

lecture 10: image retrieval and manifold learning

deep learning for vision

Yannis Avrithis

Inria Rennes-Bretagne Atlantique

Rennes, Nov. 2017 – Jan. 2018



outline

background

indexing

pooling

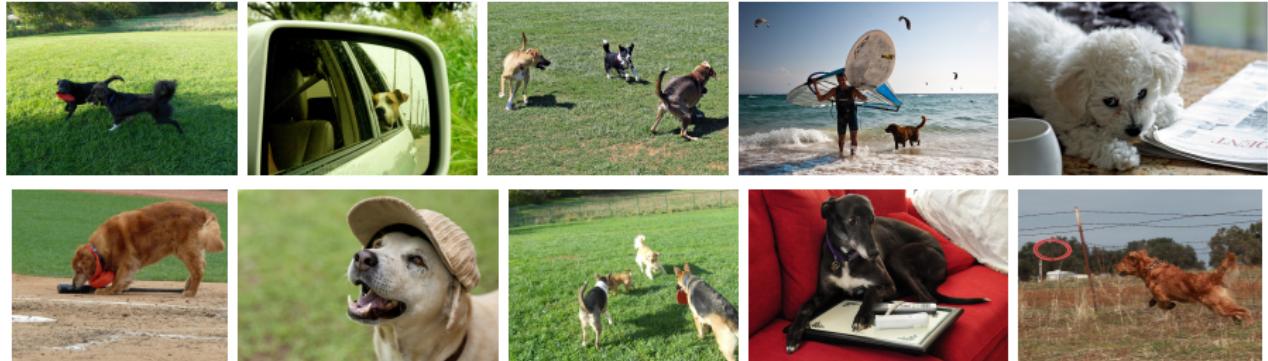
manifold learning

fine-tuning

graph-based methods

background

image classification challenges



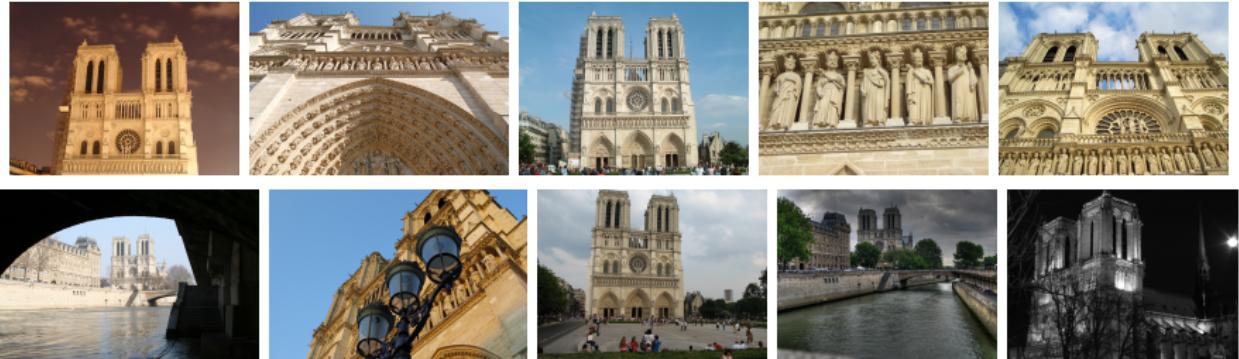
- scale
- viewpoint
- occlusion
- clutter
- lighting
- number of instances
- texture/color
- pose
- deformability
- intra-class variability

image classification challenges



- scale
- viewpoint
- occlusion
- clutter
- lighting
- number of instances
- texture/color
- pose
- deformability
- intra-class variability

image retrieval challenges



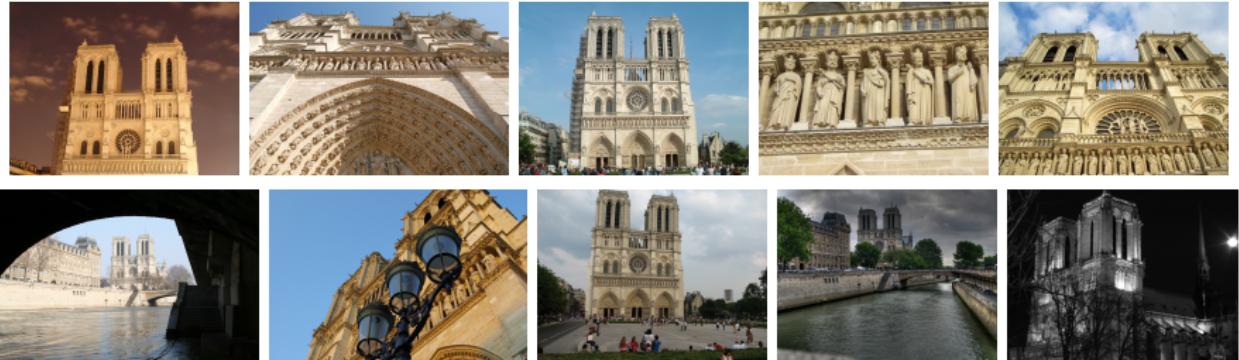
- scale
- viewpoint
- occlusion
- clutter
- lighting

- distinctiveness
- distractors

main difference to classification:

- no intra-class variability

image retrieval challenges



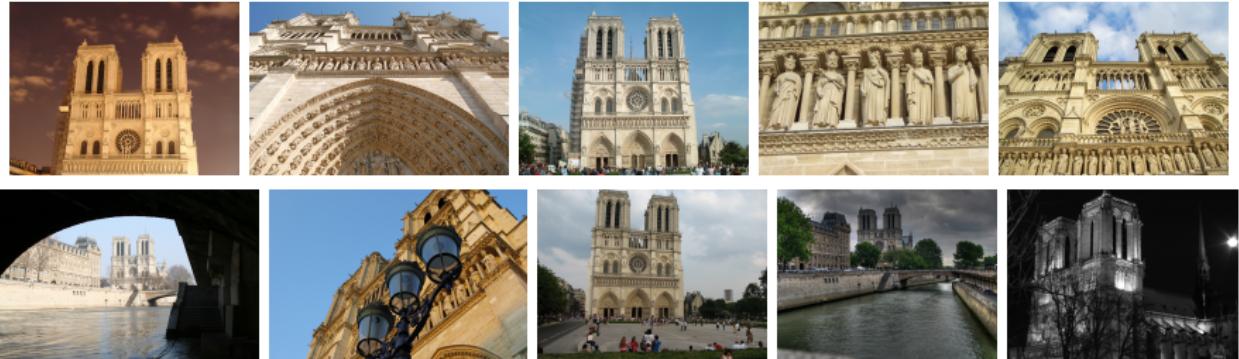
- scale
- viewpoint
- occlusion
- clutter
- lighting

- distinctiveness
- distractors

main difference to classification:

- no intra-class variability

image retrieval challenges



- scale
- viewpoint
- occlusion
- clutter
- lighting

- distinctiveness
- distractors

main difference to classification:

- no intra-class variability

vector quantization → visual words



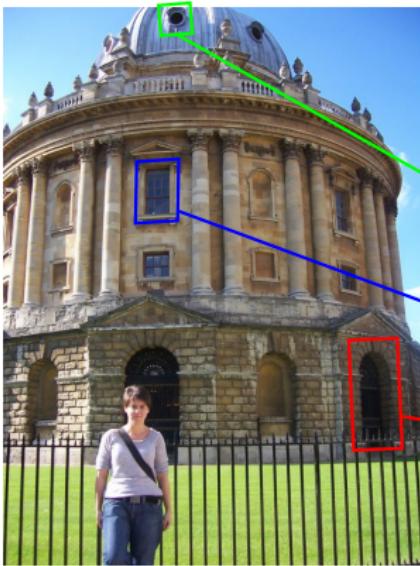
query



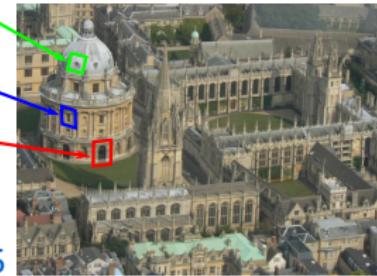
15

- query vs. dataset image

vector quantization → visual words



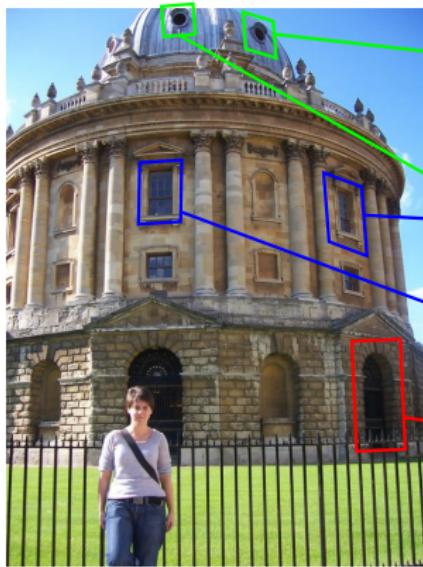
query



15

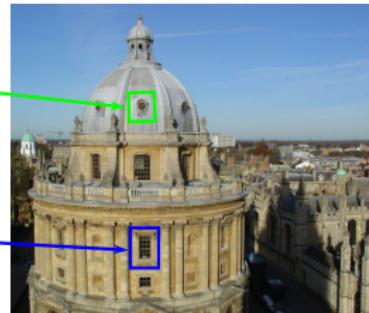
- pairwise descriptor matching

vector quantization → visual words

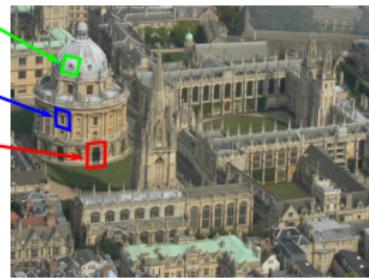


query

19

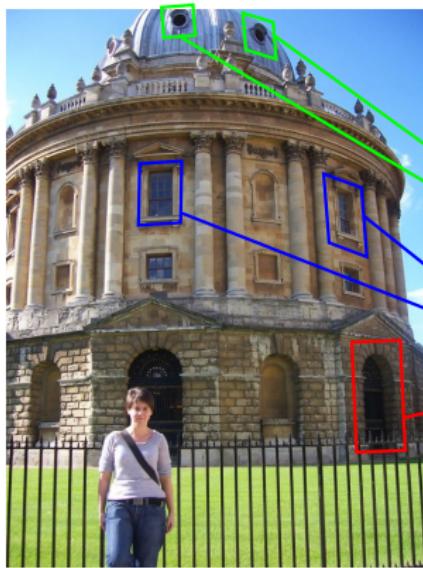


15

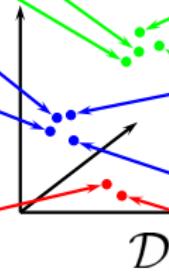


- pairwise descriptor matching for **every** dataset image

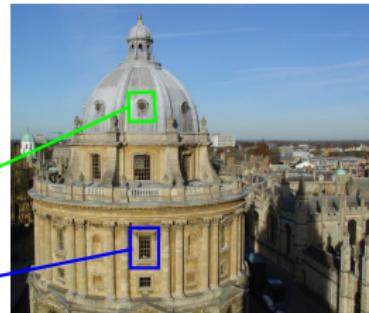
vector quantization → visual words



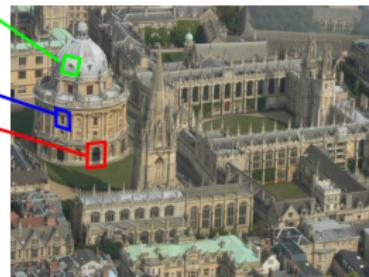
query



19

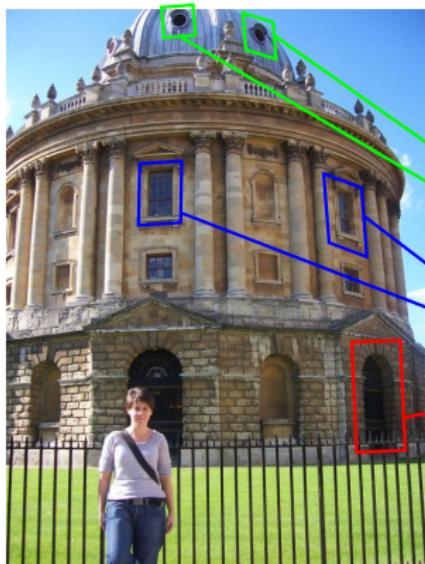


15

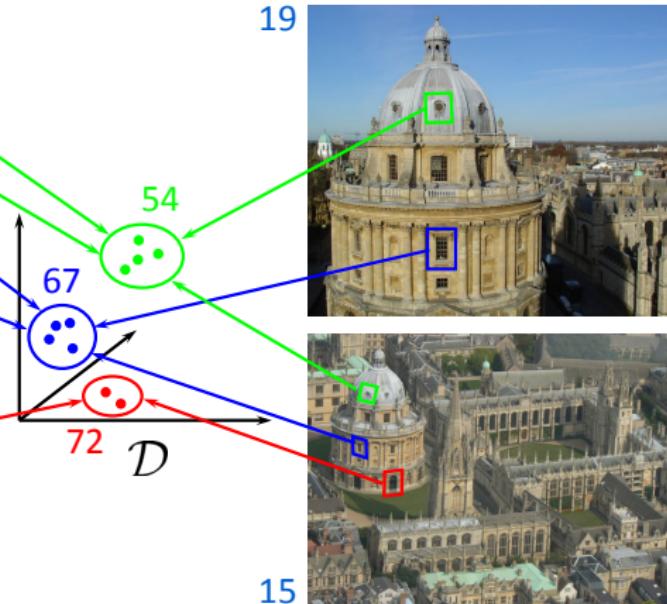


- similar descriptors should all be nearby in the descriptor space

vector quantization → visual words

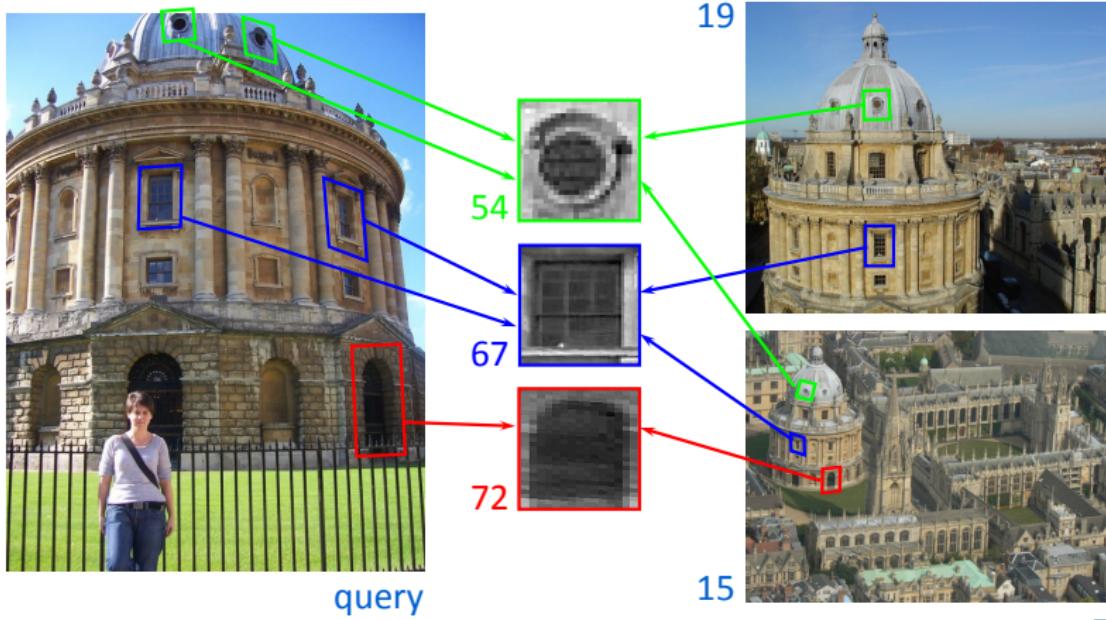


query



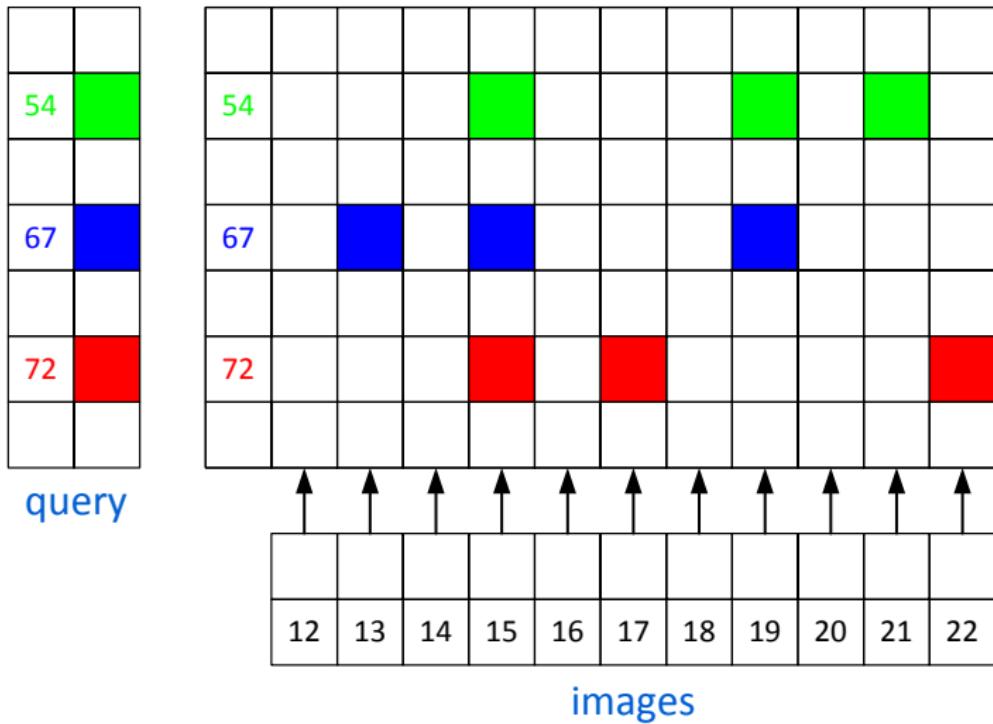
- let's quantize them into visual words

vector quantization → visual words

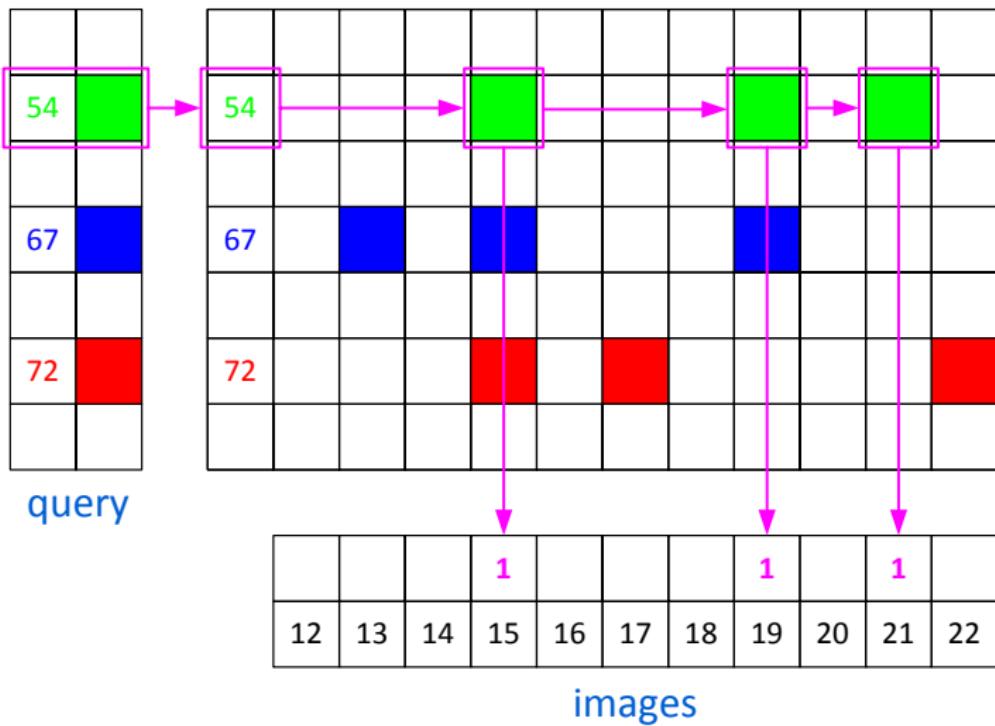


- now visual words act as a proxy; no pairwise matching needed

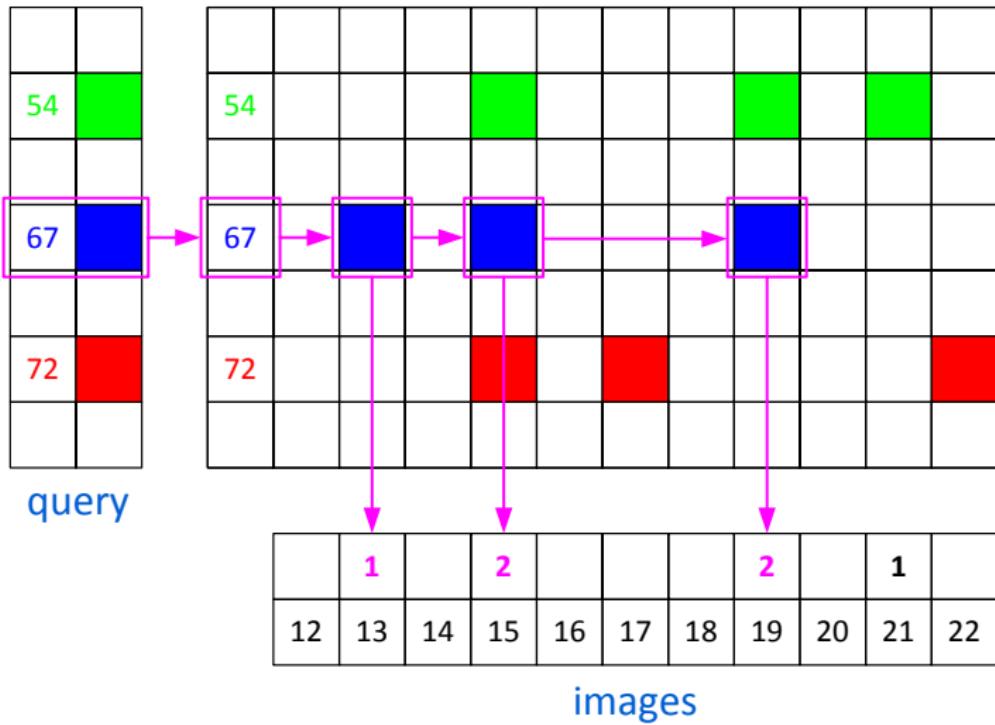
inverted file indexing



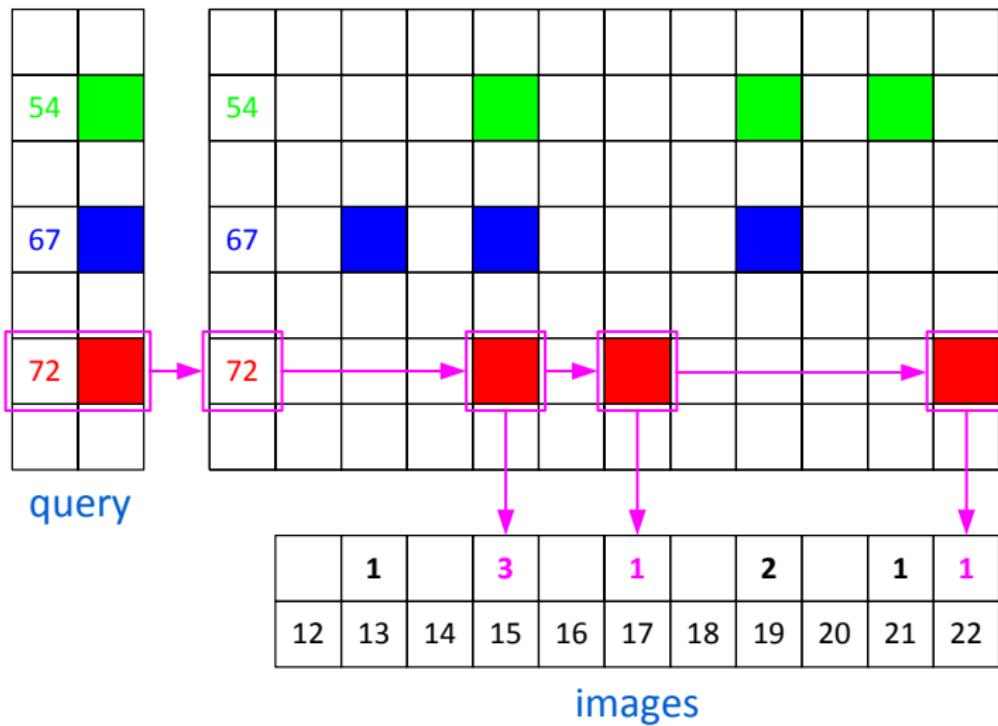
inverted file indexing



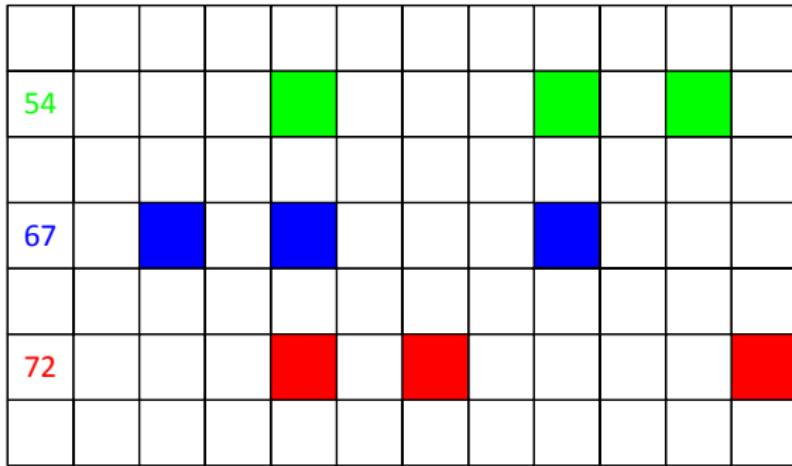
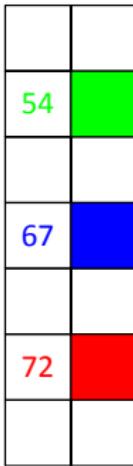
inverted file indexing



inverted file indexing

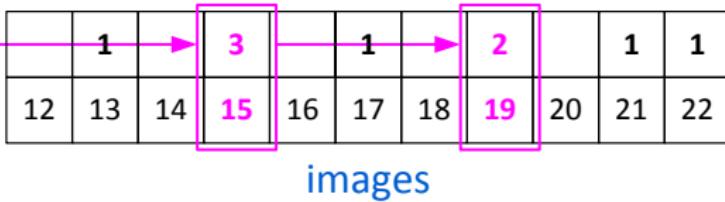


inverted file indexing



query

ranked shortlist



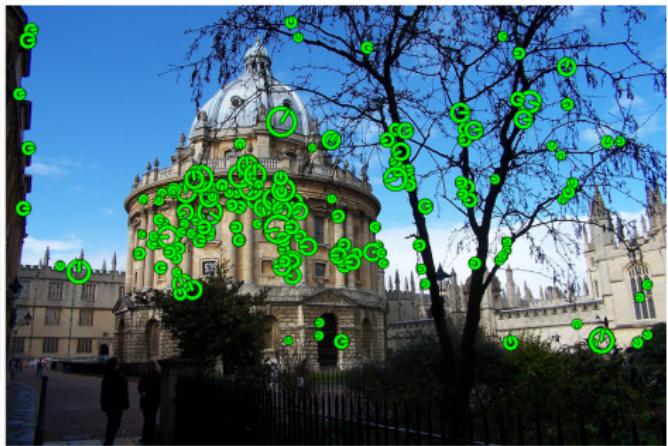
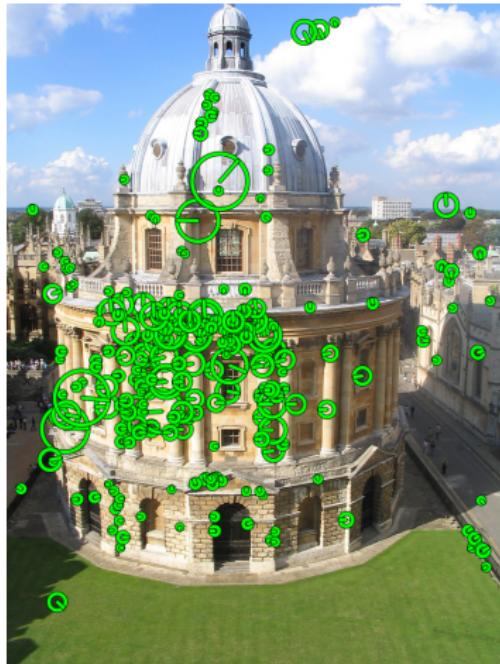
back to geometry: re-ranking



original images

Fischler and Bolles. CACM 1981. Random Sample Consensus: A Paradigm for Model Fitting With Applications to Image Analysis and Automated Cartography.

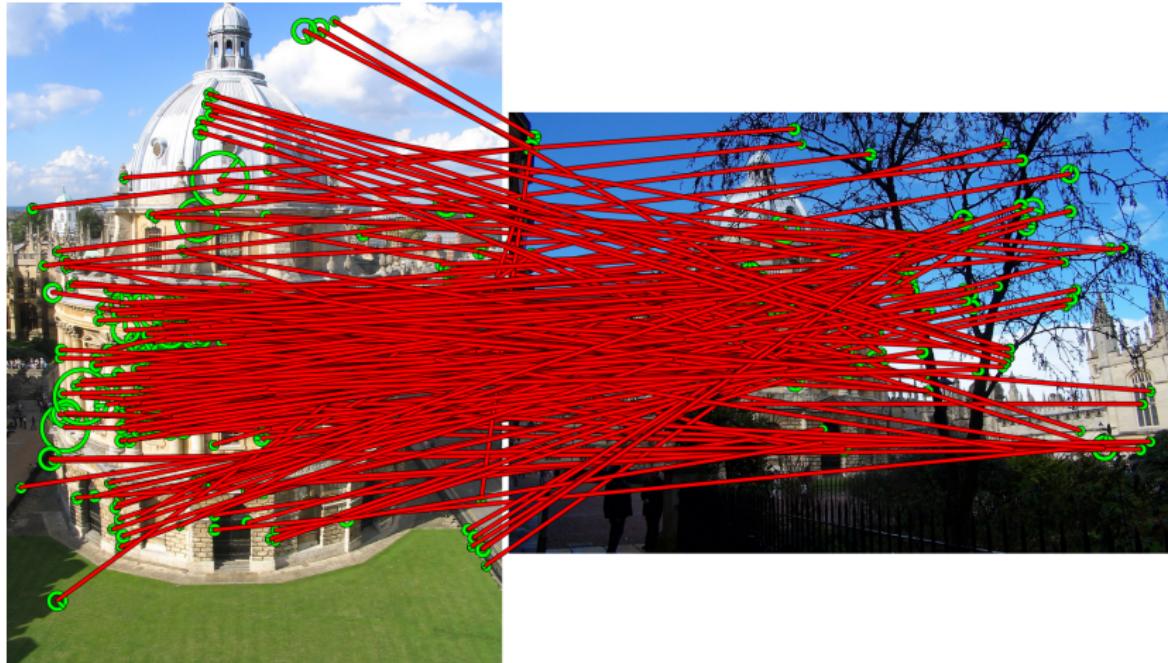
back to geometry: re-ranking



local features

Fischler and Bolles. CACM 1981. Random Sample Consensus: A Paradigm for Model Fitting With Applications to Image Analysis and Automated Cartography.

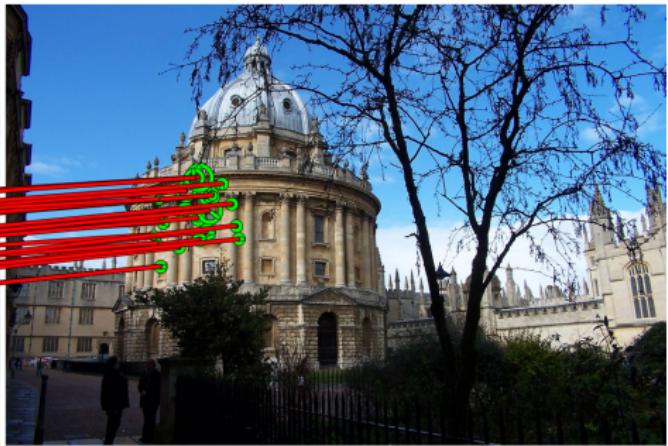
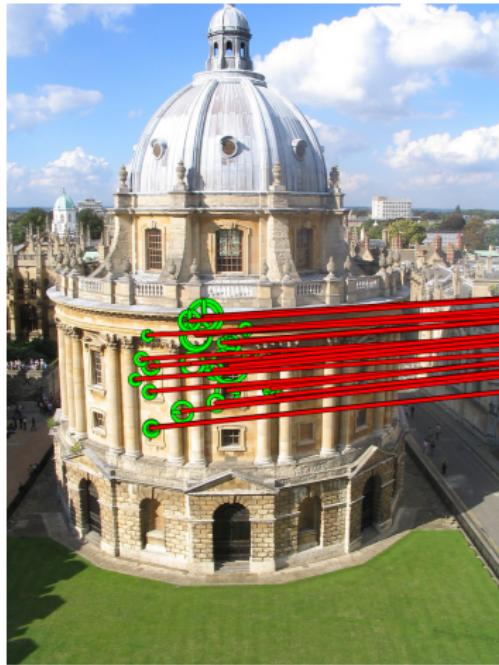
back to geometry: re-ranking



tentative correspondences: too many

Fischler and Bolles. CACM 1981. Random Sample Consensus: A Paradigm for Model Fitting With Applications to Image Analysis and Automated Cartography.

back to geometry: re-ranking

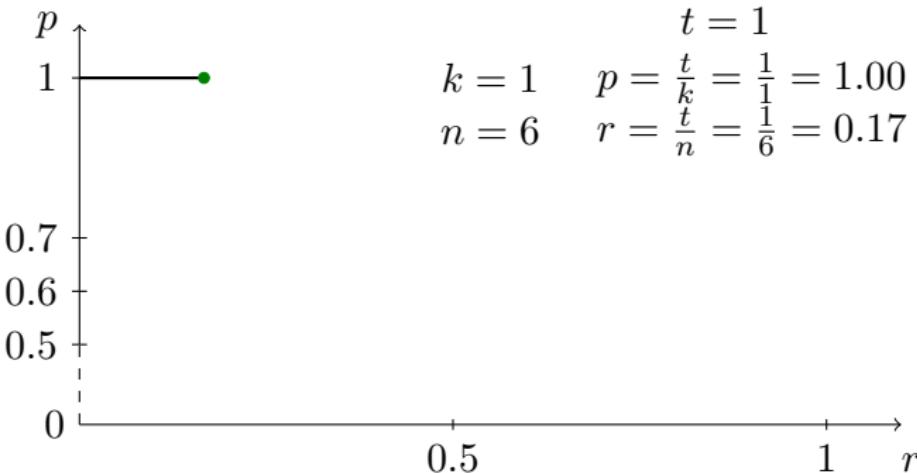


inliers: now more expensive to find

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

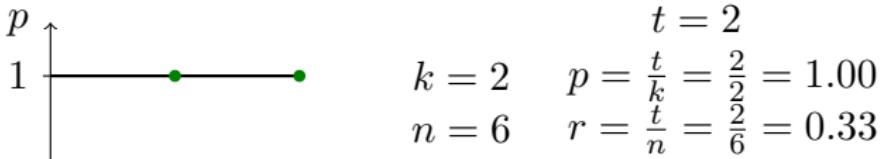


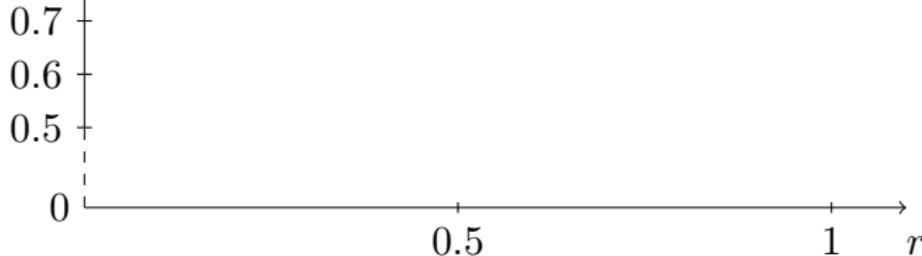
- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

$$p \uparrow$$

$$t = 2$$
$$k = 2 \quad p = \frac{t}{k} = \frac{2}{2} = 1.00$$
$$n = 6 \quad r = \frac{t}{n} = \frac{2}{6} = 0.33$$

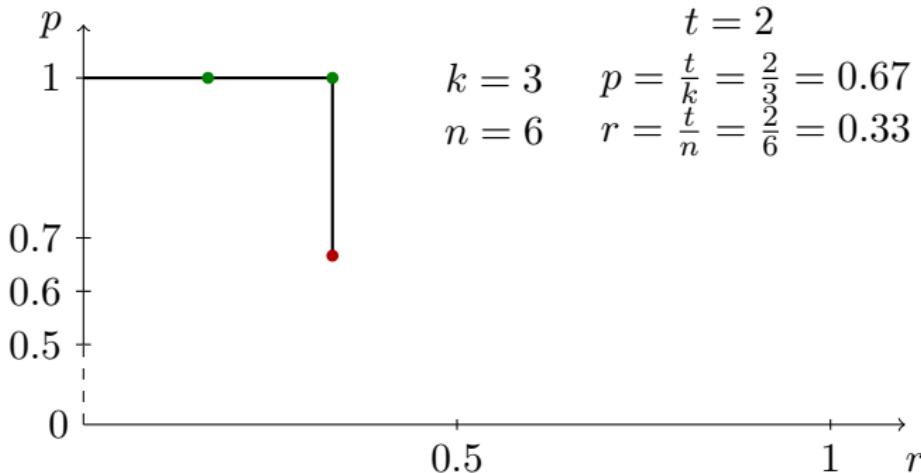


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

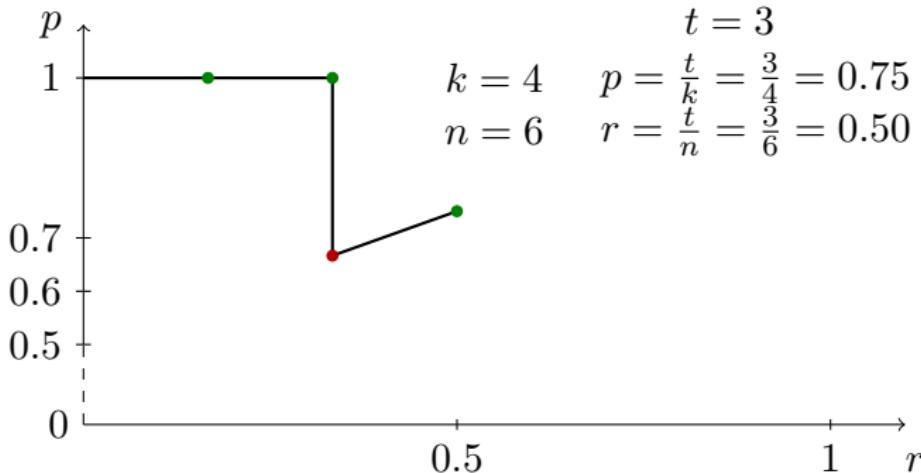


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

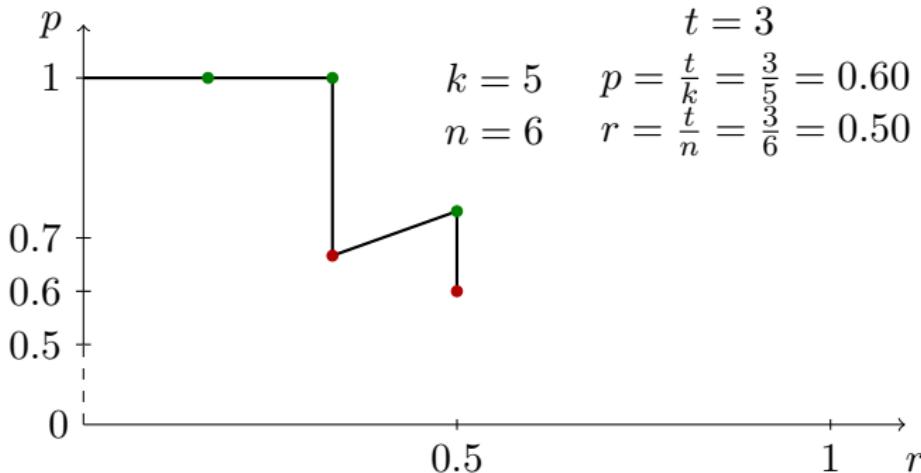


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

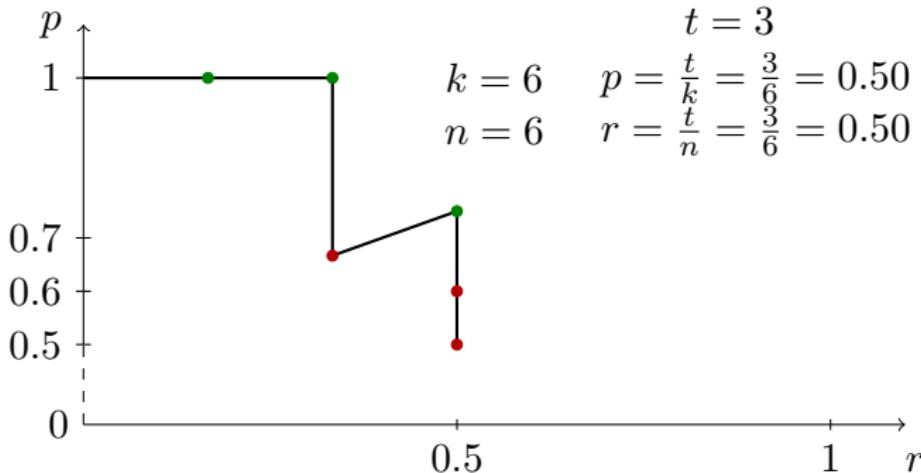


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

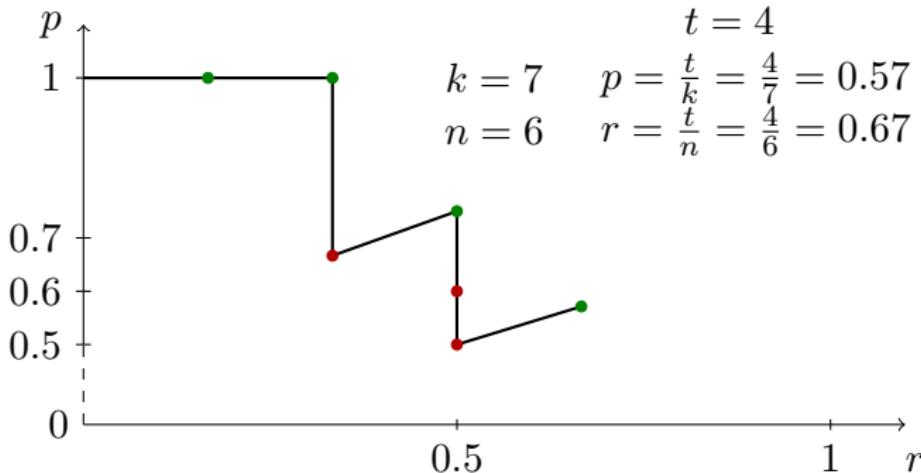


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

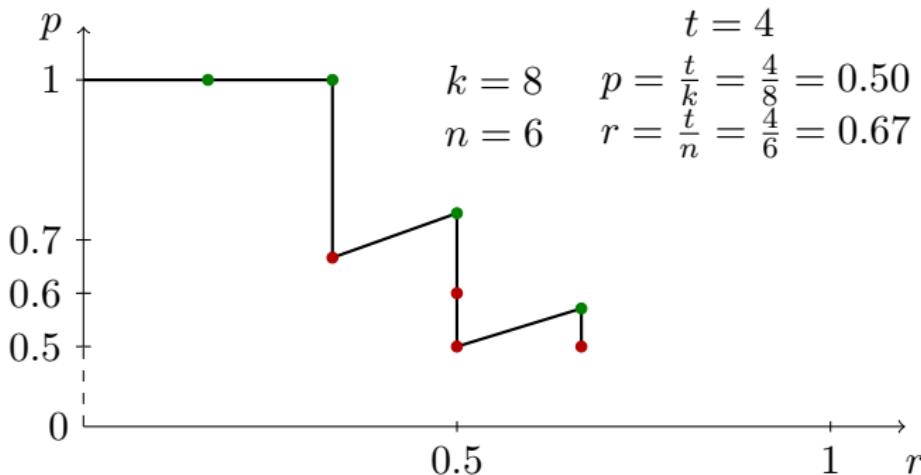


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

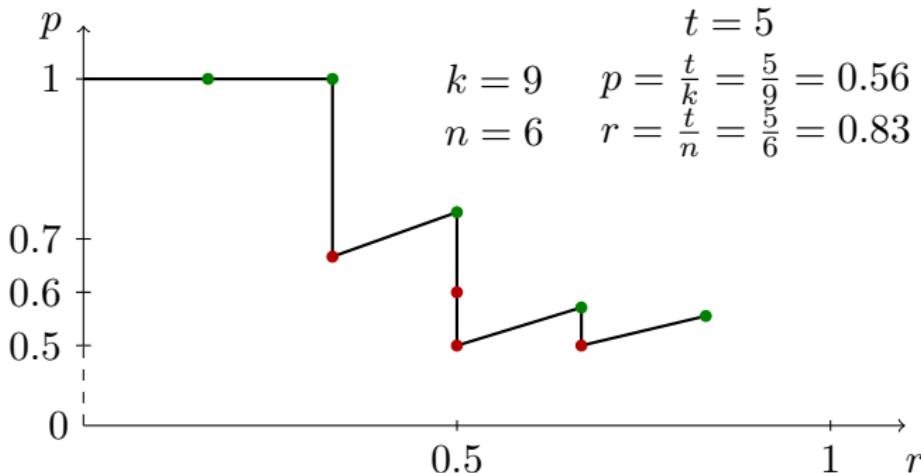


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

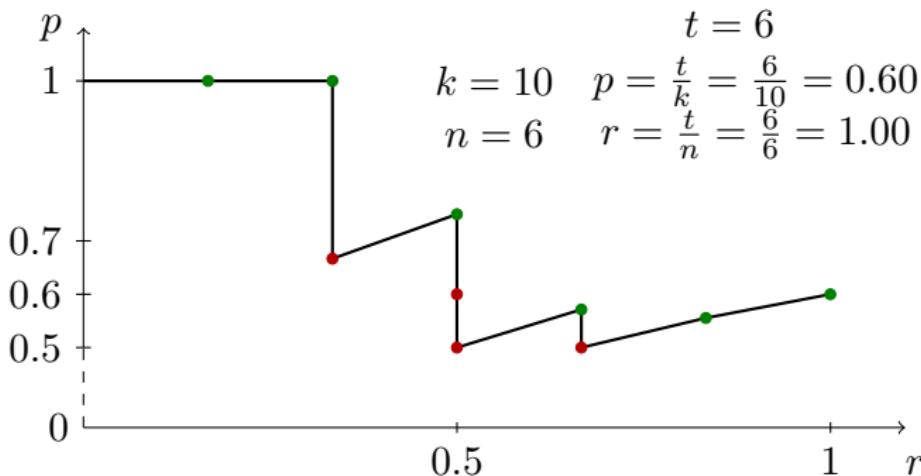


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

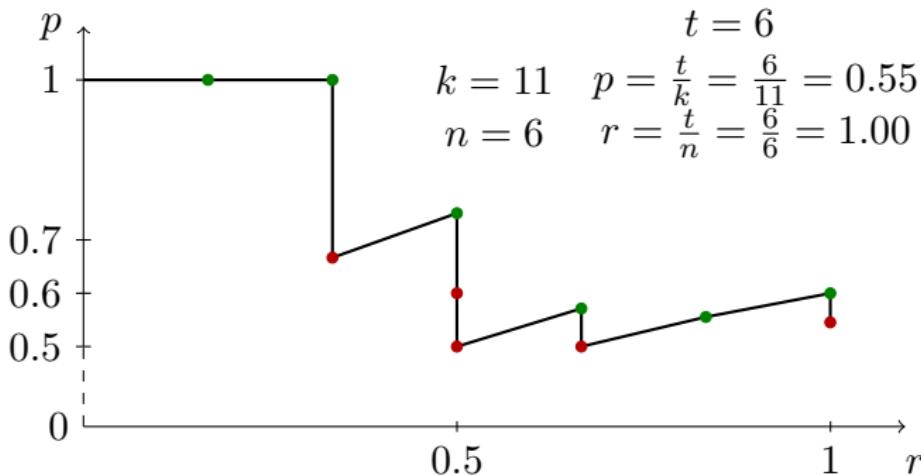


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

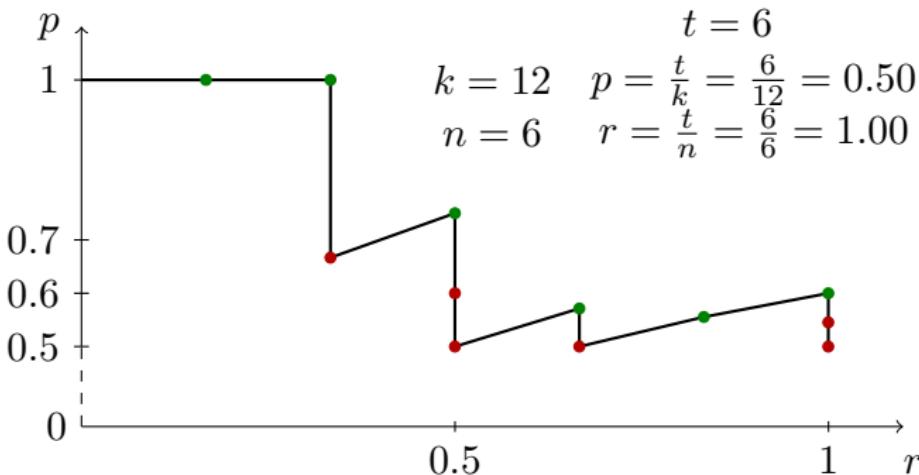


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

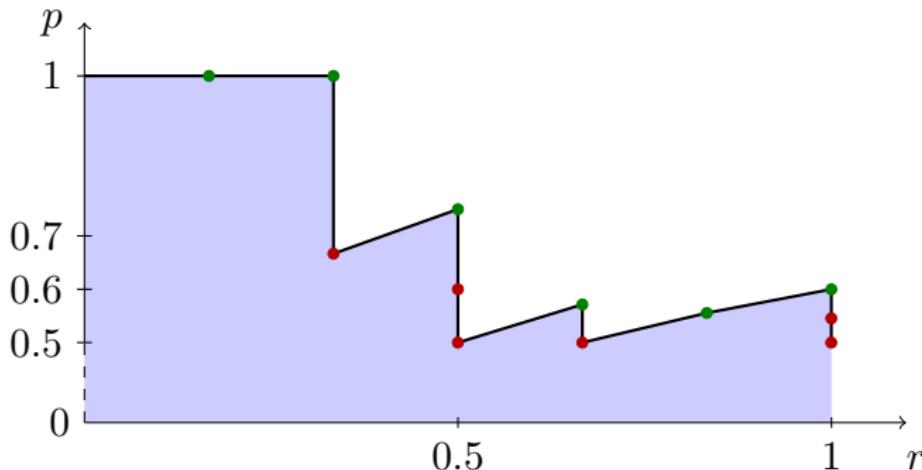


- # total ground truth n , current rank k , # true positives t
- precision $p = \frac{t}{k}$, recall $r = \frac{t}{n}$

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

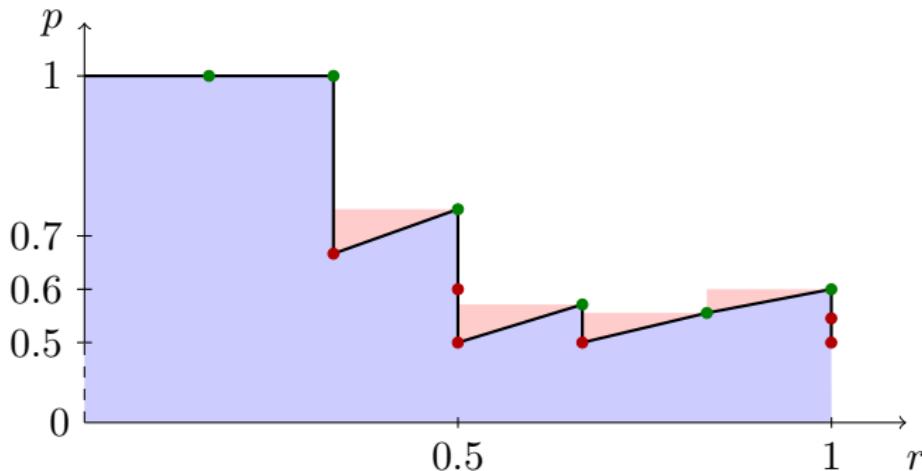


- average precision = area under curve
- the mean average precision (mAP) is the mean over queries

average precision (AP)

- ranked list of items with true/false labels

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F



- average precision = area under curve (filled-in curve)
- the mean average precision (mAP) is the mean over queries

Oxford buildings dataset

[Philbin et al. 2007]



All Souls



Ashmolean



Balliol



Bodleian



Christ Church



Cornmarket



Hertford



Keble



Magdalen



Pitt Rivers



Radcliffe Camera

- **Oxford5k:** 5k images, 11 landmarks, $5 \times 11 = 55$ queries, $10 \sim 200$ positives/query
- **Oxford105k:** 100k additional **distractor** images

Paris dataset

[Philbin et al. 2008]



Defense



Eiffel



Invalides



Louvre



Moulin Rouge



Musée d'Orsay



Notre Dame



Pantheon



Pompidou



Sacré-Cœur



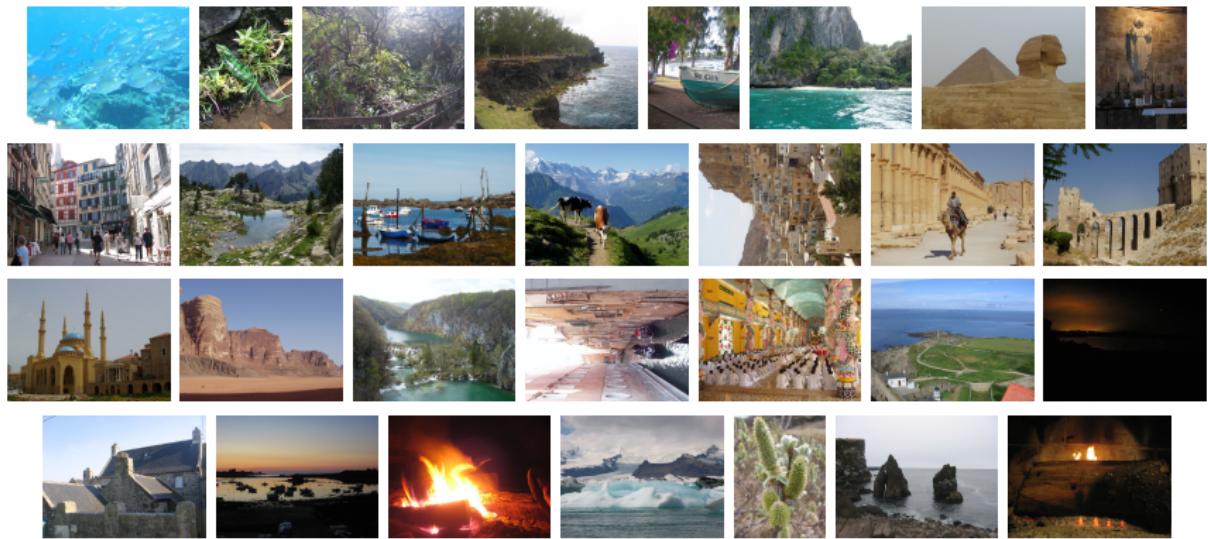
Triomphe

- **Paris6k:** 6k images, 11 landmarks, $5 \times 11 = 55$ queries, $50 \sim 300$ positives/query
- **Paris106k:** same 100k **distractor** images as Oxford

Philbin, Chum, Isard, Sivic and Zisserman. CVPR 2008. Lost in Quantization: Improving Particular Object Retrieval in Large Scale Image Databases.

Holidays dataset

[Jégou et al. 2008]



- personal holiday photos, natural and man-made scenes
- 1.5k images, 500 groups, 1 query/group, 1000 positives, 1 ~ 12 positives/query

aggregated selective match kernel (ASMK)

[Tolias et al. 2013]

- residual **pooling** within cells

$$V(X_c) := \sum_{x \in X_c} r(x) = \sum_{x \in X_c} x - q(x)$$

- nonlinear **selectivity** between cells

$$K(X, Y) := \gamma(X)\gamma(Y) \sum_{c \in C} w_c \sigma_\alpha \left(\hat{V}(X_c)^\top \hat{V}(Y_c) \right)$$

where $\hat{x} := x/\|x\|$ and σ_α a nonlinear function

- 81.7 (83.8) mAP on Oxford5k, $\sim 2.2k$ (3.8k) descriptors/image \times 128 dimensions
- no spatial verification or other post-processing

aggregated selective match kernel (ASMK)

[Tolias et al. 2013]

- residual **pooling** within cells

$$V(X_c) := \sum_{x \in X_c} r(x) = \sum_{x \in X_c} x - q(x)$$

- nonlinear **selectivity** between cells

$$K(X, Y) := \gamma(X)\gamma(Y) \sum_{c \in C} w_c \sigma_\alpha \left(\hat{V}(X_c)^\top \hat{V}(Y_c) \right)$$

where $\hat{x} := x/\|x\|$ and σ_α a nonlinear function

- 81.7 (83.8) mAP on Oxford5k, $\sim 2.2k$ (3.8k) descriptors/image \times 128 dimensions
- no spatial verification or other post-processing

triangulation embedding (T-embedding)

[Jégou and Zisserman 2014]

- normalized residuals, concatenated over cells, pooling over dataset

$$R(X) := \sum_{x \in X} (\hat{r}_1(x), \dots, \hat{r}_k(x)) = \sum_{x \in X} \left(\frac{x - c_1}{\|x - c_1\|}, \dots, \frac{x - c_k}{\|x - c_k\|} \right)$$

where $r_j(x) := x - c_j$ and $\hat{x} := x/\|x\|$

- linear kernel, written as inner product

$$K(X, Y) := (\gamma(X)R(X))^{\top} (\gamma(Y)R(Y))$$

- 57.1 (67.6) mAP on Oxford5k, global descriptor, 1k (8k) dimensions
- no spatial verification or other post-processing

triangulation embedding (T-embedding)

[Jégou and Zisserman 2014]

- normalized residuals, concatenated over cells, pooling over dataset

$$R(X) := \sum_{x \in X} (\hat{r}_1(x), \dots, \hat{r}_k(x)) = \sum_{x \in X} \left(\frac{x - c_1}{\|x - c_1\|}, \dots, \frac{x - c_k}{\|x - c_k\|} \right)$$

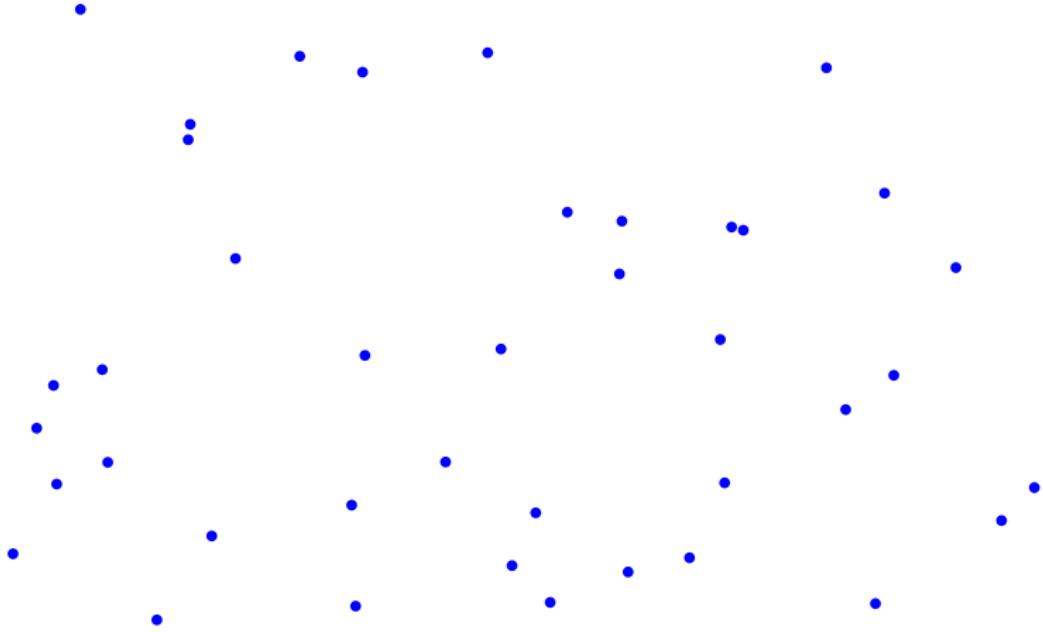
where $r_j(x) := x - c_j$ and $\hat{x} := x/\|x\|$

- linear kernel, written as inner product

$$K(X, Y) := (\gamma(X)R(X))^\top (\gamma(Y)R(Y))$$

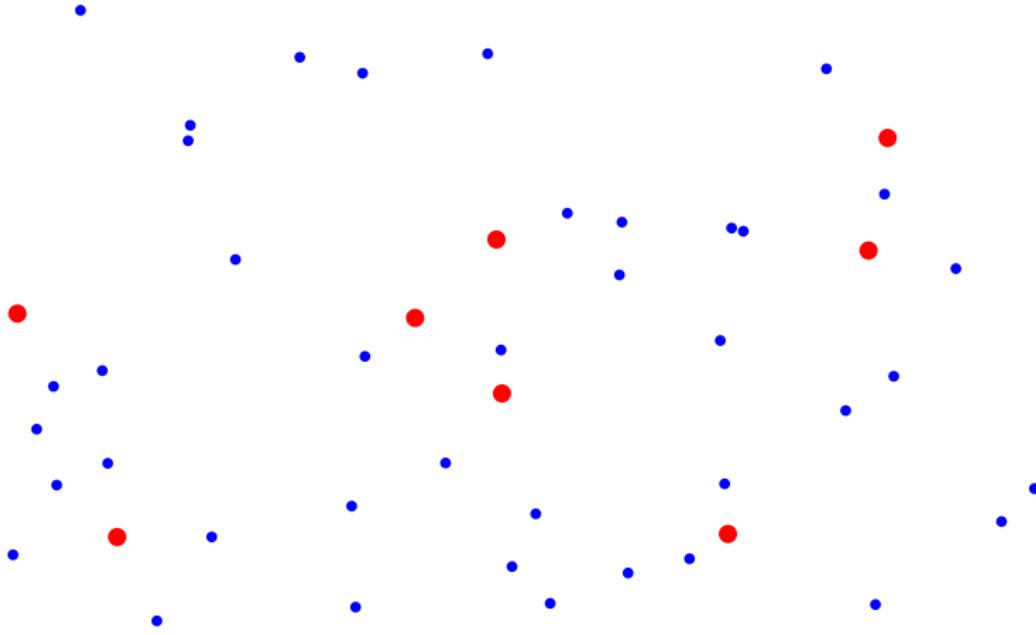
- 57.1 (67.6) mAP on Oxford5k, global descriptor, 1k (8k) dimensions
- no spatial verification or other post-processing

triangulation embedding geometry



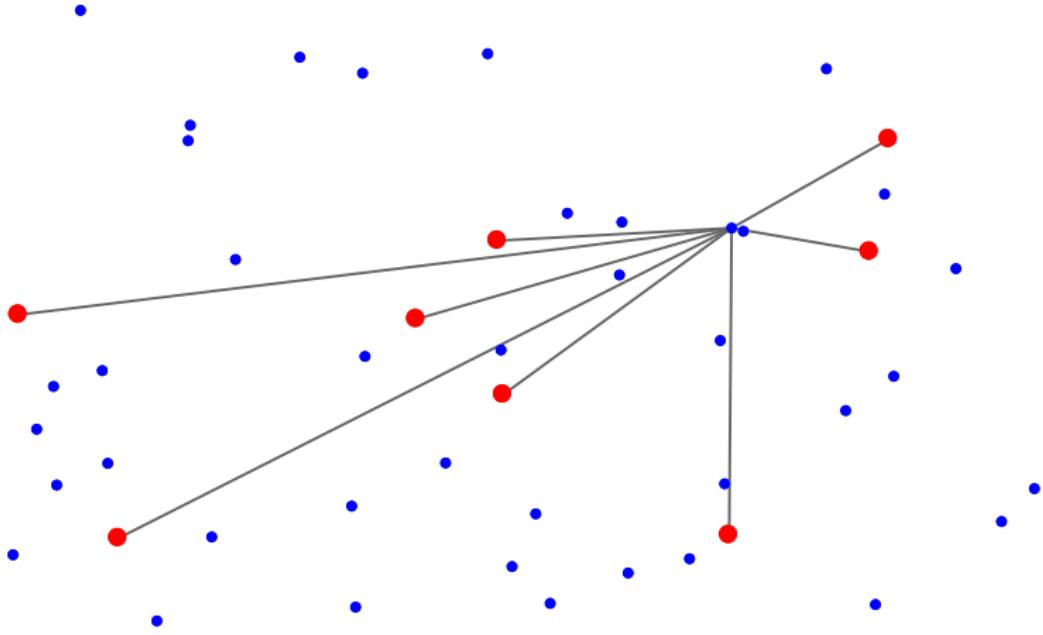
- **input vectors** – codebook – residuals – normalized residuals

triangulation embedding geometry



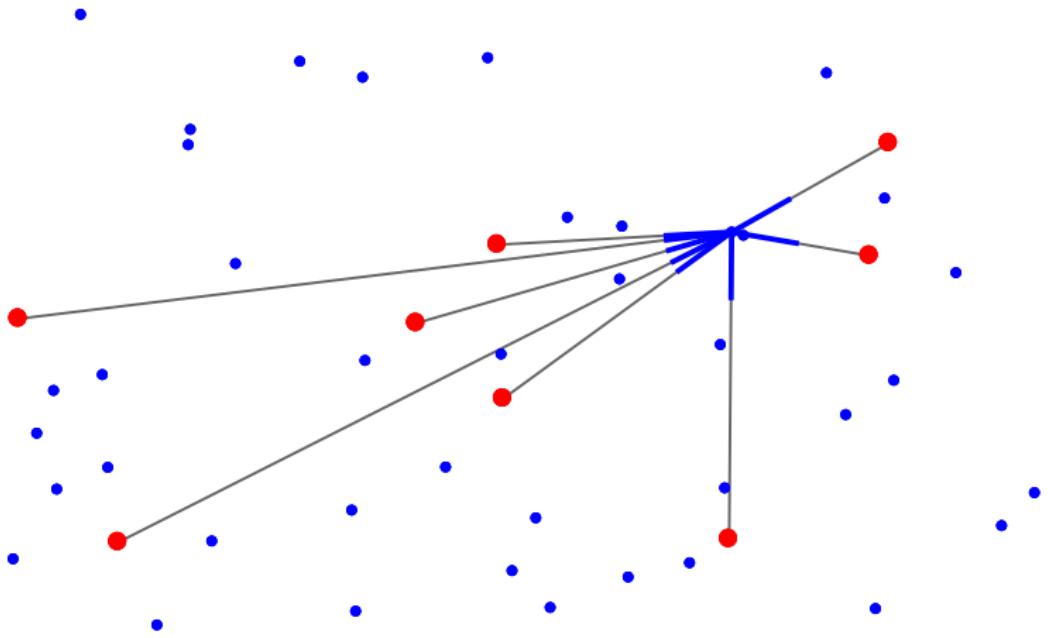
- input vectors – codebook – residuals – normalized residuals

triangulation embedding geometry



- input vectors – codebook – residuals – normalized residuals

triangulation embedding geometry



- input vectors – codebook – residuals – normalized residuals

state of the art before deep learning

- bag of words and **inverted index** is only a crude form of approximate nearest neighbor search for each local descriptor, followed by a kernel function
- for good performance, storing **descriptors** is necessary, even compressed
 - very good performance achieved with **thousands descriptors**/image
 - a **global descriptor**/image allows nearest neighbor search directly on images, but is inferior

state of the art before deep learning

- bag of words and **inverted index** is only a crude form of approximate nearest neighbor search for each local descriptor, followed by a kernel function
- for good performance, storing **descriptors** is necessary, even compressed
- very good performance achieved with **thousands descriptors**/image
- a **global descriptor**/image allows nearest neighbor search directly on images, but is inferior

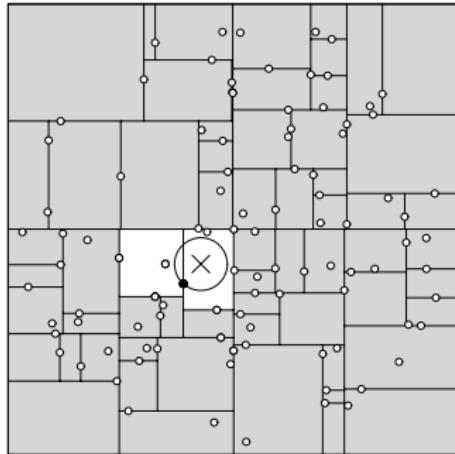
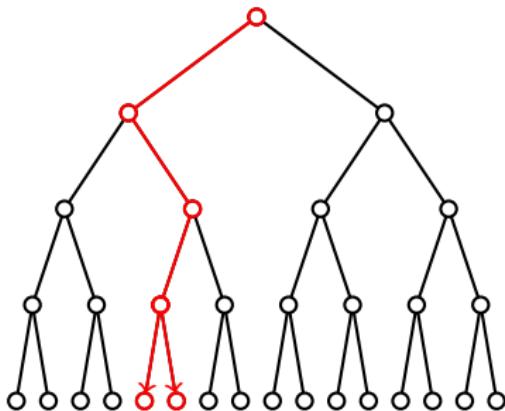
indexing

nearest neighbor search

- given query point y , find its nearest neighbor with respect to Euclidean distance within data set X in a d -dimensional space
- **image retrieval**: same problem; one or multiple queries depending on global or local representation
- **image classification**: nearest neighbor or naïve Bayes nearest neighbor classifier, again depending on representation

k-d tree

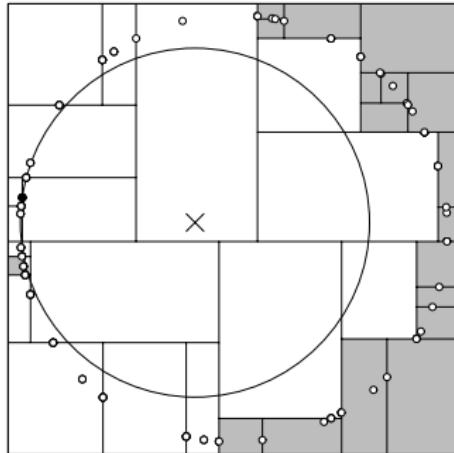
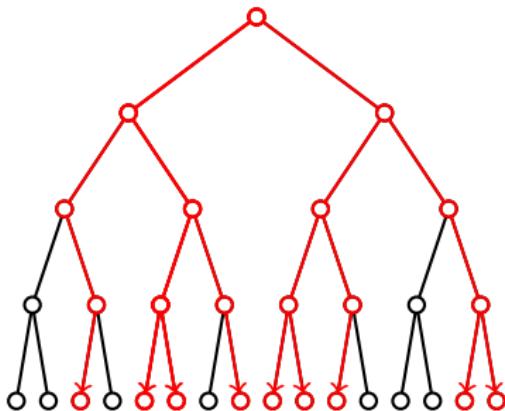
[Bentley 1975]



- **index**: recursively split at medoid on some dimension, make balanced binary tree
- **search**: descend recursively from root, choosing child according to splitting dimension and value
- backtracking becomes exhaustive at high dimensions

k-d tree

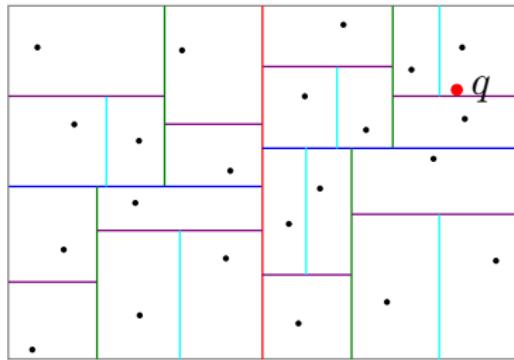
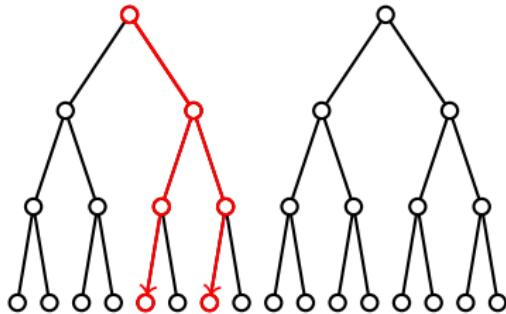
[Bentley 1975]



- **index**: recursively split at medoid on some dimension, make balanced binary tree
- **search**: descend recursively from root, choosing child according to splitting dimension and value
- backtracking becomes exhaustive at high dimensions

randomized k-d trees

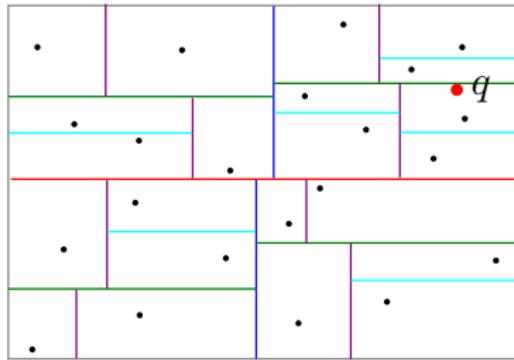
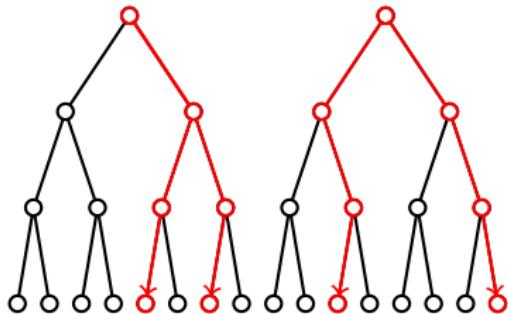
[Silpa-Anan and Hartley 1975]



- **index**: same as before, but now multiple randomized trees
- **search**: descend trees in parallel according to shared priority queue
- still, points are stored, distances are exact

randomized k-d trees

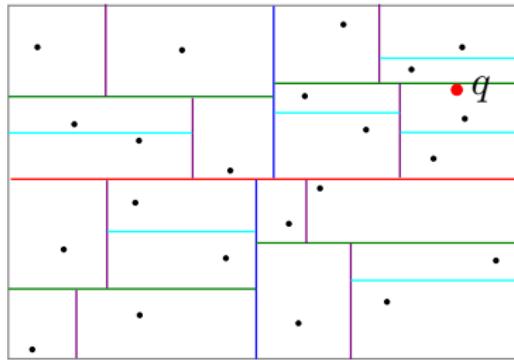
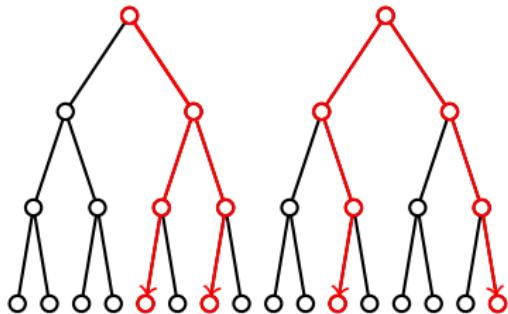
[Silpa-Anan and Hartley 1975]



- **index**: same as before, but now multiple randomized trees
- **search**: descend trees in parallel according to shared priority queue
- still, points are stored, distances are exact

randomized k-d trees

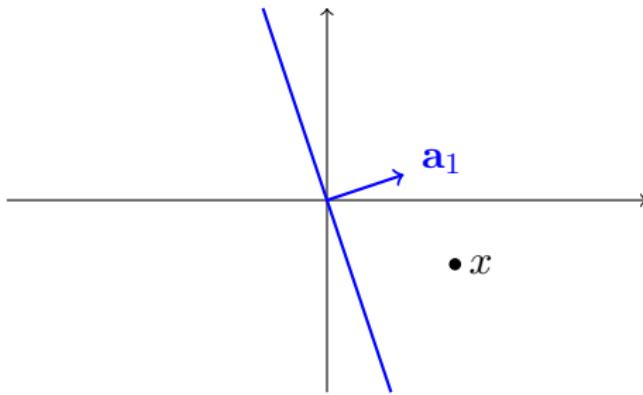
[Silpa-Anan and Hartley 1975]



- **index**: same as before, but now multiple randomized trees
- **search**: descend trees in parallel according to shared priority queue
- still, points are stored, distances are exact

locality sensitive hashing (LSH)

[Charikar 2002]



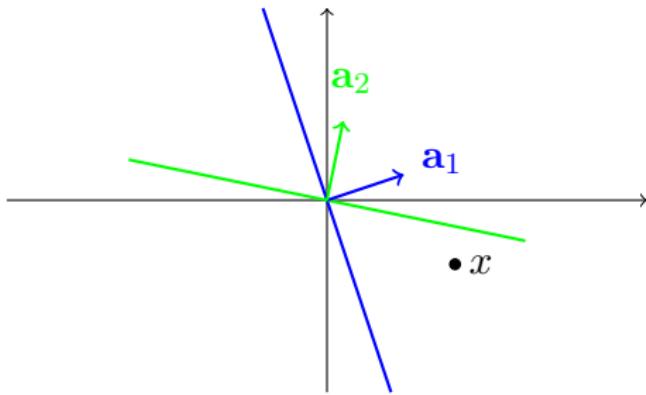
- **index:** choose $\mathbf{a}_i \sim \mathcal{N}(0, 1)$; encode each data point $x \in X$ by **binary code** $h(x) := (h_{\mathbf{a}_1}(x), \dots, h_{\mathbf{a}_k}(x)) \in \{-1, 1\}^d$ with **hash function**

$$h_{\mathbf{a}}(x) = \text{sgn}(\mathbf{a}^\top x)$$

- **search:** encode query y as $h(y)$ and search by **Hamming distance**
- not adapted to data distribution

locality sensitive hashing (LSH)

[Charikar 2002]



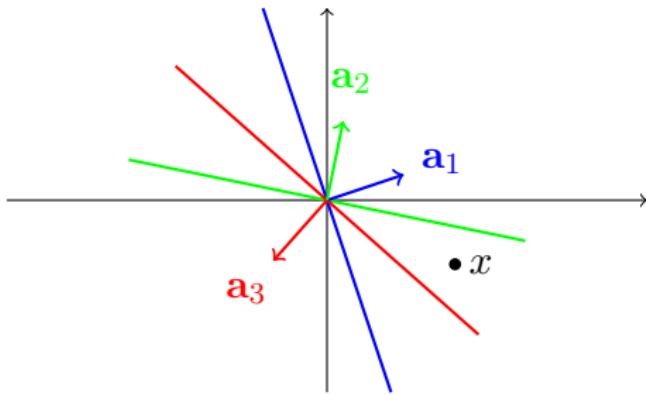
- **index:** choose $\mathbf{a}_i \sim \mathcal{N}(0, 1)$; encode each data point $x \in X$ by **binary code** $h(x) := (h_{\mathbf{a}_1}(x), \dots, h_{\mathbf{a}_k}(x)) \in \{-1, 1\}^d$ with **hash function**

$$h_{\mathbf{a}}(x) = \text{sgn}(\mathbf{a}^\top x)$$

- **search:** encode query y as $h(y)$ and search by **Hamming distance**
- not adapted to data distribution

locality sensitive hashing (LSH)

[Charikar 2002]



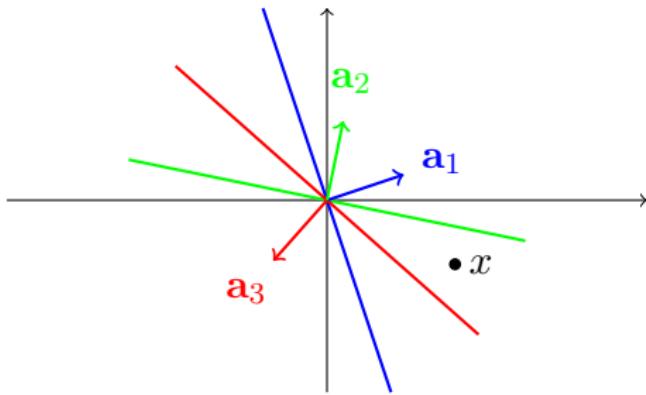
- **index:** choose $\mathbf{a}_i \sim \mathcal{N}(0, 1)$; encode each data point $x \in X$ by **binary code** $h(x) := (h_{\mathbf{a}_1}(x), \dots, h_{\mathbf{a}_k}(x)) \in \{-1, 1\}^d$ with **hash function**

$$h_{\mathbf{a}}(x) = \text{sgn}(\mathbf{a}^\top x)$$

- **search:** encode query y as $h(y)$ and search by **Hamming distance**
- not adapted to data distribution

locality sensitive hashing (LSH)

[Charikar 2002]



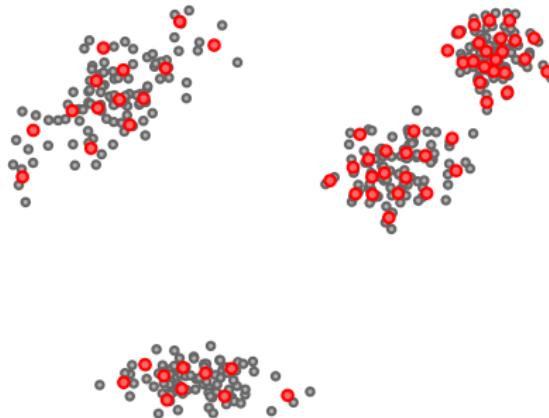
- **index:** choose $\mathbf{a}_i \sim \mathcal{N}(0, 1)$; encode each data point $x \in X$ by **binary code** $h(x) := (h_{\mathbf{a}_1}(x), \dots, h_{\mathbf{a}_k}(x)) \in \{-1, 1\}^d$ with **hash function**

$$h_{\mathbf{a}}(x) = \text{sgn}(\mathbf{a}^\top x)$$

- **search:** encode query y as $h(y)$ and search by **Hamming distance**
- not adapted to data distribution

vector quantization (VQ)

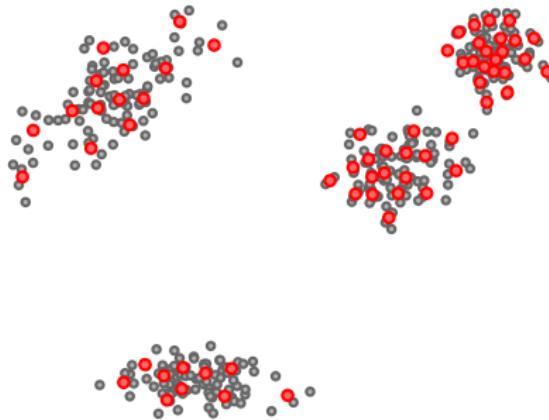
[Gray 1984]



- **index**: cluster X into codebook $C = \{c_1, \dots, c_k\}$; quantize each $x \in X$ to $q(x) = \min_{c \in C} \|x - c\|^2$ and encode it by $\log k$ bits
- **search**: pre-compute distances $\|y - c\|^2$ for $c \in C$ and approximate distances $\|y - x\|^2$ by $\|y - q(x)\|^2$ where $q(x) \in C$
- small distortion \rightarrow large k , too large to store, too slow to search

vector quantization (VQ)

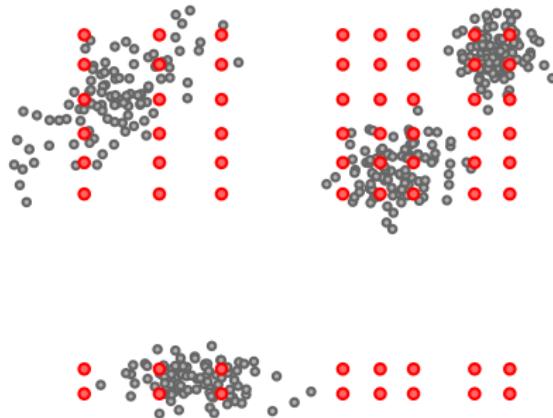
[Gray 1984]



- **index**: cluster X into codebook $C = \{c_1, \dots, c_k\}$; quantize each $x \in X$ to $q(x) = \min_{c \in C} \|x - c\|^2$ and encode it by $\log k$ bits
- **search**: pre-compute distances $\|y - c\|^2$ for $c \in C$ and approximate distances $\|y - x\|^2$ by $\|y - q(x)\|^2$ where $q(x) \in C$
- small distortion \rightarrow large k , too large to store, too slow to search

product quantization (PQ)

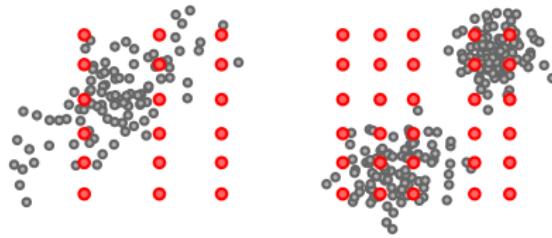
[Jégou et al. 2011]



- **index**: decompose vectors as $x = (x^1, \dots, x^m)$, cluster X into codebook $C = C^1 \times \dots \times C^m$ with k cells each and $|C| = k^m$
- **search**: pre-compute distances $\|y^j - c\|^2$ for $c \in C^j$ and approximate $\|y - x\|^2$ by $\|y - q(x)\|^2 = \sum_{j=1}^m \|y^j - q^j(x^j)\|^2$ where $q^j(x^j) \in C^j$
- a lot of centroids do not represent data and are unused

product quantization (PQ)

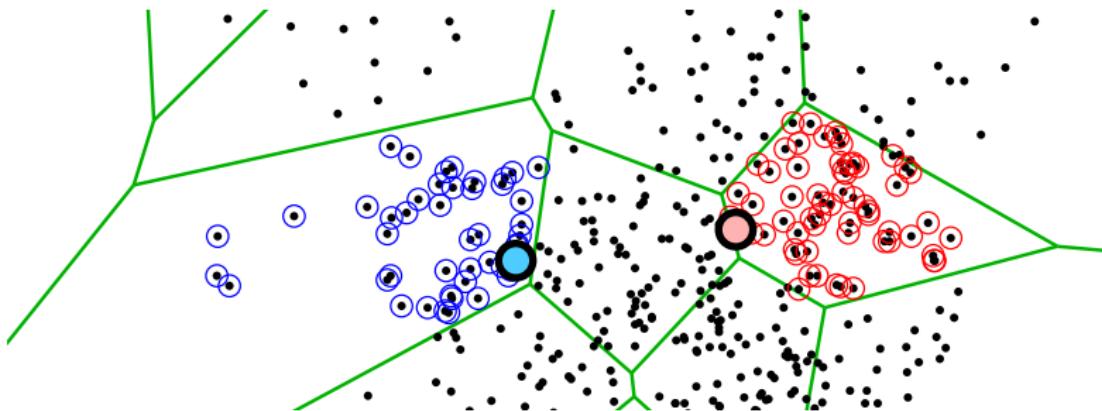
[Jégou et al. 2011]



- **index**: decompose vectors as $x = (x^1, \dots, x^m)$, cluster X into codebook $C = C^1 \times \dots \times C^m$ with k cells each and $|C| = k^m$
- **search**: pre-compute distances $\|y^j - c\|^2$ for $c \in C^j$ and approximate $\|y - x\|^2$ by $\|y - q(x)\|^2 = \sum_{j=1}^m \|y^j - q^j(x^j)\|^2$ where $q^j(x^j) \in C^j$
- a lot of centroids do not represent data and are unused

inverted index

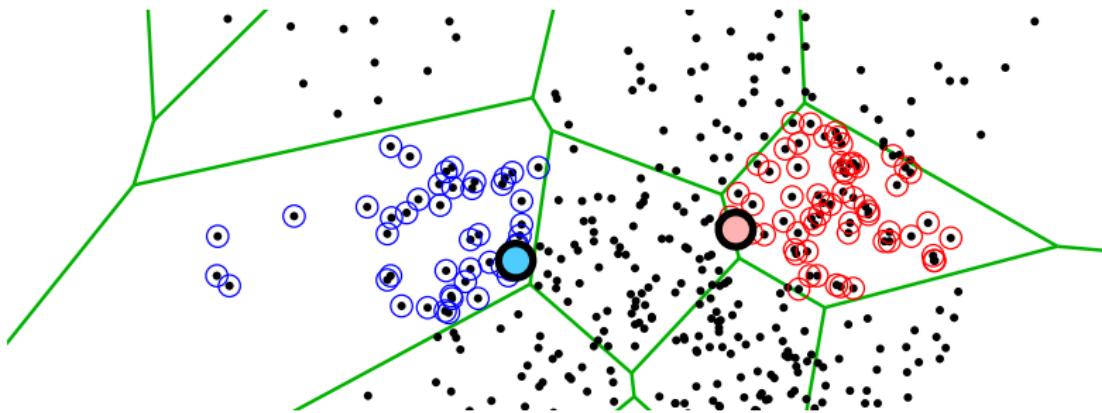
[Jégou et al. 2011]



- **index:** train a coarse quantizer Q of k cells; quantize each $x \in X$ to $Q(x)$, compute **residual** $r(x) = x - Q(x)$ and encode residuals by a product quantizer q
- **search:** quantize query y to a fixed number of nearest cells; exhaustively search by PQ only within those cells
- a lot of points in the coarse cells are too far away from query

inverted index

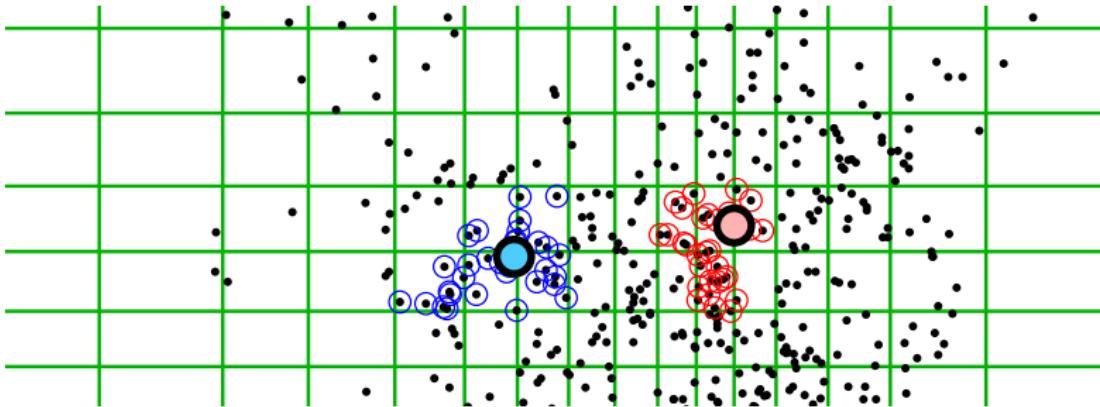
[Jégou et al. 2011]



- **index:** train a coarse quantizer Q of k cells; quantize each $x \in X$ to $Q(x)$, compute **residual** $r(x) = x - Q(x)$ and encode residuals by a product quantizer q
- **search:** quantize query y to a fixed number of nearest cells; exhaustively search by PQ only within those cells
- a lot of points in the coarse cells are too far away from query

inverted multi-index

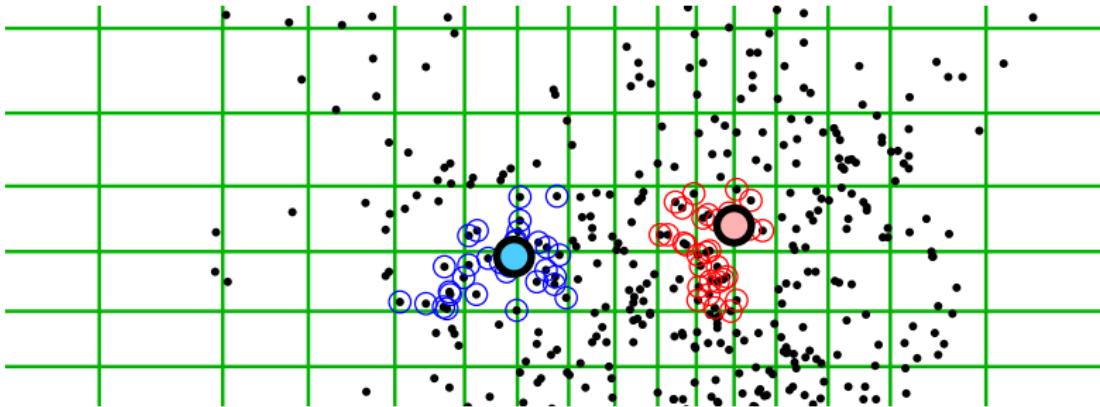
[Babenko and Lempitsky 2012]



- **index:** decompose vectors as $x = (x^1, x^2)$; train two coarse quantizers Q^1, Q^2 of k cells each, quantize each $x \in X$ to $Q^1(x^1), Q^2(x^2)$ and encode residuals by product quantizers q^1, q^2
- **search:** visit cells $(c^1, c^2) \in C^1 \times C^2$ in ascending order of distance to y by **multi-sequence** algorithm
- two coarse quantizers induce a **finer** partition than one

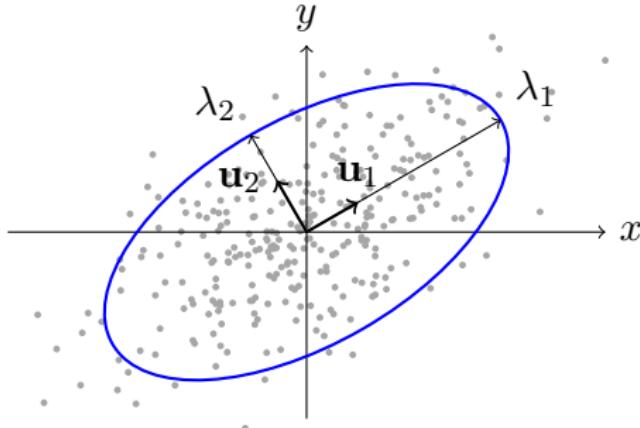
inverted multi-index

[Babenko and Lempitsky 2012]



- **index:** decompose vectors as $x = (x^1, x^2)$; train two coarse quantizers Q^1, Q^2 of k cells each, quantize each $x \in X$ to $Q^1(x^1), Q^2(x^2)$ and encode residuals by product quantizers q^1, q^2
- **search:** visit cells $(c^1, c^2) \in C^1 \times C^2$ in ascending order of distance to y by **multi-sequence** algorithm
- two coarse quantizers induce a **finer** partition than one

principal component analysis (PCA)



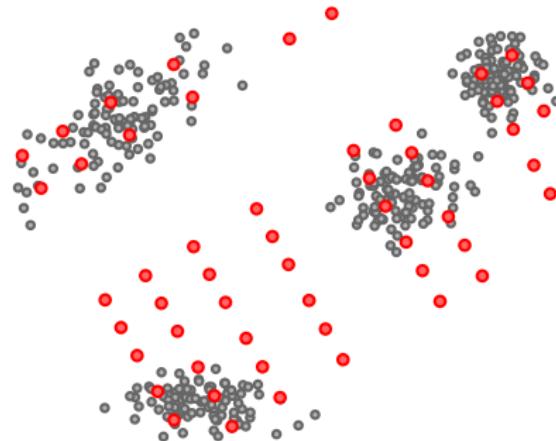
- given data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, compute empirical mean $\bar{\mathbf{x}} := \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$ and covariance matrix

$$S := \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top$$

- then diagonalize S by $S = U \Lambda U^\top$ where $U = (\mathbf{u}_1 \ \mathbf{u}_2)$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2)$

optimized product quantization (OPQ)

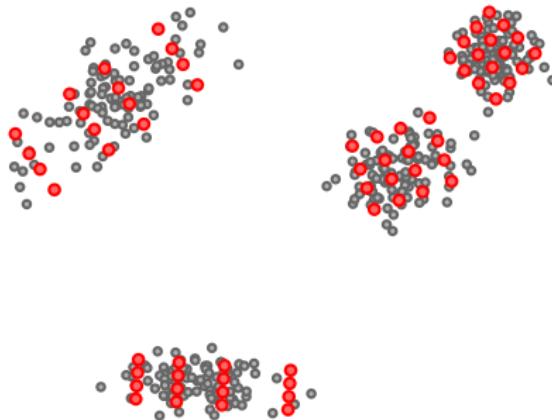
[Ge et al. 2013]



- **no correlation**: PCA-align by diagonalizing $\text{cov}(X)$ as $U\Lambda U^\top$
- **balanced variance**: shuffle eigenvalues Λ by permutation π such that the product $\prod_i \lambda_i$ is constant in each subspace
- find codebook \hat{C} by PQ on rotated data $\hat{X} := RX$ where $R := UP_\pi^\top$ and P_π is the permutation matrix of π

locally optimized product quantization (LOPQ)

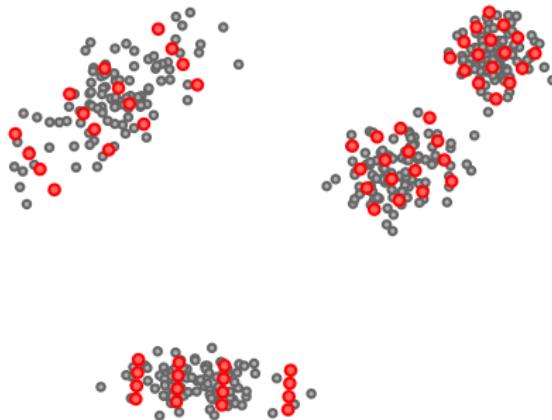
[Kalantidis and Avrithis 2014]



- same as PQ with inverted index (or multi-index), but residuals are encoded by OPQ
- better on multimodal data: residual distributions closer to Gaussian assumption

locally optimized product quantization (LOPQ)

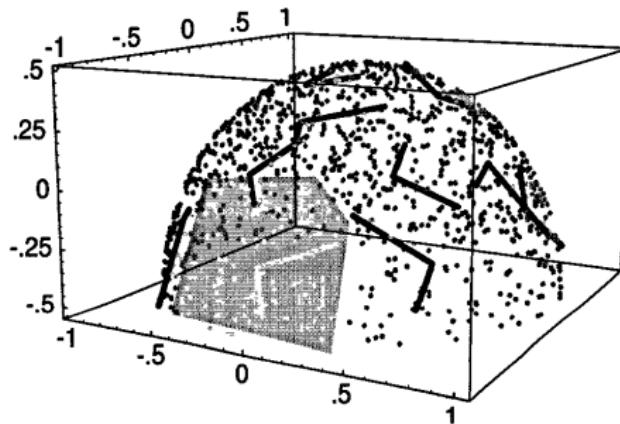
[Kalantidis and Avrithis 2014]



- same as PQ with inverted index (or multi-index), but residuals are encoded by OPQ
- better on multimodal data: residual distributions closer to Gaussian assumption

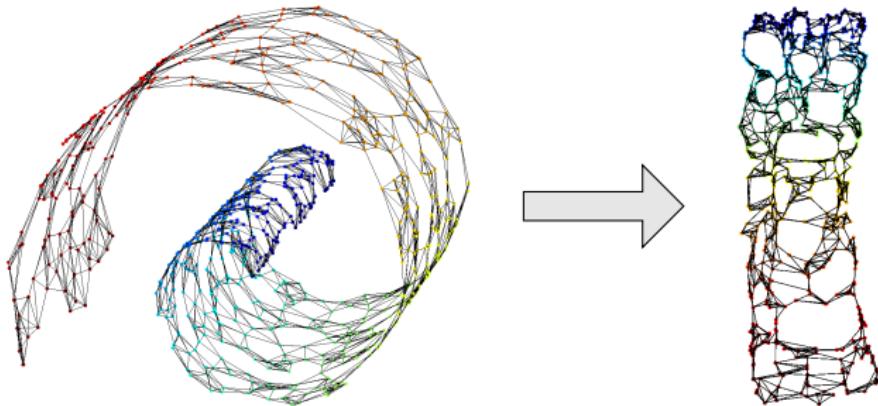
local principal component analysis

[Kambhatla & Leen 1997]



- cluster data, then apply PCA per cell
- captures the support of data distribution
 - multimodal (e.g. mixture) distributions
 - distributions on nonlinear manifolds

manifold learning

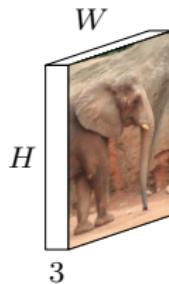


- e.g. Isomap: apply PCA to the geodesic (graph) distance matrix
- e.g. kernel PCA: apply PCA to the Gram matrix of a nonlinear kernel
- other **topology-preserving** methods are only focusing on distances to nearest neighbors
- many classic methods use eigenvalue decomposition and most do **not** learn and **explicit mapping** from the input to the embedding space

pooling

image ranking by CNN features

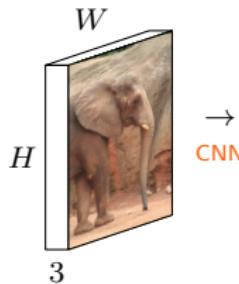
[Krizhevsky et al. 2012]



- 3-channel RGB input, 224×224
- AlexNet pre-trained on ImageNet for classification
- last fully connected layer, **global descriptor** of dimension $k = 4096$

image ranking by CNN features

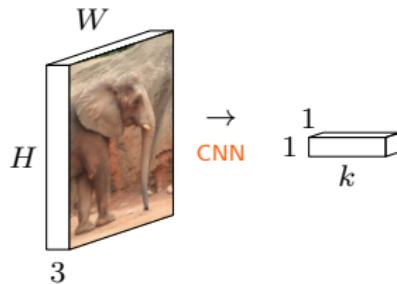
[Krizhevsky et al. 2012]



- 3-channel RGB input, 224×224
- AlexNet pre-trained on ImageNet for classification
- last fully connected layer, **global descriptor** of dimension $k = 4096$

image ranking by CNN features

[Krizhevsky et al. 2012]



- 3-channel RGB input, 224×224
- AlexNet pre-trained on ImageNet for classification
- last fully connected layer, **global descriptor** of dimension $k = 4096$

image ranking by CNN features



- query images
- nearest neighbors in ImageNet according to Euclidean distance

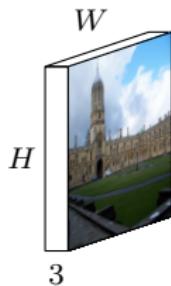
image ranking by CNN features



- query images
 - nearest neighbors in ImageNet according to Euclidean distance

neural codes for image retrieval

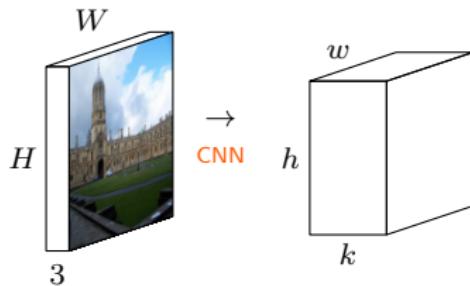
[Babenko et al. 2014]



- 3-channel RGB input, 224×224
- AlexNet last pooling layer, *global descriptor* of dimension $w \times h \times k = 6 \times 6 \times 256 = 9216$
- alternatively: fully convolutional layers fc_6, fc_7 , *global descriptors* of dimension $k' = 4096$ (besides fc_5)
- in each case: PCA-whitening, ℓ_2 normalization

neural codes for image retrieval

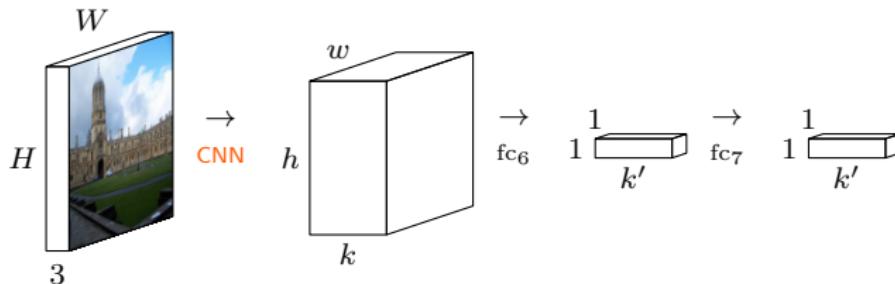
[Babenko et al. 2014]



- 3-channel RGB input, 224×224
- AlexNet last pooling layer, **global descriptor** of dimension $w \times h \times k = 6 \times 6 \times 256 = 9216$
- alternatively: fully convolutional layers fc_6, fc_7 , **global descriptors** of dimension $k' = 4096$ (instead fc_7)
- in each case: PCA-whitening, ℓ_2 normalization

neural codes for image retrieval

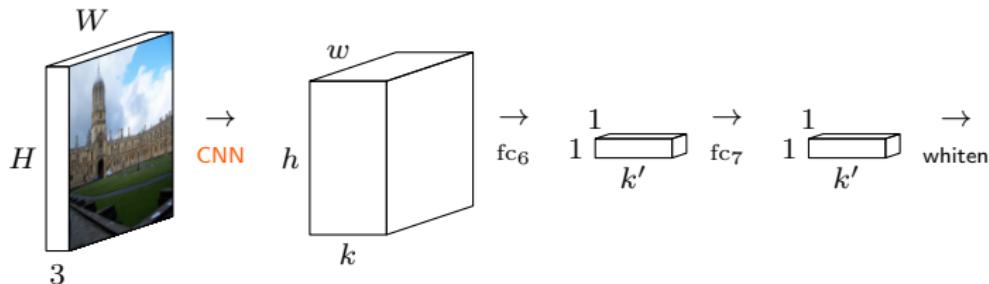
[Babenko et al. 2014]



- 3-channel RGB input, 224×224
- AlexNet last pooling layer, **global descriptor** of dimension $w \times h \times k = 6 \times 6 \times 256 = 9216$
- alternatively: fully convolutional layers fc_6, fc_7 , **global descriptors** of dimension $k' = 4096$ (**best is fc_6**)
- in each case: PCA-whitening, ℓ_2 normalization

neural codes for image retrieval

[Babenko et al. 2014]



- 3-channel RGB input, 224×224
- AlexNet last pooling layer, **global descriptor** of dimension $w \times h \times k = 6 \times 6 \times 256 = 9216$
- alternatively: fully convolutional layers fc₆, fc₇, **global descriptors** of dimension $k' = 4096$ (**best is fc₆**)
- in each case: PCA-whitening, ℓ_2 normalization

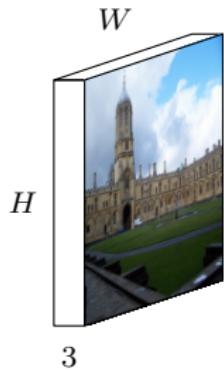
neural codes for image retrieval



- **fine-tuning** by softmax on 672 classes of 200k landmark photos
- outperforms VLAD and Fisher vectors on standard retrieval benchmarks, but still inferior to SIFT local descriptors

regional CNN features

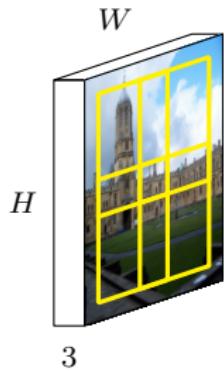
[Razavian et al. 2015]



- 3-channel RGB input, largest square region extracted
- fixed multiscale overlapping regions, warped into $w \times h = 227 \times 227$
- each region yields a $w' \times h' \times k = 36 \times 36 \times 256$ dimensional feature at the last convolutional layer of AlexNet
- global spatial max-pooling
- ℓ_2 -normalization, PCA-whitening of each descriptor

regional CNN features

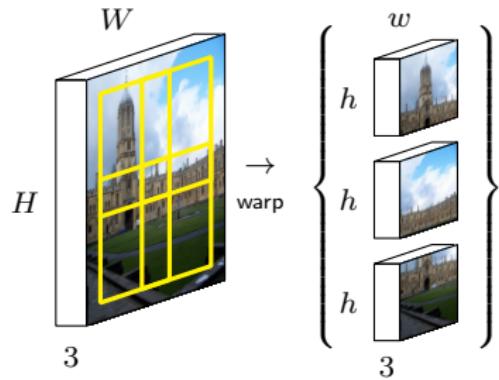
[Razavian et al. 2015]



- 3-channel RGB input, largest square region extracted
- fixed multiscale overlapping regions, warped into $w \times h = 227 \times 227$
- each region yields a $w' \times h' \times k = 36 \times 36 \times 256$ dimensional feature at the last convolutional layer of AlexNet
- global spatial max-pooling
- ℓ_2 -normalization, PCA-whitening of each descriptor

regional CNN features

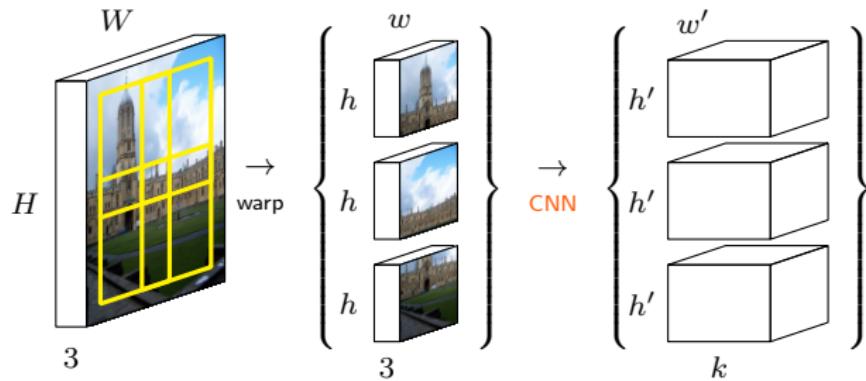
[Razavian et al. 2015]



- 3-channel RGB input, largest square region extracted
- fixed multiscale overlapping regions, **warped** into $w \times h = 227 \times 227$
- each region yields a $w' \times h' \times k = 36 \times 36 \times 256$ dimensional feature at the last convolutional layer of AlexNet
- global spatial **max**-pooling
- ℓ_2 -normalization, PCA-whitening of each descriptor

regional CNN features

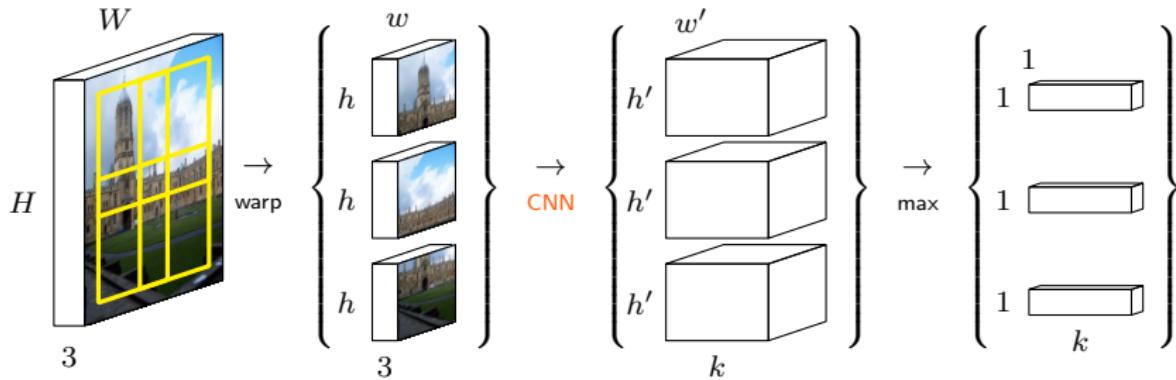
[Razavian et al. 2015]



- 3-channel RGB input, largest square region extracted
- fixed multiscale overlapping regions, **warped** into $w \times h = 227 \times 227$
- **each region** yields a $w' \times h' \times k = 36 \times 36 \times 256$ dimensional feature at the last convolutional layer of AlexNet
- global spatial **max-pooling**
- ℓ_2 -normalization, PCA-whitening of each descriptor

regional CNN features

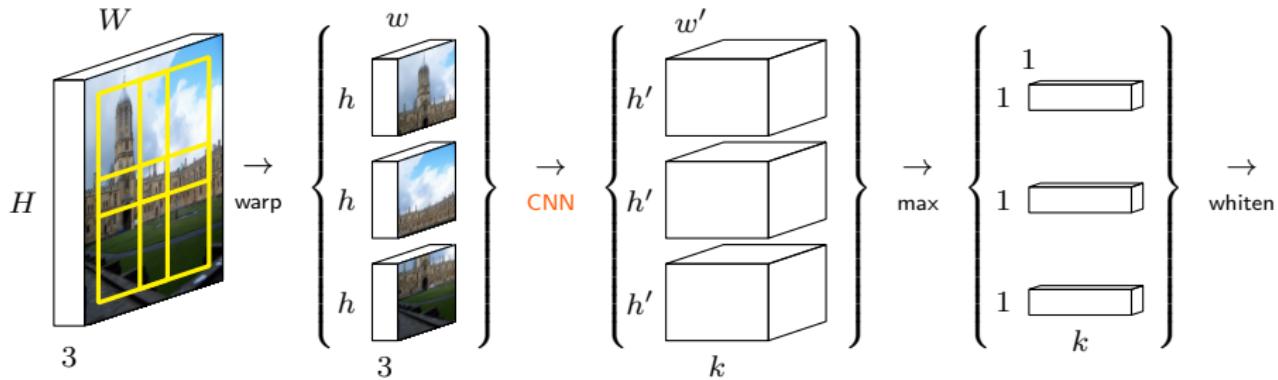
[Razavian et al. 2015]



- 3-channel RGB input, largest square region extracted
 - fixed multiscale overlapping regions, **warped** into $w \times h = 227 \times 227$
 - **each region** yields a $w' \times h' \times k = 36 \times 36 \times 256$ dimensional feature at the last convolutional layer of AlexNet
 - global spatial **max**-pooling
 - ℓ_2 -normalization, PCA-whitening of each descriptor

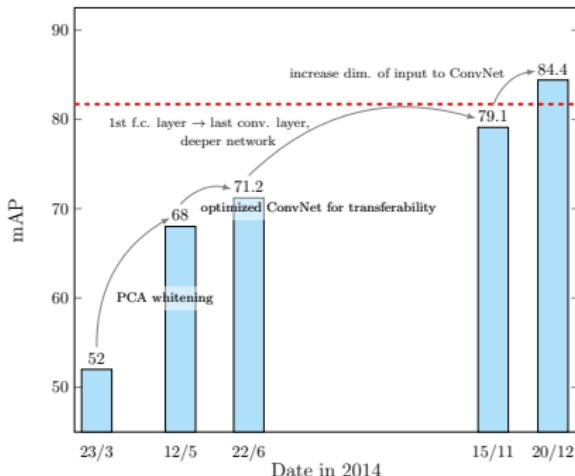
regional CNN features

[Razavian et al. 2015]



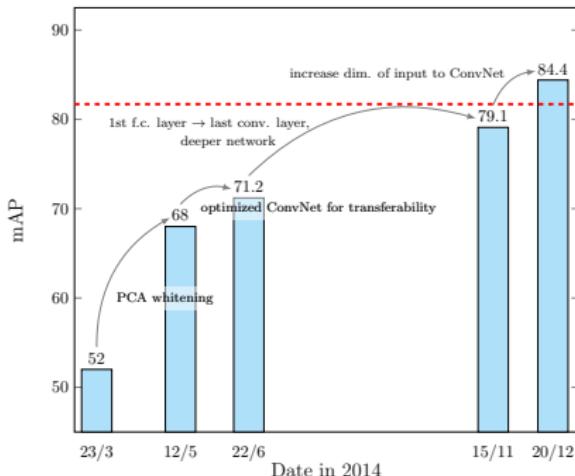
- 3-channel RGB input, largest square region extracted
- fixed multiscale overlapping regions, **warped** into $w \times h = 227 \times 227$
- **each region** yields a $w' \times h' \times k = 36 \times 36 \times 256$ dimensional feature at the last convolutional layer of AlexNet
- global spatial **max**-pooling
- ℓ_2 -normalization, PCA-whitening of each descriptor

regional CNN features



- CNN visual representation jumps by more than 30% mAP to outperform standard SIFT pipeline in a few months
- however, this is based on **multiple** regional descriptors per image and **exhaustive** pairwise matching of all descriptors of query and all dataset images, which is not practical

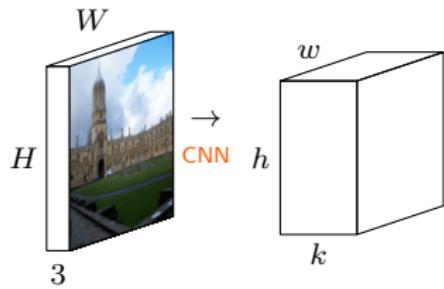
regional CNN features



- CNN visual representation jumps by more than 30% mAP to outperform standard SIFT pipeline in a few months
- however, this is based on **multiple** regional descriptors per image and **exhaustive** pairwise matching of all descriptors of query and all dataset images, which is not practical

regional max-pooling (R-MAC)

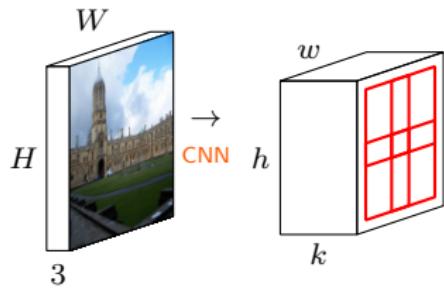
[Tolias et al. 2016]



- VGG-16 last convolutional layer, $k = 512$
 - fixed multiscale overlapping regions spatial max-pooling
 - ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
 - sum-pooling over all descriptors, ℓ_2 -normalization

regional max-pooling (R-MAC)

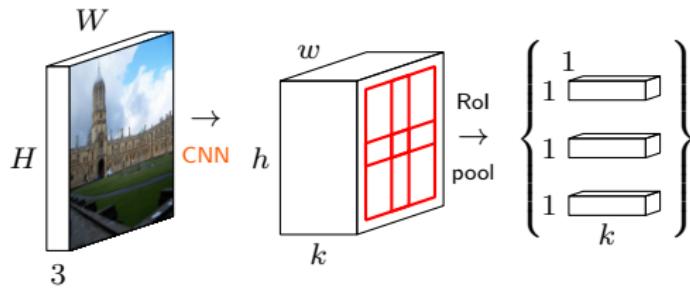
[Tolias et al. 2016]



- VGG-16 last convolutional layer, $k = 512$
- fixed multiscale overlapping regions, spatial max-pooling
- ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
- sum-pooling over all descriptors, ℓ_2 -normalization

regional max-pooling (R-MAC)

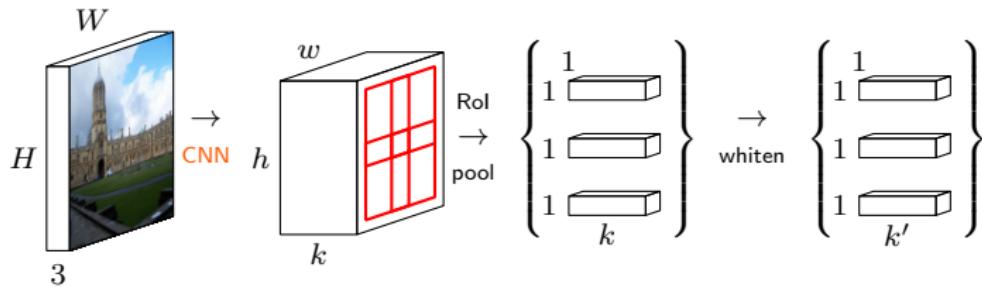
[Tolias et al. 2016]



- VGG-16 last convolutional layer, $k = 512$
 - fixed multiscale overlapping regions, spatial **max**-pooling
 - ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
 - sum-pooling over all descriptors, ℓ_2 -normalization

regional max-pooling (R-MAC)

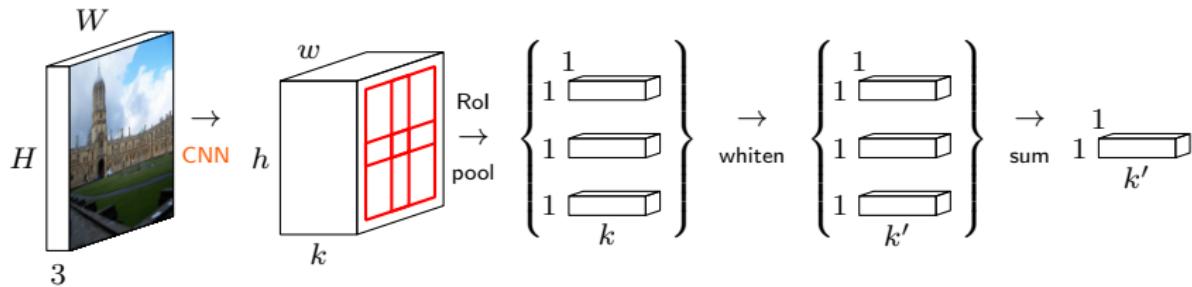
[Tolias et al. 2016]



- VGG-16 last convolutional layer, $k = 512$
 - fixed multiscale overlapping regions, spatial **max**-pooling
 - ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
 - sum-pooling over all descriptors, ℓ_2 -normalization

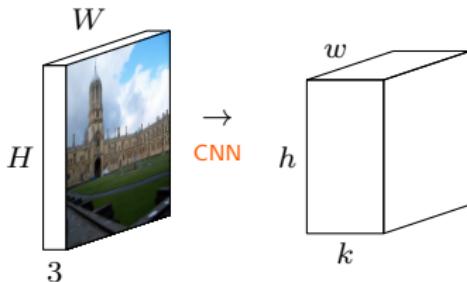
regional max-pooling (R-MAC)

[Tolias et al. 2016]



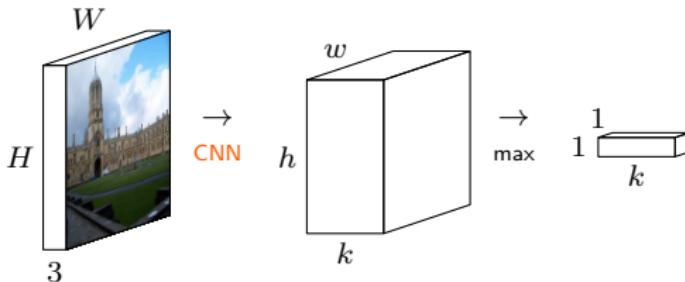
- VGG-16 last convolutional layer, $k = 512$
 - fixed multiscale overlapping regions, spatial max-pooling
 - ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
 - sum-pooling over all descriptors, ℓ_2 -normalization

global max-pooling (MAC)



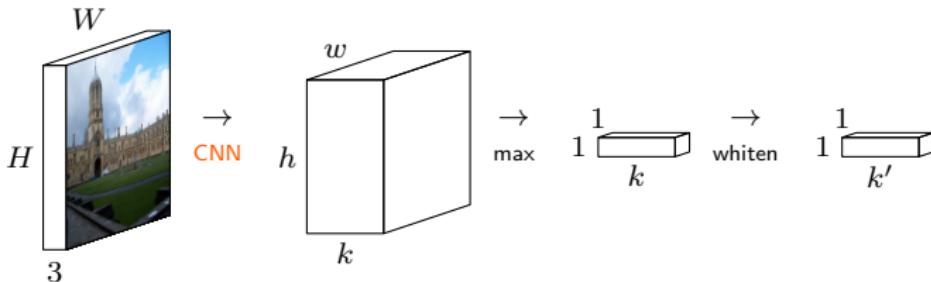
- VGG-16 last convolutional layer, $k = 512$
- global spatial max-pooling
- ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
- MAC: maximum activation of convolutions

global max-pooling (MAC)



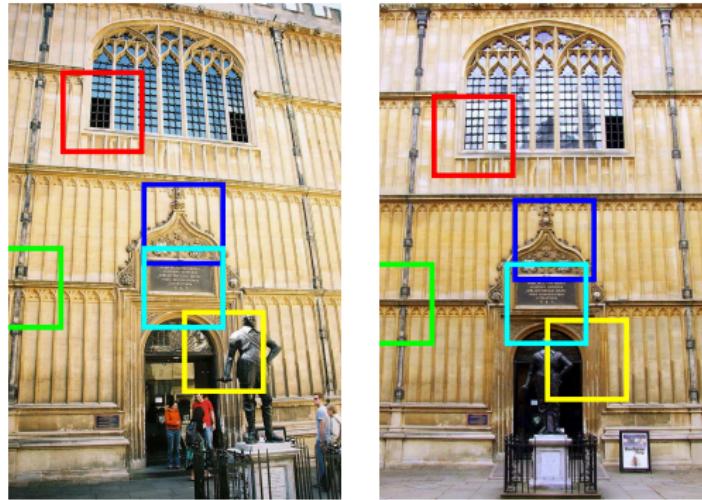
- VGG-16 last convolutional layer, $k = 512$
- global spatial max-pooling
- ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
- MAC: maximum activation of convolutions

global max-pooling (MAC)



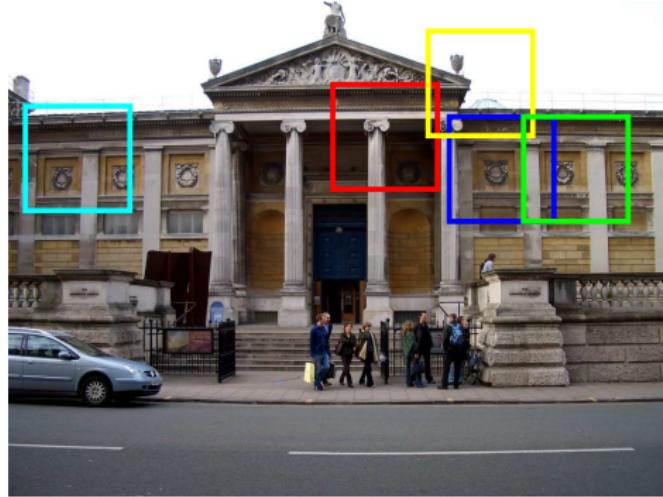
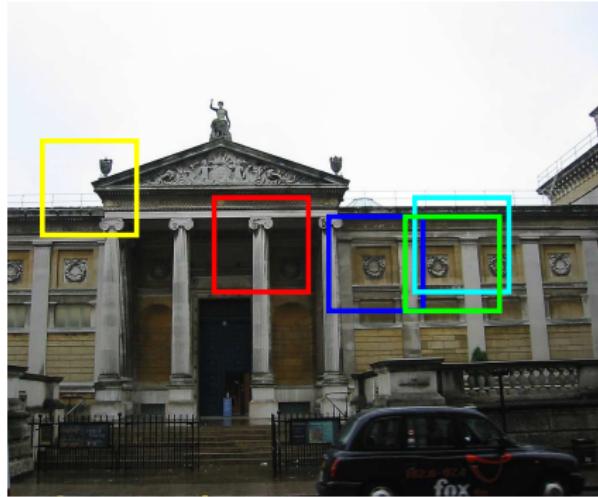
- VGG-16 last convolutional layer, $k = 512$
 - global spatial **max**-pooling
 - ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
 - **MAC**: maximum activation of convolutions

global max-pooling: matching



- receptive fields of 5 components of MAC vectors that contribute most to image similarity

global max-pooling: matching



- receptive fields of 5 components of MAC vectors that contribute most to image similarity

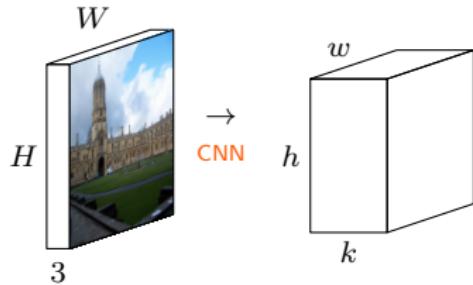
global max-pooling: matching



- receptive fields of 5 components of MAC vectors that contribute most to image similarity

global sum-pooling (SPoC)

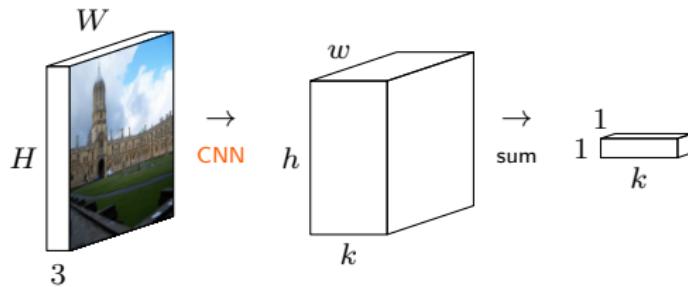
[Babenko and Lempitsky 2015]



- VGG-19 last convolutional layer, $k = 512$
- global spatial sum-pooling
- ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
- SPoC: sum-pooled convolutional features

global sum-pooling (SPoC)

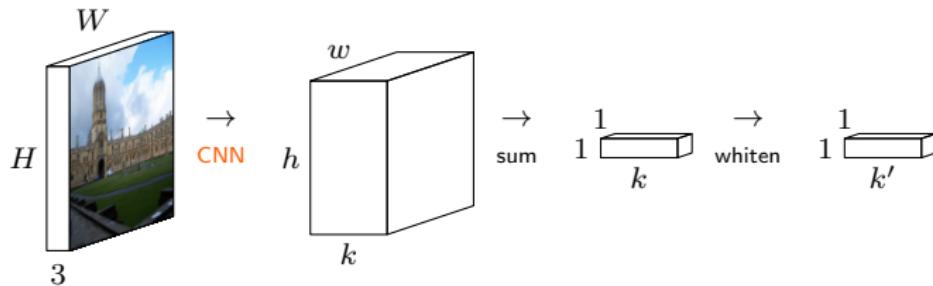
[Babenko and Lempitsky 2015]



- VGG-19 last convolutional layer, $k = 512$
- global spatial sum-pooling
- ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
- SPoC: sum-pooled convolutional features

global sum-pooling (SPoC)

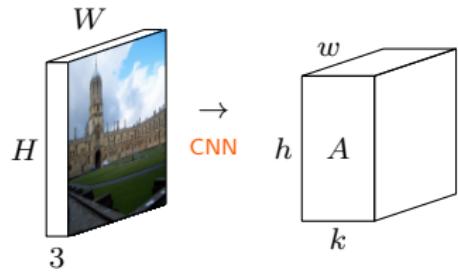
[Babenko and Lempitsky 2015]



- VGG-19 last convolutional layer, $k = 512$
- global spatial **sum**-pooling
- ℓ_2 -normalization, PCA-whitening, ℓ_2 -normalization
- **SPoC**: sum-pooled convolutional features

cross-dimensional weighting (CroW)

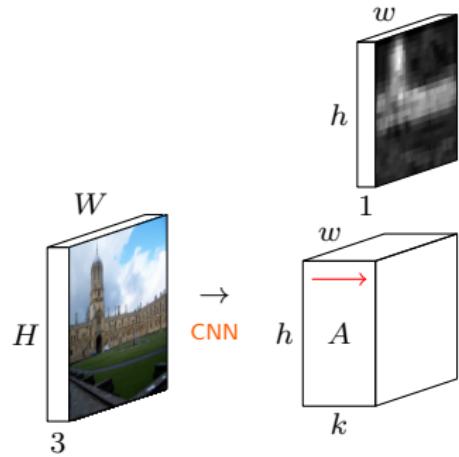
[Kalantidis et al. 2016]



- VGG-16 feature map A , last **pooling** layer, $k = 512$
- spatial weights F , channel weights w , weighted feature map
- global spatial **sum-pooling**
- ℓ_p -normalization, PCA-whitening, ℓ_2 -normalization

cross-dimensional weighting (CroW)

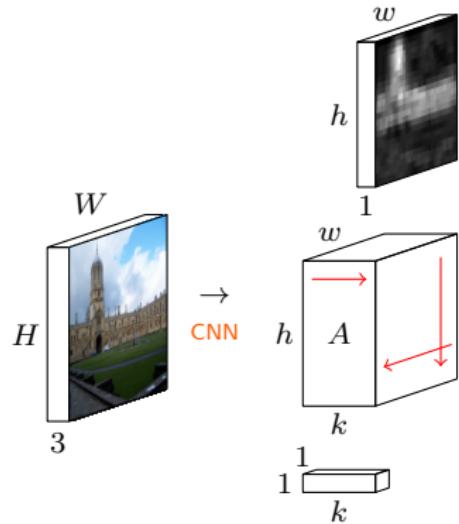
[Kalantidis et al. 2016]



- VGG-16 feature map A , last **pooling** layer, $k = 512$
- **spatial** weights F , **channel** weights w , **weighted** feature map
- global spatial **sum-pooling**
- ℓ_p -normalization, PCA-whitening, ℓ_2 -normalization

cross-dimensional weighting (CroW)

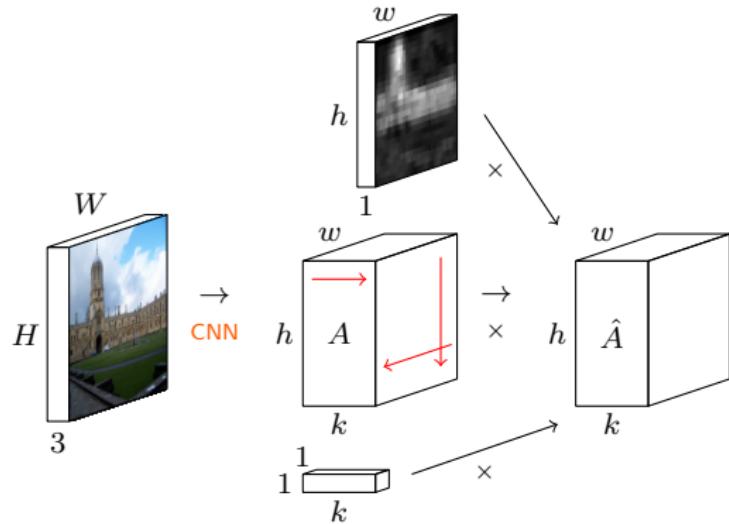
[Kalantidis et al. 2016]



- VGG-16 feature map A , last **pooling** layer, $k = 512$
- **spatial** weights F , **channel** weights \mathbf{w} , **weighted** feature map
- global spatial **sum-pooling**
- ℓ_p -normalization, PCA-whitening, ℓ_2 -normalization

cross-dimensional weighting (CroW)

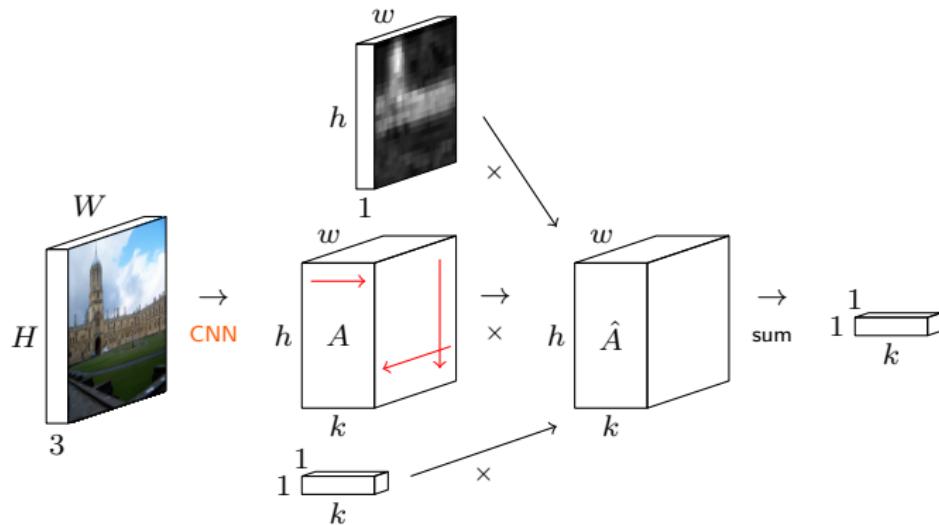
[Kalantidis et al. 2016]



- VGG-16 feature map A , last **pooling** layer, $k = 512$
- **spatial** weights F , **channel** weights w , **weighted** feature map
- global spatial **sum-pooling**
- ℓ_p -normalization, PCA-whitening, ℓ_2 -normalization

cross-dimensional weighting (CroW)

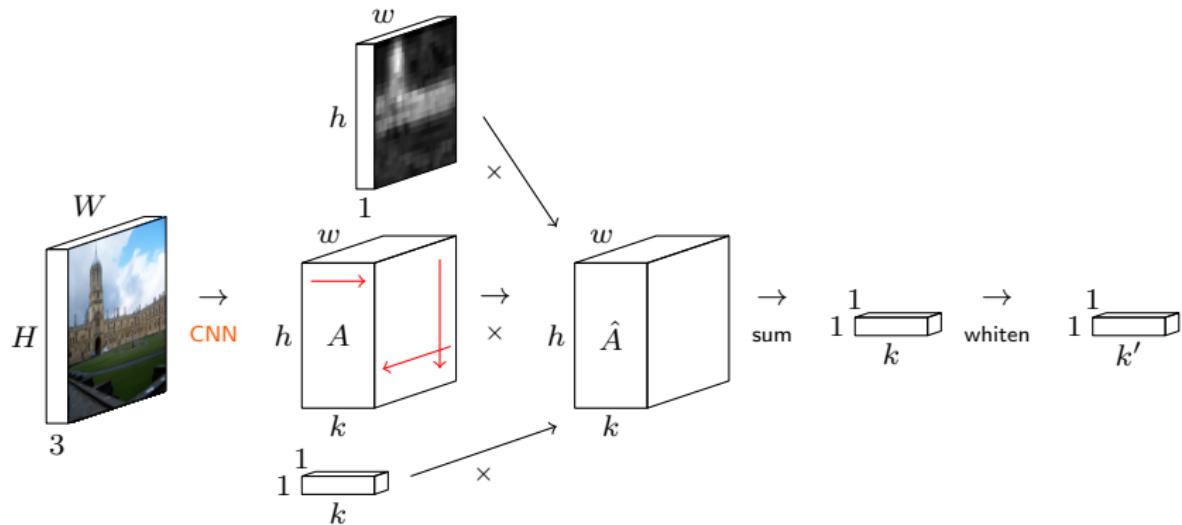
[Kalantidis et al. 2016]



- VGG-16 feature map A , last **pooling** layer, $k = 512$
- **spatial** weights F , **channel** weights w , **weighted** feature map
- global spatial **sum-pooling**
- ℓ_p -normalization, PCA-whitening, ℓ_2 -normalization

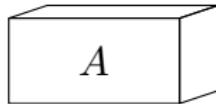
cross-dimensional weighting (CroW)

[Kalantidis et al. 2016]



- VGG-16 feature map A , last **pooling** layer, $k = 512$
- **spatial** weights F , **channel** weights w , **weighted** feature map
- global spatial **sum-pooling**
- ℓ_p -normalization, PCA-whitening, ℓ_2 -normalization

cross-dimensional weighting (CroW)



- **spatial** weights (visual saliency)

$$F(x, y) = \sum_k A_k(x, y)$$

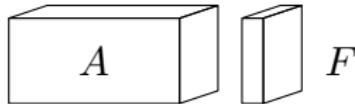
- **channel** weights (sparsity sensitive)

$$w_j = -\log \left(\epsilon + \sum_{x,y} \mathbb{1}[A_j(x, y)] \right)$$

- **weighted** feature map

$$\hat{A} = A \times F \times \mathbf{w}$$

cross-dimensional weighting (CroW)



- **spatial** weights (visual saliency)

$$F(x, y) = \sum_k A_k(x, y)$$

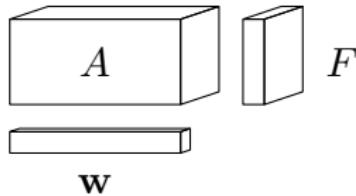
- **channel** weights (sparsity sensitive)

$$w_j = -\log \left(\epsilon + \sum_{x,y} \mathbb{1}[A_j(x, y)] \right)$$

- **weighted** feature map

$$\hat{A} = A \times F \times \mathbf{w}$$

cross-dimensional weighting (CroW)



- **spatial** weights (visual saliency)

$$F(x, y) = \sum_k A_k(x, y)$$

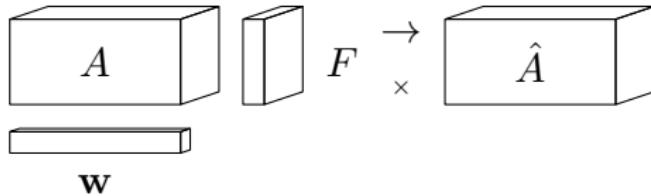
- **channel** weights (sparsity sensitive)

$$w_j = -\log \left(\epsilon + \sum_{x,y} \mathbb{1}[A_j(x, y)] \right)$$

- **weighted** feature map

$$\hat{A} = A \times F \times w$$

cross-dimensional weighting (CroW)



- **spatial** weights (visual saliency)

$$F(x, y) = \sum_k A_k(x, y)$$

- **channel** weights (sparsity sensitive)

$$w_j = -\log \left(\epsilon + \sum_{x,y} \mathbb{1}[A_j(x, y)] \right)$$

- **weighted** feature map

$$\hat{A} = A \times F \times \mathbf{w}$$

cross-dimensional weighting (CroW)



- input image

cross-dimensional weighting (CroW)



- receptive fields of nonzero elements of the 10 channels with the highest sparsity-sensitive weights

manifold learning

siamese architecture

[Chopra et al. 2005]

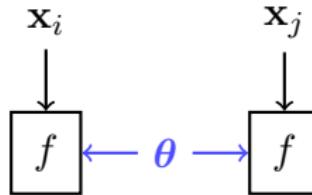
\mathbf{x}_i

\mathbf{x}_j

- an input sample is a pair $(\mathbf{x}_i, \mathbf{x}_j)$
- both $\mathbf{x}_i, \mathbf{x}_j$ go through the same function f with shared parameters θ
- loss ℓ_{ij} is measured on output pair $(\mathbf{y}_i, \mathbf{y}_j)$ and target t_{ij}

siamese architecture

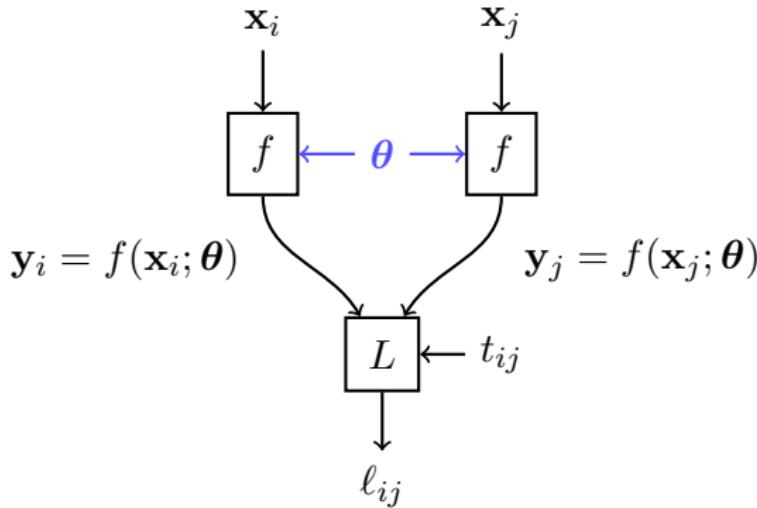
[Chopra et al. 2005]



- an input sample is a pair $(\mathbf{x}_i, \mathbf{x}_j)$
- both $\mathbf{x}_i, \mathbf{x}_j$ go through the same function f with shared parameters θ
- loss ℓ_{ij} is measured on output pair $(\mathbf{y}_i, \mathbf{y}_j)$ and target t_{ij}

siamese architecture

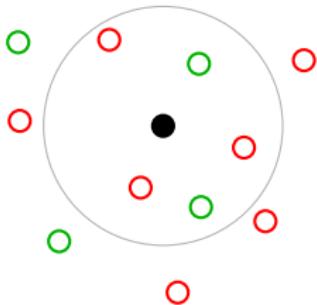
[Chopra et al. 2005]



- an input sample is a **pair** $(\mathbf{x}_i, \mathbf{x}_j)$
- both $\mathbf{x}_i, \mathbf{x}_j$ go through the **same** function f with **shared** parameters θ
- loss ℓ_{ij} is measured on output pair $(\mathbf{y}_i, \mathbf{y}_j)$ and target t_{ij}

contrastive loss

[Hadsel et al. 2006]



- input samples \mathbf{x}_i , output vectors $\mathbf{y}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$
- target variables $t_{ij} = \mathbb{1}[\text{sim}(\mathbf{x}_i, \mathbf{x}_j)]$
- **contrastive loss** is a function of distance $\|\mathbf{y}_i - \mathbf{y}_j\|$ only

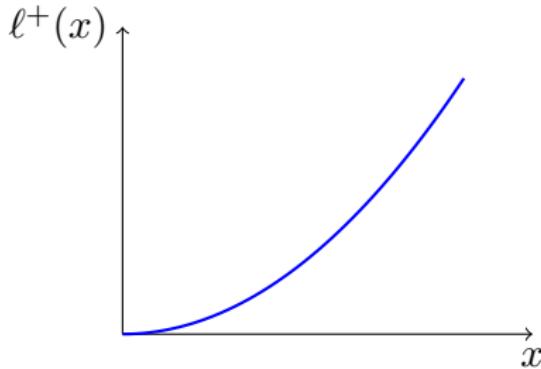
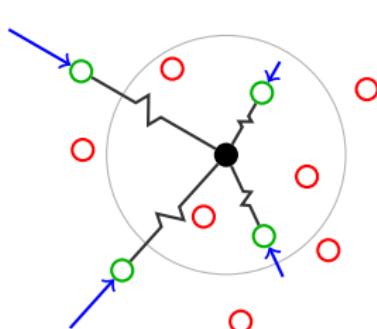
$$\ell_{ij} = L((\mathbf{y}_i, \mathbf{y}_j), t_{ij}) = \ell(\|\mathbf{y}_i - \mathbf{y}_j\|, t_{ij})$$

- similar samples are attracted

$$\ell(x, t) = t\ell^+(x) + (1-t)\ell^-(x) = tx^2 + (1-t)[m - x]_+^2$$

contrastive loss

[Hadsel et al. 2006]



- input samples \mathbf{x}_i , output vectors $\mathbf{y}_i = f(\mathbf{x}_i; \theta)$
- target variables $t_{ij} = \mathbb{1}[\text{sim}(\mathbf{x}_i, \mathbf{x}_j)]$
- **contrastive loss** is a function of distance $\|\mathbf{y}_i - \mathbf{y}_j\|$ only

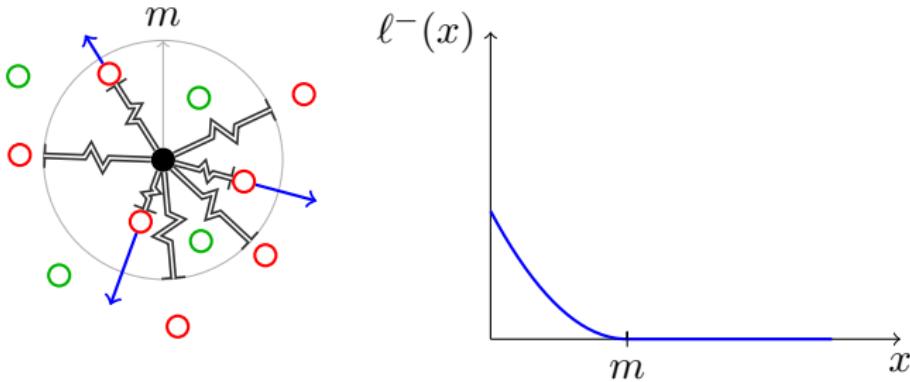
$$\ell_{ij} = L((\mathbf{y}_i, \mathbf{y}_j), t_{ij}) = \ell(\|\mathbf{y}_i - \mathbf{y}_j\|, t_{ij})$$

- similar samples are attracted

$$\ell(x, t) = t\ell^+(x) + (1-t)\ell^-(x) = tx^2 + (1-t)[m - x]_+^2$$

contrastive loss

[Hadsel et al. 2006]



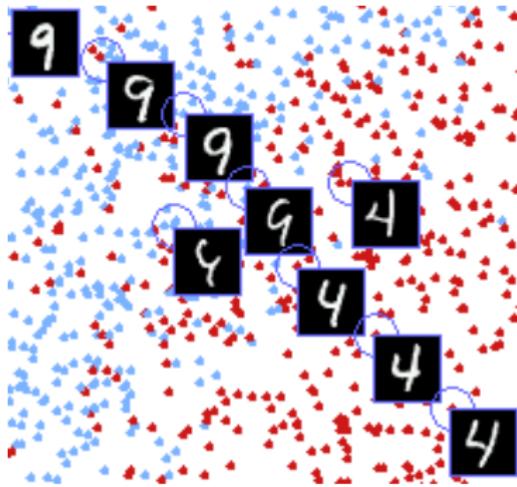
- input samples \mathbf{x}_i , output vectors $\mathbf{y}_i = f(\mathbf{x}_i; \theta)$
- target variables $t_{ij} = \mathbb{1}[\text{sim}(\mathbf{x}_i, \mathbf{x}_j)]$
- **contrastive loss** is a function of distance $\|\mathbf{y}_i - \mathbf{y}_j\|$ only

$$\ell_{ij} = L((\mathbf{y}_i, \mathbf{y}_j), t_{ij}) = \ell(\|\mathbf{y}_i - \mathbf{y}_j\|, t_{ij})$$

- dissimilar samples are repelled if closer than margin m

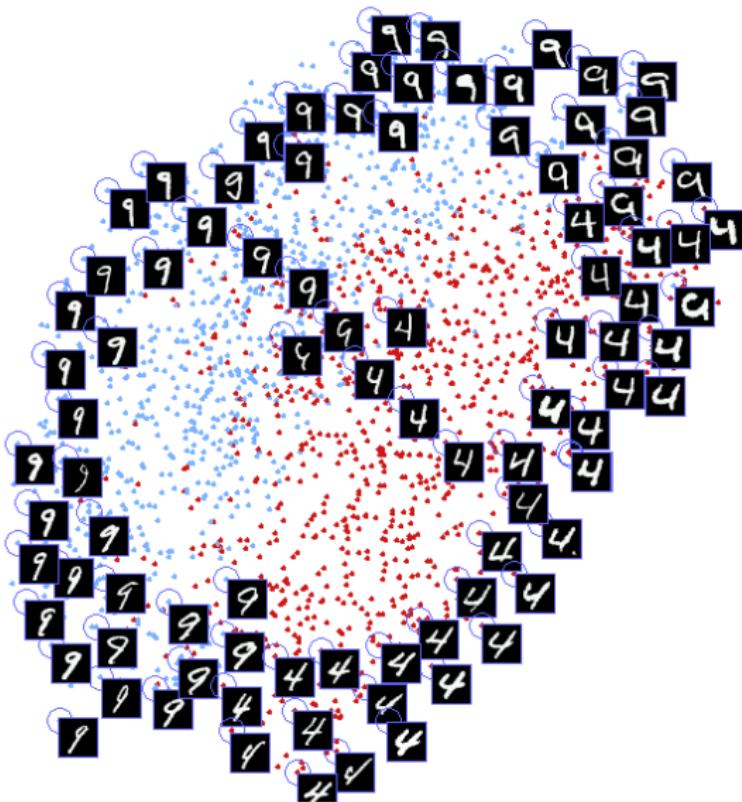
$$\ell(x, t) = t\ell^+(x) + (1-t)\ell^-(x) = tx^2 + (1-t)[m - x]_+^2$$

manifold learning: MNIST

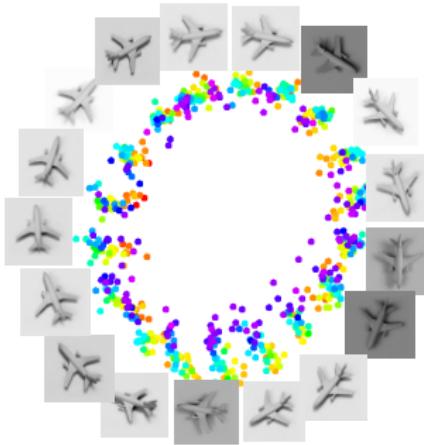


- 3k samples of each of digits 4, 9
- each sample similar to its 5 Euclidean nearest neighbors, and dissimilar to all other points
- 30k similar pairs, 18M dissimilar pairs

manifold learning: MNIST

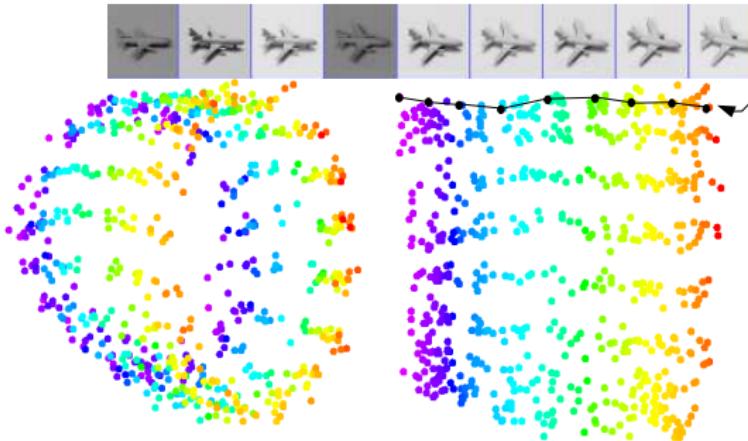


manifold learning: NORB



- 972 images of airplane class: 18 azimuths (every 20°), 9 elevations (in $[30^\circ, 70^\circ]$, every 5°), 6 lighting conditions
- samples similar if taken from contiguous azimuth or elevation, regardless of lighting
- 11k similar pairs, 206M dissimilar pairs
- cylinder in 3d: **azimuth on circumference, elevation on height**

manifold learning: NORB



- 972 images of airplane class: 18 azimuths (every 20°), 9 elevations (in $[30^\circ, 70^\circ]$, every 5°), 6 lighting conditions
- samples similar if taken from contiguous azimuth or elevation, regardless of lighting
- 11k similar pairs, 206M dissimilar pairs
- cylinder in 3d: azimuth on circumference, elevation on height

triplet architecture

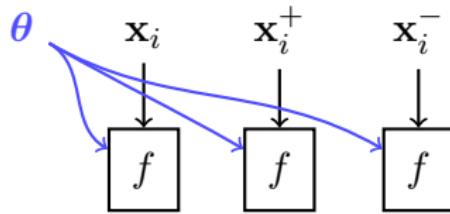
[Wang et al. 2014]

$$\mathbf{x}_i \quad \mathbf{x}_i^+ \quad \mathbf{x}_i^-$$

- an input sample is a **triplet** $(\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)$
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through the **same** function f with **shared** parameters θ
- loss ℓ_i measured on output triplet $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

triplet architecture

[Wang et al. 2014]

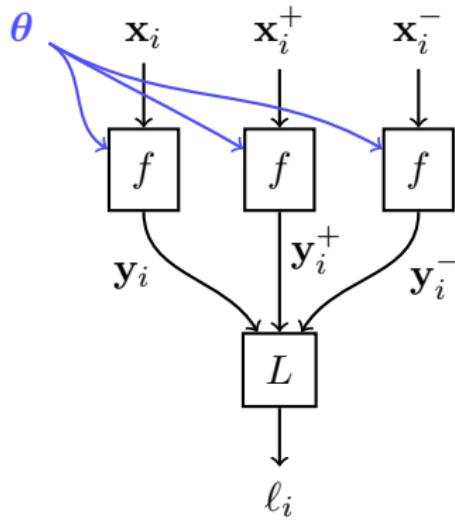


- an input sample is a **triplet** $(\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)$
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through the **same** function f with **shared** parameters θ
- loss ℓ_i measured on output triplet $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

Wang, Song, Leung, Rosenberg, Wang, Philbin, Chen, Wu. CVPR 2014. Learning Fine-Grained Image Similarity with Deep Ranking.

triplet architecture

[Wang et al. 2014]



- an input sample is a **triplet** $(\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-)$
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through the **same** function f with **shared** parameters θ
- loss ℓ_i measured on output triplet $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

triplet loss

- input “anchor” \mathbf{x}_i , output vector $\mathbf{y}_i = f(\mathbf{x}_i; \theta)$
- positive $\mathbf{y}_i^+ = f(\mathbf{x}_i^+; \theta)$, negative $\mathbf{y}_i^- = f(\mathbf{x}_i^-; \theta)$
- triplet loss is a function of distances $\|\mathbf{y}_i - \mathbf{y}_i^+\|, \|\mathbf{y}_i - \mathbf{y}_i^-\|$ only

$$\ell_i = L(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-) = \ell(\|\mathbf{y}_i - \mathbf{y}_i^+\|, \|\mathbf{y}_i - \mathbf{y}_i^-\|)$$

$$\ell(x^+, x^-) = [m + (x^+)^2 - (x^-)^2]_+$$

so distance $\|\mathbf{y}_i - \mathbf{y}_i^+\|$ should be less than $\|\mathbf{y}_i - \mathbf{y}_i^-\|$ by margin m

- by taking two pairs $(\mathbf{x}_i, \mathbf{x}_i^+)$ and $(\mathbf{x}_i, \mathbf{x}_i^-)$ at a time with targets 1, 0 respectively, the contrastive loss can be written similarly

$$\ell(x^+, x^-) = (x^+)^2 + [m - x^-]^2_+$$

so distance $\|\mathbf{y}_i - \mathbf{y}_i^+\|$ should small and $\|\mathbf{y}_i - \mathbf{y}_i^-\|$ larger than m

triplet loss

- input “anchor” \mathbf{x}_i , output vector $\mathbf{y}_i = f(\mathbf{x}_i; \theta)$
- positive $\mathbf{y}_i^+ = f(\mathbf{x}_i^+; \theta)$, negative $\mathbf{y}_i^- = f(\mathbf{x}_i^-; \theta)$
- triplet loss is a function of distances $\|\mathbf{y}_i - \mathbf{y}_i^+\|, \|\mathbf{y}_i - \mathbf{y}_i^-\|$ only

$$\ell_i = L(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-) = \ell(\|\mathbf{y}_i - \mathbf{y}_i^+\|, \|\mathbf{y}_i - \mathbf{y}_i^-\|)$$

$$\ell(x^+, x^-) = [m + (x^+)^2 - (x^-)^2]_+$$

so distance $\|\mathbf{y}_i - \mathbf{y}_i^+\|$ should be less than $\|\mathbf{y}_i - \mathbf{y}_i^-\|$ by margin m

- by taking two pairs $(\mathbf{x}_i, \mathbf{x}_i^+)$ and $(\mathbf{x}_i, \mathbf{x}_i^-)$ at a time with targets 1, 0 respectively, the contrastive loss can be written similarly

$$\ell(x^+, x^-) = (x^+)^2 + [m - x^-]^2_+$$

so distance $\|\mathbf{y}_i - \mathbf{y}_i^+\|$ should small and $\|\mathbf{y}_i - \mathbf{y}_i^-\|$ larger than m

unsupervised learning by context prediction: task

[Doersch et al. 2015]

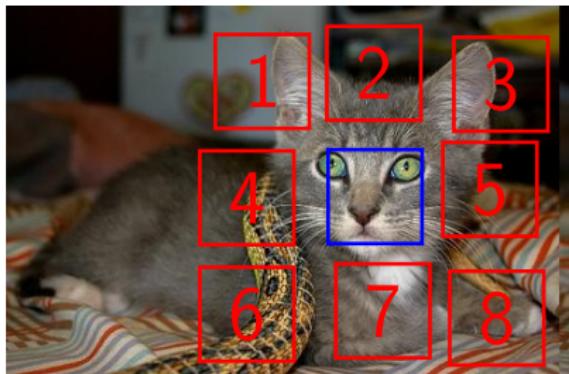


- sample random pairs of patches in one of eight spatial configurations
- patches are randomly jittered and do not overlap
- like **solving a puzzle**, learn to predict the relative position

$$f\left(\quad, \quad \right) = 3$$

unsupervised learning by context prediction: task

[Doersch et al. 2015]

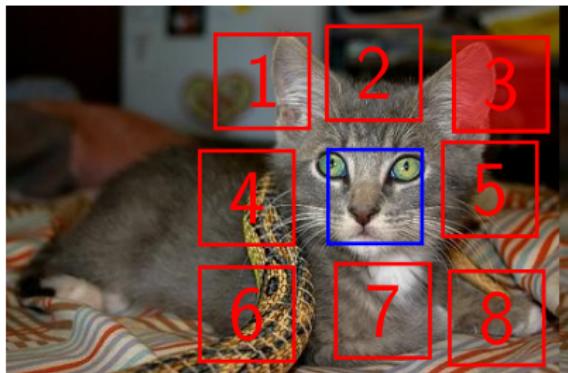


- sample random pairs of patches in one of eight spatial configurations
- patches are randomly jittered and do not overlap
- like *solving a puzzle*, learn to predict the relative position

$$f\left(\quad, \quad \right) = 3$$

unsupervised learning by context prediction: task

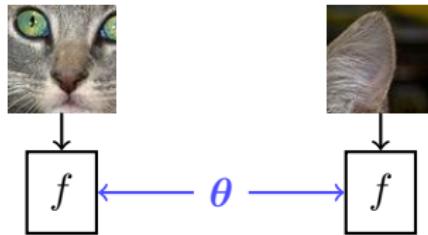
[Doersch et al. 2015]



- sample random pairs of patches in one of eight spatial configurations
- patches are randomly jittered and do not overlap
- like **solving a puzzle**, learn to predict the relative position

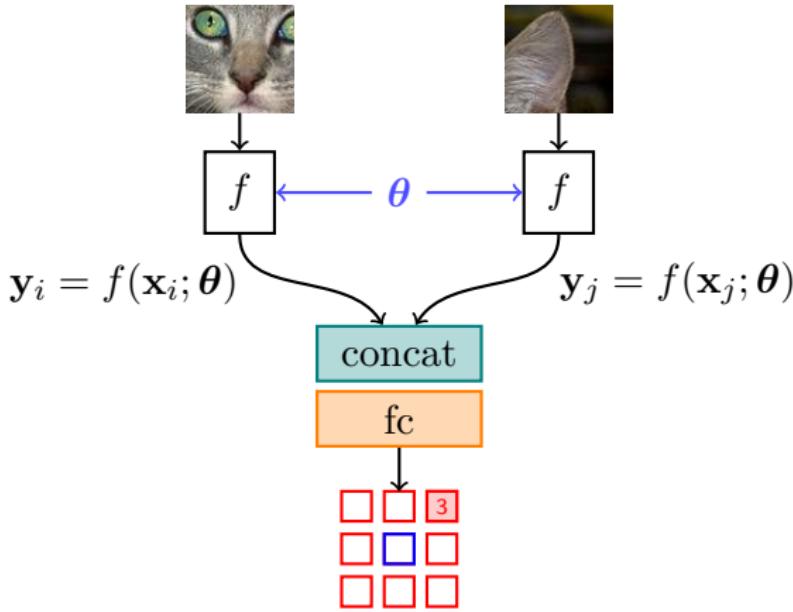
$$f\left(\begin{array}{c} \text{[Patch 1]} \\ , \\ \text{[Patch 2]} \end{array}\right) = 3$$

unsupervised learning by context prediction: architecture



- network f learned by siamese architecture
- representations are concatenated and followed by softmax classifier, where each spatial configuration is a class

unsupervised learning by context prediction: architecture



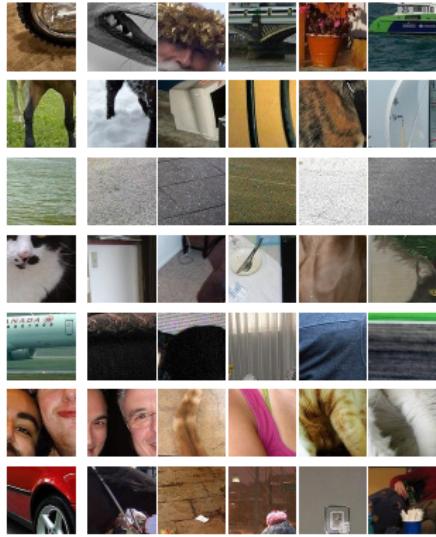
- network f learned by siamese architecture
 - representations are concatenated and followed by softmax classifier, where each spatial configuration is a class

unsupervised learning by context prediction: examples



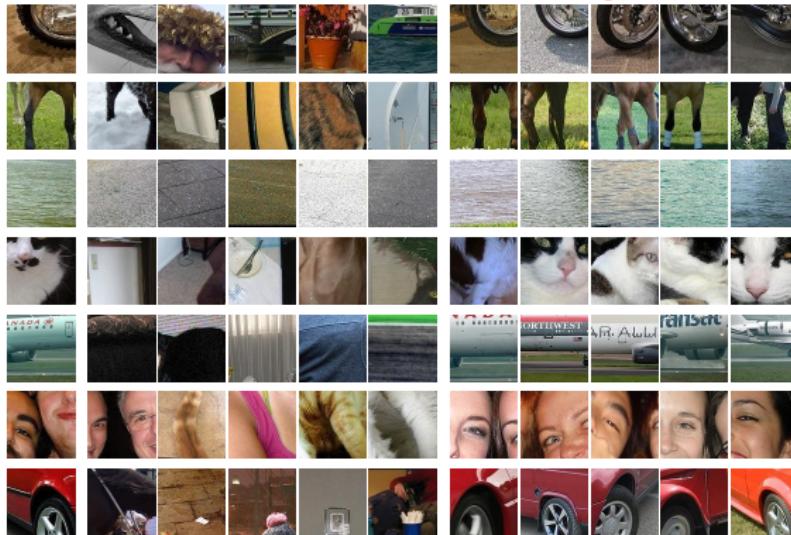
- input image
- nearest neighbors with randomly initialized network
- trained by supervised classification on ImageNet
- unsupervised training from scratch on the context prediction task

unsupervised learning by context prediction: examples



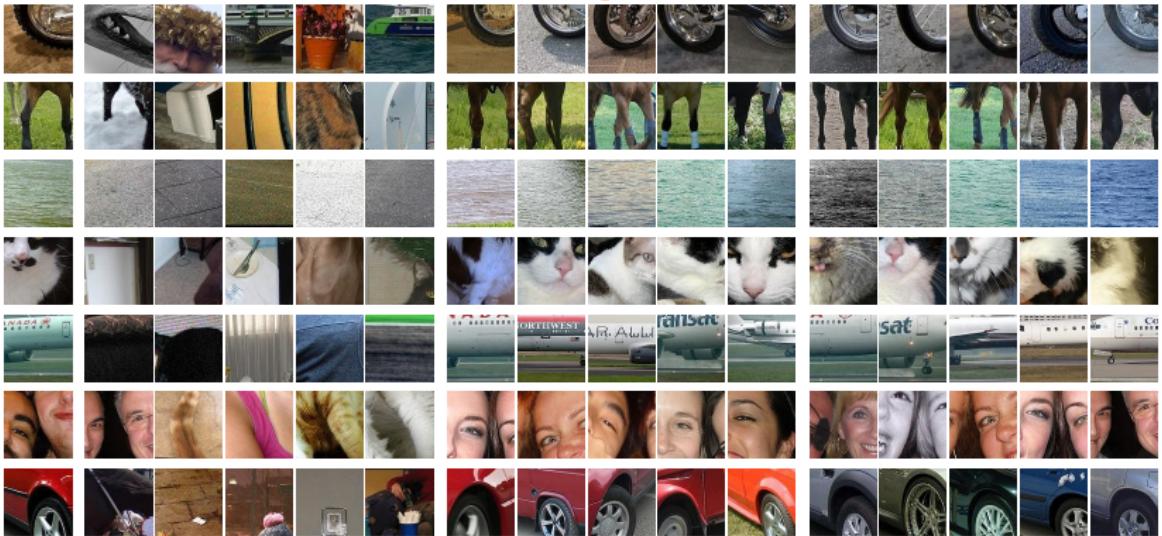
- input image
- nearest neighbors with randomly initialized network
- trained by supervised classification on ImageNet
- unsupervised training from scratch on the context prediction task

unsupervised learning by context prediction: examples



- input image
- nearest neighbors with randomly initialized network
- trained by supervised classification on ImageNet
- unsupervised training from scratch on the context prediction task

unsupervised learning by context prediction: examples



- input image
 - nearest neighbors with randomly initialized network
 - trained by supervised classification on ImageNet
 - unsupervised training from scratch on the context prediction task

unsupervised learning on video: tracking

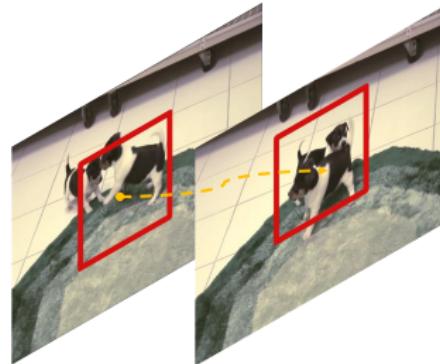
[Wang et al. 2015]



- estimate motion and find the region that contains most motion
- track this region for a number of frames
- generate a pair of matching patches on the first and last frames

unsupervised learning on video: tracking

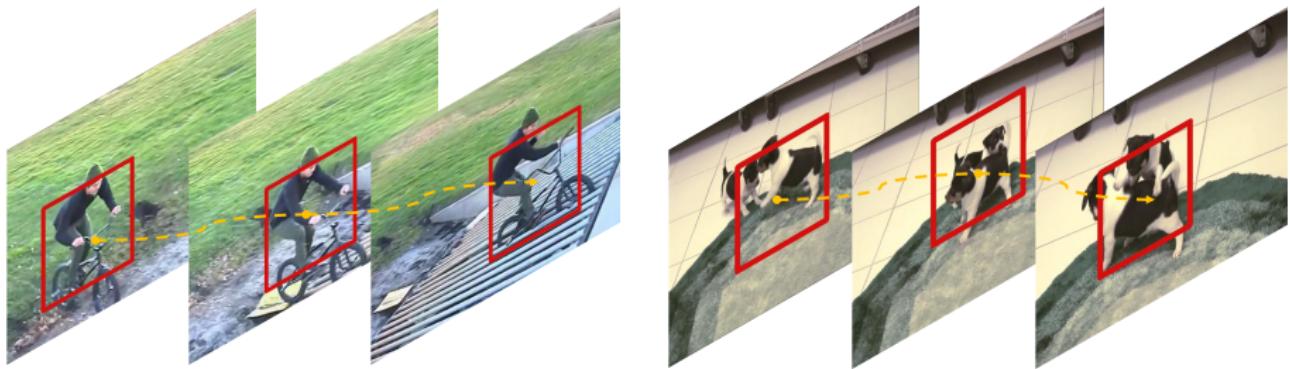
[Wang et al. 2015]



- estimate motion and find the region that contains most motion
- track this region for a number of frames
- generate a pair of matching patches on the first and last frames

unsupervised learning on video: tracking

[Wang et al. 2015]



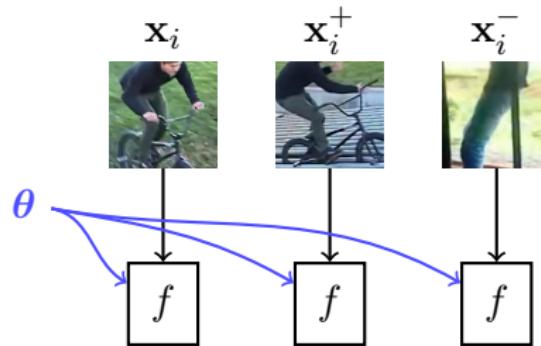
- estimate motion and find the region that contains most motion
 - track this region for a number of frames
 - generate a pair of matching patches on the first and last frames

unsupervised learning on video: architecture



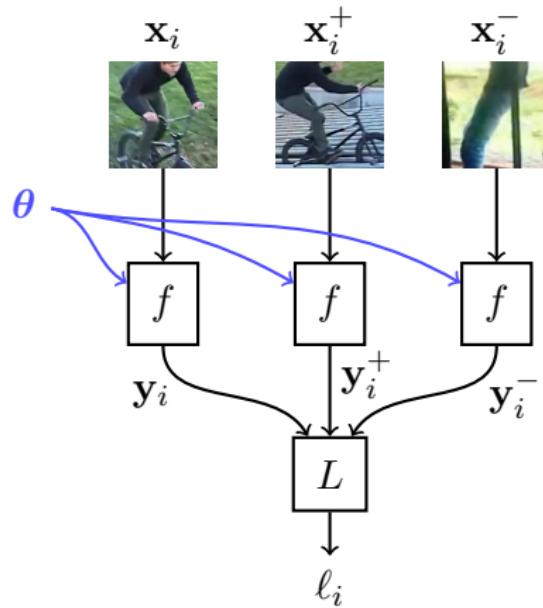
- input query \mathbf{x}_i (first frame), tracked \mathbf{x}_i^+ (last frame), random \mathbf{x}_i^-
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through the same function f with shared parameters θ
- triplet loss ℓ_i measured on output triplet $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

unsupervised learning on video: architecture



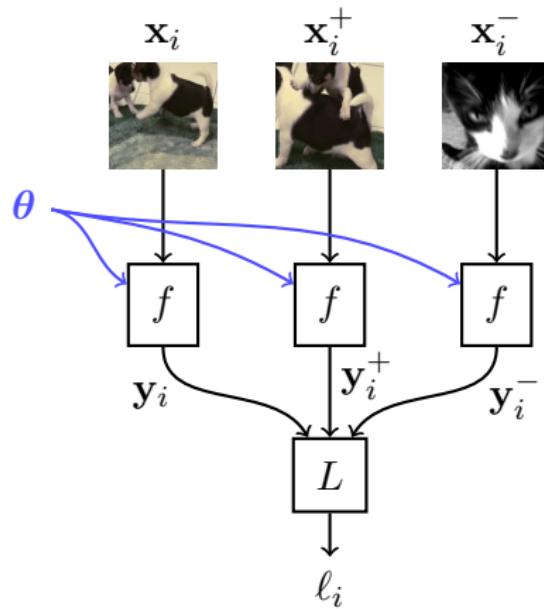
- input query \mathbf{x}_i (first frame), tracked \mathbf{x}_i^+ (last frame), random \mathbf{x}_i^-
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through the **same** function f with **shared** parameters θ
- triplet loss ℓ_i measured on output triplet $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

unsupervised learning on video: architecture



- input query \mathbf{x}_i (first frame), tracked \mathbf{x}_i^+ (last frame), random \mathbf{x}_i^-
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through the **same** function f with **shared** parameters θ
- **triplet loss** ℓ_i measured on output triplet $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

unsupervised learning on video: architecture



- input query \mathbf{x}_i (first frame), tracked \mathbf{x}_i^+ (last frame), random \mathbf{x}_i^-
 - $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through the **same** function f with **shared** parameters θ
 - **triplet loss** ℓ_i measured on output triplet $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

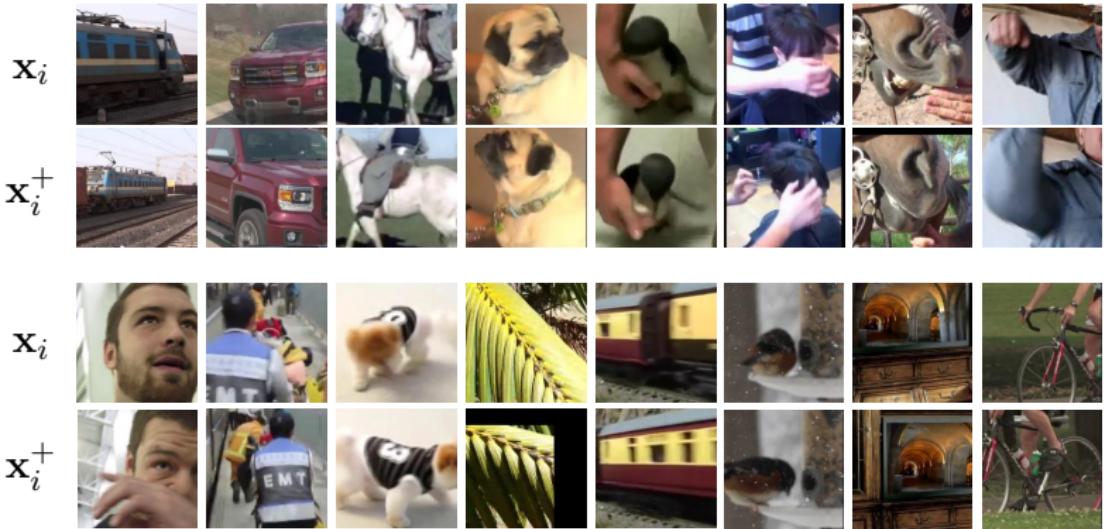
unsupervised learning on video: objective

$$\left\| f\left(\begin{array}{c} \text{person on bike} \\ \text{person on bike} \end{array} \right), f\left(\begin{array}{c} \text{person on bike} \\ \text{person on grass} \end{array} \right) \right\|^2 < \left\| f\left(\begin{array}{c} \text{person on bike} \\ \text{person on grass} \end{array} \right), f\left(\begin{array}{c} \text{person on bike} \\ \text{person on grass} \end{array} \right) \right\|^2 - m$$

$$\left\| f\left(\begin{array}{c} \text{dog} \\ \text{dog} \end{array} \right), f\left(\begin{array}{c} \text{dog} \\ \text{cat} \end{array} \right) \right\|^2 < \left\| f\left(\begin{array}{c} \text{dog} \\ \text{cat} \end{array} \right), f\left(\begin{array}{c} \text{dog} \\ \text{cat} \end{array} \right) \right\|^2 - m$$

- so, the objective is that squared distance $\|\mathbf{y}_i^+ - \mathbf{y}_i^+\|^2$ is less than $\|\mathbf{y}_i^- - \mathbf{y}_i^-\|^2$ by margin m

unsupervised learning on video: more examples

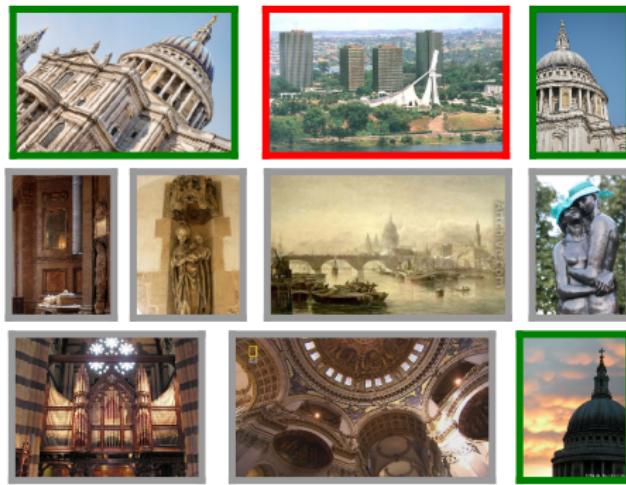


- input query x_i (first frame), tracked x_i^+ (last frame)

fine-tuning

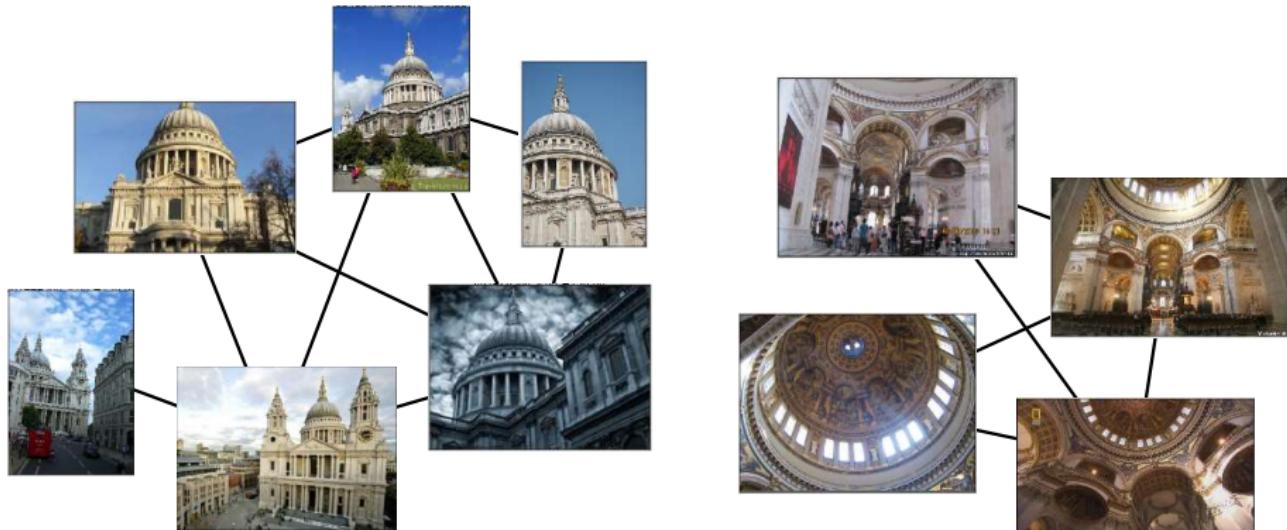
deep image retrieval: dataset cleaning

[Gordo et al. 2016]



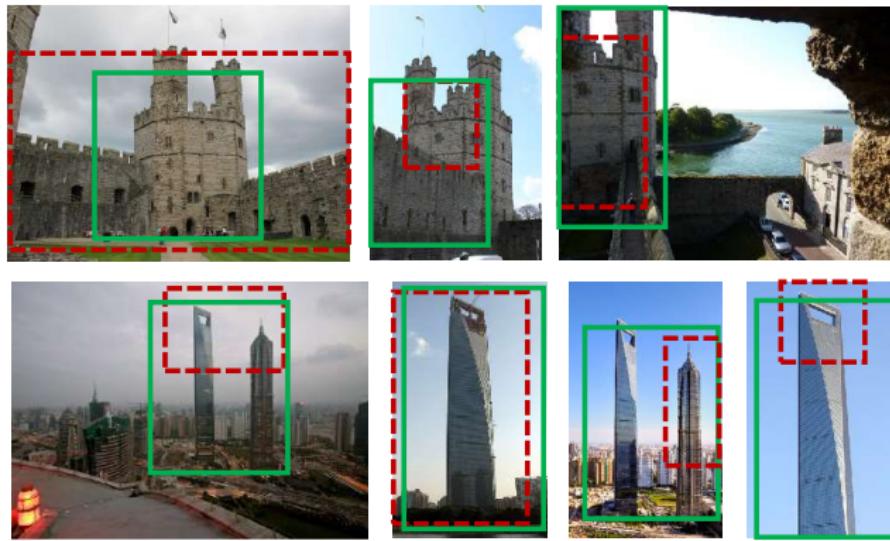
- start from landmark dataset (192k images) and **clean** it (49k images)
- use it to fine-tune a network pre-trained on ImageNet for classification
- **prototypical, non-prototypical** and incorrect images per class
- only prototypical are kept to reduce intra-class variability

deep image retrieval: prototypical views



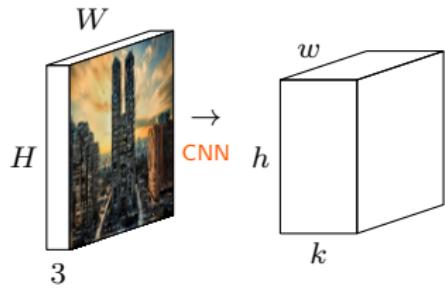
- pairwise match images per class by SIFT descriptors and fast spatial matching
- connect images into a graph and compute the connected components
- keep only the largest component

deep image retrieval: bounding boxes



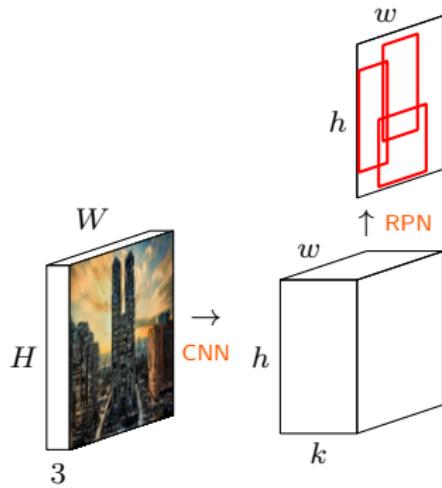
- automatically find object bounding boxes
 - initialize with inlier features per image
 - update such that boxes are consistent over all matching pairs
- use bounding boxes to train a region proposal network

deep image retrieval: network, regions, pooling



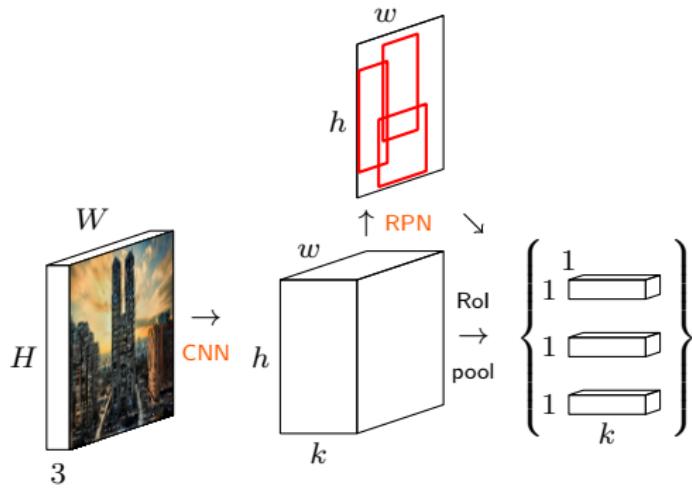
- VGG-16 or ResNet-101 feature maps
- proposals detected on feature maps by RPN and max-pooled
- ℓ_2 -normalization, PCA-whitening (FC layer), ℓ_2 -normalization
- sum-pooling, ℓ_2 -normalization (as in R-MAC)

deep image retrieval: network, regions, pooling



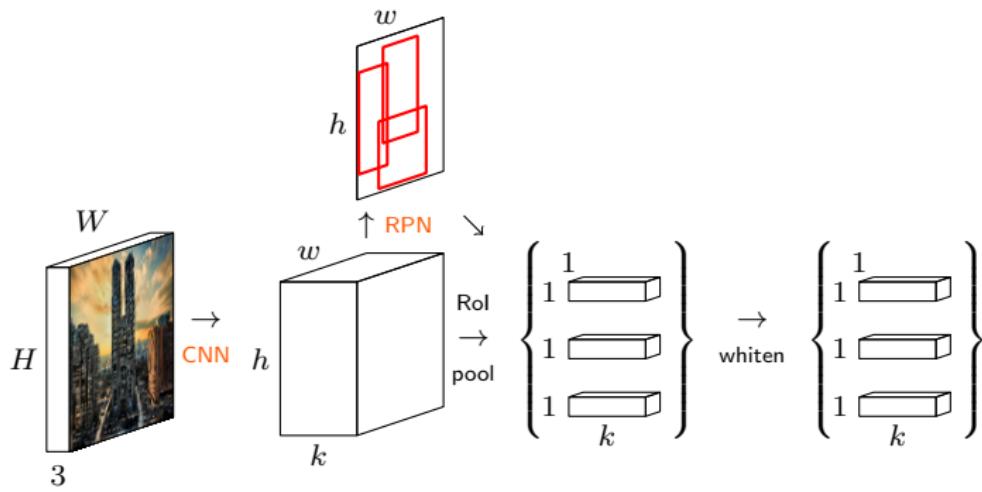
- VGG-16 or ResNet-101 feature maps
 - proposals detected on feature maps by **RPN** and **max**-pooled
 - ℓ_2 -normalization, PCA-whitening (**FC layer**), ℓ_2 -normalization
 - **sum**-pooling, ℓ_2 -normalization (as in R-MAC)

deep image retrieval: network, regions, pooling



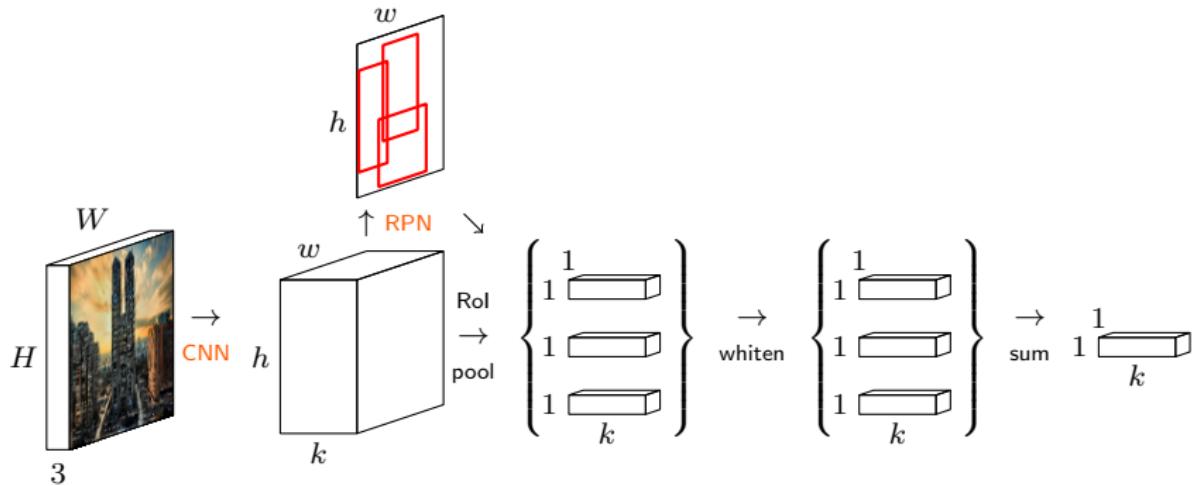
- VGG-16 or ResNet-101 feature maps
 - proposals detected on feature maps by **RPN** and **max**-pooled
 - ℓ_2 -normalization, PCA-whitening (**FC layer**), ℓ_2 -normalization
 - **sum**-pooling, ℓ_2 -normalization (as in R-MAC)

deep image retrieval: network, regions, pooling



- VGG-16 or ResNet-101 feature maps
 - proposals detected on feature maps by **RPN** and **max-pooled**
 - ℓ_2 -normalization, PCA-whitening (**FC layer**), ℓ_2 -normalization
 - **sum-pooling**, ℓ_2 -normalization (as in R-MAC)

deep image retrieval: network, regions, pooling



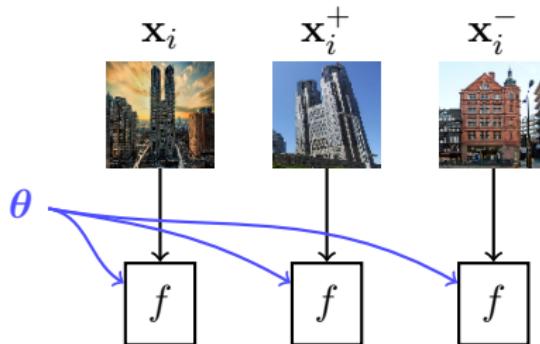
- VGG-16 or ResNet-101 feature maps
- proposals detected on feature maps by **RPN** and **max**-pooled
- ℓ_2 -normalization, PCA-whitening (**FC layer**), ℓ_2 -normalization
- **sum**-pooling, ℓ_2 -normalization (as in R-MAC)

deep image retrieval: architecture



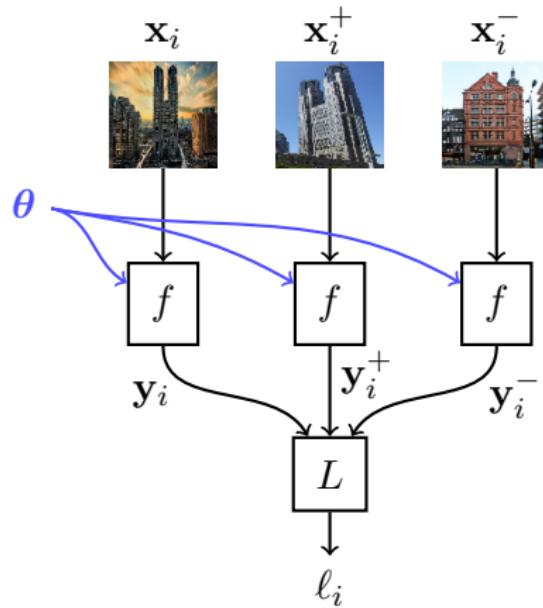
- query \mathbf{x}_i , relevant \mathbf{x}_i^+ (same building), irrelevant \mathbf{x}_i^- (other building)
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through function f including features, RPN, pooling
- triplet loss ℓ_i measured on output $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

deep image retrieval: architecture



- query \mathbf{x}_i , relevant \mathbf{x}_i^+ (same building), irrelevant \mathbf{x}_i^- (other building)
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through function f including features, RPN, pooling
- triplet loss ℓ_i measured on output $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

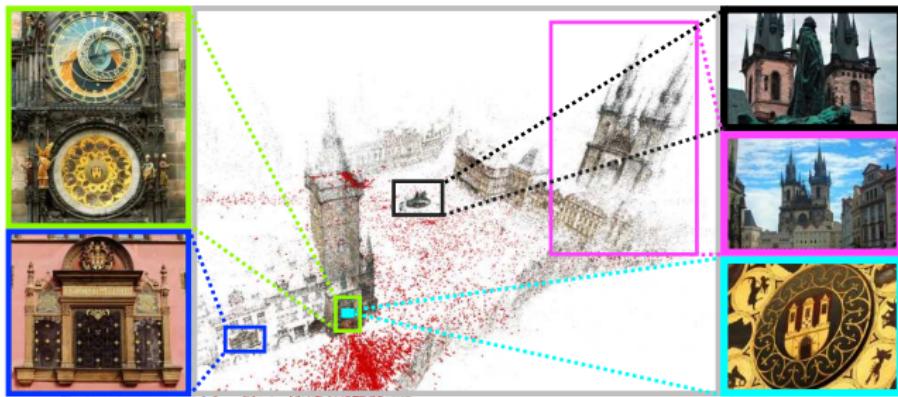
deep image retrieval: architecture



- query \mathbf{x}_i , relevant \mathbf{x}_i^+ (same building), irrelevant \mathbf{x}_i^- (other building)
- $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ go through function f including features, RPN, pooling
- **triplet loss** ℓ_i measured on output $(\mathbf{y}_i, \mathbf{y}_i^+, \mathbf{y}_i^-)$

learning from bag-of-words: 3d reconstruction

[Radenovic et al. 2016]



- start from an independent dataset of 7.4M images, no class labels
- clustering, pairwise matching and reconstruction of 713 3d models containing 165k unique images
- 3d models playing the same role as classes in deep image retrieval
- again, fine-tune a network pre-trained on ImageNet for classification

Radenovic, Tolias, Chum. ECCV 2016. CNN Image Retrieval Learns From BoW: Unsupervised Fine-Tuning with Hard Examples.
Schonberger, Radenovic, Chum and Frahm. CVPR 2015. From Single Image Query to Detailed 3D Reconstruction.

learning from bag-of-words: positive pairs



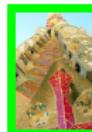
- input query
- positive images found in **same model** by minimum MAC distance
maximum inliers, or drawn at random from images having at least a given number of inliers (more challenging)

learning from bag-of-words: positive pairs



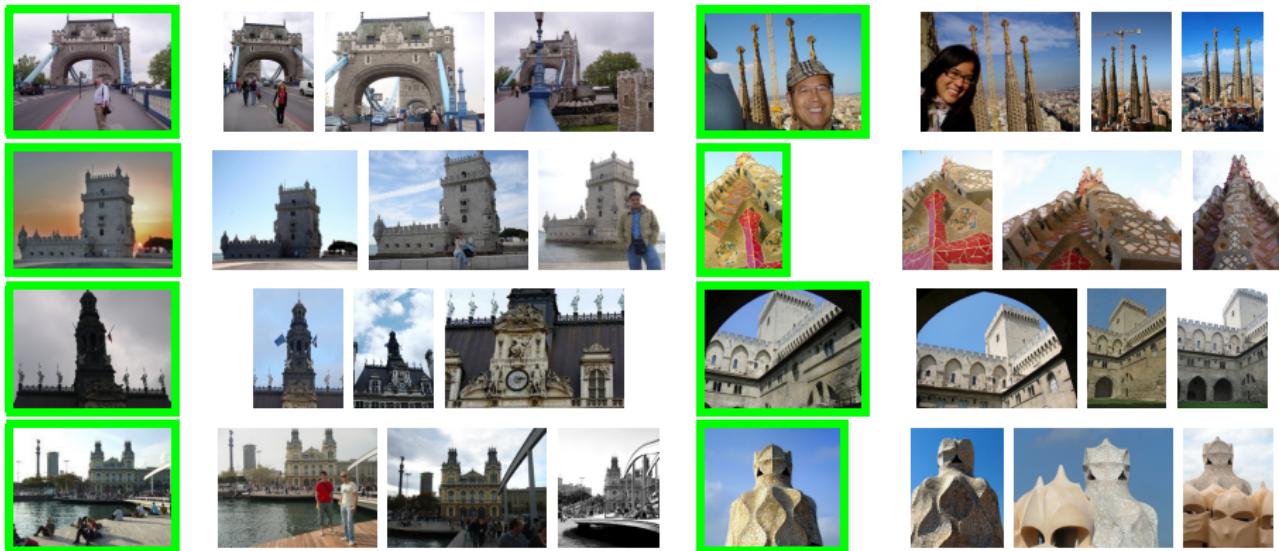
- input query
- positive images found in **same model** by minimum MAC distance, maximum inliers, or drawn at random from images having at least a given number of inliers (more challenging)

learning from bag-of-words: positive pairs



- input query
- positive images found in **same model** by minimum MAC distance, **maximum inliers**, or drawn at random from images having at least a given number of inliers (more challenging)

learning from bag-of-words: positive pairs



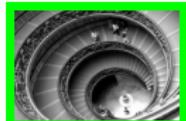
- input query
 - positive images found in **same model** by minimum MAC distance, maximum inliers, or drawn at random from images having at least a given number of inliers (more challenging)

learning from bag-of-words: negative pairs



- input query
- negative images found in different models
- **hard negatives** are most similar to query, *i.e.* with minimum MAC distance
- hardest negative, nearest neighbors from all other models, or nearest neighbors, one per model (higher variability)

learning from bag-of-words: negative pairs



- input query
- negative images found in different models
- **hard negatives** are most similar to query, *i.e.* with minimum MAC distance
- **hardest negative**, nearest neighbors from all other models, or nearest neighbors, one per model (higher variability)

learning from bag-of-words: negative pairs



- input query
- negative images found in **different models**
- **hard negatives** are most similar to query, *i.e.* with minimum MAC distance
- hardest negative, nearest neighbors from all other models, or nearest neighbors, one per model (higher variability)

learning from bag-of-words: negative pairs



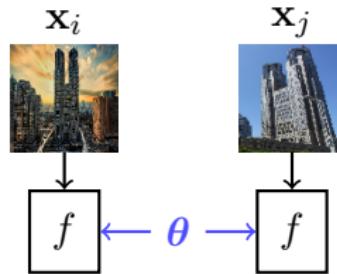
- input query
- negative images found in **different models**
- **hard negatives** are most similar to query, *i.e.* with minimum MAC distance
- hardest negative, nearest neighbors from all other models, or nearest neighbors, one per model (higher variability)

learning from bag-of-words: architecture



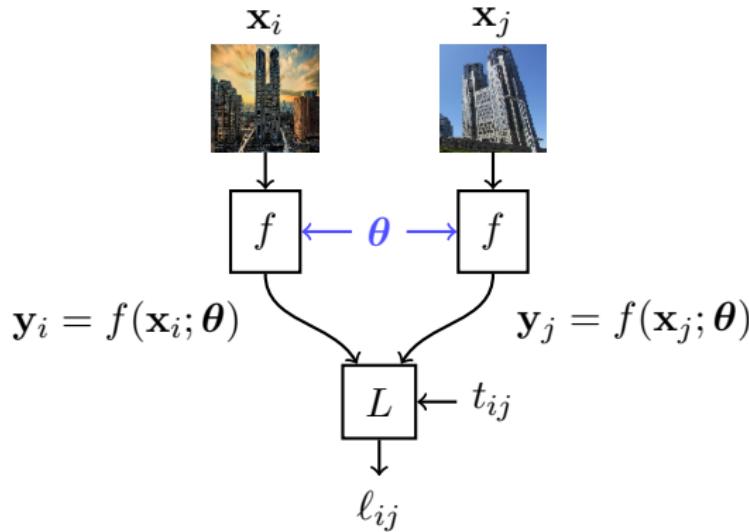
- input $(\mathbf{x}_i, \mathbf{x}_j)$ of relevant or irrelevant images
- both $\mathbf{x}_i, \mathbf{x}_j$ go through function f including features and MAC pooling
- **contrastive loss** ℓ_{ij} measured on output $(\mathbf{y}_i, \mathbf{y}_j)$ and target t_{ij}

learning from bag-of-words: architecture



- input $(\mathbf{x}_i, \mathbf{x}_j)$ of relevant or irrelevant images
- both $\mathbf{x}_i, \mathbf{x}_j$ go through function f including features and MAC pooling
- **contrastive loss** ℓ_{ij} measured on output $(\mathbf{y}_i, \mathbf{y}_j)$ and target t_{ij}

learning from bag-of-words: architecture



- input $(\mathbf{x}_i, \mathbf{x}_j)$ of relevant or irrelevant images
- both $\mathbf{x}_i, \mathbf{x}_j$ go through function f including features and MAC pooling
- **contrastive loss** ℓ_{ij} measured on output $(\mathbf{y}_i, \mathbf{y}_j)$ and target t_{ij}

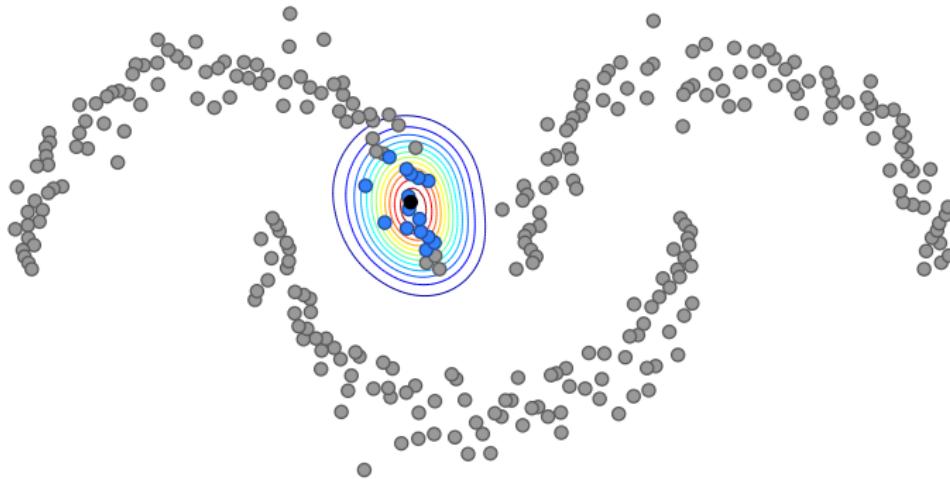
graph-based methods

ranking on manifolds: single query



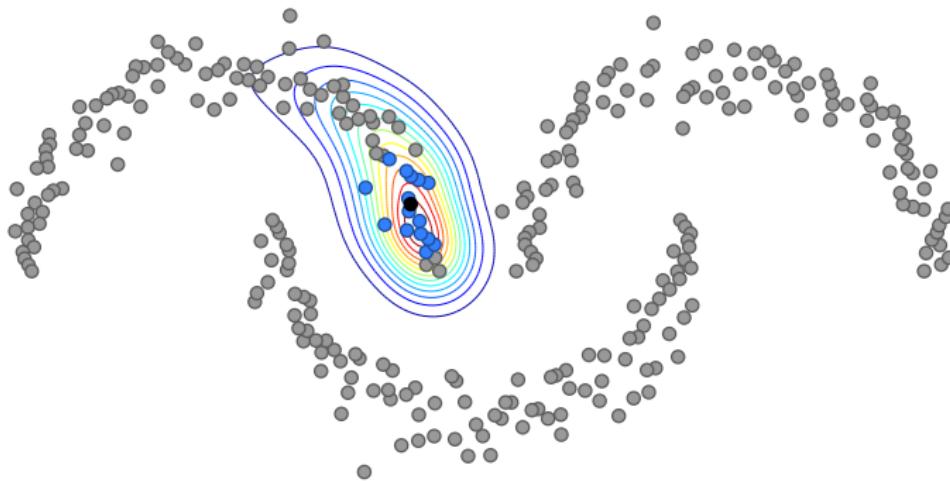
- data points (●), query point (○), nearest neighbors (●)
- iteration $\times 30$

ranking on manifolds: single query



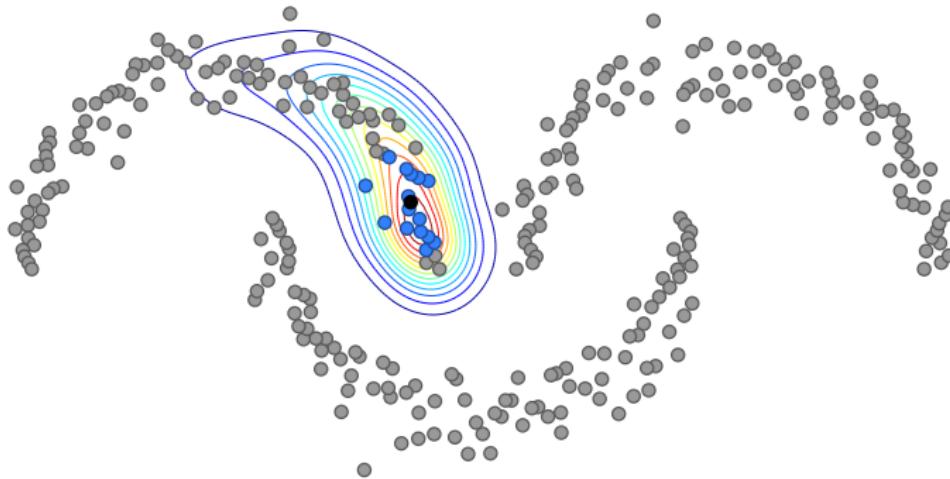
- data points (●), query point (•), nearest neighbors (○)
- iteration 0 × 30

ranking on manifolds: single query



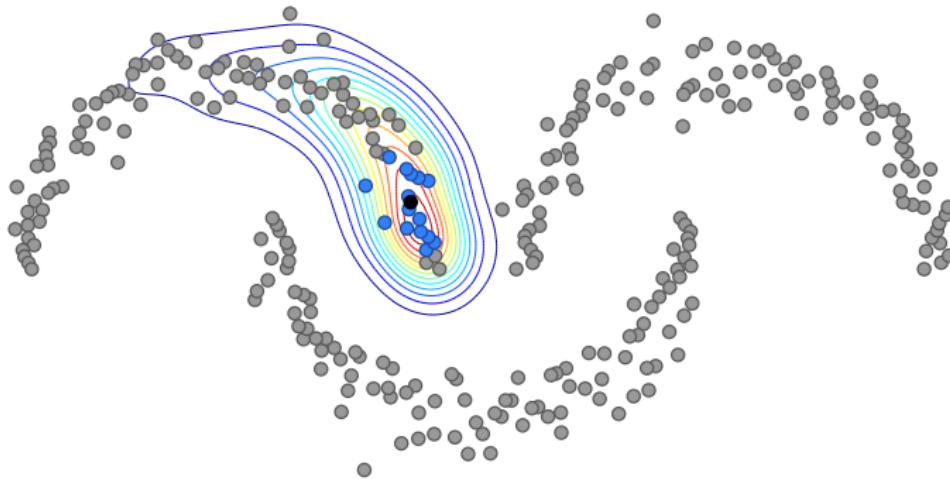
- data points (●), query point (•), nearest neighbors (○)
- iteration 1×30

ranking on manifolds: single query



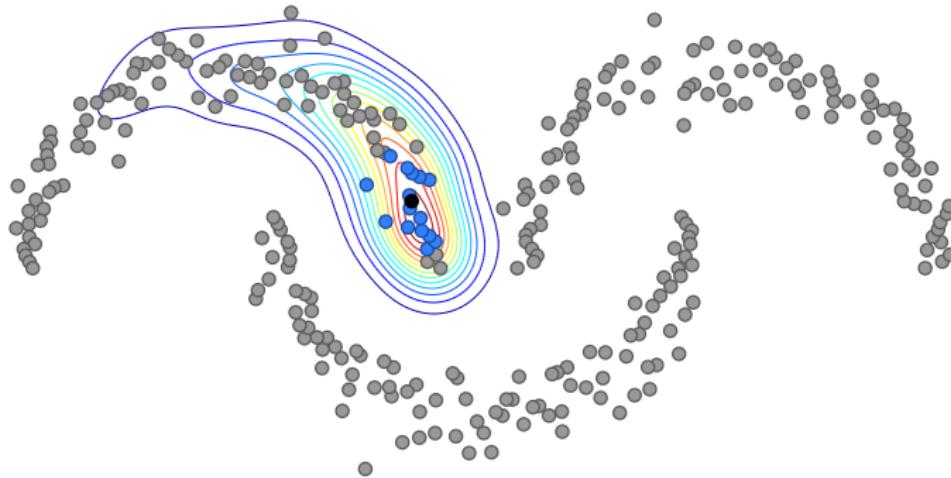
- data points (●), query point (•), nearest neighbors (○)
- iteration 2×30

ranking on manifolds: single query



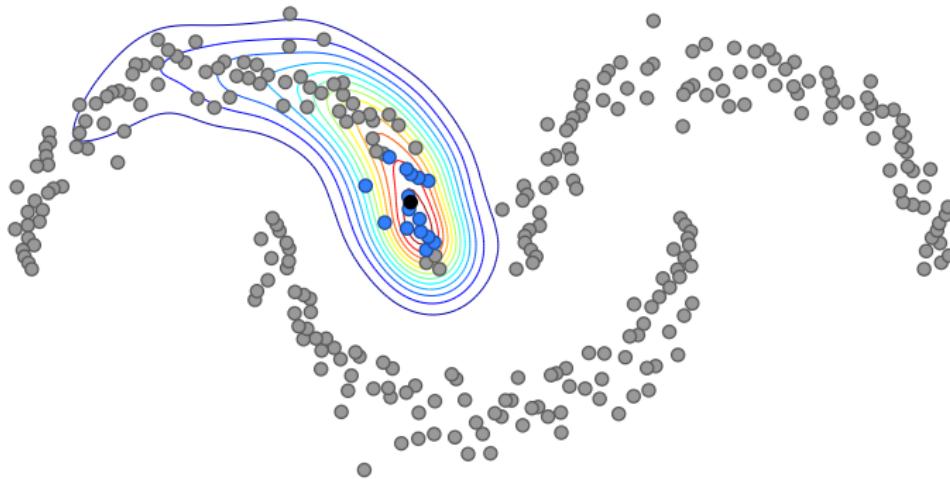
- data points (●), query point (•), nearest neighbors (○)
- iteration 3×30

ranking on manifolds: single query



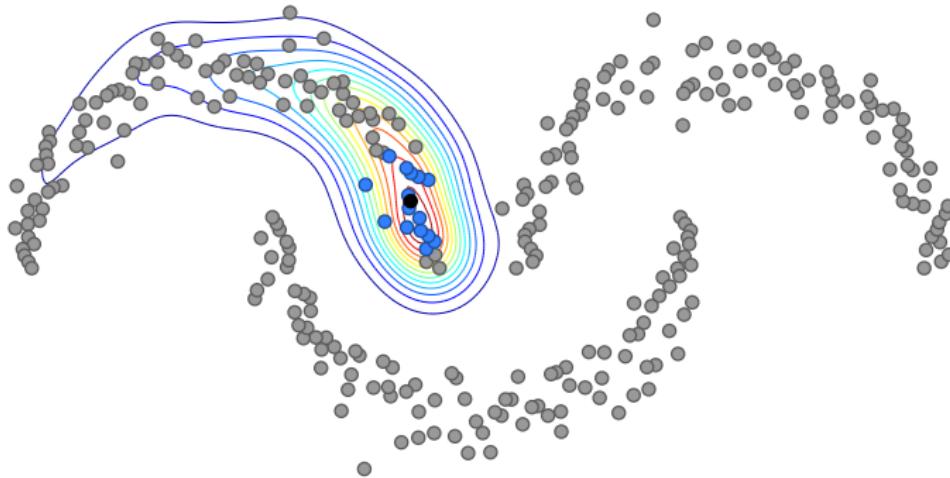
- data points (•), query point (•), nearest neighbors (•)
- iteration 4×30

ranking on manifolds: single query



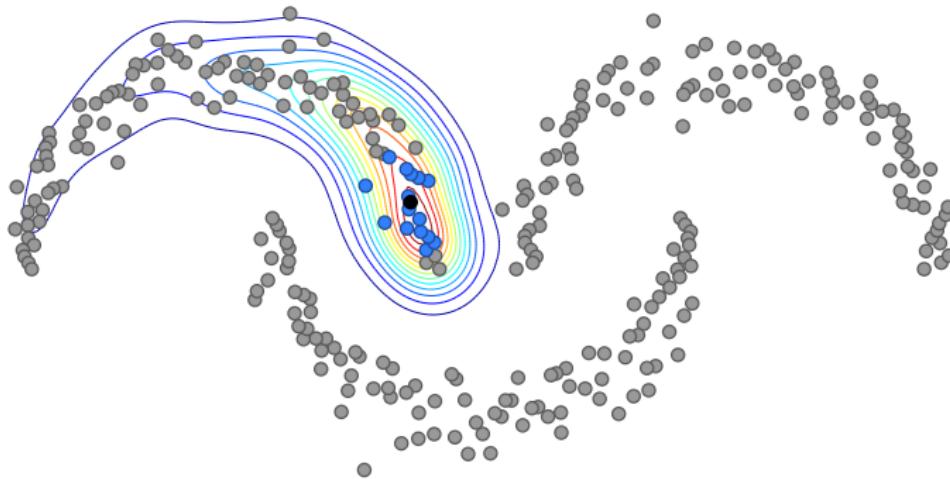
- data points (●), query point (•), nearest neighbors (○)
- iteration 5×30

ranking on manifolds: single query



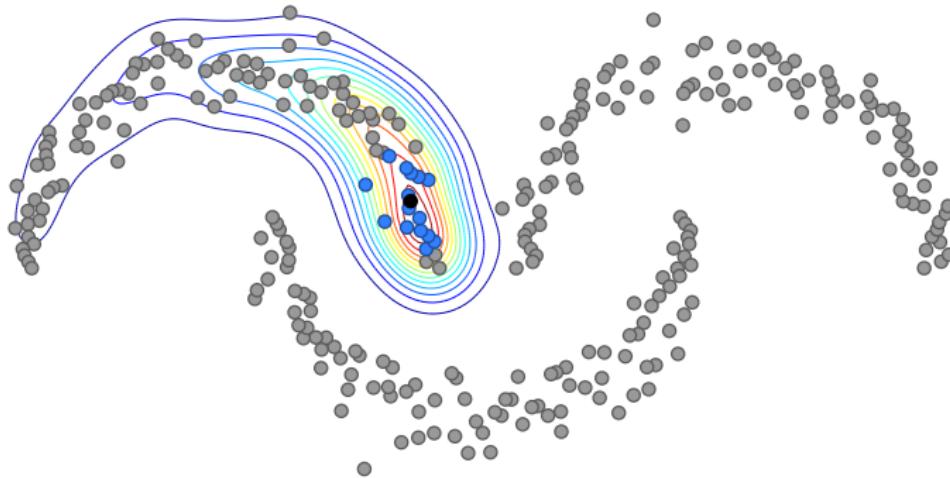
- data points (●), query point (•), nearest neighbors (○)
- iteration 6×30

ranking on manifolds: single query



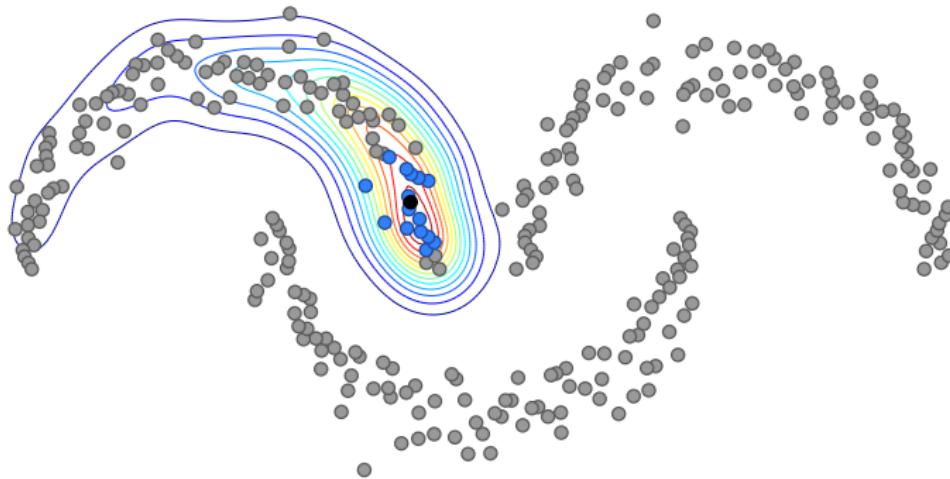
- data points (●), query point (•), nearest neighbors (○)
- iteration 7×30

ranking on manifolds: single query



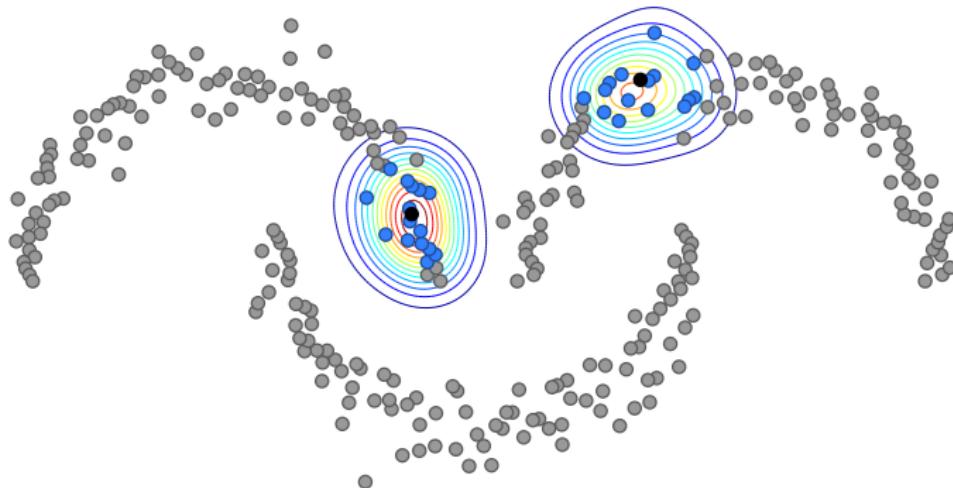
- data points (●), query point (•), nearest neighbors (○)
- iteration 8×30

ranking on manifolds: single query



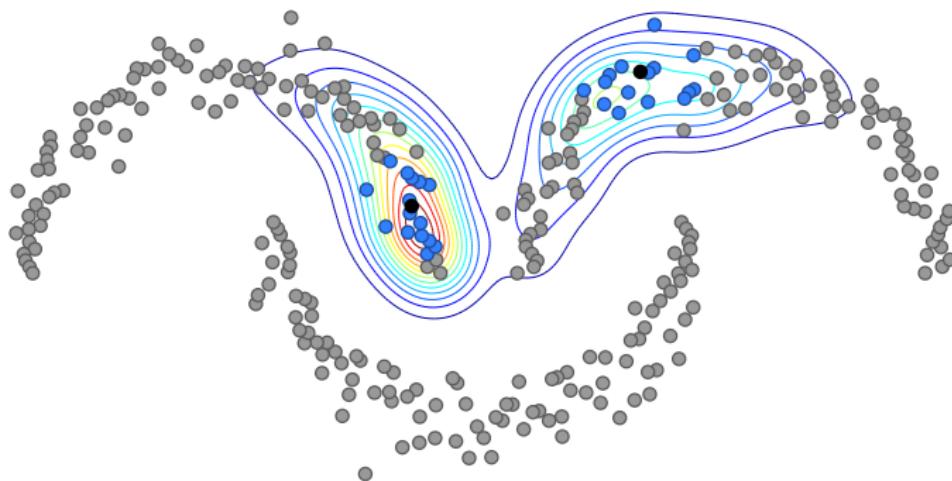
- data points (●), query point (•), nearest neighbors (○)
- iteration 9×30

ranking on manifolds: multiple queries



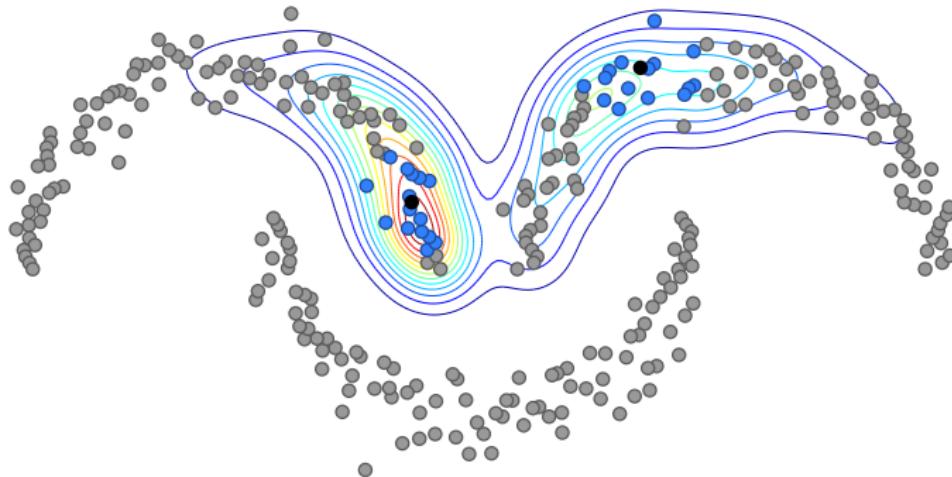
- data points (●), query points (●), nearest neighbors (○)
- iteration 0 × 30

ranking on manifolds: multiple queries



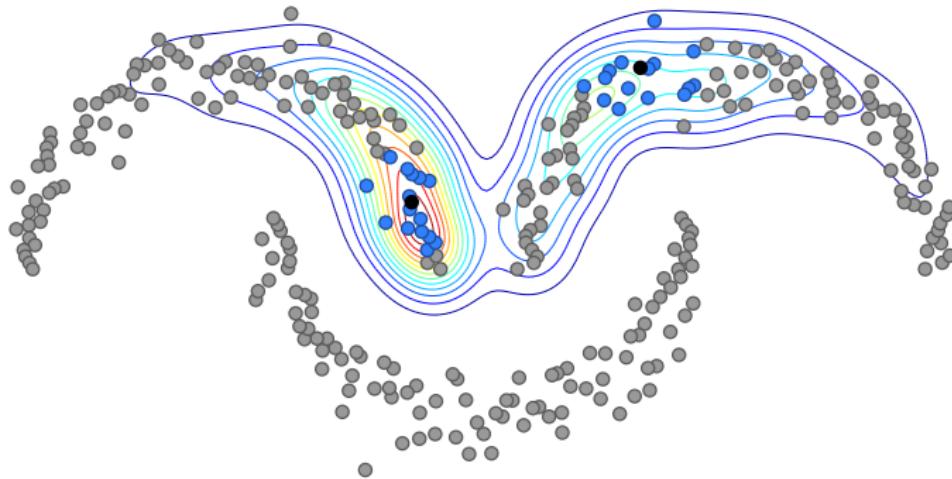
- data points (•), query points (●), nearest neighbors (○)
- iteration 1×30

ranking on manifolds: multiple queries



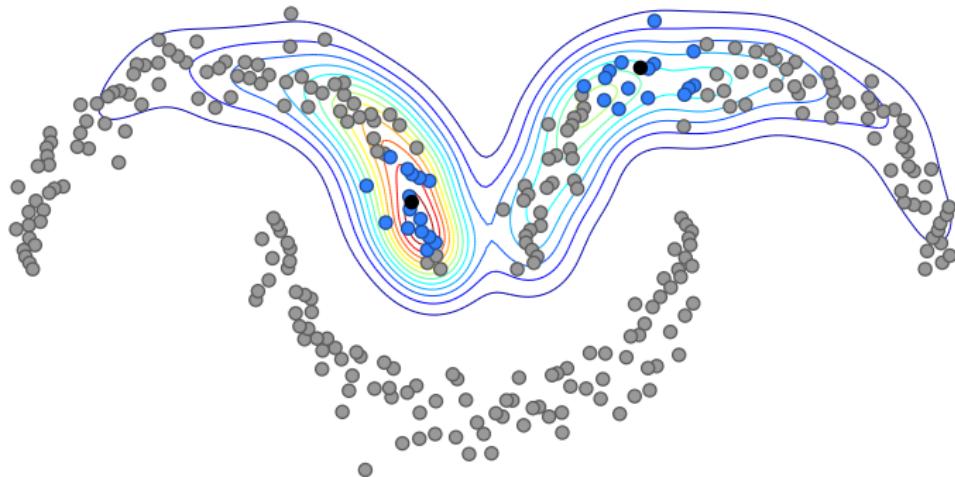
- data points (•), query points (•), nearest neighbors (•)
- iteration 2×30

ranking on manifolds: multiple queries



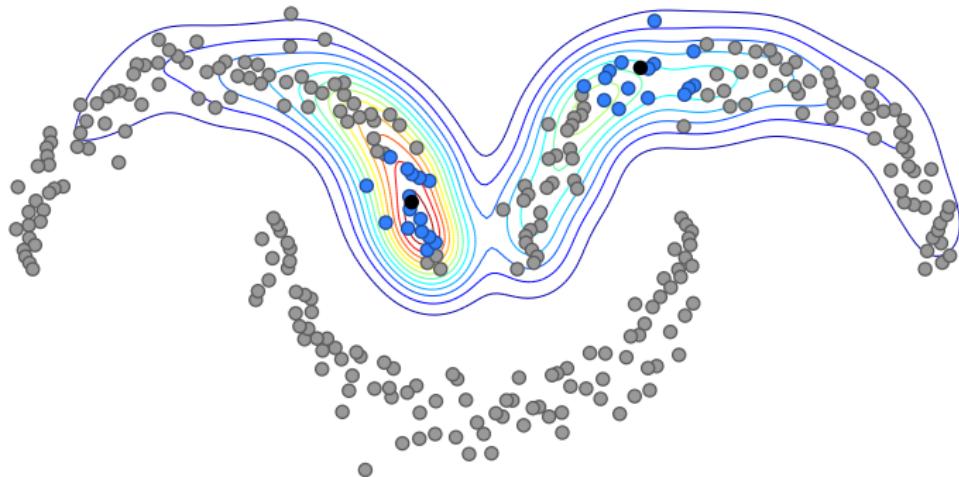
- data points (•), query points (•), nearest neighbors (•)
- iteration 3×30

ranking on manifolds: multiple queries



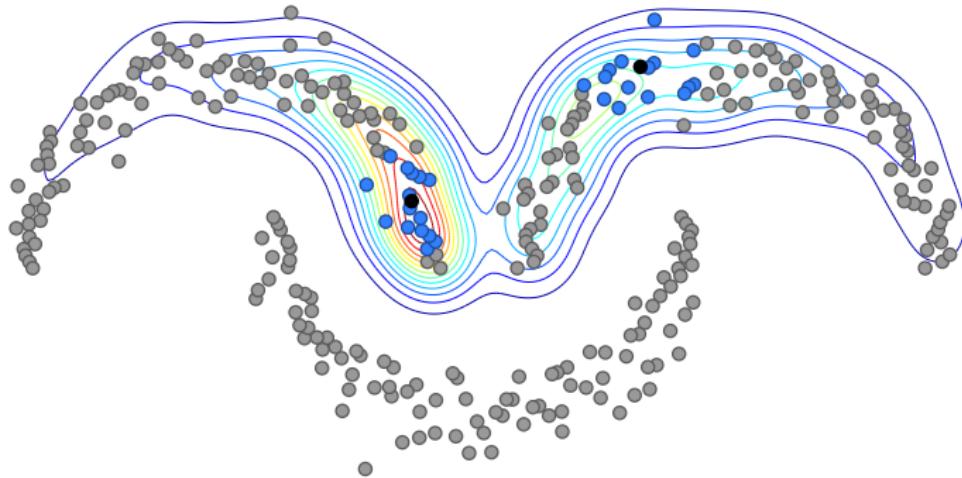
- data points (•), query points (•), nearest neighbors (•)
- iteration 4×30

ranking on manifolds: multiple queries



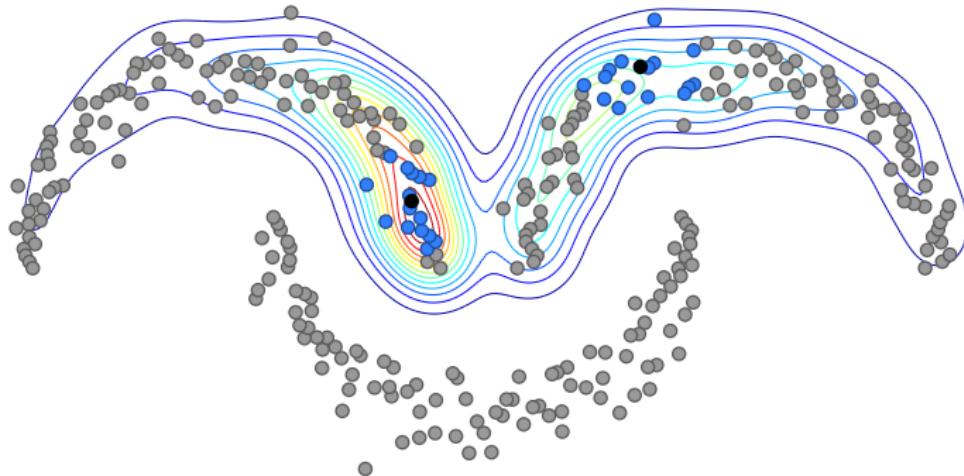
- data points (●), query points (●), nearest neighbors (●)
- iteration 5×30

ranking on manifolds: multiple queries



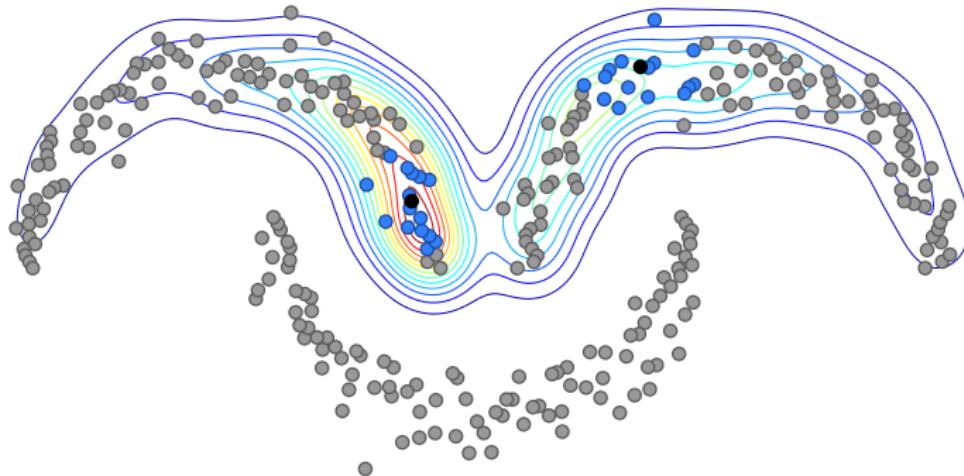
- data points (•), query points (•), nearest neighbors (•)
- iteration 6×30

ranking on manifolds: multiple queries



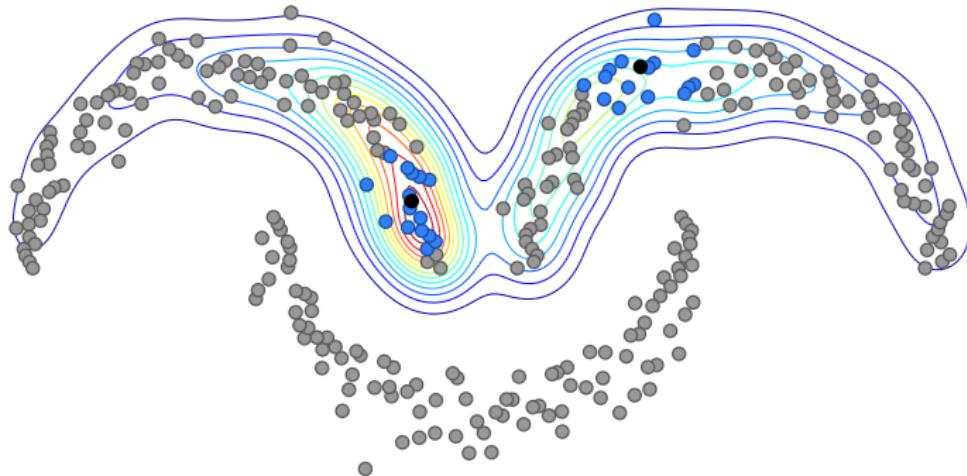
- data points (•), query points (•), nearest neighbors (•)
- iteration 7×30

ranking on manifolds: multiple queries



- data points (•), query points (•), nearest neighbors (•)
- iteration 8×30

ranking on manifolds: multiple queries



- data points (•), query points (•), nearest neighbors (•)
- iteration 9×30

ranking on manifolds: random walk

[Zhou et al. 2003]

- reciprocal nearest neighbor graph on n data points
- non-negative, symmetric, sparse adjacency matrix $W \in \mathbb{R}^{n \times n}$, with zero diagonal (no self-loops)
- symmetrically normalized adjacency matrix

$$\mathcal{W} := D^{-1/2} W D^{-1/2}$$

where $D = \text{diag}(W\mathbf{1})$ is the degree matrix

- query: vector $\mathbf{y} \in \mathbb{R}^n$ with $y_i = \mathbb{1}[i \text{ is query}]$
- random walk: starting with any $\mathbf{f}^{(0)} \in \mathbb{R}^n$, iterate

$$\mathbf{f}^{(\tau)} = \alpha \mathcal{W} \mathbf{f}^{(\tau-1)} + (1 - \alpha) \mathbf{y}$$

where $\alpha \in [0, 1)$ (typically close to 1)

- rank data points by descending order of \mathbf{f}

ranking on manifolds: random walk

[Zhou et al. 2003]

- reciprocal nearest neighbor graph on n data points
- non-negative, symmetric, sparse adjacency matrix $W \in \mathbb{R}^{n \times n}$, with zero diagonal (no self-loops)
- symmetrically normalized adjacency matrix

$$\mathcal{W} := D^{-1/2} W D^{-1/2}$$

where $D = \text{diag}(W\mathbf{1})$ is the degree matrix

- query: vector $\mathbf{y} \in \mathbb{R}^n$ with $y_i = \mathbb{1}[i \text{ is query}]$
- random walk: starting with any $\mathbf{f}^{(0)} \in \mathbb{R}^n$, iterate

$$\mathbf{f}^{(\tau)} = \alpha \mathcal{W} \mathbf{f}^{(\tau-1)} + (1 - \alpha) \mathbf{y}$$

where $\alpha \in [0, 1)$ (typically close to 1)

- rank data points by descending order of \mathbf{f}

ranking on manifolds: random walk

[Zhou et al. 2003]

- reciprocal nearest neighbor graph on n data points
- non-negative, symmetric, sparse adjacency matrix $W \in \mathbb{R}^{n \times n}$, with zero diagonal (no self-loops)
- symmetrically normalized adjacency matrix

$$\mathcal{W} := D^{-1/2} W D^{-1/2}$$

where $D = \text{diag}(W\mathbf{1})$ is the degree matrix

- query: vector $\mathbf{y} \in \mathbb{R}^n$ with $y_i = \mathbb{1}[i \text{ is query}]$
- random walk: starting with any $\mathbf{f}^{(0)} \in \mathbb{R}^n$, iterate

$$\mathbf{f}^{(\tau)} = \alpha \mathcal{W} \mathbf{f}^{(\tau-1)} + (1 - \alpha) \mathbf{y}$$

where $\alpha \in [0, 1)$ (typically close to 1)

- rank data points by descending order of \mathbf{f}

ranking on manifolds: random walk

[Zhou et al. 2003]

- reciprocal nearest neighbor graph on n data points
- non-negative, symmetric, sparse adjacency matrix $W \in \mathbb{R}^{n \times n}$, with zero diagonal (no self-loops)
- symmetrically normalized adjacency matrix

$$\mathcal{W} := D^{-1/2} W D^{-1/2}$$

where $D = \text{diag}(W\mathbf{1})$ is the degree matrix

- query: vector $\mathbf{y} \in \mathbb{R}^n$ with $y_i = \mathbb{1}[i \text{ is query}]$
- random walk: starting with any $\mathbf{f}^{(0)} \in \mathbb{R}^n$, iterate

$$\mathbf{f}^{(\tau)} = \alpha \mathcal{W} \mathbf{f}^{(\tau-1)} + (1 - \alpha) \mathbf{y}$$

where $\alpha \in [0, 1)$ (typically close to 1)

- rank data points by descending order of \mathbf{f}

ranking on manifolds: random walk

[Zhou et al. 2003]

- reciprocal nearest neighbor graph on n data points
- non-negative, symmetric, sparse adjacency matrix $W \in \mathbb{R}^{n \times n}$, with zero diagonal (no self-loops)
- symmetrically normalized adjacency matrix

$$\mathcal{W} := D^{-1/2} W D^{-1/2}$$

where $D = \text{diag}(W\mathbf{1})$ is the degree matrix

- **query**: vector $\mathbf{y} \in \mathbb{R}^n$ with $y_i = \mathbb{1}[i \text{ is query}]$
- **random walk**: starting with any $\mathbf{f}^{(0)} \in \mathbb{R}^n$, iterate

$$\mathbf{f}^{(\tau)} = \alpha \mathcal{W} \mathbf{f}^{(\tau-1)} + (1 - \alpha) \mathbf{y}$$

where $\alpha \in [0, 1)$ (typically close to 1)

- **rank** data points by descending order of \mathbf{f}

ranking as solving a linear system

[Iscen et al. 2017]

- **query**: sparse vector $\mathbf{y} \in \mathbb{R}^n$ with

$$y_i = \sum_{\mathbf{q} \in Q} s(\mathbf{q}, \mathbf{x}_i) \times \mathbb{1}[\mathbf{x}_i \in \text{NN}_X^k(\mathbf{q})]$$

where $Q, X \subset \mathbb{R}^d$ query/data points, $\mathbf{x}_i \in X$, s similarity function

- regularized Laplacian

$$\mathcal{L}_\alpha = \frac{I - \alpha \mathcal{W}}{1 - \alpha}$$

- solve linear system

$$\mathcal{L}_\alpha \mathbf{f} = \mathbf{y}$$

by conjugate gradient method

ranking as solving a linear system

[Iscen et al. 2017]

- **query**: sparse vector $\mathbf{y} \in \mathbb{R}^n$ with

$$y_i = \sum_{\mathbf{q} \in Q} s(\mathbf{q}, \mathbf{x}_i) \times \mathbb{1}[\mathbf{x}_i \in \text{NN}_X^k(\mathbf{q})]$$

where $Q, X \subset \mathbb{R}^d$ query/data points, $\mathbf{x}_i \in X$, s similarity function

- regularized **Laplacian**

$$\mathcal{L}_\alpha = \frac{I - \alpha \mathcal{W}}{1 - \alpha}$$

- solve **linear system**

$$\mathcal{L}_\alpha \mathbf{f} = \mathbf{y}$$

by conjugate gradient method

ranking as solving a linear system

[Iscen et al. 2017]

- **query**: sparse vector $\mathbf{y} \in \mathbb{R}^n$ with

$$y_i = \sum_{\mathbf{q} \in Q} s(\mathbf{q}, \mathbf{x}_i) \times \mathbb{1}[\mathbf{x}_i \in \text{NN}_X^k(\mathbf{q})]$$

where $Q, X \subset \mathbb{R}^d$ query/data points, $\mathbf{x}_i \in X$, s similarity function

- regularized **Laplacian**

$$\mathcal{L}_\alpha = \frac{I - \alpha \mathcal{W}}{1 - \alpha}$$

- solve **linear system**

$$\mathcal{L}_\alpha \mathbf{f} = \mathbf{y}$$

by conjugate gradient method

ranking as solving a linear system

- represent image by global descriptor or multiple regional descriptors
- perform initial query based on Euclidean nearest neighbors
- re-rank by solving linear system
- fine-tuned ResNet-101 + re-ranking: 87.1 (**95.8**) mAP on Oxford, 1 (**21**) descriptors/image \times 2048 dimensions

mining on manifolds

[Iscen et al. 2018]

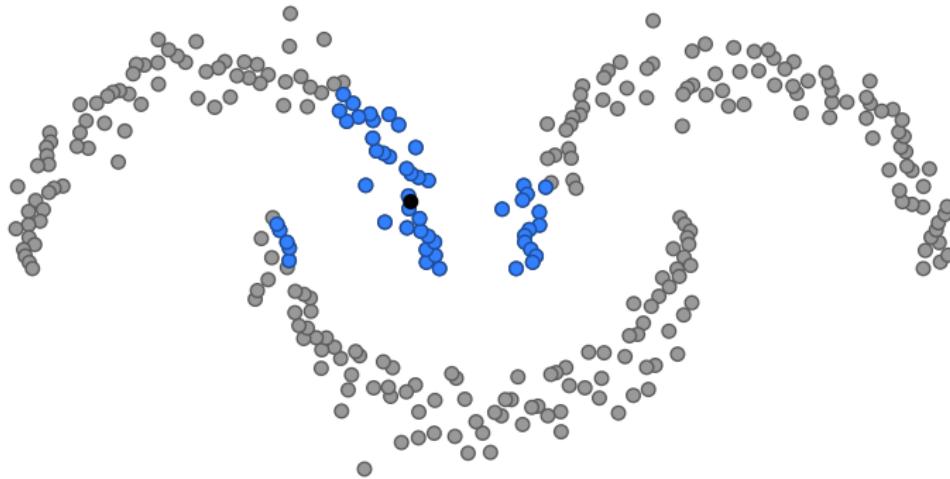


- data points (◦), query point (•)



mining on manifolds

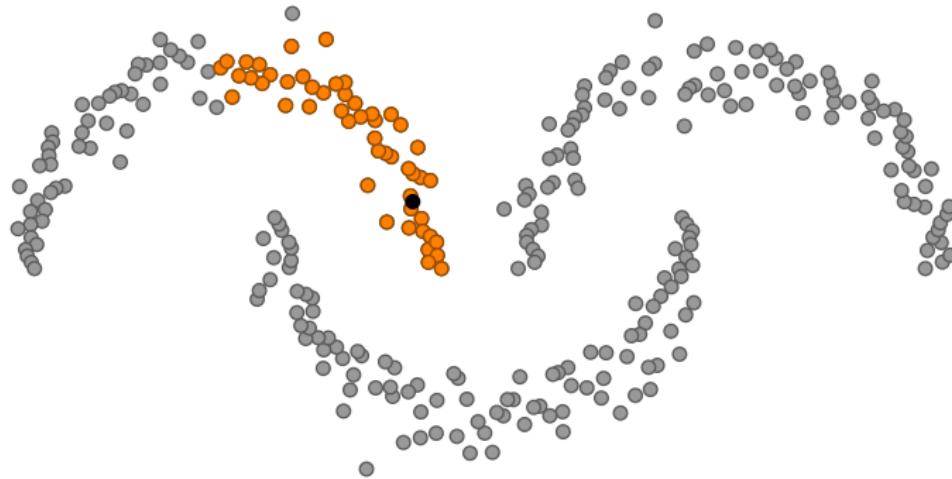
[Iscen et al. 2018]



- data points (\circ), query point (\bullet)
- Euclidean nearest neighbors E (\bullet)

mining on manifolds

[Iscen et al. 2018]



- data points (\circ), query point (\bullet)
 - manifold nearest neighbors M (\bullet)

mining on manifolds

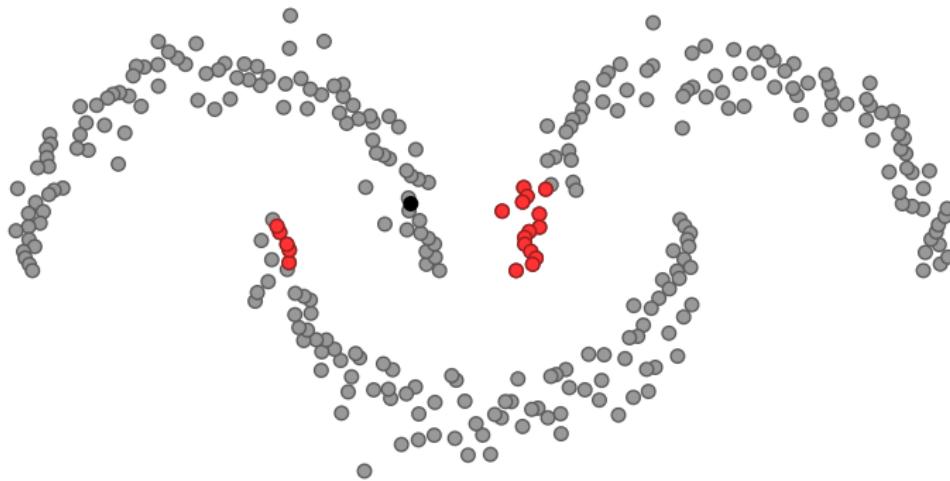
[Iscen et al. 2018]



- data points (\circ), query point (\bullet)
- hard positives $P = M \setminus E$ (\bullet)

mining on manifolds

[Iscen et al. 2018]



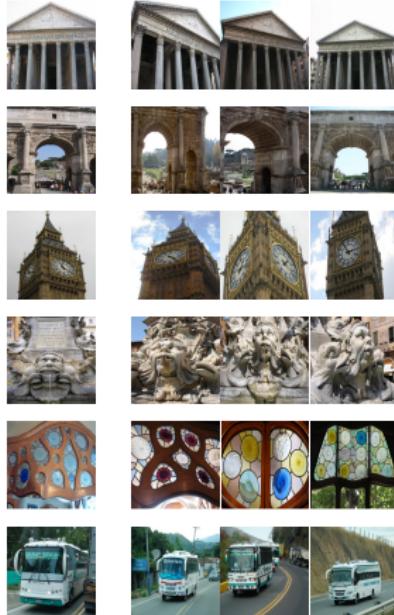
- data points (\circ), query point (\bullet)
- hard negatives $N = E \setminus M$ (\bullet)

mining on manifolds



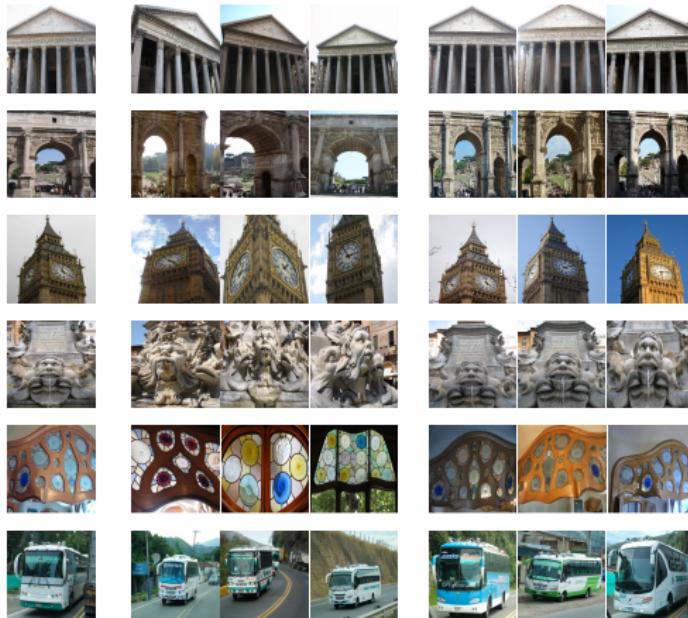
- query (**anchor**), positives P , Euclidean neighbors E , negatives N , Euclidean non-neighbors $X \setminus E$

mining on manifolds



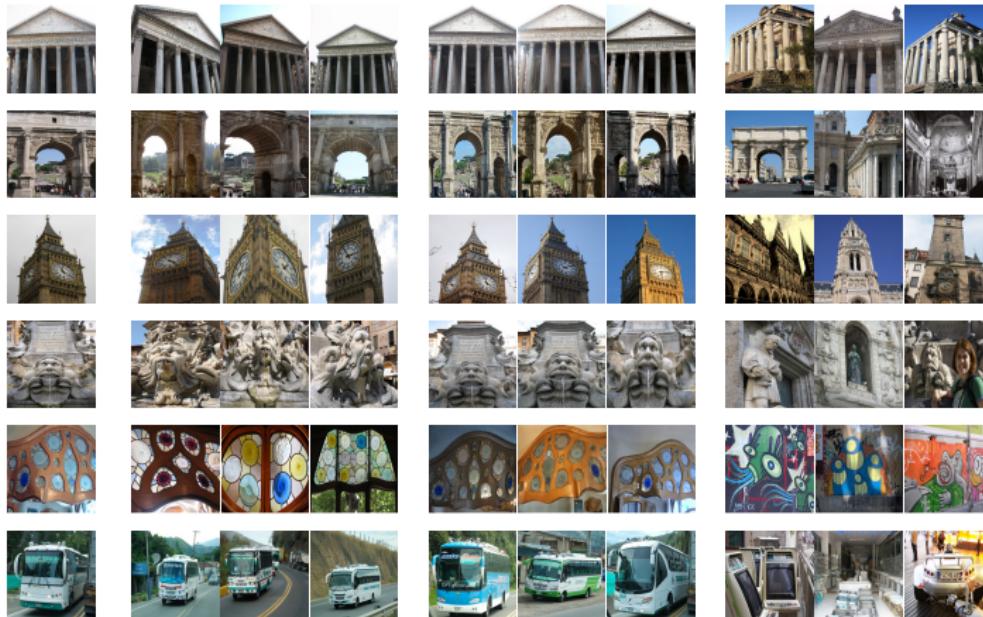
- query (anchor), positives P , Euclidean neighbors E , negatives N , Euclidean non-neighbors $X \setminus E$

mining on manifolds



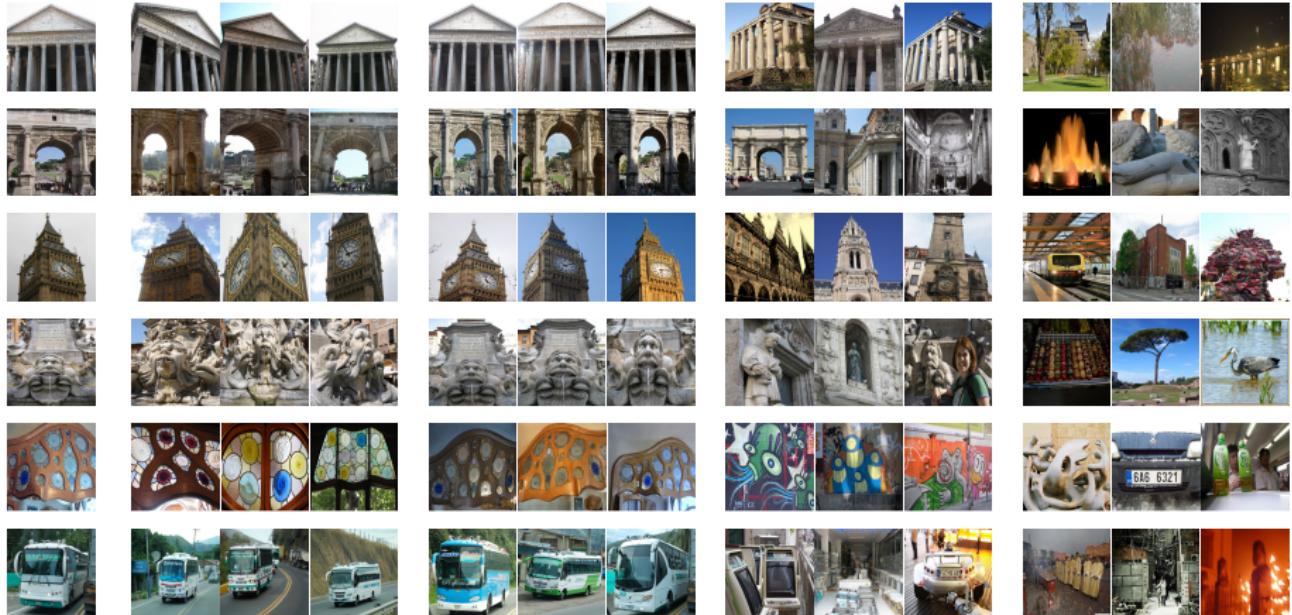
- query (anchor), positives P , Euclidean neighbors E , negatives N , Euclidean non-neighbors $X \setminus E$

mining on manifolds



- query (anchor), positives P , Euclidean neighbors E , negatives N , Euclidean non-neighbors $X \setminus E$

mining on manifolds



- query (anchor), positives P , Euclidean neighbors E , negatives N , Euclidean non-neighbors $X \setminus E$

mining on manifolds

- pre-train network
- extract descriptors on **unlabeled** dataset
- construct nearest neighbor graph
- sample **anchors**, measure Euclidean and manifold distances
- sample **positives** and **negatives**
- fine-tune using **contrastive** or **triplet** loss
- VGG-16 + R-MAC, mAP on Oxford:
 - pre-trained on ImageNet: 68
 - fine-tuning with SIFT + 3d reconstruction pipeline: 77.8
 - **unsupervised fine-tuning**: 78.2

mining on manifolds

- pre-train network
- extract descriptors on **unlabeled** dataset
- construct nearest neighbor graph
- sample **anchors**, measure Euclidean and manifold distances
- sample **positives** and **negatives**
- fine-tune using **contrastive** or **triplet** loss
- VGG-16 + R-MAC, mAP on Oxford:
 - pre-trained on ImageNet: 68
 - fine-tuning with SIFT + 3d reconstruction pipeline: 77.8
 - **unsupervised fine-tuning**: 78.2

mining on manifolds

- pre-train network
- extract descriptors on **unlabeled** dataset
- construct nearest neighbor graph
- sample **anchors**, measure Euclidean and manifold distances
- sample **positives** and **negatives**
- fine-tune using **contrastive** or **triplet** loss
- VGG-16 + R-MAC, mAP on Oxford:
 - pre-trained on ImageNet: 68
 - fine-tuning with SIFT + 3d reconstruction pipeline: 77.8
 - **unsupervised fine-tuning**: 78.2

summary

- bag-of-words and inverted index is only a crude form of approximate nearest neighbor search
- global descriptors are compact and fast, but do not perform as well as local descriptors
- compressed representation for nearest neighbor search are effective if manifold is captured correctly
- pooling CNN representations is best at last convolutional layers: MAC, R-MAC, SPoC, CroW
- fine-tuning with contrastive or triplet loss allows transferring to a new domain and learning to rank as opposed to classify
- modeling the manifold explicitly allows unsupervised fine-tuning without labels, auxiliary systems (e.g. SIFT pipeline), or other information (e.g. temporal neighborhood in video)