Computer Vision by Learning

Cees Snoek
Laurens van der Maaten
Arnold W.M. Smeulders

University of Amsterdam
Delft University of Technology

Advanced School for Computing and Imaging
University of Amsterdam
TU Delft

Overview – Day 1

1. Introduction, types of concepts, relation to tasks, invariance
2. Observables, color, space, time, texture, Gaussian family
3. Invariance, the need, invariants, color, SIFT, Harris, HOG
4. BoW overview, what matters
5. On words and codebooks, internal and local structure, soft assignment, synonyms, convex reduction, Fisher & VLAD

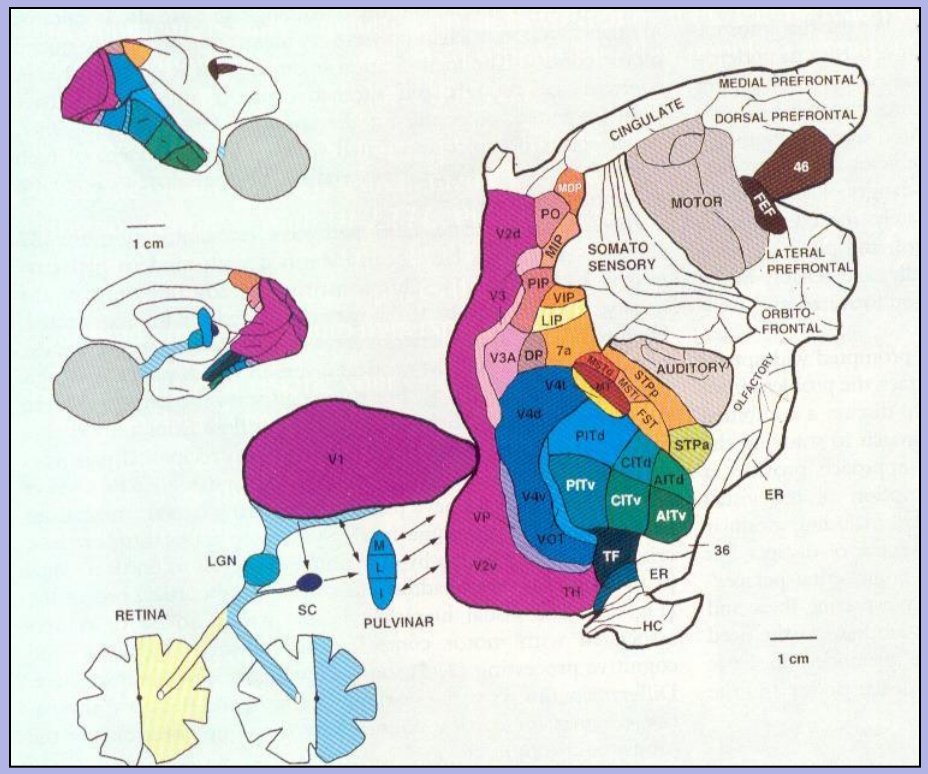
6. Object and scene classification, recap chapters 1 to 5.
7. Support vector machine, linear, nonlinear, kernel trick.
8. Codemaps, L2-norm for regions, nonlinear kernel pooling.

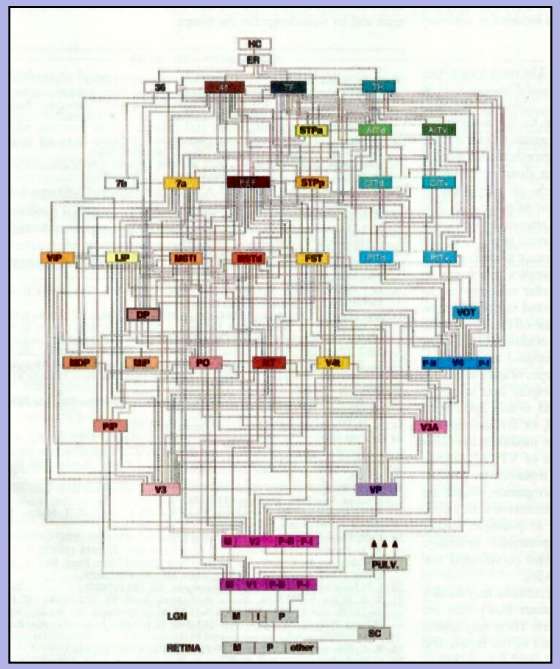
6. Object and scene classification

Computer vision by learning is important for accessing visual information on the level of objects and scene types. The common paradigm for object and scene detection during the past ten years rests on observables, invariance, bag of words, codebooks and labeled examples to learn from. We briefly summarize the first two lectures and explain what is needed to learn reliable object and scene classifiers with the bag of words paradigm.

How difficult is the problem?

Human vision consumes 50% brain power…





Van Essen, Science 1992

Object and scene classification

Testing: Does this image contain any bicycle?

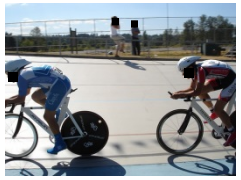
Object Classfication System

Bicycle

Training:

Bicycles



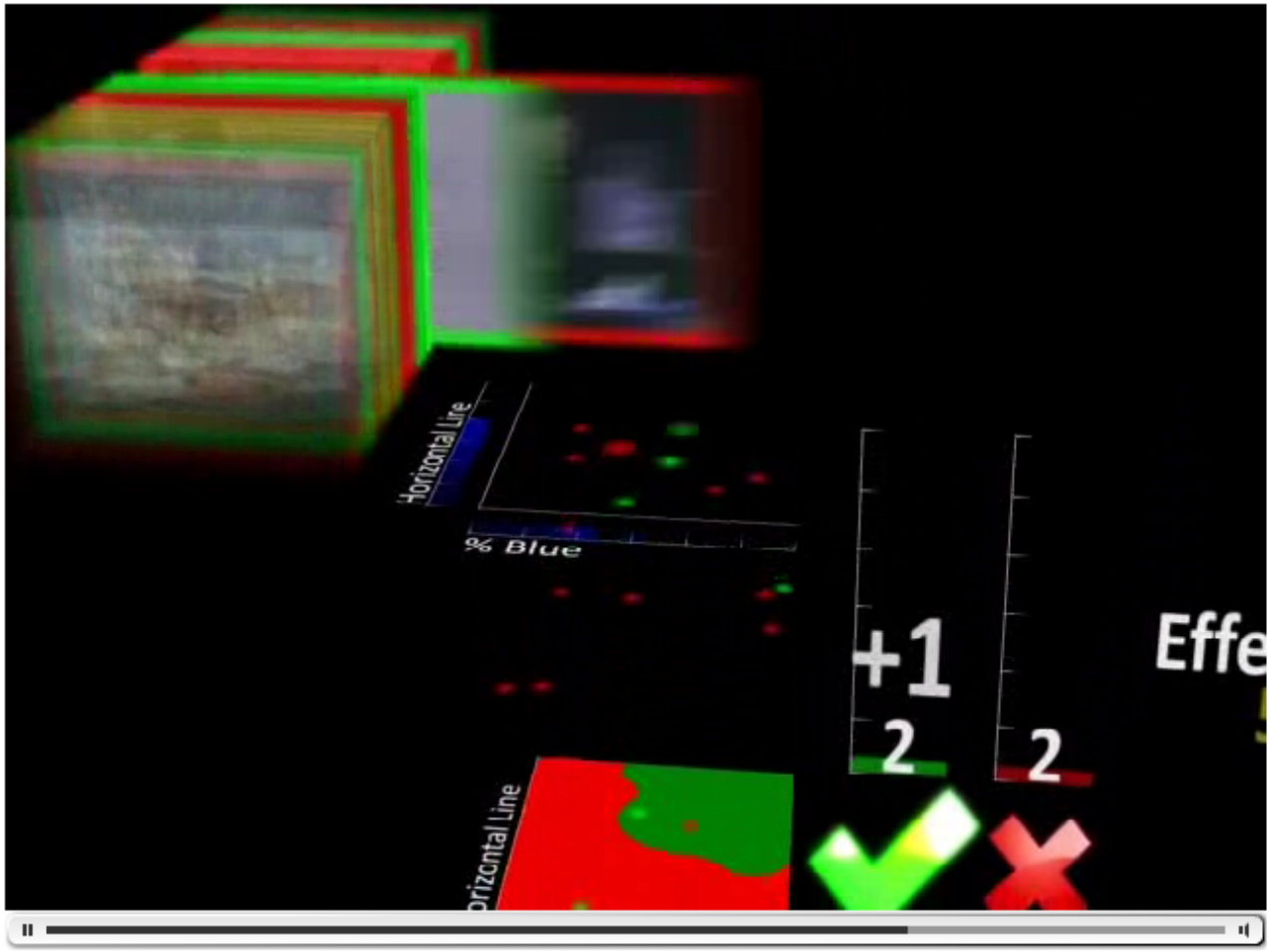



Not bicycles






Simple example



Visualization by Jasper Schulte

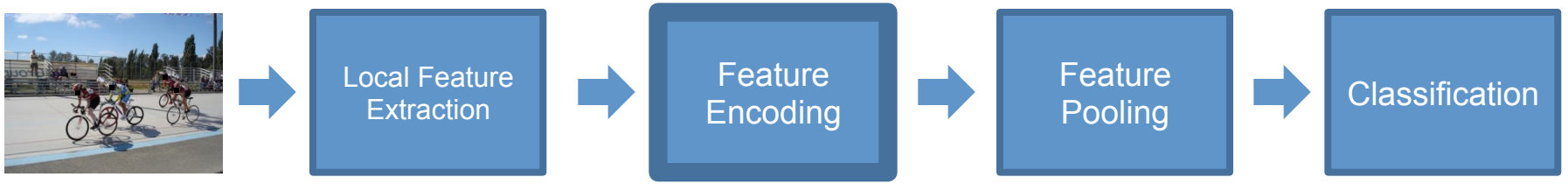
Object and scene classification

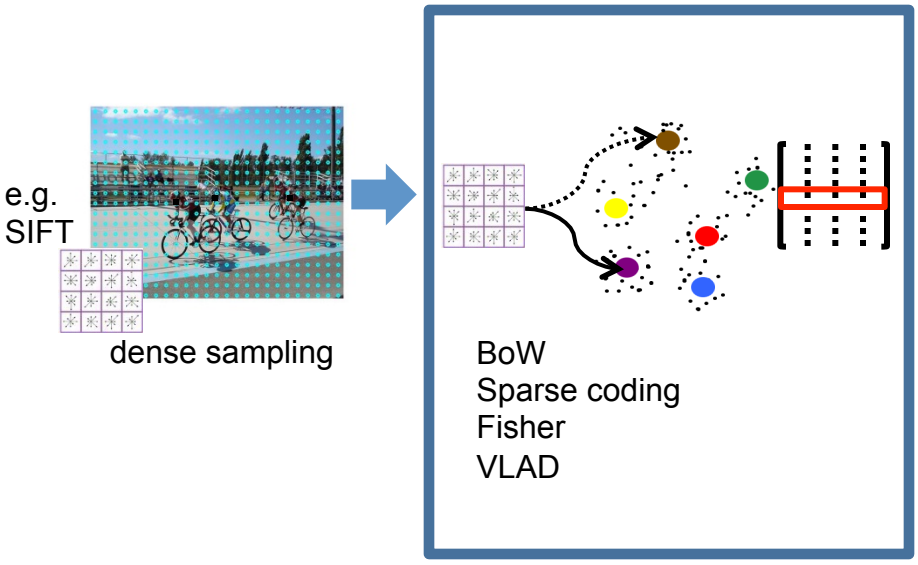
Local Feature Extraction → Feature Encoding → Feature Pooling → Classification

e.g.
SIFT

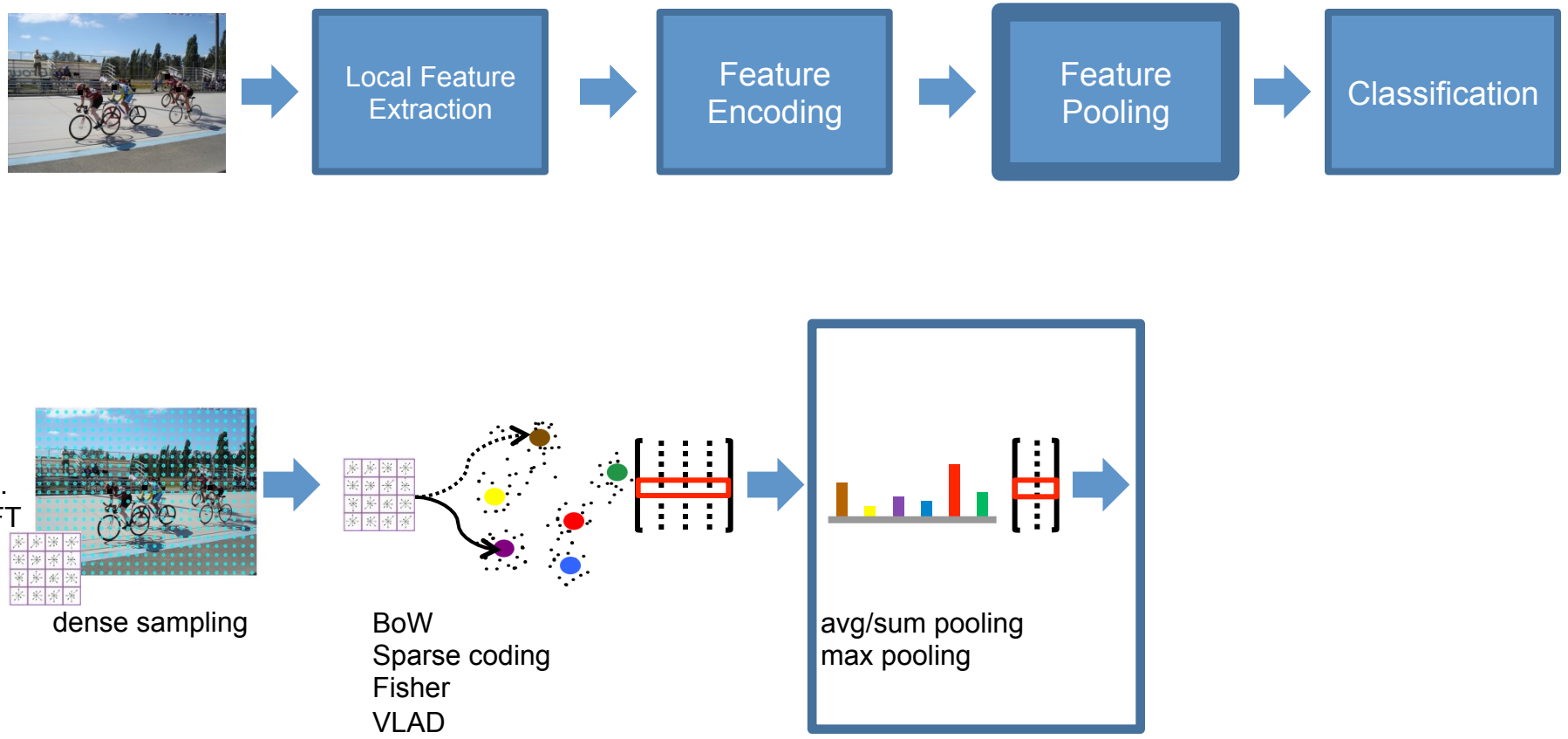
dense sampling

Object and scene classification

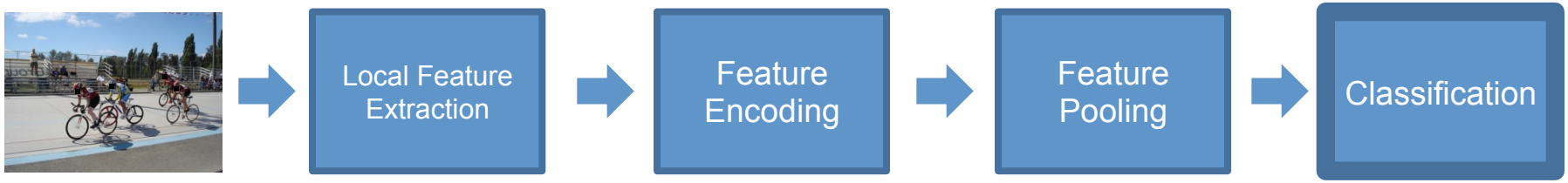


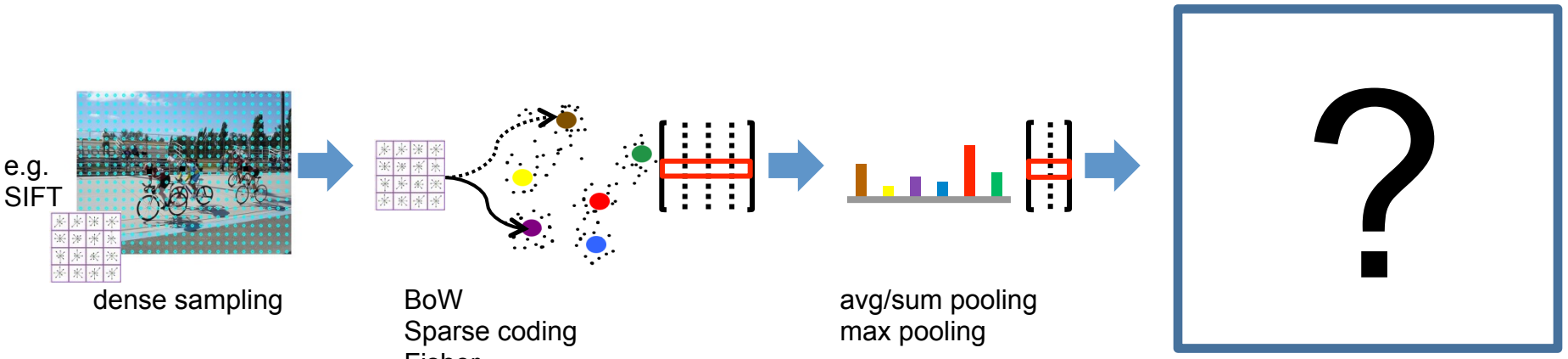


Object and scene classification



Object and scene classification





Classifiers

Nearest neighbor methods

Neural networks

Support vector machines

Randomized decision trees

…

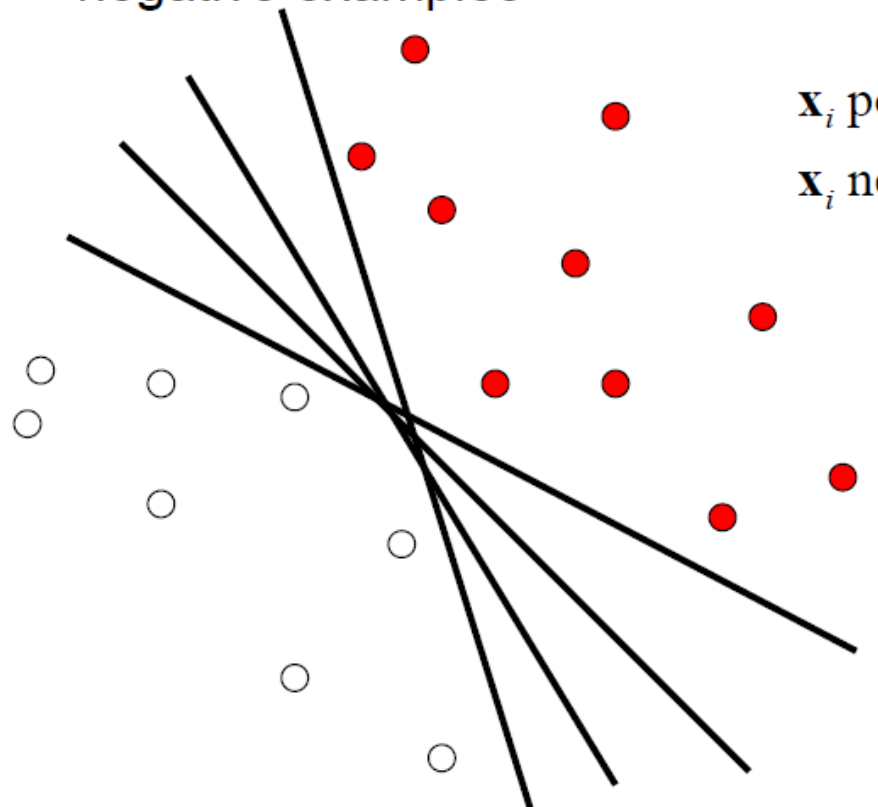
7. Support Vector Machine

The support vector machine separates an *n*-dimensional feature space into a class of interest and a class of disinterest by means of a hyperplane. A hyperplane is considered optimal when the distance to the closest training examples is maximized for both classes. The examples determining this margin are called the support vectors. For nonlinear margins, the SVM exploits the kernel trick. It maps the distance between feature vectors into a higher dimensional space in which the hyperplane separator and its support vectors are obtained as easy as in the linear case. Once the support vectors are known, it is straightforward to define a decision function for an unseen test sample.

Vapnik, 1995

Quiz: What linear classifier is best?

Find linear function (*hyperplane*) to separate positive and negative examples



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

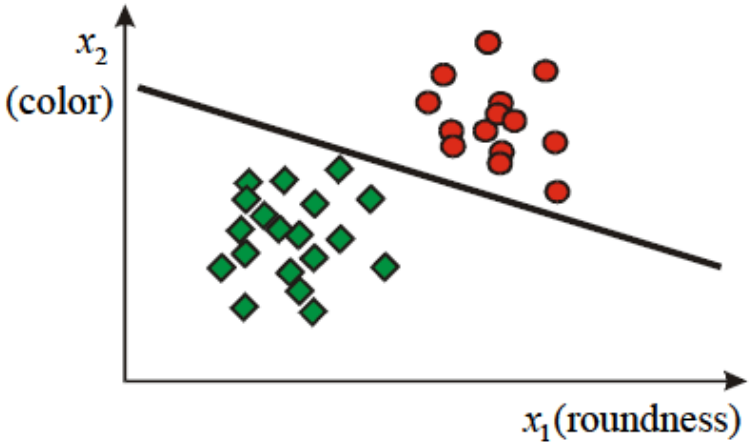
$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} + b < 0$

Which hyperplane is best?

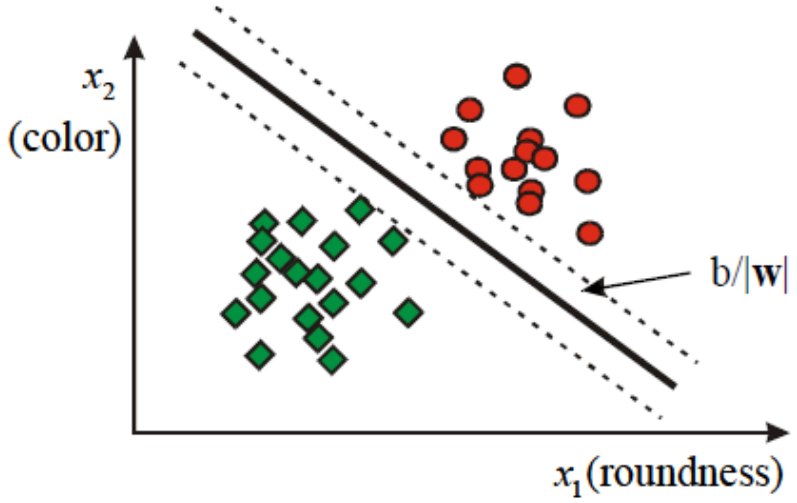
Slide credit: Cordelia Schmid

Linear classifiers - margin

Generalization is not good in this case:



Better if a margin is introduced:



Slide credit: Cordelia Schmid

Training a linear SVM

To find the maximum margin separator, we have to solve the following optimization problem:

$$\mathbf{w}.\mathbf{x}^c + b > +1 \quad for\ positive\ cases$$

$$\mathbf{w}.\mathbf{x}^c + b < -1 \quad for\ negative\ cases$$

$$and \quad \| \mathbf{w} \|^2 \ is\ as\ small\ as\ possible$$

Convex problem. Solved by quadratic programming.

Software available: LIBSVM, LIBLINEAR

Testing a linear SVM

The separator is defined as the set of points for which:

$$\mathbf{w}.\mathbf{x} + b = 0$$

$$so\ if\ \ \mathbf{w}.\mathbf{x}^c + b > 0\ \ say\ its\ a\ positive\ case$$

$$and\ if\ \ \mathbf{w}.\mathbf{x}^c + b < 0\ \ say\ its\ a\ negative\ case$$

L2 Normalization

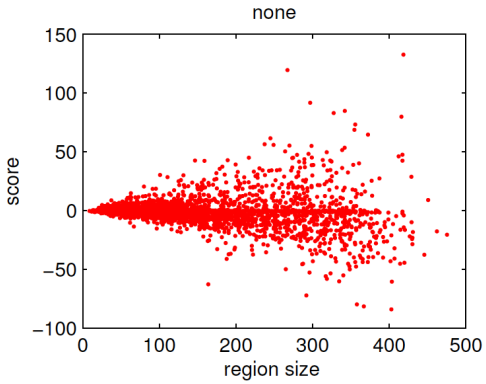
Linear classifier for object and scene classification prefers L2 normalization [Vedaldi ICCV09]

$$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2},$$

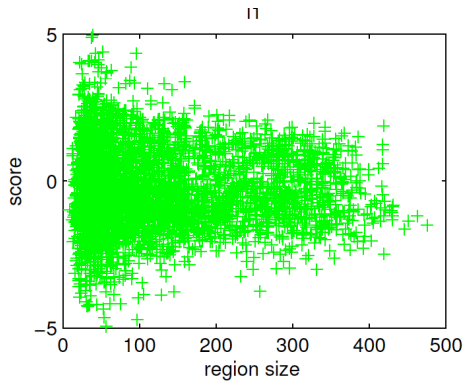
Important for Fisher vector

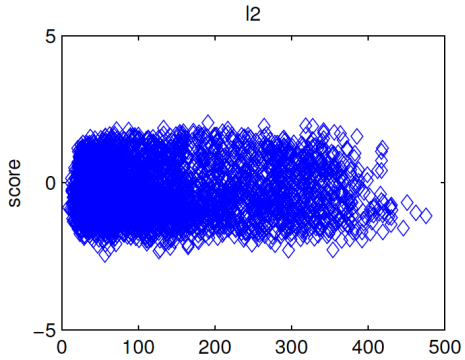
Acts as scale invariant

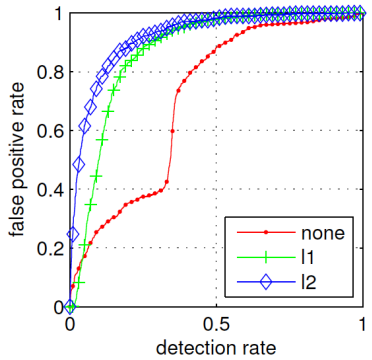
Large object bias



Small object bias

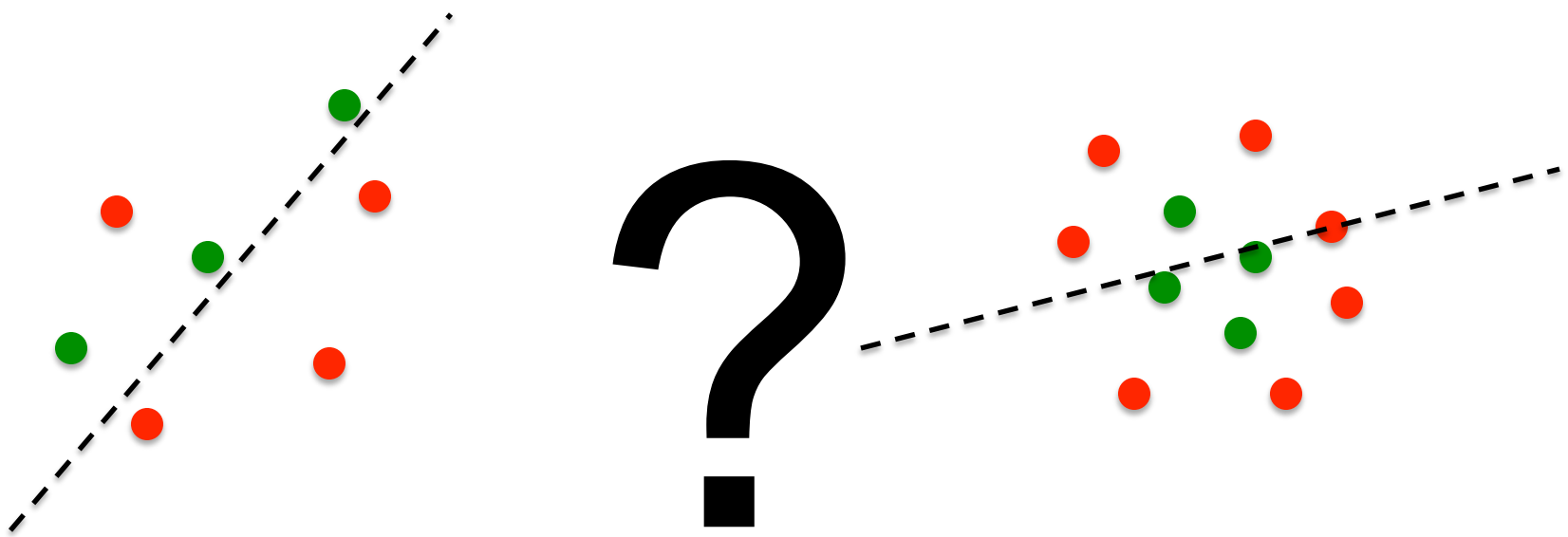






No scale bias

Quiz: What if data is not linearly separable?



Solutions for non separable data

1. Slack variables

2. Feature transformation

1. Introducing slack variables

Slack variables are constrained to be non-negative. When they are greater than zero they allow us to cheat by putting the plane closer to the datapoint than the margin. So we need to **minimize the amount of cheating**. This means we have to pick a value for lambda

$$\mathbf{w}.\mathbf{x}^c + b \geq +1 - \xi^c \quad \textit{for positive cases}$$

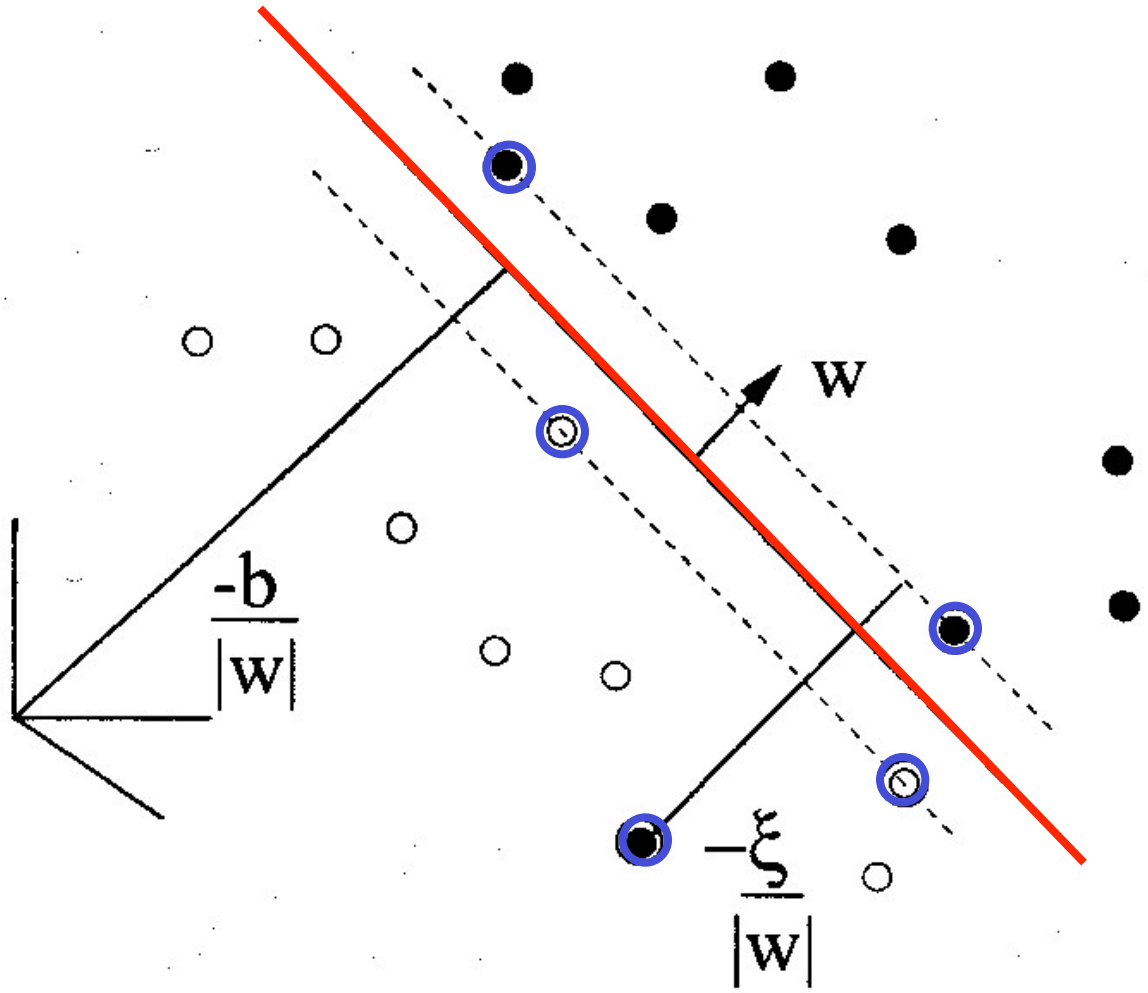
$$\mathbf{w}.\mathbf{x}^c + b \leq -1 + \xi^c \quad \textit{for negative cases}$$

$$\textit{with} \quad \xi^c \geq 0 \quad \textit{for all } c$$

$$\textit{and} \quad \frac{\|\mathbf{w}\|^2}{2} + \lambda \sum_c \xi^c \quad \textit{as small as possible}$$

Slide credit: Geoff Hinton

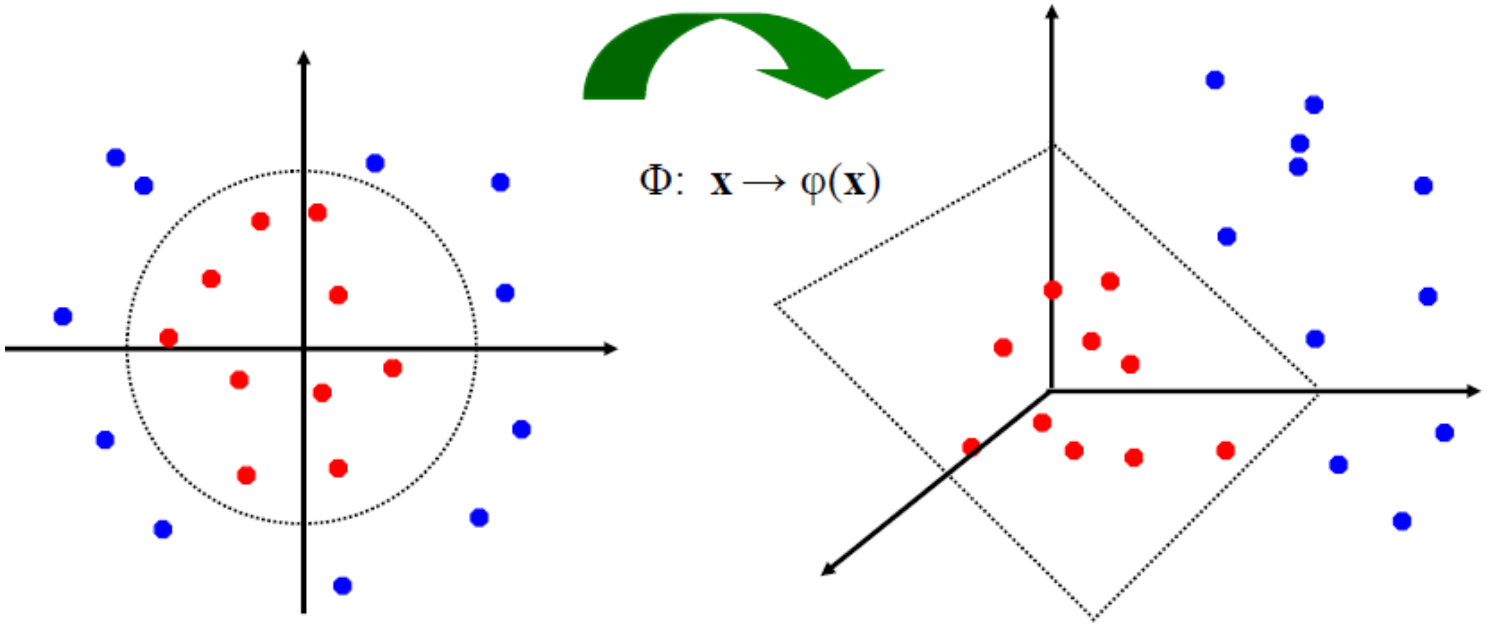
Separator with slack variable



Slide credit: Geoff Hinton

2. Feature transformations

Transform the feature space in order to achieve linear separability after the transformation.



Slide credit: Andrew Moore

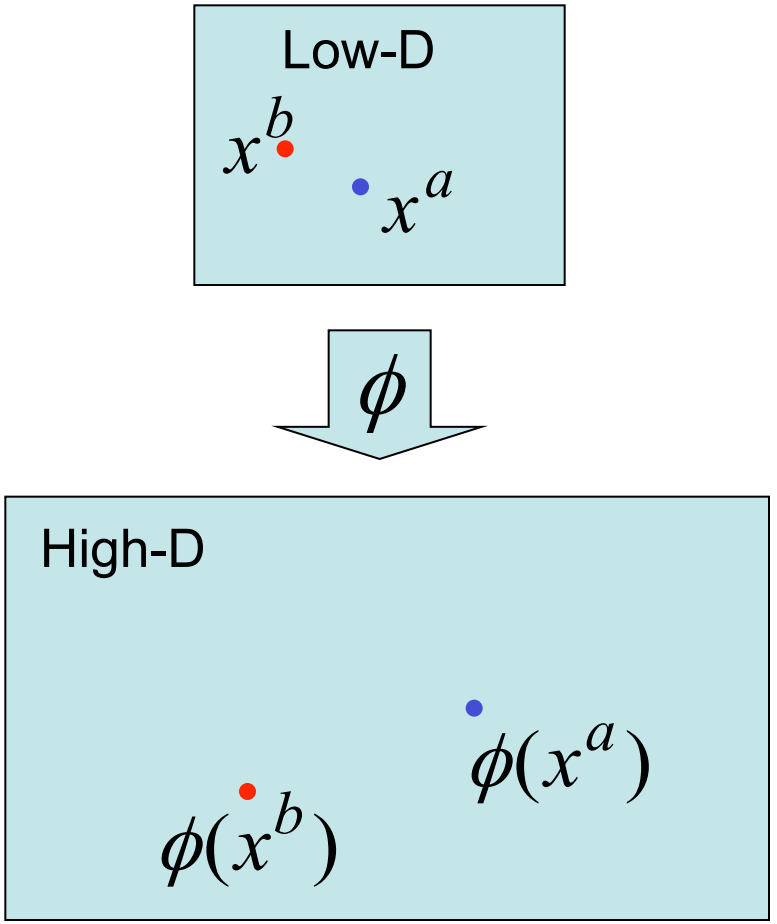
The kernel trick

For many mappings from a low-D space to a high-D space, there is a simple operation on two vectors in the low-D space that can be used to compute the scalar product of their two images in the high-D space.

$$K(x^a, x^b) = \phi(x^a).\phi(x^b)$$

Letting the kernel do the work

doing the scalar product in the obvious way



Slide credit: Geoff Hinton

The classification rule

The final classification rule is quite simple:

$$bias + \sum_{s \,\varepsilon\, SV} w_s K(x^{test}, x^s) \quad > \quad 0$$

The set of support vectors

All the cleverness goes into selecting the support vectors that maximize the margin and computing the weight to use on each support vector.

.

Slide credit: Geoff Hinton

Popular kernels for computer vision

Histogram intersection kernel:

$$I(h_1, h_2) = \sum_{i=1}^{N} \min(h_1(i), h_2(i))$$

Generalized Gaussian kernel:

$$K(h_1, h_2) = \exp\left(-\frac{1}{A} D(h_1, h_2)^2 \right)$$

D can be Euclidean distance, χ^2 distance, Earth Mover's Distance, etc.

$$D(h_1, h_2) = \sum_{i=1}^{N} \frac{(h_1(i) - h_2(i))^2}{h_1(i) + h_2(i)}$$

Slide credit: Cordelia Schmid

Quiz: linear vs non-linear kernels

	Linear	Non-linear
Training speed		
Training scalability		
Testing speed		
Test accuracy		

Quiz: linear vs non-linear kernels

	Linear	Non-linear
Training speed	Very fast	Very slow
Training scalability	Very high	Low
Testing speed	Very fast	Very slow
Test accuracy	Lower	Higher

Slide credit: Jianxin Wu

Nonlinear kernel speedups

Many have proposed speedups for nonlinear kernels.

Exploiting two basic properties:

Additivity

$$K(p, q) = K(p_1, q_1) + \cdots + K(p_D, q_D)$$

Homogeneity

$$K(c \cdot p, c \cdot q) = c^{\gamma} K(p, q)$$

Nonlinear as fast as linear kernel exploiting additivity

Feature maps for all additive homogeneous kernels.

Maji et al. PAMI 2013

Vedaldi et al. PAMI 2012

Selecting and weighting dimensions

For additive kernels *all dimensions are equal*

$$K_{\mathbf{X},\,\mathbf{Y}} = 1 \cdot k^1_{\mathbf{X},\,\mathbf{Y}} + \cdots + 1 \cdot k^D_{\mathbf{X},\,\mathbf{Y}}$$

We introduce scaling factor *c*

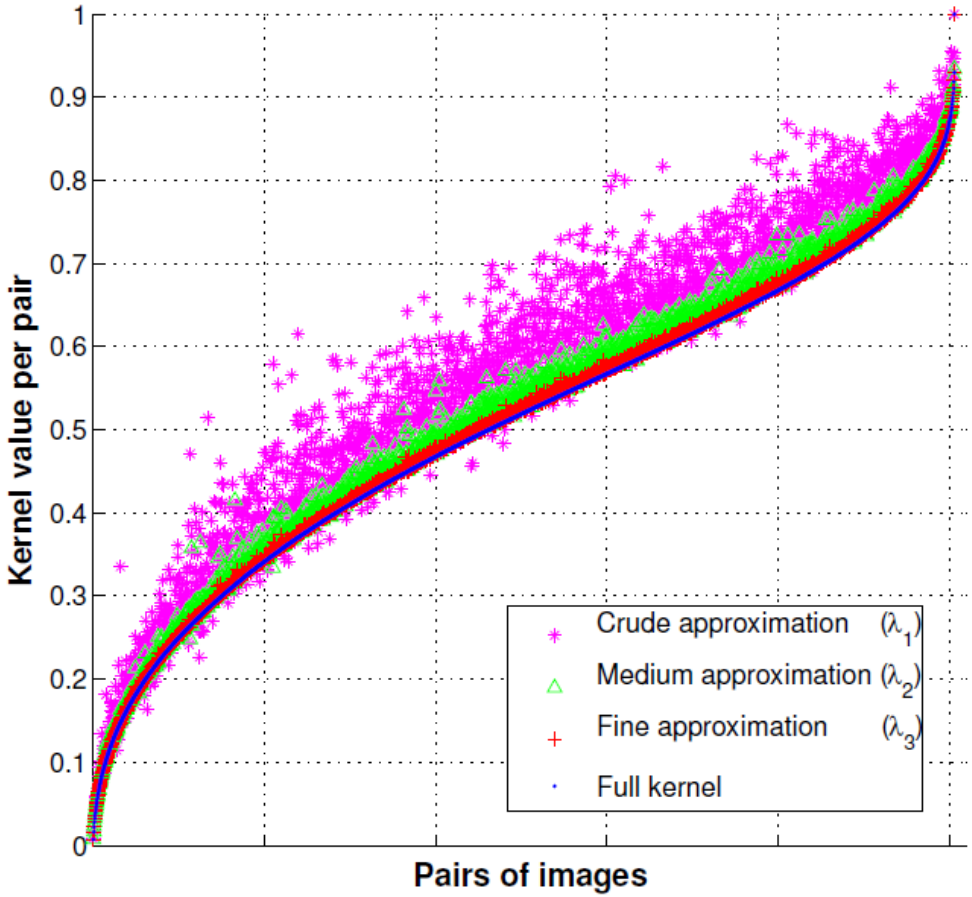
$$K_{\mathbf{X},\,\mathbf{Y}} = c_1 \cdot k^1_{\mathbf{X},\,\mathbf{Y}} + \cdots + c_D \cdot k^D_{\mathbf{X},\,\mathbf{Y}} = \mathbf{c}^T \cdot \mathbf{k}_{\mathbf{X},\,\mathbf{Y}}$$

Kernel reduction as convex optimization problem

$$\arg\min_{\mathbf{c}} \left\| \mathbf{K}_{\mathbf{X},\,\mathbf{Y}} - \mathbf{c}^T \cdot \mathbf{k}_{\mathbf{X},\,\mathbf{Y}} \right\|^2 + \lambda \left\| \mathbf{c} \right\|_{\ell 1}$$

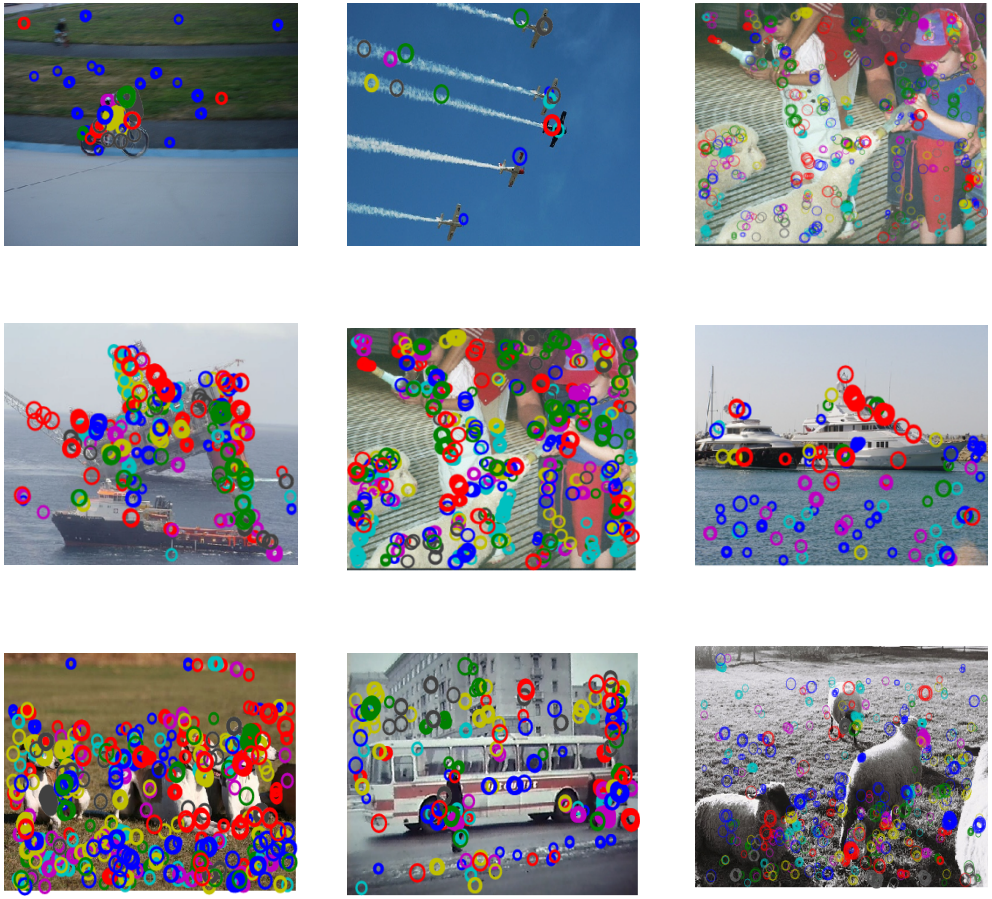
Convex reduced kernels

Similar accuracy with a 45-85% smaller size.



Equally accurate and 10x faster as PCA codebook reduction.
Applies also to Fisher vectors.

Selected kernel dimensions



Note: descriptors originally dense sampled

Performance

Support Vector Machines work very well in practice.

- The user must choose the kernel function and its parameters, but the rest is automatic.
- The test performance is very good.

They can be expensive in time and space for big datasets

- The computation of the maximum-margin hyper-plane depends on the square of the number of training cases.
- We need to store all the support vectors.
- Exploit kernel additivity and homogenity for speedup

SVM's are very good if you have no idea about what structure to impose on the task.

Quiz: what is remarkable about bag-of-words with SVM?

Local Feature Extraction → Feature Encoding → Feature Pooling → Kernel Classification

Bag-of-words ignores locality

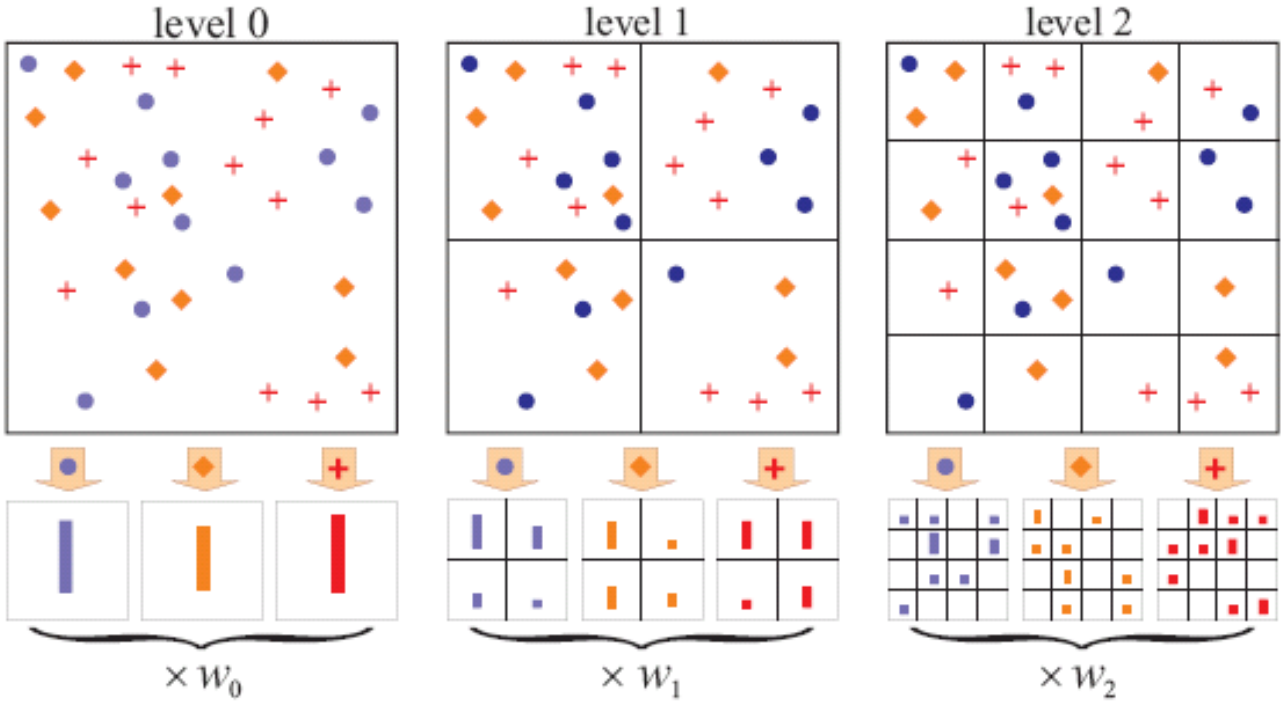
Solution: spatial pyramid

– aggregate statistics of local features over fixed subregions

Grauman, ICCV 2005, Lazebnik, CVPR 2006

Spatial pyramid kernel

For **homogeneous** kernels the spatial pyramid is simply obtained by concatenating the appropriately weighted histograms of all channels at all resolutions.



Lazebnik, CVPR 2006

Problem posed by Hinton

Suppose we have images that may contain a tank, but with a cluttered background.

To recognize which ones contain a tank, it is no good computing a global similarity

We need local features that are appropriate for the task.

Its very appealing to convert a learning problem to a convex optimization problem, but we may end up by ignoring aspects of the real learning problem in order to make it convex.

8. Codemaps

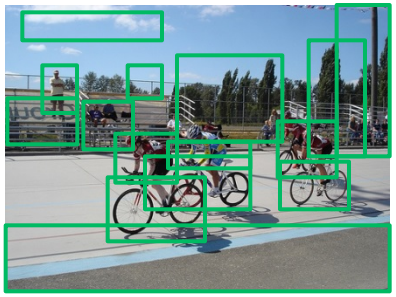
Codemaps integrate locality into the bag-of-words paradigm. Codemaps are a joint formulation of the classification score and the local neighborhood it belongs to in the image. We obtain the codemap by reordering the encoding, pooling and SVM classification steps over lattice elements. Codemaps include L2 normalization for arbitrarily shaped image regions and embed nonlinearities by explicit or approximate feature mappings. Many computer vision by learning problems may profit from codemaps.

Slides Credit: Zhenyang Li ICCV13

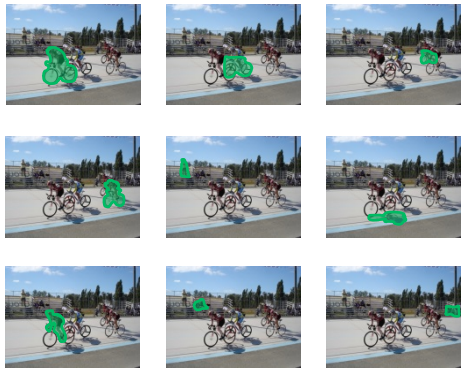
Local object classification

Requires **repetitive** computations on **overlapping** regions

Spatial Pyramids [*Lazebnik, CVPR06*]
(#regions: 10-100)



Object Detection [*Sande, ICCV11*]
(#regions: 1,000-10,000)



Semantic Segmentation [*Carreira, CVPR09*]
(#regions: 100-1,000)

Repeat for each region

Local Feature Extraction → Feature Encoding → Feature Pooling → Kernel Classification

Decompose BoW + linear SVM

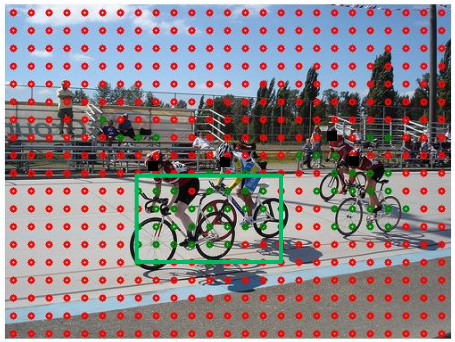
Efficient window/region search for detection

if feature c_i mapped into j-th word

$$y(R) = \sum_{j=1}^{K} w^j \sum_{i=1}^{M} c_i^j = \sum_{i=1}^{M} y_i, \quad for\ y_i = \sum_{j=1}^{K} w^j c_i^j$$

SVM weight for j-th word

Per-descriptor classification score



Lampert, PAMI09; Vijayanarasimhan, CVPR11

Problem 1: **Kernel classifier requires normalization**

- Linear classifier prefers L2 normalization [Vedaldi, ICCV09]

Problem 2: **Object classification profits from nonlinearities**

- BoW+Intersection Kernel [Maji, ICCV09]
- Fisher+power norm [Perronnin, ECCV10]

Codemaps

Decomposes any encoding with sum pooling + linear classifier

L2 normalization for arbitrarily shaped image regions

Nonlinearities by local kernel pooling for object classification

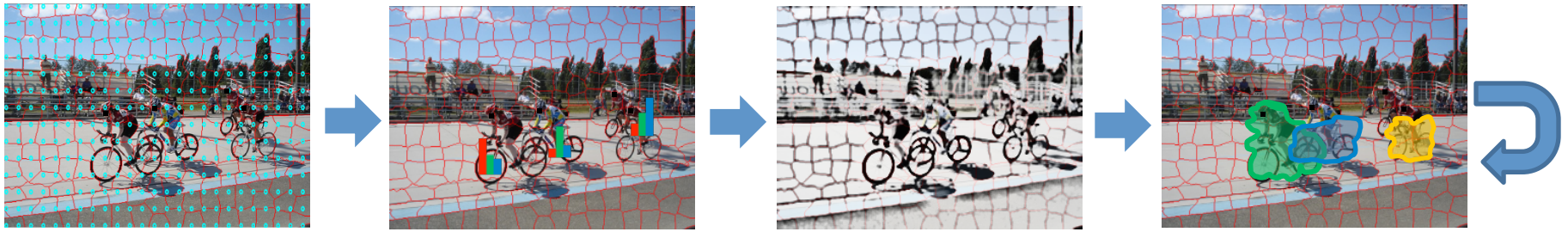
Codemaps

Lattice $G = \{g_i\}, \quad i = 1, \ldots, N$; Sum pooling $h = \sum \cdot$; Linear classifier $f = w^T \cdot$

Goal: reorder the encoding, pooling, classification of general object classification

Decomposition

Lattice $G = \{g_i\}, \quad i = 1, \ldots, N$; Sum pooling $h = \sum \cdot$; Linear classifier $f = w^T \cdot$



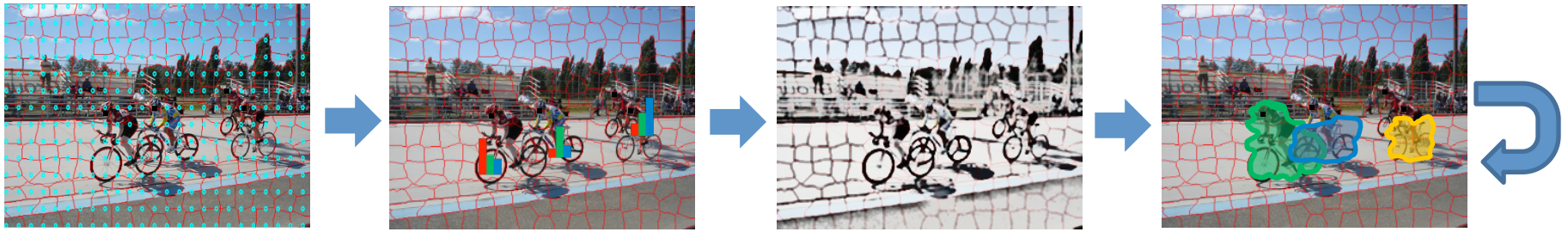
Feature Encoding | Lex Pooling | Lex Classification | Classification Pooling

$$f(h(R)) = w^T \sum_{i=1}^{l} h(g_i) = \sum_{i=1}^{l} w^T h(g_i) = \sum_{i=1}^{l} f(h(g_i))$$

$$s.t.\ \mathrm{R} = \{g_1, \ldots, g_l\}$$

L2 normalization for regions

Lattice $G = \{g_i\}, \quad i = 1, \ldots, N$; Sum pooling $h = \sum \cdot$; Linear classifier $f = w^T \cdot$



Feature Encoding | Lex Pooling | Lex Classification | Normalized Classification Pooling

$$\left\| L_R \right\|_2 = \sqrt{h(R)^T h(R)} = \sqrt{\sum_{i=1}^{l} \sum_{j=1}^{l} h(g_i)^T \cdot h(g_j)}$$

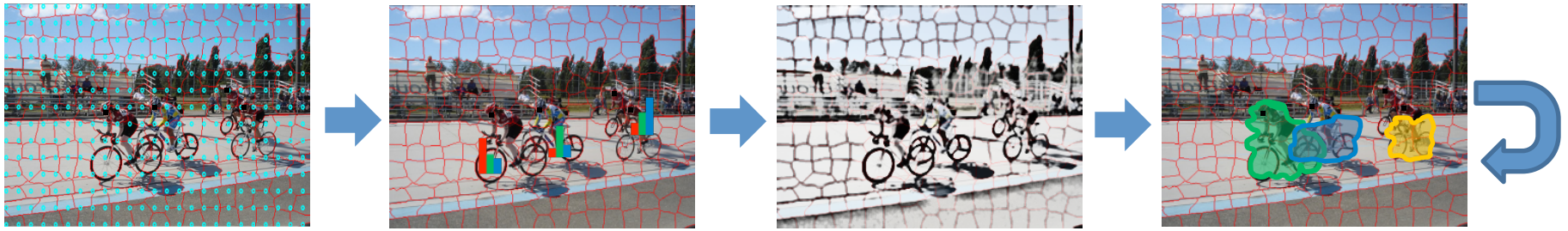
$$s.t. \ \mathrm{R} = \{g_1, \ldots, g_l\}$$

L2 normalized classification score:

$$y(R) = f\left(\frac{1}{\left\| L_R \right\|_2} \cdot h(R)\right) = \frac{1}{\left\| L_R \right\|_2} \cdot \sum_{i=1}^{l} w^T h(g_i)$$

L2 normalization for regions

Lattice $G = \{g_i\}, \quad i = 1,\ldots,N$; Sum pooling $h = \sum \cdot$; Linear classifier $f = w^T \cdot$



Feature Encoding | Lex Pooling | Lex Classification | Normalized Classification Pooling

$$\left\| L_R \right\|_2 = \sqrt{h(R)^T h(R)} = \sqrt{\sum_{i=1}^{l} \sum_{j=1}^{l} h(g_i)^T \cdot h(g_j)}$$

pair-wise lex similarity

$$s.t.\ \mathrm{R} = \{g_1,\ldots,g_l\}$$

L2 normalized classification score:

$$y(R) = f\left(\frac{1}{\left\| L_R \right\|_2} \cdot h(R)\right) = \frac{1}{\left\| L_R \right\|_2} \cdot \sum_{i=1}^{l} w^T h(g_i)$$

per-lex classification score

Embed nonlinearity

Similarity between two codemaps for image X and Z can be reduced into pair-wise similarity between lexes

$$K_L(X,Z) = h(X)^T h(Z)$$
$$= \sum_{g_x \in X} \sum_{g_z \in Z} h(g_x)^T h(g_z)$$

Kernel Trick

Replace **linear** kernel with more sophisticated **nonlinear** ones for lexes

$$\tilde{K}(X,Z) = \sum_{g_x \in X} \sum_{g_z \in Z} k(h(g_x),\ h(g_z))$$

Nonlinear kernel pooling

$$\tilde{K}(X,Z) = \sum_{g_x \in X} \sum_{g_z \in Z} k(h(g_x),\ h(g_z))$$

approximated feature map

$$k(x,y) = \psi(x)^T \psi(y)$$

Vedaldi, PAMI 2012

$$\tilde{K}(X,Z) = \sum_{g_x \in X} \sum_{g_z \in Z} \psi(h(g_x))^T\ \psi(h(g_z)))$$

$$= \tilde{h}(X)^T \tilde{h}(Z)$$

where $\tilde{h}(X) = \sum_{g_x \in X} \psi(h(g_x))$

Nonlinear kernel pooling

$$\tilde{K}(X,Z) = \sum_{g_x \in X} \sum_{g_z \in Z} k(h(g_x),\ h(g_z))$$

approximated feature map

$$k(x,y) = \psi(x)^T \psi(y)$$

Vedaldi, PAMI 2012

$$\tilde{K}(X,Z) = \sum_{g_x \in X} \sum_{g_z \in Z} \psi(h(g_x))^T\ \psi(h(g_z)))$$

$$= \tilde{h}(X)^T \tilde{h}(Z)$$

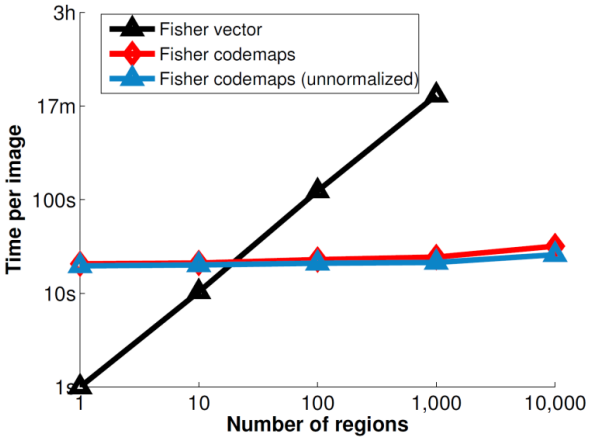
linear classifier

where $\tilde{h}(X) = \sum_{g_x \in X} \psi(h(g_x))$

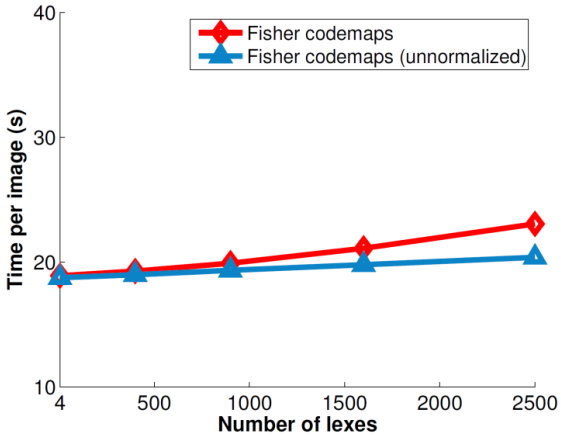
local nonlinear kernel pooling on each lex

global sum pooling

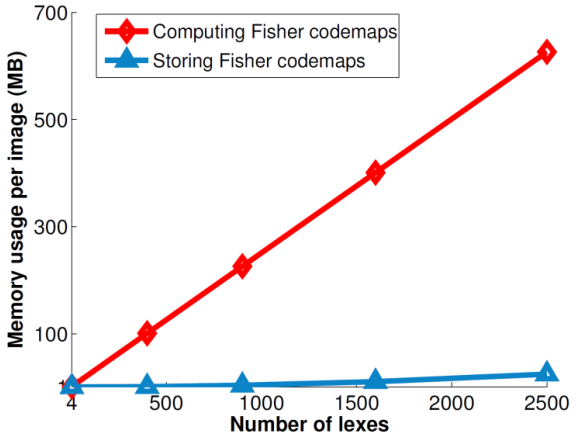
Timing and memory usages



(a)



(b)



(c)

Using Fisher encoding

L2 normalized codemaps are up to 56x faster than Fisher vectors

L2 normalization for arbitrary regions is as efficient for 4-500 lexes

Computing codemaps ~600MB/image, while storing ~30MB/image

Codemap segment classification

Codemaps

Computer vision by learning challenges involving repetitive computations over overlapping image regions may profit from codemaps.

Connection to convolutional networks?

Overview – Day 1

1. Introduction, types of concepts, relation to tasks, invariance
2. Observables, color, space, time, texture, Gaussian family
3. Invariance, the need, invariants, color, SIFT, Harris, HOG
4. BoW overview, what matters
5. On words and codebooks, internal and local structure, soft assignment, synonyms, convex reduction, Fisher & VLAD
6. Object and scene classification, recap chapters 1 to 5.
7. Support vector machine, linear, nonlinear, kernel trick.
8. Codemaps, L2-norm for regions, nonlinear kernel pooling.

Tomorrow

Laurens van der Maaten on

1. Pictorial structures
2. Latent and Structured SVMs
3. Convolutional networks