Online Product Quantization

Donna Xu, Ivor W. Tsang, and Ying Zhang

Abstract—Approximate nearest neighbor (ANN) search has achieved great success in many tasks. However, existing popular methods for ANN search, such as hashing and quantization methods, are designed for static databases only. They cannot handle well the database with data distribution evolving dynamically, due to the high computational effort for retraining the model based on the new database. In this paper, we address the problem by developing an online product quantization (online PQ) model and incrementally updating the quantization codebook that accommodates to the incoming streaming data. Moreover, to further alleviate the issue of large scale computation for the online PQ update, we design two budget constraints for the model to update partial PQ codebook instead of all. We derive a loss bound which guarantees the performance of our online PQ model. Furthermore, we develop an online PQ model over a sliding window with both data insertion and deletion supported, to reflect the real-time behaviour of the data. The experiments demonstrate that our online PQ model is both time-efficient and effective for ANN search in dynamic large scale databases compared with baseline methods and the idea of partial PQ codebook update further reduces the update cost.

Index Terms—Online indexing model, product quantization, nearest neighbour search.

1 Introduction

Approximate nearest neighbor (ANN) search in a static database has achieved great success in supporting many tasks, such as information retrieval, classification and object detection. However, due to the massive amount of data generation at an unprecedented rate daily in the era of big data, databases are dynamically growing with data distribution evolving over time, and existing ANN search methods would achieve unsatisfactory performance without new data incorporated in their models. In addition, it is impractical for these methods to retrain the model from scratch for the continuously changing database due to the large scale computational time and memory. Therefore, it is increasingly important to handle ANN search in a dynamic database environment.

ANN search in a dynamic database has a widespread applications in the real world. For example, a large number of news articles are generated and updated on hourly/daily basis, so a news searching system [1] requires to support news topic tracking and retrieval in a frequently changing news database. For object detection in video surveillance [2], video data is continuously recorded, so that the distances between/among similar or dissimilar objects are continuously changing. For image retrieval in dynamic databases [3], relevant images are retrieved from a constantly changing image collection, and the retrieved images could therefore be different over time given the same image query. In such an environment, real-time query needs to be answered based on all the data collected to the database so far.

In recent years, there has been an increasing concern over the computational cost and memory requirement dealing with continuously growing large scale databases, and therefore there are many online learning algorithm works [4], [5], [6] proposed to update the model each time streaming data coming in. Therefore, we consider the following problem. Given a dynamic database environment, develop an online learning model accommodating the new streaming data with low computational cost for ANN search.

Recently, several studies on online hashing [7], [8], [9], [10], [11], [12], [13] show that hashing based ANN approaches can be adapted to the dynamic database environment by updating hash functions accommodating new streaming data and then updating the hash codes of the exiting stored data via the new hash functions. Searching is performed in the Hamming space which is efficient and has low computational cost. However, an important problem that these works have not addressed is the computation of the hash code maintenance. To handle the streaming fashion of the data, the hash functions are required to be frequently updated, which will result in constant hash code recomputation of all the existing data in the reference database. This will inevitably incur an increasing amount of update time as the data volume increases. In addition, these online hashing approaches require the system to keep the old data so that the new hash code of the old data can be updated each time, leading to inefficiency in memory and computational load. Therefore, computational complexity and storage cost are still our major concerns in developing an online indexing model.

Product quantization (PQ) [14] is an effective and successful alternative solution for ANN search. PQ partitions the original space into a Cartesian product of low dimensional subspaces and quantizes each subspace into a number of sub-codewords. In this way, PQ is able to produce a large number of codewords with low storage cost and perform ANN search with inexpensive computation. Moreover, it preserves the quantization error and can achieve satisfactory recall performance. Most importantly, unlike hashing-based methods representing each data instance by a hash code, which depends on a set of hash functions, quantization-based methods represent each data instance by an index, which associates with a codeword that is in the same vector space with the data instance. However, PQ is a batch mode method which is not designed for the problem of accommo-

- *D. Xu, I. W. Tsang and Y. Zhang are with the Centre for Artificial Intelligence, FEIT, University of Technology Sydney, NSW, Australia. E-mail: donna.xu@student.uts.edu.au, ivor.tsang@uts.edu.au, ying.zhang@uts.edu.au*

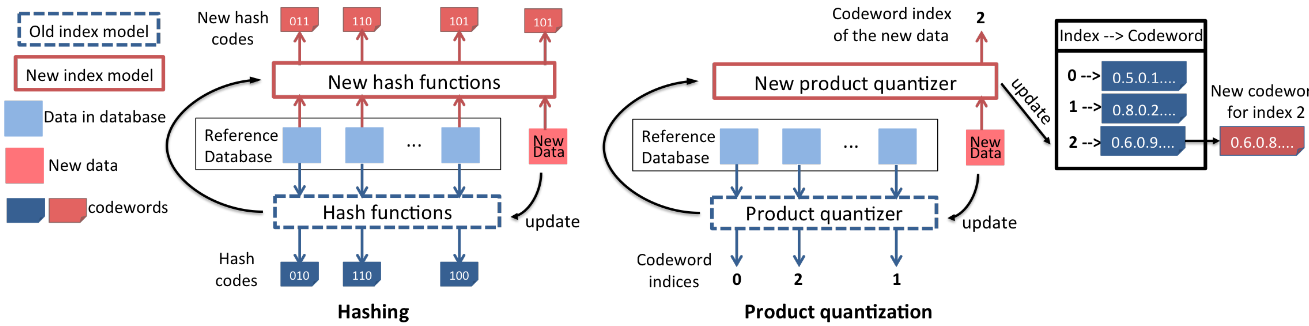


Fig. 1. Hashing vs PQ in online update. The hash codes of the data points in the reference database will get updated if the hash functions get updated by the new data. *The index of the codewords in the PQ codebook, on the other hand, will remain the same even though the codebook gets updated by the new data.* Thus online PQ is able to save severely much time by avoiding codewords maintenance of the reference database. (Best viewed in colors)

dating streaming data in the model. Therefore, to address the problem of handling streaming data for ANN search and tackle the challenge of hash code recomputation, we develop an online PQ approach, which updates the codewords by streaming data without the need to update the indices of the existing data in the reference database, to further alleviate the issue of large scale update computational cost.

Figure 1 compares hashing method and PQ in the code representation and maintenance, which illustrates the advantage of PQ over hashing in computational cost and memory efficiency. Once the index models get updated by the streaming data, the updated hash functions in hashing methods will produce new hash codes for each data point in the reference database, which will incur expensive cost for large scale databases. The updated product quantizer in PQ, on the other hand, updates the codewords in the codebook, but it does not change the index of the updated codewords of each data point in the reference database. To further reduce the update computational cost, we illustrate the idea of partial codebook update and present two budget constraints for the model to update the codebook partially instead of all. Furthermore, we derive a loss bound which guarantees the performance of our online PQ model. Unlike traditional analysis, our model is a non-convex problem with matrices as the variables, so its theoretical analysis is not trivial to be handled. To emphasize the real-time data for querying, we also propose an online PQ model over a sliding window, which support both data insertion and deletion.

2 Related Works

Hashing methods generate a set of hash functions to map a data instance to a hash code in order to facilitate fast nearest neighbor search. Existing hashing methods are grouped in data-independent hashing and data-dependent hashing. One of the most representative work for data-independent hashing is Locality Sensitive Hashing (LSH) [20], where its hashing functions are randomly generated. LSH has the theoretical performance guarantee that similar data instances will be mapped to similar hash codes with a certain probability. Since data-independent hashing methods are independent from the input data, they can be easily adopted in an online fashion. Data-dependent hashing, on the other hand, learns the hash functions from the given data, which can achieve better performance than data-independent hashing methods. Its representative works are Spectral Hashing (SH) [19], which uses spectral method to encode similarity graph of the input into hash functions, and IsoH [18] which finds a rotation matrix for equal variance in the projected dimensions.

To handle nearest neighbor search in a dynamic database, online hashing methods [7], [8], [9], [10], [11], [12], [13] have attracted a great attention in recent years. They allow their models to accommodate to the new data coming sequentially, without retraining all stored data points. Specifically, Online Hashing [7], [8], AdaptHash [11] and Online Supervised Hashing [13] are online supervised hashing methods, requiring label information, which might not be commonly available in many real-world applications. Stream Spectral Binary Coding (SSBC) [9] and Online Sketching Hashing (OSH) [10] are the only two existing online unsupervised hashing methods which do not require labels, where both of them are matrix sketch-based methods to learn to represent the data seen so far by a small sketch. However, all the online hashing methods suffer from the existing data storage and the high computational cost of hash code maintenance on the existing data. Each time new data comes, they update their hash functions accommodating to the new data and then update the hash codes of all stored data according to the new hash functions, which could be very time-consuming for a large scale database.

Multi-codebook quantization (MCQ) methods [14], [21], [22], [23], [24], [25] are derived by minimizing the quantization error between the original input data and their corresponding codewords. Each codeword is represented by a set of sub-codewords selected from multiple codebooks. To the best of our knowledge, no MCQ methods have been explored to an online fasion. However, methods such as Composite Quantization (CQ) [22] and Sparse Composite Quantization (SQ) [23] can not be extended to handle streaming data, because the constant inter-dictionary-element-product in the constraint of CQ and SQ results in non-separable variables during codebook update. Additive Quantization (AQ) [21] is an unconstrained approach, but its lack of constraints leads to a high computational encoding procedure due to its use of heuristic beam search algorithm, which will dominate the update process in a dynamic database environment.

TABLE 1
Comparison between the existing methods and ours

Method	Applicable to streaming data	Preserves the quantization error	Does not require codewords maintenance	Does not require labels	Does not require to keep old data
Supervised data-dependent hashing ([15], [16])	✗	✗	✗	✗	✗
Unsupervised data-dependent hashing ([17], [18], [19])	✗	✗	✗	✓	✗
Data-independent hashing ([20])	✓	✗	✗	✓	✓
Online supervised hashing ([7], [8], [11], [12], [13])	✓	✗	✗	✗	✗
Online unsupervised hashing ([9], [10])	✓	✗	✗	✓	✗
Quantization that can not be adapted to an online setting: AQ, CQ, SQ, TQ ([21], [22], [23], [24])	✗	✓	✗	✓	✗
Quantization that can be adapted to an online setting: PQ, OPQ ([14], [25])	✗	✓	✓	✓	✗
Proposed model	✓	✓	✓	✓	✓

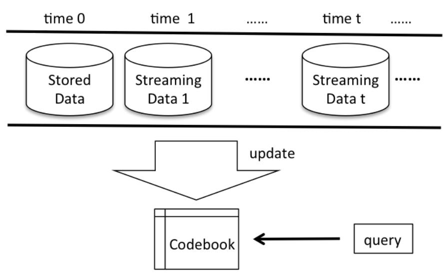


Fig. 2. A general procedure for Online Product Quantization update. At each iteration, the new codebook gets updated by the streaming data. Searching can be performed using the latest codebook.

Tree Quantization (TQ) [24], extended on the AQ work, achieves efficient encoding by adding more constraints to the codebook to form a tree structure. However, it is difficult for TQ to be extended to handle streaming data as the tree graph needs to be updated together with the codebook and the indices of the stored data. Product Quantization (PQ) [14], as one of the most classical MCQ method for fast nearest neighbor search, decomposes the input space into a Cartesian product of subspaces. The codeword of a data instance is represented by the concatenation of the sub-codeword of the data in all subspaces. More specifically, each sub-codeword is represented by an index. Thus the codeword of a data instance is the concatenation of the sub-codeword indices of all subspaces. Assuming that the codebook will not change much with an update by a streaming data, an online PQ model can be proposed. Under this assumption, there is no need of codewords maintenance of the existing data, as the indices of the sub-codewords of the data remain the same even though the actual sub-codewords are updated. It is different from online hashing methods, which requires the hash code update for the existing data each time the hash functions get updated. Extension works of PQ such as Optimized Product Quantization (OPQ) [25] can also be developed to an online fashion. In this paper, we focus on PQ method, so a novel online paradigm for PQ is proposed. The difference between existing methods and ours are summarized in Table 1.

TABLE 2
Summary of Notations

Notation	Definition
$\mathcal{X}$	a set of data points
$\mathcal{Z}$	codebook
$\|\|\cdot\|\|$	l_2-norm
$\|\|\cdot\|\|_F$	Frobenius norm
t	iteration number. $t = 1, 2, ..., T$
M	the number of subspaces (subquantizers)
K	the number of sub-codewords in each subspace
$z^t_{m,k}$	the kth sub-codeword of the mth subspace in iteration t
$\mathcal{C}^t_{m,k}$	the set of vectors assigned to the sub-codeword $z^t_{m,k}$
$n_{m,k}$	the number of vectors assigned to the sub-codeword $z^t_{m,k}$
$\Delta z^t_{m,k}$	the difference of the $z^t_{m,k}$ in two consecutive iterations i.e., $z^t_{m,k} = z^{t-1}_{m,k} + \Delta z^t_{m,k}$
ΔZ^t_m	differences concatenation $[\Delta z^t_{m,1}, ... z^t_{m,K}]$
$\mathbb{1}_{[condition]}$	indicator function with value 1 if the condition is true and 0 otherwise
B	size of the mini-batch
α	the number of subspaces to be updated
λ	the percentage of sub-codewords to be updated
L	size of the sliding window

3 Online Product Quantization with Budget Constraints

The traditional PQ method assumes that the database is static and hence it is not suitable for non-stationary data setting especially when the data is time-varying. Therefore, it is crucial to develop an online version of PQ to deal with dynamic database. A general procedure of our online product quantization framework is illustrated in Figure 2. The codebook at each iteration gets updated by the streaming data without retraining all the collected data. ANN search can be conducted against the latest codebook in terms of user queries. Unlike online hashing methods which update hashing functions and hash codes of the existing data, online PQ updates codebooks only and the codeword index of the existing data remains the same.

3.1 Preliminaries

We first define the vector quantization approach [26] and the concept of quantization error, and then introduce Product quantization. Table 2 summarizes the notations frequently used in this paper.

Definition 1 (Vector quantization [26]). *Vector quantization approach quantizes a vector* $x \in \mathbb{R}^D$ *to its codeword* z_k *in a codebook* $\mathcal{Z} = \{z_k\}$ *where* $k \in \{1...K\}$.

Definition 2 (Quantization error). Given a finite set of data points $\mathcal{X}$, vector quantization aims to minimize the *quantization error* which is defined in the following:

$$\min_{\substack{\mathcal{C}_1,\ldots\mathcal{C}_K\\ z_1,\ldots,z_K}} \sum_{i=1}^{|\mathcal{X}|} \|x^i - z_k\|^2$$

where $\mathcal{C}_k$ represents the set of data points assigned to the codeword z_k.

According to the first Lloyd's condition, x^i should be mapped to its nearest codeword z_k in the codebook. A codeword z_k can be computed as the centroid of the vectors in $\mathcal{C}_k$. All of the codewords form the codebook $\mathcal{Z}$ with size K.

Unlike vector quantization which uses one quantizer to map a vector, product quantization (PQ) uses M sub-quantizers. It represents any $x \in \mathbb{R}^D$ as a concatenation of M sub-vectors $(x_1, \ldots, x_m, \ldots x_M)$ where $x_m \in \mathbb{R}^{D/M}$, assuming that D is a multiple of M for simplicity. The PQ codebook is then composed of M sub-codebooks and each of the sub-codebook contains K sub-codewords quantized from a distinct subquantizer. Any codeword belongs to the Cartesian product of the sub-codewords in each sub-codebook. The codeword of x is constructed by the concatenation of M sub-codewords $z = [z_{1,k1}, \ldots z_{m,km}, \ldots z_{M,kM}]$, where $z_{m,km}$ is the sub-codeword of x_m.

3.2 Online Product Quantization

Inspired by product quantization and online learning, the objective function of the online product quantization at each iteration t is shown in the following:

$$\min_{\substack{\mathcal{C}^t_{1,1},\ldots,\mathcal{C}^t_{m,k},\ldots,\mathcal{C}^t_{M,K}\\ z^t_{1,1},\ldots,z^t_{m,k},\ldots,z^t_{M,K}}} = \sum_{m=1}^{M} \|x^t_m - z^t_{m,k}\|^2 \tag{1}$$

where x^t_m is the streaming data in the mth subspace in the tth iteration and its nearest sub-codeword is $z^t_{m,k}$. We expect to minimize the quantization error of the data at the current iteration t. Inspired by sequential vector quantization algorithm [27] to update the codebook, the solution of online PQ is shown in Algorithm 1:

3.3 Mini-batch Extension

In addition to processing one streaming data at a time, our framework can also handle a mini-batch of data at a time. In the case of processing mini-batch of data, we assume that each time we get a new batch of data points $X^t \in \mathbb{R}^{B\times D}$ where B is the size of the mini-batch. Its objective function is stated as the following:

$$\min_{\substack{\mathcal{C}^t_{1,1},\ldots,\mathcal{C}^t_{m,k},\ldots,\mathcal{C}^t_{M,K}\\ z^t_{1,1},\ldots,z^t_{m,k},\ldots,z^t_{M,K}}} = \sum_{m=1}^{M}\sum_{i=1}^{B} \|x^{t,i}_m - z^t_{m,k}\|^2 \tag{2}$$

where $x^{t,i}_m$ is the ith streaming data of the current mini-batch in the mth subspace at the tth iteration and its nearest sub-codeword is $z^t_{m,k}$.

Follow the aforementioned algorithm, we determine the sub-codeword for each sub-vector in each subspace for

Algorithm 1 Online Product Quantization Solutions for Eq.(1)

initialize PQ with the $M * K$ sub-codewords $z_{1,1}, \ldots, z_{m,k}, \ldots z_{M,K}$ using a initial set of data
create counters $n_{1,1}, \ldots, n_{m,k}, \ldots n_{M,K}$ for each sub-codeword and initialize each $n_{m,k}$ to be the number of initial data points assigned to the corresponding $z_{m,k}$
for $t = 1, 2, 3, \ldots$ **do**
  get a new data x^t
  partition x^t into M subspaces $(x^t_1, \ldots x^t_M)$
  in each subspace, determine and assign the nearest sub-codeword $z^t_{m,k}$ for each sub-vector x^t_m
  update the number of points for each sub-codeword: $n_{m,k} \leftarrow n_{m,k} + 1\ \forall m \in \{1, \ldots M\}$
  update the sub-codeword: $z^{t+1}_{m,k} \leftarrow z^t_{m,k} + \frac{1}{n_{m,k}}(x^t_m - z^t_{m,k})\ \forall m \in \{1, \ldots M\}$
end for

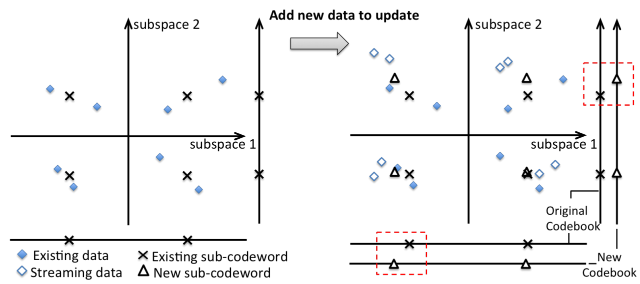


Fig. 3. A schematic figure of online product quantization with budget constraints. There are two subspaces where each subspace has two sub-codewords. After the codebook adapting to the new data, two of the four sub-codewords get hugely changed (highlighted in a red dashed rectangle) and the rest two sub-codewords barely changed.

all data in the mini-batch, and update the counters accordingly for the determined sub-codewords. Finally, each determined sub-codewords can be updated as $z^{t+1}_{m,k} \leftarrow z^t_{m,k} + \frac{1}{n_{m,k}}\sum_{\{x^{t,i}_m \in \mathcal{C}^t_{m,k}\}}(x^{t,i}_m - z^t_{m,k})\ \forall m \in \{1, \ldots M\}$ where $x^{t,i}_m$ is a streaming data point in the mth subspace in the tth iteration with $z^t_{m,k}$ as its nearest sub-codeword.

3.4 Partial Codebook Update

As we mentioned in the introduction, one of the issues is that online indexing model might incur high computational cost in update. Each new incoming data point might contribute in different significance of changes in different subspaces of nearest sub-codeword update. An obvious example of this is that, given a new streaming data, one of its sub-vector is far from its nearest sub-codeword and another of its sub-vector is close to its nearest sub-codeword, then the first one contributes more in PQ index update than the second one. Moreover, mini-batch streaming data sometimes would result in a number of sub-codewords to be updated across different subspaces, and different sub-codewords (within or outside the same subspace) would have different significance of changes in update. It is worthless to update the sub-codeword when the update change is minimal. To better illustrate the idea, we show the update of a mini-batch of streaming data in Figure 3. After assigning the nearest codeword to the new data, it shows that there is one sub-codeword in each subspace that is hugely different from its previous sub-codeword. The other two sub-codewords barely changed. Therefore, the update cost can be further reduced by ignoring the update of these sub-codewords that have less significant changes. Thus we can tackle the issue of possible high computational cost of update as we mentioned in the introduction by employing partial codebook update strategy, which can be achieved by adding one of the two budget constraints: the number of subspaces and the number of sub-codewords to be updated.

3.4.1 Constraint On Subspace Update

Each streaming data point is assigned to a sub-codeword in each subspace, so at least one sub-codeword in each subspace needs to be updated at each iteration. It is possible that the features in some subspaces of the new data have a vital contribution in their corresponding sub-codeword update and the features in some other subspaces have trivial contribution. Therefore, we target at updating the sub-codewords in the subspaces with significant update changes only. The subspace update constraint we add to our framework is:

$$\phi_M = \{m : m \in \{1, ...M\} \; and \; \|\Delta Z^t_m\| \neq 0\}, |\phi_M| \leq \alpha$$

where α represents the number of subspaces to be updated and $1 \leq \alpha \leq M$. In terms of linear programming, we select the top α subspaces with the most significant update to get the optimal solution. The significance of the subspace update can be computed by the sum of the quantization errors of streaming data. Thus Steps 7 and 8 in Algorithm 1 are applied for determined sub-codewords in the selected top α subspaces.

3.4.2 Constraint On Sub-codeword Update

Specifically in the case of mini-batch streaming data update, it is likely that each mini-batch consists of different classes of data, which results in a number of sub-codewords in each subspace to be updated. Similar to subspace update constraint, we propose a sub-codeword update constraint to ignore those sub-codewords with minimal update changes:

$$\sum_m^M \sum_k^K \mathbb{1}_{[\|\Delta z^t_{m,k}\|_0 \neq 0]} \leq \lambda M K$$

where λ represents the percentage of sub-codewords to be updated and $0 \leq \lambda \leq 1$. Similar to the solution for handling the subspace update constraint, we select the top $\lambda M K$ sub-codewords with the highest quantization error and apply the update steps 7 and 8 to the selected sub-codwords.

3.5 Complexity Analysis

For our online PQ model with M subspaces and K sub-codewords in each subspace, the codebook update complexity for each iteration by a mini-batch of streaming data with size B ($B \geq 1$) and D dimensions is $\mathcal{O}(BKD + BM + BD)$, where these three elements represent the complexity of streaming data encoding, codewords counter update and codewords update respectively. The complexity of codewords update will get proportionally decreased to the budget constraint parameters if they are applied in our model.

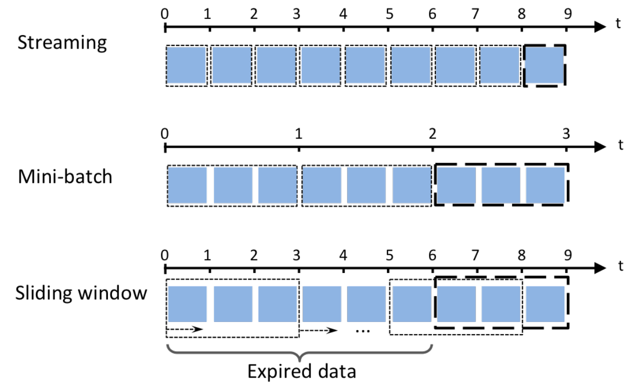


Fig. 4. Approaches of Handling Data Streams. Streaming: data streams one at a time. Mini-batch: a mini-batch of data with size 3 is processed by the model at each iteration t. Sliding window: a moving window with size 3 applied on continuously changing data.

Note the overall update complexity does not depend on the volume of the database at the current iteration.

4 Online Product Quantization over a Sliding Window

In the context of data streams, data distribution evolves over time. Some applications may aim at capturing the real-time behaviour of the data streams and thus emphasize the most recent data points in the stream. As the data stream evolves, some data will be expired based on their arrival times. To reflect the real-time and evolving behaviour of the data streams, we leverage a sliding window to facilitate the most recent data maintenance. Data insertion is performed to the online indexing model using the recent data in the sliding window, and data deletion removes the contributions of the codebook made by the expired data. Figure 4 compares three different approaches of handling data streams. In this section, we present the online PQ model over a time-based sliding window with both data insertion and deletion supported.

4.1 Online Product Quantization with Data Insertion and Deletion

Assume we are given a sliding window of size L. For window at the tth iteration, it consists of a stream of data $x^{t,1}, x^{t,2}, ...x^{t,L}$, where $x^{t,L}$ is the newly inserted data point to the window at the current iteration, and $x^{t-1,1}$ is just expired and removed from the window. Therefore, the objective function of the Online PQ over a sliding window at the tth iteration is stated as the following:

$$\min_{\substack{\mathcal{C}^t_{1,1},...,\mathcal{C}^t_{m,k},...,\mathcal{C}^t_{M,K} \\ z^t_{1,1},...,z^t_{m,k},...,z^t_{M,K}}} = \sum_{m=1}^M \sum_{i=1}^L \|x^{t,i}_m - z^t_{m,k}\|^2 \qquad (3)$$

where $x^{t,i}_m$ is the ith streaming data of the window in the mth subspace at the tth iteration and its nearest sub-codeword is $z^t_{m,k}$. To emphasize the real-time data in the stream, we want our model only affected by the data in the sliding window at the current iteration. Therefore, each time a new data streaming into the system, it moves to the sliding window. We first update the codebook by adding the contributions made by the new data. Correspondingly, the oldest data in the sliding window will be removed. We tackle the issue of data expiry by deleting the contribution to the codebook made by the data point that is just removed from the window. The solution of online PQ over a sliding window to handle insertion and deletion to the codebook is shown in Algorithm 2:

Algorithm 2 Online Product Quantization over Time-based Sliding Window Solutions for Eq.(3)

initialize PQ with the $M * K$ sub-codewords $z_{1,1}, ..., z_{m,k}, ...z_{M,K}$
create counters $n_{1,1}, ..., n_{m,k}, ...n_{M,K}$ for each sub-codeword and initialize each $n_{m,k}$ to be the number of initial data points assigned to the corresponding $z_{m,k}$
initialize the size of the sliding window L
for $t = 1, 2, 3, ...$ **do**
  remove $x^{t-1,1}$ from the sliding window
  $x^{t-1,i} (2 \leq i \leq L)$ from the $(t-1)$th sliding window is now represented as $x^{t,i-1}$ at the current iteration
  insert the new data $x^{t,L}$ to the sliding window at position L
  partition the sliding window data X^t and $x^{t-1,1}$ into M subspaces $(X_1^t, ..., X_M^t)$ and $(x_1^{t-1,1}, ..., x_M^{t-1,1})$, respectively
  in each subspace, determine and assign the nearest sub-codeword $z_{m,k}^t$ for each sub-vector $x_m^{t,i}$ of X_m^t and $x_m^{t-1,1}$
  Insertion:
  update the number of points for each sub-codeword of X^t: $n_{m,k} \leftarrow n_{m,k} + 1 \; \forall m \in \{1, ..., M\}$
  update the sub-codeword: $z_{m,k}^{t+1} \leftarrow z_{m,k}^t + \frac{1}{n_{m,k}} (x_m^{t,i} - z_{m,k}^t) \; \forall m \in \{1, ..., M\}$
  Deletion:
  update the number of points for each sub-codeword of $x^{t-1,1}$: $n_{m,k} \leftarrow n_{m,k} - 1 \; \forall m \in \{1, ..., M\}$
  update the sub-codeword: $z_{m,k}^{t+1} \leftarrow z_{m,k}^t - \frac{1}{n_{m,k}} (x_m^{t-1,1} - z_{m,k}^t) \; \forall m \in \{1, ...M\}$
end for

5 Loss Bound

In this section we study the relative loss bounds for our online product quantization, assuming our framework processes streaming data one at a time. Traditional analysis for online models are convex problems with vectors as variables. Our model, on the other hand, is non-convex and has matrices as variables, which makes the analysis non-trivial to be handled. Moreover, each of the continuously learned codewords may not be consistently matching with each codeword in the best fixed batch model. For example, a new incoming data may be assigned to the codeword with index 1 in our model but to the codeword with index 3 in the best batch model. This will make the loss bound even more difficult to study. Without using the properties of the convex function, we derive the loss bound of our model.

Here we define "loss" and "codeword" analogous to "prediction loss" and "prediction", and follow the analysis in [4]. Since all M subquantizers are independent to each other, we focus on the loss bound in a subquantizer m. We use ℓ_m^t to denote the instantaneous quantization error produced by the mth subquantizer in our model during iteration t, and ℓ^{t*} to denote the quantization error by the mth subquantizer of a fixed batch mode quantization model we are comparing with. We concatenate all sub-codewords together and denote them by $Z_m^t = [z_{m,1}^t, ..., z_{m,K}^t]$. For an arbitrary concatenated sub-codewords U in $\mathbb{R}^{\lceil \frac{D}{M} \rceil \times K}$, $U = [u_1, ..., u_K]$, where column vectors of Z_m^t and U are paired by minimum distance, *i.e.* $z_{m,k}^t$ matches u_k, and $\lceil x \rceil$ is the ceiling function that maps x to its nearest integer up. Formally, we define

$$\ell_m^t = ||x_m^t - z_{m,k}^t||^2 \quad and \quad \ell^{t*} = ||x_m^t - u_{k*}||^2 \tag{4}$$

where x_m^t is the streaming data in the mth subspace for the current iteration, $z_{m,k}^t$ is the codeword of x_m^t in our model and u_{k*} is the codeword of x_m^t in the comparison model.

Following Lemma 1 in [4], we derive the following lemma:

Lemma 1. Let $x_m^1, ..., x_m^T$ be a sequence of examples for the mth subspace where $x_m^t \in \mathbb{R}^{\lceil \frac{D}{M} \rceil}$. Assume $||x_m^t - u_k|| - ||x_m^t - u_{k*}|| \leq \beta$ where β is a constant. Let $\mathcal{T}_n$ be the set of iteration numbers where $z_{m,k}^t$ does not match u_{k*}, *i.e.* $||u_k - u_{k*}|| \neq 0$. Using the notation provided in Eq.4, then

$$\sum_{t=1}^{T} \frac{1}{n_{m,k}^t} ((1 - \frac{1}{n_{m,k}^t}) \ell_m^t - \ell^{t*} - ||x_m^t||^2 - ||u_k||^2) \\ - \beta \sum_{t \in \mathcal{T}_n} \frac{1}{n_{m,k}^t} \leq ||Z_m^1 - U||^2$$

where Z_m^1 is initialized to be nonzero matrix.

Proof Following the proof of Lemma 1 in [4], define Δ_t to be $||Z_m^t - U||_F^2 - ||Z_m^{t+1} - U||_F^2$. We obtain that,

$$\sum_{t=1}^{T} \Delta_t \leq ||Z_m^1 - U||_F^2$$

We now try to bound Δ_t. If a sub-codeword of a streaming data point is not selected to be updated due to the budget constraint during iteration t, *i.e.* Z_m^{t+1} is the same as Z_m^t, then this $\Delta_t = 0$. Therefore, we focus on the iterations for which $||\Delta Z_m^t||_F > 0$. As $z_{m,k}^{t+1} = z_{m,k}^t + \Delta z_{m,k}$ where $\Delta z_{m,k} = \frac{x_m^t - z_{m,k}^t}{n_{m,k}^t}$ when $z_{m,k}^t$ is the codeword for x_m^t, we define $\Phi_k(\Delta z_{m,k})$ as the difference between Z_m^{t+1} and Z_m^t, where $\Delta z_{m,k}$ is the difference vector between the kth column vector of Z_m^{t+1} and Z_m^t. We can therefore write Δ_t as,

$$\begin{aligned} \Delta_t &= ||Z_m^t - U||_F^2 - ||Z_m^{t+1} - U||_F^2 \\ &= ||Z_m^t - U||_F^2 - ||Z_m^t - U + \Phi_k(\Delta z_{m,k})||_F^2 \\ &= -2trace((Z_m^t - U)^T \Phi_k(\Delta z_{m,k})) \\ &\qquad - trace(\Phi_k(\Delta z_{m,k})^T \Phi_k(\Delta z_{m,k})) \\ &= -2(z_{m,k}^t - u_k) \cdot \Delta z_{m,k} - ||\Delta z_{m,k}||^2 \\ &= \frac{1}{n_{m,k}^t} (-2z_{m,k}^t \cdot x_m^t + 2x_m^t \cdot u_k + 2||z_{m,k}^t||^2 \\ &\qquad - 2z_{m,k}^t \cdot u_k - \frac{||x_m^t - z_{m,k}^t||^2}{n_{m.k}^t}) \end{aligned}$$

From Eq.4, we obtain that $\ell_m^t - ||x_m^t||^2 = ||z_{m,k}^t||^2 - 2x_m^t \cdot z_{m,k}^t$ and $-\ell^{t*} \leq 2x_m^t \cdot u_{k*}$. In addition, $-||u_k||^2 \leq ||z_{m,k}^t||^2 - 2z_{m,k}^t \cdot u_k$, If $z_{m,k}^t$ matches u_{k*}, then u_{k*} is the same as u_k, then

$$\Delta_t \geq \frac{1}{n_{m,k}^t}(\ell_m^t - \frac{\ell_m^t}{n_{m,k}^t} - \ell^{t*} - ||x_m^t||^2 - ||u_k||^2)$$

If $z_{m,k}^t$ does not match u_{k*}, then based on the assumption $||x_m^t - u_k|| - ||x_m^t - u_{k*}|| \leq \beta$,

$$\Delta_t \geq \frac{1}{n_{m,k}^t}(\ell_m^t - \frac{\ell_m^t}{n_{m,k}^t} - \ell^{t*} - \beta - ||x_m^t||^2 - ||u_k||^2)$$

Overall, we obtain our conclusion.

Following the Theorem 2 in [4] and our Lemma 1, we derive our theorem.

Theorem 1. Let $x_m^1, ..., x_m^T$ be a sequence of examples for the mth subspace where $x_m^t \in \mathbb{R}^{\lceil \frac{D}{M} \rceil}$ and $||x_m^t||^2 \leq R^2$. Assume that there exists a matrix U such that ℓ^{t*} is minimized for all t, and $\max_{1 \leq k \leq K} ||u_k||^2 \leq F^2$. Then, the cumulative quantization error of our algorithm is bounded by

$$\sum_{t=1}^{T} \ell_m^t < 4(||Z_m^1 - U||_F^2 + \beta \sum_{t \in \mathcal{T}_n} \frac{1}{n_{m,k}^t}) + 4T(R^2 + F^2) + 8 + 4\sum_{t=1}^{T} \ell^{t*}$$

Proof Since $1 \leq n_t^{m,k} \leq t$, then $\frac{1}{n_{m,k}^t} \leq 1$. Using the facts that $||x_m^t||^2 \leq R^2$ and $\max_{1 \leq k \leq K} ||u_k||^2 \leq F^2$, Lemma 1 implies that,

$$\sum_{t=1}^{T} \frac{1}{n_{m,k}^t}(1 - \frac{1}{n_{m,k}^t})\ell_m^t - \sum_{t=1}^{T} \ell^{t*} \leq ||Z_m^1 - U||_F^2 + \beta \sum_{t \in \mathcal{T}_n} \frac{1}{n_{m,k}^t} + T(R^2 + F^2) + 2$$

Since $\frac{1}{n_{m,k}^t}(1 - \frac{1}{n_{m,k}^t}) \leq \frac{1}{4}$, we get our relative loss bound.

If there exists a concatenated sub-codewords U that is produced by the best fixed batch algorithm in hindsight, ℓ^{t*} is minimal. A tighter bound can be achieved if $z_{m,k}^t$ matches u_k in all iterations and the initialized sub-codewords is close to the optimal sub-codewords. The performance of online PQ model is guaranteed for unseen data, and the loss bound scales linearly with the squared norm of the streaming data and the sub-codeword with the maximum norm in the optimal sub-codebook.

6 Experiments

We conduct a series of experiments on several real-world datasets to evaluate the efficiency and effectiveness of the online PQ model. In this section, we first introduce the datasets used in the experiments. We then compare the online version and the mini-batch version of our online PQ model. After that, we analyze the impact of the parameters α and λ in update constraints. Finally, we compare our proposed model with existing related hashing methods for different applications.

TABLE 3
Detailed datasets information

Dataset	Class no.	Size	Feature
News20	20	18,845	Doc2vec (300)
Caltech-101	101	9,144	GIST (512)
Half dome	28,086	107,732	GIST (512)
Sun397	397	108,753	GIST (512)
Video Dataset	**Video**	**Frame**	**Feature**
YoutubeFaces (CSLBP)	3,425	621,126	CSLBP (480)
YoutubeFaces (FPLBP)	3,425	621,126	FPLBP (560)
UQ_VIDEO	169,952	3,304,554	HSV (162)

6.1 Datasets and evaluation criterion

There are one text dataset, three image datasets and two video datasets employed to evaluate the proposed method. **20 Newsgroups Data** (News20)[1] [28] consists of chronologically ordered 18,845 newsgroup messages. **Caltech-101** [2] [29] consists of 9144 images and each image belongs to one of the 101 categories. **Half dome** [3] [30] includes 107,732 image patches obtained from Photo Tourism reconstructions from Half Dome (Yosemite). **Sun397** [4] [31] contains around 108K images in 397 scenes. **YoutubeFaces** [5] contains 3,425 videos of 1,595 different people, with a total of 621,126 frames. **UQ_VIDEO** [6] consists of 169,952 videos with 3,305,525 frames in total. We use 300-D doc2vec features to represent each news article in News20 and 512-D GIST features to represent each image in the three image datasets. We use two different features, 480-D Center-Symmetric LBP (CSLBP) and 560-D Four-Patch LBP (FPLBP) to represent each frame in YoutubeFaces. 162-D HSV feature is used in UQ_VIDEO dataset. Table 3 shows detailed statistical information about datasets used in evaluation.

We measure the performance of our proposed model by the model update time and the search quality measurement recall@R adopted in [14]. We use recall@20 which indicates that fraction of the query for which the nearest neighbor is in the top 20 retrieved images by the model.

6.2 Online vs mini-batch

In real-world applications, streaming data might be processed one at a time or in small batches. For example, real-time topic detection in streaming media can be applied on texts, images or videos. If the streaming data is Twitter post, it might be processed one at a time. If the media is video, then the streaming data can be processed in mini-batches of video frames. Our model can process streaming data either one at a time or in mini-batches. We compare these two versions of our model on Caltech-101 dataset. For our model, we use $M = 8$ and $K = 256$. Since the memory cost of storing each codeword is $M\lceil log_2 K \rceil$ bits [14], then the number of bits used in our model is 64. We split the data into twelve groups, where one of the groups is used for learning the codebook, one of the groups is used as the query and

1. http://people.csail.mit.edu/jrennie/20Newsgroups(20news-bydate.tar.gz)
2. http://www.vision.caltech.edu/Image Datasets/Caltech101/
3. http://phototour.cs.washington.edu/patches/default.htm
4. http://groups.csail.mit.edu/vision/SUN/
5. https://www.cs.tau.ac.il/ wolf/ytfaces/
6. http://staff.itee.uq.edu.au/shenht/UQ_VIDEO/

each one of the rest of the ten groups is used to update the original codebook, so that we have the performance for ten iterations. Figure 5 shows the comparison of the online version and the mini-batch (MB) version in update time and recall@R measurements. It indicates that the mini-batch version takes much less update time than the mini-batch version but they have similar search quality. Therefore, we adopt mini-batch version of our model in the rest of the experiments.

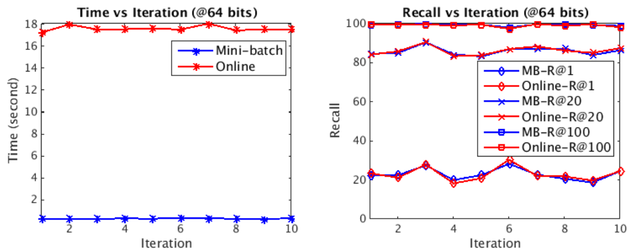


Fig. 5. The left figure shows the update time for each iteration of update. The time of the online version for each iteration sums up the update time of the streaming data corresponding to the ones in the mini-batch. The right figure shows the recall@1, 20 and 100 for each iteration.

6.3 Update time complexity vs search accuracy

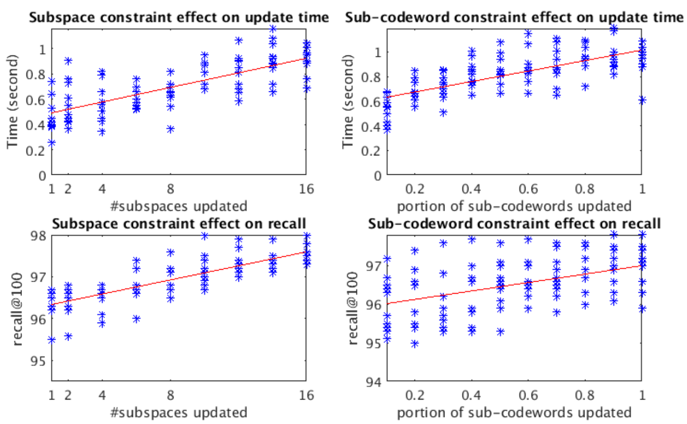


Fig. 6. Trade-offs between update time cost and the search accuracy. The first column shows the impact of the subspace update constraint. The second column shows the impact of the sub-codeword update constraint. The red line is the reference line for the scatter plot.

There are two budget constraints we proposed for the codebook update to further reduce the time cost: number of subspaces and number of sub-codewords to be updated. In this experiment, we evaluate the impact of these two constraints and the trade-offs between update time cost and the search quality using a synthetics dataset. We randomly sampled 12000 data points with 128-D features from multivariate normal distribution. We use recall@50 as the performance measurement which indicates that fraction of the query for which the nearest neighbor is in the top 50 retrieved images by the model. We set $M = 16$ and $K = 256$, and vary the number of updated subspaces from 1 to 16 and the portion of updated sub-codewords from 0.1 to 1 respectively. We split the dataset evenly into twelve groups and set one of the groups as the learning set to learn the codebook and another one as the query set, and use each of the rest of ten groups to update the learned codebook and record the update time cost and the search accuracy for 10 times while varying the update constraints. From Figure 6, we observe that the search quality and update time cost strongly depend on these two update constraints. As we increase the number of subspaces or the portion of sub-codewords to be updated, the update time cost is increasing, along with the search accuracy. Therefore, higher update time cost is required for better search accuracy.

6.4 Baseline methods

To verify that our online PQ model is time-efficient in update and effective in nearest neighbor search, we make comparison with several related online indexing and batch learning methods. We evaluate the performance in both search accuracy and update time cost. We select SSBC [9] and OSH [10] as two of the baseline methods, as they are the only unsupervised online indexing methods to the best of our knowledge. Specifically, SSBC is only applied on two of our smallest datasets, News20 and Caltech-101, as it takes a significant amount of time to train. In addition, two supervised online indexing methods OH [7], [8] and AdaptHash [11] are selected, with the top 5 percentile nearest neighbors in Euclidean space as the ground truth following the setting as in [10]. Further, four batch learning indexing methods are selected. Three of them are unsupervised data-dependent methods: PQ [14], spectral hashing (SH) [19] and IsoH [18]. Each of these three methods is compared with online PQ in two ways. The first way uses all the data points seen so far to retrain the model (batch) at each iteration. The second way uses the model trained from the initial iteration to assign the codeword to the streaming data in all the rest of the iterations (no update). The rest of the batch learning comparison methods is a data-independent method, LSH, which can be easily adapted in the online setting. We do not apply the batch learning methods on our largest dataset, UQ_VIDEO, as it takes too much retraining time at each iteration.

6.5 Object tracking and retrieval in a dynamic database

In many real-world applications, data is continuously generated everyday and the database needs to get updated dynamically by the newly available data. For example, news articles can be posted any time and it is important to enhance user experience in news topic tracking and related news retrieval. New images with new animal species may be inserted to the large scale image database. Index update needs to be supported to allow users to retrieve images with expected animal in a dynamically changing database. A live video or a surveillance video may generate several frame per second, which makes the real-time object tracking or face recognition a crucial task to solve. In this experiment, we evaluate our model on how it handles dynamic updates in both time efficiency and search accuracy in three different types of data: text, image and video.

6.5.1 Setting

For each dataset, we split data into several mini-batches, and stream each mini-batch into the models in turn. Since

TABLE 4
Number of iterations and average mini-batch size for each dataset

Dataset	News20	Caltech-101	Half dome	Sun397	YoutubeFaces	UQ_VIDEO
Iteration No.	20	12	12	21	67	25
Avg mini-batch size	942.25	762	8,977.7	5,178.7	8,628.4	132,182

the news articles in News20 dataset are ordered by date, we stream the data to the models in its chronological order. Image datasets consist of different classes. To simulate data evolution over time, we stream images by classes and each pair of two consecutive mini-batches have half of the images from the same class. In the video datasets, videos are ordered by their labels, such as the videos belonging to the same person are grouped together and then sets of videos stream to the models in turn. For text and image datasets, we have dynamic query set in each iteration. After the initial mini-batch of data used to initialize the model, each time a mini-batch of streaming data comes, we use each of them as the query to search for the nearest neighbors from the existing database, and then update the current model by accommodating these streaming data. For YoutubeFace video dataset, we have a randomly sampled fixed set of queries consisting 226 videos with 43,020 frames in total. UQ_VIDEO dataset provides a fixed set of 24 videos with 902 frames in total. Table 4 shows detailed information about the data streams.

In this experiment, we compare our method with subspace update constraint (Online PQ - SS) and sub-codeword update constraint (Online PQ - SC) to several batch mode methods and online methods for the task of continuous update in a dynamic setting. In our model, we set $M = 8$ and $K = 256$. then the number of bits used for vector encoding is 64. We constraint the number of the updated subspaces α to be 4 and the portion of the updated sub-codewords λ to be 0.5 respectively. The first batch is used for codebook initialization and the rest of the batches are used to update the existing codebook one at a time. All the key parameters in the baseline methods are set to the ones recommended in the corresponding papers. All the methods compared are implemented in Matlab provided by the authors and all experiments are conducted on a workstation with a 3.10GHZ Intel CPU and 120GB main memory running on a Linux platform. We use 64 bits for vector encoding in all of the comparison models for fair comparisons.

6.5.2 Online methods comparison

Figure 7 demonstrate the performance of indexing update using different number of bits of the model on News20 dataset compared with four online hashing methods. It clearly shows that our proposed models consistently outperforms other online hashing methods, with the lowest update time cost in different number of bits. In particular, when the number of bits increases, the difference between online PQ and other online methods gets increasing. SSBC achieves comparable search accuracy with online PQ in some of the iterations using 32 bits, but its update time cost is significantly higher than other methods.

As shown in Figure 8 and Figure 9, it is evident that our proposed method with two different budget constraints

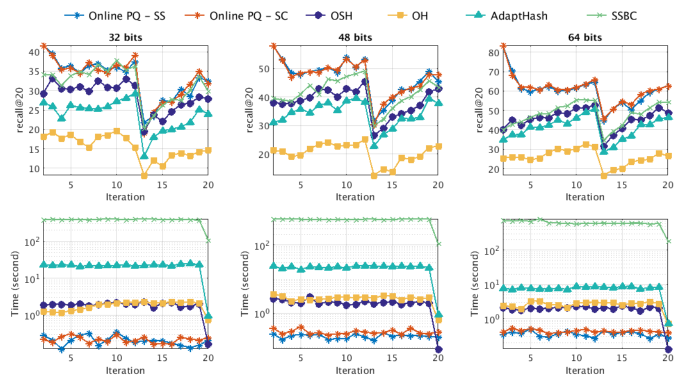


Fig. 7. Recall@20 performance (1st row) and Update time cost (2nd row) comparison against online hashing methods at each iteration for News20 datasets using different number of bits. 1st column: 32 bits. 2nd column: 48 bits. 3rd column: 64 bits. Time cost is in log scale.

can achieve superior performance in both efficiency and effectiveness compared to other online methods. Specifically, online PQ significantly outperforms the second best online method, OSH, in search accuracy and is much faster in model update.

As OH and AdaptHash are supervised online hashing methods, and OSH performs the best over all baseline methods, we compare our proposed model with two different budget constraints with OSH in Figure 10. It is obvious that our method achieves better search accuracy with lower update time cost. Moreover, although the difference of the performance between the two budget constraints of our model is minimum, updating sub-codewords in half of the subspaces performs slightly better than updating half of the sub-codewords of all in both search accuracy and update time.

6.5.3 Batch methods comparison

To further evaluate the performance of nearest neighbor search of our online model on how well it approaches to the search accuracy of batch mode methods and to the model update time of ”no update” methods, we compare our model with each of the batch mode methods in two ways: retrain the model at each iteration (batch) and using the model trained on the initial iteration once for all (no update). The comparison results displayed in Figure 11 and Figure 12 are revealing in several ways. First, as the update time cost graphs are plot in log scale, the update time of online PQ is only slightly more than the one of the ”no update” methods, but significantly lower than the one of the ”batch” methods. Second, online PQ and PQ methods significantly outperform other batch hashing methods, and online PQ performs slightly worse than ”batch” PQ and better than ”no update” PQ. Therefore, we can conclude that our online model can achieves comparable search accuracy

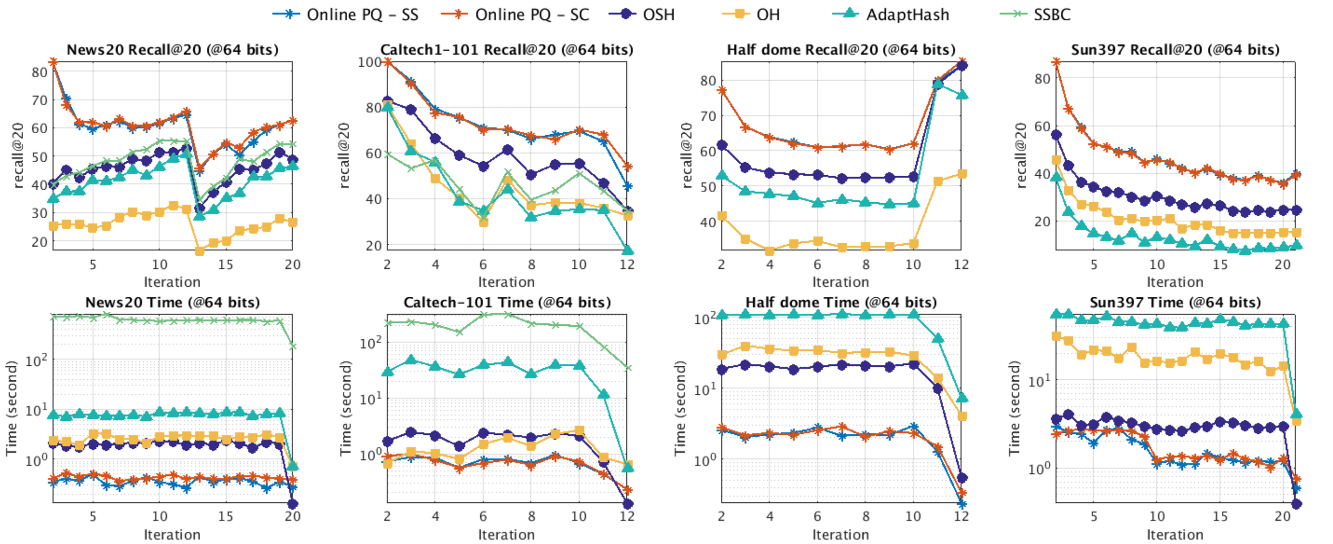


Fig. 8. Results for news and image retrieval in a dynamic database comparison against online hashing methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: News20. 2nd column: Caltech-101. 3rd column: Sun397. 4th column: Half dome. Time cost is in log scale.

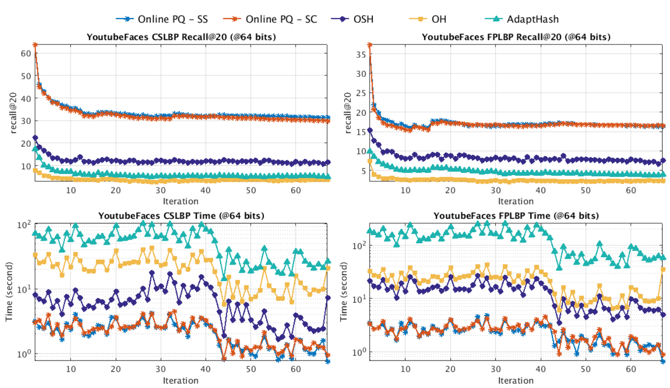


Fig. 9. Results for YoutubeFaces dataset on CSLBP and FPLBP features in a dynamic database comparison against online hashing methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: CSLBP feature. 2nd column: FPLBP feature. Time cost is in log scale.

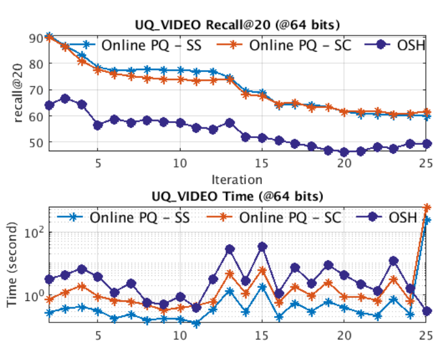


Fig. 10. Results for UQ_VIDEO dataset in a dynamic database comparison against online hashing methods. Recall@20 performance (1st row) and Update time cost (2nd row). Time cost is in log scale.

to batch mode methods with taking only slightly more time than "no update" batch model in update.

6.6 Continuous querying on dynamic real-time data

In many emerging application environments, they commonly require to emphasize the most recent data and ignore the expired data in retrieval search. Examples of such applications include network monitoring, cyber security analytics and online portfolio selection. Furthermore, applications such as hot topic news article retrieval system, photo search or object tracking given a recent period of time from social network albums or live videos require the real-time behaviour of the data. Therefore, to reflect this requirement in an online indexing system, we employ sliding window technique. In this experiment, we investigate the comparison between with and without employing the sliding window technique, and presents the comparison results on different hash methods.

6.6.1 Setting

We follow the same way as in the setting in Section 6.5.1 to order the data in text and video datasets and the image data is ordered by classes without the overlapping of classes in each pair of two consecutive mini-batches this time. We set the number of iterations to be 11 for text and image datasets, and 65 and 34 for YoutubeFaces and UQ_VIDEO respectively. The sliding window size is set to be 2000 for News20, 1000 for Caltech-101, 10000 for Sun397, Half dome and YoutubeFaces, and 100000 for UQ_VIDEO. Except for News20 dataset which contains news articles in chronological order, for the rest of the datasets, the sliding window contains images/videos belonging to a certain amount of classes/people at each iteration, so that the contribution of the classes/people for the expired data are removed from our proposed model. We use the dynamic query set in this setting for all the datasets, so we use each new coming mini-batch of data as the query set first to retrieve similar data

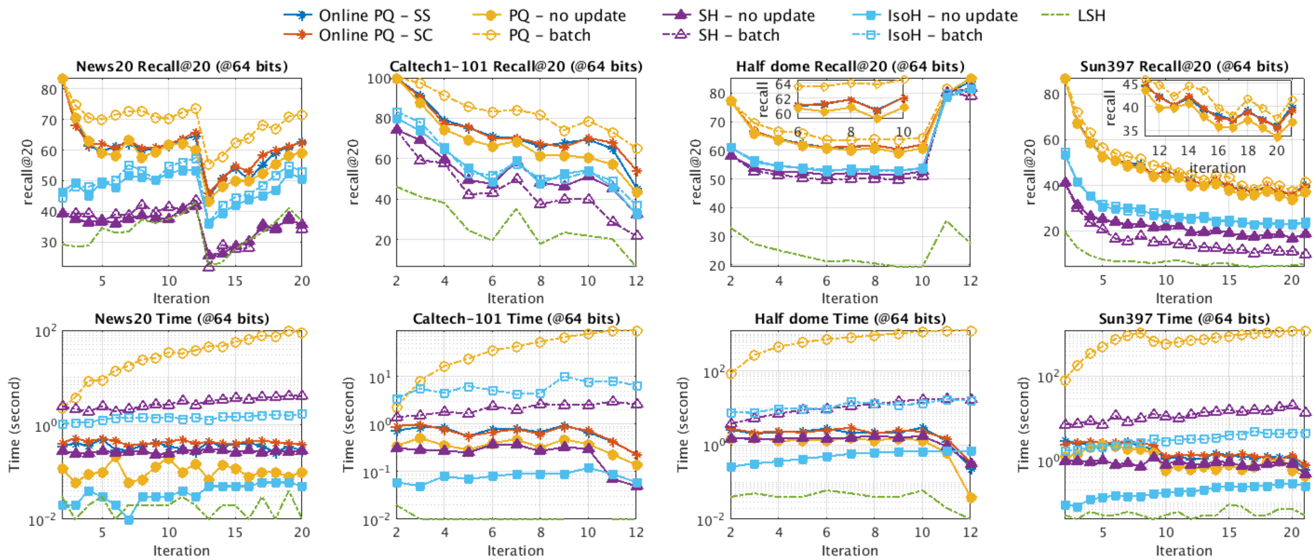


Fig. 11. Results for news and image retrieval in a dynamic database comparison against batch methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: News20. 2nd column: Caltech-101. 3rd column: Sun397. 4th column: Half dome. Part of the recall plots of online PQ and baseline PQ methods for sun397 and half dome datasets are enlarged. Time cost is in log scale.

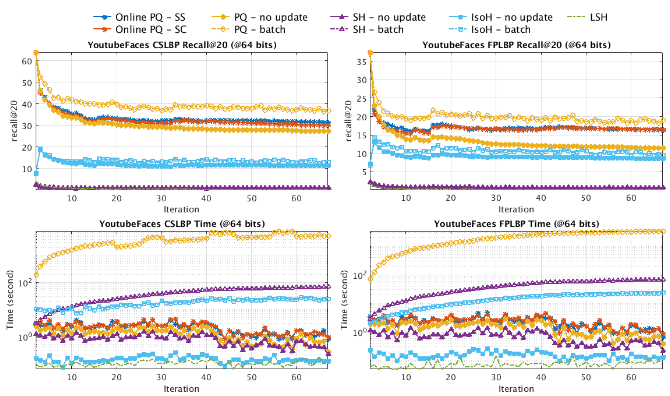


Fig. 12. Results for YoutubeFaces dataset on CSLBP and FPLBP features in a dynamic database comparison against batch methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: CSLBP feature. 2nd column: FPLBP feature. Time cost is in log scale.

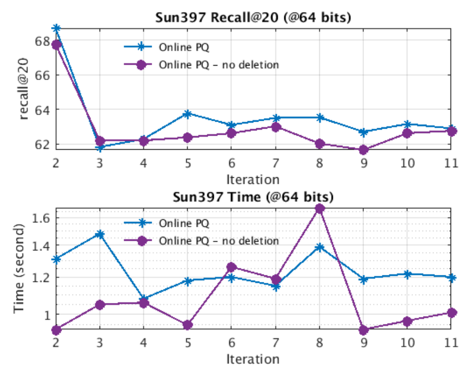


Fig. 13. Online PQ over a sliding window approach between deletion and without deletion of the expired data to the model for Sun397. Recall@20 performance (1st row) and Update time cost (2nd row).

from the sliding window of the previous iteration and then use this mini-batch of data to update the model. In our model, we remove the contribution of the data once it is removed from the sliding window. We set $M = 8$ and $K = 256$, and update all the codewords over all subspaces. Batch mode baseline methods retrain the model using the data in the sliding window at each iteration.

6.6.2 *Online methods comparison*

Our proposed method over a sliding window approach adds the contribution of the new incoming data to the index model and removes the contribution of the expired data to the index model at each iteration. To evaluate our approach on data deletion using the sliding window technique, we highlight the difference between the models with and without expired data deletion in nearest neighbor search task in Figure 13. The update time of online PQ with expired data deletion is reasonably slightly higher than that of online PQ without expired data deletion and the search accuracy of online PQ with expired data deletion is slightly better as it emphasizes on the ”real-time” data.

From Figure 14 and Figure 15, we can see that online PQ over a sliding window approach achieves the best performance over other online methods in terms of update time efficiency and search effectiveness. Specifically, the search accuracy for YoutubeFaces dataset of CSLBP feature is significantly higher than OSH, with low update time cost. Figure 16 further verifies our proposed method in processing a large scale dataset by comparing our method with the state-of-the-art unsupervised online hashing method OSH.

6.6.3 *Batch methods comparison*

To investigate how well our proposed method over a sliding window technique with expired data deletion approaches the performance of the batch mode methods where the batch models will be retrained on the data from the sliding window at each iteration, we compare our model with batch mode methods. In addition, we compare with ”no update”

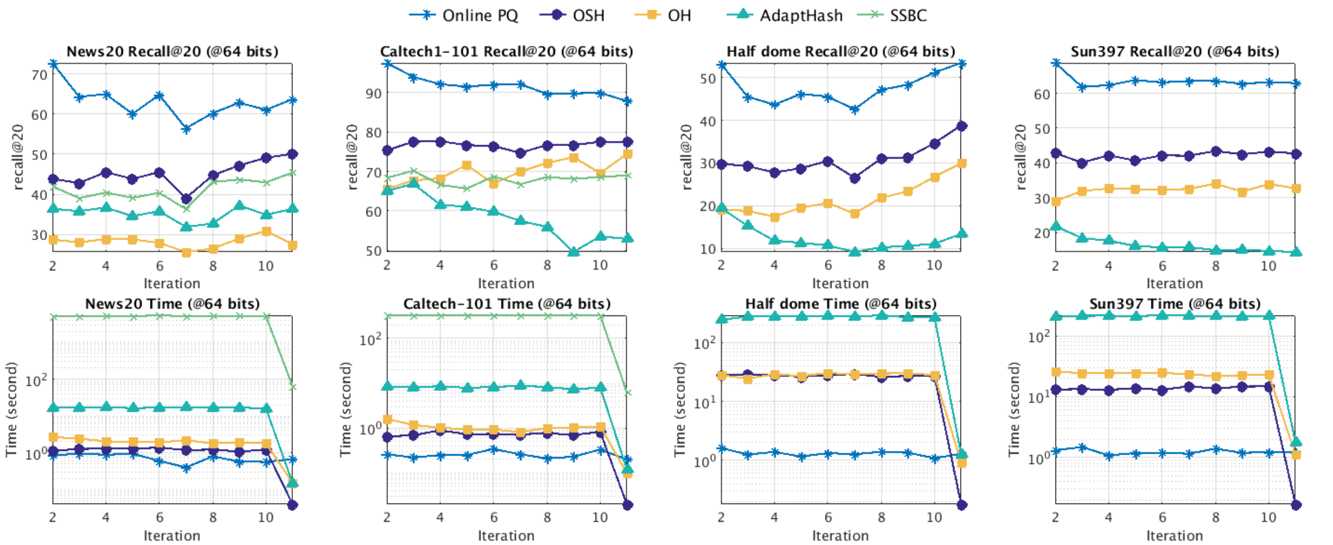


Fig. 14. Results for news and image retrieval over a sliding window comparison against online hashing methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: News20. 2nd column: Caltech-101. 3rd column: Sun397. 4th column: Half dome. Time cost is in log scale.

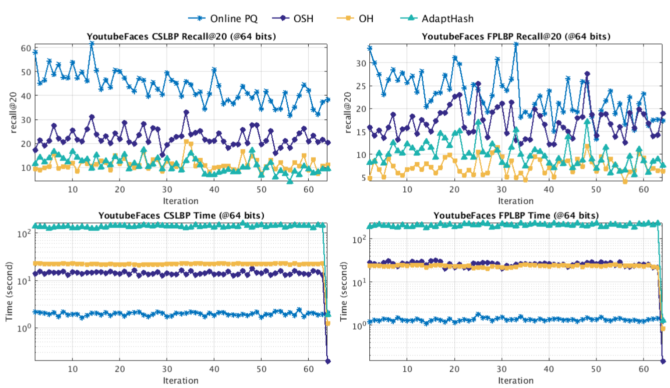


Fig. 15. Results for YoutubeFaces dataset on CSLBP and FPLBP features over a sliding window comparison against online hashing methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: CSLBP feature. 2nd column: FPLBP feature. Time cost is in log scale.

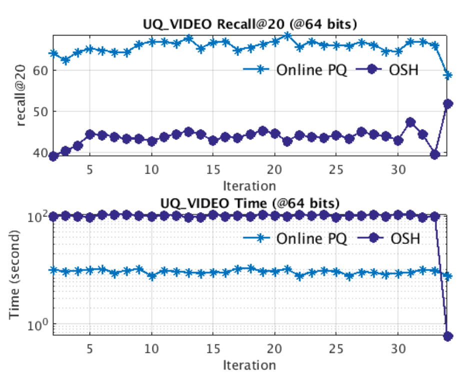


Fig. 16. Results for UQ_VIDEO dataset over a sliding window comparison against online hashing methods. Recall@20 performance (1st row) and Update time cost (2nd row). Time cost is in log scale.

model to show the update time complexity of our method. These comparisons are shown in Figure 17 and Figure 18. They reveal that our proposed method achieves comparable search accuracy with batch mode PQ with the update time cost only slightly more than ”no update” PQ.

7 Conclusion and Future Work

In this paper, we have presented our online PQ method to accommodate streaming data. In addition, we employ two budget constraints to facilitate partial codebook update to further alleviate the update time cost. A relative loss bound has been derived to guarantee the performance of our model. In addition, we propose an online PQ over sliding window approach, to emphasize on the real-time data. Experimental results show that our method is significantly faster in accommodating the streaming data, outperforms the competing online hashing methods and unsupervised batch mode hashing method in terms of search accuracy and update time cost, and attains comparable search quality with batch mode PQ. In our future work, we will extend the online update for other MCQ methods, leveraging the advantage of them in a dynamic database environment to enhance the search performance. Moreover, the theoretical bound for the online model will be further investigated.

References

[1] A. Moffat, J. Zobel, and N. Sharman, “Text compression for dynamic document databases,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 2, pp. 302–313, 1997.

[2] R. Popovici, A. Weiler, and M. Grossniklaus, “On-line clustering for real-time topic detection in social media streaming data,” in *SNOW 2014 Data Challenge*, 2014, pp. 57–63.

[3] A. Dong and B. Bhanu, “Concept learning and transplantation for dynamic image databases,” in *International Conference on Multimedia and Expo*, 2003, pp. 765–768.

[4] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, “Online passive-aggressive algorithms,” *Journal of Machine Learning Research*, vol. 7, pp. 551–585, 2006.

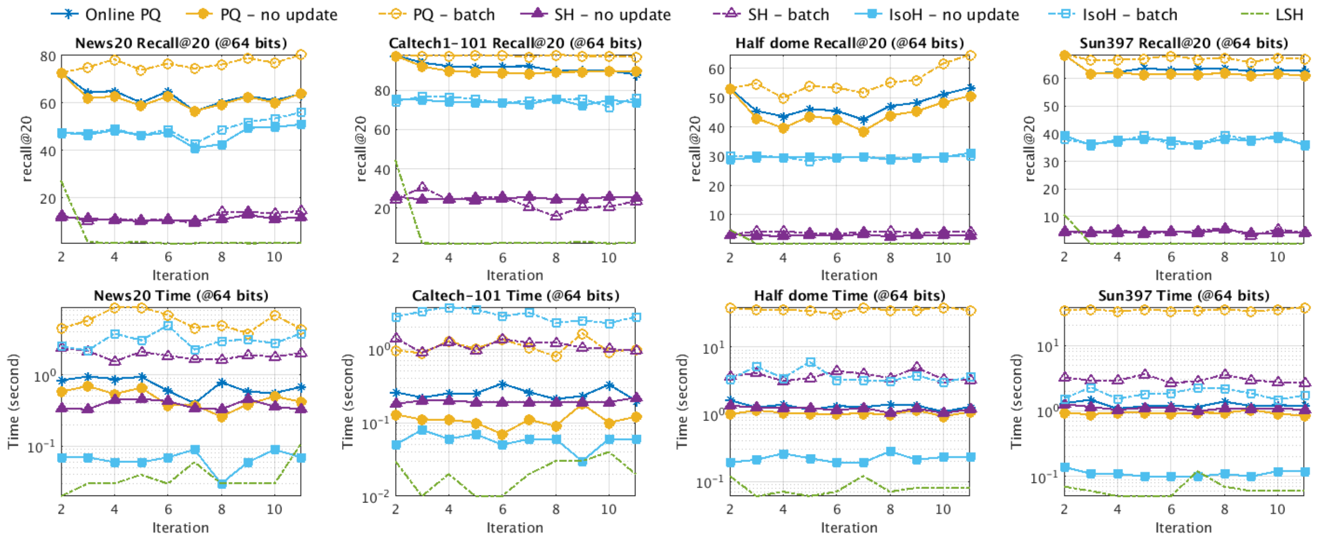


Fig. 17. Results for news and image retrieval over a sliding window comparison against batch methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: News20. 2nd column: Caltech-101. 3rd column: Sun397. 4th column: Half dome. Time cost is in log scale.

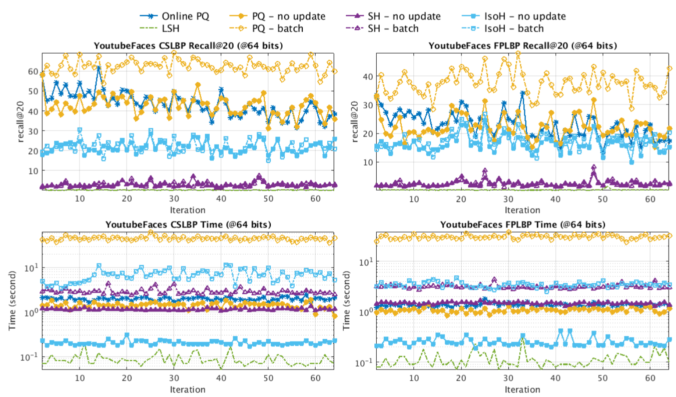


Fig. 18. Results for YoutubeFaces dataset on CSLBP and FPLBP features over a sliding window comparison against batch methods. Recall@20 performance (1st row) and Update time cost (2nd row). 1st column: CSLBP feature. 2nd column: FPLBP feature. Time cost is in log scale.

[5] C. Li, W. Dong, Q. Liu, and X. Zhang, "Mores: online incremental multiple-output regression for data streams," *CoRR, vol. abs/1412.5732*, 2014.

[6] L. Zhang, T. Yang, R. Jin, Y. Xiao, and Z. Zhou, "Online stochastic linear optimization under one-bit feedback," in *Proceedings of the 33nd International Conference on Machine Learning*, 2016, pp. 392–401.

[7] L. Huang, Q. Yang, and W. Zheng, "Online hashing," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2013, pp. 1422–1428.

[8] ——, "Online hashing," *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[9] M. Ghashami and A. Abdullah, "Binary coding in stream," *CoRR*, vol. abs/1503.06271, 2015.

[10] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu, "Online sketching hashing," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2503–2511.

[11] F. Cakir and S. Sclaroff, "Adaptive hashing for fast similarity search," in *International Conference on Computer Vision*, 2015, pp. 1044–1052.

[12] Q. Yang, L. Huang, W. Zheng, and Y. Ling, "Smart hashing update for fast response," in *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2013, pp. 1855–1861.

[13] F. Cakir, S. A. Bargal, and S. Sclaroff, "Online supervised hashing," *Computer Vision and Image Understanding*, 2016.

[14] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[15] M. Norouzi and D. J. Fleet, "Minimal loss hashing for compact binary codes," in *International Conference on Machine Learning*, 2011, pp. 353–360.

[16] W. Liu, J. Wang, R. Ji, Y. Jiang, and S. Chang, "Supervised hashing with kernels," in *Computer Vision and Pattern Recognition*, 2012, pp. 2074–2081.

[17] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *omputer Vision and Pattern Recognition*, 2011, pp. 817–824.

[18] W. Kong and W. Li, "Isotropic hashing," in *Advances in Neural Information Processing Systems*, 2012, pp. 1655–1663.

[19] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, 2008, pp. 1753–1760.

[20] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of 25th International Conference on Very Large Data Bases*, 1999, pp. 518–529.

[21] A. Babenko and V. S. Lempitsky, "Additive quantization for extreme vector compression," in *Computer Vision and Pattern Recognition*, 2014, pp. 931–938.

[22] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *International Conference on Machine Learning*, 2014, pp. 838–846.

[23] T. Zhang, G. Qi, J. Tang, and J. Wang, "Sparse composite quantization," in *Computer Vision and Pattern Recognition*, 2015, pp. 4548–4556.

[24] A. Babenko and V. S. Lempitsky, "Tree quantization for large-scale similarity search and classification," in *Computer Vision and Pattern Recognition*, 2015, pp. 4240–4248.

[25] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 744–755, 2014.

[26] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[27] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2010.

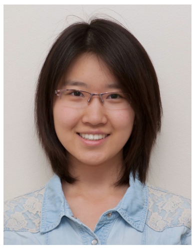
[28] K. Lang, "Newsweeder: Learning to filter NETnews," in *ICML*, 1995.

[29] F. Li, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *Computer Vision and Pattern Recognition Workshops*, 2004, p. 178.

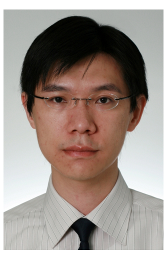
[30] S. A. J. Winder and M. A. Brown, "Learning local image descriptors," in *Computer Vision and Pattern Recognition*, 2007.

[31] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "SUN

database: Large-scale scene recognition from abbey to zoo," in *Computer Vision and Pattern Recognition*, 2010, pp. 3485–3492.



Donna Xu received a BCST (Honours) in computer science from the University of Sydney in 2014. She is currently pursuing a PhD degree under the supervision of Prof. Ivor W. Tsang at the Centre for Artificial Intelligence, FEIT, University of Technology Sydney, NSW, Australia. Her current research interests include multiclass classification, online hashing and information retrieval.



Ivor W. Tsang is an ARC Future Fellow and Professor at University of Technology Sydney (UTS). He is also the Research Director of the UTS Priority Research Centre for Artificial Intelligence (CAI). He received his PhD degree in computer science from the Hong Kong University of Science and Technology in 2007. In 2009, Dr Tsang was conferred the 2008 Natural Science Award (Class II) by Ministry of Education, China. In addition, he had received the prestigious IEEE Transactions on Neural Networks Outstanding 2004 Paper Award in 2007, the 2014 IEEE Transactions on Multimedia Prize Paper Award.

Ying Zhang is a senior lectuer and ARC DECRA research fellow (2014-2016) at QCIS, the University of Technology, Sydney (UTS). He received his BSc and MSc degrees in Computer Science from Peking University, and PhD in Computer Science from the University of New South Wales. His research interests include query processing on data stream, uncertain data and graphs. He was an Australian Research Council Australian Postdoctoral Fellowship (ARC APD) holder (2010-2013).