Aggregating local descriptors into a compact representation

Hervé Jégou¹, Matthijs Douze², Cordelia Schmid² and Patrick Pérez³

¹: INRIA Rennes, TEXMEX 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
  - billions of images

queries

relevant answers
Related work on large scale image search

- Most systems build upon the BoF framework [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”

- Alternative: GIST descriptors with Spectral Hashing or similar techniques
→ very limited invariance to scale/rotation/crop
Objective and proposed approach

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

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

```
extract SIFT → [5 34 14] → aggregate descriptors → [−0.234 0.452 0.134 0.001] → dimension reduction → [0.155 0.230 ... 0.435] → vector encoding /indexing → code
```

n SIFTS (128 dim) → D → D'
Aggregation of local descriptors

- Problem: represent an image by a single fixed-size vector:
  
  set of $n$ local descriptors $\rightarrow$ 1 vector

- Most popular idea: BoF representation [Sivic & Zisserman 03]
  - sparse vector
  - highly dimensional
  $\rightarrow$ strong dimensionality reduction introduces loss

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

- **Learning:** \( k \)-means
  - output: \( k \) centroids: \( c_1, \ldots, c_i, \ldots c_k \)

- **VLAD computation:**
  1. \( c(x) = \arg \min_{c_i} ||c_i - x||^2 \)
  2. \( v_i = \sum_{x : c(x) = c_i} (x - c_i) \)
  3. \( v = [v_1, \ldots, v_i, \ldots, v_k] \), \( v_i \in \mathbb{R}^{128} \)

\( \Rightarrow \) dimension \( D = k \times 128 \)

- L2-normalized
- Typical parameter: \( k=64 \) (\( D=8192 \))

\[\text{INRIA}\]
VLADs for corresponding images

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

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

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

<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:**
- VLAD better than BoF for a given descriptor size
  → comparable to Fisher kernels for these operating points
- Choose a small $D$ if output dimension $D'$ is small
Indexing algorithm: searching with quantization [Jegou et al. 10]

- 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
  - needs many centroids
  - regular k-means and approximate k-means can not be used
    → we typically want $k=2^{64}$ for 64-bit codes
Product quantization for nearest neighbor search

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

- Subvectors are quantized separately by quantizers

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

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

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

⇒ 64-bit quantization index
Product quantizer: 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
  - 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
Optimizing the dimension reduction and quantization together

- VLAD vectors suffer two approximations
  - mean square error from PCA projection: $e_p(D')$
  - mean square error from quantization: $e_q(D')$

- Given $k$ and bytes/image, choose $D'$ minimizing their sum

<table>
<thead>
<tr>
<th>Ex, $k=16$:</th>
<th>$D'$</th>
<th>$e_p(D')$</th>
<th>$e_q(D')$</th>
<th>$e_p(D')+e_q(D')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0.0632</td>
<td>0.0164</td>
<td>0.0796</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0.0508</td>
<td>0.0248</td>
<td>0.0757</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>0.0434</td>
<td>0.0321</td>
<td><strong>0.0755</strong></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>0.0386</td>
<td>0.0458</td>
<td>0.0844</td>
<td></td>
</tr>
</tbody>
</table>
# 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>40</td>
<td>3.10</td>
<td>49.5</td>
</tr>
</tbody>
</table>

miniBOF: “Packing Bag-of-Features”, ICCV’09
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
- VLAD+Spectral Hashing, 16 bytes

Timings:
- ADC: 0.286s
- IVFADC: 0.014s
- SH ≈ 0.267s
Conclusion

- Competitive search accuracy with a few dozen bytes per indexed image

- Tested on up to 220 million video frames
  - extrapolation for 1 billion images: 20GB RAM, query < 1s on 8 cores

- Matlab package available, includes:
  - VLAD
  - Indexing algorithm (ADC/IVFADC)
  - extracted descriptors

- Improved Fisher kernels by Perronnin et al., CVPR’2010

- 10 million images indexed on my laptop: