This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Web Scale Photo Hash Clustering on A Single Machine

Yunchao Gong, Marcin Pawlowski, Fei Yang, Louis Brandy, Lubomir Boundev, Rob Fergus
Facebook

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.

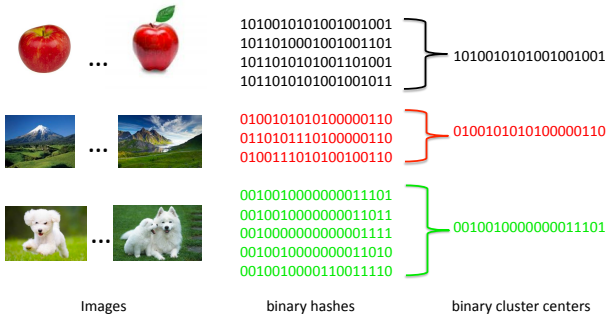


Figure 1. The problem setting of this paper. We are interested in clustering a large amount of image hash codes into compact binary centers.

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

hash table [31] to index the centers so that the nearest center lookup becomes extremely efficient. This can reduce the time complexity of the traditional $k$means from $O(nk)$ to $O(n)$, assuming we have n data points and k centers. This also significantly reduces the storage space of the centers, which we are representing as compact binary codes. In addition to the batch clustering method, to handle online photo streaming, we also propose a simple online version of the method, which is capable of handling hundreds of millions of photos a day. Based on the online clustering method, we show two applications of spam detection and trending cluster detection.

In our experiments, we evaluated the proposed method on the ILSVRC dataset [7] as well as several large Internet datasets. On ILSVRC, we have found that our method usually produces clusters whose quality is only slightly worse (1-2% in terms of purity) than k-means but with a much faster clustering speed. It usually improves the speed for 10-100 times, depending on the dataset size and cluster number. We also tested the online clustering method on two large Internet photo stream datasets (flickr 100M dataset [39] and a proprietary large photo stream dataset) and demonstrated two applications for spam detection and finding trending image clusters. We have found the proposed method is efficient, can handle hundreds of millions photos a day, and can produce high quality image clusters.

2. Related Work

Image clustering is one of the fundamental research areas in computer vision research. It has been widely used in many different vision applications such as automatic discovery of object categories [17, 24], finding trending or iconic photos [33, 4, 10], reconstructing story lines from photo streams [19, 18, 20], and 3D reconstruction from photo collections [37, 1, 10].

One important application of image clustering in recent years is to automatically organize Internet photo collections. Many very large-scale datasets have been proposed to better study the problem of Internet computer vision. For example, ImageNet dataset [7] contains more than 12 million images and 22K classes, and all the images in it have been manually annotated with ground truth labels. The 80 million tiny image dataset [40, 41], which contains 80 million images with noisy labels, is another example. A considerable amount of work has been devoted to how to better organize such huge Internet photo collections. For example, [9] proposed a very efficient algorithm to perform semi-supervised learning on 80 million large photo collections. The work by Liu et al. [25] proposed to use mapreduce to cluster billions of photos offline. Chun et al. [6] applied min-hash to perform large-scale clustering to find near duplicate images. Image clustering has been applied to discover iconic views of famous landmarks and reconstructing 3D models for these landmarks [1, 10]. In particular, the work from [10] used k-medoids to discover iconic image clusters from millions of photos and reconstruct 3D models from the clusters. In addition, parallel kd-tree based k-means [32] have been developed to cluster a large number of local image patches.

Deep learning methods have been extremely successful for visual recognition and retrieval in recent years. In particular, training deep convolutional neural networks on large Internet collections has achieved great success [21, 23, 36]. Also, the activations from hidden layers have also been shown to have very good generalization ability as a generic image feature. For example, the activations have been successfully applied to outdoor scene recognition [8, 38, 34, 45], indoor scene recognition [13, 34], and image retrieval [13, 34]. In this work, we also use such deep activation features as our basic image feature and build our system on top of it. In particular, for large photo collections, we directly hash these features when they are computed and only store the hashes, which are compact representations of the images.

There has been considerable amount of work on learning similarity preserving binary hash codes for Internet images. By representing the images as short binary codes, we are able to store hundreds of millions or even billions of image data points in the RAM of one single machine. In addition, distance computation between these hash codes can be done using fast Hamming distance. This can significantly improve the scalability of analyzing image collections. Many such hash algorithms have been proposed, such as spectral hashing [44], iterative quantization [12], minimal loss hashing [30], graph hashing [26, 27], and semi-supervised hashing [42]. Also, quantization methods such as product quantization [16], optimized product quantization [11], k-means hashing [14], and Cartesian k-means [29] have been proposed to better quantize the data into codes for fast distance computation. Based on the binary hash codes, one significant advantage is that by building multi-index hash tables [31] on top of the codes, we are able to perform nearest neighbor lookup in constant time . However, in almost all these works, the learned hash codes are only applied to image retrieval and has not been applied to speedup other learning algorithms. In this work, we build our clustering method directly on top of the hashes and try to improve the speed of the clustering algorithms by exploring the special properties of hash codes.

Online clustering [3, 5, 47] is another important problem that has been extensively studied in machine learning. Usually, it assumes one has very large data stream, and is unable to perform batch processing to such streams. For example, [2] have proposed online k-means clustering algorithms to cluster large data streams or very large datasets that cannot easily fit into memory. However, their main concern is

how to process huge datasets (i.e. n is large), but they did not consider the scenario when k is large. When both n and k are large, these algorithms are still computationally very inefficient and cannot be easily applied. Also, different from our online clustering setting that has an increasing number of clusters, their online clustering has a fixed number of clusters. Several other works have investigated parallel clustering algorithms, such as parallel k-means [46]. These methods mostly rely on using multiple machines to speedup batch clustering.

3. Fast Clustering on Binary Hash Codes

We introduce the methods for batch clustering on hashes in this section and will introduce online clustering later in Section 5. First, we show that traditional k-means and k-medoids clustering methods can be adapted to this case to cluster hashes efficiently. Then we introduce the Binary k-means (Bk-means) algorithm. We assume we have a set of binary data points $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \in \{-1, +1\}$ and want to cluster the data points to a set of k centers $C = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_k\}$.

3.1. k-means on Hashes

We first briefly discuss how to perform standard k-means clustering on hashes. The hash codes use a binary representation and lie on the vertices of the hypercube. Every hash corresponds to one binary vector[1] with values $+1$ and -1. Thus it is natural to perform k-means clustering on these vectors by treating them as floating binary vectors. In practice, we are not able to store large amount of such floating vectors in memory but can only store the compact hashes. Thus, we need to perform a decoding during the k-means algorithm. We first construct a lookup table from compact hash keys to floating vectors, and during the optimization of k-means, we directly lookup the floating vectors for the hash keys and perform updates. This method produces exactly the same result as standard k-means and has the same running time.

We can also use kd-tree [32, 28] to index the real valued centers and perform fast lookup to speedup the clustering process. However, note that the cluster centers may not lie on vertices of the hypercube, so they are not binary. This requires us to store all the floating value centers in the memory, which might take a huge amount of storage when the number of centers is large. Also, we will show that the binary k-means method produces better clustering accuracy with faster speed than kd-tree based k-means. We will refer to the kd-tree version of k-means as KDk-means.

[1]For binary codes, since we use Hamming diastase, we can represent them as $0/1$ or $1/-1$ interchangeably.

3.2. k-medoids on Hashes

Another approach to explore the special properties of hashes to speedup clustering is applying k-medoids clustering, as described in [10]. Specifically, k-medoids directly picks original data points as centers, which guarantees that the centers are binary. This enables us to use fast distance computation to find the nearest center for every data point during optimization. In [10], they proposed to directly apply Hamming distance to compute the distance between every point to the centers.

Instead, since the centers are binary, we can directly build a multi-index hash table [31] on the centers, so we can perform 1NN lookup very efficiently in constant time. When the nearest centers for every data point have been found, we can then apply the standard k-medoids step within each cluster to find its center point. Specifically, we find each cluster's center by finding the data point having average distance to all the other points in the cluster that is minimal. However, this step is computationally inefficient, especially when there are many points in one cluster. The time complexity of the k-medoids method described in [10] is $O(nk + n^2/k)$. By using the hash lookup for finding the nearest center, we are able to reduce it to $O(n+n^2/k)$. Notice that we will have a quadratic term on n for this method, which is not favorable for large datasets.

3.3. Binary k-means on Hashes

We introduce a fast clustering algorithm on binary codes in this section. For standard k-means, the computationally most inefficient step is finding the nearest center for each point in the dataset. This step needs to compute the distance between every point to all the centers and pick the closest center. The time complexity of this step is $O(nk)$. This will become extremely inefficient when n and k are both large. However, if we can build a hash table to index all the centers, it makes it possible to efficiently find the nearest center for any point in $O(1)$, ideally. This can potentially reduce the time complexity of k-means to $O(n)$.

To enable efficient lookup of the nearest center, we use a constrained k-means formulation that constrains the mean to be binary. Given the means are binary, we can directly build a multi-index hash table [31] on the centers, and can efficiently find the *exact* nearest mean for any binary data point in constant time. It also significantly saves storage of the centers. We can have the following objective function:

$$\begin{aligned} &\min_{\boldsymbol{c}_j} \textstyle\sum_i^n \sum_j^k \|\boldsymbol{x}_i - \boldsymbol{c}_j\|_2^2 \\ &\text{s.t.} \quad \boldsymbol{c}_j \in \{-1, +1\}. \end{aligned} \tag{1}$$

The main difference between this formation to the standard k-means is that we have added a binary constrains to the centers $\boldsymbol{c}$. This makes both the data points $\boldsymbol{x}$ and the means

Algorithm 1: Binary k-means (Bk-means)

Data: $X = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ and k
Randomly initialize centers as $C = \{\boldsymbol{c}_i, \dots, \boldsymbol{c}_k\}$ where $\boldsymbol{c}_i$ are randomly selected from X.
for $i = 1 \dots iter$
  // Step 1: Update nearest center index.
  Build a multi-index hash table M on C.
  for $j = 1 \dots n$
    idx = M.Lookup1NN(x_j).
  endfor
  // Step 2: Update cluster centers.
  for $j = 1 \dots k$
    Compute mean $\boldsymbol{m}_j$ for points in each cluster.
  endfor
  Update all $\boldsymbol{c}_j$ as $\boldsymbol{c}_j = \text{sign}(\boldsymbol{m}_j)$.
endfor

$\boldsymbol{c}$ binary, which enables us to perform fast hashing operations.

The question remained is how to solve Eq. (1) with integer constrains on $\boldsymbol{c}$. We show below that it can be similarly solved by an EM style alternative optimization algorithm. Assuming $\boldsymbol{c}_j$ has already been computed, the problem is reduced to the assignment step of k-means, which can be easily accomplished by assigning each point $\boldsymbol{x}_i$ to its nearest center $\boldsymbol{c}_j$. This can be done by building a multi-index hash table [31] on $\boldsymbol{c}_j$ and perform fast lookups for each $\boldsymbol{x}_i$. When all the points have been assigned to its nearest center, the problem is how to optimize $\boldsymbol{c}_j$ with respect to the binary constrainss. By expanding Eq. (1), and only consider one cluster $\boldsymbol{c}_j$ and p points belonging to it, we have

$$\begin{aligned} & \min_{\boldsymbol{c}_j} \textstyle\sum_i^p \|\boldsymbol{x}_i - \boldsymbol{c}_j\|_2^2 \\ = \quad & \textstyle\sum_i^p \|\boldsymbol{x}_i\|_2^2 + \sum_i^p \|\boldsymbol{c}_j\|_2^2 - \sum_i^p \boldsymbol{x}_i \boldsymbol{c}_j^T. \end{aligned} \tag{2}$$

We notice that $\sum_i^p \|\boldsymbol{x}_i\|_2^2$ and $\sum_i^p \|\boldsymbol{c}_j\|_2^2$ are both constants, because they are both binary variables. Thus the optimization problem can be reduced to:

$$\begin{aligned} & \max_{\boldsymbol{c}_j} \textstyle\sum_i^p \boldsymbol{x}_i \boldsymbol{c}_j^T = \max_{\boldsymbol{c}_j} (\sum_i^p \boldsymbol{x}_i) \boldsymbol{c}_j^T \\ & \text{s.t.} \quad \boldsymbol{c}_j \in \{-1, +1\}. \end{aligned} \tag{3}$$

The above problem can be solved by first computing the sum of all $\boldsymbol{x}_i$ as $\boldsymbol{m}_j = \sum_i^p \boldsymbol{x}_i$, and $\boldsymbol{c}_j$ can be obtained by:

$$\boldsymbol{c}_{jk} = \text{sign}(\boldsymbol{m}_{jk}) = \begin{cases} +1 & \text{if } \boldsymbol{m}_{jk} \geq 0 \\ -1 & \text{if } \boldsymbol{m}_{jk} < 0. \end{cases} \tag{4}$$

This gives an alternative optimization algorithm for solving Eq. (1), which iteratively solves two subproblems. The solution of each sub-problem is exact and optimal, which guarantees not to increase the objective function value. Thus with a finite number of iterations, this optimization problem will converge to a local minimum or a stationary point. The full algorithm is described in Algorithm 1, which we refer to as Bk-means. It is interesting to mention that this method, is indeed the same as running a k-median clustering [15] on binary data ($+1/-1$). This is because when we run k-median on binary data, computing the median is equivalent to first computing mean and taking sign of it. In particular, k-median is optimizing L1-norm, which is more robust than L2-norm, suggesting our method might be more robust to outliers.

k-means	KDk-means	k-medoids	Bk-means
$O(nk)$	$O(n \log(k))$	$O(n + n^2/k)$	$O(n)$

Table 1. Time complexity comparison between different methods.

3.4. Analysis and Discussion

We discuss the time complexity of different methods, and it is shown in Table 1. First, k-means is probably the slowest method since when both n and k are large, it will be very inefficient, and it cannot take advantage of the fast computation of binary codes. KDk-means is usually much faster than k-means, since we used kd-tree to index the centers. k-medoids are also more efficient than k-means, because when the centers are also binary, we can build hash tables on top of them and perform fast lookup very efficiently. However, the main limitation of k-medoids is that it is very inefficient to find the medoids within each cluster. Thus, when k is large, the cost is affordable, and when k is small, the cost becomes extremely inefficient. The proposed Bk-means, which has linear time complexity with the number of data points n, is the most efficient method. This is under the assumption that the hash lookup is constant time.

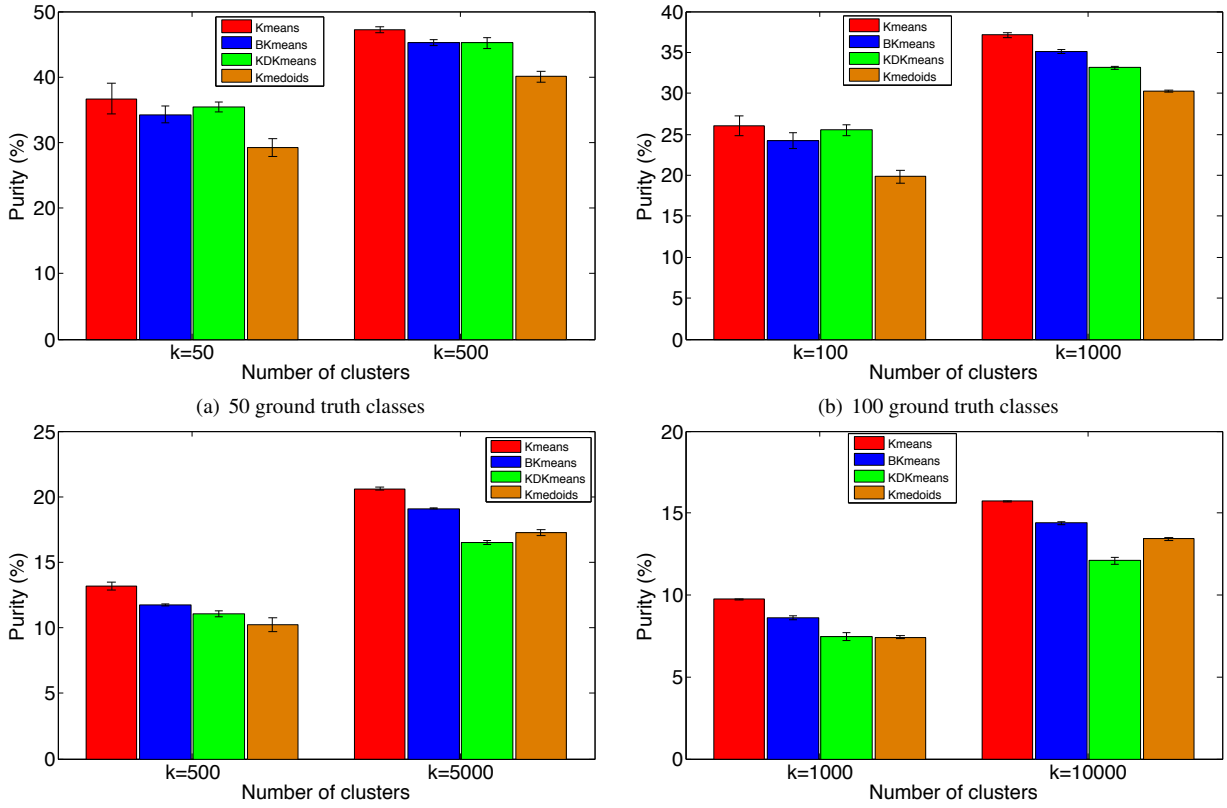
In terms of space complexity, k-means and KDk-means both need to store the floating valued centers in memory, which is prohibitive when the number of centers becomes large. k-medoids and Bk-means make the centers binary, which makes storing the centers much more compact. We will show quantitative comparison of memory consumption in Table 3.

4. Experiments

4.1. Datasets, Features, and Protocols

We perform quantitative experiments on the ILSVRC2012 dataset, which is a subset of ImageNet [7]. It contains 1000 object categories and more than 1.2 million images. Each category roughly has more than 1000 images. We randomly pick 50, 100, 500, and 1000 classes from all 1000 classes, and construct four different datasets.

To represent the images, we use the activations from a deep neural network as the image features. The network is trained using a similar architecture as the Krizhevsky et al. [22]. We extract the 4096 dimensional activations from the last hidden layer as our image feature and normalize them



(a) 50 ground truth classes

(b) 100 ground truth classes

(c) 500 ground truth classes

(d) 1000 ground truth classes

Figure 2. Comparison of clustering purity on subsets of ILSVRC2012 dataset. Each subset contains different number of classes.

to have unit norm. Then we apply the Iterative Quantization (ITQ) method [12] to quantize the features into 256-bit binary codes. These binary codes are the input of all the algorithms studied below.

We use purity to evaluate the clustering algorithms. Given a dataset with c ground truth classes, we perform clustering algorithms to cluster the images into k clusters. Each cluster is assigned to the class, which is most frequent, and then counting the number of correctly assigned documents and dividing by the number of points in this cluster to measure the accuracy of this assignment. Then mean purity is computed by averaging the purities for all clusters. In our experiments, given c ground truth classes, we will cluster them into $k = c, 10c$ centers and report purity for each of them. We run different methods for three random trials and report the mean purity and standard deviation. For all our implementations, we have heavily used multithreading (16 cores) to speedup the computations. For all methods compared here, we set the number of iterations to 5. For KDk-means, we use the FLANN implementation [28], and tuned the tree to have roughly the same running time as hashing. We fixed the number of backtracking to 32, which means each tree will check at most 32 different leaf nodes.

4.2. Clustering Accuracy

We first evaluate the clustering performance using purity on various datasets, and the results are shown in Figure 2. We can find k-means always performs the best, as most other methods are approximating it or introduce certain constrainss. The Bk-means method is usually slightly (1-2%) worse than k-means. KDk-means works well for small number of centers, while its performance becomes worse when the dataset size and number of centers become large. Overall, Bk-means is still better than KDk-means, and we will show later that it is also faster. k-medoids works reasonably well, but is always worse than Bk-means. This is probably because Bk-means is not restricted to using data points as centers and allows more degrees of freedom. Some sample image clusters on ILSVRC are shown in Figure 3.

4.3. Clustering Speed

We report the running time for different clustering methods on different dataset sizes in Table 2. All methods have achieved a faster speed than k-means. Overall, Bk-means and k-medoids are fastest and are slightly faster than KDk-means. In particular, Bk-means method has achieved a significant speedup over k-means. For relatively small datasets, such as those with 50 classes and 50 clusters,

method / c/k	50/50	50/500	100/100	100/1000	500/500	500/5000	1000/1000	1000/10000
k-means	3.6	18.0	11.4	64.6	188.5	1492.9	640.9	5721.1
Bk-means	4.0	4.1	8.3	8.4	41.4	41.8	85.4	86.9
k-medoids	5.8	3.5	12.6	7.1	71.7	37.1	149.6	75.6
KDk-means	4.5	4.8	9.5	9.6	48.7	49.2	95.8	96.3

Table 2. Timing (seconds) for clustering (one iteration) on subset of ILSVRC with different number of classes c and clusters k.

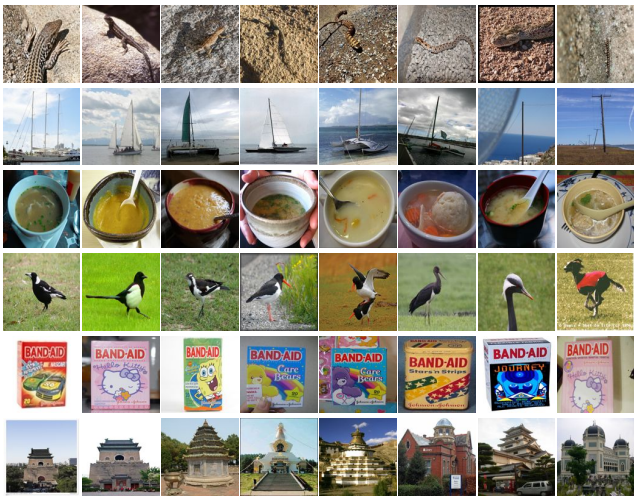


Figure 3. Sample clusters from the ILSVRC2012 dataset. Each line shows images in one cluster.

method / center size	10000	1M	100M
k-means / kd-kmeans	9.7	976	97600
Bk-means / k-medoids	0.31	30.5	3000.5

Table 3. Memory consumption (Mbytes) for storing centers for different methods, assuming 256-bit hash code.

Bk-means achieved comparable speed to k-means. This is mainly because the number of centers is small, which limits the speedup factor of hashing. When we use more centers, the speedup becomes more significant. For example, with 1000 classes and 10,000 centers, we are able to achieve about 60-70 times speedup. We have also tested k-means and bk-measuring 20M images and 1M centers, and bk-means can achieve over 400 times speedup than k-means (5.5 hours vs. 2444 hours for one iteration).

In terms of memory consumption, the proposed hashing approach and k-medoids always *saves 32 times* more storage than k-means or kdtree based methods. This is because they can directly work with binary centers, while others need to store the floating valued cluster centers. This is especially important when we work with millions of even more centers. We show quantitative comparison in Table 3.

5. Online Clustering of Large Photo Streams

In this section, we introduce an online clustering algorithm that performs clustering on large photo stream. This has real world applications since photos are always uploaded as a stream. Our work is similar to previous works on online k-means clustering, but the main difference is we do not know the number of clusters and the number of clusters will continue to increase.

We describe the online clustering algorithm in Algorithm 2. The algorithm works as follows. We have an online photo stream coming in every second, and we use a fixed size Least Recently Used (LRU) cache to store the data. When the size of the cache grows to n, we run our Bk-means clustering method on this batch of data $\hat{X}$and cluster them into $n/10$ centers. Then we filter out images that are not visually consistent in each cluster. For every point in each cluster, we filter out images that having distance r away from its center. After filtering, all the clusters smaller than size t will be removed. In the meantime of clustering, the LRU cache collects another set of data, and when the clustering is done, we perform clustering to the new batch. For this new batch, we first assign its points to all the existing clusters when they are within r Hamming distance to the cluster centers. Then we run Bk-means to the rest and repeat the above steps. In the following experiments, we will set $n = 1000000$, $t = 10$, and r depends on particular application. This online clustering setting makes the need of compact cluster centers a even more desirable property, as the number of centers will grow significantly after running it for years. It enables us to process large online photo streams, and we will show two applications for it. Clustering 1 million images into 100,000 clusters usually only takes 4 minutes. This system enables us to cluster about 360 million photos in a day on a single machine.

5.1. Detecting Spam Photo Clusters

One direct application of this online photo stream clustering algorithm is spam photo detection. Spam photo sare usually near duplicate to each other. Once we have clustered these near-duplicate images into one cluster, as long as one of them is detected as spam, we can find the whole cluster of images. For this particular case, since we are interested in finding near duplicate images, we set the radius threshold for each cluster to a small value $r = 7$. We have found this produced reasonably good clusters in which images are

Algorithm 2: Online Binary k-means

Initialize centers as $C = \{\}$.
for every data batch $i = 1 \ldots$
  // Step 1: Update nearest center index.
  Build a multi-index hash table M on C.
  // Step 2: Assign points to nearest center.
  for $j = 1 \ldots n$
    z = M.Lookup1NN($\boldsymbol{x}_j$).
    if ($\|\boldsymbol{x}_j - \boldsymbol{c}_z\|_H < r$)
      Assign $\boldsymbol{x}_j$ to $\boldsymbol{c}_z$.
    endif
  endfor
  // Step 3: Update cluster centers.
  for $j = 1 \ldots k$
    Compute mean $\boldsymbol{m}_j$ for points in each cluster.
  endfor
  Update all $\boldsymbol{c}_j$ as $\boldsymbol{c}_j = \text{sign}(\boldsymbol{m}_j)$.
  // Step 4: Perform batch Bk-means to rest points not assigned.
  Get new clusters C_n using Bk-means for points not assigned.
  Prune each new cluster based on its radius and size.
  // Step 5: Merge clusters.
  $C = \{C, C_n\}$
endfor

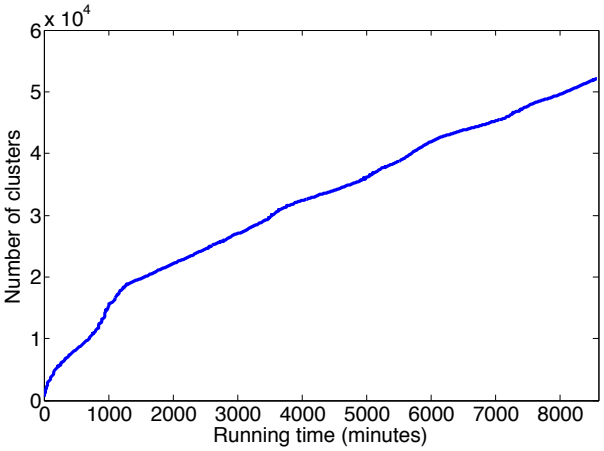


Figure 4. Number of clusters with respect to running time.

very similar but not identical.

We run this algorithm on a large photo stream with hundreds of millions of photos every day. We show the number of discovered clusters over 6 days in Figure 4. By running the system over 6 days, it has scanned over 2 billion photos. We can clearly find that when we continue aggregating new clusters, the growth speed becomes slightly slower. It is also clear that we will accumulate a very large number of centers after running this method for years. This emphasizes the importance of having binary hashes as centers. We also show some sample clusters we have discovered during the process in Figure 5. This clearly shows we are able to find very consistent near-duplicate clusters, but images in it are not identical.

Figure 5. Some near-duplicate clusters we have found. Each line corresponds to one cluster. We show a few examples in each cluster.

5.2. Online Semantic Clustering

The second problem we consider is finding trending photo clusters or events from online photo streams. Our online clustering method fits naturally to this problem, since all the new images will first match against existing old clusters, and then we cluster images that do not have a match. Thus, whatever new clusters we have found do not comply with the old distribution of the clusters and can be considered as trending new clusters.

To study this application, we used 10 million images from Flickr [39][2] and sorted them by their upload time. Then we split these images into 10 splits, and each split contains 1 million images. We ran our method on these 10 splits, and show some example clusters in Figure 6 and Figure 7. For Figure 6, we selected clusters that are found in very early batches. We can find most clusters contain regular objects, and they are the non-trending clusters. In Figure 7, we show some clusters our method found in later batches, which clearly corresponds to some trending events. For example, one photo cluster found on July 4th, 2013, directly corresponds to the fireworks on Independence day. Please note that most photos do not belong to the same person. Our current results indicate that we are able to find trending events using this online clustering method. However, it is still preliminary, and in the future, we will incorporate other information such as location and other text meta data to improve the results.

6. Discussions

This paper studies very large-scale photo clustering using hashes. In particular, we propose to cluster image binary codes into a set of binary cluster centers so we can perform clustering efficiently and store and index huge number of cluster centers easily. The proposed binary k-means algo-

[2]They are public photo albums uploaded to Flickr, and all photos have Creative Commons Attributions License.

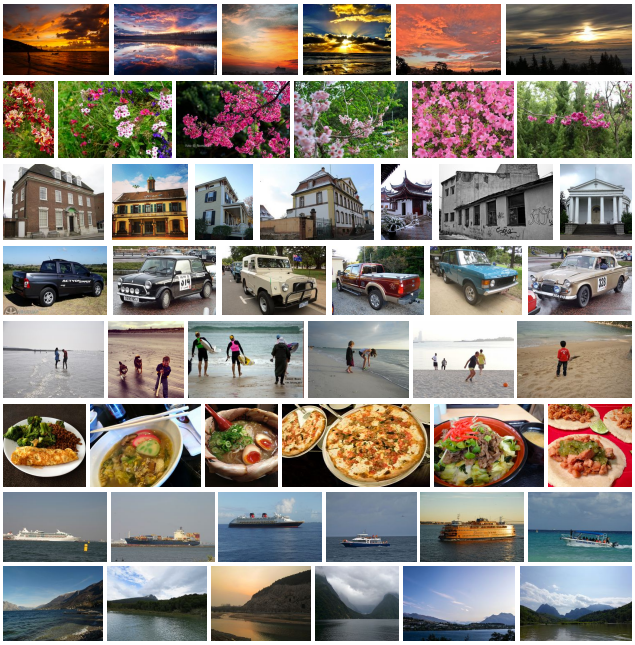


Figure 6. Example clusters (one cluster per line) that are found in older batches. These clusters are very common in the dataset.

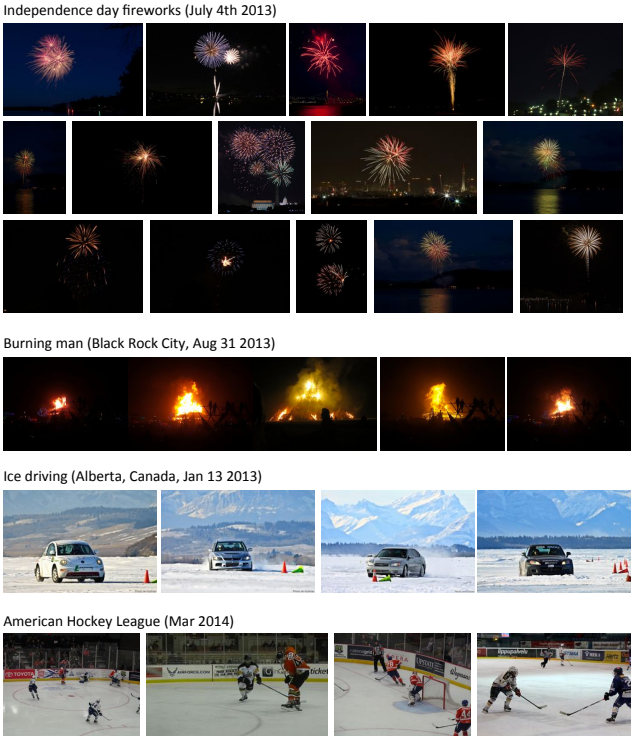


Figure 7. Example trending clusters our method found.

rithm is able to perform clustering in linear time $O(n)$, and significantly improved the scalability of the clustering process. This makes clustering large photo collections on a single machine possible.

We have also proposed an online binary k-means method to handle hundreds of millions of online photo uploads every day. We show that this method can handle more than 300 million photos in a day, on a single machine. We show two applications of the online clustering algorithm: finding near duplicate photos for spam detection, and finding trending photo clusters from photo streams.

References

[1] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building rome in a day. In *Commun. ACM*. 2011.

[2] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *IJCNN*. 2009.

[3] W. Barbakh and C. Fyfe. Online clustering algorithms. In *JMLR Workshops*. 2008.

[4] T. Berg and A. Berg. Finding iconic images. In *CVPR Workshops*. 2009.

[5] A. Choromanska and C. Monteleoni. Online clustering with experts. In *JMLR Workshops*. 2005.

[6] O. Chum and J. Matas. Web scale image clustering–large scale discovery of spatially related images. *Research Report. Czech Technical University*, 2008.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. *CVPR*, 2009.

[8] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[9] R. Fergus, A. Torralba, and Y. Weiss. Semi-supervised learning in gigantic image collections. *NIPS*, 2009.

[10] J.-M. Frahm, P. Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, and M. Pollefeys. Building Rome on a cloudless day. In *ECCV*, 2010.

[11] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *PAMI*, 2014.

[12] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A Procrustean approach to learning binary codes for large-scale image retrieval. *PAMI*, 2012.

[13] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *ECCV*. 2014.

[14] K. He, F. Wen, and J. Sun. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. *CVPR*, 2013.

[15] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[16] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 2011.

[17] G. Kim, C. Faloutsos, and M. Hebert. Unsupervised modeling of object categories using link analysis techniques. In *CVPR*. 2008.

[18] G. Kim and E. P. Xing. Joint summarization of large-scale collections of web images and videos for storyline reconstruction. In *CVPR*. 2014.

[19] G. Kim and E. P. Xing. Reconstructing storyline graphs for image recommendation from web community photos. In *CVPR*. 2014.

[20] G. Kim and E. P. Xing. Visualizing brand associations from web community photos. In *WSDM*. 2014.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 2012.

[23] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[24] Y. Lee and K. Grauman. Shape discovery from unlabeled image collections. In *CVPR*. 2009.

[25] T. Liu, C. Rosenberg, and H. R. and. Clustering billions of images with large scale nearest neighbor search. In *WACV)*, 2007.

[26] W. Liu, S. Kumar, and S.-F. Chang. Hashing with graphs. *ICML*, 2011.

[27] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. *CVPR*, 2012.

[28] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[29] M. Norouzi and D. Fleet. Cartesian $k$means. *CVPR*, 2013.

[30] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. *ICML*, 2011.

[31] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*. 2012.

[32] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[33] R. Raguram and S. Lazebnik. Computing iconic summaries for general visual concepts. In *CVPR Workshops*. 2008.

[34] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[35] H. Senuma. K-means clustering with feature hashing. In *ACL (Student Session)*, 2011.

[36] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[37] I. Simon, N. Snavely, and S. M. Seitz. Scene summarization for online image collections. In *ICCV*. 2007.

[38] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep fisher networks for large-scale image classification. In *Advances in Neural Information Processing Systems*, pages 163–171, 2013.

[39] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.

[40] A. Torralba, R. Fergus, and W. Freenman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *PAMI*, 2008.

[41] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. *CVPR*, 2008.

[42] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. *CVPR*, 2010.

[43] X.-J. Wang, L. Zhang, and C. Liu. Duplicate discovery on 2 billion internet images. In *CVPR*, 2012.

[44] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *NIPS*, 2008.

[45] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. *arXiv preprint arXiv:1311.2901*, 2013.

[46] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on mapreduce. In *Cloud Computing*. 2009.

[47] S. Zhong. Efficient online spherical k-means clustering. In *IJCNN*. 2005.