Auto-Encoding Twin-Bottleneck Hashing

Yuming Shen* [1], Jie Qin*† [1], Jiaxin Chen* [1], Mengyang Yu [1],
Li Liu [1], Fan Zhu [1], Fumin Shen [2], and Ling Shao [1]

[1]Inception Institute of Artificial Intelligence (IIAI), Abu Dhabi, UAE
[2]Center for Future Media, University of Electronic Science and Technology of China, China
ymcidence@gmail.com

Abstract

Conventional unsupervised hashing methods usually take advantage of similarity graphs, which are either pre-computed in the high-dimensional space or obtained from random anchor points. On the one hand, existing methods uncouple the procedures of hash function learning and graph construction. On the other hand, graphs empirically built upon original data could introduce biased prior knowledge of data relevance, leading to sub-optimal retrieval performance. In this paper, we tackle the above problems by proposing an efficient and adaptive code-driven graph, which is updated by decoding in the context of an auto-encoder. Specifically, we introduce into our framework twin bottlenecks (i.e., latent variables) that exchange crucial information collaboratively. One bottleneck (i.e., binary codes) conveys the high-level intrinsic data structure captured by the code-driven graph to the other (i.e., continuous variables for low-level detail information), which in turn propagates the updated network feedback for the encoder to learn more discriminative binary codes. The auto-encoding learning objective literally rewards the code-driven graph to learn an optimal encoder. Moreover, the proposed model can be simply optimized by gradient descent without violating the binary constraints. Experiments on benchmarked datasets clearly show the superiority of our framework over the state-of-the-art hashing methods. Our source code can be found at `https://github.com/ymcidence/TBH`.

1. Introduction

Approximate Nearest Neighbour (ANN) search has attracted ever-increasing attention in the era of big data. Thanks to the extremely low costs for computing Hamming distances, binary coding/hashing has been appreciated as

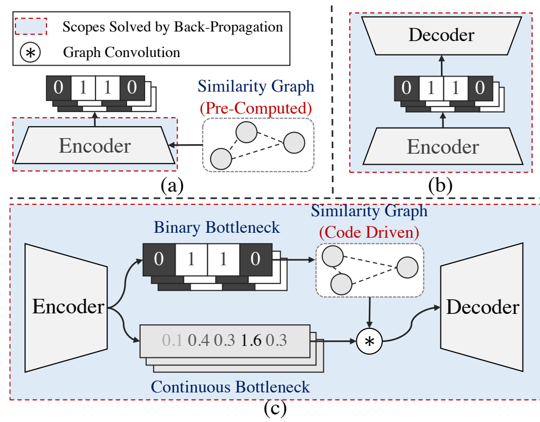

Figure 1. Illustration of the differences between **(a)** graph-based hashing, **(b)** unsupervised generative auto-encoding hashing, and **(c)** the proposed model (TBH).

an efficient solution to ANN search. Similar to other feature learning schemes, hashing techniques can be typically subdivided into supervised and unsupervised ones. Supervised hashing [11, 23, 25, 39, 45, 48], which highly depends on labels, is not always preferable since large-scale data annotations are unaffordable. Conversely, unsupervised hashing [12, 16, 15, 20, 33, 47, 46], provides a cost-effective solution for more practical applications. To exploit data similarities, existing unsupervised hashing methods [29, 30, 43, 35, 41] have extensively employed graph-based paradigms. Nevertheless, existing methods usually suffer from the '**static graph**' problem. More concretely, they often adopt explicitly pre-computed graphs, introducing biased prior knowledge of data relevance. Besides, graphs cannot be adaptively updated to better model the data structure. The interaction between hash function learning and graph construction can be hardly established. The 'static graph' problem greatly hinders the effectiveness of graph-based unsupervised hashing mechanisms.

In this work, we tackle the above long-standing chal-

*Equal Contribution
†Corresponding Author

lenge by proposing a novel adaptive graph, which is directly driven by the learned binary codes. The graph is then seamlessly embedded into a generative network that has recently been verified effective for learning reconstructive binary codes [8, 10, 40, 50]. In general, our network can be regarded as a variant of Wasserstein Auto-Encoder [42] with two kinds of bottlenecks (*i.e.*, latent variables). Hence, we call the proposed method Twin-Bottleneck Hashing (**TBH**). Fig. 1 illustrates the differences between TBH and the related models. As shown in Fig. 1 (c), the binary bottleneck (**BinBN**) contributes to constructing the code-driven similarity graph, while the continuous bottleneck (**ConBN**) mainly guarantees the reconstruction quality. Furthermore, Graph Convolutional Networks (GCNs) [19] are leveraged as a 'tunnel' for the graph and the ConBN to fully exploit data relationships. As a result, similarity-preserving latent representations are fed into the decoder for high-quality reconstruction. Finally, as a reward, the updated network setting is back-propagated through the ConBN to the encoder, which can better fulfill our ultimate goal of binary coding.

More concretely, TBH tackles the '**static graph**' problem by directly leveraging the latent binary codes to adaptively capture the intrinsic data structure. To this end, an adaptive similarity graph is computed directly based on the Hamming distances between binary codes, and is used to guide the ConBN through neural graph convolution [14, 19]. This design provides an optimal mechanism for efficient retrieval tasks by directly incorporating the Hamming distance into training. *On the other hand*, as a side benefit of the twin-bottleneck module, TBH can also overcome another important limitation in generative hashing models [5, 8], *i.e.*, directly inputting the BinBN to the decoder leads to poor data reconstruction capability. For simplicity, we call this problem as '**deficient BinBN**'. Particularly, we address this problem by leveraging the ConBN, which is believed to have higher encoding capacity, for decoding. In this way, one can expect these continuous latent variables to preserve more entropy than binary ones. Consequently, the reconstruction procedure in the generative model becomes more effective.

In addition, during the optimization procedure, existing hashing methods often employ alternating iteration for auxiliary binary variables [34] or even discard the binary constraints using some relaxation techniques [9]. In contrast, our model employs the distributional derivative estimator [8] to compute the gradients across binary latent variables, ensuring that the binary constraints are not violated. Therefore, the whole TBH model can be conveniently optimized by the standard Stochastic Gradient Descent (SGD) algorithm.

The main contributions of this work are summarized as:

- We propose a novel unsupervised hashing framework by incorporating twin bottlenecks into a unified generative network. The binary and continuous bottlenecks work collaboratively to generate discriminative binary codes without much loss of reconstruction capability.
- A code-driven adjacency graph is proposed with efficient computation in the Hamming space. The graph is updated adaptively to better fit the inherent data structure. Moreover, GCNs are leveraged to further exploit the data relationships.
- The auto-encoding framework is novelly leveraged to determine the **reward** of the encoding quality on top of the code-driven graph, shaping the idea of *learning similarities by decoding*.
- Extensive experiments show that the proposed TBH model massively boosts the state-of-the-art retrieval performance on four large-scale image datasets.

2. Related Work

Learning to hash, including the supervised and unsupervised scenario [5, 6, 9, 12, 24, 27, 29, 30], has been studied for years. This work is mostly related to the graph-based approaches [43, 29, 30, 35] and deep generative models based ones [5, 8, 10, 36, 40, 50].

Unsupervised hashing with graphs. As a well-known graph-based approach, Spectral Hashing (SpH) [43] determines pair-wise code distances according to the graph Laplacians of the data similarity affinity in the original feature space. Anchor Graph Hashing (AGH) [30] successfully defines a small set of anchors to approximate this graph. These approaches assume that the original or mid-level data feature distance reflects the actual data relevance. As discussed in the problem of '**static graph**', this is not always realistic. Additionally, the pre-computed graph is isolated from the training process, making it hard to obtain optimal codes. Although this issue has already been considered in [35] by an alternating code updating scheme, its similarity graph is still only dependent on real-valued features during training. We decide to build the batch-wise graph directly upon the Hamming distance so that the learned graph is automatically optimized by the neural network.

Unsupervised generative hashing. Stochastic Generative Hashing (SGH) [8] is closely related to our model in the way of employing the auto-encoding framework and the discrete stochastic neurons. SGH [8] simply utilizes the binary latent variables as the encoder-decoder bottleneck. This design does not fully consider the code similarity and may lead to high reconstruction error, which harms the training effectiveness ('**deficient BinBN**'). Though auto-encoding schemes apply deterministic decoding error, we are also aware that some existing models [10, 40, 50] are proposed with implicit reconstruction likelihood such as the discriminator in Generative Adversarial Network (GAN) [13]. Note that TBH also involves adversarial training, but only for regularization purpose.

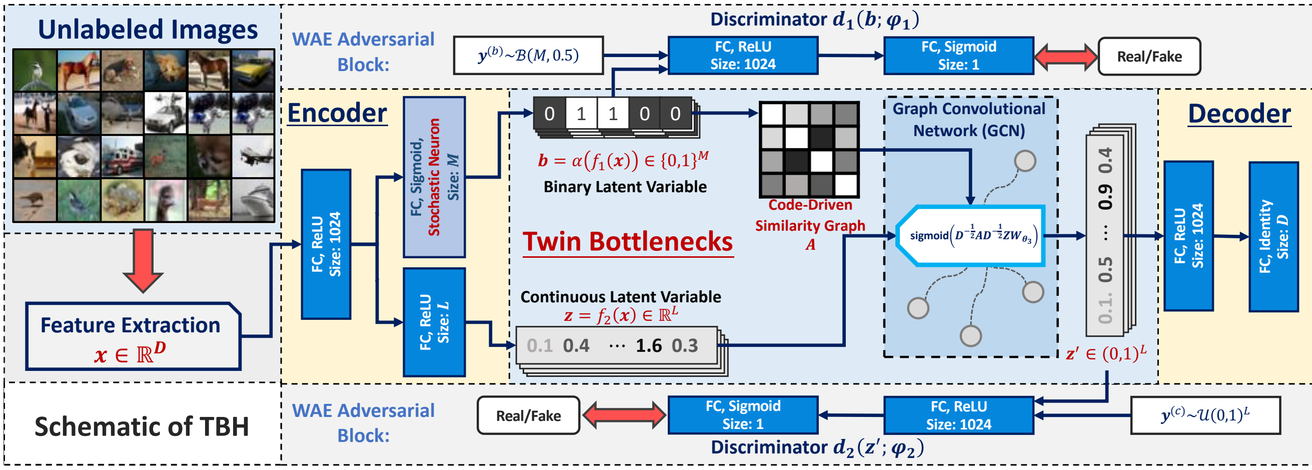

Figure 2. The schematic of TBH. Our model generally shapes an auto-encoding framework with twin bottlenecks (*i.e.*, binary and continuous latent variables). Note that the graph adjacency $\mathbf{A}$ is dynamically built according to the code Hamming distances and is adjusted upon the 'reward' from decoding. Thus, the whole process can be briefed as *optimizing code similarities by decoding*.

3. Proposed Model

TBH produces binary features of the given data for efficient ANN search. Given a data collection $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N \in \mathbb{R}^{N \times D}$, the goal is to learn an encoding function $h : \mathbb{R}^D \to (0, 1)^M$. Here N refers to the set size; D indicates the original data dimensionality and M is the target code length. Traditionally, the code of a data point, *e.g.*, an image or a feature vector, is obtained by applying an element-wise sign function (*i.e.*, $\text{sign}(\cdot)$) to the encoding function:

$$\mathbf{b} = (\text{sign}(h(\mathbf{x}) - 0.5) + 1)/2 \in \{0, 1\}^M, \qquad (1)$$

where $\mathbf{b}$ is the binary code. Some auto-encoding hashing methods [8, 38] introduce stochasticity on the encoding layer (see Eq. (2)) to estimate the gradient across $\mathbf{b}$. We also adopt this paradigm to make TBH fully trainable with SGD, while during test, Eq. (1) is used to encode out-of-sample data points.

3.1. Network Overview

The network structure of TBH is illustrated in Fig. 2. It typically involves a twin-bottleneck auto-encoder for our unsupervised feature learning task and two WAE [42] discriminators for latent variable regularization. The network setting, including the numbers of layers and hidden states, is also provided in Fig. 2.

An arbitrary datum $\mathbf{x}$ is firstly fed to the encoders to produce two sets of latent variables, *i.e.*, the binary code $\mathbf{b}$ and the continuous feature $\mathbf{z}$. Note that the back-propagatable discrete stochastic neurons are introduced to obtain the binary code $\mathbf{b}$. This procedure will be explained in Sec. 3.2.1. Subsequently, a similarity graph within a training batch is built according to the Hamming distances between binary codes. As shown in Fig. 2, we use an adjacency matrix $\mathbf{A}$ to represent this batch-wise similarity graph. The continuous latent variable $\mathbf{z}$ is tweaked by Graph Convolutional Network (GCN) [19] with the adjacency $\mathbf{A}$, resulting in the final latent variable $\mathbf{z}'$ (see Sec. 3.2.2) for reconstruction. Following [42], two discriminators $d_1(\cdot)$ and $d_2(\cdot)$ are utilized to regularize the latent variables, producing informative 0-1 balanced codes.

3.1.1 Why Does This Work?

Our key idea is to utilize the reconstruction loss of the auxiliary decoder side as sort of **reward/critic** to score the encode quality through the GCN layer and encoder. Hence, TBH directly solves both '**static graph**' and '**deficient BinBN**' problems. First of all, the utilization of continuous latent variables mitigates the information loss on the binary bottleneck in [5, 8], as more detailed data information can be kept. This design promotes the reconstruction quality and training effectiveness. Secondly, a direct back-propagation pathway from the decoder to the binary encoder $f_1(\cdot)$ is established through the GCN [19]. The GCN layer selectively mixes and tunes the latent data representation based on code distances so that data with similar binary representations have stronger influences on each other. Therefore, the binary encoder is effectively trained upon successfully detecting relevant data for reconstruction.

3.2. Auto-Encoding Twin-Bottleneck

3.2.1 Encoder: Learning Factorized Representations

Different from conventional auto-encoders, TBH involves a *twin-bottleneck* architecture. Apart from the M-bit binary code $\mathbf{b} \in \{0, 1\}^M$, the continuous latent variable $\mathbf{z} \in \mathbb{R}^L$ is introduced to capture detailed data information. Here L

refers to the dimensionality of $\mathbf{z}$. As shown in Fig. 2, two encoding layers, respectively for $\mathbf{b}$ and $\mathbf{z}$, are topped on the identical fully-connected layer which receives original data representations $\mathbf{x}$. We denote these two encoding functions, *i.e.*, $[\mathbf{b}, \mathbf{z}] = f(\mathbf{x}; \theta)$, as follows:

$$\begin{aligned} \mathbf{b} &= \alpha\left(f_1(\mathbf{x}; \theta_1), \epsilon\right) \in \{0, 1\}^M, \\ \mathbf{z} &= f_2\left(\mathbf{x}; \theta_2\right) \in \mathbb{R}^L, \end{aligned} \tag{2}$$

where θ_1 and θ_2 indicate the network parameters. Note that θ_1 overlaps with θ_2 w.r.t. the weights of the shared fully-connected layer. The first layer of $f_1(\cdot)$ and $f_2(\cdot)$ comes with a ReLU [32] non-linearity. The activation function for the second layer of $f_1(\cdot)$ is the sigmoid function to restrict its output values within an interval of $(0, 1)$, while $f_2(\cdot)$ uses ReLU [32] non-linearity again on the second layer.

More importantly, $\alpha(\cdot, \epsilon)$ in Eq. (2) is the element-wise discrete stochastic neuron activation [8] with a set of random variables $\epsilon \sim \mathcal{U}(0, 1)^M$, which is used for back-propagation through the binary variable $\mathbf{b}$. A discrete stochastic neuron is defined as:

$$\mathbf{b}^{(i)} = \alpha\left(f_1(\mathbf{x}; \theta_1), \epsilon\right)^{(i)} = \begin{cases} 1 & f_1(\mathbf{x}; \theta_1)^{(i)} \geqslant \epsilon^{(i)}, \\ 0 & f_1(\mathbf{x}; \theta_1)^{(i)} < \epsilon^{(i)}, \end{cases} \tag{3}$$

where the superscript (i) denotes the i-th element in the corresponding vector. During the training phase, this operation preserves the binary constraints and allows gradient estimation through distributional derivative [8] with Monte Carlo sampling, which will be elaborated later.

3.2.2 Bottlenecks: Code-Driven Hamming Graph

Different from the existing graph-based hashing approaches [29, 30, 48] where graphs are basically *fixed* during training, TBH automatically detects relevant data points in a graph and mixes their representations for decoding with a back-propagatable scheme.

The outputs of the encoder, *i.e.*, $\mathbf{b}$ and $\mathbf{z}$, are utilized to produce the final input $\mathbf{z}'$ to the decoder. For simplicity, we use batch-wise notations with capitalized letters. In particular, $\mathbf{Z}_B = [\mathbf{z}_1; \mathbf{z}_2; \cdots; \mathbf{z}_{N_B}] \in \mathbb{R}^{N_B \times L}$ and $\mathbf{B}_B = [\mathbf{b}_1; \mathbf{b}_2; \cdots; \mathbf{b}_{N_B}] \in \{0, 1\}^{N_B \times M}$ respectively refer to the continuous and binary latent variables for a batch of N_B data points. The inputs to the decoder are therefore denoted as $\mathbf{Z}'_B = [\mathbf{z}'_1; \mathbf{z}'_2; \cdots; \mathbf{z}'_{N_B}] \in \mathbb{R}^{N_B \times L}$. We construct the graph based on the whole training batch with each datum as a vertex, and the edges are determined by the Hamming distances between the binary codes. The normalized graph adjacency $\mathbf{A} \in [0, 1]^{N_B \times N_B}$ is computed by:

$$\mathbf{A} = \mathbb{J} + \frac{1}{M}\left(\mathbf{B}_B(\mathbf{B}_B - \mathbb{J})^\top + (\mathbf{B}_B - \mathbb{J})\mathbf{B}_B^\top\right), \tag{4}$$

where $\mathbb{J} = \mathbb{1}^\top \mathbb{1}$ is a matrix full of ones. Eq. (4) is an equilibrium of $\mathbf{A}_{ij} = 1 - \mathrm{Hamming}(\mathbf{b}_i, \mathbf{b}_j)/M$ for each entry of $\mathbf{A}$. Then this adjacency, together with the continuous variables $\mathbf{Z}_B$, is processed by the GCN layer [19], which is defined as:

$$\mathbf{Z}'_B = \mathrm{sigmoid}\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{Z}_B\mathbf{W}_{\theta_3}\right). \tag{5}$$

Here $\mathbf{W}_{\theta_3} \in \mathbb{R}^{L \times L}$ is a set of trainable projection parameters and $\mathbf{D} = \mathrm{diag}\left(\mathbf{A}\mathbb{1}^\top\right)$.

As the batch-wise adjacency $\mathbf{A}$ is constructed exactly from the codes, a trainable pathway is then established from the decoder to $\mathbf{B}_B$. Intuitively, the reconstruction penalty scales up when unrelated data are closely located in the Hamming space. Ultimately, only relevant data points with similar binary representations are linked during decoding. Although GCNs [19] are utilized as well in [38, 48], these works generally use pre-computed graphs and cannot handle the '**static graph**' problem.

3.2.3 Decoder: Rewarding the Hashing Results

The decoder is an auxiliary component to the model, determining the code quality produced by the encoder. As shown in Fig. 2, the decoder $g(\cdot)$ of TBH consists of two fully-connected layers, which are topped on the GCN [19] layer. We impose an ReLU [32] non-linearity on the first layer and an identity activation on the second layer. Therefore, the decoding output $\hat{\mathbf{x}}$ is represented as $\hat{\mathbf{x}} = g(\mathbf{z}'; \theta_4) \in \mathbb{R}^D$, where θ_4 refers to the network parameters within the scope of the decoder and $\mathbf{z}'$ is a row vector of $\mathbf{Z}'_B$ generated by the GCN [19]. We elaborate the detailed loss of the decoding side in Sec. 3.4.

To keep the content concise, we do not propose a large convolutional network receiving and generating raw images, since our goal is to learn compact binary features. The decoder provides deterministic reconstruction penalty, *e.g.*, the ℓ_2-norm, back to the encoders during optimization. This ensures a more stable and controllable training procedure than the implicit generation penalties, *e.g.*, the discriminators in GAN-based hashing [10, 40, 50].

3.3. Implicit Bottleneck Regularization

The latent variables in the bottleneck are regularized to avoid wasting bits and align representation distributions. Different from the deterministic regularization terms such as bit de-correlation [9, 27] and entropy-like loss [8], TBH mimics WAE [42] to adversarially regularize the latent variables with auxiliary discriminators. The detailed settings of the discriminators, *i.e.*, $d_1(\cdot; \varphi_1)$ and $d_2(\cdot; \varphi_2)$ with network parameters φ_1 and φ_2, are illustrated in Fig. 2, particularly involving two fully-connected layers successively with ReLU [32] and sigmoid non-linearities.

In order to balance zeros and ones in a binary code, we assume that $\mathbf{b}$ is priored by a binomial distribution $\mathcal{B}(M, 0.5)$, which could maximize the code entropy. Meanwhile, regularization is also applied to the continuous variables $\mathbf{z}'$ after the GCN for decoding. We expect $\mathbf{z}'$ to obey a uniform distribution $\mathcal{U}(0, 1)^L$ to fully explore the latent space. To that end, we employ the following two discriminators d_1 and d_2 for $\mathbf{b}$ and $\mathbf{z}'$, respectively:

$$\begin{aligned} d_1\left(\mathbf{b};\varphi_1\right) \in (0,1);\ d_1(\mathbf{y}^{(b)};\varphi_1) \in (0,1), \\ d_2\left(\mathbf{z}';\varphi_2\right) \in (0,1);\ d_2(\mathbf{y}^{(c)};\varphi_2) \in (0,1), \end{aligned} \tag{6}$$

where $\mathbf{y}^{(b)} \sim \mathcal{B}\left(M, 0.5\right)$ and $\mathbf{y}^{(c)} \sim \mathcal{U}\left(0,1\right)^L$ are random signals sampled from the targeted distributions for implicit regularizing $\mathbf{b}$ and $\mathbf{z}'$ respectively.

The WAE-like regularizers focus on minimizing the distributional discrepancy between the produced feature space and the target one. This design fits TBH better than deterministic regularizers [8, 9], since such kinds of regularizers (*e.g.*, bit de-correlation) impose direct penalties on each sample, which may heavily skew the similarity graph built upon the codes and consequently degrades the training quality. Experiments also support our insight (see Table 3).

3.4. Learning Codes and Similarity by Decoding

As TBH involves adversarial training, two learning objectives, *i.e.*, $\mathcal{L}_{AE}$ for the *auto-encoding* step and $\mathcal{L}_D$ for the *discriminating* step, are respectively introduced.

3.4.1 *Auto-Encoding* Objective

The *Auto-Encoding* objective $\mathcal{L}_{AE}$ is written as follows:

$$\begin{aligned} \mathcal{L}_{AE} = \frac{1}{N_B}\sum_{i=1}^{N_B}\mathbb{E}_{\mathbf{b_i}}\Big[\frac{1}{2M}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \\ - \lambda \log d_1(\mathbf{b}_i;\varphi_1) - \lambda \log d_2(\mathbf{z}'_i;\varphi_2)\Big], \end{aligned} \tag{7}$$

where λ is a hyper-parameter controlling the penalty of the discriminator according to [42]. $\mathbf{b}$ is obtained from Eq. (3), $\mathbf{z}'$ is computed according to Eq. (5), and the decoding result $\hat{\mathbf{x}} = g\left(\mathbf{z}';\theta_4\right)$ is obtained from the decoder. $\mathcal{L}_{AE}$ is used to optimize the network parameters within the scope of $\theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$. Eq. (7) comes with an expectation term $\mathbb{E}[\cdot]$ over the latent binary code, since $\mathbf{b}$ is generated by a sampling process.

Inspired by [8], we estimate the gradient through the binary bottleneck with distributional derivatives by utilizing a set of random signals $\epsilon \sim \mathcal{U}\left(0,1\right)^M$. The gradient of $\mathcal{L}_{AE}$ w.r.t. θ is estimated by:

$$\begin{aligned} \nabla_\theta\mathcal{L}_{AE} \approx \frac{1}{N_B}\sum_{i=1}^{N_B}\mathbb{E}_\epsilon\Big[\nabla_\theta\big(\frac{1}{2M}\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \\ - \lambda \log d_1(\mathbf{b}_i;\varphi_1) - \lambda \log d_2(\mathbf{z}'_i;\varphi_2)\big)\Big]. \end{aligned} \tag{8}$$

We refer the reader to [8] and our **Supplementary Material** for the details of Eq. (8). Notably, a similar approach of sampling-based gradient estimation for discrete variables was employed in [2], which has been proved as a special case of the REINFORCE algorithm [44].

3.4.2 *Discriminating* Objective

The *Discriminating* objective $\mathcal{L}_D$ is defined by:

Algorithm 1: The Training Procedure of TBH

Input: Dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ and the maxinum number of iterations T.
Output: Network parameters θ_1.
repeat
    Randomly select a mini-batch $\{\mathbf{x}_i\}_{i=1}^{N_B}$ from $\mathbf{X}$
    for $i = 1, \cdots, N_B$ **do**
        Sample $\epsilon_i \sim \mathcal{U}\left(0,1\right)^M$
        Sample $\mathbf{y}_i^{(b)} \sim \mathcal{B}\left(M, 0.5\right); \mathbf{y}_i^{(c)} \sim \mathcal{U}\left(0,1\right)^L$
        Sample a set of $\mathbf{b}_i$ according to Eq. (3) using ϵ_i
    end
    ***Discriminating* step:**
        $\mathcal{L}_D \leftarrow$ Eq. (9)
        $\varphi \leftarrow \varphi - \mathbf{\Gamma}\left(\nabla_\varphi\mathcal{L}_D\right)$
    ***Auto-encoding* step:**
        $\mathcal{L}_{AE} \leftarrow$ Eq. (7)
        $\theta \leftarrow \theta - \mathbf{\Gamma}\left(\nabla_\theta\mathcal{L}_{AE}\right)$ according to Eq. (8)
until *convergence or reaching the maximum iteration* T;

$$\begin{aligned} \mathcal{L}_D = -\frac{\lambda}{N_B}\sum_{i=1}^{N_B}\Big(\log d_1(\mathbf{y}_i^{(b)};\varphi_1) \\ + \log d_2(\mathbf{y}_i^{(c)};\varphi_2) + \log\big(1 - d_1(\mathbf{b}_i;\varphi_1)\big) \\ + \log\big(1 - d_2(\mathbf{z}'_i;\varphi_2)\big)\Big). \end{aligned} \tag{9}$$

Here λ refers to the same hyper-parameter as in Eq. (7). Similarly, $\mathcal{L}_D$ optimizes the network parameters within the scope of $\varphi = \{\varphi_1, \varphi_2\}$. As the *discriminating* step does not propagate error back to the auto-encoder, there is no need to estimate the gradient through the indifferentiable binary bottleneck. Thus the expectation term $\mathbb{E}[\cdot]$ in Eq. (7) is deprecated in Eq. (9).

The training procedure of TBH is summarized in Algorithm 1, where $\mathbf{\Gamma}\left(\cdot\right)$ refers to the adaptive gradient scaler, for which we adopt the Adam optimizer [18]. Monte Carlo sampling is performed on the binary bottleneck, once a data batch is fed to the encoder. Therefore, the learning objective can be computed using the network outputs.

3.5. Out-of-Sample Extension

After TBH is trained, we can obtain the binary codes for any out-of-sample data as follows:

$$\mathbf{b}^{(q)} = (\mathrm{sign}(f_1(\mathbf{x}^{(q)};\theta_1) - 0.5) + 1)/2 \in \{0,1\}^M, \tag{10}$$

where $\mathbf{x}^{(q)}$ denotes a query data point. During the test phase, only $f_1\left(\cdot\right)$ is required, which considerably eases the binary coding process. Since only the forward propagation process is involved for test data, the stochasticity on the encoder $f_1\left(\cdot\right)$ used for training in Eq. (2) is not needed.

4. Experiments

We evaluate the performance of the proposed TBH on four large-scale image benchmarks, *i.e.*, CIFAR-10, NUS-WIDE, MS COCO. We additionally present results for image reconstruction on the MNIST dataset.

Table 1. Performance comparison (w.r.t. MAP) of TBH and the state-of-the-art **unsupervised** hashing methods.

		CIFAR-10			**NUS-WIDE**			**MSCOCO**		
Method	**Reference**	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH [6]	STOC02	0.106	0.102	0.105	0.239	0.266	0.266	0.353	0.372	0.341
SpH [43]	NIPS09	0.272	0.285	0.300	0.517	0.511	0.510	0.527	0.529	0.546
AGH [30]	ICML11	0.333	0.357	0.358	0.592	0.615	0.616	0.596	0.625	0.631
SpherH [17]	CVPR12	0.254	0.291	0.333	0.495	0.558	0.582	0.516	0.547	0.589
KMH [15]	CVPR13	0.279	0.296	0.334	0.562	0.597	0.600	0.543	0.554	0.592
ITQ [12]	PAMI13	0.305	0.325	0.349	0.627	0.645	0.664	0.598	0.624	0.648
DGH [29]	NIPS14	0.335	0.353	0.361	0.572	0.607	0.627	0.613	0.631	0.638
DeepBit [27]	CVPR16	0.194	0.249	0.277	0.392	0.403	0.429	0.407	0.419	0.430
SGH [8]	ICML17	0.435	0.437	0.433	0.593	0.590	0.607	0.594	0.610	0.618
BGAN [40]	AAAI18	0.525	0.531	0.562	0.684	0.714	0.730	0.645	0.682	0.707
BinGAN [50]	NIPS18	0.476	0.512	0.520	0.654	0.709	0.713	0.651	0.673	0.696
GreedyHash [41]	NIPS18	0.448	0.473	0.501	0.633	0.691	0.731	0.582	0.668	0.710
HashGAN [10]*	CVPR18	0.447	0.463	0.481	-	-	-	-	-	-
DVB [37]	IJCV19	0.403	0.422	0.446	0.604	0.632	0.665	0.570	0.629	0.623
DistillHash [46]	CVPR19	0.284	0.285	0.288	0.667	0.675	0.677	-	-	-
TBH	Proposed	**0.532**	**0.573**	**0.578**	**0.717**	**0.725**	**0.735**	**0.706**	**0.735**	**0.722**

*Note the duplicate naming of HashGAN, *i.e.*, the unsupervised one [10] and the supervised one [3].

4.1. Implementation Details

The proposed TBH model is implemented with the popular deep learning toolbox Tensorflow [1]. The hidden layer sizes and the activation functions used in TBH are all provided in Fig. 2. The gradient estimation of Eq. (8) can be implemented with a single Tensorflow decorator in Python, following [8]. TBH only involves **two** hyper-parameters, *i.e.*, λ and L. We set $\lambda = 1$ and $L = 512$ by default. For all of our experiments, the $\texttt{fc\_7}$ features of the AlexNet [22] network are utilized for data representation. The learning rate of Adam optimizer $\mathbf{\Gamma}(\cdot)$ [18] is set to 1×10^{-4}, with default decay rates $\texttt{beta}_1 = 0.9$ and $\texttt{beta}_2 = 0.999$. We fix the training batch size to 400. Our implementation can be found at https://github.com/ymcidence/TBH.

4.2. Datasets and Setup

CIFAR-10 [21] consists of 60,000 images from 10 classes. We follow the common setting [10, 41] and select 10,000 images (1000 per class) as the query set. The remaining 50,000 images are regarded as the database.

NUS-WIDE [7] is a collection of nearly 270,000 images of 81 categories. Following the settings in [45], we adopt the subset of images from 21 most frequent categories. 100 images of each class are utilized as a query set and the remaining images form the database. For training, we employ 10,500 images uniformly selected from the 21 classes.

MS COCO [28] is a benchmark for multiple tasks. We adopt the pruned set as with [4] with 12,2218 images from 80 categories. We randomly select 5,000 images as queries with the remaining ones the database, from which 10,000 images are chosen for training.

Standard metrics [4, 41] are adopted to evaluate our method and other state-of-the-art methods, *i.e.*, Mean Average Precision (MAP), Precision-Recall (P-R) curves, Precision curves within Hamming radius 2 (P@H$\leq$2), and Precision w.r.t. 1,000 top returned samples (P@1000). We adopt MAP@1000 for CIFAR-10, and MAP@5000 for MS COCO and NUS-WIDE according to [4, 49].

Table 2. P@1000 results of TBH and compared methods on CIFAR-10 and MSCOCO.

	CIFAR-10			**MSCOCO**		
Method	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
KMH	0.242	0.252	0.284	0.557	0.572	0.612
SpherH	0.228	0.256	0.291	0.525	0.571	0.612
ITQ	0.276	0.292	0.309	0.607	0.637	0.662
SpH	0.238	0.239	0.245	0.541	0.548	0.567
AGH	0.306	0.321	0.317	0.602	0.635	0.644
DGH	0.315	0.323	0.324	0.623	0.642	0.650
HashGAN	0.418	0.436	0.455	-	-	-
SGH	0.387	0.380	0.367	0.604	0.615	0.637
GreedyHash	0.322	0.403	0.444	0.603	0.624	0.675
TBH (Ours)	**0.497**	**0.524**	**0.529**	**0.646**	**0.698**	**0.701**

4.3. Comparison with Existing Methods

We compare TBH with several state-of-the-art unsupervised hashing methods, including **LSH** [6], **ITQ** [12], **SpH** [43], **SpherH** [17], **KMH** [15], **AGH** [30], **DGH** [29], **DeepBit** [27], **BGAN** [40], **HashGAN** [10], **SGH [8]**, **BinGAN** [50], **GreedyHash** [41], **DVB** [37] and **DistillHash** [46]. For fair comparisons, all the methods are reported with identical training and test sets. Additionally, the shallow methods are evaluated with the same deep features as the ones we are using.

4.3.1 Retrieval results

The MAP and P@1000 results of TBH and other methods are respectively provided in Tables 1 and 2, while the

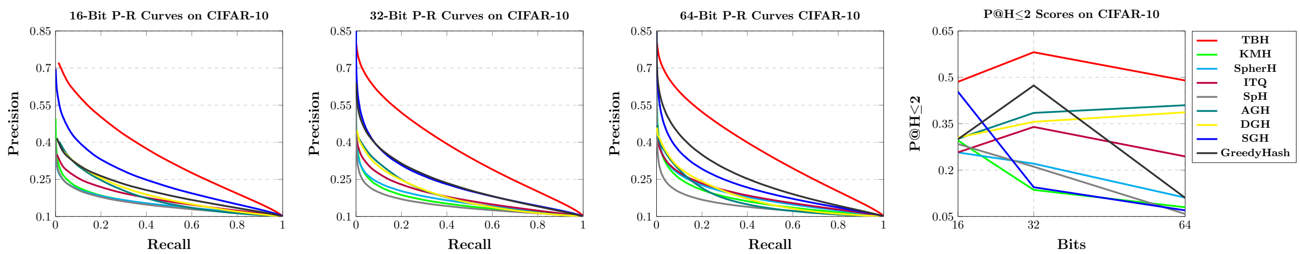

Figure 3. P-R curves and P@H≤2 results of TBH and compared methods on CIFAR-10.

Table 3. Ablation study of MAP@1000 results by using different variants of the proposed TBH on CIFAR-10.

	Baseline	16 bits	32 bits	64 bits
1	Single bottleneck	0.435	0.437	0.433
2	Swapped bottlenecks	0.466	0.471	0.475
3	Explicit regularization	0.524	0.559	0.560
4	Without regularization	0.521	0.535	0.547
5	Without stochastic neuron	0.408	0.412	0.463
6	Fixed graph	0.442	0.464	0.459
7	Attention equilibrium	0.477	0.503	0.519
	TBH (full model)	**0.532**	**0.573**	**0.578**

respective P-R curves and P@H≤2 results are illustrated in Fig. 3. The performance gap between TBH and existing unsupervised methods can be clearly observed. Particularly, TBH obtains remarkable MAP gain with 16-bit codes (*i.e.*, $M = 16$). Among the unsupervised baselines, GreedyHash [41] performs closely next to TBH. It bases the produced code similarity on pair-wise feature distances. As is discussed in Sec 1, this design is straightforward but sub-optimal since the original feature space is not fully revealing data relevance. On the other hand, as a generative model, HashGAN [10] significantly underperforms TBH, as the binary constraints are violated during its adversarial training. TBH differs SGH [8] by leveraging the twin-bottleneck scheme. Since SGH [8] only considers the reconstruction error and in auto-encoder, it generally does not produce convincing retrieval results.

4.3.2 Extremely short codes

Inspired by [41], we illustrate the retrieval performance with extremely short bit length in Fig. 4 (a). TBH works well even when the code length is set to $M = 4$. The significant performance gain over SGH can be observed. This is due to that, during training, the continuous bottleneck complements the information discarded by the binary one.

4.4. Ablation Study

In this subsection, we validate the contribution of each component of TBH, and also show some empirical analysis. Different baseline network structures are visualized in the **Supplementary Material** for better understanding.

4.4.1 Component Analysis

We compare TBH with the following baselines. **(1) Single bottleneck.** This baseline coheres with SGH. We remove the twin-bottleneck structure and directly feed the binary codes to the decoder. **(2) Swapped bottlenecks.** We swap the functionalities of the two bottlenecks, *i.e.*, using the continuous one for adjacency building and the binary one for decoding. **(3) Explicit regularization.** The WAE regularizers are replaced by conventional regularization terms. An entropy loss similar to SGH is used to regularize $\mathbf{b}$, while an ℓ_2-norm is applied to $\mathbf{z}'$. **(4) Without regularization.** The regularization terms for $\mathbf{b}$ and $\mathbf{z}'$ are removed in this baseline. **(5) Without stochastic neuron.** The discrete stochastic neuron $\alpha(\cdot, \epsilon)$ is deprecated on the top of $f_1(\cdot)$, and bit quantization loss [9] is appended to $\mathcal{L}_{AE}$. **(6) Fixed graph.** $\mathbf{A}$ is pre-computed using feature distances. The continuous bottleneck is removed and the GCN is applied to the binary bottleneck with the fixed $\mathbf{A}$. **(7) Attention equilibrium.** This baseline performs weighted average on $\mathbf{Z}$ according to $\mathbf{A}$, instead of employing GCN in-between.

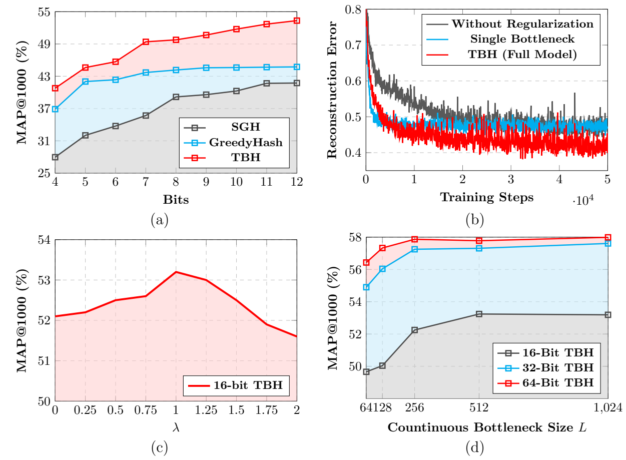

Figure 4. **(a)** MAP@1000 results with extremely short code lengths on CIFAR-10. **(b)** 16-bit normalized reconstruction errors of TBH and its variants on CIFAR-10. **(c)** and **(d)** Effects of the hyper-parameters λ and L on CIFAR-10.

Table 3 shows the performance of the baselines. We can observe that the model undergoes significant performance drop when modifying the twin-bottleneck structure. Specifically, our trainable adaptive Hamming graph plays an important role in the network. When removing this (*i.e.*, `baseline_6`), the performance decreases by ~9%. This accords with our motivation in dealing with the '**static**

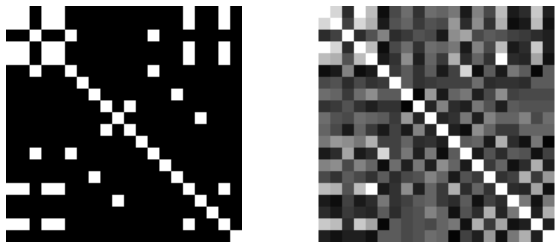

Figure 5. Adjacency matrices of 20 randomly-sampled data points of a training batch on CIFAR-10, computed based on ground-truth similarity (**Left**) and the Hamming distances between binary codes using Eq. (4) (**Right**).

graph' problem.In practice, we also experience training perturbations when applying different regularization and quantization penalties to the model.

4.4.2 Reconstruction Error

As mentioned in the '**deficient BinBN**' problem, decoding from a single binary bottleneck is less effective. This is illustrated in Fig. 4 (b), where the normalized reconstruction errors of TBH, `baseline_1` and `baseline_4` are plotted. TBH produces lower decoding error than the single bottleneck baseline. Note that `baseline_1` structurally coheres SGH [8].

4.4.3 Hyper-Parameter

Only two hyper-parameters are involved in TBH. The effect of the adversarial loss scaler λ is illustrated in Fig. 4 (c). A large regularization penalty slightly influences the overall retrieval performance. The results w.r.t. different settings of L on CIFAR-10 are shown in Fig. 4 (d). Typically, no dramatic performance drop is observed when squeezing the bottleneck, as data relevance is not only reflected by the continuous bottleneck. Even when setting L to 64, TBH still outperforms most existing unsupervised methods, which also endorses our twin-bottleneck mechanism.

4.5. Qualitative Results

We provide some intuitive results to further justify the design. The implementation details are given in the **Supplementary Material** to keep the content concise.

4.5.1 The Constructed Graph by Hash Codes

We show the effectiveness of the code-driven graph learning process in Fig. 5. 20 random samples are selected from a training batch to plot the adjacency. The twin-bottleneck mechanism automatically tunes the codes, constructing $\mathbf{A}$ based on Eq. (4). Though TBH has no access to the labels, the constructed adjacency simulates the label-based one. Here brighter color indicates closer Hamming distances.

4.5.2 Visualization

Fig. 6 (a) shows the t-SNE [31] visualization results of TBH. Most of the data from different categories are clearly scattered. Interestingly, TBH successfully locates visually similar categories within short Hamming distances, *e.g.*, *Automobile*/*Truck* and *Deer*/*Horse*. Some qualitative image retrieval results w.r.t. 32-bit codes are shown in Fig. 6 (b).

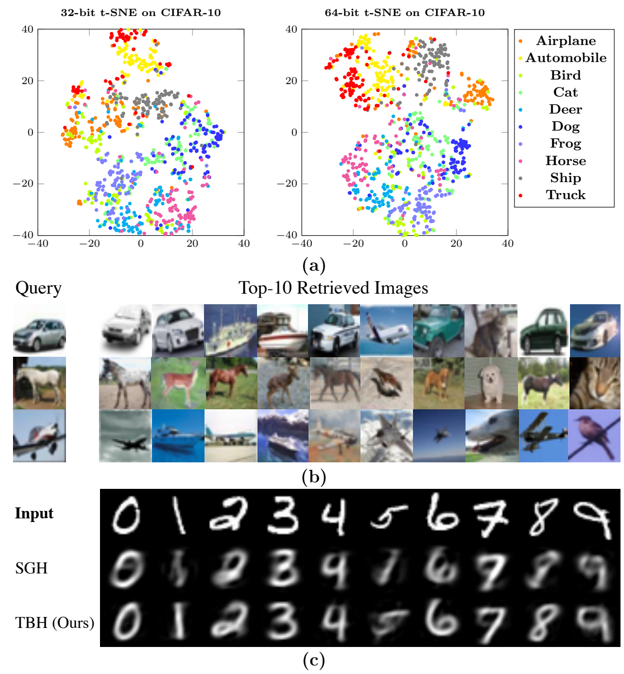

Figure 6. **(a)** t-SNE visualization on CIFAR-10. **(b** 32-bit retrieval results on CIFAR-10. **(c)** 64-bit image reconstruction results of TBH and SGH on MNIST.

4.5.3 Toy Experiment on MNIST

Following [8], another toy experiment on image reconstruction with the MNIST [26] dataset is conducted. For this task, we directly use the flattened image pixels as the network input. The reconstruction results are reported in Fig. 6 (c). Some bad handwritings are falsely decoded to wrong numbers by SGH [8], while this phenomenon is not frequently observed in TBH. This also supports our insight in addressing the '**deficient BinBN**' problem.

5. Conclusion

In this paper, a novel unsupervised hashing model was proposed with an auto-encoding twin-bottleneck mechanism, namely Twin-Bottleneck Hashing (TBH). The binary bottleneck explored the intrinsic data structure by adaptively constructing the code-driven similarity graph. The continuous bottleneck interactively adopted data relevance information from the binary codes for high-quality decoding and reconstruction. The proposed TBH model was fully trainable with SGD and required no empirical assumption on data similarity. Extensive experiments revealed that TBH remarkably boosted the state-of-the-art unsupervised hashing schemes in image retrieval.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467*, 2016. 6

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 5

[3] Yue Cao, Bin Liu, Mingsheng Long, and Jianmin Wang. Hashgan: Deep learning to hash with pair conditional wasserstein gan. In *CVPR*, 2018. 6

[4] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Hashnet: Deep learning to hash by continuation. In *ICCV*, 2017. 6

[5] Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. Hashing with binary autoencoders. In *CVPR*, 2015. 2, 3

[6] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002. 2, 6

[7] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *CIVR*, 2009. 6

[8] Bo Dai, Ruiqi Guo, Sanjiv Kumar, Niao He, and Le Song. Stochastic generative hashing. In *ICML*, 2017. 2, 3, 4, 5, 6, 7, 8

[9] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *CVPR*, 2015. 2, 4, 5, 7

[10] Kamran Ghasedi Dizaji, Feng Zheng, Najmeh Sadoughi, Yanhua Yang, Cheng Deng, and Heng Huang. Unsupervised deep generative adversarial hashing network. In *CVPR*, 2018. 2, 4, 6, 7

[11] 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. 1

[12] Y. Gong, S. Lazebnik, A. Gordo, and F. 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. 1, 2, 6

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014. 2

[14] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011. 2

[15] Kaiming He, Fang Wen, and Jian Sun. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *CVPR*, 2013. 1, 6

[16] Xiangyu He, Peisong Wang, and Jian Cheng. K-nearest neighbors hashing. In *CVPR*, 2019. 1

[17] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *CVPR*, 2012. 6

[18] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5, 6

[19] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. 2, 3, 4

[20] Weihao Kong and Wu-Jun Li. Isotropic hashing. In *NeurIPS*, 2012. 1

[21] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2009. 6

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 6

[23] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *NeurIPS*, 2009. 1

[24] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009. 2

[25] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, 2015. 1

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. 8

[27] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *CVPR*, 2016. 2, 4, 6

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 6

[29] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *NeurIPS*, 2014. 1, 2, 4, 6

[30] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *ICML*, 2011. 1, 2, 4, 6

[31] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. 8

[32] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010. 4

[33] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7), 2009. 1

[34] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *CVPR*, 2015. 2

[35] Fumin Shen, Yan Xu, Li Liu, Yang Yang, Zi Huang, and Heng Tao Shen. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):3034–3044, 2018. 1, 2

[36] Yuming Shen, li Liu, and Ling Shao. Unsupervised deep generative hashing. In *BMVC*, 2017. 2

[37] Yuming Shen, Li Liu, and Ling Shao. Unsupervised binary representation learning with deep variational networks. *International Journal of Computer Vision*, 127(11-12):1614–1628, 2019. 6

[38] Yuming Shen, Li Liu, Fumin Shen, and Ling Shao. Zero-shot sketch-image hashing. In *CVPR*, 2018. 3, 4

[39] Yuming Shen, Jie Qin, Jiaxin Chen, Li Liu, Fan Zhu, and Ziyi Shen. Embarrassingly simple binary representation learning. In *ICCV Workshops*, 2019. 1

[40] Jingkuan Song, Tao He, Lianli Gao, Xing Xu, Alan Hanjalic, and Heng Tao Shen. Binary generative adversarial networks for image retrieval. In *AAAI*, 2018. 2, 4, 6

[41] Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *NeurIPS*, 2018. 1, 6, 7

[42] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. In *ICLR*, 2018. 2, 3, 4, 5

[43] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NeurIPS*, 2009. 1, 2, 6

[44] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 5

[45] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2014. 1, 6

[46] Erkun Yang, Tongliang Liu, Cheng Deng, Wei Liu, and Dacheng Tao. Distillhash: Unsupervised deep hashing by distilling data pairs. In *CVPR*, 2019. 1, 6

[47] Mengyang Yu, Li Liu, and Ling Shao. Structure-preserving binary representations for rgb-d action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(8):1651–1664, 2016. 1

[48] X. Zhou, F. Shen, L. Liu, W. Liu, L. Nie, Y. Yang, and H. T. Shen. Graph convolutional network hashing. *IEEE Transactions on Cybernetics*, pages 1–13, 2018. 1, 4

[49] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *AAAI*, 2016. 6

[50] Maciej Zieba, Piotr Semberecki, Tarek El-Gaaly, and Tomasz Trzcinski. Bingan: Learning compact binary descriptors with a regularized gan. In *NeurIPS*, 2018. 2, 4, 6