Approximate search as a source coding problem

Hervé Jégou
INRIA, Rennes

Seminar@Oxford Visual Geometry Group
September 6th 2012
What is a LARGE database?
What is a LARGE database?

- **Image search**
  - Local descriptors: 1 million images ≈ 2 billion vectors (d=128)
  - Global description: 100 million images (1 vector/image, d=100-1M)
What is a LARGE database?

- **Image search**
  - Local descriptors: 1 million images ≈ 2 billion vectors (d=128)
  - Global description: 100 million images (1 vector/image, d=100-1M)

- **Video search**: hundred/thousand hours of video
  - in TRECVID evaluation tasks
  - Billions of audio/image local descriptors
  - E.g., 3.5 billions SIFTs in INRIA’s last TRECVID submission
What is a LARGE database?

- **Image search**
  - Local descriptors: 1 million images ≈ 2 billion vectors (d=128)
  - Global description: 100 million images (1 vector/image, d=100-1M)

- **Video search**: hundred/thousand hours of video
  - in TRECVID evaluation tasks
  - Billions of audio/image local descriptors
  - E.g., 3.5 billions SIFTs in INRIA’s last TRECVID submission

- **Music retrieval**, e.g., Columbia Million-song database
  - About 1 billion descriptors
What is a LARGE database?

- **Image search**
  - Local descriptors: 1 million images $\approx$ 2 billion vectors ($d=128$)
  - Global description: 100 million images (1 vector/image, $d=100-1M$)

- **Video search**: in TRECVID
  - Billions of audio/image local descriptors
  - E.g., 3.5 billions SIFTs in INRIA’s last TRECVID submission

- **Music retrieval**, e.g., Columbia Million-song database
  - About 1 billion descriptors

In the order of billions of vectors.
Indexing with local descriptors: a typical system

- Local description of images
  - Detect regions of interest [Matas’02, Lowe’04]
  - Compute local descriptors [Lowe’04]

- Search similar vectors in the database
  - Most popular approach is bag-of-words [Sivic’03]
  - \( \approx \) approximation of a voting system

- Spatial verification
  - Re-rank a short-list using geometrical matching [Philbin’07, Perdoc’h 09]
In this talk

- Local description of images
  - Detect regions of interest [Matas’02, Lowe’04]
  - Compute local descriptors [Lowe’04]

- Search similar vectors in the database
  - Most popular approach is bag-of-words [Sivic’03]
  - \( \approx \) approximation of a voting system

- Spatial verification
  - Re-rank a short-list using geometrical matching [Philbin’07, Perdoc’h 09]
The cost of (efficient) exact matching

- Exhaustive linear search: $O(N \times D)$

- 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

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

- Exhaustive linear search: $O(N*D)$

- 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
Yet, still a need for approximate nearest neighbors

- 1 million images, 1000 descriptors per image
  - 1 billion distances per local descriptor
  - $10^{12}$ distances in total
  - Direct matching for one query image: 1.5 hour
• Exact search with sublinear complexity is infeasible in high dimensional spaces.

• 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
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

![Diagram of LSH concept]
Hash functions – Structured vs Learned

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

![Graph showing comparison between different hash functions.](image)

- HKM: loss compared with k-means
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, Slaney 12]

- Excellent package: high integration quality and interface!
- Still high memory requirement for large vector sets


---

**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](https://github.com/PointCloudLibrary/pcl) project page. Thanks to the PCL developers!
- Mac OS X users can install flann through 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 MPI support
- New release introducing new C++ templated API, thread-safe search, save/load of indexes and more.
- The FLANN license was changed from LGPL to BSD.
**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

- **Many variants,** e.g., Spectral Hashing [Weiss 09]
LSH: the two modes – approximate guidelines

Partitioning technique

- **Sublinear/non exhaustive** search
- **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)

Binarization technique

- **Linear** search, Hamming distance
- **Very compact**
  - bit-vectors, concatenated (no ids)
- **Very fast comparison**
  - XOR + popcnt SSE4
  - > 1 billion comparisons/second
- Interesting
  - very high-dimensional vectors
  - But not too many
  - When memory is critical
A bit more on LSH

- Multi-probe LSH [Lv 07] – use less hash functions (possibly 1)
  - LSH requires to store an id per vector and per hash table: at least an id
  - Multi-probe Probes several (closest) cells per hash function
    ⇒ save a lot of memory

- Binary LSH (=searching binary vectors)
  - Like E2LSH but: random subset of bits instead of projections
  - In last CVPR: [Nourouzi 12]. **Exact** binary search variant!

- Optimized jointly with dimensionality reduction
  - ITQ [Gong 11], similar in spirit to PCA+variance balancing [J’ 10]

- Asymmetric distances – proposed along with sketches [Dong’08]
  - Idea: do **not** approximate the query – only the database is binarized
  - Significant improvement for any binarisation technique [Gordo 11]
An hybrid scheme: 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

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

Where $h(., .)$ denotes the Hamming distance
HE provides a much better trade-off between recall and remove false positives.

This scheme is advocated in the BMVC’12 paper:

« Image Retrieval for Image-Based Localization revisited », Sattler et al.

Almost as good as direct matching + Lowe’s distance ratio criterion.
Matching points with k-means quantization indexes

K=20,000 visual words

201 matches

240 matches

Many matches with the non-corresponding image
With a finer partition

K=200,000 visual words

69 matches

35 matches

Still many matches with the non-corresponding one
Matching points - HE

K=20,000 visual words

83 matches

10x more matches with the corresponding image!
Outline

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

- To a code $e(x)$ is associated a unique reconstruction value $q(x)$
  $\Rightarrow$ i.e., the visual word
A typical source coding system

- To a code $e(x)$ is associated a unique reconstruction value $q(x)$
  ⇒ i.e., the visual word
- Focus on quantization (lossy step)
Relationship: 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 quantisation

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

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

- 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 quantiser**

- See, e.g., [Gray’98]
- 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

\[ y_1: 2 \text{ components} \]

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

3 bits \( \Rightarrow \) 24-bit quantization index

- In practice: 8 bits/subquantiser (256 centroids),
  - \( m=4\text{-}512 \), depending on input vectors and desired precision
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 \textbf{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
The estimator $d(X,q(Y))^2$ of $d(X,Y)^2$ is biased:
- The bias can be removed by quantization error terms
- but does not improve the NN search quality (still an open question!)

Estimated distances versus true distances
Combination with an inverted file system

- Coarse k-means hash function
  
  Select k’ closest centroids $c_i$ and corresponding cells

- Compute the residual vector $x - c_i$ of the query vector
  
  Encode the residual vector by PQ

- 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

Searching in 1M GIST descriptors
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
  - Then used to encoding a global image representation (Vlad/Fisher) [J’10]

- [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 (on-the-fly decompression)
  - Won the ILSVRC competition in 2011

- Learning in the PQ-compressed domain: 2 papers at CVPR’12
  - [Vedaldi ’12, Harchaoui’12]
  - In both cases, PQ used for approximate matrix/vector or matrix/matrix multiplication in the compressed domain
Variants/Extensions – 1) Adapted codebook for residual

[Ichida 11] considers the IVFADC variant
- Inverted file + encoding of the residual with product quantization

- **Idea:** Learn a product quantiser separately in each coarse Voronoi region

- **Pros:** Better quantization/search performance

- **Cons:** Longer training with many vectors, many codebooks to store in memory
Variants/Extensions – 2) Adapted codebook for residual

- Re-ranking with source coding [J., Icassp 11]
  - Exploit the explicit reconstruction of PQ
  - Refine the database vector by a short code

\[ \hat{y} = q_c(y) + q_r(r(y)) \]
Other distances/kernels

- Approximate search for more general kernels
  - Kernelized LSH [Kulis 09], RMMH [Joly 11], …
  - These techniques rely on implicit embeddings (and the kernel trick)

- Alternative: explicit embeddings of a kernel K
  - Idea: implicit feature space projected on a finite-dimensional subspace
  - Done with KPCA
  - or approximation [Vedaldi’10, Perronnin’10]
  - Cosine in the embedded space $\approx K$
Searching with explicit embeddings [Bourrier’12]

- L2/Cosine-search in the embedded space ≈ search for kernel K!

![Graph of recall vs rank for Imagenet (1000-D BOW, 1.261 million images)]
Searching with explicit embeddings [Bourrier’12]

- L2/Cosine-search in the embedded space \( \approx \) search for kernel \( K \)

Remark:
KPCA Explicit embedding + LSH > KLSH
Product quantiser vs binary encoding

- Product quantiser: more general formulation
  - Which includes binary encoder

- Yet: binary encoding
  - Easier to optimize
  - Specific processors operations: x3 faster in practice

- Assume that we want to design a binary code such that
  - the reconstruction is as good as possible
  - Hamming distance is the compressed domain is meaningful
  - Estimation of a true Euclidean distance is possible
Anti-sparse coding: spread representations

- Fuchs, “Spread representations”, ASILOMAR’11

- Consider $A = [a_1 | \ldots | a_m]$: $d \times n$ full rank matrix

- The system $Ax = y$ admits an infinite number of solutions

- Add one constraint to single out a unique solution, e.g.:
  - Typical choice: solution of lowest energy
  - Sparse coding: minimize $L^0$ norm of the solution
    → or, in practice, minimal $L^1$ norm

- Anti-sparse coding minimizes instead the $L^\infty$ norm

$$x^* = \min_{x: Ax = y} \| x \|_\infty$$
Properties

Examples of encoded vectors:
- **Simple frame projection** (as in LSH)
- **Anti-sparse coding**: spread representation
Properties

Examples of encoded vectors:

- **Simple frame projection** (as in LSH)
- **Anti-sparse coding**: spread representation

Most components are equal to $-\|\mathbf{x}\|_\infty$ or $+\|\mathbf{x}\|_\infty$

- At least $n-d+1$ components are stuck to the limit
  $\rightarrow$ “natural” binarisation

- The original vector is reconstructed up to a scaling factor
  $\Rightarrow$ **The quantization is error is significantly reduced**
Properties

Examples of encoded vectors:
- **Simple frame projection** (as in LSH)
- **Anti-sparse coding**: spread representation

Assume an input vector is 1) Encoded, 2) Binarized, 3) reconstructed

The error vector is as follows:
Anti-sparse coding for ANN

Three comparison strategies with anti-sparse coding [J. Furon Fuchs, Icassp’12]

- $N_b$: Use binary vectors and Hamming distance
- $N_a$: Asymmetric comparison (similar to [Dong’08], now improved)
- $N_e$: Comparison after explicit reconstruction
Anti-sparse coding for ANN

Three comparison strategies with anti-sparse coding [J. Furon Fuchs, Icassp’12]

- $N_b$: Use binary vectors and Hamming distance
- $N_a$: Asymmetric comparison (similar to [Dong’08], now improved)
- $N_e$: Comparison after explicit reconstruction
Anti-sparse coding for ANN

Three comparison strategies with anti-sparse coding [J. Furon Fuchs, Icassp’12]

- $N_b$: Use binary vectors and Hamming distance
- $N_a$: Asymmetric comparison (similar to [Dong’08], now improved)
- $N_e$: Comparison after explicit reconstruction

A few remarks

- Not as good a PQ on real data
- Encoding cost is a problem for large $n$
- Yet some improvement can still be expected
Conclusion

Nearest neighbor search is a key component of most recognition

Strong relationship between
- Quality of the distance estimation
- Quality of the reconstruction

Product quantization-based approach offers
- Compressed-domain search
  ⇒ Competitive search accuracy
- Compact footprint: few bytes per indexed vector
- A broad interest: any matrix/vector or matrix/matrix operation

DEMO!

10 million images indexed on my laptop:
  21 bytes per indexed image