Contrastive Quantization with Code Memory for Unsupervised Image Retrieval

Jinpeng Wang[1,2,4], **Ziyun Zeng**[1,4], **Bin Chen**[2*], **Tao Dai**[3], **Shu-Tao Xia**[1,4]

[1]Tsinghua Shenzhen International Graduate School, Tsinghua University
[2]Harbin Institute of Technology, Shenzhen
[3]Shenzhen University
[4]Research Center of Artificial Intelligence, Peng Cheng Laboratory
{wjp20, zengzy21}@mails.tsinghua.edu.cn, chenbin2021@hit.edu.cn, daitao.edu@gmail.com, xiast@sz.tsinghua.edu.cn

Abstract

The high efficiency in computation and storage makes hashing (including binary hashing and quantization) a common strategy in large-scale retrieval systems. To alleviate the reliance on expensive annotations, unsupervised deep hashing becomes an important research problem. This paper provides a novel solution to *unsupervised deep quantization*, namely **C**ontrastive **Q**uantization with Code **Me**mory (**MeCoQ**). Different from existing reconstruction-based strategies, we learn unsupervised binary descriptors by contrastive learning, which can better capture discriminative visual semantics. Besides, we uncover that *codeword diversity regularization* is critical to prevent contrastive learning-based quantization from model degeneration. Moreover, we introduce a novel *quantization code memory module* that boosts contrastive learning with lower feature drift than conventional feature memories. Extensive experiments on benchmark datasets show that MeCoQ outperforms state-of-the-art methods. Code and configurations are publicly released.

Introduction

Hashing (Wang et al. 2017) plays a key role in Approximate Nearest Neighbor (ANN) search and has been widely applied in large-scale systems to improve search efficiency. There are two technical branches in hashing, namely binary hashing and quantization. Binary hashing methods (Charikar 2002; Heo et al. 2012) transform data into the Hamming space such that distances are measured quickly with bitwise operations. Quantization methods (Jegou, Douze, and Schmid 2010) divide real data space into disjoint cells. Then the data points in each cell are approximately represented as the centroid. Since the inter-centroid distances can be pre-computed as a lookup table, quantization methods can efficiently calculate pairwise distance.

With the progress in deep learning, the past few years have seen many deep hashing methods (Yuan et al. 2020; Liu et al. 2016) with impressive performance. Unfortunately, annotating tons of data in real-world applications is expensive, making it hard to apply these supervised methods. Recent research interests have arisen in unsupervised deep hashing to address this issue, but existing works are not satisfactory

enough. On the one hand, most existing studies in unsupervised deep hashing focus on preserving the information from continuous features. They mostly use quantization loss (Lin et al. 2016; Chen, Cheung, and Wang 2018) and similarity reconstruction loss (Shen et al. 2018) as the learning objectives, resulting in heavy reliance on the quality of extracted features from pre-trained backbones (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2015; He et al. 2016). If an adopted backbone generalizes poorly in the target domain, the unsatisfactory features will degrade the output binary codes fundamentally. On the other hand, Lin et al. (2016); Huang et al. (2017) introduced rotation invariance of images to learn deep hashing, but weak negative samples and ineffective training schemes led to inferior performance.

This paper focuses on *unsupervised deep quantization*. To make better use of unlabeled training data, we perform Contrastive Learning (CL) (Hadsell, Chopra, and LeCun 2006) that learns representations by mining visual-semantic invariance from inputs (Chen et al. 2020; He et al. 2020). CL has become a promising direction toward deep unsupervised representation, but CL-based deep quantization remains non-trivial. Specifically, we find three challenges in this task: (**i**) *Sampling bias*. Without label supervision, a randomly sampled batch may contain positive samples that are falsely taken as negatives. (**ii**) *Model degradation*. We observe that quantization codewords of the same codebook tend to get closer during CL, which gradually degrades the representation ability and harms the model. (**iii**) *The conflict between effect and efficiency in training*. CL benefits from a large batch size that ensures enough negative samples, while a single GPU can afford a limited batch size. Training by CL often requires multi-GPU synchronization, which is complex in engineering and less efficient. To improve the efficiency, some recent studies (Wu et al. 2018; Misra and Maaten 2020) enable small-batch CL by caching embeddings in a memory bank and reusing them as negatives in later iterations. However, as the encoder updates, the cached embeddings will expire and affect the effect of CL.

To tackle the problems, we propose **C**ontrastive **Q**uantization with Code **Me**mory (**MeCoQ**) that combines memory-based CL and deep quantization in a *mutually beneficial* framework. Specifically, (**i**) *MeCoQ is bias-aware*. We adopt a debiased framework (Chuang et al. 2020) that can correct the sampling bias in CL. (**ii**) *MeCoQ avoids degeneration*.

We find that codeword diversity is critical to prevent the CL-based quantization model from degeneration. Hence, we design a codeword regularization to reinforce MeCoQ. (**iii**) *MeCoQ boosts CL effectively and efficiently.* We propose a novel memory bank for quantization codes that shows lower feature drift (Wang et al. 2020) than existing feature memories. Thus, it can retain cached negatives valid for a longer period and enhance the effect of CL, without heavy computations from the momentum encoder (He et al. 2020).

Our contributions can be summarized as follows.

- We provide a novel solution to unsupervised deep quantization, which combines contrastive learning and deep quantization in a mutually beneficial framework.
- We show that codeword diversity is critical to prevent contrastive deep quantization from model degeneration.
- We propose a quantization code memory to enhance the effect of memory-based contrastive learning.
- Extensive experiments on public benchmarks show that MeCoQ outperforms state-of-the-art methods.

Related Work

Unsupervised Hashing Deep hashing methods with Convolutional Neural Networks (CNNs) (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2015; He et al. 2016) usually perform better than non-deep hashing methods. Existing deep hashing methods can be categorized into generative (Dai et al. 2017; Duan et al. 2017; Zieba et al. 2018; Song et al. 2018; Dizaji et al. 2018; Shen, Liu, and Shao 2019; Shen et al. 2020; Li and van Gemert 2021; Qiu et al. 2021) or discriminative (Lin et al. 2016; Huang et al. 2017; Su et al. 2018; Chen, Cheung, and Wang 2018; Yang et al. 2018, 2019; Tu, Mao, and Wei 2020) series. Most of them impose various constraints (*i.e.*, loss or regularization terms) such as pointwise constraints: (**i**) quantization error (Duan et al. 2017; Chen, Cheung, and Wang 2018), (**ii**) even bit distribution (Zieba et al. 2018; Shen, Liu, and Shao 2019), (**iii**) bit irrelevance (Dizaji et al. 2018), (**iv**) maximizing mutual information between features and codes (Li and van Gemert 2021; Qiu et al. 2021); and pairwise constraints: (**v**) preserving similarity among continuous feature vectors (Su et al. 2018; Yang et al. 2018, 2019; Tu, Mao, and Wei 2020). They merely explore statistical characteristics of hash codes or focus on preserving the semantic information from continuous features, which leads to heavy dependence on high-quality pre-trained features. On the other hand, DeepBit (Lin et al. 2016) and UTH (Huang et al. 2017) introduce rotation invariance of images to improve deep hashing. Unfortunately, rotation itself is not enough to construct informative negative samples and the training schemes of DeepBit and UTH are less effective, which leads to inferior performances. Most recently, Qiu et al. (2021) combined contrastive learning with deep binary hashing. By taking effective data augmentations and engaging more negative samples, it shows promising results. Different from them, we explore the combination of contrastive learning and deep quantization that is more challenging. We propose a codeword diversity regularization to prevent model degeneration. Besides, we adopt a debiasing mechanism and

propose a quantization code memory to enhance contrastive learning, yielding better results.

Contrastive Learning Contrastive Learning (CL) (Hadsell, Chopra, and LeCun 2006) based representation learning has drawn increasing attention. Wu et al. (2018) proposed an instance discrimination method that combines a non-parametric classifier (*i.e.*, a memory bank) with a cross-entropy loss (*aka* InfoNCE (Oord, Li, and Vinyals 2018) or contrastive loss (He et al. 2020)). Positive samples from the same image are pulled closer and negative samples from other images are pushed apart. The subsequent instance-wise CL methods focus on designing *end-to-end* (Chen et al. 2020), *memory bank*-based (Misra and Maaten 2020), or *momentum encoder*-based (He et al. 2020) architectures. Besides, cluster-wise contrastive methods (Caron et al. 2020; Li et al. 2021) integrate the clustering objective into CL, which shows promising results. Moreover, there are some works addressing sampling bias (Chuang et al. 2020) or exploring effective negative sampling (Robinson et al. 2021) in CL, which improve the training effect. We uncover that codeword diversity is critical to enable CL in deep quantization. Besides, we propose a novel memory that stores quantization codes. Interestingly, without needing a momentum encoder, it can show lower feature drift (Wang et al. 2020) than existing feature memories (Wu et al. 2018; Misra and Maaten 2020) and thus boosts CL effectively.

Modeling Framework
Problem Formulation and Model Overview
Given an unlabeled training set $\mathcal{D}$ of $N_{\mathcal{D}}$ images where each image x can be flattened as a P-dimensional vector, the goal of unsupervised deep quantization is to learn a neural quantizer $Q : \mathbb{R}^P \mapsto \{0, 1\}^B$ that encodes images as B-bit semantic quantization codes (*aka* the binary representations) for efficient image retrieval. To this end, we propose **Co**ntrastive **Q**uantization with Code **Me**mory (**MeCoQ**) in an end-to-end deep learning architecture. As shown in Figure 1, MeCoQ consists of: (**i**) Two operators sampled from the same data augmentation family ($T \sim \mathcal{T}$ and $T' \sim \mathcal{T}$), which are applied to each training image to obtain two correlated views (x_q and $x_k \in \mathbb{R}^P$). (**ii**) A deep embedding module h combined with a standard CNN (Simonyan and Zisserman 2015) and a transform layer, which produces a continuous embedding $z \in \mathbb{R}^D$ for an input view $x \in \mathbb{R}^P$. (**iii**) A trainable quantization module that produces a *soft* quantization code vector p and the reconstruction $\hat{z} \in \mathbb{R}^D$ for $z \in \mathbb{R}^D$. In inference, it directly outputs the *hard* quantization code vector $b \in \{0, 1\}^B$ for image x. (**iv**) A code memory bank $\mathcal{M}$ to cache the quantization code vectors of images, which serves as an additional source of negative training keys and is not involved in inference.

Debiased Contrastive Learning for Quantization
Trainable Quantization It is hard to integrate traditional quantization (Jegou, Douze, and Schmid 2010) into the deep learning framework because the codeword assignment step is clustering-based and can not be trained by backpropagation. To enable end-to-end learning in MeCoQ, we
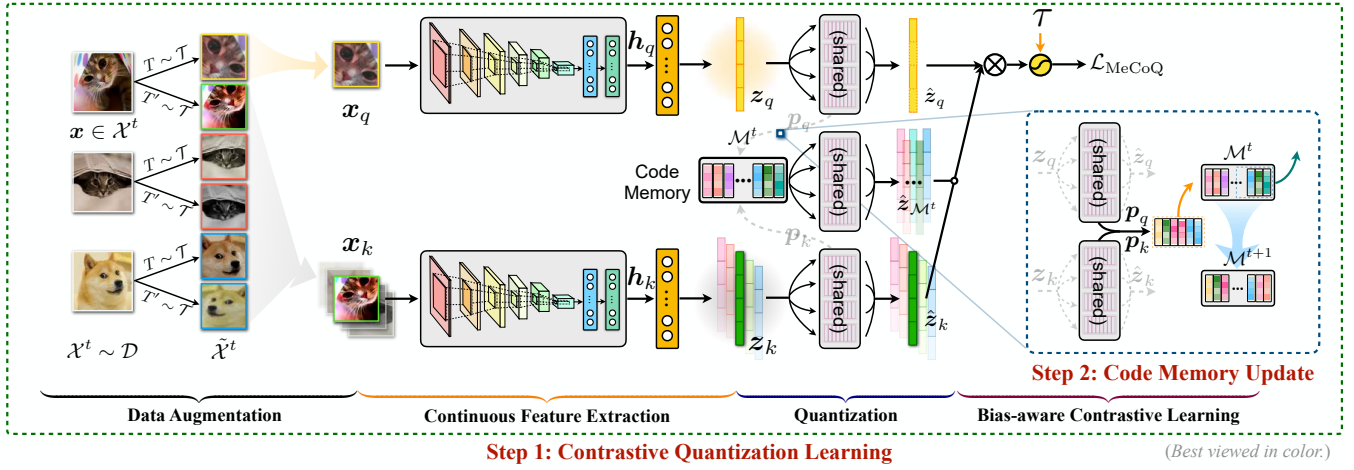
Figure 1: The framework of MeCoQ. After augmentation, we set one view for an image as the training *query* x_q and leave the other view x_k along with the views of other images as the *keys* in contrastive learning. Then, we extract the embeddings for these images and forward them to the quantization module to get quantized reconstructions. Embeddings reconstructed from the code memory bank serve as additional negative keys that boost contrastive learning. Next, we maximize the similarity between the query and the positive key (*i.e.*, the other view from the same image) and minimize the similarities of negative query-key pairs. Finally, we update the code memory bank with the quantization codes of the views of current image batch.

apply a trainable quantization scheme. Denote the quantization codebooks as $\mathcal{C} = C^1 \times C^2 \times \cdots \times C^M$, where the m-th codebook $C^m \in \mathbb{R}^{K \times d}$ consists of K codewords $c_1^m, c_2^m, \cdots, c_K^m \in \mathbb{R}^d$. Assume that $z \in \mathbb{R}^D$ can be divided into M equal-length d-dimensional segments, *i.e.*, $z \in \mathbb{R}^D \equiv [z^1, \cdots, z^M], z^m \in \mathbb{R}^d, d = D/M, 1 \le m \le M$. Given a vector, each codebook is used to quantize one segment respectively. In the m-th d-dimensional subspace, the segment and codewords are first normalized:

$$z^m \leftarrow z^m / \|z^m\|_2 \,, \; c_i^m \leftarrow c_i^m / \|c_i^m\|_2 \,. \quad (1)$$

Then each segment is quantized with codebook attention by

$$\hat{z}^m = \text{Attention}(z^m, C^m, C^m) = \sum_{i=1}^{K} p_i^m c_i^m, \quad (2)$$

where attention score p_i^m is computed with the α-softmax:

$$p_i^m = \text{softmax}_\alpha \left(z^{m\top} c_i^m \right) = \frac{\exp \left(\alpha \cdot z^{m\top} c_i^m \right)}{\sum_{j=1}^{K} \exp \left(\alpha \cdot z^{m\top} c_j^m \right)}. \quad (3)$$

The α-softmax is a differentiable alternative to the argmax that relaxes the discrete optimization of hard codeword assignment to a trainable form. Finally, we get *soft* quantization code and the *soft* quantized reconstruction of z as

$$p = \text{concatenate} \left(p^1, p^2, \cdots, p^M \right) \in \mathbb{R}^{KM}, \quad (4)$$

$$\hat{z} = \text{concatenate} \left(\hat{z}^1, \hat{z}^2, \cdots, \hat{z}^M \right). \quad (5)$$

Debiased Contrastive Learning We conduct Contrastive Learning (CL) based on the soft quantized reconstruction vectors. Because negative keys for a training query are randomly sampled from unlabeled training set, there are unavoidably some *false*-negative keys that harm CL. To tackle

this problem, we adopt a bias-aware framework (Chuang et al. 2020). The debiased CL loss is defined as

$$\mathcal{L}_{\text{DCL}} = - \sum_{q=1}^{2N} \log \frac{\exp(\frac{s_{q,k^+}}{\tau})}{\exp(\frac{s_{q,k^+}}{\tau}) + \mathcal{N}_{\text{In-Batch}}}, \quad (6)$$

where N is the batch size, q and k^+ denote the indices of the training query and the positive key in the augmented batch, τ is the temperature hyper-parameter. The similarity between a query q and a key k is defined as $s_{q,k} \triangleq \hat{z}_q^\top \hat{z}_k$. The debiased in-batch negative term in Eq.(6) is defined as

$$\mathcal{N}_{\text{In-Batch}} \triangleq \sum_{\substack{k^-=1, \\ k^- \notin \{q, k^+\}}}^{2N} \left[\frac{\exp(\frac{s_{q,k^-}}{\tau})}{1 - \rho^+} - \frac{\rho^+ \cdot \exp(\frac{s_{q,k^+}}{\tau})}{1 - \rho^+} \right], \quad (7)$$

where k^- denotes the indices of negative key and ρ^+ is the positive prior for bias correction.

Regularization to Avoid Model Degeneration

Model Degeneration We find that deep quantization is prone to degenerate as CL goes on. To show the isuue, we reduce irrelevant factors[1] and train a simplified 32-bit MeCoQ by vanilla CL on the Flickr25K dataset. As shown in Figure 2(a), the model performance (solid red line) declines while the loss (dash red line) rises with fluctuation. It seems that the success of CL for continuous representation doesn't generalize to quantization directly. Considering the difference in models, we investigate the behavior of quantization

---

[1]We have done pretest about the effect of debiasing mechanism to the issue. As the results showed that it does not change the phenomenon, we exclude it for a concise interpretation in this section.

(a) Loss and MAP Curves.

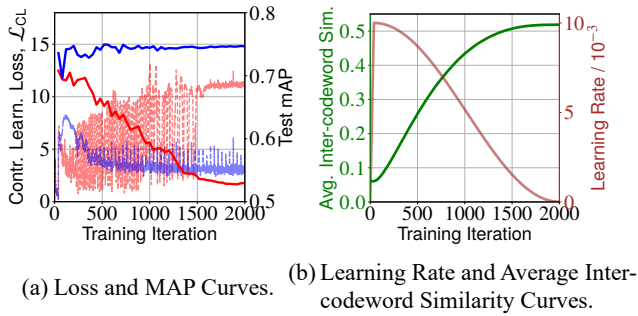(b) Learning Rate and Average Inter-codeword Similarity Curves.

Figure 2: MeCoQ degenerates during contrastive learning because codewords in the same codebook are getting closer. The codeword diversity regularization can avoid it.

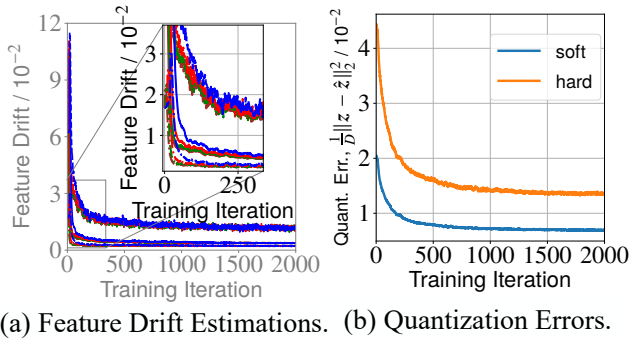(a) Feature Drift Estimations. (b) Quantization Errors.

Figure 3: We estimate the feature drifts of original features, soft quantized features, and hard quantized features. The soft features show lower drift than the original features because moderate quantization error can compensate for the drift.

codebooks during CL. Intuitively, we observe the changes of average inter-codeword similarity, namely,

$$\Omega_{\mathcal{C}} = \frac{1}{MK^2} \sum_{m=1}^{M} \sum_{i=1}^{K} \sum_{j=1}^{K} \boldsymbol{c}_i^{m\top} \boldsymbol{c}_j^m. \quad (8)$$

Figure 2(b) shows a monotonic increase of $\Omega_{\mathcal{C}}$, which slows down as the learning rate decreases. It suggests that the optimization leads to a degenerated solution.

Here we discuss what may cause this phenomenon. Recall that vanilla CL loss *w.r.t.* a training query $\boldsymbol{x}_q$ is

$$\mathcal{L}_{\mathrm{CL}}(\boldsymbol{x}_q) = -\log \frac{\exp\left(\frac{s_{q,k+}}{\tau}\right)}{\exp\left(\frac{s_{q,k+}}{\tau}\right) + \sum_{\substack{k^-=1, \\ k^- \notin \{q,k^+\}}}^{2N} \exp\left(\frac{s_{q,k^-}}{\tau}\right)}. \quad (9)$$

Proposition 1. *Suppose that $\boldsymbol{x}_{k+}$ is the positive training key and $\boldsymbol{x}_{k-}$ is a negative key to $\boldsymbol{x}_q$. $\hat{\boldsymbol{z}}_{k+}$ and $\hat{\boldsymbol{z}}_{k-}$ are the reconstructed embeddings* w.r.t. *$\boldsymbol{x}_{k+}$ and $\boldsymbol{x}_{k-}$, then we have*

$$\left\| \frac{\partial \mathcal{L}_{\mathrm{CL}}(\boldsymbol{x}_q)}{\partial \hat{\boldsymbol{z}}_{k+}} \right\| > \left\| \frac{\partial \mathcal{L}_{\mathrm{CL}}(\boldsymbol{x}_q)}{\partial \hat{\boldsymbol{z}}_{k-}} \right\| > 0. \quad (10)$$

It suggests that the scale of loss gradient *w.r.t.* the positive key is greater than that *w.r.t.* any negative key. Engaging more negatives leads to a larger gap between such

scales. Besides, to reduce the deviation between soft quantization in training and hard quantization in inference, we take a relatively large α (10 by default) in the codeword attention (Eq.(2)). It makes each soft reconstruction and assigned codeword approximate at the forward step. In the back-propagation, their gradients are also approximate. Thus, it can hold when replacing $\hat{\boldsymbol{z}}_{k+}$ and $\hat{\boldsymbol{z}}_{k-}$ in Proposition 1 with assigned codewords. If the query and the positive key are assigned to different codewords, there will be a large gradient to pull these codewords closer. We find it irreversible without explicit control, because of the insufficient frequency of the same assignment for negative pairs along with subtle gradients to push the same of codewords away. As a result, the representation ability of codebooks degrades and the model degenerates.

Codeword Diversity Regularization To avoid degeneration, we regularize the optimization by imposing $\Omega_{\mathcal{C}} \leq \epsilon$, where ϵ is a fixed bound. In practice, we set it as a loss term that encourages codeword diversity and guides deep quantization model to pay more attention to proper codeword assignment rather than violently moving the codewords. The blue lines in Figure 2(a) show that the regularization effectively avoids the issue and also helps to calm the loss down.

Quantization Code Memory to Boost Training

Effective CL methods rely on sufficient negative samples to learn discriminative representations. Therefore, existing memory-based methods cache the image embeddings and serve them as the negatives in the later training. However, as the model keeps updating, the early cached embeddings expire and become noises to CL. To enhance the effect of memory, MoCo (He et al. 2020) introduces a momentum encoder that mitigates the embedding aging issue at a higher computation cost. In contrast, we find an elegant solution that achieves a similar effect more efficiently.

Feature Drift We follow Wang et al. (2020) to investigate the embedding aging issue. We define the *feature drift* of a neural embedding model h by

$$\mathrm{Drift}(\mathcal{X}', t; \Delta t) \triangleq \frac{1}{|\mathcal{X}'|} \sum_{\boldsymbol{x} \in \mathcal{X}'} \left\| h(\boldsymbol{x}; \boldsymbol{\theta}_h^t) - h(\boldsymbol{x}; \boldsymbol{\theta}_h^{t-\Delta t}) \right\|_2^2,$$

(11)

where $\mathcal{X}'$ is a given image set for estimation, $\boldsymbol{\theta}_h$ is the parameters of h, t and Δt denote the number and the interval of training iterations (*i.e.*, batches) respectively. We train a 32-bit CL-based quantization model and compute its feature drift based on original embeddings, soft quantized reconstructions, and hard quantized reconstructions.

As shown in Figure 3(a), the embeddings violently change at the early stage, after which they become relatively stable. Moreover, it is surprising that the *soft* quantized embeddings show even lower feature drifts than the original embeddings. We realize that undesirable quantization error (illustrated in Figure 3(b)) that used to be eliminated in quantization models *partially*[2] offsets the feature drift, keeping the cached information valid for a longer time.

---

[2]Note that *hard* quantization shows higher feature drift. It doesn't benefit from the offset because of large quantization error.

Memory-augmented Training Base on the above facts, we start using a memory bank $\mathcal{M}$ with $N_\mathcal{M}$ slots to store *soft quantization codes* after the warm-up stage. In each training iteration, we fetch the cached codes $\boldsymbol{p}_{\mathcal{M}_1}, \boldsymbol{p}_{\mathcal{M}_2}, \cdots, \boldsymbol{p}_{\mathcal{M}_{N_\mathcal{M}}}$ from the memory bank. Then, we forward them to the quantization module and respectively reconstruct the embeddings $\hat{\boldsymbol{z}}_{\mathcal{M}_1}, \hat{\boldsymbol{z}}_{\mathcal{M}_2}, \cdots, \hat{\boldsymbol{z}}_{\mathcal{M}_{N_\mathcal{M}}}$ by Eq.(2) and (5). Finally, we integrate these embeddings to Eq.(6) and formulate the memory-augmented loss as

$$\mathcal{L}_{\text{MeCoQ}} = -\sum_{q=1}^{2N} \log \frac{\exp(\frac{s_{q,k+}}{\tau})}{\exp(\frac{s_{q,k+}}{\tau}) + \mathcal{N}_{\text{In-Batch}} + \mathcal{N}_{\text{Memory}}},$$
(12)

where the added negative term about code memory is

$$\mathcal{N}_{\text{Memory}} \triangleq \sum_{i=1}^{N_\mathcal{M}} \left[\frac{\exp(\frac{s_{q,\mathcal{M}_i}}{\tau})}{1-\rho^+} - \frac{\rho^+ \cdot \exp(\frac{s_{q,k+}}{\tau})}{1-\rho^+} \right]. \quad (13)$$

At the end of each training iteration, we update the memory as a queue, *i.e.*, the current batch is enqueued and an equal number of the oldest slots are removed. To simplify engineering, we set the queue size $N_\mathcal{M}$ to a multiple of the batch size N so that we update the memory bank by batches.

Learning Algorithm

The learning objective of MeCoQ is

$$\min_{\boldsymbol{\theta}_h, \mathcal{C}} \mathbb{E} \, \mathcal{L}_{\text{MeCoQ}} + \beta \|\boldsymbol{\theta}_h\|_2^2 + \gamma \Omega_\mathcal{C}, \quad (14)$$

where $\mathcal{L}_{\text{MeCoQ}}$ is formulated as Eq.(12), $\Omega_\mathcal{C}$ is defined as Eq.(8). $\boldsymbol{\theta}_h$ denotes the network parameters of the embedding module h, β and γ are the trade-off hyper-parameters. The training process is quite efficient as we formulate the whole problem in a deep learning framework. Many off-the-shelf optimizers can be applied within a few code lines.

Encoding and Retrieval

In inference, we encode the database with hard quantization:

$$i_{db}^m = \arg\max_{1 \le i \le K} \boldsymbol{z}_{db}^{m\top} \boldsymbol{c}_i^m, \qquad \hat{\boldsymbol{z}}_{db}^m = \boldsymbol{c}_{i_{db}^m}^m. \quad (15)$$

We can aggregate the indices $[i_{db}^1, i_{db}^2, \cdots, i_{db}^M]$ and convert it into a code vector $\boldsymbol{b}_{db}$ for the database image $\boldsymbol{x}_{db}$. Given a retrieval query image $\boldsymbol{x}_q$, we extract its deep embedding and cut the vector into M equal-length segments, *i.e.*, $\boldsymbol{z}_q = [\boldsymbol{z}_q^1; \boldsymbol{z}_q^2; \cdots; \boldsymbol{z}_q^M], \boldsymbol{z}_q^m \in \mathbb{R}^d$. We adopt Asymmetric Quantized Similarity (AQS) (Jegou, Douze, and Schmid 2010) as the metric, which computes the similarity between $\boldsymbol{z}_q$ and the reconstruction of a database point $\hat{\boldsymbol{z}}_{db}$ by

$$\text{AQS}(\boldsymbol{x}_q, \boldsymbol{x}_{db}) = \sum_{m=1}^{M} \frac{\boldsymbol{z}_q^{m\top} \hat{\boldsymbol{z}}_{db}^m}{\|\boldsymbol{z}_q^m\|_2} = \sum_{m=1}^{M} \frac{\boldsymbol{z}_q^{m\top} \boldsymbol{c}_{i_{db}^m}^m}{\|\boldsymbol{z}_q^m\|_2}. \quad (16)$$

We can set up a query-specific lookup table $\boldsymbol{\Xi}_q \in \mathbb{R}^{M \times K}$ for each $\boldsymbol{x}_q$, which stores the pre-computed similarities between the segments of $\boldsymbol{x}_q$ and all codewords. Specifically,

$\Xi_{q,i_{db}^m}^m = \boldsymbol{z}_q^{m\top} \boldsymbol{c}_{i_{db}^m}^m / \|\boldsymbol{z}_q^m\|_2$. Hence, the AQS can be efficiently computed by summing chosen items from the lookup table according to the quantization code, *i.e.*,

$$\text{AQS}(\boldsymbol{x}_q, \boldsymbol{x}_{db}) = \sum_{m=1}^{M} \Xi_{q,i_{db}^m}^m, \quad (17)$$

where i_{db}^m is the index of codeword in the m-th codebook.

Experiments

Setup

Datasets (i) **Flickr25K** (Huiskes and Lew 2008) contains 25k images from 24 categories. We follow Li and van Gemert (2021) to randomly pick 2,000 images as the testing queries, while another 5,000 images are randomly selected from the rest of the images as the training set. (ii) **CIFAR-10** (Krizhevsky and Hinton 2009) contains 60k images from 10 categories. We consider two typical experiment protocols. *CIFAR-10 (I)*: We follow Li and van Gemert (2021) to use 1k images per class (totally 10k images) as the test query set, and the remaining 50k images are used for training. *CIFAR-10 (II)*: Following Qiu et al. (2021) we randomly select 1,000 images per category as the testing queries and 500 per category as the training set. All images except those in the query set serve as the retrieval database. (iii) **NUS-WIDE** (Chua et al. 2009) is a large-scale image dataset containing about 270k images from 81 categories. We follow Li and van Gemert (2021) to use the 21 most popular categories for evaluation. 100 images per category are randomly selected as the testing queries while the remaining images form the database and the training set.

Metrics We adopt the typical metric, Mean Average Precision (**MAP**), from previous works (Yang et al. 2018; Li and van Gemert 2021; Qiu et al. 2021). It is defined as

$$\text{MAP@N} = \frac{1}{|\mathcal{Q}|} \sum_{\boldsymbol{x}_q \in \mathcal{Q}} \left(\frac{\sum_{n=1}^{N} \text{Prec}_q(n) \cdot \mathbb{I}_{\{\boldsymbol{x}_n \in \mathcal{R}_q\}}}{|\mathcal{R}_q|} \right),$$
(18)

where $\mathcal{Q}$ is test query image set and n is the index of a database image in a returned rank list. $\text{Prec}_q(n)$ is the precision at cut-off n in the rank list *w.r.t.* $\boldsymbol{x}_q$. $\mathcal{R}_q$ is the set of all relevant images *w.r.t.* $\boldsymbol{x}_q$. $\mathbb{I}_{\{\cdot\}}$ is an indicator function. We follow previous works to adopt MAP@1000 for CIFAR-10 (I) and (II), MAP@5000 for Flickr25K and NUS-WIDE.

Models We compare the retrieval performance of **MeCoQ** with **24** classic or state-of-the-art unsupervised baselines, including: (i) 5 shallow hashing methods: **LSH** (Charikar 2002), **SpeH** (Weiss, Torralba, and Fergus 2008), **SH** (Salakhutdinov and Hinton 2009), **SphH** (Heo et al. 2012) and **ITQ** (Gong et al. 2012). (ii) 2 shallow quantization methods: **PQ** (Jegou, Douze, and Schmid 2010) and **OPQ** (Ge et al. 2013). (iii) 15 deep binary hashing methods: **Deep-Bit** (Lin et al. 2016), **UTH** (Huang et al. 2017), **SAH** (Do et al. 2017), **SGH** (Dai et al. 2017), **HashGAN** (Dizaji et al. 2018), **GreedyHash** (Su et al. 2018), **BinGAN** (Zieba et al. 2018), **BGAN** (Song et al. 2018), **SSDH** (Yang et al. 2018), **DVB** (Shen, Liu, and Shao 2019), **DistillHash** (Yang et al.

Method ↓	Venue ↓	Type ↓	Flickr25K			CIFAR-10 (I)			CIFAR-10 (II)			NUS-WIDE		
			16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH+VGG	STOC'02	BH	56.11	57.08	59.26	14.38	15.86	18.09	12.55	13.76	15.07	38.52	41.43	43.89
SpeH+VGG	NeurIPS'08	BH	59.77	61.36	64.08	27.09	29.44	32.65	27.20	28.50	30.00	51.70	51.10	51.00
SH+VGG	IJAR'09	BH	60.02	63.30	64.17	28.28	28.86	28.51	24.68	25.34	27.16	46.85	53.63	56.28
SphH+VGG	CVPR'12	BH	61.32	62.47	64.49	26.90	31.75	35.25	25.40	29.10	33.30	49.50	55.80	58.20
ITQ+VGG	TPAMI'12	BH	63.30	65.92	68.86	34.41	35.41	38.82	30.50	32.50	34.90	62.70	64.50	66.40
PQ+VGG	TPAMI'10	Q	62.75	66.63	69.40	27.14	33.30	37.67	28.16	30.24	30.61	65.39	67.41	68.56
OPQ+VGG	CVPR'13	Q	63.27	68.01	69.86	27.29	35.17	38.48	32.17	33.50	34.46	65.74	68.38	69.12
DeepBit	CVPR'16	DBH	62.04	66.54	68.34	19.43	24.86	27.73	20.60	28.23	31.30	39.20	40.30	42.90
UTH	ACMMMW'17	DBH	-	-	-	-	-	-	-	-	-	45.00	49.50	54.90
SAH	CVPR'17	DBH	-	-	-	41.75	45.56	47.36	-	-	-	-	-	-
SGH	ICML'17	DBH	72.10	72.84	72.83	34.51	37.04	38.93	43.50	43.70	43.30	59.30	59.00	60.70
HashGAN	CVPR'18	DBH	72.11	73.25	75.46	44.70	46.30	48.10	42.81	47.54	47.29	68.44	70.56	71.71
GreedyHash	NeurIPS'18	DBH	69.91	70.85	73.03	44.80	47.20	50.10	45.76	48.26	53.34	63.30	69.10	73.10
BinGAN	NeurIPS'18	DBH	-	-	-	-	-	-	47.60	51.20	52.00	65.40	70.90	71.30
BGAN	AAAI'18	DBH	-	-	-	-	-	-	52.50	53.10	56.20	68.40	71.40	73.00
SSDH	IJCAI'18	DBH	75.65	77.10	76.68	36.16	40.37	44.00	33.30	38.29	40.81	58.00	59.30	61.00
DVB	IJCV'19	DBH	-	-	-	-	-	-	40.30	42.20	44.60	60.40	63.20	66.50
DistillHash	CVPR'19	DBH	-	-	-	-	-	-	-	-	-	62.70	65.60	67.10
TBH	CVPR'20	DBH	74.38	76.14	77.87	54.68	58.63	62.47	53.20	57.30	57.80	71.70	72.50	73.50
MLS³RDUH	IJCAI'20	DBH	-	-	-	-	-	-	-	-	-	71.30	72.70	75.00
Bi-half Net	AAAI'21	DBH	76.07	77.93	78.62	56.10	57.60	59.50	49.97	52.04	55.35	76.86	78.31	79.94
CIBHash	IJCAI'21	DBH	77.21	78.43	79.50	59.39	63.67	65.16	59.00	62.20	64.10	79.00	80.70	81.50
DBD-MQ	CVPR'17	DQ	-	-	-	21.53	26.50	31.85	-	-	-	-	-	-
DeepQuan	IJCAI'18	DQ	-	-	-	39.95	41.25	43.26	-	-	-	-	-	-
MeCoQ (Ours)	AAAI'22	DQ	**81.31**	**81.71**	**82.68**	**68.20**	**69.74**	**71.06**	**62.88**	**64.09**	**65.07**	**80.18**	**82.16**	**83.24**

Table 1: Mean Average Precision (MAP, %) results for different number of bits on Flickr25K, CIFAR-10 (I and II) and NUS-WIDE datasets. 'D', 'Q' and 'BH' indicate 'Deep', 'Quantization' and 'Binary Hashing' for short in 'Type' column.

2019), **TBH** (Shen et al. 2020), **MLS³RDUH** (Tu, Mao, and Wei 2020), **Bi-half Net** (Li and van Gemert 2021) and **CIBHash** (Qiu et al. 2021). (**iv**) 2 deep quantization methods: **DBD-MQ** (Duan et al. 2017) and **DeepQuan** (Chen, Cheung, and Wang 2018). We carefully collect their results from related literature. When results about some baselines on a certain benchmark are not available (*e.g.* CIBHash on Flickr25K dataset), we try to run their open-sourced codes (if *available* and *executable*) and report the results.

Implementation Settings We implement MeCoQ with Pytorch (Paszke et al. 2019). We follow the standard evaluation protocol (Qiu et al. 2021; Li and van Gemert 2021) of unsupervised deep hashing to use the VGG16 (Simonyan and Zisserman 2015). Specifically, for shallow models, we extract 4096-dimensional deep $fc7$ features as the model input. For deep models, we directly use raw image pixels as input and adopt the pre-trained VGG16 ($conv1 \sim fc7$) as the backbone network. We use the data augmentation scheme in Qiu et al. (2021) that combines random cropping, horizontal flipping, image graying, and randomly applied color jitter and blur. The default hyper-parameter settings are as follows. (**i**) We set the batch size as 128 and the maximum epoch as 50. (**ii**) The queue length (*i.e.*, the memory bank size), $N_{\mathcal{M}} = 384$. (**iii**) The smoothness factor of codeword assignment in Eq.(3), $\alpha = 10$. (**iv**) The codeword number of each codebook, $K = 256$ such that each image is encoded by $B = M \log_2 K = 8M$ bits (*i.e.*, M bytes). (**v**) The positive prior, $\rho^+ = 0.1$ for CIFAR-10 (I and II), $\rho^+ = 0.15$
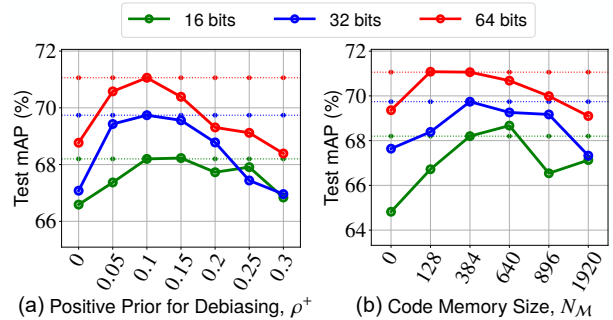

Figure 4: Sensitivities of ρ^+ and $N_{\mathcal{M}}$ on CIFAR-10 (I). The dotted lines indicate the MAP results of default settings.

for Flickr25K and NUS-WIDE. (**vi**) The starting epoch for the memory module are set to 5 on Flickr25K, 10 on NUS-WIDE and 15 on CIFAR-10 (I and II).

Results and Analysis

Comparison with Existing Methods The MAP results in Tables 1 show that MeCoQ substantially outperforms all the compared methods. Specifically, compared with CIBHash, a latest and strong baseline, MeCoQ achieves average MAP increases of **3.52**, **6.92**, **2.24** and **1.46** on Flickr25K, CIFAR-10 (I), (II) and NUS-WIDE datasets, respectively. Besides, we can get two findings from the MAP results. (**i**) Deep methods do not always outperform shallow methods with

Dataset →	Flickr25K			CIFAR-10 (I)			CIFAR-10 (II)			NUS-WIDE		
Method ↓	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
MeCoQ	**81.31**	**81.71**	**82.68**	**68.20**	**69.74**	**71.06**	**62.88**	**64.09**	**65.07**	**80.18**	**82.16**	**83.24**
MeCoQ w/o debiasing	80.03	80.19	80.67	66.59	67.08	68.77	60.51	62.81	62.85	79.64	80.68	82.30
MeCoQ w/o $\Omega_{\mathcal{C}}$	54.00	57.53	58.84	39.03	36.20	42.75	32.54	35.67	36.40	51.33	57.29	61.01
MeCoQ w/o $\mathcal{M}$	78.77	79.50	80.76	64.82	67.64	69.36	59.78	61.22	63.80	76.70	79.13	81.43
MeCoQ feature $\mathcal{M}$	78.39	79.61	80.13	65.39	67.62	70.57	62.55	63.32	64.84	77.71	79.90	81.64
MeCoQ hard code $\mathcal{M}$	60.77	63.10	68.49	33.95	40.65	46.76	34.40	37.55	44.90	69.46	72.23	74.95
MeCoQ w/o delaying $\mathcal{M}$	79.02	78.44	78.17	66.76	67.82	69.04	59.79	61.42	62.60	79.72	81.39	82.67

Table 2: Mean Average Precision (MAP, %) results for different MeCoQ variants with different number of bits on Flickr25K, CIFAR-10 (I and II) and NUS-WIDE datasets. The subscript results are the MAP drops compared with full MeCoQ.

CNN features. For instance, DeepBit and UTH do not outperform PQ and OPQ with CNN features on NUS-WIDE. It implies that without label supervision, some deep hashing methods are less effective to take good advantage of pretrained CNNs. **(ii)** Contrastive Learning (CL) is effective to learn deep hashing models. The two CL-based methods in Table 1, CIBHash and MeCoQ, perform best on all datasets. Moreover, with debiasing mechanism and code memory, MeCoQ shows notable improvements over CIBHash.

Component Analysis We set 6 MeCoQ variants to analyse the contributions of components: **(i)** MeCoQ $_{\text{w/o debiasing}}$ removes debiasing mechanism by setting $\rho^+ = 0$ in Eq.(7) and (13); **(ii)** MeCoQ $_{\text{w/o } \Omega_{\mathcal{C}}}$ removes codeword regularization, $\Omega_{\mathcal{C}}$; **(iii)** MeCoQ $_{\text{w/o } \mathcal{M}}$ removes the code memory, $\mathcal{M}$; **(iv)** MeCoQ $_{\text{feature } \mathcal{M}}$ replaces soft code memory by feature memory; **(v)** MeCoQ $_{\text{hard code } \mathcal{M}}$ replaces soft code memory by hard code memory; **(vi)** MeCoQ $_{\text{w/o delaying } \mathcal{M}}$ begins memory-augmented training at the very start of the learning. We can make the following summaries based on the Table 2.

Debiasing improves performance. MeCoQ outperforms MeCoQ $_{\text{w/o debiasing}}$ by 1.60, 2.19, 1.96 and 0.99 of average MAPs on Flickr25K, CIFAR-10 (I), (II) and NUS-WIDE, which shows that correcting the sampling bias can improve model training. Figure 4(a) shows that the optimal ρ^+ on CIFAR-10 (I) is about 0.1, which means that we may randomly drop a false-negative sample from the training set with a 10% probability. It is consistent with the property of CIFAR-10 (I) that each category accounts for 10%.

The codeword diversity regularization avoids model degeneration. MeCoQ outperforms MeCoQ $_{\text{w/o } \Omega_{\mathcal{C}}}$ by 25.11, 30.34, 29.14 and 25.32 of average MAPs on 4 datasets. It demonstrates the importance of regularization.

Soft code memory is effective and efficient to enhance contrastive learning. MeCoQ outperforms MeCoQ $_{\text{w/o } \mathcal{M}}$ by 2.22, 2.39, 2.41 and 2.77 of average MAPs on 4 datasets, which verifies the worth of using $\mathcal{M}$. Besides, Figure 4(b) shows that enlarging memory to cache more negatives improves MeCoQ, while the improvement tends to drop as memory size exceeds a certain range. The reason is that the low feature drift phenomenon only holds within a limited period. We can also learn that fewer bits with larger quantization errors allow a slightly larger memory because the quantization error can partially offset the feature drift. Moreover, as shown in Table 3, using code memory is efficient in GPU memory and computation. It can achieve better results

Method	#Neg.	Time / Ep.	GPU Mem.	MAP
N=128, w/o $\mathcal{M}$	128	528s	5925MB	67.64
N=256, w/o $\mathcal{M}$	256	550s	10927MB	68.66
N=128, $N_{\mathcal{M}}$=128	256	536s	5927MB	68.39
N=128, $N_{\mathcal{M}}$=384	512	547s	5933MB	69.74
N=128, $N_{\mathcal{M}}$=896	1024	563s	5947MB	69.17

Table 3: Model profiling results on CIFAR-10 (I) dataset, including the number of negative samples per training query, average training time per epoch in seconds, GPU memory demand in megabytes and MAP for 32 bits, under different batch size (N) and memory size ($N_{\mathcal{M}}$) settings.

with much less GPU memory than enlarging batch size. The marginal increase of time is caused by similarity computations between training queries and cached negatives.

Soft code memory is better than feature memory and hard code memory. MeCoQ outperforms MeCoQ $_{\text{feature } \mathcal{M}}$ by 2.52, 1.81, 0.44 and 2.11 of average MAPs on 4 datasets, because the soft code memory has lower feature drift than feature memory. Surprisingly, MeCoQ $_{\text{hard code } \mathcal{M}}$ fails. It seems that reconstructed features from hard codes become adverse noises rather than valid negatives because the large error of hard quantization leads to an over-large feature drift.

It is better to delay the usage of memory in the learning process. MeCoQ outperforms MeCoQ $_{\text{w/o delaying } \mathcal{M}}$ by 3.36, 1.79, 2.74 and 0.6 of average MAPs on 4 datasets. It implies that using code memory from the very beginning leads to sub-optimal solutions because reusing unstable representations in initial training stage is not recommended.

Conclusions

In this paper, we propose **C**ontrastive **Q**uantization with Code **Me**mory (**MeCoQ**) for unsupervised deep quantization. Different from existing reconstruction-based unsupervised deep hashing methods, MeCoQ learns quantization by contrastive learning. To avoid model degeneration when optimizing MeCoQ, we introduce a codeword diversity regularization. We further improve the memory-based contrastive learning by designing a novel quantization code memory, which shows lower feature drift than existing feature memories without using momentum encoder. Extensive experiments show the superiority of MeCoQ over the state-of-the-art methods.

Acknowledgments

References

Caron, M.; Misra, I.; Mairal, J.; Goyal, P.; Bojanowski, P.; and Joulin, A. 2020. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. In *NeurIPS*.

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*, 380–388.

Chen, J.; Cheung, W. K.; and Wang, A. 2018. Learning Deep Unsupervised Binary Codes for Image Retrieval. In *IJCAI*, 613–619.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. 2020. A simple framework for contrastive learning of visual representations. In *ICML*, 1597–1607. PMLR.

Chua, T.-S.; Tang, J.; Hong, R.; Li, H.; Luo, Z.; and Zheng, Y. 2009. 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*, 1–9.

Chuang, C.-Y.; Robinson, J.; Lin, Y.-C.; Torralba, A.; and Jegelka, S. 2020. Debiased Contrastive Learning. In *NeurIPS*, volume 33, 8765–8775.

Dai, B.; Guo, R.; Kumar, S.; He, N.; and Song, L. 2017. Stochastic generative hashing. In *ICML*, 913–922. PMLR.

Dizaji, K. G.; Zheng, F.; Sadoughi, N.; Yang, Y.; Deng, C.; and Huang, H. 2018. Unsupervised deep generative adversarial hashing network. In *CVPR*, 3664–3673.

Do, T.-T.; Le Tan, D.-K.; Pham, T. T.; and Cheung, N.-M. 2017. Simultaneous feature aggregating and hashing for large-scale image search. In *CVPR*, 6618–6627.

Duan, Y.; Lu, J.; Wang, Z.; Feng, J.; and Zhou, J. 2017. Learning deep binary descriptor with multi-quantization. In *CVPR*, 1183–1192.

Ge, T.; He, K.; Ke, Q.; and Sun, J. 2013. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, 2946–2953.

Gong, Y.; Lazebnik, S.; Gordo, A.; and Perronnin, F. 2012. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, 35(12): 2916–2929.

Hadsell, R.; Chopra, S.; and LeCun, Y. 2006. Dimensionality reduction by learning an invariant mapping. In *CVPR*, volume 2, 1735–1742. IEEE.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 9729–9738.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Heo, J.-P.; Lee, Y.; He, J.; Chang, S.-F.; and Yoon, S.-E. 2012. Spherical hashing. In *CVPR*, 2957–2964. IEEE.

Huang, S.; Xiong, Y.; Zhang, Y.; and Wang, J. 2017. Unsupervised triplet hashing for fast image retrieval. In *ACM MM Workshops*, 84–92.

Huiskes, M. J.; and Lew, M. S. 2008. The mir flickr retrieval evaluation. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, 39–43.

Jegou, H.; Douze, M.; and Schmid, C. 2010. Product quantization for nearest neighbor search. *TPAMI*, 33(1): 117–128.

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, volume 25, 1097–1105.

Li, J.; Zhou, P.; Xiong, C.; and Hoi, S. 2021. Prototypical Contrastive Learning of Unsupervised Representations. In *ICLR*.

Li, Y.; and van Gemert, J. 2021. Deep Unsupervised Image Hashing by Maximizing Bit Entropy. In *AAAI*.

Lin, K.; Lu, J.; Chen, C.-S.; and Zhou, J. 2016. Learning compact binary descriptors with unsupervised deep neural networks. In *CVPR*, 1183–1192.

Liu, H.; Wang, R.; Shan, S.; and Chen, X. 2016. Deep supervised hashing for fast image retrieval. In *CVPR*, 2064–2072.

Misra, I.; and Maaten, L. v. d. 2020. Self-supervised learning of pretext-invariant representations. In *CVPR*, 6707–6717.

Oord, A. v. d.; Li, Y.; and Vinyals, O. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 8024–8035.

Qiu, Z.; Su, Q.; Ou, Z.; Yu, J.; and Chen, C. 2021. Unsupervised Hashing with Contrastive Information Bottleneck. In *IJCAI*.

Robinson, J. D.; Chuang, C.-Y.; Sra, S.; and Jegelka, S. 2021. Contrastive Learning with Hard Negative Samples. In *ICLR*.

Salakhutdinov, R.; and Hinton, G. 2009. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7): 969–978.

Shen, F.; Xu, Y.; Liu, L.; Yang, Y.; Huang, Z.; and Shen, H. T. 2018. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *TPAMI*, 40(12): 3034–3044.

Shen, Y.; Liu, L.; and Shao, L. 2019. Unsupervised binary representation learning with deep variational networks. *IJCV*, 127(11): 1614–1628.

Shen, Y.; Qin, J.; Chen, J.; Yu, M.; Liu, L.; Zhu, F.; Shen, F.; and Shao, L. 2020. Auto-encoding twin-bottleneck hashing. In *CVPR*, 2818–2827.

Simonyan, K.; and Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*.

Song, J.; He, T.; Gao, L.; Xu, X.; Hanjalic, A.; and Shen, H. T. 2018. Binary generative adversarial networks for image retrieval. In *AAAI*.

Su, S.; Zhang, C.; Han, K.; and Tian, Y. 2018. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *NeurIPS*, 806–815.

Tu, R.-C.; Mao, X.-L.; and Wei, W. 2020. MLS3RDUH: Deep Unsupervised Hashing via Manifold based Local Semantic Similarity Structure Reconstructing. In *IJCAI*, 3466–3472.

Wang, J.; Zhang, T.; Sebe, N.; Shen, H. T.; et al. 2017. A survey on learning to hash. *TPAMI*, 40(4): 769–790.

Wang, X.; Zhang, H.; Huang, W.; and Scott, M. R. 2020. Cross-batch memory for embedding learning. In *CVPR*, 6388–6397.

Weiss, Y.; Torralba, A.; and Fergus, R. 2008. Spectral hashing. In *NeurIPS*, 1753–1760.

Wu, Z.; Xiong, Y.; Yu, S. X.; and Lin, D. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *CVPR*, 3733–3742.

Yang, E.; Deng, C.; Liu, T.; Liu, W.; and Tao, D. 2018. Semantic structure-based unsupervised deep hashing. In *IJCAI*, 1064–1070.

Yang, E.; Liu, T.; Deng, C.; Liu, W.; and Tao, D. 2019. Distillhash: Unsupervised deep hashing by distilling data pairs. In *CVPR*, 2946–2955.

Yuan, L.; Wang, T.; Zhang, X.; Tay, F. E.; Jie, Z.; Liu, W.; and Feng, J. 2020. Central similarity quantization for efficient image and video retrieval. In *CVPR*, 3083–3092.

Zieba, M.; Semberecki, P.; El-Gaaly, T.; and Trzcinski, T. 2018. BinGAN: learning compact binary descriptors with a regularized GAN. In *NeurIPS*, 3612–3622.