Scalable Learning and Indexing for Retrieval in Large Image Databases

Philippe-Henri Gosselin

ETIS Lab., CNRS UMR 8051
Professor at ENSEA, Cergy, France

October 2, 2014
Outline

1. Scalable indexing

2. Decentralized learning
Vote-based Systems¹

- Objective: Metric to compare two images
- Idea: "count" the number of matching descriptors
- Very effective for copy search (>99%)

Consider matching-based similarity

\[ K(B_i, B_j) = \sum_r \sum_s \text{matches}(b_{ri}, b_{sj}) \]

with \( b_{ri} \) visual descriptor \( r \) for document \( i \).
From images to small index

1. Consider matching-based similarity
2. Implicit embedding in Hilbert spaces of matching-based similarity
   ⇒ Kernel functions on Bags
\[
K(B_i, B_j) = \sum_r \sum_s \langle \phi(b_{ri}), \phi(b_{sj}) \rangle^p
\]
   with \( b_{ri} \) visual descriptor \( r \) for document \( i \).

---

From images to small index

1. Consider matching-based similarity
2. Implicit embedding in Hilbert spaces of matching-based similarity
3. Improve matching accuracy thanks to visual dictionaries

⇒ Distribute and center descriptors in clusters

\[ K(B_i, B_j) = \sum_c \sum_r \sum_s \langle \phi(b_{rci}) - \mu_c, \phi(b_{scj} - \mu_c) \rangle_p \]

\[ = \sum_c K_c(B_i, B_j) \]

with \( b_{ri} \) visual descriptor \( r \) in cluster \( c \) for document \( i \), and \( \mu_c \) mean of cluster \( c \).

---

From images to small index

1 Consider matching-based similarity
2 Implicit embedding in Hilbert spaces of matching-based similarity
3 Improve matching accuracy thanks to visual dictionaries
4 Explicit embedding thanks to linearisation

\[
K_c(B_i, B_j) = \sum_r \sum_s \langle \phi(b_{rci}), \phi(b_{scj}) \rangle^p \\
= \sum_r \sum_s \langle \otimes_p \phi(b_{rci}), \otimes_p \phi(b_{scj}) \rangle \\
= \langle \sum_r \otimes_p \phi(b_{rci}), \sum_s \otimes_p \phi(b_{scj}) \rangle
\]

with \( \otimes_p \) the tensor product.

\( p = 1 \) \( \Rightarrow \) Vectors of Locally Aggregated Descriptors\(^1\)
\( p = 2 \) \( \Rightarrow \) Vectors of Locally Aggregated Tensors\(^2^3\)

---

\(^1\) Jegou et al. *Aggregating local image descriptors into compact codes*. In *PAMI*, 2012.
\(^3\) Demonstration software: http://www.vlat.fr
From images to small index

1. Consider matching-based similarity
2. Implicit embedding in Hilbert spaces of matching-based similarity
3. Improve matching accuracy thanks to visual dictionaries
4. Explicit embedding thanks to linearisation
5. Dimensionality reduction
   ⇒ PCA¹, Kernel PCA²
   ⇒ Cluster-wise PCA³, Local coordinate systems⁴
   ⇒ Sparse projectors for dimensionality reduction⁵

---

From images to small index

1. Consider matching-based similarity
2. Implicit embedding in Hilbert spaces of matching-based similarity
3. Improve matching accuracy thanks to visual dictionaries
4. Explicit embedding thanks to linearisation
5. Dimensionality reduction
6. Binary encoding / compression
   \[\Rightarrow\] Product Quantizers\(^1\)

From images to small index

1. Consider matching-based similarity
2. Implicit embedding in Hilbert spaces of matching-based similarity
3. Improve matching accuracy thanks to visual dictionaries
4. Explicit embedding thanks to linearisation
5. Dimensionality reduction
6. Binary encoding / compression

⇒ A final index size up to 1k bytes.
Computational and memory complexity

1. Descriptors extraction in one image
   - Complexity depends on image sizes, descriptor type
   - Existing fast extractors: basic HoG, SURF

2. Computation of one feature
   - Memory: storage of model (dictionary, GMM)
   - CPU: cluster assignment

3. Dimensionality reduction of one feature
   - Memory: storage of projection matrix
   - CPU: Projection

with:
- $N$: size of features (from 4k to 512k)
- $d$: size of reduced features (from 16 to 256)

$O(N) \rightarrow O(16k \text{ to } 128M)$
⇒ Save memory and CPU with sparse projection matrix
Computational and memory complexity

1. Descriptors extraction in one image
   - Complexity depends on image sizes, descriptor type
   - Existing fast extractors: basic HoG, SURF

2. Computation of one feature
   - Memory: storage of model (dictionary, GMM) $O(kD)$
   - Cpu: cluster assignment $O(nkD)$

   with:
   - $D$ size of descriptors ($\simeq 64$)
   - $k$ size of model (from 64 to 4k)
   - $n$ number of descriptors ($\simeq 8k$)
   - $O(kD)$ from 4k to 256k
   - $O(nkD)$ from 32k to 2M

⇒ Save memory and cpu with sparse projection matrix
Computational and memory complexity

1. Descriptors extraction in one image
   - Complexity depends on image sizes, descriptor type
   - Existing fast extractors: basic HoG, SURF

2. Computation of one feature
   - Memory: storage of model (dictionary, GMM) $O(kD)$
   - Cpu: cluster assignment $O(nkD)$

3. Dimensionality reduction of one feature
   - Memory: storage of projection matrix $O(Nd)$
   - Cpu: Projection $O(Nd)$

with:
   - $N$ size of features (from 4k to 512k)
   - $d$ size of reduced features (from 16 to 256)
   - $O(Nd)$ from 64k to 128M

⇒ Save memory and cpu with sparse projection matrix
Problem formulation

- Linear projection:
  \[ y = P^T x \]

with:
- \( x \) feature vector (VLAT, Fisher, ...) of size \( N \)
- \( P \) projection matrix of size \( N \times d \)
- \( y \) reduced feature vector of size \( d \)
Problem formulation

- Linear projection:
  \[ y = P^T x \]

- Problem formulation:
  \[
  P^* = \operatorname{argmin}_P \| X^T X - X^T P P^T X \|_F \\
  \text{s.t. } \forall i, \| p_i \|_0 = M
  \]

- \( X = (x_1 \ldots x_T) \) feature vectors in training set
- \( \| \cdot \|_F \) Frobenius norm, \( \| \cdot \|_0 \) \( \ell_0 \) norm
- \( M \) number of non-zero values in each projector \( p_i \)
Problem formulation

- Linear projection:
  \[ y = P^T x \]

- Problem formulation:
  \[
  P^* = \arg\min_P \| X^T X - X^T P P^T X \|_F
  \text{s.t. } \forall i, \| p_i \|_0 = M
  \]

- \( X^T X \): all similarities between training features
- \( X^T PP^T X \): all similarities between reduced training features
- \( \Rightarrow \) This formulation aims at saving image similarities
Proposed method

Two steps method:\(^1\):

- **Step 1:**
  1. Solve Eq. 1 without sparsity constraint
  2. Find the closest sparse matrix

---

Proposed method

Two steps method\(^1\):

- **Step 1:**
  1. Solve Eq. 1 without sparsity constraint
  2. Find the closest sparse matrix

- **Step 2:** Build a \(d \times d\) correction matrix \(R\) in reduced space that correct similarities:

\[
W^* = \arg \min_W \| G_{\text{dense}} - G_{\text{sparse}} \|_F
\]

with:
- \(G_{\text{dense}}\) similarity matrix of reduced features with dense projectors
- \(G_{\text{sparse}}\) similarity matrix of reduced features with sparse projectors
- \(R^\top R = W\)

---

Proposed method

Two steps method\(^1\):

- **Step 1:**
  1. Solve Eq. 1 without sparsity constraint
  2. Find the closest sparse matrix

- **Step 2:** Build a \(d \times d\) correction matrix \(R\) in reduced space that correct similarities:

\[
W^* = \arg\min_W \| G_{\text{dense}} - G_{\text{sparse}} \|_F
\]  

\(\Rightarrow\) Fast sparse projection matrix computation

---

Near-duplicate retrieval

<table>
<thead>
<tr>
<th>Query image</th>
<th>Similar images</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Query image" /></td>
<td><img src="image2.jpg" alt="Similar images" /></td>
</tr>
<tr>
<td><img src="image3.jpg" alt="Query image" /></td>
<td><img src="image4.jpg" alt="Similar images" /></td>
</tr>
<tr>
<td><img src="image5.jpg" alt="Query image" /></td>
<td><img src="image6.jpg" alt="Similar images" /></td>
</tr>
<tr>
<td><img src="image7.jpg" alt="Query image" /></td>
<td><img src="image8.jpg" alt="Similar images" /></td>
</tr>
<tr>
<td><img src="image9.jpg" alt="Query image" /></td>
<td><img src="image10.jpg" alt="Similar images" /></td>
</tr>
<tr>
<td><img src="image11.jpg" alt="Query image" /></td>
<td><img src="image12.jpg" alt="Similar images" /></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Features</th>
<th>Model Size</th>
<th>Red. Dim.</th>
<th>Sparsity</th>
<th>Holidays +100k</th>
<th>Oxford +100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher</td>
<td>64</td>
<td>128</td>
<td>Dense</td>
<td>77.4</td>
<td>36.6</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>128</td>
<td>Dense</td>
<td>79.9</td>
<td>38.0</td>
</tr>
<tr>
<td>VLAT</td>
<td>64</td>
<td>128</td>
<td>Dense</td>
<td>75.4</td>
<td>42.1</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>128</td>
<td>Dense</td>
<td>78.2</td>
<td>39.0</td>
</tr>
<tr>
<td>Fisher</td>
<td>64</td>
<td>128</td>
<td>90%</td>
<td>77.5</td>
<td>35.9</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>128</td>
<td>90%</td>
<td>80.6</td>
<td>37.8</td>
</tr>
<tr>
<td>VLAT</td>
<td>64</td>
<td>128</td>
<td>99%</td>
<td>75.6</td>
<td>40.9</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>128</td>
<td>99.9%</td>
<td>76.9</td>
<td>35.6</td>
</tr>
</tbody>
</table>

Results in mean Average Precision (mAP %).
Computational complexity

Dashed lines: without correction, Continuous lines: with correction. Colours: sparsity rates (99.9%,99%,90%,0%).

mAP (%) vs Computational cost
Conclusion

Scalable search:
- Reduced features
- Fast similarity (dot product)

Scalable pipeline for feature computation:
- Descriptors extraction
- Feature computation
- Feature reduction & encoding

Scalable models learning?
- Dictionary learning (KMeans, GMM, ...)
- Principal component analysis
Decentralized environment

Training data not hosted on a single machine!
Decentralized environment

Training data not hosted on a single machine!

Multiple connected data sources

- Mobile devices
- Sensor networks

Data often not transferable

Amounts

YouTube

100h/min

Size

Privacy / Rights
Decentralized environment

Training data not hosted on a single machine!

Multiple connected data sources

- Mobile devices
- Sensor networks

Data often not transferable

Amounts: YouTube 100h/min

Size

Privacy / Rights

Training data rather spread over an unreliable network without any central coordination
Case of K-Means algorithm

Objective

- Compute a global codebook $\mathcal{M}$ from local datasets $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}$ distributed on $N$ networking nodes

Constraints

- **Decentralization:** All nodes and links shall play the same role.
- **Consensus:** All nodes must get exactly the same codebook. A given input vector should fall in the same cell whichever the node we use.
- **Asynchrony:** Exchanged messages must be independent $\iff$ a communication round must contain at most one message

Solution

$\Rightarrow$ Gossip protocol
Gossip K-Means
Gossip K-Means
Gossip K-Means

Scalable Learning and Indexing for Large Image Databases
Philippe-Henri Gosselin
October 2, 2014
Gossip K-Means
Gossip K-Means
Gossip K-Means
Gossip K-Means
Gossip K-Means

Results

- Enhancement of Epidemic K-Means
  - No global communication clock
  - Relax initialization constraints
  - Decentralized Codeword-Shifting
  - Convergence proof to a consensus

- Experiments: same retrieval quality for image indexing (Holidays dataset)

---

Gossip PCA

Results\(^4\)

- Learning method in dual space
  - Only projectors need to be shared
  - Minimize communication cost
- Validated on MNIST handwritten digits
- Image experiments: todo!

---

Decentralized learning

Results
- Gossip K-Means
- Gossip PCA

Current work
- Joint learning of dictionary and dimensionality reduction
- Complete state-of-the-art indexing system

Perspectives
- Very large-scale experiments
- Decentralized supervised learning