

Product Quantization Network for Fast Visual Search

Tan Yu · Jingjing Meng · Chen Fang · Hailin Jin · Junsong Yuan

Received: date / Accepted: date

Abstract Product quantization has been widely used in fast image retrieval due to its effectiveness of coding high-dimensional visual features. By constructing the approximation function, we extend the hard-assignment quantization to soft-assignment quantization. Thanks to the differentiable property of the soft-assignment quantization, the product quantization operation can be integrated as a layer in a convolutional neural network, constructing the proposed product quantization network (PQN). Meanwhile, by extending the triplet loss to the asymmetric triplet loss, we directly optimize the retrieval accuracy of the learned representation based on asymmetric similarity measurement. Utilizing PQN, we can learn a discriminative and compact image representation in an end-to-end manner, which further enables a fast and accurate image retrieval. By revisiting residual quantization, we further extend the proposed PQN to residual product quantization network (RPQN). Benefited from the resid-

ual learning triggered by residual quantization, RPQN achieves a higher accuracy than PQN using the same computation cost. Moreover, we extend PQN to temporal product quantization network (TPQN) by exploiting temporal consistency in videos to speed up the video retrieval. It integrates frame-wise feature learning, frame-wise features aggregation and video-level feature quantization in a single neural network. Comprehensive experiments conducted on multiple public benchmark datasets demonstrate the state-of-the-art performance of the proposed PQN, RPQN and TPQN in fast image and video retrieval.

Keywords product quantization · image retrieval · deep learning · video retrieval

1 Introduction

Visual search has been a fundamental research topic in computer vision. Given a query (an image or a video), it aims to retrieve the query's relevant items from a database. Accuracy and efficiency are two key aspects for a retrieval system. These two aspects drive the research on visual search to progress in two directions.

The first direction focuses on representation designing or learning for a higher search accuracy [43, 42, 26, 3, 2, 39, 16, 62, 4, 5]. A good representation maintains a large distance between irrelevant items in feature space and a close distance between relevant ones. Traditional retrieval systems first extract hand-crafted local features like SIFT and then aggregate local features into a global feature [43, 42, 26]. With the progress of deep learning, the convolutional neural network provides an effective representation [3, 2, 39, 60, 63], which is trained by the semantic information and thus is robust to low-level image transformations.

Tan Yu
Cognitive Computing Lab
Baidu Research, Seattle
E-mail: tyu008@e.ntu.edu.sg

Jingjing Meng
Computer Science and Engineering Department
University at Buffalo, State University of New York
E-mail: jmeng2@buffalo.edu

Chen Fang
Adobe Research, San Jose
E-mail: cfang@adobe.com

Hailin Jin
Adobe Research, San Jose
E-mail: hljin@adobe.com

Junsong Yuan
Computer Science and Engineering Department
University at Buffalo, State University of New York
E-mail: jsyuan@buffalo.edu

The second direction aims to boost the retrieval speed, especially when dealing with a large-scale dataset. In this circumstance, a compact visual representation is necessary. Generically speaking, there are two types of schemes to gain a compact representation, hashing and quantization. Hashing maps the real-value vectors into binary codes, which enables a faster distance computation and lower memory cost. One of widely used hashing methods is locality sensitivity hashing (LSH) [10]. Nevertheless, LSH is data-independent, which ignores the data distribution and is sub-optimal to a specific dataset. To further improve the performance, some hashing methods [45, 54, 15] learn the projection from the data, which caters better to a specific dataset and achieves a higher retrieval precision. Note that traditional hashing methods are based on a two-step procedure. To be specific, they first conduct feature extraction and then carry on Hamming embedding based on the extracted features. These two steps are independently conducted and their mutual influence are ignored. More recently, inspired by the progress of deep learning, some deep hashing methods [57, 31, 35, 32] are proposed, which simultaneously conduct feature learning and feature compression through a unified neural network, achieving much better retrieval performance compared with methods based on off-the-shelf features.

Nevertheless, hashing methods are only able to produce a few distinct distances, limiting its discriminative power in characterizing distances between data points. In parallel to hashing methods, another widely used data compression method in visual retrieval is product quantization. It quantizes each feature vector into a Cartesian product of several codewords and the similarity can be efficiently computed through looking up cached similarity table. Thanks to the asymmetric distance calculation mechanism, product quantization enables a more accurate distance calculation than hashing methods using the same code length. The product quantization (PQ) [25] and its optimized versions like OPQ [14], CKmeans [40], APQ [1] and CQ [66, 19] are originally designed for unsupervised scenarios where no labeled data are provided. SQ [53] extends product quantization to supervised scenarios. However, SQ is based on the hand-crafted features or CNN features from the pretrained model, therefore it might not be optimal with respect to a specific dataset.

To jointly optimize feature learning and product quantization, Cao *et al* [8] propose a deep quantization network (DQN) which supports an end-to-end training. It optimizes a weighted sum of similarity preserving loss and product quantization loss. It iteratively updates codewords and other parameters of a neural network. Therefore, in each iteration, the codewords are directly

updated by k-means whereas the label information is ignored. Recently, Klein *et al.* [28] propose a deep product quantization (DPQ). They learn a cascade of two fully-connected layers followed by a softmax layer to determine a soft codeword assignment. It is different from original product quantization as the codeword assignment is no longer determined by distance between the original feature and codewords. Nevertheless, the additional parameters introduced in the cascade of fully-connected layers make DPQ more prone to over-fitting.

In this paper, we also attempt to incorporate the product quantization in a neural network and train it in an end-to-end manner. We construct a soft product quantization layer through an approximation function $f(\mathbf{x}, \alpha)$, which is differentiable and supports an end-to-end training. The original product quantization function $q(\mathbf{x})$ is a special case of the constructed approximation function when $\alpha \rightarrow +\infty$. Different from DPQ, fully-connected layers are no longer needed for codebook assignment, instead, in our method, the codeword assignment is determined by the similarity between features and the codewords. Therefore, we significantly reduce the number of parameters to be trained, making our product quantization network (PQN) less prone to over-fitting compared with DPQ. Meanwhile, inspired by the success of the triplet loss in metric learning and the advantage of the asymmetric similarity measurement in feature compression, we propose a novel asymmetric triplet loss to directly optimize the representation’s adaptability to asymmetric similarity measurement in an end-to-end manner. Interestingly, our experiments show that product quantization not also boosts retrieval efficiency, but also serves as a regularization approach, which improves the model’s generalizability and improves the retrieval precision.

Meanwhile, we revisit the residual quantization and extend PQN to residual product quantization network (RPQN). In general, residual quantization cannot achieve comparable distortion error as product quantization, which limits its effectiveness in retrieval based on hand-crafted features. Nevertheless, there are large redundancy in features obtained from deep neural network and thus the quantization error is less important. Benefited from residual learning triggered through residual quantization, the proposed RPQN achieves a higher accuracy than PQN. Fig. 1 visualizes the architecture of the proposed RPQN. It takes a triplet consisting of an anchor image, a positive image and a negative image $[I, I_+, I_-]$ as input. The backbone network, *e.g.*, AlexNet, obtains the images’ real-value feature vectors $[\mathbf{v}, \mathbf{v}_+, \mathbf{v}_-]$. The feature vectors of the positive image and negative image are further feed in the proposed residual product quantization layer (RPQL) to generate

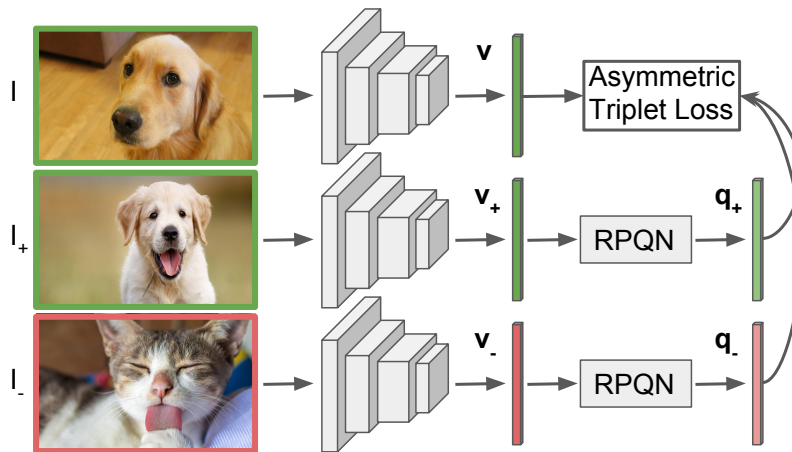


Fig. 1: The architecture of the proposed residual product quantization network (RPQN). It takes triplets as input. Each triplet consists of an anchor image I , a positive image I_+ and a negative image I_- . It obtains real-value feature vectors $[\mathbf{v}, \mathbf{v}_+, \mathbf{v}_-]$ through backbone and further generates soft-quantized representations for positive image and negative image, $[\mathbf{q}_+, \mathbf{q}_-]$, by residual product quantization layer (RPQN). $[\mathbf{v}, \mathbf{q}_+, \mathbf{q}_-]$ constructs asymmetric triplet loss to train the network.

their quantized features $[\mathbf{q}_+, \mathbf{q}_-]$. The anchor image’s real-value feature and the quantized features of the negative image and the positive image constitutes a triplet $[\mathbf{v}, \mathbf{q}_+, \mathbf{q}_-]$ to compute the asymmetric loss which aims to enlarge the distance between \mathbf{v} and \mathbf{q}_- , and decrease the distance between \mathbf{v} and \mathbf{q}_+ .

Moreover, we extend PQN to temporal product quantization network (TPQN) by exploiting the temporal consistency inherited in videos through a temporal pyramid pooling. The temporal pyramid pooling not only enhances the discriminativeness of the video representation but also provides a natural feature partition for product quantization. It integrates frame feature learning, frame features aggregation and video feature product quantization in a single neural network.

This paper is an extension of our conference paper [64]. New contributions are summarized as follows:

- We give a deeper insight into making the quantization differentiable. We propose to construct a function to approximate the hard quantization to achieve the functionality of the quantization. The softmax function used in the original conference paper [64] is a special case of the proposed approximation function in this paper. We give a more detailed derivation for the gradient back-propagation and conduct a more comprehensive ablation study.
- We extend the proposed product quantization network (PQN) in the conference paper to residual product quantization network (RPQN) by revisiting the residual quantization. The residual quantization not only brings a finer feature space partition but also triggers the residual learning, enhancing the

representation’s discriminability. We conduct comprehensive experiments on RPQN and demonstrate that it achieves higher accuracies than the PQN proposed in our conference paper.

- We tackle a new task, video retrieval. By exploiting the temporal consistency inherited in videos, we extend the PQN proposed in our conference paper to temporal product quantization (TPQN). The proposed TPQN integrates frame feature learning, frame features pooling and video feature compression in a single network. We conduct systematic experiments based on the proposed TPQN and achieve state-of-the-art performance in video retrieval task.

The remainder of the paper is organized as follows: Related work for Hashing, quantization and fast video retrieval are discussed in Section 2. Product quantization network (PQN) is introduced in Section 3. The proposed residual product quantization network (RPQN) is introduced in Section 4 and temporal product quantization network (TPQN) is introduced in Section 5. Section 6 and Section 7 provide experimental results in fast image retrieval and video retrieval, respectively.

2 Related Work

2.1 Hashing

Hashing [10, 45, 54, 15, 57, 31, 35, 32, 22, 21, 20, 61] aims to map a feature vector into a short code consisting of a sequence of bits, which enables fast distance computation

as well as small memory cost. One of traditional hashing methods, locality sensitivity hashing (LSH) [10], directly utilizes random projection to map the real-value vectors into binary codes. Since the random projection in LSH is independent with data distribution, it might not be optimal for a specific dataset. To optimize the projection for a specific dataset, spectral hashing (SH) [54] and iterative quantization (ITQ) [15] learn the projection by minimizing the distortion errors in an unsupervised manner. They achieve better performance than LSH. To further exploit the supervision information, some supervised hashing methods [36, 41, 47] are proposed, achieving better performance than their unsupervised counterpart such as ITQ and SH.

Nevertheless, the above-mentioned hashing methods all adopt a two-step procedure. They first obtain real-value image features and then compress the features into binary codes. In that case, the representation learning and feature compression are conducted separately and the mutual influence between them is ignored. Recently, motivated by the success of deep learning, some works [57, 31, 35, 32] propose deep hashing methods by incorporating hashing as a layer into a deep neural network. The end-to-end training mechanism of deep hashing simultaneously optimizes the representation learning and feature compression, achieving better performance than traditional hashing methods based on the two-step procedure.

2.2 Product Quantization

Since the hashing methods are only able to produce a few distinct distances, it has limited capability of describing the distance between data points. Parallely, another scheme termed product quantization (PQ) [25] possesses a more powerful representation capability. It decomposes the space into a Cartesian product of subspaces and quantizes each subspace individually. Thanks to the product settings, it could partition the feature space into a huge number of fine cells. By utilizing the asymmetric similarity measurement, it only causes quantization error in reference images' features and uses the original feature of the query without any distortion. Meanwhile, through looking up tables, it achieves comparable search speed as Hashing methods. Some following work [14, 1, 66] further optimize the product quantization through reducing the distortion and achieve higher retrieval precision and recall. Note that, production quantization and its optimized versions such as OPQ [14], AQ [1] and CQ [66] are designed for unsupervised scenarios where no supervision is provided, limiting their effectiveness in the supervised scenario.

To exploit the labels provided in the supervised scenario, Wang *et al* [53] propose supervised quantization (SQ). Nevertheless, SQ conducts feature extraction and quantization individually, whereas the interaction between these two steps are ignored. To simultaneously learn image representation and product quantization, deep quantization network (DQN) [8] adds a fully connected bottleneck layer in the convolutional network. It optimizes a combined loss consisting of a similarity-preserving loss and a product quantization loss. Nevertheless, the codebook in DPQ is trained through k-means clustering and thus the supervision is ignored. Recently, deep product quantization (DPQ) [28] is proposed where the codebook and network parameters are learned in an end-to-end manner. Different from original product quantization which determines the codeword assignment according to distances between features and codewords, DPQ determines the codeword assignment through a fully connected layer with parameters learned from data. Nevertheless, the additional parameters in the cascade of fully connected layers make the network more prone to over-fitting.

Our work is also an attempt of incorporating product quantization in a neural network. We propose a soft product quantization layer and build our product quantization network (PQN), which can be trained in an end-to-end manner. Different from DPQ, our PQN determines the codeword assignment according to the similarity between features for coding and codewords, which can be seen as a soft extension of original product quantization. Unlike DPQ, we do not need additional fully-connected layers to determine the codeword assignment and the only parameters in our soft product quantization layer are codewords. Therefore, ours is less prone to over-fitting. Besides, we exploit residual quantization besides product quantization in RPQN. The residual quantization triggers the residual learning, which boosts the representation's discriminability. Moreover, we extend product quantization to temporal product quantization for speeding up video retrieval. The temporal pyramid used in temporal product quantization not only provides a natural feature partition but also enhances the representation's discriminability.

2.3 Fast Video Search

Traditional fast video search methods [7, 59] follow a two-step pipeline by extracting the video feature [50, 49] followed by the feature compression. Nevertheless, this two-step pipeline ignores the interaction between two steps, feature learning and compression, and thus the obtained compact video representation might be sub-optimal. To improve the performance of learned com-

pact code, Wu *et al.* [56] propose a deep video hashing method which jointly optimize frame feature learning and hashing. Similarly, Liong *et al.* [34] incorporate video feature learning and hashing function optimization in a single neural network. Recently, Liu *et al.* [37] propose a deep video hashing framework as well as a category mask to increase the discriminativity of hash codes. Different from these deep hashing methods, we exploit product quantization to obtain a compact video representation and incorporate it in a neural network. Thanks to asymmetric distance measurement, product quantization results in a smaller distortion error than hashing. Moreover, we exploit the temporal consistency inherited in videos and extend the product quantization to temporal product quantization, constructing the proposed temporal product quantization network (TPQN).

3 Product Quantization Network

3.1 Limitation of Product Quantization

Let us denote by $\mathbf{v} \in \mathbb{R}^d$ the feature of a reference image I , we divide the feature \mathbf{v} into M subvectors $[\mathbf{v}_1, \dots, \mathbf{v}_m, \dots, \mathbf{v}_M]$ in the feature space where $\mathbf{v}_m \in \mathbb{R}^{d/M}$ is a subvector. The product quantization further approximates \mathbf{v} by

$$\mathbf{q} = [q_1(\mathbf{v}_1), \dots, q_m(\mathbf{v}_m), \dots, q_M(\mathbf{v}_M)], \quad (1)$$

where $q_m(\cdot)$ is m -th quantizer defined as

$$q_m(\mathbf{v}_m) = \mathbf{c}_m^{k^*}, \quad (2)$$

in which,

$$k^* = \operatorname{argmax}_{k \in [1, K]} \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle. \quad (3)$$

In Eq. (3), $\{\mathbf{c}_m^1, \dots, \mathbf{c}_m^K\}$ are codewords for $q_m(\cdot)$. Originally, the codewords are learned through unsupervised k-means by minimizing the distortion error between each data point and its nearest codeword. Nevertheless, only minimizing distortion errors between data points and codewords ignores the supervision information. To exploit the provided supervision when incorporated product quantization in a neural network, we seek to learn the codewords through back-propagation in an end-to-end manner.

Let us attempt to derive the back-propagation of gradients in product quantization operation. We define L as the training loss, which we will introduce in Section 3.5. Let us assume we have already obtained

$\partial L / \partial q_m(\mathbf{v}_m)$, *i.e.*, the derivative of loss \mathcal{L} with respect to the output. The back-propagation seeks to compute the derivative of loss \mathcal{L} with respect to the codewords, $\partial L / \partial \mathbf{c}_m^{k^*}$, and the derivative of loss \mathcal{L} with respect to the input, $\partial L / \partial \mathbf{v}_m$. From the definition, it is straightforward to derive that $\partial L / \partial \mathbf{c}_m^{k^*} = \partial L / \partial q_m(\mathbf{v}_m)$. Intuitively, it back-propagates the gradients of the output to the codewords it is assigned to and leave the other codewords unchanged. Nevertheless, $q_m(\mathbf{v}_m)$ is not a continuous function of \mathbf{v}_m , making it infeasible to back-propagate $\partial L / \partial q_m(\mathbf{v}_m)$ to $\partial L / \partial \mathbf{v}_m$. It drives us to seek other solutions to make the back-propagation in product quantization feasible.

3.2 From Hard Quantization to Soft Quantization

To overcome the non-differentiable problem, we generalize the quantization defined in Eq. (2) by constructing a continuously differential approximation function defined as $f_m(\mathbf{v}_m, \alpha)$ which satisfies $\lim_{\alpha \rightarrow +\infty} f_m(\mathbf{v}_m, \alpha) = q_m(\mathbf{v}_m)$, where α is a scalar controlling the consistency between $f_m(\mathbf{v}_m, \alpha)$ and $q_m(\mathbf{v}_m)$. In practice, we can choose a large α to achieve a good approximation. There are multiple choices to construct the function $f_m(\mathbf{v}_m, \alpha)$. Below we specify two instances and the soft quantization function proposed in our conference paper [64] corresponds to the second one:

$$\begin{aligned} f_m^1(\mathbf{v}_m, \alpha) &= \sum_{k=1}^K \frac{\langle \mathbf{v}_m, \mathbf{c}_m^k \rangle^\alpha}{\sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^\alpha} \mathbf{c}_m^k, \\ f_m^2(\mathbf{v}_m, \alpha) &= \sum_{k=1}^K \frac{e^{\alpha \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle}}{\sum_{k'=1}^K e^{\alpha \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle}} \mathbf{c}_m^k \end{aligned} \quad (4)$$

Note that, $\lim_{\alpha \rightarrow +\infty} f_m^1(\mathbf{v}_m, \alpha) = q_m(\mathbf{v}_m)$ might not be satisfied if $\langle \mathbf{v}_m, \mathbf{c}_m^k \rangle < 0$. To tackle this issue, we clip $\langle \mathbf{v}_m, \mathbf{c}_m^k \rangle$ to ensure it is non-negative. Below we derive the gradient back-propagation for $f_m^1(\mathbf{v}_m, \alpha)$ and that of $f_m^2(\mathbf{v}_m, \alpha)$ can be derived in a similar manner. To facilitate the derivation, we introduce a series of intermediate variables $\{a_m^k\}_{m=1, k=1}^{M, K}$ defined as

$$a_m^k = \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle^\alpha / \sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^\alpha. \quad (5)$$

Meanwhile, let us define $\mathbf{q}_m = f_m^1(\mathbf{v}_m, \alpha)$ as output of the approximation function taking \mathbf{v}_m as input. By plugging Eq. (5) into Eq. (4), we can obtain

$$\mathbf{q}_m = \sum_{k=1}^K a_m^k \mathbf{c}_m^k. \quad (6)$$

Since \mathbf{q}_m is a function of \mathbf{v}_m and $\{\mathbf{c}_m^k\}_{k=1}^K$, it is straightforward to obtain

$$\begin{aligned} dL &= \sum_{m=1}^M \left(d\mathbf{q}_m \right)^\top \frac{\partial L}{\partial \mathbf{q}_m} \\ &= \sum_{m=1}^M \left(\frac{\partial \mathbf{q}_m}{\partial \mathbf{v}_m} d\mathbf{v}_m + \sum_{k=1}^K \frac{\partial \mathbf{q}_m}{\partial \mathbf{c}_m^k} d\mathbf{c}_m^k \right)^\top \frac{\partial L}{\partial \mathbf{q}_m}. \end{aligned} \quad (7)$$

Meanwhile, according to the definition of \mathbf{q}_m in Eq. (6),

$$\begin{aligned} \frac{\partial \mathbf{q}_m}{\partial \mathbf{v}_m} &= \sum_{k=1}^K \frac{\partial \mathbf{q}_m}{\partial a_m^k} \left(\frac{\partial a_m^k}{\partial \mathbf{v}_m} \right)^\top = \sum_{k=1}^K \mathbf{c}_m^k \left(\frac{\partial a_m^k}{\partial \mathbf{v}_m} \right)^\top, \\ \frac{\partial \mathbf{q}_m}{\partial \mathbf{c}_m^k} &= \sum_{k''=1}^K \frac{\partial \mathbf{q}_m}{\partial a_m^{k''}} \left(\frac{\partial a_m^{k''}}{\partial \mathbf{c}_m^k} \right)^\top + a_m^k \frac{\partial \mathbf{c}_m^k}{\partial \mathbf{c}_m^k} \\ &= \sum_{k''=1}^K \mathbf{c}_m^{k''} \left(\frac{\partial a_m^{k''}}{\partial \mathbf{c}_m^k} \right)^\top + a_m^k \mathbf{I}, \end{aligned} \quad (8)$$

where \mathbf{I} is an identity matrix. By plugging Eq. (8) into Eq. (7), we further obtain

$$\begin{aligned} dL &= \sum_{m=1}^M \left\{ \sum_{k=1}^K \mathbf{c}_m^k \left(\frac{\partial a_m^k}{\partial \mathbf{v}_m} \right)^\top d\mathbf{v}_m \right. \\ &\quad \left. + \sum_{k=1}^K \left[\sum_{k''=1}^K \mathbf{c}_m^{k''} \left(\frac{\partial a_m^{k''}}{\partial \mathbf{c}_m^k} \right)^\top + a_m^k \mathbf{I} \right] d\mathbf{c}_m^k \right\}^\top \frac{\partial L}{\partial \mathbf{q}_m}. \end{aligned} \quad (9)$$

Meanwhile, L is the function of \mathbf{v}_m and \mathbf{c}_m^k , therefore,

$$dL = \sum_{m=1}^M \left[\left(d\mathbf{v}_m \right)^\top \frac{\partial L}{\partial \mathbf{v}_m} + \sum_{k=1}^K \left(d\mathbf{c}_m^k \right)^\top \frac{\partial L}{\partial \mathbf{c}_m^k} \right]. \quad (10)$$

Comparing Eq. (10) with Eq. (9), we obtain

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{v}_m} &= \sum_{k=1}^K \frac{\partial a_m^k}{\partial \mathbf{v}_m} \left(\mathbf{c}_m^k \right)^\top \frac{\partial L}{\partial \mathbf{q}_m}, \\ \frac{\partial L}{\partial \mathbf{c}_m^k} &= \sum_{k''=1}^K \frac{\partial a_m^{k''}}{\partial \mathbf{c}_m^k} \left(\mathbf{c}_m^{k''} \right)^\top \frac{\partial L}{\partial \mathbf{q}_m} + a_m^k \frac{\partial L}{\partial \mathbf{q}_m}. \end{aligned} \quad (11)$$

where

$$\begin{aligned} \frac{\partial a_m^k}{\partial \mathbf{v}_m} &= \frac{\alpha \sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^{\alpha-1} \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle^{\alpha-1} \mathbf{c}_m^k \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle}{\left(\sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^\alpha \right)^2} \\ &\quad - \frac{\alpha \sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^{\alpha-1} \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle^{\alpha-1} \mathbf{c}_m^{k'} \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle}{\left(\sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^\alpha \right)^2}, \\ \frac{\partial a_m^{k''}}{\partial \mathbf{c}_m^k} &= \frac{\alpha \mathbb{I}(k=k'') \mathbf{c}_m^{k''} \langle \mathbf{v}_m, \mathbf{c}_m^{k''} \rangle^{\alpha-1} \sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^\alpha}{\left(\sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^\alpha \right)^2} \\ &\quad - \frac{\alpha \mathbf{c}_m^k \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle^{\alpha-1} \langle \mathbf{v}_m, \mathbf{c}_m^{k''} \rangle^\alpha}{\left(\sum_{k'=1}^K \langle \mathbf{v}_m, \mathbf{c}_m^{k'} \rangle^\alpha \right)^2}. \end{aligned}$$

(12)

$\mathbb{I}(k=k') = 1$ only if $k=k'$ otherwise 0. In fact, our experiments in Section 6.1.1 show that, $f_m^1(\cdot, \alpha)$ and $f_m^2(\cdot, \alpha)$ achieve comparable performance. For convenience of implementation, we select $f_m^2(\cdot, \alpha)$ as default approximation function.

3.3 Regularization

Interestingly, the quantization mechanism works as a regularization, effectively suppressing over-fitting. Different from traditional regularization method putting constraints on the weights of the network such as ℓ_1/ℓ_2 regularization, quantization, instead, puts constraints on the activations. We will show that even though the quantized features can be seen as an approximation of the original features with inevitable distortion, it achieves a much better retrieval precision than that using the original features.

3.4 Initialization

We initialize the parameters of convolutional layers by fine-tuning a standard convolutional neural network without quantization, *e.g.*, Alexnet, on the specific dataset. Since we adopt inner-product similarity, we add an intra-normalization layer after the fully-connected layer to fine-tune the network to make it compatible with the proposed product quantization network. After that, we extract the features from the fine-tuned neural network and conduct k-means followed by ℓ_2 -normalization to obtain the initialized codewords $\{\mathbf{c}_{mk}\}_{k=1, m=1}^{K, M}$ in the soft product quantization layer.

3.5 Asymmetric Triplet Loss

We extend triplet loss originally used in metric learning to asymmetric triplet loss to boost the performance of learned representation in asymmetric similarity measurement. We define (I, I_+, I_-) as a training triplet, where I_+ represents a relevant (positive) image and I_- is an irrelevant (negative) image with respect to I . We denote by \mathbf{v} the feature of I before soft product quantization and denote by \mathbf{q}_+ and \mathbf{q}_- the features of I_+ and I_- after soft product quantization. We define asymmetric similarity between I and I_+ as $\langle \mathbf{v}, \mathbf{q}_+ \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner-product between two vectors. The proposed asymmetric triplet loss is defined as

$$\mathcal{L}_{ATL} = \langle \mathbf{v}, \mathbf{q}_- \rangle - \langle \mathbf{v}, \mathbf{q}_+ \rangle. \quad (13)$$

Intuitively, it aims to increase the asymmetric similarity between pairs of relevant images and decrease that of pairs of irrelevant images. It is a natural extension of the original triplet loss to asymmetric distance. The difference is that, a training triplet used in the original triplet loss consists of three features of the same type, whereas a training triplet used in the proposed asymmetric triplet loss consists of one feature without quantization and two features after quantization. In fact, our experiments in Section 6.1.1 show that a better performance is achieved by processing the above loss through a sigmoid function and a revised loss function is defined as Eq. (14). The better performance might be attributed to the fact that the sigmoid function can normalize the original loss so that the training will not be biased by some samples with huge loss. By default, the asymmetric triplet loss (ATL) we mention in this paper is \mathcal{L}_{ATL}^+ in Eq. (14).

$$\mathcal{L}_{ATL}^+ = \frac{1}{1 + e^{\langle \mathbf{v}, \mathbf{q}^+ \rangle - \langle \mathbf{v}, \mathbf{q}^- \rangle}}. \quad (14)$$

3.6 Encoding and retrieval

After training the proposed product quantization network, the reference images in the database will be encoded by hard product quantization. We define the layer before the soft product quantization layer as embedding layer. Given a reference image I , we obtain its feature from embedding layer $\mathbf{v} = [\mathbf{v}_1, \dots, \mathbf{v}_m, \dots, \mathbf{v}_M]$ and then obtain its PQ code $\mathbf{i} = [i_1, \dots, i_m, \dots, i_M]$ where i_m is computed by

$$i_m = \operatorname{argmax}_{k \in [1, K]} \langle \mathbf{v}_m, \mathbf{c}_m^k \rangle, \quad (15)$$

where $\{\mathbf{c}_m^k\}_{m=1, k=1}^{M, K}$ are codewords learned from our product quantization network. In the retrieval phase, we obtain the query's feature from the embedding layer $\mathbf{v}^q = [\mathbf{v}_1^q, \dots, \mathbf{v}_m^q, \dots, \mathbf{v}_M^q]$. The relevance between the query and a reference image represented by its product quantization code $\mathbf{i} = [i_1, \dots, i_m, \dots, i_M]$ is computed by the asymmetric similarity $s(\mathbf{v}^q, \mathbf{i})$ defined as

$$s(\mathbf{v}^q, \mathbf{i}) = \sum_{m=1}^M \langle \mathbf{v}_m^q, \mathbf{c}_m^{i_m} \rangle. \quad (16)$$

Since $\{\langle \mathbf{v}_m^q, \mathbf{c}_m^k \rangle\}_{k=1}^K$ is computed only once for all the reference images in the database and thus obtaining $s(\mathbf{v}^q, \mathbf{i})$ only requires to sum up the pre-computed similarity scores in the look-up table, considerably speeding up the image retrieval process. Meanwhile, storing the

product quantization code \mathbf{i} only requires $M \log_2 K$ bits, which considerably reduces the memory cost.

3.7 Relation to existing methods

DQN [8] is the first attempt of incorporating product quantization in the neural network. It alternately optimizes codewords and other parameters of the network. It is worth noting that when updating codewords, it only minimizes the quantization errors through k-means and the supervision information is ignored.

SUBIC [24] integrates the one-hot block encoding layer in the deep neural network. It represents each image by a product of one-hot blocks, following the spirit of product quantization. Nevertheless, the sparse property limits its representation capability, making it perform not as well as ours as shown in Table 11.

DPQ [28] is another attempt of incorporating product quantization into the neural network. It determines the codeword assignment through a cascade of two fully-connected layers. In contrast, our method determines the codeword assignment according to the similarity between original features and the codewords. Note that, the additional parameters from these two fully-connected layers in DPQ not only increase the computation complexity in training the neural network, but also make the network more prone to over-fitting. Our experiments show that our proposed PQN considerably outperforms DPQ.

4 Residual Product Quantization Network

Different from PQ, residual quantization (RQ) [11] performs quantization on the entire feature space, and then recursively applies vector quantization (VQ) to the residuals of the previous quantization level. It is a stacked quantization model. In other words, different from PQ conducting VQ independently on several partitioned sub-spaces, RQ conducts VQ on multiple levels instead. For the first level, RQ simply applies VQ to quantize a feature vector \mathbf{v} into one of K centroids through $q_0(\mathbf{v})$. The feature vector's residual vector \mathbf{r}_1 can be obtained by subtracting its corresponding centroid as $\mathbf{v} - q_0(\mathbf{v})$. After that, \mathbf{r}_1 will be further quantized into K centroids through $q_1(\mathbf{r}_1)$. By repeatedly quantizing the residuals $M_r - 1$ times, we can approximate \mathbf{v} by $q_0(\mathbf{v}) + \sum_{m=1}^{M_r-1} q_m(\mathbf{r}_m)$. Through combining codewords in M_r levels, we obtain a considerably huge codebook consisting of K^{M_r} codewords. Generally, residual quantization cannot achieve comparable distortion error as

product quantization, which limits its effectiveness in retrieval based on off-the-shelf features.

Nevertheless, in a deep learning network, there existing huge amount of redundancy in activations and the quantization error is less important. Meanwhile, we observe that the residual quantization naturally triggers residual learning [18], which can boost the features' discriminability. Based on the above motivation, we attempt to integrate the residual quantization in a deep neural network. We demonstrate the advantage of residual quantization over product quantization when incorporated into a deep neural network. In fact, the residual quantization in our framework not only minimizes the distortion error caused by the quantization, but more importantly, it triggers residual learning and enhances the discriminability of the learned compact representation.

4.1 Residual Product Quantization Layer

Fig. 2 illustrates the proposed residual product quantization layer (RPQL). In this case, we set the residual level $M_r = 2$. We denote by $\mathbf{v} \in \mathbb{R}^d$ the input feature vector. RPQL divides \mathbf{v} into M_p sub-vectors:

$$\mathbf{v} \rightarrow [\mathbf{v}_1, \dots, \mathbf{v}_m, \dots, \mathbf{v}_{M_p}] \quad (17)$$

For each sub-vector \mathbf{v}_m , it goes through the following forward path:

1. $\mathbf{q}_m^1 \leftarrow f(\mathbf{v}_m, \alpha)$
2. $\mathbf{r}_m^1 \leftarrow \mathbf{v}_m - \mathbf{q}_m^1$
3. $\mathbf{q}_m^2 \leftarrow f(\mathbf{r}_m^1, \alpha)$
4. $\mathbf{q}_m \leftarrow \mathbf{q}_m^1 + \mathbf{q}_m^2$

where $f(\cdot, \alpha)$ is the soft function defined in section 3.

4.2 Indexing and Retrieval

After training the network, in the indexing phase, we encode each reference image I in the database through hard-assignment residual product quantization. Let us define the layer before residual product quantization layer as embedding layer. We denote by \mathbf{v} the output of embedding layer when input is I . \mathbf{v} is partitioned into $[\mathbf{v}_1, \dots, \mathbf{v}_m, \dots, \mathbf{v}_{M_p}]$ and each sub-vector \mathbf{v}_m is indexed as follows:

1. $i_m^1 \leftarrow \operatorname{argmax}_{k \in [1, K]} \langle \mathbf{x}_m, \mathbf{c}_m^{1, k} \rangle$
2. $\mathbf{r}_m^1 \leftarrow \mathbf{x}_m - \mathbf{c}_m^{1, i_m^1}$
3. $i_m^2 \leftarrow \operatorname{argmax}_{k \in [1, K]} \langle \mathbf{r}_m^1, \mathbf{c}_m^{2, k} \rangle$

In the indexing phase, for each reference image I , we only need to store indices of their corresponding code-words $\{i_m^1\}_{m=1}^{M_p}$ and $\{i_m^2\}_{m=1}^{M_p}$, taking $2M_p \log_2 K$ bits in total. In the retrieval phase, we utilize the asymmetric similarity scores to rank the reference images in the dataset. Let denote by \mathbf{v}^q the output of embedding layer when the input is query image q . Let denote by $[i_1^1, \dots, i_{M_p}^1]$ and $[i_1^2, \dots, i_{M_p}^2]$ indices in two-level residual product quantization of the reference image I . The asymmetric similarity is defined as

$$\text{sim}_{asym}(q, I) = \sum_{m=1}^{M_p} [\langle \mathbf{v}_q, \mathbf{c}_m^{1, i_m^1} \rangle + \langle \mathbf{v}_q, \mathbf{c}_m^{2, i_m^2} \rangle]. \quad (18)$$

In this case, for each query, we only need to compute its similarity with all the codewords for only once and then similarity between the query and each reference image can be efficiently obtained through $2M_p$ table look-ups.

5 Temporal Product Quantization Network

The proposed PQN and RPQN in the previous section are designed for fast image retrieval and are not optimal for video retrieval task. In this section, we introduce our temporal product quantization network (TPQN) for effective and efficient video retrieval. The proposed TPQN is an extension of the proposed product quantization network by further exploiting the temporal consistency inherited in videos. It integrates frame-wise feature learning, frame-wise features aggregation and video-level feature quantization in a single neural network, and supports an end-to-end training. Fig. 3 visualizes the architecture of the proposed TPQN. In Section 5.1 and 5.2, we will introduce two core modules of TPQN sequentially. In Section 5.3, the indexing and retrieval strategy will be introduced.

5.1 Convolutional and pooling layers

Given a video V , we uniformly sample N frames from it and define a global set of sampled frames as $\mathcal{S} = \{f_1, \dots, f_N\}$. We feed frames $\{f_i\}_{i=1}^N$ in parallel into the backbone network consisting of convolutional and pooling layers and obtain a set of frame-level features $\{F(f_i) | f_i \in \mathcal{S}\}$. We uniformly partition the global set of frames \mathcal{S} into T subsets $\{\mathcal{S}^t\}_{t=1}^T$ where $\mathcal{S}^t = \{f_i | i \in [\lfloor N/T \rfloor(t-1) + 1, \lfloor N/T \rfloor t]\}$. Each subset \mathcal{S}^t represents a continuous interval of the video. Features of frames in each subset \mathcal{S}^t are max-pooled into an interval's feature:

$$\mathbf{v}^t = \max\text{pool}(\{F(f) | f \in \mathcal{S}^t\}), \quad t = 1, \dots, T+1 \quad (19)$$

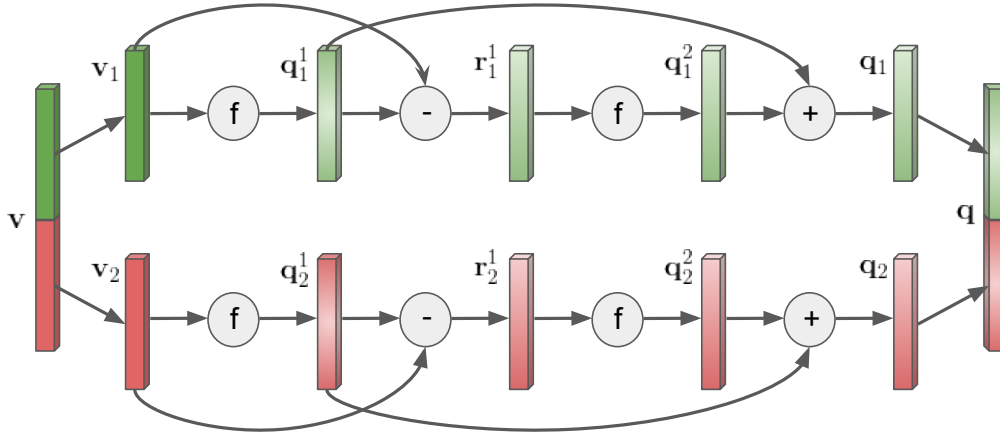


Fig. 2: The architecture of the proposed residue product quantization layer. It splits the real-value feature \mathbf{v} to sub-features $[\mathbf{v}_1, \mathbf{v}_2]$. Through approximation function $f(\cdot, \alpha)$, each sub-feature \mathbf{v}_m is quantized into \mathbf{q}_m^1 and then obtains the first-level residual vector $\mathbf{r}_m^1 \leftarrow \mathbf{v}_m - \mathbf{q}_m^1$. Each residual vector \mathbf{r}_m^1 is further quantized into \mathbf{q}_m^2 . After that, \mathbf{q}_m^1 and \mathbf{q}_m^2 are summed up to obtain the \mathbf{q}_m . \mathbf{q}_1 and \mathbf{q}_2 are concatenated into \mathbf{q} .

where $\mathcal{S}^{T+1} = \mathcal{S}$ accounts for the global set containing all sampled frames. All the interval-level features will be concatenated into a global feature $\mathbf{v} = [\mathbf{v}^1, \dots, \mathbf{v}^{T+1}]$, which is the input of the proposed temporal product quantization layer. Intuitively, \mathbf{v} is a two-level temporal pyramid pooling feature which exploits the temporal consistency inherited in the videos. But more importantly, the pyramid structure provides a natural feature partition for product quantization.

5.2 Temporal Product Quantization Layer

For each interval-level feature \mathbf{v}^t , following the standard product quantization, we equally split it into M sub-vectors as $\mathbf{v}^t = [\mathbf{v}_1^t, \dots, \mathbf{v}_M^t]$. Each \mathbf{v}_m^t goes through the following pipeline consisting of 4 steps:

1. $\mathbf{y}_{tm} = \text{relu}(\mathbf{W}_m^t \mathbf{v}_m^t + \mathbf{b}_m^t)$
2. $\bar{\mathbf{y}}_m^t = \frac{\mathbf{y}_{tm}}{\|\mathbf{y}_{tm}\|_2}$
3. $\mathbf{z}_m^t = f_m^t(\bar{\mathbf{y}}_m^t, \alpha)$
4. $\bar{\mathbf{z}}_m^t = \beta \frac{\mathbf{z}_m^t}{\|\mathbf{z}_m^t\|_2}$

where \mathbf{W}_m^t and \mathbf{b}_m^t in step 1 are parameters of a fully-connected layer used to further boost the feature's discriminability. β in step 4 is a scalar to speed up the training and quantizer $f_m^t(\bar{\mathbf{y}}_m^t, \alpha)$ is defined as Eq. 4.

5.3 Indexing and Retrieval

After training the neural network, we will use the learned codewords but adopt hard quantization for indexing. To be specific, given a reference video V in the database, let $\mathbf{v}^t = [\mathbf{v}_1^t, \dots, \mathbf{v}_m^t, \dots, \mathbf{v}_M^t]$ denote the max-pooled

feature of the intervals I_t where \mathbf{v}_m^t is the m -th sub-vector. For each \mathbf{v}_m^t , we obtain its index i_m^t by

1. $\mathbf{y}_m^t = \text{relu}(\mathbf{W}_m^t \mathbf{b}_m^t + \mathbf{b}_m^t)$
2. $\bar{\mathbf{y}}_m^t = \frac{\mathbf{y}_m^t}{\|\mathbf{y}_m^t\|_2}$
3. $i_m^t = \underset{k \in [1, K]}{\text{argmax}} \langle \bar{\mathbf{y}}_m^t, \mathbf{c}_m^{t,k} \rangle$

where K is the number of codewords used by each quantizer $q_m^t(\cdot)$. It takes $\log_2(K)$ bits to store i_m^t and $(T+1)M \log_2(K)$ bits to store all indices $\{[i_1^t, \dots, i_M^t]\}_{t=1}^{T+1}$ for the video V . The advantage of the proposed temporal product quantization over the original product quantization is that it exploits the temporal consistency inherited in the video, significantly boosting the representation's discriminability.

In the retrieval phase, given a query video q , we can directly obtain its concatenated max-pooled feature $[\mathbf{v}_{q,1}^1, \dots, \mathbf{v}_{q,M}^{T+1}]$ through the trained backbone network, further post-process each $\mathbf{v}_{q,m}^t$ through $\{\mathbf{W}_m^t, \mathbf{b}_m^t\}$ followed by ℓ_2 normalization and obtain the final feature $[\bar{\mathbf{y}}_{q,1}^1, \dots, \bar{\mathbf{y}}_{q,M}^{T+1}]$. Note that we do not conduct quantization on the query's feature due to the asymmetric similarity measurement. The similarity between the query q and a reference video V is computed by

$$\text{sim}(q, V) = \sum_{t=1}^{T+1} \sum_{m=1}^M \langle \bar{\mathbf{y}}_{q,m}^t, \mathbf{c}_m^{t, i_m^t} \rangle. \quad (20)$$

Computing Eq. (20) is considerably efficient by utilizing look-up table [25]. To be specific, we only need to compute the similarity between query video's feature with each codeword $\langle \bar{\mathbf{y}}_{q,m}^t, \mathbf{c}_m^{t,k} \rangle$ for one time and then store the similarity in a look-up table. Computing the

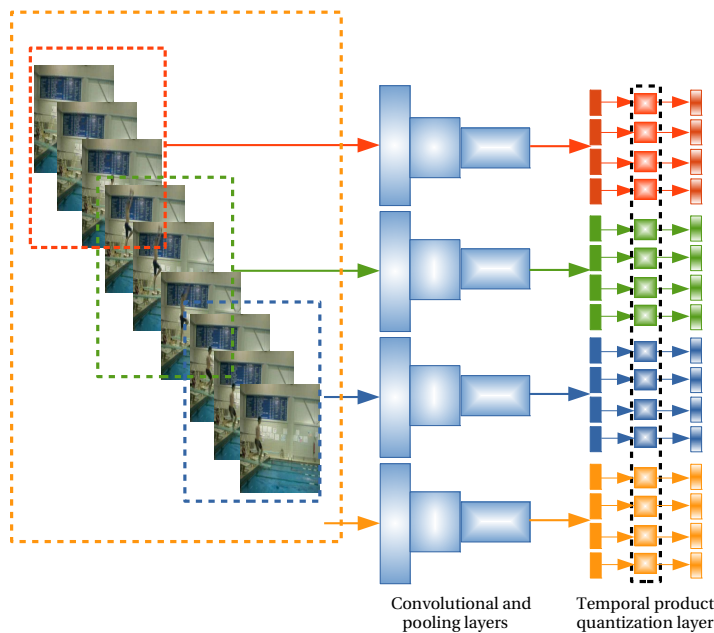


Fig. 3: The architecture of the proposed temporal product quantization network (TPQN). The input video V is segmented into multiple intervals. An interval’s representation is obtained through convolutional and pooling layers and the video’s feature is obtained through temporal pyramid pooling. The temporal product quantization layer takes the obtained representation of the video as input and further conducts soft product quantization.

similarity between the query video q and a reference video V defined in Eq. (20) only need $M \times (T + 1)$ times similarity table look-ups.

We summarize architectures of PQN, RPQN, and TPQN in Table 1.

Table 1: Comparisons between PQN, RPQN, and TPQN. PQN adopts the product quantization architecture, RPQN exploits the residual quantization besides the product quantization, and TPQN utilizes the temporal structure besides product quantization.

| | Product | Residual | Temporal |
|------|---------|----------|----------|
| PQN | ✓ | | |
| RPQN | ✓ | ✓ | |
| TPQN | ✓ | | ✓ |

6 Experiments on Image Retrieval

We evaluate the performance of our PQN on two public benchmark datasets, CIFAR-10 and NUS-WIDE. CIFAR-10 [29] is a dataset containing 60,000 color images in 10 classes, and each class has 6,000 images in size 32×32 . Different from CIFAR-10, NUS-WIDE [12] is a dataset for evaluating multi-class classification, in which one sample is assigned to one or multiple labels.

We follow the settings in [31,8] and use the subset of 195,834 images that are associated with the 21 most frequent concepts, where each concept consists of at least 5,000 images. We resize all images into 256×256 .

On the CIFAR-10 dataset, the performance reported by different baselines are based on different base convolutional neural networks, making it unfair to directly compare their reported retrieval accuracy. To make a fair comparison, we evaluate our method based on two types of convolutional neural networks. The first convolutional neural network we use is 3CNet which is also used by SUBIC [24] and DPQ [28]. 3CNet is proposed in [35], which consists of $L = 3$ convolutional layers with 32, 32 and 64 filters of size 5×5 respectively, followed by a fully connected layer with $d = 500$ nodes. The second convolutional neural network we choose is AlexNet. It is worth noting that the baselines we compare may apply different models. For example, DQN [8] adopts AlexNet whereas other work [52,32] adopt VGG-F model. These two models are similar in the architecture. To be specific, both the CNN-F and AlexNet consist of five convolutional layers and two fully connected layers. As shown in [27], the CNN-F generally performs better than Alexnet in image retrieval, therefore, the better performance of ours based on AlexNet than existing state-of-art methods based on CNN-F is not owing to better base network. In other words, our method can achieve better performance even with

Table 2: Influence of M and K on PQN.

| M | 1 | 2 | 4 | 8 |
|--------------|-------|-------|--------------|-------|
| $K = 2^3$ | 0.539 | 0.650 | 0.741 | 0.708 |
| $K = 2^6$ | 0.696 | 0.741 | 0.782 | 0.724 |
| $K = 2^9$ | 0.712 | 0.750 | 0.787 | 0.713 |
| $K = 2^{12}$ | 0.735 | 0.763 | 0.786 | 0.737 |

Table 3: Influence of α on PQN.

| α | 1 | 5 | 20 | 40 | 80 |
|--------------|-------|--------------|-------|-------|-------|
| $K = 2^3$ | 0.731 | 0.741 | 0.741 | 0.732 | 0.734 |
| $K = 2^6$ | 0.771 | 0.782 | 0.782 | 0.780 | 0.760 |
| $K = 2^9$ | 0.779 | 0.787 | 0.786 | 0.784 | 0.780 |
| $K = 2^{12}$ | 0.785 | 0.786 | 0.782 | 0.782 | 0.781 |

Table 4: Performance comparisons between \mathcal{L}_{ATL} in Eq. (13) and its sigmoid-version \mathcal{L}_{ATL}^+ in Eq. (14).

| | 12 bits | 24 bits | 36 bits | 48 bits |
|-----------------------|--------------|--------------|--------------|--------------|
| \mathcal{L}_{ATL} | 0.732 | 0.773 | 0.781 | 0.781 |
| \mathcal{L}_{ATL}^+ | 0.741 | 0.782 | 0.787 | 0.786 |

Table 5: Performance comparisons between $f_1(\cdot, \cdot)$ and $f_2(\cdot, \cdot)$ in Eq. (4).

| | 12 bits | 24 bits | 36 bits | 48 bits |
|---------------------|--------------|--------------|--------------|--------------|
| $f_1(\cdot, \cdot)$ | 0.745 | 0.784 | 0.785 | 0.783 |
| $f_2(\cdot, \cdot)$ | 0.741 | 0.782 | 0.787 | 0.786 |

an inferior base network. On the NUS-WIDE dataset, we also adopt AlexNet as our base model. On both datasets, we report the performance of the proposed method through mAP, which is a standard metric in evaluating the performance of retrieval algorithms.

6.1 CIFAR-10 using 3CNet

Following the experimental setup in SUBIC [24] and DPQ [28], the training is conducted on 50K image training set. The test set is split into 9K database images and 1K query images (100 per class).

6.1.1 Ablation study on PQN

We first evaluate the influence of the number of subvectors M and the number of codewords per sub-codebook K on the retrieval precision of PQN. In experiments,

Table 6: Comparisons with PQ and LSQ.

| | 8bits | 16bits | 24bits | 32bits |
|---------|--------------|--------------|--------------|--------------|
| TL+Full | 0.779 | | | |
| TL+PQ | 0.621 | 0.741 | 0.773 | 0.780 |
| TL+LSQ | 0.720 | 0.752 | 0.753 | 0.763 |
| PQN | 0.729 | 0.778 | 0.782 | 0.786 |

we change M among $\{1, 2, 4, 8\}$, and vary K among $\{2^3, 2^6, 2^9, 2^{12}\}$. As shown in Table 2, PQN achieves the best performance when $M = 4$. By default, we set $M = 4$. Note that when $M \in \{1, 2\}$, the performance of PQN increases as K increases. This is expected since the larger K can partition the feature space into finer cells. Nevertheless, when $M = 4$, the performance drops when K increases from 2^9 to 2^{12} . Meanwhile, when $M = 8$, there is also a performance drop when K increases from 2^6 to 2^9 . The worse performance might be caused by over-fitting when both M and K are large. α controls the quantization softness of the soft product quantization layer.

We further evaluate the performance of our method when α varies. We test the influence of α by fixing $M = 4$ and varying K among $\{2^3, 2^6, 2^9, 2^{12}\}$. As shown in Table 3, the performance of the proposed PQN is relatively stable when α increases from 1 to 80. Note that, when $\alpha = 1$, the performance is slightly worse than that when $\alpha = 5$. The worse performance is due to the fact a small α will make the quantization too soft and thus the soft quantization in training phase differs too much from the hard quantization in the testing phase. Meanwhile, we also observe a performance drop when α increases from 5 to 80. This drops might be caused by the fact that a huge α tends to push the input of soft-max function to the saturation region and lead to gradient vanishing. By default, we set $\alpha = 5$.

We compare the \mathcal{L}_{ATL} in Eq. (13) and its sigmoid-version \mathcal{L}_{ATL}^+ in Eq. (14). As shown in Table 4, \mathcal{L}_{ATL}^+ achieves slightly better performance than \mathcal{L}_{ATL} . The better performance might be attributed to the fact that the sigmoid function can balance the losses of different samples and thus the network training will not be dominated by samples generating huge loss.

We compare the performance of two different approximation function $f_1(\cdot, \cdot)$ and $f_2(\cdot, \cdot)$ in Eq. (4). As shown in Table 5, these two approximation functions achieve comparable performance. Considering the complexity of implementation, we select $f_2(\cdot, \cdot)$ as default approximation function.

We compare PQN with unsupervised PQ and LSQ [38] based on fine-tuned features trained through triplet loss. As shown in Table 6, ours considerably outperforms both TL+PQ and TL+LSQ. Meanwhile, we also show the performance of original features trained through triplet loss without quantization (TL+Full) in Table 6. Note that, to make a fair comparison, the triplet loss we used for full feature without quantization is also the revised one as Eq. (14). The performance of ours is even better than that of features without quantization, this is owing to the regularization imposed by quantization, which suppresses over-fitting.

We evaluate the proposed PQN in the open-set setting [44] on CIFAR-10 dataset. In the testing phase, the query and reference images are of different classes from that in the training phase. We train the PQN using samples with labels $c \in [0, 4]$ and conduct the retrieval using samples with labels in $c \in [5, 9]$. In the training data, each class contains 5000 samples. In the testing data, each class contains 1000 classes. Among 5000 testing samples, 1000 samples from the testing data are randomly selected as query and the rest 4000 samples are used as reference images. We compare with DTSH [52] and Triplet + PQ. To be specific, Triplet + PQ is implemented by training the backbone without quantization and then conducted the unsupervised PQ. As shown in Table 7, our method consistently outperforms the DTSH [52] and Triplet + PQ in the open-set setting.

Table 7: Comparisons with other methods in the open-set setting.

| | 8bits | 16bits | 24bits | 32bits |
|--------------|--------------|--------------|--------------|--------------|
| DTSH [52] | 0.261 | 0.269 | 0.297 | 0.341 |
| Triplet + PQ | 0.282 | 0.316 | 0.332 | 0.341 |
| DPQ | 0.329 | 0.349 | 0.357 | 0.356 |

We evaluate the proposed PQN using another two backbones, ResNet34 and ResNet50. The feature dimension of ResNet-34 is 512 and that of the ResNet-50 is 2048. As shown in Table 8, when $M \in \{1, 2\}$, the best performance is achieved when $K = 64$. On the other hand, when $M \in \{4, 8\}$, the best performance is achieved when $K = 16$. Note that, when $K = 256$, the performance is worse than that when $K = 64$. The worse performance might be caused by over-fitting.

We evaluate the influence of the sampling strategy of the triplet loss on the proposed PQN. We compare the uniform sampling, hard negative sampling and distance weighted sampling [55]. As shown in Table 9, the hard negative sampling is worse than uniform sampling, and the distance weighted sampling is slightly better than uniform sampling. Considering the effectiveness and simplicity, we use uniform sampling as default.

6.1.2 Ablation study on RPQN

We then evaluate the influence of residual level M_r on the performance of our RPQN. Note that, when $M_r = 1$, the residual product quantization will degenerate to the product quantization. We compare the case when $M_r = 1$ and that when $M_r = 2, 4$. Since the code length $L = M_p M_r \log_2 K$, to achieve an identical code length, we set $M_p = 4$ when $M_r = 1$, set $M_p = 2$ when

$M_r = 2$, and set $M_p = 1$ when $M_r = 4$. As shown in Table 10, the performance when $M_r = 2$ consistently outperforms others. By default, we set $M_r = 2$ on all testing dataset. Meanwhile, the better performance of $M_r = 2$ than $M_r = 1$ verifies the advantage of residual product quantization over product quantization.

We also evaluate the influence of M_p . In implementation, we fix $M_r = 2$, change M_p among $\{1, 2, 4\}$. We comprehensively test cases when K varies among $\{8, 64, 512, 4096\}$. As shown in Figure 4, when $M_p = 2$, it achieves the highest mAP on all cases and meanwhile it takes only a half number of bits of that when $M_p = 4$. By default, we set $M_p = 2$.

6.1.3 Compare with state-of-the-art methods.

We compare the proposed PQN and RPQN with two state-of-the-art methods (SUBIC and DPQ), which adopt the same 3CNet as well as the same experimental settings. We change bit length L among $\{12, 24, 36, 48\}$. We set $M = 4$ on PQN, and set $M_r = 2$ and $M_p = 2$ on RPQN. Since SUBIC adopts cross-entropy loss, it is unfair to directly compare it with ours using asymmetric triplet loss. Therefore, we report the performance of our PQN and RPQN based on the cross-entropy loss (CEL) as well as the proposed asymmetric triplet loss (ATL). As shown in Table 11, our PQN and RPQN based on both CEL and ATL significantly outperform the existing state-of-the-art methods including SUBIC and DPQ.

6.2 CIFAR-10 using AlexNet

Following the experimental settings in [52, 32], we randomly sample 1000 images per class (10000 images in total) as the testing query images, and the remaining 50000 images are used as the training set as well as reference images in the database. We set $M = 4$ on PQN, and set $M_r = 2$ and $M_p = 2$ on RPQN. We vary K among $\{2^4, 2^6, 2^9, 2^{12}\}$, and thus the code length L varies among $\{16, 24, 36, 48\}$.

6.2.1 Comparisons with state-of-the-art methods.

As shown in Table 12, ours consistently outperforms the existing state-of-the-art methods, especially when the bit length is small. For instance, when the bit length is 16, our PQN/RPQN achieves a 0.947/0.950 mAP whereas DSDH only achieves a 0.935 mAP.

Table 8: The influence of M and K on the ResNet34 and ResNet50 features.

| | ResNet34 (512-d) | | | | ResNet50 (2048-d) | | | |
|---------|------------------|----------|----------|-----------|-------------------|----------|----------|-----------|
| | $K = 4$ | $K = 16$ | $K = 64$ | $K = 256$ | $K = 4$ | $K = 16$ | $K = 64$ | $K = 256$ |
| $M = 1$ | 0.353 | 0.914 | 0.914 | 0.908 | 0.376 | 0.918 | 0.919 | 0.914 |
| $M = 2$ | 0.451 | 0.921 | 0.924 | 0.919 | 0.627 | 0.928 | 0.931 | 0.925 |
| $M = 4$ | 0.635 | 0.935 | 0.930 | 0.929 | 0.797 | 0.938 | 0.936 | 0.933 |
| $M = 8$ | 0.773 | 0.927 | 0.924 | 0.922 | 0.782 | 0.931 | 0.930 | 0.927 |

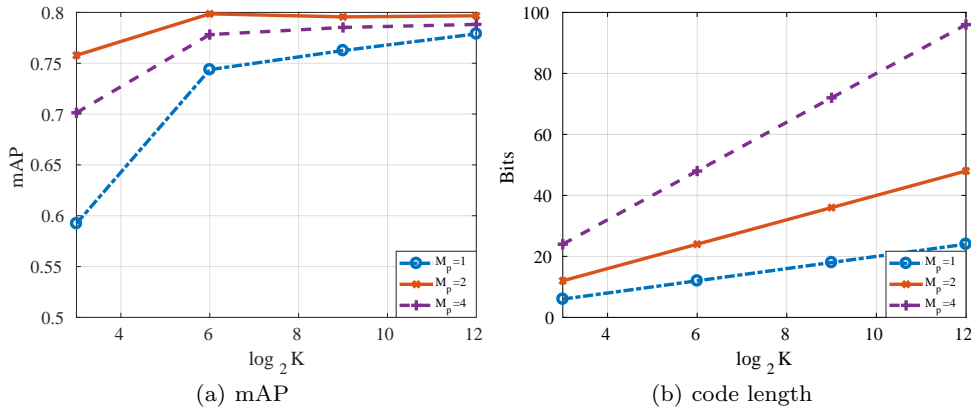
Fig. 4: The influence of M_p .

Table 9: Comparisons among different sampling methods.

| method | 8bits | 16bits | 24bits | 32bits |
|-------------------|-------|--------|--------|--------|
| Uniform | 0.729 | 0.778 | 0.782 | 0.787 |
| Hard Negative | 0.632 | 0.679 | 0.692 | 0.701 |
| Distance Weighted | 0.733 | 0.783 | 0.786 | 0.788 |

Table 10: Influence of M_r . When $M_r = 1$, the product residual product quantization degenerates into product quantization.

| M_r | 12 bits | 24 bits | 36 bits | 48 bits |
|-------|--------------|--------------|--------------|--------------|
| 1 | 0.741 | 0.782 | 0.787 | 0.786 |
| 2 | 0.758 | 0.799 | 0.796 | 0.797 |
| 4 | 0.711 | 0.762 | 0.765 | 0.776 |

Table 11: mAP comparisons with state-of-the-art methods using 3CNet.

| Method | 12 bits | 24 bits | 36 bits | 48 bits |
|------------|--------------|--------------|--------------|--------------|
| SUBIC [24] | 0.635 | 0.672 | 0.682 | 0.686 |
| DPQ [28] | 0.673 | 0.692 | 0.695 | 0.693 |
| PQN+CEL | 0.737 | 0.771 | 0.768 | 0.762 |
| RPQN+CEL | 0.742 | 0.785 | 0.784 | 0.786 |
| PQN+ATL | 0.741 | 0.782 | 0.787 | 0.786 |
| RPQN+ATL | 0.758 | 0.799 | 0.796 | 0.797 |

6.2.2 Extremely short code evaluation.

As shown in Table 12, the mAP achieved by our method does not drop when the bit length decreases from 48 to

Table 12: mAP comparisons with existing state-of-the-art methods using AlexNet base model on the CIFAR10 dataset.

| Method | 16 bits | 24 bits | 36 bits | 48 bits |
|------------|--------------|--------------|--------------|--------------|
| DRSCH [65] | 0.615 | 0.622 | 0.629 | 0.631 |
| DSCH [65] | 0.609 | 0.613 | 0.617 | 0.686 |
| DSRH [68] | 0.608 | 0.611 | 0.617 | 0.618 |
| VDSH [67] | 0.845 | 0.848 | 0.844 | 0.845 |
| DPSH [33] | 0.903 | 0.885 | 0.915 | 0.911 |
| DTSH [52] | 0.915 | 0.923 | 0.925 | 0.926 |
| DSDH [32] | 0.935 | 0.940 | 0.939 | 0.939 |
| PQN | 0.947 | 0.947 | 0.946 | 0.947 |
| RPQN | 0.950 | 0.949 | 0.949 | 0.948 |

16. In contrast, the performance of other methods in Table 12 all turn worse due to decrease of the bit length. To fully exploit the potential of the proposed product quantization network on the CIFAR-10 dataset, we evaluate it by setting the code length L extremely small. We vary M among 1, 2 and 4, and meanwhile vary the code length (bit number) L within $\{4, 6, 8, 10, 12\}$. As shown in Table 13, when code length is extremely small, *e.g.*, $L = 4$, the performance of PQN when $M = 1$ significantly outperforms that when $M = 2, 4$. Meanwhile, when $M = 1$, there is not significant performance drop when L decreases from 12 to 4. Note that, when $M = 1$, the proposed PQN achieves a 0.945 mAP when using only 4 bits per code. It considerably outperforms the existing state-of-the-art method DSDH [32] which only achieves 0.935 mAP using 16 bits.

Table 13: Evaluation on the extremely short code.

| L | 4 bits | 6 bits | 8 bits | 10 bits | 12 bits |
|---------|--------|--------|--------|---------|---------|
| $M = 1$ | 0.945 | 0.945 | 0.946 | 0.946 | 0.946 |
| $M = 2$ | 0.674 | 0.882 | 0.946 | 0.946 | 0.947 |
| $M = 4$ | 0.672 | — | 0.947 | — | 0.947 |

Table 14: mAP comparisons with existing state-of-the-art methods using AlexNet base model on the NUS-WIDE dataset. The mAP is based on top 5000 nearest neighbors.

| Method | 12 bits | 24 bits | 36 bits | 48 bits |
|----------------|--------------|--------------|--------------|--------------|
| SH + CNN [32] | 0.621 | 0.616 | 0.615 | 0.612 |
| ITQ + CNN [32] | 0.719 | 0.739 | 0.747 | 0.756 |
| LFH + CNN [32] | 0.695 | 0.734 | 0.739 | 0.759 |
| KSH + CNN [32] | 0.768 | 0.786 | 0.790 | 0.799 |
| SDH+ CNN [32] | 0.780 | 0.804 | 0.815 | 0.824 |
| FASTH+CNN [32] | 0.779 | 0.807 | 0.816 | 0.825 |
| CNNH [57] | 0.611 | 0.618 | 0.625 | 0.608 |
| NINH [31] | 0.674 | 0.697 | 0.713 | 0.715 |
| DHN [69] | 0.708 | 0.735 | 0.748 | 0.758 |
| DQN [8] | 0.768 | 0.776 | 0.783 | 0.792 |
| DPSH [33] | 0.752 | 0.790 | 0.794 | 0.812 |
| DTSH [52] | 0.773 | 0.808 | 0.812 | 0.824 |
| DSDH [32] | 0.776 | 0.808 | 0.820 | 0.829 |
| PQN | 0.795 | 0.819 | 0.823 | 0.830 |
| RPQN | 0.797 | 0.822 | 0.829 | 0.831 |

6.3 NUS-WIDE

Following the experiment setup in [52, 32], we randomly sample 100 images per class (2100 images in total) as the test query set and the remaining images are used as database images. 500 database images per label are randomly sampled as training images. The mAP is calculated based on the top 5000 returned neighbors. Due to multi-label settings, the cross-entropy loss used in SUBIC [24] and the softmax loss in DPQ [28] are no longer feasible, which explains neither SUBIC [24] nor DPQ [28] conducts the experiments on the NUS-WIDE dataset. Inspired by the success of label embedding proposed in [32], we also adopt a combined loss, which is a weighed sum of our asymmetric triplet loss and a mean square loss defined as

$$\mathcal{L} = \frac{1}{1 + e^{\langle \mathbf{v}, \mathbf{q}_+ \rangle - \langle \mathbf{v}, \mathbf{q}_- \rangle}} + \beta \|\mathbf{W}\mathbf{v} - \mathbf{y}\|_2^2, \quad (21)$$

where \mathbf{W} represents the weights of an additional fully-connected layer and \mathbf{y} is the label of the sample I . We set $\beta = 10$ by default.

We compare our method with two types of baselines. The first type extracts the features from CNN and then convert the extracted features into binary codes. We directly copy the reported results in [32] which conducts experiments on several traditional hashing meth-

Table 15: Comparisons with state-of-the-art methods on ImageNet100 dataset.

| | 16bits | 32bits | 48bits |
|--------------|--------------|--------------|--------------|
| HashNet [9] | 0.506 | 0.631 | 0.663 |
| MIHash [6] | 0.569 | 0.661 | 0.685 |
| TALR-AP [17] | 0.589 | 0.669 | 0.699 |
| PQN | 0.613 | 0.682 | 0.707 |
| RPQN | 0.624 | 0.691 | 0.712 |

ods such as SH [45], ITQ [15], KSH [36], SDH [46], *etc.* The baselines of the second type are deep hashing/quantization methods, where the binary codes are learned in an end-to-end manner. We compare several methods of the second type such as DQN [8], DPSH [33], DTSH [52], DSDH [32], *etc.* As shown in Table 14, the proposed PQN consistently outperforms these two types of baselines when code length L varies among $\{12, 24, 36, 48\}$. The advantage of our PQN over other methods is more obvious when the code length L is short. For instance, when $L = 24$, our PQN/ RPQN achieves a 0.819/0.822 mAP whereas DSDH [32] only achieves a 0.808 mAP.

6.4 ImageNet100

ImageNet100 [9] randomly select 130K images from 100 categories of ImageNet. All images in the validation set as the queries and 100 images per category are selected from the database as the training points. We conduct experiments based on AlexNet backbone and the evaluation metric is mAP@100. We compare with three recent state-of-the-art methods including HashNet [9], MIHash [6] and TALR-AP [17]. As shown in Table 15, our PQN and RPQN consistently outperform the compared methods.

7 Experiments on Video Retrieval

We evaluate our method on two public benchmark datasets: UCF101 [48] and HMDB51 [30]. **UCF101** dataset consists of 101 categories containing 13320 total realistic videos. **HMDB51** dataset contains 6766 clips divided into 51 categories, each containing a minimum of 101 clips. Both UCF101 and HMDB51 provide 3 training/testing splits. Following [37], we use the third split of UCF101 and first split of the HMDB51. The training data not only is used for training the network but also serves as the reference videos in the database for retrieval. The testing data are the query videos. A reference video is related to the query video if they share the same semantic label.

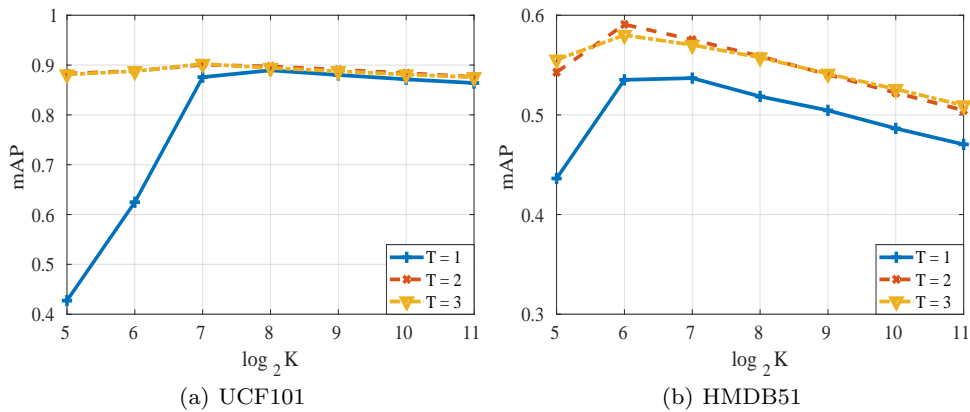
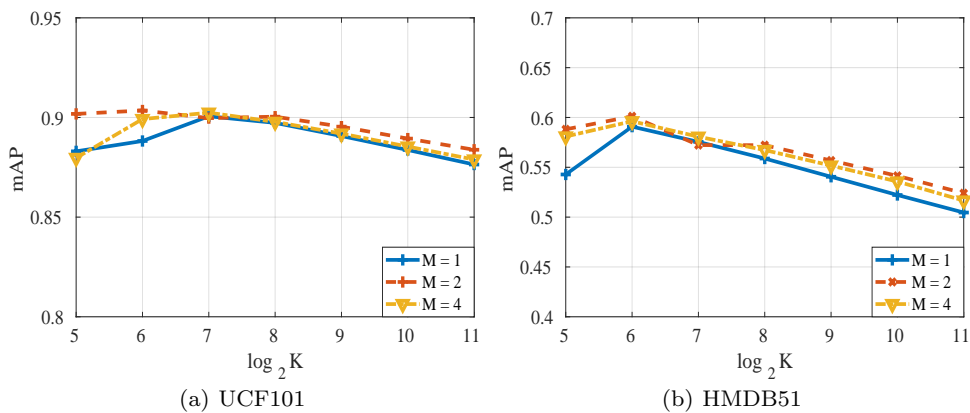
Fig. 5: The influence of T .Fig. 6: The influence of M .

Table 16: Regularization function of the product quantization structure.

| Quantization? | UCF101 | | | HMDB51 | | |
|---------------|--------|--------|--------|--------|--------|--------|
| | $T=1$ | $T=2$ | $T=3$ | $T=1$ | $T=2$ | $T=3$ |
| NO | 0.8547 | 0.8642 | 0.8622 | 0.4628 | 0.4927 | 0.4945 |
| YES | 0.8639 | 0.9035 | 0.9023 | 0.5352 | 0.6012 | 0.5801 |

Backbone network we adopt is an old-fashioned network, BN-Inception [23], but we remove its last fully-connected layer and softmax layer. To suppress overfitting, we add a dropout layer with ratio $r = 0.8$ after the last layer of the backbone network. Even though there are many more advanced deep learning architectures available, we select BN-Inception due to the limitation of our computing resources. Laterly, we will show that the performance achieved by ours using BN-Inception is considerably better than another work [37] using a deeper ResNet50 as the backbone. Despite that a temporal stream network taking optical flow as input can achieve higher performance in action retrieval, we do not do that since the baselines we compare only take the RGB frames as input. On both datasets, we set N , the number of sampled frames per video, as 9.

The batch size is set to be 128. The initial learning rate is 0.001 and the learning rate will be divided by 10 after every 30 epochs and the training process finishes in 120 epochs. We use SGD as the optimizer and set the momentum as 0.9. The loss function used in training the model is standard cross-entropy loss.

7.1 Ablation Study

In this section, we conduct ablation study and evaluate the influence of T , M and α on the proposed TPQN, respectively.

Table 17: Comparison with existing hashing methods.

| | UCF101 | | | | HMDB | | | |
|-----------------|--------|---------|---------|---------|--------|---------|---------|---------|
| | 6 bits | 12 bits | 18 bits | 36 bits | 6 bits | 12 bits | 18 bits | 36 bits |
| LSH | 0.090 | 0.220 | 0.326 | 0.521 | 0.068 | 0.116 | 0.149 | 0.231 |
| SH | 0.234 | 0.448 | 0.613 | 0.777 | 0.155 | 0.255 | 0.344 | 0.440 |
| ITQ | 0.282 | 0.588 | 0.728 | 0.828 | 0.206 | 0.360 | 0.449 | 0.521 |
| SPBE | 0.266 | 0.532 | 0.717 | 0.805 | 0.209 | 0.327 | 0.408 | 0.512 |
| KSH | 0.450 | 0.752 | 0.842 | 0.878 | 0.325 | 0.436 | 0.532 | 0.572 |
| DSH | 0.513 | 0.807 | 0.856 | 0.882 | 0.331 | 0.457 | 0.546 | 0.586 |
| Unsupervised PQ | 0.562 | 0.831 | 0.862 | 0.891 | 0.485 | 0.533 | 0.568 | 0.582 |
| Ours | 0.626 | 0.864 | 0.888 | 0.904 | 0.535 | 0.563 | 0.591 | 0.601 |

Table 18: Comparison with deep video hashing methods.

| | SUBIC [24] | | | DHCM [37] | | | TPQN (Ours) | | |
|--------|------------|-------|-------|-----------|-------|-------|-------------|-------|-------|
| | 64 | 128 | 256 | 64 | 128 | 256 | 12 | 18 | 36 |
| UCF101 | 0.324 | 0.432 | 0.449 | 0.759 | 0.817 | 0.843 | 0.864 | 0.888 | 0.904 |
| HMDB51 | 0.192 | 0.247 | 0.298 | 0.356 | 0.367 | 0.368 | 0.563 | 0.591 | 0.601 |

7.1.1 Influence of T

We vary T among $\{1, 2, 3\}$. Note that when $T = 1$, it will be equivalent to a one-level global max-pooling. In this case, the temporal product quantization network degenerates to product quantization network. On both datasets, we fix $M = 1$ and increase the number of code-words K from 32 to 2048. As shown in Figure 5, when $T = 2, 3$, it consistently outperform that when $T = 1$, which validates the advantage of the proposed temporal product quantization network over product quantization network. Nevertheless, a higher T will take more bits, leading to a higher memory and computation cost. To balance the precision and efficiency, we set $T = 2$ by default on both datasets. Meanwhile, from Figure 5 we can also observe that as K increases, the mAP increases in the early stage and then drops. The increase of mAP is due to that a higher K is capable of representing richer information. In contrast, the mAP decreases when $K > 128$ is caused by over-fitting. We will show in section 7.2 that the quantization can serve as a regularization mechanism which suppresses the over-fitting. How to select K is dependent on the scale of the dataset. As a rule of thumb, we pick a larger K for a larger dataset. As shown in Figure 5, when T varies among $\{1, 2, 3\}$, it consistently achieves the best performance when $K = 128$ on UCF101 dataset and when $K = 64$ on HMDB51 dataset.

7.1.2 Influence of M

A larger M brings a richer codebook as well as a greater complexity. We vary M among $\{1, 2, 4\}$. On both datasets, we set $T = 2$ and increase K from 32 to 2048. In this scenario, the number of bits required for representing

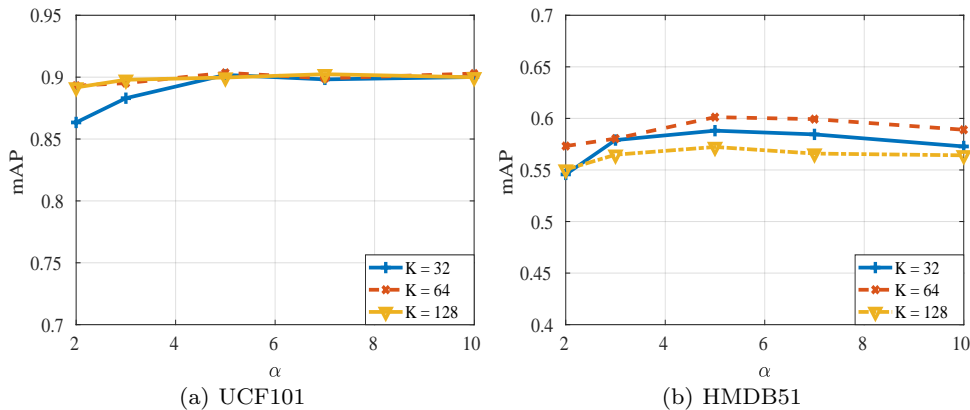
a video is $3M \log_2 K$. As shown in Figure 6, when K is small, $M = 2, 4$ achieves much better performance than $M = 1$, this is expected since a small K will have a limited capability of representing and therefore need a larger M to enrich the codewords. On the contrary, when K is large, the advantage of $M = 2, 4$ over $M = 1$ is not so obvious. Meanwhile, in consistency with the previous experimental results shown in Figure 6, when M varies among $\{1, 2, 4\}$, it continuously achieves the highest precision on UCF101 dataset when $K = 128$ and on HMDB51 when $K = 64$.

7.1.3 Influence of α

α controls the consistency between the approximation function $f_{tm}(\cdot, \alpha)$ and original quantization function $q_{tm}(\cdot)$. We fix $T = 2$ and $M = 2$ and test our TPQN by increasing α from 2 to 10 and vary K among $\{32, 64, 128\}$. As shown in Figure 7, the performance, when α is within the range $[5, 10]$, the performance of TPQN is considerably stable. Nevertheless, we can observe a significant performance drop when α decreases from 5 to 2. This drop is due to the fact that a small α brings a large inconsistency between the approximation function $f_{tm}(\cdot, \alpha)$ in the training phase and the hard quantization in the indexing phase. Meanwhile, on the HMDB51 dataset, we also observe a slight performance drop when α increases from 7 to 10. This performance drop is caused by the fact that a large α will tend to make the training unstable. By default, we set $\alpha = 5$ on both datasets.

7.1.4 Action recognition.

We also evaluate the performance of the proposed TPQN for action recognition. To be specific, we also the near-

Fig. 7: The influence of α .

est neighbor classifier for recognition. We compare with TSN [51], using the same backbone network, BN-Inception [23]. To make a fair comparison, both ours and TSN only take RGB frames as input and do not use optical flows. As shown in Table 19, benefited from temporal pyramid pooling, our TPQN considerably outperforms TSN in the action recognition.

Table 19: The performance in action recognition.

| | UCF101 | HMDB51 |
|----------|--------|--------|
| TSN [51] | 85.3 | 51.0 |
| Ours | 90.9 | 62.3 |

7.2 Complexity Analysis and Regularization

In this section, we analyze the complexity reduction brought by quantization and also show the regularization functionality of the quantization. We will show that our TPQN not only significantly boosts the efficiency but also serves as a regularization mechanism, improving the precision. To make a fair comparison, we compare with a baseline by directly removing the quantization operation, *i.e.*, the third step in temporal product quantization layer and keep other parts of TPQN fixed. As for our TPQN, we set $M = 2$ and $K = 64$ on both datasets.

Without quantization, the video features are high-dimension real-value vectors. For instance, when $T = 2$, the feature dimension is $D(T+1) = 1024 \times 3 = 3072$ and it takes $3072 \times 4 = 12288$ bytes to store the feature in a float-type array. In contrast, using the quantization, it only takes $(T+1)M \lceil \log_2 K/8 \rceil = 6$ bytes, achieving an approximate $2000 \times$ memory reduction.

Meanwhile, we find the quantization also significantly improves the retrieval precision as shown in Table 16. This is due to the regularization mechanism brought by the quantization, which suppresses overfitting and improves the generalization capability of the trained model. For instance, when $T = 2$, our TPQN achieves 0.9035 mAP on UCF101 dataset and 0.6012 mAP on HMDB51 dataset. In contrast, after removing the quantization, it only achieves 0.8642 mAP on UCF101 dataset and 0.4927 mAP on HMDB51 dataset.

7.3 Comparison with state-of-the-art methods

To further demonstrate the effectiveness of the proposed TPQN, we compare it with the state-of-the-art methods. The compared methods can be categorized into two types. The first type of methods are based on a two-step process: video feature extraction followed by hashing. To make a fair comparison, we directly use the features without quantization used in Table 16 for hashing. We implement multiple hashing methods including LSH [13], SH [54], ITQ [15], SPBE [58], KSH [36], SDH [47]. Among them, LSH [13], SH [54], ITQ [15] and SPBE [58] are unsupervised hashing methods, whereas KSH [36] and SDH [47] are supervised hashing methods. Meanwhile, we also compare with unsupervised PQ. As show in Table 17, our method consistently outperforms other methods on both UCF101 and HMDB51 datasets, especially when bit length is small. For instance, on UCF101 dataset, our method achieves a 0.864 mAP when bit length is only 12 whereas the second best DSH only achieves a 0.807 mAP using the same bit length. On HMDB51 dataset, our method achieves a 0.535 mAP when bit length is only 12 whereas the second best DSH only achieves a 0.331 mAP.

The second type of methods is deep video hashing. We mainly compare with two most recent methods,

SUBIC [24] and Deep Hashing with Category Mask (DHCM) [37]. Note that even if SUBIC and DHCM are based on a deeper ResNet50 backbone, our result consistently outperforms both of them using a shallower backbone BN-Inception as show in Table 18. To be specific, on the UCF101 dataset, we achieve a 0.904 mAP using only 36 bits, whereas DHCM only achieves a 0.843 using 256 bits. Meanwhile, on the HMDB51 dataset, we achieve a 0.563 mAP using only 12 bits, whereas DHCM only achieves a 0.368 using 256 bits.

8 Conclusion

In this paper, by constructing an approximate function, we make the product quantization differentiable and feasible to be incorporated in a neural network. Product quantization network (PQN) is introduced, which learns a discriminative and compact image representation in an end-to-end manner. Asymmetric triplet loss extended from triplet loss is introduced, which directly optimizes the representation’s adaptability to retrieval based on asymmetric distance. By revisiting residual quantization, we extend PQN to residual product quantization (RPQN) which triggers the residual learning and further improves the discriminativeness of the representation. Moreover, by exploiting the temporal consistency inherited in videos, we extend PQN to temporal product quantization network (TPQN) for fast video retrieval. Interestingly, our experiments show that the product quantization not only improves the retrieval efficiency but also improves the model’s generalizability and retrieval accuracy. Systematic experiments conducted on benchmark datasets demonstrate state-of-the-art performance of the proposed PQN, RPQN and TPQN in fast image and video retrieval.

Acknowledgements This work is supported in part by a gift grant from Adobe and startup funds from University at Buffalo.

References

1. Babenko, A., Lempitsky, V.: Additive quantization for extreme vector compression. In: CVPR, pp. 931–938 (2014)
2. Babenko, A., Lempitsky, V.: Aggregating local deep features for image retrieval. In: ICCV, pp. 1269–1277 (2015)
3. Babenko, A., Slesarev, A., Chigorin, A., Lempitsky, V.: Neural codes for image retrieval. In: ECCV, pp. 584–599. Springer (2014)
4. Bai, S., Bai, X., Tian, Q., Latecki, L.J.: Regularized diffusion process on bidirectional context for object retrieval. TPAMI (2018)
5. Bai, S., Zhou, Z., Wang, J., Bai, X., Latecki, L.J., Tian, Q.: Ensemble diffusion for retrieval
6. Cakir, F., He, K., Bargal, S.A., Sclaroff, S.: Mhash: On-line hashing with mutual information. In: ICCV (2017)
7. Cao, L., Li, Z., Mu, Y., Chang, S.F.: Submodular video hashing: a unified framework towards video pooling and indexing. In: Proceedings of the 20th ACM international conference on Multimedia, pp. 299–308. ACM (2012)
8. Cao, Y., Long, M., Wang, J., Zhu, H., Wen, Q.: Deep quantization network for efficient image retrieval. In: AAAI (2016)
9. Cao, Z., Long, M., Wang, J., Yu, P.S.: Hashnet: Deep learning to hash by continuation. In: ICCV (2017)
10. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing, pp. 380–388 (2002)
11. Chen, Y., Guan, T., Wang, C.: Approximate nearest neighbor search by residual vector quantization. Sensors **10**(12), 11259–11273 (2010)
12. Chua, T.S., Tang, J., Hong, R., Li, H., Luo, Z., Zheng, Y.: Nus-wide: a real-world web image database from national university of singapore. In: Proceedings of the ACM international conference on image and video retrieval, p. 48 (2009)
13. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the twentieth annual symposium on Computational geometry, pp. 253–262 (2004)
14. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization for approximate nearest neighbor search. In: CVPR, pp. 2946–2953. IEEE (2013)
15. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE T-PAMI **35**(12), 2916–2929 (2013)
16. Gordo, A., Almazán, J., Revaud, J., Larlus, D.: Deep image retrieval: Learning global representations for image search. In: ECCV, pp. 241–257. Springer (2016)
17. He, K., Cakir, F., Bargal, S.A., Sclaroff, S.: Hashing as tie-aware learning to rank. In: CVPR (2018)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778 (2016)
19. Hong, W., Meng, J., Yuan, J.: Distributed composite quantization. In: AAAI (2018)
20. Hong, W., Meng, J., Yuan, J.: Tensorized projection for high-dimensional binary embedding. In: AAAI (2018)
21. Hong, W., Yuan, J.: Fried binary embedding: From high-dimensional visual features to high-dimensional binary codes. IEEE Transactions on Image Processing **27**(10) (2018)
22. Hong, W., Yuan, J., Bhattacharjee, S.D.: Fried binary embedding for high-dimensional visual features. CVPR **11**, 18 (2017)
23. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015)
24. Jain, H., Zepeda, J., Perez, P., Gribonval, R.: Subic: A supervised, structured binary code for image search. In: ICCV, pp. 833–842 (2017)
25. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE T-PAMI **33**(1), 117–128 (2011)
26. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: CVPR, pp. 3304–3311 (2010)

27. Jiang, Q.Y., Li, W.J.: Asymmetric deep supervised hashing. AAAI (2018)
28. Klein, B., Wolf, L.: In defense of product quantization. arXiv preprint arXiv:1711.08589 (2017)
29. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
30. Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., Serre, T.: Hmdb: a large video database for human motion recognition. In: Computer Vision (ICCV), 2011 IEEE International Conference on, pp. 2556–2563. IEEE (2011)
31. Lai, H., Pan, Y., Liu, Y., Yan, S.: Simultaneous feature learning and hash coding with deep neural networks. arXiv preprint arXiv:1504.03410 (2015)
32. Li, Q., Sun, Z., He, R., Tan, T.: Deep supervised discrete hashing. In: NIPS, pp. 2479–2488 (2017)
33. Li, W.J., Wang, S., Kang, W.C.: Feature learning based deep supervised hashing with pairwise labels. arXiv preprint arXiv:1511.03855 (2015)
34. Liong, V.E., Lu, J., Tan, Y.P., Zhou, J.: Deep video hashing. IEEE Transactions on Multimedia **19**(6), 1209–1219 (2017)
35. Liu, H., Wang, R., Shan, S., Chen, X.: Deep supervised hashing for fast image retrieval. In: CVPR, pp. 2064–2072 (2016)
36. Liu, W., Wang, J., Ji, R., Jiang, Y.G., Chang, S.F.: Supervised hashing with kernels. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2074–2081. IEEE (2012)
37. Liu, X., Zhao, L., Ding, D., Dong, Y.: Deep hashing with category mask for fast video retrieval. CoRR **abs/1712.08315** (2017). URL <http://arxiv.org/abs/1712.08315>
38. Martinez, J., Clement, J., Hoos, H.H., Little, J.J.: Revisiting additive quantization. In: European Conference on Computer Vision, pp. 137–153. Springer (2016)
39. Ng, J.Y.H., Yang, F., Davis, L.S.: Exploiting local features from deep networks for image retrieval. arXiv preprint arXiv:1504.05133 (2015)
40. Norouzi, M., Fleet, D.J.: Cartesian k-means. In: CVPR, pp. 3017–3024 (2013)
41. Norouzi, M., Fleet, D.J., Salakhutdinov, R.R.: Hamming distance metric learning. In: Advances in neural information processing systems, pp. 1061–1069 (2012)
42. Perronnin, F., Liu, Y., Sánchez, J., Poirier, H.: Large-scale image retrieval with compressed fisher vectors. In: CVPR, pp. 3384–3391 (2010)
43. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: CVPR, pp. 1–8 (2007)
44. Sablayrolles, A., Douze, M., Jégou, H., Usunier, N.: How should we evaluate supervised hashing? In: ICASSP (2017)
45. Salakhutdinov, R., Hinton, G.: Semantic hashing. RBM **500**(3), 500 (2007)
46. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. IEEE T-PAMI **35**(12), 2916–2929 (2013)
47. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. In: CVPR, vol. 2, p. 5 (2015)
48. Soomro, K., Zamir, A.R., Shah, M.: Ucf101: A dataset of 101 human actions classes from videos in the wild. arXiv preprint arXiv:1212.0402 (2012)
49. Tu, Z., Li, H., Zhang, D., Dauwels, J., Li, B., Yuan, J.: Action-stage emphasized spatio-temporal vlad for video action recognition. IEEE Transactions on Image Processing (2019)
50. Tu, Z., Xie, W., Qin, Q., Veltkamp, R.C., Li, B., Yuan, J.: Multi-stream cnn: Learning representations based on human-related regions for action recognition. Pattern Recognition (2019)
51. Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., Van Gool, L.: Temporal segment networks: Towards good practices for deep action recognition. In: European Conference on Computer Vision, pp. 20–36. Springer (2016)
52. Wang, X., Shi, Y., Kitani, K.M.: Deep supervised hashing with triplet labels. In: ACCV, pp. 70–84. Springer (2016)
53. Wang, X., Zhang, T., Qi, G.J., Tang, J., Wang, J.: Supervised quantization for similarity search. In: CVPR, pp. 2018–2026 (2016)
54. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS, pp. 1753–1760 (2009)
55. Wu, C.Y., Manmatha, R., Smola, A.J., Krähenbühl, P.: Sampling matters in deep embedding learning. In: ICCV (2017)
56. Wu, G., Liu, L., Guo, Y., Ding, G., Han, J., Shen, J., Shao, L.: Unsupervised deep video hashing with balanced rotation. IJCAI (2017)
57. Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S.: Supervised hashing for image retrieval via image representation learning. In: AAAI, pp. 2156–2162. AAAI Press (2014)
58. Xia, Y., He, K., Kohli, P., Sun, J.: Sparse projections for high-dimensional binary codes. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3332–3339 (2015)
59. Ye, G., Liu, D., Wang, J., Chang, S.F.: Large-scale video hashing via structure learning. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2272–2279 (2013)
60. Yu, T., Meng, J., Yuan, J.: Is my object in this video? reconstruction-based object search in videos. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 4551–4557. AAAI Press (2017)
61. Yu, T., Wang, Z., Yuan, J.: Compressive quantization for fast object instance search in videos. In: ICCV, pp. 833–842 (2017)
62. Yu, T., Wu, Y., Bhattacharjee, S.D., Yuan, J.: Efficient object instance search using fuzzy objects matching. AAAI (2017)
63. Yu, T., Wu, Y., Yuan, J.: Hope: Hierarchical object prototype encoding for efficient object instance search in videos. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2424–2433 (2017)
64. Yu, T., Yuan, J., Fang, C., Jin, H.: Product quantization network for fast image retrieval. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 186–201 (2018)
65. Zhang, R., Lin, L., Zhang, R., Zuo, W., Zhang, L.: Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. IEEE TIP **24**(12), 4766–4779 (2015)
66. Zhang, T., Du, C., Wang, J.: Composite quantization for approximate nearest neighbor search. In: ICML, 2, pp. 838–846 (2014)
67. Zhang, Z., Chen, Y., Saligrama, V.: Efficient training of very deep neural networks for supervised hashing. In: CVPR, pp. 1487–1495 (2016)
68. Zhao, F., Huang, Y., Wang, L., Tan, T.: Deep semantic ranking based hashing for multi-label image retrieval. In: CVPR, pp. 1556–1564 (2015)
69. Zhu, H., Long, M., Wang, J., Cao, Y.: Deep hashing network for efficient similarity retrieval. In: AAAI (2016)