Large / larger-scale image search

Introduction

Hervé Jégou, INRIA

Special Acknowledgments to
Florent Perronnin, Matthijs Douze, Cordelia Schmid, Patrick Pérez, Ondrej Chum
General outline

PART I: Introduction
   Applications and datasets
       Image description and matching

PART II: Large-scale image search
   The bag-of-word representation and some extensions

PART III: Larger-scale image search
   Novel aggregation mechanisms
   Efficient indexing

Conclusions
Image search

Scenario: Query-by-example:

On a large (largest) scale:
- short response time
- Millions to billions of images
Visual Search

- Inria’s BigImBaz (2008-)
  
  http://bigimbaz.inrialpes.fr/
  
  10 millions images on a big server
Visual Search

- TinEye.com

194 Results

Searched over 1.9305 billion images in 2.106 seconds.
for file: eiffel.png

blog.beneth.fr
  eiffel_tower_m.jpg
  http://blog.beneth.fr/

blog.pixnet.net
  1203141301.jpg
  http://blog.pixnet.net/66a1/66a1/14354493

amphetamine.deviantart.com
  La_Tour_Eiffel_by_amphetamine.jpg
  http://amphetamine.deviantart.com/la-tour-eiffel-

Lizacontagious.deviantart.com
  la_tour_eiffel__majestueuse__by_lizac...___.jpg
  http://lizacontagious.deviantart.com/art/La-Tour-

renmiked.wordpress.com
  paris-2004-068.jpg
  http://renmiked.wordpress.com/2010/04/11/annee-
Visual Search

- Google’s goggles on Android
Scalability for the image search problem

Scalable systems with Global descriptors (image-level)

- QBIC’95: 7.5K (but in 1995!)
- Cortina: Quack et al. – ACMM’04
  3 million images (10M)
- Torralba et al. – CVPR’08
  12.9M – 74ms with 30bit codes
- Douze et al. ’2009 – CIVR’09
  110 million images – 180ms

Scalable systems local descriptors (object instance)

- Sivic et al. – CVPR’03: “Video-Google”
  5k images
- Joly et al. – CIVR’03
  6M video keyframes – 120M descriptors
- Nister et al. – CVPR’06
  50k images (then 1M images)
- J. et al. – CVPR’10
  10M images (then 100M)
Datasets – Oxford5k/Paris6k

Oxford5k dataset: find images of the same famous building
55 queries (11*5 buildings), varying number of relevant results (6-221)
Oxford105K = Oxford5k + a image set of 100k “distractors” for large scale tests

Philbin, Chum, Isard, Sivic and Zisserman,
« Object retrieval with large vocabularies and fast spatial matching », ICCV’07
Datasets– Holidays

INRIA Holidays dataset: 1491 shots of personal Holiday snapshot
500 queries, each associated with a small number of results 1-11 results
1 million distracting images (with some “false false” positives)

Hervé Jégou, Matthijs Douze and Cordelia Schmid
Hamming Embedding and Weak Geometric consistency for large-scale image search, ECCV’08
Univ. Kentucky object recognition benchmark

Nister & Stewenius 2006
2550 objects, represented each by 4 images
10200 images in total

Images shot for the purpose of the benchmark

Each query is submitted in turn

Typical performance measure: average number of images returned in first 4 positions
Datasets – Stanford Mobile

Stanford Mobile Visual dataset:
1200 reference images
3000 queries: images shot by mobile devices (queries) – of lower quality
Tutorial: large scale image search
Image description & matching

Hervé Jégou, INRIA
General Outline

- PART I: Introduction
  - Applications and datasets
  - Image description and matching

- PART II: Large-scale image search
  - The bag-of-word representation and some extension

- PART III: Larger-scale image search
  - Novel aggregation mechanisms
  - Efficient indexing

- Conclusions
**Image description**

- Image processing: analysis step (=description)
  - Convert an image to a mathematical representation
  - Similar images have the “similar” representations, but not dissimilar ones

- Difficulty: Finding the object despite possibly large changes in scale, viewpoint, lighting and partial occlusion
  \[ \Rightarrow \text{needs invariant description} \]
Image description

- Image processing: analysis step (=description)
  - Convert an image to a mathematical representation
  - Similar images have the “similar” representations, but not dissimilar ones

- But the representation should be discriminative enough
  ⇒ careful selection of what should be invariant for the application

http://labs.ideeinc.com/multicolour/
Global descriptors

- Highly scalable: 1 vector matched with a set of N database vectors

- Color Histogram, e.g.: [Swain 91]
  - High invariance to many transformation
  - But limited discriminative power

- The “gist” of a scene [Oliva 01]
  - Several frequency bands and orientations for each image location
  - Tiling of the image, for example 4x4, and at different resolution
Matching local descriptors [Lowe04]

- Image content is transformed into local features that are invariant to geometric and photometric transformations
Local description: image detector

- The detector provides the desired invariance to transformations

- Popular detectors:
  - MSER: Wide-baseline matching [Matas 02]
  - Difference of Gaussian [Lowe 99]
  - Hessian-Affine [Mikolajczyk 01]

- Renewed interest for dense descriptors
  - [Leung 99, Fei-Fei 05, Lazebnik 06]
  - Mainly for Image classification
  - But also for image/scene/object retrieval
    E.g., [Gordo 12] at CVPR’12

[Lowe, IJCV 2004]
Local image descriptor

- Description of patch
  - After orientation/scale/photometric normalization

- SIFT [Lowe 99]
  - 8 orientations of the gradient
  - 4x4 spatial grid ⇒ 128 dimensions
  - Normalized to L2-norm one, compared with Euclidean distance
  - Component-wise “Power-law” [Jain’12, Arandjelovic 12]

- Most descriptors derive from SIFT:
  - More efficient: SURF [Bay 08]
  - More compact: many, e.g., DAISY
  - With color: [Burghouts 09]

- Learned descriptors [Winder’07, Brown’10]
  - Used training sets of 1) matching and 2) non-matching patches
Geometric matching with local descriptors

- Use a global geometrical constraint to filter out the outliers

Interest points extracted with Harris detector (~ 500 points)

Match points using descriptors

99 inliers \[\Rightarrow\] score \[\Rightarrow\] 89 outliers

- Precise matching, but not scalable (100—1000 images)
Large-scale image search
Bag-of-words and extensions

Hervé Jégou, INRIA
General outline

- PART I: Introduction
  - Applications and datasets
  - Image description and matching

- PART II: Large-scale image search
  - The bag-of-word representation and some extensions

- PART III: Larger-scale image search
  - Novel aggregation mechanisms
  - Efficient indexing

- Conclusion
Direct matching: the complexity issue

- Assume an image described by $m=1000$ descriptors (dimension $d=128$)
  - $N \cdot m = 1 \text{ billion descriptors to index}$

- Database representation in RAM: 128 GB with 1 byte per dimension

- Search: $m^2 \cdot N \cdot d$ elementary operations
  - i.e., $> 10^{14}$ \Rightarrow computation not tractable
  - The quadratic term $m^2$: severely impacts the efficiency
Bag-of-visual-words

- The BOV representation
  - First introduced for texture classification [Malik’99]

- “Video-Google paper” – Sivic and Zisserman, ICCV’2003
  - Mimick a text retrieval system for image/video retrieval
  - High retrieval efficiency and excellent recognition performance

- “Visual categorization with bag of keypoints” – Dance’04
  - Show its interest when used jointly with a (kernelized) SVM

- Key idea: n local descriptor describing the image → 1 vector
  - sparse vectors ⇒ efficient comparison
  - inherits invariance of the local descriptors
Bag-of-visual words

- The goal: “put the images into words”, namely visual words
  - Input local descriptors are continuous
  - Need to define what a “visual word is”
  - Done by a quantizer $q$
    \[
    q: \mathbb{R}^d \rightarrow \omega
    \]
    \[
    x \rightarrow c(x) \in \omega
    \]
  - $q$ is typically a k-means

- $\omega$ is called a “visual dictionary”, of size $k$
  - A local descriptor is assigned to its nearest neighbor
    \[
    q(x) = \arg \min_w \|x-w\|^2
    \]
    \[
    w \in \omega
    \]
  - Quantization is lossy: we can not get back to the original descriptor
  - But much more compact: typically 2-4 bytes/descriptor
Video Google [Sivic & Zisserman’03]

- Extract local descriptors
  - Detector
  - Describe the patch

- Quantize all descriptors
  - Subsequently compute the vector of frequencies
  - Weight by IDF (rare = more important)

$\Rightarrow$ TF-IDF vectors

- Search similar vectors

- Optionally: Re-ranking

Inverted file: sparse vectors

find most similar vectors

results
Inverted file

- Set of lists
  - That stores the sparse vector components
  - Use to compute the cosine similarity (or any Lp-norm, see [Nister 06])

- Two implementations

  store one image id per descriptor

  ![Diagram 1]

  Can easily incorporate meta information per descriptor (geometry, bundled features, etc)

  ![Diagram 2]

  Store image id+nb of descriptors

  ![Diagram 3]

  Easily implemented with Matlab using sparse matrices/vectors

- Complexity: approximated by the number of visited items
Interest of the voting interpretation

- And the corresponding implementation of the inverted file

- Easy extended to incorporate
  - A better matching method [J’08]
  - Partial Geometrical information [J’08, Zhao 10, …]
  - Neighborhood information [Wu 09]
  - … any method that requires to handle individual descriptors
Inverted file – Complexity

- Denote
  - $p_i = P(\text{assign a descriptor to word } i)$
  - $N = \text{number of image in database}$
  - $m = \text{average # of descriptors / image}$

  $\Rightarrow$ The expected length of List $i$ is given by: $N \cdot m \cdot p_i$

- The expected cost is: 
  \[ N \cdot m^2 \sum_{i=1}^{k} p_i^2 \]

- Clusters of variable sizes negatively impacts this cost [Nister 06]
  - Imbalance factor: 
    \[ k \sum p_i^2 \]
  - measures the divergence from (optimal) uniform distribution (=1)

- Strategies proposed to balance the clusters [Tavenard 11]
  $\Rightarrow$ but these impact the search quality
Inverted file – Complexity

- Complexity is **linear** in the number of images
  - but small constant, in order of $\frac{m}{k}$
  - E.g., $C=0.01$

- **Memory usage** of an inverted file
  - 1 million images $\approx 8\text{ GB}$ (depending on $m$)
  - Can be compressed [J’09], “Packing bag-of-features”
    - As previously proposed for text search engines [Zobel’06, Zhang’08]
Inverted file – Boosting efficiency

- **Stop-words**
  - Method used in Text retrieval to discard uninformative words
  - In image search: remove the s most frequent ones [Sivic 03]
  - Impact on efficiency: assuming \( p_i \) in decreasing order
    
    \[
    Nm^2 \sum_{i=1}^{k} p_i^2 \quad \text{by} \quad Nm^2 \sum_{i=s+1}^{k} p_i^2
    \]

  - But most frequent **visual** words are not that uninformative
Inverted file – Boosting efficiency

- Large vocabularies
  - Unlike in text, we decide the vocabulary size by choosing $k$
    - for search quality and/or efficiency
  - Querying complexity: linear in $1/k$
  - Efficiency boosted by using a very large dictionary [Nister 06]
Large vocabularies: assignment cost

- Large vocabularies are preferred [Nister 06]: high retrieval efficiency
  - But increased assignment cost, e.g., for k-means: $C(k) = C_1 \times k + \frac{C_2}{k}$

- Structured quantizers: low quantization cost even for huge vocabularies
  - Grid lattice quantizer [Tuytelaars 07]
  - But poor performance in retrieval [Philbin 08]
  - And very unbalanced [Pauleve 10]:
Large vocabularies with learned quantizer

- Hierarchical k-means [Nister 06]
  - K-means tree of height $h$
    - Branching factor $b$: $k = b^h$
    - Assignment Complexity:
      $$\mathcal{O}(d hb) = \mathcal{O}(d h k^{\frac{1}{h}})$$

- Approximate k-means [Philbin 07]
  - Based on approximate nearest neighbor search
  - With parallel tree structures
  - See later in this tutorial
Bag-of-words: another interpretation

- « Visual words » are a view of mind
- BOV \( \approx \) approximate k-NN search+voting
  - Implicitly define the neighborhood \( N(x) \) of a vector \( x \) as
    \[
    N(x) = \{ y_i \in Y : c(y_i) = c(q) \}
    \]

- But, let assume:
  - 2 descriptors in query
  - 3 descriptors on database side
  \( \Rightarrow \) 6 votes for 2x3 descriptors
    = contribution to the cosine similarity
- Partial solution: pre-process BOV with component-wise square rooting
  \( \Rightarrow \) Linear contribution w.r.t the number of matches
Compromise on vocabulary size: $k=20000$
Compromise on vocabulary size: $k=200000$
Impact of the vocabulary size on accuracy

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

  \[ k=1,000 \quad k=200,000 \]

- No good trade-off between “small” and “large”!
  - Intrinsic matching method of BOV is relatively poor in all cases

- Partially solved by multiple [J’07] or soft assignment [Philbin 08]
  - Preferably on query side only [J’09, Arandjelovic’12] to save memory
Compromise on vocabulary size: $k=20000$
But with a better matching method (HE)...
Compromise on vocabulary size: k=200000
Geometrical verification

- Re-ranking based on full geometric verification [Philbin 07]
  - works very well but very costly
  - Applied to a short-list only (typically, 100 images)
  - for very large datasets, the number of distracting images is so high that relevant images are not even short-listed!
BOV search in 1M images – ranks

Query

BOV 2

BOV 5890

BOV 43064
Geometrical verification on a large scale

- Important activity on the topic
  - Weak geometry consistency [Jegou 08]
  - Geometrical Min-hash [Chum 09]
  - Bundling features [Wu 09]
  - Spatial inverted file [Lin 10]
  - ... 

- In classification
  - Most of these methods does not correspond to a vector model
  - not useable for classification with SVM
  - Geometry in classification: spatial pyramid matching [Lazebnik 06]
Geometrical verification on a large scale

- Important activity on the topic
  - Weak geometry consistency [Jegou 08]
  - Geometrical Min-hash [Chum 09]
  - Bundling features [Wu 09]
  - Spatial inverted file [Lin 10]
  - ...

- In classification
  - Most of these methods does not correspond to a vector model
  - not useable for classification with SVM
  - Geometry in classification: spatial pyramid matching [Lazebnik 06]
Weak Geometry consistency

- WGC is a Hough transform
  - But do estimate a full geometrical transformation
  - Separately estimate scalar quantities: rotation angle and log-scale
  - Just used to filter out the outliers

- Implementation
  - Store quantized dominant orientation and detector log-scale directly in the inverted file
  - Two small hough histograms to collect the votes (16–32 bins/image)

- Variation: Enhanced Weak Geometry consistency [Zhao 10]
  - a.k.a visual phrases [Zhang 11]
  - Deal with the translation (instead of angle/scale)
Weak geometric consistency

Max = rotation angle between images
Large scale: BOV search in 1M images

Query

BOV  2
HE+WGC  1

BOV  5890
HE+WGC  4

BOV  43064
HE+WGC  5
Query expansion in visual search

- [Chum 07], “Total Recall”, ICCV 07
  - Process the list of results
  - If some images are good (verified by spatial verification), use them
  - To process some other augmented queries

- Discriminative query expansion [Arandjelovic 12]
  - Learn a classifier on-the-fly
**Bag-of-words: concluding comments**

- Practical solution: same ingredients as in text can be used
  - vector model especially interesting in classification
    → useable with strong classifiers, in particular SVM
  - query expansion [Chum’07]
  - Or handle statistical phenomenons, e.g.,
    - Burstiness [Jegou’09]
    - Co-occurences [Chum’10]

- With appropriate extension, state-of-the-art:
  - Hamming Embedding
  - Re-ranking with spatial verification
  - Query-expansion
Questions ?
Towards larger scale

- BOV Limited to about a few million images on a server (memory!)

- To scale more, one may use
  - global descriptors
  - With a subsequent coding technique

```
<table>
<thead>
<tr>
<th>D</th>
<th>D'</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.234]</td>
<td>[0.155]</td>
</tr>
<tr>
<td>0.452</td>
<td>0.230</td>
</tr>
<tr>
<td>0.134</td>
<td></td>
</tr>
<tr>
<td>0.001</td>
<td>0.435</td>
</tr>
</tbody>
</table>
```

- “Small codes and large databases for recognition [Torralba’08]
  - Very compact binary codes (32-256 bits)
  - Yet limited invariance
Towards larger scale

- BOV Limited to about **a few million images** on a server (memory!)

- **To scale more**, need to jointly optimize: quality, speed, memory

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

![Diagram](image_url)
Larger-scale visual recognition
Novel aggregation mechanisms

Hervé Jégou, INRIA
Towards larger scale

BOV Limited to about a **few million images** on a server (memory!)

To scale more, one may use
- Global descriptors
- With a subsequent coding technique

“Small codes and large databases for recognition [Torralba’08]
- Very compact binary codes (32-256 bits)
- Yet limited invariance
Towards larger scale

With a representation based on local descriptors

**To scale more**, need to jointly optimize: quality, speed, memory

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

![Diagram showing the process of SIFT aggregation, dimension reduction, and vector encoding/indexing.](image)
Outline

PART I: Introduction
   Applications and datasets
   Image description and matching

PART II: Large-scale image search
   The bag-of-word representation and some extension

PART III: Larger-scale image search
   Novel aggregation mechanisms
   Efficient indexing

Conclusions
Motivation for new aggregation mechanisms

BOV is only about counting the number of local descriptors assigned to each Voronoi region

Why not including other statistics?

Motivation

BOV is only about **counting** the number of local descriptors assigned to each Voronoi region

Why not including **other statistics**? For instance:

- mean of local descriptors  \( \times \)

Motivation

BOV is only about **counting** the number of local descriptors assigned to each Voronoi region

Why not including **other statistics**? For instance:
- mean of local descriptors
- (co)variance of local descriptors

A first example: the VLAD

Given a codebook \( \{ \mu_i, i = 1 \ldots N \} \), e.g. learned with K-means, and a set of local descriptors \( X = \{ x_t, t = 1 \ldots T \} \):

- ① assign: \( \text{NN}(x_t) = \arg \min_{\mu_i} |x_t - \mu_i| \)

- ② ③ compute: \( v_i = \sum_{x_t: \text{NN}(x_t) = \mu_i} x_t - \mu_i \)

- concatenate \( v_i \)'s + \( \ell_2 \) normalize

A first example: the VLAD

A graphical representation of

\[ v_i = \sum_{x_t: \text{NN}(x_t) = \mu_i} x_t - \mu_i \]

The Fisher vector
Score function

Given a likelihood function $u_\lambda$ with parameters $\lambda$, the score function of a given sample $X$ is given by:

$$G^X_\lambda = \nabla_\lambda \log u_\lambda(X)$$

→ Fixed-length vector whose dimensionality depends only on # parameters.

Intuition: direction in which the parameters $\lambda$ of the model should be modified to better fit the data.
The Fisher vector
Fisher information matrix

Fisher information matrix (FIM) or negative Hessian:

\[ F_\lambda = E_{x \sim u_\lambda} \left[ \nabla_\lambda \log u_\lambda(x) \nabla_\lambda \log u_\lambda(x)' \right] \]

Measure similarity between using the Fisher Kernel (FK):

\[ K(X, Y) = G_X^\lambda F_\lambda^{-1} G_Y^\lambda \]


→ can be interpreted as a score whitening

The Fisher information matrix can be decomposed as

\[ F_\lambda^{-1} = L_\lambda' L_\lambda \]

and the FK can be rewritten as a dot product between Fisher Vectors (FV):

\[ G_X^\lambda = L_\lambda G_X^\lambda \]
The Fisher vector

Application to images

\[ X = \{ x_t, t = 1 \ldots T \} \] is the set of T i.i.d. D-dim local descriptors (e.g. SIFT) extracted from an image:

\[
G^X_\lambda = \frac{1}{T} \sum_{t=1}^{T} \nabla_\lambda \log u_\lambda(x_t)
\]

→ **average pooling** is a direct consequence of independence assumption

\[ u_\lambda(x) = \sum_{i=1}^{K} w_i u_i(x) \] is a Gaussian Mixture Model (GMM) with parameters \( \lambda = \{ w_i, \mu_i, \Sigma_i, i = 1 \ldots N \} \) trained on a large set of local descriptors

→ a probabilistic **visual vocabulary**

Perronnin and Dance, “Fisher kernels on visual categories for image categorization”, CVPR’07.
The Fisher vector
Relationship with the BOV

FV formulas:

Perronnin and Dance, “Fisher kernels on visual categories for image categorization”, CVPR’07.
The Fisher vector
Relationship with the BOV

FV formulas:

- gradient wrt to w

\[
\gamma_t(i) \approx \frac{1}{T} \sum_{t=1}^{T} \gamma_t(i)
\]

→ soft BOV

\[\gamma_t(i) = \text{soft-assignment of patch } t \text{ to Gaussian } i\]

Perronnin and Dance, “Fisher kernels on visual categories for image categorization”, CVPR’07.
FV formulas:

- gradient wrt to \( w \)

\[
\gamma_t(i) = \frac{1}{T} \sum_{t=1}^{T} \gamma_t(i) \]

\[\Rightarrow\text{soft BOV}\]

- gradient wrt to \( \mu \) and \( \sigma \)

\[
\begin{align*}
G_{\mu,i} &= \frac{1}{T \sqrt{w_i}} \sum_{t=1}^{T} \gamma_t(i) \left( \frac{x_t - \mu_i}{\sigma_i} \right) \\
G_{\sigma,i} &= \frac{1}{T \sqrt{2w_i}} \sum_{t=1}^{T} \gamma_t(i) \left[ \frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right]
\end{align*}
\]

\( \gamma_t(i) \) = soft-assignment of patch \( t \) to Gaussian \( i \)

\[\Rightarrow\text{compared to BOV, include \textbf{higher-order statistics} (up to order 2)}\]

Let us denote: \( D = \text{feature dim}, \) \( N = \# \text{Gaussians} \)

- BOV = \( N \)-dim
- FV = \( 2DN \)-dim

Perronnin and Dance, “Fisher kernels on visual categories for image categorization”, CVPR’07.
The Fisher vector
Relationship with the BOV

FV formulas:

• gradient wrt to \( w \)

\[
\gamma(i) = \sum_{t=1}^{T} \gamma_{t}(i)
\]

• soft BOV

• gradient wrt to \( \mu \) and \( \sigma \)

\[
G_{\mu,i} = \frac{1}{T \sqrt{w_i}} \sum_{t=1}^{T} \gamma_{t}(i) \left( \frac{x_t - \mu_i}{\sigma_i} \right)
\]

\[
G_{\sigma,i} = \frac{1}{T \sqrt{2w_i}} \sum_{t=1}^{T} \gamma_{t}(i) \left[ \frac{(x_t - \mu_i)^2}{\sigma_i^2} - 1 \right]
\]

\( \gamma_{t}(i) \) = soft-assignment of patch \( t \) to Gaussian \( i \)

→ compared to BOV, include higher-order statistics (up to order 2)

→ FV much higher-dim than BOV for a given visual vocabulary size

→ FV much faster to compute than BOV for a given feature dim

Perronnin and Dance, “Fisher kernels on visual categories for image categorization”, CVPR’07.
The Fisher vector
Dimensionality reduction on local descriptors

Perform PCA on local descriptors:

→ uncorrelated features are more consistent with diagonal assumption of covariance matrices in GMM

→ FK performs whitening and enhances low-energy (possibly noisy) dimensions
The Fisher vector
Dimensionality reduction on local descriptors

Perform PCA on local descriptors:

→ uncorrelated features are more consistent with diagonal assumption of covariance matrices in GMM

→ FK performs whitening and enhances low-energy (possibly noisy) dimensions

The Fisher vector: power-law

As in BOV, the Fisher vector representation suffers from

1) Over-counting similar pattern

2) Bursty visual elements [J’09]

Effective solution: Signed component-wise Power-law

\[ f(z) = \text{sign}(z)|z|^\alpha \text{ with } 0 \leq \alpha \leq 1 \]

(with \( \alpha=0.5 \) by default)

The VLAD can be viewed as a non-probabilistic version of the FV:
→ replace GMM clustering by k-means

\[ G_{\mu,i}^X = \frac{1}{T} \sqrt{w_i} \sum_{t=1}^{T} \gamma_t(i) \left( \frac{x_t - \mu_i}{\sigma_i} \right) \]
\[ \Rightarrow v_i = \sum_{x_t: \text{NN}(x_t) = \mu_i} x_t - \mu_i \]

Main differences: in contrast to VLAD, Fisher
• Performs **soft assignment** of descriptors
• Implicitly **whiten** the components
• High-order statistics are included

→ extension of the VLAD to include 2nd order statistics: VLAT
Picard and Gosselin, “Improving image similarity with vectors of locally aggregated tensors”, ICIP ‘11.

Remark on the supervector [Zhou 11]: **SV \approx BOV + VLAD**
Examples
Retrieval

Example on Holidays:


<table>
<thead>
<tr>
<th>Descriptor</th>
<th>$K$</th>
<th>$D$</th>
<th>$D' = D$</th>
<th>$D' = 2048$</th>
<th>$D' = 512$</th>
<th>$D' = 128$</th>
<th>$D' = 64$</th>
<th>$D' = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOW</td>
<td>1000</td>
<td>1000</td>
<td>40.1</td>
<td>43.5</td>
<td>44.4</td>
<td>43.4</td>
<td>40.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>20000</td>
<td>43.7</td>
<td>41.8</td>
<td>44.9</td>
<td>45.2</td>
<td>44.4</td>
<td>41.8</td>
</tr>
<tr>
<td>Fisher ($\mu$)</td>
<td>16</td>
<td>1024</td>
<td>54.0</td>
<td>54.6</td>
<td>52.3</td>
<td>49.9</td>
<td>46.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>59.5</td>
<td>60.7</td>
<td>61.0</td>
<td>56.5</td>
<td>52.0</td>
<td>48.0</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>62.5</td>
<td>62.6</td>
<td>57.0</td>
<td>53.8</td>
<td>50.6</td>
<td>48.6</td>
</tr>
<tr>
<td>VLAD</td>
<td>16</td>
<td>1024</td>
<td>52.0</td>
<td>52.7</td>
<td>52.6</td>
<td>50.5</td>
<td>47.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>55.6</td>
<td>57.6</td>
<td>59.8</td>
<td>55.7</td>
<td>52.3</td>
<td>48.4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>58.7</td>
<td>62.1</td>
<td>56.7</td>
<td>54.2</td>
<td>51.3</td>
<td>48.1</td>
</tr>
</tbody>
</table>
Example on Holidays:


<table>
<thead>
<tr>
<th>Descriptor</th>
<th>$K$</th>
<th>$D$</th>
<th>$D' = D$</th>
<th>$D'=2048$</th>
<th>$D'=512$</th>
<th>$D'=128$</th>
<th>$D'=64$</th>
<th>$D'=32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOW</td>
<td>1000</td>
<td>1000</td>
<td>40.1</td>
<td>43.5</td>
<td>44.4</td>
<td>43.4</td>
<td>40.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>20000</td>
<td>43.7</td>
<td>41.8</td>
<td>44.9</td>
<td>45.2</td>
<td>44.4</td>
<td>41.8</td>
</tr>
<tr>
<td>Fisher ($\mu$)</td>
<td>16</td>
<td>1024</td>
<td>54.0</td>
<td>54.6</td>
<td>52.3</td>
<td>49.9</td>
<td>46.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>59.5</td>
<td>60.7</td>
<td>61.0</td>
<td>56.5</td>
<td>52.0</td>
<td>48.0</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>62.5</td>
<td>62.6</td>
<td>57.0</td>
<td>53.8</td>
<td>50.6</td>
<td>48.6</td>
</tr>
<tr>
<td>VLAD</td>
<td>16</td>
<td>1024</td>
<td>52.0</td>
<td>52.7</td>
<td>52.6</td>
<td>50.5</td>
<td>47.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>55.6</td>
<td>57.6</td>
<td>59.8</td>
<td>55.7</td>
<td>52.3</td>
<td>48.4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>58.7</td>
<td>62.1</td>
<td>56.7</td>
<td>54.2</td>
<td>51.3</td>
<td>48.1</td>
</tr>
</tbody>
</table>

→ second order statistics are not essential for retrieval
Examples
Retrieval

Example on Holidays:


<table>
<thead>
<tr>
<th>Descriptor</th>
<th>$K$</th>
<th>$D$</th>
<th>$D' = D$</th>
<th>$D' = 2048$</th>
<th>$D' = 512$</th>
<th>$D' = 128$</th>
<th>$D' = 64$</th>
<th>$D' = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOW</td>
<td>1000</td>
<td>1000</td>
<td>40.1</td>
<td>43.5</td>
<td>44.4</td>
<td>43.4</td>
<td>40.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>20000</td>
<td>43.7</td>
<td>41.8</td>
<td>44.9</td>
<td>45.2</td>
<td>44.4</td>
<td>41.8</td>
</tr>
<tr>
<td>Fisher ($\mu$)</td>
<td>16</td>
<td>1024</td>
<td>54.0</td>
<td>54.6</td>
<td>52.3</td>
<td>49.9</td>
<td>46.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>59.5</td>
<td>60.7</td>
<td>61.0</td>
<td>56.5</td>
<td>52.0</td>
<td>48.0</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>62.5</td>
<td>62.6</td>
<td>57.0</td>
<td>53.8</td>
<td>50.6</td>
<td>48.6</td>
</tr>
<tr>
<td>VLAD</td>
<td>16</td>
<td>1024</td>
<td>52.0</td>
<td>52.7</td>
<td>52.6</td>
<td>50.5</td>
<td>47.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>55.6</td>
<td>57.6</td>
<td>59.8</td>
<td>55.7</td>
<td>52.3</td>
<td>48.4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>58.7</td>
<td>62.1</td>
<td>56.7</td>
<td>54.2</td>
<td>51.3</td>
<td>48.1</td>
</tr>
</tbody>
</table>

→ second order statistics are not essential for retrieval
→ even for the same feature dim, the FV/VLAD may beat the BOV
Examples
Retrieval

Example on Holidays:


<table>
<thead>
<tr>
<th>Descriptor</th>
<th>$K$</th>
<th>$D$</th>
<th>$D' = D$</th>
<th>$D' = 2048$</th>
<th>$D' = 512$</th>
<th>$D' = 128$</th>
<th>$D' = 64$</th>
<th>$D' = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOW</td>
<td>1000</td>
<td>1000</td>
<td>40.1</td>
<td>43.5</td>
<td>44.4</td>
<td>43.4</td>
<td>40.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>20000</td>
<td>43.7</td>
<td>41.8</td>
<td>44.9</td>
<td>45.2</td>
<td>44.4</td>
<td>41.8</td>
</tr>
<tr>
<td>Fisher ($\mu$)</td>
<td>16</td>
<td>1024</td>
<td>54.0</td>
<td>54.6</td>
<td>52.3</td>
<td>49.9</td>
<td>46.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>59.5</td>
<td>60.7</td>
<td>61.0</td>
<td>56.5</td>
<td>52.0</td>
<td>48.0</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>62.5</td>
<td>62.6</td>
<td>57.0</td>
<td>53.8</td>
<td>50.6</td>
<td>48.6</td>
</tr>
<tr>
<td>VLAD</td>
<td>16</td>
<td>1024</td>
<td>52.0</td>
<td>52.7</td>
<td>52.6</td>
<td>50.5</td>
<td>47.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>55.6</td>
<td>57.6</td>
<td>59.8</td>
<td>55.7</td>
<td>52.3</td>
<td>48.4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>58.7</td>
<td>62.1</td>
<td>56.7</td>
<td>54.2</td>
<td>51.3</td>
<td>48.1</td>
</tr>
</tbody>
</table>

→ second order statistics are not essential for retrieval
→ even for the same feature dim, the FV/VLAD can beat the BOV
→ soft assignment + whitening of FV helps when number of Gaussians ↑
## Examples

### Retrieval

**Example on Holidays:**


<table>
<thead>
<tr>
<th>Descriptor</th>
<th>$K$</th>
<th>$D$</th>
<th>$D' = D$</th>
<th>$D' = 2048$</th>
<th>$D' = 512$</th>
<th>$D' = 128$</th>
<th>$D' = 64$</th>
<th>$D' = 32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOW</td>
<td>1000</td>
<td>1000</td>
<td>40.1</td>
<td>43.5</td>
<td>44.4</td>
<td>43.4</td>
<td>40.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20000</td>
<td>20000</td>
<td>43.7</td>
<td>41.8</td>
<td>44.9</td>
<td>45.2</td>
<td>44.4</td>
<td>41.8</td>
</tr>
<tr>
<td>Fisher ($\mu$)</td>
<td>16</td>
<td>1024</td>
<td>54.0</td>
<td>54.6</td>
<td>52.3</td>
<td>49.9</td>
<td>46.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>59.5</td>
<td>60.7</td>
<td>56.5</td>
<td>52.0</td>
<td>48.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>62.5</td>
<td>62.6</td>
<td>57.0</td>
<td>53.8</td>
<td>50.6</td>
<td>48.6</td>
</tr>
<tr>
<td>VLAD</td>
<td>16</td>
<td>1024</td>
<td>52.0</td>
<td>52.7</td>
<td>52.6</td>
<td>50.5</td>
<td>47.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4096</td>
<td>55.6</td>
<td>57.6</td>
<td>59.8</td>
<td>55.7</td>
<td>52.3</td>
<td>48.4</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>16384</td>
<td>58.7</td>
<td>62.1</td>
<td>56.7</td>
<td>54.2</td>
<td>51.3</td>
<td>48.1</td>
</tr>
</tbody>
</table>

→ second order statistics are not essential for retrieval
→ even for the same feature dim, the FV/VLAD can beat the BOV
→ soft assignment + whitening of FV helps when number of Gaussians ↑
→ after dim-reduction however, the FV and VLAD perform similarly
Packages for Fisher vectors

The INRIA package (VLAD also available):
http://lear.inrialpes.fr/src/inria_fisher/

The Oxford package:
http://www.robots.ox.ac.uk/~vgg/research/encoding_eval/
Questions?
Larger-scale visual recognition
Efficient matching

Hervé Jégou, INRIA
General outline

- PART I: Introduction
  - Applications and datasets
  - Image description and matching

- PART II: Large-scale image search
  - The bag-of-word representation and some extension

- PART III: Larger-scale image search
  - Novel aggregation mechanisms
  - Efficient indexing

- Conclusions
Efficient matching: outline

- Preliminary
- Locality Sensitive Hashing: the two modes
- Hamming Embedding
- Searching with Product Quantization
Finding neighbors

- Nearest neighbor search is a critical step in object recognition
  - To compute the image descriptor itself
    E.g., assignment with k-means to a large vocabulary
  - To find the most similar images/patches in a database
  - For instance, the closest one w.r.t to Euclidean distance:
    \[ \text{NN}(x) = \arg \min_{y \in \mathcal{Y}} \|x - y\|^2 \]

- Problems:
  - costly operation of exact exhaustive search: \(O(n^d)\)
  - High-dimensional vectors: for exact search the best approach is the naïve exhaustive comparison
The cost of (efficient) exact matching

- But what about the actual timings? With an efficient implementation!

- Finding the 10-NN of 1000 distinct queries in 1 million vectors
  - Assuming 128-D Euclidean descriptors
  - i.e., 1 billion distances, computed on an 8-core machine

Poll: How much time?
The cost of (efficient) exact matching

- But what about the actual timings? With an efficient implementation!

- Finding the 10-NN of 1000 distinct queries in 1 million vectors
  - Assuming 128-D Euclidean descriptors
  - i.e., 1 billion distances, computed on a 8-core machine

  **5.5 seconds**

- Assigning 2000 SIFTs to a visual vocabulary of size k=100,000
  - 1.2 second
Need for approximate nearest neighbors

- 1 million images, 1000 descriptors per image
  - 1 billion distances per local descriptor
  - $10^{12}$ distances in total
  - 1 hour 30 minutes to perform the query for Euclidean vectors

- To improve the scalability:
  - We allow to find the nearest neighbors in probability only:
    Approximate nearest neighbor (ANN) search

- Three (contradictory) performance criteria for ANN schemes
  - search quality (retrieved vectors are actual nearest neighbors)
  - speed
  - memory usage
Efficient matching: outline

- Preliminary
- Locality Sensitive Hashing: the two modes
- Hamming Embedding
- Searching with Product Quantization
Locality Sensitive Hashing (LSH)

- Most known ANN technique [Charikar 98, Gionis 99, Datar 04,…]

- But “LSH” is associated with two distinct search algorithms
  - As an indexing technique involving several hash functions
  - As a binarization technique
LSH – partitioning technique

- General idea:
  - Define m hash functions in parallel
  - Each vector: associated with m distinct hash keys
  - Each hash key is associated with a hash table

- At query time:
  - Compute the hash keys associated with the query
  - For each hash function, retrieve all the database vectors assigned to the same key (for this hash function)
  - Compute the exact distance on this short-list
What kind of hash functions/partitions?

- Any hash function can be used in LSH
  - Just need a set of functions \( f_j : \mathbb{R}^d \rightarrow \mathbb{K} \)

- Usually, random projection + scalar quantization

- Could be
  - Structured lattice quantizers [Andoni’06, J’08]
  - k-means, Hierarchical k-means, KD-trees
Hash functions – Structured vs Learned

- Learned quantizers are better than structured quantizers
- Evaluation search quality for a single hash function [Pauleve’10]:

HKM: loss compared with k-means
Multi-probe LSH

- But multiple hash functions use a lot of memory
  - Per vector and per hash table: at least an id

- Multi-probe LSH [Lv 07]
  - Use less hash functions (possibly 1)
  - But probe several (closest) cells per hash function
    - save a lot of memory
  - Similar in spirit to Multiple-assignment with BOV
FLANN

- ANN package described in Muja’s VISAPP paper [Muja 09]
  - Multiple kd-tree or k-means tree
  - With auto-tuning under given constraints
  - Remark: self-tuned LSH proposed in [Dong 07]
  - Still high memory requirement for large vector sets

- Excellent package: high integration quality and interface!


**FLANN - Fast Library for Approximate Nearest Neighbors**

*What is FLANN?*

FLANN is a library for performing fast approximate nearest neighbor searches in high dimensional spaces. It contains a collection of algorithms we found to work best for nearest neighbor search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.

FLANN is written in C++ and contains bindings for the following languages: C, MATLAB and Python.

*News*

- (20 December 2011) Version 1.7.0 is out bringing two new index types and several other improvements.
- You can find binary installers for FLANN on the Point Cloud Library project page. Thanks to the PCL developers!
- Mac OS X users can install flann though MacPorts (thanks to Mark Moll for maintaining the Portfile)
- New release introducing an easier way to use custom distances, kd-tree implementation optimized for low dimensionality search and experimental VPI support
- New release introducing new C++ templated API, thread-safe search, savoofd of indexes and more.
- The FLANN license was changed from LGPL to BSD
For this second (“re-ranking”) stage, we need raw descriptors, i.e.,
- either huge amount of memory → 128GB for 1 billion SIFTs
- either to perform disk accesses → severely impacts efficiency
Issue for large scale: final verification

- Some techniques –like BOV– keep all vectors (no verification)

- Better: use very short codes for the filtering stage
  - Hamming Embedding [J’08] or Product Quantization [J’11]
LSH for binarization [Charikar’ 98, J.’08, Weiss’09, etc]

- **Idea:** design/learn a function mapping the original space into the compact Hamming space:
  \[ e : \mathbb{R}^d \rightarrow \{0, 1\}^D \]
  \[ x \rightarrow e(x) \]

- **Objective:** neighborhood in the Hamming space try to reflect original neighborhood
  \[ \arg \min_i h(e(x), e(y_i)) \approx \arg \min_i d(x, y) \]

- **Advantages:** compact descriptor, fast comparison
LSH for binarization [Charikar’ 98, J.’08, Weiss’09, etc]

- Given $B$ random projection direction $a_i$
- Compute a binary code from a vector $x$ as

$$b_i(x) = \text{sign } a_i^T x$$

$$b(x) = (b_1(x), \ldots, b_B(x))$$

- Spectral Hashing: theoretical framework for finding hash functions
- In practice: PCA + binarization on the different axis (based on variance)
LSH: the two modes – approximate guidelines

**Partitioning technique**
- **Sublinear/non exhaustive** search
- Several hash indexes (integer)
- **Large memory overhead**
  - Hash table overhead (store ids)
- Need original vectors for re-ranking
  - Need a lot of memory
  - Or to access the disk
- Interesting when (e.g., FLANN)
  - Not too large dimensionality
  - Dataset small enough (memory)
- Very good variants/software (FLANN)

**Binarization technique**
- **Linear** search
- Produce a binary code per vector
- **Very compact**
  - bit-vectors, concatenated (no ids)
- **Very fast comparison**
  - Hamming distance (popcnt SSE4)
  - 1 billion comparisons/second
- Interesting
  - For very high-dimensional vectors
  - When memory is critical
- Simple to implement. Very active problems with many variants
LSH: the two modes – approximate guidelines

**Partitioning technique**
- **Sublinear** search
- Several hash indexes (integer)
  - Typical usage: Searching local descriptors
    - Dataset small enough (memory)
    - Very good variants/software (FLANN)

**Binarization technique**
- **Linear** search
- Produce a binary code per vector
  - Typical usage: Index global (or aggregated) descriptors
    - When memory is critical
    - Simple to implement. Very active problems with many variants

Interesting when (e.g., FLANN)
- Not too large dimensionality
- Dataset small enough (memory)
  - Very good variants/software (FLANN)
Outline

- Preliminary
- Locality Sensitive Hashing: the two modes
- Hamming Embedding
- Searching with Product Quantization
Hamming Embedding

- Introduced as an extension of BOV [J’08]

- Combination of
  - A partitioning technique (k-means)
  - A binary code that refine the descriptor

Representation of a descriptor $x$
- Vector-quantized to $q(x)$ as in standard BOV
- short binary vector $b(x)$ for an additional localization in the Voronoi cell

- Two descriptors $x$ and $y$ match iif

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

Where $h(\ldots)$ denotes the Hamming distance
ANN evaluation of Hamming Embedding

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

Hamming Embedding provides a much better trade-off between recall and remove false positives
Matching points - 20k word vocabulary

201 matches

240 matches

Many matches with the non-corresponding image!
Matching points - 200k word vocabulary

69 matches

35 matches

Still many matches with the non-corresponding one
Matching points - 20k word vocabulary + HE

83 matches

8 matches

10x more matches with the corresponding image!
Outline

- Preliminary
- Locality Sensitive Hashing: the two modes
- Hamming Embedding
- Searching with Product Quantization
A typical source coding system

Simple source coding system:
- Decorrelation, e.g., PCA
- Quantization
- Entropy coding

To a code $e(x)$ is associated a unique reconstruction value $q(x)$
\[ \Rightarrow \text{i.e., the visual word} \]

- Focus on quantization (lossy step)
Relationship between Reconstruction and Distance estimation

- Assume $y$ quantized to $q_c(y)$
  - $x$ is a query vector

- If we estimate the distance by
  \[ d(x, y) \approx d(x, q_c(y)) \]

- Then we can show that:
  \[
  \mathbb{E}_Y [(d(x, y) - d(x, q_c(y)))^2] \leq \mathbb{E}_Y [(y - q_c(y))^2] = \text{MSE}
  \]

  i.e., the error on the square distance is statistically bounded by the quantization error
Searching with quantization [J’11]

- Main idea: compressed representation of the database vectors
  - Each database vector $y$ is represented by $q_c(y)$ where $q_c(.)$ is a **product quantizer**

\[
d(x, y) \approx d(x, q_c(y))
\]

- Search = distance approximation problem

- **The key**: Estimate the distances in the **compressed domain** such that
  - Quantization is fast enough
  - Quantization is precise, i.e., many different possible indexes (ex: $2^{64}$)

- Regular k-means is not appropriate: not for $k=2^{64}$ centroids
Product Quantizer

- Vector split into $m$ subvectors: $y \rightarrow [y_1 | \cdots | y_m]$
- Subvectors are quantized separately
- Example: $y =$ 16-dim vector split in 8 subvectors of dimension 16

$y_1$: 2 components

$\Rightarrow$ 24-bit quantization index

- In practice: 8 bits/subquantizer (256 centroids),
  - SIFT: $m=4-16$
  - VLAD/Fisher: 4-128 bytes per indexed vector
Asymmetric distance computation (ADC)

- Compute the square distance approximation in the compressed domain
  \[ d(x, y)^2 \approx \sum_{i=1}^{m} d(x_i, q_i(y_i))^2 \]

- To compute distance between query \( x \) and many codes
  - compute \( d(x_i, c_{i,j})^2 \) for each subvector \( x_i \) and all possible centroids
    - stored in look-up tables
    - fixed cost for quantization
  - for each database code: sum the elementary square distances

- Each 8x8=64-bits code requires only \( m=8 \) additions per distance
- IVFADC: combination with an inverted file to avoid exhaustive search
Estimated distances versus true distances
Combination with an inverted file system

ALGORITHM

1. Coarse k-means hash function

   Select $k'$ closest centroids $c_i$ and corresponding cells

2. Compute the residual vector $x-c_i$ of the query vector

3. Encode the residual vector by PQ

4. Apply the PQ search method. Distance is approximated by $d(x,y) = d(x-c_i, q(y-c_i))$

Example timing: 3.5 ms per vector for a search in 2 billion vectors
Performance evaluation

- Comparison with other memory efficient approximate neighbor search techniques, i.e., binarization techniques
  - Spectral Hashing [Weiss 09] – exhaustive search
  - Hamming Embedding [J’08] – non exhaustive search

- Performance measured by searching 1M vector (recall@R, varying R)

Searching in 1M SIFT descriptors

```
1
0.8
0.6
0.4
0.2
1
10
100
1k
10k
100k
1M
```

Searching in 1M GIST descriptors

```
1
10
100
1k
10k
100k
1M
```
Product Quantization: some applications

- PQ search was first proposed for searching local descriptors [J’09-11], i.e., to replace bag-of-words or Hamming Embedding
  - [J’10]: Encoding a global image representation (Vlad/Fisher)
  - [Gammeter et al’10]: Fast geometrical re-ranking with local descriptors
  - [Perronnin et al.’11]: Large scale classification (Imagenet)
    - Combined with Stochastic Gradient Descent SVM
    - Decompression on-the-fly when feeding the classifier
    - Won the ILSVRC competition in 2011

- Wider scope than pure search: Approximation of the inner product
  - Learning in the PQ-compressed domain [Vedaldi’12, Harchaoui’12]
Concluding remarks

Nearest neighbor search is a key component of image indexing systems
Must be considered jointly with the image representation!

Product quantization-based approach offers
- Competitive search accuracy
- Compact footprint: few bytes per indexed vector

Tested
- on local image descriptors (up to 2 billions)
- global or aggregated descriptors (200 millions)
- audio, text descriptors
- any descriptor compared with L2 distance/Cosine (and actually more)

Toy Matlab package available on my web page
Larger-scale visual recognition

Conclusion

Hervé Jégou, INRIA
General outline

PART I: Introduction
  • Applications and datasets
  • Image description and matching

PART II: Large-scale image search
  • The bag-of-word representation and some extension

PART III: Larger-scale image search
  • Novel aggregation mechanisms
  • Efficient indexing

Conclusion
Large-/larger-scale image search

Large-scale (1-5 millions): BOV is still state-of-the-art with proper extensions
- Improved matching extension (Soft assignment, Hamming Embedding, …)
- Re-ranking with spatial verification, or integrated geometry
- Query-expansion

Larger-scale (100M-1B+): historically global descriptors, but better to use
- SIFT extraction (better: dense)
- An improved aggregation mechanisms – The Fisher kernel (or variants)
- An efficient indexing technique – Product quantization
Large Scale Experiments

Holidays + up to 10M distractors from Flickr

- 320B / image
- exhaustive, 7s
- 16B, 45ms

Database size

mAP

BOW, K=200k
Fisher K=64, D=4096
Fisher K=64, PCA D'=96
Fisher K=64, IVFADC 64/8192, 16x8
Fisher K=256, IVFADC 64/8192, 256x10
Large Scale Experiments

Short list quality in 10M images
Very Large Scale Experiments

Copydays + 100M distractors from Exalead (copy detection setup)

Crop 50% of image surface

Strong transformations

- GIST
- GISTIS
- Fisher
- Fisher+IVFPQ

64B, 245ms

64B, 160ms
Final note: search vs classification

Query-by-example retrieval of images/objects/location/etc:

Classification / annotation:

Convergence of large-scale retrieval and classification:

→ retrieval: more and more machine learning
→ classification: more and more cost aware
General conclusions

Tools to handle large-scale datasets:
→ image representations: scaling the BOV, extensions
→ including higher order statistics (VLAD, FV)
→ scalable matching: compressed-domain indexing

Very large-scale image search does not necessarily require gigantic resources:
→ searching in 100M images in 250ms on a single processor

DEMO!