Efficient visual search of local features

Cordelia Schmid
Visual search

change in viewing angle
Matches

22 correct matches
Image search system for large datasets

Large image dataset (one million images or more)

• **Issues** for very large databases
  • to reduce the query time
  • to reduce the storage requirements
Two strategies

1. Efficient approximate nearest neighbour search on local feature descriptors.

2. Quantize descriptors into a “visual vocabulary” and use efficient techniques from text retrieval. (Bag-of-words representation)
Strategy 1: Efficient approximate NN search

1. Compute local features in each image independently
2. Describe each feature by a descriptor vector
3. Find nearest neighbour vectors between query and database
4. Rank matched images by number of (tentatively) corresponding regions
5. Verify top ranked images based on spatial consistency
Finding nearest neighbour vectors

Establish correspondences between query image and images in the database by nearest neighbour matching on SIFT vectors

Solve following problem for all feature vectors, \( \mathbf{x}_j \in \mathcal{R}^{128} \), in the query image:

\[
\forall j \quad NN(j) = \arg \min_i ||\mathbf{x}_i - \mathbf{x}_j||
\]

where, \( \mathbf{x}_i \in \mathcal{R}^{128} \), are features from all the database images.
Quick look at the complexity of the NN-search

N … images
M … regions per image (~1000)
D … dimension of the descriptor (~128)

Exhaustive linear search: $O(MN^D)$

Example:
- Matching two images (N=1), each having 1000 SIFT descriptors
  Nearest neighbors search: 0.4 s (2 GHz CPU, implementation in C)
- Memory footprint: $1000 \times 128 = 128$kB / image

<table>
<thead>
<tr>
<th># of images</th>
<th>CPU time</th>
<th>Memory req.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = 1,000$</td>
<td>~7min</td>
<td>(~100MB)</td>
</tr>
<tr>
<td>$N = 10,000$</td>
<td>~1h7min</td>
<td>(~ 1GB)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$N = 10^7$</td>
<td>~115 days</td>
<td>(~ 1TB)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>All images on Facebook:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N = 10^{10}$</td>
<td>~300 years</td>
<td>(~ 1PB)</td>
</tr>
</tbody>
</table>

...All images on Facebook:
Nearest-neighbor matching

Solve following problem for all feature vectors, $x_j$, in the query image:

$$
\forall j \quad NN(j) = \arg \min_i ||x_i - x_j||
$$

where $x_i$ are features in database images.

Nearest-neighbour matching is the major computational bottleneck

- Linear search performs $dn$ operations for $n$ features in the database and $d$ dimensions
- No exact methods are faster than linear search for $d > 10$
- Approximate methods can be much faster, but at the cost of missing some correct matches. Failure rate gets worse for large datasets.
Approximate nearest neighbour search

- kd-trees (k dim. tree)
- Binary tree in which each node is a k-dimensional point
- Every split is associated with one dimension
K-d tree

- K-d tree is a binary tree data structure for organizing a set of points.
- Each internal node is associated with an axis aligned hyper-plane splitting its associated points into two sub-trees.
- Dimensions with high variance are chosen first.
- Position of the splitting hyper-plane is chosen as the mean/median of the projected points – balanced tree.
K-d tree construction

Simple 2D example
K-d tree query
Large scale object/scene recognition

- Each image described by approximately 2000 descriptors
  - $2 \times 10^9$ descriptors to index for one million images!

- Database representation in RAM:
  - Size of descriptors: 1 TB, search+memory intractable
Bag-of-features \cite{Sivic&Zisserman'03}

- “visual words”:
  - 1 “word” (index) per local descriptor
  - only images ids in inverted file
  => 8 GB fits!

Harris-Hessian-Laplace regions + SIFT descriptors

\textbf{Set of SIFT descriptors}

\textbf{Bag-of-features processing} + \textit{tf-idf} weighting

\textbf{centroids (visual words)}

\textbf{sparse frequency vector}

\textbf{Inverted file}

\textbf{querying}

\textbf{Re-ranked list}

\textbf{Geometric verification}

\textbf{ranked image short-list}

\cite{Chum & al. 2007}
Indexing text with inverted files

Document collection:

<table>
<thead>
<tr>
<th></th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>common people</td>
<td>sculpture</td>
<td>sculpture</td>
<td>common people</td>
</tr>
<tr>
<td></td>
<td>people</td>
<td></td>
<td>common</td>
<td>people</td>
</tr>
<tr>
<td></td>
<td>common</td>
<td>sculpture</td>
<td></td>
<td>common</td>
</tr>
<tr>
<td></td>
<td>people</td>
<td></td>
<td></td>
<td>people</td>
</tr>
</tbody>
</table>

Inverted file: Term List of hits (occurrences in documents)

- People [d1:hit hit hit], [d4:hit hit] …
- Common [d1:hit hit], [d3: hit], [d4: hit hit hit] …
- Sculpture [d2:hit], [d3: hit hit hit] …

Need to map feature descriptors to “visual words”
Visual words: main idea

Map high-dimensional descriptors to tokens/words by quantizing the feature space

- Quantize via clustering, let cluster centers be the prototype “words”

K. Grauman, B. Leibe
Visual words: main idea

Map high-dimensional descriptors to tokens/words by quantizing the feature space

- Determine which word to assign to each new image region by finding the closest cluster center.

K. Grauman, B. Leibe
Visual words

• Example: each group of patches belongs to the same visual word

Figure from Sivic & Zisserman, ICCV 2003
K-means clustering

• Minimizing sum of squared Euclidean distances between points $x_i$ and their nearest cluster centers

• Algorithm:
  – Randomly initialize K cluster centers
  – Iterate until convergence:
    • Assign each data point to the nearest center
    • Recompute each cluster center as the mean of all points assigned to it

• Local minimum, solution dependent on initialization

• Initialization important, run several times, select best
Inverted file index for images comprised of visual words

- Score each image by the number of common visual words (tentative correspondences)
- Dot product between bag-of-features
- Fast for sparse vectors!
Inverted file index for images comprised of visual words

- Weighting with tf-idf score: weight visual words based on their frequency

  - Tf: normalized term (word) $t_i$ frequency in a document $d_j$

    $$ tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}} $$

  - Idf: inverse document frequency, total number of documents divided by number of documents containing the term $t_i$

    $$ idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|} $$

  - Tf-Idf: $tf - idf = tf_{ij} \cdot idf_i$
Visual words

• Map descriptors to words by quantizing the feature space
  – Quantize via k-means clustering to obtain visual words
  – Assign descriptor to closest visual word

• Bag-of-features as approximate nearest neighbor search

  Bag-of-features matching function

  \[ f_q(x, y) = \delta_{q(x), q(y)} \]

  where \( q(x) \) is a quantizer, i.e., assignment to visual word and \( \delta_{a,b} \) is the Kronecker operator (\( \delta_{a,b}=1 \) iff \( a=b \))
Approximate nearest neighbor search evaluation

- ANN algorithms usually return a short-list of nearest neighbors
  - this short-list is supposed to contain the NN with high probability
  - exact search may be performed to re-order this short-list

- Proposed quality evaluation of ANN search: trade-off between
  - **Accuracy**: NN recall = probability that the NN is in this list
  - **Ambiguity removal** = proportion of vectors in the short-list
    - the lower this proportion, the more information we have about the vector
    - the lower this proportion, the lower the complexity if we perform exact search on the short-list

- ANN search algorithms usually have some parameters to handle this trade-off
ANN evaluation of bag-of-features

• ANN algorithms returns a list of potential neighbors

• **Accuracy**: NN recall
  = probability that the NN is in this list

• **Ambiguity removal**: = proportion of vectors in the short-list

• In BOF, this trade-off is managed by the number of clusters $k$
Vocabulary size

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

- No good trade-off between “small” and “large”!
  - either the Voronoi cells are too big
  - or these cells can’t absorb the descriptor noise
  → intrinsic approximate nearest neighbor search of BOF is not sufficient
20K visual word: false matches
200K visual word: good matches missed
Hamming Embedding \cite{Jegou et al. ECCV’08}

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

Two descriptors $x$ and $y$ match iif

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

where $h(a,b)$ Hamming distance
Hamming Embedding

• Nearest neighbors for Hamming distance ≈ those for Euclidean distance
  → a metric in the embedded space reduces dimensionality curse effects

• Efficiency
  – Hamming distance = very few operations
  – Fewer random memory accesses: 3 x faster than BOF with same dictionary size!
Hamming Embedding

**Off-line** (given a quantizer)
- draw an orthogonal projection matrix $P$ of size $d_b \times d$
  - this defines $d_b$ random projection directions
- for each Voronoi cell and projection direction, compute the median value for a learning set

**On-line**: compute the binary signature $b(x)$ of a given descriptor
- project $x$ onto the projection directions as $z(x) = (z_1, \ldots z_{db})$
- $b_i(x) = 1$ if $z_i(x)$ is above the learned median value, otherwise 0
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 ambiguity removal
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!
Bag-of-features [Sivic&Zisserman’03]

- “visual words”:
  - 1 “word” (index) per local descriptor
  - only images ids in inverted file
=> 8 GB fits!

[Chum & al. 2007]
Geometric verification

Use the position and shape of the underlying features to improve retrieval quality

Both images have many matches – which is correct?
Geometric verification

We can measure **spatial consistency** between the query and each result to improve retrieval quality.

Many spatially consistent matches – **correct result**

Few spatially consistent matches – **incorrect result**
Geometric verification

Gives localization of the object
Geometric verification

- Remove outliers, matches contain a high number of incorrect ones
- Estimate geometric transformation
- Robust strategies
  - RANSAC
  - Hough transform
Example: estimating 2D affine transformation

- Simple fitting procedure (linear least squares)
- Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras
- Can be used to initialize fitting for more complex models

Matches consistent with an affine transformation
Fitting an affine transformation

Assume we know the correspondences, how do we get the transformation?

\[
\begin{pmatrix}
    x'_i \\
    y'_i
\end{pmatrix} = \begin{bmatrix}
    m_1 & m_2 \\
    m_3 & m_4
\end{bmatrix} \begin{pmatrix}
    x_i \\
    y_i
\end{pmatrix} + \begin{pmatrix}
    t_1 \\
    t_2
\end{pmatrix}
\]

\[
\begin{pmatrix}
    m_1 \\
    m_2 \\
    m_3 \\
    m_4
\end{pmatrix} \begin{pmatrix}
    x_i & y_i & 0 & 0 & 1 & 0 \\
    0 & 0 & x_i & y_i & 0 & 1 \\
    \cdots & \cdots
\end{pmatrix} = \begin{pmatrix}
    x'_i \\
    y'_i
\end{pmatrix}
\]
Fitting an affine transformation

\[
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
  m_1 \\
  m_2 \\
  m_3 \\
  m_4 \\
\end{bmatrix}
= 
\begin{bmatrix}
  x' \\
  y' \\
  t_1 \\
  t_2 \\
\end{bmatrix}
\]

Linear system with six unknowns
Each match gives us two linearly independent equations: need at least three to solve for the transformation parameters