Deep Recurrent Quantization for Generating Sequential Binary Codes

Jingkuan Song[1] , **Xiaosu Zhu**[1] , **Lianli Gao**[1] , **Xin-Shun Xu**[2] , **Wu Liu**[3] and **Heng Tao Shen**[1*]
[1]Center for Future Media, University of Electronic Science and Technology of China
[2]Shandong University
[3]JD AI Research
jingkuan.song@gmail.com, xiaosu.zhu@outlook.com, lianli.gao@uestc.edu.cn, xuxinshun@sdu.edu.cn, liuwu1@jd.com, shenhengtao@hotmail.com

Abstract

Quantization has been an effective technology in ANN (approximate nearest neighbour) search due to its high accuracy and fast search speed. To meet the requirement of different applications, there is always a trade-off between retrieval accuracy and speed, reflected by variable code lengths. However, to encode the dataset into different code lengths, existing methods need to train several models, where each model can only produce a specific code length. This incurs a considerable training time cost, and largely reduces the flexibility of quantization methods to be deployed in real applications. To address this issue, we propose a Deep Recurrent Quantization (*DRQ*) architecture which can generate sequential binary codes. To the end, when the model is trained, a sequence of binary codes can be generated and the code length can be easily controlled by adjusting the number of recurrent iterations. A shared codebook and a scalar factor is designed to be the learnable weights in the deep recurrent quantization block, and the whole framework can be trained in an end-to-end manner. As far as we know, this is the first quantization method that can be trained once and generate sequential binary codes. Experimental results on the benchmark datasets show that our model achieves comparable or even better performance compared with the state-of-the-art for image retrieval. But it requires significantly less number of parameters and training times. Our code is published online: https://github.com/cfm-uestc/DRQ.

1 Introduction

With the significant increase of the mass media contents, image retrieval has become the highly-concerned spot. Image retrieval concentrates on searching similar images from large-scale database. The direct way is to use reliable kNN (k-nearest neighbor) techniques, which usually perform brute-force searching on database. ANN (approximate nearest neighbor) search is an optimized algorithm which is actually practicable against kNN search. The main idea of ANN search is to find a compact representation of raw features *i.e*. a binary code with fixed length, which can retain structure of raw feature space and dramatically improve the computation speed.

Recently, hashing methods have been widely used in ANN search. They usually learn a hamming space which is refined to maintain similarity between features [Liu *et al.*, 2016; Zhu *et al.*, 2016; Song *et al.*, 2018c; Song *et al.*, 2018b; Song *et al.*, 2018a]. Since the computation of hamming distance is super fast, hashing methods have huge advantages on ANN search. However, hashing methods lack accuracy on feature restoration. Methods based on quantization require a codebook to store some representative features. Therefore, the main goal of quantization is to reserve more information of feature space in codebook. Then, they try to find a combination of codewords to approximate raw features and only to store indexes of these codewords.

Quantization is originated from k-means algorithm, which first clusters data points and uses the clustering centers as codebook. Each data point is represented by index of its corresponding center. In order to decrease computation cost of k-means, product quantization [Jegou *et al.*, 2011] and optimized product quantization [Ge *et al.*, 2013] split whole feature space into a set of sub-regions and perform similar algorithm on each subspace respectively. Such initial quantization methods construct restrictions and well-designed codebooks to accelerate calculation. In the deep learning era, people proposed some end-to-end deep neural networks to perform image feature learning and quantization together. Deep quantization network [Cao *et al.*, 2016] use AlexNet to learn well-separated image features and use OPQ to quantize features. Deep visual-semantic quantization [Cao *et al.*, 2017] and deep triplet quantization [Liu *et al.*, 2018] quantize features by CQ. Different from these works, product quantization network [Yu *et al.*, 2018] proposed a differentiable method to represent quantization as operations of neural network, so that gradient descent can be applied to quantization. Despite their successes, PQ and its variants have several issues. First, to generate binary codes with different code lengths, a retraining is usually unavoidable. Second, it is tricky for the decomposition of high-dimensional vector space. Different decomposition strategies may result in huge performance differ-

*Contact Author

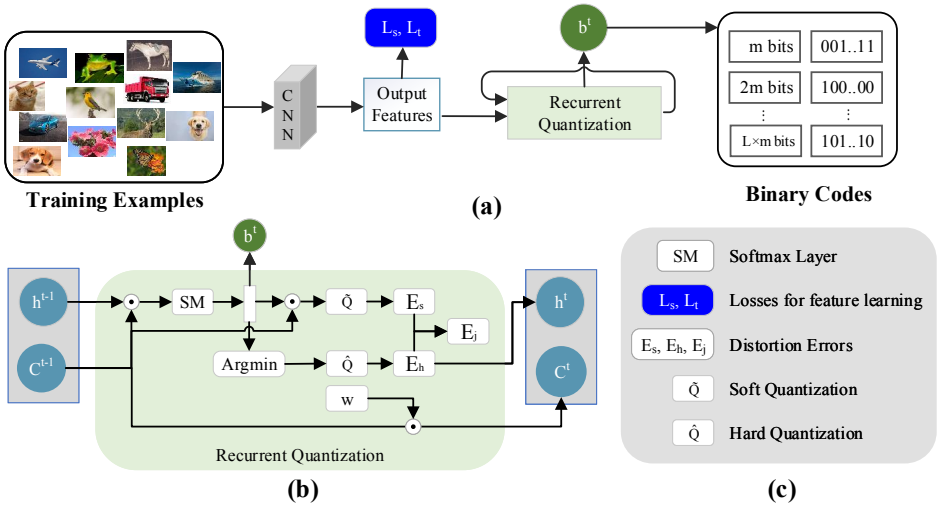


Figure 1: Illustration of our Deep Recurrent Quantization network architecture. The whole framework is depicted in (a). It contains two main models, feature refinement and recurrent quantization block (b). DRQ can generate a sequence of binary codes.

ences. To tackle these issues, we propose a deep quantization method called deep recurrent quantization, which constructs codebook that can be used recurrently to generate sequential binary codes. Extensive experiments show our method outperforms state-of-the-art methods even though they use larger codebooks.

2 Preliminaries

Quantization-based image retrieval tasks are defined as follows: Given a set of images $\mathbb{I} \in \{0, 1, \cdots, 255\}^{N \times H \times W \times C}$ which contain N images of height H, width W and channel C. We first use a CNN, *e.g.* AlexNet and VGG to learn a hyper representation $\mathbf{X} \in \mathbb{R}^{N \times D}$ of images, where D is the dimension of feature vectors. Then we apply quantization on these feature vectors, to learn a codebook $\mathbf{C} \in \mathbb{R}^{K \times D}$ which contains K codewords and each of them has D dimensions. Feature vectors are then compressed to compact binary codes $\mathbf{B} \in \{0, 1\}^{N \times L}$ where L indicates the code length.

2.1 Integrate Quantization To Deep Learning Architectures

During the procedure of quantization, to pick a closest codeword from feature representation is to compute the distance between codewords and features and find the minimum one, which can be described as:

$$\hat{\mathbf{q}} = \hat{\mathbf{Q}}(\mathbf{x}) = \arg\min_{\mathbf{C}_k} \|\mathbf{C}_k - \mathbf{x}\|_2, i = 1, 2, \cdots, K \quad (1)$$

where $\mathbf{x}$ are the features of a data point, and $\mathbf{C}_k$ is the k-th codeword, $\hat{\mathbf{Q}}(\mathbf{x})$ is quantization function and $\hat{\mathbf{q}}$ is quantized feature. Therefore, $\hat{\mathbf{q}}$ is the approximation of $\mathbf{x}$. Meanwhile, we collect the index of codeword as the quantized code, which is described as:

$$\mathbf{b} = \arg\min_{k} \|\mathbf{C}_k - \mathbf{x}\|_2, k = 1, 2, \cdots, K \quad (2)$$

Since $\mathbf{b}$ is in the range of $0 \sim K$-1, then all the codes can be binarized to a code length of $\log_2 K$. Then, the original feature $\mathbf{x}$ can be compressed to an extremely short binary code.

However, the formulation of codeword is non-differentiable, *i.e.* $\frac{\partial \hat{\mathbf{q}}}{\partial \mathbf{C}}$ does not exist. It cannot be directed integrated into deep learning architectures. To tackle this issue, we use a convex combination of codewords to approximate features, which is defined as follows:

$$p_k(\mathbf{x}) = \frac{e^{-\gamma \|\mathbf{C}_k - \mathbf{x}\|_2}}{\sum_{j=1}^{K} e^{-\gamma \|\mathbf{C}_j - \mathbf{x}\|_2}} \quad (3)$$

$$\tilde{\mathbf{q}} = \tilde{\mathbf{Q}}(\mathbf{x}) = \sum_{k=1}^{K} p_k(\mathbf{x}) \mathbf{C}_k \quad (4)$$

Here, $p_k(\mathbf{x})$ indicates the confidences of each codewords w.r.t. $\mathbf{x}$, *i.e.* the closer one codeword $\mathbf{C}_k$ is to a feature $\mathbf{x}$, the higher $p_k(\mathbf{x})$ will be. Then, $\hat{\mathbf{q}}$ is approximated by $\tilde{\mathbf{q}}$, which is the weighted sum of all codewords. We define $\hat{\mathbf{Q}}(\mathbf{x})$ as hard quantization and $\tilde{\mathbf{Q}}(\mathbf{x})$ as soft quantization.

3 Proposed Method

The whole network architecture of our deep recurrent quantization (DRQ) is demonstrated in Fig. 1. DRQ contains two main parts: feature extraction module and quantization module. In feature extraction module, we apply intermediate supervision on top of CNN, to guide the learning of semantic-embedded visual features. In quantization module, we design a recurrent quantization block and integrate it into deep learning architecture which can be trained end-to-end.

3.1 Intermediate Supervision for Features

To get the feature representation of images, we use AlexNet to extract features from the last linear layer. To leverage the clustering performance *i.e.*, to let the images with the same label have higher similarity and vice versa, we apply two losses with intermediate supervision. Specifically, we first collect a triplet in dataset which contains an anchor image I^o, a positive sample I^+ and a negative sample I^- w.r.t. anchor (for multi-label images, we define a positive image as one which shares at least one label with anchor, and a negative image as

one which does not share any label with an anchor), and feed them into AlexNet to obtain the 4096-d features from $fc7$ layer. Then we add two linear layers $fc8$ of 1748-d and $fc9$ of 300-d. We concatenate $fc8$ and $fc9$ to get a final feature $\mathbf{x}$ of 2048-d. Since we feed the triplet into the network, the output features are represented as $\mathbf{x}^o, \mathbf{x}^+, \mathbf{x}^-$.

We apply two supervised objective function on these layers: 1) Adaptive margin loss ℓ_s, which is from DVSQ [Cao *et al.*, 2017] and applied to $fc9$ outputs of triplet, and 2) Triplet loss ℓ_t defined to final feature $\mathbf{x}$, which is a concatenated feature of $fc8$ and $fc9$. ℓ_s is defined as:

$$\ell_s(\mathbf{x}) = \sum_{i\in\mathcal{Y}_n}\sum_{j\notin\mathcal{Y}_n}(0, \delta_{ij} - \frac{\boldsymbol{v_i}^\intercal \boldsymbol{z}_n}{\|\boldsymbol{v}_i\|\,\|\boldsymbol{z}_n\|} + \frac{\boldsymbol{v_j}^\intercal \boldsymbol{z}_n}{\|\boldsymbol{v}_j\|\,\|\boldsymbol{z}_n\|})$$
$$\delta_{ij} = 1 - \frac{\boldsymbol{v}_i^\intercal \boldsymbol{v}_j}{\|\boldsymbol{v}_i\|\,\|\boldsymbol{v}_j\|} \tag{5}$$

Triplet loss ℓ_t can adjust features to adapt to clustering, which uses the triplet of x^o, x^+, x^-. It is defined as:

$$\ell_t(\mathbf{x}^o, \mathbf{x}^+, \mathbf{x}^-) = \max\left(\left\|\mathbf{x}^o - \mathbf{x}^+\right\|_2 - \left\|\mathbf{x}^o - \mathbf{x}^-\right\|_2 + \delta, 0\right) \tag{6}$$

3.2 Recurrent Quantization Block

In recurrent quantization model, we adopt a shared codebook that contains K codewords. We denote the level of quantization code as M, which indicates how many iterations the codebook is reused. For each level, we pick a proper codeword as the approximation of feature vectors, and we take the index of picked codeword as the quantization code. For example, if we set $K = 256, M = 4$, the index range of each level quantization code is $0 - 255$, represented as a binary code of $\log_2 K$=8 bits. The total length of quantization code is $M \times \log_2 K = 4 \times 8 = 32$. The position $0 - 7$ is the index of first level codeword, $8 - 15$ is the index of second level, *etc*. Therefore, the feature vector can be approximated by a combination of a few codewords in the codebook.

As we described in Sec. 2.1, to perform a quantization, input $\mathbf{x}$ and codebook $\mathbf{C}$ are necessary. Output is the code $\mathbf{b}$. Inspired by the hierarchical codebooks in stacked quantizer [Martinez *et al.*, 2014], we observe the residual of $\mathbf{x}$ can be used as an input to the next quantizer. Therefore, a basic idea is to perform quantization step-by-step:

$$\begin{aligned}
\hat{\mathbf{q}}^1 &= \hat{\mathbf{Q}}^1(\mathbf{x}),\ \mathbf{r}^1 = \mathbf{x} - \hat{\mathbf{q}}^1 \\
\hat{\mathbf{q}}^2 &= \hat{\mathbf{Q}}^2(\mathbf{r}^1),\ \mathbf{r}^2 = \mathbf{x} - \hat{\mathbf{q}}^1 - \hat{\mathbf{q}}^2 \\
&\vdots \\
\hat{\mathbf{q}}^M &= \hat{\mathbf{Q}}^M(\mathbf{r}^{M-1})
\end{aligned} \tag{7}$$

Specifically, $\hat{\mathbf{q}}^m$ is quantized feature explained above, and $\mathbf{r}^1, \mathbf{r}^2, \cdots, \mathbf{r}^{M-1}$ are residuals of $\mathbf{x}$. We put $\mathbf{r}^m$ to the next quantization to get the $\mathbf{q}^{m+1}$ which approximates $\mathbf{r}^m$. Therefore, $\sum_m \mathbf{q}^m$ can be described as an approximation of $\mathbf{x}$, which is much preciser than $\hat{q}^1$. Notice that processing of $\mathbf{q}^m$ is similar. If we use a shared codebook, the computation

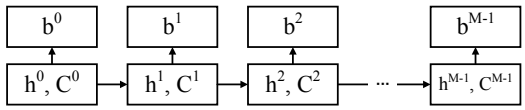


Figure 2: The unfolded version of recurrent quantization.

in Eq. 7 can be rewritten recurrently:

$$\begin{aligned}
&\mathbf{b}^m = \arg\min_k \left\|\mathbf{C}_k - \mathbf{h}^{m-1}\right\|_2, k = 1, 2, \cdots, K, m \geq 0 \\
&\mathbf{h}^m = \mathbf{h}^{m-1} - \mathbf{C}^{m-1}_{\mathbf{b}_{m-1}}, 1 \leq m \leq M \\
&\mathbf{C}^m = w \times \mathbf{C}^{m-1}, 1 \leq m \leq M \\
&\mathbf{h}^0 = \mathbf{x}, \mathbf{C}^0 = \mathbf{C},
\end{aligned} \tag{8}$$

And the soft and hard quantization of $\mathbf{h}^{m-1}$ is defined as:

$$\begin{aligned}
\hat{\mathbf{q}}^m &= \hat{\mathbf{Q}}(\mathbf{h}^{m-1}), 1 \leq m \leq M \\
\tilde{\mathbf{q}}^m &= \tilde{\mathbf{Q}}(\mathbf{h}^{m-1}), 1 \leq m \leq M
\end{aligned} \tag{9}$$

The unfolded structure of recurrent quantization is depicted in Fig. 2. Here, $\mathbf{h}^0, \mathbf{C}^0$ is the raw features and initial codebook. w is a shared learnable parameter with random initialization. In iteration m, we compute $\mathbf{b}^m$ to find the best-fitted codeword, then we use $\mathbf{C}^m, \mathbf{b}^m, \mathbf{h}^m$ to compute residual of $\mathbf{h}^m$ and treat residual as next input $\mathbf{h}^{m+1}$. Since the residual is one or more order of magnitudes lower than $\mathbf{h}^m$, the next input should be much smaller than codewords in codebook, so we use $w \in \mathbb{R}$ as a scale factor to adjust the norm of codebook in order to fit the new input. In next iteration $m+1$, we use the scaled codebook $\mathbf{C}^{m+1}$ to complete another similar computation. Finally, we learn a codebook $\mathbf{C}$, a scale factor w and sequential binary codes $\mathbf{b}^0, \mathbf{b}^1, \cdots, \mathbf{b}^M$. The hard and soft quantization of $\mathbf{x}$ can be computed as:

$$\begin{aligned}
\hat{\mathbf{x}} &= \hat{\mathbf{q}}^0 + \hat{\mathbf{q}}^1 + \cdots + \hat{\mathbf{q}}^M \\
&= \mathbf{C}^0_{\mathbf{b}_0} + w \times \mathbf{C}^1_{\mathbf{b}_1} + \cdots + w^{M-1} \times \mathbf{C}^M_{\mathbf{b}_M}
\end{aligned} \tag{10}$$
$$\tilde{\mathbf{x}} = \tilde{\mathbf{q}}^0 + \tilde{\mathbf{q}}^1 + \cdots + \tilde{\mathbf{q}}^M \tag{11}$$

By reusing codebook $\mathbf{C}$, we can reduce the number of parameters by M times.

Objective Function

Since $\hat{\mathbf{q}}(\mathbf{x})$ and $\tilde{\mathbf{q}}(\mathbf{x})$ are approximation of feature $\mathbf{x}$, we define a distortion error as:

$$E_h^m = ||\sum\nolimits_{i=1}^m \hat{\mathbf{q}}^m - \mathbf{x}||_2,\ E_s^m = ||\sum\nolimits_{i=1}^m \tilde{\mathbf{q}}^m - \mathbf{x}||_2 \tag{12}$$

where E_h^m is the distortion error between $\hat{\mathbf{q}}^m$ and $\mathbf{x}$ at iteration m and E_s^m is the distortion error between $\tilde{\mathbf{q}}^m$ and $\mathbf{x}$. We sum distortions for each level and the total distortion error is:

$$E_h = \sum\nolimits_{m=1}^M E_h^m,\ E_s = \sum\nolimits_{m=1}^M E_s^m \tag{13}$$

We also design a joint central error E_j to align E_h and E_s:

$$E_j = \|E_h - E_s\|_2 \tag{14}$$

3.3 Optimization

In DRQ, there are two main losses: 1) ℓ_t and ℓ_s which refine features, 2) E_h, E_s, E_j, which control the quantization effectiveness. We split the training procedure into three stages. Firstly, we minimize ℓ_t, ℓ_s together to pre-train our preceding neural network. Then, we add recurrent quantization block into network but only perform one recurrent iteration *i.e*. set $M = 1$ and optimize $\ell_t, \ell_s, E_h, E_s, E_j$ together. This is to get an initial codebook which are optimized for short binary codes. Finally, we set M to a specified value and optimize the whole network with all losses, until it converges or we reach the max number of training iterations.

4 Experiments

To validate the effectiveness and efficiency of our adopted deep recurrent quantization, we perform extensive experiments on three public datasets: **CIFAR-10**, **NUS-WIDE** and **ImageNet**. *Since existing methods use different settings, to make a thorough comparison with them, we follow these works and compare with them using separate settings.* We implement our model with Tensorflow, using a pre-trained AlexNet and construct intermediate layers on top of the $fc7$ layer. Meanwhile, we randomly initialize codebook with specified M and K, which will be described below. We use Adam optimizer with $lr = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ for training.

4.1 Comparison Results Using Setting 1

Settings

We first conduct results and make comparisons with state-of-the art methods on two benchmark datasets: **CIFAR-10** and **NUS-WIDE**. **CIFAR-10** is a public dataset labeled in 10 classes. It consists of 50,000 images for training and 10,000 images for validation. We follow [Yu *et al.*, 2018] to combine the training and validation set together, and randomly sample 5,000 images per class as database. The remaining 10,000 images are used as queries. Meanwhile, we use the whole database to train the network. **NUS-WIDE** is a public dataset consisting of 81 concepts, and each image is annotated with one or more concepts. We follow [Yu *et al.*, 2018] to use the subset of 195,834 images from the 21 most frequent concepts. We randomly sample 1,000 images per concept as the query set, and use the remaining images as the database. Furthermore, we randomly sample 5,000 images per concept from the database as the training set. We use mean Average Precision (mAP@5000) as the evaluation metric.

Results

On CIFAR-10, we compare our DRQ with a few state-of-the-art methods, including DRSCH [Zhang *et al.*, 2015], DSCH [Zhang *et al.*, 2015], DSRH [Zhao *et al.*, 2015], VDSH [Zhang *et al.*, 2016], DPSH [Li *et al.*, 2015], DTSH [Li *et al.*, 2015], DTSH [Wang *et al.*, 2016], DSDH [Li *et al.*, 2017] and PQNet [Yu *et al.*, 2018], using 16, 24, 36, 48 bits. We set $M = 4$ and $K = 2^{\frac{L}{M}} = 16, 64, 512, 2048$. The results on CIFAR dataset are shown in Table 1. Results show our network achieves comparable mAP performance against state-of-the-art methods, *i.e*. PQNet. Our mAP is only 0.3%-0.5% lower than PQNet. Results also show our performance is stable with variable bit-lengths. Our method only get 0.1% decrease when bit-length shrinks to 16 bits. Noticed that our recurrent quantization only use a single codebook with K codewords, and therefore our method requires much less parameters compared with other methods, as shown in Tab. 4.

Method	16 bits	24 bits	36 bits	48 bits
DRSCH	0.615	0.622	0.629	0.631
DSCH	0.609	0.613	0.617	0.686
DSRH	0.608	0.611	0.617	0.618
VDSH	0.845	0.848	0.844	0.845
DPSH	0.903	0.885	0.915	0.911
DTSH	0.915	0.923	0.925	0.926
DSDH	0.935	0.940	0.939	0.939
PQNet	**0.947**	**0.947**	**0.946**	**0.947**
DRQ	0.942	0.943	0.943	0.943

Table 1: Retrieval performance on CIFAR-10. The scores reported are mean Average Precision values.

Method	12 bits	24 bits	36 bits	48 bits
SH	0.621	0.616	0.615	0.612
ITQ	0.719	0.739	0.747	0.756
LFH	0.695	0.734	0.739	0.759
KSH	0.768	0.786	0.790	0.799
SDH	0.780	0.804	0.815	0.824
FASTH	0.779	0.807	0.816	0.825
NINH	0.674	0.697	0.713	0.715
DHN	0.708	0.735	0.748	0.758
DQN	0.768	0.776	0.783	0.792
DPSH	0.752	0.790	0.794	0.812
DTSH	0.773	0.808	0.812	0.824
DSDH	0.776	0.808	0.820	0.829
PQNet	**0.795**	0.819	0.823	0.830
DRQ	0.772	**0.838**	**0.840**	**0.843**

Table 2: Retrieval performance on NUS-WIDE. The scores reported are mean Average Precision values.

On NUS-WIDE dataset, we compare our method with a few shallow and deep methods. Shallow methods include SH [Salakhutdinov and Hinton, 2009], ITQ [Gong *et al.*, 2013], LFH [Zhang *et al.*, 2014], KSH [Liu *et al.*, 2012], SDH [Shen *et al.*, 2015], FASTH [Lin *et al.*, 2014]. Deep methods include NINH [Lai *et al.*, 2015a], DHN [Zhu *et al.*, 2016], DQN [Cao *et al.*, 2016], DPSH [Li *et al.*, 2015], DTSH [Wang *et al.*, 2016], DSDH [Li *et al.*, 2017] and PQNet [Yu *et al.*, 2018]. The results are generated in 12, 24, 36, 48 bits. We fix $K = 2048, M = 4$ to generate 48 bits codes, and then slice these codes to get shorter binary codes. The results are shown in Table 2. On NUS-WIDE, our method achieves the highest mAP compared with state-of-the-art methods when the code length is longer than 12 bits. Noticed that our method uses a shared codebook for all code lengths, and it is trained once.

The codebook size w.r.t. code-length comparison between multiple methods is shown in Tab. 4. Our method obtains the smallest codebook size compared with the other methods. Also, to generate binary codes with different lengths, our methods is trained once.

Method	CIFAR-10				NUS-WIDE				ImageNet			
	8 bits	16 bits	24 bits	32 bits	8 bits	16 bits	24 bits	32 bits	8 bits	16 bit	24 bits	32 bits
ITQ-CCA	0.315	0.354	0.371	0.414	0.526	0.575	0.572	0.594	0.189	0.270	0.339	0.436
BRE	0.306	0.370	0.428	0.438	0.550	0.607	0.605	0.608	0.251	0.363	0.404	0.453
KSH	0.489	0.524	0.534	0.558	0.618	0.651	0.672	0.682	0.228	0.398	0.499	0.547
SDH	0.356	0.461	0.496	0.520	0.645	0.688	0.704	0.711	0.385	0.516	0.570	0.605
SQ	0.567	0.583	0.602	0.615	0.653	0.691	0.698	0.716	0.465	0.536	0.592	0.611
CNNH	0.461	0.476	0.465	0.472	0.586	0.609	0.628	0.635	0.317	0.402	0.453	0.476
DNNH	0.525	0.559	0.566	0.558	0.638	0.652	0.667	0.687	0.347	0.416	0.497	0.525
DHN	0.512	0.568	0.594	0.603	0.668	0.702	0.713	0.716	0.358	0.426	0.531	0.556
DSH	0.592	0.625	0.651	0.659	0.653	0.688	0.695	0.699	0.332	0.398	0.487	0.537
DVSQ	0.715	0.727	0.730	0.733	0.780	0.790	0.792	0.797	0.500	0.502	0.505	0.518
DTQ	0.785	0.789	0.790	0.792	**0.795**	0.798	0.800	0.801	**0.641**	**0.644**	**0.647**	**0.651**
DRQ	**0.803**	**0.824**	**0.832**	**0.837**	0.748	**0.810**	**0.817**	**0.821**	0.551	0.583	0.585	0.587

Table 3: Quantitative comparison with state-of-the-art methods on three datasets. The scores reported are mean Average Precision values.

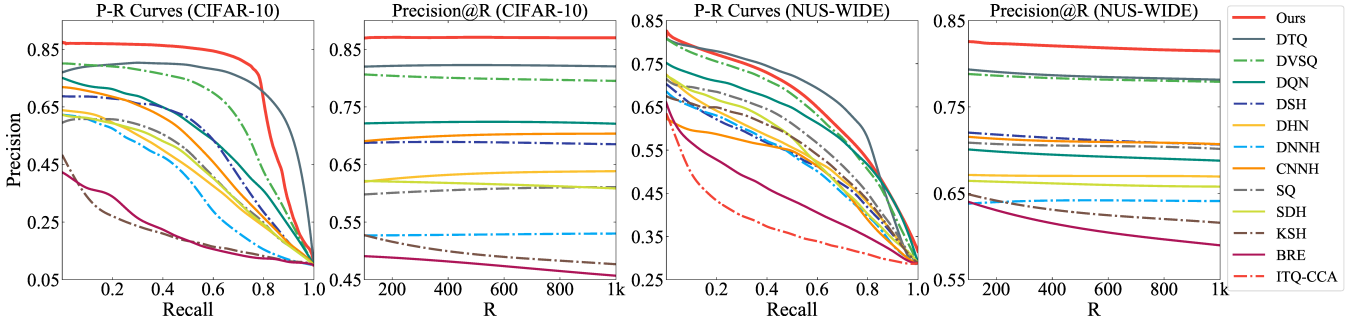


Figure 3: Quantitative comparison with state-of-the-art methods on two datasets: CIFAR-10 and NUS-WIDE. For each data set, we demonstrate Precision-Recall (P-R) curves and Precision@R curves. All results are based on 32-bit.

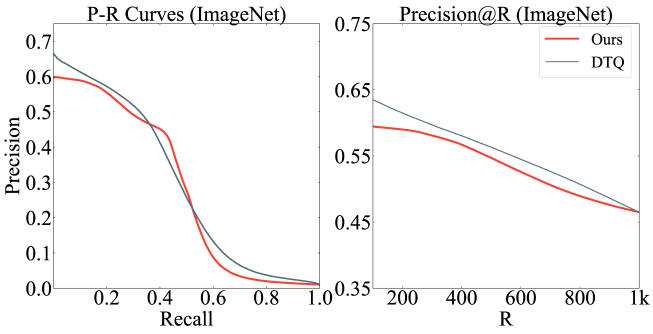


Figure 4: Comparison on ImageNet. All results are based on 32-bit.

Methods	8 bits	16 bits	24 bits	32 bits	40 bits	48 bits
PQ (PQNet)	524k	524k	524k	524k	524k	524k
OPQ	4.72M	4.72M	4.72M	4.72M	4.72M	4.72M
AQ (DTQ)	524k	1.05M	1.57M	2.10M	2.62M	3.15M
DRQ	**524k**					

Table 4: Codebook size w.r.t. code-length comparison among multiple methods (we set $D = 2048$ and $K = 256$). PQNet uses similar codebook structure to PQ, and so does DTQ to AQ.

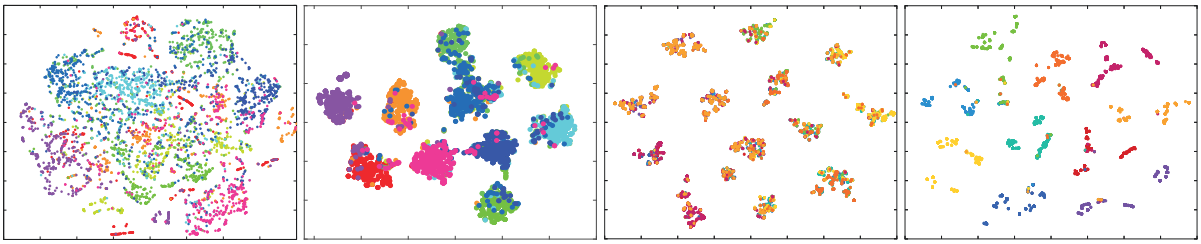
4.2 Comparison Results Using Setting 2

Setting

Following DTQ [Liu *et al.*, 2018], on **CIFAR-10**, we combine the training and validation set together, and randomly select 500 images per class as the training set, 100 images per class as the query set. The remaining images are used as the database. On **NUS-WIDE**, we use the subset of 195,834 images from the 21 most frequent concepts. We randomly sample 5,000 images as the query set, and use the remaining images as the database. Furthermore, we randomly select 10,000 images from the database as the training set. On **ImageNet**, we follow [Cao *et al.*, 2017] to randomly choose 100 classes. We use all the images of these classes in the training set as the database, and use all the images of these classes in the validation set as the queries. Furthermore, we randomly select 100 images for each class in the database for training. We compare our method with 11 classical hash or quantization methods, including 5 shallow methods: ITQ-CCA [Gong *et al.*, 2013], BRE [Kulis and Darrell, 2009], KSH [Liu *et al.*, 2012], SDH [Shen *et al.*, 2015] and SQ [Martinez *et al.*, 2014], and 6 deep architecture: CNNH [Xia *et al.*, 2014], DNNH [Lai *et al.*, 2015b], DHN [Zhu *et al.*, 2016], DSH [Liu *et al.*, 2016], DVSQ [Cao *et al.*, 2017], DTQ [Liu *et al.*, 2018].

Results

We use mAP@54000 on CIFAR-10 and mAP@5000 on NUS-WIDE and ImageNet. We use 8, 16, 24, 32-bits codes by setting $M = 4, K = 256$. We also use precision-recall curve and precision@R (returned results) curve to evaluate the retrieval quality. The results are shown in Table 3, Fig. 3 and Fig. 4.



(a) DVSQ (b) DTQ (c) Our Unsupervised (d) Ours

Figure 5: t-SNE visualization of 32 bits quantized features between DQN, DVSQ and DRQ. Images are randomly sampled on CIFAR-10 dataset, and samples with different labels are marked with different colors.

It can be observed that: 1) Our DRQ significantly outperforms the other methods in CIFAR-10 and NUS-WIDE datasets. Specifically, it outperforms the best counterpart (DTQ) by 1.8%, 3.5%, 4.2% and 4.5% on CIFAR-10, and by 1.2%, 1.7% and 2.0% on NUS-WIDE dataset. DRQ is outperformed by DTQ on NUS-WIDE for 8-bit codes. The possible reason is that in DRQ, codebooks are shared by different code lengths, and may lose some accuracy especially for short binary codes. On ImageNet, our method is outperformed by DTQ, which may be caused by the random selection. Also, our DRQ requires much less parameters than DTQ, and our model is only trained once. 2) With the increase of code length, the performance of most indexing methods is improved accordingly. For our DRQ, the mAP increased by 3.4%, 7.3% and 3.6% for CIFAR-10, NUS-WIDE and ImageNet dataset respectively. This verifies our DRQ can generate sequential binary codes to gradually improve accuracy. 3) The performances of precision-recall curves for different methods are consistent with their performances of mAP. The precision curves represent the retrieval precision with respect to number of return results.

4.3 Ablation Study

In this subsection, we study the effect of each part in our architecture using the following settings. 1) Unsupervised quantization: we use raw $fc7$ output without additional $fc8, fc9$. 2) Remove L_s: we remove $fc9$ and L_s, and change $fc8$ to 2048-d and apply L_t on it directly, to validate the role of L_s. 3) Remove L_t: we remove $fc9$ and L_t and change $fc8$ to 300-d with L_s, to validate the role of L_t. 4) $\tilde{\mathbf{q}}$ only: we remove the construction of $\hat{\mathbf{q}}$ and associated losses, to validate the role of $\hat{\mathbf{q}}$. 5) Remove E_j: we remove E_j to validate the effectiveness of joint central loss. 6) Intermediate supervision: we use only $fc9$, which is 300-d and also apply L_t to $fc9$, to validate effectiveness of the concatenation of $fc8, fc9$. We perform ablation study on NUS-WIDE and show results in Tab. 5. In general, DRQ performs the best, and 'Remove ℓ_s' ranks the second. By removing ℓ_s, the supervision information is still utilized in ℓ_t, so mAP drop is not that significant. The unsupervised architecture also achieves good results, indicating the validity of pretrained AlexNet. Notice that if we remove any of the objective functions in the structure, mAP will have a huge loss. This indicates the effectiveness of each part of our DRQ. We get the worst result when we remove the concat, this may be of the significant information loss in 300-d features.

Structure	8 bits	16 bits	24 bits	32 bits
Unsupervised	0.582	0.630	0.629	0.626
Remove ℓ_s	**0.750**	0.759	0.761	0.762
Remove ℓ_t	0.629	0.672	0.678	0.680
$\tilde{\mathbf{q}}$ only	0.654	0.725	0.731	0.734
Remove E_j	0.604	0.628	0.628	0.628
Remove concat	0.544	0.569	0.571	0.575
DRQ	0.748	**0.810**	**0.817**	**0.821**

Table 5: Ablation study of our method on NUS-WIDE dataset.

4.4 Qualitatively Results

To qualitatively validate the performance of quantization methods, we also perform t-SNE visualization on DTQ, DVSQ and DRQ, and show the results in Fig.5. Visualizations are created on CIFAR-10, we randomly sample 5,000 images from database and adopt 32 bits quantized features. Our DRQ has a similar performance to DTQ, and they both show distinct clusters in their visualization, which is much better than DVSQ. Our unsupervised structure also has a promising performance since data points are concretely clustered. However, some of the data points with different labels are wrongly clustered together. This indicates the importance of supervision information.

5 Conclusion

In this paper, we propose a Deep Recurrent Quantization (DRQ) architecture to generate sequential binary codes. When the model is trained once, a sequence of binary codes can be generated and the code length can be easily controlled by adjusting the number of recurrent iterations. A shared codebook and a scalar factor is designed to be the learnable weights in the deep recurrent quantization block, and the whole framework can be trained in an end-to-end manner. Experimental results on the benchmark datasets show that our model achieves comparable or even better performance compared with the state-of-the-art for image retrieval, but with much less parameters and training time.

Acknowledgements

This work is supported by the Fundamental Research Funds for the Central Universities (Grant No. ZYGX2014J063, No. ZYGX2016J085), the National Natural Science Foundation of China (Grant No. 61772116, No. 61872064, No. 61632007, No. 61602049).

References

[Cao *et al.*, 2016] Yue Cao, Mingsheng Long, Jianmin Wang, Han Zhu, and Qingfu Wen. Deep quantization network for efficient image retrieval. In *AAAI*, pages 3457–3463, 2016.

[Cao *et al.*, 2017] Yue Cao, Mingsheng Long, Jianmin Wang, and Shichen Liu. Deep visual-semantic quantization for efficient image retrieval. In *CVPR*, volume 2, 2017.

[Ge *et al.*, 2013] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, pages 2946–2953, 2013.

[Gong *et al.*, 2013] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[Jegou *et al.*, 2011] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

[Kulis and Darrell, 2009] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009.

[Lai *et al.*, 2015a] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015.

[Lai *et al.*, 2015b] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015.

[Li *et al.*, 2015] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. *arXiv preprint arXiv:1511.03855*, 2015.

[Li *et al.*, 2017] Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. Deep supervised discrete hashing. In *NIPS*, pages 2482–2491, 2017.

[Lin *et al.*, 2014] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, pages 1963–1970, 2014.

[Liu *et al.*, 2012] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012.

[Liu *et al.*, 2016] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Deep supervised hashing for fast image retrieval. In *CVPR*, pages 2064–2072, 2016.

[Liu *et al.*, 2018] Bin Liu, Yue Cao, Mingsheng Long, Jianmin Wang, and Jingdong Wang. Deep triplet quantization. In *ACM MM*, pages 755–763. ACM, 2018.

[Martinez *et al.*, 2014] Julieta Martinez, Holger H Hoos, and James J Little. Stacked quantizers for compositional vector compression. *arXiv preprint arXiv:1411.2173*, 2014.

[Salakhutdinov and Hinton, 2009] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[Shen *et al.*, 2015] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *CVPR*, pages 37–45, 2015.

[Song *et al.*, 2018a] Jingkuan Song, Lianli Gao, Li Liu, Xiaofeng Zhu, and Nicu Sebe. Quantization-based hashing: a general framework for scalable image and video retrieval. *Pattern Recognition*, 75:175 – 187, 2018.

[Song *et al.*, 2018b] Jingkuan Song, Tao He, Lianli Gao, Xing Xu, Alan Hanjalic, and Heng Tao Shen. Binary generative adversarial networks for image retrieval. In *AAAI*, pages 394–401, 2018.

[Song *et al.*, 2018c] Jingkuan Song, Hanwang Zhang, Xiangpeng Li, Lianli Gao, Meng Wang, and Richang Hong. Self-supervised video hashing with hierarchical binary auto-encoder. *IEEE Transactions on Image Processing*, 27(7):3210–3221, 2018.

[Wang *et al.*, 2016] Xiaofang Wang, Yi Shi, and Kris M Kitani. Deep supervised hashing with triplet labels. In *ACCV*, pages 70–84. Springer, 2016.

[Xia *et al.*, 2014] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, page 2, 2014.

[Yu *et al.*, 2018] Tan Yu, Junsong Yuan, Chen Fang, and Hailin Jin. Product quantization network for fast image retrieval. In *ECCV*, pages 191–206. Springer, 2018.

[Zhang *et al.*, 2014] Peichao Zhang, Wei Zhang, Wu-Jun Li, and Minyi Guo. Supervised hashing with latent factor models. In *ACM SIGIR*, pages 173–182. ACM, 2014.

[Zhang *et al.*, 2015] Ruimao Zhang, Liang Lin, Rui Zhang, Wangmeng Zuo, and Lei Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Transactions on Image Processing*, 24(12):4766–4779, 2015.

[Zhang *et al.*, 2016] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient training of very deep neural networks for supervised hashing. In *CVPR*, pages 1487–1495, 2016.

[Zhao *et al.*, 2015] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, pages 1556–1564, 2015.

[Zhu *et al.*, 2016] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *AAAI*, pages 2415–2421, 2016.