Large scale clustering
and nearest neighbor search

Yannis Avrithis

Inria Rennes-Bretagne Atlantique

Prague, March 2016
Problem

ANN search

• Given query point $q$, find its nearest neighbor with respect to Euclidean distance within data set $X$ in a $d$-dimensional space
• Encode (compress) vectors, speed up distance computations
• Fit underlying distribution with little space & time overhead

Vector quantization

• Given data set $X'$, map it to discrete codebook $C$ such that distortion is minimized
• Use ANN search to assign points to centroids
• Use vector quantization to improve ANN search
Problem

ANN search

- Given query point $q$, find its nearest neighbor with respect to Euclidean distance within data set $\mathcal{X}$ in a $d$-dimensional space
- Encode (compress) vectors, speed up distance computations
- Fit underlying distribution with little space & time overhead

Vector quantization

- Given data set $\mathcal{X}$, map it to discrete codebook $C$ such that distortion is minimized
- Use ANN search to assign points to centroids
- Use vector quantization to improve ANN search
### Applications in vision

Retrieval (image as point) [Jégou et al. ’10][Perronnin et al. ’10]

![Figure 1. Images and corresponding VLAD descriptors, for k=16 centroids (D =16×128). The components of the descriptor are represented like SIFT, with negative components (see Equation 1) in red.](image)

- Words $k$: we consider values ranging from $k=16$ to $k=256$.
- Figure 1 depicts the VLAD representations associated with a few images, when aggregating $128$-dimensional SIFT descriptors. The components of our descriptor map to components of SIFT descriptors. Therefore we adopt the usual $4 \times 4$ spatial grid representation of oriented gradients for each $v_i=1..k$.
- We have accumulated the descriptors in 16 of them, one per visual word. In contrast to SIFT descriptors, a component may be positive or negative, due to the difference in Equation 1.

- One can observe that the descriptors are relatively sparse (few values have a significant energy) and very structured: most high descriptor values are located in the same cluster, and the geometrical structure of SIFT descriptors is observable. Intuitively and as shown later, a principal component analysis is likely to capture this structure. For sufficiently similar images, the closeness of the descriptors is obvious.

### 3. From vectors to codes

This section addresses the problem of coding an image vector. Given a $D$-dimensional input vector, we want to produce a code of $B$ bits encoding the image representation, such that the nearest neighbors of a (non-encoded) query vector can be efficiently searched in a set of $n$ encoded database vectors.

We handle this problem in two steps, that must be optimized jointly: 1) a projection that reduces the dimensionality of the vector and 2) a quantization used to index the resulting vectors. For this purpose, we consider the recent approximate nearest neighbor search method of [7], which is briefly described in the next section. We will show the importance of the joint optimization by measuring the mean squared Euclidean error generated by each step.

#### 3.1. Approximate nearest neighbor

Approximate nearest neighbors search methods [4, 11, 15, 24, 27] are required to handle large databases in computer vision applications [22]. One of the most popular techniques is Euclidean Locality-Sensitive Hashing [4], which has been extended in [11] to arbitrary metrics. However, these approaches and the one of [15] are memory consuming, as several hash tables or trees are required. The method of [27], which embeds the vector into a binary space, better satisfies the memory constraint. It is, however, significantly outperformed in terms of the trade-off between memory and accuracy by the product quantization-based approximate search method of [7]. In the following, we use this method, as it offers better accuracy and because the search algorithm provides an explicit approximation of the indexed vectors. This allows us to compare the vector approximations introduced by the dimensionality reduction and the quantization. We use the asymmetric distance computation (ADC) variant of this approach, which only encodes the vectors of the database, but not the query vector.

This method is summarized in the following.
Abstract

A wide range of properties and assumptions determine the most appropriate spatial matching model for an application, e.g., recognition, detection, registration, or large scale image retrieval. Most notably, these include discriminative power, geometric invariance, rigidity constraints, mapping constraints, assumptions made on the underlying features or descriptors and, of course, computational complexity. Having image retrieval in mind, we present a very simple model inspired by Hough voting in the transformation space, where votes arise from single feature correspondences. A relaxed matching process allows for multiple matching surfaces or non-rigid objects under one-to-one mapping, yet is linear in the number of correspondences. We apply it to geometry re-ranking in a search engine, yielding superior performance with the same space requirements but a dramatic speed-up compared to the state of the art.

1. Introduction

Discriminative local features have made sub-linear indexing of appearance possible, but geometry indexing still appears elusive if one targets invariance, global geometry verification, high precision and low space requirements. Large scale image retrieval solutions typically consider geometry in a second, re-ranking phase. The latter is linear in the number of images to match, hence its speed is crucial. Exploiting local shape of features (e.g., local scale, orientation, or affine parameters) to extrapolate relative transformations, it is either possible to construct RANSAC hypotheses by single correspondences [14], or to see correspondences as Hough votes in a transformation space [12]. In the former case one still has to count inliers, so the process is quadratic in the number of (tentative) correspondences. In the latter, voting is linear but further verification with inlier count seems unavoidable.

Flexible spatial models are more typical in recognition; these are either not invariant to geometric transformations, or use pairwise constraints to detect inliers without any rigid motion model [11]. The latter are at least quadratic in the number of correspondences and their practical running time is still prohibitive if our target for re-ranking is thousands of matches per second.

We develop a relaxed spatial matching model which applies the concept of pyramid match [8] to the transformation space. Using local feature shape to generate votes, it is invariant to similarity transformations, free of inlier-count verification and linear in the number of correspondences. It imposes one-to-one mapping and is flexible, allowing non-rigid motion and multiple matching surfaces or objects. Fig. 1 compares our Hough pyramid matching (HPM) to fast spatial matching (FSM) [14]. Both buildings are matched by HPM, while inliers from one surface are only found by FSM. But our major achievement is speed: in a given query time, HPM can re-rank one order of magnitude more images than the state of the art in geometry re-ranking. We give a more detailed account of our contribution in section 2 after discussing the most related prior work.
Applications in vision

Localization, pose estimation [Sattler et al. '12][Li et al. '12]
Applications in vision

Classification [Boiman et al. '08][McCann & Lowe '12]

query image $Q$

$KL(p_Q | p_c) = 8.35$

$KL(p_Q | p_1) = 17.54$

$KL(p_Q | p_2) = 18.20$

$KL(p_Q | p_3) = 14.56$
Applications in vision

BoW (patch quantization) [Sivic et al. '03][Philbin et al. '07]
Applications in vision

BoW (codebook construction) [Philbin et al. '07][Avrithis '12]

iteration=3, clusters=8
Applications in vision

Image clustering [Gong et al. ’15][Avrithis ’15]

Web Scale Photo Hash Clustering on A Single Machine
Yunchao Gong, Marcin Pawlowski, Fei Yang, Louis Brandy, Lubomir Boundev, Rob Fergus

Abstract
This paper addresses the problem of clustering a very large number of photos (i.e. hundreds of millions a day) in a stream into millions of clusters. This is particularly important as the popularity of photo sharing websites, such as Facebook, Google, and Instagram. Given large number of photos available online, how to efficiently organize them is an open problem.

To address this problem, we propose to cluster the binary hash codes of a large number of photos into binary cluster centers. We present a fast binary k-means algorithm that works directly on the similarity-preserving hashes of images and clusters them into binary centers on which we can build hash indexes to speedup computation. The proposed method is capable of clustering millions of photos on a single machine in a few minutes. We show that this approach is usually several magnitude faster than standard k-means and produces comparable clustering accuracy. In addition, we propose an online clustering method based on binary k-means that is capable of clustering large photo stream on a single machine, and show applications to spam detection and trending photo discovery.

1. Introduction
Photo sharing websites are becoming extremely popular, hundreds of millions of photos are uploaded every day. For example, Facebook announced it has about 300 million photo uploads every day. However, how to efficiently organize such huge online photo collections is becoming a challenge. In this paper, we propose to study the problem of clustering large photo collections at the scale of hundreds millions a day, This process has many practical applications. For example, clustering large photo collections into near-duplicate image clusters can help find spam photos. Online clustering photos into semantic clusters can be used to find time-sensitive photo clusters and trending events. For these scenarios, we need online clustering methods which can handle hundreds of millions photos a day and can store a very large number of centers in memory.

Image clustering is a well-studied problem in the literature [17, 24, 10, 37, 43, 35]. However, how to efficiently cluster such huge collections of photos on a single machine has received little attention. This problem is challenging because 1) it is hard to compactly represent such huge photo collections; 2) it is computationally very inefficient to perform clustering on large datasets; and 3) it is very inefficient to store and index increasing large number of cluster centers. The first problem has been addressed by recent works on similarity preserving hashing [44, 12, 30], that try to represent images as compact hash codes. For the second challenge, there is work using kd-tree [32] to speed up the clustering process, but it does not address the third challenge, as kd-tree needs to store all the real valued centers in memory. Photo clustering will become infeasible when the number of clusters accumulates to tens of millions or even more.

In this paper, we try to address three challenges by developing a method that clusters image similarity binary codes into a set of compact binary centers, which can be easily indexed. The basic idea is illustrated in Figure 1. We first represent the photos using similarity preserving binary codes [44, 12, 30], enabling us to store large number of photos in memory. Then we propose a variant of the classic k-means algorithm denoted as Binary k-means (Bk-means) that constrains the centers to be binary. The centers also live on the Hamming cube. This enables us to easily use a multi-index.

Figure 1. The problem setting of this paper. We are interested in clustering a large amount of image hash codes into compact binary centers.
Overview (1)

Binary codes

- spectral hashing [Weiss et al. '08]
- iterative quantization [Gong & Lazebnik '11]

Quantization

- vector quantization (VQ) [Gray '84]
- product quantization (PQ) [Jégou et al. '11]
- optimized product quantization (OPQ) [Ge et al. '13]
  Cartesian $k$-means [Norouzi & Fleet '13]
- locally optimized product quantization (LOPQ) [Kalantidis & Avrithis '14]
Overview (1)

**Binary codes**
- spectral hashing [Weiss et al. ’08]
- iterative quantization [Gong & Lazebnik ’11]

**Quantization**
- vector quantization (VQ) [Gray ’84]
- product quantization (PQ) [Jégou et al. ’11]
- optimized product quantization (OPQ) [Ge et al. ’13]
  Cartesian $k$-means [Norouzi & Fleet ’13]
- locally optimized product quantization (LOPQ) [Kalantidis & Avrithis ’14]
Overview (2)

Non-exhaustive search

- non-exhaustive PQ [Jégou et al. '11]
- inverted multi-index [Babenko & Lempitsky '12]
- multi-LOPQ [Kalantidis & Avrithis '14]

Clustering

- hierarchical \(k\)-means [Nister & Stewenius '06]
- approximate \(k\)-means [Philbin et al. '07]
- approximate Gaussian mixtures [Kalantidis & Avrithis '12]
- dimensionality-recursive vector quantization [Avrithis '13]
- ranked retrieval [Broder et al. '14]
- inverted-quantized \(k\)-means [Avrithis et al. '15]
Overview (2)

Non-exhaustive search

• non-exhaustive PQ [Jégou et al. '11]
• inverted multi-index [Babenko & Lempitsky '12]
• multi-LOPQ [Kalantidis & Avrithis '14]

Clustering

• hierarchical $k$-means [Nister & Stewenius '06]
• approximate $k$-means [Philbin et al. '07]
• approximate Gaussian mixtures [Kalantidis & Avrithis '12]
• dimensionality-recursive vector quantization [Avrithis '13]
• ranked retrieval [Broder et al. '14]
• inverted-quantized $k$-means [Avrithis et al. '15]
Binary codes
Spectral hashing

[Weiss et al. ’08]

- Given a set of \( n \) data points \( x_i \in \mathbb{R}^d \), encode each by binary code \( y_i \)
- Define similarity matrix \( S \) with \( S_{ij} = \exp(-\|x_i - x_j\|^2/t^2) \)
- Require binary codes to be similarity preserving, balanced, and uncorrelated:

\[
\begin{align*}
\text{minimize} & \quad \sum_{ij} S_{ij} \|y_i - y_j\|^2 \\
\text{subject to} & \quad y_i \in \{-1, 1\}^k \\
& \quad \sum_i y_i = 0 \\
& \quad \frac{1}{n} \sum_i y_i y_i^\top = I.
\end{align*}
\]
Spectral hashing
[Weiss et al. ’08]

- Given a set of \( n \) data points \( x_i \in \mathbb{R}^d \), encode each by binary code \( y_i \)
- Define similarity matrix \( S \) with \( S_{ij} = \exp(-\|x_i - x_j\|^2 / t^2) \)
- Require binary codes to be similarity preserving, balanced, and uncorrelated:

\[
\begin{align*}
\text{minimize} & \quad \sum_{ij} S_{ij} \|y_i - y_j\|^2 \\
\text{subject to} & \quad y_i \in \{-1, 1\}^k \\
& \quad \sum_i y_i = 0 \\
& \quad \frac{1}{n} \sum_i y_i y_i^\top = I.
\end{align*}
\]
Spectral hashing

Example

- **Red**: outer-product eigenfunctions: excluded
- Better to cut long dimension first
- Lower spatial frequencies are better than higher ones
Spectral hashing

Example

- **Red**: outer-product eigenfunctions: excluded
- Better to cut long dimension first
- Lower spatial frequencies are better than higher ones

<table>
<thead>
<tr>
<th>LSH</th>
<th>Boosting SSC</th>
<th>LSH</th>
<th>Boosting SSC</th>
<th>LSH</th>
<th>Boosting SSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBM (two hidden layers)</td>
<td>Spectral hashing</td>
<td>RBM (two hidden layers)</td>
<td>Spectral hashing</td>
<td>RBM (two hidden layers)</td>
<td>Spectral hashing</td>
</tr>
</tbody>
</table>

- **Red**: radius = 0; **green**: radius = 1; **blue**: radius = 2
Iterative quantization
[Gong and Lazebnik ’11]

Quantize each data point to the closest vertex of the binary cube, 
$(±1, ±1)$.

(a) PCA aligned. (b) Random Rotation. (c) Optimized Rotation.
Vector quantization
Vector quantization

[Gray '84]

minimize $E(C) = \sum_{x \in X} \min_{c \in C} \|x - c\|^2 = \sum_{x \in X} \|x - q(x)\|^2$

distortion dataset codebook quantizer
Vector quantization

[Gray '84]

minimize \( E(C) = \sum_{x \in \mathcal{X}} \min_{c \in C} \| x - c \|^2 = \sum_{x \in \mathcal{X}} \| x - q(x) \|^2 \)

distortion  dataset  codebook  quantizer
Vector quantization

[Gray ’84]

- For small distortion $\rightarrow$ large $k = |\mathcal{C}|$:
  - hard to train
  - too large to store
  - too slow to search
Product quantization

[Jégou et al. ’11]

\[\text{minimize} \quad \sum_{x \in \mathcal{X}} \min_{c \in C} \|x - c\|^2\]

subject to \(C = C^1 \times \cdots \times C^m\)
Product quantization

[Jégou et al. ’11]

- **train:** $q = (q^1, \ldots, q^m)$ where $q^1, \ldots, q^m$ obtained by VQ
- **store:** $|\mathcal{C}| = k^m$ with $|\mathcal{C}^1| = \cdots = |\mathcal{C}^m| = k$
- **search:** $\|\mathbf{y} - q(\mathbf{x})\|^2 = \sum_{j=1}^{m} \|\mathbf{y}^j - q^j(\mathbf{x}^j)\|^2$ where $q^j(\mathbf{x}^j) \in \mathcal{C}^j$
Optimized product quantization
[Ge et al. ’13]

\[
\text{minimize} \quad \sum_{x \in X} \min_{\hat{c} \in \hat{C}} \|x - R^\top \hat{c}\|^2
\]
\[
\text{subject to} \quad \hat{C} = C^1 \times \cdots \times C^m
\]
\[
R^\top R = I
\]
Optimized product quantization

Parametric solution for $x \sim \mathcal{N}(0, \Sigma)$

- **independence**: PCA-align by diagonalizing $\Sigma$ as $U \Lambda U^\top$

- **balanced variance**: permute $\Lambda$ by $\pi$ such that $\prod_i \lambda_i$ is constant in each subspace; $R \leftarrow U P_{\pi}^\top$

- find $\hat{C}$ by PQ on rotated data $\hat{X} = RX$
Locally optimized product quantization

[Kalantidis & Avrithis '14]

- compute residuals \( r(x) = x - Q(x) \) on coarse quantizer \( Q \)
- collect residuals \( Z_i = \{ r(x) : Q(x) = c_i \} \) per cell
- train \((R_i, q_i) \leftarrow \text{OPQ}(Z_i)\) per cell
Locally optimized product quantization

[Kalantidis & Avrithis ’14]

- residual distributions closer to Gaussian assumption
- better captures the support of data distribution, like local PCA
  - multimodal (e.g. mixture) distributions
  - distributions on nonlinear manifolds
Local principal component analysis

[Kambhatla & Leen '97]

But, we are not doing dimensionality reduction!
Non-exhaustive search
Inverted multi-index

[Babenko & Lempitsky '12]

- train codebook $C$ from dataset $\{x_n\}$, defining a coarse quantizer $Q$
- quantize each point $x$ to $Q(x)$ and encode its residual $r(x) = x - Q(x)$ by product quantizer $q$
- given query $y$, visit $w$ coarse cells closest to $y$
• decompose vectors as $x = (x^1, x^2)$
• train codebooks $C^1, C^2$ from datasets $\{x^1_n\}, \{x^2_n\}$
• induced codebook $C^1 \times C^2$ gives a finer partition
• given query $y$, visit cells $(c^1, c^2) \in C^1 \times C^2$ in ascending order of distance to $y$
The overview of the query process within the inverted multi-index. First, the two halves of the query \( q^1 \) and \( q^2 \) are determined.

\[
\begin{array}{c|c|c}
 i & u_{\alpha(i)} & r \\
\hline
 1 & u_3 & 0.5 \\
 2 & u_4 & 0.7 \\
 3 & u_5 & 4 \\
 4 & u_2 & 6 \\
 5 & u_1 & 8 \\
 6 & u_6 & 9 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
 j & v_{\beta(j)} & s \\
\hline
 1 & v_4 & 0.1 \\
 2 & v_3 & 2 \\
 3 & v_5 & 3 \\
 4 & v_2 & 6 \\
 5 & v_6 & 7 \\
 6 & v_1 & 11 \\
\end{array}
\]

The multi-sequence algorithm produces a sequence of \( \alpha \) pairs \((i, j)\) that correspond to the space subdivision into cells, while black symbols give original exact distances. The queue is initialized with a predefined length. The queue is reinitialized after each \( \alpha \) iteration. The lists associated with those pairs are concatenated to produce the answer to the query. One can prove the correctness of the algorithm:

\[
[u_{\alpha(i)} \ v_{\beta(j)}] = (i, j) \quad r(i) + s(j)
\]

The multi-sequence algorithm are given in the supplementary material. The relative efficiency of the two indexing structures, standard inverted index and for the inverted multi-index (Fig. 1), is also discussed.

Output:

- \((1, 1)\) → \(W_{3,4}\)
- \((2, 1)\) → \(W_{4,4}\)
- \((1, 2)\) → \(W_{3,3}\)
- \((2, 2)\) → \(W_{4,3}\)
- \((1, 3)\) → \(W_{3,5}\)
Inverted multi-index

Result on SIFT1B: are NN in candidate lists?

![Graph showing recall as a function of list length for different index configurations.](image)

- **Multi-index K=2^{14}**
- **Index+kd-tree K=2^{14}**
- **Index K=2^{14}**
- **Multi-index K=2^{12}**
- **Index+kd-tree K=2^{12}**
- **Index K=2^{12}**
Multi-LOPQ

[Kalantidis & Avrithis '14]
## Multi-LOPQ

Result on SIFT1B, 128-bit codes

<table>
<thead>
<tr>
<th>$T$</th>
<th>Method</th>
<th>$R = 1$</th>
<th>10</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>20K</td>
<td>IVFADC+R [Jégou et al. ’11]</td>
<td>0.262</td>
<td>0.701</td>
<td>0.962</td>
</tr>
<tr>
<td></td>
<td>LOPQ+R [Kalantidis &amp; Avrithis ’14]</td>
<td>0.350</td>
<td>0.820</td>
<td>0.978</td>
</tr>
<tr>
<td></td>
<td>Multi-D-ADC [Babenko &amp; Lempitsky ’12]</td>
<td>0.304</td>
<td>0.665</td>
<td>0.740</td>
</tr>
<tr>
<td></td>
<td>OMulti-D-OADC [Ge et al. ’13]</td>
<td>0.345</td>
<td>0.725</td>
<td>0.794</td>
</tr>
<tr>
<td></td>
<td>Multi-LOPQ [Kalantidis &amp; Avrithis ’14]</td>
<td>0.430</td>
<td>0.761</td>
<td>0.782</td>
</tr>
<tr>
<td>10K</td>
<td>Multi-D-ADC [Babenko &amp; Lempitsky ’12]</td>
<td>0.328</td>
<td>0.757</td>
<td>0.885</td>
</tr>
<tr>
<td></td>
<td>OMulti-D-OADC [Ge et al. ’13]</td>
<td>0.366</td>
<td>0.807</td>
<td>0.913</td>
</tr>
<tr>
<td></td>
<td>Multi-LOPQ [Kalantidis &amp; Avrithis ’14]</td>
<td>0.463</td>
<td>0.865</td>
<td>0.905</td>
</tr>
<tr>
<td>30K</td>
<td>Multi-D-ADC [Babenko &amp; Lempitsky ’12]</td>
<td>0.334</td>
<td>0.793</td>
<td>0.959</td>
</tr>
<tr>
<td></td>
<td>OMulti-D-OADC [Ge et al. ’13]</td>
<td>0.373</td>
<td>0.841</td>
<td>0.973</td>
</tr>
<tr>
<td></td>
<td>Multi-LOPQ [Kalantidis &amp; Avrithis ’14]</td>
<td>0.476</td>
<td>0.919</td>
<td>0.973</td>
</tr>
</tbody>
</table>
Application: image search
Deep learned image features

[Krizhevsky et al. ’12] [Babenko et al. ’14]

2 A. Babenko, A. Slesarev, A. Chigorin, V. Lempitsky
trained to recognize Image-Net [1] classes. We measure such performance on four standard benchmark datasets: INRIA Holidays [8], Oxford Buildings, Oxford Building 105K [19], and the University of Kentucky benchmark (UKB) [16].

Perhaps unsurprisingly, these deep features perform well, although not better than other state-of-the-art holistic features (e.g. Fisher vectors). Interestingly, the relative performance of different layers of the CNN varies in different retrieval setups, and the best performance on the standard retrieval datasets is achieved by the features in the middle of the fully-connected layers hierarchy.

Fig. 1. The convolutional neural network architecture used on our experiments. Purple nodes correspond to input (an RGB image of size $224 \times 224$) and output (1000 class labels). Green units correspond to outputs of convolutions, red units correspond to the outputs of max pooling, and blue units correspond to the outputs of rectified linear (ReLU) transform. Layers 6, 7, and 8 (the output) are fully connected to the preceding layers. The units that correspond to the neural codes used in our experiments are shown with red arrows. Stride=4 are used in the first convolutional layer, and stride=1 in the rest.

The good performance of neural codes demonstrate their universality, since the task the network was trained for (i.e. classifying Image-Net classes) is quite different from the retrieval task we consider. Despite the evidence of such universality, there is an obvious possibility to improve the performance of deep features by adapting them to the task, and such adaptation is the subject of the second part of the paper. Towards this end, we assemble a large-scale image dataset, where the classes correspond to landmarks (similar to [14]), and retrain the CNN on this collection using the original image-net network parameters as initialization. After such training, we observe a considerable improvement of the retrieval performance on the datasets with similar image statistics, such as INRIA Holidays and Oxford Buildings, while the performance on the unrelated UKB dataset degrades. In the second experiment of this kind, we retrain the initial network on the Multi-view RGB-D dataset [12] of turntable views of different objects. As expected, we observe the improvement on the more related UKB dataset, while the performance on other datasets degrades or stays the same.

Finally, we focus our evaluation on the performance of the compact versions of the neural codes. We evaluate the performance of the PCA compression
Image search on CNN activations

[Razavian '14, Babenko '15, Kalantidis '15, Tolias '16]
Multi-LOPQ on CNN activations
Image query on Flickr 100M (4k → 128 dimensions)
Clustering
Hierarchical $k$-means

[Nister & Stewenius '06]

---

**Abstract**

A recognition scheme that scales efficiently to a large number of objects is presented. The efficiency and quality is exhibited in a live demonstration that recognizes CD-covers from a database of 40000 images of popular music CD's. The scheme builds upon popular techniques of indexing descriptors extracted from local regions, and is robust to background clutter and occlusion. The local region descriptors are hierarchically quantized in a vocabulary tree. The vocabulary tree allows a larger and more discriminatory vocabulary to be used efficiently, which we show experimentally leads to a dramatic improvement in retrieval quality. The most significant property of the scheme is that the tree directly defines the quantization. The quantization and the indexing are therefore fully integrated, essentially being one and the same.

The recognition quality is evaluated through retrieval on a database with ground truth, showing the power of the vocabulary tree approach, going as high as 1 million images.

---

**1. Introduction**

Object recognition is one of the core problems in computer vision, and it is a very extensively investigated topic. Due to appearance variabilities caused for example by non-rigidity, background clutter, differences in viewpoint, orientation, scale or lighting conditions, it is a hard problem.

One of the important challenges is to construct methods that scale well with the size of the database, and can select one out of a large number of objects in acceptable time. In this paper, a method handling a large number of objects is presented. The approach belongs to a currently very popular class of algorithms that work with local image regions and...
Approximate $k$-means
[Philbin et al. '07][Gong et al. '15]

- centroids updated as in $k$-means
- points assigned to centroids by approximate search
- index rebuilt in every $k$-means iteration
Ranked retrieval
[Broder et al. ’14]

- points assigned by inverse search from centroids to points
- needs conflict resolution; points may remain unassigned
- index built only once; resembles mean shift [Cheng et al. ’95]
Dimensionality-recursive vector quantization

[Avrithis ’13]

- points quantized as in multi-index
- cells assigned exhaustively by distance map from centroids
- points assigned by lookup
Approximate Gaussian mixtures

[Kalantidis & Avrithis ’12]

- centroids & variances updated as in EM
- points soft-assigned by approximate search
- $k$ dynamically estimated
Inverted-quantized $k$-means

[Avrithis et al. ’15]

- inverse search as in RR
- points quantized as in DRVQ; search as in multi-index
- $k$ dynamically estimated as in AGM
Inverted-quantized $k$-means

representation: for each cell $u_\alpha$, with $X_\alpha = \{x \in X : q(x) = u_\alpha\}$
- probability $p_\alpha = |X_\alpha|/n$
- mean $\mu_\alpha = \frac{1}{|X_\alpha|} \sum_{x \in X_\alpha} x$ of all points in $X_\alpha$

update: for each centroid $c_m$, with $A_m = \{\alpha \in I : a(u_\alpha) = m\}$

$$c_m \leftarrow \frac{1}{\sum_{\alpha \in A_m} p_\alpha} \sum_{\alpha \in A_m} p_\alpha \mu_\alpha,$$

assignment: for each centroid $c_m$,
- find the $w$ nearest sub-codewords in each of two sub-codebooks
- run multi-sequence independently in $w \times w$ search block
- assign visited cells $m \leftarrow a(u_\alpha)$; resolve conflicts
Centroid-to-cell search

(a) visited cells on original grid

(b) search block of $c_1$

(c) search block of $c_2$
Dynamic IQ-means
Dynamic IQ-means
Dynamic IQ-means
Dynamic IQ-means
Dynamic IQ-means

- quantize each centroid to closest cell just before search
- get centroid-to-centroid search at no extra cost
- greedily delete centroids as in EGM [Avrithis & Kalantidis ’12]
Comparison on SIFT1M with $k \in \{10^3, \ldots, 10^4\}$
Comparison on YFCC100M, initial $k = 10^5$

AlexNet fc7 features, 128 dimensions, optimized decomposition

<table>
<thead>
<tr>
<th></th>
<th>Cell-KM</th>
<th>DKM ($\times 300$)</th>
<th>D-IQ-Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k/k'$</td>
<td>100000</td>
<td>100000</td>
<td>85742</td>
</tr>
<tr>
<td>time (s)</td>
<td>13068.1</td>
<td>7920.0</td>
<td>140.6</td>
</tr>
<tr>
<td>precision</td>
<td>0.474</td>
<td>0.616</td>
<td>0.550</td>
</tr>
</tbody>
</table>

**Cell-KM** $k$-means on points quantized to cell

**DKM** distributed $k$-means on 300 machines
Mining on YFCC100M

Paris500k

Paris500k + YFCC100M

http://image.ntua.gr/iva/research/

Thank you!