

lecture 4: matching and indexing

deep learning for vision

Yannis Avrithis

Inria Rennes-Bretagne Atlantique

Rennes, Nov. 2017 – Jan. 2018



outline

bag of words

codebooks

beyond codebooks

pyramid matching

nearest neighbor search

discussion

bag of words

image matching

- so far, we have a representation that is very robust in matching different views of the same object or scene—same **instance**—to be used e.g. for **retrieval**
- the same representation can be used in matching views of different instances of the same category—same **class**—to be used e.g. for **classification** or **detection**
- main differences

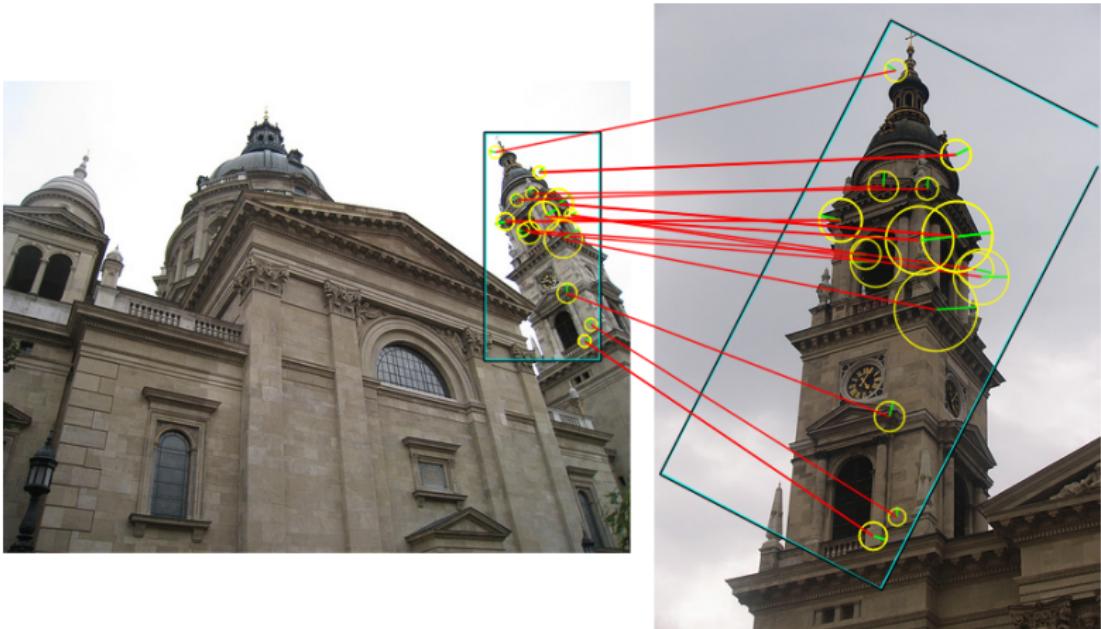
	instance	class
features	sparse	dense
descriptors		same
vocabulary	fine	coarse
geometry	rigid	flexible

image matching

- so far, we have a representation that is very robust in matching different views of the same object or scene—same **instance**—to be used e.g. for **retrieval**
- the same representation can be used in matching views of different instances of the same category—same **class**—to be used e.g. for **classification** or **detection**
- main differences

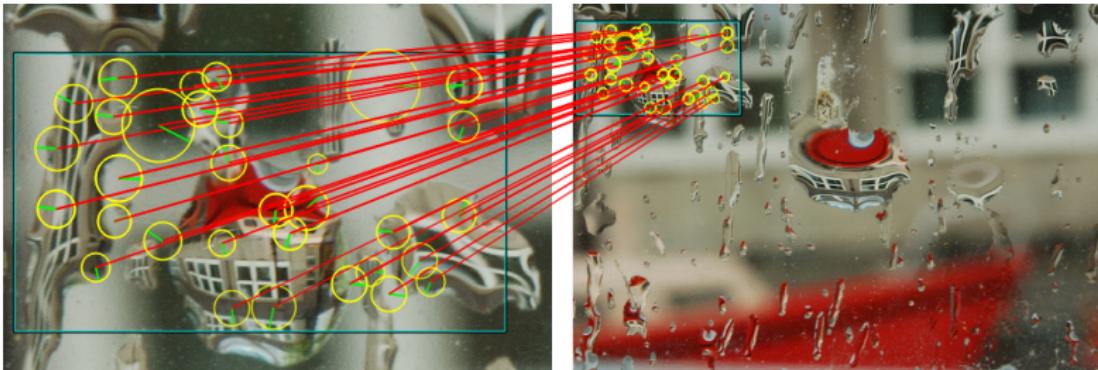
	instance	class
features	sparse	dense
descriptors		same
vocabulary	fine	coarse
geometry	rigid	flexible

spatial matching—same instance



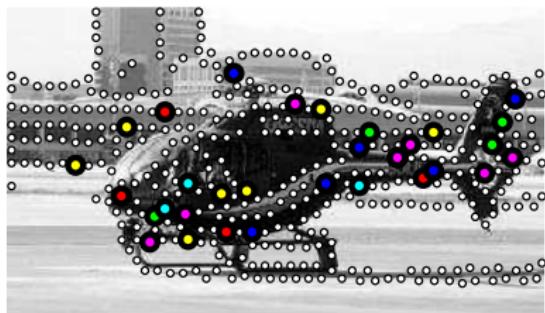
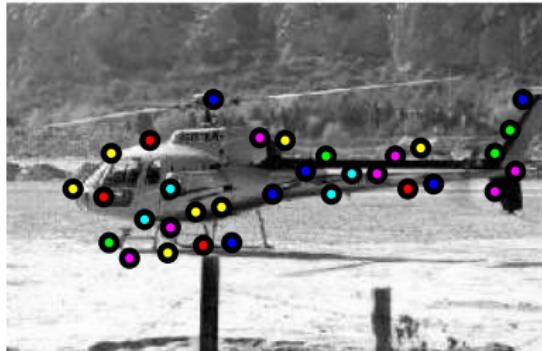
- now robust to scale, viewpoint, occlusion, clutter, lighting
- and very fast

spatial matching—same instance



- now robust to scale, viewpoint, occlusion, clutter, lighting
- and very fast

spatial matching—same class



- solve for feature correspondence, flexible transformation and outliers on all possible **correspondence pairs** by joint optimization
- very expensive

spatial matching—same class



- solve for feature correspondence, flexible transformation and outliers on all possible **correspondence pairs** by joint optimization
- very expensive and error prone

geometry

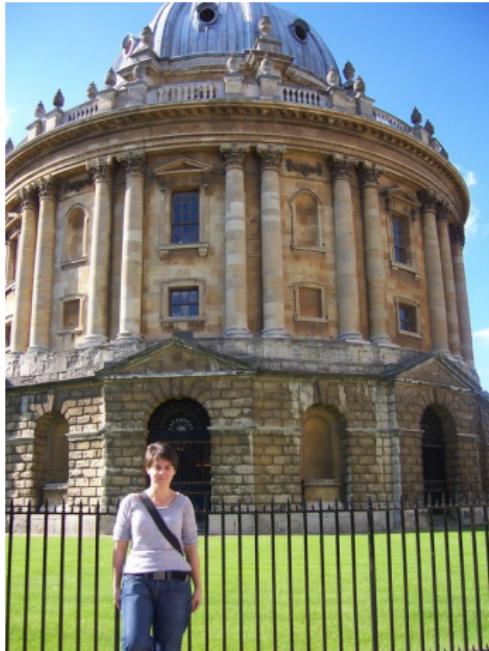
- spatial matching on **same instance** is robust, but expensive
- we can
 - encode position, e.g. with dense features; easier to match, but we loose invariance
 - discard geometry altogether and use a global representation; even easier, we maintain invariance, but loose discriminative power
 - discard geometry as a first step, then bring it back
 - make it more efficient?
- rigid transformations won't work for **classification**, and matching is even more expensive
 - make it more flexible?
 - make it more efficient?
 - maintain invariance?

geometry

- spatial matching on **same instance** is robust, but expensive
- we can
 - encode position, e.g. with dense features; easier to match, but we loose invariance
 - discard geometry altogether and use a global representation; even easier, we maintain invariance, but loose discriminative power
 - discard geometry as a first step, then bring it back
 - make it more efficient?
- rigid transformations won't work for **classification**, and matching is even more expensive
 - make it more flexible?
 - make it more efficient?
 - maintain invariance?

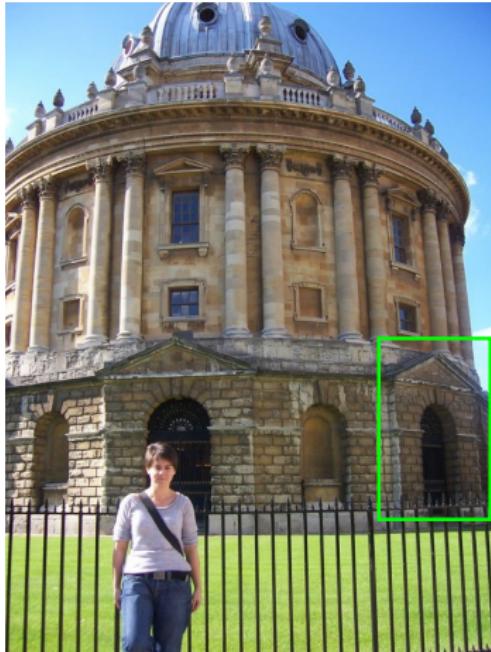
matching discriminative local features

[Lowe, ICCV 1999]

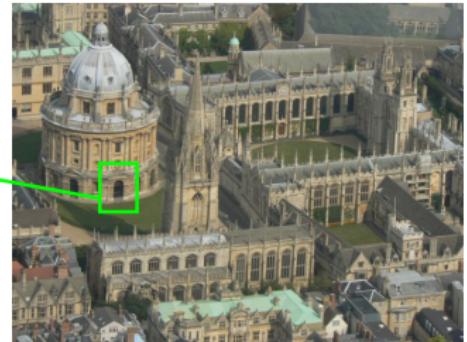


matching discriminative local features

[Lowe, ICCV 1999]

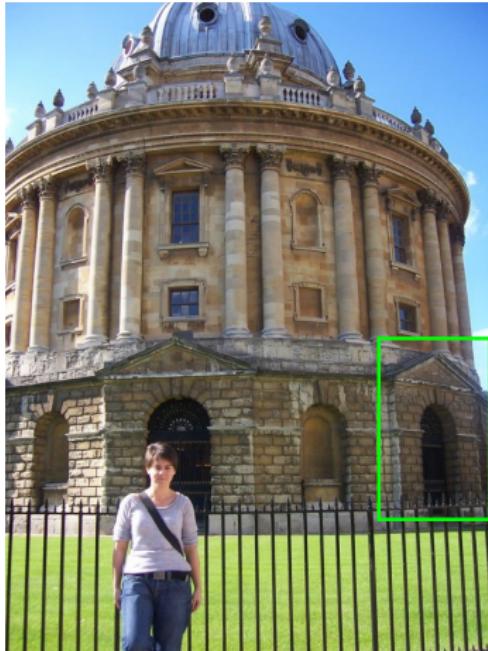


features

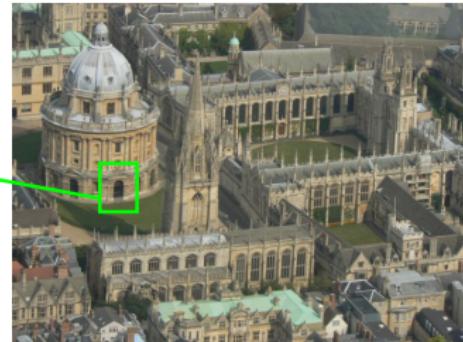
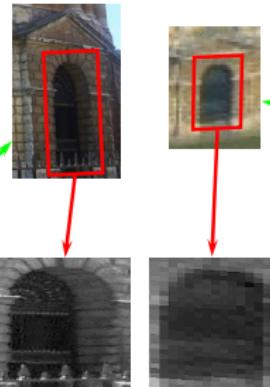


matching discriminative local features

[Lowe, ICCV 1999]



features



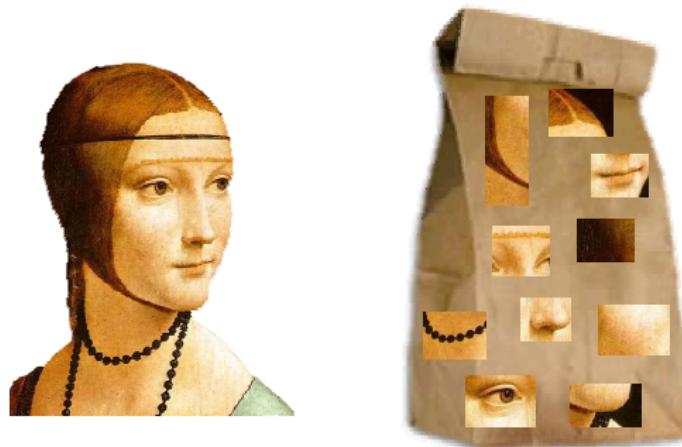
normalized features

appearance

- matching appearance via descriptors should be easier than geometry
- but
 - if we have positions e.g. with dense features, we know what to match (but we loose invariance)
 - otherwise, we need to find correspondences (expensive)
 - we can apply some **pooling** in image space or in descriptor space; more efficient; it may help or not
 - **global** pooling is the most efficient (but is not as discriminative)
 - local descriptors take up a lot of space; with pooling or not, we can **compress** them

forget about geometry: bag-of-words

[Sivic and Zisserman 2003]



- in fact, discarding geometry (**bag**) is one thing and quantizing descriptors (**words**) is another

vector quantization → visual words



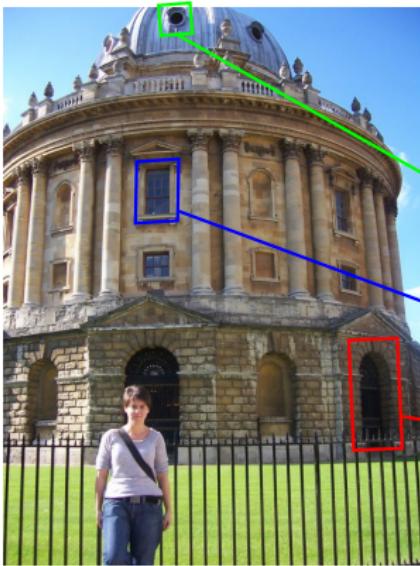
query



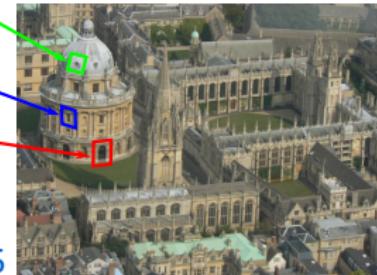
15

- query vs. dataset image

vector quantization → visual words



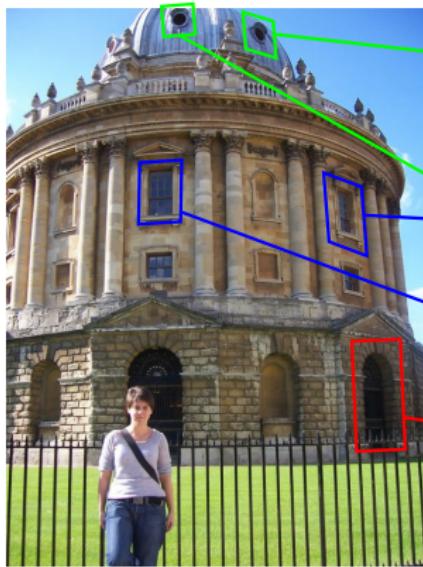
query



15

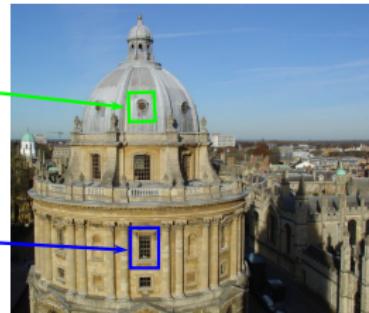
- pairwise descriptor matching

vector quantization → visual words

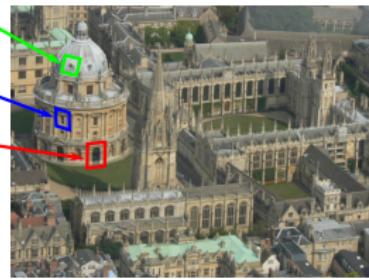


query

19

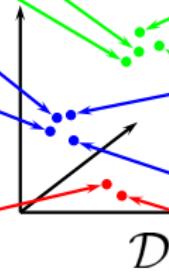
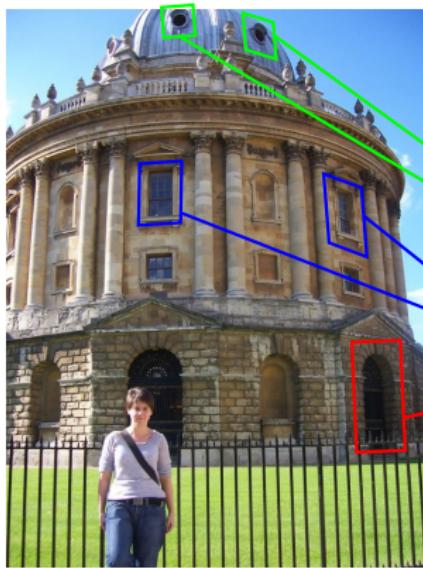


15

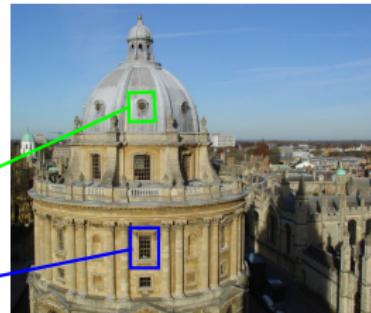


- pairwise descriptor matching for **every** dataset image

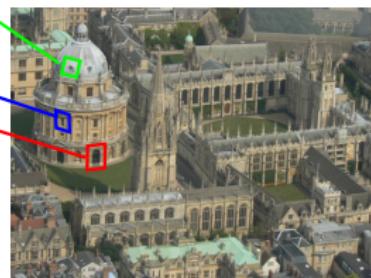
vector quantization → visual words



19

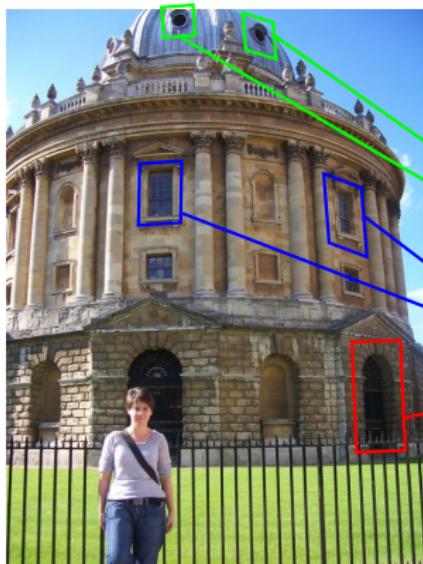


15

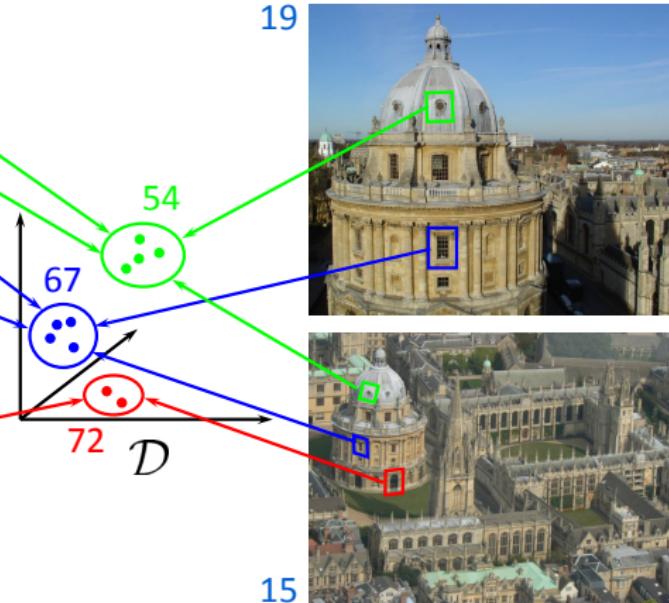


- similar descriptors should all be nearby in the descriptor space

vector quantization → visual words

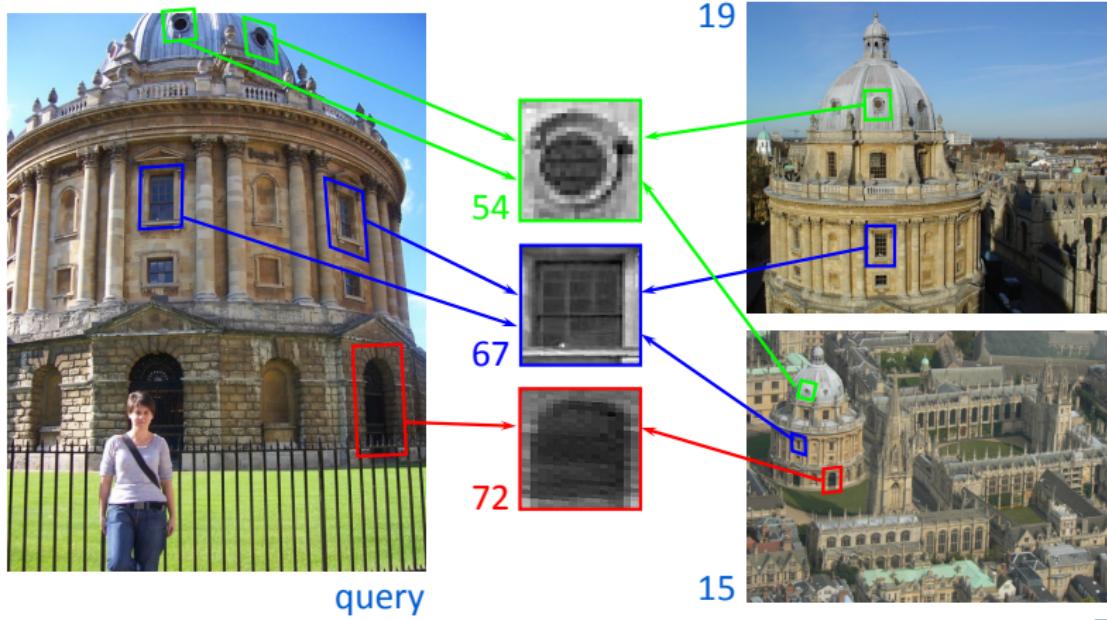


query



- let's quantize them into visual words

vector quantization → visual words



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

bag-of-words and “cosine” similarity

- each image is represented by a single vector $\mathbf{z} \in \mathbb{R}^k$, where k is the size of the codebook
- each element $z_i = w_i n_i$ where w_i fixed weight per visual word and n_i the number of occurrences of this word in the image
- this vector then typically normalized, e.g. $\|\mathbf{z}\|_1 = 1$ or $\|\mathbf{z}\|_2 = 1$
- given two images represented by \mathbf{z}, \mathbf{y} , similarity is measured by dot product

$$s(\mathbf{z}, \mathbf{y}) = s_{\text{BoW}}(\mathbf{z}, \mathbf{y}) := \mathbf{z}^\top \mathbf{y}$$

- with ℓ_2 normalization, this is equivalent to measuring Euclidean distance $\|\mathbf{z} - \mathbf{y}\|$ because $\|\mathbf{z} - \mathbf{y}\|^2 = 2(1 - \mathbf{z}^\top \mathbf{y})$

bag-of-words and “cosine” similarity

- each image is represented by a single vector $\mathbf{z} \in \mathbb{R}^k$, where k is the size of the codebook
- each element $z_i = w_i n_i$ where w_i fixed weight per visual word and n_i the number of occurrences of this word in the image
- this vector then typically normalized, e.g. $\|\mathbf{z}\|_1 = 1$ or $\|\mathbf{z}\|_2 = 1$
- given two images represented by \mathbf{z}, \mathbf{y} , similarity is measured by dot product

$$s(\mathbf{z}, \mathbf{y}) = s_{\text{BoW}}(\mathbf{z}, \mathbf{y}) := \mathbf{z}^\top \mathbf{y}$$

- with ℓ_2 normalization, this is equivalent to measuring Euclidean distance $\|\mathbf{z} - \mathbf{y}\|$ because $\|\mathbf{z} - \mathbf{y}\|^2 = 2(1 - \mathbf{z}^\top \mathbf{y})$

bag-of-words for retrieval

- given a set of n images represented by matrix $Z \in \mathbb{R}^{k \times n}$ (each image as a column) and query image \mathbf{q} , we need a vector of similarities

$$\mathbf{s} = S_{\text{BoW}}(Z, \mathbf{q}) := Z^\top \mathbf{q}$$

and then sort \mathbf{s} by descending order

- when $k \gg m$, where p is the number of features per image on average, Z and \mathbf{q} are sparse
- rather than whether a word is contained in an image, ask which images contain a given word

bag-of-words for retrieval

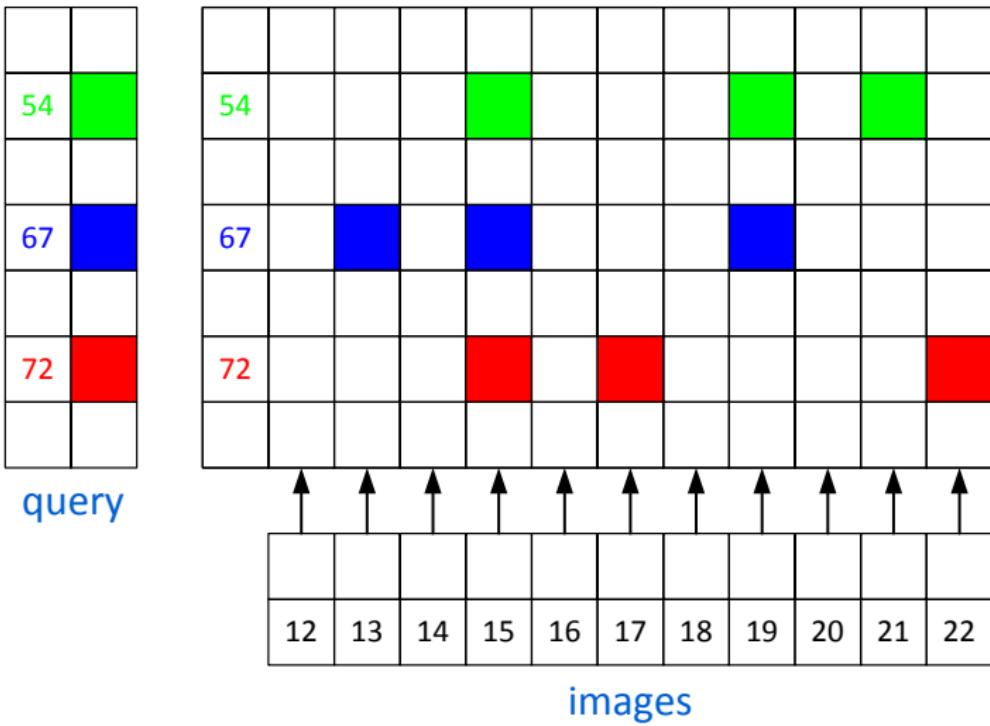
- given a set of n images represented by matrix $Z \in \mathbb{R}^{k \times n}$ (each image as a column) and query image \mathbf{q} , we need a vector of similarities

$$\mathbf{s} = S_{\text{BoW}}(Z, \mathbf{q}) := Z^\top \mathbf{q}$$

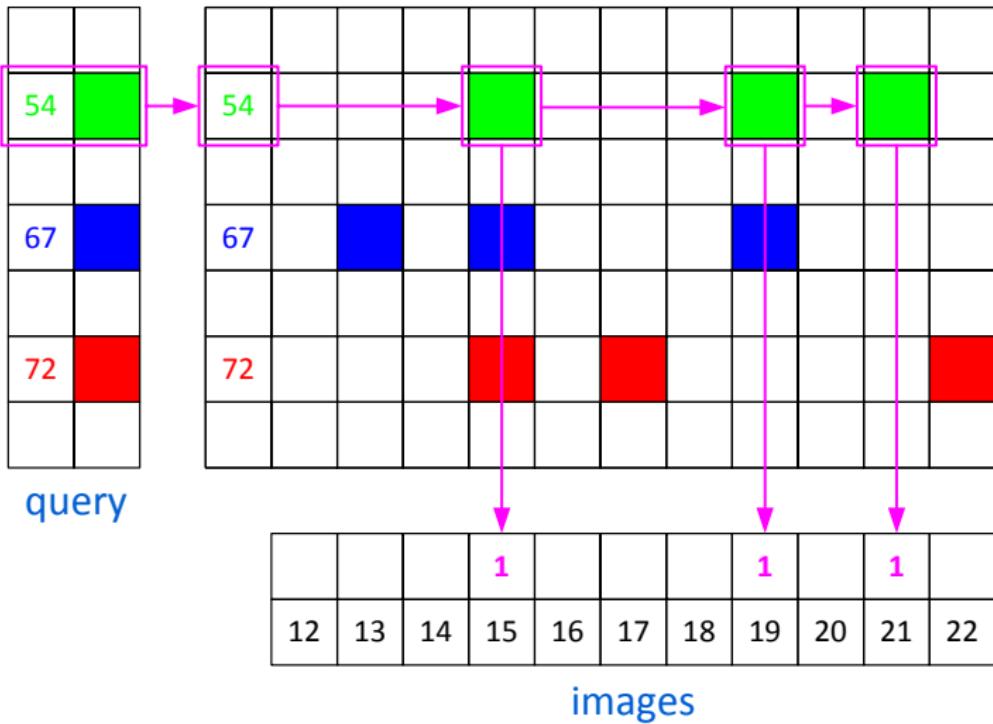
and then sort \mathbf{s} by descending order

- when $k \gg m$, where p is the number of features per image on average, Z and \mathbf{q} are sparse
- rather than whether a word is contained in an image, ask **which images contain a given word**

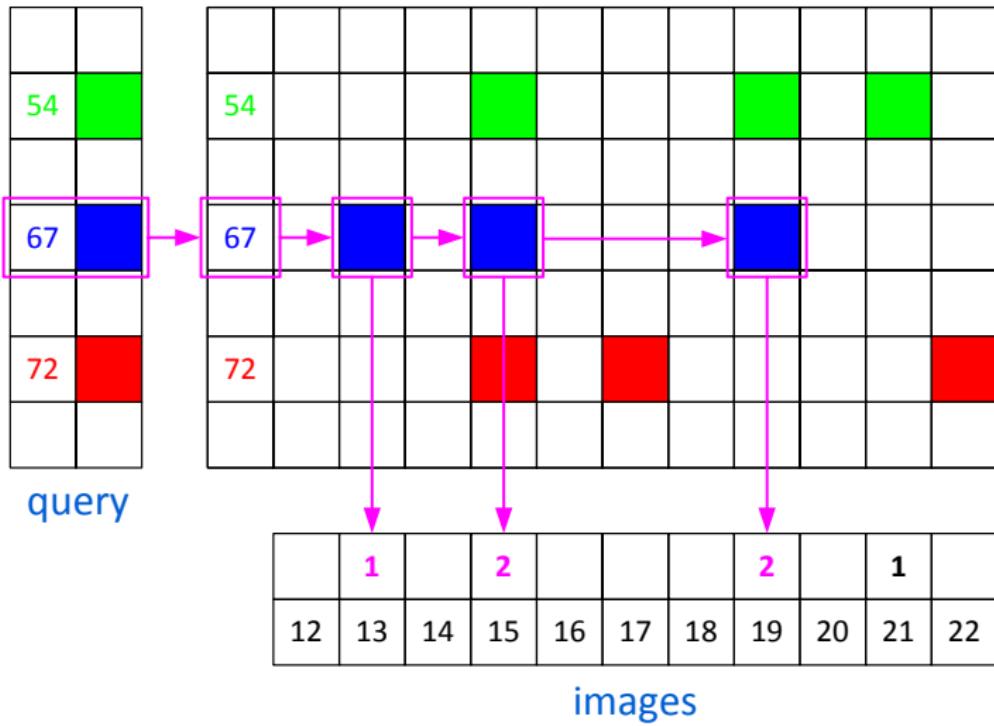
inverted file indexing



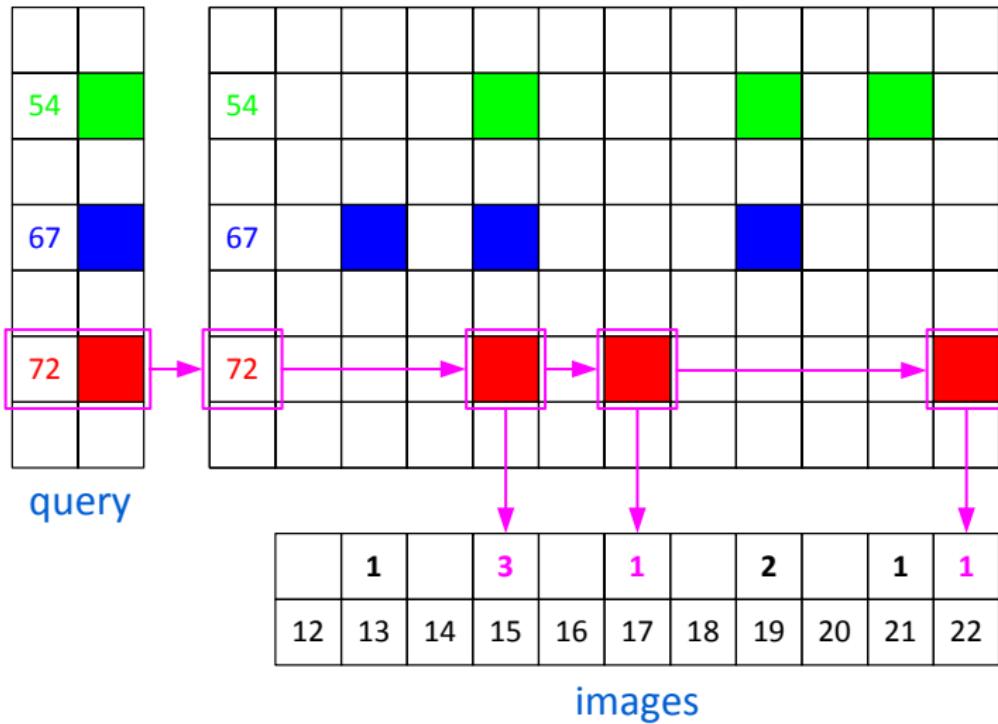
inverted file indexing



inverted file indexing



inverted file indexing



inverted file indexing

54	
67	

54																							
67																							

query

ranked
shortlist

12	13	14	15	16	17	18	19	20	21	22
1			3	1			2	1	1	

images

back to geometry: re-ranking

- dot product similarity is fast but quantized descriptors are not discriminative enough; performs poorly in the presence of distractors
- perform spatial matching only on **top-ranking** images, and re-ranking according to a score based on geometry, e.g. number of inliers
- but to save space, **descriptors are not available**: tentative correspondences are based on visual words, and there are too many (too many features are in correspondence if they are assigned to the same visual word)

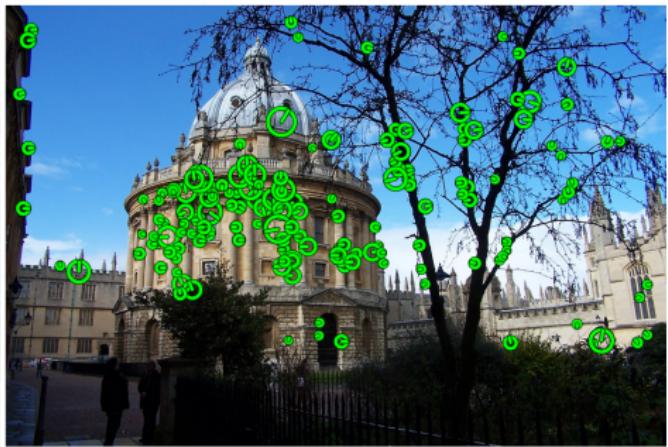
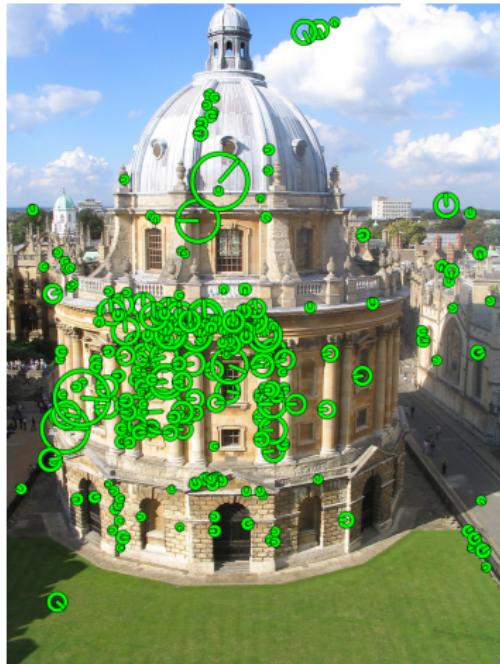
back to geometry: re-ranking



original images

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

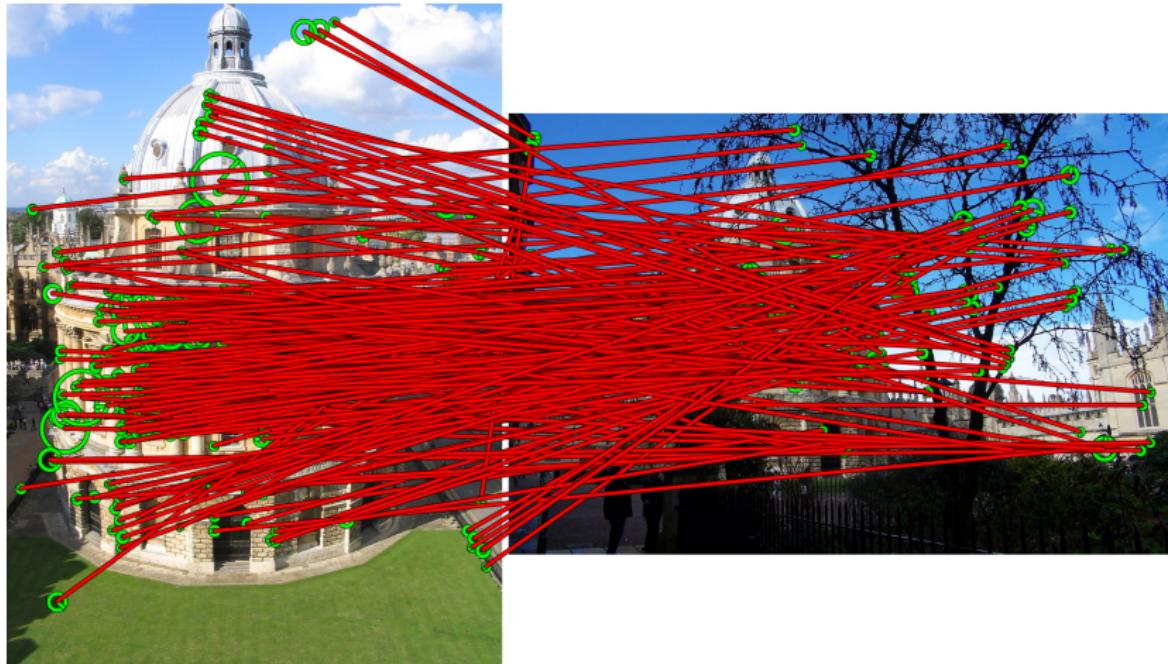
back to geometry: re-ranking



local features

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

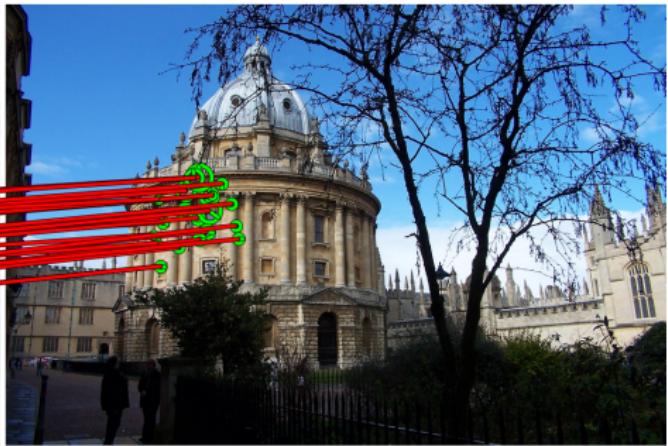
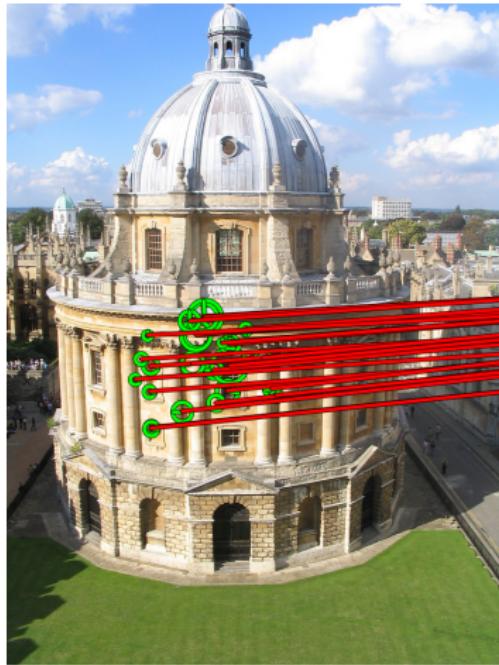
back to geometry: re-ranking



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

bag of words for classification

- each image represented by $\mathbf{z} \in \mathbb{R}^k$; each element z_i the number of occurrences of visual word c_i in the image
- Naïve Bayes: chose maximum posterior probability of class C given image \mathbf{z} assuming features are independent → linear classifier with parameters estimated by visual word statistics on training set
- support vector machine (SVM): images \mathbf{z}, \mathbf{y} compared by kernel function $\kappa(\mathbf{z}, \mathbf{y})$; if $\kappa(\mathbf{z}, \mathbf{y}) = \mathbf{z}^\top \mathbf{y}$, this is again a linear classifier and is a standard choice at large scale

bag of words for classification

- each image represented by $\mathbf{z} \in \mathbb{R}^k$; each element z_i the number of occurrences of visual word c_i in the image
- Naïve Bayes: chose maximum posterior probability of class C given image \mathbf{z} assuming features are independent → linear classifier with parameters estimated by visual word statistics on training set
- support vector machine (SVM): images \mathbf{z}, \mathbf{y} compared by kernel function $\kappa(\mathbf{z}, \mathbf{y})$; if $\kappa(\mathbf{z}, \mathbf{y}) = \mathbf{z}^\top \mathbf{y}$, this is again a linear classifier and is a standard choice at large scale

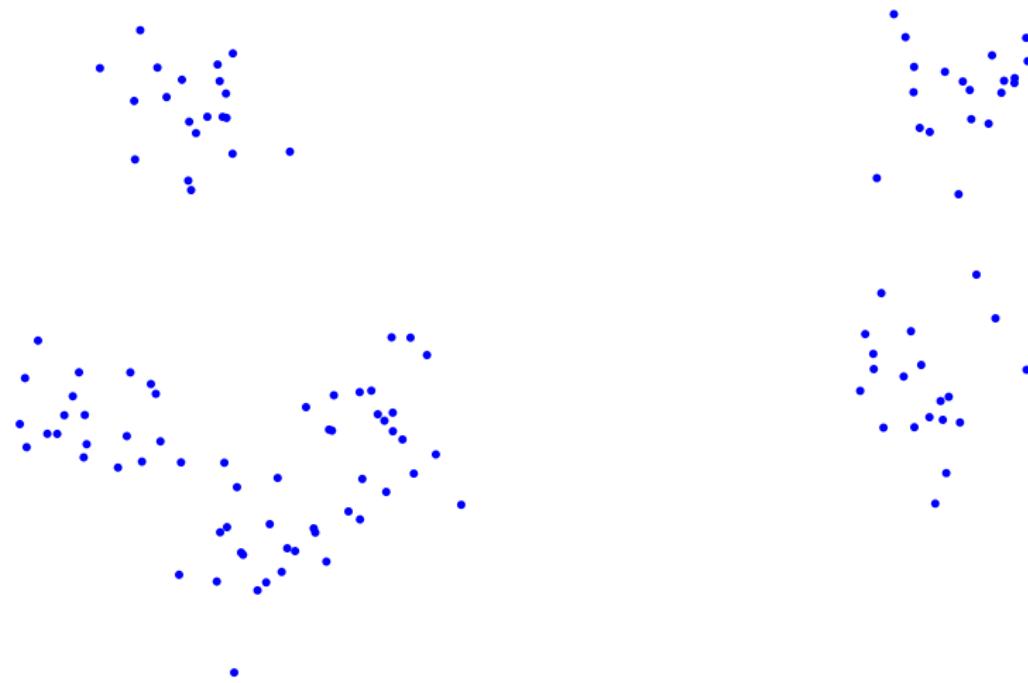
bag of words for classification

- each image represented by $\mathbf{z} \in \mathbb{R}^k$; each element z_i the number of occurrences of visual word c_i in the image
- Naïve Bayes: chose maximum posterior probability of class C given image \mathbf{z} assuming features are independent → linear classifier with parameters estimated by visual word statistics on training set
- support vector machine (SVM): images \mathbf{z}, \mathbf{y} compared by kernel function $\kappa(\mathbf{z}, \mathbf{y})$; if $\kappa(\mathbf{z}, \mathbf{y}) = \mathbf{z}^\top \mathbf{y}$, this is again a linear classifier and is a standard choice at large scale

codebooks

vector quantization: *k*-means clustering

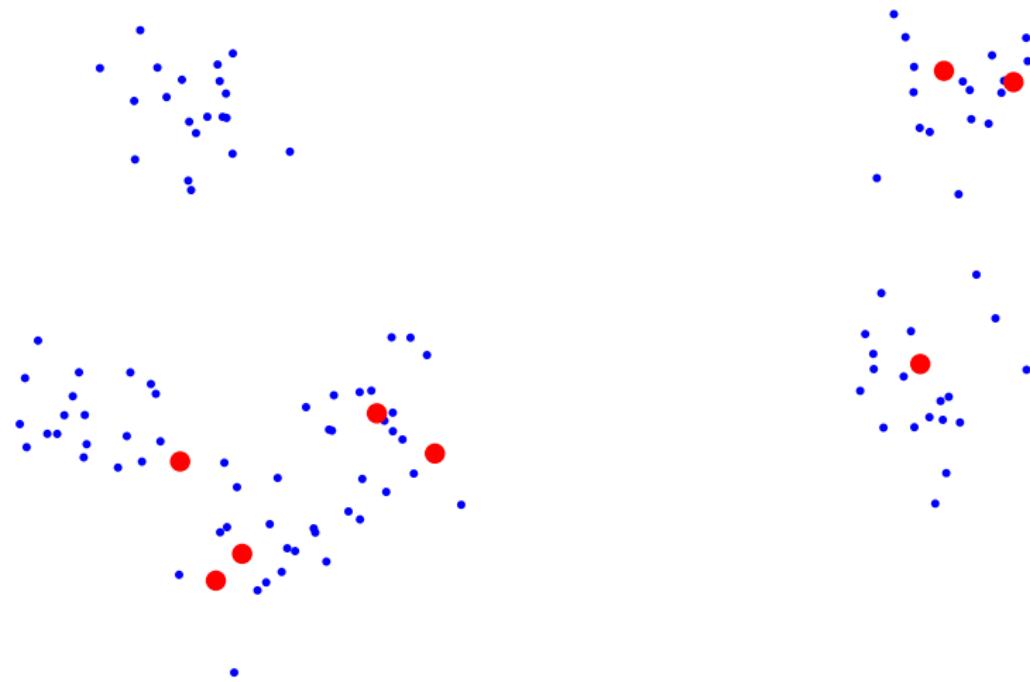
[MacQueen 1967]



dataset

vector quantization: k -means clustering

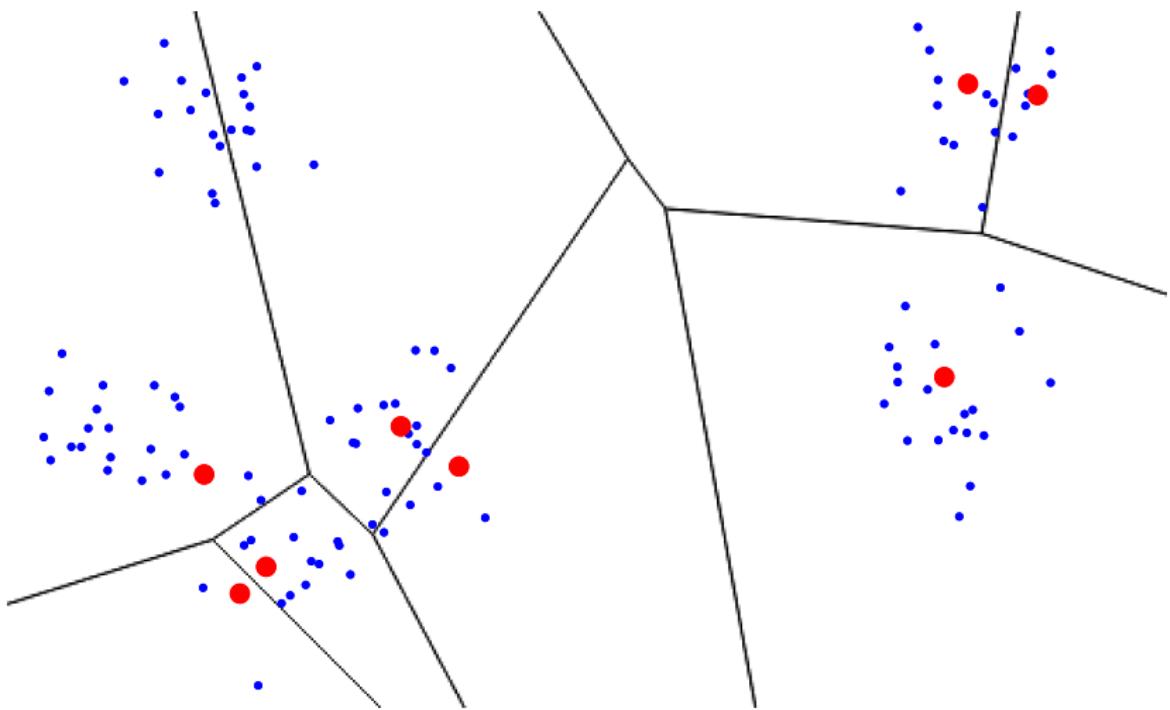
[MacQueen 1967]



initial centroids

vector quantization: *k*-means clustering

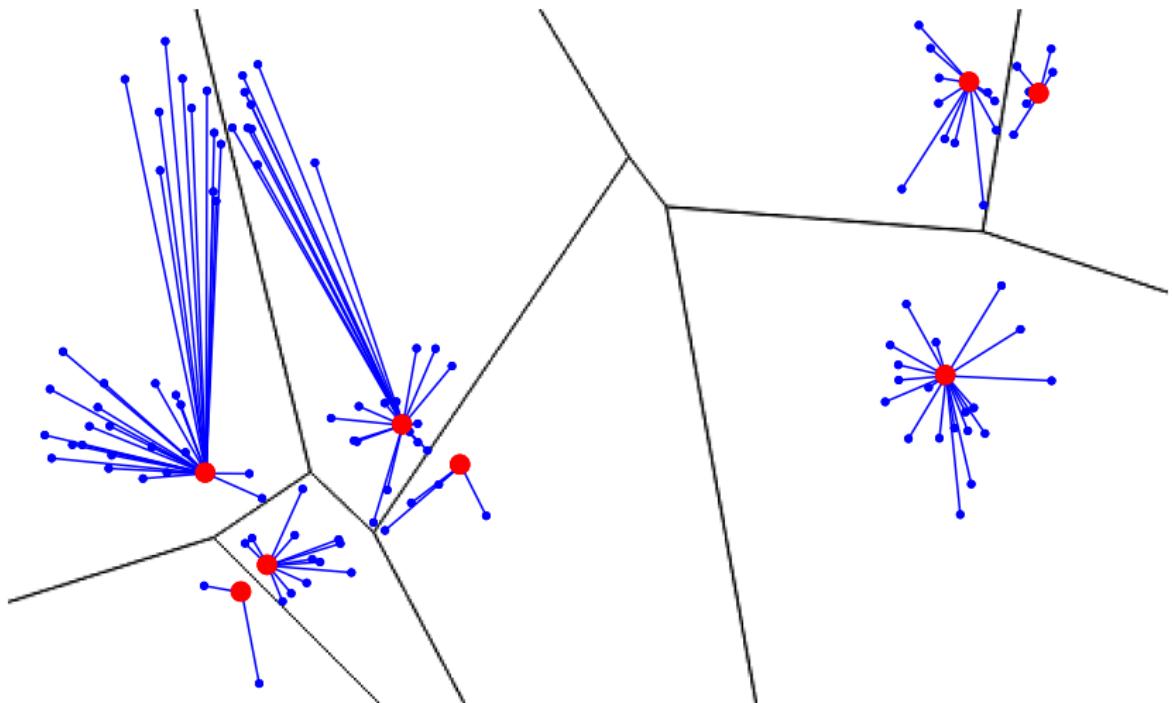
[MacQueen 1967]



Voronoi cells

vector quantization: k -means clustering

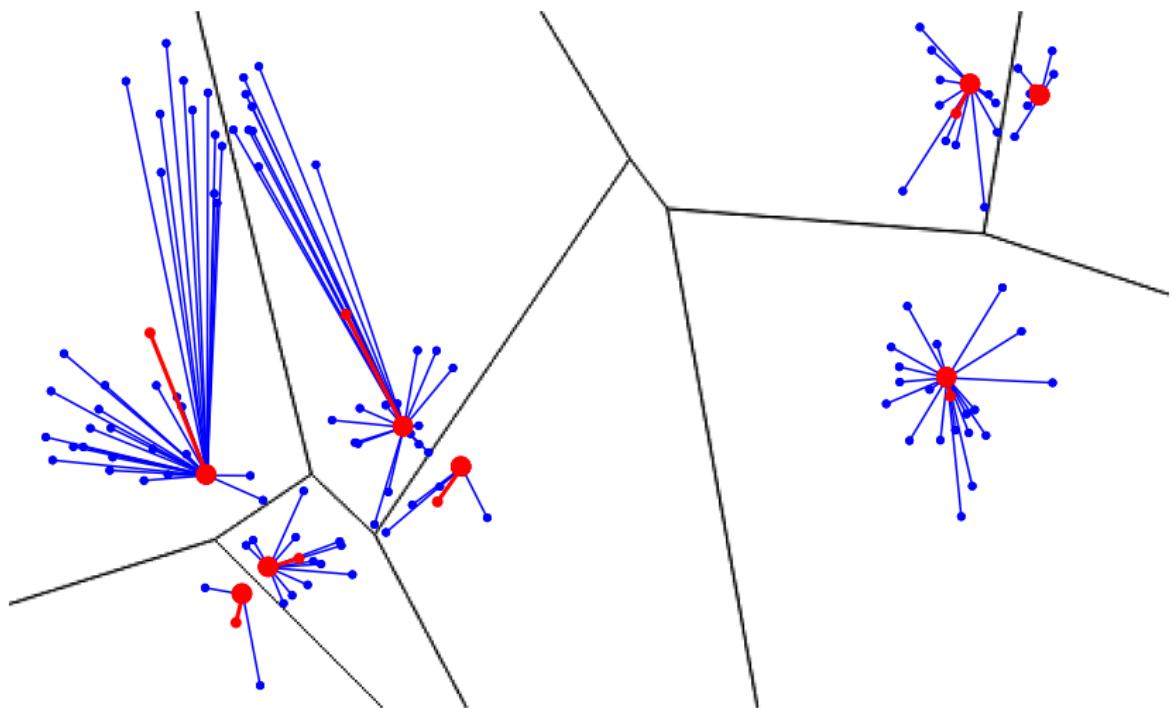
[MacQueen 1967]



points assigned to nearest centroids

vector quantization: *k*-means clustering

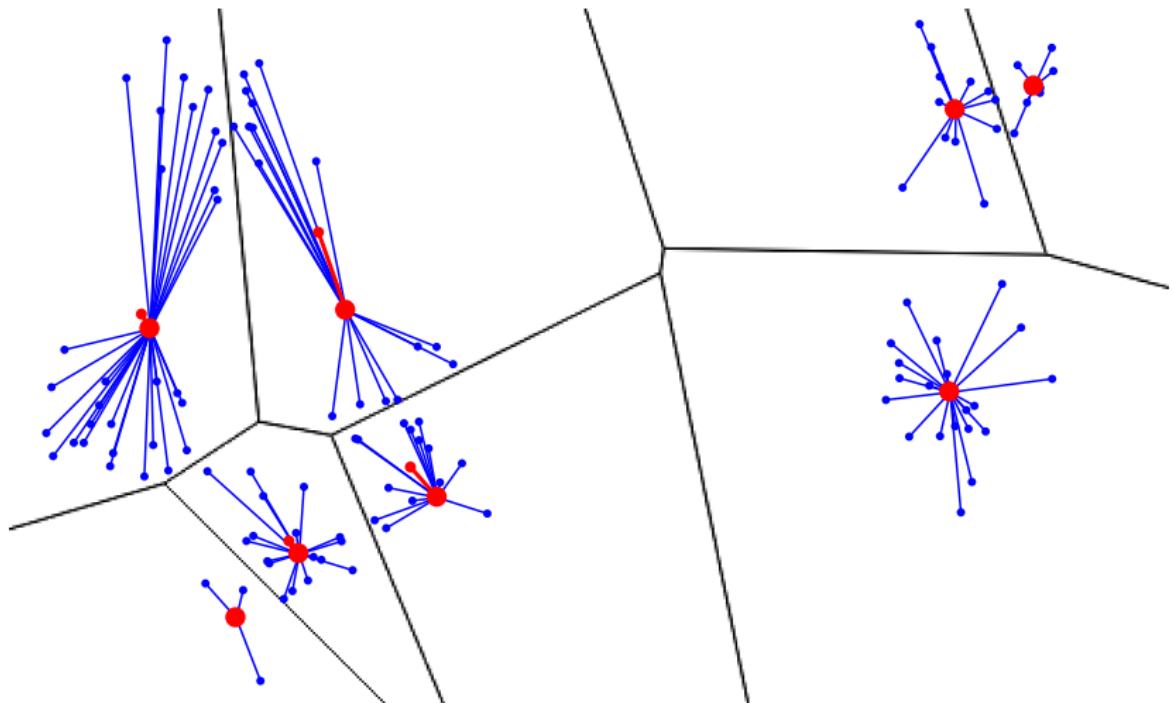
[MacQueen 1967]



centroids move to mean per cell

vector quantization: k -means clustering

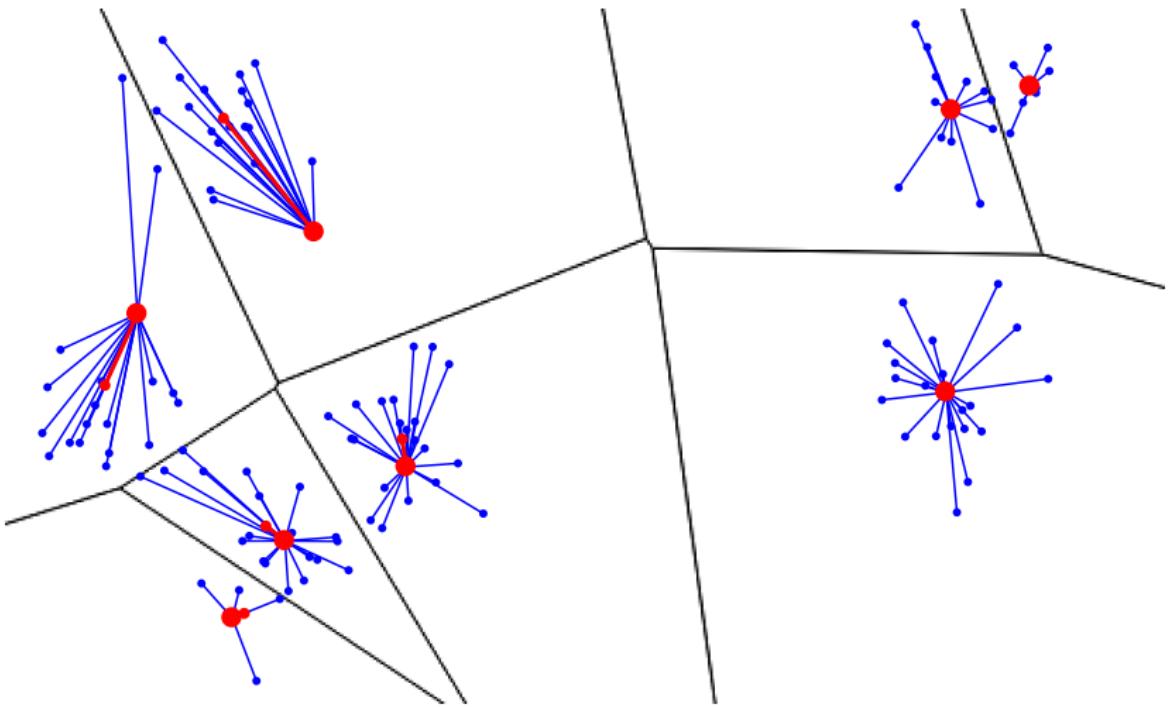
[MacQueen 1967]



iterate until convergence

vector quantization: k -means clustering

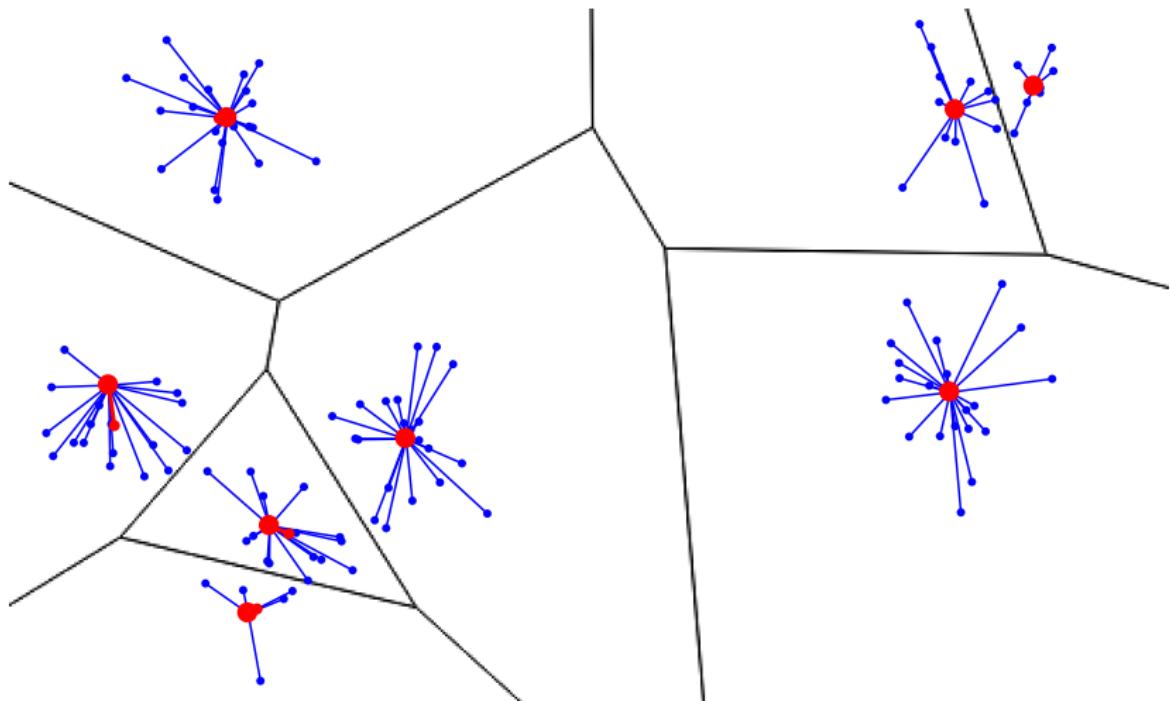
[MacQueen 1967]



iterate until convergence

vector quantization: *k*-means clustering

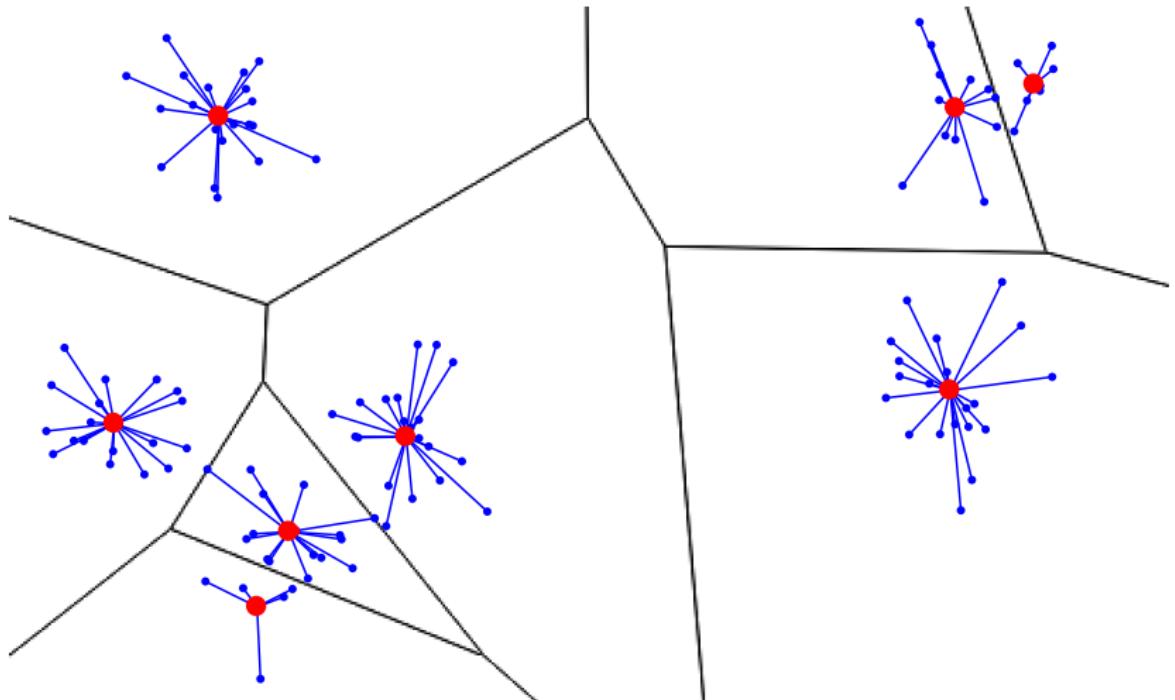
[MacQueen 1967]



iterate until convergence

vector quantization: *k*-means clustering

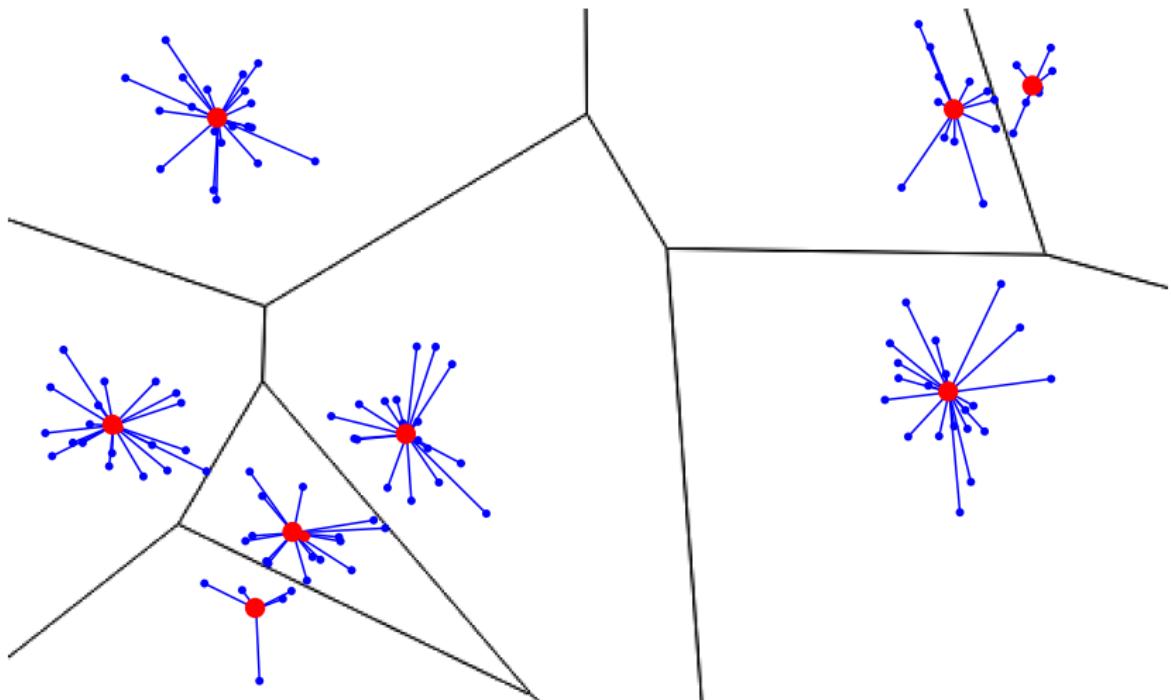
[MacQueen 1967]



iterate until convergence

vector quantization: *k*-means clustering

[MacQueen 1967]



iterate until convergence

vector quantization: *k*-means clustering

- objective: given dataset $X \subset \mathbb{R}^d$, find codebook $C \subset \mathbb{R}^d$, with $|C| = k$, and quantizer function $q : \mathbb{R}^d \rightarrow C$, minimizing distortion

$$E(C, q) := \sum_{x \in X} \|x - q(x)\|^2$$

- regardless of C , q should map vector x to its nearest centroid

$$q(x) = \arg \min_{c \in C} \|x - c\|$$

- algorithm: at each iteration, given the set $X_c = \{x \in X : q(x) = c\}$ of points assigned to centroid c , (assignment step), c moves to their mean (update step)

$$c \leftarrow \frac{1}{|X_c|} \sum_{x \in X_c} x$$

vector quantization: *k*-means clustering

- objective: given dataset $X \subset \mathbb{R}^d$, find codebook $C \subset \mathbb{R}^d$, with $|C| = k$, and quantizer function $q : \mathbb{R}^d \rightarrow C$, minimizing distortion

$$E(C, q) := \sum_{x \in X} \|x - q(x)\|^2$$

- regardless of C , q should map vector x to its nearest centroid

$$q(x) = \arg \min_{c \in C} \|x - c\|$$

- algorithm: at each iteration, given the set $X_c = \{x \in X : q(x) = c\}$ of points assigned to centroid c , (assignment step), c moves to their mean (update step)

$$c \leftarrow \frac{1}{|X_c|} \sum_{x \in X_c} x$$

vector quantization: *k*-means clustering

- objective: given dataset $X \subset \mathbb{R}^d$, find codebook $C \subset \mathbb{R}^d$, with $|C| = k$, and quantizer function $q : \mathbb{R}^d \rightarrow C$, minimizing distortion

$$E(C, q) := \sum_{x \in X} \|x - q(x)\|^2$$

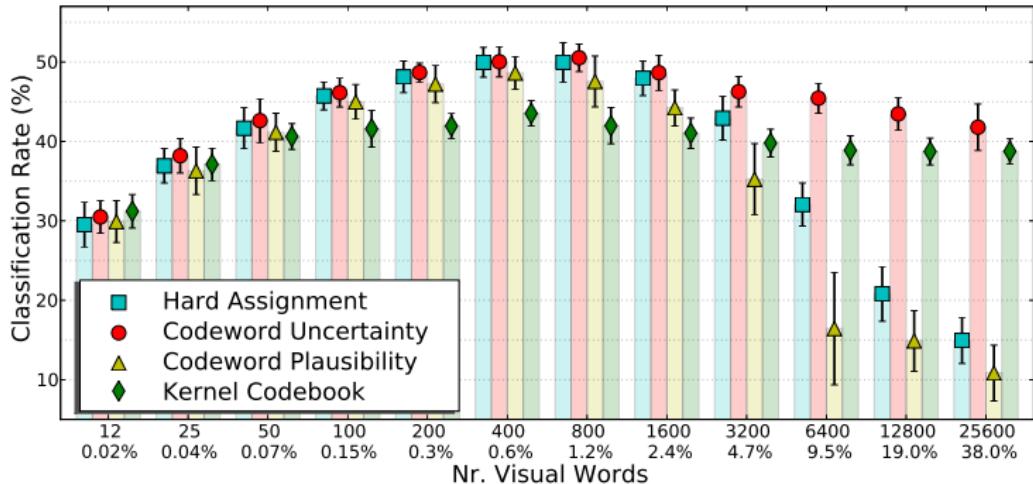
- regardless of C , q should map vector x to its nearest centroid

$$q(x) = \arg \min_{c \in C} \|x - c\|$$

- algorithm: at each iteration, given the set $X_c = \{x \in X : q(x) = c\}$ of points assigned to centroid c , (**assignment step**), c moves to their mean (**update step**)

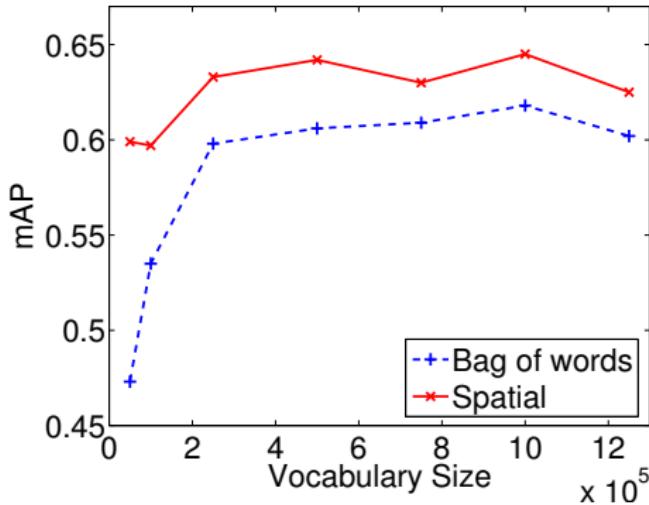
$$c \leftarrow \frac{1}{|X_c|} \sum_{x \in X_c} x$$

codebook size



- classification: **thousands**
- depends on a lot of factors e.g. the number of features in the image representation and size and variability of the dataset

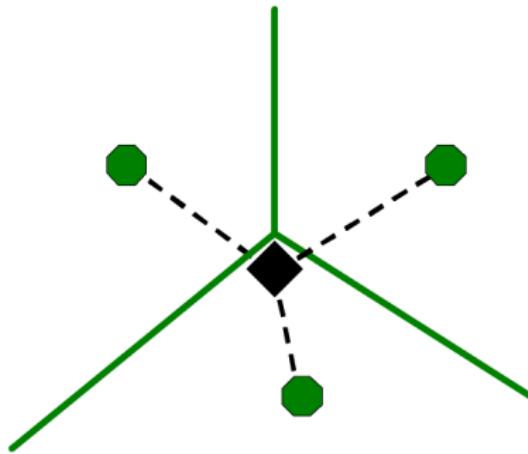
codebook size



- instance retrieval: millions
- depends on a lot of factors e.g. the number of features in the image representation and size and variability of the dataset

hierarchical k -means (HKM)

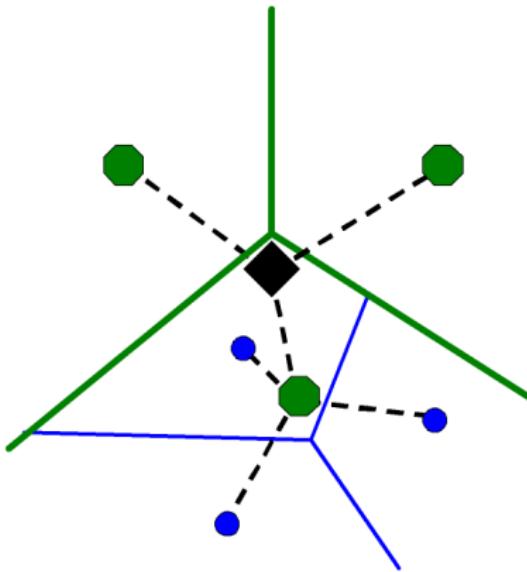
[Fukunaga and Narendra 1975]



- partition data into b clusters using k -means

hierarchical k -means (HKM)

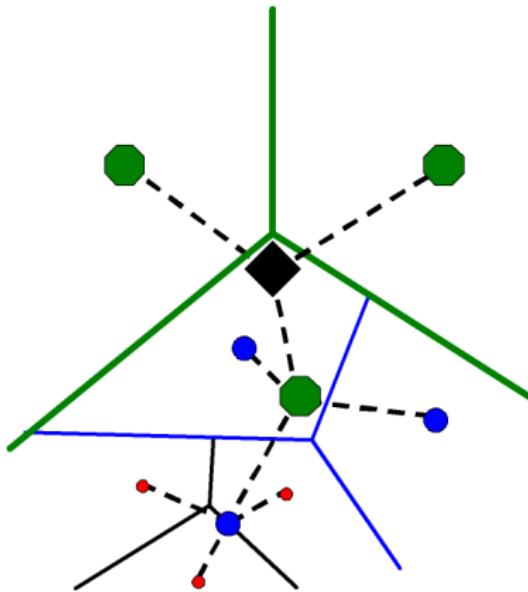
[Fukunaga and Narendra 1975]



- within each cluster, partition data into b clusters

hierarchical k -means (HKM)

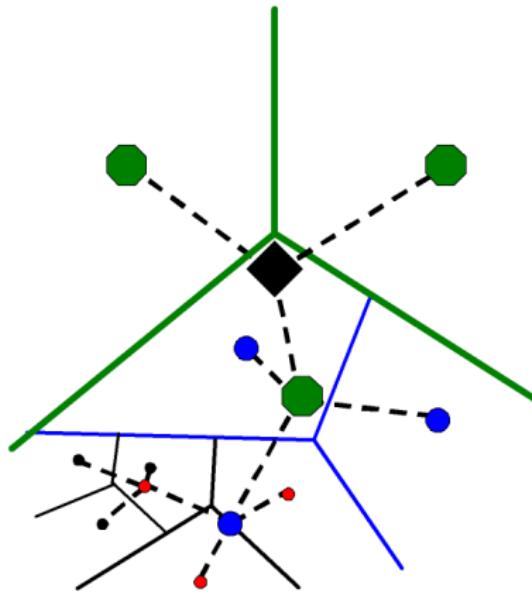
[Fukunaga and Narendra 1975]



- and repeat; b is called the **branching factor**

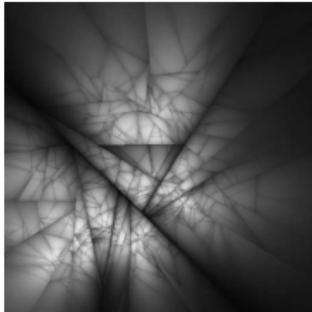
hierarchical k -means (HKM)

[Fukunaga and Narendra 1975]

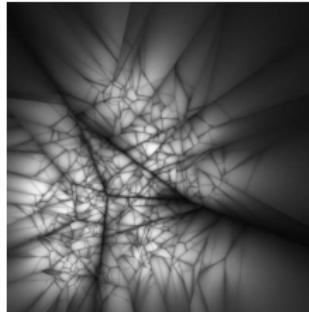


- at ℓ levels, there are b^ℓ total clusters

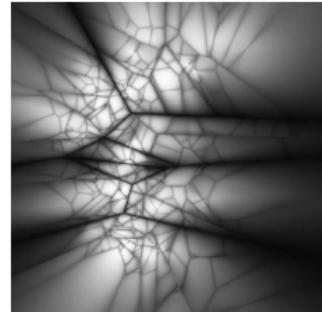
hierarchical k -means



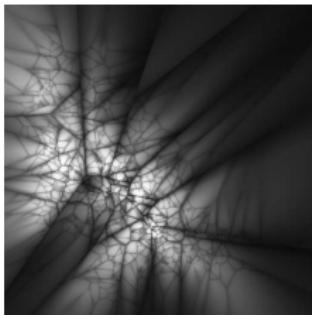
$b = 2$



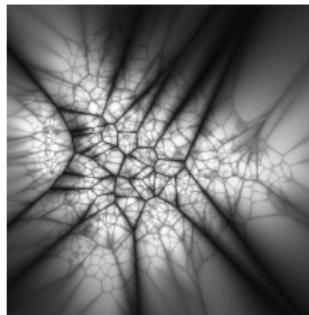
$b = 4$



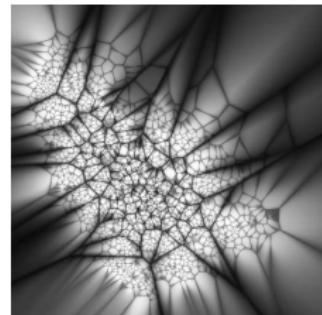
$b = 8$



$b = 16$



$b = 32$

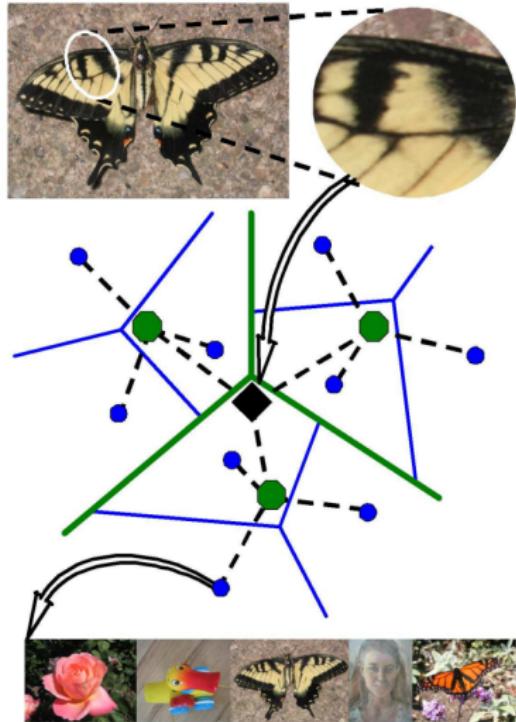


$b = 128$

- intensity: ratio of first to second neighbor distance

vocabulary tree

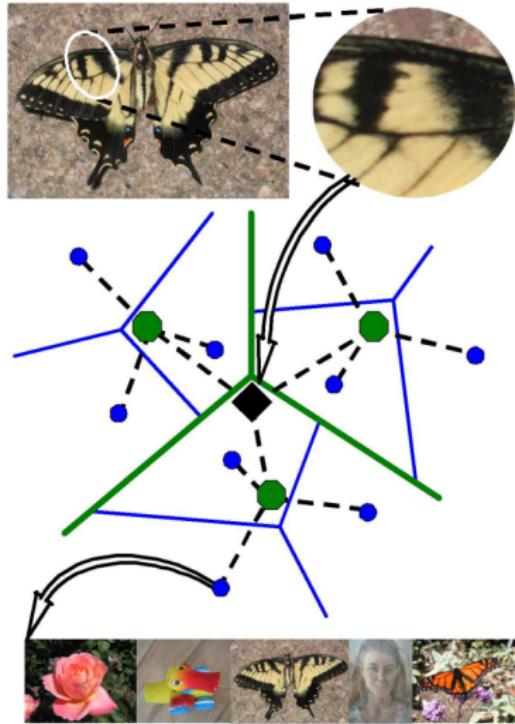
[Nister and Stewenius. CVPR 2006]



- apply k -means hierarchically and build a fine partition tree
- descriptors descend from root to leaves by finding nearest node at each level
- image represented by $x_i = w_i n_i$ as in BoW, but now there is one element per node including internal nodes
- dataset searched by inverted files at leaves

vocabulary tree

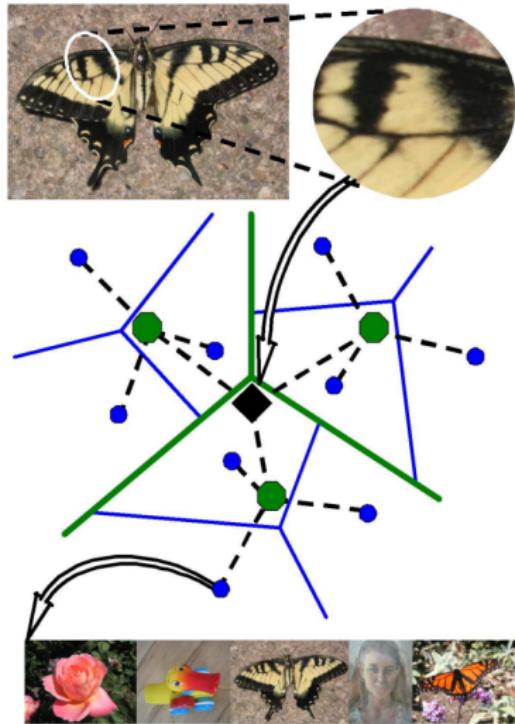
[Nister and Stewenius. CVPR 2006]



- apply k -means hierarchically and build a fine partition tree
- descriptors descend from root to leaves by finding nearest node at each level
- image represented by $x_i = w_i n_i$ as in BoW, but now there is one element per node including internal nodes
- dataset searched by inverted files at leaves

vocabulary tree

[Nister and Stewenius. CVPR 2006]

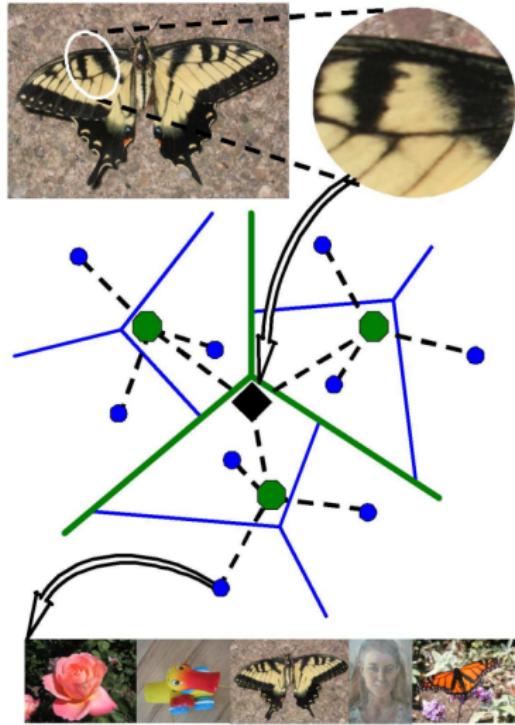


however:

- no principled way of defining w_i across levels
- distortion minimized only locally; points get assigned to leaves that are not globally nearest

vocabulary tree

[Nister and Stewenius. CVPR 2006]

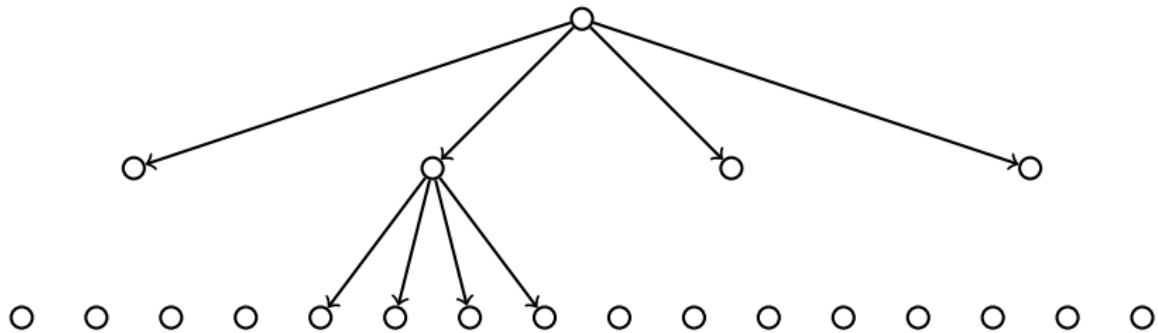


however:

- no principled way of defining w_i across levels
- distortion minimized only locally; points get assigned to leaves that are not globally nearest

approximate k -means (AKM)

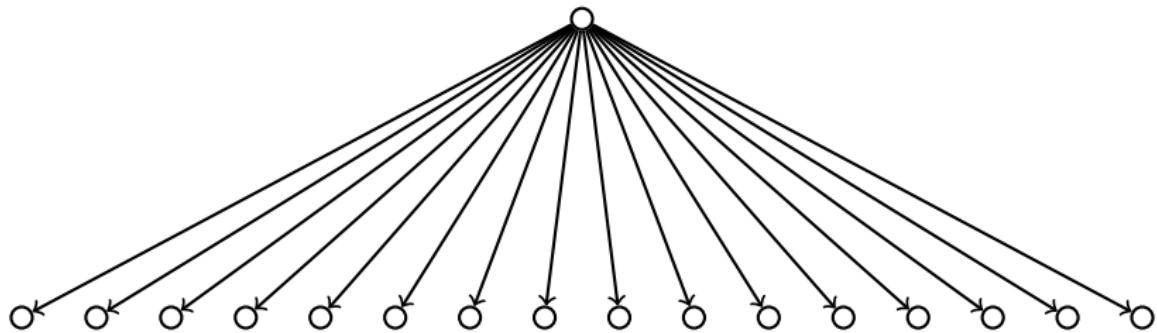
[Philbin et al. 2007]



- with branching factor $b = 10$ and $\ell = 6$ levels, HKM yields $k = 10^6$ visual words; complexity is $O(nbl)$
- search through multiple randomized trees (comparison to HKM in color)

approximate k -means (AKM)

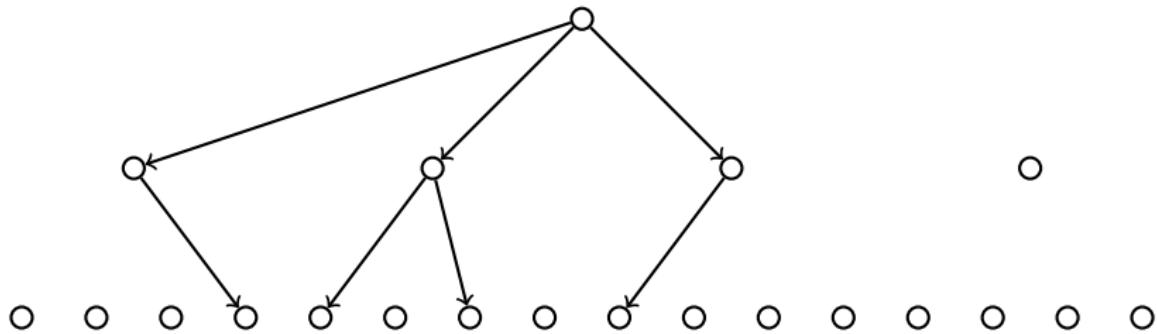
[Philbin et al. 2007]



- flat k -means with e.g. $n = 10^7$ points and $k = 10^6$ centroids is prohibitive; complexity is $O(nk)$, because each assignment is $O(k)$
- search through multiple randomized trees (comparison to HKM in color)

approximate k -means (AKM)

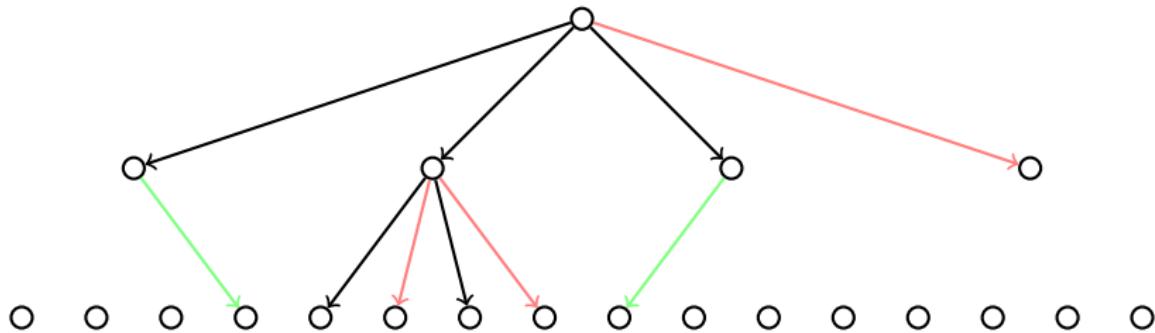
[Philbin et al. 2007]



- **approximate nearest neighbor search** to find the nearest centroid: each assignment is now $O(\log k)$, and complexity drops to $O(n \log k)$
- search through multiple randomized trees (comparison to HKM in color)

approximate k -means (AKM)

[Philbin et al. 2007]



- approximate nearest neighbor search to find the nearest centroid: each assignment is now $O(\log k)$, and complexity drops to $O(n \log k)$
- search through multiple randomized trees (comparison to HKM in color)

approximate k -means (AKM)

- if the sole purpose of the hierarchy is to **accelerate assignment**, both at learning and at search, it is better to use a flat vocabulary combined with a more principled nearest neighbor search method
- however, with appropriate **node weighting**, a hierarchical structure can help (see pyramid matching later on)

pipeline, again

- given **codebook** $C = \{c_1, \dots, c_k\} \subset \mathbb{R}^d$
- given image with descriptors $x_i \in \mathbb{R}^d$ at positions $y_i \in \mathbb{R}^2$, $i = 1, \dots, n$ into $\mathbf{a}_i \in \mathbb{R}^k$
- encode each descriptor x_i into $\mathbf{a}_i \in \mathbb{R}^k$

$$\mathbf{a}_i := F(x_i; C) := (f(x_i, c_1; C), \dots, f(x_i, c_k; C))$$

- pool each spatial region $R_j, j = 1, \dots, m$ into $\mathbf{z}^j \in \mathbb{R}^k$

$$\mathbf{z}^j := g(\{\mathbf{a}_i : y_i \in R_j\})$$

- concatenate into $\mathbf{z} \in \mathbb{R}^{km}$

$$\mathbf{z} := (\mathbf{z}^1; \dots; \mathbf{z}^m)$$

- global pooling is just $m = 1$

pipeline, again

- given **codebook** $C = \{c_1, \dots, c_k\} \subset \mathbb{R}^d$
- given image with descriptors $x_i \in \mathbb{R}^d$ at positions $y_i \in \mathbb{R}^2$, $i = 1, \dots, n$ into $\mathbf{a}_i \in \mathbb{R}^k$
- **encode** each descriptor x_i into $\mathbf{a}_i \in \mathbb{R}^k$

$$\mathbf{a}_i := F(x_i; C) := (f(x_i, c_1; C), \dots, f(x_i, c_k; C))$$

- **pool** each spatial region $R_j, j = 1, \dots, m$ into $\mathbf{z}^j \in \mathbb{R}^k$

$$\mathbf{z}^j := g(\{\mathbf{a}_i : y_i \in R_j\})$$

- concatenate into $\mathbf{z} \in \mathbb{R}^{km}$

$$\mathbf{z} := (\mathbf{z}^1; \dots; \mathbf{z}^m)$$

- **global pooling** is just $m = 1$

pipeline, again

- given **codebook** $C = \{c_1, \dots, c_k\} \subset \mathbb{R}^d$
- given image with descriptors $x_i \in \mathbb{R}^d$ at positions $y_i \in \mathbb{R}^2$, $i = 1, \dots, n$ into $\mathbf{a}_i \in \mathbb{R}^k$
- **encode** each descriptor x_i into $\mathbf{a}_i \in \mathbb{R}^k$

$$\mathbf{a}_i := F(x_i; C) := (f(x_i, c_1; C), \dots, f(x_i, c_k; C))$$

- **pool** each spatial region $R_j, j = 1, \dots, m$ into $\mathbf{z}^j \in \mathbb{R}^k$

$$\mathbf{z}^j := g(\{\mathbf{a}_i : y_i \in R_j\})$$

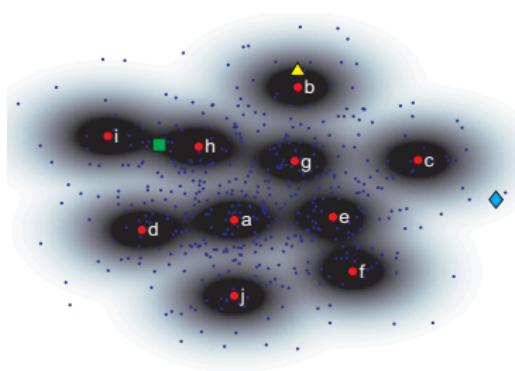
- **concatenate** into $\mathbf{z} \in \mathbb{R}^{km}$

$$\mathbf{z} := (\mathbf{z}^1; \dots; \mathbf{z}^m)$$

- **global pooling** is just $m = 1$

soft assignment

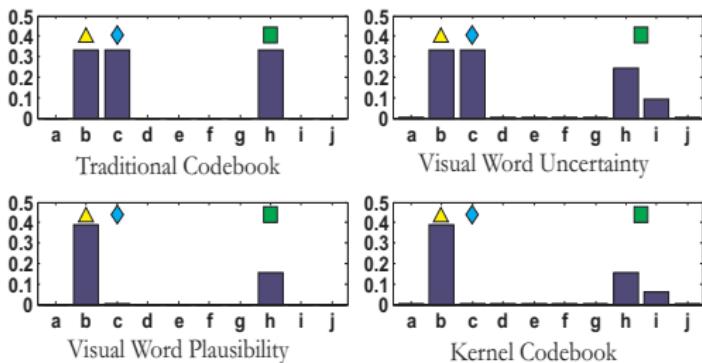
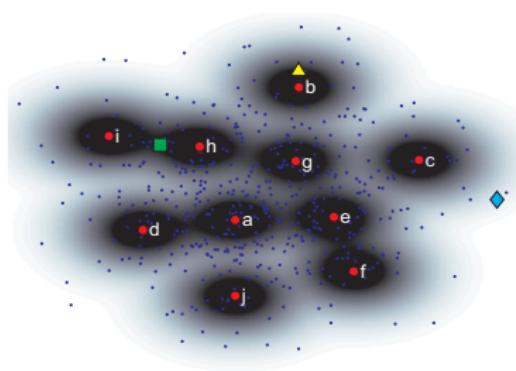
[van Gemert et al. 2008]



- ▲: ok; ■: ambiguous; ◆: not represented
- left: assigned to nearest neighbor; right: to all visual words with different weights
- top: total weight normalized to one; bottom: depends on distance

soft assignment

[van Gemert et al. 2008]



- △: ok; ■: ambiguous; ◆: not represented
- left: assigned to nearest neighbor; right: to all visual words with different weights
- top: total weight normalized to one; bottom: depends on distance

soft assignment

- r -nearest neighbors of x in C : $\text{NN}_C^r(x)$
- kernel function

$$h(x) = h_G(x; \sigma) := \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})(x) \propto \exp\left(-\frac{\|x\|^2}{2\sigma^2}\right)$$

- encoding descriptor x into visual word c

$f(x, c; C)$	visual word	
	nearest	all
fixed weight	$\mathbb{1}[c \in \text{NN}_C^1(x)]$ “BoW”	$\frac{h(x-c)}{\sum_j h(x-c_j)}$ “uncertainty”
variable weight	$\mathbb{1}[c \in \text{NN}_C^1(x)]h(x - c)$ “plausibility”	$h(x - c)$ “kernel”

soft assignment

- r -nearest neighbors of x in C : $\text{NN}_C^r(x)$
- kernel function

$$h(x) = h_G(x; \sigma) := \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})(x) \propto \exp\left(-\frac{\|x\|^2}{2\sigma^2}\right)$$

- encoding descriptor x into visual word c

$f(x, c; C)$	visual word	
	nearest	all
fixed weight	$\mathbb{1}[c \in \text{NN}_C^1(x)]$ “BoW”	$\frac{h(x-c)}{\sum_j h(x-c_j)}$ “uncertainty”
variable weight	$\mathbb{1}[c \in \text{NN}_C^1(x)]h(x - c)$ “plausibility”	$h(x - c)$ “kernel”

soft assignment

- r -nearest neighbors of x in C : $\text{NN}_C^r(x)$
- kernel function

$$h(x) = h_G(x; \sigma) := \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})(x) \propto \exp\left(-\frac{\|x\|^2}{2\sigma^2}\right)$$

- encoding descriptor x into visual word c

$f(x, c; C)$	visual word	
	nearest	all
fixed weight	$\mathbb{1}[c \in \text{NN}_C^1(x)]$ “BoW”	$\frac{h(x-c)}{\sum_j h(x-c_j)}$ “uncertainty”
variable weight	$\mathbb{1}[c \in \text{NN}_C^1(x)]h(x - c)$ “plausibility”	$h(x - c)$ “kernel”

soft assignment

- on classification: best model is “uncertainty”

$$f(x, c; C) = \frac{h(x - c)}{\sum_j h(x - c_j)}$$

- it is better to contribute to visual words even if all are far away
- we shall see this is the softmax of negative distances $-\|x - c\|^2$
- it is also the responsibility of visual word c for descriptor x in a Gaussian mixture model with C as components

soft assignment

- on classification: best model is “uncertainty”

$$f(x, c; C) = \frac{h(x - c)}{\sum_j h(x - c_j)}$$

- it is better to contribute to visual words even if all are far away
- we shall see this is the softmax of negative distances $-\|x - c\|^2$
- it is also the responsibility of visual word c for descriptor x in a Gaussian mixture model with C as components

soft assignment

[Liu et al. 2011]

- on classification: it turns out, it is better to limit contributions to r nearest neighbors

$$f(x, c; C) = \mathbb{1}[c \in \text{NN}_C^r(x)] \frac{h(x - c)}{\sum_j h(x - c_j)}$$

- this is attributed to respecting the manifold structure of the data, and it superior to more expensive sparse coding that have been proposed in the meantime

soft assignment

[Philbin et al. 2008]

- on retrieval: “kernel” is followed on r nearest neighbors

$$f(x, c; C) = \mathbb{1}[c \in \text{NN}_C^r(x)] h(x - c)$$

- it is better to discard descriptors if they are not well represented
- r should be small: this applies to dataset images and increases the required index space and query time (including spatial matching) by r

soft assignment

[Philbin et al. 2008]

- on retrieval: “kernel” is followed on r nearest neighbors

$$f(x, c; C) = \mathbb{1}[c \in \text{NN}_C^r(x)] h(x - c)$$

- it is better to discard descriptors if they are not well represented
- r should be small: this applies to dataset images and increases the required index space and query time (including spatial matching) by r

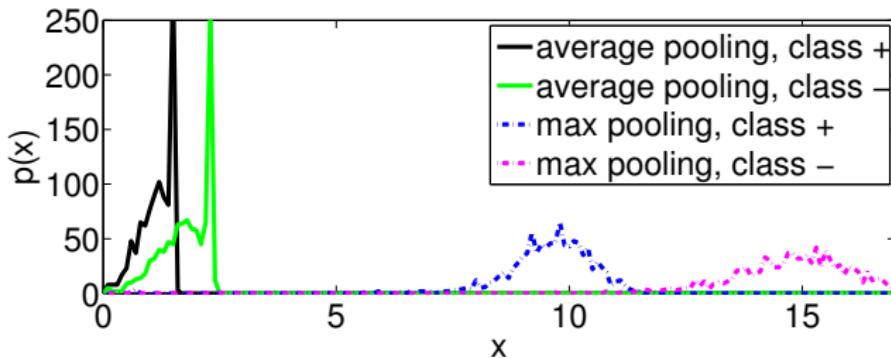
multiple assignment

[Jégou et al. 2010]

- on retrieval: same as before, but now applies only to query images
- $f(x, c; C)$ further limited to visual words at distance $\leq \alpha d_1$ from x , where d_1 is the distance of $\text{NN}_C^1(x)$
- index space maintained as in standard hard assignment, but query time is still increased by r

max pooling vs. average pooling

[Boureau et al. 2010]



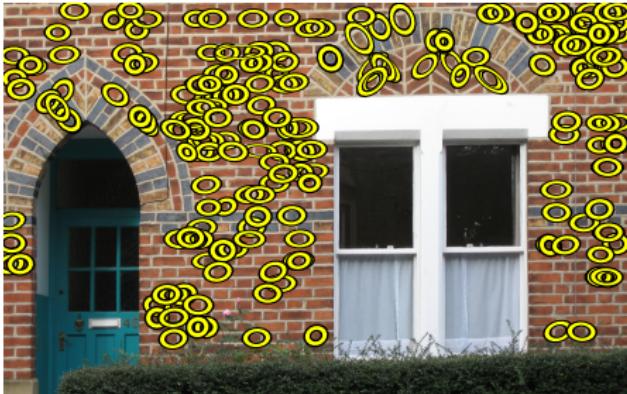
- on classification: max-pooling superior to average pooling

$$g_{\max}(A) = \left(\max_{\mathbf{a} \in A} a_1, \dots, \max_{\mathbf{a} \in A} a_k \right) \quad g_{\text{avg}}(A) = \frac{1}{|A|} \sum_{\mathbf{a} \in A} \mathbf{a}$$

- with max-pooling, SVM with linear and nonlinear kernel perform nearly the same

burstiness

[Jégou et al. 2009]

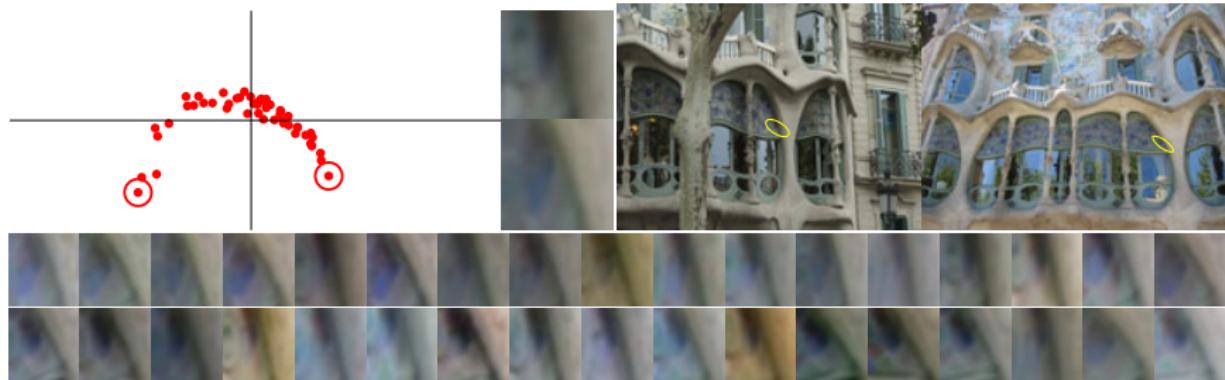


- **burstiness:** descriptors appear more frequently than a statistically independent model predicts; it hurts performance because bursty features dominate the image similarity
- **on retrieval:** the situation is more complex here; max-pooling would be like keeping only one representative per cell, average pooling like keeping all, but none is the best choice

beyond codebooks

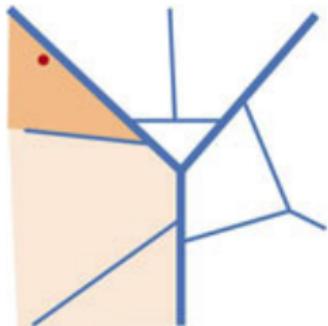
learning cell shapes

[Mikulik et al. 2010]

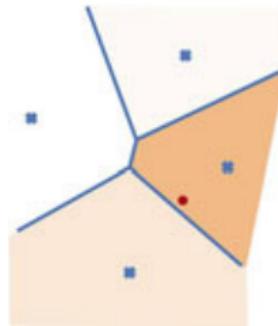


- **on retrieval:** matched across images in an entire dataset, features are connected into **feature tracks**
- feature tracks have curved shape in descriptor space, contrary to the Gaussian assumption—an example of **manifold structure**
- even if such structure cannot be captured by k -means, cells can still be connected via feature tracks → vocabulary of **16M words**

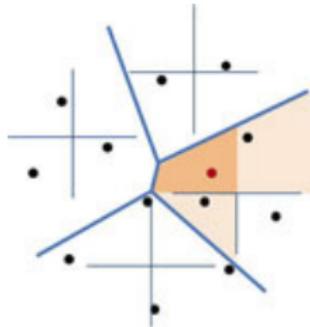
learning cell shapes



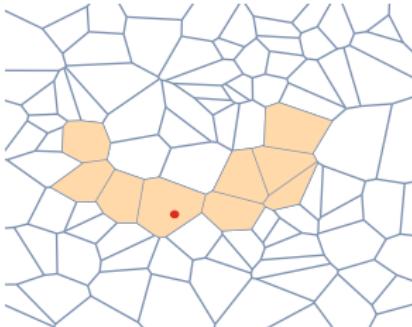
HKM



soft assignment



Hamming



learned

descriptor matching

- **on retrieval:** given two images with descriptors $X, Y \subset \mathbb{R}^d$, and recalling $X_c = \{x \in X : q(x) = c\}$, bag-of-words similarity on C is

$$\begin{aligned}s_{\text{BoW}}(X, Y) &\propto \sum_{c \in C} w_c |X_c| |Y_c| \\&= \sum_{c \in C} w_c \sum_{x \in X_c} \sum_{y \in Y_c} 1\end{aligned}$$

- if descriptors are available in some form (**more space**), it is better to use a more general function of the form

$$K(X, Y) := \gamma(X)\gamma(Y) \sum_{c \in C} w_c M(X_c, Y_c)$$

where M is a **within-cell** matching function and $\gamma(X)$ serves for normalization

descriptor matching

- **on retrieval:** given two images with descriptors $X, Y \subset \mathbb{R}^d$, and recalling $X_c = \{x \in X : q(x) = c\}$, bag-of-words similarity on C is

$$\begin{aligned}s_{\text{BoW}}(X, Y) &\propto \sum_{c \in C} w_c |X_c| |Y_c| \\&= \sum_{c \in C} w_c \sum_{x \in X_c} \sum_{y \in Y_c} 1\end{aligned}$$

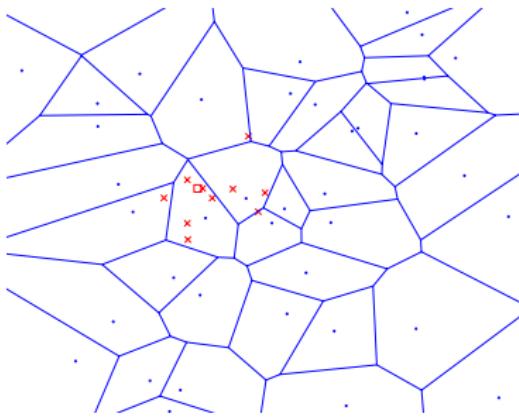
- if descriptors are available in some form (**more space**), it is better to use a more general function of the form

$$K(X, Y) := \gamma(X)\gamma(Y) \sum_{c \in C} w_c M(X_c, Y_c)$$

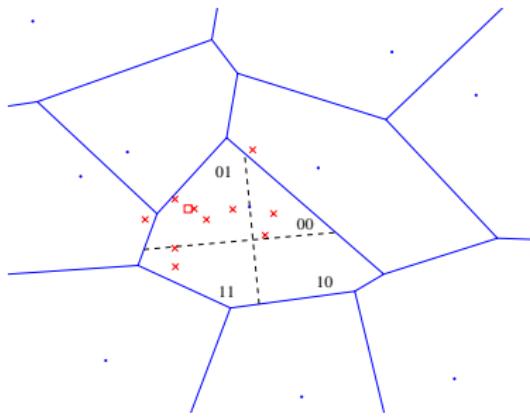
where M is a **within-cell** matching function and $\gamma(X)$ serves for normalization

Hamming embedding (HE)

[Jégou et al. 2008]



fine vocabulary



Hamming embedding

- each descriptor x is **binarized** into $b(x) \in \{0, 1\}^d$
- pairs within cells are kept only if **Hamming distance** is at most τ

$$M_{\text{HE}}(X_c, Y_c) := \sum_{x \in X_c} \sum_{y \in Y_c} \mathbb{1}[d_{\text{H}}(b(x), b(y)) \leq \tau]$$

aggregated selective match kernel (ASMK)

[Tolias et al. 2013]

- borrow from HE the idea that descriptor pairs are **selected** by a nonlinear function

$$M_{\text{HE}}(X_c, Y_c) := \sum_{x \in X_c} \sum_{y \in Y_c} \mathbb{1}[d_{\mathcal{H}}(b(x), b(y)) \leq \tau]$$

- borrow from VLAD the idea that residuals are **pooled** per cell

$$M_{\text{VLAD}}(X_c, Y_c) := V(X_c)^\top V(Y_c) = \sum_{x \in X_c} \sum_{y \in Y_c} r(x)^\top r(y)$$

- combine pooling **within** cells with selectivity **between** cells

$$M_{\text{ASMK}}(X_c, Y_c) := \sigma_\alpha(\hat{V}(X_c)^\top \hat{V}(Y_c))$$

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

aggregated selective match kernel (ASMK)

[Tolias et al. 2013]

- borrow from HE the idea that descriptor pairs are **selected** by a nonlinear function

$$M_{\text{HE}}(X_c, Y_c) := \sum_{x \in X_c} \sum_{y \in Y_c} \mathbb{1}[d_{\mathcal{H}}(b(x), b(y)) \leq \tau]$$

- borrow from VLAD the idea that residuals are **pooled** per cell

$$M_{\text{VLAD}}(X_c, Y_c) := V(X_c)^\top V(Y_c) = \sum_{x \in X_c} \sum_{y \in Y_c} r(x)^\top r(y)$$

- combine pooling **within** cells with selectivity **between** cells

$$M_{\text{ASMK}}(X_c, Y_c) := \sigma_\alpha(\hat{V}(X_c)^\top \hat{V}(Y_c))$$

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

aggregated selective match kernel (ASMK)



- apart from saving space, pooling and normalizing per cell helps fight burstiness
- still, unlike VLAD, due to the nonlinearity we cannot have a low dimensional embedding

efficient match kernels (EMK)

[Bo and Sminchisescu. NIPS 2009]

- on classification: given two images with descriptors $X, Y \subset \mathbb{R}^d$, bag-of-words similarity on C is

$$s_{\text{BoW}}(X, Y) \propto \sum_{c \in C} |X_c| |Y_c| = \sum_{x \in X} \sum_{y \in Y} \mathbb{1}[q(x) = q(y)]$$

- use a continuous function $\kappa(x, y)$ instead, with no codebook

$$K(X, Y) := \gamma(X) \gamma(Y) \sum_{x \in X} \sum_{y \in Y} \kappa(x, y)$$

- derive an approximate finite-dimensional feature map ϕ such that $\kappa(x, y) = \phi(x)^\top \phi(y)$, and

$$K(X, Y) = \left(\gamma(X) \sum_{x \in X} \phi(x) \right) \left(\gamma(Y) \sum_{y \in Y} \phi(y) \right) = \Phi(X)^\top \Phi(Y)$$

efficient match kernels (EMK)

[Bo and Sminchisescu. NIPS 2009]

- on classification: given two images with descriptors $X, Y \subset \mathbb{R}^d$, bag-of-words similarity on C is

$$s_{\text{BoW}}(X, Y) \propto \sum_{c \in C} |X_c| |Y_c| = \sum_{x \in X} \sum_{y \in Y} \mathbb{1}[q(x) = q(y)]$$

- use a continuous function $\kappa(x, y)$ instead, with no codebook

$$K(X, Y) := \gamma(X) \gamma(Y) \sum_{x \in X} \sum_{y \in Y} \kappa(x, y)$$

- derive an approximate finite-dimensional feature map ϕ such that $\kappa(x, y) = \phi(x)^\top \phi(y)$, and

$$K(X, Y) = \left(\gamma(X) \sum_{x \in X} \phi(x) \right) \left(\gamma(Y) \sum_{y \in Y} \phi(y) \right) = \Phi(X)^\top \Phi(Y)$$

efficient match kernels (EMK)

[Bo and Sminchisescu. NIPS 2009]

- on classification: given two images with descriptors $X, Y \subset \mathbb{R}^d$, bag-of-words similarity on C is

$$s_{\text{BoW}}(X, Y) \propto \sum_{c \in C} |X_c| |Y_c| = \sum_{x \in X} \sum_{y \in Y} \mathbb{1}[q(x) = q(y)]$$

- use a continuous function $\kappa(x, y)$ instead, with no codebook

$$K(X, Y) := \gamma(X) \gamma(Y) \sum_{x \in X} \sum_{y \in Y} \kappa(x, y)$$

- derive an approximate finite-dimensional feature map ϕ such that $\kappa(x, y) = \phi(x)^\top \phi(y)$, and

$$K(X, Y) = \left(\gamma(X) \sum_{x \in X} \phi(x) \right) \left(\gamma(Y) \sum_{y \in Y} \phi(y) \right) = \Phi(X)^\top \Phi(Y)$$

efficient match kernels (EMK)

- given a function $K(X, Y)$ on sets X, Y in the form of a pairwise sum of **nonlinear** functions $\kappa(x, y)$ of the elements $x \in X, y \in Y$, we can decompose it into an inner product of $\Phi(X), \Phi(Y)$
- this can be done by
 - **encoding** $x \mapsto \phi(x)$
 - **pooling** $X \mapsto \Phi(X) = \gamma(X) \sum_{x \in X} \phi(x)$
- this is always possible for **positive-definite** functions κ but ϕ may be infinite-dimensional; in nonlinear SVM, it is only implicit through κ
- here, we are interested in an **explicit, low-dimensional** feature map ϕ , which can be designed or learned

efficient match kernels (EMK)

- given a function $K(X, Y)$ on sets X, Y in the form of a pairwise sum of **nonlinear** functions $\kappa(x, y)$ of the elements $x \in X, y \in Y$, we can decompose it into an inner product of $\Phi(X), \Phi(Y)$
- this can be done by
 - **encoding** $x \mapsto \phi(x)$
 - **pooling** $X \mapsto \Phi(X) = \gamma(X) \sum_{x \in X} \phi(x)$
- this is always possible for **positive-definite** functions κ but ϕ may be infinite-dimensional; in nonlinear SVM, it is only implicit through κ
- here, we are interested in an **explicit, low-dimensional** feature map ϕ , which can be designed or learned

efficient match kernels (EMK)

- given a function $K(X, Y)$ on sets X, Y in the form of a pairwise sum of **nonlinear** functions $\kappa(x, y)$ of the elements $x \in X, y \in Y$, we can decompose it into an inner product of $\Phi(X), \Phi(Y)$
- this can be done by
 - **encoding** $x \mapsto \phi(x)$
 - **pooling** $X \mapsto \Phi(X) = \gamma(X) \sum_{x \in X} \phi(x)$
- this is always possible for **positive-definite** functions κ but ϕ may be infinite-dimensional; in nonlinear SVM, it is only implicit through κ
- here, we are interested in an **explicit, low-dimensional** feature map ϕ , which can be designed or learned

pyramid matching

histogram intersection

[Swain and Ballard 1991]

- the sum $\sum_{x \in X_c} \sum_{y \in Y_c} 1$ appearing in $s_{\text{BoW}}(X, Y)$ implies an **all-all** matching; it is often preferable to have an **one-one** matching instead



- given two histograms x, y of b bins, their **histogram intersection** is

$$\kappa_{\text{HI}}(x, y) = \sum_{i=1}^b \min(x_i, y_i)$$

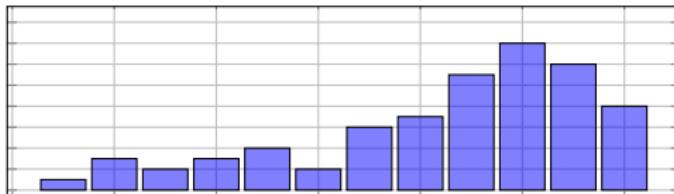
- this is related to ℓ_1 distance by

$$\|x - y\|_1 = \|x\|_1 + \|y\|_1 - 2\kappa_{\text{HI}}(x, y)$$

histogram intersection

[Swain and Ballard 1991]

- the sum $\sum_{x \in X_c} \sum_{y \in Y_c} 1$ appearing in $s_{\text{BoW}}(X, Y)$ implies an **all-all** matching; it is often preferable to have an **one-one** matching instead



- given two histograms x, y of b bins, their **histogram intersection** is

$$\kappa_{\text{HI}}(x, y) = \sum_{i=1}^b \min(x_i, y_i)$$

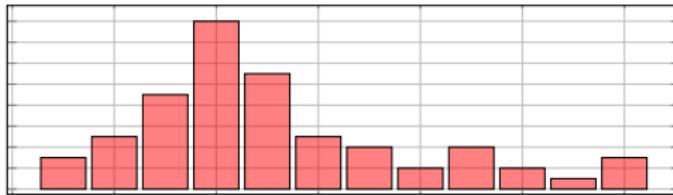
- this is related to ℓ_1 distance by

$$\|x - y\|_1 = \|x\|_1 + \|y\|_1 - 2\kappa_{\text{HI}}(x, y)$$

histogram intersection

[Swain and Ballard 1991]

- the sum $\sum_{x \in X_c} \sum_{y \in Y_c} 1$ appearing in $s_{\text{BoW}}(X, Y)$ implies an **all-all** matching; it is often preferable to have an **one-one** matching instead



- given two histograms x, y of b bins, their **histogram intersection** is

$$\kappa_{\text{HI}}(x, y) = \sum_{i=1}^b \min(x_i, y_i)$$

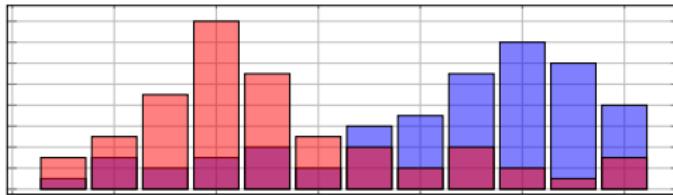
- this is related to ℓ_1 distance by

$$\|x - y\|_1 = \|x\|_1 + \|y\|_1 - 2\kappa_{\text{HI}}(x, y)$$

histogram intersection

[Swain and Ballard 1991]

- the sum $\sum_{x \in X_c} \sum_{y \in Y_c} 1$ appearing in $s_{\text{BoW}}(X, Y)$ implies an **all-all** matching; it is often preferable to have an **one-one** matching instead



- given two histograms x, y of b bins, their **histogram intersection** is

$$\kappa_{\text{HI}}(x, y) = \sum_{i=1}^b \min(x_i, y_i)$$

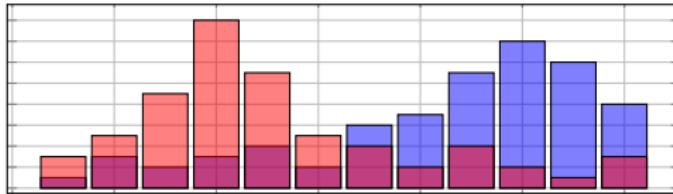
- this is related to ℓ_1 distance by

$$\|x - y\|_1 = \|x\|_1 + \|y\|_1 - 2\kappa_{\text{HI}}(x, y)$$

histogram intersection

[Swain and Ballard 1991]

- the sum $\sum_{x \in X_c} \sum_{y \in Y_c} 1$ appearing in $s_{\text{BoW}}(X, Y)$ implies an **all-all** matching; it is often preferable to have an **one-one** matching instead



- given two histograms x, y of b bins, their **histogram intersection** is

$$\kappa_{\text{HI}}(x, y) = \sum_{i=1}^b \min(x_i, y_i)$$

- this is related to ℓ_1 distance by

$$\|x - y\|_1 = \|x\|_1 + \|y\|_1 - 2\kappa_{\text{HI}}(x, y)$$

pyramid match kernel (PMK)

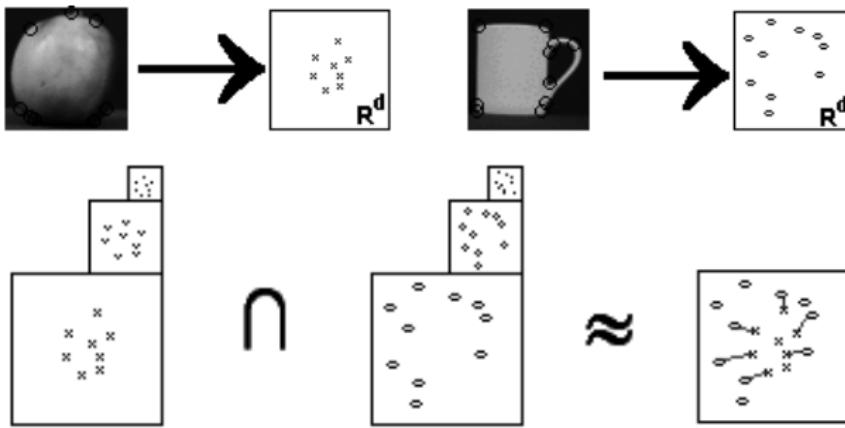
[Grauman and Darrell 2005]



- given the descriptors of two images as point sets X, Y in \mathbb{R}^d
- a weighted sum of histogram intersections at different levels approximates their optimal pairwise matching

pyramid match kernel (PMK)

[Grauman and Darrell 2005]



- given the descriptors of two images as point sets X, Y in \mathbb{R}^d
- a weighted sum of histogram intersections at different levels approximates their optimal pairwise matching

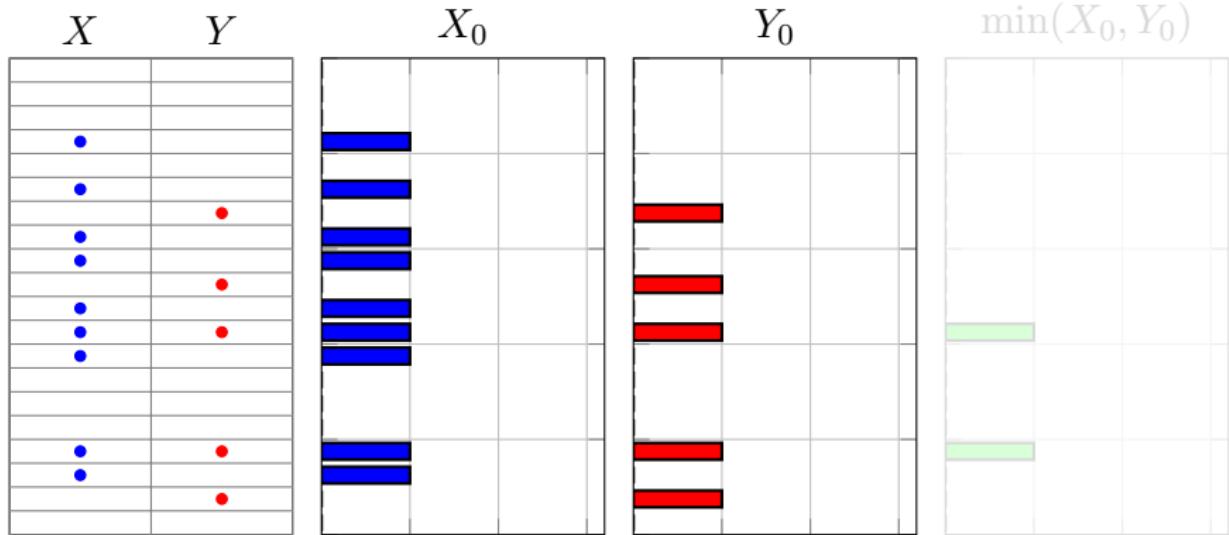
pyramid match kernel (PMK)



- 1d point sets X, Y on grid of size 1



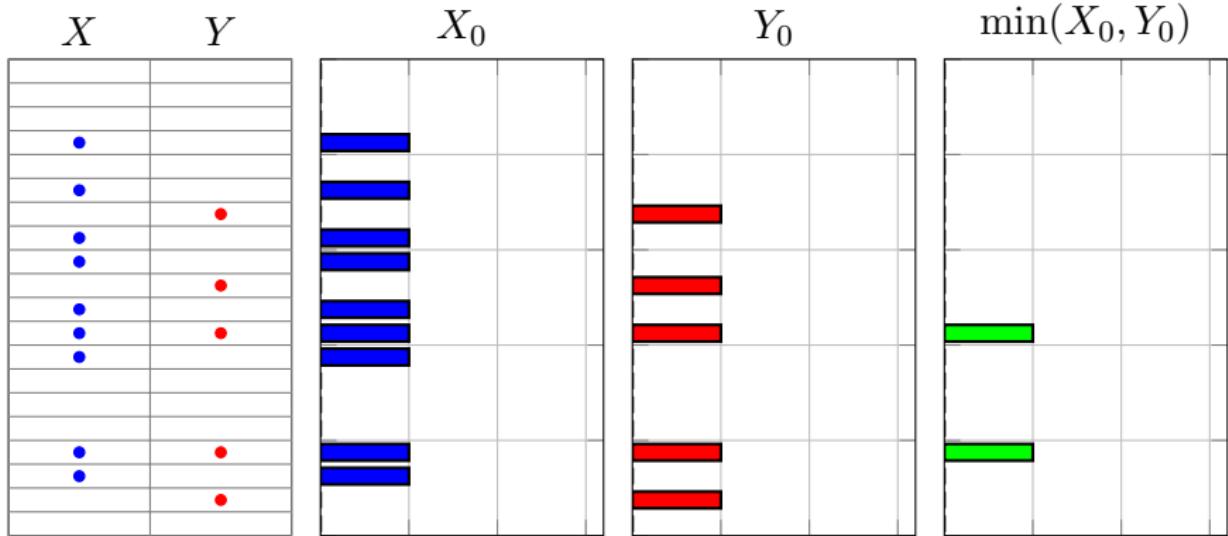
pyramid match kernel (PMK)



- 1d point sets X, Y on grid of size 1 - level 0 histograms



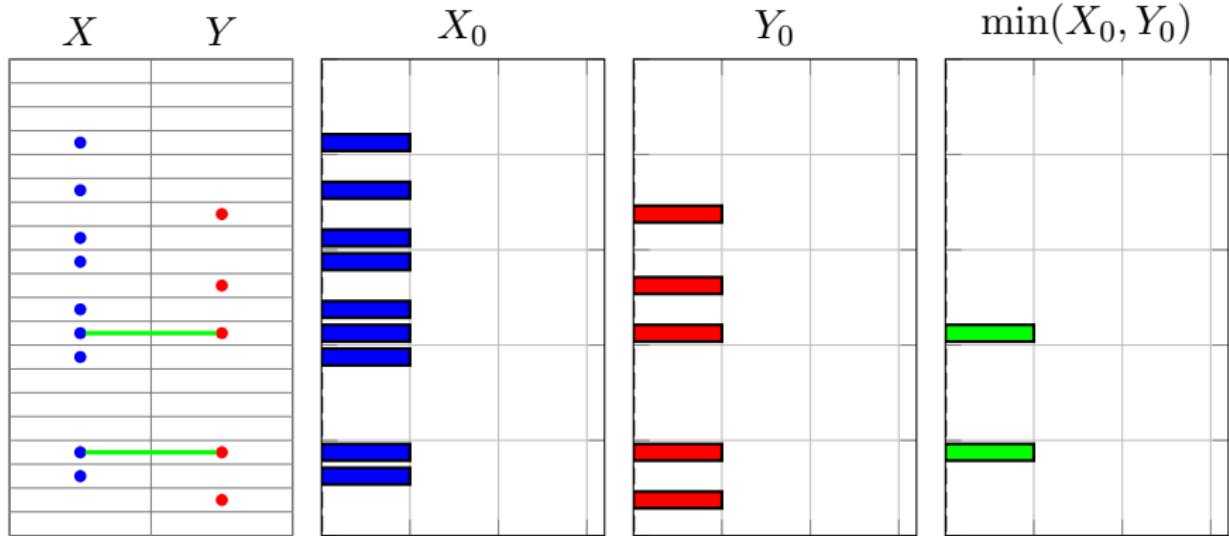
pyramid match kernel (PMK)



- 1d point sets X, Y on grid of size 1 - level 0 histograms - intersection

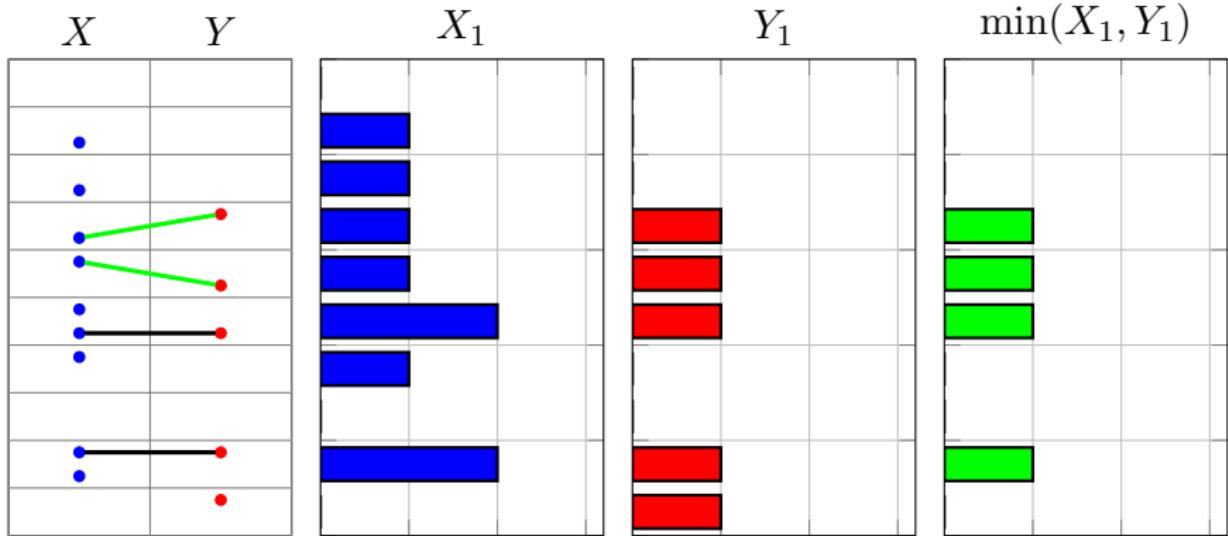


pyramid match kernel (PMK)



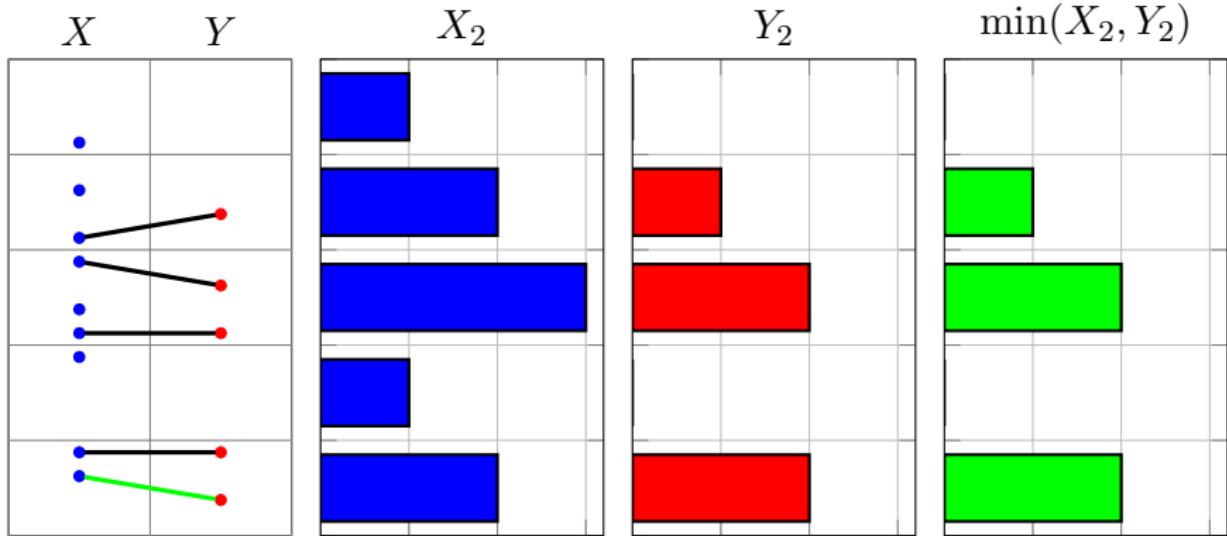
- 1d point sets X, Y on grid of size 1 - level 0 histograms - intersection
- (2 matches weighted by 1)
- total score 2×1

pyramid match kernel (PMK)



- 1d point sets X, Y on grid of size 2 - level 1 histograms - intersection
- (2 matches weighted by 1) + (2 weighted by $\frac{1}{2}$)
- total score $2 \times 1 + 2 \times \frac{1}{2}$

pyramid match kernel (PMK)



- 1d point sets X, Y on grid of size 4 - level 2 histograms - intersection
- $(2 \text{ matches weighted by } 1) + (2 \text{ weighted by } \frac{1}{2}) + (1 \text{ weighted by } \frac{1}{4})$
- total score $2 \times 1 + 2 \times \frac{1}{2} + 1 \times \frac{1}{4}$

pyramid match kernel (PMK)

- given a set $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$, where distances of elements range in $[1, D]$
- let X_i be a histogram of X in \mathbb{R}^d on a regular grid of side length 2^i
- i ranges from -1 , where each bin has at most one element, to $L = \lceil \log_2 D \rceil$, where X is contained in a single bin
- given two images with descriptors $X, Y \subset \mathbb{R}^d$, their **pyramid match** is

$$\begin{aligned} K_{\Delta}(X, Y) &= \gamma(X)\gamma(Y) \sum_{i=0}^L \frac{1}{2^i} (\kappa_{\text{HI}}(X_i, Y_i) - \kappa_{\text{HI}}(X_{i-1}, Y_{i-1})) \\ &= \gamma(X)\gamma(Y) \left(\frac{1}{2^L} \kappa_{\text{HI}}(X_L, Y_L) + \sum_{i=0}^{L-1} \frac{1}{2^{i+1}} \kappa_{\text{HI}}(X_i, Y_i) \right) \end{aligned}$$

where $\gamma(X)$ serves for normalization

pyramid match kernel (PMK)

- given a set $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$, where distances of elements range in $[1, D]$
- let X_i be a histogram of X in \mathbb{R}^d on a regular grid of side length 2^i
- i ranges from -1 , where each bin has at most one element, to $L = \lceil \log_2 D \rceil$, where X is contained in a single bin
- given two images with descriptors $X, Y \subset \mathbb{R}^d$, their **pyramid match** is

$$\begin{aligned} K_{\Delta}(X, Y) &= \gamma(X)\gamma(Y) \sum_{i=0}^L \frac{1}{2^i} (\kappa_{\text{HI}}(X_i, Y_i) - \kappa_{\text{HI}}(X_{i-1}, Y_{i-1})) \\ &= \gamma(X)\gamma(Y) \left(\frac{1}{2^L} \kappa_{\text{HI}}(X_L, Y_L) + \sum_{i=0}^{L-1} \frac{1}{2^{i+1}} \kappa_{\text{HI}}(X_i, Y_i) \right) \end{aligned}$$

where $\gamma(X)$ serves for normalization

pyramid match kernel (PMK)

- given a set $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$, where distances of elements range in $[1, D]$
- let X_i be a histogram of X in \mathbb{R}^d on a regular grid of side length 2^i
- i ranges from -1 , where each bin has at most one element, to $L = \lceil \log_2 D \rceil$, where X is contained in a single bin
- given two images with descriptors $X, Y \subset \mathbb{R}^d$, their **pyramid match** is

$$\begin{aligned} K_{\Delta}(X, Y) &= \gamma(X)\gamma(Y) \sum_{i=0}^L \frac{1}{2^i} (\kappa_{\text{HI}}(X_i, Y_i) - \kappa_{\text{HI}}(X_{i-1}, Y_{i-1})) \\ &= \gamma(X)\gamma(Y) \left(\frac{1}{2^L} \kappa_{\text{HI}}(X_L, Y_L) + \sum_{i=0}^{L-1} \frac{1}{2^{i+1}} \kappa_{\text{HI}}(X_i, Y_i) \right) \end{aligned}$$

where $\gamma(X)$ serves for normalization

PMK as an embedding

[Indyk and Thaper 2003]

- there is an explicit embedding for κ_{HI} , therefore also for κ_{Δ}
- if $|X| \leq |Y|$ and $\pi : X \rightarrow Y$ is one-to-one, then $K_{\Delta}(X, Y)$ approximates the optimal pairwise matching

$$\max_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1^{-1}$$

- this was first shown on the earth mover's distance

$$\min_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1$$

- but PMK is a similarity measure; it allows partial matching and does not penalize clutter, except for the normalization

PMK as an embedding

[Indyk and Thaper 2003]

- there is an explicit embedding for κ_{HI} , therefore also for κ_{Δ}
- if $|X| \leq |Y|$ and $\pi : X \rightarrow Y$ is one-to-one, then $K_{\Delta}(X, Y)$ approximates the optimal pairwise matching

$$\max_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1^{-1}$$

- this was first shown on the earth mover's distance

$$\min_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1$$

- but PMK is a similarity measure; it allows partial matching and does not penalize clutter, except for the normalization

PMK as an embedding

[Indyk and Thaper 2003]

- there is an explicit embedding for κ_{HI} , therefore also for κ_{Δ}
- if $|X| \leq |Y|$ and $\pi : X \rightarrow Y$ is one-to-one, then $K_{\Delta}(X, Y)$ approximates the optimal pairwise matching

$$\max_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1^{-1}$$

- this was first shown on the earth mover's distance

$$\min_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1$$

- but PMK is a similarity measure; it allows partial matching and does not penalize clutter, except for the normalization

PMK as an embedding

[Indyk and Thaper 2003]

- there is an explicit embedding for κ_{HI} , therefore also for κ_{Δ}
- if $|X| \leq |Y|$ and $\pi : X \rightarrow Y$ is one-to-one, then $K_{\Delta}(X, Y)$ approximates the optimal pairwise matching

$$\max_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1^{-1}$$

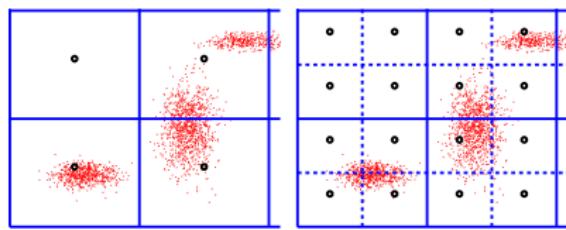
- this was first shown on the earth mover's distance

$$\min_{\pi} \sum_{x \in X} \|x - \pi(x)\|_1$$

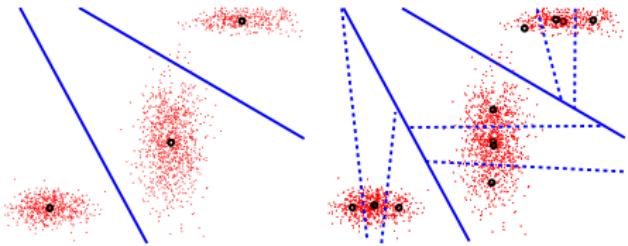
- but PMK is a similarity measure; it allows partial matching and does not penalize clutter, except for the normalization

PMK and vocabulary tree

[Grauman and Darrell 2007]



uniform bins

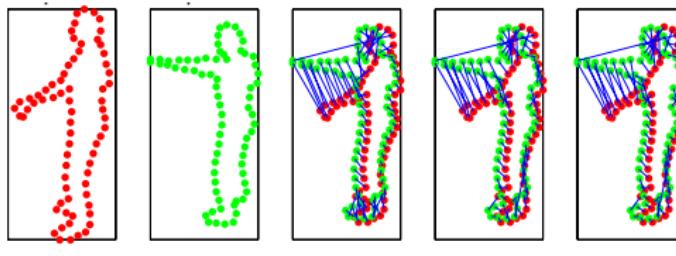


vocabulary-guided bins

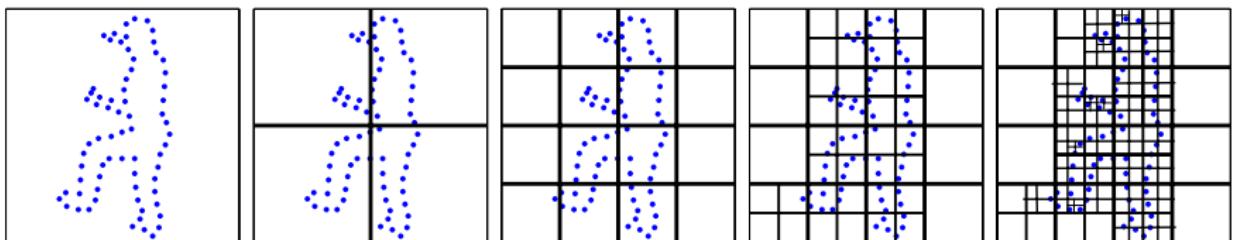
- replace regular grid with hierarchical vocabulary cells
- compared to vocabulary tree, there is a principle in assigning cell weights
- still, its approximation quality suffers at high dimensions

PMK and spatial matching

[Grauman and Darrell 2004]



optimal matching

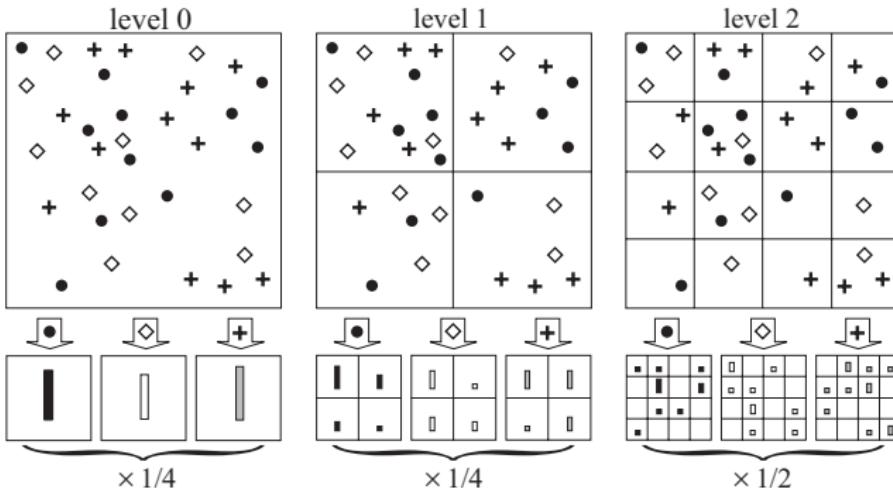


representation

- same idea, applied to image 2d coordinate space for spatial matching
- matching cost is only based on point coordinates; no appearance

spatial pyramid matching (SPM)

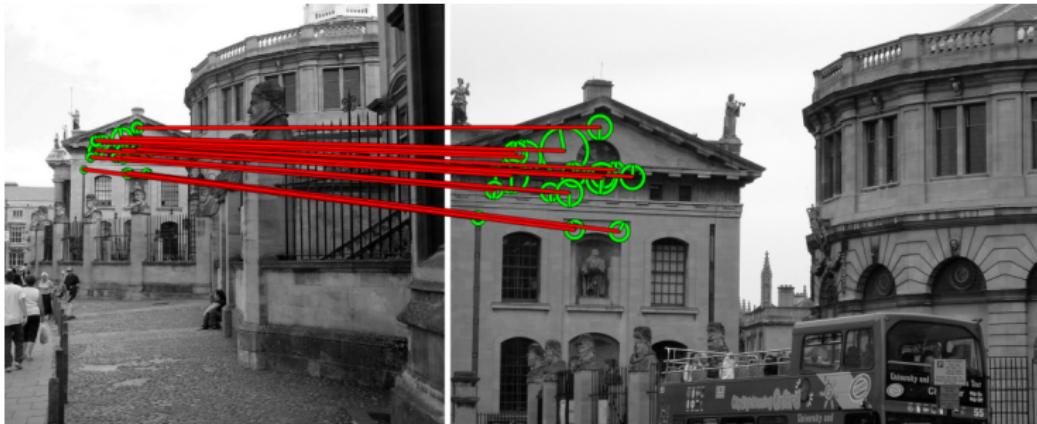
[Lazebnik et al. 2006]



- coupled with BoW, it is a set of joint appearance-geometry histograms
- robust to deformation but not invariant to transformations; applied to global scene classification

Hough pyramid matching (HPM)

[Tolias and Avrithis 2011]

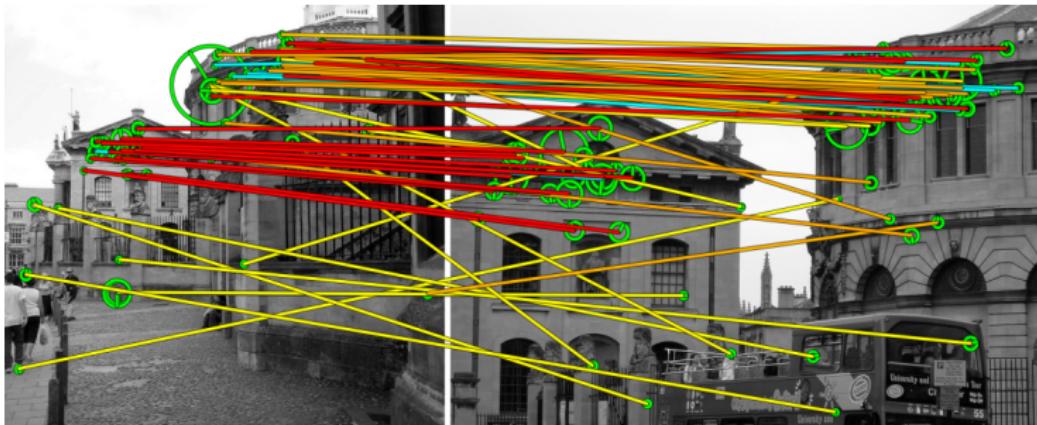


fast spatial matching

- work with a single set of correspondences instead of two sets of features
- determine a transformation hypothesis by a pair of features and then use histograms to collect votes in the transformation space

Hough pyramid matching (HPM)

[Tolias and Avrithis 2011]

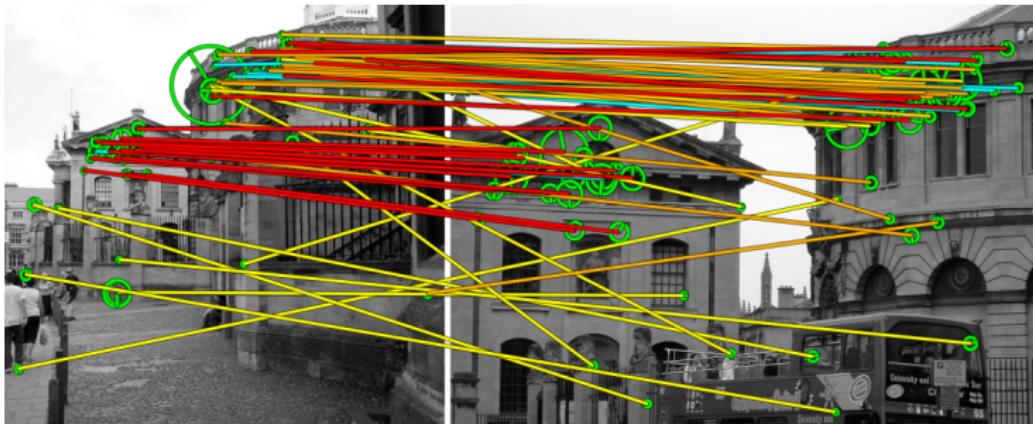


Hough pyramid matching

- work with a single set of correspondences instead of two sets of features
- determine a transformation hypothesis by a pair of features and then use histograms to collect votes in the transformation space

Hough pyramid matching (HPM)

[Tolias and Avrithis 2011]



Hough pyramid matching

- work with a single set of correspondences instead of two sets of features
- determine a transformation hypothesis by a pair of features and then use histograms to collect votes in the transformation space

Hough pyramid matching (HPM)

- a **local feature** p in image P has position $\mathbf{t}(p)$, scale $s(p)$ and orientation $\theta(p)$ given by matrix $R(p) \in \mathbb{R}^{2 \times 2}$

$$F(p) = \begin{pmatrix} s(p)R(p) & \mathbf{t}(p) \\ \mathbf{0}^\top & 1 \end{pmatrix}$$

- a **correspondence** $c = (p, q)$ is a pair of features $p \in P, q \in Q$ of two images P, Q and determines relative similarity transformation from p to q

$$F(c) = F(q)F(p)^{-1} = \begin{pmatrix} s(c)R(c) & \mathbf{t}(c) \\ \mathbf{0}^\top & 1 \end{pmatrix}$$

with translation $\mathbf{t}(c) = \mathbf{t}(q) - s(c)R(c)\mathbf{t}(p)$, relative scale $s(c) = s(q)/s(p)$ and rotation $R(c) = R(q)R(p)^{-1}$ or
 $\theta(c) = \theta(q) - \theta(p)$

Hough pyramid matching (HPM)

- the 4-dof relative transformation represented by 4d vector

$$f(c) = (\mathbf{t}(c), s(c), \theta(c))$$

- mode seeking: we are looking for regions where density is maximized in the transformation space
 - a correspondence c contributes to a similarity score according to weight $w(c)$, based e.g. on visual word
 - to enforce one-to-one mapping, two correspondences $c = (p, q)$, $c' = (p', q')$ are conflicting if $p = p'$ or $q = q'$

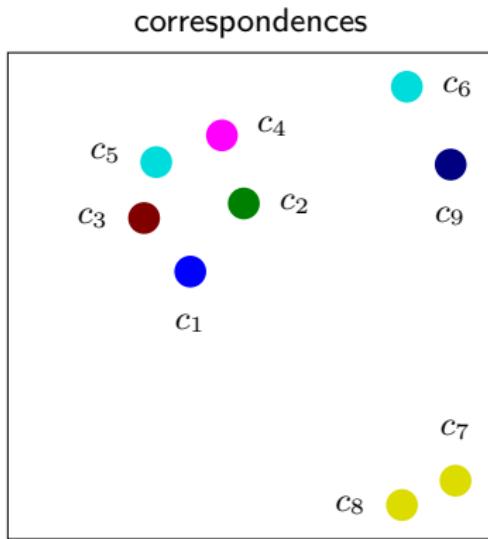
Hough pyramid matching (HPM)

- the 4-dof relative transformation represented by 4d vector

$$f(c) = (\mathbf{t}(c), s(c), \theta(c))$$

- mode seeking: we are looking for regions where density is maximized in the transformation space
- a correspondence c contributes to a similarity score according to weight $w(c)$, based e.g. on visual word
- to enforce one-to-one mapping, two correspondences $c = (p, q)$, $c' = (p', q')$ are conflicting if $p = p'$ or $q = q'$

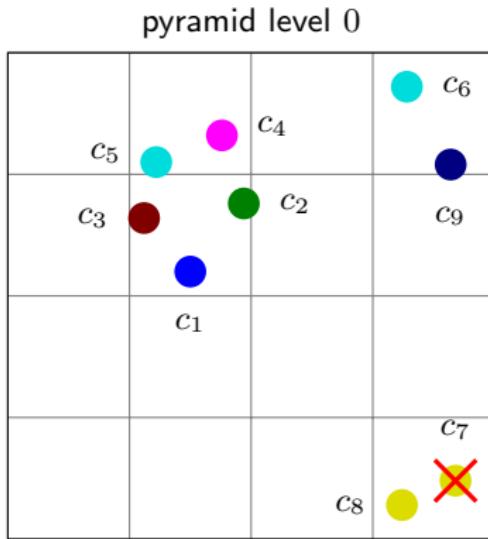
Hough pyramid matching (HPM)



	p	q	similarity score
c_1			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_1)$
c_2			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_2)$
c_3			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_3)$
c_4			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_4)$
c_5			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_5)$
c_6			0
c_7			0
c_8			$\frac{1}{4}6w(c_8)$
c_9			$\frac{1}{4}6w(c_9)$

- conflicting correspondences in the same bin b are erased
- in a bin b with $|h(b)|$ correspondences, each groups with $g(b) = [|h(b)| - 1]_+$ others
- level 0 weight 1

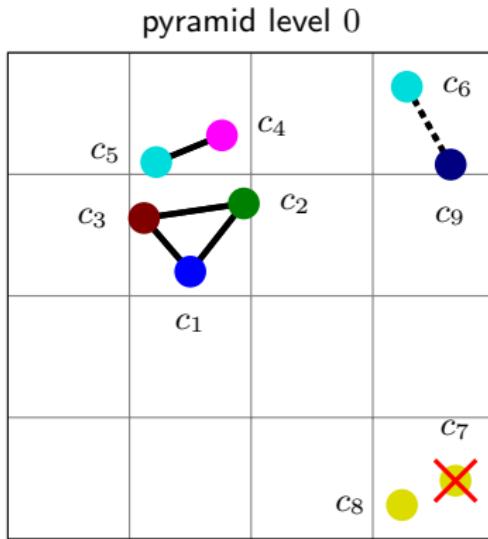
Hough pyramid matching (HPM)



	p	q	similarity score
c_1			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_1)$
c_2			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_2)$
c_3			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_3)$
c_4			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_4)$
c_5			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_5)$
c_6			0
c_7			0
c_8			$\frac{1}{4}6w(c_8)$
c_9			$\frac{1}{4}6w(c_9)$

- conflicting correspondences in the same bin b are erased
- in a bin b with $|h(b)|$ correspondences, each groups with $g(b) = [|h(b)| - 1]_+$ others
- level 0 weight 1

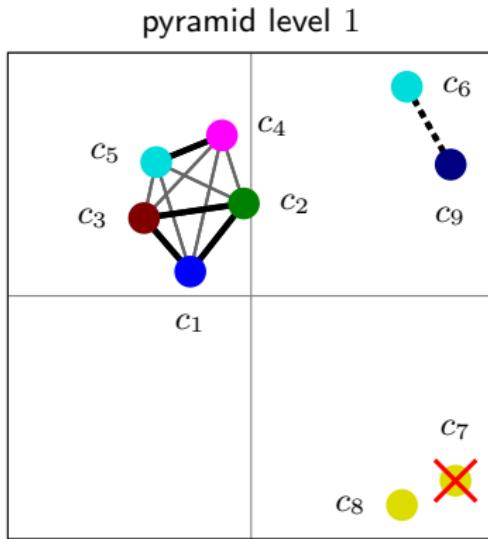
Hough pyramid matching (HPM)



	p	q	similarity score
c_1			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_1)$
c_2			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_2)$
c_3			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_3)$
c_4			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_4)$
c_5			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_5)$
c_6			0
c_7			0
c_8			$\frac{1}{4}6w(c_8)$
c_9			$\frac{1}{4}6w(c_9)$

- conflicting correspondences in the same bin b are erased
- in a bin b with $|h(b)|$ correspondences, each groups with $g(b) = [|h(b)| - 1]_+$ others
- level 0 weight 1

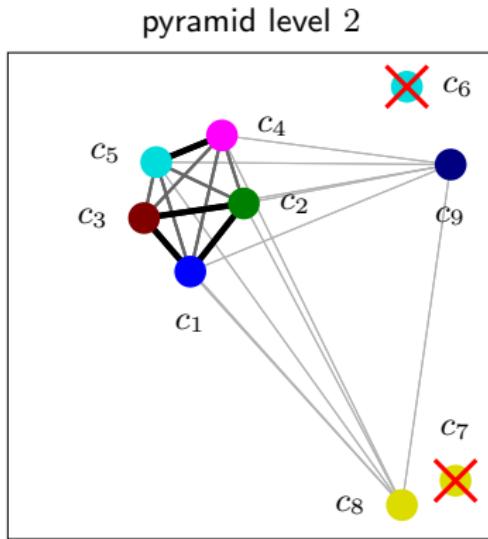
Hough pyramid matching (HPM)



	p	q	similarity score
c_1			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_1)$
c_2			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_2)$
c_3			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_3)$
c_4			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_4)$
c_5			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_5)$
c_6			0
c_7			0
c_8			$\frac{1}{4}6w(c_8)$
c_9			$\frac{1}{4}6w(c_9)$

- conflicting correspondences in the same bin b are erased
- in a bin b with $|h(b)|$ correspondences, each groups with $g(b) = [|h(b)| - 1]_+$ others
- level 1 weight $\frac{1}{2}$

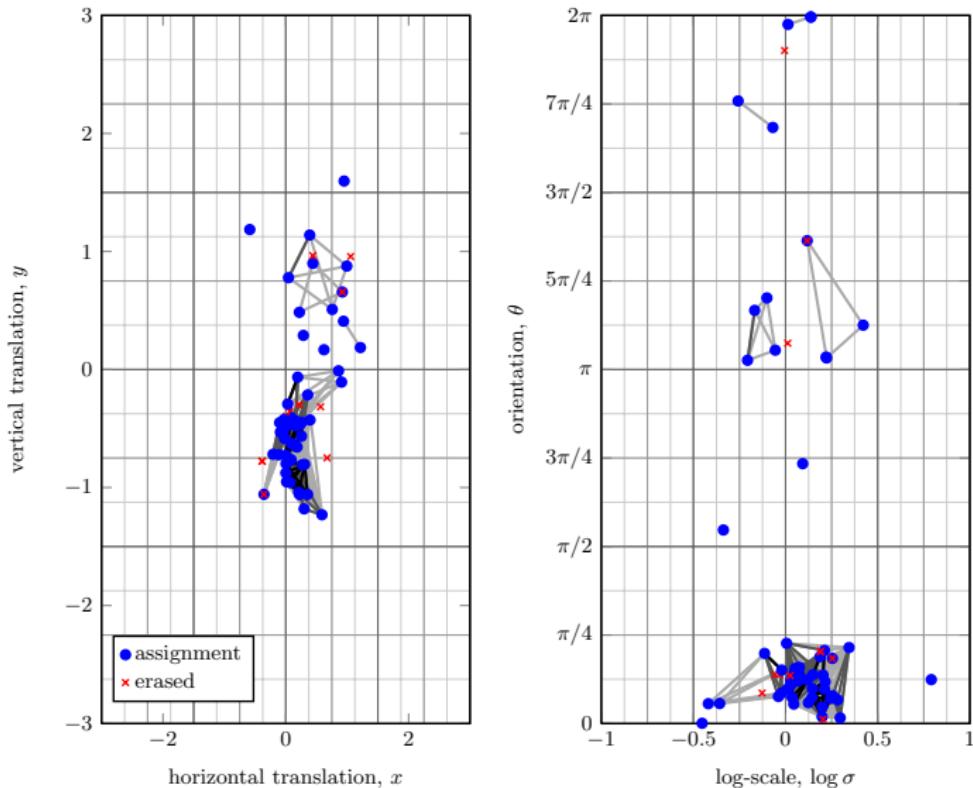
Hough pyramid matching (HPM)



	p	q	similarity score
c_1			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_1)$
c_2			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_2)$
c_3			$(2 + \frac{1}{2}2 + \frac{1}{4}2)w(c_3)$
c_4			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_4)$
c_5			$(1 + \frac{1}{2}3 + \frac{1}{4}2)w(c_5)$
c_6			0
c_7			0
c_8			$\frac{1}{4}6w(c_8)$
c_9			$\frac{1}{4}6w(c_9)$

- conflicting correspondences in the same bin b are erased
- in a bin b with $|h(b)|$ correspondences, each groups with $g(b) = [|h(b)| - 1]_+$ others
- level 2 weight $\frac{1}{4}$

Hough pyramid matching (HPM)



Hough pyramid matching (HPM)

- linear in the number of correspondences; no need to count inliers
- robust to deformations and multiple matching surfaces, invariant to transformations
- only applies to same instance matching

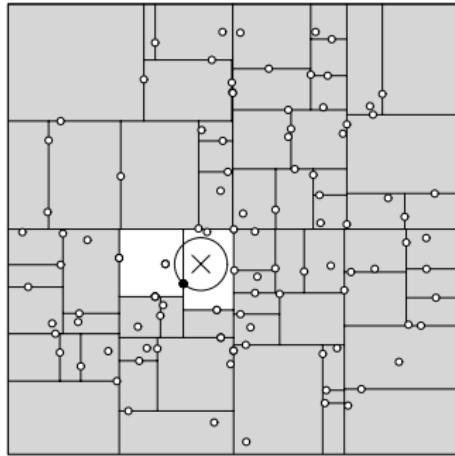
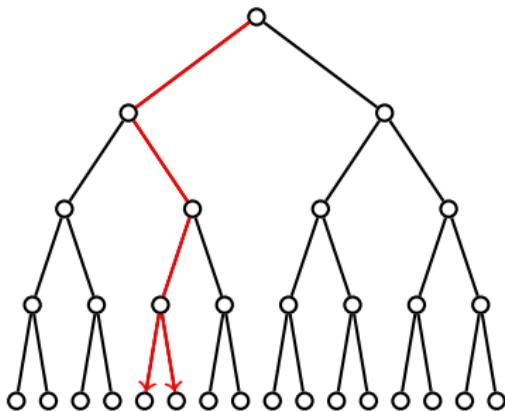
nearest neighbor search

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

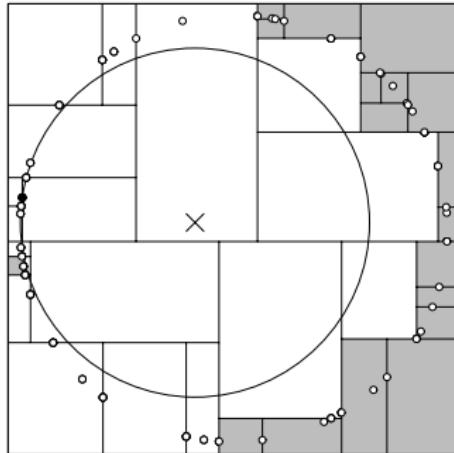
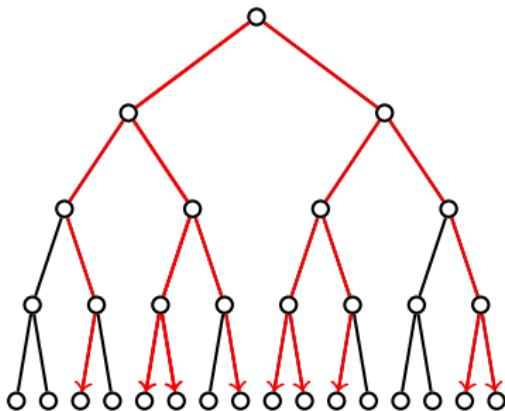
[Bentley 1975]



- **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

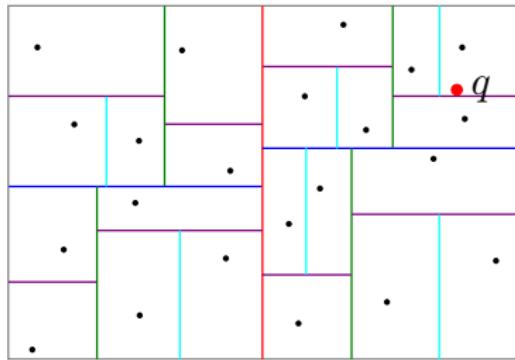
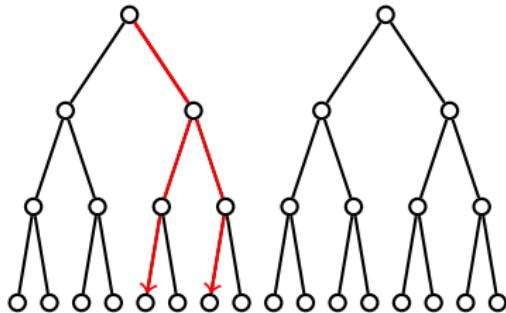
[Bentley 1975]



- **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

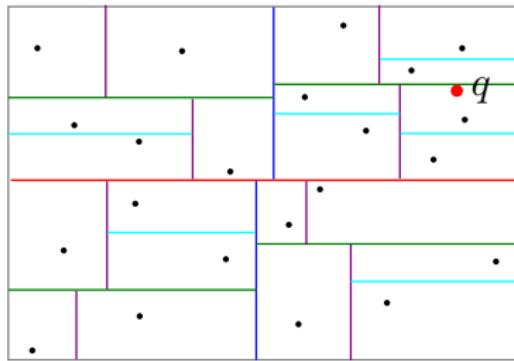
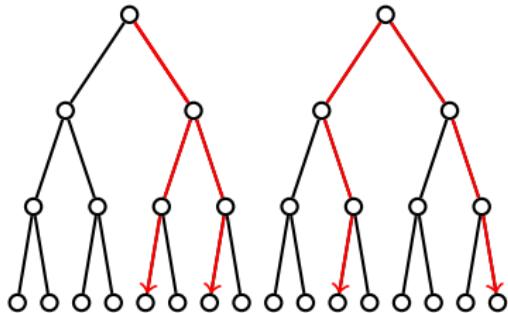
[Silpa-Anan and Hartley 1975]



- **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

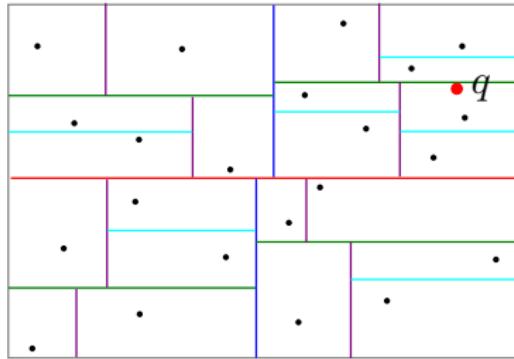
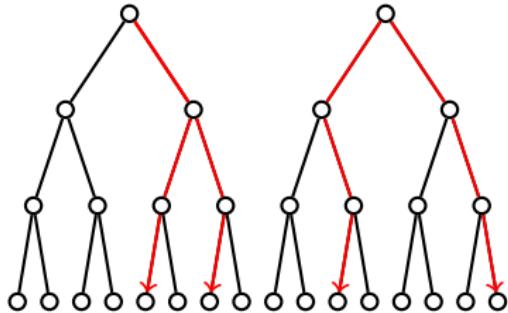
[Silpa-Anan and Hartley 1975]



- **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

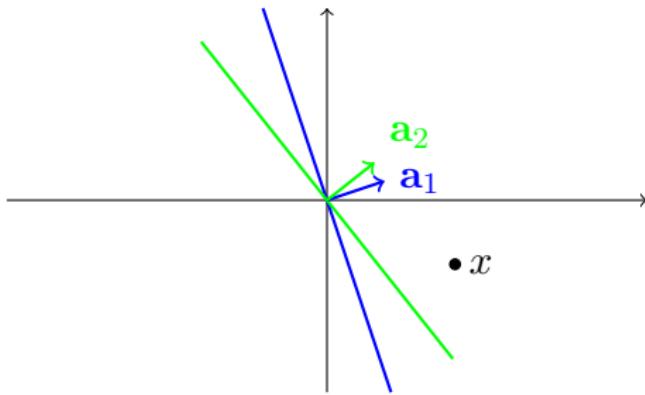
[Silpa-Anan and Hartley 1975]



- **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

[Charikar 2002]



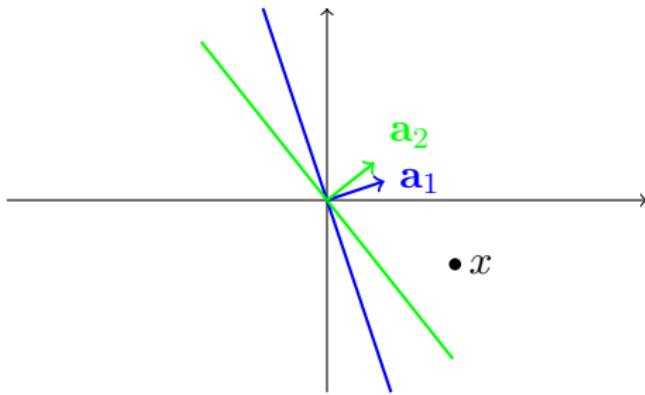
- **index:** choose $\mathbf{a}_i \sim \mathcal{N}(0, 1)$; encode each data point $x \in X$ by **binary code** $h(x) := (h_{\mathbf{a}_1}(x), \dots, h_{\mathbf{a}_k}(x)) \in \{-1, 1\}^d$ with **hash function**

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

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

locality sensitive hashing

[Charikar 2002]



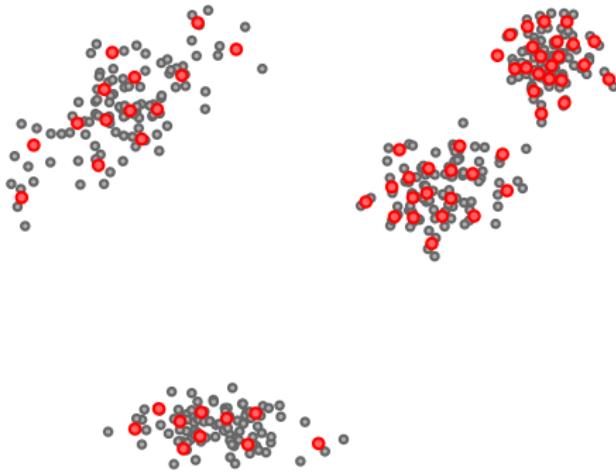
- **index:** choose $\mathbf{a}_i \sim \mathcal{N}(0, 1)$; encode each data point $x \in X$ by **binary code** $h(x) := (h_{\mathbf{a}_1}(x), \dots, h_{\mathbf{a}_k}(x)) \in \{-1, 1\}^d$ with **hash function**

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

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

vector quantization (VQ)

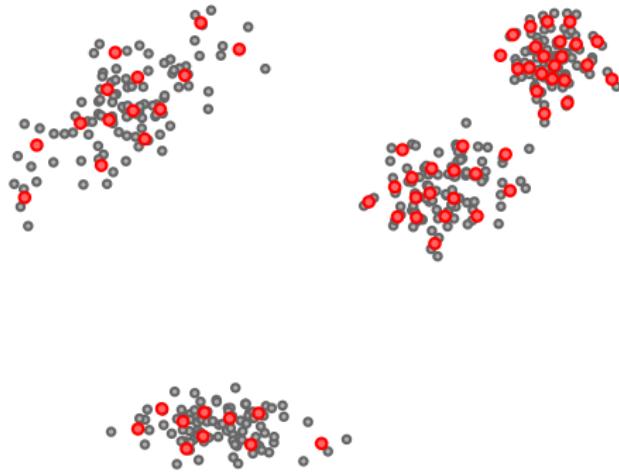
[Gray 1984]



- **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)

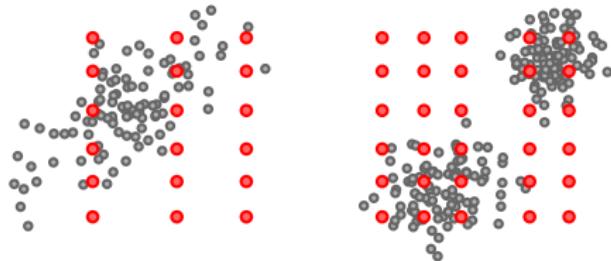
[Gray 1984]



- **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)

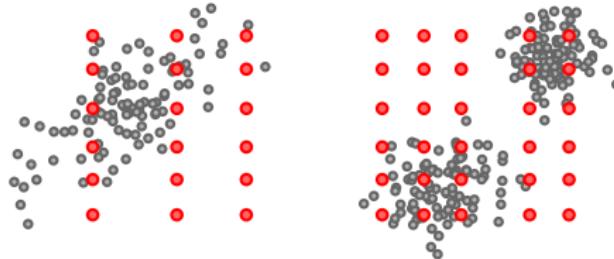
[Jégou et al. 2011]



- **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)

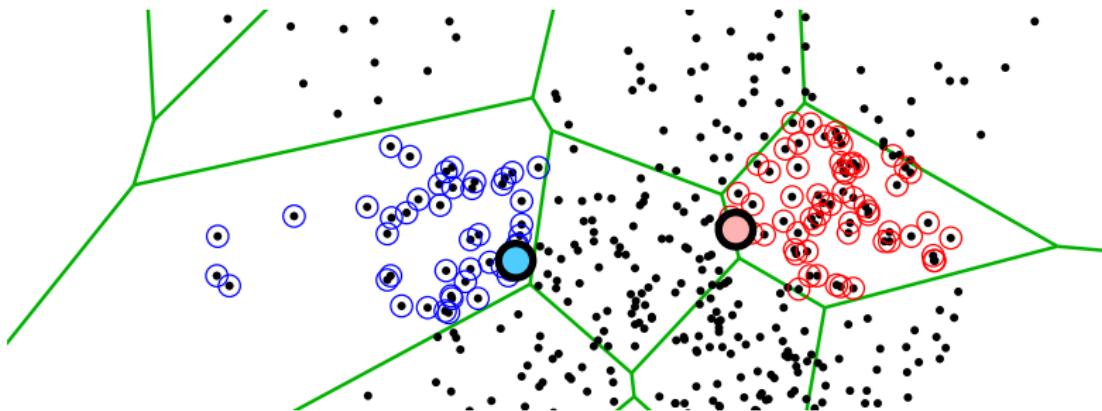
[Jégou et al. 2011]



- **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

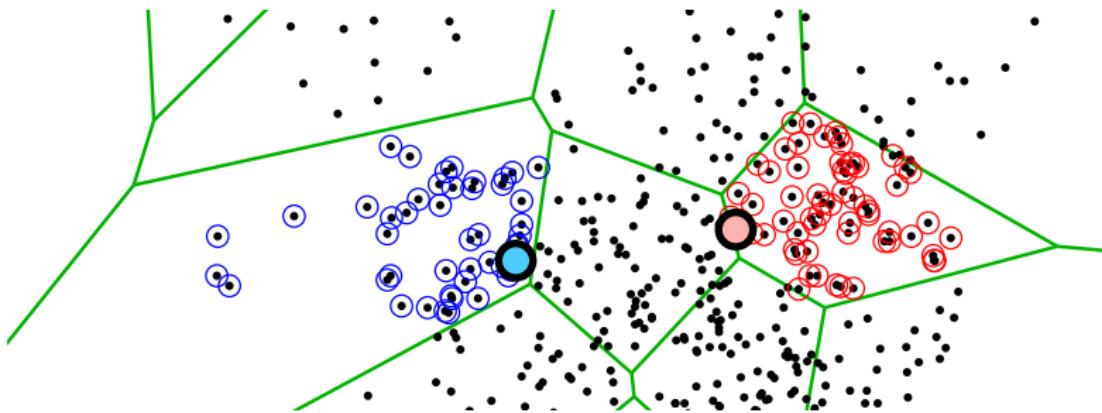
[Jégou et al. 2011]



- **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

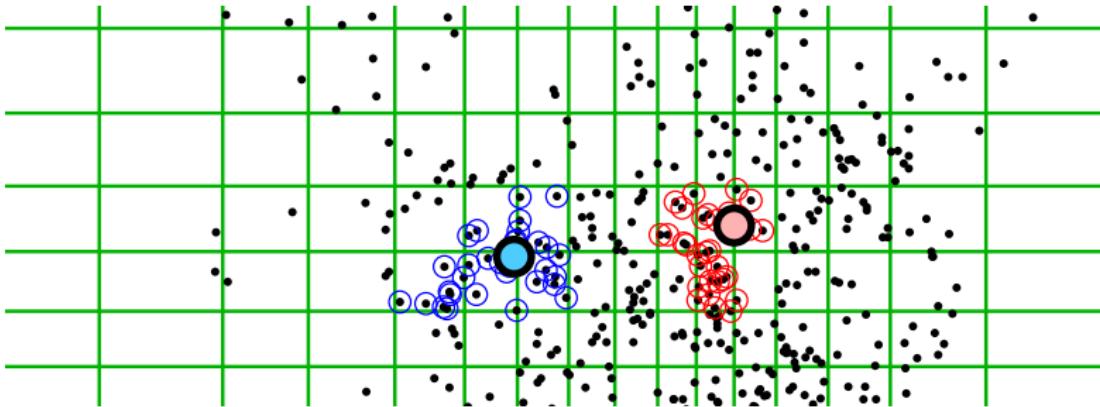
[Jégou et al. 2011]



- **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

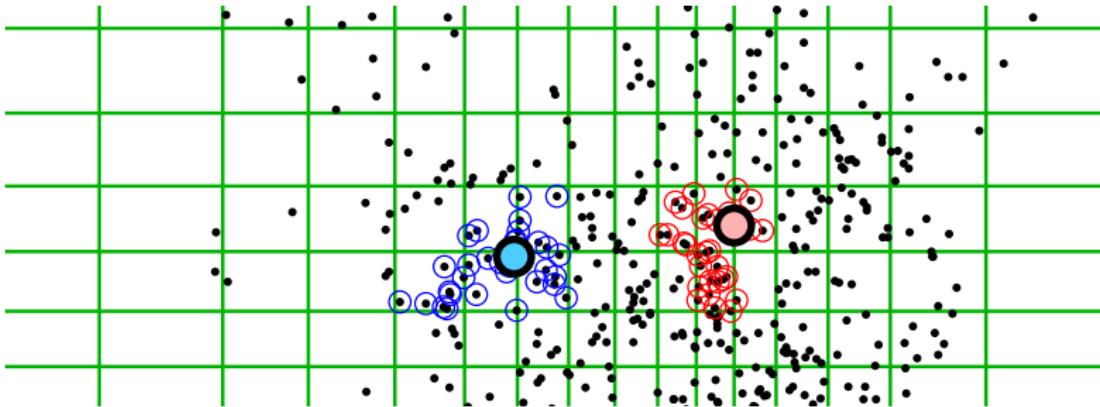
[Babenko and Lempitsky 2012]



- **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

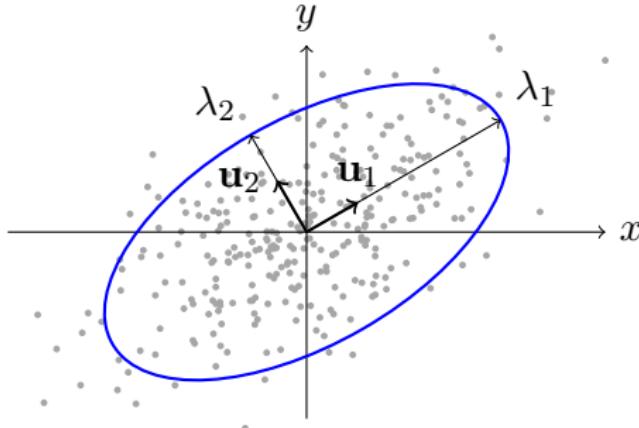
inverted multi-index

[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)



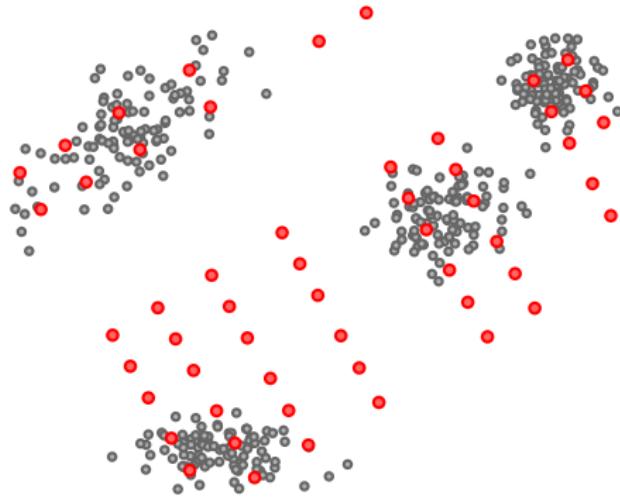
- given data $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, compute empirical mean $\bar{\mathbf{x}} := \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$ and covariance matrix

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

- 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)

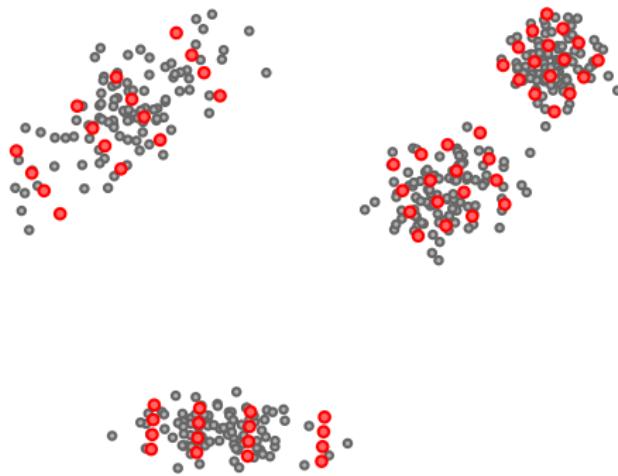
[Ge et al. 2013]



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

locally optimized product quantization (LOPQ)

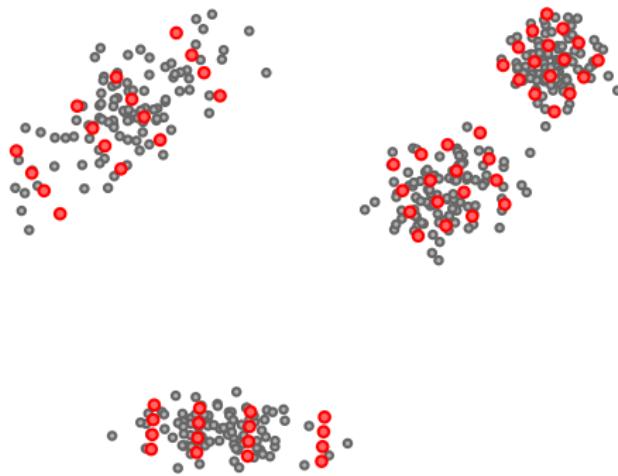
[Kalantidis and Avrithis 2014]



- 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)

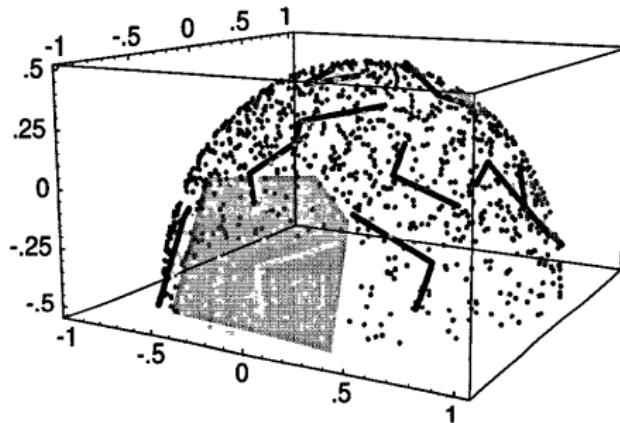
[Kalantidis and Avrithis 2014]



- 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

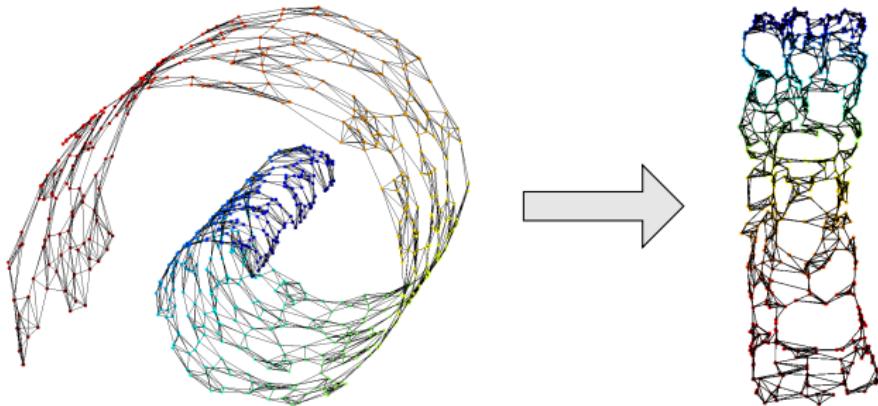
local principal component analysis

[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 an explicit mapping from the input to the embedding space

summary

- bag of words: treating geometry separately from appearance, and quantizing descriptors
- bag of words for instance and class recognition: what is common, what is different
- k -means, HKM, vocabulary tree, AKM, soft/multiple assignment, max pooling, burstiness
- beyond bag of words: pairwise matching between sets of features/descriptors that cannot be expressed as dot product
- HE, VLAD, ASMK, EMK
- design or learn embeddings: PMK, SPM, HPM
- a sum of similarities is better than a sum of distances
- nearest neighbor search: inverted index, multi-index, trees, forests, hashing, compression
- PCA and beyond: we should learn the manifold

discussion

representation

- convolution is linear + translation invariant (or equivariant) and is the **only** function having these properties
- Gabor filters or histograms of gradient orientations are more or less the same thing and are just the **first layer** of extracting a representation
- they record responses at every possible position, **scale and orientation**, resulting in a 4-dimensional representation; rotation and change of scale in the image behave like translation in the representation space
- convolution means that for every pixel we are looking at some **spatial neighborhood** (in the image domain), but the image has only **one channel** (grayscale)
- histograms can be expressed as two stages of **encoding + pooling**; then we can generalize these operations for the next layers

representation

- convolution is linear + translation invariant (or equivariant) and is the **only** function having these properties
- Gabor filters or histograms of gradient orientations are more or less the same thing and are just the **first layer** of extracting a representation
- they record responses at every possible position, **scale and orientation**, resulting in a 4-dimensional representation; rotation and change of scale in the image behave like translation in the representation space
- convolution means that for every pixel we are looking at some **spatial neighborhood** (in the image domain), but the image has only **one channel** (grayscale)
- histograms can be expressed as two stages of **encoding + pooling**; then we can generalize these operations for the next layers

representation

- convolution is linear + translation invariant (or equivariant) and is the **only** function having these properties
- Gabor filters or histograms of gradient orientations are more or less the same thing and are just the **first layer** of extracting a representation
- they record responses at every possible position, **scale and orientation**, resulting in a 4-dimensional representation; rotation and change of scale in the image behave like translation in the representation space
- convolution means that for every pixel we are looking at some **spatial neighborhood** (in the image domain), but the image has only **one channel** (grayscale)
- histograms can be expressed as two stages of **encoding + pooling**; then we can generalize these operations for the next layers

representation

- convolution is linear + translation invariant (or equivariant) and is the **only** function having these properties
- Gabor filters or histograms of gradient orientations are more or less the same thing and are just the **first layer** of extracting a representation
- they record responses at every possible position, **scale and orientation**, resulting in a 4-dimensional representation; rotation and change of scale in the image behave like translation in the representation space
- convolution means that for every pixel we are looking at some **spatial neighborhood** (in the image domain), but the image has only **one channel** (grayscale)
- histograms can be expressed as two stages of **encoding + pooling**; then we can generalize these operations for the next layers

representation

- convolution is linear + translation invariant (or equivariant) and is the **only** function having these properties
- Gabor filters or histograms of gradient orientations are more or less the same thing and are just the **first layer** of extracting a representation
- they record responses at every possible position, **scale and orientation**, resulting in a 4-dimensional representation; rotation and change of scale in the image behave like translation in the representation space
- convolution means that for every pixel we are looking at some **spatial neighborhood** (in the image domain), but the image has only **one channel** (grayscale)
- histograms can be expressed as two stages of **encoding + pooling**; then we can generalize these operations for the next layers

codebooks

- so, for the **second layer** we still have histograms of some kind but now they are over vectors (the filter responses of the first stage) rather than scalars (orientation and scale)
- to make a histogram we need a finite set of such vectors, and this we obtain through **vector quantization** (or sampling) of the layer one responses of a given dataset
- so, the concept that such representations are “**hand-crafted**” is incorrect; codebooks are learned from data in an unsupervised fashion
- codebook size, parameters in the encoding and pooling stages *etc.* are just **hyperparameters** that will we learn through **cross-validation**
- in contrast to layer one, there is **no spatial neighborhood** here (with the exception of HMAX) but there is **depth**, i.e. a number of channels corresponding to the dimensions of these vectors; we will combine both, resulting in 3-dimensional filter kernels

codebooks

- so, for the **second layer** we still have histograms of some kind but now they are over vectors (the filter responses of the first stage) rather than scalars (orientation and scale)
- to make a histogram we need a finite set of such vectors, and this we obtain through **vector quantization** (or sampling) of the layer one responses of a given dataset
- so, the concept that such representations are “**hand-crafted**” is incorrect; codebooks are learned from data in an unsupervised fashion
- codebook size, parameters in the encoding and pooling stages etc. are just **hyperparameters** that will we learn through **cross-validation**
- in contrast to layer one, there is **no spatial neighborhood** here (with the exception of HMAX) but there is **depth**, i.e. a number of channels corresponding to the dimensions of these vectors; we will combine both, resulting in 3-dimensional filter kernels

codebooks

- so, for the **second layer** we still have histograms of some kind but now they are over vectors (the filter responses of the first stage) rather than scalars (orientation and scale)
- to make a histogram we need a finite set of such vectors, and this we obtain through **vector quantization** (or sampling) of the layer one responses of a given dataset
- so, the concept that such representations are “**hand-crafted**” is incorrect; codebooks are learned from data in an **unsupervised** fashion
- codebook size, parameters in the encoding and pooling stages etc. are just **hyperparameters** that will we learn through **cross-validation**
- in contrast to layer one, there is **no spatial neighborhood** here (with the exception of HMAX) but there is **depth**, i.e. a number of channels corresponding to the dimensions of these vectors; we will combine both, resulting in 3-dimensional filter kernels

codebooks

- so, for the **second layer** we still have histograms of some kind but now they are over vectors (the filter responses of the first stage) rather than scalars (orientation and scale)
- to make a histogram we need a finite set of such vectors, and this we obtain through **vector quantization** (or sampling) of the layer one responses of a given dataset
- so, the concept that such representations are “**hand-crafted**” is incorrect; codebooks are learned from data in an unsupervised fashion
- codebook size, parameters in the encoding and pooling stages etc. are just **hyperparameters** that will we learn through **cross-validation**
- in contrast to layer one, there is **no spatial neighborhood** here (with the exception of HMAX) but there is **depth**, i.e. a number of channels corresponding to the dimensions of these vectors; we will combine both, resulting in 3-dimensional filter kernels

codebooks

- so, for the **second layer** we still have histograms of some kind but now they are over vectors (the filter responses of the first stage) rather than scalars (orientation and scale)
- to make a histogram we need a finite set of such vectors, and this we obtain through **vector quantization** (or sampling) of the layer one responses of a given dataset
- so, the concept that such representations are “**hand-crafted**” is incorrect; codebooks are learned from data in an unsupervised fashion
- codebook size, parameters in the encoding and pooling stages etc. are just **hyperparameters** that will we learn through **cross-validation**
- in contrast to layer one, there is **no spatial neighborhood** here (with the exception of HMAX) but there is **depth**, i.e. a number of channels corresponding to the dimensions of these vectors; we will combine both, resulting in 3-dimensional filter kernels

local features

- depending on the task (e.g. stereopsis, motion estimation, instance recognition compared to class recognition), not all spatial regions are equally important
- classification works best with dense features, but still, through encoding, the responses to most “visual words” are zero; so there some **sparsity** in the representation, at least before pooling
- in order to make change of scale really behave like translation in the representation space, we also need **scale normalization** and a **logarithm**
- operators that detect local features can be expressed as convolution followed by some kind of competition, but they can require **more than one layers** with **nonlinearities** in between; we will follow this idea for more complex patterns
- when it comes a sparse set of local features, matching becomes easier to formulate compared to e.g. continuous distributions

local features

- depending on the task (e.g. stereopsis, motion estimation, instance recognition compared to class recognition), not all spatial regions are equally important
- classification works best with dense features, but still, through encoding, the responses to most “visual words” are zero; so there some **sparsity** in the representation, at least before pooling
- in order to make change of scale really behave like translation in the representation space, we also need **scale normalization** and a **logarithm**
- operators that detect local features can be expressed as convolution followed by some kind of competition, but they can require **more than one layers** with **nonlinearities** in between; we will follow this idea for more complex patterns
- when it comes a sparse set of local features, matching becomes easier to formulate compared to e.g. continuous distributions

local features

- depending on the task (e.g. stereopsis, motion estimation, instance recognition compared to class recognition), not all spatial regions are equally important
- classification works best with dense features, but still, through encoding, the responses to most “visual words” are zero; so there some **sparsity** in the representation, at least before pooling
- in order to make change of scale really behave like translation in the representation space, we also need **scale normalization** and a **logarithm**
- operators that detect local features can be expressed as convolution followed by some kind of competition, but they can require **more than one layers** with **nonlinearities** in between; we will follow this idea for more complex patterns
- when it comes a sparse set of local features, matching becomes easier to formulate compared to e.g. continuous distributions

local features

- depending on the task (e.g. stereopsis, motion estimation, instance recognition compared to class recognition), not all spatial regions are equally important
- classification works best with dense features, but still, through encoding, the responses to most “visual words” are zero; so there some **sparsity** in the representation, at least before pooling
- in order to make change of scale really behave like translation in the representation space, we also need **scale normalization** and a **logarithm**
- operators that detect local features can be expressed as convolution followed by some kind of competition, but they can require **more than one layers** with **nonlinearities** in between; we will follow this idea for more complex patterns
- when it comes a sparse set of local features, matching becomes easier to formulate compared to e.g. continuous distributions

local features

- depending on the task (e.g. stereopsis, motion estimation, instance recognition compared to class recognition), not all spatial regions are equally important
- classification works best with dense features, but still, through encoding, the responses to most “visual words” are zero; so there some **sparsity** in the representation, at least before pooling
- in order to make change of scale really behave like translation in the representation space, we also need **scale normalization** and a **logarithm**
- operators that detect local features can be expressed as convolution followed by some kind of competition, but they can require **more than one layers** with **nonlinearities** in between; we will follow this idea for more complex patterns
- when it comes a sparse set of local features, matching becomes easier to formulate compared to e.g. continuous distributions

matching

- descriptors are really meant to be used for matching one image to another (e.g. for instance recognition) or one image to a pattern (for classification)
- we want to learn a descriptor such that dot product will be good enough for matching
- we can start by thinking about pairwise matching between two sets of descriptors and come up with (design or learn) a representation, maybe at a higher dimension, such that dot product will be approximating this pairwise matching process
- there should be some invariance to geometric transformations; whether this should be designed or learned is up to discussion

matching

- descriptors are really meant to be used for matching one image to another (e.g. for instance recognition) or one image to a pattern (for classification)
- we want to learn a descriptor such that **dot product** will be good enough for matching
- we can start by thinking about **pairwise matching** between two **sets of descriptors** and come up with (design or learn) a representation, maybe at a higher dimension, such that dot product will be approximating this pairwise matching process
- there should be some **invariance** to geometric transformations; whether this should be designed or learned is up to discussion

matching

- descriptors are really meant to be used for matching one image to another (e.g. for instance recognition) or one image to a pattern (for classification)
- we want to learn a descriptor such that **dot product** will be good enough for matching
- we can start by thinking about **pairwise matching** between two **sets of descriptors** and come up with (design or learn) a representation, maybe at a higher dimension, such that dot product will be approximating this pairwise matching process
- there should be some **invariance** to geometric transformations; whether this should be designed or learned is up to discussion

matching

- descriptors are really meant to be used for matching one image to another (e.g. for instance recognition) or one image to a pattern (for classification)
- we want to learn a descriptor such that **dot product** will be good enough for matching
- we can start by thinking about **pairwise matching** between two **sets of descriptors** and come up with (design or learn) a representation, maybe at a higher dimension, such that dot product will be approximating this pairwise matching process
- there should be some **invariance** to geometric transformations; whether this should be designed or learned is up to discussion