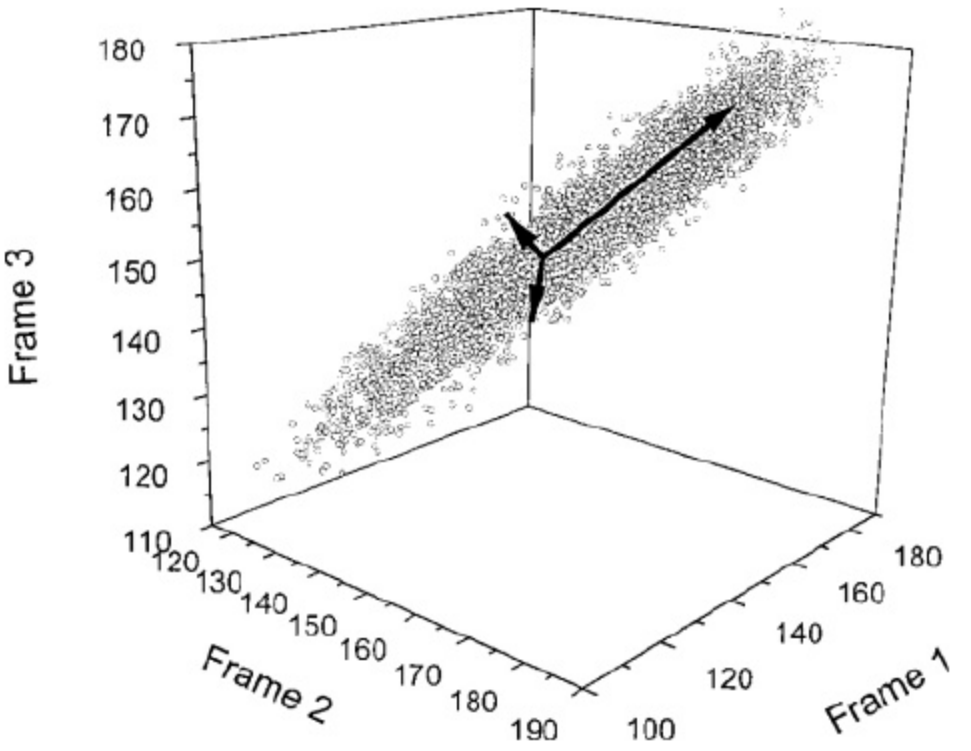


Fig. 1.11. *Principal component analysis seeks to fit a subspace of dimension less than n to a set of points in* $\mathbb{R}^n$.

PCA algorithm.

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

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

(ii) Form the scatter matrix

$$\mathtt{Y} = \sum_{i=1}^{k} (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^{\mathsf{T}} \ .$$

(iii) Let $\mathtt{A}$ be the matrix whose columns are the eigenvectors corresponding to the m **largest** eigenvalues of $\mathtt{Y}$.

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

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

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

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

Table 1.1. PCA algorithm

Review: PCA Algorithm

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

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

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

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

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

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

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

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

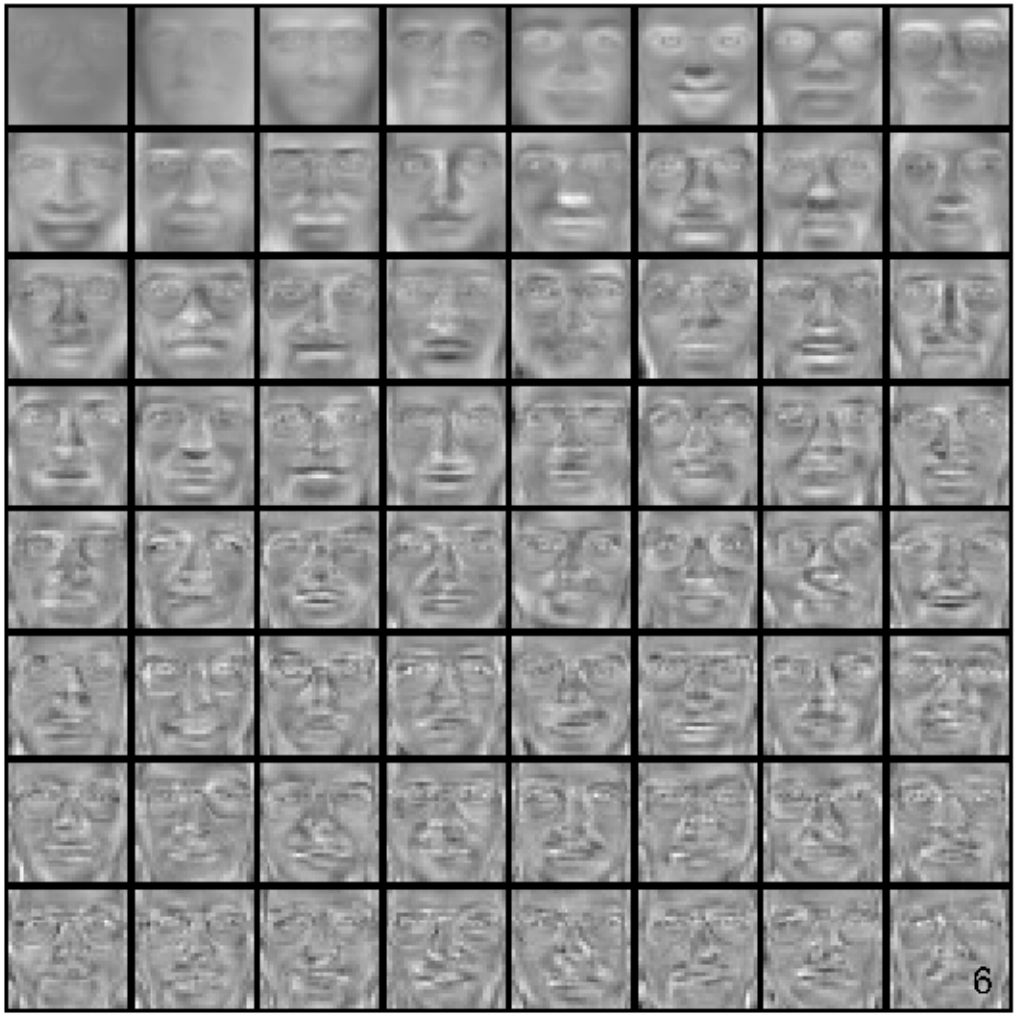
Review: Eigenfaces example

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

Review: Eigenfaces example

Mean: μ

Top eigenvectors: $u_1,\ldots u_k$



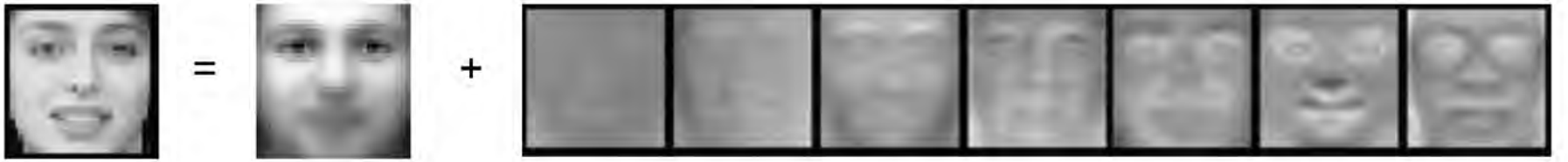
Representation and Reconstruction

- Represent a face **x** in "face space" as coordinates:

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

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

- Reconstruction:



$\hat{x} = \mu + w_1u_1+w_2u_2+w_3u_3+w_4u_4+ \ldots$

Eigen-faces for Recognition

Process labeled M training images

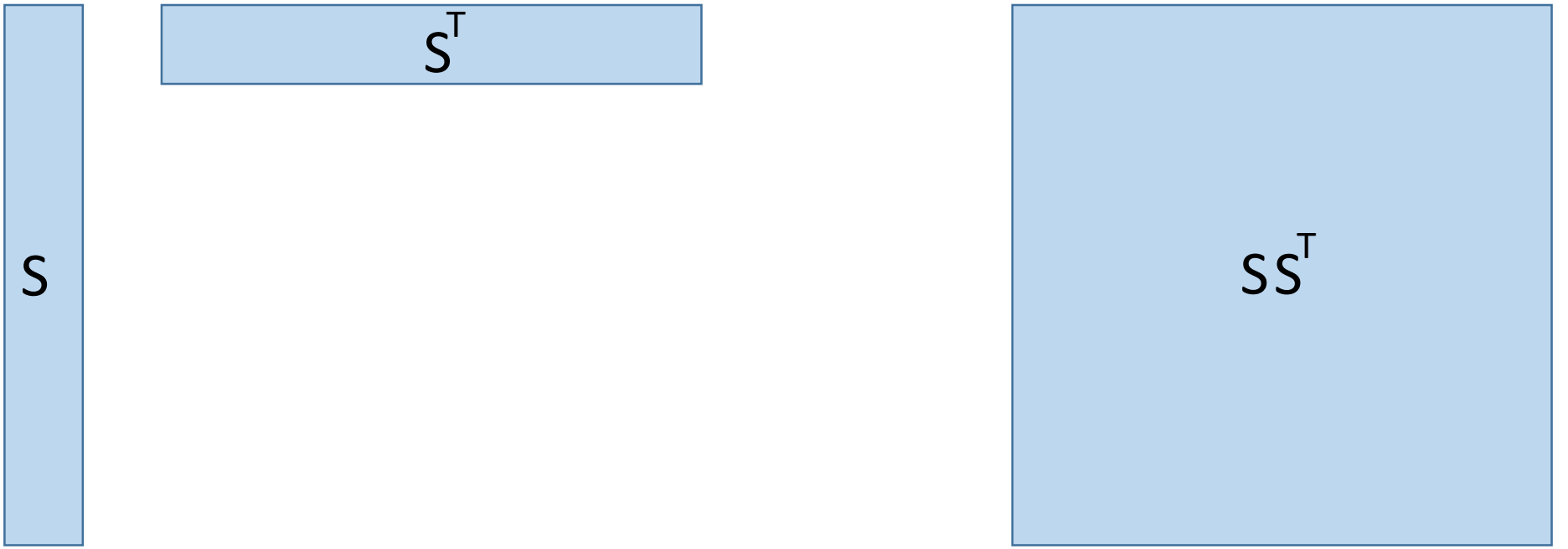
- Find mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
- Find k principal components (eigenvectors of $\boldsymbol{\Sigma}$) $\mathbf{u}_1,\ldots\mathbf{u}_k$
- Project each training image $\mathbf{x}_i$ onto subspace spanned by principal components:
  $(w_{i1},\ldots,w_{ik}) = (\mathbf{u}_1^T(\mathbf{x}_i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

Given novel image $\mathbf{x}$

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

Issues?

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



S^TS

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

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

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

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

PCA dimension for high-dimensional ambient space.

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

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

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

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

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

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

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

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

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

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

Solution

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

- Step 7: Normalize the eigenvalue by dividing by its norm.

PCA using Singular Value Decomposition.

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

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

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

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

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

Table 1.3. PCA algorithm using the Singular Value Decomposition.

Projection of a vector y onto the PCA subspace. Most often, once a PCA subspace fitting a set of points $\mathbf{y}_i$ has been found, it is used to project further points $\mathbf{y}$ onto that subspace.

Recall that the PCA subspace is determined by a mean vector $\bar{\mathbf{y}}$ and a matrix $\mathtt{A}$ consisting of the largest eigenvectors of the scatter-matrix $\mathtt{Y}$. Now, given a point $\mathbf{y}$, how do we project it to a point lying in this hyperplane?

The projected point $\pi(\mathbf{y})$ that we are looking at is the point given by

$$\begin{aligned}\pi(\mathbf{y}) &= \bar{\mathbf{y}} + \sum_{i=1}^{m} \langle \mathbf{a}_i, \mathbf{y} - \bar{\mathbf{y}} \rangle \, \mathbf{a}_i \\ &= \bar{\mathbf{y}} + \mathtt{A}\mathtt{A}^{\mathsf{T}}(\mathbf{y} - \bar{\mathbf{y}}) \ .\end{aligned} \tag{1.2}$$

where $\mathbf{a}_i$ is the i-th eigenvalue, column of $\mathtt{A}$. Note that in the case of the algorithm in table 1.2, this is the vector $\mathtt{S}\mathbf{v}_i / \|\mathtt{S}\mathbf{v}_i\|$.

The justification for this formula is that the $\langle \mathbf{y} - \bar{\mathbf{y}}, \mathbf{a}_i \rangle$ is the projection of the vector $\mathbf{y} - \bar{\mathbf{y}}$ on the basis vector direction of $\mathbf{a}$. We use here that the eigenvectors are orthogonal and of unit length.

Note. In computing the projection given in (1.2), it should be computed using the first line of the formula, or equivalently as

$$\pi(\mathbf{y}) = \bar{\mathbf{y}} + \mathtt{A}\left(\mathtt{A}^{\mathsf{T}}(\mathbf{y} - \bar{\mathbf{y}})\right) \ ,$$

without computing the product $\mathtt{A}\mathtt{A}^{\mathsf{T}}$, which can be a very big matrix.

Recognition Using Eigenfaces

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

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

Step 2: project on the eigenspace

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

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

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

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

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

Detection Using Eigenfaces

- Given an unknown image Γ

  Step 1: compute $\Phi = \Gamma - \Psi$

  Step 2: compute $\hat{\Phi} = \sum_{i=1}^{K} w_i u_i \quad (w_i = u_i^T \Phi) \qquad (where \| u_i \| = 1)$

  Step 3: compute $e_d = \|\Phi - \hat{\Phi}\|$

  Step 4: if $e_d < T_d$, then Γ is a face.

- The distance e_d is called distance from face space (dffs)

Object detection, face detection

Visual Object Detection

Task

Given an input image, determine if there are objects of a certain category (e.g. faces, people, cars..) in the image and where they are located.

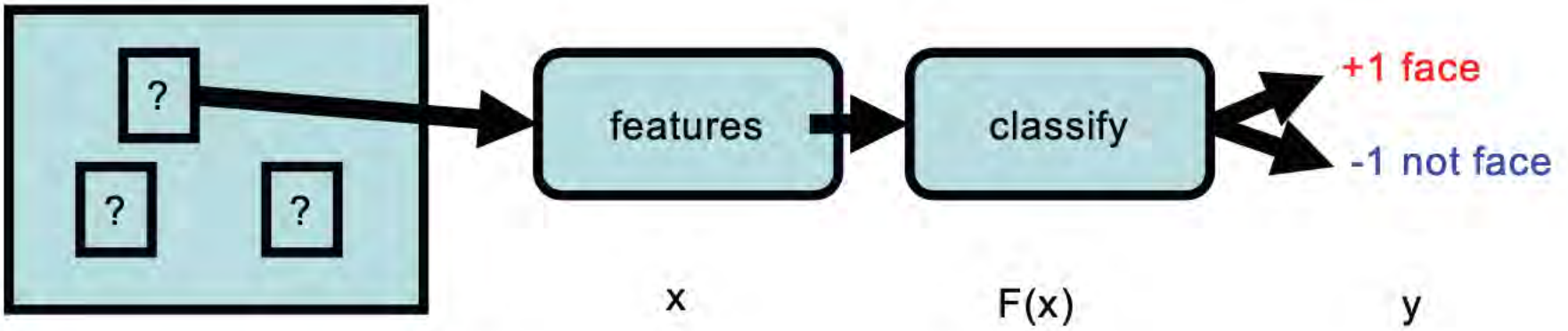
Face detection methods

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

Face detection

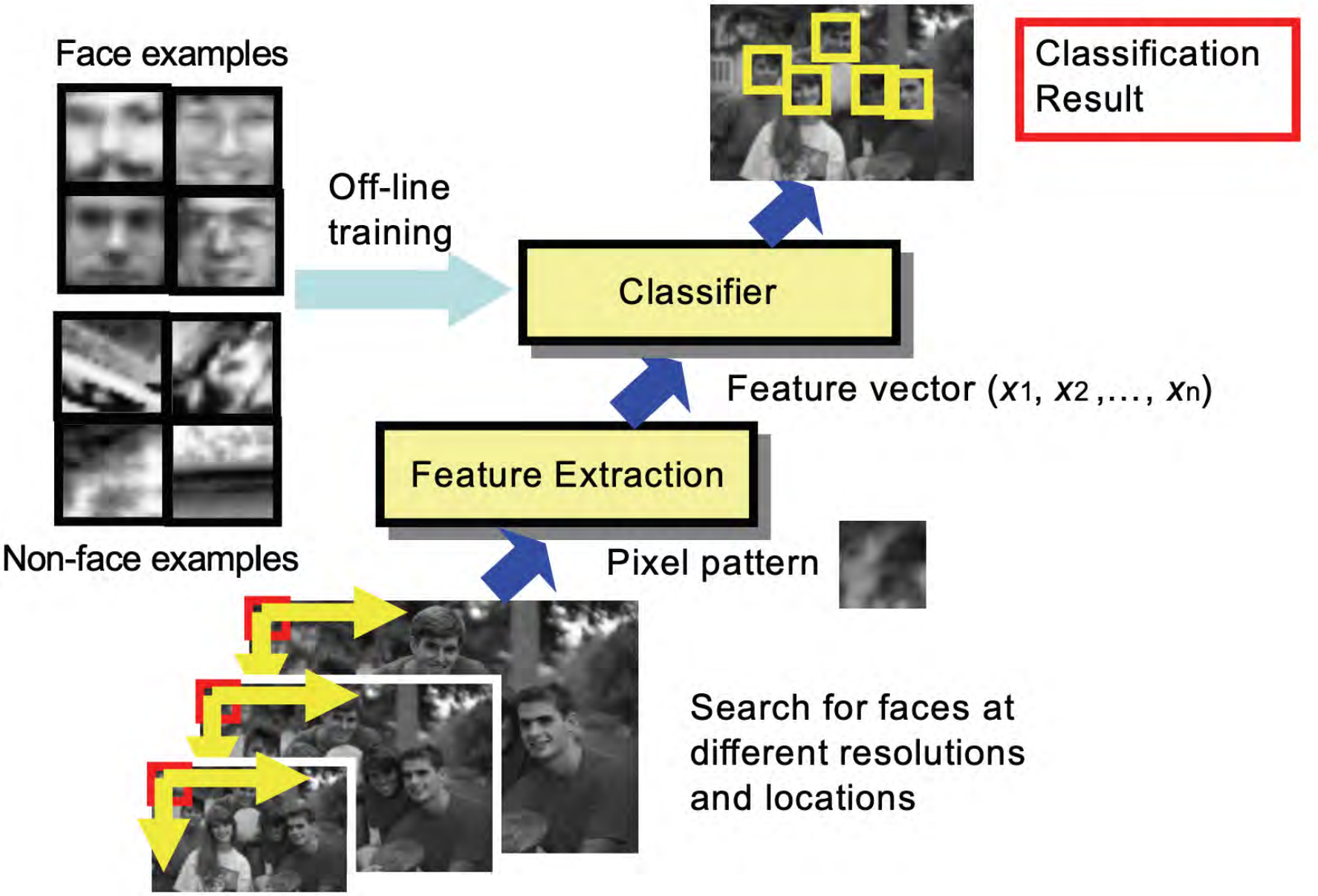
Basic idea: slide a window across image and classify (evaluate) a facial feature representation at every location

Face detection -- basic scheme

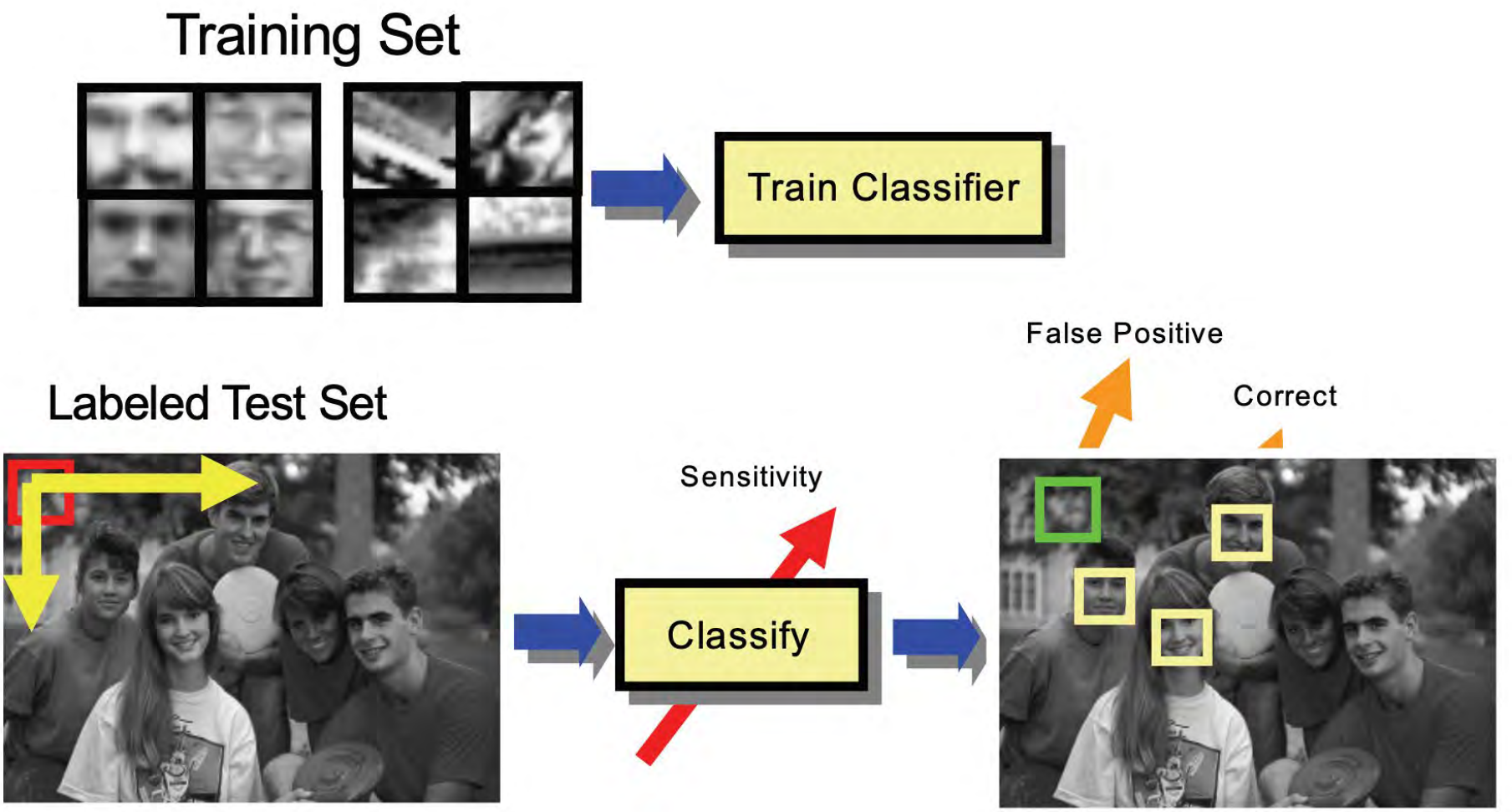


- We slide a window over the image
- Extract features for each window
- Classify each window into face/non-face

Face detection – basic scheme



Training and Testing



Viola-Jones face detection

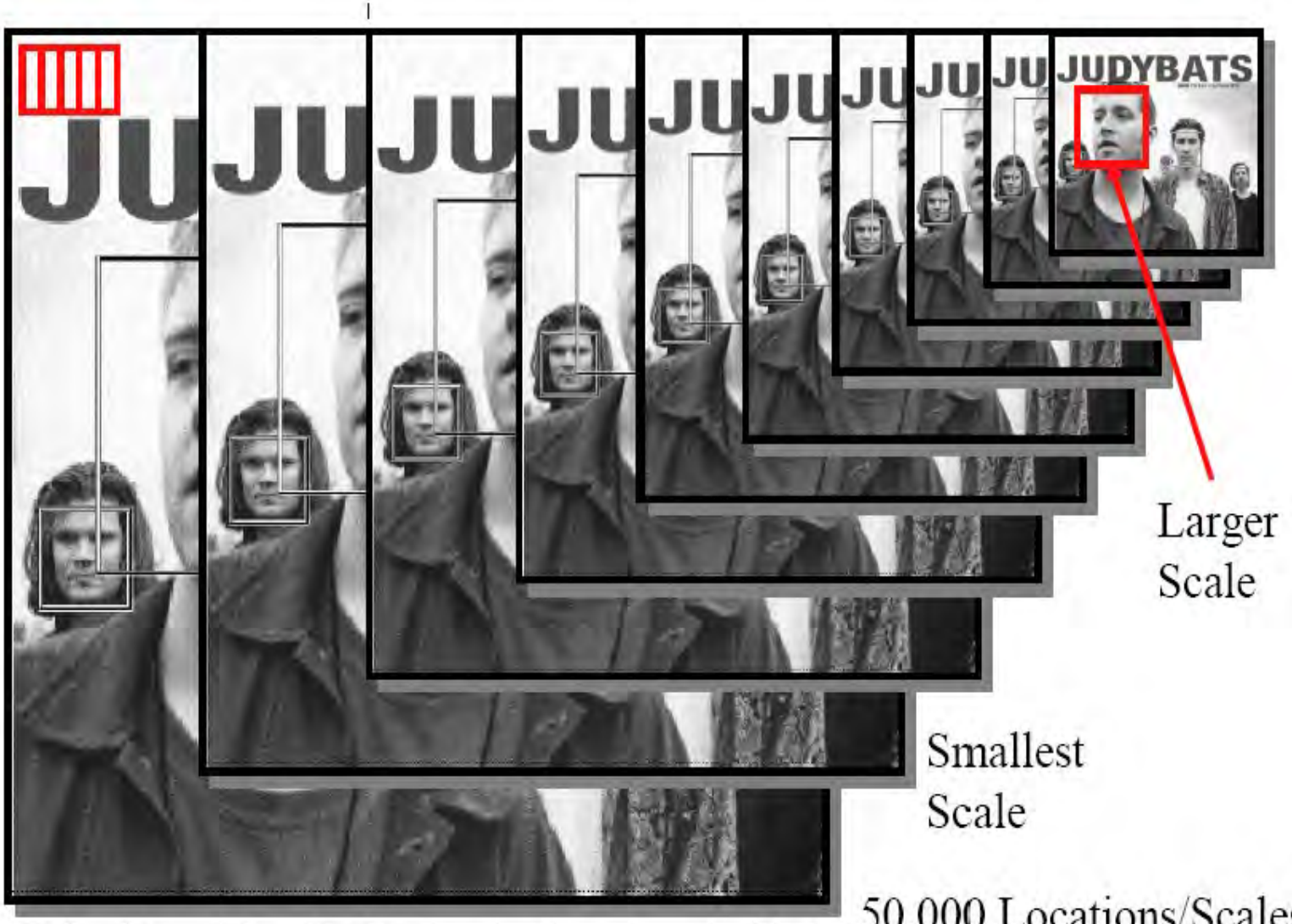
Paul A. Viola and Michael J. Jones

Intl. J. Computer Vision

57(2), 137–154, 2004

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

Scan classifier over locs. & scales



Learn feature & classifier from data

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

Characteristics of the algorithm

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

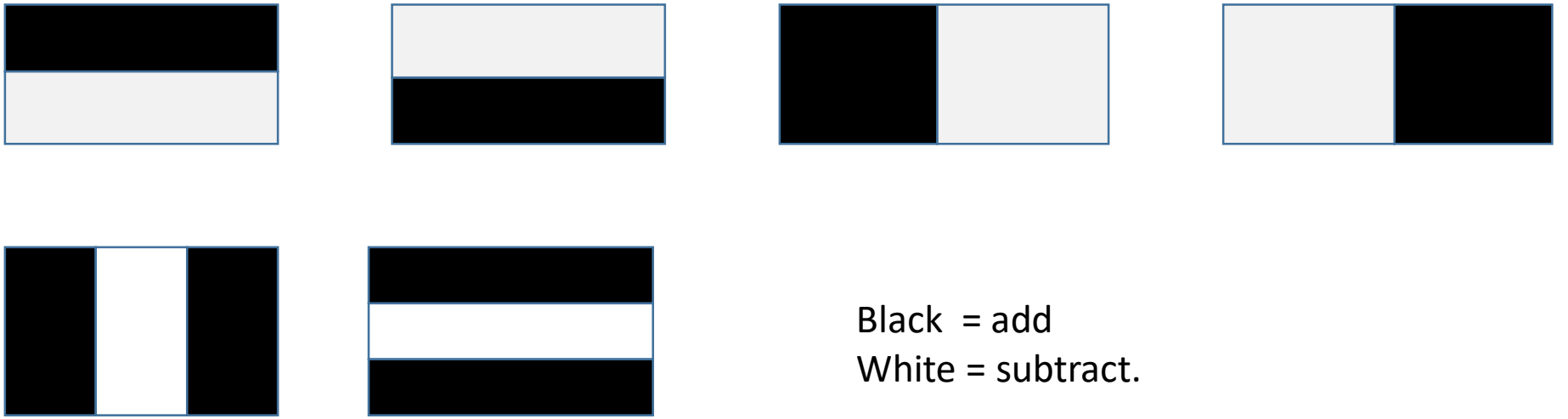
➢ Fastest known face detector for gray scale images (circa 2001, Marr Prize Winner).

Viola-Jones face detection

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

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

Haar features

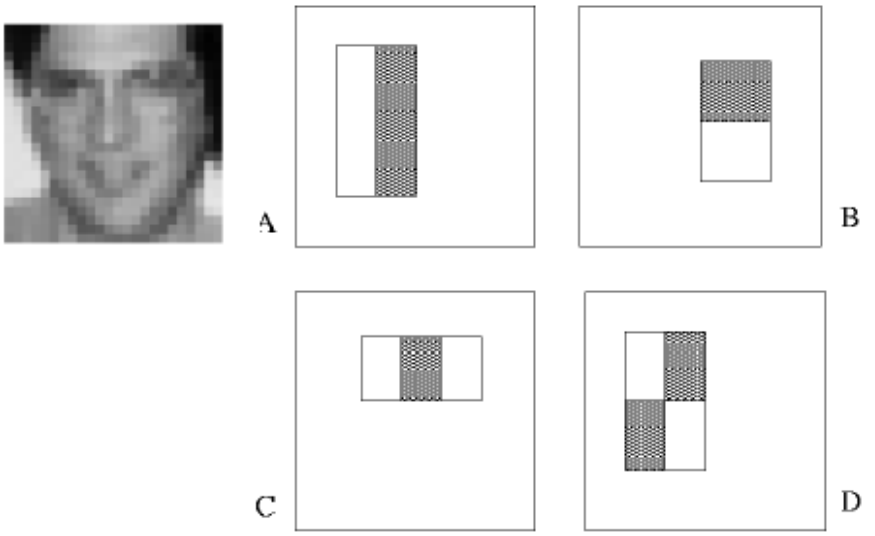


Black = add
White = subtract.

Lots of kernels at different scales and shapes.

Image features

- "Rectangle filters"
- Differences between sums of pixels in adjacent rectangles



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

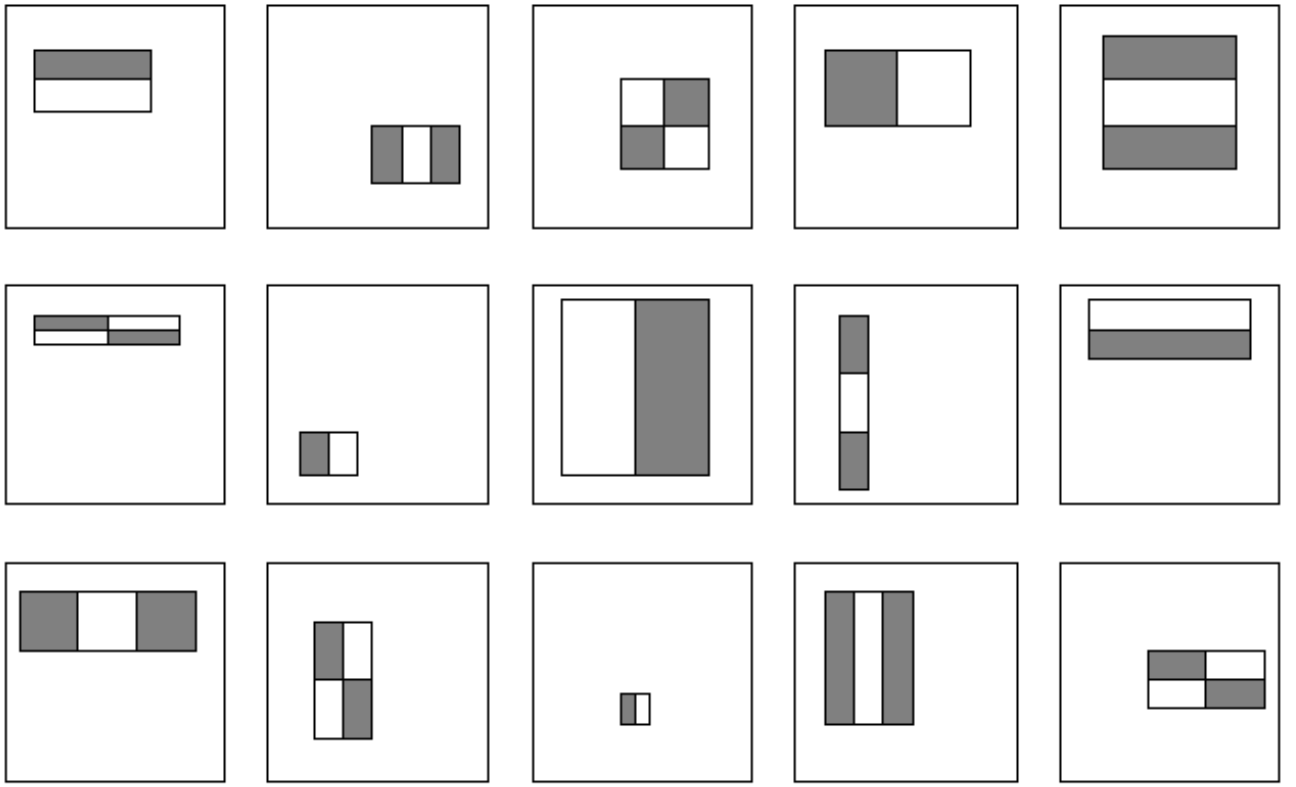
Threshold determined by training

What is a face?

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

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

Huge library of Haar kernels



The integral image

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

Given an image …

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

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

Construct the “integral image”

Integral image

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

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

Integral image

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

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

Integral image

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

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

Integral image

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

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

Integral image

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

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

Integral image

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

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

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

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

Integral image

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

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

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

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

Integral image

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

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

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

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

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

Integral image

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

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

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

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

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

Integral image

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

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

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

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

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

Integral image

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

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

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

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

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

Integral image

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

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

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

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

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

Integral image

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

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

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

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

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

Integral image

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

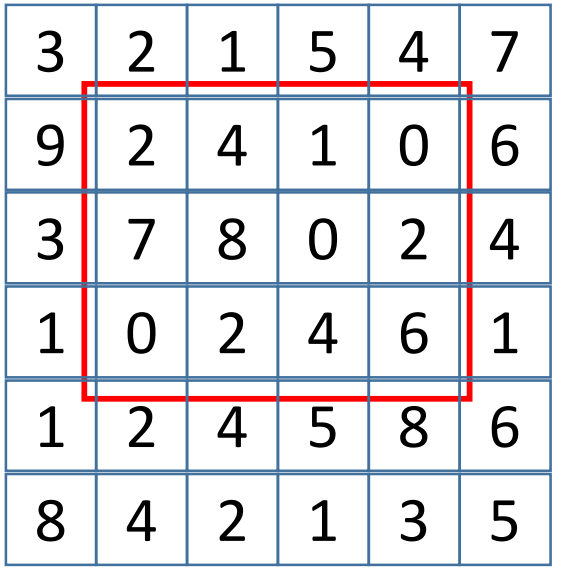
Each value A[i][j] in the integral image represents the sum of the pixels above and to the right.

Eg 45 = Sum of pixel in the original image:

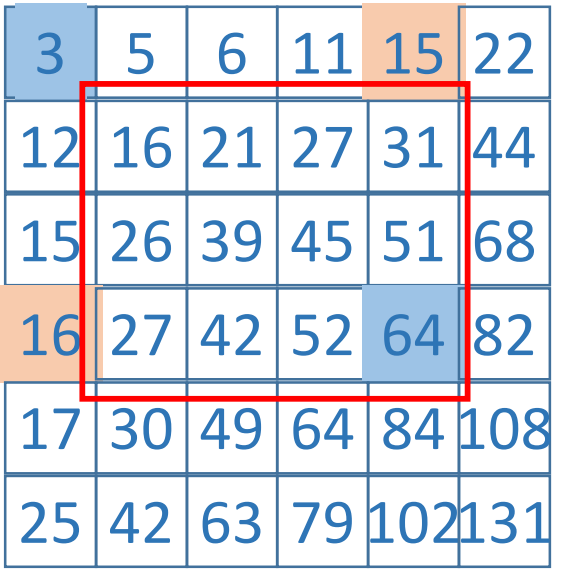
3	5	6	11	15	22
12	16	21	27	31	44
15	26	39	45	51	68
16	27	42	52	64	82
17	30	49	64	84	108
25	42	63	79	102	131

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

Finding the sum of all pixels in any box

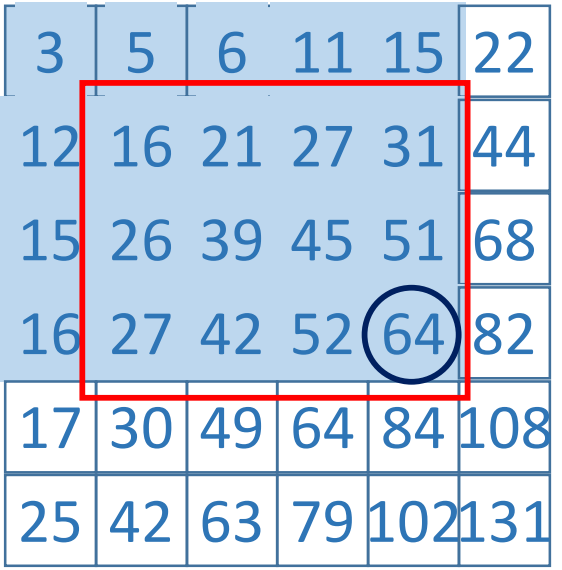


Sum = 36

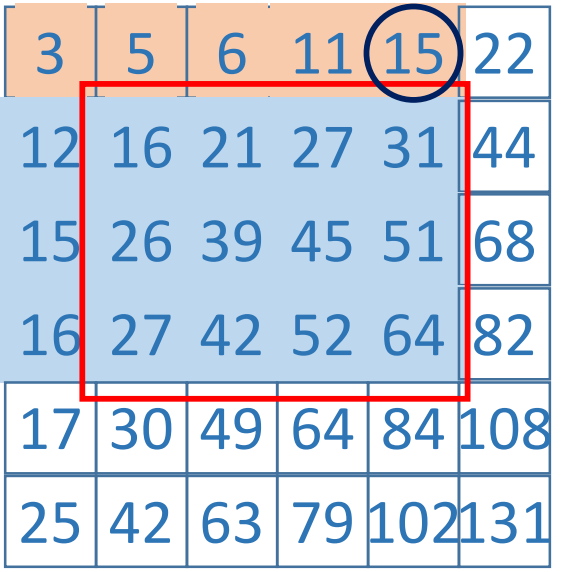


3	5	6	11	15	22
12	16	21	27	31	44
15	26	39	45	51	68
16	27	42	52	64	82
17	30	49	64	84	108
25	42	63	79	102	131

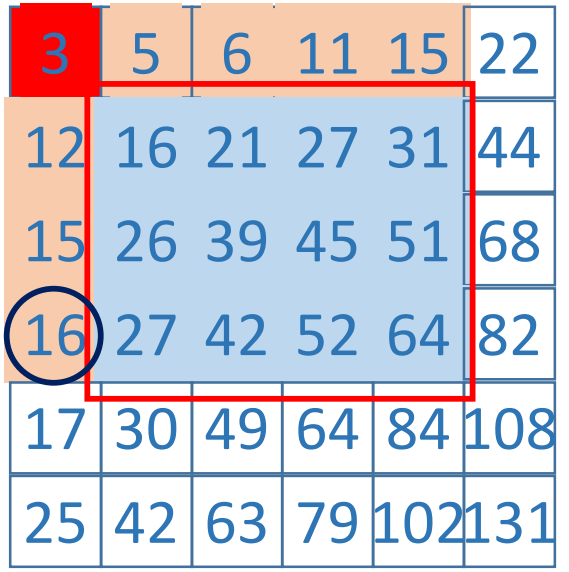
Sum = 36 = 64 + 3 – 15 - 16


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

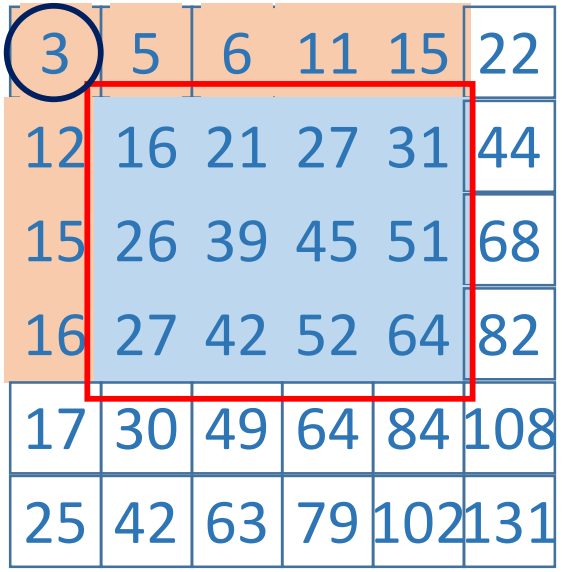
Sum = 36 = 64


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

Sum = 36 = 64 – 15



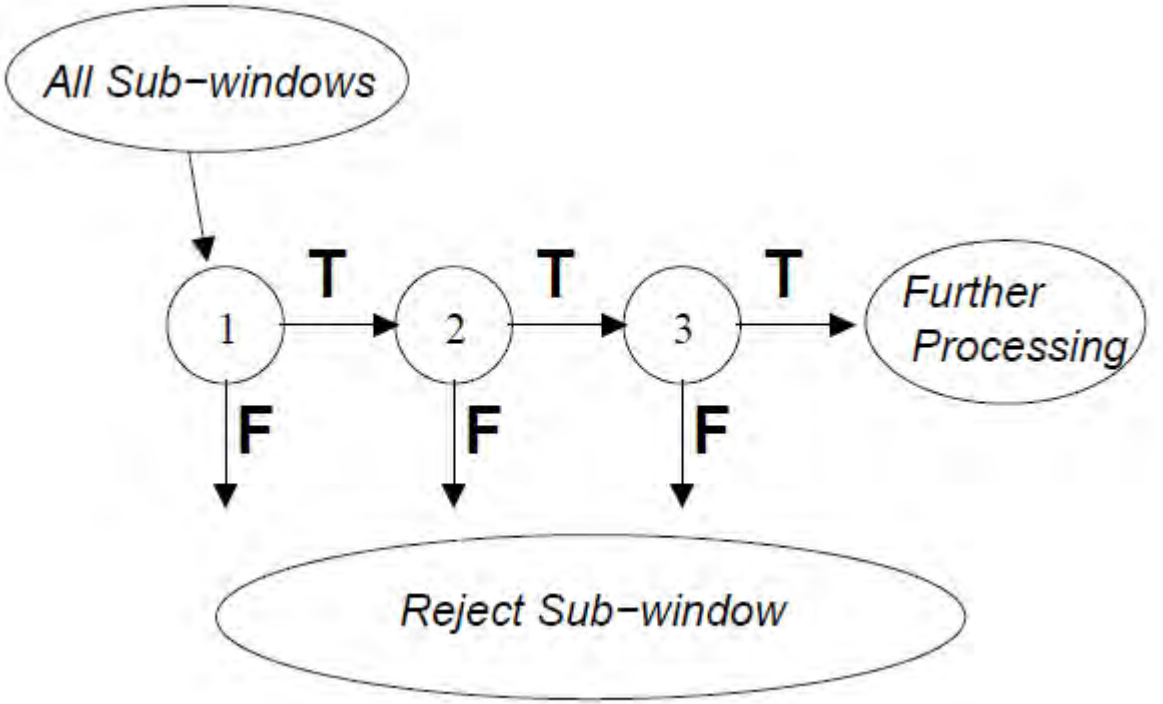
Sum = 36 = 64 - 15 -16



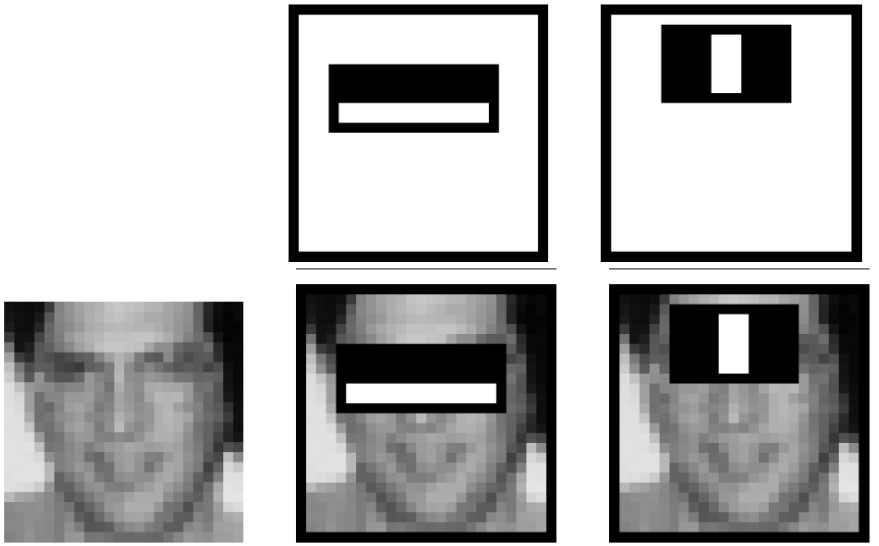
Sum = 36 = 64 - 15 -16 + 3

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

Cascade classifier

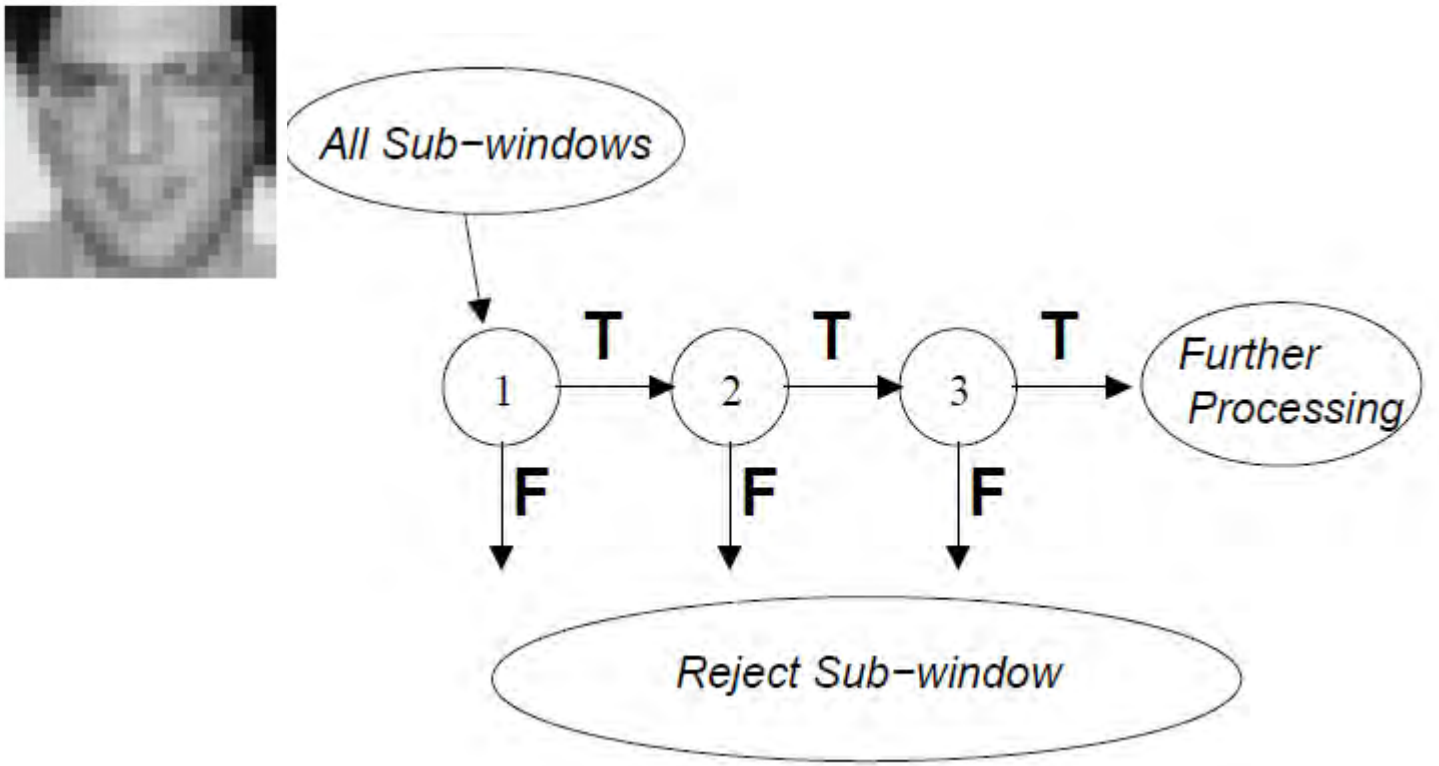

All Sub−windows
1
2
3
T
T
T
F
F
F
Further
Processing
Reject Sub−window

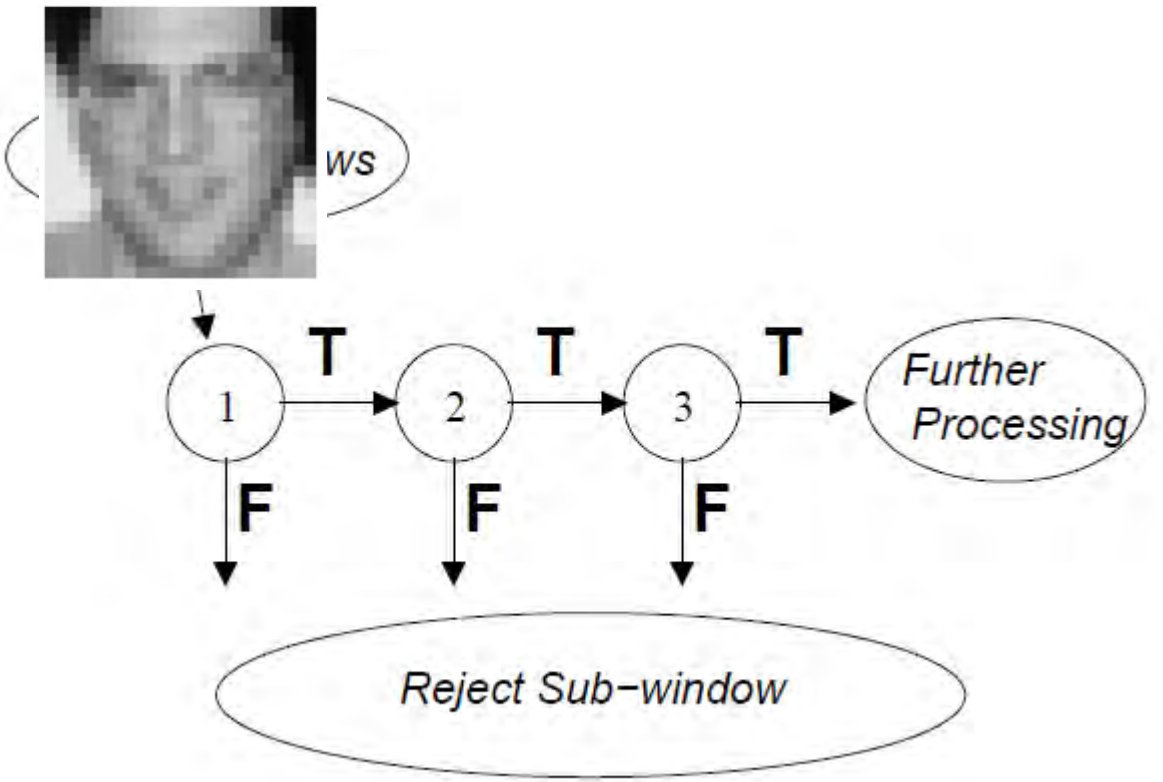
Viola-Jones Face Detector: Results

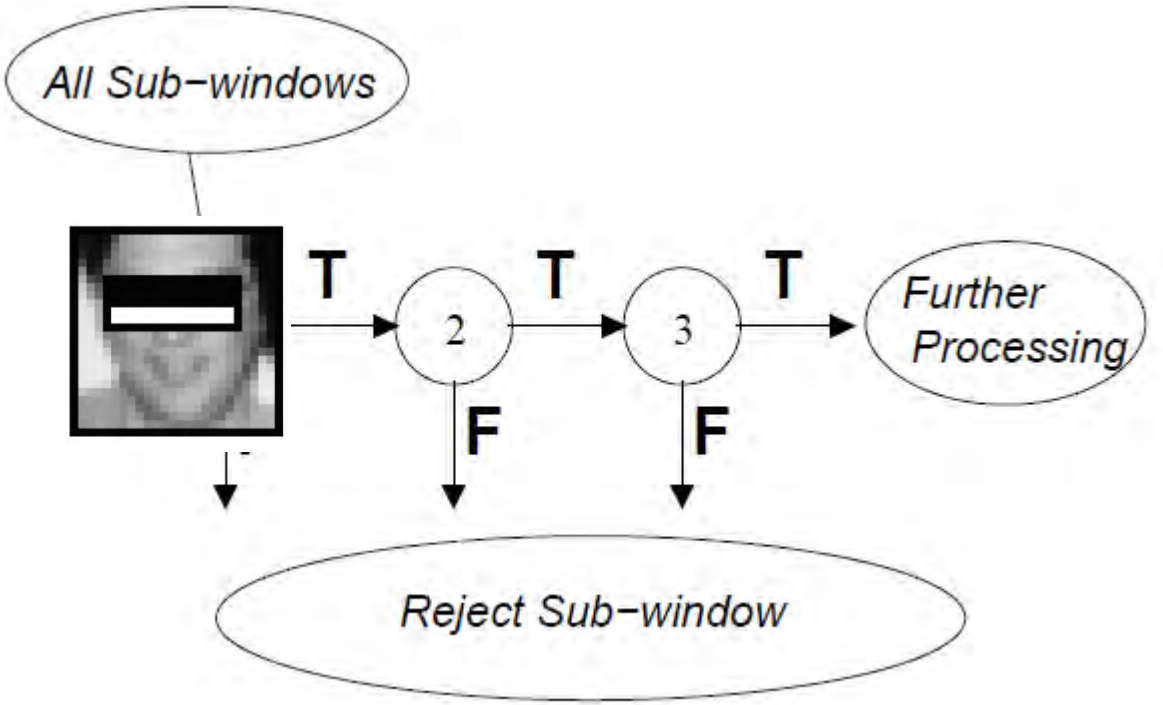


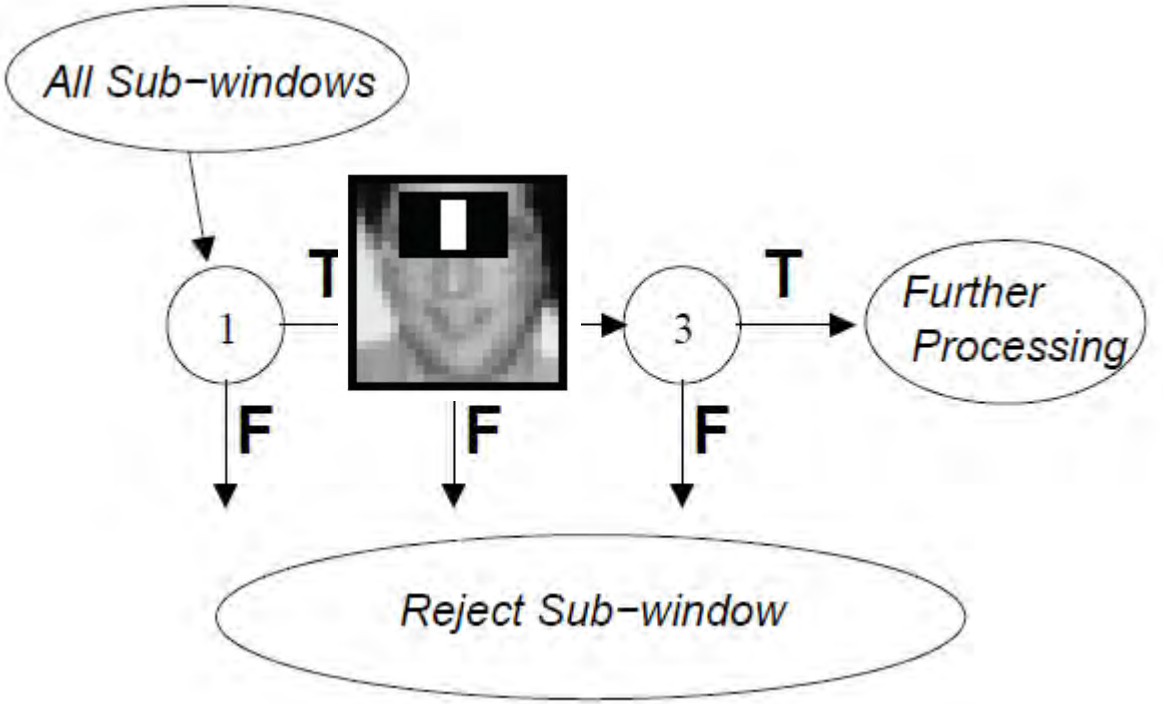
First two features selected

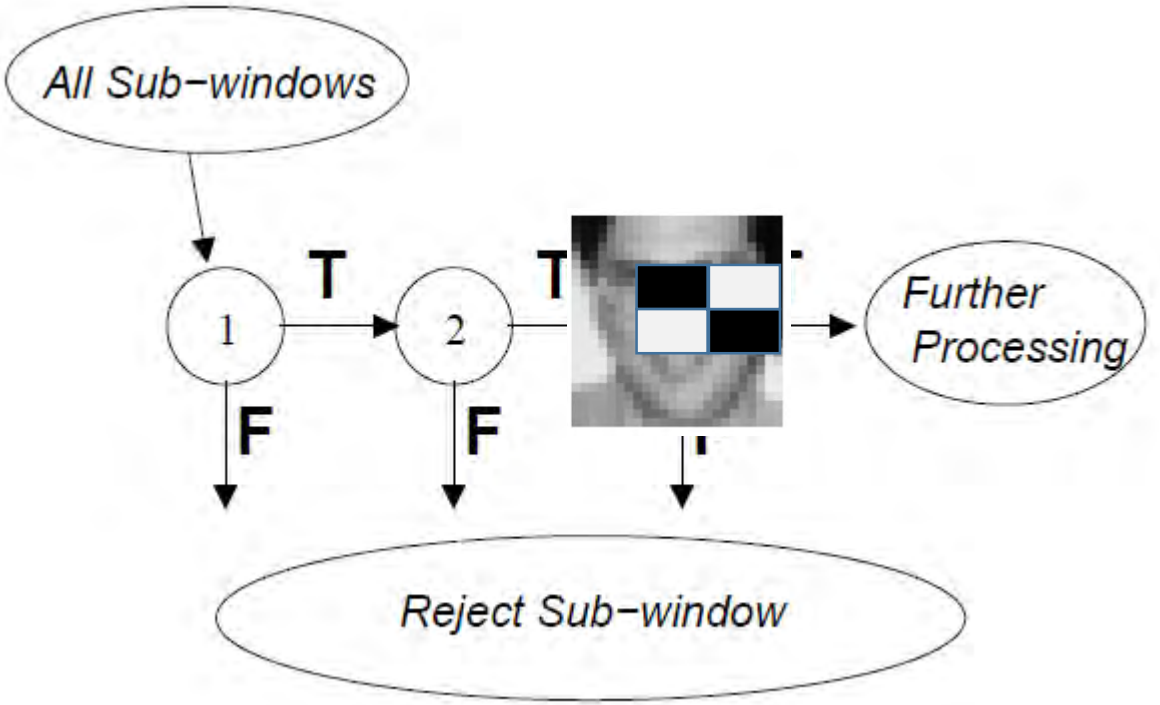
59

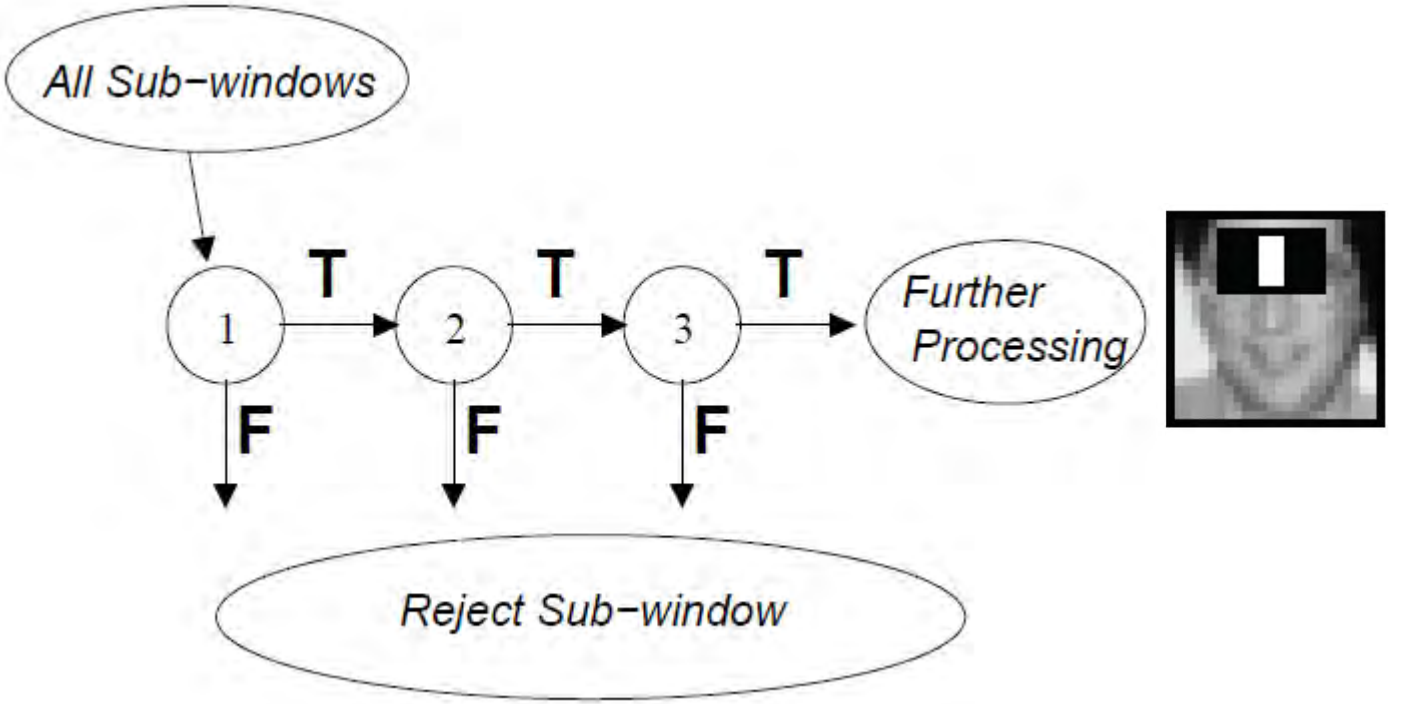

All Sub−windows
1
2
3
T
T
T
F
F
F
Further
Processing
Reject Sub−window

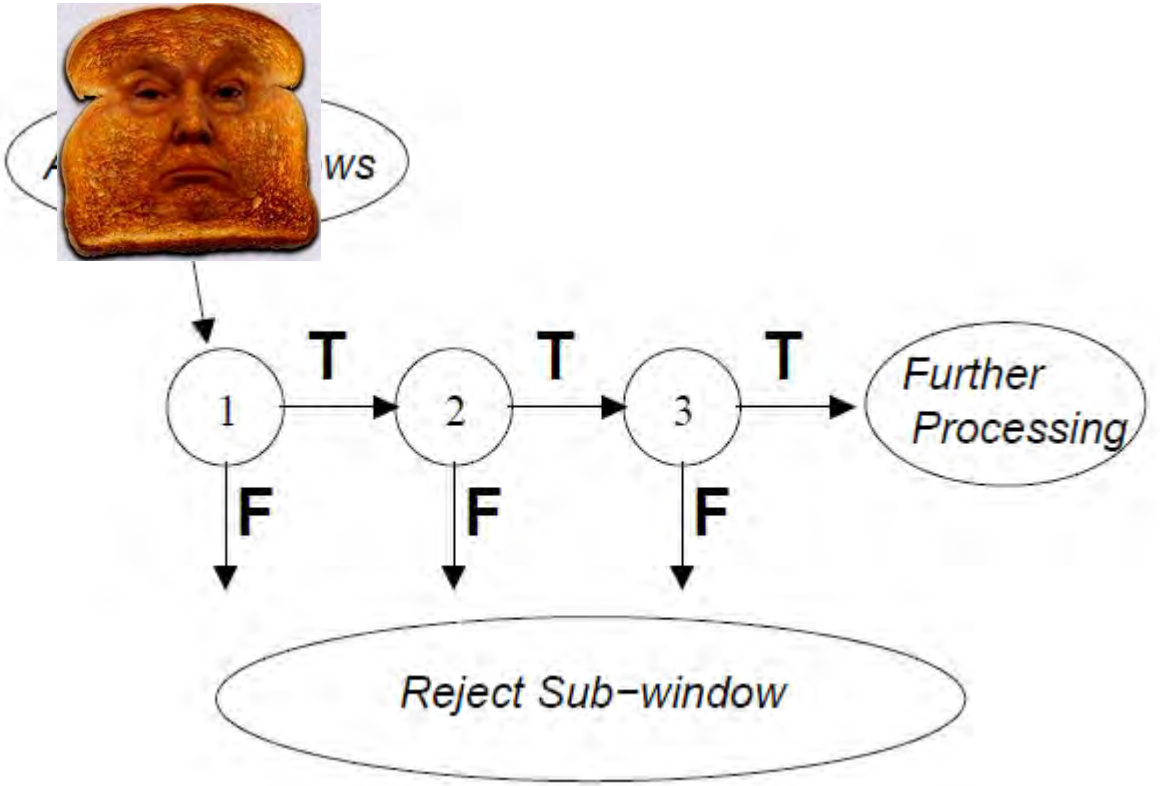

ws
1
2
3
T
T
T
F
F
F
Further
Processing
Reject Sub−window

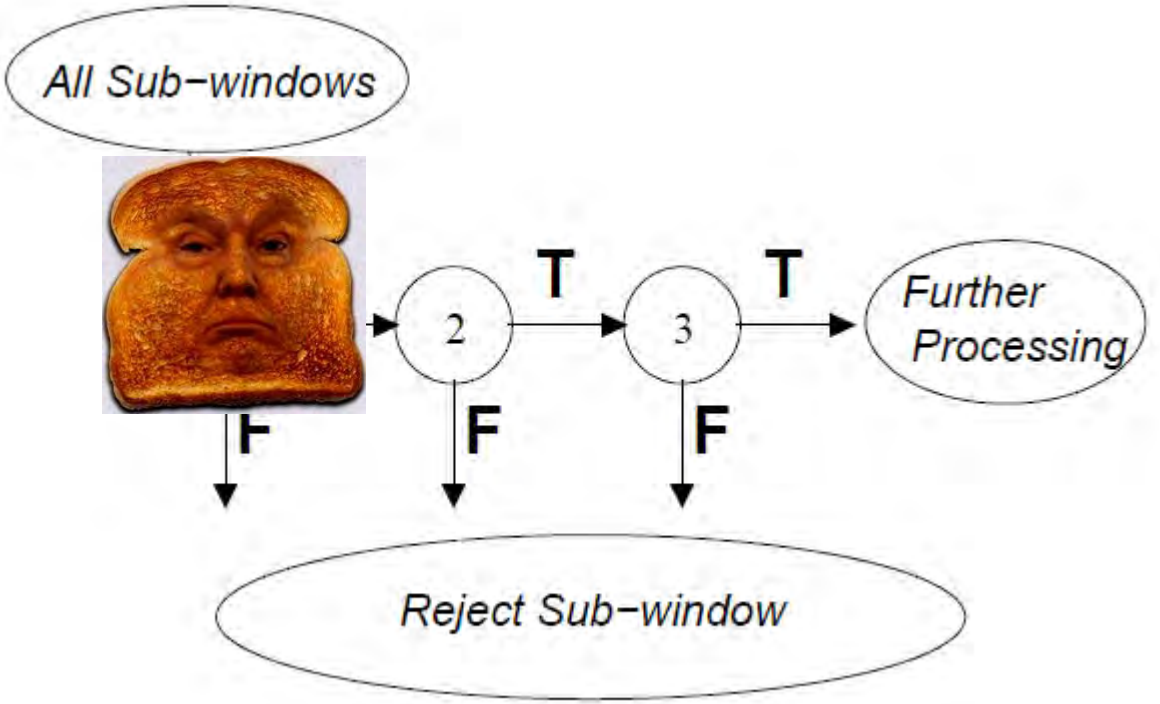

All Sub−windows
T
2
T
3
T
Further
Processing
F
F
Reject Sub−window

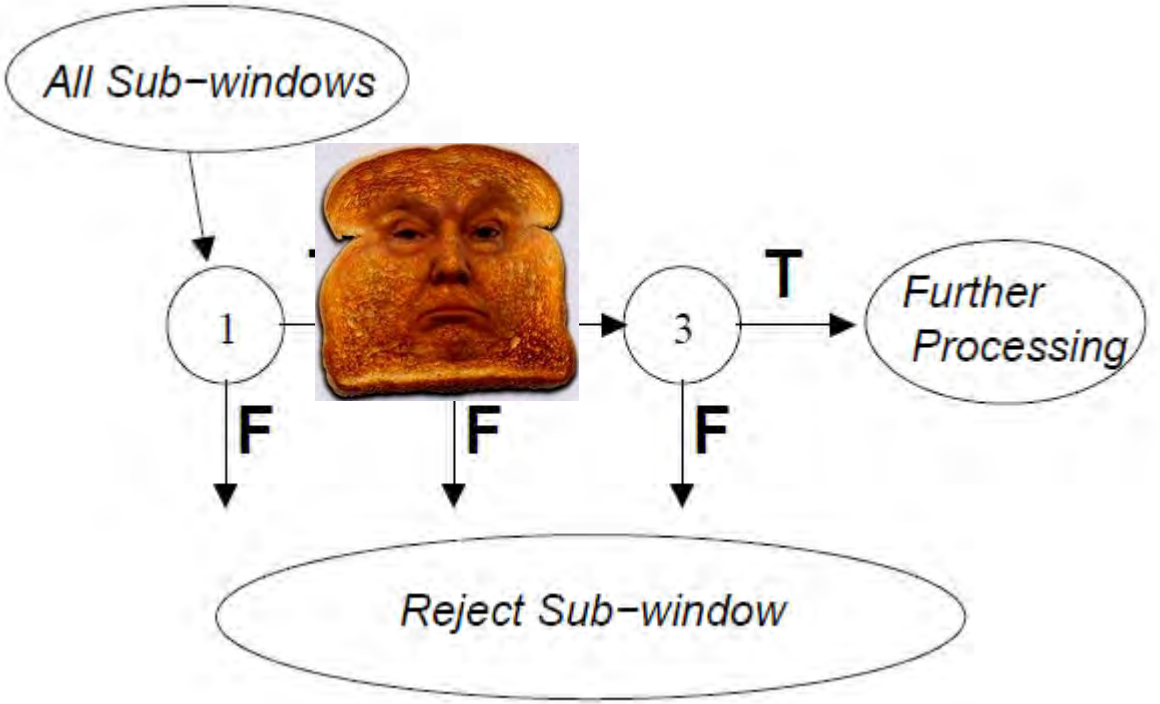

All Sub-windows
1
T
F
3
T
F
F
Further Processing
Reject Sub-window

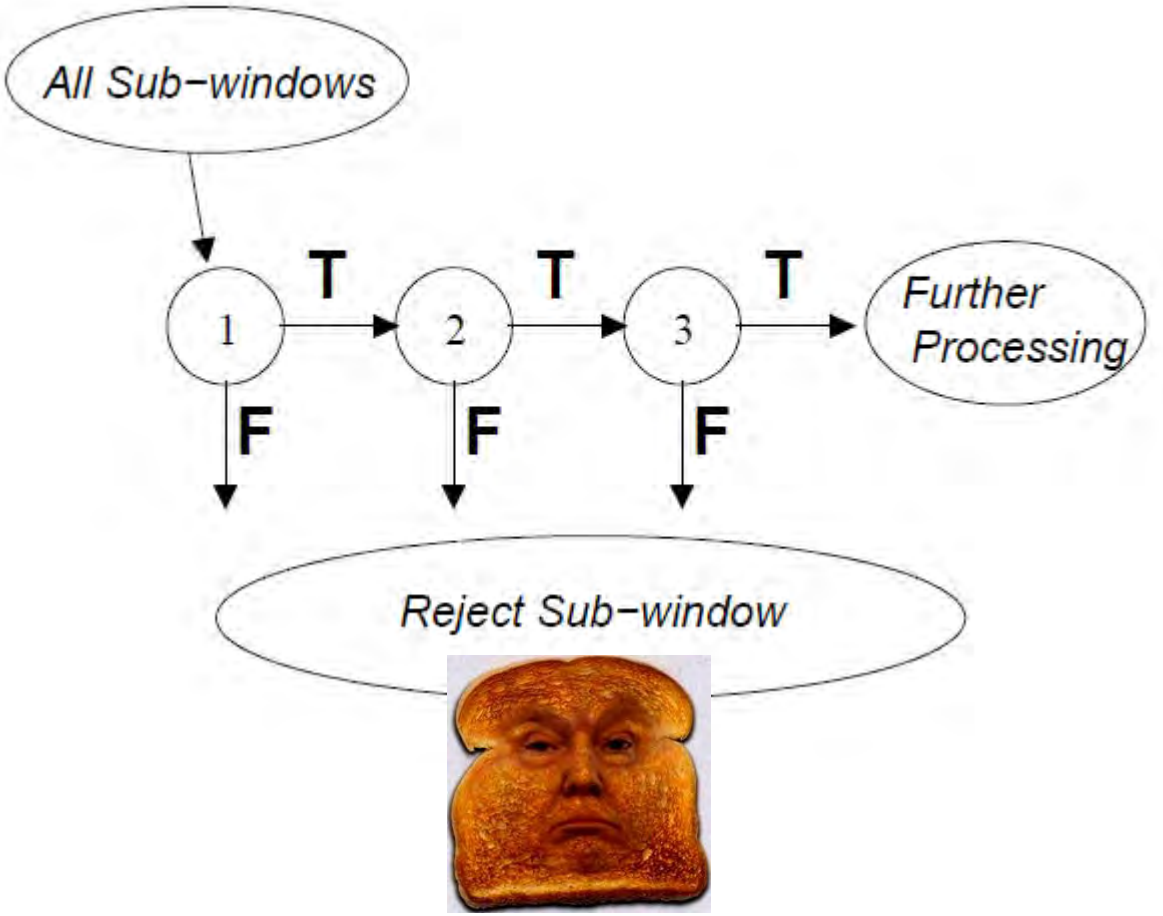

All Sub-windows
1
T
2
T
F
F
F
Further Processing
Reject Sub-window


All Sub-windows
1
2
3
T
T
T
F
F
F
Further Processing
Reject Sub-window


ws
1
T
2
T
3
T
Further
Processing
F
F
F
Reject Sub−window


All Sub−windows
2
T
3
T
Further
Processing
F
F
F
Reject Sub−window


All Sub−windows
1
3
T
Further
Processing
F
F
F
Reject Sub−window


All Sub−windows
1
2
3
T
T
T
F
F
F
Further Processing
Reject Sub−window

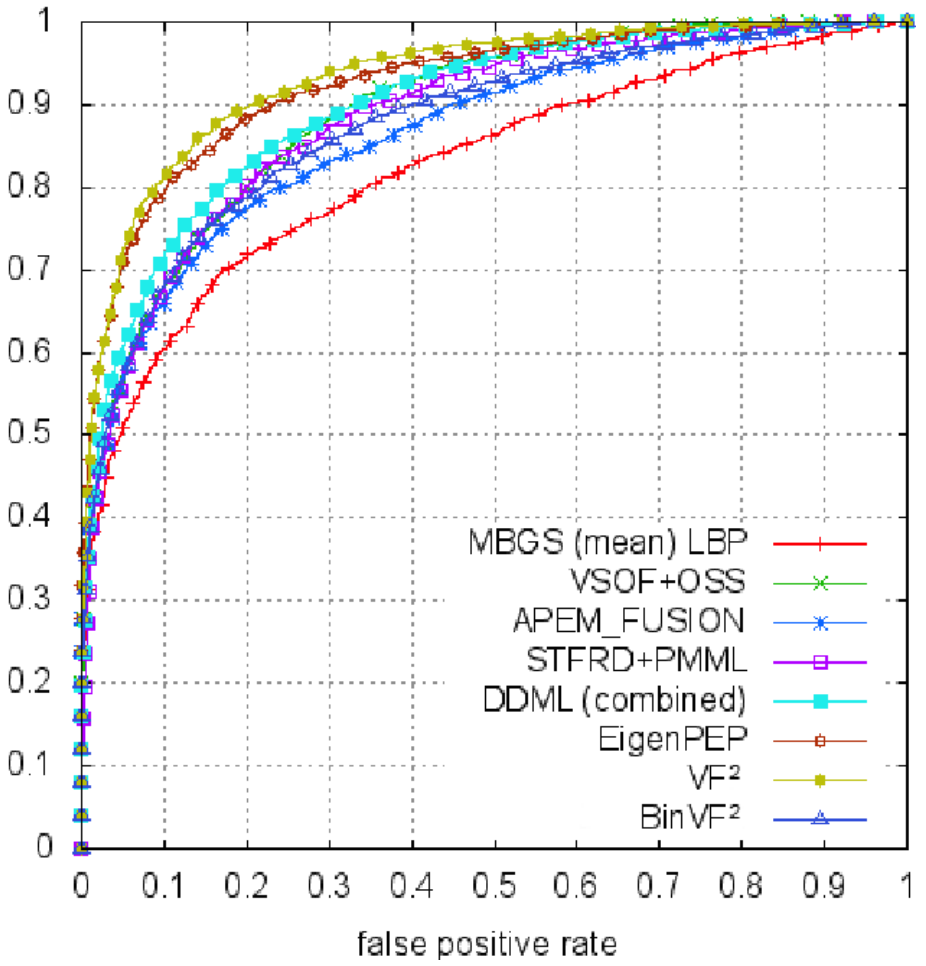
Possible outcomes of a classifier

	is face	is NOT face
says face	TP	FP
Says NOT face	FN	TN

TP : true positive
FP : false positive
TN : true negative
FN : false negative

$$\mathrm{TP} + \mathrm{FN} = 1.0$$

$$\mathrm{FP} + \mathrm{TN} = 1.0$$



ROC curve for face recognition.

Assuming tests $i = 1, \ldots, n$ are independent:

$$\text{Probability of true face passing test } i = \text{TP}_i$$

$$\text{Probability of true face passing all tests} = \text{TP}_{\text{all}} = \prod_i \text{TP}_i$$

$$\text{Probility of false face passing test } i = \text{FP}_i$$

$$\text{Probability of false face passing all tests} = \text{FP}_{\text{all}} = \prod_i \text{FP}_i$$

Assuming tests $i = 1, \ldots, n$ are independent:

$$\text{Probability of true face passing test } i = \text{TP}_i$$

$$\text{Probability of true face passing all tests} = \text{TP}_{\text{all}} = \prod_i \text{TP}_i$$

$$\text{Probility of false face passing test } i = \text{FP}_i$$

$$\text{Probability of false face passing all tests} = \text{FP}_{\text{all}} = \prod_i \text{FP}_i$$

We want TP_{all} to be large, and FP_{all} small.
For example, suppose that

$$\begin{aligned} \text{TP}_i &= 0.999, \quad \text{FN}_i = 0.001 \\ \text{FP}_i &= 0.95, \quad \text{TN}_i = 0.05 \end{aligned}$$

and suppose there are 100 tests. Then

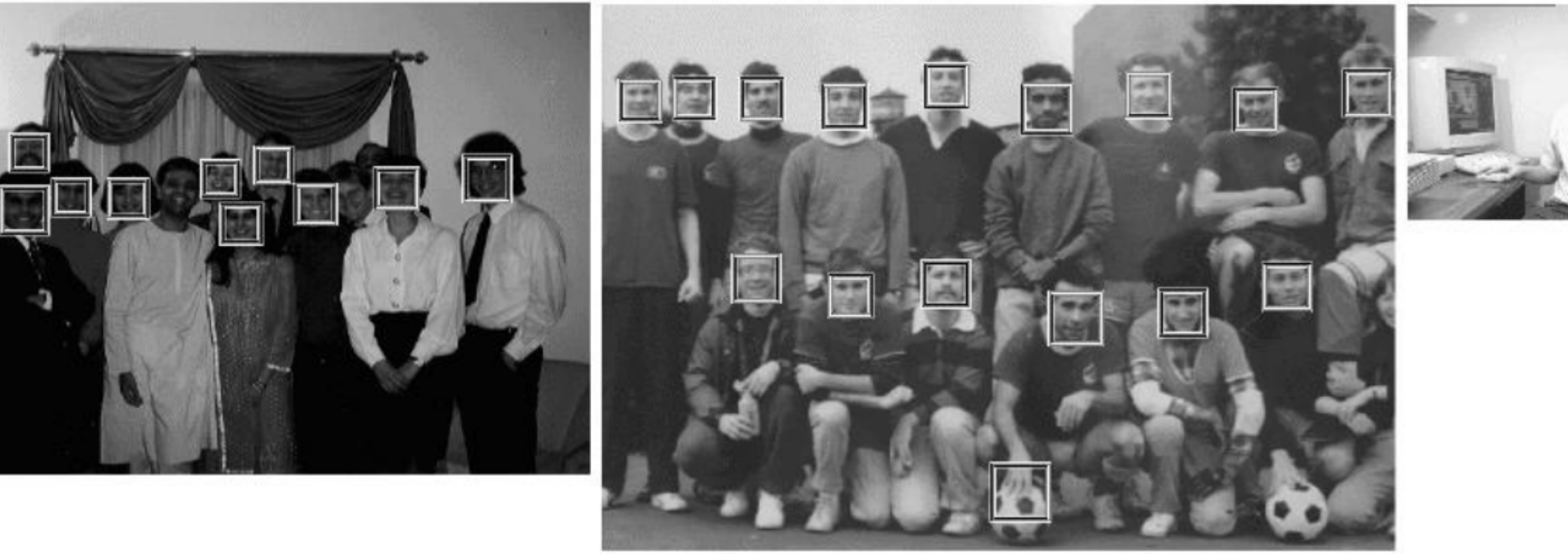
$$\begin{aligned} \text{TP}_{\text{all}} &= 0.999^{100} &&= 0.904 \\ \text{FP}_{\text{all}} &= 0.95^{100} &&= 0.006 \end{aligned}$$

Test is relatively cheap on negative examples.

Expected number of tests applied to a non-face is

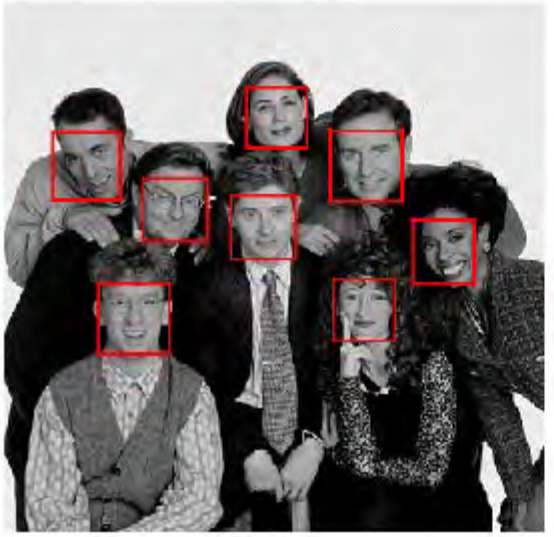
$$E[\text{number tests}] = \frac{\text{FP}_i}{\text{TN}_i}$$

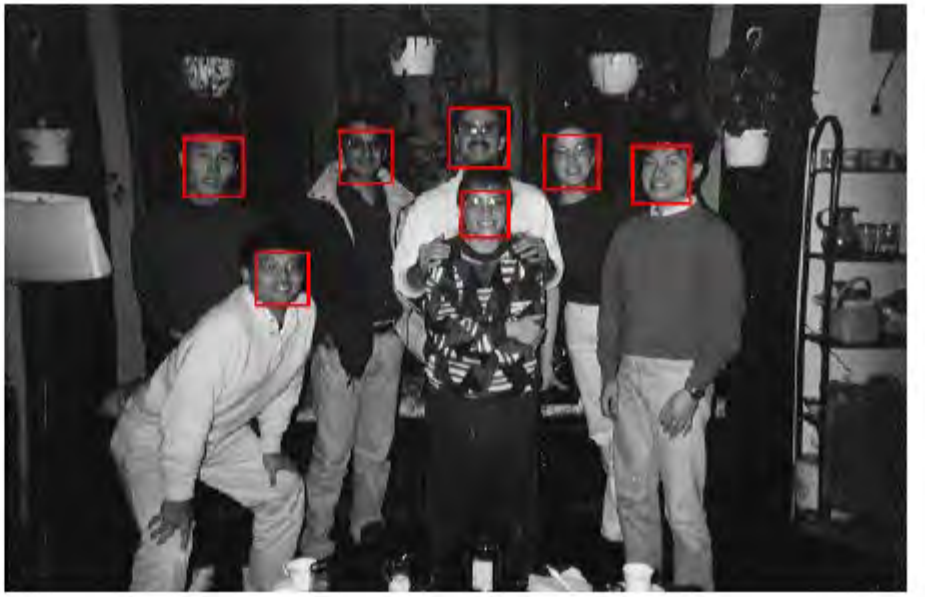
which is 19 if $\text{FP}_i = 0.95$.



re 7: Output of our face detector on a number of test images from the MIT+CMU tes

Viola-Jones Face Detector: Results





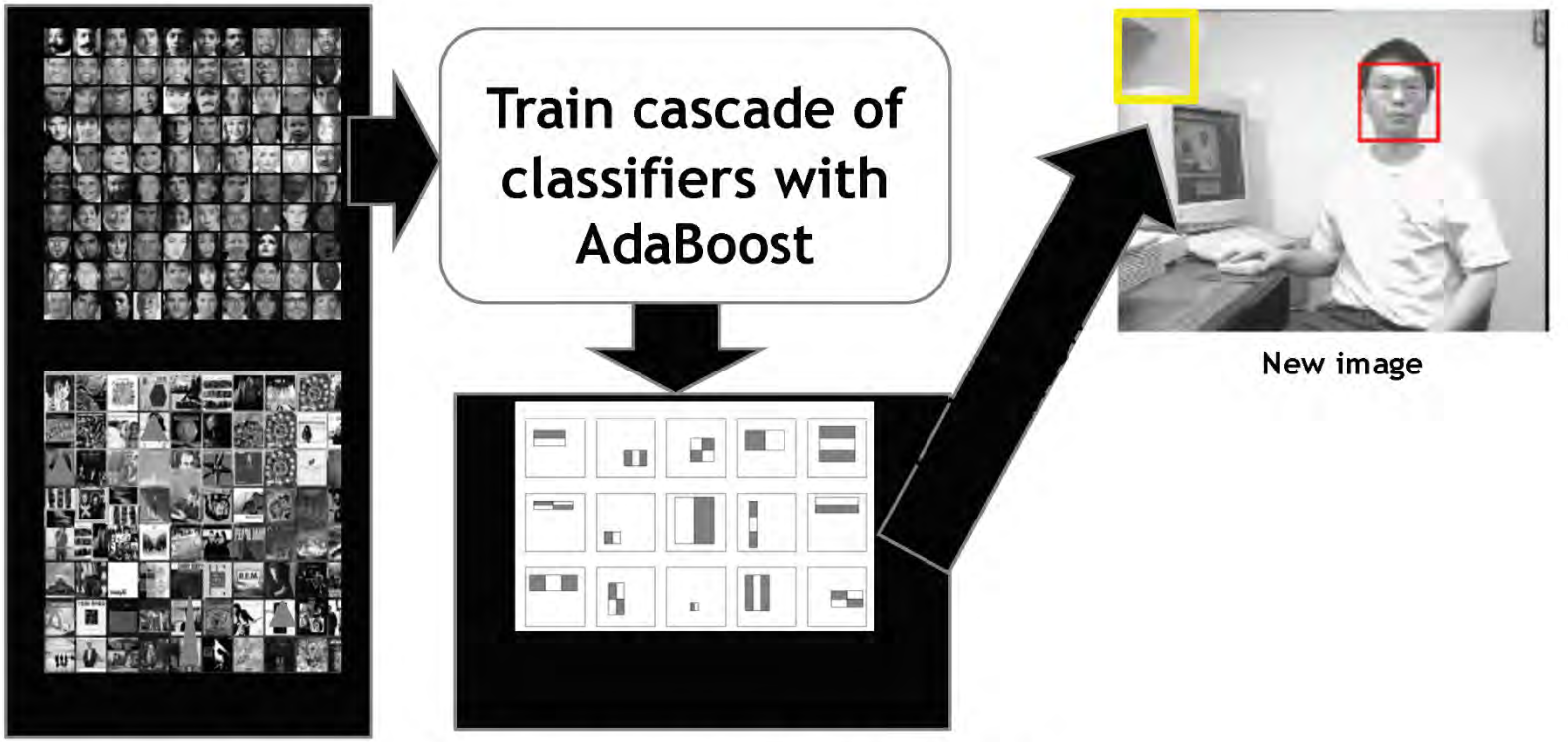
Speed of face detector (2001)

- Speed is proportional to the average number of features computed per sub-window.
- On the MIT+CMU test set, an average of 9 features (/ 6061) are computed per sub-window.
- On a 700 Mhz Pentium III (2001 PC), a 384x288 pixel image takes about 0.067 seconds to process (15 fps).
- Roughly 15 times faster than Rowley-Baluja-Kanade and 600 times faster than Schneiderman-Kanade.

Summary (Viola-Jones)

- Fastest known real-time face detector for gray images (in 2001)
- Three contributions with broad applicability:
  - ❖Cascaded classifier yields rapid classification
  - ❖AdaBoost as an extremely efficient feature selector
  - ❖Rectangle Features + Integral Image can be used for rapid image analysis

Viola-Jones Face Detector: Summary



- Train with 5K positives, 350M negatives
- Real-time detector using 38 layer cascade
- 6061 features in final layer
- [Implementation available in OpenCV: http://www.intel.com/technology/computing/opencv/]

61

Constructing the classifier

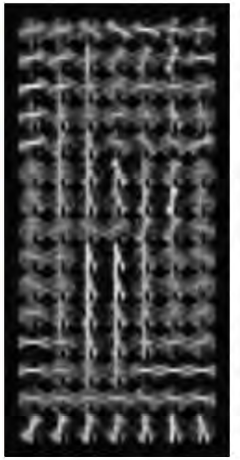
- For each round of boosting:
- Evaluate each rectangle filter on each example
- Sort examples by filter values
- Select best threshold for each filter (min error)
- Use sorting to quickly scan for optimal threshold
- Select best filter/threshold combination
- Weight is a simple function of error rate
- Reweight examples

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

Table 1: The boosting algorithm for learning a query online. T hypotheses are constructed each using a single feature. The final hypothesis is a weighted linear combination of the T hypotheses where the weights are inversely proportional to the training errors.

Linear SVM and HOG for pedestrian detection

Person detection with HoG's & linear SVM's

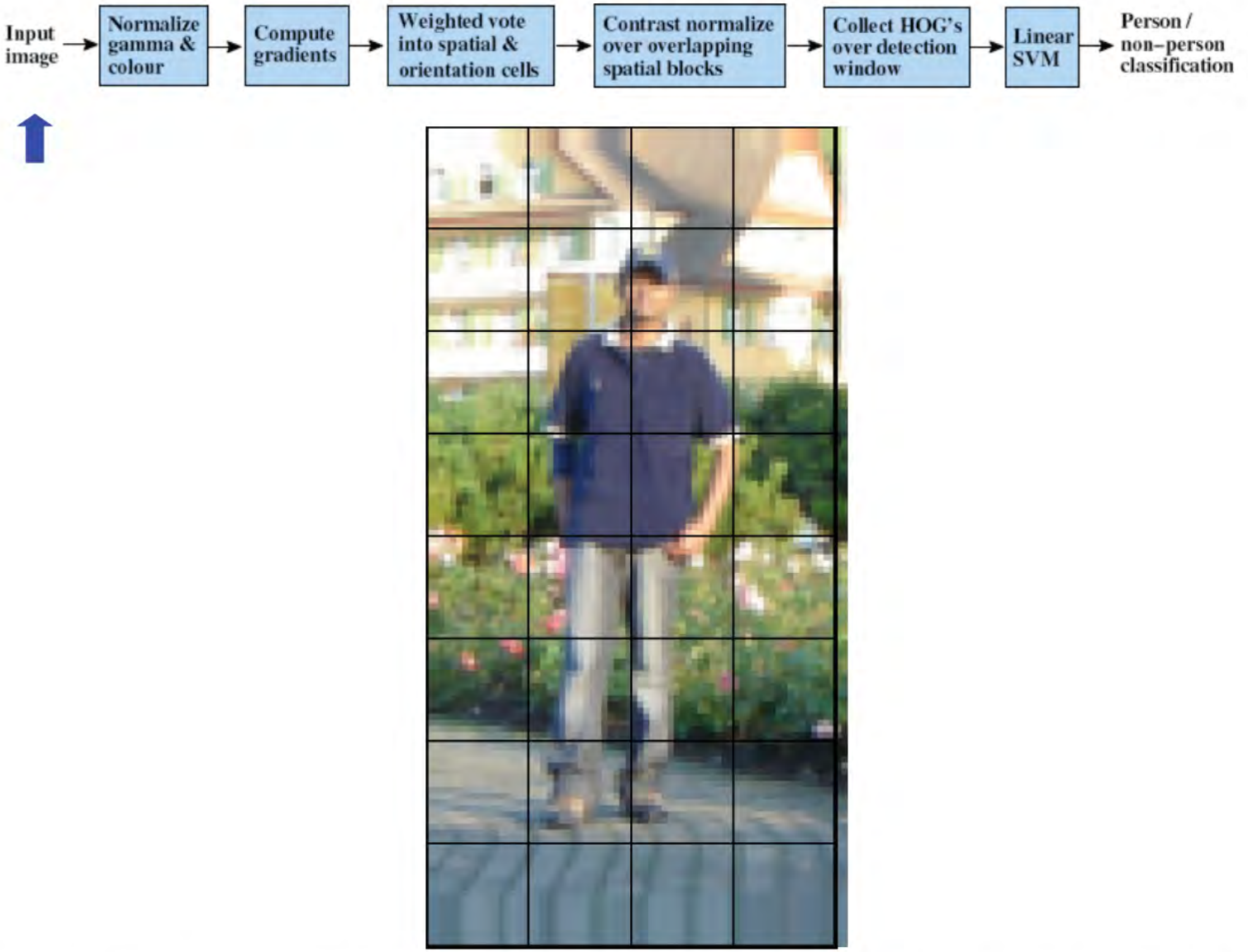


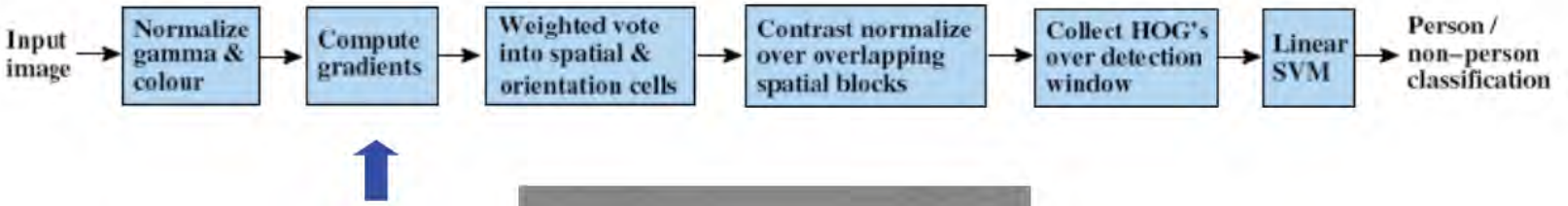
- Histogram of oriented gradients (HoG): Map each grid cell in the input window to a histogram counting the gradients per orientation.
- Train a linear SVM using training set of pedestrian vs. non-pedestrian windows.

Dalal & Triggs, CVPR 2005

Person detection with HoGs & linear SVMs

- Histograms of Oriented Gradients for Human Detection, Navneet Dalal, Bill Triggs, International Conference on Computer Vision & Pattern Recognition - June 2005
- http://lear.inrialpes.fr/pubs/2005/DT05/


Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non-person classification



-1	0	1

centered

-1	1

uncentered

1	-8	0	8	-1

cubic-corrected

0	1
-1	0

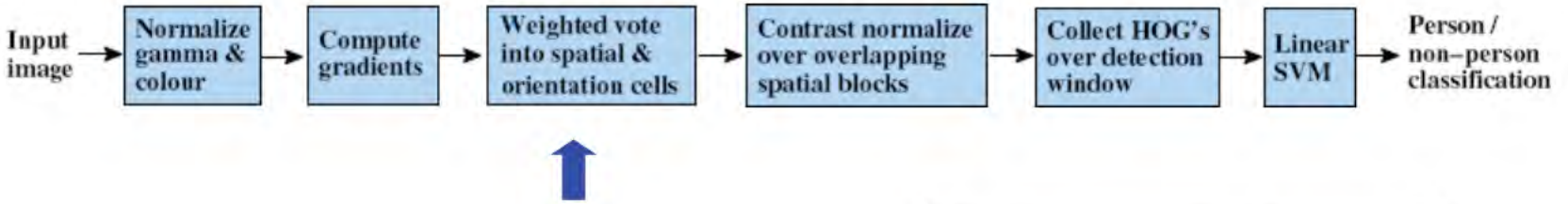
diagonal

-1	0	1
-2	0	2
-1	0	1

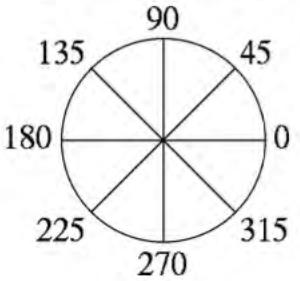
Sobel

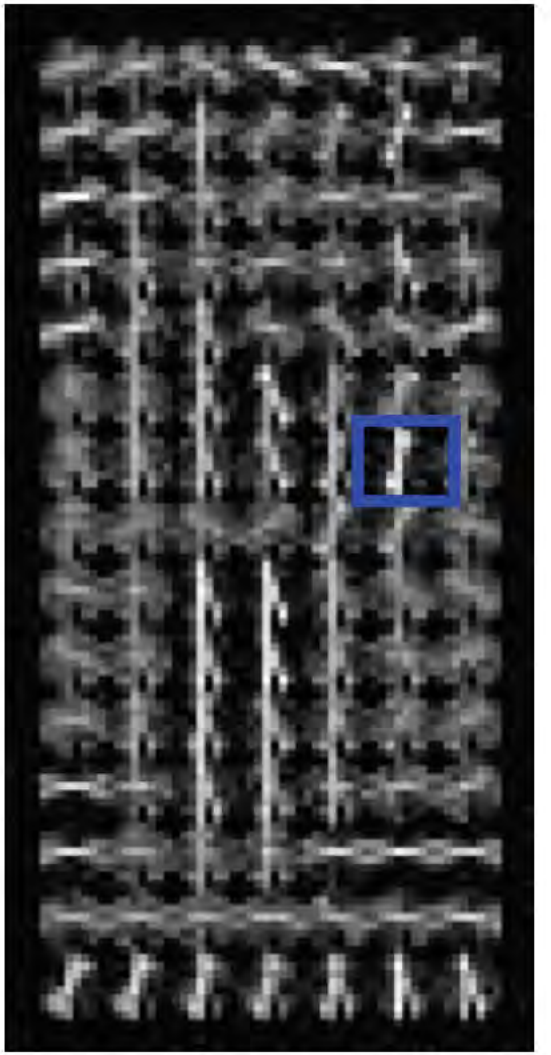
Slides by Pete Barnum

Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05



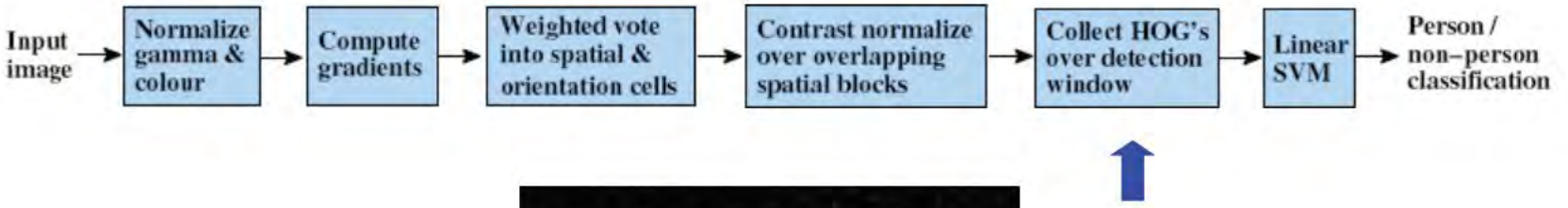
- Histogram of gradient orientations
  - -Orientation



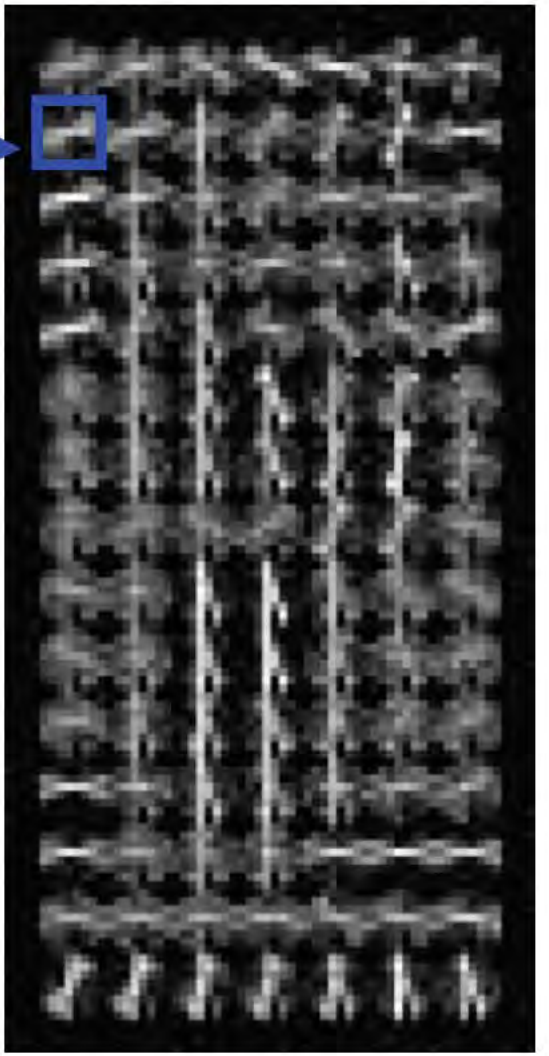


Slides by Pete Barnum

Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05



8 orientations



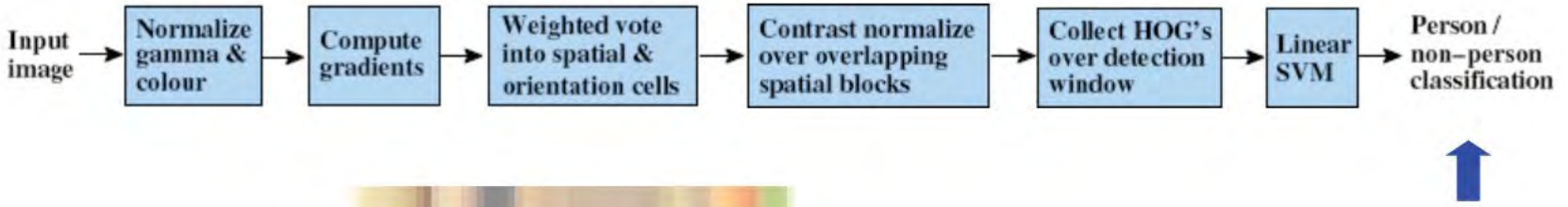
X=

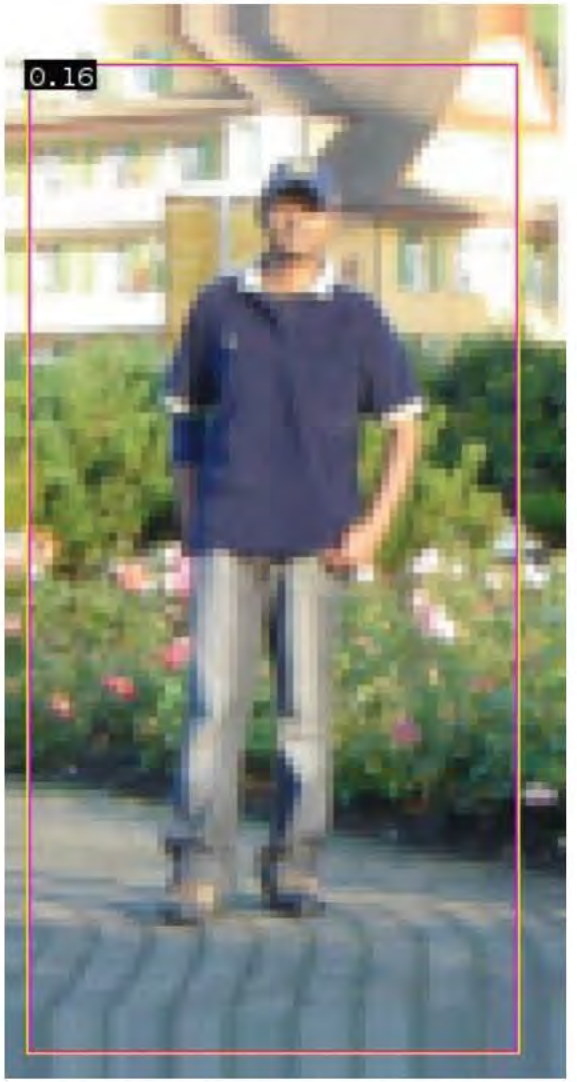
$\in R^{840}$

15x7 cells

Slides by Pete Barnum

Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05





$$0.16 = w^T x - b$$

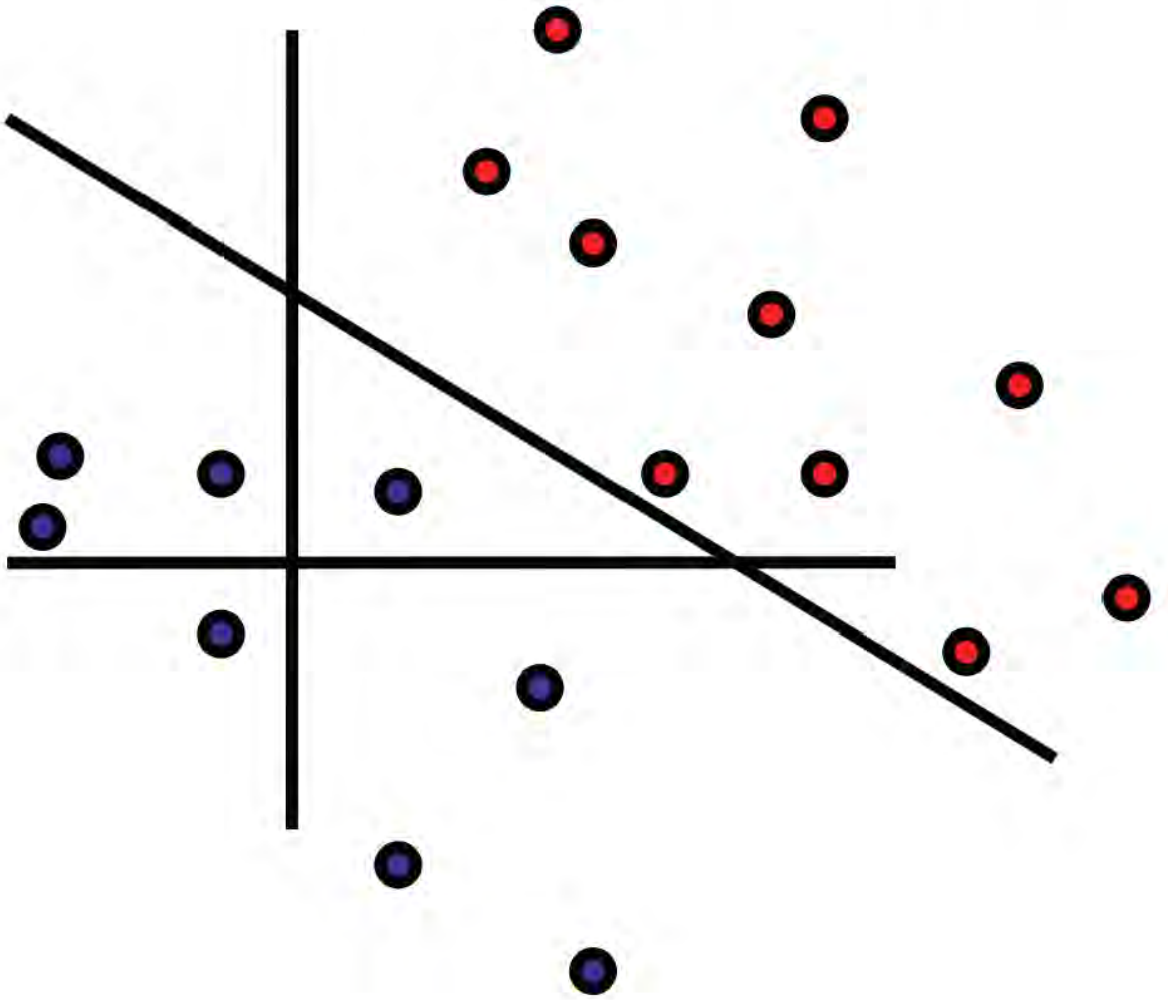
$$sign(0.16) = 1$$

$$=>$$ pedestrian

Slides by Pete Barnum

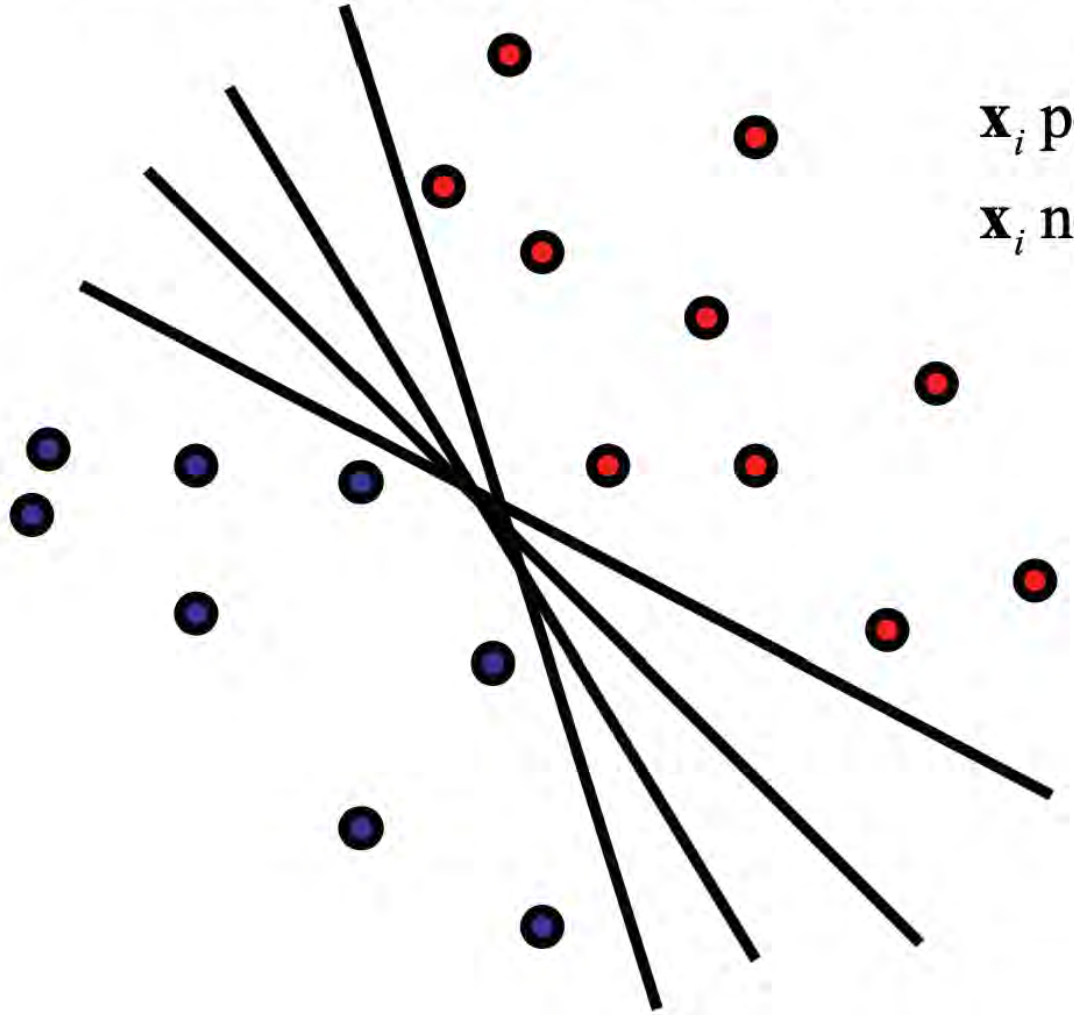
Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05

Linear classifiers



Linear classifiers

- Find linear function to separate positive and negative examples

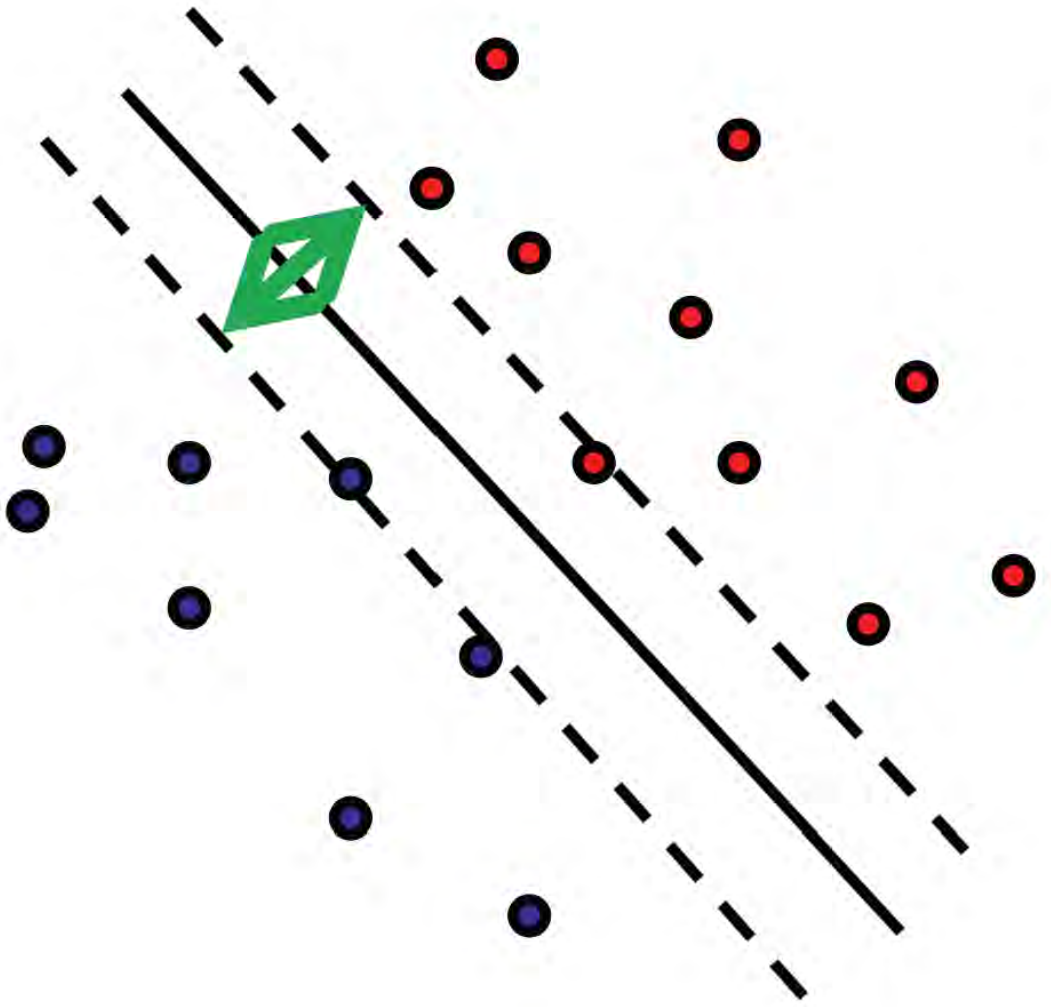


$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$

Which line is best?

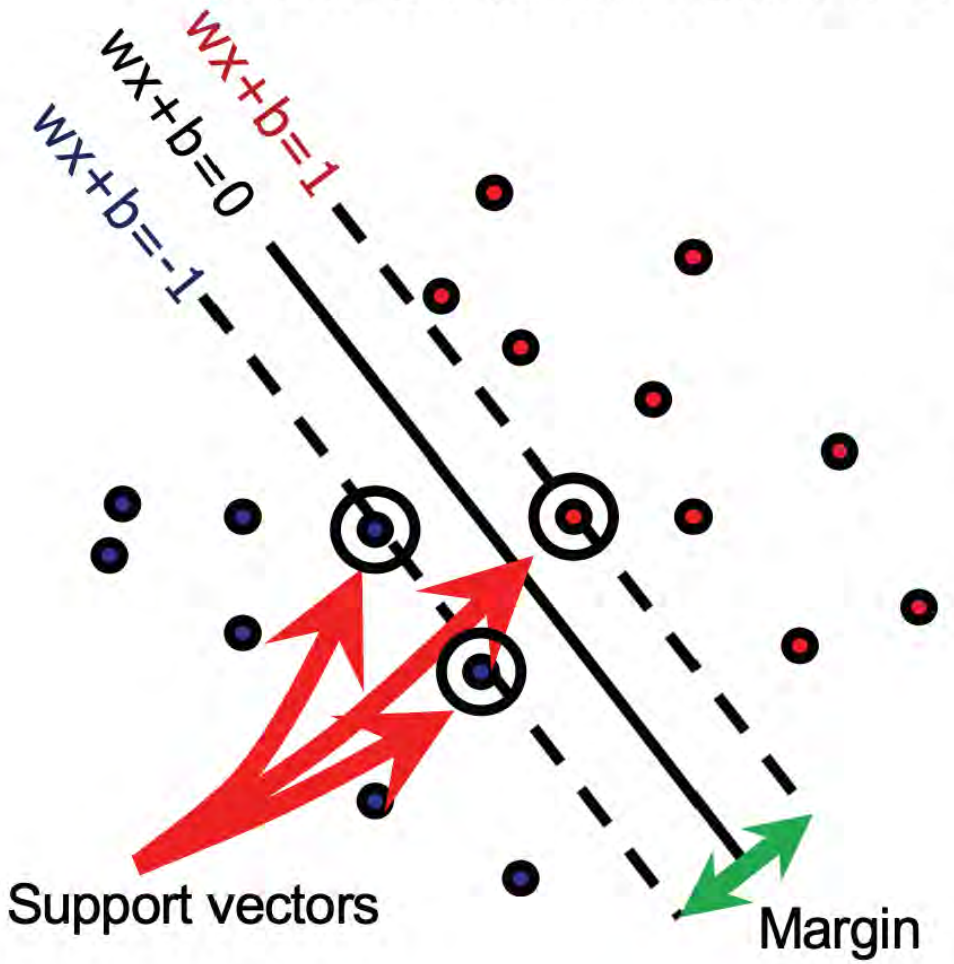
Support Vector Machines (SVMs)



- Discriminative classifier based on *optimal separating line (for 2d case)*
- Maximize the *margin* between the positive and negative training examples

Support vector machines

- Want line that maximizes the margin.



$\mathbf{x}_i$ positive $(y_i = 1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

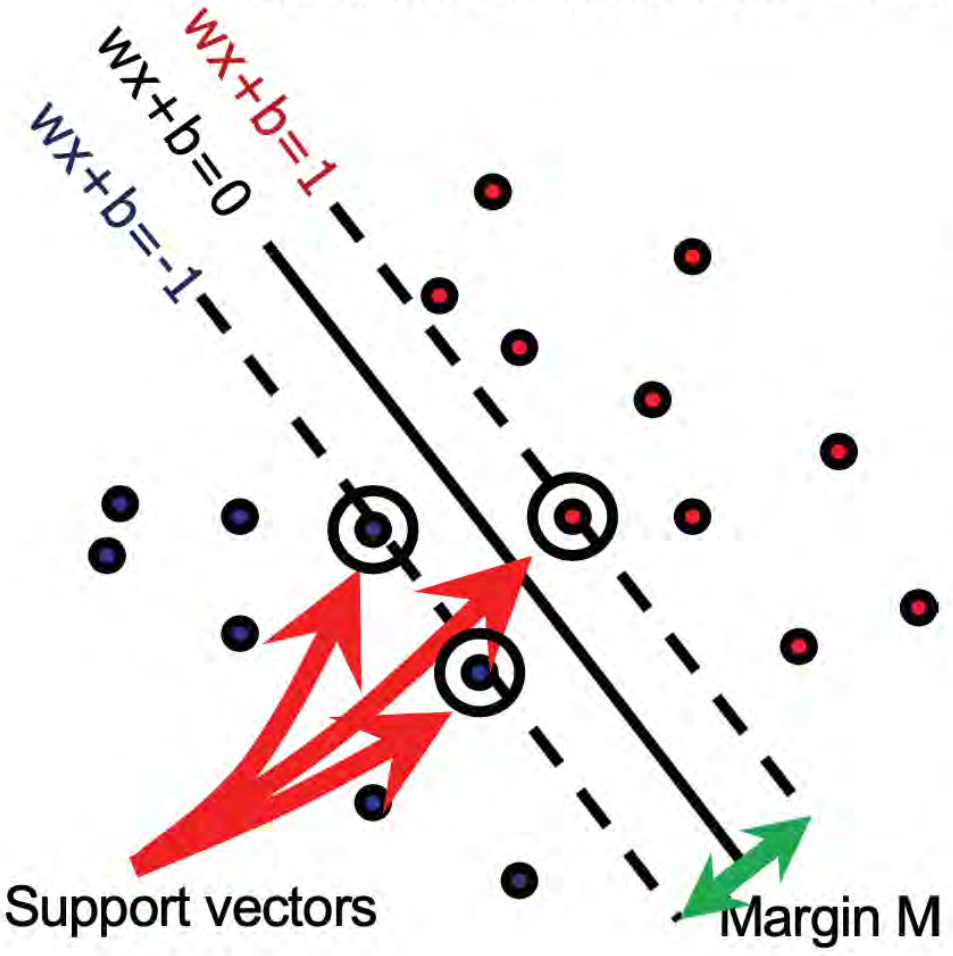
$\mathbf{x}_i$ negative $(y_i = -1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

Support vector machines

- Want line that maximizes the margin.



$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

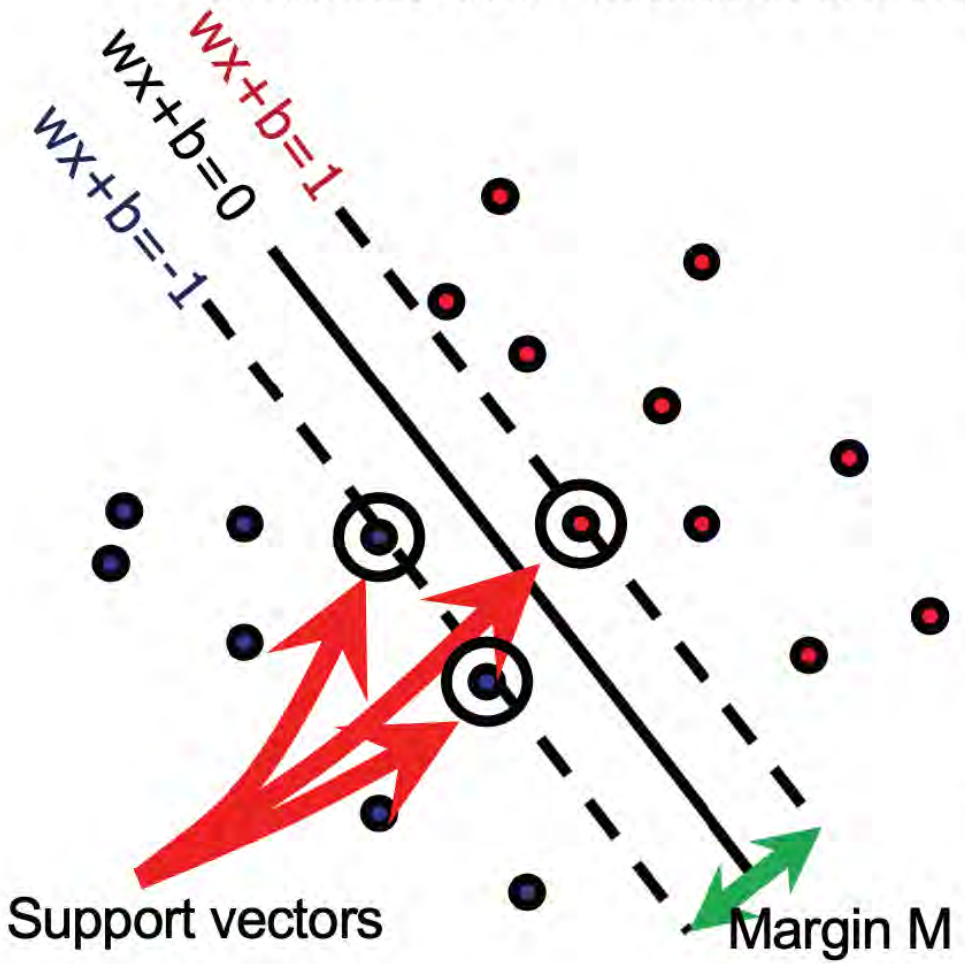
Distance between point and line: $\quad \dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

For support vectors:

$$\frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|} = \frac{\pm 1}{\|\mathbf{w}\|} \qquad M = \left| \frac{1}{\|\mathbf{w}\|} - \frac{-1}{\|\mathbf{w}\|} \right| = \frac{2}{\|\mathbf{w}\|}$$

Support vector machines

- Want line that maximizes the margin.



$\mathbf{x}_i$ positive $(y_i = 1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

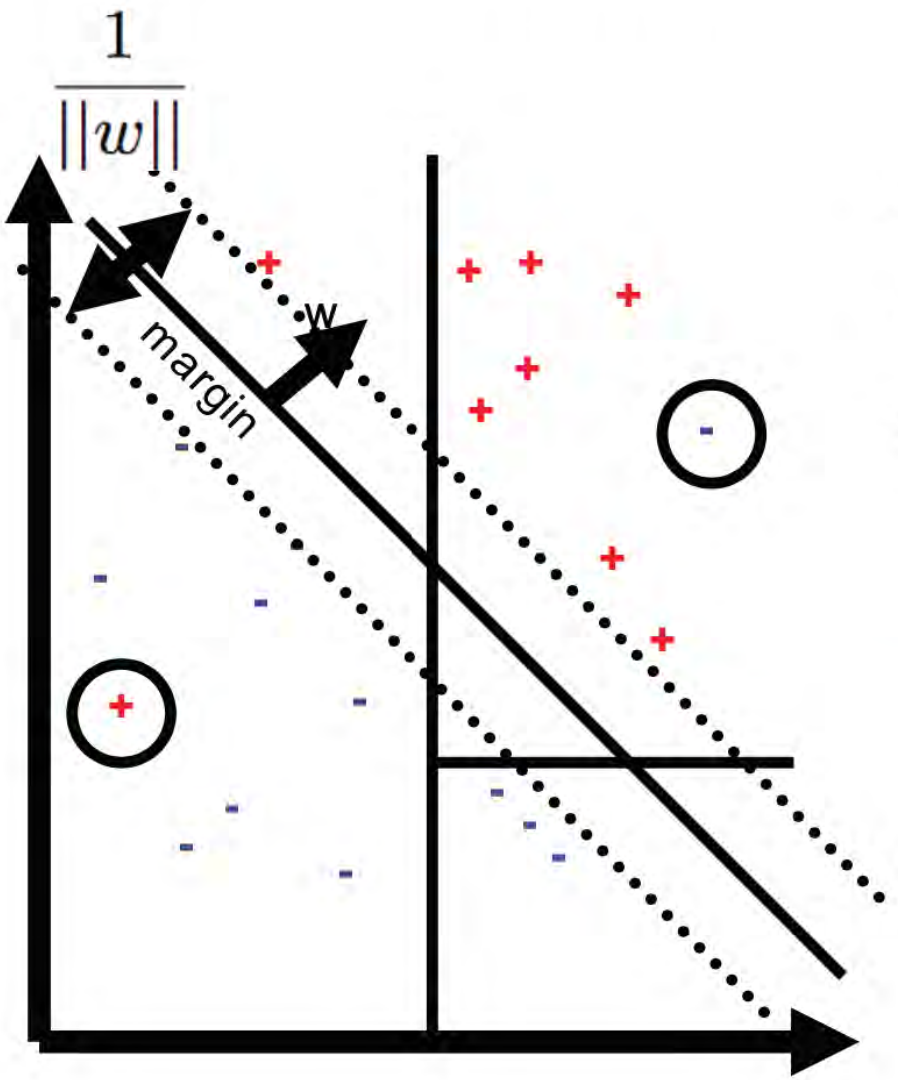
$\mathbf{x}_i$ negative $(y_i = -1)$: $\mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Distance between point and line: $\dfrac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

Therefore, the margin is $2 / \|\mathbf{w}\|$

Support vector machines



- Simple decision
- Good classification
- Good generalization

$$\min_{w,\xi} \frac{||w||^2}{2} + C \sum_j \xi_j$$

$$y_j w^T x_j \geq 1 - \xi_j$$

$$\xi_j \geq 0$$

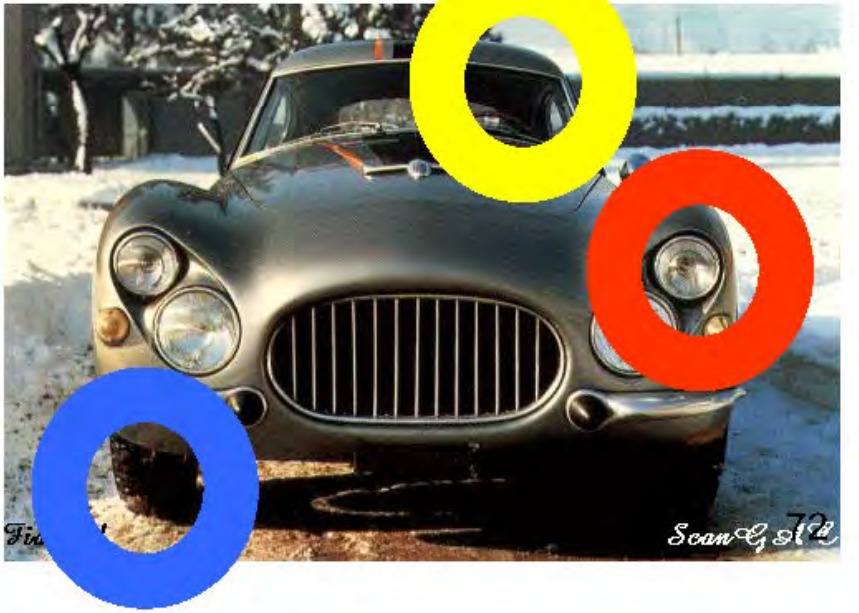
How do I solve the SVM problem?

- It's a convex optimization problem
  - Can solve in Matlab (don't)
- Download from the web
  - SMO: Sequential Minimal Optimization
  - SVM-Light http://svmlight.joachims.org/
  - LibSVM http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - LibLinear http://www.csie.ntu.edu.tw/~cjlin/liblinear/
  - SVM-Perf http://svmlight.joachims.org/
  - Pegasos http://ttic.uchicago.edu/~shai/

Bag of Words model for object detection

Adapted from slides by Rob Fergus

Different appearance, similar parts



Object → Bag of 'words'

Bag-of-words / Video google

Slides provided by Andrew Zisserman

Retrieve image/key frames containing the same object

query

?

Approach

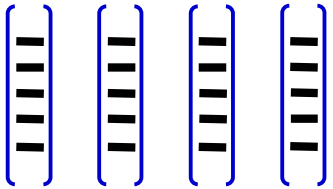
- Use image matching engine based on SIFT descriptors and geometric consistency

- Rank images based on # correspondences from query to image

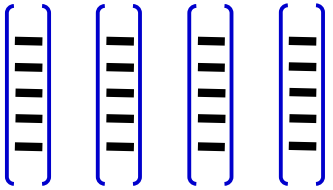
Outline of retrieval algorithm

all images

invariant descriptor vectors



invariant descriptor vectors

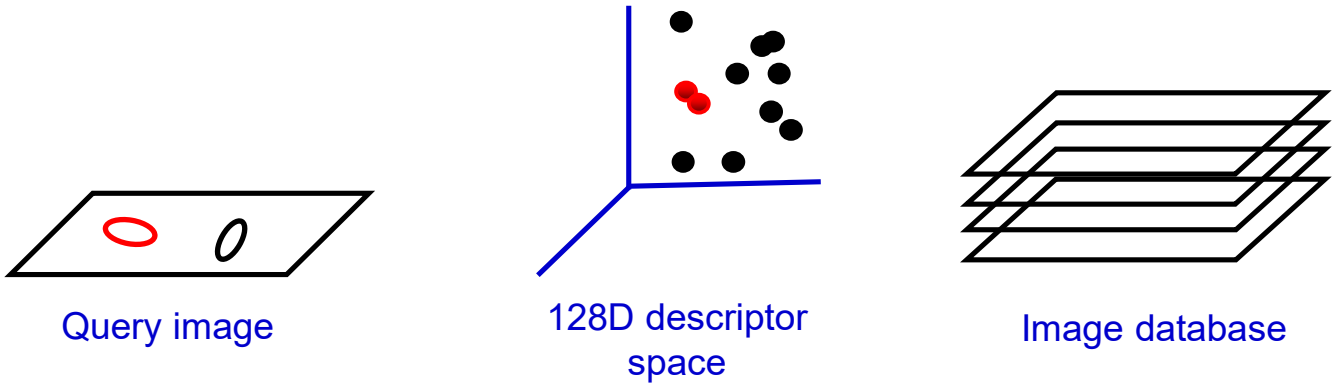


1. Compute regions in each image independently
2. “Label” each region by a SIFT descriptor vector
3. Find corresponding regions by matching to closest descriptor vector
4. Verify matches using a geometric transformation
5. Score images by # of correspondences

Finding corresponding regions cast as finding nearest neighbour vectors

Finding nearest neighbour vectors

Establish correspondences between object model image and images in the database by **nearest neighbour matching** on SIFT vectors



Query image

128D descriptor space

Image database

Solve following problem for all feature vectors, $\mathbf{x}_j \in \mathcal{R}^{128}$, in the query image:

$$\forall j \; NN(j) = \arg\min_i ||\mathbf{x}_i - \mathbf{x}_j||$$

where, $\mathbf{x}_i \in \mathcal{R}^{128}$, are features from all the database images.

How to search over a large number of images?

- So far, we have a strategy for matching a query image to a handful of target image using local features.

- How to generalize this strategy to many target images with reasonable complexity?
  - 10, 10^2, 10^3, …, 10^7, … 10^{10} images?

- Goal: 1 s on a single machine

- Issues:
  - memory footprint
  - matching cost (time)
  - precision and recall

Quick look at the complexity of the NN-search

N … images

M … regions per image (~1000)

D … dimension of the descriptor (~128)

Exhaustive linear search: O(M NMD)

Example:

- Matching two images (N=1), each having 1000 SIFT descriptors
  Nearest neighbors search: 0.4 s (2 GHz CPU, implementation in C)
- Memory footprint: 1000 * 128 = 128kB / image

# of images	CPU time	Memory req.
N = 1,000	… ~7min	(~100MB)
N = 10,000	… ~1h7min	(~ 1GB)
…		
N = 10^7	~115 days	(~ 1TB)
…		
All images on Facebook:		
N = 10^{10}	… ~300 years	(~ 1PB)

Nearest-neighbour matching

Solve following problem for all feature vectors, $\mathbf{x_j}$, in the query image:

$$\forall j \; NN(j) = \arg\min_i ||\mathbf{x}_i - \mathbf{x}_j||$$

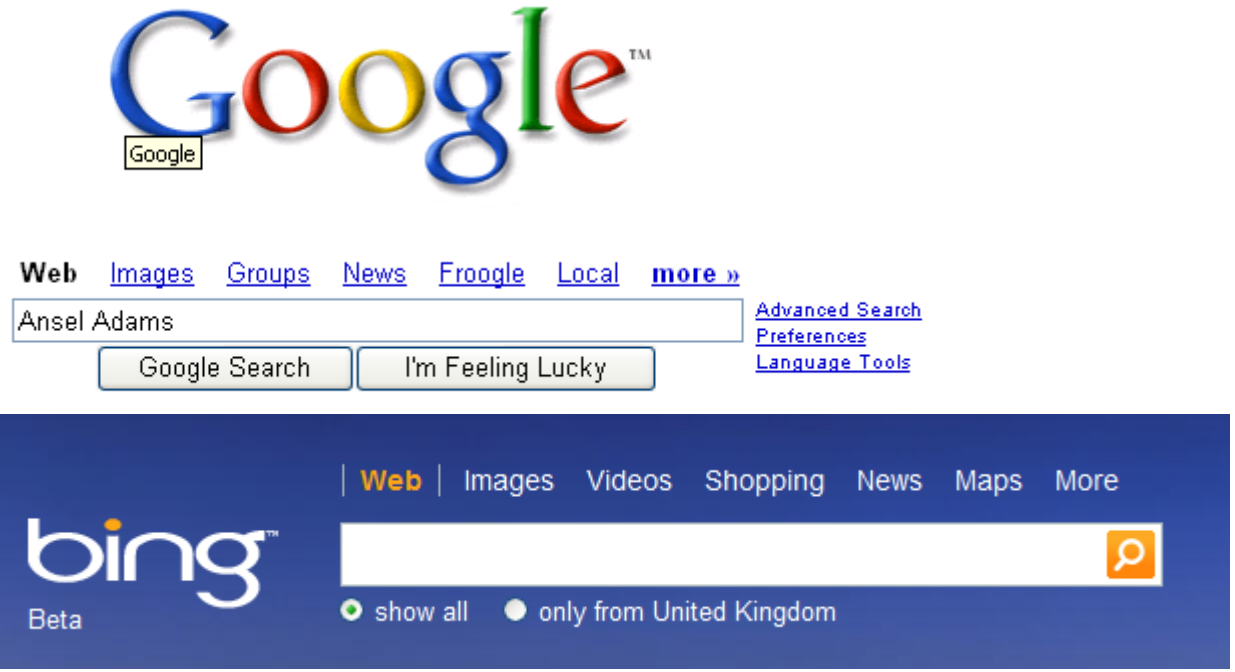
where $\mathbf{x_i}$ are features in database images.

Nearest-neighbour matching is the major computational bottleneck

- Linear search performs *dN* operations for *N* features in the database and *d* dimensions
- No exact methods are faster than linear search for d>10
- Approximate methods can be much faster, but at the cost of missing some correct matches. Failure rate gets worse for large datasets.

Success of text retrieval

- efficient
- high precision
- scalable



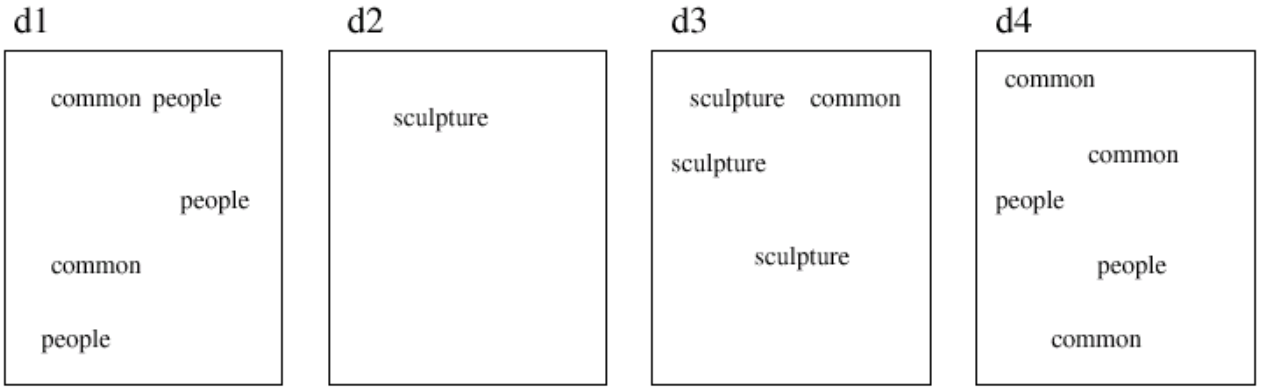
- Use retrieval mechanisms from text for visual retrieval

Text retrieval lightning tour

Stemming Represent words by stems, e.g. “walking”, “walks” ⇨ “walk”

Stop-list Reject the very common words, e.g. “the”, “a”, “of”

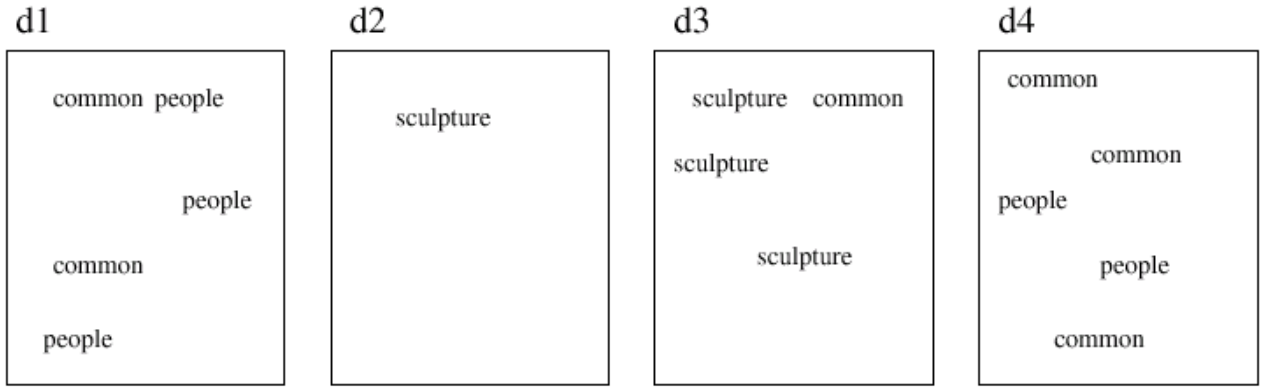
Inverted file



Ideal book index:

Term	**List of hits** (occurrences in documents)
People	[d1:hit hit hit], [d4:hit hit] …
Common	[d1:hit hit], [d3: hit], [d4: hit hit hit] …
Sculpture	[d2:hit], [d3: hit hit hit] …

- word matches are pre-computed



Ranking

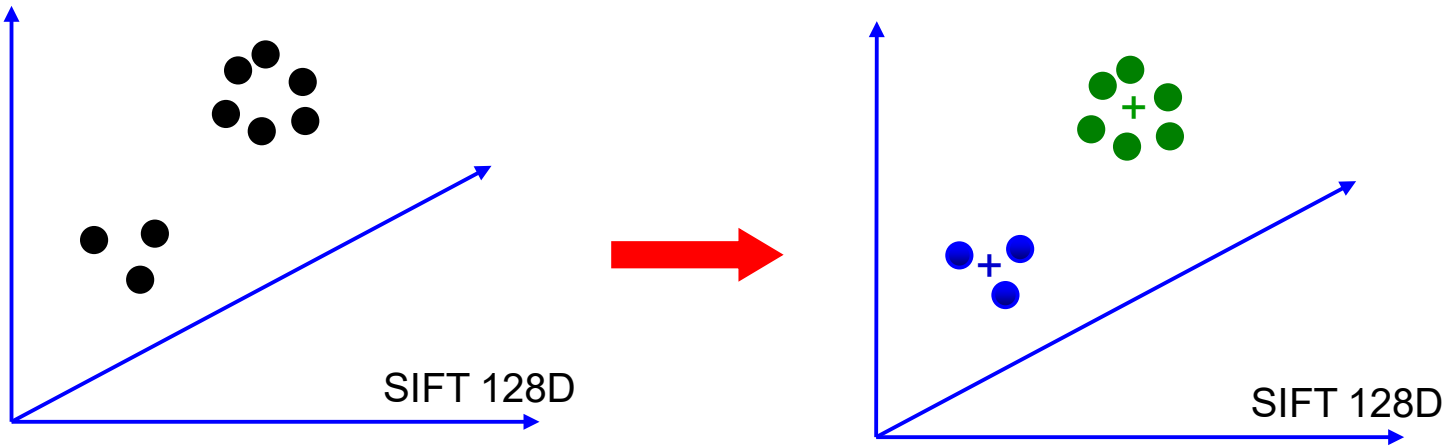
- tf-idf
  - frequency of words in document
  - weighted by inverse document frequency

Need to map feature descriptors to “visual words”.

Build a visual vocabulary for a set of images

Vector quantize descriptors

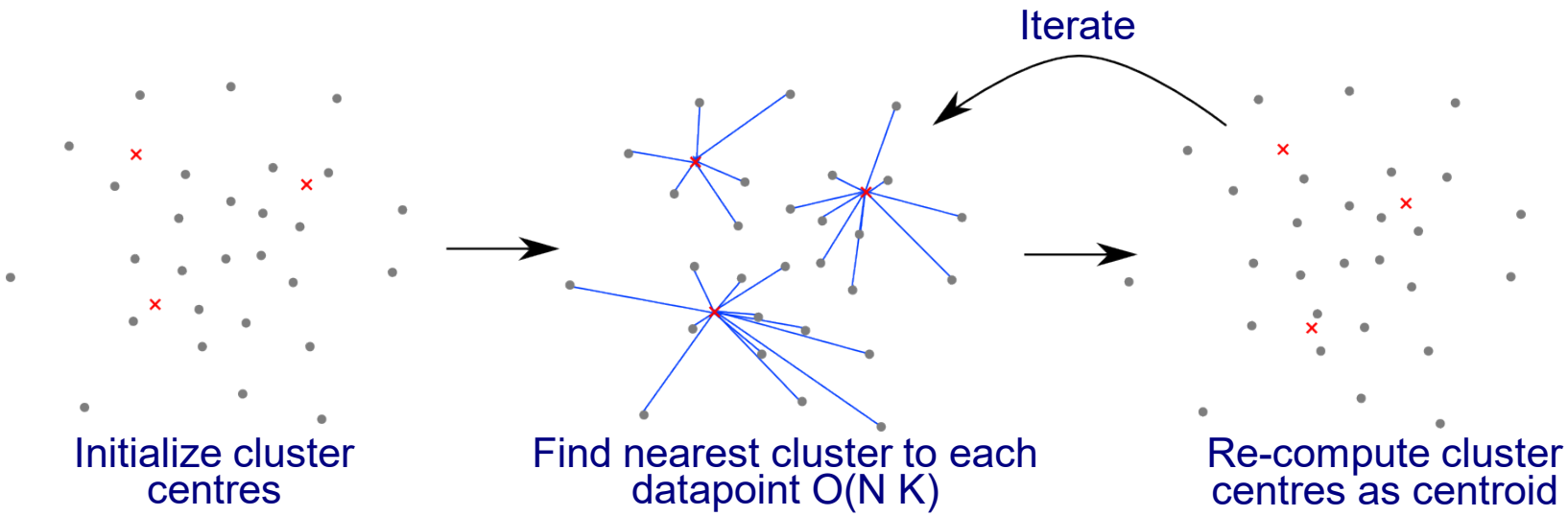
- k-means clustering



Implementation

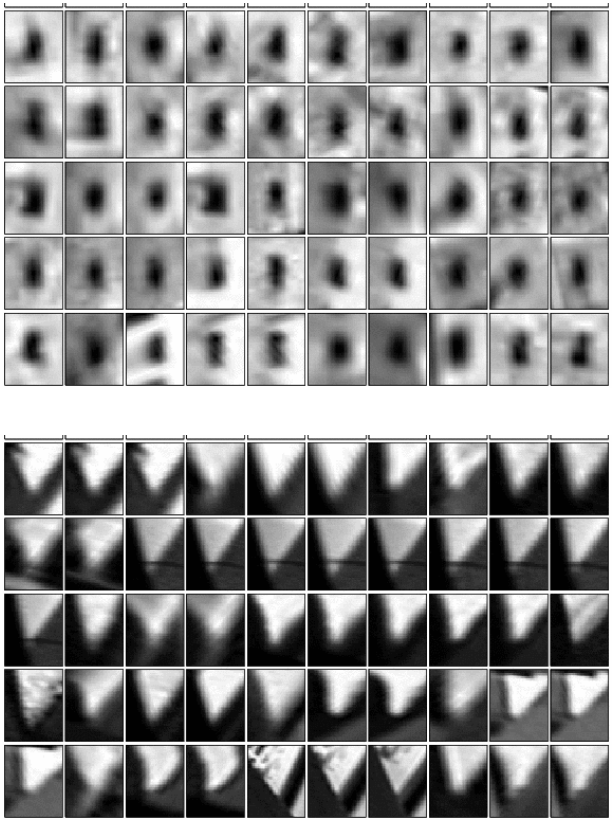
- compute SIFT features on images of dataset
- k-means cluster
- each cluster is a “visual word”

K-means overview

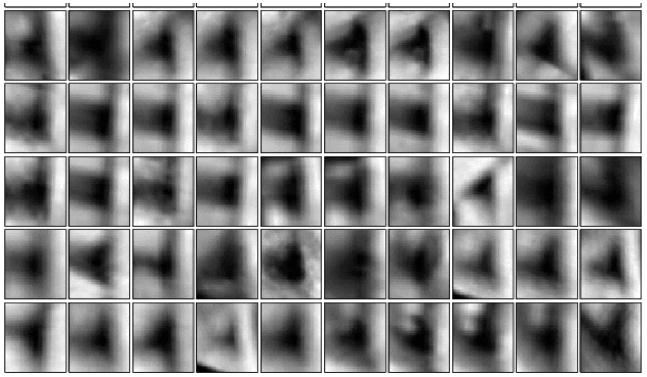


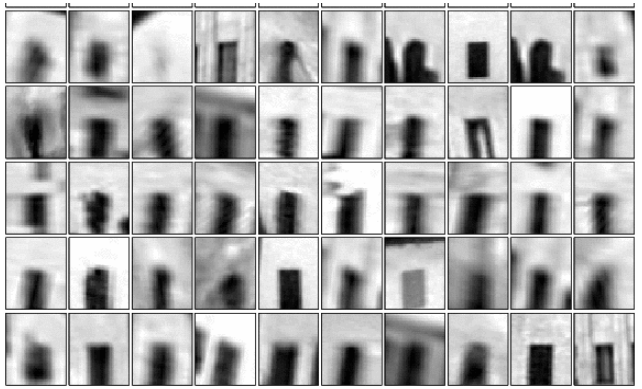
- K-means provably locally minimizes the sum of squared errors (SSE) between a cluster centre and its points

Samples of visual words (clusters on SIFT descriptors):



Shape adapted regions



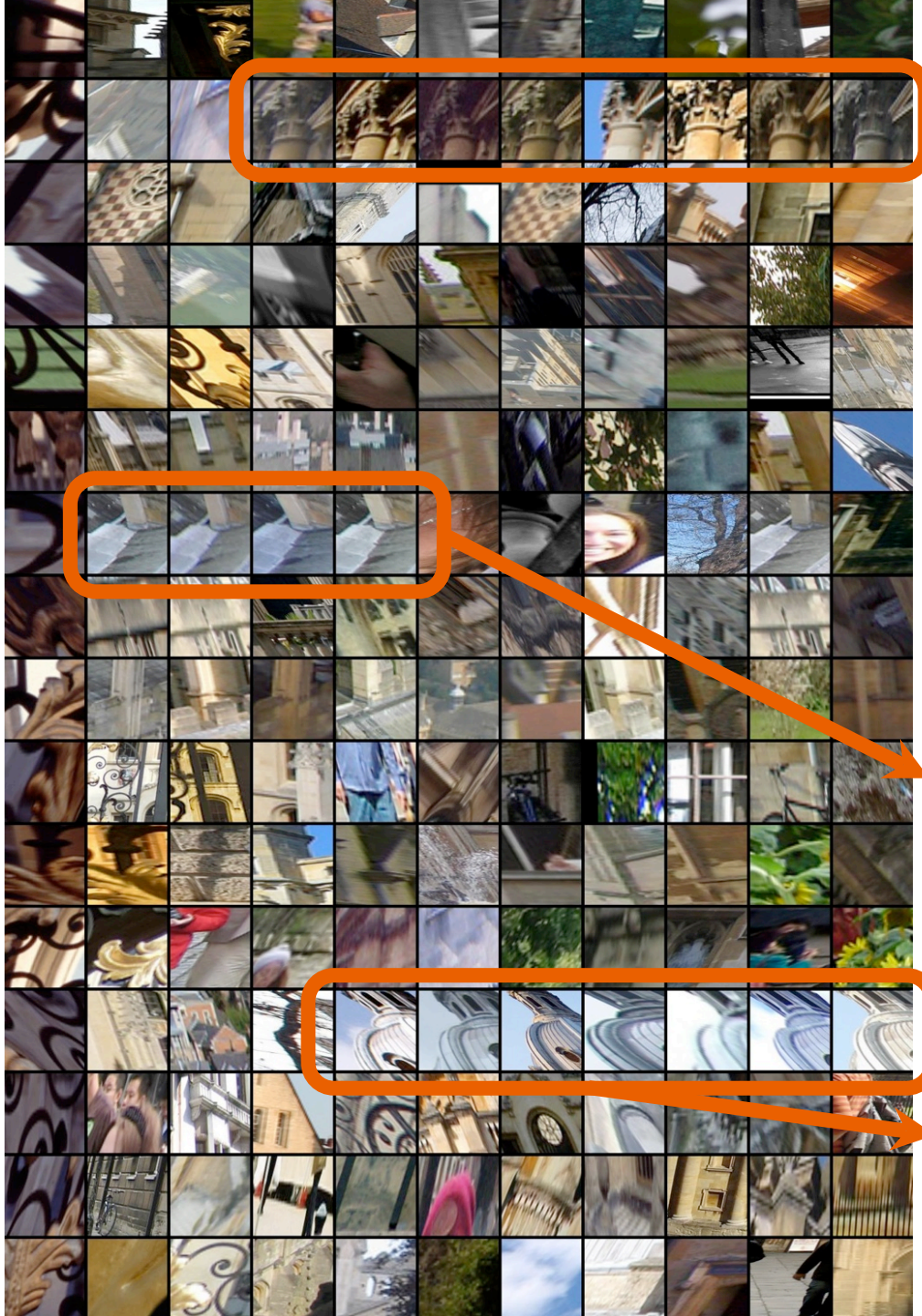


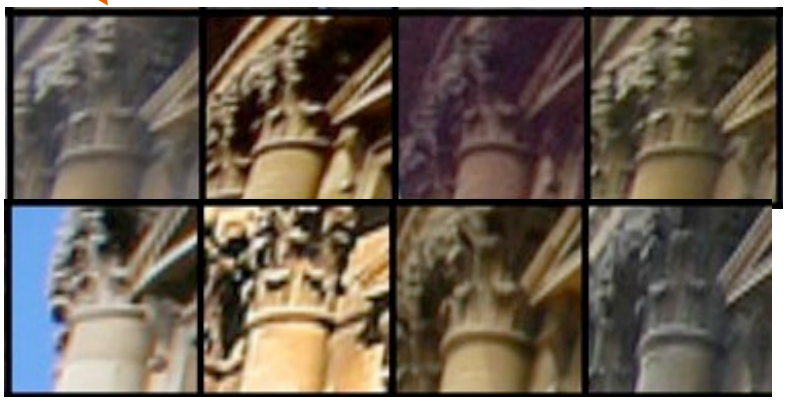
Maximally stable regions

generic examples – cf textons

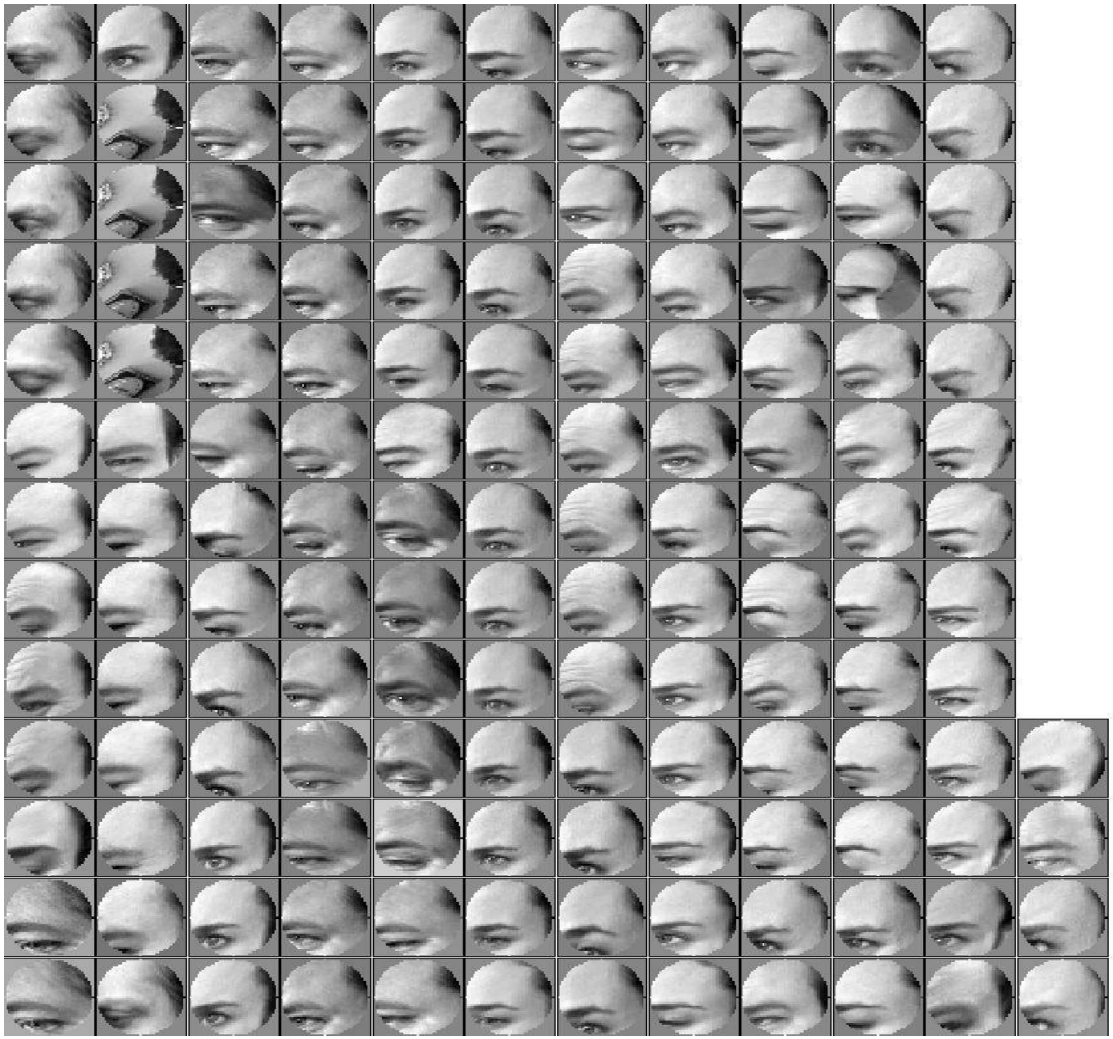
Visual word examples

- shows patched mapped to the same visual word



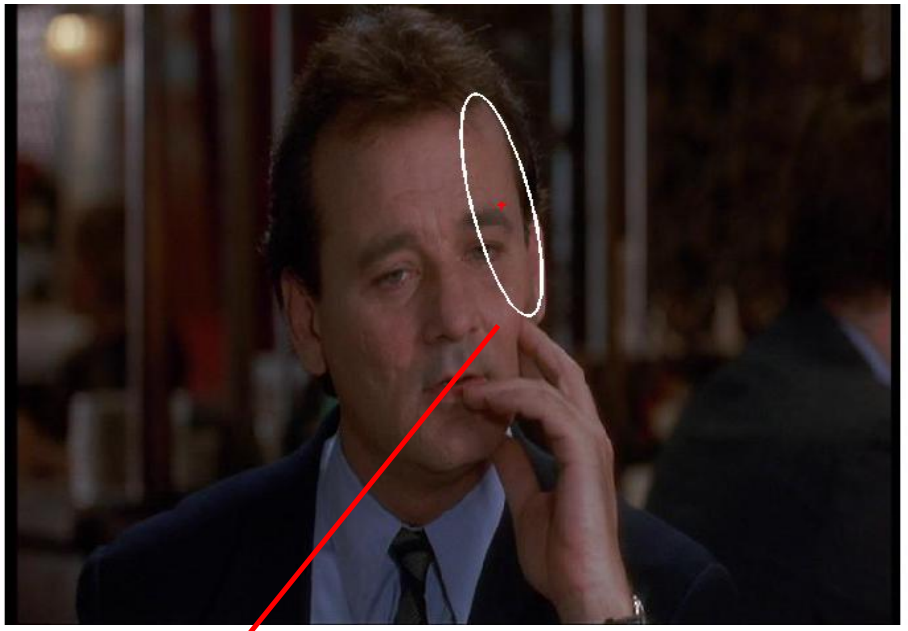


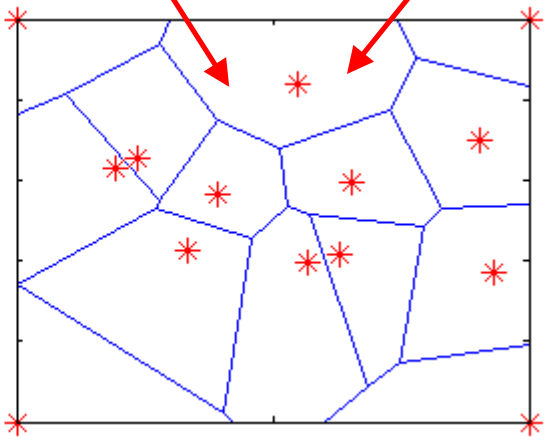
Samples of visual words (clusters on SIFT descriptors):



More specific example

Vector quantize the descriptor space (SIFT)





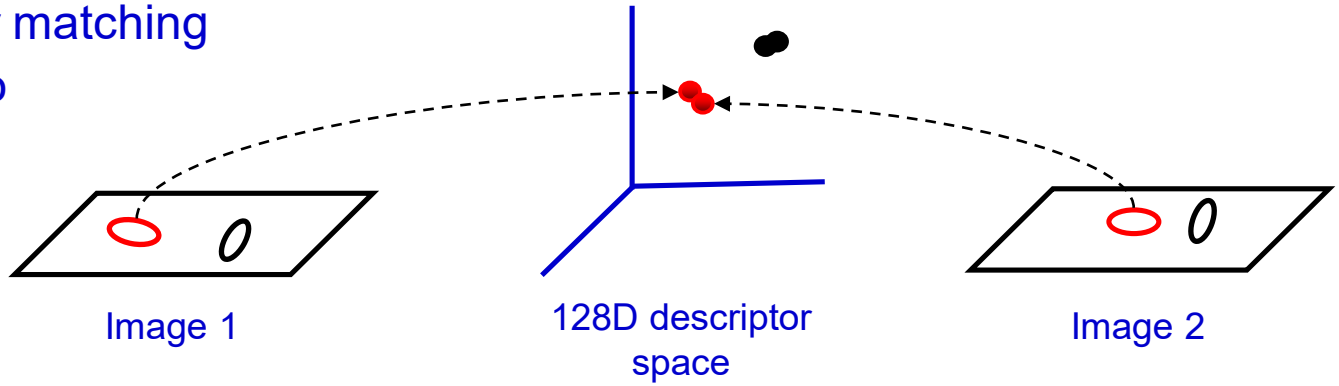
The same visual word

Visual words: quantize descriptor space

Sivic and Zisserman, ICCV 2003

Nearest neighbour matching

- expensive to do for all frames

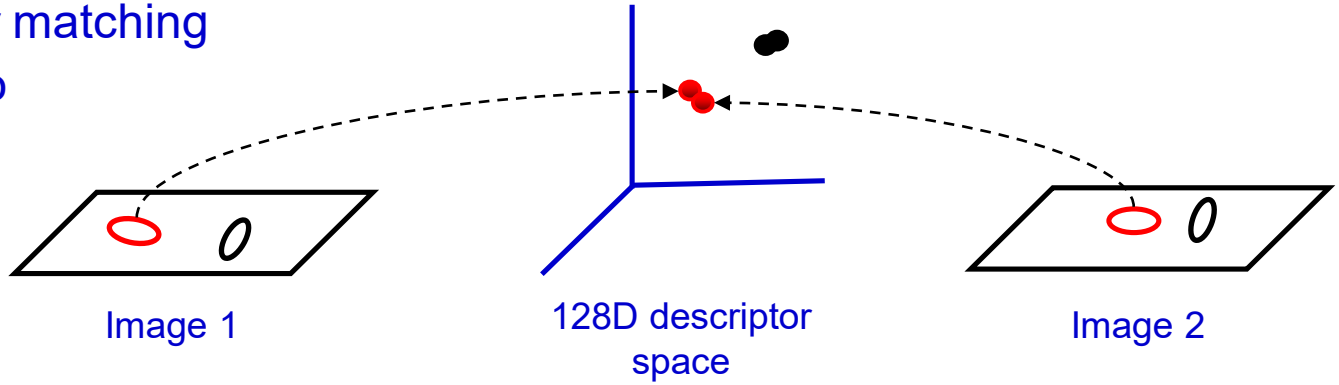


Visual words: quantize descriptor space

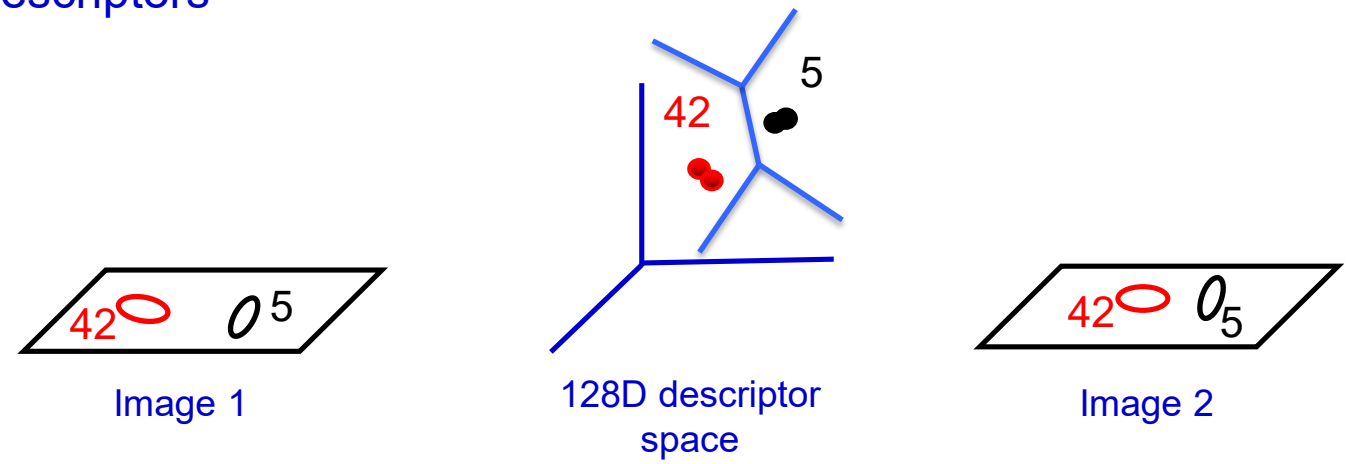
Sivic and Zisserman, ICCV 2003

Nearest neighbour matching

- expensive to do for all frames



Vector quantize descriptors

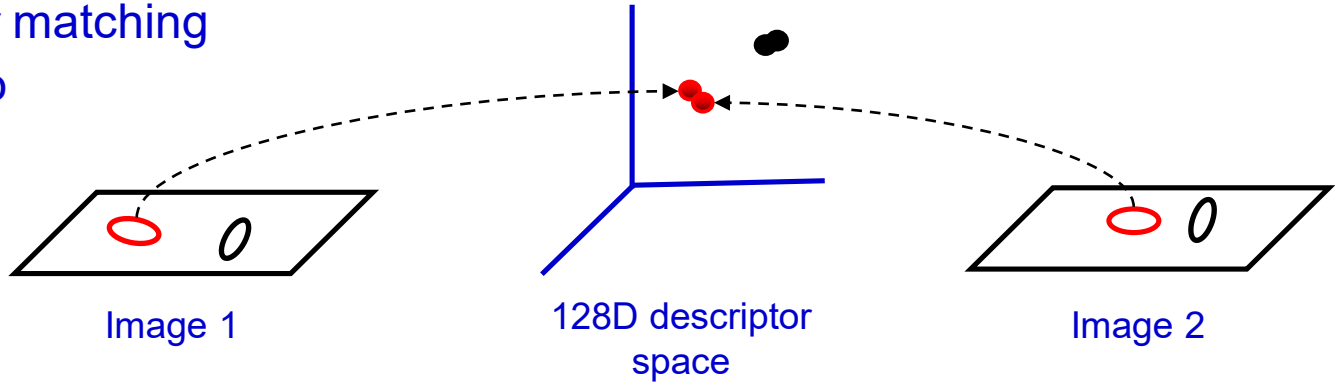


Visual words: quantize descriptor space

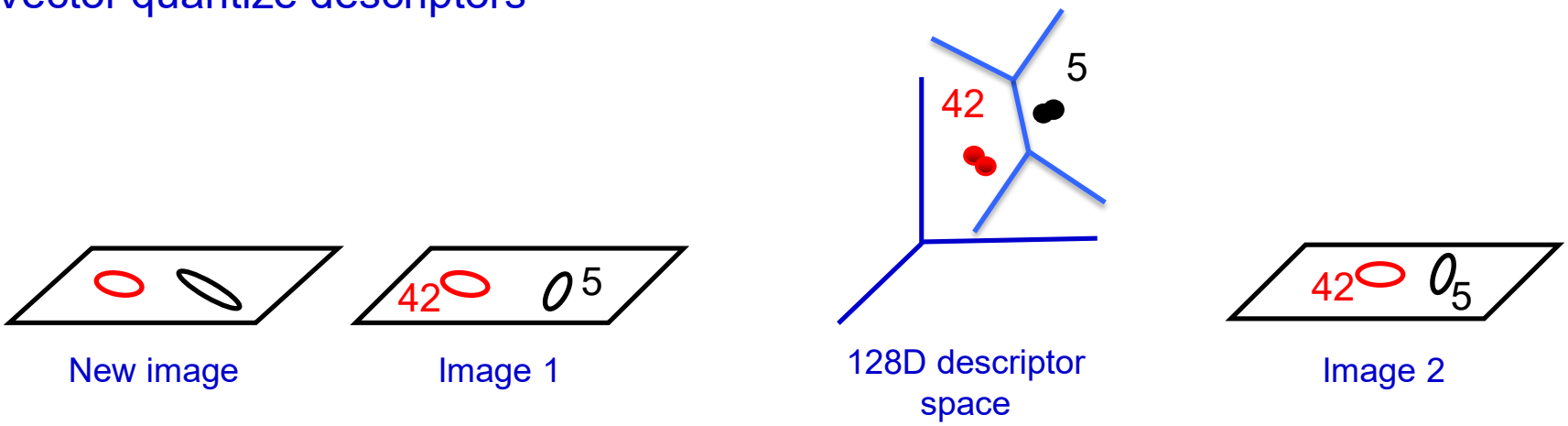
Sivic and Zisserman, ICCV 2003

Nearest neighbour matching

- expensive to do for all frames



Vector quantize descriptors

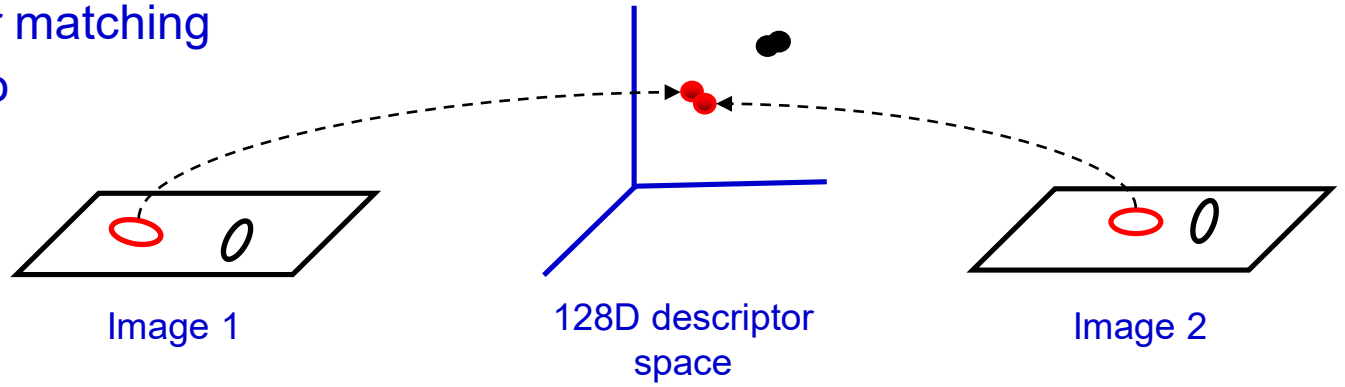


Visual words: quantize descriptor space

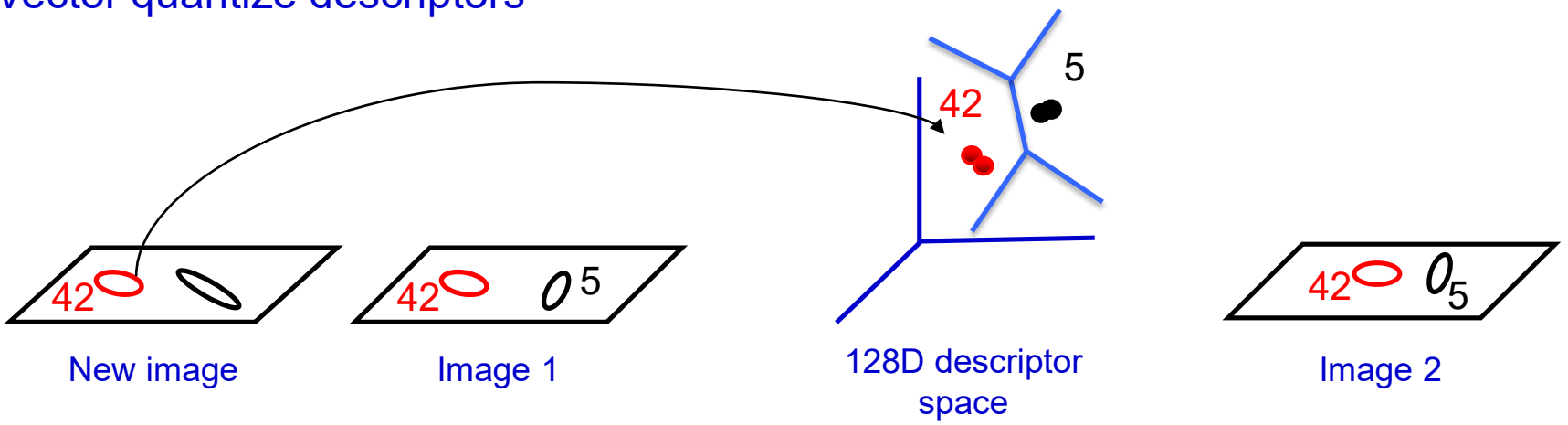
Sivic and Zisserman, ICCV 2003

Nearest neighbour matching

- expensive to do for all frames



Vector quantize descriptors



Offline: create an index

- For fast search, store a “posting list” for the dataset
- This maps word occurrences to the documents they occur in

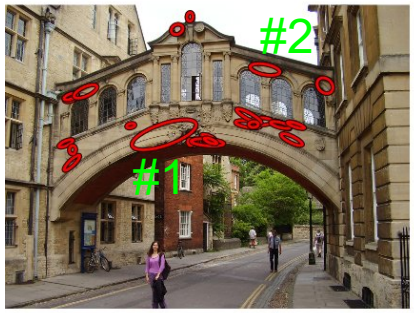


image #5

image #10

Posting list

1	5,10,...
2	5,10,...
...	...

At run time …

- User specifies a query region
- Generate a short list of images using visual words in region

1. Accumulate all visual words within the query region
2. Use “book index” to find other frames with these words
3. Compute similarity for frames which share at least one word

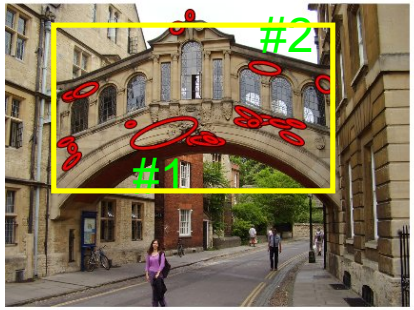
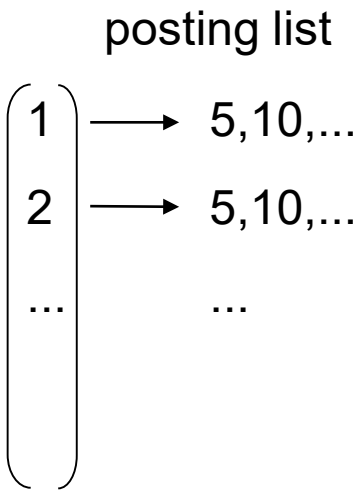


image #5

image #10



- Generates a ranked list of all the images in dataset

Image ranking using the bag-of-words model

For a vocabulary of size K, each image is represented by a K-vector

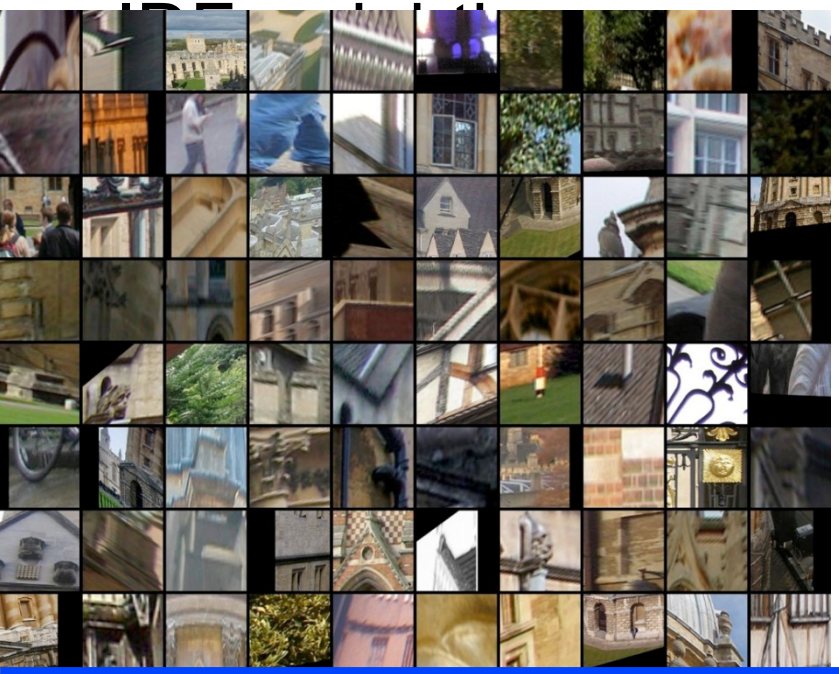
$$\mathbf{v}_d = (t_1, \ldots, t_i, \ldots, t_K)^\top$$

where t_i is the (weighted) number of occurrences of visual word i.

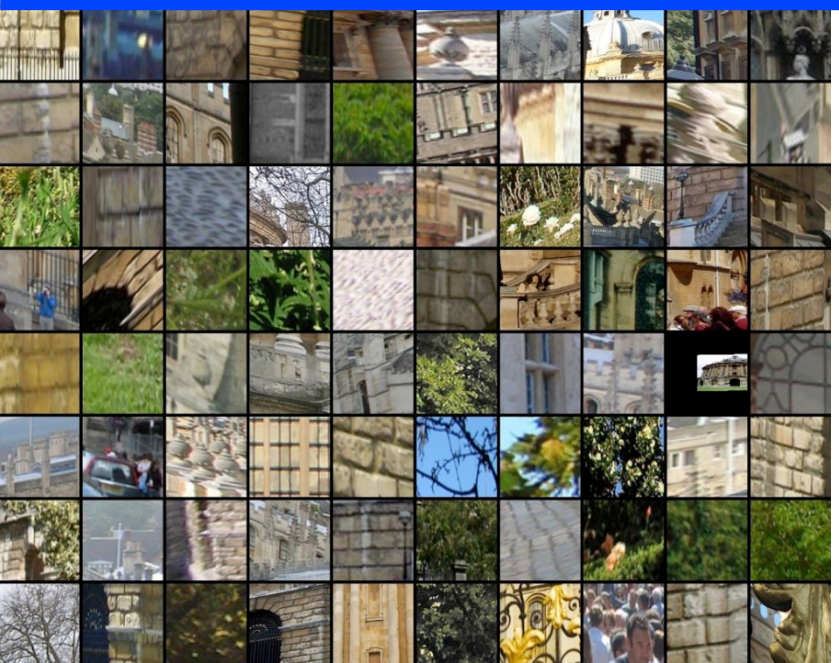
Images are ranked by the normalized scalar product between the query vector v_q and all vectors in the database v_d:

$$f_d = \frac{\mathbf{v}_q{}^\top \mathbf{v}_d}{\|\mathbf{v}_q\|_2 \; \|\mathbf{v}_d\|_2}$$

Scalar product can be computed efficiently using inverted file.



large IDF



small IDF

Common words are *not* informative

- e.g. grass, bricks, ...

Captured by **inverse document frequency (IDF)** $\mathrm{idf}(q) = \log \frac{\#\text{images}}{\#\text{images} \ni q}$

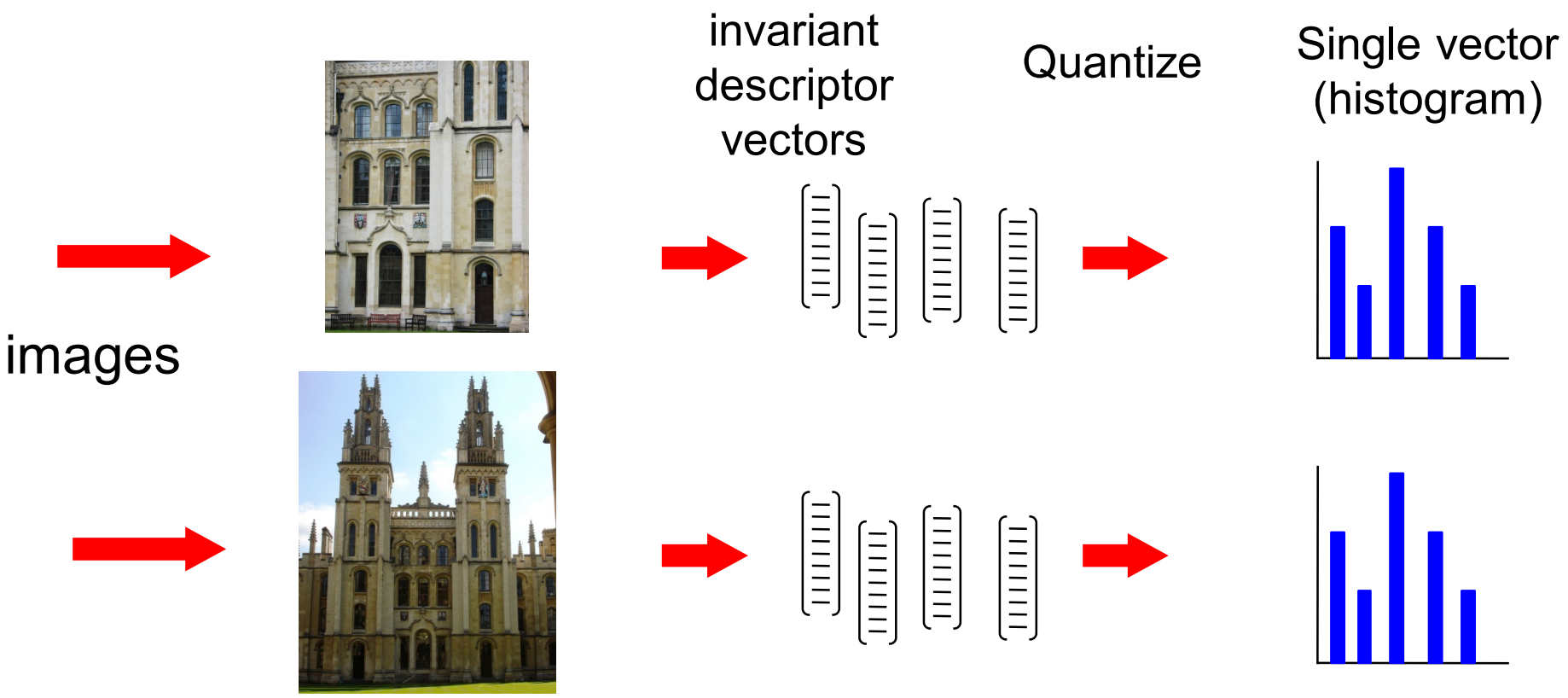
$$\mathbf{h}_q \leftarrow \mathrm{idf}(q) \times \mathbf{h}_q$$

Weight histogram components by IDF

$$\mathbf{h} \leftarrow \frac{\mathbf{h}}{\|\mathbf{h}\|_2}$$

And do not forget to normalise

Summary: Match histograms of visual words



1. Compute affine covariant regions in each frame independently (offline)
2. "Label" each region by a vector of descriptors based on its intensity (offline)
3. **Build histograms of visual words by descriptor quantization (offline)**
4. **Rank retrieved frames by matching vis. word histograms using inverted files.**

Films = common dataset

"Pretty Woman"

"Casablanca"

"Groundhog Day"

"Charade"

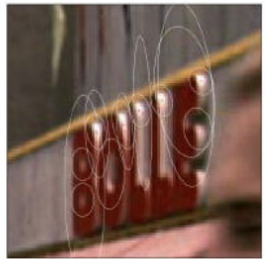
Video Google: The Results

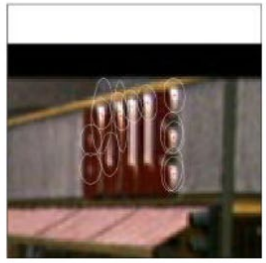
A supermarket logo query in the movie 'Run Lola Run'

Query

Query frame with the outlined query region

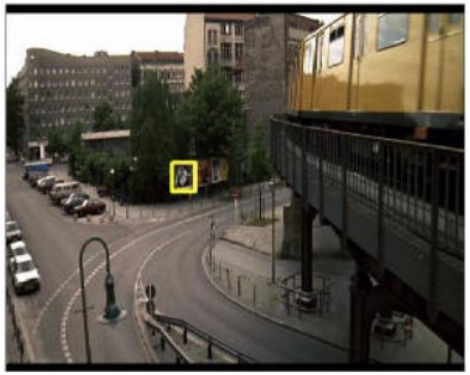
Close up of the region





A poster query in the movie 'Run Lola Run'

Query



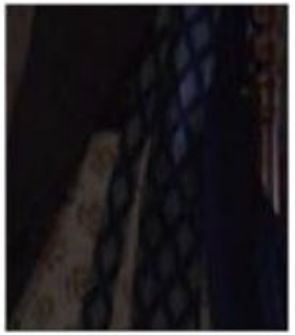
Query frame with the outlined query region

Close up of the region

Query

Query frame with the outlined query region

Close up of the region









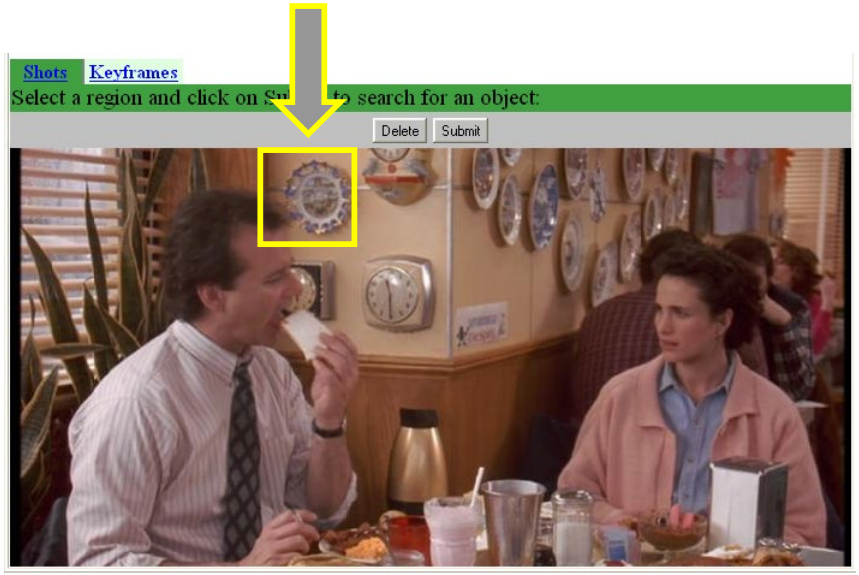





Video Google Demo

Example

Select a region



retrieved shots

Search in film “Groundhog Day”

Investigate …

Vocabulary size: number of visual words in range 10K to 1M

Use of spatial information to re-rank

Oxford buildings dataset

- Landmarks plus queries used for evaluation

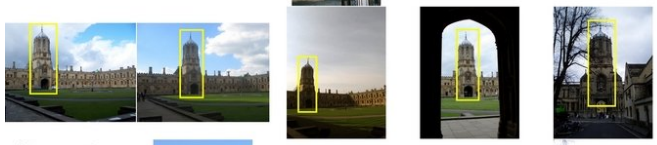
All Soul's

Ashmolean

Balliol

Bodleian

Thom Tower



Cornmarket

Bridge of Sighs

Keble

Magdalen

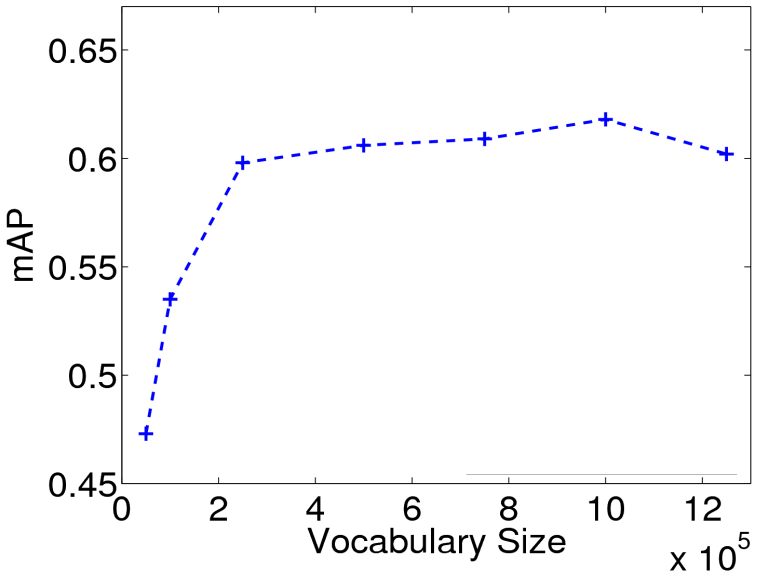
University Museum

Radcliffe Camera

- Ground truth obtained for 11 landmarks over 5062 images
- Evaluate performance by mean Average Precision over 55 queries

Performance against vocabulary size

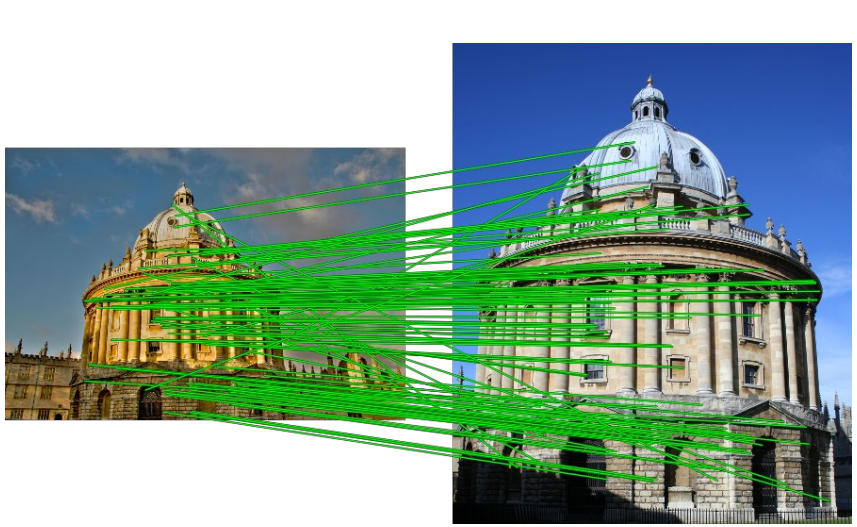
- Using large vocabularies gives a big boost in performance (peak @ 1M words)

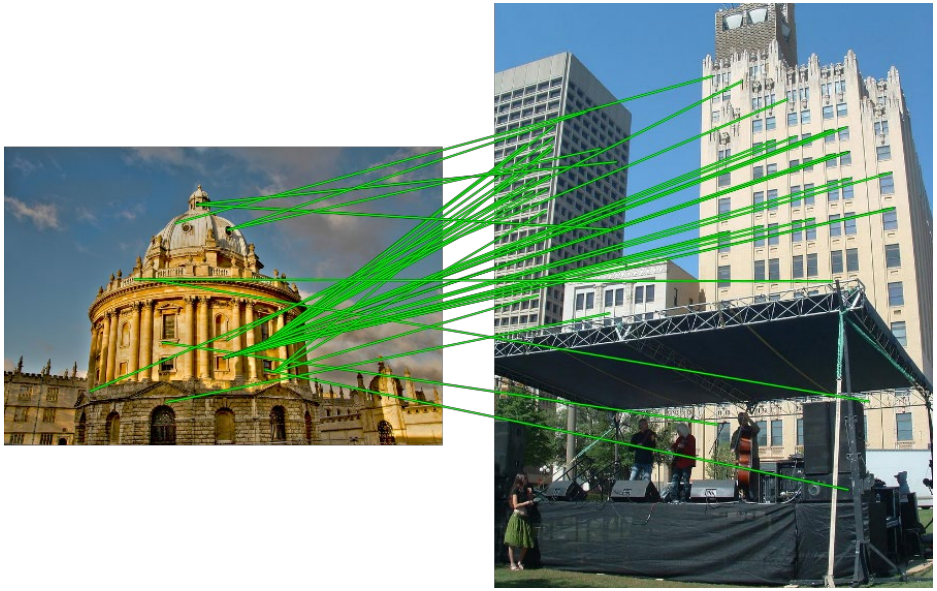


- More discriminative vocabularies give:
  - Better retrieval quality
  - Increased search speed – documents share less words, so fewer documents need to be scored
  - Closer approximation of original SIFTs

Beyond Bag of Words

- Use the **position** and **shape** of the underlying features to improve retrieval quality

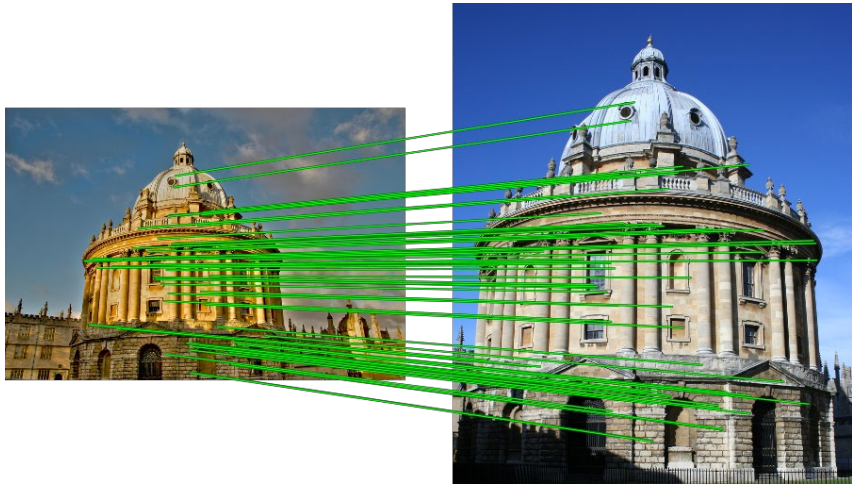




- Both images have many matches – which is correct?

Beyond Bag of Words

- We can measure **spatial consistency** between the query and each result to improve retrieval quality



Many spatially consistent matches – **correct result**

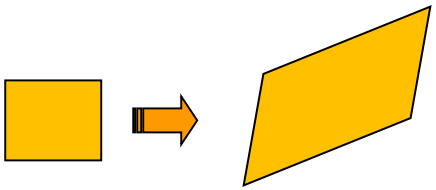
Few spatially consistent matches – **incorrect result**

Compute 2D affine transformation

- between query region and target image

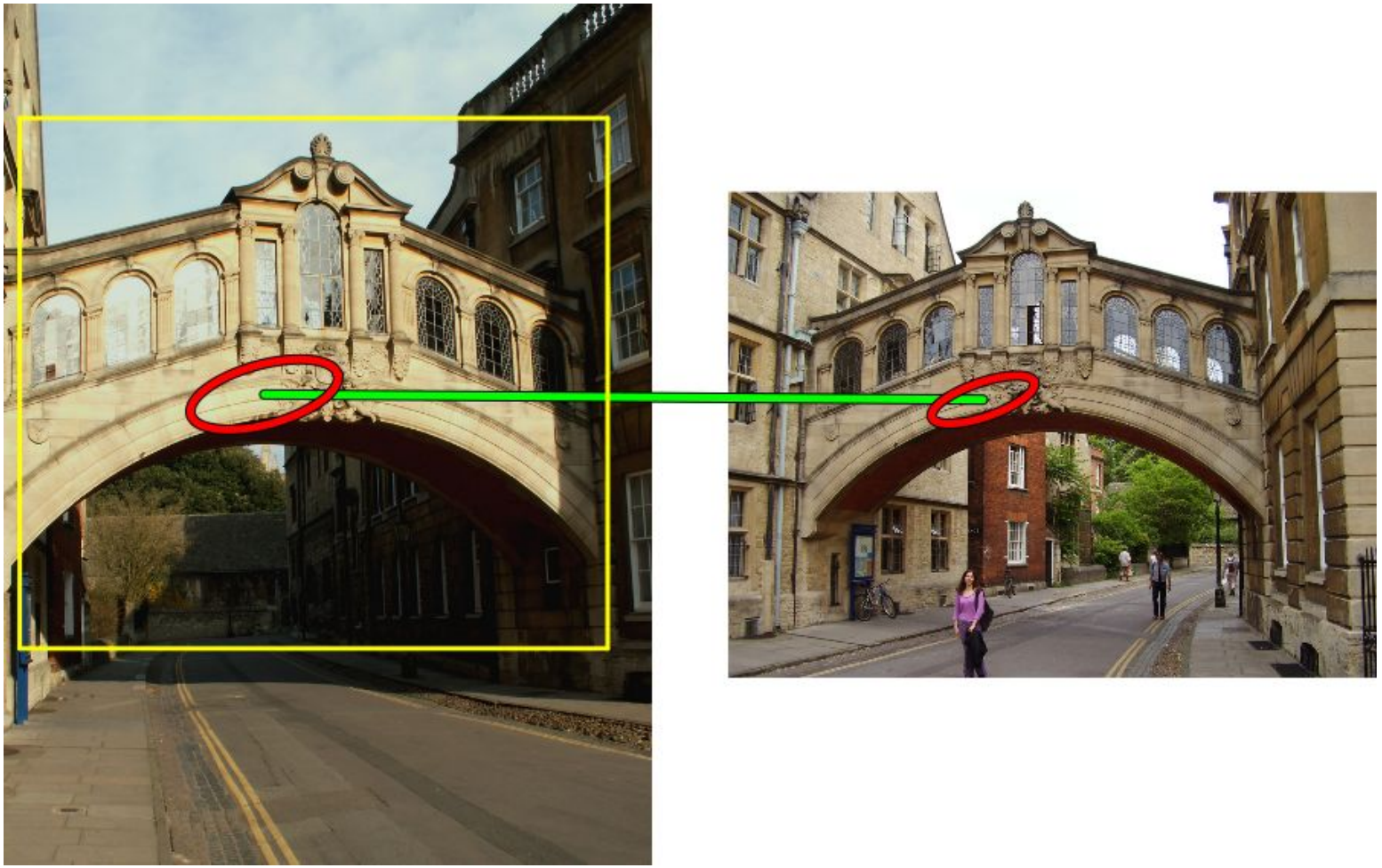
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \mathtt{A} \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

where A is a 2x2 non-singular matrix



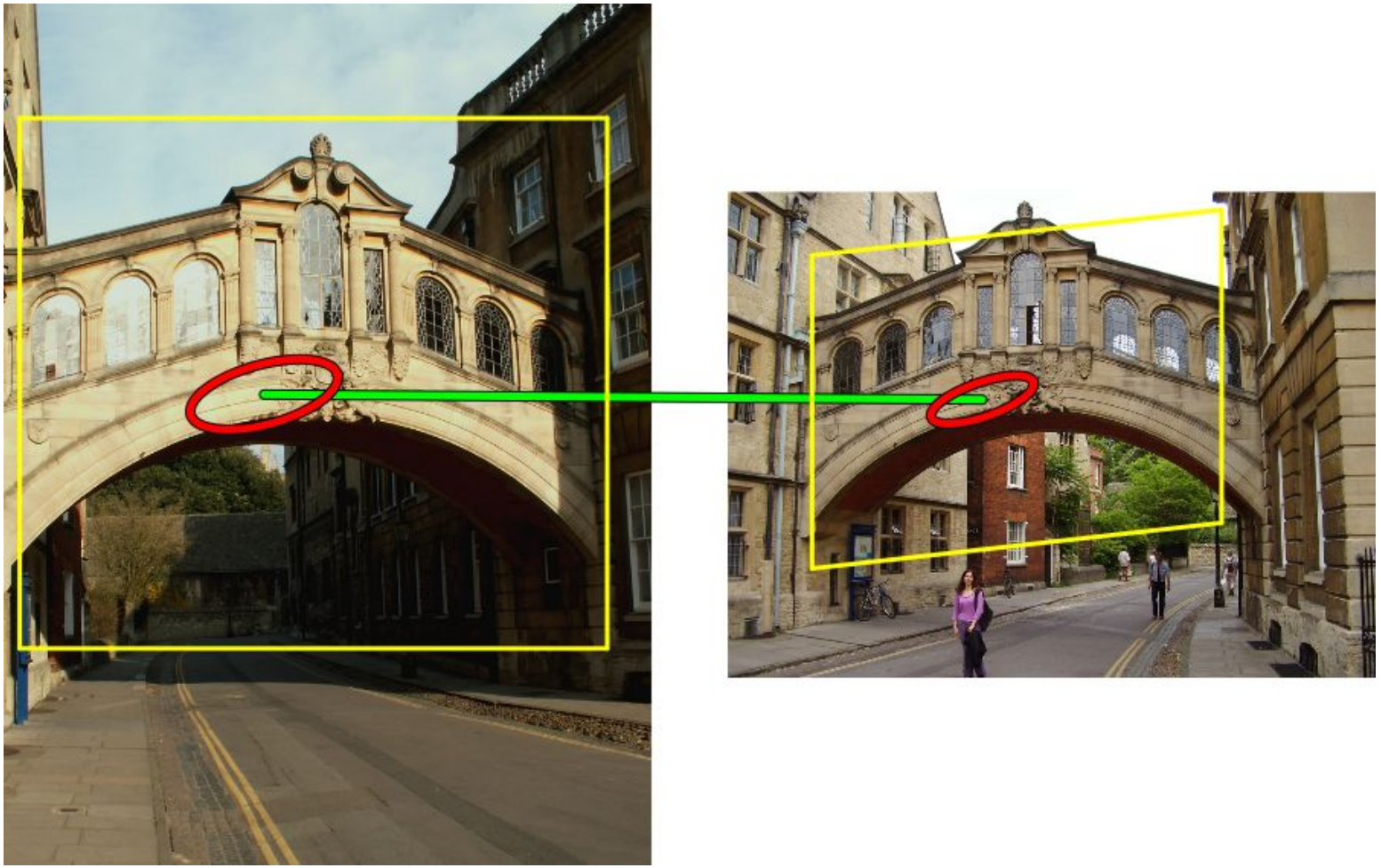
Estimating spatial correspondences

1. Test each correspondence



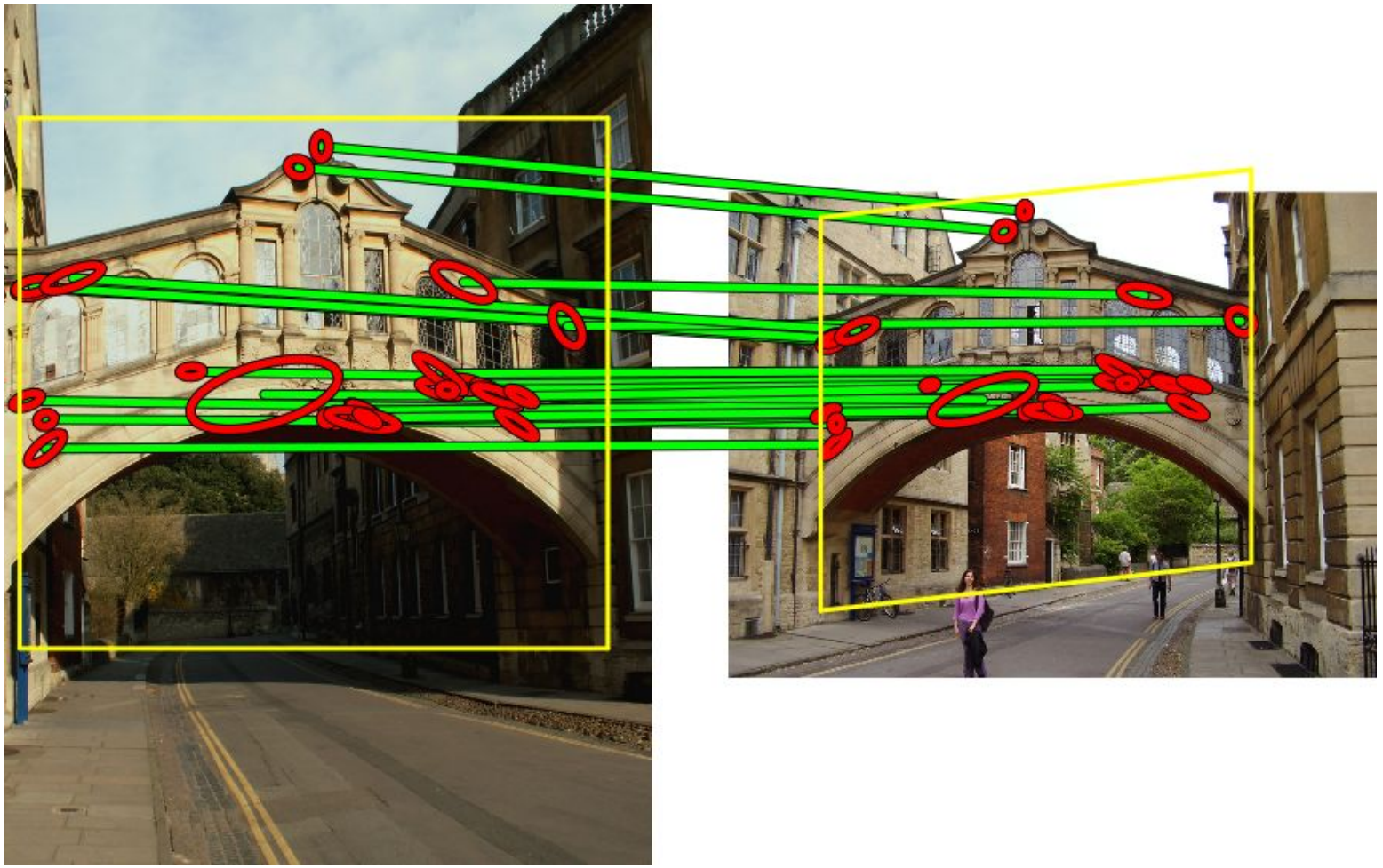
Estimating spatial correspondences

2. Compute a (restricted) affine transformation (5 dof)



Estimating spatial correspondences

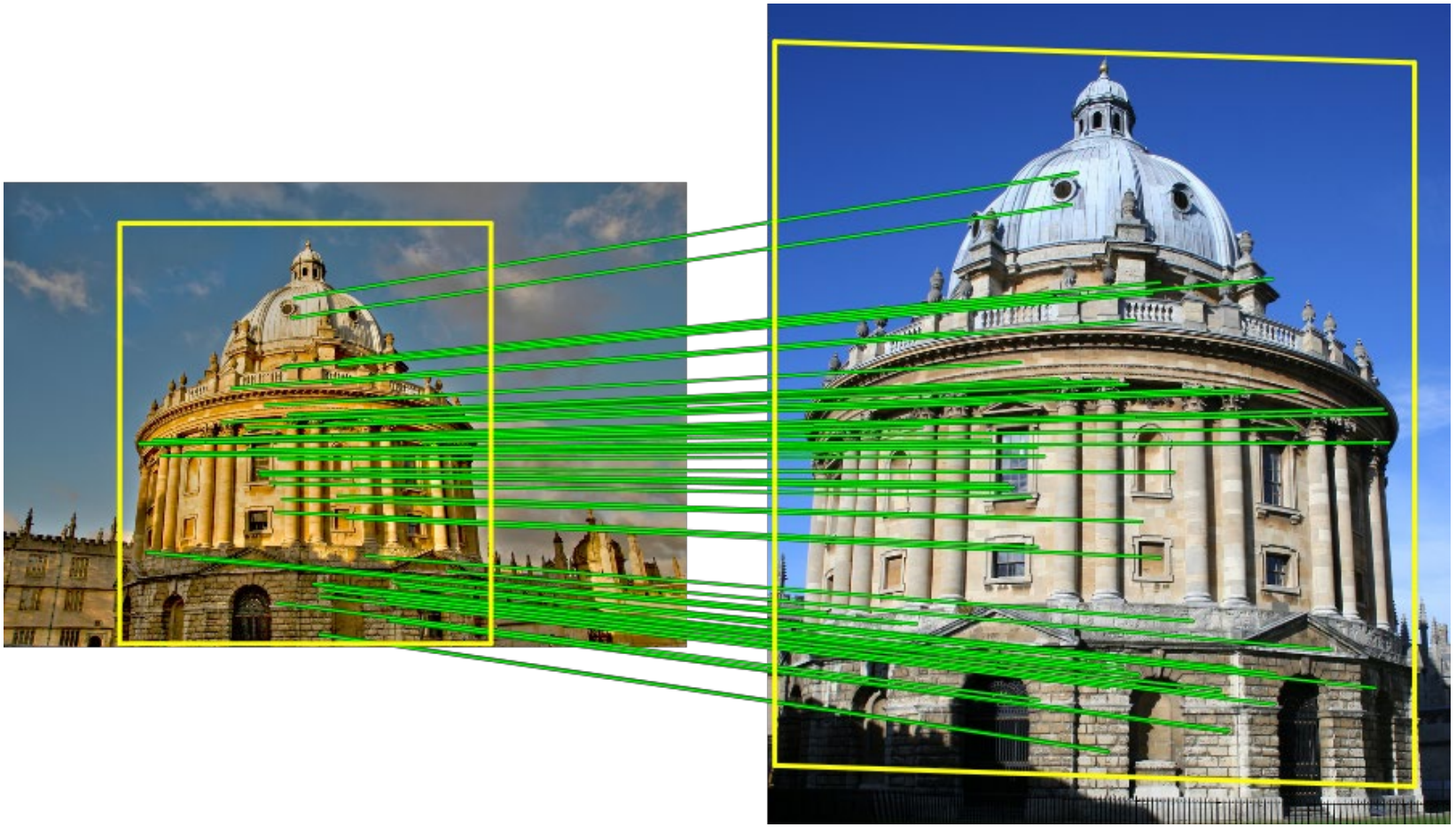
3. Score by number of consistent matches



Use RANSAC on full affine transformation (6 dof)

Beyond Bag of Words

- Extra bonus – gives **localization** of the object

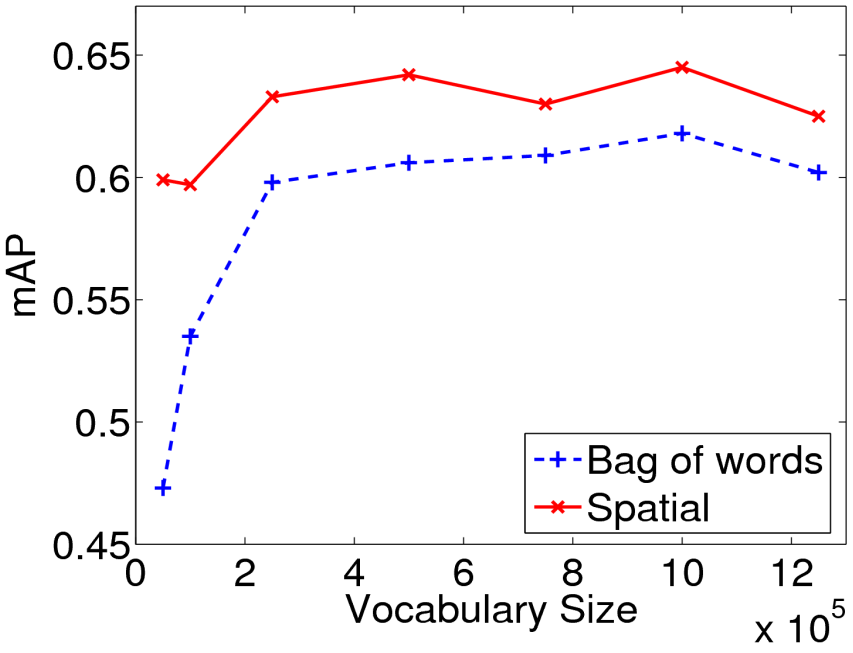


Example Results

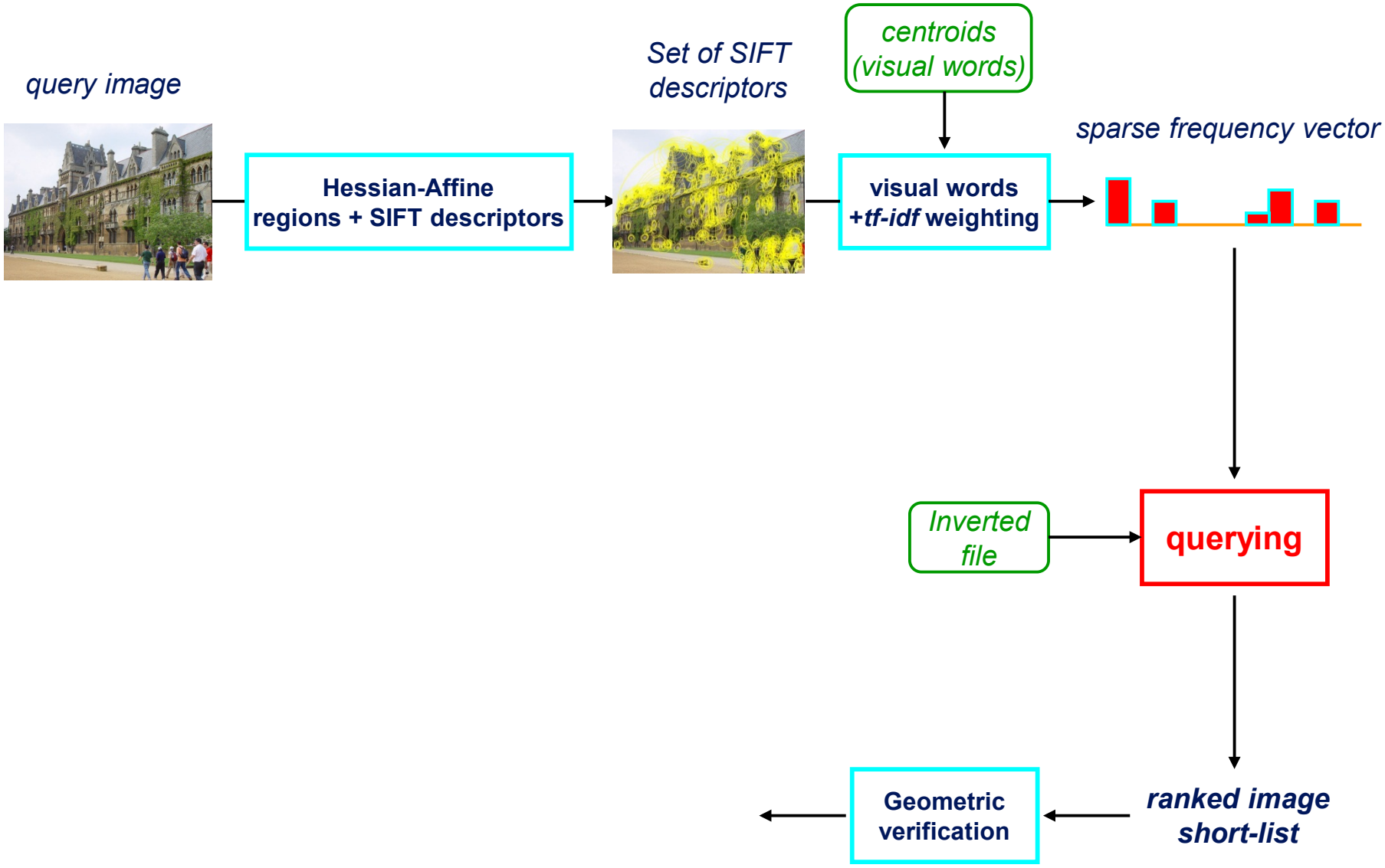
Rank short list of retrieved images on number of correspondences

Mean Average Precision variation with vocabulary size

vocab size	bag of words	spatial
50K	0.473	0.599
100K	0.535	0.597
250K	0.598	0.633
500K	0.606	0.642
750K	0.609	0.630
1M	0.618	0.645
1.25M	0.602	0.625

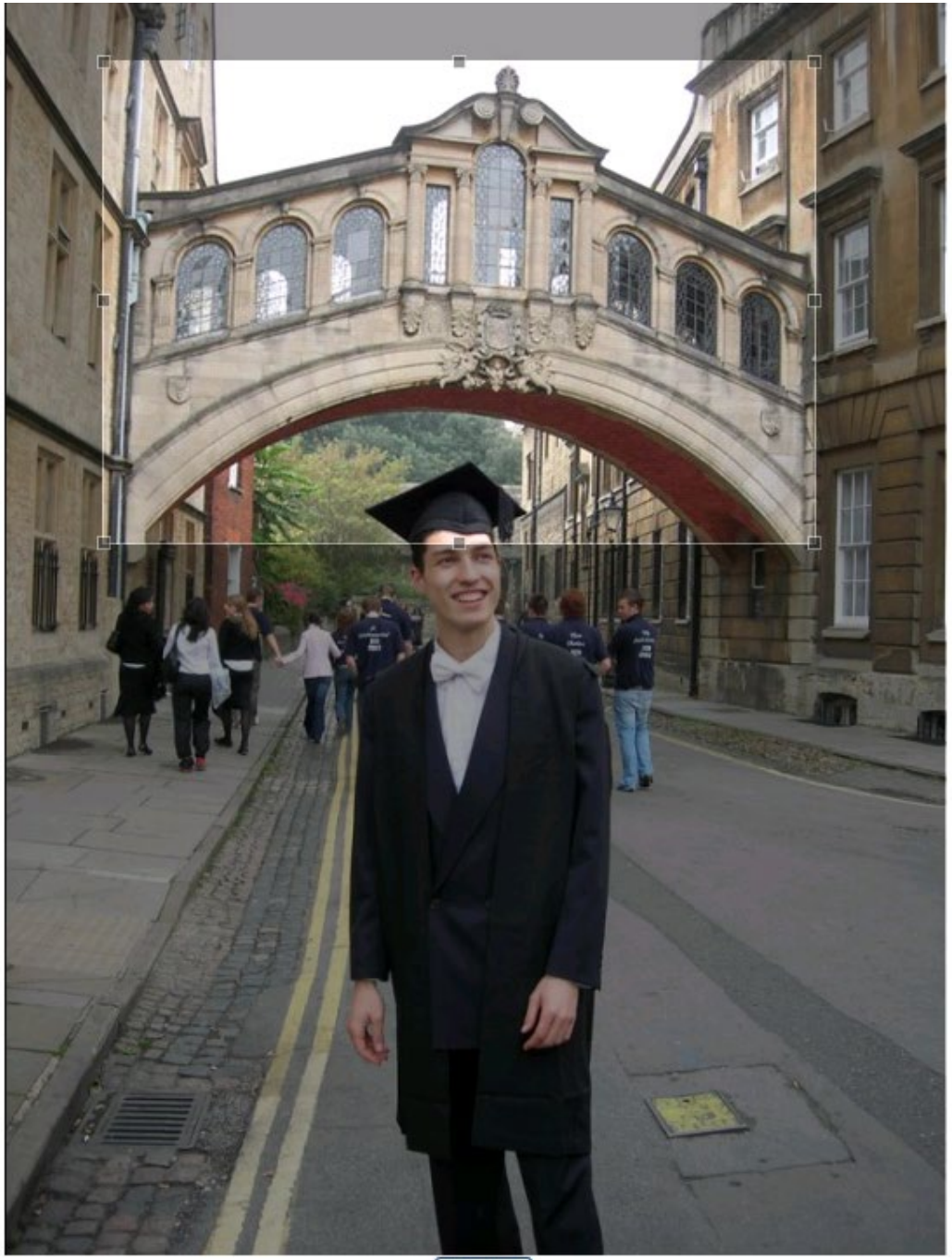


Bag of visual words particular object retrieval



Demo

Example

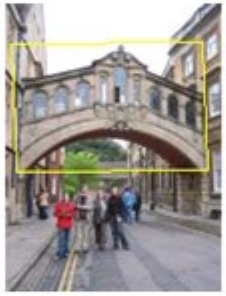


Search results 1 to 20 of 104844

1

ID: oxc1_hertford_000011
Score: 1816.000000
Putative: 2325
Inliers: 1816
Hypothesis: 1.000000 0.000000 0.000015 0.000000 1.000000 0.000031
Detail

2

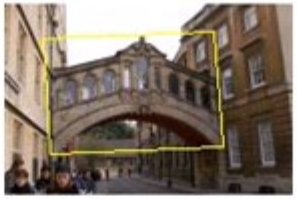


ID: oxc1_all_souls_000075
Score: 352.000000
Putative: 645
Inliers: 352
Hypothesis: 1.162245 0.041211 -70.414459 -0.012913 1.146417 91.276093
Detail

3

ID: oxc1_hertford_000064
Score: 278.000000
Putative: 527
Inliers: 278
Hypothesis: 0.928686 0.026134 169.954620 -0.041703 0.937558 97.962112
Detail

4

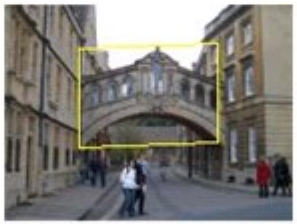


ID: oxc1_oxford_001612
Score: 252.000000
Putative: 451
Inliers: 252
Hypothesis: 1.046026 0.069416 51.576881 -0.044949 1.046938 76.264442
Detail

5

ID: oxc1_hertford_000123
Score: 225.000000
Putative: 446
Inliers: 225
Hypothesis: 1.361741 0.090413 -34.673317 -0.084659 1.301689 -32.281090
Detail

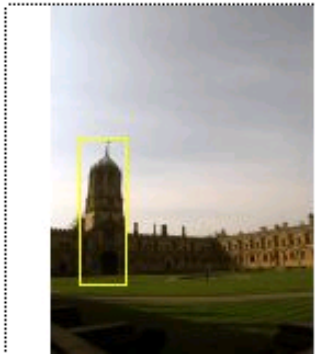
6



ID: oxc1_oxford_001085
Score: 224.000000
Putative: 389
Inliers: 224
Hypothesis: 0.848997 0.000000 195.707611 -0.031077 0.895546 114.583961
Detail

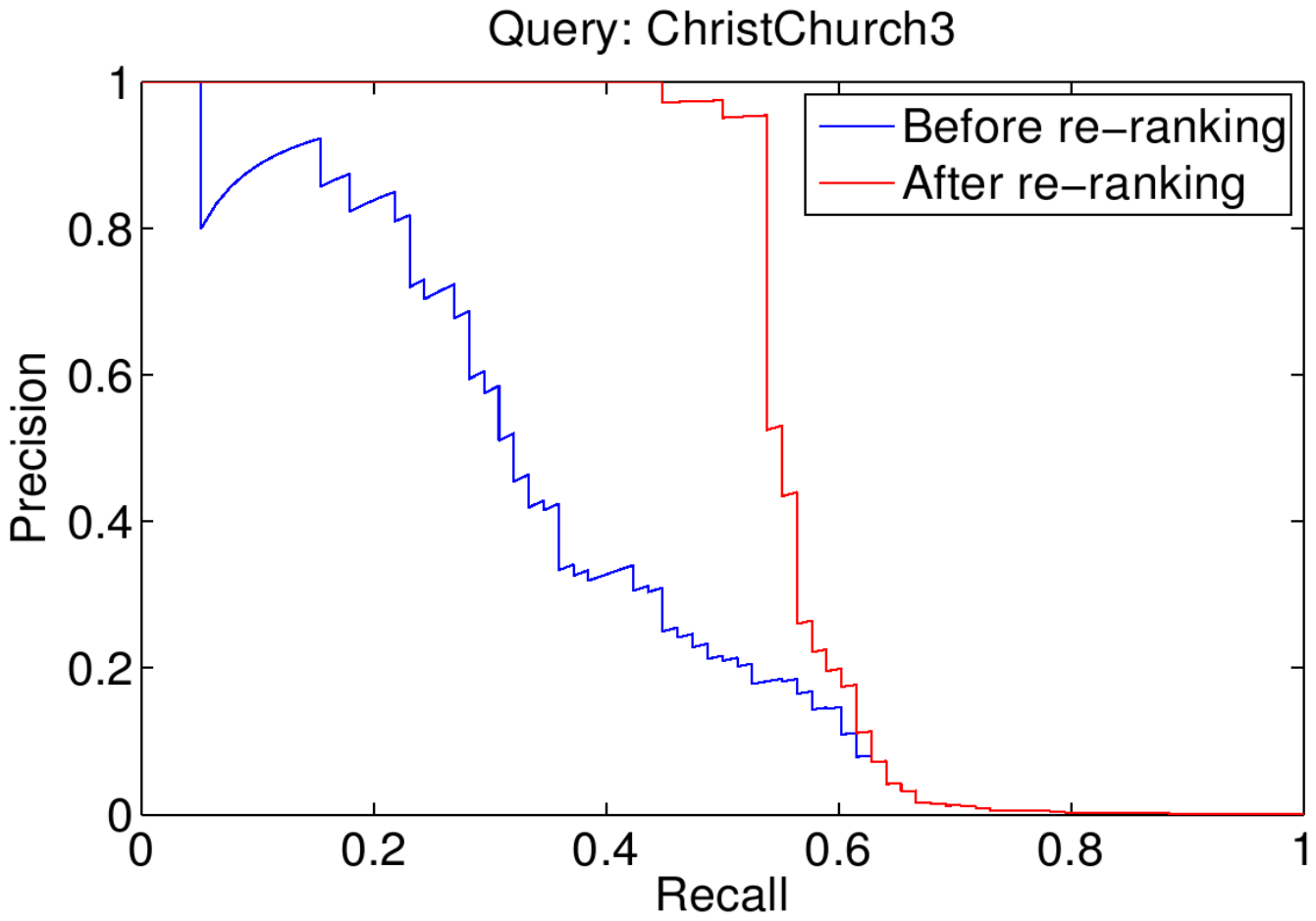
7

ID: oxc1_hertford_000077
Score: 195.000000
Putative: 386
Inliers: 195
Hypothesis: 1.465144 0.069286 -108.473091 -0.097598 1.461877 -30.205191
Detail



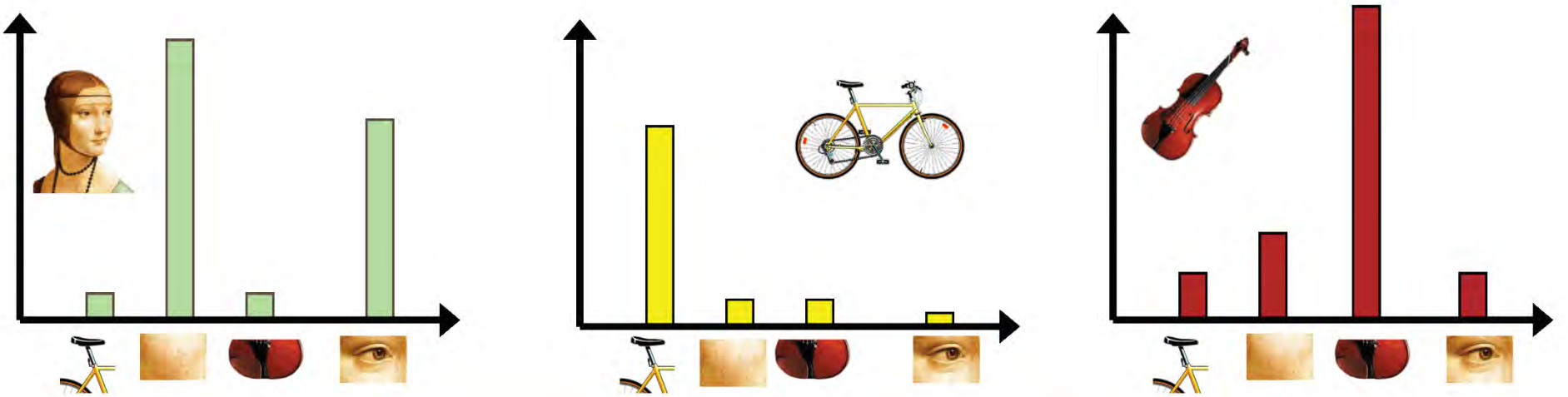
Spatial re-ranking

- improves precision
- but not recall …

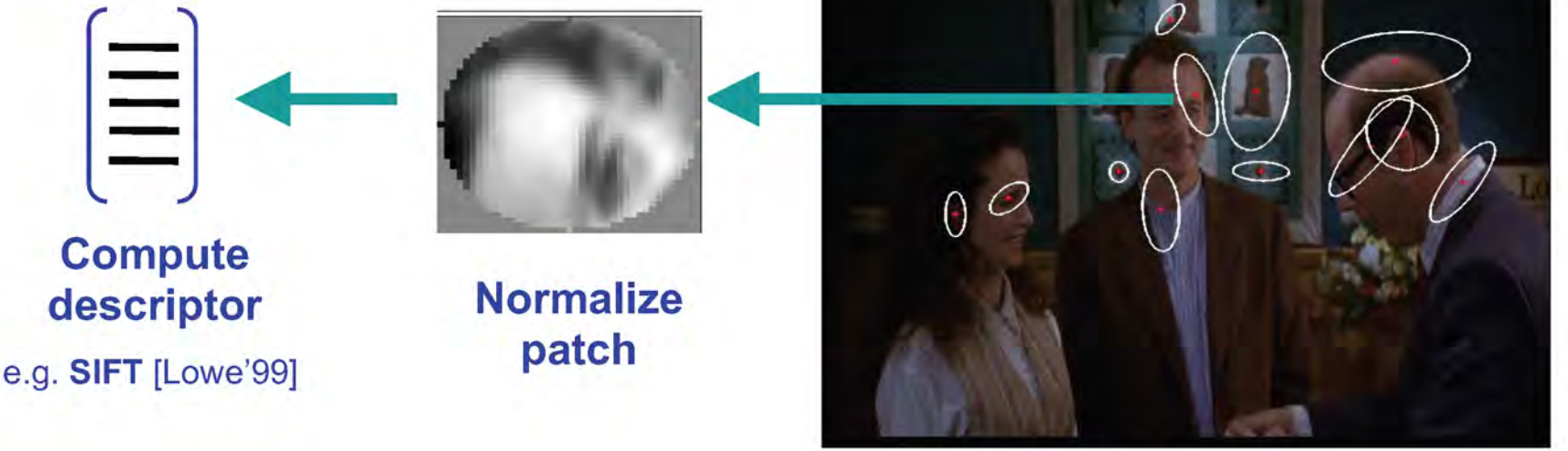


Bag of Words

- Independent features
- Histogram representation



1.Feature detection and representation



Compute descriptor

e.g. **SIFT** [Lowe'99]

Normalize patch

Detect patches

[Mikojaczyk and Schmid '02]

[Mata, Chum, Urban & Pajdla, '02]

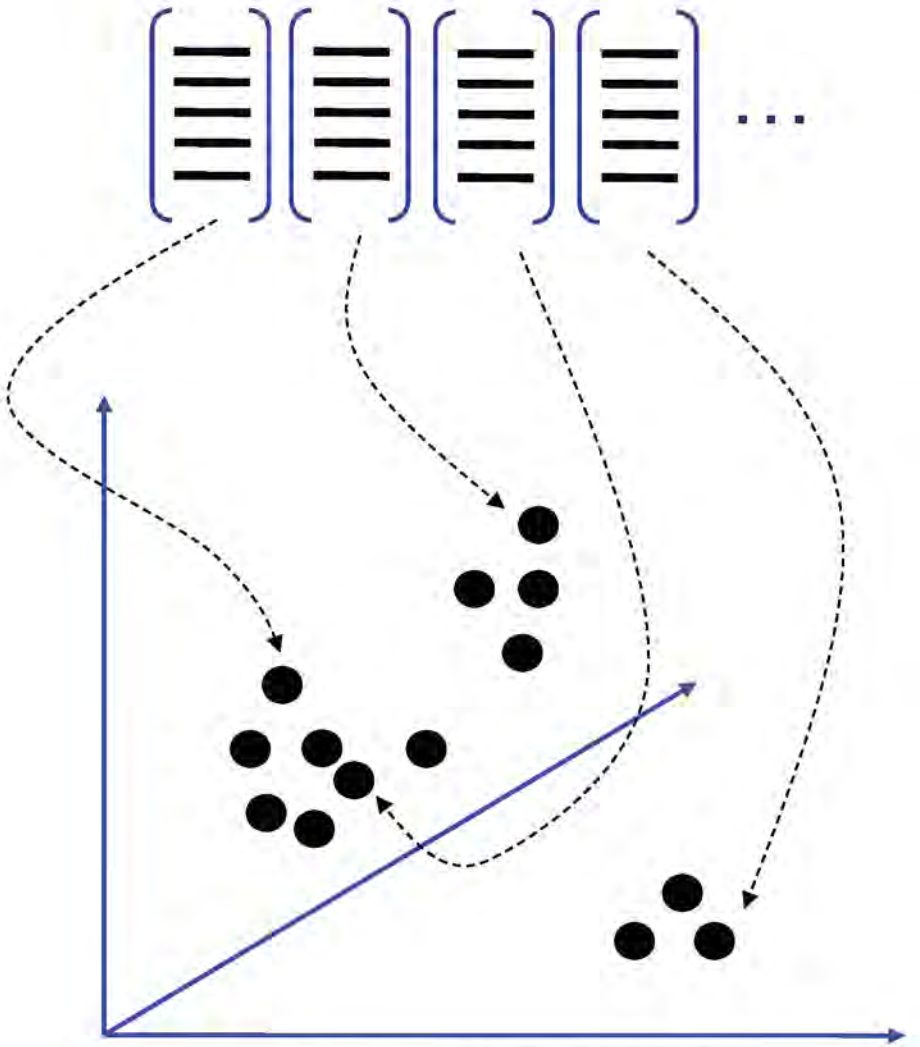
[Sivic & Zisserman, '03]

Local interest operator
or
Regular grid

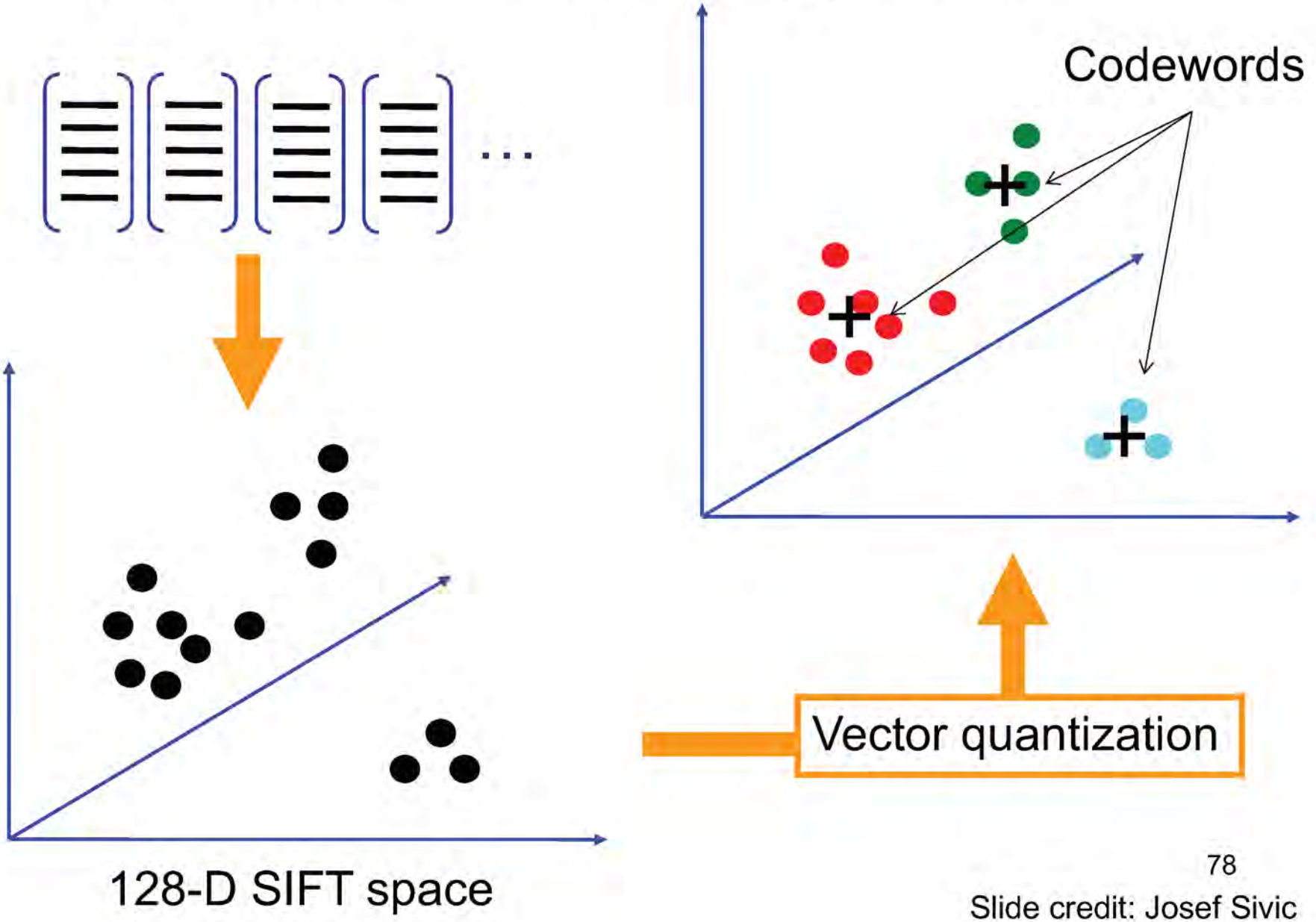
Slide credit: Josef Sivic

1.Feature detection and representation

2. Codewords dictionary formation

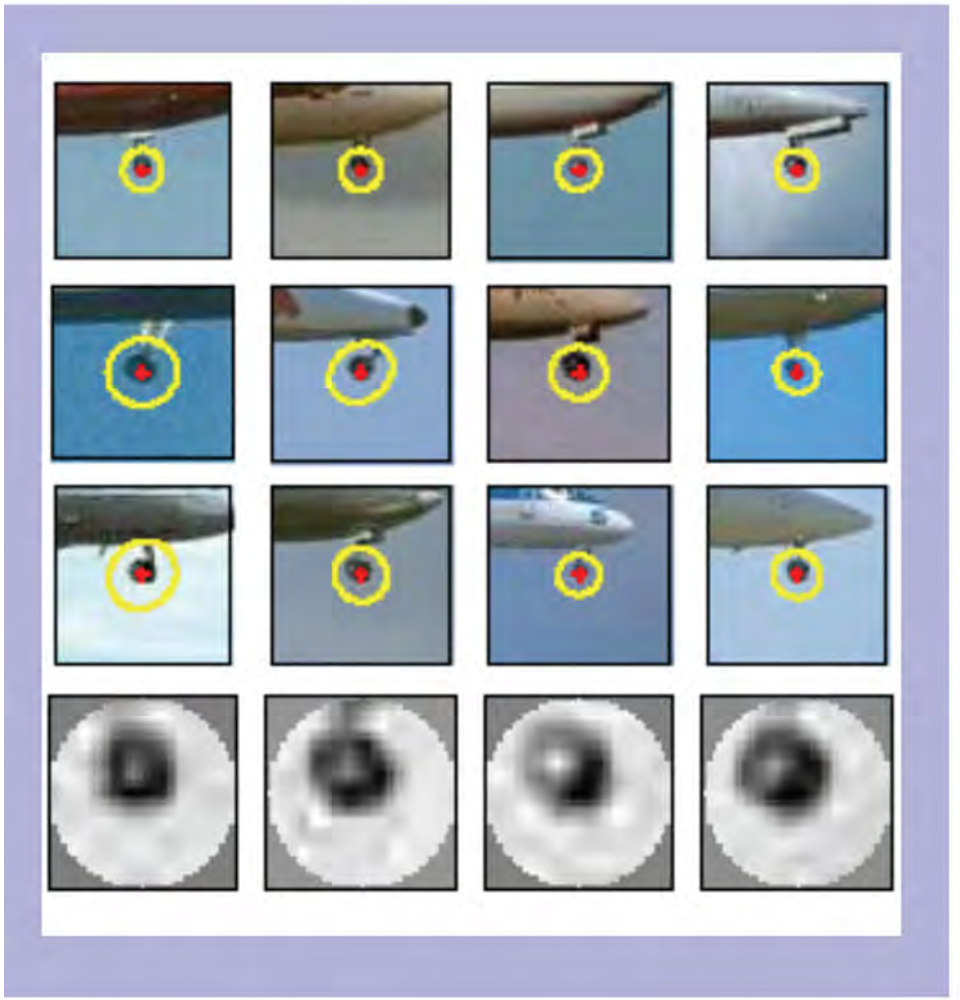


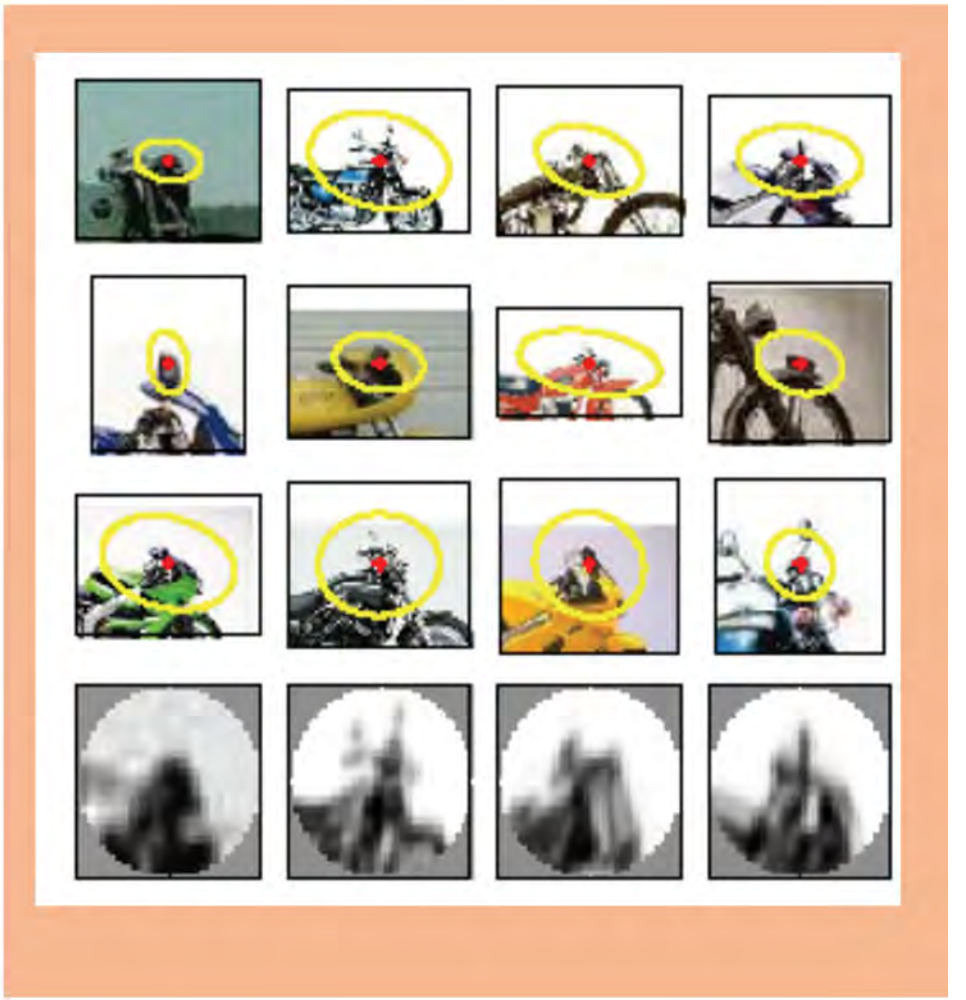
2. Codewords dictionary formation



Slide credit: Josef Sivic

Image patch examples of codewords

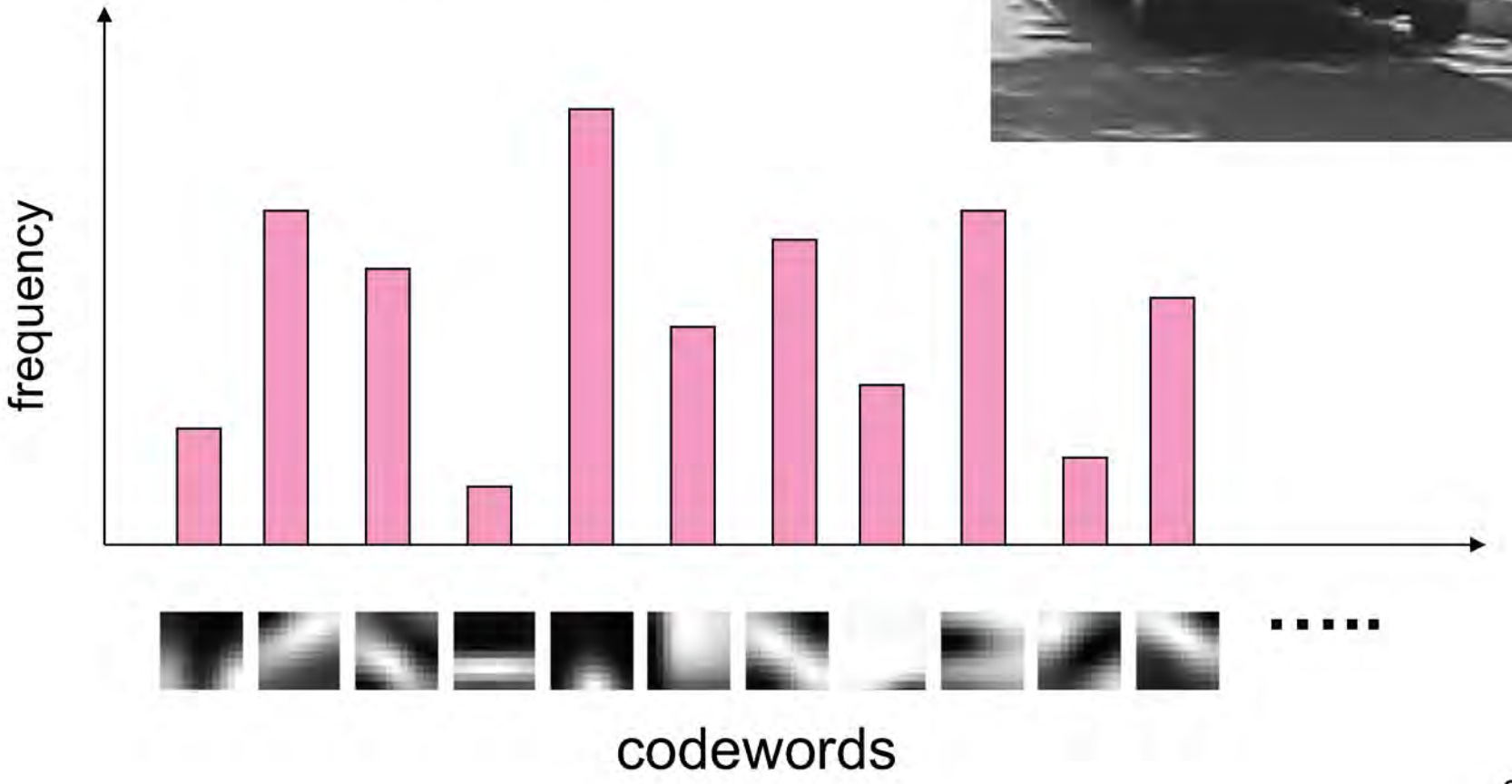




Sivic et al. 2005

Image representation

Histogram of features
assigned to each cluster

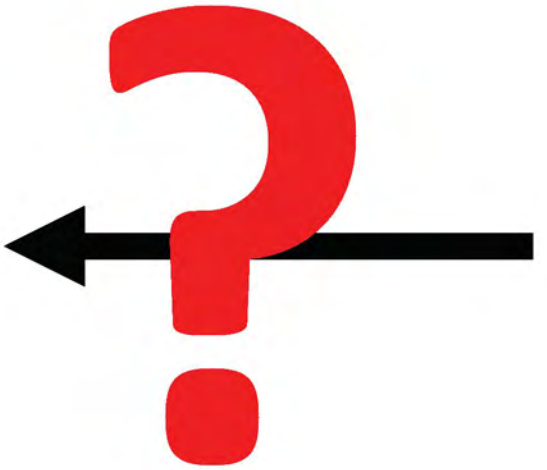


Uses of BoW representation

- Treat as feature vector for standard classifier
  - e.g k-nearest neighbors, support vector machine

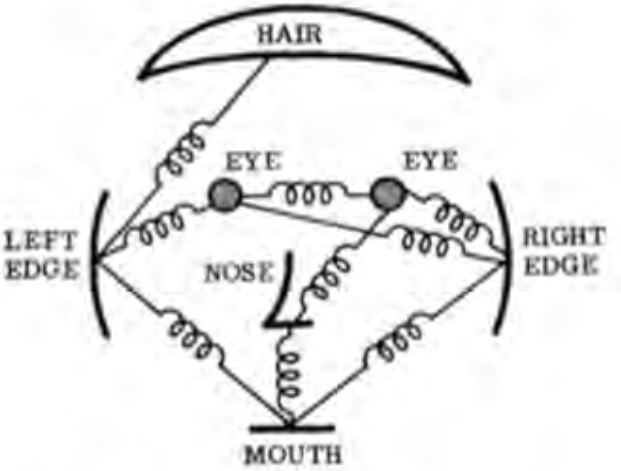
- Cluster BoW vectors over image collection
  - Discover visual themes

What about spatial info?



Deformable Part-based Models

- Felzenszwalb et al. PAMI'10
- Winner of the PASCAL detection challenge (2008,2009)
- Objects are decomposed into parts and spatial relations among parts



Fischler and Elschlager '73

Deformable Part-based Models

- Score of hypothesis

$$\text{score}(p_0, \ldots, p_n) = \underbrace{\sum_{i=0}^{n} F_i \cdot \phi(H, p_i)}_{\text{“data term”}} - \underbrace{\sum_{i=1}^{n} d_i \cdot (dx_i^2, dy_i^2)}_{\text{“spatial prior”}}$$

filters

displacements

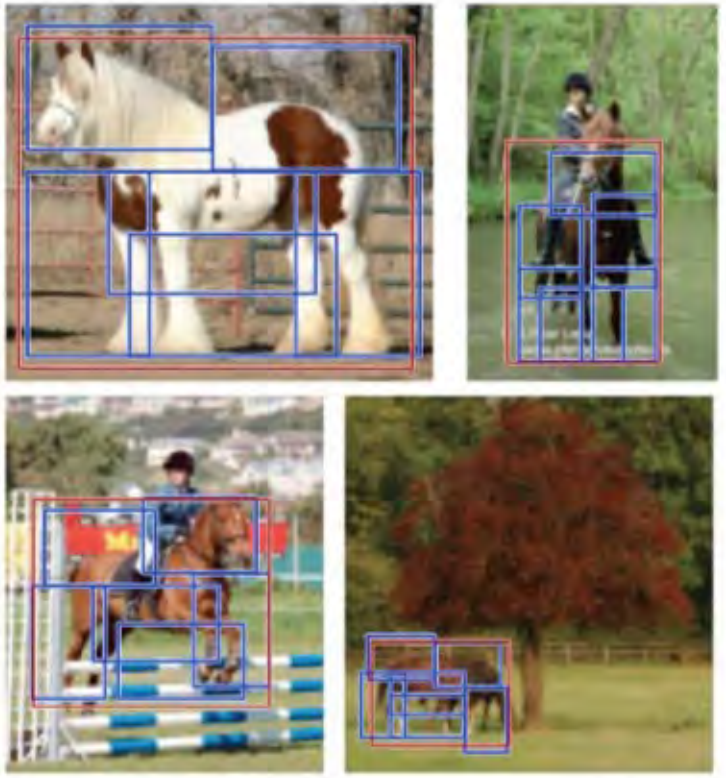
deformation parameters

Score is sum of filter scores minus deformation costs

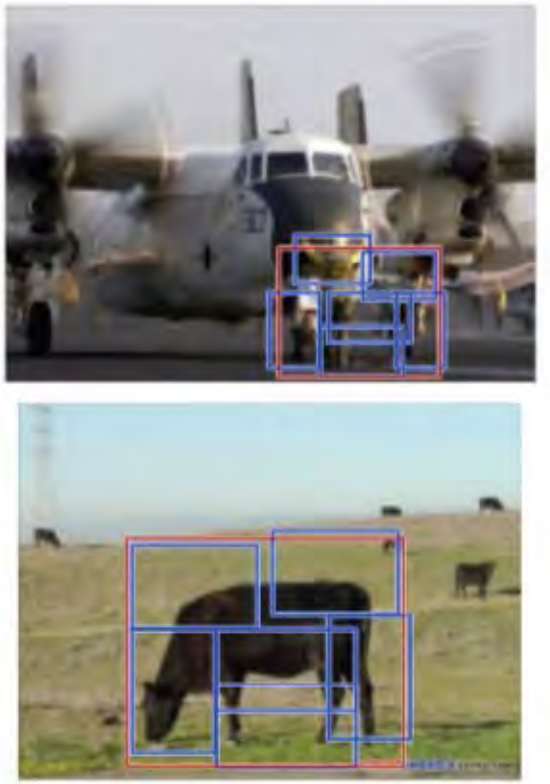
Deformable Part-based Models: Results

- Horse detections

high scoring true positives

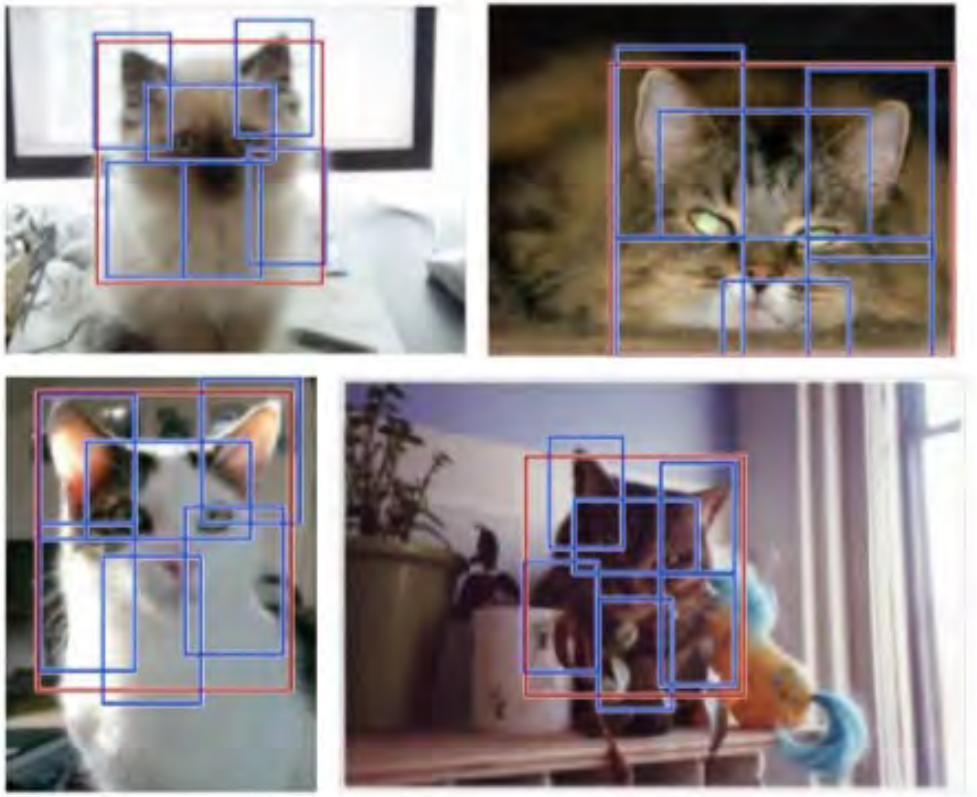


high scoring false positives

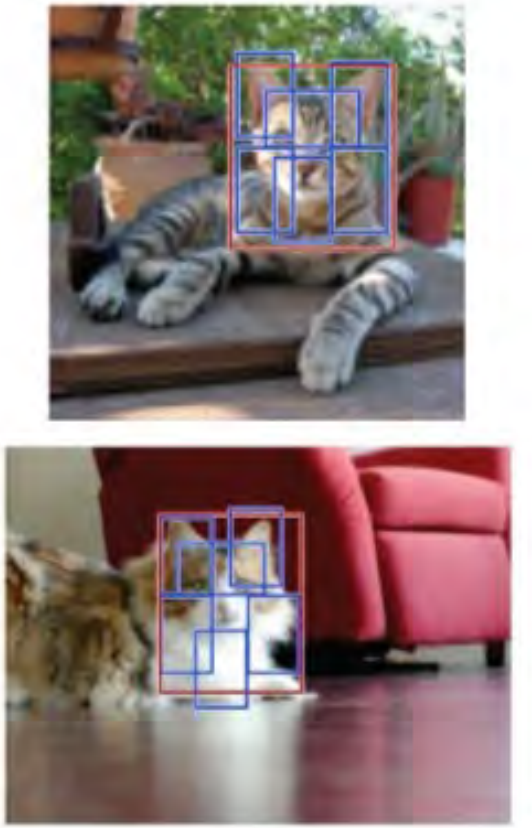


- Cat detections

high scoring true positives

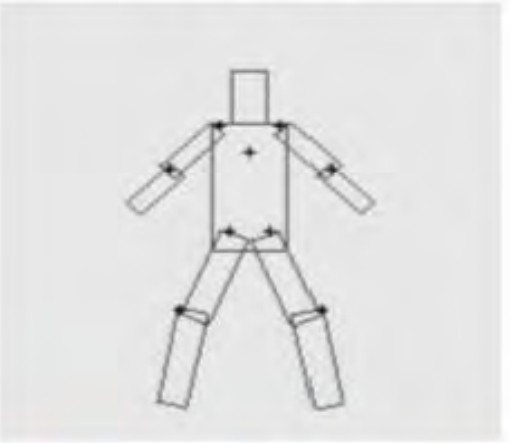


high scoring false positives
(not enough overlap)



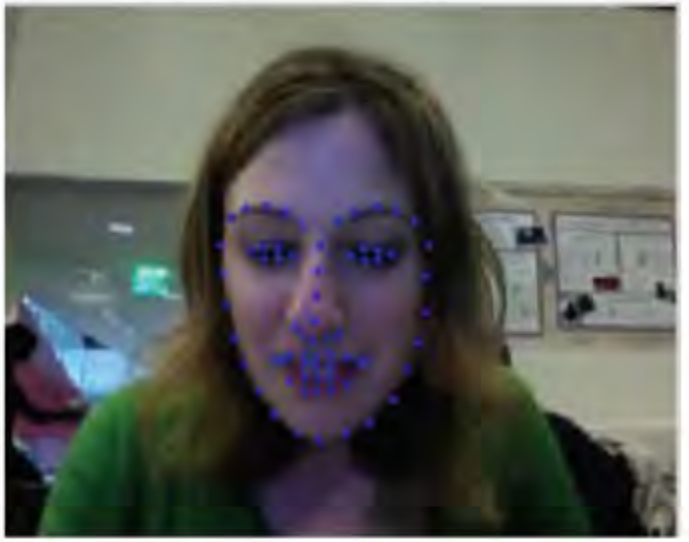
Part-based representation

- Tree model ➔ Articulated objects



Part-based models: Pose estimation

- Pose estimation in video (Ramanan et al, 2007; Yang et al, 2015)

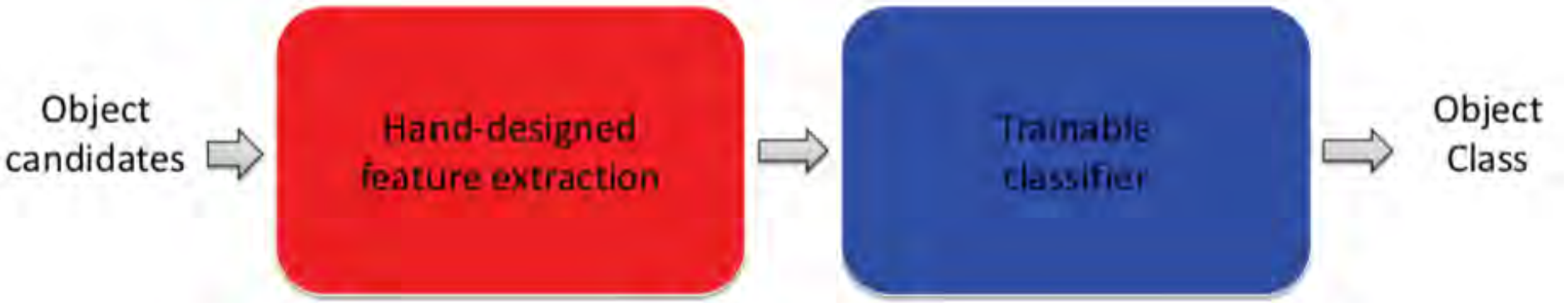


Face capturing

Dancing

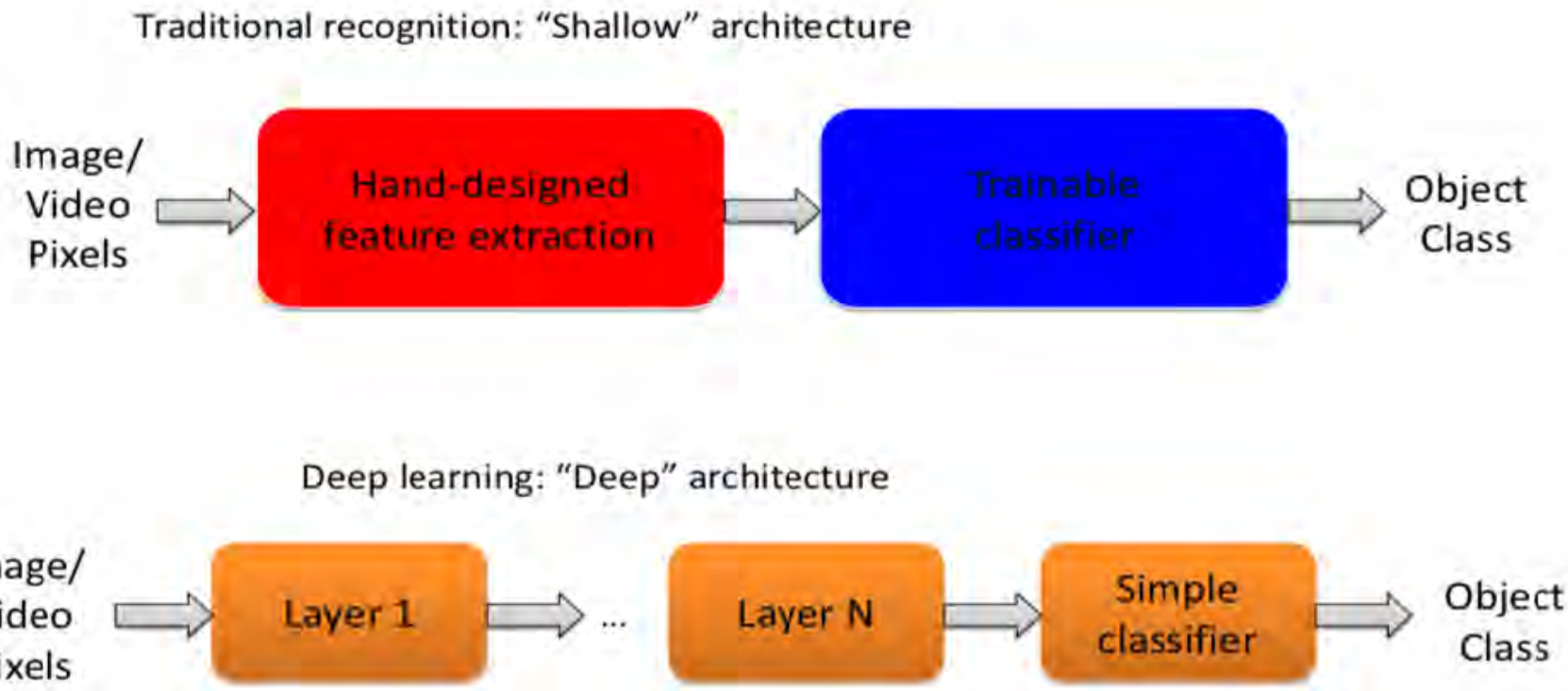
DPMs: Limitations

- Manually designed feature (HOG)
- Pre-defined object-part structure
- Shallow structure: limited representation capacity



Move into “deep learning” regime

Next: “deep” architectures

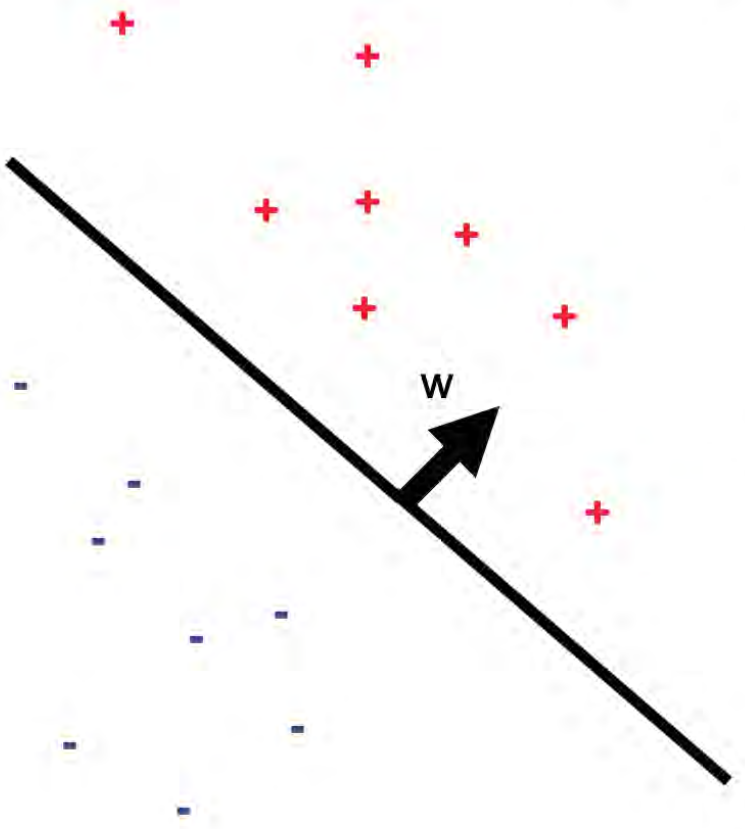


Readings

- M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991
- M. Turk and A. Pentland, Eigenfaces for recognition, Journal of Cognitive Neuroscience, 1991
- P. Viola and M. Jones, Robust Real-time Object Detection, IJCV 2001
- N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005
- P. Felzenszwalb, R. Girshick, D. McAllester and D. Ramanan, Object Detection with Discriminatively Trained Part Based Models, TPAMI 2010

Classifier design

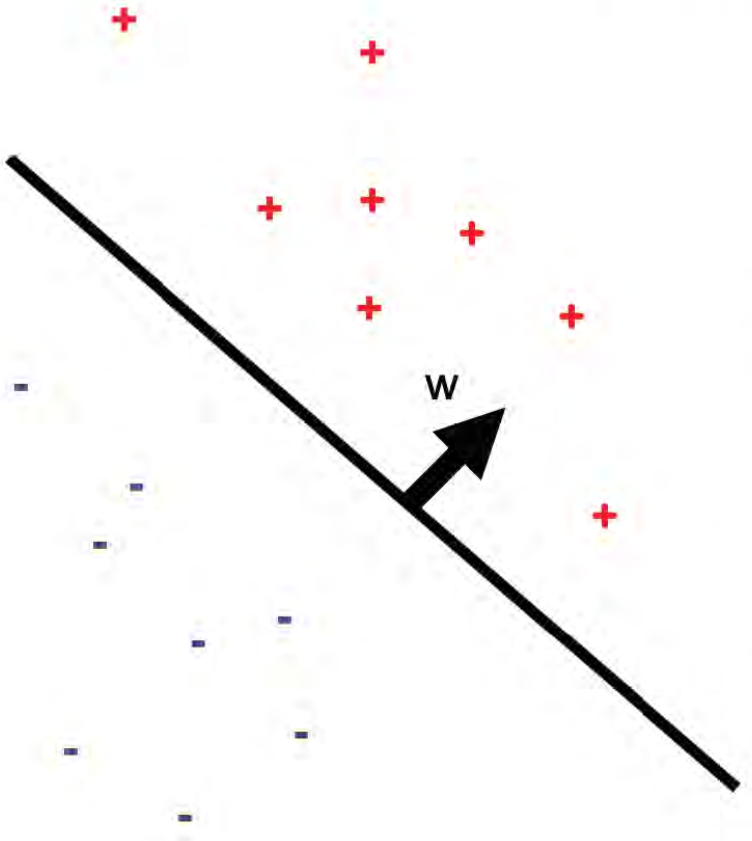
Classification



- Examples are points in R^n
- Positives are separated from negatives by the hyperplane **w**

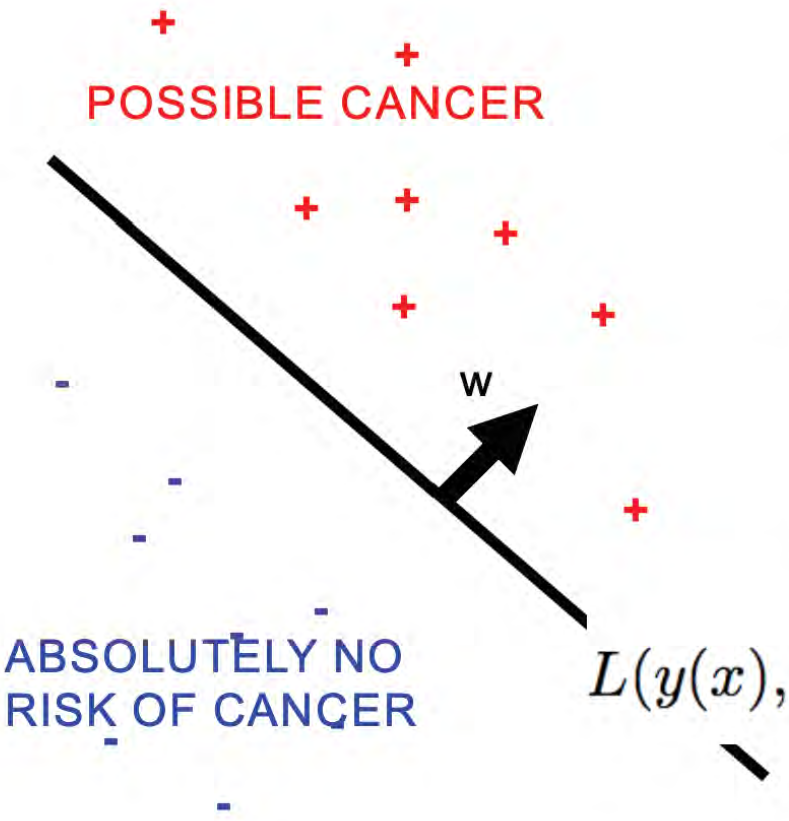
- $f(x)=\text{sign}(w^T x-b)$

Classification



- $x \in R^n$ - data points
- P(x) - distribution of the data
- y(x) - true value of y for each x
- F – classifier decision function:
  $y=F(x, \theta)$
- θ - parameters of F,
  e.g. $\theta=(w,b)$
- We want F that makes few mistakes

Loss function



- Our decision may have severe implications
- L(y(x),F(x, θ)) - loss function
  How much we pay for predicting F(x,θ), when the true value is y(x)
- Classification error:

$$L(y(x), F(x,\theta)) = \begin{cases} 0, & y(x) = sign(w^T x - b) \\ 1, & otherwise \end{cases}$$

- Hinge loss

$$L(y(x), F(x,\theta)) = max(0, 1 - y(x)F(x,\theta))$$

Learning

- Total loss shows how good a function (F, θ) is:

$$L(f) = \int_x L(y, F(x)) P(x) dx$$

- Learning is to find a function to minimize the loss:

$$(F, \theta) = \arg\min_{F,\theta} \int_x L(y, F(x, \theta)) P(x) dx$$

- How can we see all possible x?

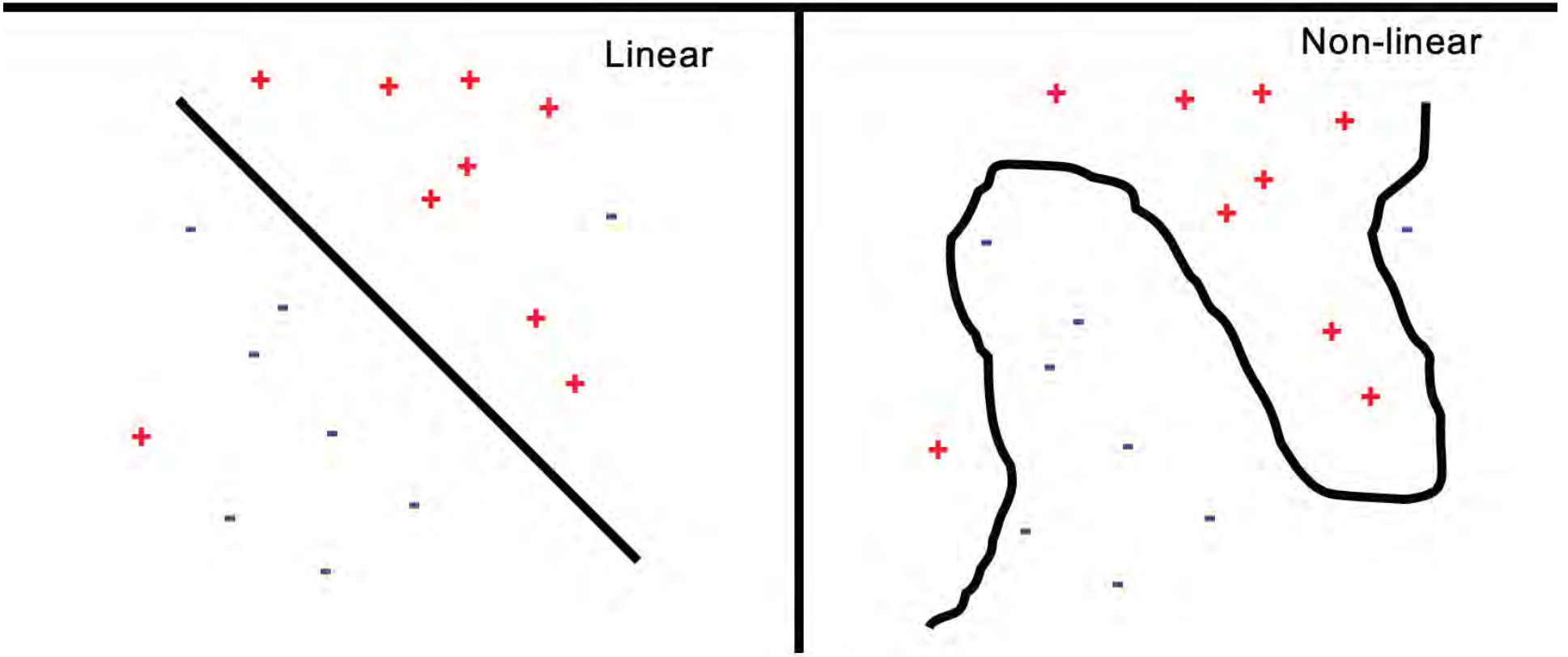
Datasets

- Dataset is a finite sample $\{x_i\}$ from P(x)
- Dataset has labels $\{(x_i,y_i)\}$
- Datasets today are big to ensure the sampling is fair

	#images	#classes	#instances
Caltech 256	30608	256	30608
Pascal VOC	4340	20	10363
LabelMe	176975	???	414687

Overfitting

- A simple dataset.
- Two models

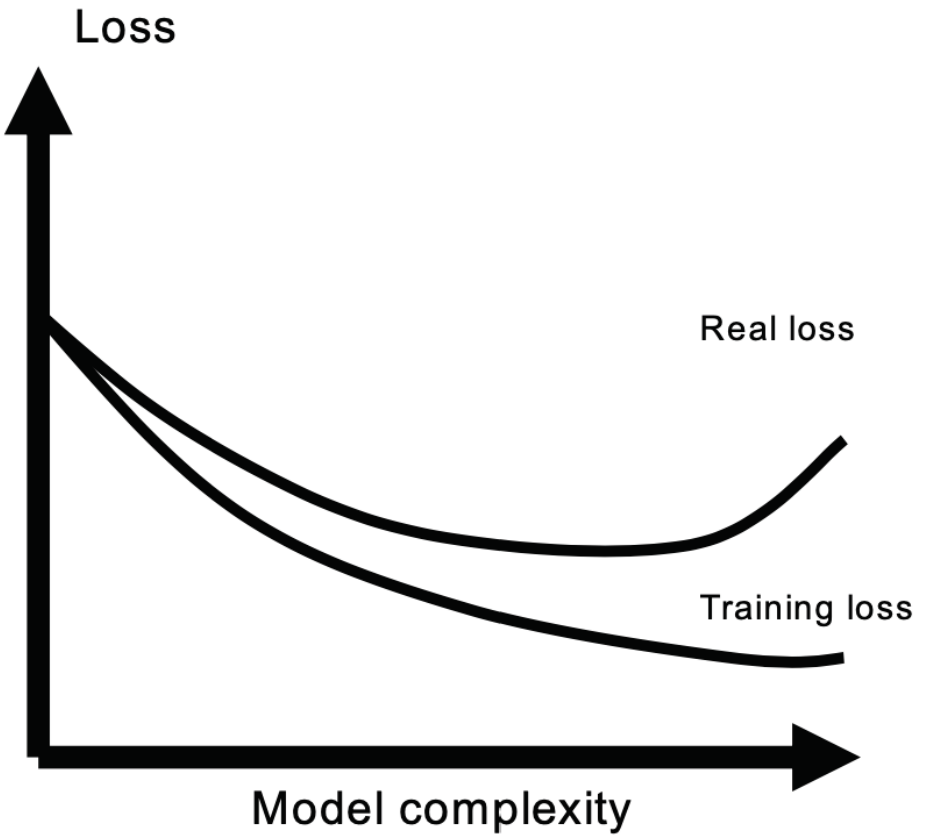


Overfitting

- Let's get more data.
- Simple model has better generalization.

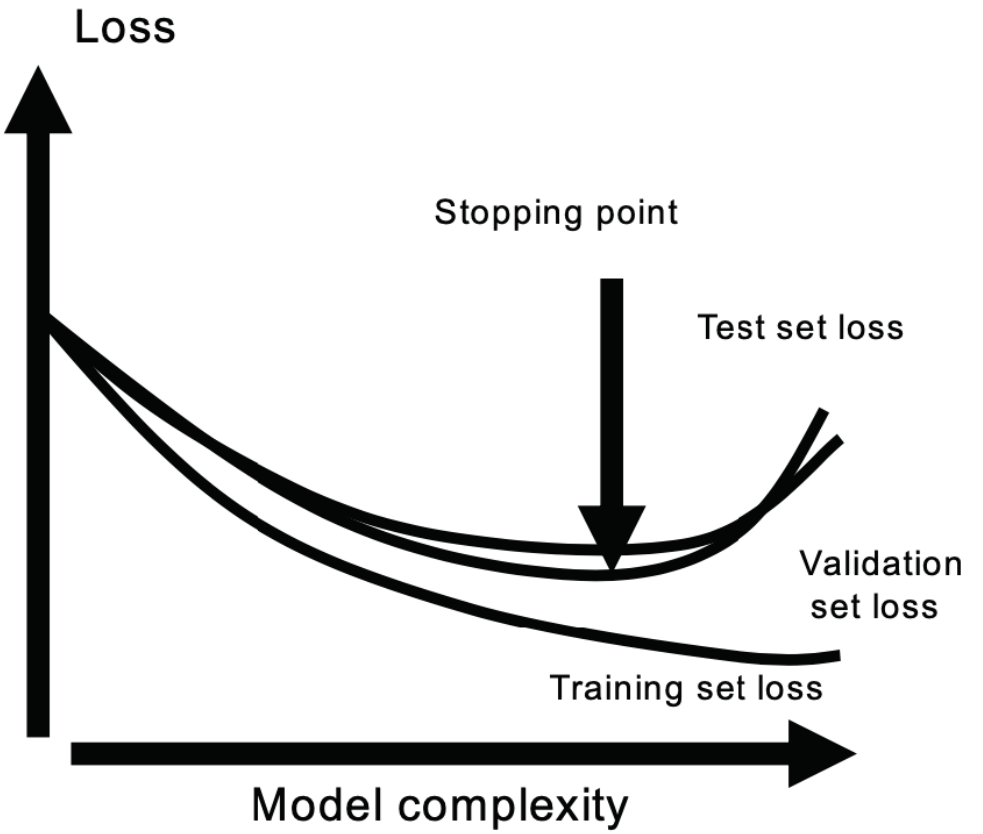
Overfitting

- As complexity increases, the model overfits the data
- Training loss decreases
- Real loss increases
- We need to penalize model complexity
  = to regularize

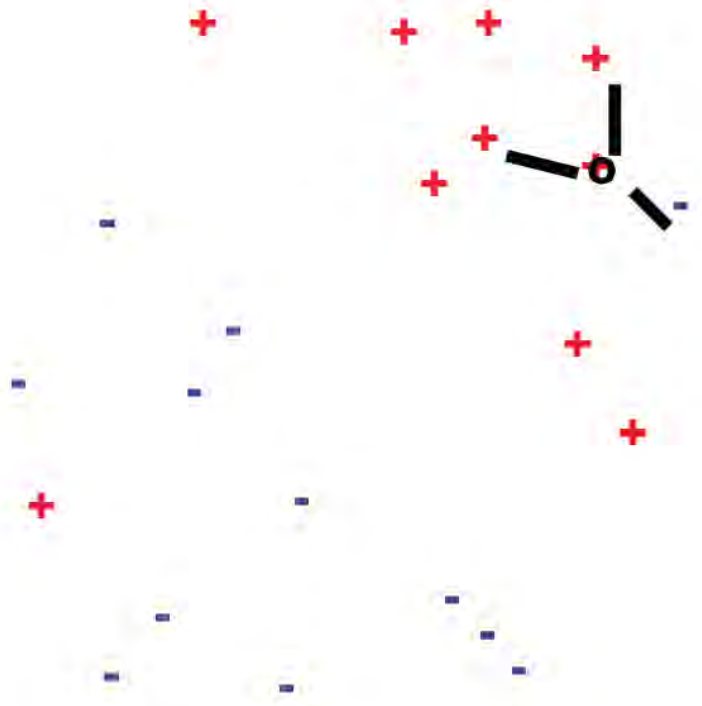


Overfitting

- Split the dataset
  - Training set
  - Validation set
  - Test set
- Use training set to **optimize** model parameters
- Use validation test to **choose** the best model
- Use test set only to **measure** the expected loss



K Nearest Neighbors

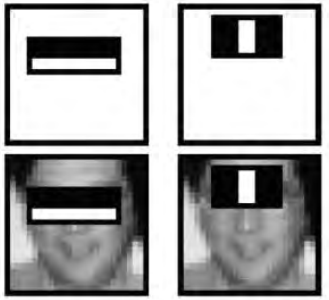


- Memorize all training data
- Find K closest points to the query
- The neighbors vote for the label:

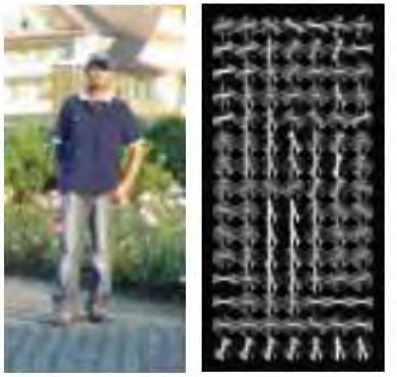
  Vote(+)=2

  Vote(–)=1

Case Studies



Boosting + face detection



SVM + person detection

Dalal & Triggs

Bag of Words model

Viola & Jones face detector