Efficient Image Search with Correspondence Kernels and Learned Metrics

Kristen Grauman
University of Texas at Austin
Dept. of Computer Sciences

Work with Trevor Darrell, Prateek Jain, Brian Kulis

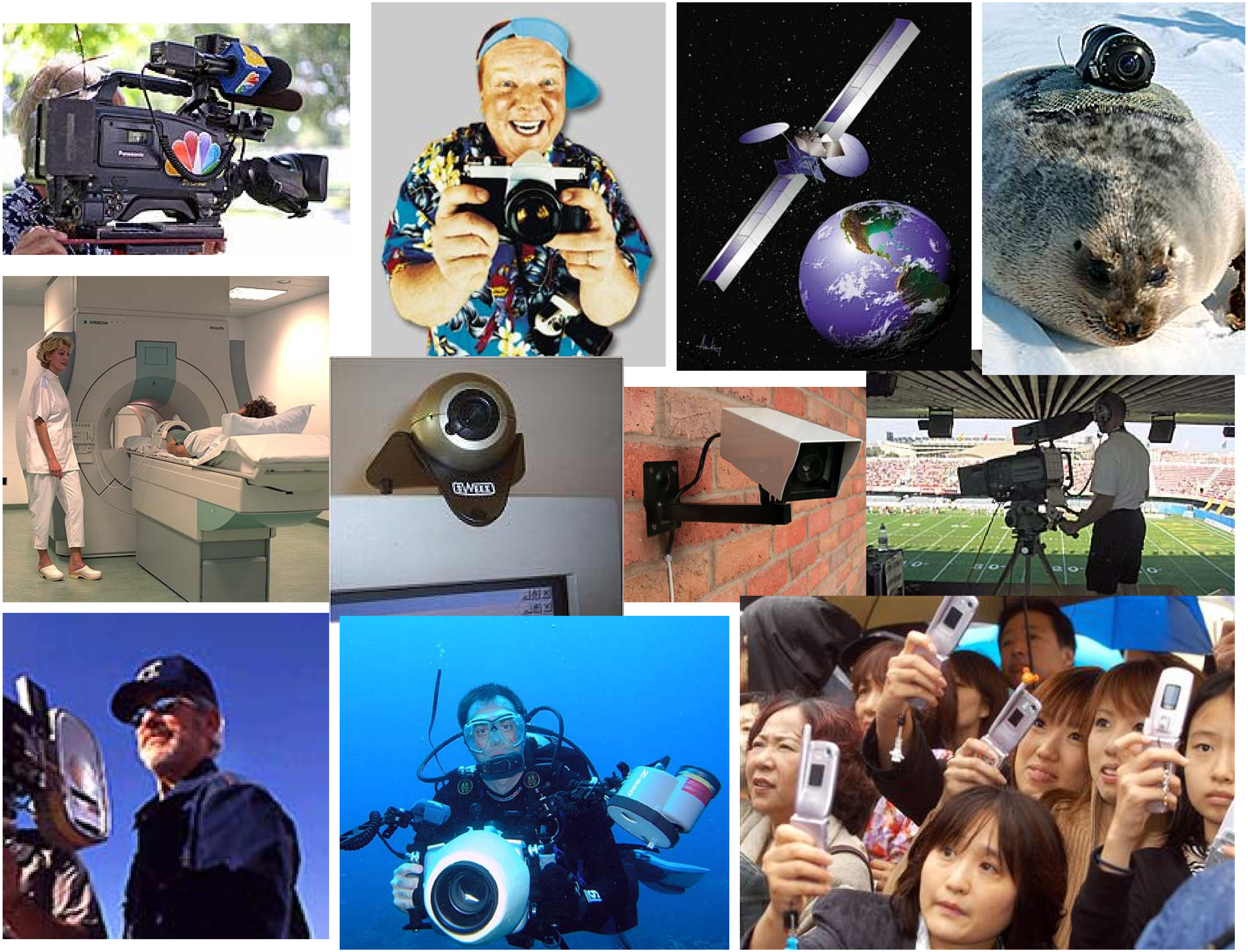


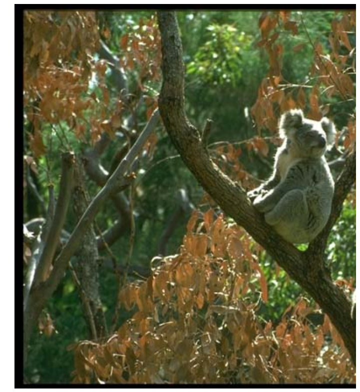
Image search vs. generic information retrieval

- What are the tokens? (segmentation problem)
- Plenty of nuisance parameters (lighting, pose, background clutter, etc.)

Nuisance parameters

Illumination

Object pose



Clutter

Occlusions

Intra-class appearance

Viewpoint

Image search vs. generic information retrieval

- What are the tokens? (segmentation problem)
- Plenty of nuisance parameters (lighting, pose, background clutter, etc.)
- Context can be critical to interpretation, but itself is difficult to extract
- Good representations may be structured (not vectors) and/or very high-dimensional

Outline

Scalable image search

- Fast correspondence-based search with local features
- Fast similarity search for learned metrics

Techniques motivated by image search problems, though not specific to image data.

Image representations

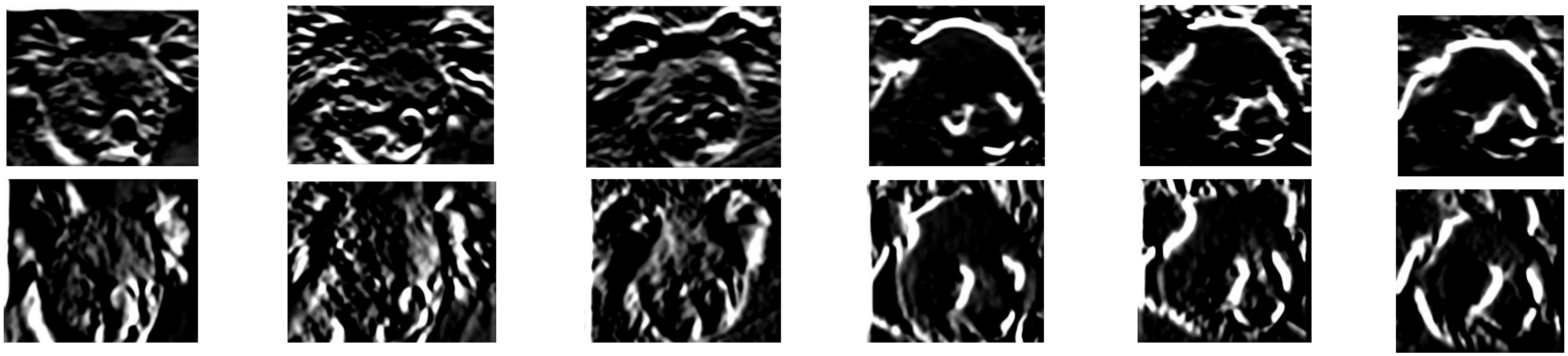
- Global:

  Describe the image as a whole

- Local:

  Decompose the image into multiple parts or fragments

Global representations

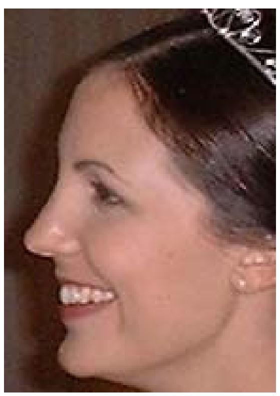


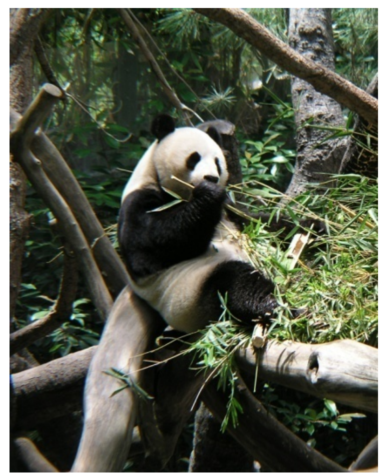
Map image to a single vector based on overall characteristics

- vector of pixel intensities
- grayscale / color histogram
- bank of filter responses …

Limitations of global representations

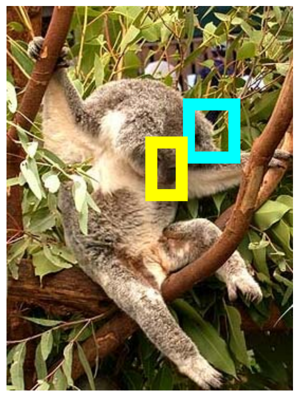
- Success may rely on alignment
- All parts of image impact description

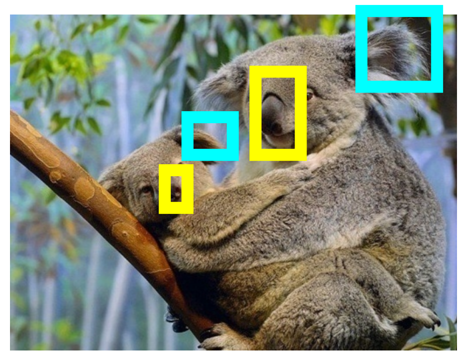




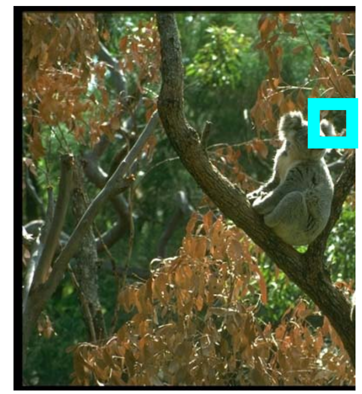

Local image features

Illumination

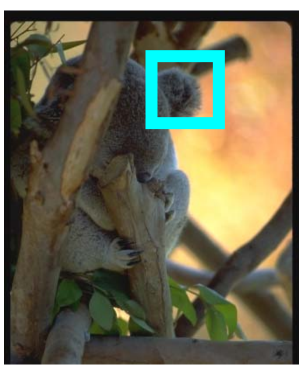




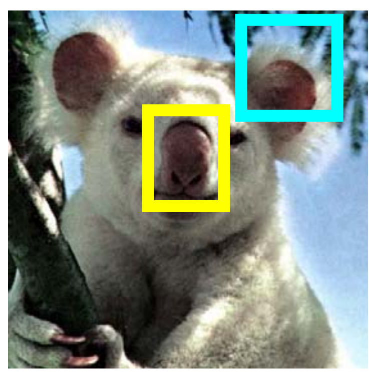
Object pose



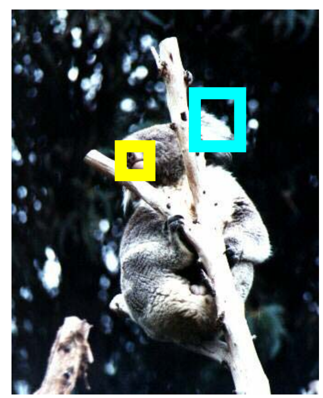
Clutter



Occlusions



Intra-class appearance

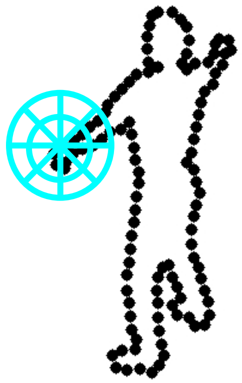


Viewpoint

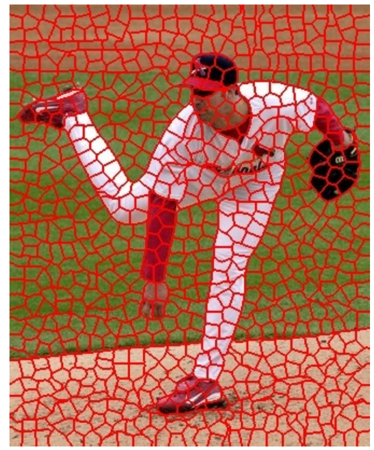
Local representations

Describe component regions or patches separately

SIFT [Lowe]

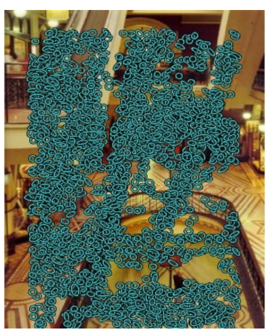


Shape context
[Belongie et al.]



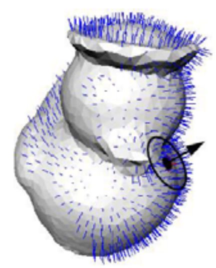
Superpixels
[Ren et al.]

Maximally Stable Extremal
Regions [Matas et al.]

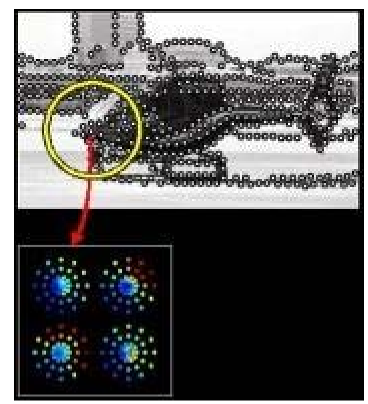


Salient regions
[Kadir et al.]

Harris-Affine
[Schmid et al.]



Spin images
[Johnson and Hebert]

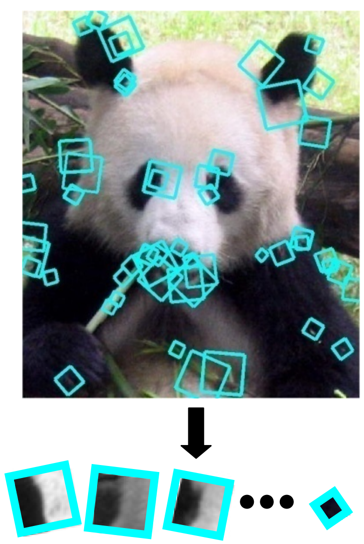


Geometric Blur
[Berg et al.]

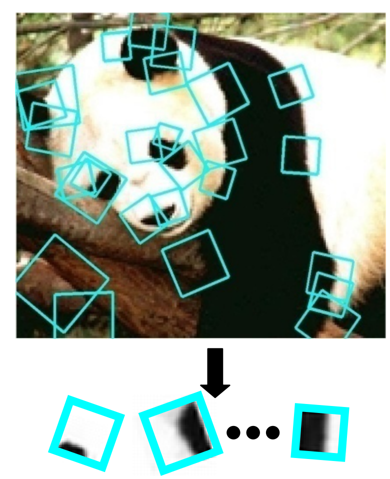
How to handle sets of features?

Want to compare, index, cluster, etc. local representations, but:

- Each instance is unordered set of vectors
- Varying number of vectors per instance



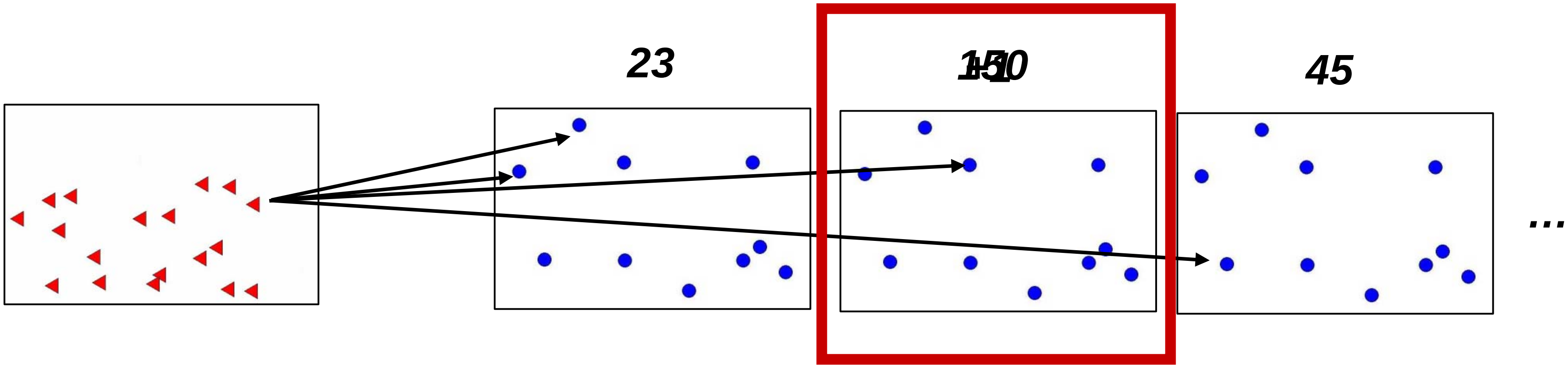
$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}$$



$$\mathbf{Y} = \{\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_n\}$$

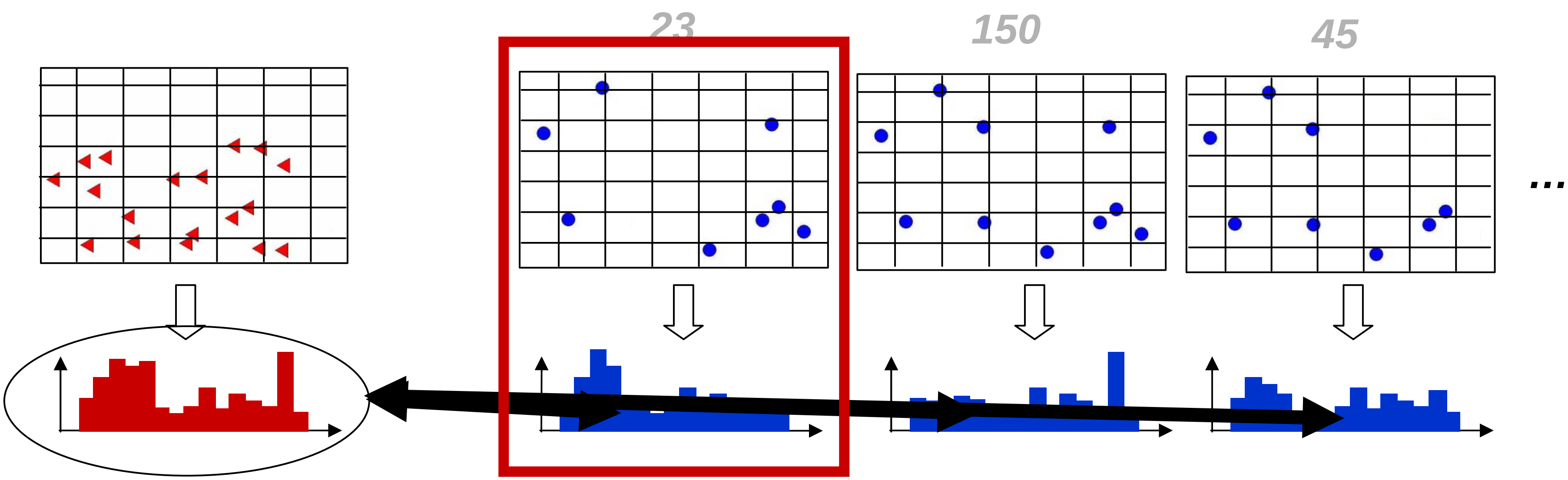
Conventional Approaches

“Voting” – for each patch, find the most similar patch in database, and vote for the image containing that patch. *[Schmid, Lowe, Tuytelaars et al.]*



Conventional Approaches

"Voting" – for each patch, find the most similar patch in database, and vote for the image containing that patch. *[Schmid, Lowe, Tuytelaars et al.]*



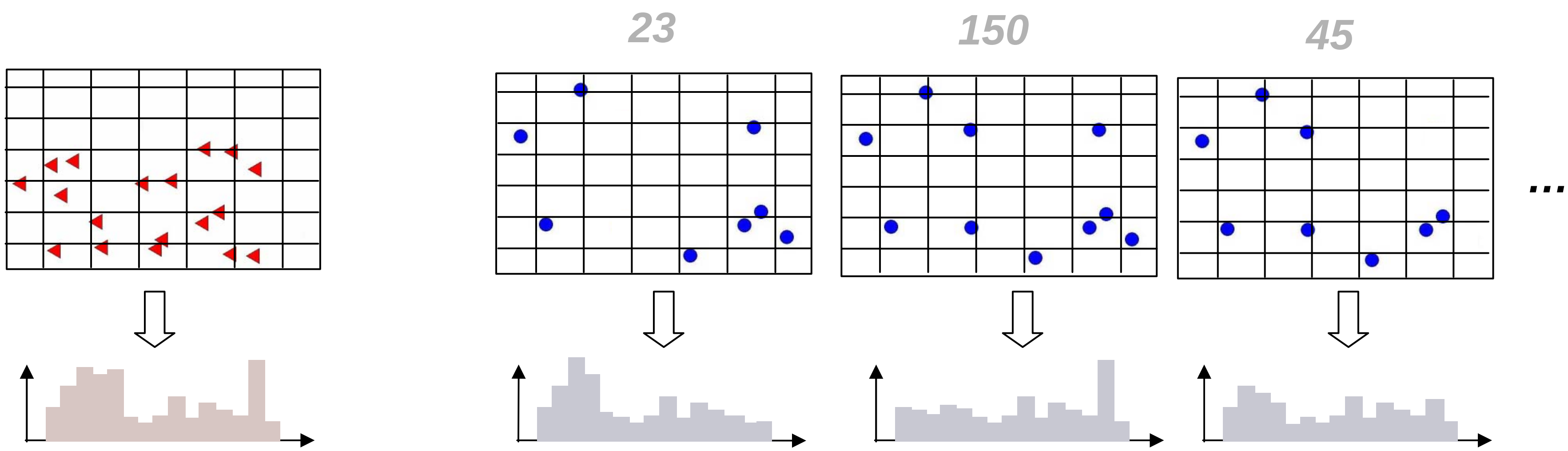
"Bag of Words" – quantize descriptor space, represent each image as a histogram over prototypes; use NN indexing, SVM recognition. *[Csurka et al., Sivic & Zisserman, Lazebnik & Ponce, Agarwal & Triggs, …]*

Conventional Approaches

“Voting” – for each p
database, and vote
patch.

Ignores co-occurrence; can be costly;
works well for instance matching

[Schmid, Lowe, Tuytelaars et al.]



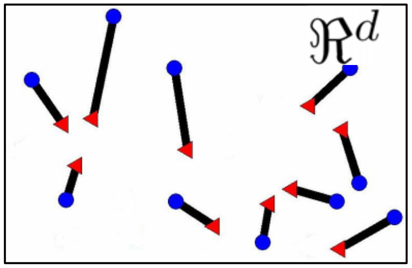
“Bag of Words” –
each image as a

Can be sensitive to choice of quantization;
how many/which “visual words”?

indexing, SVM recognition. *[Csurka et al., Sivic & Zisserman, Lazebnik & Ponce, Agarwal & Triggs, …]*

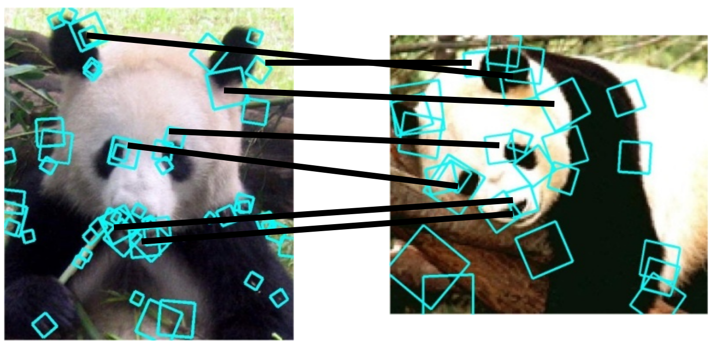
Correspondence / matching

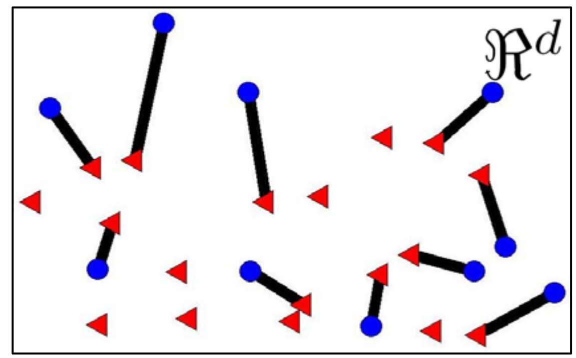
Explicitly search for good correspondences
[Wallraven et al., Lyu, Boughhorbel et al., Belongie et al., Rubner et al., Berg et al., Gold & Rangarajan, Shashua & Hazan,…]



Multi-resolution matching approximations
[Indyk & Thaper 2003, Charikar 2002, Agarwal & Varadarajan 2004, Grauman & Darrell 2005]

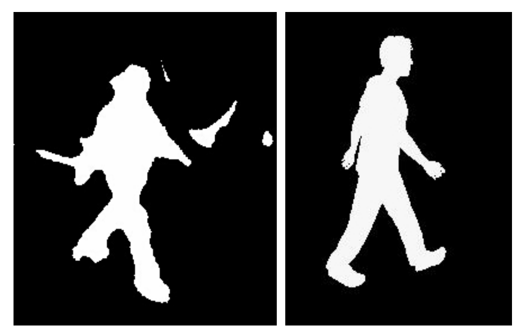
Partially matching sets of features





$$\max_{\pi:\mathbf{X}\rightarrow\mathbf{Y}} \sum_{\mathbf{x}_i\in\mathbf{X}} \mathcal{S}(\mathbf{x}_i, \pi(\mathbf{x}_i))$$

“Extra” features do not hurt the matching score

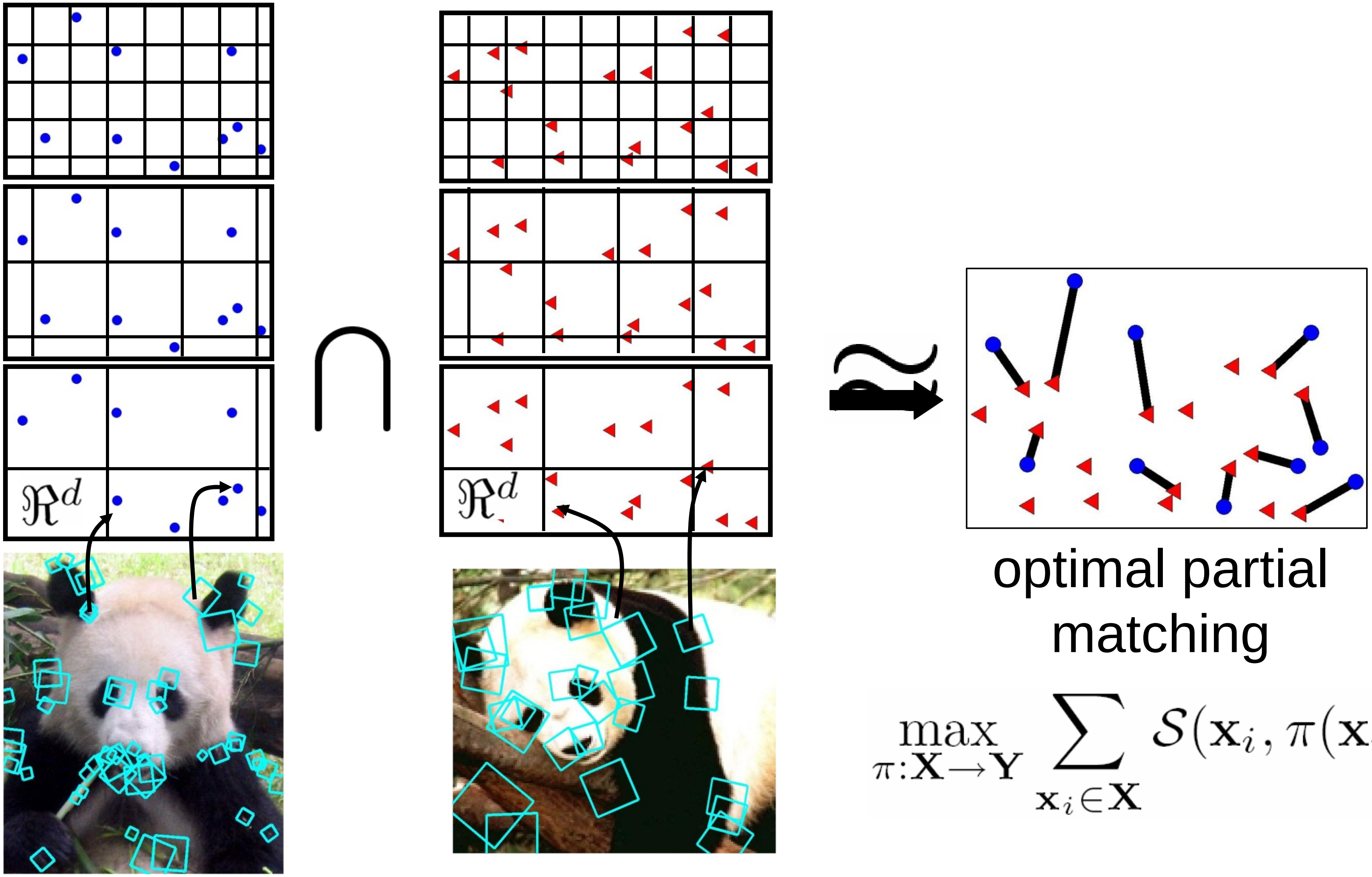


Computing the partial matching

- Optimal matching $O(dm^3)$
- Greedy matching $O(dm^2 \log m)$
- Pyramid match $O(dm)$
  [Grauman and Darrell, ICCV 2005]

for sets with $O(m)$ features of dimension d

Review: pyramid match

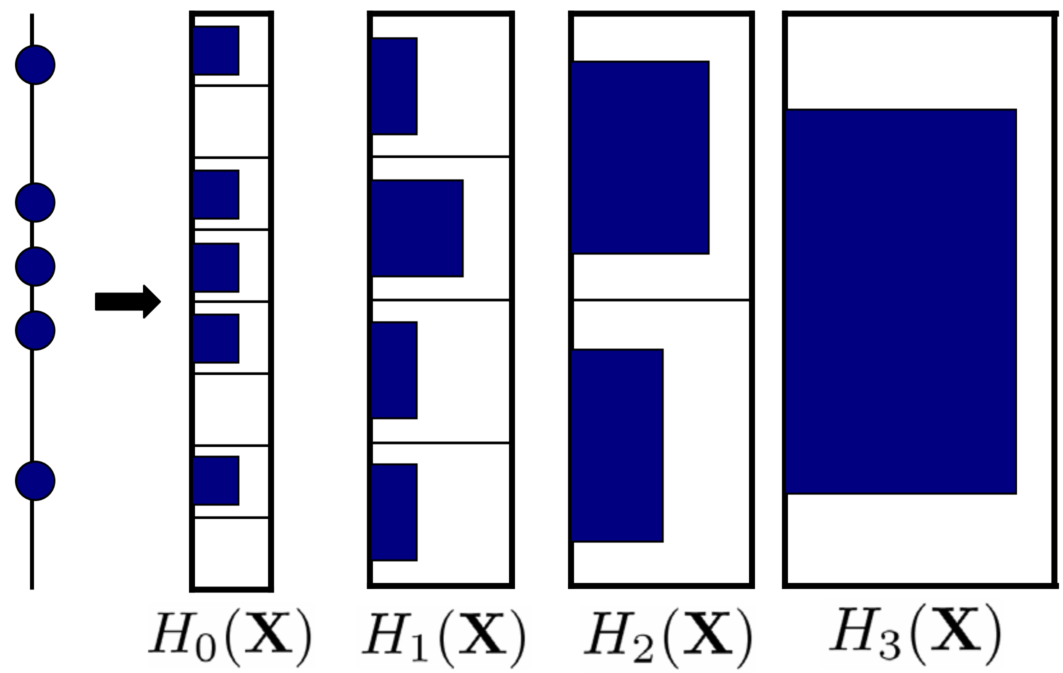


$$\max_{\pi:\mathbf{X}\rightarrow\mathbf{Y}} \sum_{\mathbf{x}_i\in\mathbf{X}} \mathcal{S}(\mathbf{x}_i, \pi(\mathbf{x}_i))$$

$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\} \qquad \mathbf{Y} = \{\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_n\}$$

Extracting histogram pyramids

$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}, \quad \vec{\mathbf{x}}_i \in \Re^d$$

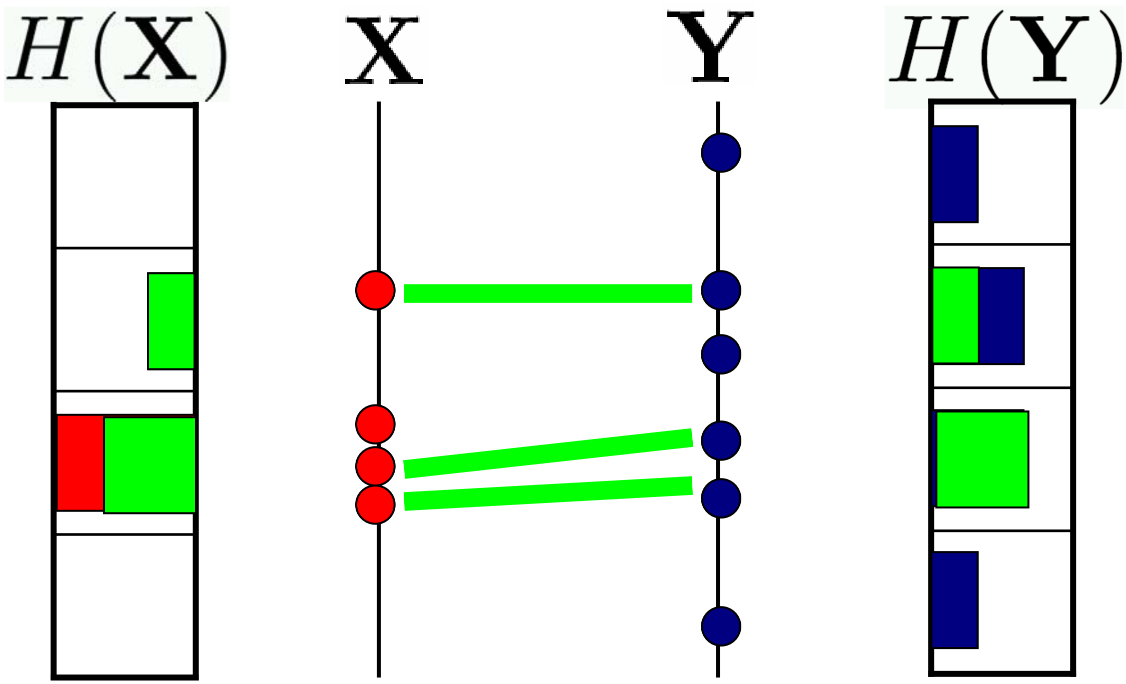


Histogram pyramid: level i has bins of size 2^i

$$\Psi(\mathbf{X}) = [H_0(\mathbf{X}), \ldots, H_{L-1}(\mathbf{X})]$$

Counting matches

Histogram intersection $$\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = \sum_{j=1}^{r} \min\left(H(\mathbf{X})_j, H(\mathbf{Y})_j\right)$$



$$\mathcal{I}(H(\mathbf{X}), H(\mathbf{Y})) = 3$$

Counting new matches

Histogram intersection

$$\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = \sum_{j=1}^{r} \min\left(H(\mathbf{X})_j, H(\mathbf{Y})_j\right)$$

$$N_i = \underbrace{\mathcal{I}\left(H_i(\mathbf{X}), H_i(\mathbf{Y})\right)}_{\text{matches at this level}} - \underbrace{\mathcal{I}\left(H_{i-1}(\mathbf{X}), H_{i-1}(\mathbf{Y})\right)}_{\text{matches at previous level}}$$

Difference in histogram intersections across levels counts *number of new pairs* matched

Kernel definition

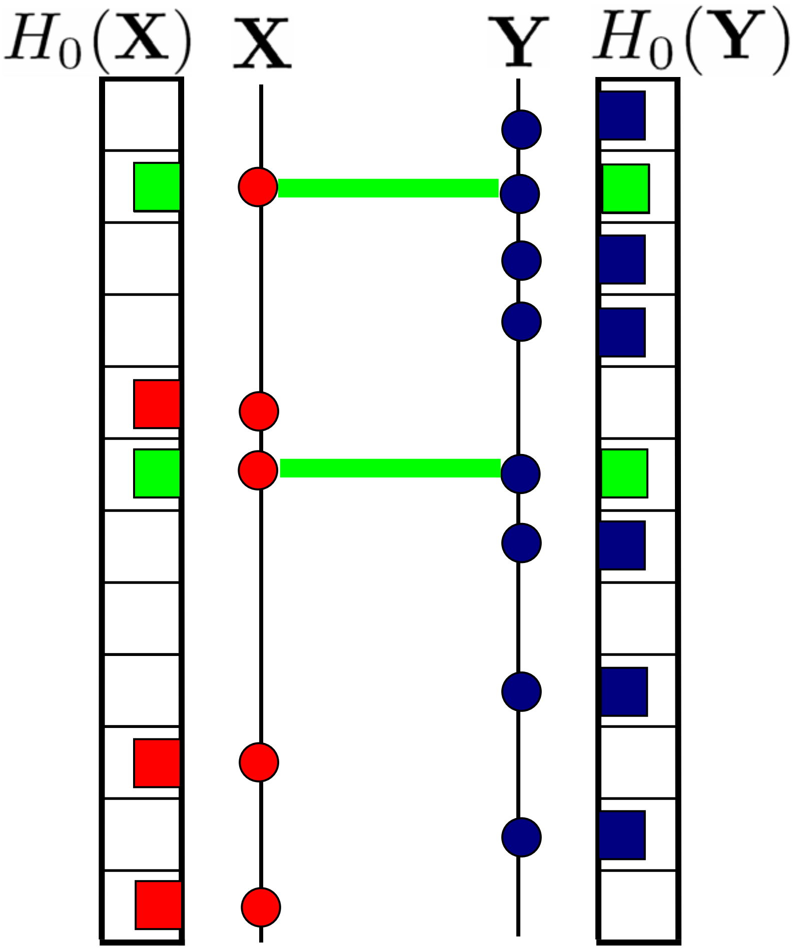
$$\tilde{K}_{\Delta}\big(\overbrace{\Psi(\mathbf{X}), \Psi(\mathbf{Y})}^{\text{histogram pyramids}}\big) =$$

$$\underbrace{\sum_{i=0}^{L} \frac{1}{2^i}}_{w_i\text{: measure of difficulty of a match at level } i} \Big(\underbrace{\mathcal{I}\left(H_i(\mathbf{X}), H_i(\mathbf{Y})\right) - \mathcal{I}\left(H_{i-1}(\mathbf{X}), H_{i-1}(\mathbf{Y})\right)}_{\text{number of newly matched pairs at level } i}\Big)$$

- For similarity, weights inversely proportional to bin size (or learned…)
- Normalize kernel values to avoid favoring large sets

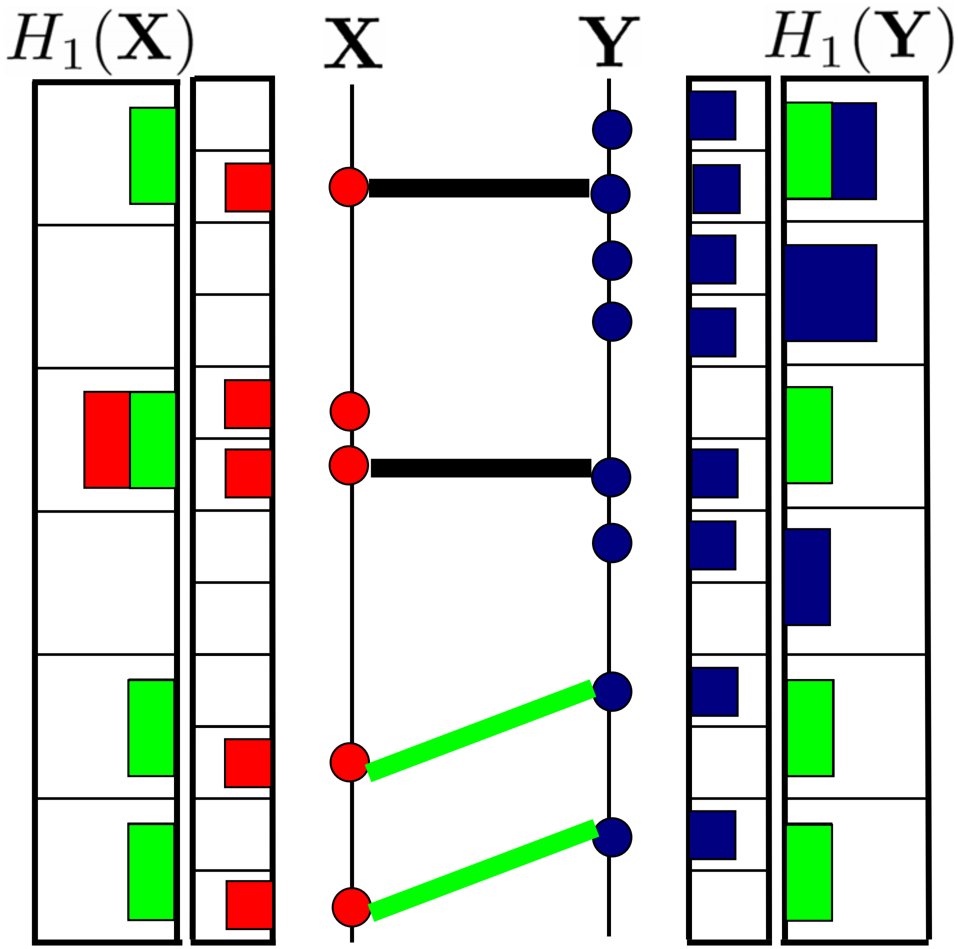
Example pyramid match

$$\mathcal{I}\left(H_0(\mathbf{X}), H_0(\mathbf{Y})\right) = 2 \longrightarrow \begin{matrix} N_0 = 2 \\ w_0 = 1 \end{matrix}$$



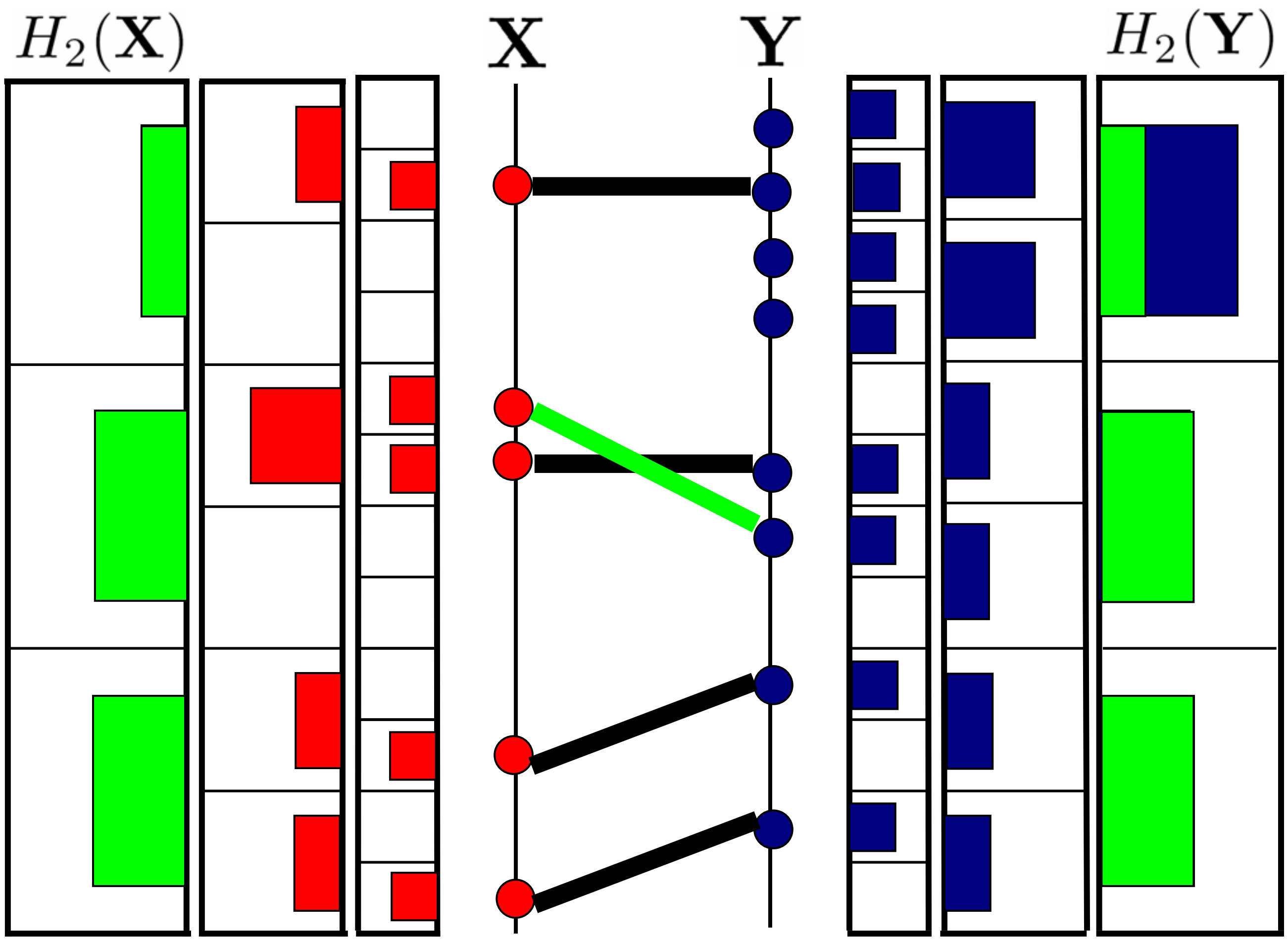
Example pyramid match

$$\mathcal{I}\left(H_1(\mathbf{X}), H_1(\mathbf{Y})\right) = 4 \longrightarrow \begin{array}{l} N_1 = 4 - 2 = 2 \\ w_1 = \frac{1}{2} \end{array}$$



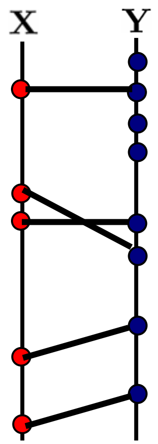
Example pyramid match

$$\mathcal{I}\left(H_2(\mathbf{X}), H_2(\mathbf{Y})\right) = 5 \longrightarrow \begin{array}{l} N_2 = 5 - 4 = 1 \\ w_2 = \frac{1}{4} \end{array}$$



Example pyramid match

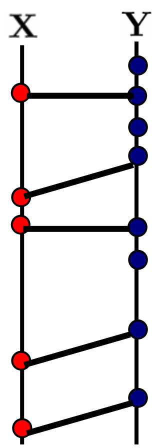
pyramid match



$$K_\Delta = \sum_{i=0}^{L} w_i N_i$$

$$= 1(2) + \tfrac{1}{2}(2) + \tfrac{1}{4}(1) = 3.25$$

optimal match



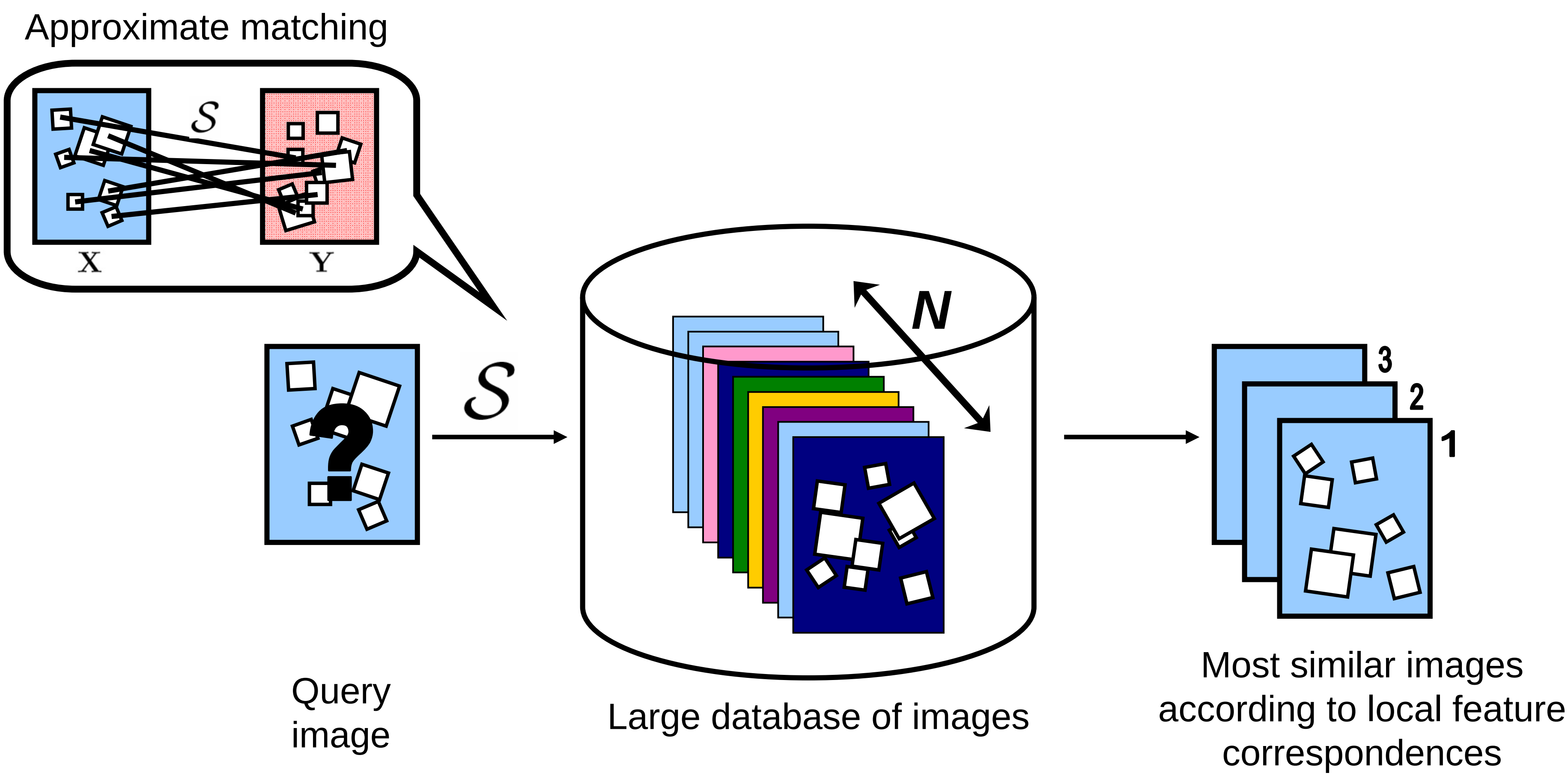
$$K = \max_{\pi:\mathbf{X}\to\mathbf{Y}} \sum_{\mathbf{x}_i \in \mathbf{X}} \mathcal{S}(\mathbf{x}_i, \pi(\mathbf{x}_i))$$

$$= 1(2) + \tfrac{1}{2}(3) = 3.5$$

Highlights of pyramid matching

- Time complexity linear in number of points per set
- Formal bounds on expected error
- Empirical evidence: preserves rank of optimal partial match
- Mercer kernel
- Strong performance on benchmark object recognition datasets, orders of magnitude speed improvements

Goal: index according to the matching



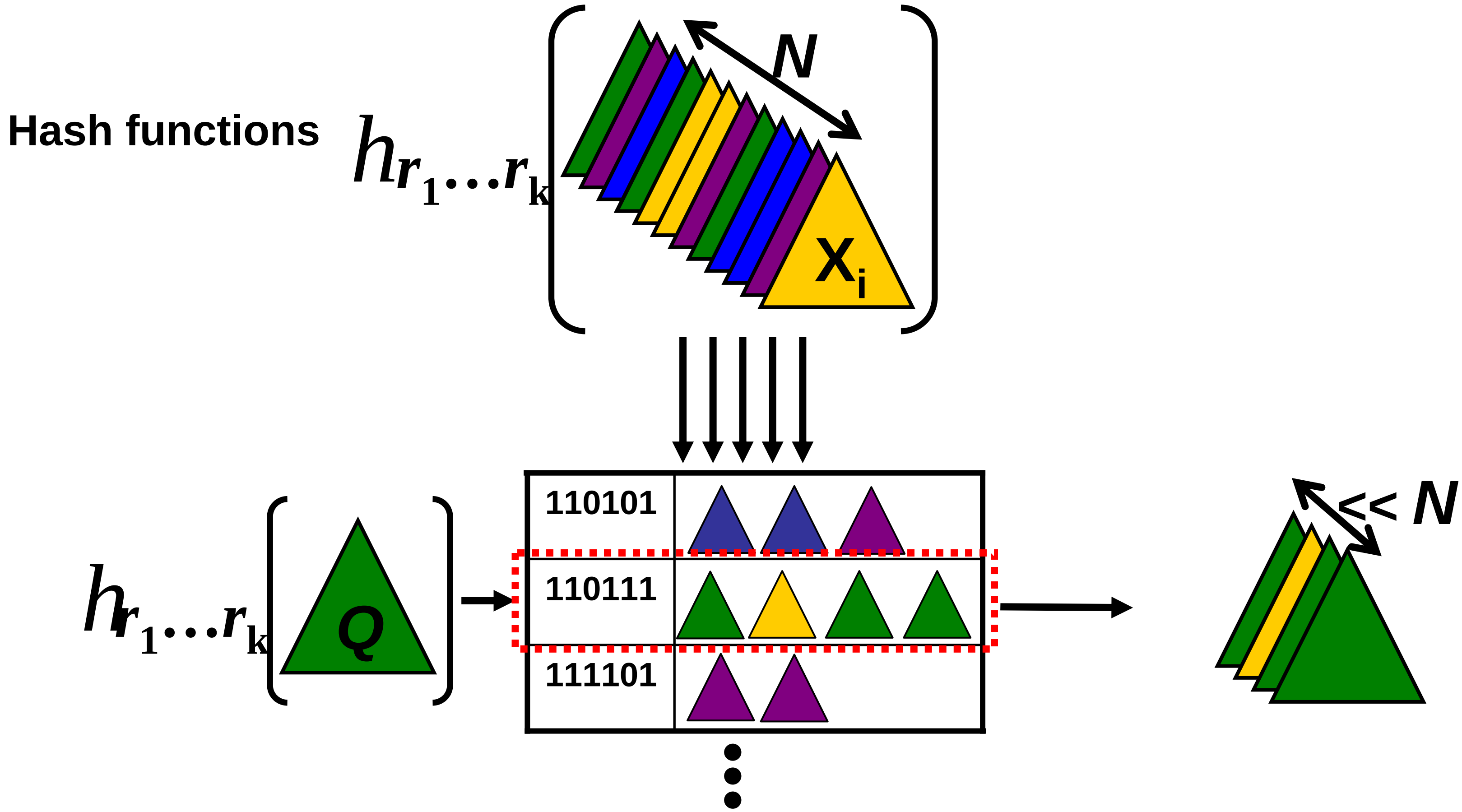
Linear scan infeasible, even with efficient metric.
How to index over *set correspondences*?

Image search with matching-sensitive hash functions

- Main idea:
  - Map point sets to a vector space in such a way that a dot product reflects partial match similarity (normalized PMK value).
  - Exploit random hyperplane properties to construct *matching-sensitive hash functions*.
  - Perform approximate similarity search on hashed examples.

[Grauman & Darrell 2007]

Sub-linear time search



Locality sensitive hash functions

A **locality-sensitive hash** (LSH) function guarantees similar examples collide in the hash table with high probability:

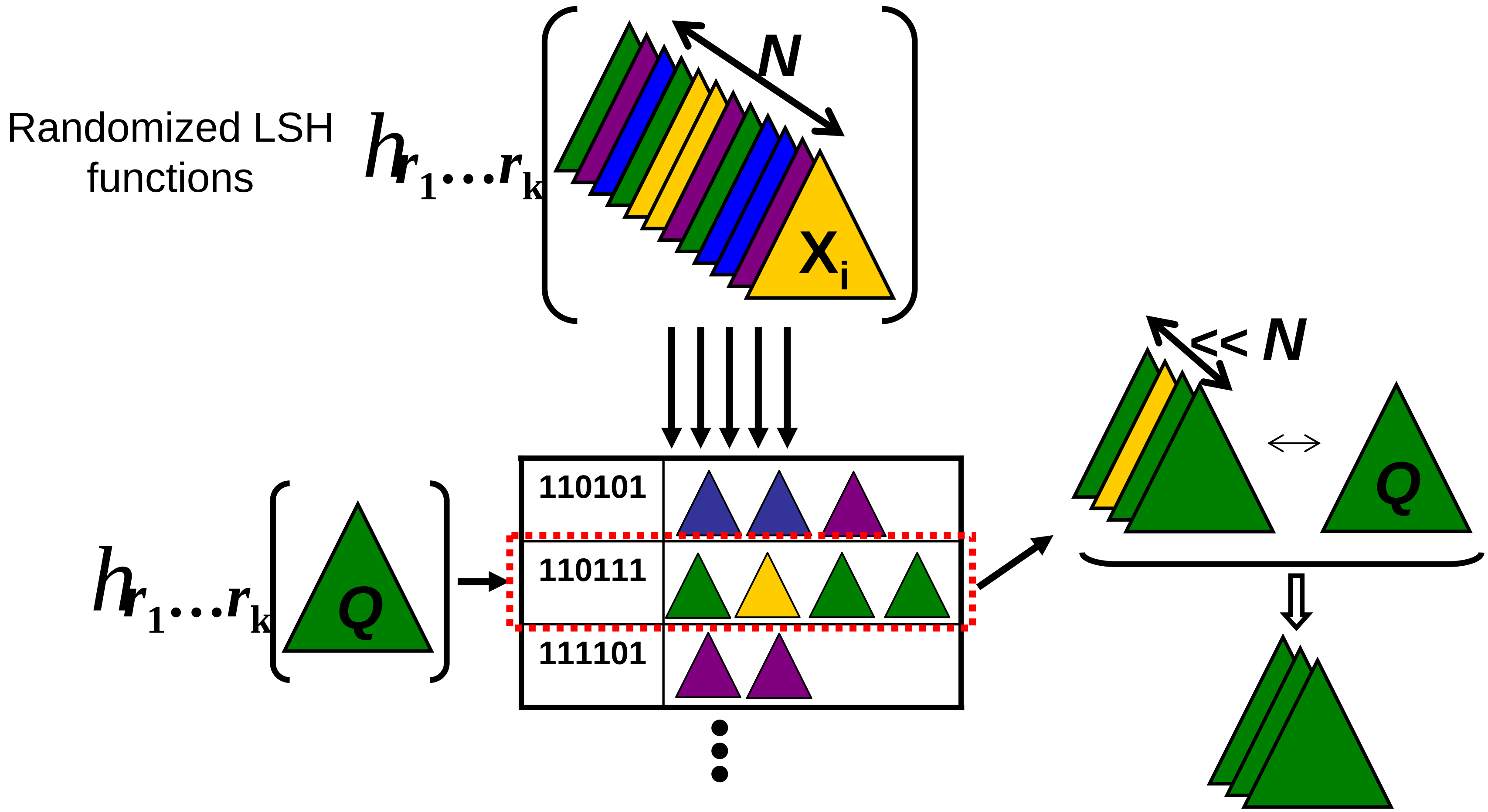
$$\Pr_{h \in \mathcal{F}} [\underbrace{h(x) = h(y)}_{\textit{Hash keys equal}}] = sim(x, y)$$

Existing methods guarantee retrieval of "approximate"-nearest neighbors in sub-linear time, *given appropriate hash functions.*

[Indyk and Motwani 1998]

Sub-linear time search with LSH

$$\Pr_{h \in \mathcal{F}} [h(x) = h(y)] = sim(x, y)$$



LSH functions for dot products

The probability that a *random hyperplane* separates two unit vectors is related to the angle between them.

$$\Pr\left[\mathrm{sgn}(\vec{v}_i \cdot \vec{r}) \neq \mathrm{sgn}(\vec{v}_j \cdot \vec{r})\right] = \frac{1}{\pi} \cos^{-1} \boxed{(\vec{v}_i \cdot \vec{v}_j)}$$

for $\vec{r}_i \sim N(\mu = 0, \sigma^2 = 1)$

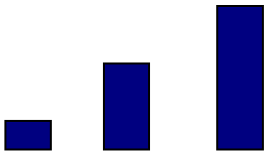
→ enables LSH function for **dot product similarity**.

[Goemans and Williamson 1995, Charikar 2004]

A useful property of intersection

<u>histograms</u>

<u>padded unary encoding</u>



x = [1, 3, 5]

$\mathcal{U}(x)$= [1 0 0 0 0

y = [2, 0, 3]

$\mathcal{U}(y)$= [1 1 0 0 0 0 0 0 0 0 1 1 1 0 0]

[1, 0, 3]

[1+0+0+0+0+0+0+0+0+0+1+1+1+0+0]

$\sum_i \min(x_i, y_i) = 4$

$\mathcal{U}(x) \cdot \mathcal{U}(y) = 4$

$$\boxed{\sum_i \min(x_i, y_i) = \mathcal{U}(x) \cdot \mathcal{U}(y)}$$

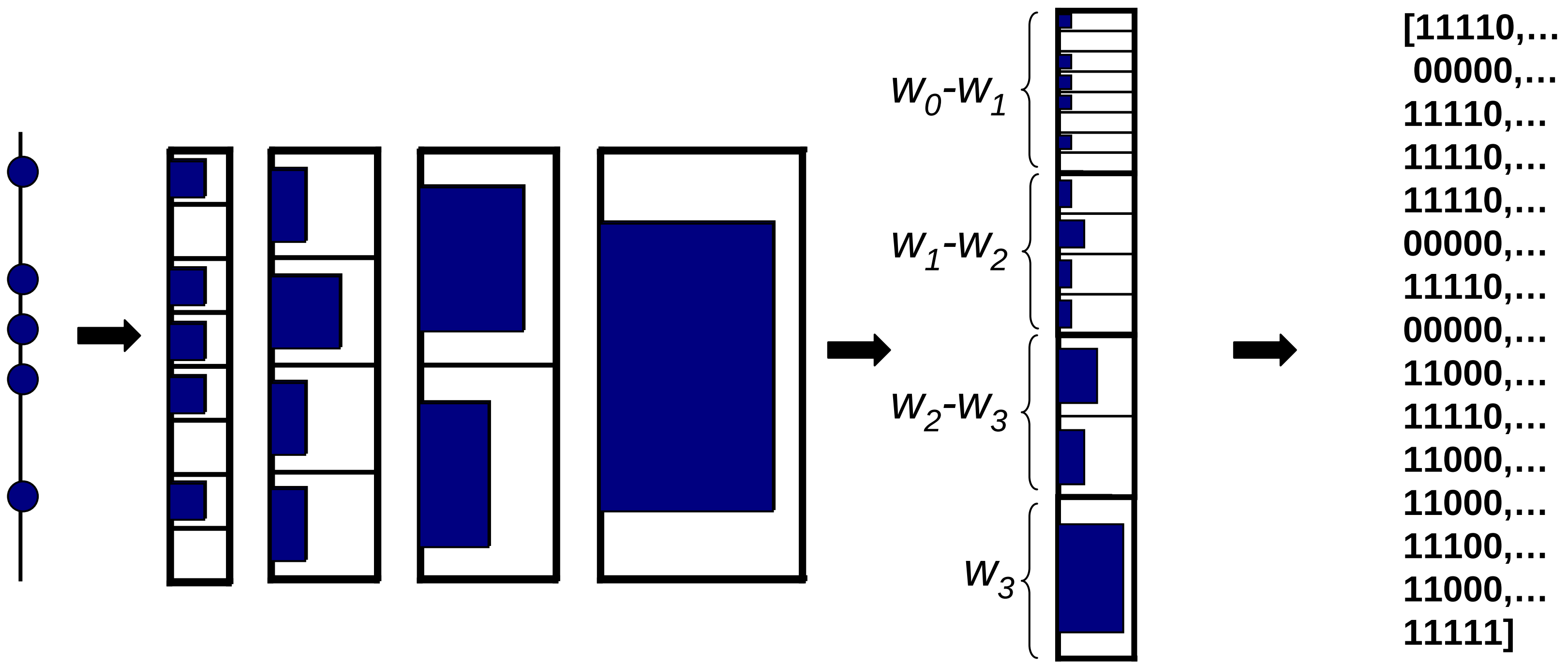
Vector encoding of pyramids

$$\tilde{K}_{\Delta}(\Psi(\mathbf{X}), \Psi(\mathbf{Y})) = \sum_{i=0}^{L-1} w_i \left(\mathcal{I}_i - \mathcal{I}_{i-1}\right)$$

$$= w_{L-1}\boxed{\mathcal{I}_{L-1}} + \sum_{i=0}^{L-2} (w_i - w_{i+1})\boxed{\mathcal{I}_i}$$

Pyramid match (un-normalized) expressed as sum of weighted intersections

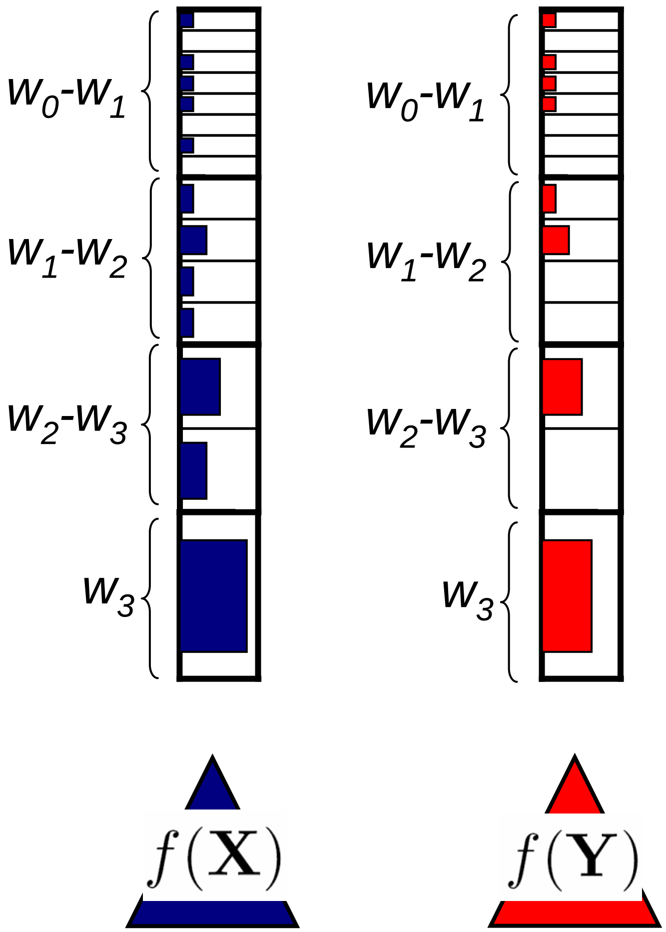
Vector encoding of pyramids



Point set ⟶ Multi-resolution histogram ⟶ **Weighted** sparse count vector ⟶ Implicit unary encoding

$\mathbf{X}$ $f(\mathbf{X})$

Vector encoding of pyramids



$$f(\mathbf{X}) \cdot f(\mathbf{Y}) = \tilde{K}_{\Delta}(\Psi(\mathbf{X}), \Psi(\mathbf{Y}))$$

Dot product between embedded point sets yields pyramid match kernel value

$$|f(\mathbf{X})| = \tilde{K}_{\Delta}(\Psi(\mathbf{X}), \Psi(\mathbf{X}))$$

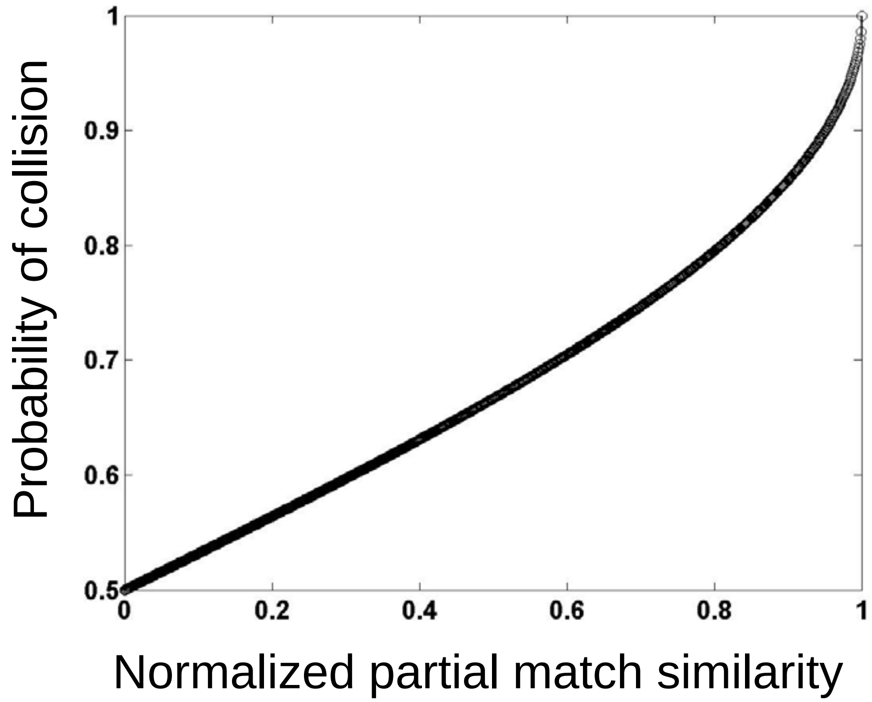
Length of an embedded point set is equivalent to its self-similarity

Matching-sensitive hash functions

$$h_{\vec{r}}(f(\mathbf{X})) = \begin{cases} 1, & \text{if } \vec{r} \cdot f(\mathbf{X}) \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

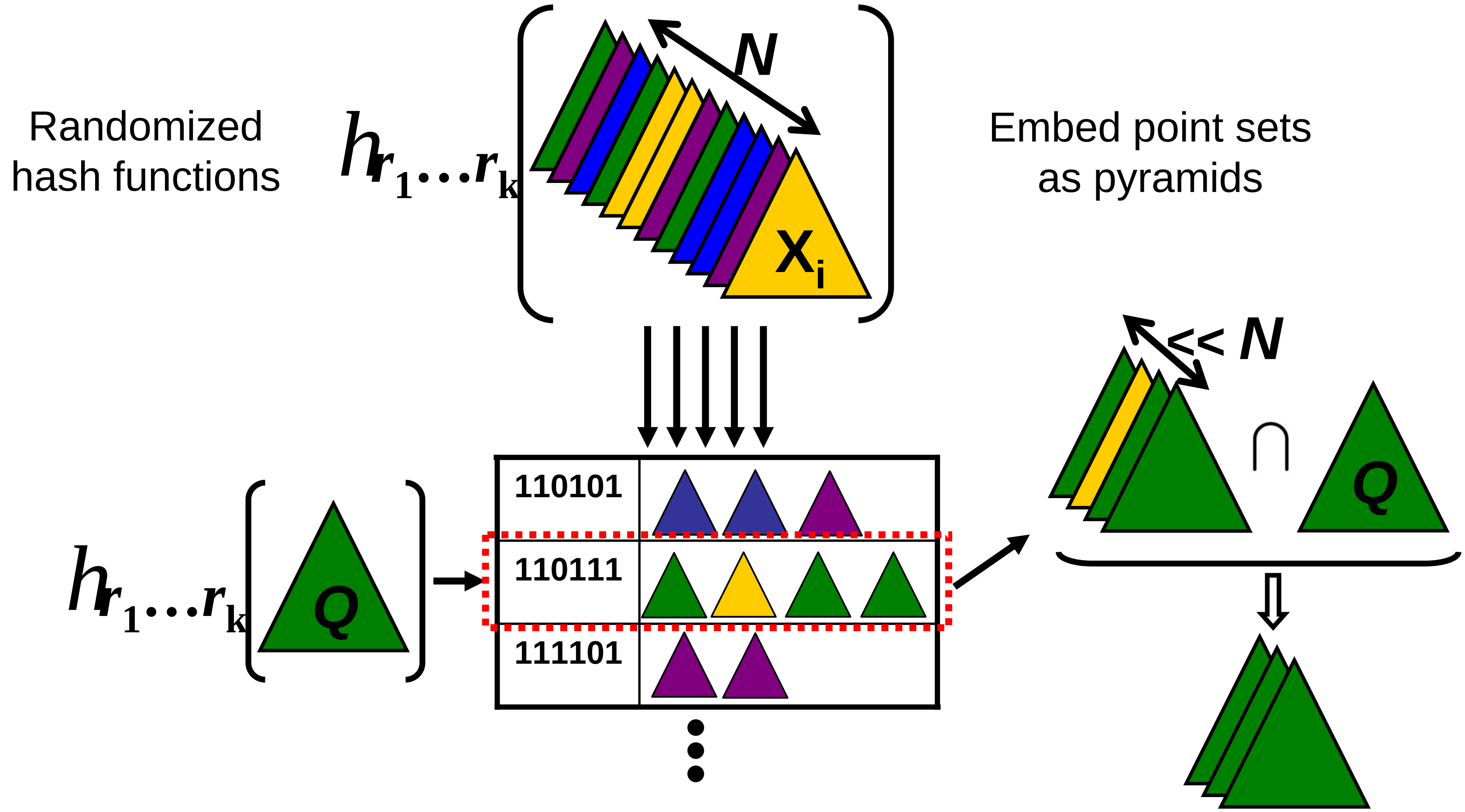
$$\underbrace{\Pr\left[h_{\vec{r}}(f(\mathbf{X})) = h_{\vec{r}}(f(\mathbf{Y}))\right]}_{\text{Probability of collision (hash bits equal)}} = 1 - \frac{1}{\pi}\cos^{-1}\left(\underbrace{\frac{f(\mathbf{X}) \cdot f(\mathbf{Y})}{\sqrt{|f(\mathbf{X})||f(\mathbf{Y})|}}}_{\text{Normalized pyramid match kernel value}}\right)$$

Matching-sensitive hash functions



$$\Pr\left[h_{\vec{r}}(f(\mathbf{X})) = h_{\vec{r}}(f(\mathbf{Y}))\right] = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{f(\mathbf{X}) \cdot f(\mathbf{Y})}{\sqrt{|f(\mathbf{X})||f(\mathbf{Y})|}}\right)$$

Sub-linear time search according to the partial match

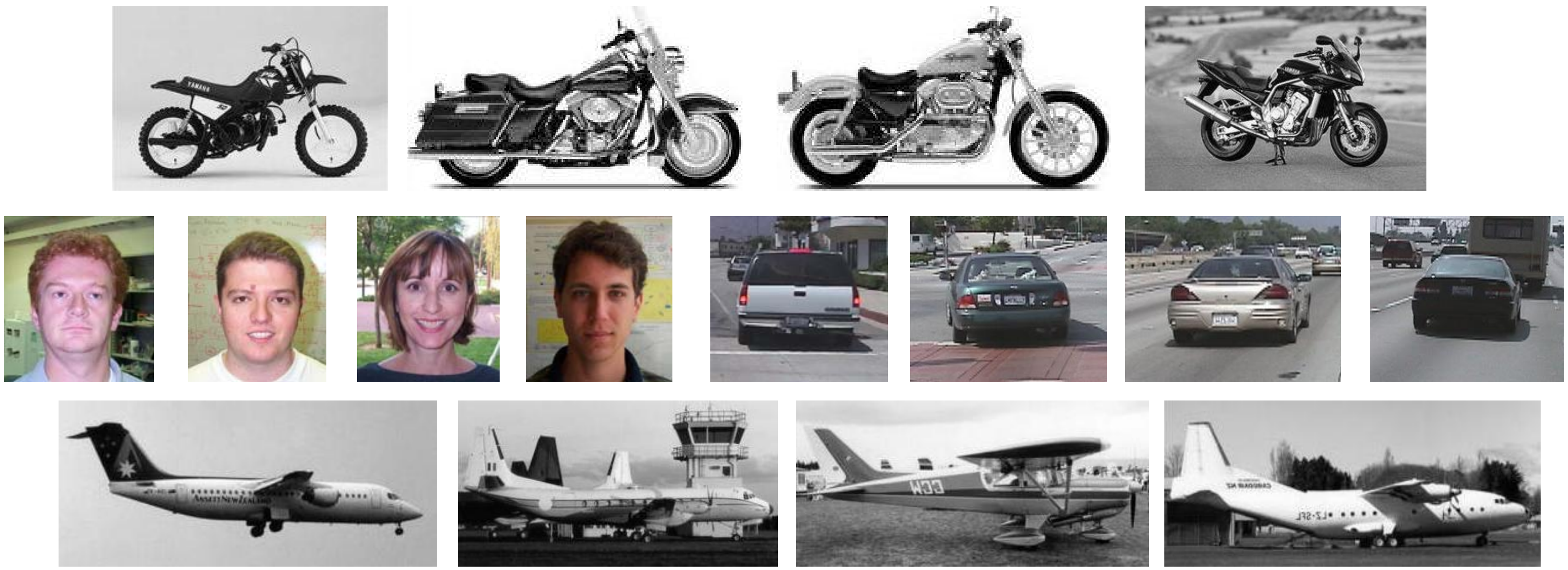


Probability of collision = normalized partial match similarity

Matching-sensitive image search

- Guaranteed retrieval of $(1+\epsilon)$-nearest neighbors in $O(N^{1/1+\epsilon})$ time.
- Applicable whether bins uniform in size or not (adjust weighting accordingly)
- LSH functions do not exist for *un-normalized* partial match

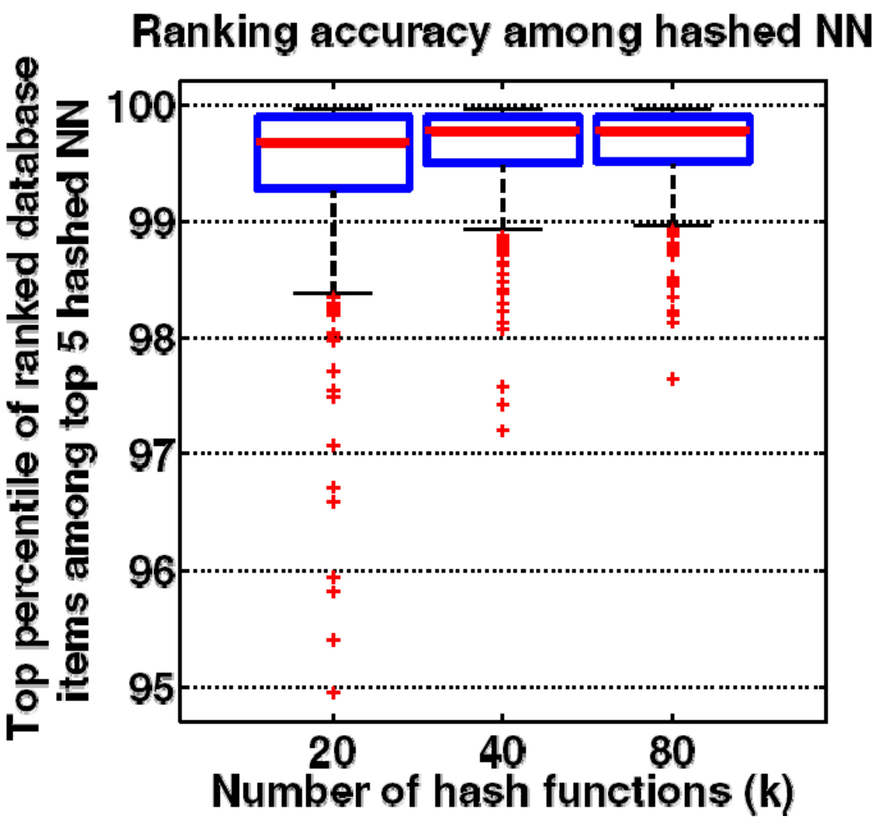
Caltech-4 image data set



- 4 categories
- 3188 total images

- Dense local SIFT features
- Uniform bin pyramids
- $\epsilon = 1.0$
- Consider 5 NN

Hashing results: Caltech-4 data

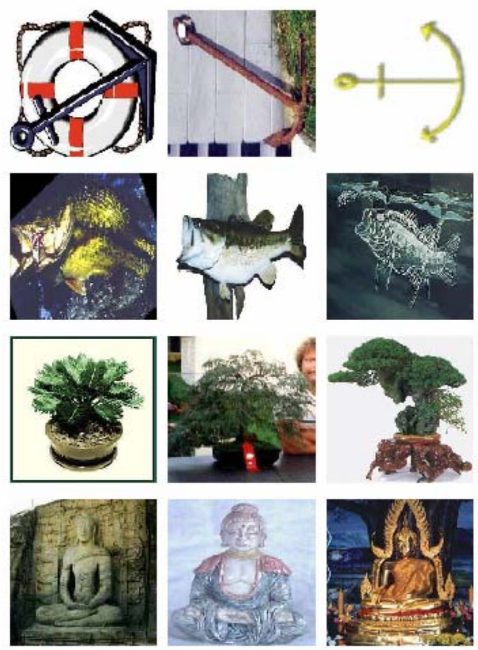


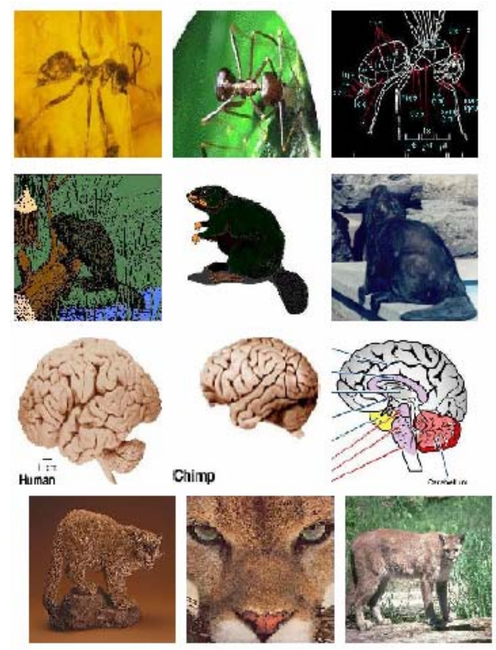
Search only 2.5% of database, get results very close to linear scan.

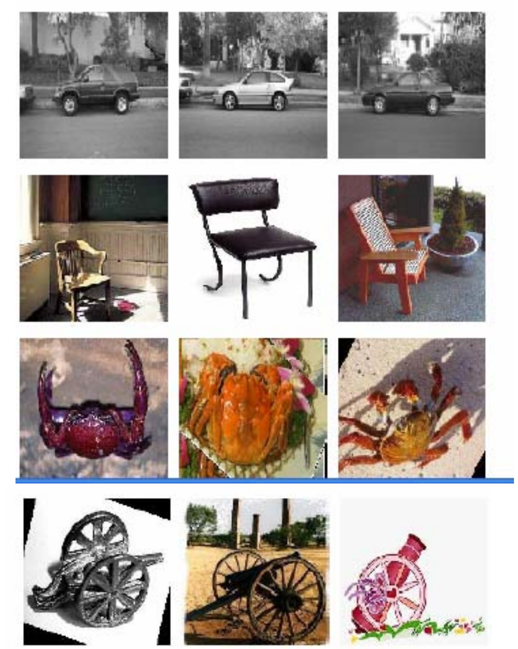
Mean relevance ratio = 0.97 (median = 1)

11*x* speedup, for *N*=3188

Caltech-101 image data set



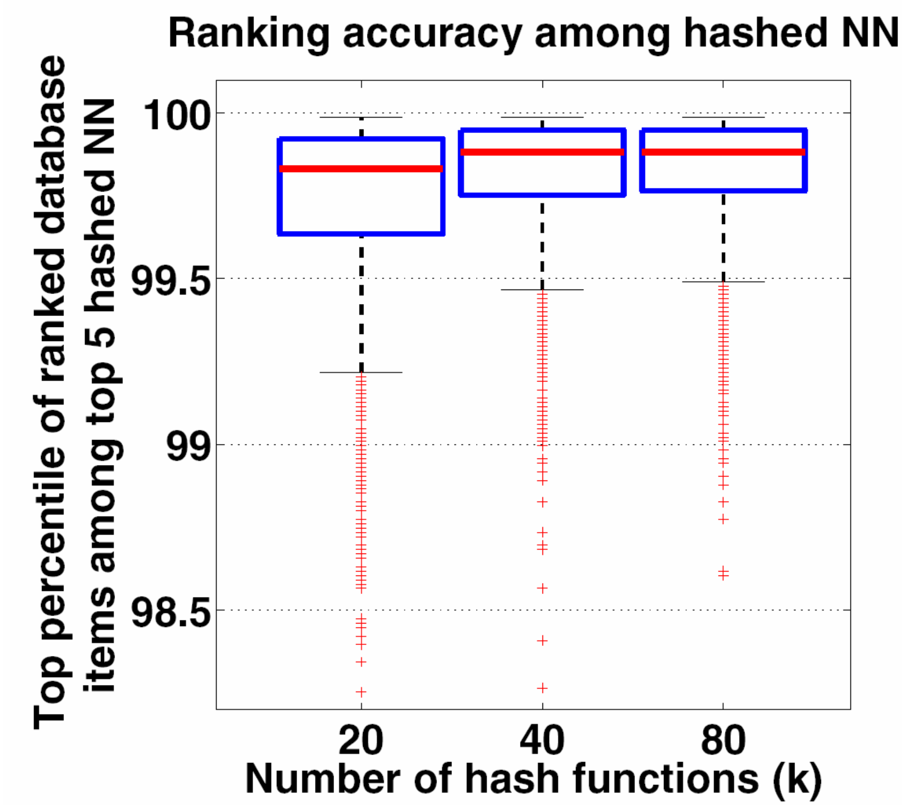




- 100 categories
- 40-800 images/class

- Dense local SIFT features+spatial position
- Uniform bin pyramids
- $\epsilon = 1.0$
- Consider 5 NN

Hashing results: Caltech-101 data

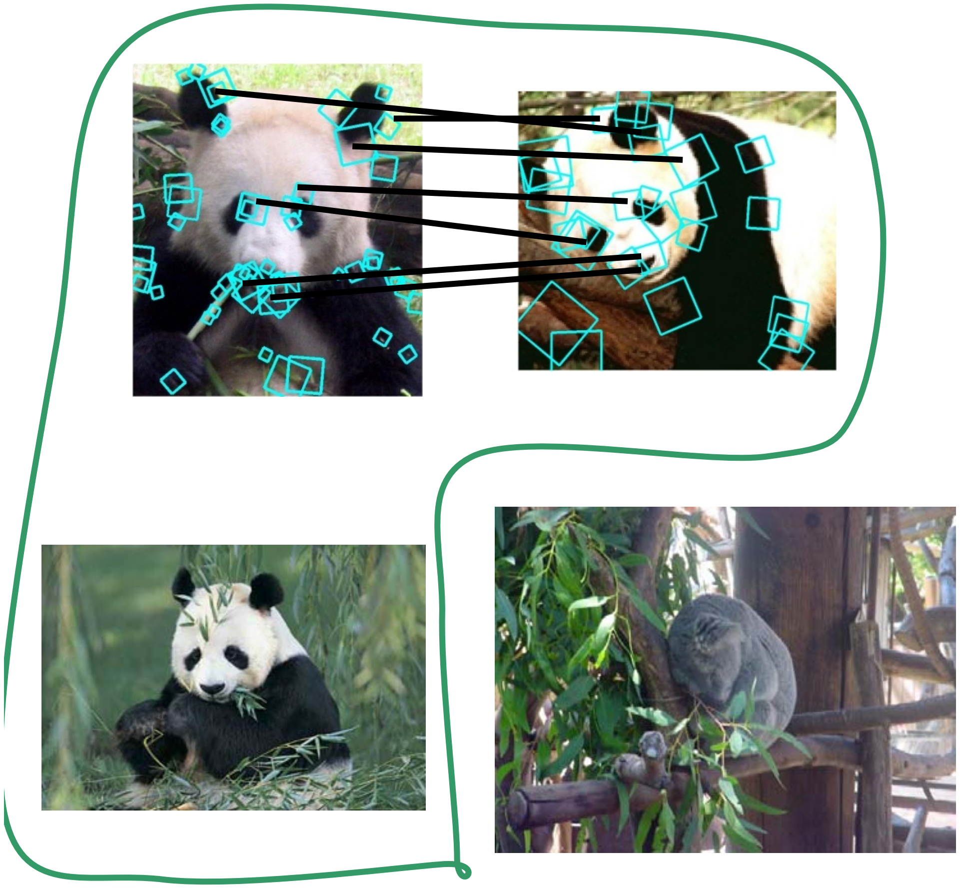


Search only 1.5% of database, get results very close to linear scan.

Mean relevance ratio = 0.76 (median = 1)

20*x* speedup, for *N*=6657

Considering external constraints



Correspondence measures are robust way to judge appearance/shape similarity…
but often we know more about (some) data than just their appearance.

How should available constraints affect how correspondences are interpreted?

Or any other image distance measure?

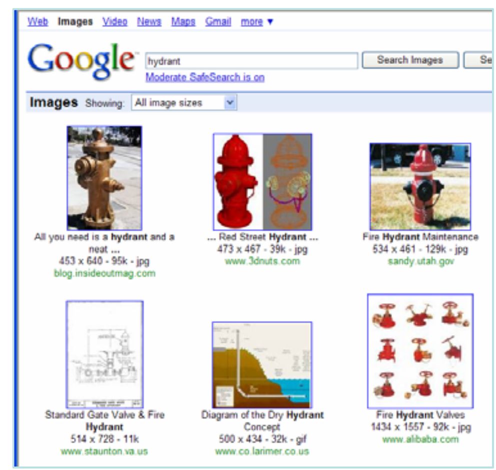
Similarity constraints for image collections

Fully labeled image databases

Problem-specific knowledge

Detected video shots, tracked objects

Partially labeled image databases



User feedback

Metric learning

- Exploit partially labeled data and/or (dis)similarity constraints to construct more useful distance function

- Can dramatically boost performance on clustering, indexing, classification tasks

- Number of existing techniques
  [Xing et al., Globerson and Roweis, Weinberger et al, Bar-Hillel et al., Schultz and Joachims, Frome et al., Davis et al.]

Problem

- Specialized distance measures have limited applicability for searching large databases
  - Exact search in high-d spaces break down, need good partitioning heuristics, can degenerate to linear scan in worst case
  - Approximate search techniques defined for particular “generic” measures, e.g., Hamming distance, L_p norms, inner product

- Related work: Shakhnarovich et al. 2003: select feature dimensions to hash that are more indicative of hidden parameter space

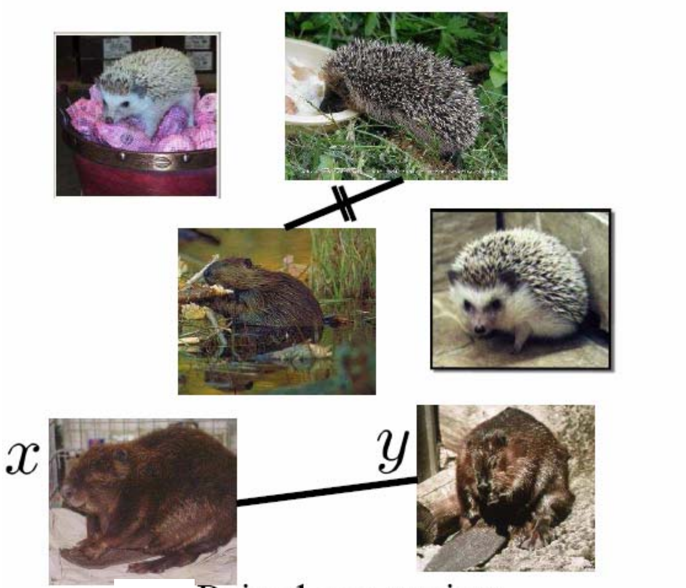
Fast similarity search for learned metrics

- Goal:
  - Maintain query time guarantees while performing approximate search with a learned metric

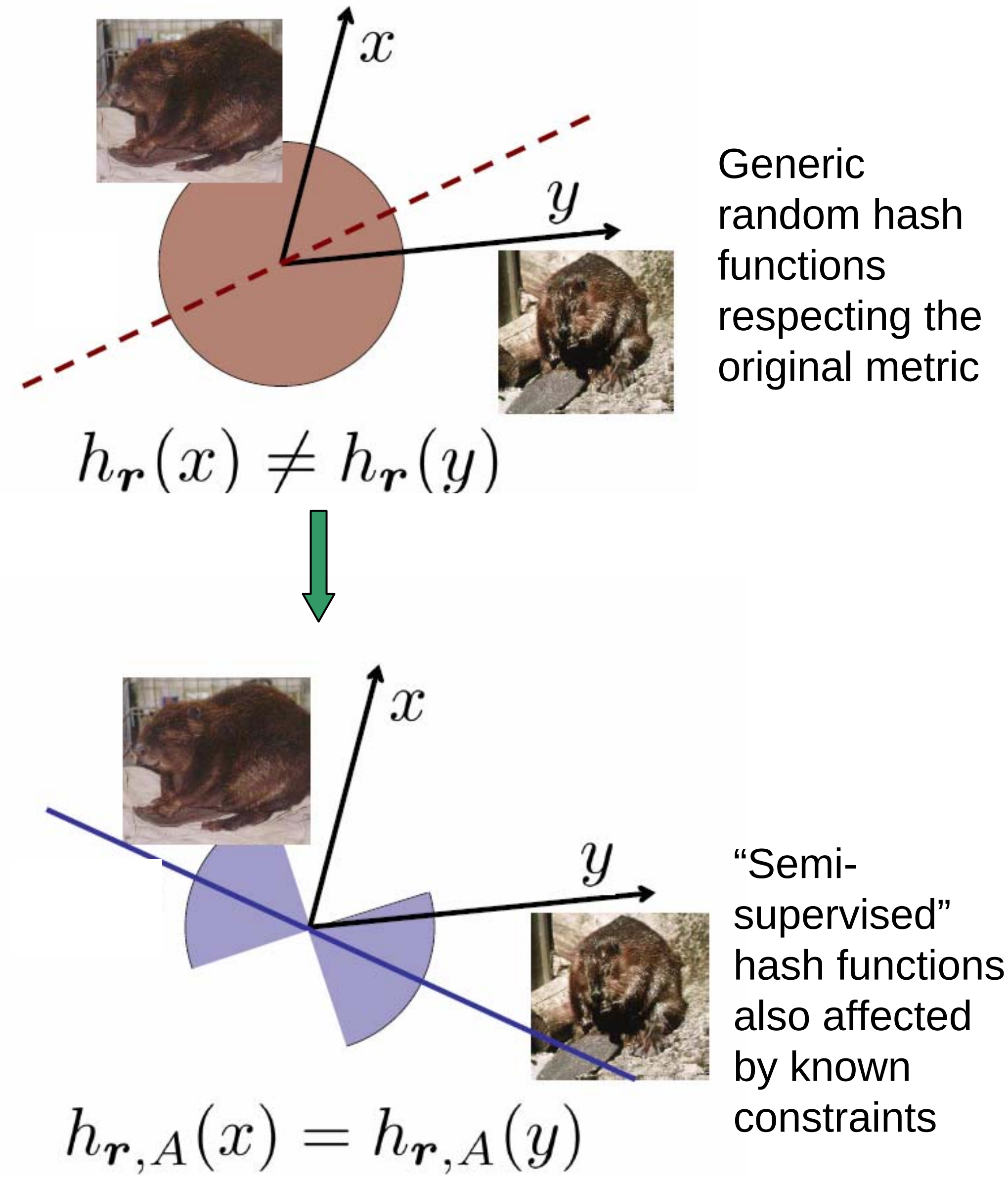
- Main idea:
  - Learn Mahalanobis distance parameterization
  - Use it to affect distribution from which random hash functions are selected
    - LSH functions that preserve the learned metric
    - Approximate NN search with existing methods

[Jain, Kulis, & Grauman, 2007]

Use partial knowledge given by the constraints to influence hash function selection.



Paired constraints



Mahalanobis distances

- Distance parameterized by p.d. $d \times d$ matrix A:

$$d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^T A (\boldsymbol{x}_i - \boldsymbol{x}_j)$$

- Similarity measure is associated generalized inner product (kernel)

$$s_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \;=\; \boldsymbol{x}_i^T A \boldsymbol{x}_j.$$

A matrix can be learned so that it reflects available constraints

[Xing et al. 2002, Schultz & Joachims 2003, Bar-Hillel et al. 2005, Globerson & Roweis 2005, Weinberger et al. 2006, Davis et al. 2007]

Consider two situations

- Explicit:
  - Relatively low-dimensional inputs
  - *A* is representable
- Implicit:
  - Very high-dimensional but sparse inputs (e.g., bag of words, multi-resolution histograms)
  - *A* cannot be explicitly represented

Explicit formulation

- Given learned metric with $A = G^T G$
- Generate parameterized hash functions:

$$h_{\boldsymbol{r},A}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T G\boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\Pr\left[h_{\boldsymbol{r},A}(\boldsymbol{x}_i) = h_{\boldsymbol{r},A}(\boldsymbol{x}_j)\right] = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\boldsymbol{x}_i^T A \boldsymbol{x}_j}{\sqrt{|G\boldsymbol{x}_i||G\boldsymbol{x}_j|}}\right)$$

Implicit formulation

- High-d inputs are sparse, but $A = G^T G$ may be dense $\longrightarrow$ we can't work with $\boldsymbol{r}^T G\boldsymbol{x}$.
- Intuition: need to express parameterization in terms of constrained data points.
  - Adopt *information-theoretic metric learning* approach of Davis, Kulis, Jain, Sra, and Dhillon
    - Efficient learning, flexibility in constraint specs, *kernelizable*, shown to perform well empirically
  - Derive new implicit update to evaluate $\boldsymbol{r}^T G\boldsymbol{x}$ indirectly

Information-theoretic metric learning

[Davis, Kulis, Jain, Sra, and Dhillon, 2007]

Minimize LogDet divergence while enforcing desired constraints:

$$\begin{aligned}
\min_{A \succeq 0} \quad & D_{\ell \mathrm{d}}(A, A_0) \\
\text{s. t.} \quad & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u \quad (i,j) \in \mathcal{S}, \\
& d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq \ell \quad (i,j) \in \mathcal{D}.
\end{aligned}$$

Optimized with iterative updates for each constraint:

$$\begin{aligned}
A_{t+1} &= A_t + \beta_t A_t (\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})(\boldsymbol{x}_{i_t} - \boldsymbol{x}_{j_t})^T A_t \\
&= A_t + \beta_t A_t \boldsymbol{v}_t \boldsymbol{v}_t^T A_t
\end{aligned}$$

Implicit hashing formulation

Work in kernel space,
but still think in terms
of factorized A matrix:

$$A_t = G_t^T G_t$$

Have metric
learning update:

$$A_{t+1} = A_t + \beta_t A_t \boldsymbol{v}_t \boldsymbol{v}_t^T A_t$$

Using this update, we derive **implicit update** that iteratively computes *G* in terms of kernel values with constrained inputs.

Implicit hashing formulation

And then use it to compute corresponding hash value as before:

$$\boldsymbol{r}^T G \underbrace{\phi(\boldsymbol{x})}_{\text{sparsely represented high-d input}} = \boxed{\boldsymbol{r}^T \phi(x)} + \sum_{i=1}^{c} \sum_{j=1}^{c} \underbrace{S_{ij} \boldsymbol{r}^T}\, \boxed{\phi(\boldsymbol{x}_i)}\, \underbrace{\phi(\boldsymbol{x}_j)^T}\, \boxed{\phi(\boldsymbol{x})}$$

sparsely represented high-d input

Efficiently computed as soon as we have learned metric

S is *c* x *c* matrix of coefficients that determine how much weight each pair of constrained inputs contributes to *G*

Implicit hashing formulation

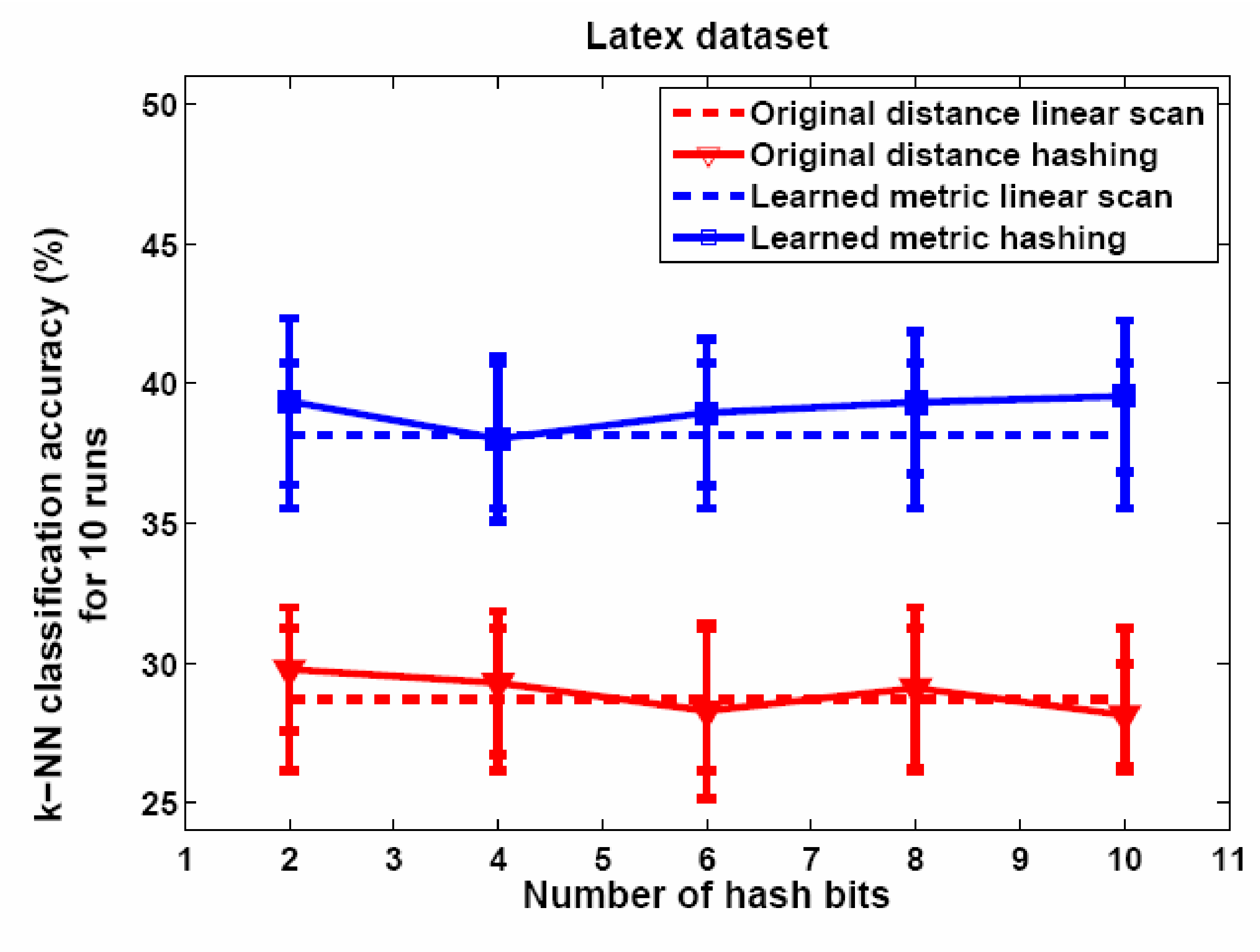
$$h_{\boldsymbol{r},A}(\phi(\boldsymbol{x})) = \begin{cases} 1, & \text{if } \boxed{\boldsymbol{r}^T\phi(\boldsymbol{x})} + \sum_{i=1}^{c} \gamma_i^r \boxed{\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x})} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$h_{\boldsymbol{r},A}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T G\boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Results: Systems dataset

- *Clarify* system of Ha et al. [PLDI 2007] uses machine learning to diagnose programmer errors, for example in Latex code.
- Representation: System collects program features during run-time
  - Function counts, Call-site counts, etc.
- Class labels: Program execution errors
- Nearest neighbor software support
  - Find error reports with similar program features
  - Point programmer to users with similar problems

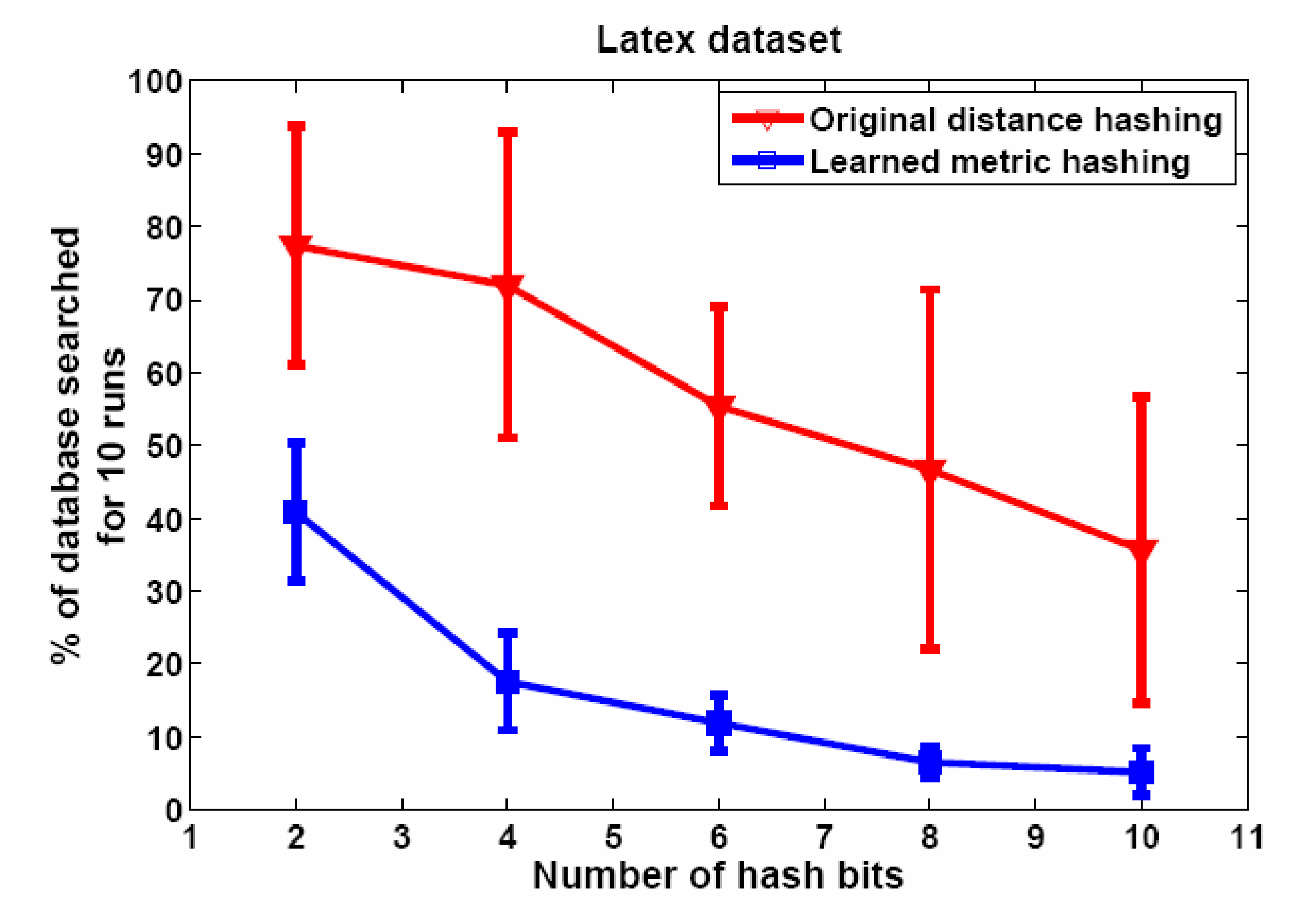
Results: Systems dataset



k = 4

- explicit
- d=20, N=3825
- Search about 5% of data
- 13*x* average speedup (up to 34*x*)

Results: Systems dataset

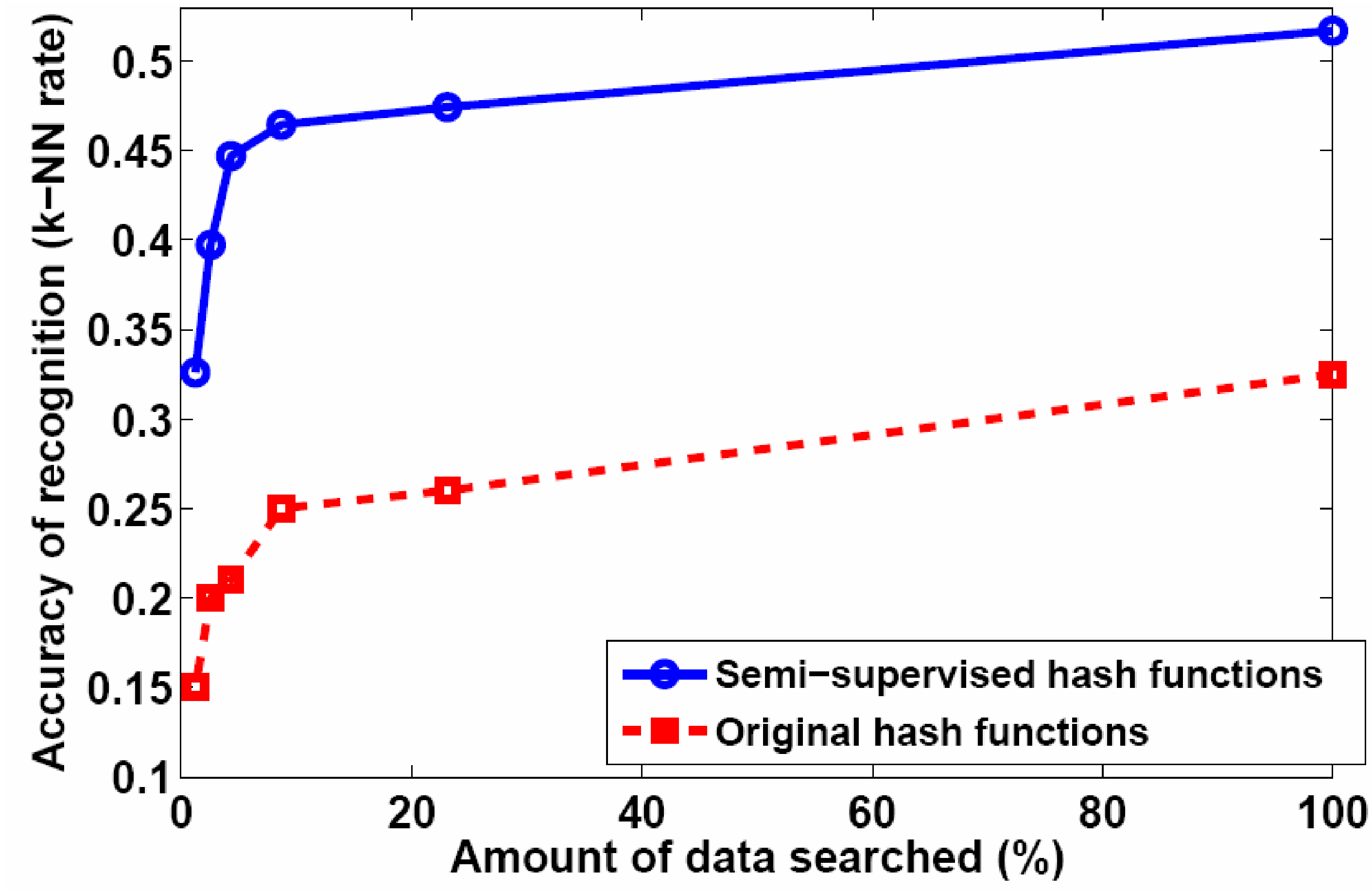


Epsilon = 1.5

- explicit
- *d*=20, *N*=3825
- Search about 5% of data
- 13*x* average speedup (up to 34*x*)

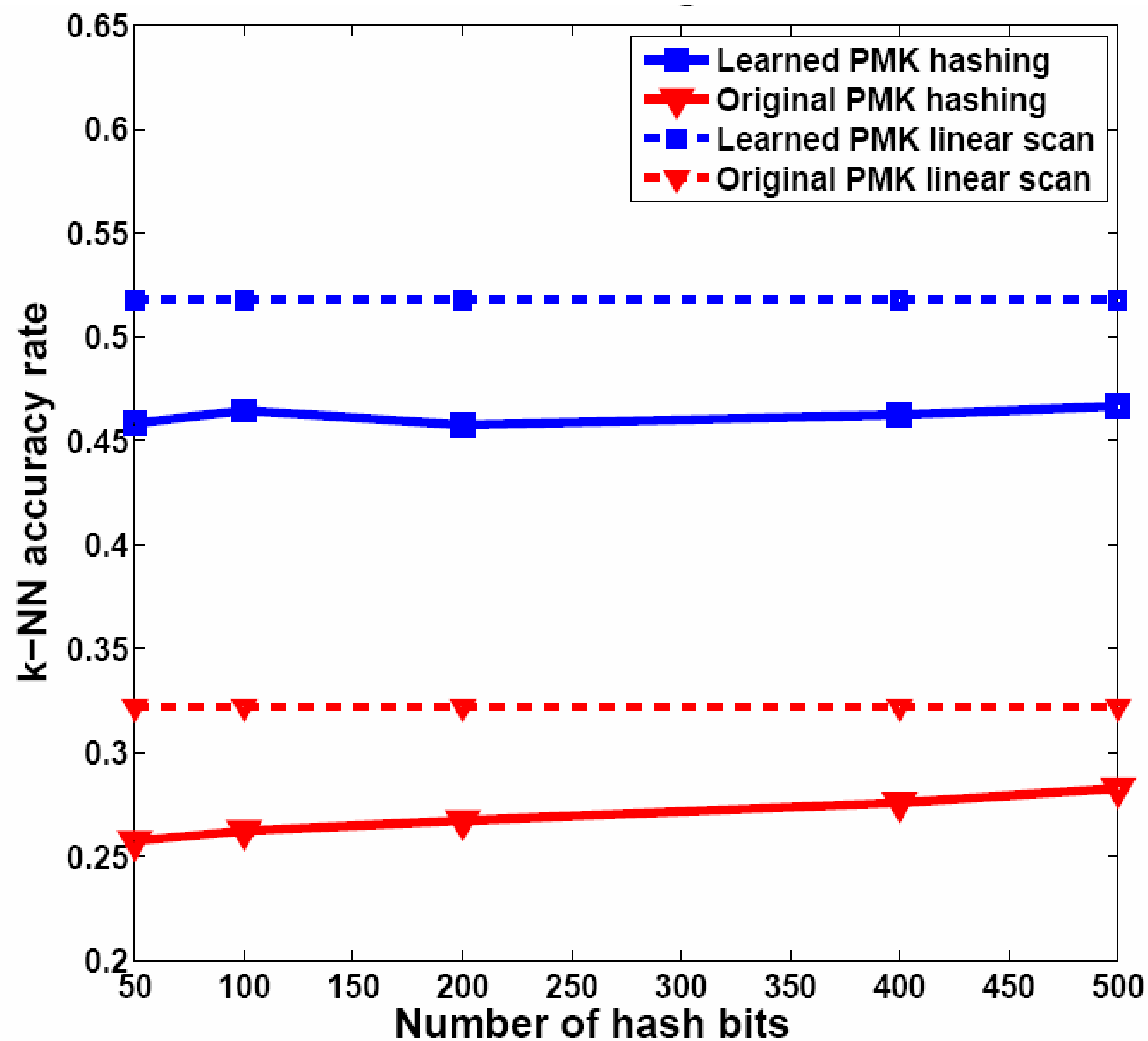
Results: Image dataset

Learn kernel for Caltech-101, initialize with PMK



- implicit
- d=O(10^6), N=1515
- Search about 5% of data
- 10x average speedup

Results: Image dataset

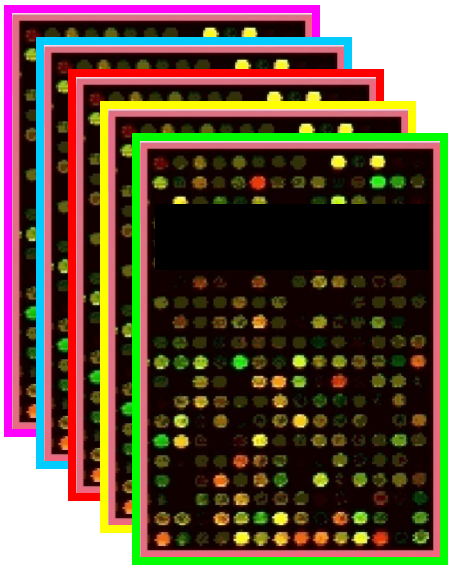


- implicit
- d=O(10^6), N=1515
- Search about 5% of data
- 10*x* average speedup

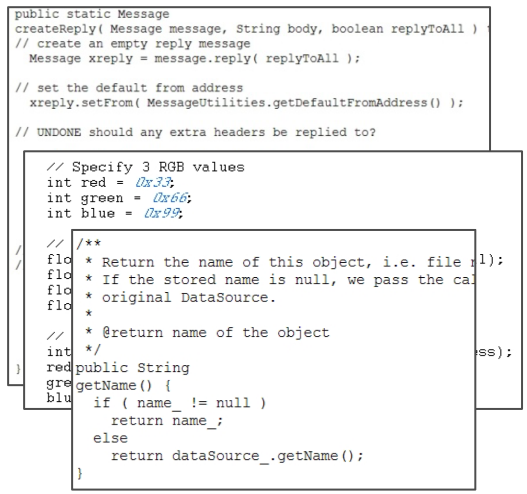
Summary

- Local image features useful, important to handle efficiently
- Introduced scalable methods to allow fast similarity search methods with
  - Local feature matching
  - Learned Mahalanobis metrics
- Key idea: design hash functions that encode matching process, or the constraints provided

Sets of features elsewhere



diseases as sets of gene expressions



methods as sets of instructions

documents as bags of word meanings

Future work

- Generalization of implicit formulation?
- Searching with local (learned) metrics
- Constraint selection strategies
- Practical impact for large-scale clustering, local learning
- Further evaluation with very large databases

Relevant papers

- Matching kernel:
  - K. Grauman and T. Darrell. The Pyramid Match Kernel: Efficient Learning with Sets of Features. Journal of Machine Learning Research (JMLR), 8 (Apr): 725--760, 2007. pdf

- Search with the matching:
  - K. Grauman and T. Darrell. Pyramid Match Hashing: Sub-Linear Time Indexing Over Partial Correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), Minneapolis, MN, June 2007. pdf

- Search for learned metrics:
  - P. Jain, B. Kulis, K. Grauman. Fast Similarity Search for Learned Metrics. *UTCS Technical Report #TR-07-48*, September, 2007. pdf