LOH and Behold: Web-scale visual search, recommendation and clustering using Locally Optimized Hashing

Yannis Kalantidis[†], Lyndon Kennedy[‡*], Huy Nguyen[†],
Clayton Mellina[†] and David A. Shamma[§*]

[†]Computer Vision and Machine Learning Group, Flickr, Yahoo
[‡]Futurewei Technologies Inc.
[§]CWI: Centrum Wiskunde & Informatica, Amsterdam

{ykal,huyng,clayton}@yahoo-inc.com,
lyndonk@acm.org, aymans@acm.org

Abstract. We propose a novel hashing-based matching scheme, called Locally Optimized Hashing (LOH), based on a state-of-the-art quantization algorithm that can be used for efficient, large-scale search, recommendation, clustering, and deduplication. We show that matching with LOH only requires set intersections and summations to compute and so is easily implemented in generic distributed computing systems. We further show application of LOH to: a) large-scale search tasks where performance is on par with other state-of-the-art hashing approaches; b) large-scale recommendation where queries consisting of thousands of images can be used to generate accurate recommendations from collections of hundreds of millions of images; and c) efficient clustering with a graph-based algorithm that can be scaled to massive collections in a distributed environment or can be used for deduplication for small collections, like search results, performing better than traditional hashing approaches while only requiring a few milliseconds to run. In this paper we experiment on datasets of up to 100 million images, but in practice our system can scale to larger collections and can be used for other types of data that have a vector representation in a Euclidean space.

1 Introduction

The rapid rise in the amount of visual multimedia created, shared, and consumed requires the development of better large-scale methods for querying and mining large data collections. Similarly, with increased volume of data comes a greater variety of use cases, requiring simple and repurposeable pipelines that can flexibly adapt to growing data and changing requirements.

Recent advances in computer vision have shown a great deal of progress in analyzing the content of very large image collections, pushing the state-of-the-art for classification [27], detection [7, 11] and visual similarity search [3, 20, 22, 31, 37]. Critically, deep Convolutional Neural Networks (CNNs) [23] have allowed processing pipelines

---

[*] Work done while author was at Yahoo Labs.

to become much simpler by reducing complex engineered systems to simpler systems learned end-to-end and by providing powerful, generic visual representations that can be used for a variety of downstream visual tasks. Recently it has been shown that such deep features can be used to reduce visual search to nearest neighbor search in the deep feature space [5]. Complimentary work has recently produced efficient algorithms for approximate nearest neighbor search that can scale to billions of vectors [10, 17, 21].

In this paper, we present a novel matching signature, called *Locally Optimized Hashing* (*LOH*). LOH extends LOPQ [21], a state-of-the-art nearest neighbor search algorithm, by treating the quantization codes of LOPQ as outputs of hashing functions. When applied to deep features, our algorithm provides a very flexible solution to a variety of related large-scale search and data mining tasks, including fast visual search, recommendation, clustering, and deduplication. Moreover, unlike [10, 17, 21], our system does not necessarily require specialized resources (*i.e.* dedicated cluster nodes and indexes for visual search) and is easily implemented in generic distributed computing environments.

Our approach sacrifices precision for speed and generality as compared to more exact quantization approaches, but it enables applications that wouldn't be computationally feasible with more exact approaches. LOH can trivially cope with large multi-image query sets. In practice, our approach allows datasets of *hundreds of millions* of images to be efficiently searched with query sets of *thousands of images*. We are in fact able to query with multiple large query sets, *e.g.* from Flickr groups, simultaneously and get visual recommendations for all the sets in parallel. We are also able to cluster web-scale datasets with MapReduce by simply thresholding LOH matches and running a connected components algorithm. The same approach can be used for deduplication of, *e.g.* search results.

Our contributions can be summarized as follows:

1. We propose Locally Optimized Hashing (LOH), a novel hashing-based matching method that competes favorably with the state-of-the-art hashing methods for search and allows approximate ordering of results.
2. We extend LOH to multiple image queries and provide a simple and scalable algorithm that can provide visual recommendations in batch for query sets of thousands of images.
3. We show that this same representation can be used to efficiently deduplicate image search results and cluster collections of hundreds of millions of images.

Although in this paper we experiment on datasets of up to 100 million images (*i.e.* using the YFCC100M dataset [29], the largest publicly available image dataset), in practice our system is suited to web-scale multimedia search applications with billions of images. In fact, on a Hadoop cluster with 1000 nodes, our approach can find and rank similar images *for millions of users from a search set of hundreds of millions of images in a runtime on the order of one hour*. The method can be adapted to other data types that have vector representations in Euclidean space.

2   Related Work

Large scale nearest neighbor search was traditionally based on hashing methods [6, 26] because they offer low memory footprints for index codes and fast search in Hamming space [25]. However, even recent hashing approaches [19, 28, 33] suffer in terms of performance compared to quantization-based approaches [16, 25] for the same amount of memory. On the other hand, quantization-based approaches traditionally performed worse in terms of search times, and it was only recently with the use of novel indexing methods [4] that quantization-based search was able to achieve search times of a few milliseconds in databases of billions of vectors [10, 21, 24].

A benefit of quantization approaches is that, unlike classic hashing methods, they provide a ranking for the retrieved points. Recently, approaches for binary code re-ranking have been proposed in [32, 36]; both papers propose a secondary, computationally heavier re-ranking step that, although is performed on only the retrieved points, makes search slower than state-of-the-art quantization-based approaches. In the approach presented here, we try to keep the best of both worlds by producing an approximate ordering of retrieved points without re-ranking. We argue that for use cases involving multiple queries, this approximation can be tolerated since many ranked lists are aggregated in this case.

A similar approach to ours, *i.e.* an approach that aims to produce multipurpose, *polysemous* codes [8] is presented at the current ECCV conference. After training a product quantizer, the authors then propose to optimize the so-called index assignment of the centroids to binary codes, such that distances between similar centroids are small in the Hamming space.

For multi-image queries, there are two broad categories based on the semantic concepts that the query image set represents. The first is query sets that share the same semantic concept or even the same specific object (*i.e.* a particular building in Oxford) [1, 9, 31, 38]. The second category is multi-image queries with *multiple* semantics. This category has been recently studied [15] and the authors propose a Pareto-depth approach on top Efficient Manifold Ranking [35] for such queries. Their approach is however not scalable to very large databases and they limit query sets to just be image pairs.

The current work uses visual features from a CNN trained for classification, thus similarities in our visual space capture broader category-level semantics. We focus on the first category of multi-image queries, *i.e.* multiple-image query sets with a single semantic concept, and provide a simple and scalable approach which we apply to Flickr group set expansion. However, it is straightforward to tackle the second category with our approach by introducing a first step of (visual or multi-modal) clustering on the query set with multiple semantics before proceeding with the LOH-based set expansion.

3   Locally Optimized Hashing

3.1   Background

Product quantization. A *quantizer* q maps a d-dimensional vector $\mathbf{x} \in \mathbb{R}^d$ to vector $q(\mathbf{x}) \in \mathcal{C}$, where $\mathcal{C}$ is a finite subset of $\mathbb{R}^d$, of cardinality k. Each vector $\mathbf{c} \in \mathcal{C}$ is called

a *centroid*, and $\mathcal{C}$ a *codebook*. Assuming that dimension d is a multiple of m, we may write any vector $\mathbf{x} \in \mathbb{R}^d$ as a concatenation $(\mathbf{x}^1, \ldots, \mathbf{x}^m)$ of m sub-vectors, each of dimension d/m. If $\mathcal{C}^1, \ldots, \mathcal{C}^m$ are m sub-codebooks of k sub-centroids in subspace $\mathbb{R}^{d/m}$, a *product quantizer* [16] constrains $\mathcal{C}$ to be a Cartesian product

$$\mathcal{C} = \mathcal{C}^1 \times \cdots \times \mathcal{C}^m, \tag{1}$$

making it a codebook of k^m centroids of the form $\mathbf{c} = (\mathbf{c}^1, \ldots, \mathbf{c}^m)$ with each sub-centroid $\mathbf{c}^j \in \mathcal{C}^j$ for $j \in \mathcal{M} = \{1, \ldots, m\}$. An optimal product quantizer q should minimize distortion $E = \sum_{\mathbf{x} \in \mathcal{X}} \|\mathbf{x} - q(\mathbf{x})\|^2$. as a function of $\mathcal{C}$, subject to $\mathcal{C}$ being of the form (1) [10]. This is typically done with a variant of k-means.

When codebook $\mathcal{C}$ is expressed as a Cartesian product, for each vector $\mathbf{x} \in \mathbb{R}^d$, the nearest centroid in $\mathcal{C}$ is

$$q(\mathbf{x}) = (q^1(\mathbf{x}^1), \ldots, q^m(\mathbf{x}^m)), \tag{2}$$

where $q^j(\mathbf{x}^j)$ is the nearest sub-centroid of sub-vector $\mathbf{x}^j$ in $\mathcal{C}^j$, for $j \in \mathcal{M}$ [10]. Hence finding an optimal product quantizer q in d dimensions amounts to solving m optimal sub-quantizer problems $q^j, j \in \mathcal{M}$, each in d/m dimensions.

Given a new *query* vector $\mathbf{y}$, the (squared) Euclidean distance to every point $\mathbf{x} \in \mathcal{X}$ may be approximated by

$$\delta^{SDC}(\mathbf{y}, \mathbf{x}) = \sum_{j=1}^{m} \|q^j(\mathbf{y}^j) - q^j(\mathbf{x}^j)\|^2, \tag{3}$$

or

$$\delta^{ADC}(\mathbf{y}, \mathbf{x}) = \sum_{j=1}^{m} \|\mathbf{y}^j - q^j(\mathbf{x}^j)\|^2, \tag{4}$$

where $q^j(\mathbf{x}^j) \in \mathcal{C}^j = \{\mathbf{c}_1^j, \ldots, \mathbf{c}_k^j\}$ for $j \in \mathcal{M}$. The superscripts SDC and ADC correspond to the symmetric and asymmetric distance computations of [16], respectively. In the latter case the query vector is not quantized using the product quantizer, distances $\|\mathbf{y}^j - \mathbf{c}_i^j\|^2$ are computed and stored for $i \in \mathcal{K}$ and $j \in \mathcal{M}$ prior to search, so (4) amounts to only $O(m)$ operations. Sacrificing distortion for speed, in this approach we are mostly exploring the symmetric approximation (3), where the query is also in quantized form. In this case, sub-quantizer distances $\|\mathbf{c}_l^j - \mathbf{c}_i^j\|^2$ can be pre-computed and stored for all $i, l \in \mathcal{K}$ and $j \in \mathcal{M}$ and again only $O(m)$ operations are needed for distance computations.

Locally Optimized Product Quantization. In their recent paper [21], the authors further extend product quantization by optimizing multiple product quantizers *locally*, after some initial, coarse quantization of the space. Similar to the IVFADC version of [16] or multi-index [4], they adopt a two-stage quantization scheme, where local optimization follows independently inside each cluster of a coarse quantizer Q, learnt on the residual vectors with respect to the cluster's centroid. They learn an *optimized* product

quantizer [10] per cluster, jointly optimizing the subspace decomposition together with the sub-quantizers. Constraint (1) of the codebook is relaxed to

$$\mathcal{C} = \{R\hat{\mathbf{c}} : \hat{\mathbf{c}} \in \mathcal{C}^1 \times \cdots \times \mathcal{C}^m, R^{\mathrm{T}}R = I\}, \tag{5}$$

where the $d \times d$ matrix R is orthogonal and allows for arbitrary rotation and permutation of vector components.

Given the coarse quantizer Q and the associated codebook $\mathcal{E} = \{\mathbf{e}_1, \ldots, \mathbf{e}_K\}$ of K *clusters*, for $i \in \mathcal{K} = \{1, \ldots, K\}$ we may define the set of residuals of all data points $\mathbf{x} \in \mathcal{X}_i$ quantized to cluster i as $\mathcal{Z}_i = \{\mathbf{x} - \mathbf{e}_i : \mathbf{x} \in \mathcal{X}_i, Q(\mathbf{x}) = \mathbf{e}_i\}$. Given a set $\mathcal{Z} \in \{\mathcal{Z}_1, \ldots, \mathcal{Z}_K\}$, the problem of locally optimizing both space decomposition and sub-quantizers can be expressed as minimizing distortion as a function of orthogonal matrix $R \in \mathbb{R}^{d \times d}$ and sub-codebooks $\mathcal{C}^1, \ldots, \mathcal{C}^m \subset \mathbb{R}^{d/m}$ per cell,

$$\begin{aligned} \text{minimize} \quad & \sum_{\mathbf{z} \in \mathcal{Z}} \min_{\hat{\mathbf{c}} \in \hat{\mathcal{C}}} \|\mathbf{z} - R\hat{\mathbf{c}}\|^2 \\ \text{subject to} \quad & \hat{\mathcal{C}} = \mathcal{C}^1 \times \cdots \times \mathcal{C}^m \\ & R^{\mathrm{T}}R = I, \end{aligned} \tag{6}$$

where $|\mathcal{C}^j| = k$ for $j \in \mathcal{M} = \{1, \ldots, m\}$. Assuming a d-dimensional, zero-mean normal distribution $\mathcal{N}(\mathbf{0}, \Sigma)$ of residual data $\mathcal{Z}$, we can efficiently solve the problem by first aligning the data with PCA and then using the *eigenvalue allocation* [10] algorithm to assign dimensions to subspaces.

To achieve the state-of-the-art results on a billion-scale dataset, the inverted multi-index [4] is used with local optimization. In this setting, the original space is split into two subspaces first and then LOPQ follows within each one of the two subspaces separately, on the residual vectors.

Each data point now gets assigned in *two* clusters, one in each subspace. The intersection of two clusters gives a multi-index *cell* in the product space. However, as the space overhead to locally optimize per cell is prohibitive, in [21] the authors separately optimize *per cluster* in each of the two subspaces. They refer to this type of local optimization as *product optimization* and the complete algorithm as *Multi-LOPQ*, which is the approach we also adopt for training. As was shown in [21], using local rotations together with a single set of global sub-quantizers gave only a small drop in performance. We therefore choose to have a global set of sub-quantizers $q(\mathbf{x})$ defined as in (2) and trained on the projected residual vectors $\mathbf{x}$ from (10). We will refer to the two quantization stages as *coarse* and *fine* quantization, respectively.

3.2 Locally Optimized Hashing

The symmetric distance computation of (3) yields an approximation of the true distance in the quantized space. In pursuit of an even more scalable, fast and distributed-computing friendly approach, we propose to treat the sub-quantizer centroid indices as *hash codes*. This allows us to approximate the true distance via *collisions* without any explicit numeric computations apart from a final summation. As we demonstrate below, the proposed formulation further generalizes to querying with multiple vector queries, and, in fact, can perform many such queries in parallel very efficiently.

Lets begin by assuming a single coarse quantizer Q. The LOPQ model contains local rotations R_c and global sub-quantizers $q_c^j, j \in \mathcal{M}$ and $c \in \mathcal{K}$ that operate on the projected residuals. Let $\mathbf{x}, \mathbf{y} \in \mathcal{Z}_c$ be such residual vectors with respect to the same centroid of cluster c of the coarse quantizer. The symmetric distance computation of (3) is now given by

$$\delta_c^{SDC}(\mathbf{y}, \mathbf{x}) = \|q_c(\mathbf{y}) - q_c(\mathbf{x})\|^2 = \sum_{j=1}^{m} \|q_c^j(\mathbf{y}^j) - q_c^j(\mathbf{x}^j)\|^2, \tag{7}$$

where q_c^j is the j-th sub-quantizer for cluster c, with $c \in \mathcal{K}$, $j \in \mathcal{M}$ and $\mathbf{x}, \mathbf{y} \in \mathcal{Z}_c$. A residual vector $\mathbf{x}$ is mapped via q_c to the corresponding sub-centroid indices $i_c(\mathbf{x}) = (i_c(\mathbf{x})^1, \ldots, i_c(\mathbf{x})^m)$. We may treat the indices as values of a set of hash functions $h_c = (h_c^1, \ldots, h_c^m)$, *i.e.* a mapping $h_c : \mathbb{R}^d \to \mathbb{Z}^m$ such that $h_c(\mathbf{x}) = i_c(\mathbf{x})$. We can then estimate the *similarity* between residual vectors $\mathbf{y}, \mathbf{x}$ using the function:

$$\sigma_h(\mathbf{y}, \mathbf{x}) = \sum_{j=1}^{m} \mathbb{1}[h_c(\mathbf{y})^j = h_c(\mathbf{x})^j], \tag{8}$$

where $\mathbb{1}[a = b]$ equals to 1, iff $a = b$ and 0 otherwise. Our hash functions are defined locally, *i.e.* on residual vectors for a specific cluster, and therefore we call the proposed matching scheme *Locally Optimized Hashing* or *LOH*.

The LOH approach can be extended to work on top of a multi-LOPQ [21] model. Instead of having a single coarse quantizer, we learn two subspace quantizers Q^1, Q^2 of K centroids, with associated codebooks $\mathcal{E}^j = \{\mathbf{e}_1^j, \ldots, \mathbf{e}_K^j\}$ for $j = 1, 2$ in a product quantization fashion.

Each data point $\tilde{\mathbf{x}} = (\tilde{\mathbf{x}}^1, \tilde{\mathbf{x}}^2) \in \mathcal{X}$ is quantized using the two coarse quantizers into the tuple

$$(c_1, c_2) = (\arg\min_i \|\tilde{\mathbf{x}}^1 - \mathbf{e}_i^1\|, \arg\min_i \|\tilde{\mathbf{x}}^2 - \mathbf{e}_i^2\|), \tag{9}$$

with $c_j \in [1, K]$ referring to the indices of the nearest clusters for the two subspaces $j = 1, 2$. We will refer to the tuple $c(\tilde{\mathbf{x}}) = (c_1, c_2)$ as the *coarse codes* of a data point. Following LOPQ, the residual vector $\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2)$ of point $\tilde{\mathbf{x}}$ is equal to:

$$\mathbf{x} = (R_{c_1}^1(\tilde{\mathbf{x}}^1 - \mathbf{e}_{c_1}^1), R_{c_2}^2(\tilde{\mathbf{x}}^2 - \mathbf{e}_{c_2}^2)), \tag{10}$$

where R_i^j correspond to the local rotation of cluster i in subspace j. We can split the concatenated vector $\mathbf{x}$ into m subvectors $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^m)$ and use the global sub-quantizers for encoding into m codes. Therefore, given sub-quantizer $q(\mathbf{x})$ with codebooks $\mathbf{c} = (\mathbf{c}^1, \ldots, \mathbf{c}^m)$ and each sub-centroid $\mathbf{c}^j \in \mathcal{C}^j = \{\mathbf{c}_1^j, \ldots, \mathbf{c}_k^j\}$ for $j \in \mathcal{M} = \{1, \ldots, m\}$, we can compute

$$f_j = \arg\min_i \|\mathbf{x}^j - \mathbf{c}_i^j\|, \tag{11}$$

for all $j \in \mathcal{M}$ and get the sub-quantizer indices for the m subspaces in set $f(\mathbf{x}) = (f_1, \ldots, f_m)$. We will refer to this set as the *fine codes* of a data point. An overview of the encoding process is shown in Algorithm 1.

Algorithm 1: Data encoding for the multi-index case

input : data point $\tilde{\mathbf{x}} \in \mathcal{X}$, number of subspaces m, coarse quantizer codebooks $\mathcal{E}^j$, local rotations R_i^j where $i = 1, \ldots, K$ and $j = 1, 2$ and sub-quantizer codebooks $\mathbf{c} = (\mathbf{c}^1, \ldots, \mathbf{c}^m)$.

output: sets of coarse and fine codes

// calculate coarse codes
1 $(c_1, c_2) = (\arg\min_i \|\tilde{\mathbf{x}}^1 - \mathbf{e}_i^1\|, \arg\min_i \|\tilde{\mathbf{x}}^2 - \mathbf{e}_i^2\|)$

// calculate locally projected residuals
2 $\mathbf{x} = (R_{c_1}^1(\tilde{\mathbf{x}}^1 - \mathbf{e}_{c_1}^1), R_{c_2}^2(\tilde{\mathbf{x}}^2 - \mathbf{e}_{c_2}^2))$

// split residual to m subvectors
3 $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^m)$

// calculate fine codes
4 $(f_1, \ldots, f_m) = (\arg\min_i \|\mathbf{x}^1 - \mathbf{c}_i^1\|, \ldots, \arg\min_i \|\mathbf{x}^m - \mathbf{c}_i^m\|)$

Now, given residual vectors $\mathbf{x}, \mathbf{y}$ with respect to the same set of coarse codes, and their sets of fine codes $f(\mathbf{x}), f(\mathbf{y})$, the similarity function of (8) can be expressed as

$$\sigma_h(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{m} \mathbb{1}[f_i(\mathbf{y}) = f_i(\mathbf{x})] \tag{12}$$

i.e. the sum of similarities for the m subspaces. We should note that since fine codes are calculated on residuals, *i.e.* given a coarse centroid, they are comparable only for points that share at least one of the coarse codes. If two points, for example share only the first coarse code of the two, only the first half of the fine codes are comparable.

3.3 Ranking with LOH

After we encode all database points using the approach summarized in Algorithm 1, each one will be assigned to the *cell* of the multi-index that corresponds to the pair of coarse codes of each data point. For indexing, an inverted list $\mathcal{L}_c$ is kept for each cell $c = c_{ij} = (c_i, c_j)$ of the multi-index, giving K^2 inverted lists in total.

For search one visits the inverted lists of multiple cells. The query vector is therefore not projected to just the closest coarse cluster for each of the coarse quantizers, but to multiple, and cells are visited in a sequence dictated by the multi-sequence algorithm [4]. As we are only counting collisions of fine codes within a coarse cluster, we need a way of incorporating the distance of the query to the centroid for each cell visited to further rank points *across* cells. The multi-sequence algorithm provides us with an (approximate) distance d_c for each cell which we use to extend the similarity function presented in the previous section. We introduce a weight w_c for each cell c visited that encodes the similarity of the query's residual to that cell's centroid, and modify the similarity function to be:

$$\sigma_w(\mathbf{y}, \mathbf{x}) = w_c + \sigma_h(\mathbf{y}, \mathbf{x}) = w_c + \sum_{i=1}^{m} \mathbb{1}[f_i(\mathbf{y}) = f_i(\mathbf{x})] \tag{13}$$

Algorithm 2: Pseudo-code for batch search in PIG

---

    input : queries, documents: flattened query and database files. Each row contains some id
           and a triplet of a fine code, its position in the ordered set and its corresponding
           coarse code.

    output: scores: list of documents sorted by LOH similarity to the query set

    // load flattened files for query set

1 Q = LOAD 'queries' AS (`user_id`, `image_id_q`,
    (`coarse_code`, `position`, `fine_code`) as `code_q`);

    // load flattened files for the database set

2 D = LOAD 'documents' AS (`image_id_d`, (`coarse_code`, `position`, `fine_code`)
    as `code_d`);

    // join by code

3 matches = JOIN Q by `code_q`, D BY `code_d`;

    // group by document

4 grouped = GROUP matches BY (`user_id`, `image_id_d`);

    // count the matches within each group

5 scores = FOREACH grouped GENERATE group.$0 as `user_id`, group.$1 as
    `image_id`, COUNT(matches) as `n_matches`;

    // order by number of matches

6 scores = ORDER scores BY `user_id` ASC, `n_matches` DESC;

We choose to set these weight to a simple exponential function of the approximate distances d_c used by the multi-sequence algorithm.

Similarities are evaluated for every vector in list $\mathcal{L}_c$, for every cell c returned by the multi-sequence algorithm. In [4], search is terminated if a quota of at least T vectors have been evaluated. For a query vector $\mathbf{y}$, we may assume that cells $\{c^1, \ldots, c^W\}$ were visited by the multi-index algorithm until termination. Let that sequence of lists visited be $\mathcal{L}^{\mathbf{y}} = \{\mathcal{L}_{c^1}, \ldots, \mathcal{L}_{c^W}\}$. Also let $\mathcal{X}^{\mathbf{y}} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ be the sequence of the T database vectors evaluated during search, concatenated from W disjoint inverted lists. Let also set $\mathcal{S}^{\mathbf{y}} = \{\sigma_w(\mathbf{y}, \mathbf{x}_1), \ldots, \sigma_w(\mathbf{y}, \mathbf{x}_T)\}$, hold the similarity values of the query vector $\mathbf{y}$ with each of the top T database vectors returned by the multi-index. The ranked list of nearest neighbors returned for the query is the top elements of $\mathcal{S}^{\mathbf{y}}$ in descending order according to the approximate similarity values.

3.4   Searching with large query sets

An application like recommendation requires the ability to jointly search with multiple query vectors and get aggregated results. Taking visual recommendation as an example, one can define query sets in multiple ways, *e.g.* the set of images in a given Flickr group, or the set of images that a given user has favorited. If each image is represented as a vector, *e.g.* using CNN-based global visual features, queries with *image sets* correspond to queries with *multiple vectors* and can produce results that are visually similar to the whole query image set.

Now let's suppose that the query is a set of Y vectors, $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_Y\}$. If we query the index for each of the query vectors, we get sets $\mathcal{L}^{\mathbf{y}}$, $\mathcal{X}^{\mathbf{y}}$ and $S^{\mathbf{y}}$ for each vector corresponding to cells, database vectors and similarities, respectively, with $\mathbf{y} = 1, \ldots, Y$. We can now define the set of similarity values

$$S_{\mathcal{Y}} = \{\sigma(\mathcal{Y}, \mathbf{x}_t)\} \text{ for } t \in \bigcup_{\mathbf{y} \in Y} \mathcal{X}^{\mathbf{y}}, \tag{14}$$

between the query set $\mathcal{Y}$ and all database vectors evaluated in any of the single-vector queries. The aggregation function $\sigma(\mathcal{Y}, \mathbf{x}) = g(\sigma_w(\mathbf{y}_1, \mathbf{x}), \ldots, \sigma_w(\mathbf{y}_Y, \mathbf{x}))$ measures the similarity of the query set $\mathcal{Y}$ to database vector $\mathbf{x}$, where g can be any pooling function, for example

$$\sigma_{SUM}(\mathcal{Y}, \mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}} \sigma_w(\mathbf{y}, \mathbf{x}), \tag{15}$$

for sum-pooling or $\sigma_{MAX}(\mathcal{Y}, \mathbf{x}) = \arg\max_{\mathbf{y} \in \mathcal{Y}} \sigma_w(\mathbf{y}, \mathbf{x})$ for max-pooling. We experimentally found that sum-pooling performs better than max-pooling, which is understandable since the latter tends to under-weight results that appeared in the result sets of many query vectors. We also experimented with more complex functions, *e.g.* functions that combine max-pooling with the frequency that each database image appears in the result sets, but since the improvements were minimal we end up using the simpler sum-pooling function of (15) for the rest of the paper.

An advantage of representing images as multiple hash codes is that they can be naturally manipulated for a variety of tasks with MapReduce using tools such as PIG or HIVE for Hadoop. Returning to our example of image recommendations in Flickr, we might use a user's favorited images as a query set to produce a ranked set of recommended images that are visually similar to images the user has favorited. In this case we would like to produce recommendations for *all* users, and we would like an algorithm that can run many searches efficiently in batch to periodically recompute image recommendations for all users.

We show PIG pseudo-code for such a batch search in Algorithm 2. The algorithm assumes that the coarse and fine codes for each document have already been computed and are available in a *flattened* form. To get this form, we first split the m fine codes and create triplets by appending each fine code with its position in f and the corresponding coarse code. That is, for coarse and fine codes (c_1, c_2) and $(f_1, \ldots, f_m)$, respectively, we would get the set of m triplets $((c_1, 1, f_1), (c_1, 2, f_2), \ldots, (c_2, m, f_m))$ or *LOH codes*.

3.5 Clustering and deduplication with LOH

LOH can also be used for efficient and scalable clustering. Unlike recent approaches that try to cluster data on a single machine [2, 13], we are interested in the distributed case. Our clustering algorithm first constructs a graph of documents from an input set $\mathcal{Y}$ such that a pair of documents $\mathbf{x}, \mathbf{y} \in \mathcal{Y}$ is connected by an edge iff the LOH similarity of the pair is above some threshold t, *i.e.* $\sigma_h(\mathbf{y}, \mathbf{x}) > t$. Clustering then amounts to finding connected components in this graph.

Algorithm 3: Pseudo-code for LOH clustering

> **input** : documents $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ with the set of flattened triplets, threshold t
> **output**: clusters $\mathcal{R} = \{r_1, \ldots, r_n\}$

1 Groups = HashMap<List>()
2 Matches = HashMap<Int>()
3 DocumentGraph = Graph()

  // group documents by matching LOH codes
4 **foreach** $d \in D$ **do**
5   **foreach** $(c_i, j, f_j) \in d$ **do**
6   Groups((c_i, j, f_j)).append(d)

  // count num of matching codes for pairs of documents in each group
7 **foreach** $group \in Groups$ **do**
8   **foreach** $(d_a, d_b) \in allPairs(group)$ **do**
9   Matches((d_a, d_b))++
10   **if** *Matches((d_a, d_b))* $> t$ **then**
11   DocumentGraph.addEdge(d_a, d_b)

  // find connected components in the document graph
12 $\mathcal{R} \longleftarrow$ DocumentGraph.findConnectedComponents()
13 **return** $\mathcal{R}$

We present pseudo-code for LOH clustering in Algorithm 3. The algorithm first groups documents by flattened code triplets. This grouping is used to efficiently count the number of matches for document pairs by greatly reducing the number of pairs we consider when constructing the graph. Like the batch search algorithm, LOH clustering is easily implemented in MapReduce frameworks. When running with a high threshold for a small set of images, *e.g.* for the top hundred or thousand results after an image search, this algorithm can deduplicate the set in real-time, requiring only a few milliseconds.

4   Experiments

We use the following four datasets:
SIFT1M dataset [16]. This dataset contains of 1 million 128-dimensional SIFT vectors and 10K query vectors and is a common benchmark dataset in related work [21, 25, 33].
Yahoo Flickr Creative Commons 100M dataset [30]. This dataset (YFCC100M) contains a subset of 100 million public images with a creative commons license from Flickr and is the largest such publicly available collection of social multimedia images.
Flickr Brad Pitt Search Dataset. This dataset contains the top 1048 photo results from a query for the search term "Brad Pitt" on the Flickr website. At the time that we collected this data, results from this query exhibited a large amount of "near-duplicate" results. We manually grouped each photo that looked visually similar into the same cluster. The dataset contains a total of 30 clusters with more than 1 image.
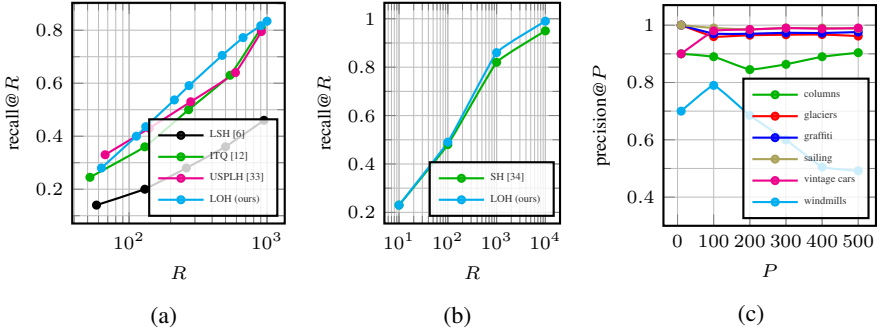
Fig. 1: *Left*: Recall@R on SIFT1M with 64bit codes. Recall is measured here as the percentage of times the true nearest neighbor is returned within the top R results returned by the index for all $10K$ queries. $K = 1024$ for LOH. *Center*: Recall@R on SIFT1M for the proposed LOH and Spectral Hashing. LOH only takes into account the top $T = 10000$ results returned by the multi-index, while SH is exhaustive. *Right*: Precision@P for the 7 Flickr Group Photos dataset. Recommendations for the group "Portraits & Faces" are not depicted because they were flawless.

7 Flickr Groups dataset. This dataset contains $70K$ images in total and was constructed by collecting $10K$ images from 7 popular Flickr groups: *Graffiti of the world*, *Sailboats and sailing*, *Glaciers, Icefields and Icebergs*, *Windmills*, *Columns and Columns*, *Vintage Cars and Trucks* and *Portraits and Faces*.

We use the $fc7$ features of the pretrained AlexNet model [23] from Caffe [18] and use PCA to reduce them to 128 dimensions. We learn a covariance matrix from 100 million images of the YFCC100M dataset and further permute the dimensions in order to balance variance between the two subspaces before multi-indexing [10]. As in all related work [10, 16, 21], we set the number of sub-quantizer centroids $k = 256$, *i.e.* we require m bytes of memory in total per vector.

We adopt the Multi-LOPQ [21] approach to train[1] and index the database points. We use parameters $K = 1024$ ($K = 8192$), $m = 8$ ($m = 16$) and for $T = 10K$ ($T = 100K$) for SIFT1M (YFCC100M).

We use the recall metric to measure the performance of LOH against related methods and conduct experiments on the SIFT1M dataset. To compare with hashing methods that do not return any ordering of the retrieved points, we measure the percentage of times the true nearest neighbor is within the top R results returned by the index (therefore varying parameter T of the multi-index for LOH) for all $10K$ queries of the SIFT1M dataset. We use the precision-recall metric for evaluating deduplication.

4.1 Approximate nearest neighbor search with LOH

We first investigate how the LOH approach compares with the hashing literature for the task of retrieving the true nearest neighbor within the first R samples seen. We compare

---

[1] https://github.com/yahoo/lopq
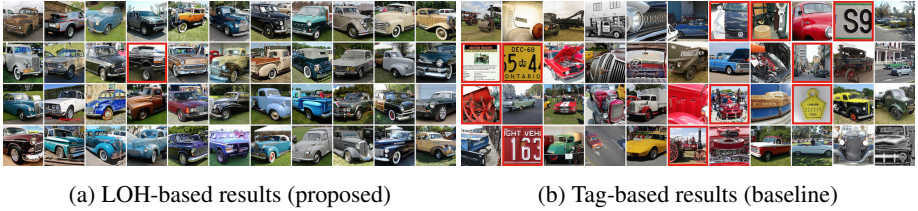
(a) LOH-based results (proposed)   (b) Tag-based results (baseline)

Fig. 2: Suggestions for Flickr group "vintage cars and trucks", false positives are marked by a red border. Figure 2a: Our visual similarity-based suggestions; only 1 of the top results is false and the aesthetics fit the images in the group. Figure 2b: Top images returned by tag-based search for "vintage cars and trucks"; we see more false positives in this case.

against classic hashing methods like Locality Sensitive Hashing (LSH) [6], Iterative Quantization [12] and the recent Sequential Projection Learning Hashing (USPLH) [33] and report results in Figure 1a. One can see that LOH, built on the inverted multi-index after balancing the variance of the two subspaces, compares well with the state-of-the-art in the field, even outperforming recently proposed approaches like [33] for large enough R.

In 1b, we evaluate LOH ranking, *i.e.* how well LOH orders the true nearest neighbor after looking at a fraction of the database. We compare against Spectral Hashing (SH) [34] which, like LOH, also provides a ranking of the results. LOH performs similarly for small values of R but outperforms SH when retrieving more than $R = 100$ results, which is the most common case.

4.2   Visual recommendations for Flickr groups

We conduct an experiment to evaluate the ability of the proposed approach to visually find images that might be topically relevant to a group of photos already curated by a group of users. On Flickr, such activity is common as users form groups around topical photographic interests and seek out high-quality photos relevant to the group. Group moderators may contact photo owners to ask them to submit to their group.

To evaluate this, we select 7 public Flickr groups that are representative of the types of topical interests common in Flickr groups, selected due to their clear thematic construction (*graffiti*, *sailing*, *glacier*, *windmill*, *columns*, *cars & trucks*, *portrait & face*), for ease of objective evaluation. For each group, we construct a large query of $10,000$ images randomly sampled from the group pool. We perform visual search using our proposed method on the YFCC100M dataset, aggregate results from all 10 thousand images and report precision after manual inspection of the top $k = 500$ results. We visually scanned the photo pools of the groups and consider true positives all images that look like images in the photo pool and follow the group rules as specified by the administrators of each group.

Precision for each group is shown in Figure 1c. For group "Portraits & Faces", for example *all* 500 top results were high-quality portraits. We see some confusion due
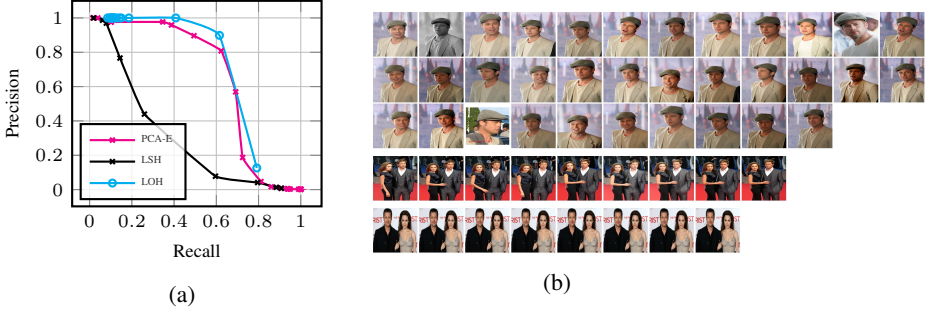
Fig. 3: *Left:* Precision-Recall curve on the Brad Pitt dataset. *Right:* Deduplication results on the Brad Pitt dataset; three clusters of duplicates are shown where each cluster shares at least 3 LOH codes.

to the nature of the visual representation chosen (*e.g.* our visual representation may confuse desert and cloud images with snow images), but overall, Precision@500 was over 0.96 for five out of the seven groups we tested.

Example results for the set expansion with our method and a baseline tag-based search are shown in Figure 2 for Flickr group "vintage cars and trucks". For the proposed approach, precision is high, as is the aesthetic quality of the results. The tag-based Flickr search returns more false positives for such a specific group, as irrelevant images are likely improperly tagged.

4.3 Clustering and deduplication results

We evaluate the performance of LOH on the deduplication task using a dataset of Flickr searches for the query "Brad Pitt" with the LOH codes learned on the YFCC100M dataset. To measure precision and recall, we enumerate all pairs of images in the dataset and define a "positive" sample as a pair of images that belong to the same group in our dataset, and a "negative" sample as a pair of images that belong to different groups in our dataset.

In figure 3a we plot the precision-recall for LOH versus LSH [6] and PCA-E [14]. For LSH, we transform our PCA'd 128 dimensional image descriptor into a 128-bit binary code computed from random binary projection hash functions. For PCA-E we compute a 128-bit binary code by subtracting the mean of our PCA'd 128 dimensional image descriptor and taking the sign.

We run LOH clustering for the 100 million images of the YFCC100M dataset on a small Hadoop cluster and show sample clusters in Figure 4. We first did some cleaning and preprocessing by using a stoplist of codes (*i.e.* remove all triplets that appear more than $10K$ or less than 10 times) for efficiency. For a threshold of $t = 3$, we get 74 million edges and approximately 7 million connected components. Of those, about 6.5 million are small components of size smaller than 5. The graph construction for the 100M images took a couple of hours, while connected components runs in a few minutes.
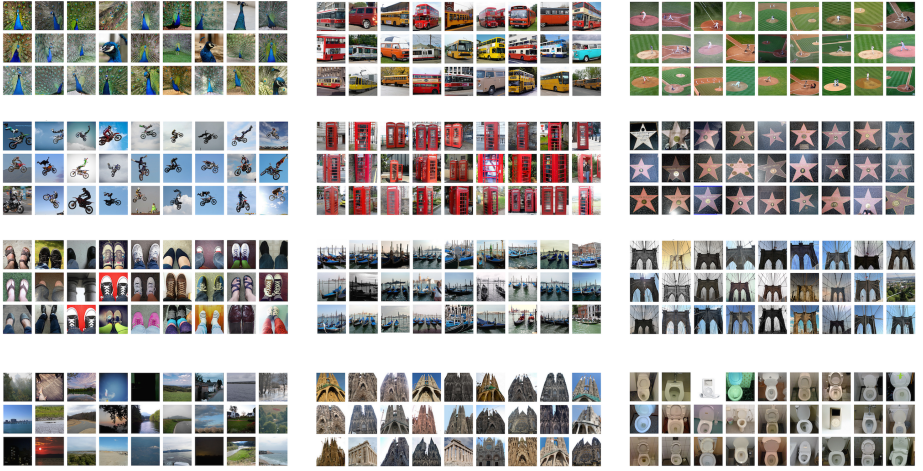
Fig. 4: Random images from sample clusters of the YFCC100M dataset. The first three rows show (mostly) coherent clusters that can be used for learning classifiers, while the bottom row shows failure cases. The sizes of the clusters depicted are (row-wise): 1194,976,920 / 272,320,873 / 139,151,164 / 6.8M, 1363 and 94.

By visual inspection, we notice that a large set of medium-sized clusters (*i.e.* clusters with $10^2 - 10^4$ images) contain visually consistent higher level concepts (*e.g.* from Figure 4: "motorbikes in the air", "Hollywood St stars" or "British telephone booths"). Such clusters can be used to learn classifiers in a semi-supervised framework that incorporates noisy labels. Clustering YFCC100M gives us about 32K such clusters.

5   Conclusions

In this paper we propose a novel matching scheme, Locally Optimized Hashing or LOH, that is computed on the very powerful and compact LOPQ codes. We show how LOH can be used to efficiently perform visual search, recommendation, clustering and deduplication for web-scale image databases in a distributed fashion.

While LOPQ distance computation gives high quality, fast distance estimation for nearest neighbor search, it is not as well suited for large-scale, batch search and clustering tasks. LOH, however, enables these use-cases by allowing implementations that use only highly parallelizable set operations and summations. LOH can therefore be used for massively parallel visual recommendation and clustering in generic distributed environments with only a few lines of code. Its speed also allows its use for deduplication of, *e.g.* , search result sets at query time, requiring only a few milliseconds to run for sets of thousands of results.

References

1. Arandjelovic, R., Zisserman, A.: Multiple queries for large scale specific object retrieval. In: BMVC (2012) 3
2. Avrithis, Y., Kalantidis, Y., Anagnostopoulos, E., Emiris, I.Z.: Web-scale image clustering revisited. In: ICCV (2015) 9
3. Avrithis, Y., Kalantidis, Y., Tolias, G., Spyrou, E.: Retrieving landmark and non-landmark images from community photo collections. In: ACM Multimedia (2010) 1
4. Babenko, A., Lempitsky, V.: The inverted multi-index. In: CVPR (2012) 3, 4, 5, 7, 8
5. Babenko, A., Slesarev, A., Chigorin, A., Lempitsky, V.: Neural codes for image retrieval. In: ECCV (2014) 2
6. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-sensitive hashing scheme based on p-stable distributions. In: Symposium on Computational Geometry (2004) 3, 11, 12, 13
7. Dollár, P., Appel, R., Belongie, S., Perona, P.: Fast feature pyramids for object detection. PAMI 36(8), 1532–1545 (2014) 1
8. Douze, M., Jégou, H., Perronnin, F.: Polysemous codes. In: ECCV (2016) 3
9. Fernando, B., Tuytelaars, T.: Mining multiple queries for image retrieval: On-the-fly learning of an object-specific mid-level representation. In: ICCV (2013) 3
10. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization. Tech. Rep. 4 (2014) 2, 3, 4, 5, 11
11. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR (2014) 1
12. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. In: CVPR (2011) 11, 12
13. Gong, Y., Pawlowski, M., Yang, F., Brandy, L., Bourdev, L., Fergus, R.: Web scale photo hash clustering on a single machine. In: CVPR (2015) 9
14. Gordo, A., Perronnin, F., Gong, Y., Lazebnik, S.: Asymmetric distances for binary embeddings. PAMI 36(1), 33–47 (2014) 13
15. Hsiao, K., Calder, J., Hero, A.O.: Pareto-depth for multiple-query image retrieval. arXiv preprint arXiv:1402.5176 (2014) 3
16. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. PAMI 33(1) (2011) 3, 4, 10, 11
17. Jégou, H., Tavenard, R., Douze, M., Amsaleg, L.: Searching in one billion vectors: Re-rank with source coding. In: ICASSP (2011) 2
18. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014) 11
19. Jin, Z., Hu, Y., Lin, Y., Zhang, D., Lin, S., Cai, D., Li, X.: Complementary projection hashing. In: ICCV (2013) 3
20. Kalantidis, Y., Tolias, G., Avrithis, Y., Phinikettos, M., Spyrou, E., Mylonas, P., Kollias, S.: Viral: Visual image retrieval and localization. MTAP (2011) 1
21. Kalantidis, Y., Avrithis, Y.: Locally optimized product quantization for approximate nearest neighbor search. In: CVPR (2014) 2, 3, 4, 5, 6, 10, 11
22. Kennedy, L., Naaman, M., Ahern, S., Nair, R., Rattenbury, T.: How flickr helps us make sense of the world: Context and content in community-contributed media collections. In: ACM Multimedia. vol. 3, pp. 631–640 (2007) 1
23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS (2012) 1, 11
24. Norouzi, M., Fleet, D.: Cartesian k-means. In: CVPR (2013) 3

25. Norouzi, M., Punjani, A., Fleet, D.J.: Fast search in Hamming space with multi-index hashing. In: CVPR (2012) 3, 10
26. Paulevé, L., Jégou, H., Amsaleg, L.: Locality sensitive hashing: a comparison of hash function types and querying mechanisms. Pattern Recognition Letters 31(11), 1348–1358 (Aug 2010) 3
27. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. arXiv preprint arXiv:1409.0575 (2014) 1
28. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. CVPR (2015) 3
29. Thomee, B., Elizalde, B., Shamma, D.A., Ni, K., Friedland, G., Poland, D., Borth, D., Li, L.J.: Yfcc100m: The new data in multimedia research. Communications of the ACM 59(2), 64–73 (2016) 2
30. Thomee, B., Shamma, D.A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D., Li, L.J.: The new data and new challenges in multimedia research. arXiv preprint arXiv:1503.01817 (2015) 10
31. Tolias, G., Avrithis, Y., Jégou, H.: Image search with selective match kernels: Aggregation across single and multiple images. International Journal of Computer Vision pp. 1–15 (2015) 1, 3
32. Wang, J., Shen, H.T., Yan, S., Yu, N., Li, S., Wang, J.: Optimized distances for binary code ranking. In: Proceedings of the ACM International Conference on Multimedia. pp. 517–526. ACM (2014) 3
33. Wang, J., Kumar, S., Chang, S.F.: Sequential projection learning for hashing with compact codes. In: ICML (2010) 3, 10, 11, 12
34. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS (2008) 11, 12
35. Xu, B., Bu, J., Chen, C., Cai, D., He, X., Liu, W., Luo, J.: Efficient manifold ranking for image retrieval. In: SIGIR (2011) 3
36. Zhang, L., Zhang, Y., Tang, J., Lu, K., Tian, Q.: Binary code ranking with weighted hamming distance. In: CVPR (2013) 3
37. Zheng, Y., Zhao, M., Song, Y., Adam, H., Buddemeier, U., Bissacco, A., Brucher, F., Chua, T.S., Neven, H.: Tour the world: Building a web-scale landmark recognition engine. In: CVPR (2009) 1
38. Zhu, C.Z., Huang, Y.H., Satoh, S.: Multi-image aggregation for better visual object retrieval. In: ICASSP (2014) 3