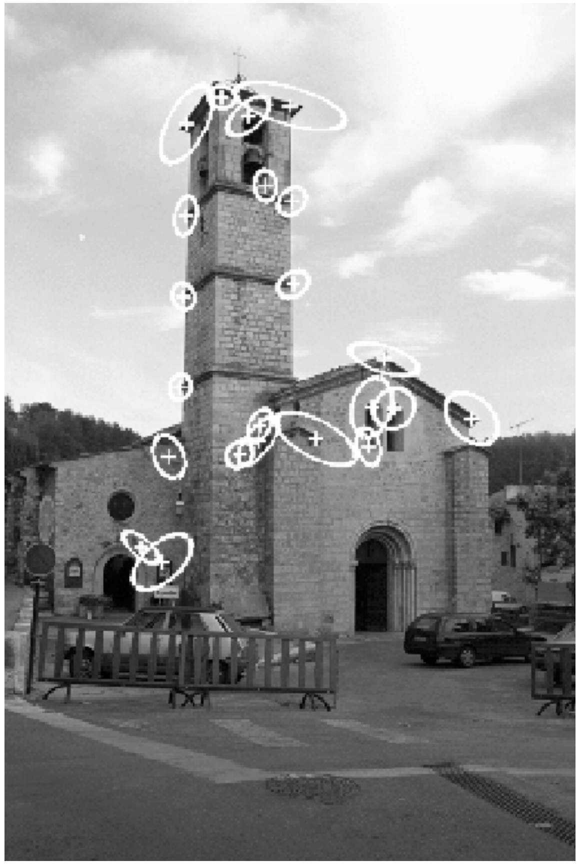
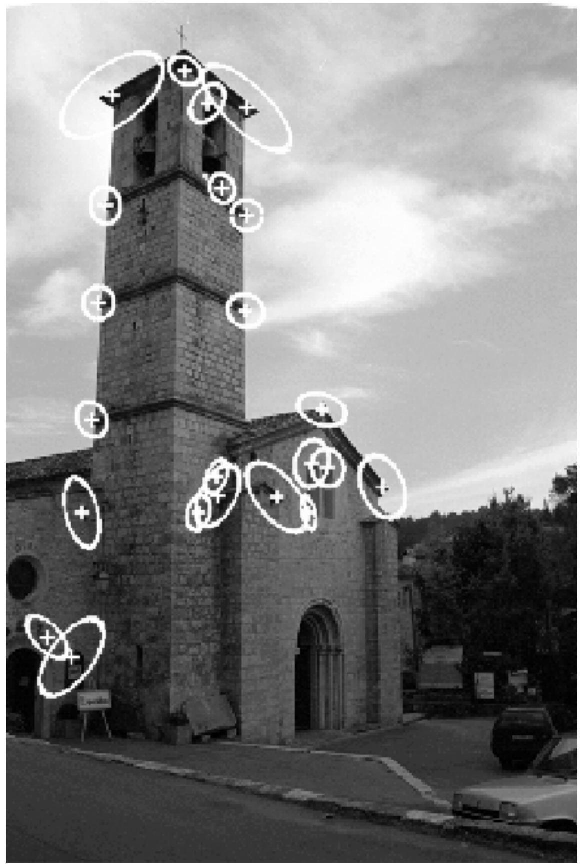
Efficient visual search of local features

Cordelia Schmid

Matches





22 correct matches

Visual search

change in viewing angle

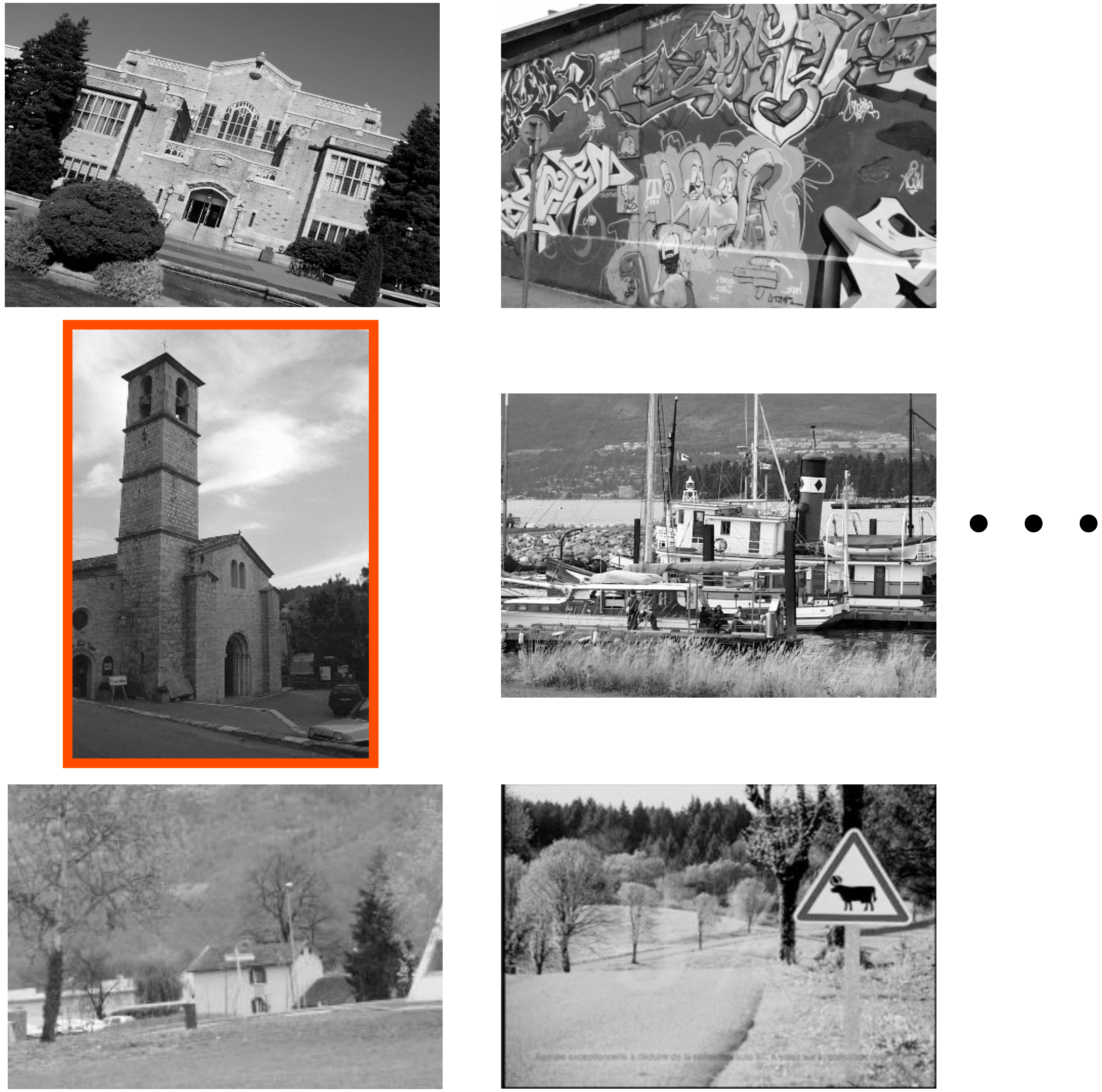
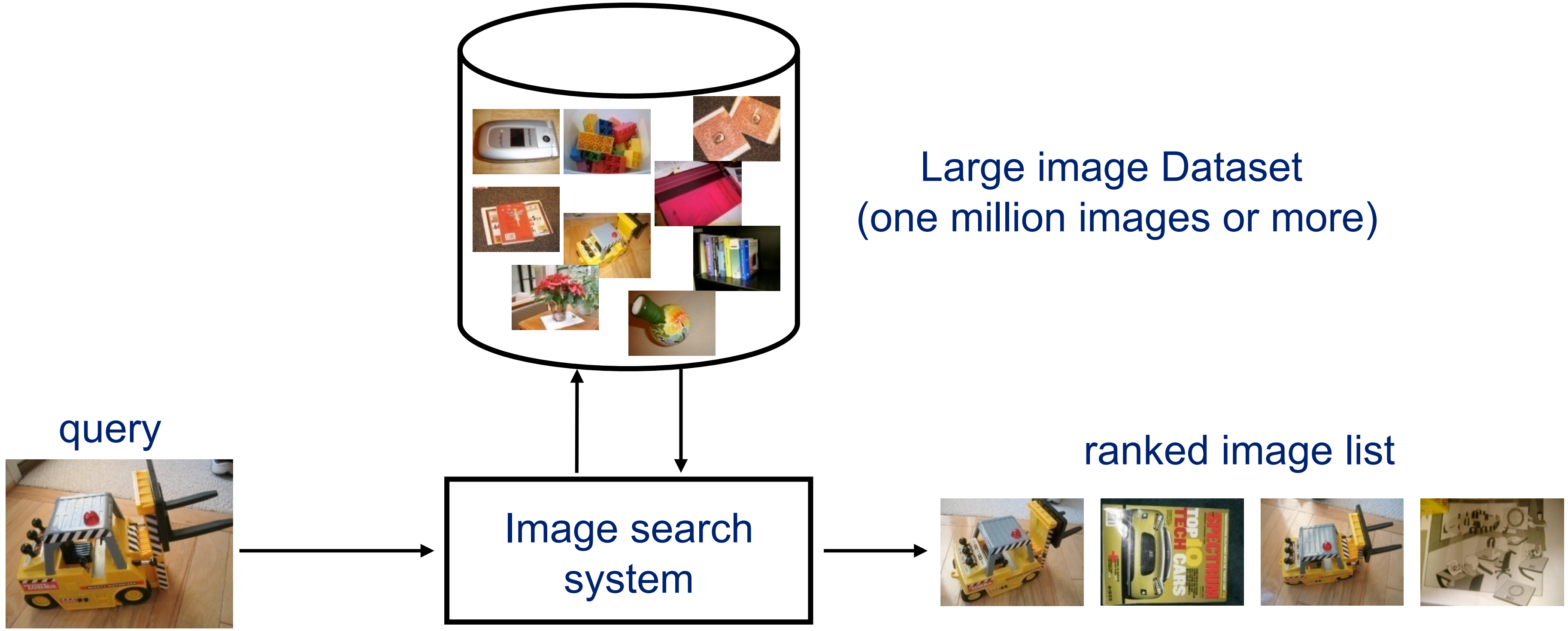


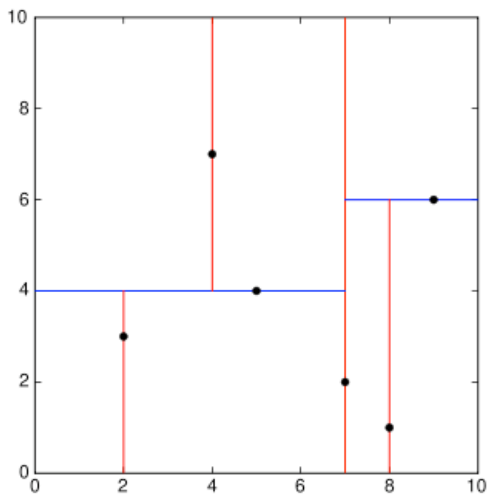
Image search system for large datasets



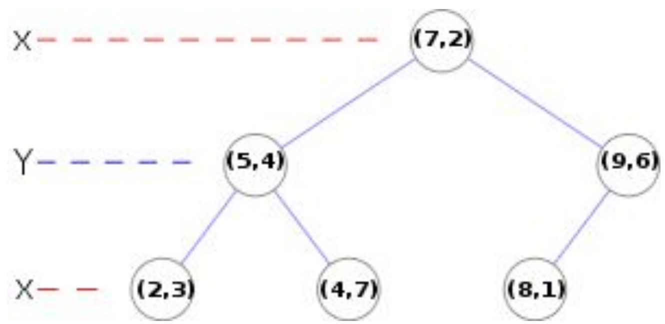
- **Issues** for very large databases
  - to reduce the query time
  - to reduce the storage requirements

Solution: fast descriptor search

- Complexity
  - O(nd) for n features and d dimensions
  - Linear in the number of features / images
- Speed up individual descriptor vector search
  - kd-trees (k dim. tree), approximate nearest neighbor search
  - Binary tree in which each node is a k-dimensional point
  - Every split is associated with one dimension

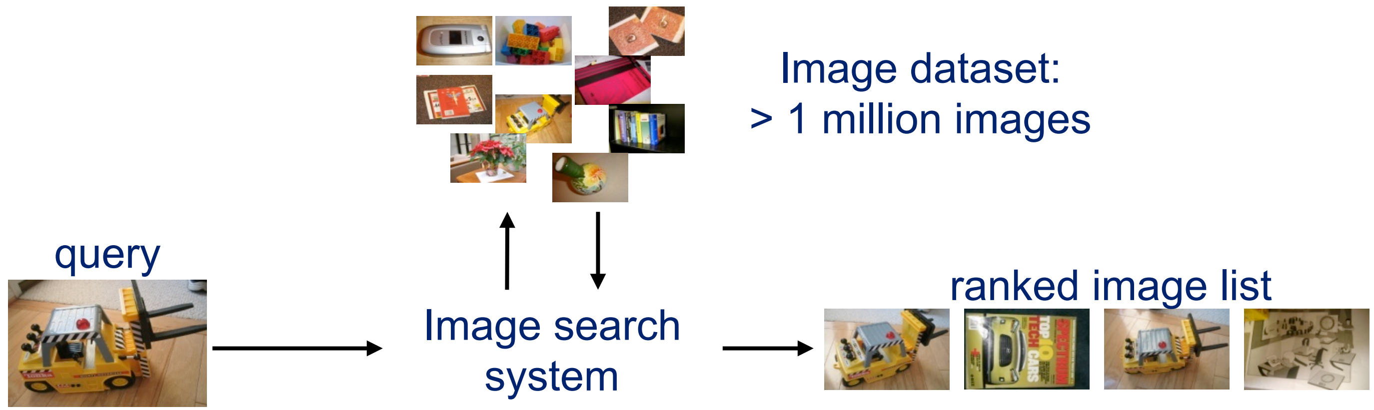


kd-tree decomposition



kd-tree

Large scale object/scene recognition



- Each image described by approximately 2000 descriptors
  - $2 * 10^9$ descriptors to index for one million images!

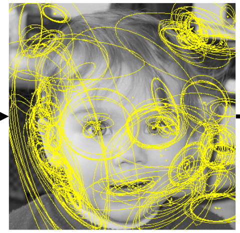
- Database representation in RAM:
  - Size of descriptors : 1 TB, search+memory intractable

Bag-of-features [Sivic&Zisserman'03]

Query image

Harris-Hessian-Laplace regions + SIFT descriptors

Set of SIFT descriptors



centroids (visual words)

Bag-of-features processing +*tf-idf* weighting

sparse frequency vector

Inverted file

querying

ranked image short-list

Geometric verification

[Chum & al. 2007]

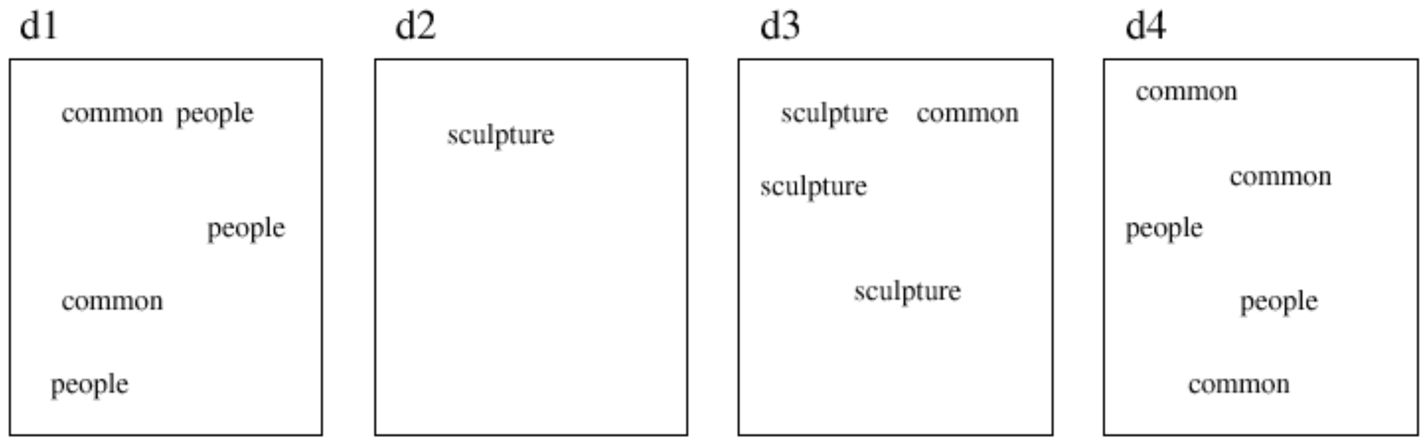
Re-ranked list

- "visual words":
  - 1 "word" (index) per local descriptor
  - only images ids in inverted file

  => 8 GB fits!

Indexing text with inverted files

Document collection:

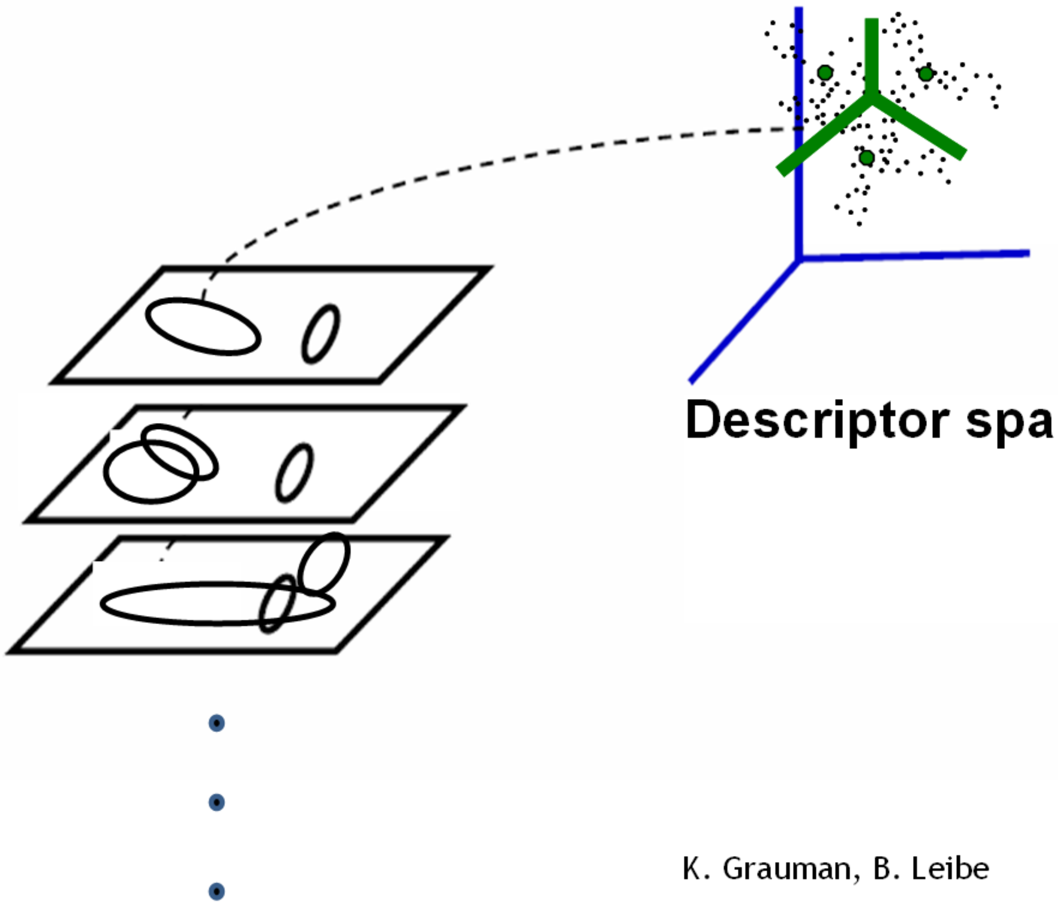


Inverted file:	**Term**	**List of hits** (occurrences in documents)
	People	[d1:hit hit hit], [d4:hit hit] …
	Common	[d1:hit hit], [d3: hit], [d4: hit hit hit] …
	Sculpture	[d2:hit], [d3: hit hit hit] …

Need to map feature descriptors to “visual words”

Visual words: main idea

Map high-dimensional descriptors to tokens/words by quantizing the feature space

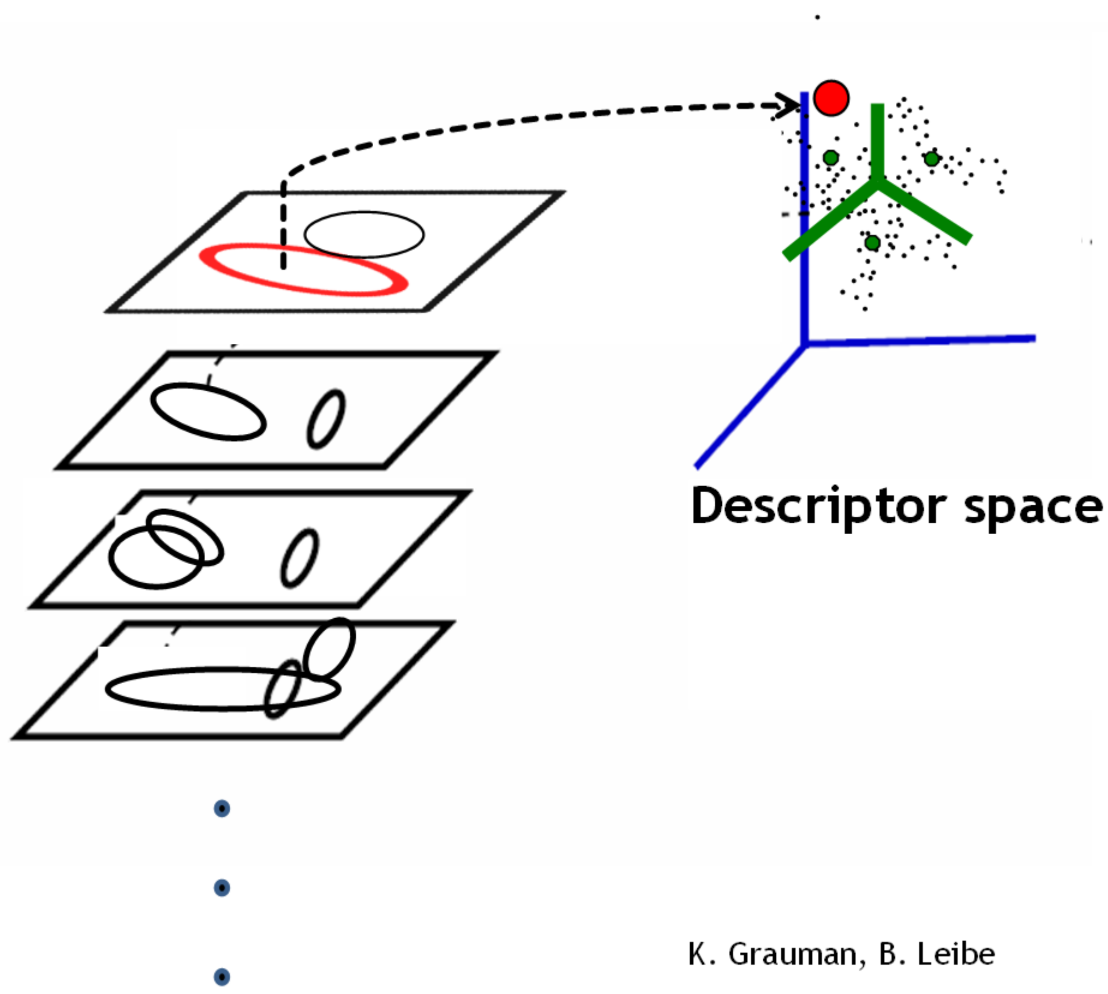


- Quantize via clustering, let cluster centers be the prototype "words"

K. Grauman, B. Leibe

Visual words: main idea

Map high-dimensional descriptors to tokens/words by quantizing the feature space

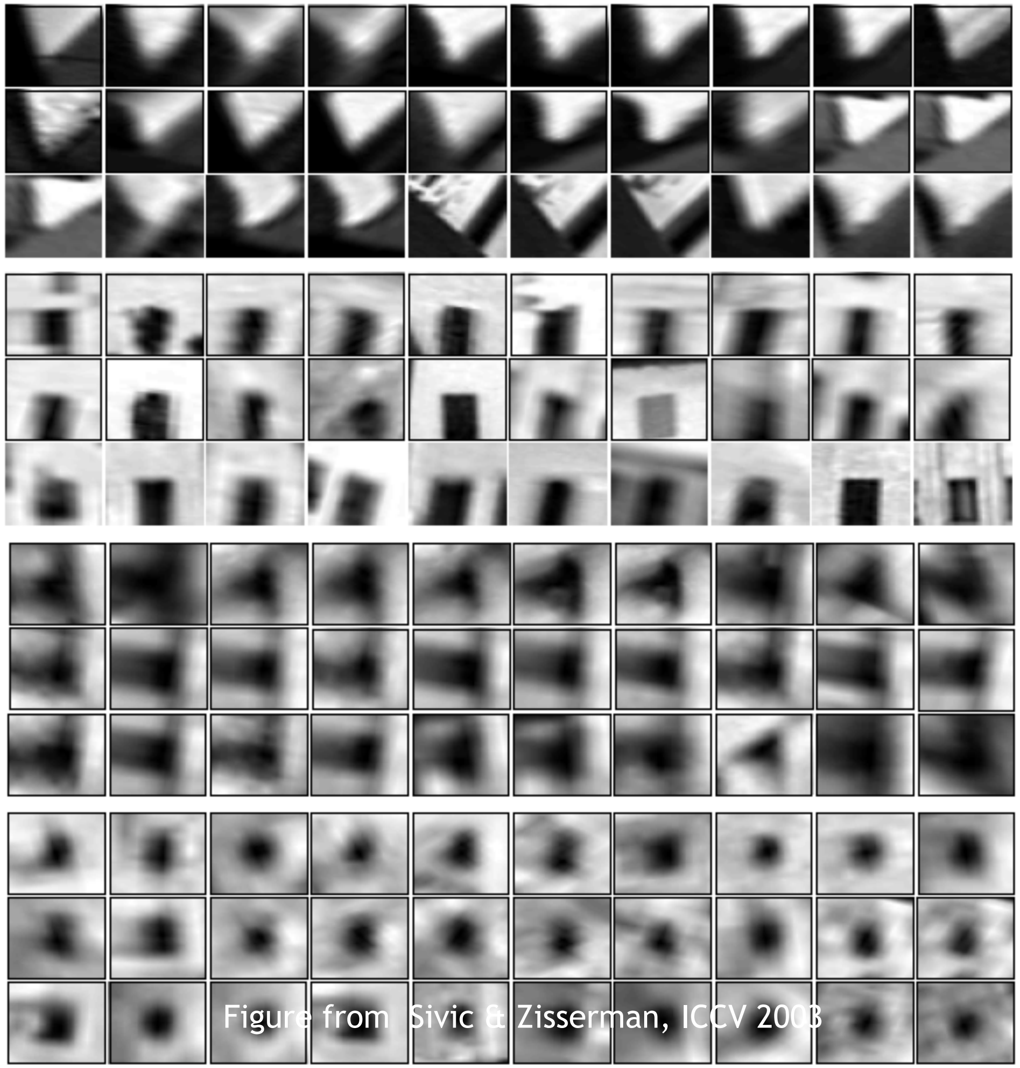


- Determine which word to assign to each new image region by finding the closest cluster center.

K. Grauman, B. Leibe

Visual words

- Example: each group of patches belongs to the same visual word



K-means clustering

- Minimizing sum of squared Euclidean distances between points x_i and their nearest cluster centers

- Algorithm:
  - Randomly initialize K cluster centers
  - Iterate until convergence:
    - Assign each data point to the nearest center
    - Recompute each cluster center as the mean of all points assigned to it

- Local minimum, solution dependent on initialization

- Initialization important, run several times, select best

Visual words

- Map descriptors to words by quantizing the feature space
  - Quantize via k-means clustering to obtain visual words
  - Assign descriptor to closest visual word

- Bag-of-features as approximate nearest neighbor search

Bag-of-features matching function $f_q(x, y) = \delta_{q(x),q(y)}$

where q(x) is a quantizer, i.e., assignment to visual word and $\delta_{a,b}$ is the Kronecker operator ($\delta_{a,b}$=1 iff a=b)

Inverted file index for images comprised of visual words

frame #5

frame #10

Word number		List of image numbers
1	→	5,10, ...
2	→	10,...
...		...

• Score each image by the number of common visual words (tentative correspondences)

• Dot product between bag-of-features

• Fast for sparse vectors !

Inverted file index for images comprised of visual words

- Weighting with tf-idf score: weight visual words based on their frequency

  - Tf: normalized term (word) ti frequency in a document dj

$$tf_{ij} = n_{ij} / \sum_k n_{kj}$$

  - Idf: inverse document frequency, total number of documents divided by number of documents containing the term ti

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

Tf-Idf: $$tf - idf_{ij} = tf_{ij} \cdot idf_i$$

Approximate nearest neighbor search evaluation

- ANN algorithms usually returns a short-list of nearest neighbors
  - this short-list is supposed to contain the NN with high probability
  - exact search may be performed to re-order this short-list

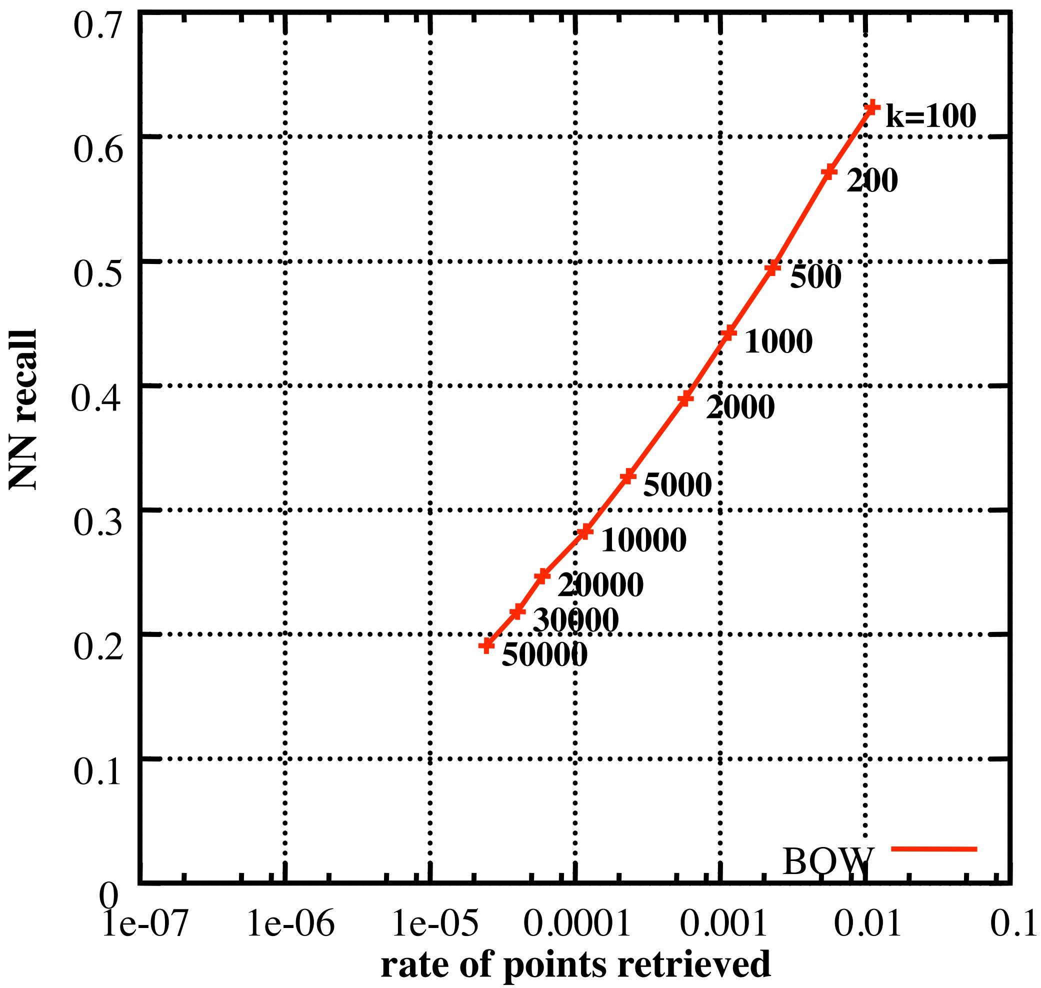
- Proposed quality evaluation of ANN search: trade-off between
  - **Accuracy**: **NN recall** = probability that *the* NN is in this list

    against

  - **Ambiguity removal** = proportion of vectors in the short-list
    - the lower this proportion, the more information we have about the vector
    - the lower this proportion, the lower the complexity if we perform exact search on the short-list

- ANN search algorithms usually have some parameters to handle this trade-off

ANN evaluation of bag-of-features



- ANN algorithms returns a list of potential neighbors
- **Accuracy**: **NN recall** = probability that *the* NN is in this list
- **Ambiguity removal**: = proportion of vectors in the short-list
- In BOF, this trade-off is managed by the number of clusters *k*

Vocabulary size

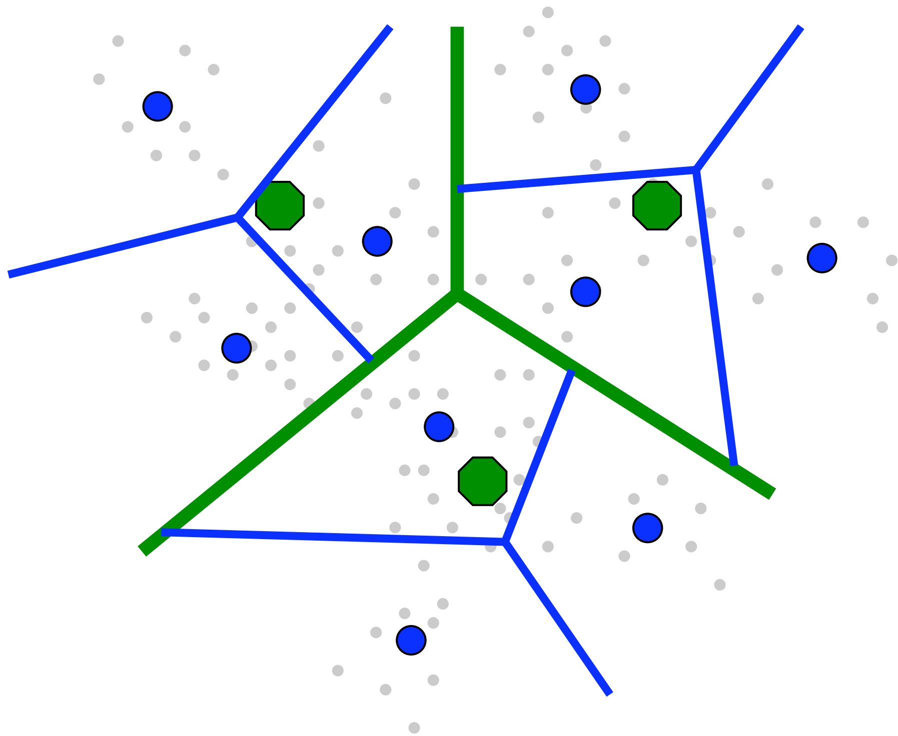
- The intrinsic matching scheme performed by BOF is weak
  - for a “small” visual dictionary: too many false matches
  - for a “large” visual dictionary: complexity, true matches are missed

- No good trade-off between “small” and “large” !
  - either the Voronoi cells are too big
  - or these cells can’t absorb the descriptor noise

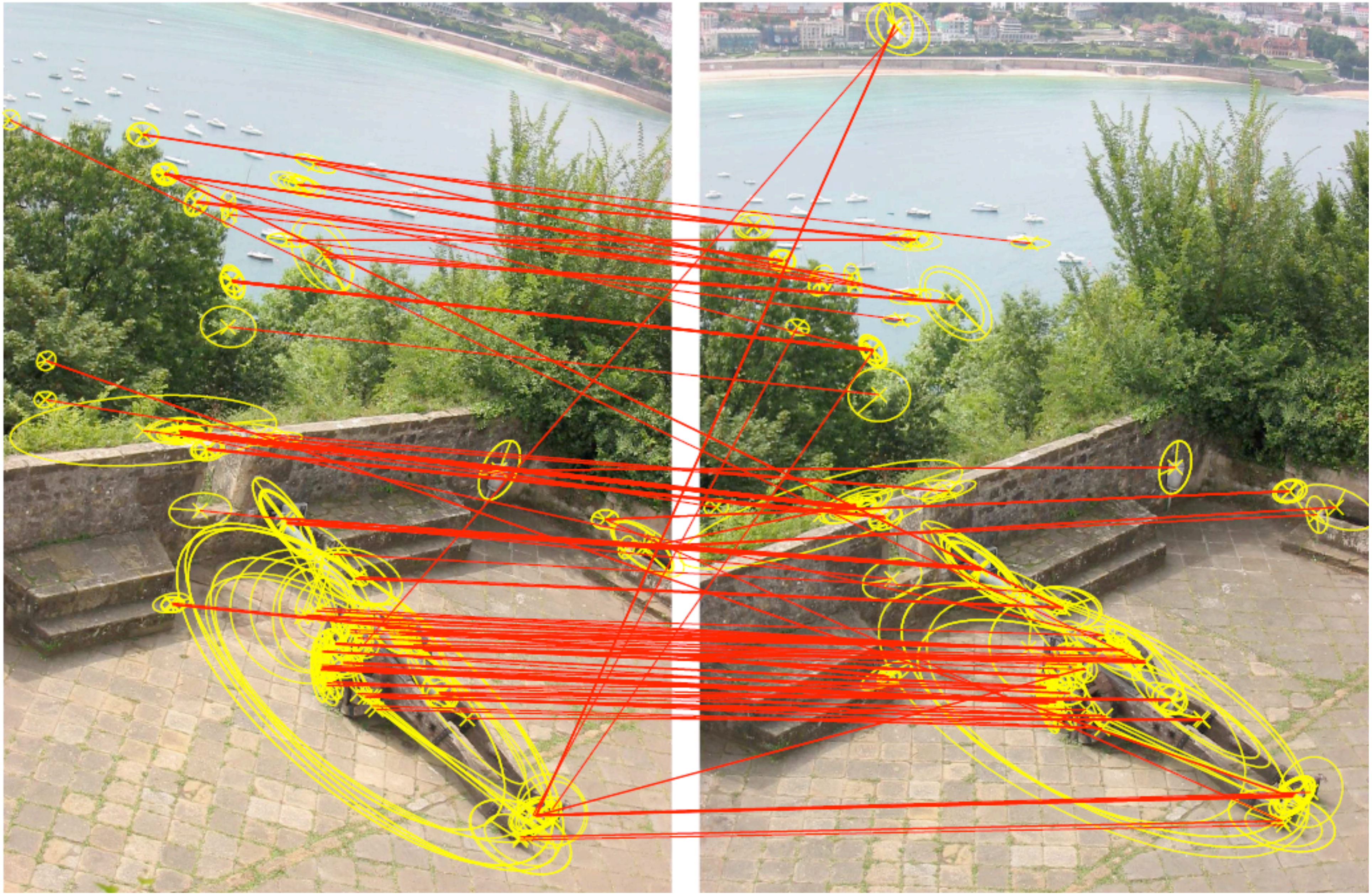
  → intrinsic approximate nearest neighbor search of BOF is not sufficient

Hierarchical clustering

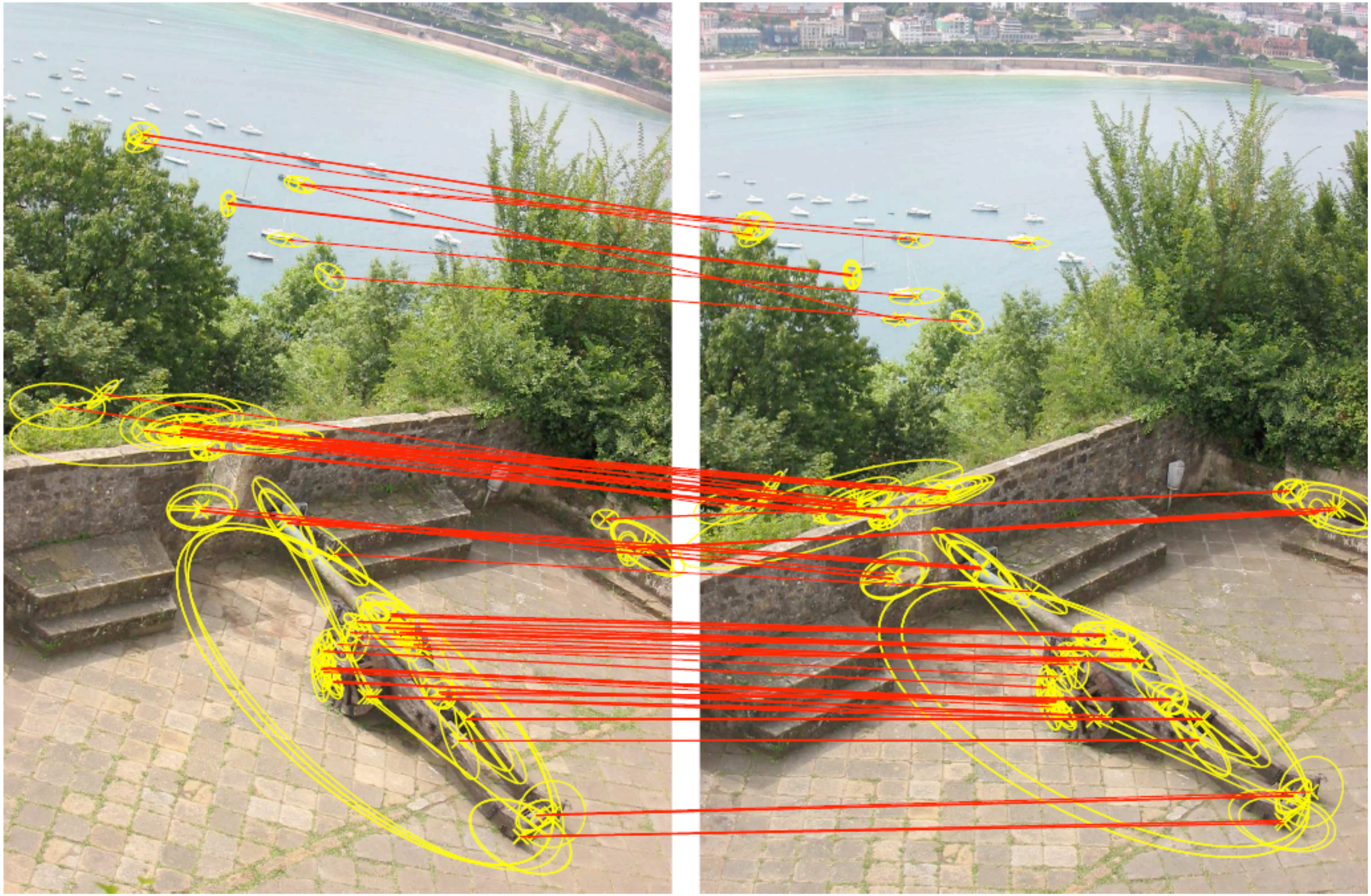
- Hierarchical clustering: fast assignment in case of large vocabularies
  – Vocabulary tree [Nister & Stewenius, CVPR 2006]
- Combined with multiple assignment



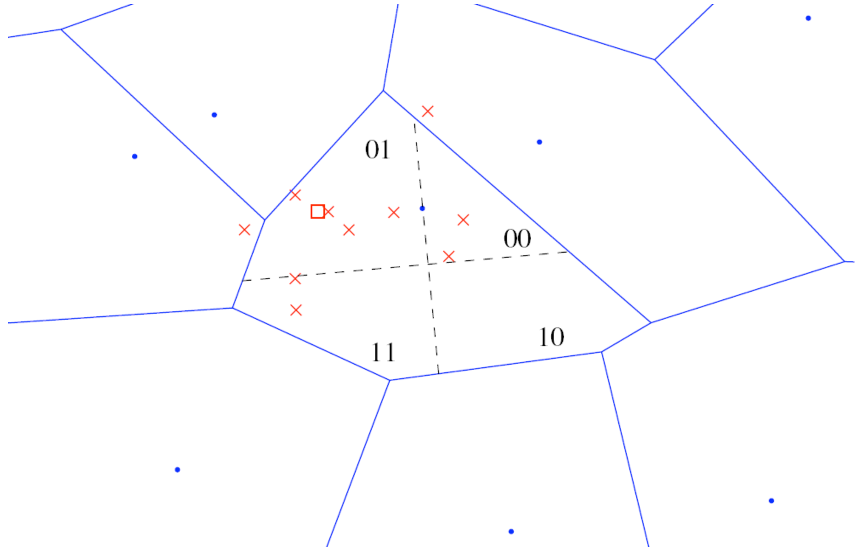
20K visual word: false matches



200K visual word: good matches missed



Hamming Embedding [Jegou et al. ECCV'08]



Representation of a descriptor *x*

- Vector-quantized to *q(x)* as in standard BOF
- \+ short binary vector *b(x)* for an additional localization in the Voronoi cell

Two descriptors x and y match iif

$$f_{\text{HE}}(x,y) = \begin{cases} (\text{tf-idf}(q(x)))^2 & \text{if } q(x) = q(y) \\ & \quad \text{and } h\left(b(x), b(y)\right) \leq h_t \\ 0 & \text{otherwise} \end{cases}$$

where h(*a*,*b*) Hamming distance

Hamming Embedding

•Nearest neighbors for Hamming distance ≈ those for Euclidean distance
→ a metric in the embedded space reduces dimensionality curse effects

•Efficiency

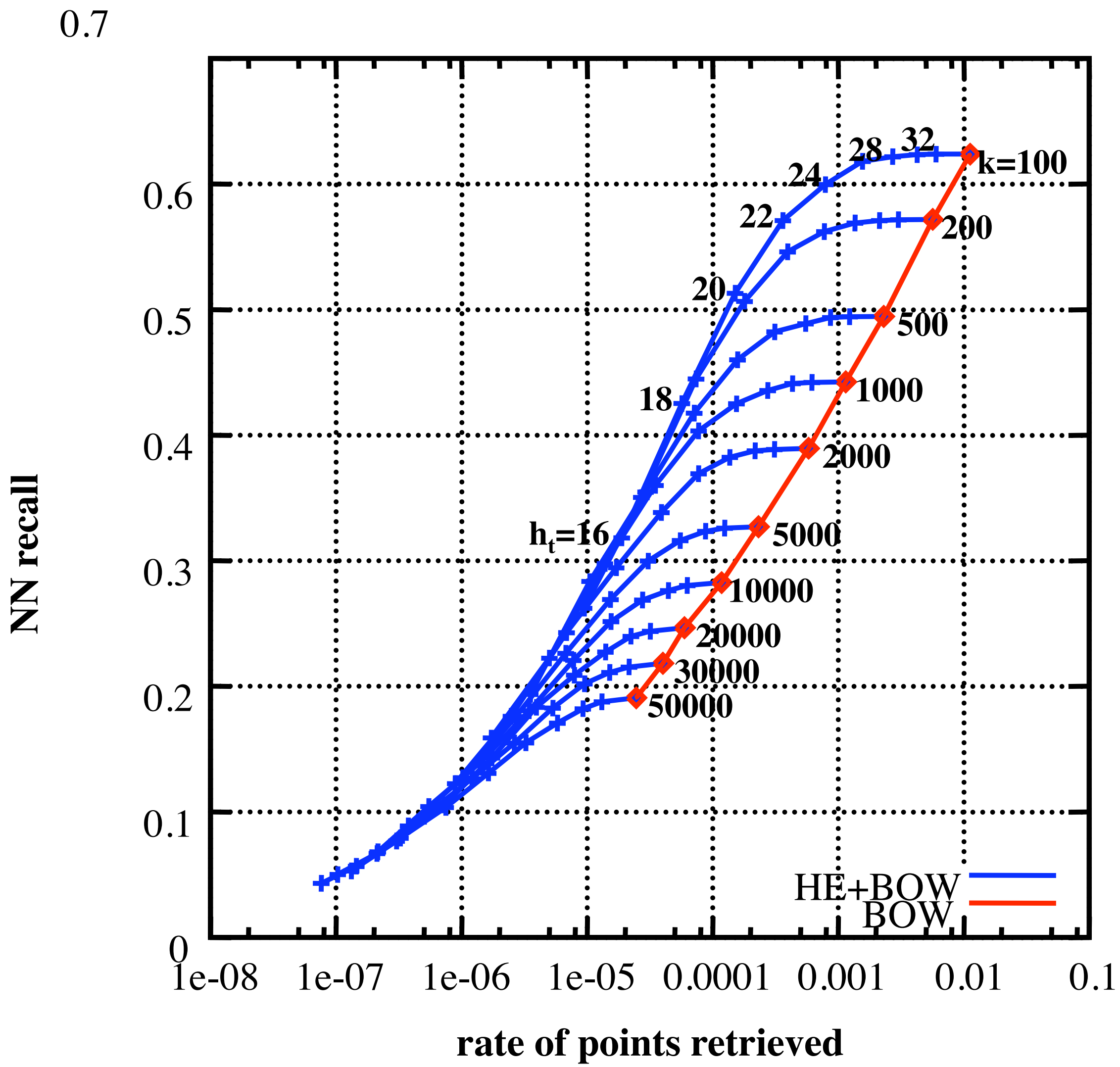
– Hamming distance = very few operations
– Fewer random memory accesses: 3 x faster that BOF with same dictionary size!

Hamming Embedding

- **Off-line** (given a quantizer)
  - draw an orthogonal projection matrix P of size $d_b \times d$
  - $\rightarrow$ this defines d_b random projection directions
  - for each Voronoi cell and projection direction, compute the median value for a learning set

- **On-line**: compute the binary signature b(x) of a given descriptor
  - project x onto the projection directions as $z(x) = (z_1, \ldots z_{db})$
  - $b_i(x) = 1$ if $z_i(x)$ is above the learned median value, otherwise 0

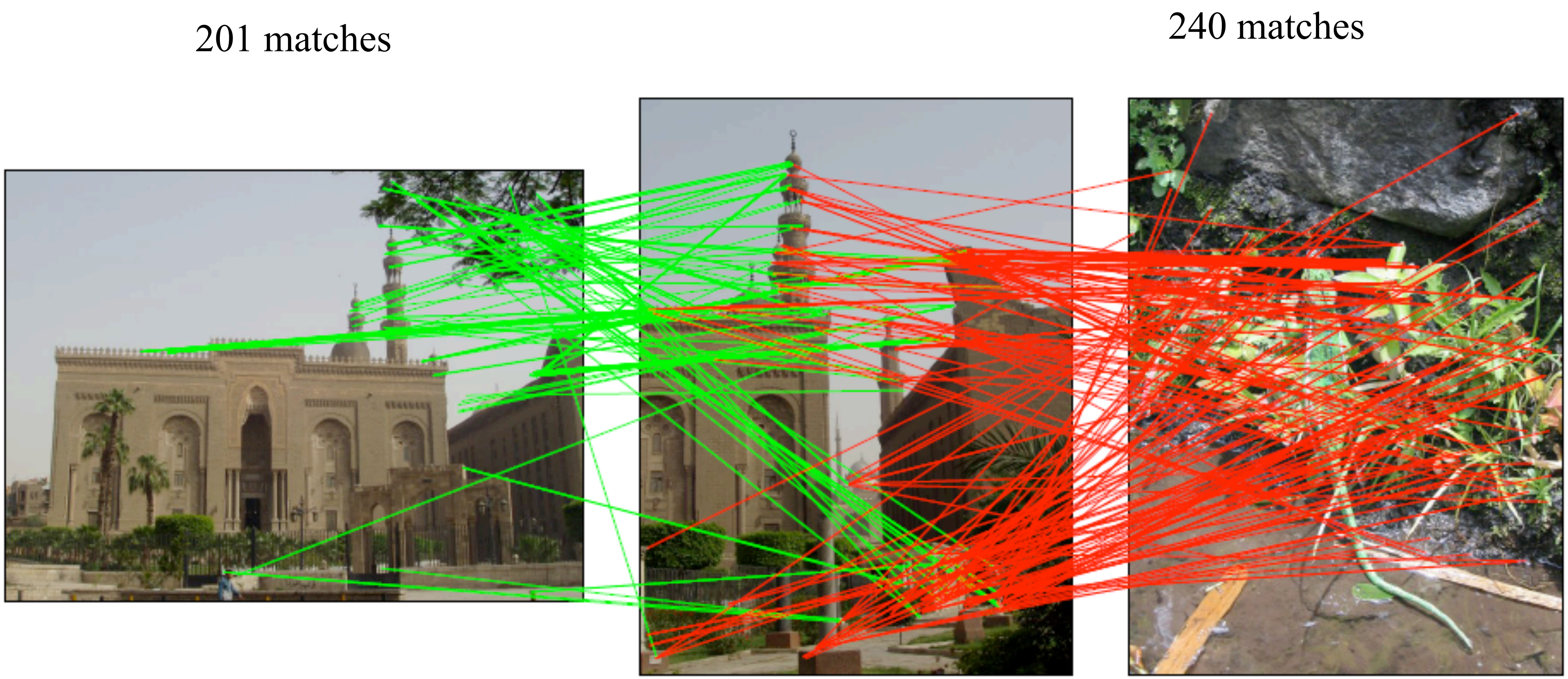
ANN evaluation of Hamming Embedding



compared to BOW: at least 10 times less points in the short-list for the same level of accuracy

Hamming Embedding provides a much better trade-off between recall and ambiguity removal

Matching points - 20k word vocabulary

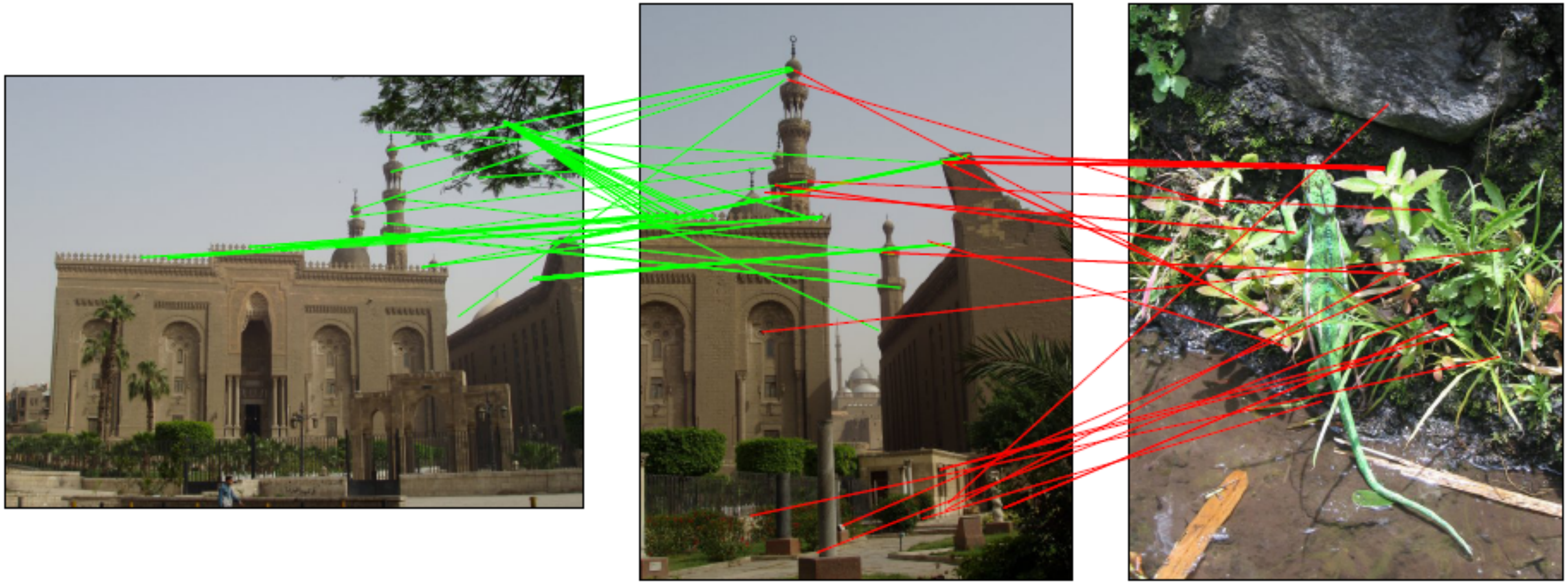


Many matches with the non-corresponding image!

Matching points - 200k word vocabulary

69 matches

35 matches

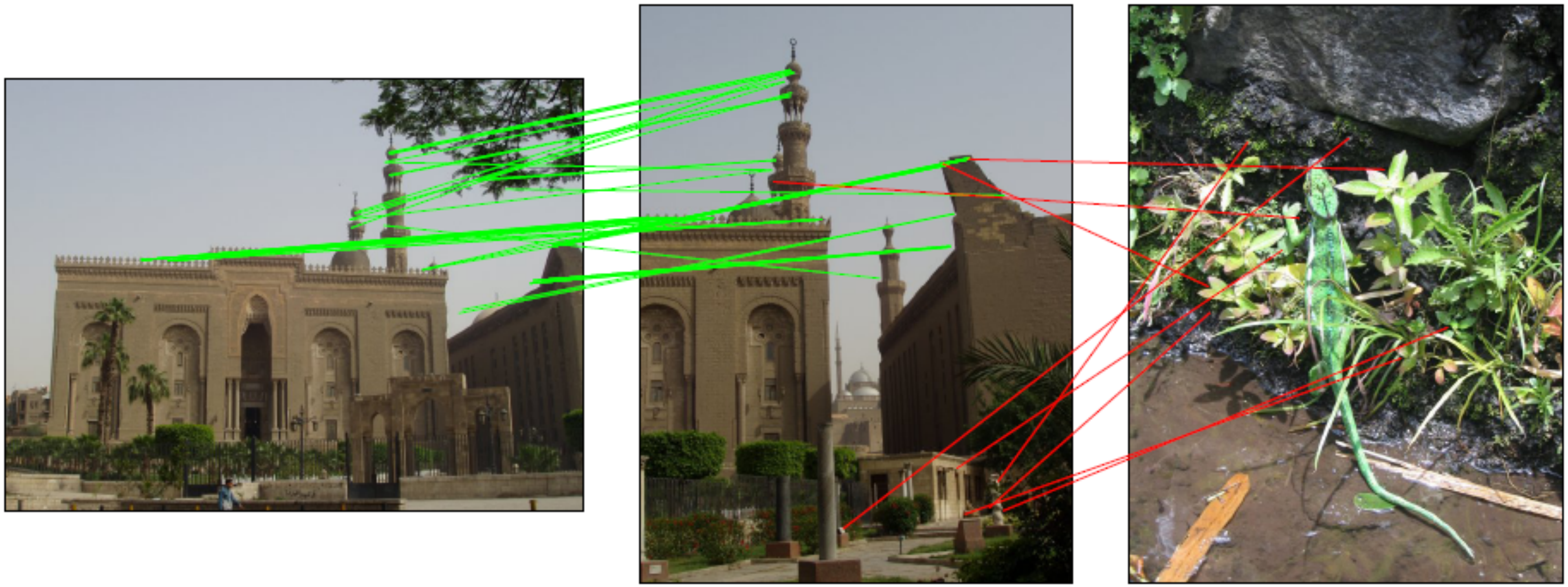


Still many matches with the non-corresponding one

Matching points - 20k word vocabulary + HE

83 matches

8 matches



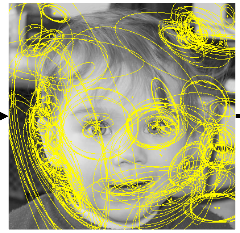
10x more matches with the corresponding image!

Bag-of-features [Sivic&Zisserman'03]

Query image

Harris-Hessian-Laplace regions + SIFT descriptors

Set of SIFT descriptors



centroids (visual words)

Bag-of-features processing +*tf-idf* weighting

sparse frequency vector

Inverted file

querying

ranked image short-list

Geometric verification

Re-ranked list

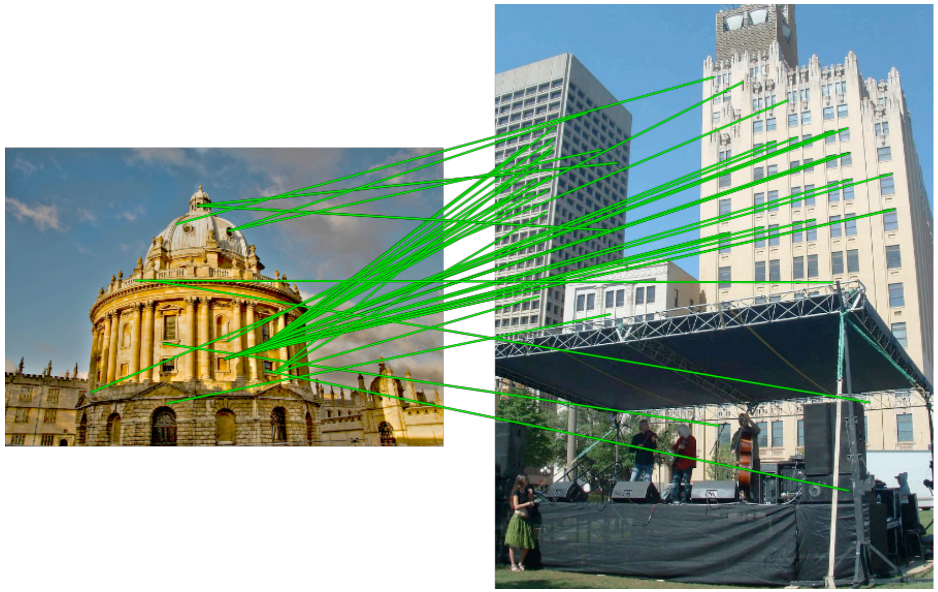
[Chum & al. 2007]

- "visual words":
  - 1 "word" (index) per local descriptor
  - only images ids in inverted file

  => 8 GB fits!

Geometric verification

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



Both images have many matches – which is correct?

Geometric verification

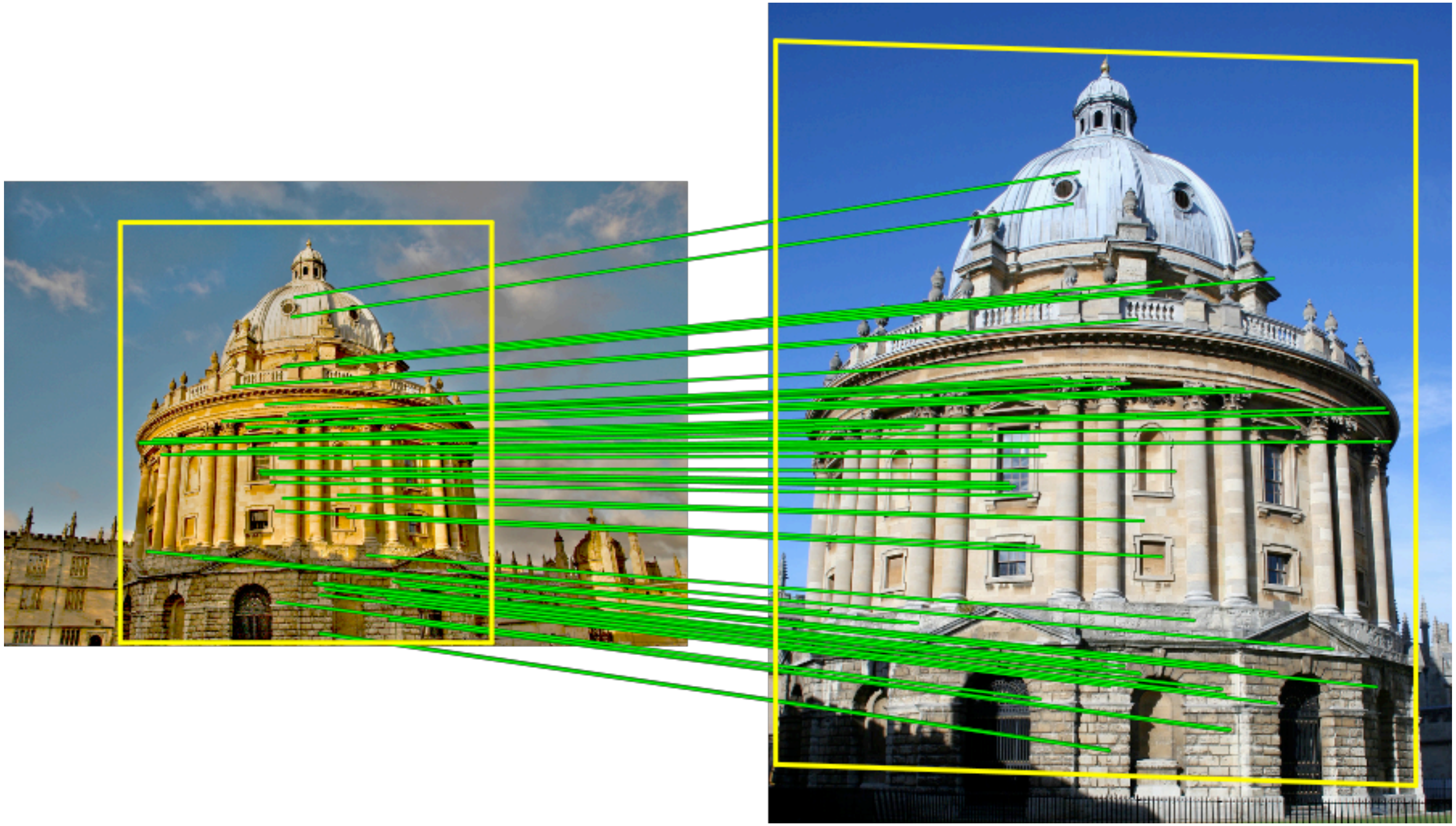
We can measure **spatial consistency** between the query and each result to improve retrieval quality

Many spatially consistent matches – **correct result**

Few spatially consistent matches – **incorrect result**

Geometric verification

Gives **localization** of the object



Geometric verification

- Remove outliers, matches contain a high number of incorrect ones

- Estimate geometric transformation

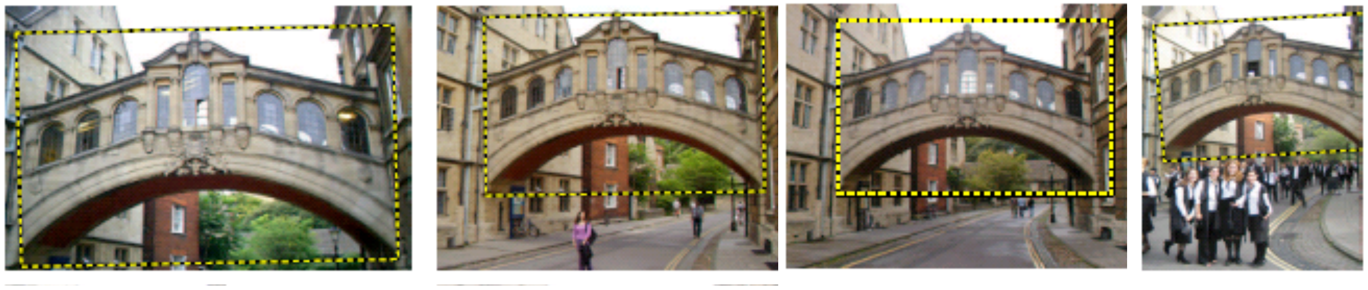
- Robust strategies
  - RANSAC
  - Hough transform

Geometric verification – example

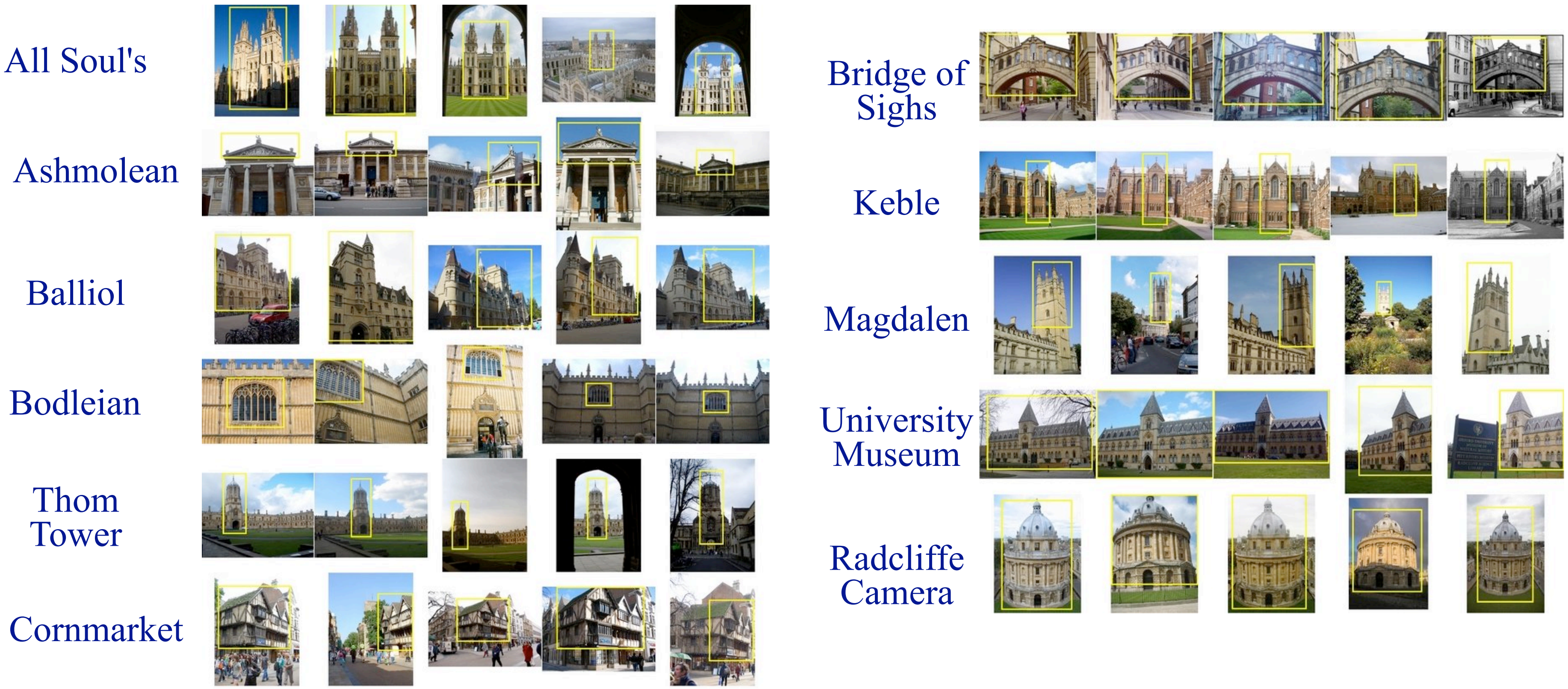
1. Query

2. Initial retrieval set (bag of words model)

3. Spatial verification (re-rank on # of inliers)



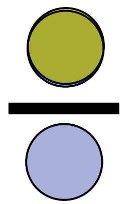
Evaluation dataset: Oxford buildings



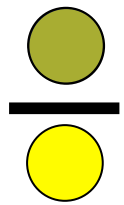
- Ground truth obtained for 11 landmarks
- Evaluate performance by mean Average Precision

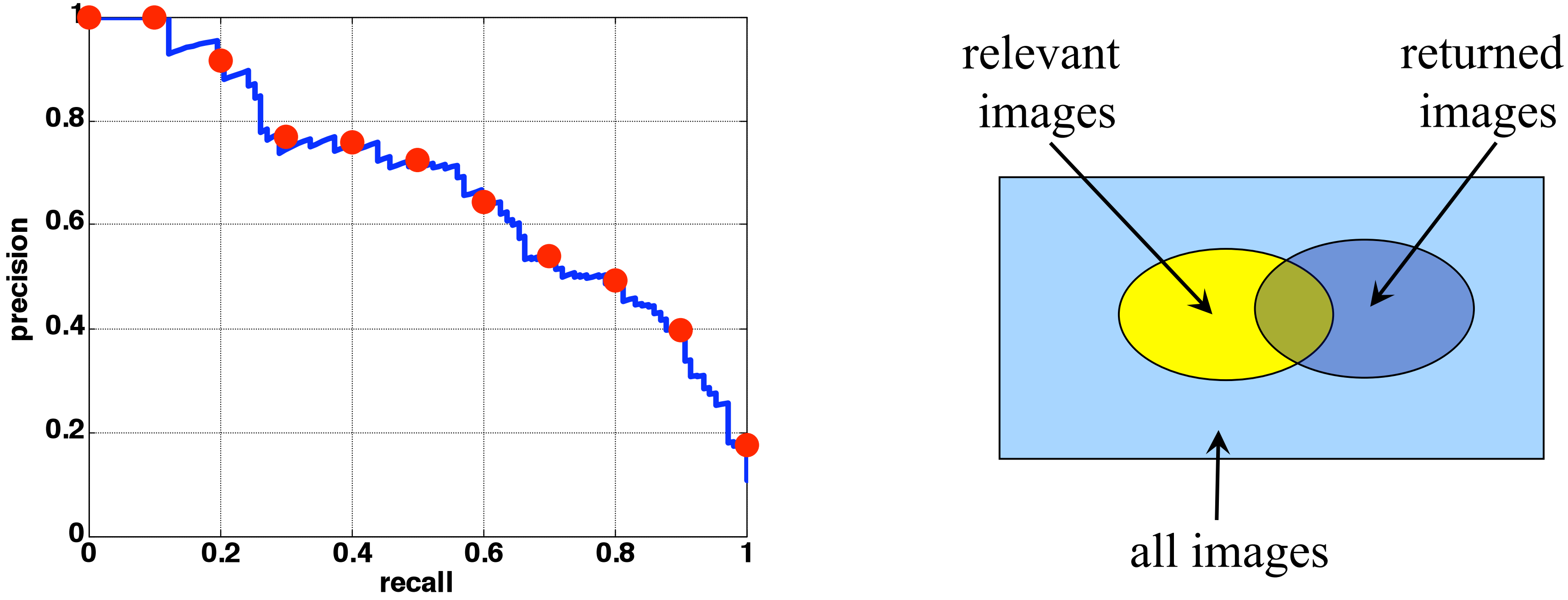
Measuring retrieval performance: Precision - Recall

- Precision: % of returned images that are relevant

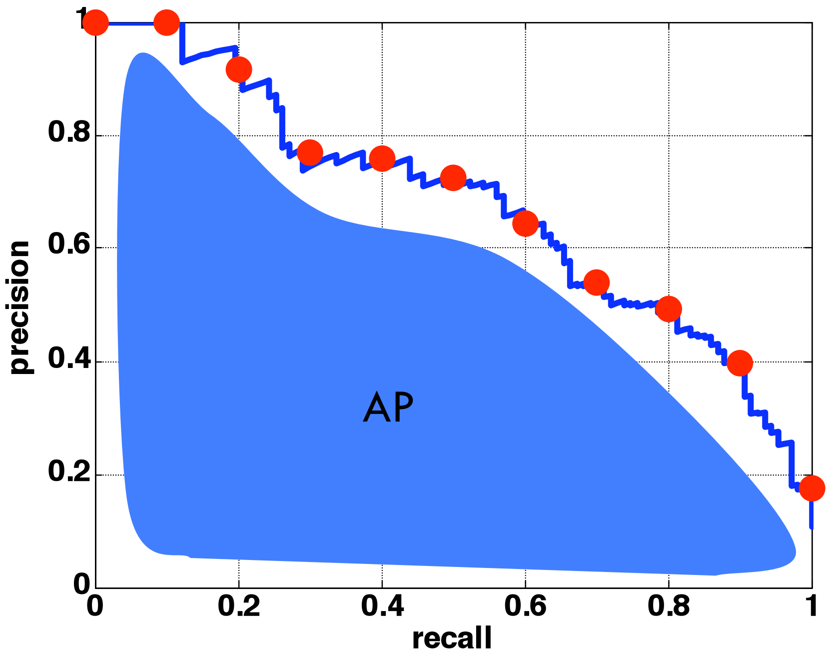


- Recall: % of relevant images that are returned



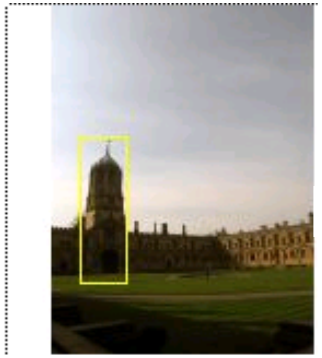


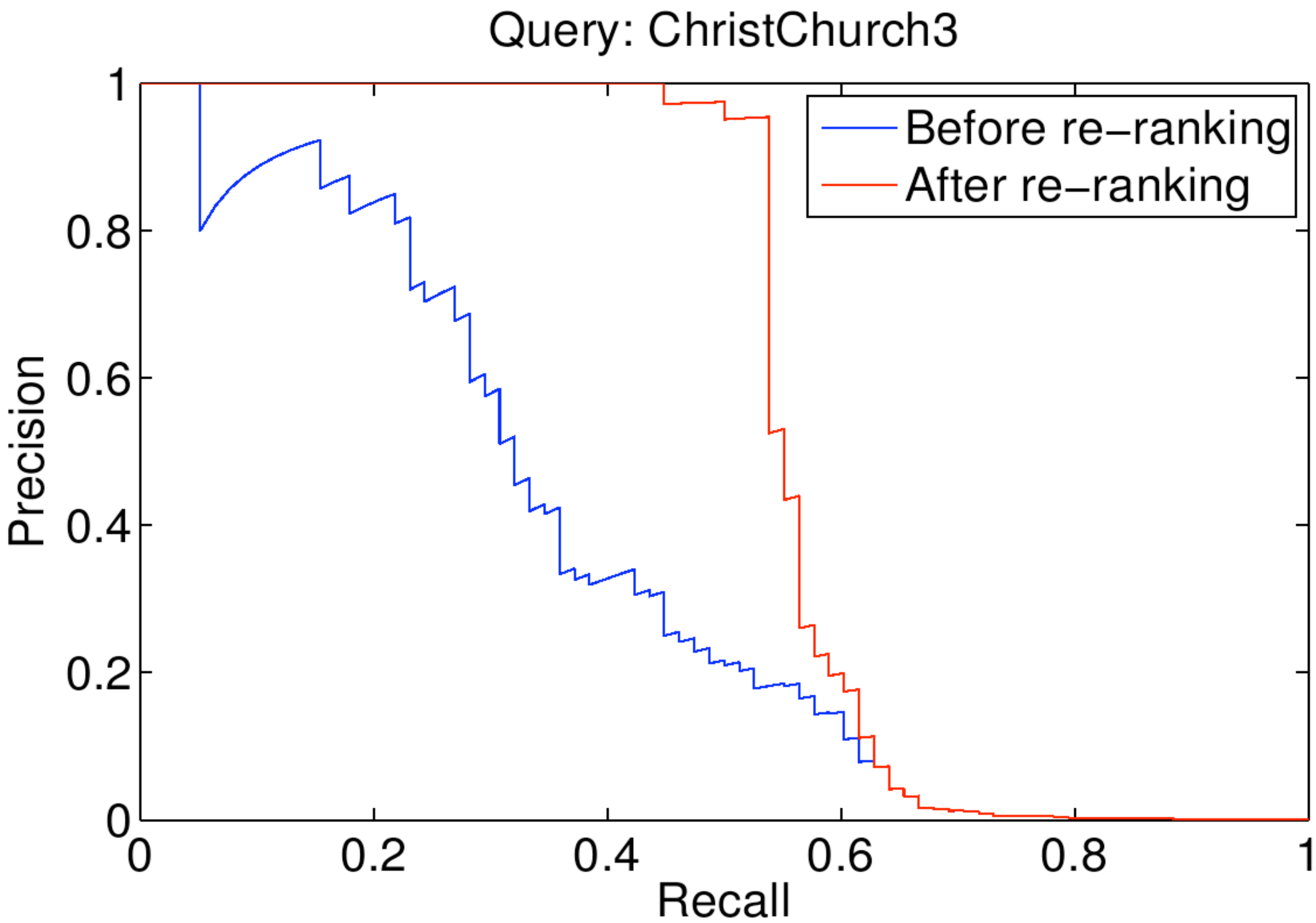
Average Precision



- A good AP score requires both high recall and high precision
- Application-independent

Performance measured by mean Average Precision (mAP) over 55 queries on 100K or 1.1M image datasets



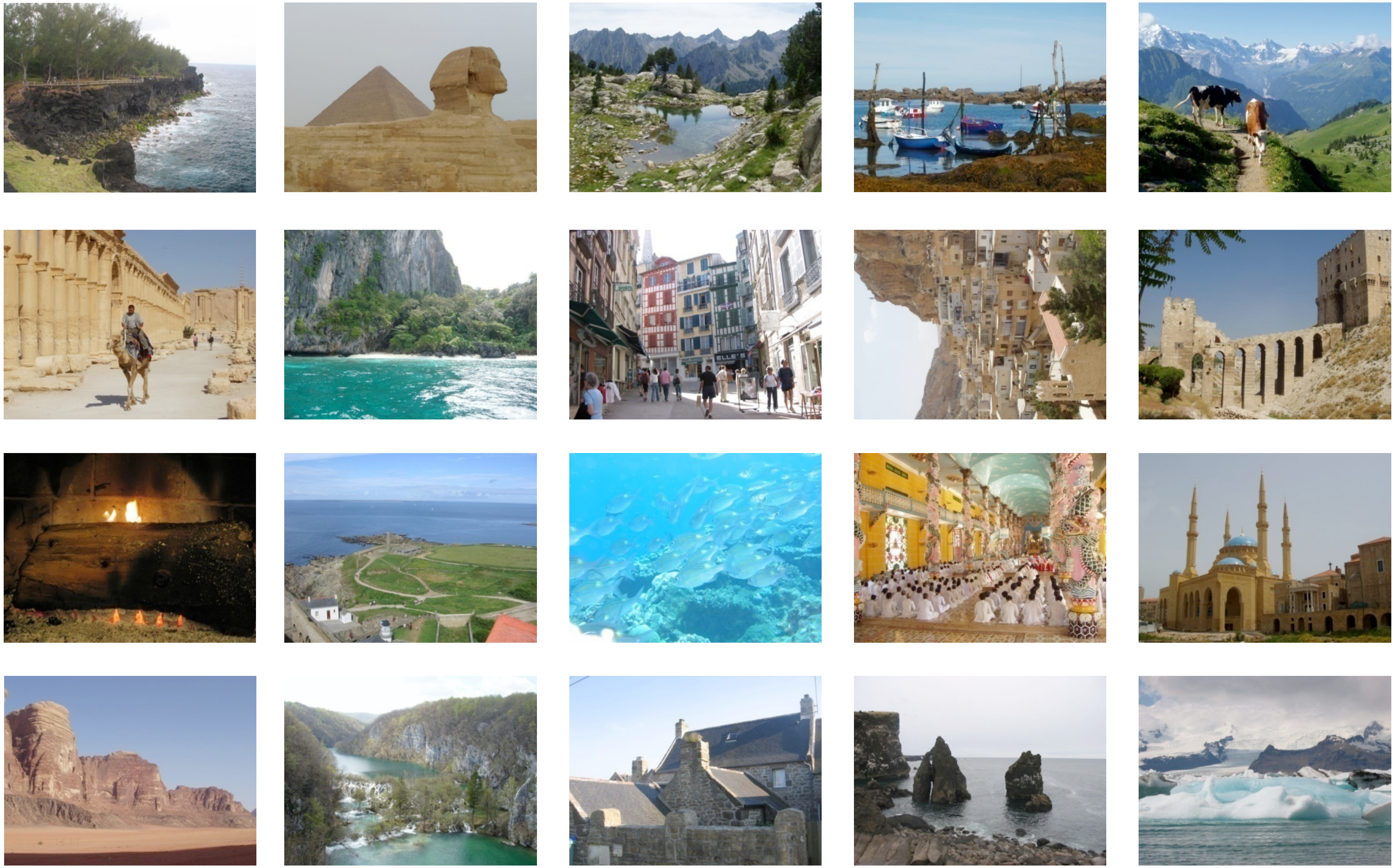

Query: ChristChurch3
Before re−ranking
After re−ranking
Precision
Recall
0
0.2
0.4
0.6
0.8
1
0
0.2
0.4
0.6
0.8
1

INRIA holidays dataset

- Evaluation for the INRIA holidays dataset, 1491 images
  - 500 query images + 991 annotated true positives
  - Most images are holiday photos of friends and family
- 1 million & 10 million distractor images from Flickr
- Vocabulary construction on a different Flickr set
- Almost real-time search speed

- Evaluation metric: mean average precision (in [0,1], bigger = better)
  - Average over precision/recall curve

Holiday dataset – example queries



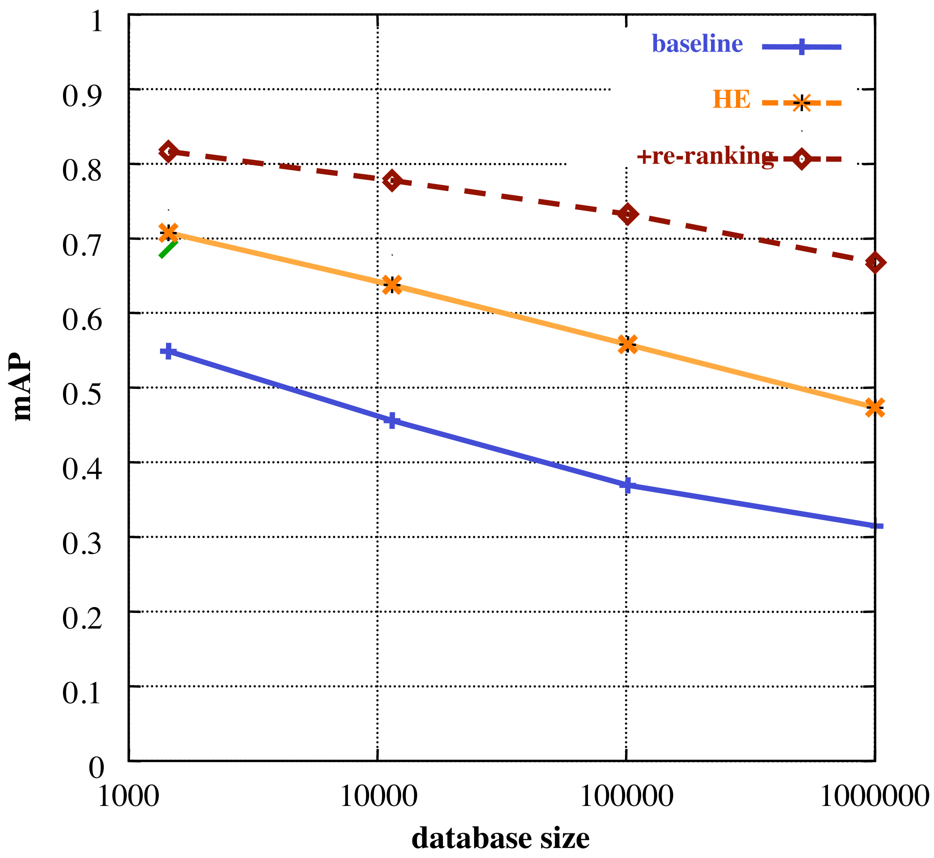
Dataset : Venice Channel

Dataset : San Marco square

Example distractors - Flickr

Experimental evaluation

- Evaluation on our holidays dataset, 500 query images, 1 million distracter images
- Metric: mean average precision (in [0,1], bigger = better)



Results – Venice Channel

Demo at http://bigimbaz.inrialpes.fr

Towards larger databases?

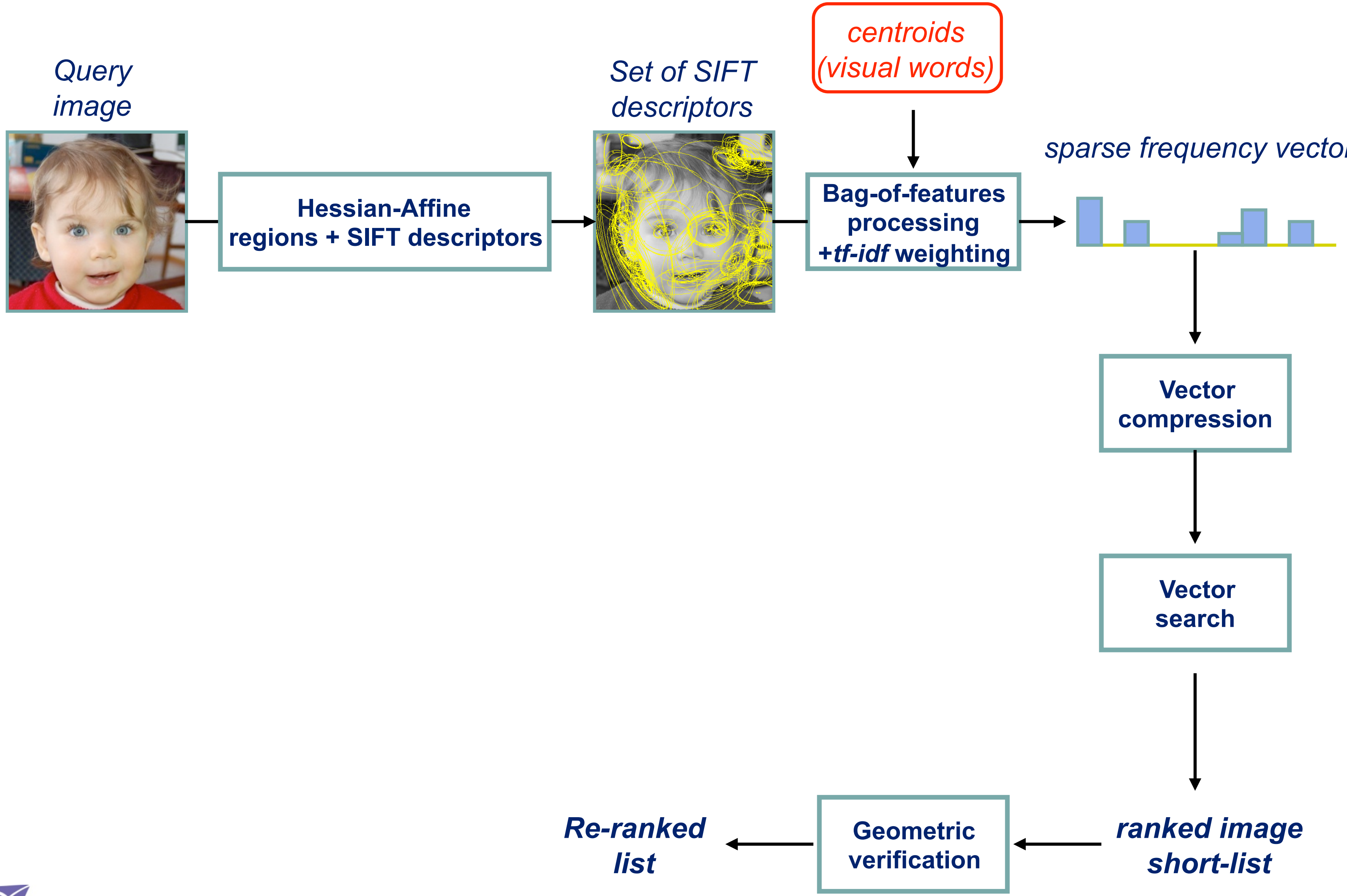
- BOF can handle up to ~10 M d'images
  - with a limited number of descriptors per image
  - 40 GB of RAM
  - search = 2 s

- Web-scale = billions of images
  - With 100 M per machine

    → search = 20 s, RAM = 400 GB

    → not tractable!

Recent approaches for very large scale indexing



INRIA

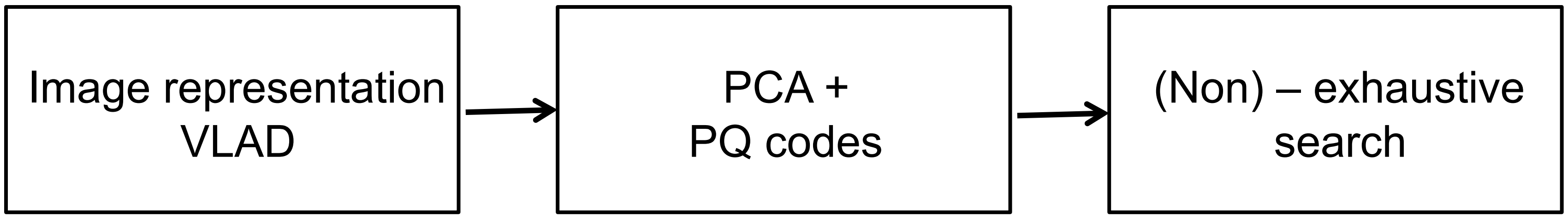
Related work on very large scale image search

- Min-hash and geometrical min-hash [Chum et al. ‘07-’09]
- GIST descriptors with Spectral Hashing [Torralba et al. ‘08]
- Compressing the BoF representation (miniBof) [Jegou et al. ‘09]
- Aggregating local desc into a compact image representation [Jegou et al. ‘10]
- Efficient object category recognition using classemes [Torresani et al.’10]

Compact image representation

- Aim: improving the tradeoff between
  - search speed
  - memory usage
  - search quality

- Approach: joint optimization of three stages
  - local descriptor aggregation
  - dimension reduction
  - indexing algorithm



[H. Jegou et al., Aggregating local desc into a compact image representation, CVPR'10]

Aggregation of local descriptors

- Problem: represent an image by a single fixed-size vector:

set of n local descriptors → 1 vector

- Most popular idea: BoF representation [Sivic & Zisserman 03]
  - sparse vector
  - highly dimensional

→ high dimensionality reduction introduces loss

- Alternative: Fisher Kernels [Perronnin et al 07]
  - non sparse vector
  - excellent results with a small vector dimensionality

  → our method (VLAD) in the spirit of this representation

VLAD : vector of locally aggregated descriptors

- Simplification of Fisher kernels

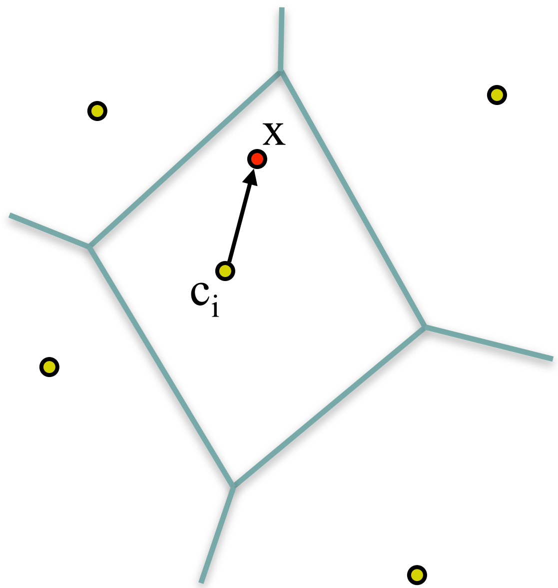
- Learning: a vector quantifier (*k*-means)
  - output: *k* centroids (visual words): $c_1,\ldots,c_i,\ldots c_k$
  - centroid c_i has dimension *d*

- For a given image
  - assign each descriptor to closest center c_i
  - accumulate (sum) descriptors per cell

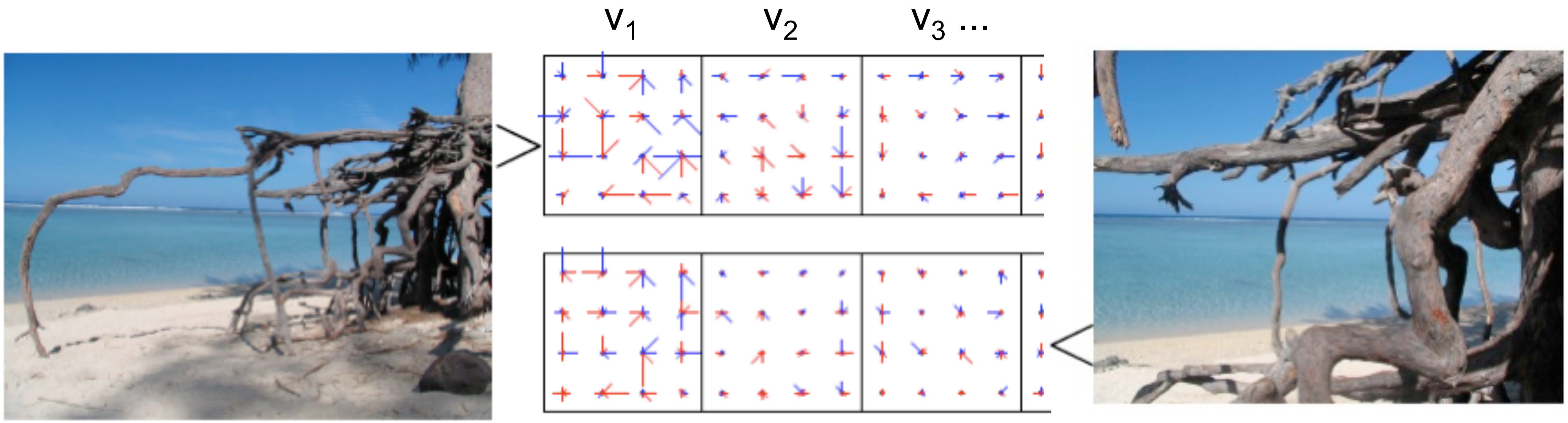
    $v_i := v_i + (x - c_i)$

- VLAD (dimension $D = k \times d$)

- The vector is L2-normalized



VLADs for corresponding images



SIFT-like representation per centroid (+ components: blue, - components: red)

- good coincidence of energy & orientations

VLAD performance and dimensionality reduction

- We compare VLAD descriptors with BoF: INRIA Holidays Dataset (mAP,%)
- Dimension is reduced to from D to D' dimensions with PCA

Aggregator	**k**	**D**	**D'=D (no reduction)**	**D'=128**	**D'=64**
BoF	1,000	1,000	41.4	44.4	43.4
BoF	20,000	20,000	44.6	45.2	44.5
BoF	200,000	200,000	54.9	43.2	41.6
VLAD	16	2,048	49.6	49.5	**49.4**
VLAD	64	8,192	52.6	**51.0**	47.7
VLAD	256	32,768	**57.5**	50.8	47.6

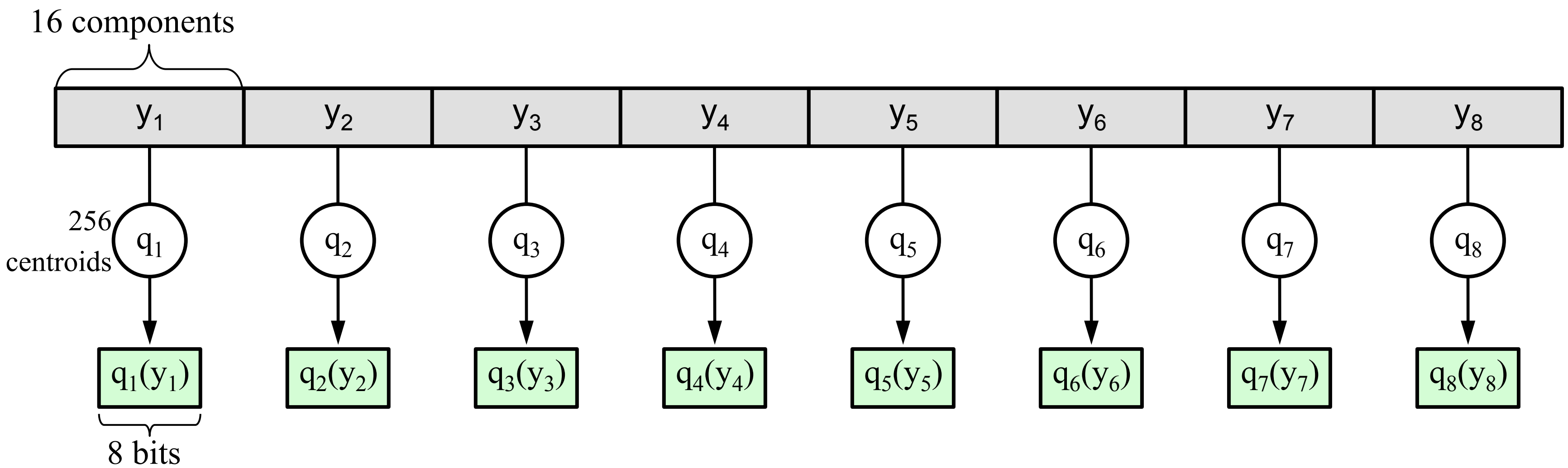
- Observations:
  - VLAD better than BoF for a given descriptor size
    → comparable to Fisher kernels for these operating points
  - Choose a small D if output dimension D' is small

Product quantization for nearest neighbor search

- Vector split into *m* subvectors: $y \rightarrow [y_1 | \ldots | y_m]$

- Subvectors are quantized separately by quantizers $q(y) = [q_1(y_1) | \ldots | q_m(y_m)]$
  where each q_i is learned by *k*-means with a limited number of centroids

- Example: y = 128-dim vector split in 8 subvectors of dimension 16
  - each subvector is quantized with 256 centroids -> 8 bit
  - very large codebook 256^8 ~ 1.8x10^19



$\Rightarrow$ 8 subvectors x 8 bits = 64-bit quantization index

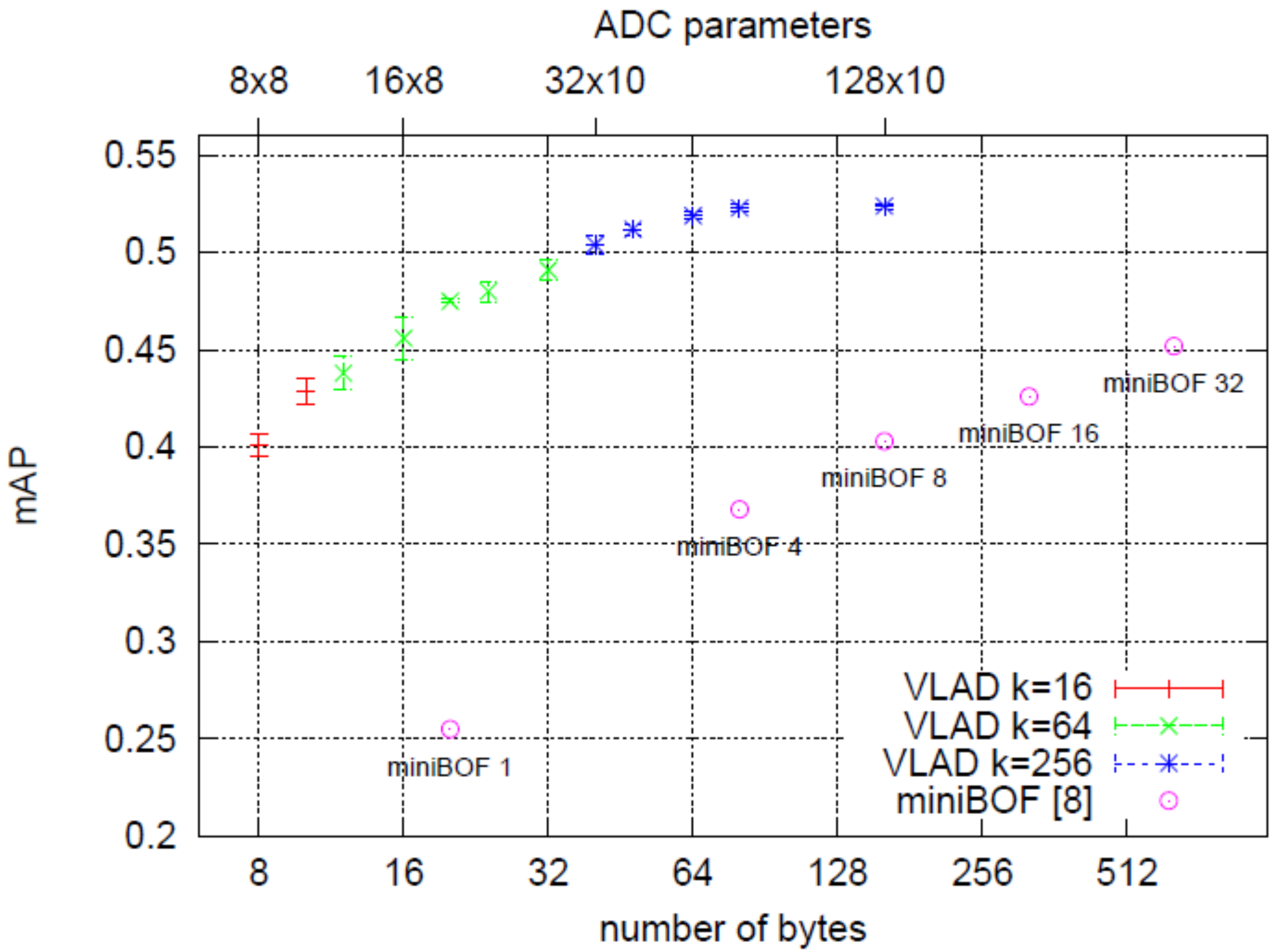
Joint optimization of VLAD and dimension reduction-indexing

- For VLAD
  - The larger k, the better the raw search performance
  - But large k produce large vectors, that are harder to index

- Optimization of the vocabulary size
  - Fixed output size (in bytes)
  - D' computed from k via the joint optimization of reduction/indexing
  - Only k has to be set

➔ end-to-end parameter optimization

Results on the Holidays dataset with various quantization parameters



Results on standard datasets

- Datasets
  - University of Kentucky benchmark — score: nb relevant images, max: 4
  - INRIA Holidays dataset — score: mAP (%)

Method	bytes	UKB	Holidays
BoF, k=20,000	10K	2.92	44.6
BoF, k=200,000	12K	3.06	54.9
miniBOF	20	2.07	25.5
miniBOF	160	2.72	40.3
VLAD k=16, ADC 16 x 8	**16**	**2.88**	**46.0**
VLAD k=64, ADC 32 x10	**40**	**3.10**	**49.5**

$D'=64$ for $k=16$ and $D'=96$ for $k=64$

ADC (subvectors) x (bits to encode each subvector)

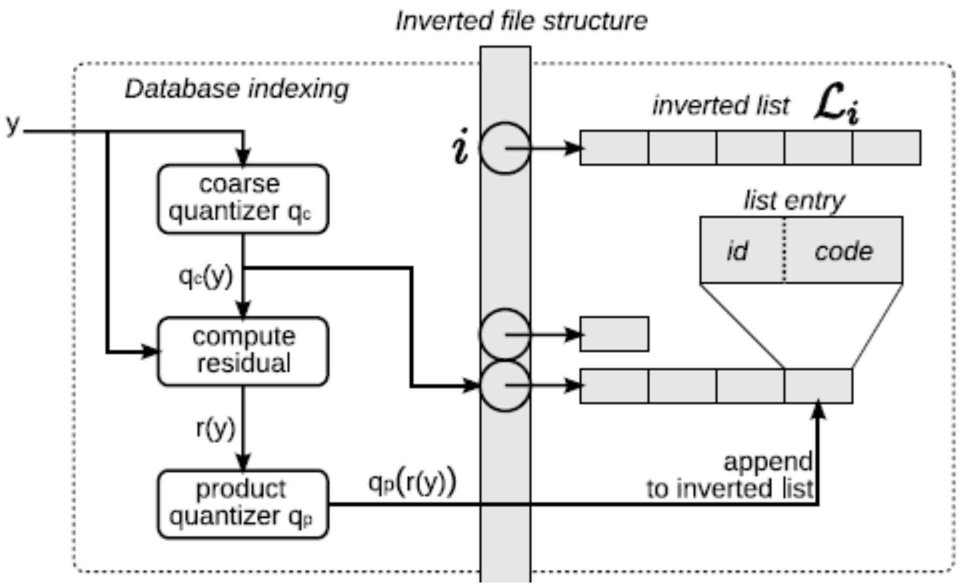
miniBOF: “Packing Bag-of-Features”, ICCV’09

Large scale experiments (10 million images)

- Exhaustive search of VLADs, D'=64
  - 4.77s

- With the product quantizer
  - Exhaustive search with ADC: 0.29s
  - Non-exhaustive search with IVFADC: 0.014s

IVFADC -- Combination with an inverted file



Large scale experiments (10 million images)