Compressed local descriptors for fast image and video search in large databases

Matthijs Douze

joint work with Hervé Jégou, Cordelia Schmid and Patrick Pérez

1: INRIA Rennes, TEMEX team, France
2: INRIA Grenoble, LEAR team, France
3: Technicolor, France
Problem setup: Image indexing

- Retrieval of images representing the same object/scene:
  - different viewpoints, backgrounds, …
  - copyright attacks: cropping, editing, …
  - short response time
  - **100s of millions** of images or 1000s of hours of video queries

relevant answers
Related work on large scale image search

- **Global descriptors:**
  - color/texture statistics
  - GIST descriptors with Spectral Hashing or similar techniques [Torralba & al 08]
    → very limited invariance to scale/rotation/crop

- **Local descriptors** → compact them: Bag of Features [Sivic & Zissermann 03]
  - Improvements: hierarchical vocabulary, compressed BoF, partial geometry...
    → But still hundreds of bytes are required to obtain a “reasonable quality”
Outline

Image description with VLAD

Indexing with the product quantizer

Porting to mobile devices

Video indexing
Objective and proposed approach [Jégou & al., CVPR 10]

- Aim: optimizing the trade-off between
  - search speed +
  - memory usage +
  - search quality -

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

![Diagram showing the process from extract SIFT to vector encoding/indexing]
Aggregation of local descriptors

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

  set of $n$ local descriptors $\rightarrow$ 1 vector

- Indexing:
  - similarity = distance between aggregated description vectors (preferably L2)
  - search = (approximate) nearest-neighbor search in descriptor space

- Most popular idea: BoF representation [Sivic & Zisserman 03]
  - sparse vector
  - highly dimensional
  $\rightarrow$ dimensionality reduction harms precision a lot

- Alternative: Fisher Kernels [Perronnin et al 07]
  - non sparse vector
  - excellent results with a small vector dimensionality
  $\rightarrow$ VLAD is in the spirit of this representation
VLAD: Vector of Locally Aggregated Descriptors

- D-dimensional descriptor space (SIFT: D=128)
- \( k \) centroids: \( c_1, \ldots, c_k \)
VLAD: Vector of Locally Aggregated Descriptors

- D-dimensional descriptor space (SIFT: D=128)
- $k$ centroids: $c_1, \ldots, c_k$

Output: $v_1 \ldots v_k$ = descriptor of size $k \times D$
- L2-normalized
- Typical $k = 16$ or 64: descriptor in 2048 or 8192 D
- Similarity measure = L2 distance.
VLADs for corresponding images

*SIFT-like representation per centroid (>0 components: blue, <0 components: red)*

- good coincidence of energy & orientations
### VLAD performance and dimensionality reduction

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

<table>
<thead>
<tr>
<th>Aggregator</th>
<th>k</th>
<th>D</th>
<th>D’=D (no reduction)</th>
<th>D’=128</th>
<th>D’=64</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoF</td>
<td>1,000</td>
<td>1,000</td>
<td>41.4</td>
<td>44.4</td>
<td>43.4</td>
</tr>
<tr>
<td>BoF</td>
<td>20,000</td>
<td>20,000</td>
<td>44.6</td>
<td>45.2</td>
<td>44.5</td>
</tr>
<tr>
<td>BoF</td>
<td>200,000</td>
<td>200,000</td>
<td>54.9</td>
<td>43.2</td>
<td>41.6</td>
</tr>
<tr>
<td>VLAD</td>
<td>16</td>
<td>2,048</td>
<td>49.6</td>
<td>49.5</td>
<td><strong>49.4</strong></td>
</tr>
<tr>
<td>VLAD</td>
<td>64</td>
<td>8,192</td>
<td>52.6</td>
<td><strong>51.0</strong></td>
<td>47.7</td>
</tr>
<tr>
<td>VLAD</td>
<td>256</td>
<td>32,768</td>
<td><strong>57.5</strong></td>
<td>50.8</td>
<td>47.6</td>
</tr>
</tbody>
</table>

- Observations:
  - performance increases with k
  - VLAD better than BoF for a given descriptor size
  - if small D’ needed: choose a smaller k
Outline

Image description with VLAD

**Indexing with the product quantizer**

Porting to mobile devices

Video indexing
Indexing algorithm: searching with quantization
[Jégou & al., PAMI to appear]

- Search/Indexing = distance approximation problem
- The distance between a query vector $x$ and a database vector $y$ is estimated by

$$d(x, y) \approx d(x, q(y))$$

where $q(.)$ is a quantizer

→ vector-to-code distance

- The choice of the quantizer is critical
  - fine quantizer → need many centroids: typically 64-bit codes → $k=2^{64}$
  - regular (and approximate) k-means cannot be used
Product quantization for nearest neighbor search

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

- Subvectors are quantized separately

\[
q(y) = [q_1(y_1) | \ldots | q_m(y_m)]
\]

where each \( q_i \) is learned by \( k \)-means with a limited number of centroids

- Example: \( y = 128 \)-dim vector split in 8 subvectors of dimension 16

\[
\begin{array}{cccccccc}
& y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 \\
\end{array}
\]

256 centroids

16 components

\[
\begin{array}{cccccccc}
q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 & q_8 \\
\end{array}
\]

8 bits

\[
\begin{array}{cccccccc}
q_1(y_1) & q_2(y_2) & q_3(y_3) & q_4(y_4) & q_5(y_5) & q_6(y_6) & q_7(y_7) & q_8(y_8) \\
\end{array}
\]

\( \Rightarrow \) 64-bit quantization index
Product quantization for nearest neighbor search

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

- Subvectors are quantized separately

$$q(y) = [q_1(y_1) | \ldots | q_m(y_m)]$$

where each $q_i$ is learned by $k$-means with a limited number of centroids

- Example: $y = 128$-dim vector split in 8 subvectors of dimension 16

16 components

256 centroids

8 bits

⇒ 64-bit quantization index
Product quantizer: asymmetric distance computation (ADC)

- Compute the 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
  - for each database code: sum up the elementary squared distances

- Each 8x8=64-bits code requires only \( m = 8 \) additions per distance!
## Results on standard datasets

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

<table>
<thead>
<tr>
<th>Method</th>
<th>bytes</th>
<th>UKB</th>
<th>Holidays</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoF, k=20,000</td>
<td>10K</td>
<td>2.92</td>
<td>44.6</td>
</tr>
<tr>
<td>BoF, k=200,000</td>
<td>12K</td>
<td>3.06</td>
<td>54.9</td>
</tr>
<tr>
<td>miniBOF</td>
<td>20</td>
<td>2.07</td>
<td>25.5</td>
</tr>
<tr>
<td>miniBOF</td>
<td>160</td>
<td>2.72</td>
<td>40.3</td>
</tr>
<tr>
<td>VLAD k=16, ADC</td>
<td>16</td>
<td>2.88</td>
<td>46.0</td>
</tr>
<tr>
<td>VLAD k=64, ADC</td>
<td>64</td>
<td>3.10</td>
<td>49.5</td>
</tr>
</tbody>
</table>

miniBOF: “Packing Bag-of-Features”, ICCV’09
IVFADC: non-exhaustive ADC

- IVFADC
  - Additional quantization level
  - Combination with an inverted file
  - visits $1/128^{th}$ of the dataset

- Timings for 10 M images
  - Exhaustive search with ADC: 0.286 s
  - Non-exhaustive search with IVFADC: 0.014 s
Large scale experiments (10 million images)

Database size: Holidays+images from Flickr

- BOF D=200k
- VLAD k=64
- VLAD k=64, D'=96
- VLAD k=64, ADC 16 bytes
Outline

Image description with VLAD

Indexing with the product quantizer

**Porting to mobile devices**

Video indexing
## On the mobile

- **Indexing on the server:**

  ![Image](image_url)

  - Extract SIFT descriptors from the image.
  - Aggregate descriptors to get `D`.
  - Perform dimension reduction to get `D'`.
  - Send descriptors to the server.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Image</th>
<th>SIFTs</th>
<th>VLAD</th>
<th>VLAD+PCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data size</td>
<td>300 kB</td>
<td>512 kB</td>
<td>32 kB</td>
<td>384 bytes</td>
</tr>
<tr>
<td>Computing time (relative)</td>
<td>NA</td>
<td>1.5 s (50 ms for CS-LBP)</td>
<td>5 ms</td>
<td>0.5 ms</td>
</tr>
</tbody>
</table>

- **Query from mobile**
  - relatively cheap to compute
  - small bandwidth
Indexing on the mobile

- The database is stored on the device

- In addition to the previous:
  - database: 20 bytes per image in RAM
  - quantize query (find closest centroids + build look-up tables)
  - scan database to find nearest neighbors

- Adapt algorithms to optimize speed

<table>
<thead>
<tr>
<th>db size (images)</th>
<th>exhaustive (ADC) / non-exhaust. (IVFADC)</th>
<th>precompute distance tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1000</td>
<td>ADC</td>
<td>no</td>
</tr>
<tr>
<td>&lt;1M</td>
<td>ADC</td>
<td>yes</td>
</tr>
<tr>
<td>&gt;1M</td>
<td>IVFADC</td>
<td>yes</td>
</tr>
</tbody>
</table>
Outline

Image description with VLAD

Indexing with the product quantizer

Porting to mobile devices

Video indexing
Video indexing [Douze & al. ECCV 2010]

- video = image sequence
  - index VLAD descriptors for all images (CS-LBP instead of SIFT for speed)
  - temporal verification

- database side: images are grouped in segments
  - 1 VLAD descriptor represents each segment
  - frame represented as refinement w.r.t. this descriptor

- query = search all frames of the query video

- Frame matches → alignment of query with database video
  - Hough transform on $\delta t = t_q - t_{db}$
  - Output: most likely $\delta t \rightarrow$ alignments
  - map back to frame matches to find aligned video segments
Video indexing results

- Comparison with Trecvid 2008 copy detection task
  - 200 h indexed video
  - 2000 queries
  - 10 “attacks” = video editing, clutter, frame dropping, camcording...
  - state of the art: competition results (score = NDCR, lower = better)

<table>
<thead>
<tr>
<th>transformation</th>
<th>best</th>
<th>ours</th>
<th>rank (/23)</th>
</tr>
</thead>
<tbody>
<tr>
<td>camcording</td>
<td>0.08</td>
<td>0.22</td>
<td>2</td>
</tr>
<tr>
<td>picture in picture</td>
<td>0.02</td>
<td>0.32</td>
<td>4</td>
</tr>
<tr>
<td>insertion of patterns</td>
<td>0.02</td>
<td>0.08</td>
<td>3</td>
</tr>
<tr>
<td>strong re-encoding</td>
<td>0.02</td>
<td>0.06</td>
<td>2</td>
</tr>
<tr>
<td>geometric attacks</td>
<td>0.07</td>
<td>0.14</td>
<td>2</td>
</tr>
<tr>
<td>5 random transformations</td>
<td>0.20</td>
<td>0.54</td>
<td>2</td>
</tr>
</tbody>
</table>

- Observations:
  - Always among 5 first results
  - 5 times faster and 100 times less memory than competing methods
  - Best localization results (due to dense temporal sampling)
Conclusion

- VLAD: compact & discriminative image descriptor
  - aggregation of SIFT, CS-LBP, SURF (ongoing),...

- Product Quantizer: generic indexing method with nearest-neighbor search function
  - works with local descriptors and GIST, audio features (ongoing)...

- Standard image and datasets
  - Holidays (different viewpoints)
  - Copydays (copyright attacks)

- Compatible with mobile applications:
  - compact descriptor, cheap to compute

- Code for VLAD and Product quantizer at \url{http://www.irisa.fr/texmex/people/jegou/src.php}

- Demo!
Searching with quantization: comparison with spectral Hashing

GIST, 64-bit codes

recall@R

R

SDC
ADC
IVFADC w=1
IVFADC w=8
IVFADC w=64
spectral hashing
Impact of $D'$ on image retrieval

- The best choice of $D'$ found by minimizing the square error criterion is reasonably consistent with the optimum obtained when measuring the image search quality.
Results on 10 million images

- exact VLAD 64
- VLAD $k=64$, ADC 16x8
- VLAD $k=64$, IVFADC 16x8
- BOF, $k=200k$
Results: comparison with « Packing BOF » (Holidays dataset)
VLAD: other examples
Combination with an inverted file
Related work on large scale image search

- **Global descriptors:**
  - GIST descriptors with Spectral Hashing or similar techniques [Torralba & al 08]
    - very limited invariance to scale/rotation/crop: use local descriptors

- **Bag-of-features [Sivic & Zisserman 03]**
  - Large (hierarchical) vocabularies [Nister Stewenius 06]
  - Improved descriptor representation [Jégou et al 08, Philbin et al 08]
  - Geometry used in index [Jégou et al 08, Perdoc’h et al 09]
  - Query expansion [Chum et al 07]
    - memory tractable for a few million images only

- **Efficiency improved by**
  - Min-hash and Geometrical min-hash [Chum et al. 07-09]
  - compressing the BoF representation [Jégou et al. 09]
    - But still hundreds of bytes are required to obtain a “reasonable quality”