Recent Advances of Compact Hashing for Large-Scale Visual Search

Shih-Fu Chang
Columbia University
January 2015

Joint work with Sanjiv Kumar (Google), Wei Liu, Jun Wang (IBM Research), Junfeng He (Facebook) Felix Yu, Guangnan Ye, Dong Liu (Columbia U)
Large-Scale Similarity Search

- Need for image search, document retrieval, biology data matching, network construction, ...
- Scale up to billions or trillions of samples

Image, document, DNA sequence ...

Query

Database
The Explosive Data Growth
-- 1.8B photos uploaded/shared per day

Daily Number of Photos Uploaded & Shared on Select Platforms, 2005 – 2014 YTD

How to find photos... not all are tagged!

Visual Search Example

141 Results

Searched over 8,463 billion images in 0.008 seconds.

for file: http://content.sportslogos.net/logos/35/882/full/2671.gif

- These results expire in 72 hours. Why?
- Share a success story!
- TinEye is free to use for non-commercial purposes.
Columbia NewsRover System

linking video over 100+ channels (10 millions images per hour)
Fine-Grained Visual Match

- Image patches (e.g., SIFT, SURF) often used in image search: invariant to geometric and photometric transformations
- Scale grows up to billions or trillions

Slide: David Lowe
Estimate the Complexity

• 500 image patches per photo
  – Feature size ~128 K Bytes
• Database
  – 5 billions patches for 10 million images
  – Finding matched patches becomes challenging
• Also hard to do this over mobile devices ...
1. Take a picture
2. Send image or features
3. Send via mobile networks
4. Visual matching with database images
5. Send results back

Mobile Visual Search
Challenges of MVS

1. Take a picture
2. Image feature extraction
3. Send via mobile networks
4. Visual matching with database images
5. Send results back

Limited power/memory/speed
Limited bandwidth
Fast response (< 1 second)

Peta-scale Database
Needs of Fast Scalable Search

• Fast index code generation
  – Minimize online query processing time

• Avoid $O(N)$ complexity of exhaustive approach
  – Sublinear or constant search complexity over database
  – Efficient storage
Traditional Indexing: K-D Tree

- Popular Public Source VLFeat, FLANN
- Threshold in max variance or random dimension at each node
- Search: best-fit-branch-first, backtrack when needed
- Search time cost: $O(c \cdot \log n)$
- But backtrack prohibitive when dimension is high (Curse of dimensionality)
- Another issue: exponential storage
Product Quantization  

Jegou, Douze, Schmid, PAMI 2011

divide to $m$ subvectors

$\text{feature dimensions (D)}$

$k^{1/m}$ clusters in each subspace

- Avoid exponential codebook size by product of subspace codebooks
- Efficient storage, only $mk^{1/m}$ codewords (3,000, $m=3$, $k=1$ billion)
- Exhaustive scan of codewords possible

$$d(q, w_i) \equiv d(q, w_i^1) + d(q, w_i^2) + d(q, w_i^3)$$

- Drawback: high query processing cost
Hash Table based Search

$H = [h_1, \ldots, h_r]$  

- Projection based hashing is very fast
- $O(1)$ search time by table lookup
- Optional step to include items in Hamming neighborhoods
Locality-Sensitive Hashing

[Indyk and Motwani 1998] [Datar et al. 2004]

hash function

\[ h(x) = \text{sgn}(w^\top x + b) \]

random

\[ P \{ H(x) = H(y) \} = l \cdot \left[ 1 - \frac{\cos^{-1} x^\top y}{\pi} \right]^K \]

• collision probability proportional to original similarity
  \( l \): # hash tables, \( K \): hash bits per table
Multi-Table Hashing

• Longer table increases precision but degrades recall

• Common practice: multi-table hashing

• Union of multi-table results increases precision and keeps recall

• But the number of hash bits 2X: bad for mobile
Beyond Point-to-Point Search

- Diverse Data: feature vectors, graphs, subspace, manifolds, dictionaries, etc.
- Search: range search, graph search, point-to-subspace, manifold to manifold, etc.
Locality-Sensitive Hashing for Subspace

(Wang et al, ICCV ‘13)

Generate K random lines, each producing a hash bit:

\[ h_{l,\theta_0} = \begin{cases} 
0, & \text{dist}_G(l, L) > \theta_0 \\
1, & \text{dist}_G(l, L) \leq \theta_0 
\end{cases} \]

- Preserve locality sensitivity, derive probability of Approximate NN
- Matching by hash table look-up \((O(1)\) complexity)
- Fast index code generation
Subspace Hashing over Face Set Database

- Treat image set of the same person as a subspace
- 60 hash bits per subspace
- Given a single image or image set as query, find the right subspace

Multi-PIE data: 750,000 face images of 337 people under pose-illumination-expression variations
So far, these are random projections...
Focus: Learning-Based Hashing

Unsupervised Hashing
- SH ‘08, KLSH ‘09,
- AGH ’10, PCAH, ITQ ’11,
- DGH ‘14

Semi-Supervised Hashing
- SSH ‘10, WeaklySH ‘10

Supervised Hashing
- RBM ‘09, BRE ‘10,
- MLH ’11, LDAH ’11,
- ITQ ’11, KSH ’12, PHC’13,
- VH’14

Fit data distribution and structure

Explore additional information
A Very Simple Form of Unsupervised Learning

• Find PCA bases as hash projection functions
• Maximize variance (discrimination) in each hash bit

• Rotate projections to minimize quantization errors (Gong&Lazebnik ’11)
Data-dependent vs. Random Hashing

- 580K tiny images

**PCA-ITQ, Gong&Lazebnik, CVPR 11**

(b) Recall precision curve@64 bits.

- PCA-random rotation
- PCA-ITQ optimal alignment
Find Additional Structures in Data

- Images are not random in the feature space
  - Explore (nonlinear) Structures

- Such structures are useful for
  - visualization, retrieval, label propagation
Tool: Sparse Graphs

- Build sparse graphs with **local** connectivity
- Use it to find approximate NN and propagate labels
Active Research: Constructing Sparse Graphs

\[
\text{max}_P \sum_{ij} P_{ij} W_{ij} \\
\text{s.t. } \sum_j P_{ij} = b, P_{ii} = 0, P_{ij} = P_{ji}, \forall i, j \in 1, \ldots, n
\]

(Huang and Jebara, AISTATS 2007)
(Jebara, Wang, and Chang, ICML 2009)
Graph Laplacian

Graph Laplacian \( \Delta = D - W \), and \( L = D^{-1/2} \Delta D^{-1/2} \)

smoothness of function \( f \) over graph

\[
< f, Lf > = f^T Lf = \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \left\| \frac{f(x_i)}{\sqrt{D_{ii}}} - \frac{f(x_j)}{\sqrt{D_{jj}}} \right\|^2
\]
Graph Hashing

\[ < h, Lh > = h^T Lh = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \| h(x_i) - h(x_j) \|^2 \]

- Eigenvectors of Laplacian \( L \) give smooth functions over points
- Can be used to discover structures approximately

Example:
Graph (12K points)

1\(^{st}\) Eigenvector
(sign: blue: +1, red: -1)

2\(^{nd}\) Eigenvector

3\(^{rd}\) Eigenvector

Hash code: [1, 1, 1]

- Locality Sensitivity over the Graph Structure
Challenge: Scale Up to Large Graph

- When graph size is large (million – billion)
  - Hard to construct/store graph \( (kN^2) \)
  - Hard to compute eigenvectors \( (N^3) \)
- Also lacks a fast function to generate index code for novel data
Idea: Low-rank anchor graph

- Use anchor points to “abstract” the graph structure
- Compute data-to-anchor similarity: sparse local embedding
  \[ Z \in \mathbb{R}^{n \times m}, \quad m \ll n \]
- Data-to-data similarity \( W = \) inner product in the embedded space

\[
W_{ij} = \sum_{k=1}^{m} Z_{ik}Z_{jk} = Z_iZ_j^\top \min_k \| x_i - \sum_{k\in\{i\}} Z_{ik}u_k \|_2^2 \quad \text{s.t.} \quad \sum_k Z_{ik} = 1, \quad Z_{ik} \geq 0
\]

(Liu, He, Chang, ICML10)
Probabilistic Intuition

• Affinity between samples i and j, $W_{ij}$ = probability of two-step Markov random walk

$$W = Z \Lambda^{-1} Z^\top,$$ where $\Lambda = \text{diag}(1^\top Z)$.

AnchorGraph: sparse, positive semi-definite
Anchor Graph

\[ W = Z \Lambda^{-1} Z^\top, \text{ where } \Lambda = \text{diag}(1^\top Z). \]

- Affinity matrix \( W \) is sparse, positive semi-definite, and low rank
- Eigenvectors of graph Laplacian can be solved efficiently over low-rank matrix \( O(m^2n) \)
  \[ E = \{e_1, \ldots, e_K\} \in R^{m \times K} \]
- Fast hash function for new data: \( \text{sgn}(Z(x)E) \)
Example of Anchor Graph Hashing

Original Graph (12K points)

• Approximate well the exact eigenvectors of the original graph
YouTube Face Dataset

Wolf et al. CVPR’11. 370K face images from 340 people, 3.8K images from 38 people as queries.

Correct: faces from the same person.

\[
F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

SH = Spectral Hashing
AGH = Anchor Graph Hashing
DGH = Discrete Graph Hashing
Recent work: Direct Graph Hashing

(Liu, Mu, Kumar, Chang, NIPS14)

- Instead of finding eigenvectors and quantize, directly find binary hash codes
- **direct discrete optimization**

\[
\min_H \sum_{i,j=1}^{n} \|H_i - H_j\|^2 W_{ij} \\
\text{s.t. } H \in \{1, -1\}^{n \times r} \quad \text{(balanced)} \\
1^\top H = 0 \\
H^\top H = nI_{r \times r} \quad \text{(orthogonal)} \\
W : \text{graph adjacency matrix}
\]
Direct Graph Hashing Framework

Anchor Graph Laplacian $L = I - A$
relax to a mixed formulation, and learn by alternate maximization

$$\min_B \ tr(B^T LB) + \frac{\rho}{2} \text{dist}^2(B, \Omega)$$

s.t. $B \in \{1, -1\}^{n \times r}$

where

$$\Omega = \{ Y \in \mathbb{R}^{n \times r} \mid \mathbf{1}^\top Y = 0, Y^\top Y = n\mathbf{I}_{r \times r} \},$$
$$\text{dist}(B, \Omega) = \min_{Y \in \Omega} \|B - Y\|_F.$$

- Generate nearly balanced and uncorrelated
  (controlled by the parameter $\rho$) hash codes $B$. 
YouTube Face Dataset

Wolf et al. CVPR’11. 370K face images from 340 people, 3.8K images from 38 people as queries.

Correct: faces from the same person.

\[
F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

SH = Spectral Hashing
AGH = Anchor Graph Hashing
DGH = Discrete Graph Hashing
Preserve Manifold Structure in Hash codes

Irie, Li, Wu, Chang, CVPR ‘14

1. Capture locally linear structures
2. Preserve locally linear structures in Hamming space

\[
\min_{w_i} \lambda \|s_i^T w_i\|_1 + \frac{1}{2} \|x_i - \sum_{j \in \mathcal{N}_E(x_i)} w_{ij} x_j\|^2
\]

\[
s_i = (s_{i1}, \ldots, s_{in})^T, \quad s_{ij} = \frac{\|x_i - x_j\|}{\sum_{j \in \mathcal{N}_E(x_i)} \|x_i - x_j\|}
\]

\[
\min_{y_1, \ldots, y_n \in \mathcal{H}} \sum_i \|y_i - \sum_j w_{ij} y_j\|^2
\]
Experiments: Locally Linear hash

Up to 6-7X performance gain
Over Yale face dataset
Extend Anchor Idea to Kernel Hashing

Kernel:
can be any nonlinear kernel, RBF, Gaussian, etc.

\[ H_l \text{ hash bits} \]
\[
\begin{pmatrix}
1 & -1 & 1 \\
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & 1 & -1
\end{pmatrix}
\]

\[ K_l \text{ kernel matrix} \]

Learn optimal projection over anchors
\[ \begin{pmatrix} a_1 & A & a_r \end{pmatrix} \]

\[ = \text{sgn} \]

\[ m \text{ anchors} \]
Supervised Learning: Explore Additional Side Information

E.g., Binary Supervision

Group M: same class /person

Group C: dissimilar

Binary Relations:

\[ S_{ij} = \begin{cases} 
1 & : (x_i, x_j) \in M \\
-1 & : (x_i, x_j) \in C \\
0 & : otherwise.
\end{cases} \]
Kernel Supervised Hashing

\[ H_l = \begin{bmatrix} H(x_1) \\ \vdots \\ H(x_l) \end{bmatrix} = \begin{bmatrix} h_1(x_1), \ldots, h_r(x_1) \\ \vdots \\ h_1(x_l), \ldots, h_r(x_l) \end{bmatrix} \]

\[ \min_{H_l \in \{1, -1\}^{l \times r}} Q = \| \frac{1}{r} H_l H_l^T - S \|^2_F, \]

S: supervised Information

Liu, et al CVPR'12
1M Tiny Image Dataset

2K query images + 1M database images. 5K (0.5%) pseudo-labeled positives are used for the supervised label matrix $S$.

KLSH (unsupervised) and KSH (supervised) use the same RBF kernel.
Comparison: Hash vs. Tree indexing

Photo Tourism Patch set (103,000 samples) 512-dim SIFT feature
Back to Visual Search Applications

1. Take a picture
2. Send image or features
3. Send via mobile networks
4. Visual matching with database images
5. Send results back
Mobile Visual Search using Bags of Hash Bits (BoHB)

- Fast Hashing
- Compact Code
- Search Large DB

Diagram:
- Extracting Local Feature
- Hashing
- Display
- Extracting Boundaries
- Feature Matching with Hash Bits
- Geometry Verification With Hash Bits
- Search Results
- Boundary Reranking
- Query data

Mobile

Server
Columbia MVS System:
Bags of Hash Bits and Boundary features

Server:
- 400,000 product images crawled from Amazon, eBay and Zappos
- Hundreds of categories; shoes, clothes, electrical devices, groceries, kitchen supplies, movies, etc.

Speed
- Feature extraction: ~1s
- Transmission: 80 bits/feature, 1KB/im
- Server Search: ~0.4s
- Download/display: 1-2s

video demo (50'')
Extension to 3D Model Search

Application: 3D printing, Robotics, etc.
**Object Proposal**
- RGBD Superpixel Merge, SVM Ranker

**3D Database**
- ~3000 CAD models over 10 categories
Other Hashing Forms
Spherical Hashing

• linear projection -> spherical partitioning

\[ h_k(x) = \begin{cases} 
-1 & \text{when } d(p_k, x) > t_k \\
+1 & \text{when } d(p_k, x) \leq t_k 
\end{cases} \]

• Asymmetrical hash bits: tighter regions for +1

• Learning: find optimal spheres (center, radius) in the space
Spherical Hashing Performance

• 1 Million Images: GIST 384-D features
Point-to-Hyperplane Search

Find points closest to the hyperplane

point query

nearest neighbor

normal vector

hyperplane query

$\mathbf{w}$
• Choose the sample closet to the SVM classifier hyperplane and ask human annotator
Hashing Principle: Point-to-Hyperplane Angle

\[ \min D \Rightarrow \min \alpha \]

\[ D(x, P_w) = \frac{|w^T x|}{\|w\|} \]

The ideal neighbors \( \perp w \)
Bilinear Hashing

Bilinear-Hyperplane Hash (BH-Hash)

\[ h^B(z) = \text{sgn}(u^\top zz^\top v), \text{ i.i.d. } u, v \sim \mathcal{N}(0, I_{d\times d}). \]

2 random projection vectors

- **bilinear** hash bit: +1 for || points, -1 for \perp points
**A Single Bit of Bilinear Hash**

\[ h^B(w) = \text{sgn}(u^Tww^Tv) = \text{sgn}(u^Tw) \text{sgn}(v^Tw) = 1 \cdot 1 = 1 \]

\[ h^B(x_1) = \text{sgn}(u^Tx_1) \text{sgn}(v^Tx_1) = 1 \cdot 1 = 1 \]

\[ h^B(x_2) = \text{sgn}(u^Tx_2) \text{sgn}(v^Tx_2) = -1 \cdot 1 = -1 \]
Locality Sensitivity

\[
\Pr [ h^B(w) \neq h^B(x) ] = \frac{1}{2} - \frac{2(\theta_{x,w} - \frac{\pi}{2})^2}{\pi^2} = \frac{1}{2} - \frac{2\alpha_{x,w}^2}{\pi^2}
\]

highest collision probability for active hashing
Active SVM Learning with Hyperplane Hashing

- SVM Learning over 1 million data points
  - actively select the most decisive training point in each iteration

(a) Learning curves

Mean average precision (MAP)

Active learning iteration #
Summary and Open Issues

• Locality Sensitive Compact Hashing
  – Applications in large-scale search, active learning, etc.

• Properties
  – Locality Sensitive
  – Fast code generation
  – Compact size: 20-64 bits per point
  – Efficient search: \(O(1)\) or sublinear cost

• Novel formulations
  – Graph hash, Kernel hash, Hyperplane hash, spherical

• Open Issues
  – Adaptive learning given new data
  – Incorporate high-order relations (e.g., spatio-temporal)
  – Hashing for heterogeneous multimodal features
Selected References

(Hashing with Graphs)

(Iterative Quantization)

(Manifold Hashing)

(Supervised Kernel Hash)

(Hash Based Mobile Product Search)
- J. He, T. Lin, J. Feng, X. Liu, S.-F. Chang, Mobile Product Search with Bag of Hash Bits and Boundary Reranking, CVPR 2012

(Semi-Supervised Hash)

(Circular Hashing)