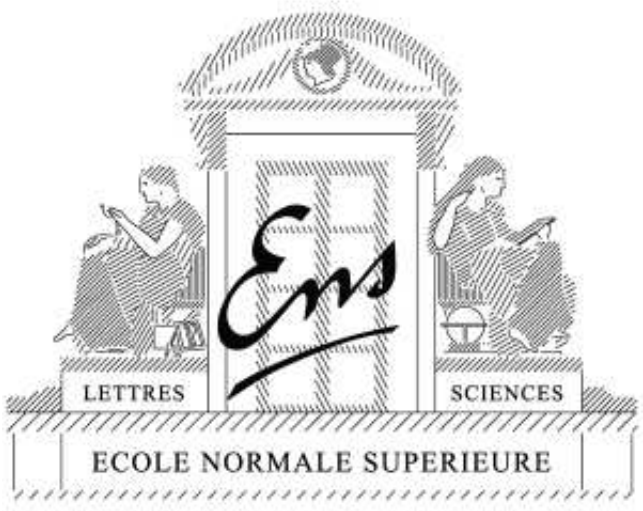
# Kernel methods & sparse methods for computer vision

**Francis Bach**

*Sierra project, INRIA - Ecole Normale Supérieure*


CVML Summer School, Paris, July 2011

# Machine learning for computer vision

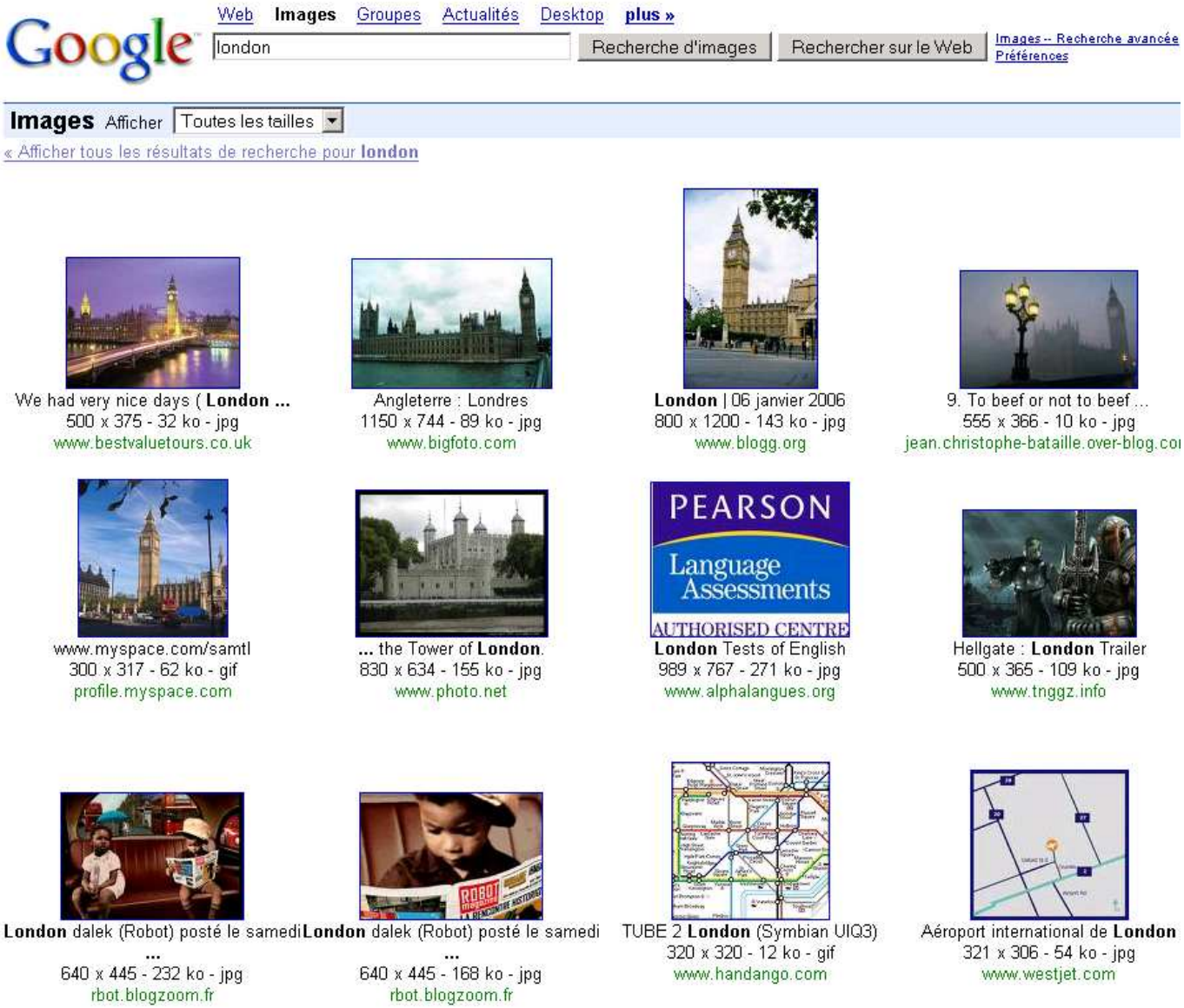
- Multiplication of digital media
- Many different tasks to be solved
  - Associated with different machine learning problems
  - Massive data to learn from

# Image retrieval

## $\Rightarrow$ Classification, ranking, outlier detection

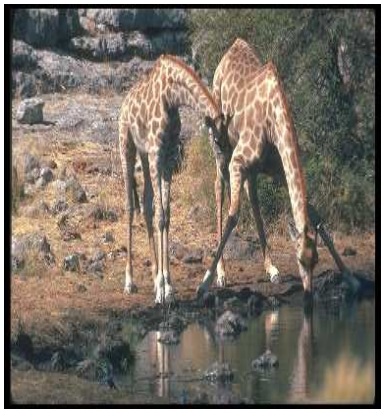
# Image retrieval

## Classification, ranking, outlier detection



# Image annotation

## Classification, clustering






# Object recognition $\Rightarrow$ Multi-label classification

# Machine learning for computer vision

- Multiplication of digital media
- Many different tasks to be solved
  - Associated with different machine learning problems
  - Massive data to learn from
- Similar situations in many fields (e.g., bioinformatics)

# Machine learning for computer vision

- Multiplication of digital media
- Many different tasks to be solved
  - Associated with different machine learning problems
  - Massive data to learn from
- Similar situations in many fields (e.g., bioinformatics)

  $\Rightarrow$ **Machine learning for high-dimensional data**

# Supervised learning and regularization

- Data: $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, $i = 1, \ldots, n$
- Minimize with respect to function $f \in \mathcal{F}$:

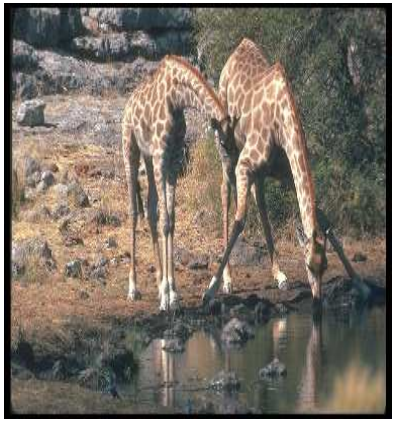
$$\sum_{i=1}^{n} \ell(y_i, f(x_i)) \quad + \quad \frac{\lambda}{2}\|f\|^2$$

Error on data + Regularization

Loss & function space ? Norm ?

- Two theoretical/algorithmic issues:
  - Loss
  - Function space / norm

# Image annotation $\Rightarrow$ multi-class classification





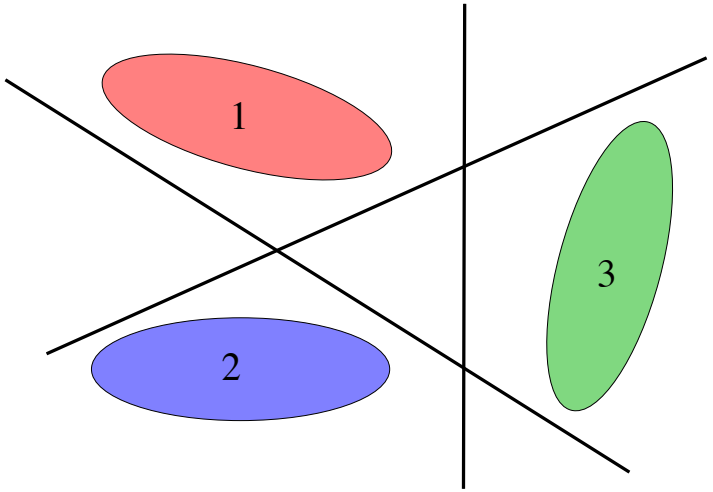

# Losses for multi-label classification (Schölkopf and Smola, 2001; Shawe-Taylor and Cristianini, 2004)

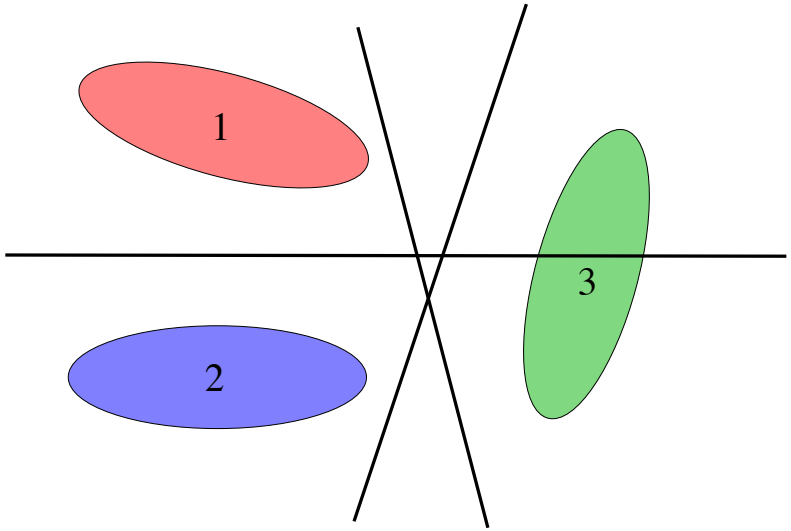
- **Two main strategies** for $k$ classes (with unclear winners)

  1. Using existing binary classifiers (efficient code!) + voting schemes
     - "one-vs-rest" : learn $k$ classifiers on the entire data
     - "one-vs-one" : learn $k(k-1)/2$ classifiers on portions of the data

# Losses for multi-label classification - Linear predictors

- Using binary classifiers (left: "one-vs-rest", right: "one-vs-one")



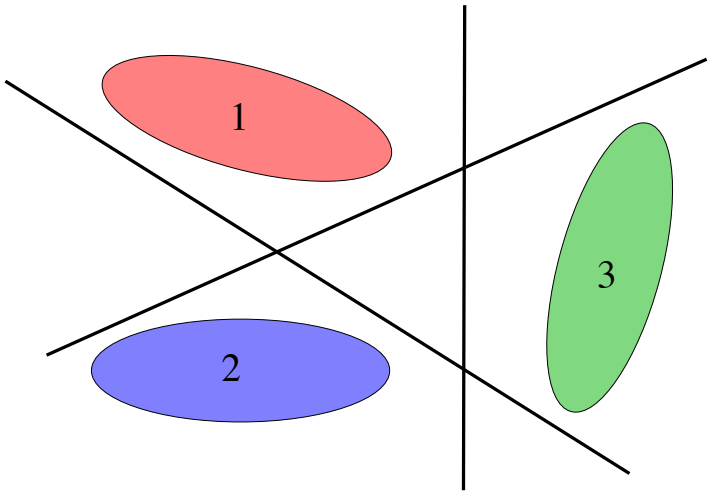
# Losses for multi-label classification (Schölkopf and Smola, 2001; Shawe-Taylor and Cristianini, 2004)

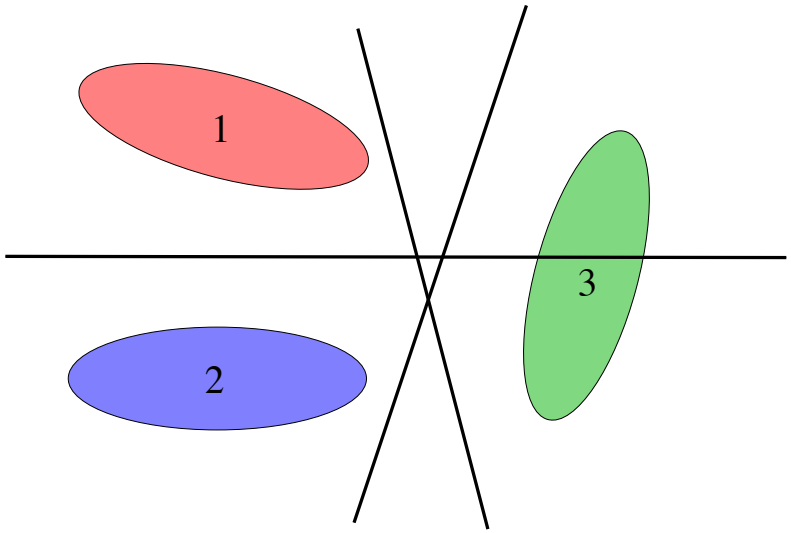
- **Two main strategies** for $k$ classes (with unclear winners)
  1. Using existing binary classifiers (efficient code!) + voting schemes
     - "one-vs-rest" : learn $k$ classifiers on the entire data
     - "one-vs-one" : learn $k(k-1)/2$ classifiers on portions of the data
  2. Dedicated loss functions for prediction using $\arg\max_{i\in\{1,\ldots,k\}} f_i(x)$
     - Softmax regression: loss $= -\log(e^{f_y(x)}/\sum_{i=1}^k e^{f_i(x)})$
     - Multi-class SVM - 1: loss $= \sum_{i=1}^k (1+f_i(x)-f_y(x))_+$
     - Multi-class SVM - 2: loss $= \max_{i\in\{1,\ldots,k\}}(1+f_i(x)-f_y(x))_+$

- Strategies do not consider same predicting functions

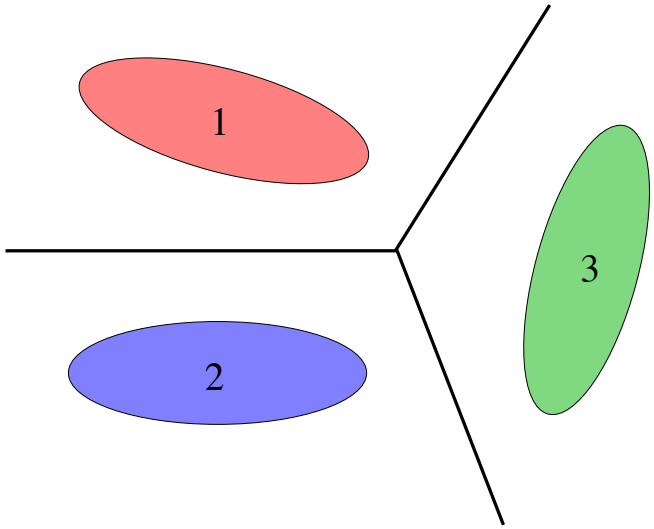
# Losses for multi-label classification - Linear predictors

- Using binary classifiers (left: "one-vs-rest", right: "one-vs-one")




- Dedicated loss function



# Image retrieval ⇒ ranking

# Losses for ther tasks

- Outlier detection (Schölkopf et al., 2001; Vert and Vert, 2006)
  - one-class SVM: learn only with positive examples
- Ranking
  - simple trick: transform into learning on pairs (Herbrich et al., 2000), i.e., predict $\{x > y\}$ or $\{x \leqslant y\}$
  - More general "structured output methods" (Joachims, 2002)
- General structured outputs
  - Very active topic in machine learning and computer vision
  - see, e.g., Taskar (2005)

# Kernel design

- Principle: **kernel on $\mathcal{X}$ = space of functions on $\mathcal{X}$ + norm**

- Two main design principles

  1. Constructing kernels from kernels by algebraic operations
  2. Using usual algebraic/numerical tricks to perform efficient kernel computation with very high-dimensional feature spaces

- Operations: $k_1(x,y) = \langle \Phi_1(x), \Phi_1(y) \rangle$, $k_2(x,y) = \langle \Phi_2(x), \Phi_2(y) \rangle$
  - **Sum = concatenation of feature spaces**:

$$k_1(x,y) + k_2(x,y) = \left\langle \begin{pmatrix} \Phi_1(x) \\ \Phi_2(x) \end{pmatrix}, \begin{pmatrix} \Phi_1(y) \\ \Phi_2(y) \end{pmatrix} \right\rangle$$

  - **Product = tensor product of feature spaces**:

$$k_1(x,y)k_2(x,y) = \left\langle \Phi_1(x)\Phi_2(x)^\top, \Phi_1(y)\Phi_2(y)^\top \right\rangle$$

# Classical kernels: kernels on vectors $x \in \mathbb{R}^d$

- Linear kernel $k(x, y) = x^\top y$
  - Linear functions
- Polynomial kernel $k(x, y) = (1 + x^\top y)^d$
  - Polynomial functions
- Gaussian kernel $k(x, y) = \exp(-\alpha \|x - y\|^2)$
  - Smooth functions
- Data are not always vectors!

# Efficient ways of computing large sums

- Goal: $\Phi(x) \in \mathbb{R}^p$ high-dimensional, compute $\sum_{i=1}^{p} \Phi_i(x)\Phi_i(y)$ **in** $o(p)$
- **Sparsity**: many $\Phi_i(x)$ equal to zero (example: pyramid match kernel)
- **Factorization and recursivity**: replace sums of many products by product of few sums (example: polynomial kernel, graph kernel)

$$(1 + x^\top y)^d = \sum_{\alpha_1 + \cdots + \alpha_k \leqslant d} \binom{d}{\alpha_1, \ldots, \alpha_k} (x_1 y_1)^{\alpha_1} \cdots (x_k y_k)^{\alpha_k}$$

# Kernels over (labelled) sets of points

- Common situation in computer vision (e.g., interest points)
- Simple approach: compute averages/histograms of certain features
  - valid kernels over histograms $h$ and $h'$ (Hein and Bousquet, 2004)
  - **intersection**: $\sum_i \min(h_i, h'_i)$, **chi-square**: $\exp\left(-\alpha \sum_i \frac{(h_i - h'_i)^2}{h_i + h'_i}\right)$

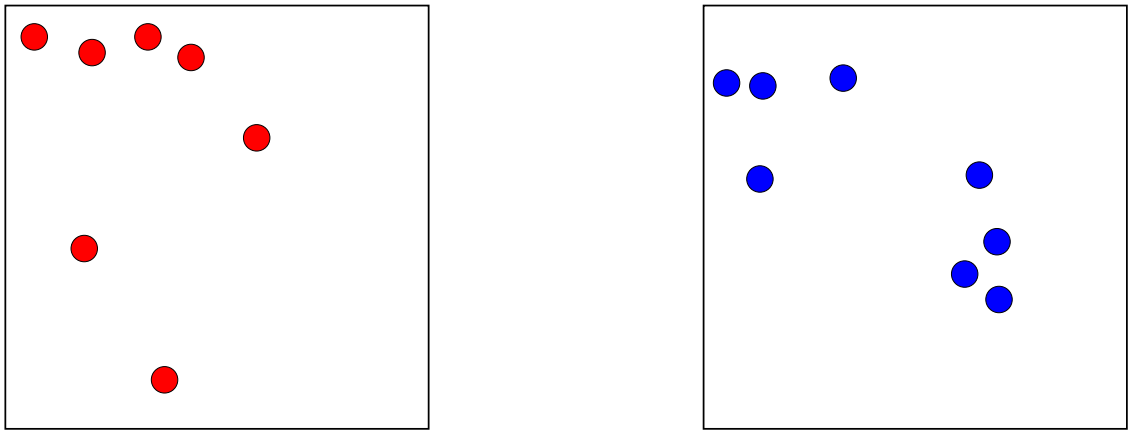
# Kernels over (labelled) sets of points

- Common situation in computer vision (e.g., interest points)
- Simple approach: compute averages/histograms of certain features
  - valid kernels over histograms $h$ and $h'$ (Hein and Bousquet, 2004)
  - **intersection**: $\sum_i \min(h_i, h'_i)$, **chi-square**: $\exp\left(-\alpha \sum_i \frac{(h_i - h'_i)^2}{h_i + h'_i}\right)$
- Pyramid match (Grauman and Darrell, 2007): efficiently introducing localization
  - Form a regular pyramid on top of the image
  - Count the number of common elements in each bin
  - Give a weight to each bin
  - Many bins but most of them are empty
    $\Rightarrow$ use sparsity to compute kernel efficiently

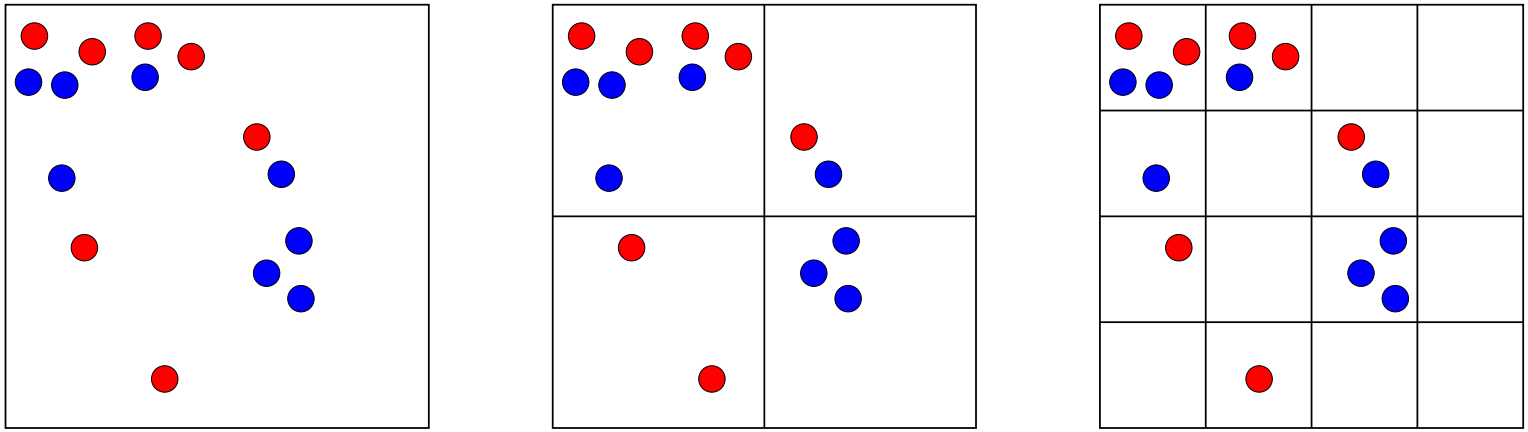
# Pyramid match kernel

## (Grauman and Darrell, 2007; Lazebnik et al., 2006)

- Two sets of points



- Counting matches at several scales: 7, 5, 4



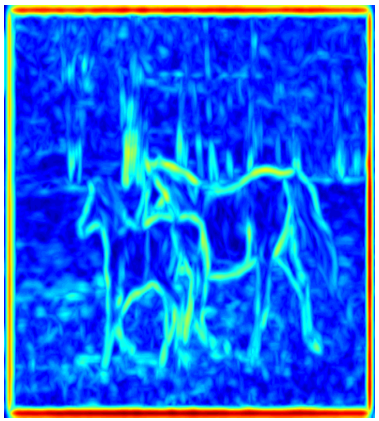
# Kernels from segmentation graphs

- Goal of segmentation: extract objects of interest
- Many methods available, ....
  - ... but, rarely find the object of interest entirely
- Segmentation graphs
  - Allows to work on "more reliable" over-segmentation
  - Going to a large square grid (millions of pixels) to a small graph (dozens or hundreds of regions)
- How to build a kernel over segmenation graphs?
  - NB: more generally, kernelizing existing representations?

# Segmentation by watershed transform (Meyer, 2001)

image

gradient



watershed

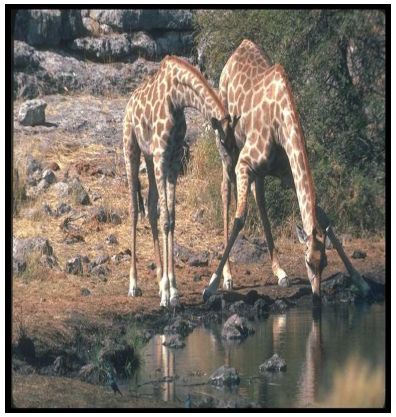
287 segments

64 segments

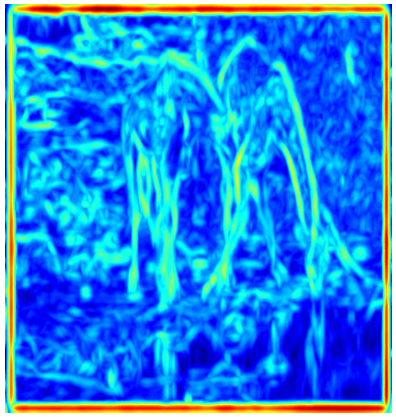
10 segments

# Segmentation by watershed transform (Meyer, 2001)

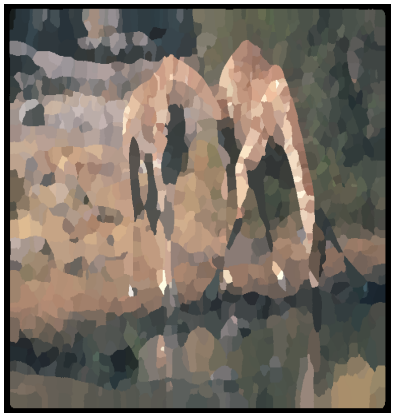
image



gradient



watershed



287 segments

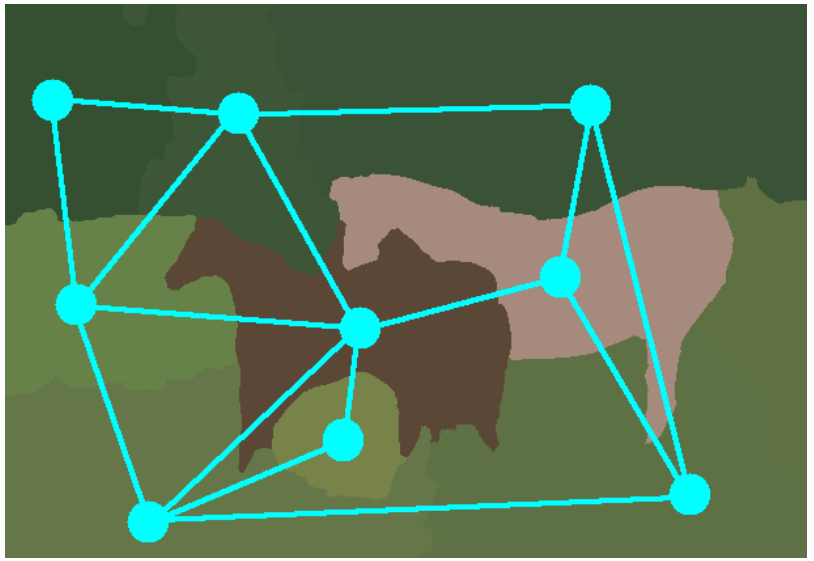
64 segments

10 segments

# Image as a segmentation graph

- Labelled undirected graph
  - Vertices: connected segmented regions
  - Edges: between spatially neighboring regions
  - Labels: region pixels

$\Rightarrow$



# Image as a segmentation graph

- Labelled undirected graph
  - Vertices: connected segmented regions
  - Edges: between spatially neighboring regions
  - Labels: region pixels

- Difficulties
  - Extremely high-dimensional labels
  - Planar undirected graph
  - Inexact matching

- **Graph kernels** (Gärtner et al., 2003; Kashima et al., 2004; Harchaoui and Bach, 2007) provide an elegant and efficient solution

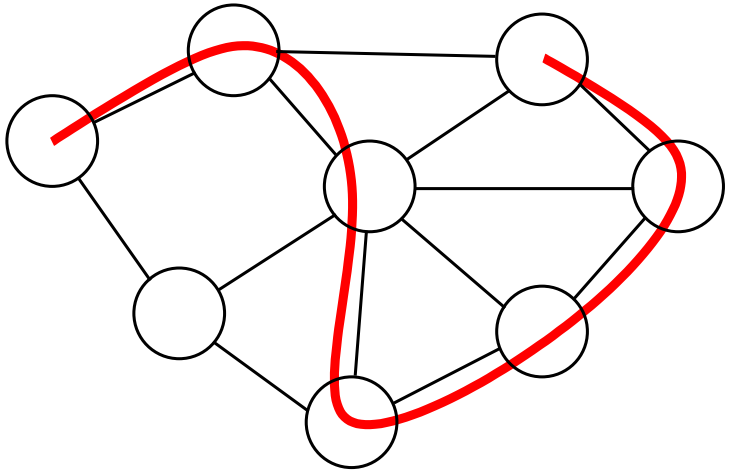
# Kernels between structured objects
# Strings, graphs, etc... (Shawe-Taylor and Cristianini, 2004)

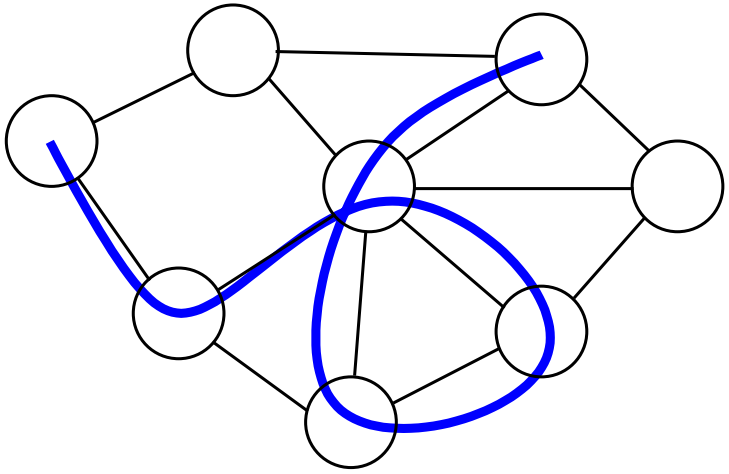
- Numerous applications (text, bio-informatics, speech, vision)
- Common design principle: **enumeration of subparts** (Haussler, 1999; Watkins, 1999)
  - Efficient for strings
  - Possibility of gaps, partial matches, very efficient algorithms
- Most approaches fails for general graphs (even for undirected trees!)
  - NP-Hardness results (Ramon and Gärtner, 2003)
  - Need specific set of subparts

# Paths and walks

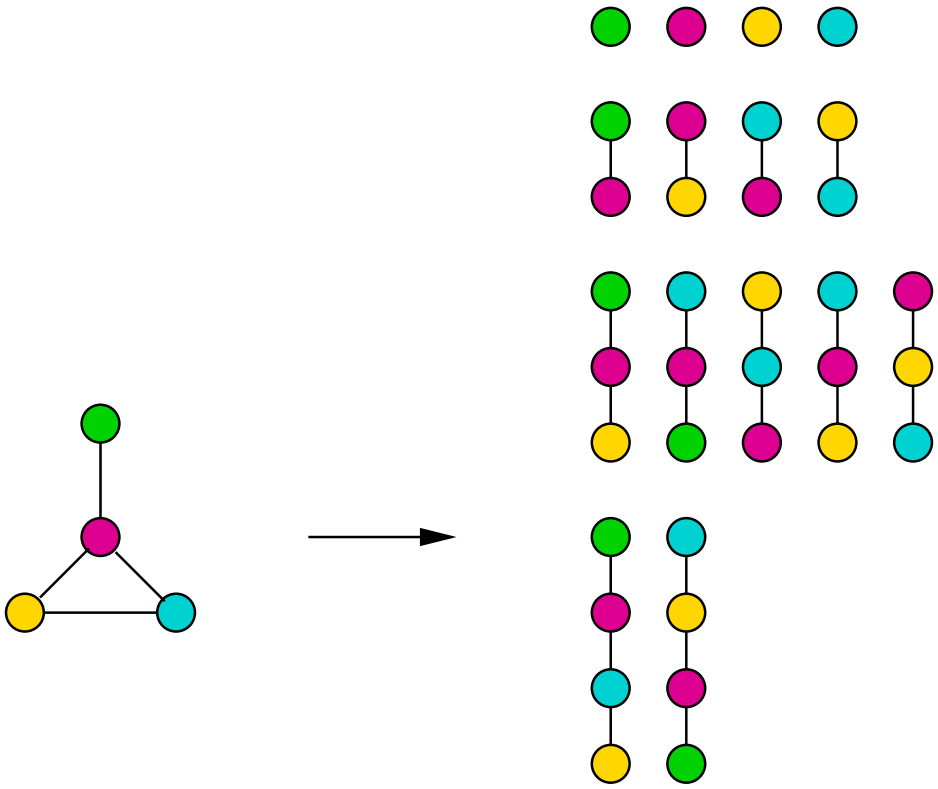
- Given a graph $G$,
  - A path is a sequence of distinct neighboring vertices
  - A walk is a sequence of neighboring vertices

- Apparently similar notions

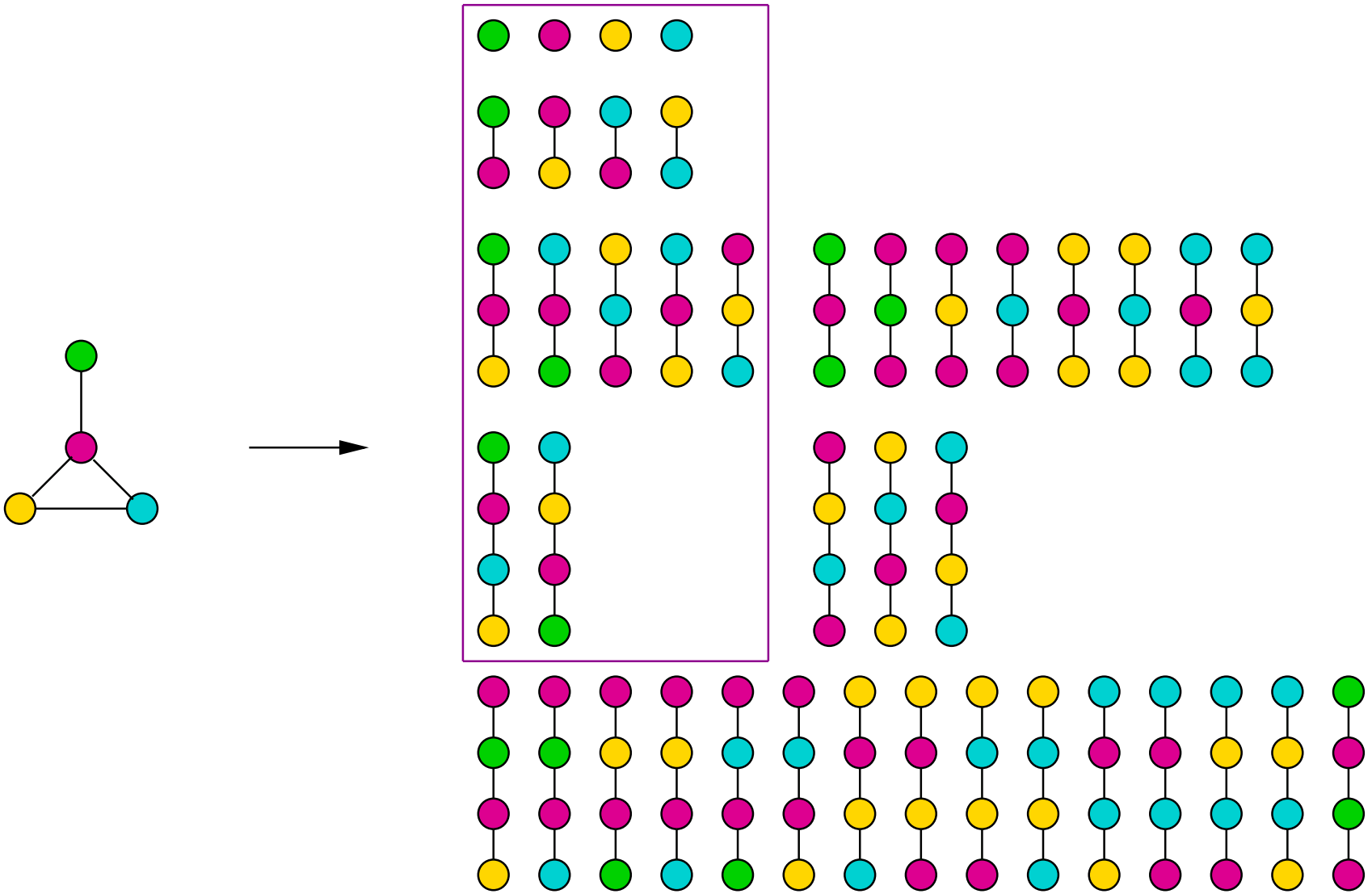




# Paths



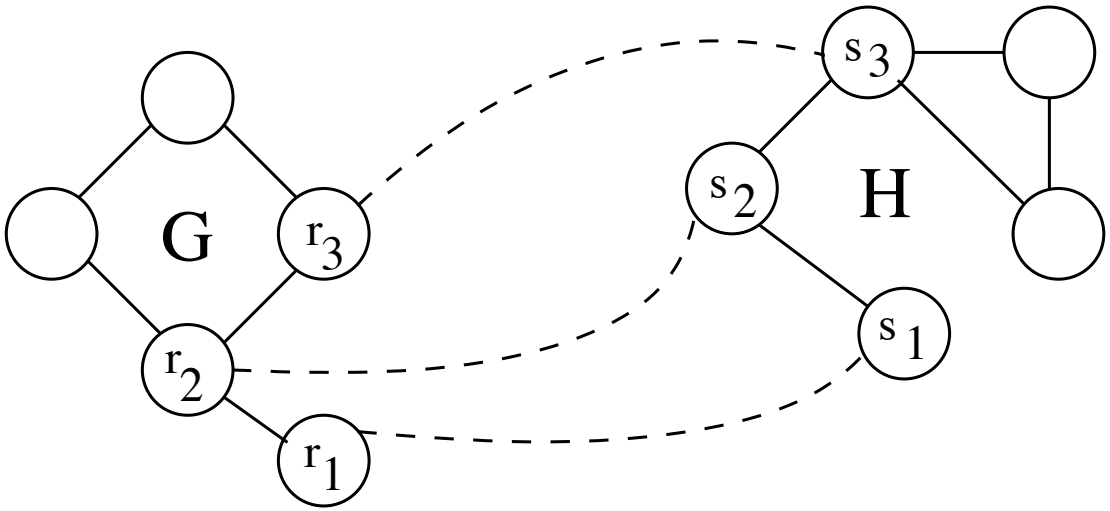
# Walks



# Walk kernel (Kashima et al., 2004; Borgwardt et al., 2005)

- $\mathcal{W}^p_{\mathbf{G}}$ (resp. $\mathcal{W}^p_{\mathbf{H}}$) denotes the set of walks of length $p$ in $\mathbf{G}$ (resp. $\mathbf{H}$)
- Given *basis kernel* on labels $k(\ell, \ell')$
- $p$-th order walk kernel:

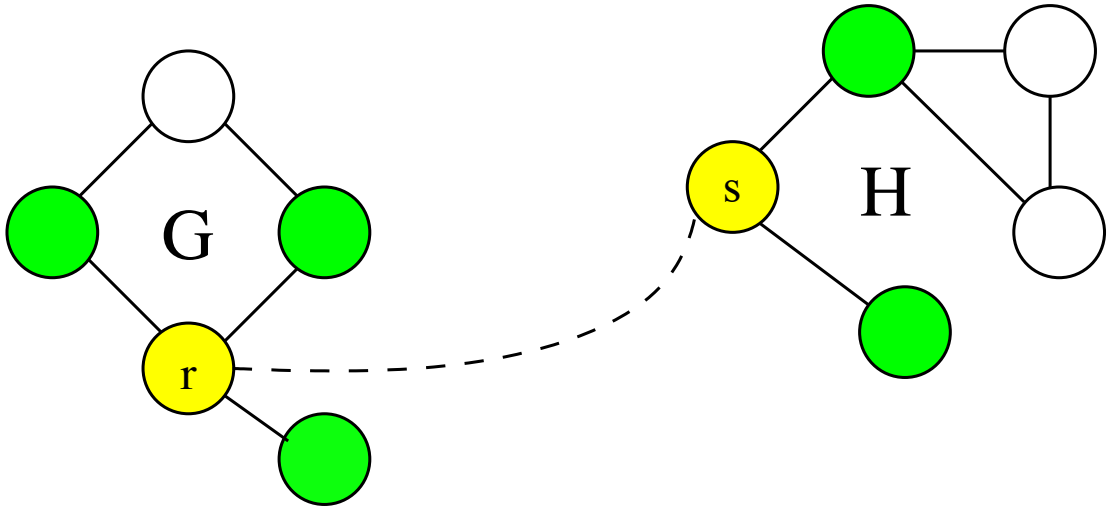
$$k^p_{\mathcal{W}}(\mathbf{G}, \mathbf{H}) = \sum_{\substack{(r_1, \ldots, r_p) \in \mathcal{W}^p_{\mathbf{G}} \\ (s_1, \ldots, s_p) \in \mathcal{W}^p_{\mathbf{H}}}} \prod_{i=1}^{p} k(\ell_{\mathbf{G}}(r_i), \ell_{\mathbf{H}}(s_i)).$$



# Dynamic programming for the walk kernel (Harchaoui and Bach, 2007)

- Dynamic programming in $O(pd_{\mathbf{G}}d_{\mathbf{H}}n_{\mathbf{G}}n_{\mathbf{H}})$
- $k^p_{\mathcal{W}}(\mathbf{G}, \mathbf{H}, r, s)$ = sum restricted to walks starting at $r$ and $s$
- recursion between $p-1$-th walk and $p$-th walk kernel

$$k^p_{\mathcal{W}}(\mathbf{G}, \mathbf{H}, r, s) = k(\ell_{\mathbf{G}}(r), \ell_{\mathbf{H}}(s)) \sum_{\substack{r' \in \mathcal{N}_{\mathbf{G}}(r) \\ s' \in \mathcal{N}_{\mathbf{H}}(s)}} k^{p-1}_{\mathcal{W}}(\mathbf{G}, \mathbf{H}, r', s').$$



# Dynamic programming for the walk kernel (Harchaoui and Bach, 2007)

- Dynamic programming in $O(pd_{\mathbf{G}}d_{\mathbf{H}}n_{\mathbf{G}}n_{\mathbf{H}})$
- $k^p_{\mathcal{W}}(\mathbf{G}, \mathbf{H}, r, s) =$ sum restricted to walks starting at $r$ and $s$
- recursion between $p-1$-th walk and $p$-th walk kernel

$$k^p_{\mathcal{W}}(\mathbf{G}, \mathbf{H}, r, s) = k(\ell_{\mathbf{G}}(r), \ell_{\mathbf{H}}(s)) \sum_{\substack{r' \in \mathcal{N}_{\mathbf{G}}(r) \\ s' \in \mathcal{N}_{\mathbf{H}}(s)}} k^{p-1}_{\mathcal{W}}(\mathbf{G}, \mathbf{H}, r', s')$$

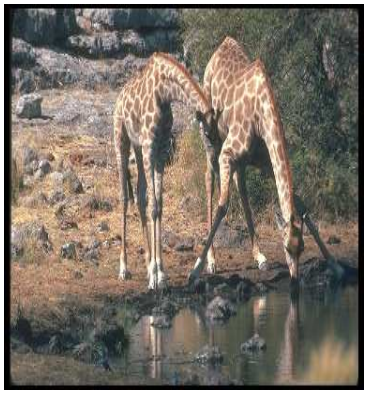
- Kernel obtained as $k^{p,\alpha}_{\mathcal{T}}(\mathbf{G}, \mathbf{H}) = \sum_{r \in \mathcal{V}_{\mathbf{G}}, s \in \mathcal{V}_{\mathbf{H}}} k^{p,\alpha}_{\mathcal{T}}(\mathbf{G}, \mathbf{H}, r, s)$

# Extensions of graph kernels

- Main principle: **compare all possible subparts of the graphs**
- Going from paths to subtrees
  - Extension of the concept of walks $\Rightarrow$ tree-walks (Ramon and Gärtner, 2003)
- Similar dynamic programming recursions (Harchaoui and Bach, 2007)
- Need to play around with subparts to obtain efficient recursions
  - Do we actually need positive definiteness?

# Performance on Corel14 (Harchaoui and Bach, 2007)

- `Corel14`: $1400$ natural images with $14$ classes






# Performance on Corel14 (Harchaoui & Bach, 2007)
# Error rates

- Histogram kernels (**H**)
- Walk kernels (**W**)
- Tree-walk kernels (**TW**)
- Weighted tree-walks (**wTW**)
- MKL (**M**)