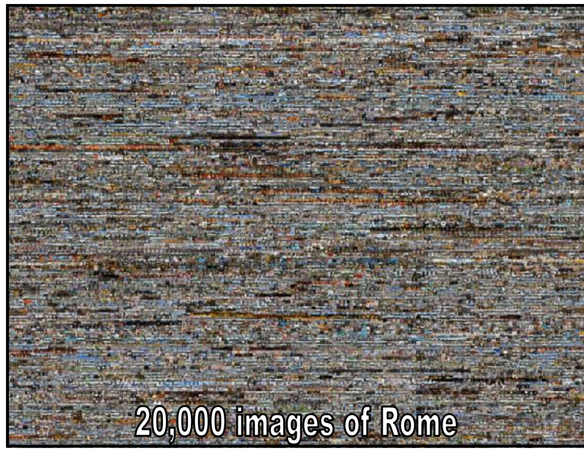
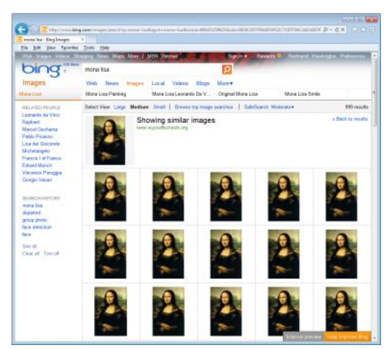
Large-scale matching

CSE P 576

Larry Zitnick (larryz@microsoft.com)



?
=



Large scale matching

How do we match millions or billions of images in under a second?

Is it even possible to store the information necessary?

1 image (640x480 jpg) = 100 kb
1 million images = 100 gigabytes
1 billion images = 100 terabytes
100 billion images = 10,000 terabytes (Flickr has 5 billion)

Interest points

Currently, interest point techniques are the main method for scaling to large databases.

Find interest points

Extract patches

Searching interest points

How do we find similar descriptors across images?

Nearest neighbor search:

Linear search:

1 million images x 1,000 descriptors = 1 billion descriptors (Too slow!)

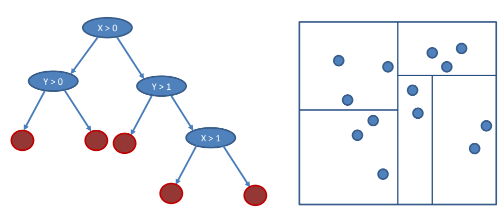
Instead use approximate nearest neighbor:

- KD-tree
- Locality sensitive hashing

KD-tree

Short for “k-dimensional tree.”

Creates a binary tree that splits the data along one dimension:



KD-tree

Algorithm for creating tree:

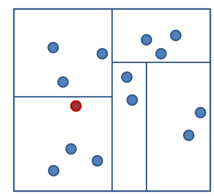
1. Find dimension with highest variance (sometimes cycle through dimensions).
2. Split at median.
3. Recursively repeat steps 1 and 2, until less than "n" data points exist in leaves.

Searching for approximate nearest neighbor:

1. Traverse down the tree until you reach the leaf node.
2. Linearly search for nearest neighbor among all data points in leaf node.

Problem:

The nearest neighbor may not be in the "found" leaf:

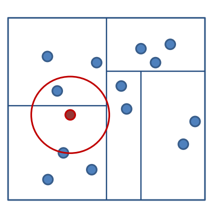


Backtracking (or priority search)

The nearest neighbor may not be in the "found" leaf:

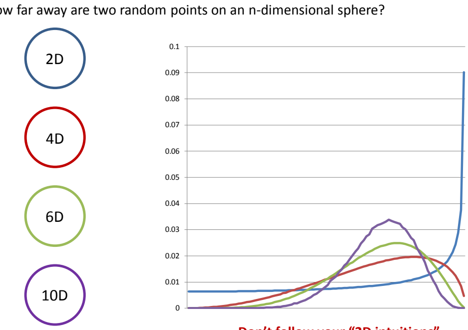
Backtrack to see if any decision boundaries are closer than your current "nearest neighbor."

In high dimensional space, the number of "backtracks" are typically limited to a fixed number. The closest decision boundaries are stored and sorted.



High-dimensional space

How far away are two random points on an n-dimensional sphere?



Don't follow your "2D intuitions"

KD-tree

Other variations:

- Use Principal Component Analysis to align the principal axes of the data with the coordinate axes
- Use multiple randomized KD-trees (by rotating data points)

Optimised KD-trees for fast image descriptor matching
Chanop Silpa-Anan Richard Hartley, CVPR 2008

Storing the descriptors

Storing the descriptors is expensive:

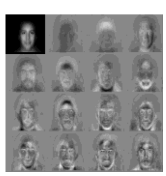
1000 descriptors x 128 dimensions x 1 byte =
128,000 bytes per image

1 million images = 120 gigabytes

PCA

Reduce the dimensionality of the descriptor using PCA.

Pick the "n" orthogonal dimensions with highest variance.



Eigenvectors of face images

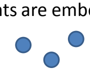
Storing the descriptors is still expensive:

1000 descriptors x 32 dimensions x 1 byte =
32,000 bytes per image

1 million images = 30 gigabytes

Locality sensitive hashing

- Assume points are embedded in Euclidean space



- How to binarize so Hamming distance approximates Euclidean distance?

`Ham_Dist(10001010,11101110)=3`

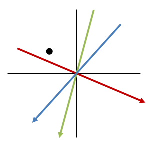
1000 descriptors x 64 dimensions x 1 bit =
8,000 bytes per image

1 million images = 7.5 gigabytes **We're in RAM!**

Finding the binary code

- For each bit:
  - Compute a random unit vector "r".
  - For input vector "v", set the bit equal to:

    h(v) = sign(v•r)



1 0 0

The following holds:

$$Pr[h(u) = h(v)] = 1 - \frac{\theta(u,v)}{\pi}$$

$1 - \frac{\theta(u,v)}{\pi}$ is closely related to cos(θ(*u*,*v*)).

P-stable distributions

- Projects onto an integer instead of bit:
  - Compute a random Gaussian vector "a".
  - For input vector "v", set the index equal to:

$$h_{\mathbf{a},b}(\boldsymbol{v}) = \left\lfloor \frac{\mathbf{a}\cdot\boldsymbol{v}+b}{r} \right\rfloor$$

  - "b" and "r" are chosen by hand.

Datar, M.; Immorlica, N., Indyk, P., Mirrokni, V.S. (2004). "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions". *Proceedings of the Symposium on Computational Geometry*.

Other methods

Spectral hashing (uses thresholded eigenvectors) to find binary codes:

h(v) = sign(cos(kw•v))

"w" is a **principal component**, "k" is chosen based on data.

Spectral Hashing, Yair Weiss, Antonio Torralba, Rob Fergus, NIPS 2008

Locality-sensitive binary codes:

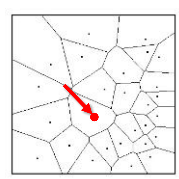
h(v) = sign(cos(kw•v))

"w" is a **randomly sampled** vector, "k" is chosen based on data.

Locality-Sensitive Binary Codes from Shift-Invariant Kernels,
Maxim Raginsky, Svetlana Lazebnik, NIPS 2009

Visual words

What if we just quantize the descriptors to create "visual words."



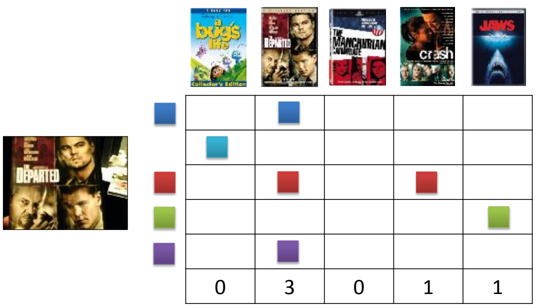
Each descriptor = one integer

1000 descriptors x 32 bits =
4,000 bytes per image

We're in RAM (on my laptop)!

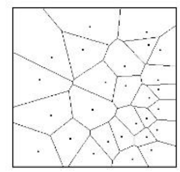
1 million images = 3.7 gigabytes

Inverse lookup table



Creating vocabulary

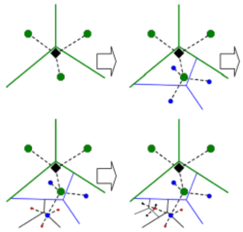
Naïve method is to use k-means clustering on the descriptors.



But this is slow to assign new descriptors to visual words. Need to match the descriptor to every cluster mean = expensive when the vocabulary has 1,000,000 words.

Vocabulary tree

Recursively split descriptor space using k-means, with k ϵ [3,10]



Only need 60 comparisons for k = 10 with 1,000,000 visual words.

Scalable Recognition with a Vocabulary Tree, D. Nister, H. Stewenius, CVPR 2006.

Stop words

If a visual word commonly occurs in many images remove it. You don't want too many images returned for a single word in the inverse look-up table.

It is common practice to have a few thousand stop words for large vocabulary sizes.

It's why search engines don't use the words "a" and "the" in their search queries…

Weighting visual words

Some visual words are more informative than others.

Use TF-IDF weighting. If a visual word occurs frequently in an image but is rare in other images give it a higher weight.

TF (term frequency) $\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$

IDF (inverse document frequency) $\mathrm{idf_i} = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$

TF-IDF $(\text{tf-idf})_{\mathrm{i,j}} = \mathrm{tf_{i,j}} \times \mathrm{idf_i}$

Commonly used in many types of document retrieval.

Reducing # of visual words

What if storing a single integer per visual word is too much?

1000 descriptors x 32 bits =
4,000 bytes per image

1 million images = 3.7 gigabytes

How can we reduce the number of visual words?

100 visual words x 32 bits =
400 bytes per image

1 million images = 380 megabytes **We're in RAM (on smartphone)!**

Randomly removing visual words

$P(v \in I_1, v \in I_2) = 0.5$

Randomly remove 2/3s of visual words:
$P(v \in I_1, v \in I_2) = 0.5*0.33*0.33 = 0.0555$

Image 1	Image 2
2	2
21	32
23	85
67	86
105	35
231	105
321	231
363	321
375	375
578	678
586	743
745	745

Image 1	Image 2
2	85
67	231
321	678
363	743

Not a good idea!

Randomly remove specific visual words

For example: Remove all even visual words.

$P(v \in I_1, v \in I_2) = 0.5$

Some images may not have any visual words remaining:

Image 1	Image 2
2	2
22	32
24	85
63	86
104	35
232	105
325	231
366	321
378	378
578	678
586	743
745	745

Image 1	Image 2
63	85
325	35
745	105
	231
	321
	743
	745

Min-hash

Maintain Jaccard similarity while keeping a constant number of visual words per image.

Jaccard similarity: $$\text{sim}_J(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$

C_1	C_2
0	1
1	0
1	1
0	0
1	1
0	1

$\text{sim}_J(C_1,C_2) = 2/5 = 0.4$

*Minhash slides based on material from Rajeev Motwani and Jeff Ullman

Key Observation

- For columns C_i, C_j, four types of rows

	C_i	C_j
A	1	1
B	1	0
C	0	1
D	0	0

- Overload notation: A = # of rows of type A
- **Claim** $$\text{sim}_J(C_i, C_j) = \frac{A}{A+B+C}$$

Min Hashing

- Randomly permute rows
- Hash $h(C_i)$ = index of first row with 1 in column C_i
- **Suprising Property**
- Why? $P[h(C_i) = h(C_j)] = \text{sim}_J(C_i, C_j)$
  - Both are A/(A+B+C)
  - Look down columns C_i, C_j until first non-Type-D row
  - $h(C_i) = h(C_j)$ <-> type A row

Min-Hash Signatures

- Pick – P random row permutations
- MinHash Signature

  sig(C) = list of P indexes of first rows with 1 in column C

- Similarity of signatures
  - Let $\text{sim}_H(\text{sig}(C_i),\text{sig}(C_j))$ = fraction of permutations where MinHash values agree
  - Observe $E[\text{sim}_H(\text{sig}(C_i),\text{sig}(C_j))] = \text{sim}_J(C_i,C_j)$

Sketches

What if hashes aren't unique enough? I.e., we return too many possible matches per hash in an inverse look-up table?

Concatenate the hashes into "sketches" of size "k".

$h_1 = 23$, $h_2 = 243$, $h_3 = 598$ ⇒ $s_1 = 598{,}243{,}023$

The probability of two sketches colliding is:

$$\mathrm{sim}_J(C_i, C_j)^k$$

Typically you have to balance the precision/recall tradeoffs when picking the sketch size and number of sketches.

Overview

1 million images	100 GB
1 million images (descriptors)	120 GB
1 million images (descriptors PCA)	30 GB
1 million images (binary descriptors)	7.5 GB
1 million images (visual words)	3.7 GB
1 million images (hashed visual words)	380 MB
10 billion images (hashed visual words)	3.6 TB