Embarrassingly Simple Binary Representation Learning

Yuming Shen , Jie Qin , Jiaxin Chen , Li Liu , Fan Zhu , and Ziyi Shen

Inception Institute of Artificial Intelligence (IIAI), Abu Dhabi, UAE

{ymcidence, qinjiebuaa, chenjiaxinx, liuli1213, fanzhu1987, joanshen0508}@gmail.com

Abstract

Recent binary representation learning models usually require sophisticated binary optimization, similarity measure or even generative models as auxiliaries. However, one may wonder whether these non-trivial components are needed to formulate practical and effective hashing models.

In this paper, we answer the above question by proposing an embarrassingly simple approach to binary representation learning. With a simple classification objective, our model only incorporates two additional fully-connected layers onto the top of an arbitrary backbone network,whilst complying with the binary constraints during training. The proposed model lower-bounds the Information Bottleneck (IB) between data samples and their semantics, and can be related to many recent 'learning to hash' paradigms. We show that, when properly designed, even such a simple network can generate effective binary codes, by fully exploring data semantics without any held-out alternating updating steps or auxiliary models. Experiments are conducted on conventional large-scale benchmarks, i.e., CIFAR-10, NUS-WIDE, and ImageNet, where the proposed simple model outperforms the state-of-the-art methods. Our codes are available at https://github.com/ymcidence/JMLH.

1. Introduction

Approximate nearest neighbour search with binary representations has been regarded as an effective and efficient solution to large-scale multimedia data retrieval. Conventionally termed as *learning to hash*, this family of techniques aims at **(a)** shrinking the embedding size of data and **(b)** producing binary features to speedup the computation of distance-based pair-wise data relevance. Similar to many other machine learning tasks, learning to hash can be either unsupervised or supervised. The former requires less labeling efforts for training, while the later obtains better performance in retrieval. We focus on supervised hashing

to fully leverage the semantic information of data.

Recent research in this field largely boosts the performance of the produced hash codes by introducing deep learning techniques. Deep hashing models typically employ an indifferentiable `sign` activation to the top of the encoding network. Various methods have been proposed to empower the encoder with the ability to properly locate data in the Hamming space.

A typical approach is to employ a held-out code learner as the network training complementary [13, 34, 45]. The code learner performs discrete optimization and alternately updates the semantic-based target codes to govern the behavior of the encoding network. This approach generally requires longer training time since the held-out discrete optimization step cannot be effectively paralleled, and consumes additional memory to cache the target codes during each round of update. Alternatively, some propose to decouple unrelated data representations by introducing similarity-based penalties to the encoders [7, 47, 48, 49]. To train an encoder with these regularizers, one may resort to continuous relaxation on the codes, which arguably degrades the training quality. One recent fashion in deep hashing is to employ generative adversarial models [5, 15, 29, 39, 50]. By distinguishing synthesized data from real ones, the encoder implicitly acknowledges the respective data distribution.

However, the above precisely-proposed approaches raise another question: *How to build an effective supervised hashing model with **minimum** auxiliary components?*

We attempt to find the answer by carefully considering the following main challenges of learning to hash:

- **Keeping the discrete nature of binary codes.** The binary constraints usually lead to an NP-hard optimization problem in parameterized models, and cannot be directly solved by gradient-based methods. This is usually addressed by conventional methods using held-out discrete optimization or relaxation techniques.

- **Enriching the information carried by the codes.** It

is always essential to make the encoder aware of the semantic information (*e.g.*, lables or tags) of data.

As a result, in this paper, we propose a simple but powerful deep hashing network. In our model, the above problems are tackled by relating data and their semantics with a binary representation bottleneck, which is thereafter used as the final hash codes. A single recognition penalty is applied for training. With a reasonable regularization term, the final learning objective forms a variational lower bound of the Information Bottleneck (IB) [2, 41] between observed data and their semantics. Importantly, one can impose stochasticity on the binary bottleneck to keep the binary constraints and apply gradient estimation methods during training. Therefore, the whole framework can be optimized end-to-end with Stochastic Gradient Descent (SGD). To this end, we find our design leads to an embarrassingly simple solution, which basically shapes a single classification neural network .

Regardless of the regularization, the proposed model just maximizes the label likelihood of data. Thus, we name our model Just-Maximizing-Likelihood Hashing (JMLH). The contributions of this paper are summarized as follows:

- We propose a simple and novel deep hashing model, *i.e.*, JMLH, and theoretically base it on the Variational Information Bottleneck (VIB) [2] method. To the best of our knowledge, JMLH is the first attempt in deep hashing to employ the IB methods.

- We show that, when properly designed and trained, a classification neural network with a discrete bottleneck already produces effective binary representations. Therefore, the proposed model requires no auxiliary components and can be optimized directly.

- Relations between JMLH and many existing hashing models are discussed in detail.

- JMLH successfully outperforms state-of-the-art hashing techniques on several benchmark datasets, *i.e.*, CIFAR-10 [22], NUS-WIDE [11] and ImageNet [33].

In the rest of this paper, we first describe our model in detail in Section 2. Subsequently, the relationships between JMLH and existing works are elaborated in Section 3. Section 4 presents the implementation details and experimental findings, with a brief conclusion given in Section 5.

2. Model

The goal of learning to hash is to find an optimal encoding function $f : X \rightarrow \{0,1\}^m$ to represent data. Here X is the variable space of data observation and m refers to the length of the hash code space B. In the context of supervised hashing, training is usually supported by the data
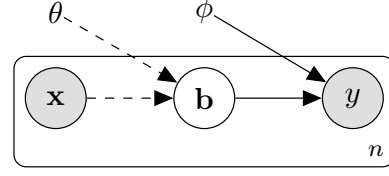

Figure 1. The directed graphical model of JMLH. We treat the hash code **b** as the latent bottleneck between data **x** and their labels y. The dotted lines define the stochastic encoding procedure of $q(B|X)$, and the solid lines denote the approximated likelihood $q(Y|B)$. n is the total number of observed data points. Note that the respective parameters θ and ϕ are jointly learned, forming an extremely simple training model.

labels Y. We intendedly use capitalized notations, *i.e.*, X, Y and B, for the (random) variable spaces, and denote each respective variable instances with lower-cased letters, *i.e.*, **x**, y and **b**.

2.1. JMLH at a Glance

JMLH involves a stochastic encoder $q(B|X)$ and a classifier $q(Y|B)$. An additional deterministic distribution $p(B)$ is used as the prior of B.[1] This model is illustrated in Figure 1 as a directed graphical model. Particularly, each datum $\mathbf{x} \in X$ is firstly associated with a latent binary code $\mathbf{b} \in B$ according to $q(B|X)$, and then the respective label $y \in Y$ can be predicted by feeding $q(Y|B)$ with **b**. Therefore, B can be regarded as the bottleneck between X and Y. Successively applying $q(B|X)$ and $q(Y|B)$ according to the above procedure specifies **a single-task neural network** with a binary layer in between, which makes JMLH extremely simple.

We firstly describe the above-mentioned probabilistic models and then discuss how they are combined as a whole for efficient end-to-end training.

2.1.1 Parameterizing the Probability Models

Given a training pair of $(\mathbf{x}, y)$, the corresponding probabilities models of $q(\mathbf{b}|\mathbf{x})$ and $q(y|\mathbf{b})$ in JMLH are defined as

$$q(\mathbf{b}|\mathbf{x}) = \mathcal{P}(\mathbf{b}|\kappa(\mathbf{x}; \theta)),$$
$$q(y|\mathbf{b}) = Cat(y|\pi(\mathbf{b}; \phi)) \text{ or } \mathcal{P}(y|\pi(\mathbf{b}; \phi)), \quad (1)$$
$$p(\mathbf{b}) = \mathcal{B}(\mathbf{b}|m, 0.5).$$

Here $\mathcal{P}(\mathbf{b}|\kappa(\mathbf{x}; \theta))$ indicates the Poisson binomial distribution, parameterized by a neural network $\kappa(\mathbf{x}; \theta)$ as follows:

$$\mathcal{P}(\mathbf{b}|\kappa(\mathbf{x}; \theta)) = \prod_{i=1}^{m} \kappa_i^{\mathbf{b}_i} (1 - \kappa_i)^{1-\mathbf{b}_i}. \quad (2)$$

---

[1]Here we use $q(\cdot)$ to denote an approximated posterior when one cannot directly model the corresponding true distribution, *e.g.*, $q(B|X)$. On the other hand, $p(\cdot)$ is used when the distribution can be deterministically defined or computed, *e.g.*, the pre-defined prior $p(B)$.

Table 1. Network settings of JMLH. All layers are sequentially applied.

Notation	Specification	Variable
Input	Arbitrary data, 256×256 images in our experiments	X
$\kappa(\mathbf{x};\theta)$	Arbitrary network backbone, Alexnet [23] before `fc_7` in our experiments	
	Fully-connected, size of m Binary stochastic activation	B
$\pi(\mathbf{b};\phi)$	Fully-connected, size of label length `softmax` (single-label datasets) `sigmoid` (multi-label datasets)	Y

On the other hand, $p(y|\mathbf{b})$ can be either categorical for single-label classification, *i.e.*, $Cat(y|\pi(\mathbf{b};\phi))$, or Poisson binomial for multi-label classification, *i.e.*, $\mathcal{P}(y|\pi(\mathbf{b};\phi))$, implemented by another network $\pi(\mathbf{b};\phi)$. We additionally introduce $p(\mathbf{b})$ of a binomial distribution $\mathcal{B}(\mathbf{b}|m,0.5)$ as the code prior for regularization purpose.

Note that we choose discrete probability models for B to avoid the use of continuous relaxation. That is to say, the input to the classifier $\pi(\cdot)$ is already binarized. Continuous relaxation, *e.g.*, activating the neurons with a `sigmoid` non-linearity, is not considered here as it skews the observation of the classifier, propagating biased semantic information measurement back to the encoder.

2.1.2 Shaping a Single Network

Sequentially stacking $\kappa(\mathbf{x};\theta)$ and $\pi(\mathbf{b};\phi)$ empirically forms a classification neural network with a binary bottleneck B, of which the briefed structure is illustrated in Table 1. It can be seen that JMLH only introduces two additional layers on the top of an arbitrary network backbone, which makes it easy to be adopted to different pre-trained models and is convenient for implementation.

Then we define the learning objective with n given training pairs $\{(\mathbf{x},y)\}^n$ of this single network as

$$\mathcal{L} = \frac{1}{n}\sum_{(\mathbf{x},y)}\underbrace{\mathbb{E}_{q(\mathbf{b}|\mathbf{x})}[-\log q(y|\mathbf{b})]}_{\text{classification objective}} + \underbrace{\lambda\,\mathrm{KL}\,(q(\mathbf{b}|\mathbf{x})||p(\mathbf{b}))}_{\text{regularization}},$$
(3)

where λ is a hyper-parameter. All the probability models are defined in Eq. (1). We first elaborate each component of it in this subsection and later show that this learning objective is supported by VIB [2] in Section 2.2.1.

The first Right-Hand-Side (RHS) term of Eq. (3), *i.e.* $-\log q(y|\mathbf{b})$, is actually a negative log-likelihood classification penalty since $q(y|\mathbf{b})$ is categorical. This loss conveys semantic label information of data to their codes during training.

Algorithm 1: The Training Procedure of JMLH

Input: Data observations X, the corresponding labels Y and the maximum number of iterations T.
Output: Network parameters θ.
repeat
    Randomly pick a batch of $\{(\mathbf{x},y)\}$ from training data
    Sample $\epsilon \sim \mathcal{U}(0,1)^m$ for each datum
    $\mathcal{L} \leftarrow$ Eq. (3)
    $(\theta,\phi) \leftarrow \left(\theta - \Gamma(\nabla_\theta\mathcal{L}), \phi - \Gamma(\nabla_\phi\mathcal{L})\right)$ according to Eq.. (6)
until *convergence or reaching the maximum iteration T*;

The second RHS term of Eq. (3) acts as a regularizer. By minimizing the Kullback-Leibler (KL) divergence between the posterior $q(\mathbf{b}|\mathbf{x})$ and the prior $p(\mathbf{b})$, the entropy carried by B is reserved. As the prior and the posterior are basically binomial, the KL divergence can be deterministically computed by two entropy terms $\mathcal{H}(\cdot)$:

$$\mathrm{KL}\,(q(\mathbf{b}|\mathbf{x})||p(\mathbf{b})) = \mathcal{H}\big(q(\mathbf{b}|\mathbf{x}),p(\mathbf{b})\big) - \mathcal{H}\big(p(\mathbf{b}),p(\mathbf{b})\big).$$
(4)

The whole network of JMLH is trained only using Eq. (3). This makes the optimization extremely simple, requiring no auxiliary module or additional complex loss function. The only problem comes from the gradient computation of the intractable expected negative log-likelihood w.r.t. θ, which is discussed in Section 2.1.3.

2.1.3 On the Tractability of JMLH

Computing the gradients of the negative log-likelihood expectation term $\nabla_\theta\mathbb{E}_{q(\mathbf{b}|\mathbf{x})}[-\log q(y|\mathbf{b})]$ of Eq. (3) is intractable. One needs to traverse the latent space of B for each sample $\mathbf{x}$ to accurately obtain the loss and corresponding gradients. Inspired by [12], we use the following reparametrization of B:[2]

$$\widetilde{\mathbf{b}}_i = \begin{cases} 1 & \kappa_i(\mathbf{x};\theta) \geqslant \epsilon_i, \\ 0 & \kappa_i(\mathbf{x};\theta) < \epsilon_i, \end{cases} \quad \text{for } i = 1 \ldots m, \quad (5)$$

where each $\epsilon_i \sim \mathcal{U}(0,1)$ is a small random signal. Eq. (5) is conventionally termed as the stochastic binary neural activation. With this reparametrization, the gradient of $\mathcal{L}$ w.r.t. the encoder parameters θ can be estimated by the distributional derivative estimator [12]:

$$\nabla_\theta\mathcal{L} = \frac{1}{n}\sum_{(\mathbf{x},y)}\Big(\mathbb{E}_\epsilon[-\nabla_\theta\log q(y|\widetilde{\mathbf{b}})]$$
$$+ \lambda\nabla_\theta\,\mathrm{KL}\,(q(\mathbf{b}|\mathbf{x})||p(\mathbf{b}))\Big)$$
(6)

---

[2]Although the reparametrization trick [21] is initially designed for continuous variables, we keep using this terminology here, because the trick proposed in [12] leads to a similar gradient estimator to the one of [21].
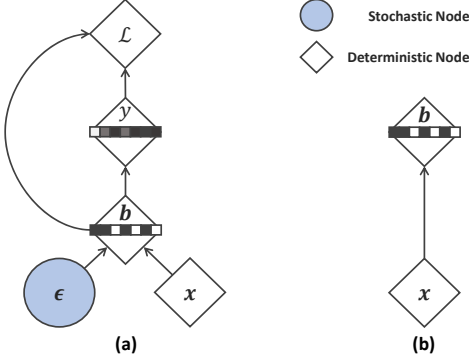
Figure 2. An analogy of the JMLH computation graphs for **(a)** training and **(b)** test.

With this estimator, the network of JMLH can be trained with SGD end-to-end. Note that $\nabla_\phi \mathcal{L}$ can be deterministically obtained and does not require approximation since $\pi(\mathbf{b}; \phi)$ does not involve stochasticity.

The whole training process is illustrated in Algorithm 1, and the respective variable feed path is illustrated in Figure 2 (a). Here we use $\Gamma(\cdot)$ to denote the gradient scaler, which is the Adam optimizer [20] in this work. It can be seen that, during training, JMLH performs identically to a normal neural classifier. The only additional step is just to sample the random signals ϵ.

2.1.4 Out-of-Sample Extension

Given a query datum $\mathbf{x}^{(q)}$, the corresponding hash code is produced by the encoder, *i.e.*,

$$\mathbf{b}^{(q)} = \big(\text{sign}(\kappa(\mathbf{x}^{(q)}; \theta) - 0.5) + 1 \big)/2, \qquad (7)$$

which is shown in Figure 2 (b).

2.2. Theoretical Analysis

2.2.1 Exploring the Information Bottleneck

In this subsection, we show that JMLH defines a special discrete extension of VIB [2] to learn information-rich codes. By empirically assigning the joint probability of X and Y with the Dirac delta function $p(\mathbf{x}, y) = \frac{1}{n} \sum_i \delta(\mathbf{x} - \mathbf{x}_i)\delta(y - y_i) = p(y|\mathbf{x})p(\mathbf{x})$, *i.e.*, data samples are independent, the negative learning objective of JMLH can be rewritten as

$$-\mathcal{L} = \frac{1}{n} \sum_{(\mathbf{x},y)} \sum_{\mathbf{b}} \Big(p(\mathbf{x})p(y|\mathbf{x})q(\mathbf{b}|\mathbf{x}) \log q(y|\mathbf{b})$$
$$-\lambda p(\mathbf{x})q(\mathbf{b}|\mathbf{x}) \log \frac{q(\mathbf{b}|\mathbf{x})}{p(\mathbf{b})} \Big), \qquad (8)$$

where the first RHS term is the variational lower bound of the mutual information $I(B, Y)$ with the second RHS

term the lower bound of the negative mutual information $-\lambda I(B, X)$ according to [2]. Consequently, $-\mathcal{L}$ literally lower-bounds the IB [41] objective $\mathcal{R}_{IB}(X, Y, B)$:

$$\mathcal{R}_{IB}(X, Y, B) = I(B, Y) - \lambda I(B, X) \geq -\mathcal{L}. \quad (9)$$

We refer to the related articles [2, 41] for more detailed definitions.

Intuitively, our learning objective allows B to maximally represent the semantic meaning of the label space Y by ascending $I(B, Y)$. Note that, though $-\lambda I(B, X)$ acts as a penalty in Eq. (9), we are not expecting zero mutual information between X and B, otherwise the produced codes would be data-independent. The purpose of introducing $-\lambda I(B, X)$ is to filter redundant information not related to the semantic meanings of data during encoding, and simultaneously preserve the essential part to support $I(B, Y)$. In this way, the learned codes can be compressed and discriminative.

2.2.2 Nearest Neighbour Search with Recognition

In the context of large-scale data retrieval, relevant data pairs are usually and conveniently defined by sharing the labels/tags, which is generally reasonable. It is trivial and inefficient to traverse all data points in a dataset and explicitly assign pair-wise similarity marks to each of them, while the labels/tags can be regarded as the similarity 'anchors' to ease this process.

JMLH favors this setting as it is literally a special classifier during training. The bottleneck latents B are directly linked to the data labels. When the model is well-trained, the codes of relevant data are naturally located with short Hamming distances. This idea has also been proved in many label-based hashing approaches [19, 34].

3. Related Work

Our work is related to various hashing techniques, of which the most popular and related ones are selectively discussed according to our motivation and design.

3.1. Solving the Discrete Constraints

Traditional solutions. We firstly look at the problem of discrete optimization. A typical example is SDH [34], which also sequentially behaves encoding and classification. However, as SDH [34] resorts to Discrete Cyclic Coordinate descent (DCC) for alternating code updating, a held-out optimization step is involved. Practically, this is hard for parallelization and batch-wise optimization. Additionally, training errors of the classification step cannot be efficiently propagated back to the encoder. A similar paradigm can be found in [44], while its objective is based on pair-wise data similarity. In both single-modal hashing [45, 13, 32, 31, 10]

and cross-modal hashing [25, 36, 9], alternating code updating is widely adopted. For those methods that have held-out code-learners, the network is regularized by the produced target code. The disadvantage of this disarticulated process is the low training quality. On the other hand, regularizing the network by quantization is also widely considered [6, 14, 19, 35]. However, these approaches ignore a severe problem of the different presence of codes. The network observes continuous codes during training, which may represent different meanings from their discrete counterparts for test. This problem is explicitly solved in JMLH as our code bottleneck is exactly binary.

Gradient estimation solutions. Some existing hashing models solve the discrete constraints for SGD by gradient estimation techniques so that the hashing model can be conveniently trained. In SGH [12], a distributional derivative estimator is proposed based on the Taylor expansion of the gradient, and the discreteness is kept by the stochastic neuron. This approach has a similar presence to the reparametrization trick [21], and is unbiased and stable during training. This is also adopted in [37], and JMLH follows the same idea. An alternative simple choice here is the Straight-Through (ST) estimator [3], which is used in GreedyHash [40]. The REINFORCE algorithm [43] is also employed for the same purpose in [46], while it undergoes high variance during training.

3.2. Enriching the Semantic Information

JMLH is not the first model that trains the hashing network with classification objectives. For instance, SUBIC [19] also employs a classification loss as its learning objective. Specifically, SUBIC [19] separates the hash code into l blocks and ground each code block on a $\Delta^{\frac{m}{l}-1}$ simplex in order to favor the discreteness. This approach considerably limits the maximal information carried by the codes. Besides, the supervised version of GreedyHash [40] is similar to JMLH both in terms of classification objective and keeping the discrete constraints. However, GreedyHash [40] only uses the quantization loss on the code bottleneck, ignoring the entropy of the codes, while we consider minimizing $\mathrm{KL}\left(q(\mathbf{b}|\mathbf{x})||\mathcal{B}(\mathbf{b}|m, 0.5)\right)$ to preserve the entropy. Moreover, GreedyHash [40] provides no theoretical clue of how the trained codes are related to data semantics.

MIHash [4] borrows the concept of mutual information as with JMLH, ending up with different designs. Our model reflects the mutual information between codes and data semantics as a part of VIB [2], while MIHash [4] considers relevant-irrelevant code distribution discrepancy and requires complex histogram binning [42] during training.

Recently, a popular idea in deep representation learning is to employ Generative Adversarial Networks (GANs) [18] during training, which has been attempted in [5, 15, 39, 50]. The discriminators or the encoders in GANs are aware of the data distribution $p(X)$ without explicitly parameterizing $p(X)$. The problem is that the auxiliary generator significantly increases the training complexity as more parameters are introduced.

We experimentally show that the above sophisticated designs are not always necessarily needed as the simple network of JMLH can already achieve the state-of-the-art retrieval performance.

4. Experiments

Extensive image retrieval experiments are conducted in this section, mainly according to the following themes:

- **Comparison with existing methods.** We show that, simple as JMLH is, it still outperforms state-of-the-art hashing models.

- **Ablation study.** The importance of each part of JMLH is evaluated and discussed.

- **Intuitive results.** Some illustrative results are provided to implicitly justify the effectiveness of JMLH.

4.1. Experimental Settings

4.1.1 Implementation Details

JMLH is implemented with the popular deep learning toolbox Tensorflow [1]. The network specifics are provided in Table 1. For our image retrieval task, AlexNet [23] before the `fc_7` layer is adopted as the network backbone, where parameters are initialized with the ImageNet [33] pre-trained results and is jointly updated during training. For multi-labeled datasets, *i.e.*, NUS-WIDE [11], we activates the last layer of $\pi(y|\mathbf{b})$ with the `sigmoid` non-linearity, while the `softmax` activation is used when training JMLH on CIFAR-10 [22] and ImageNet [33]. JMLH involves one hyper-parameter, *i.e.*, the regularization factor λ. We empirically set $\lambda = 0.1$. The learning rate of the Adam optimizer $\mathbf{\Gamma}\left(\cdot\right)$ [20] is set to 1×10^{-4}. We fix the training batch size to 256. The codes can be found at https://github.com/ymcidence/JMLH.

4.1.2 Datasets

CIFAR-10 [22] consists of 60,000 images from 10 classes. We follow the common setting [15, 24, 40] and select 1,000 images (100 per class) as the query set. The remaining 59,000 images are regarded as the database. The training set contains 5000 images, uniformly selected from the database.

NUS-WIDE [11] is a collection of nearly 270,000 Web images of 81 categories downloaded from Flickr. Following the settings in [28, 44, 24], we adopt the subset of images

Table 2. Performance comparison (w.r.t. mAP@k) of JMLH and the state-of-the-art hashing methods. The respective retrieval sequence length k is adopted according to the most popular settings [15, 40, 46]. All baselines are reported according to the identical setting.

Method	Super-vision	CIFAR-10 (mAP@all)			NUS-WIDE (mAP@5000)			ImageNet (mAP@1000)		
		16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
ITQ [16]	✗	0.201	0.207	0.235	0.627	0.645	0.664	0.217	0.317	0.391
AGH [28]	✗	0.217	0.205	0.182	0.592	0.615	0.616	0.241	0.327	0.379
DGH [26]	✗	0.199	0.200	0.212	0.572	0.607	0.627	0.270	0.341	0.373
KSH [27]	✓	0.451	0.473	0.507	0.448	0.520	0.566	0.216	0.257	0.394
ITQ-CCA [17]	✓	0.463	0.498	0.505	0.555	0.512	0.460	0.235	0.377	0.576
SDH [34]	✓	0.499	0.525	0.546	0.595	0.595	0.617	0.298	0.431	0.504
CNNH [44]	✓	0.453	0.509	0.537	0.570	0.583	0.600	0.281	0.450	0.554
DNNH [24]	✓	0.556	0.558	0.599	0.598	0.616	0.639	0.290	0.461	0.565
DHN [48]	✓	0.564	0.603	0.626	0.637	0.664	0.671	0.311	0.472	0.573
HashNet [8]	✓	0.643	0.675	0.687	0.662	0.699	0.716	0.506	0.631	0.684
MIHash [4]	✓	0.760	0.776	0.761	0.722	0.759	0.779	0.569	0.661	0.694
HashGAN [5]	✓	0.668	0.731	0.749	0.715	0.737	0.748	-	-	-
PGDH [46]	✓	0.741	0.747	0.762	0.780	0.786	0.792	0.653	0.707	0.716
GreedyHash [40]	✓	0.786	0.810	0.833	-	-	-	0.625	0.662	0.688
JMLH (Ours)	✓	**0.805**	**0.841**	**0.837**	**0.795**	**0.818**	**0.820**	**0.668**	**0.714**	**0.727**

from the 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.

ImageNet [33] is originally released for large-scale image classification purpose, and is recently used in deep hashing evaluation. Following [8, 46], we randomly select 100 categories to perform our retrieval task. All the original training images are used as the database, and all the validation images form the query set. For each category, 130 images are used for training.

4.2. Comparison with Existing Methods

We compare JMLH with existing methods using conventional evaluation metrics, including top-k mean-Average Precision (mAP@k), Precision of top-k retrieved samples (Precision@k), Precision within Hamming radius of 2 (P@H≤2) and Precision-Recall (P-R) curves.

Note that, for mAP@k, we adopt the most popular settings of $k = all, 5000, 1000$ for **CIFAR-10**, **NUS-WIDE**, and **ImageNet** respectively according to [15, 40, 46].

4.2.1 Baselines

JMLH is compared with various widely recognized hashing baselines, including ITQ [16], AGH [28], DGH [26], KSH [27], ITQ-CCA [17], SDH [34], CNNH [44], DNNH [24], DHN [48], HashNet [8], HashGAN [5] PGDH [46] and the supervised version of GreedyHash [40]. Note that the term of *HashGAN* is used both in [15] and [5]. Here we refer to the later one as it is a supervised approach and thus is more related to our work.

For feature-based models, *e.g.*, shallow hashing models, we use the AlexNet [23] fc_7 pre-trained features to represent data for training and test. As for the end-to-end baseline frameworks, we directly adopt the original training settings described in their original papers and pre-trained weights are also applied for fine-tuning when possible.

4.2.2 Results and Analysis

The retrieval mAP@k results are reported in Table 2. The respective P-R curves, Precision@k and P@H≤2 scores are illustrated in Figure 3.

It can be observed that JMLH consistently outperforms the compared baselines, though many of them consist of more trainable parameters, *e.g.*, HashGAN [5]. This result aligns with our motivation, and shows the clue that, with the current evaluation metrics, one may not require an extremely complex model to obtain the best-performing deep hashing function.

The performance margin between JMLH and Greedy-Hash [40] is not significant on CIFAR-10 [22], but this gap gets larger when it comes to a relatively more difficult situation, *i.e.*, ImageNet [33]. This raises the concern of a proper regularization term for training. Both GreedyHash [40] and JMLH are trained with classification-oriented objectives. The former literally involves a quantization penalty while JMLH considers equally distributed $\{0, 1\}$ bits to maximize the expected code entropy. This factor becomes essential when the data label space is large and the training samples are limited as the codes need to be expressive enough to be successfully classified. We find our design has better generalization ability in this case.
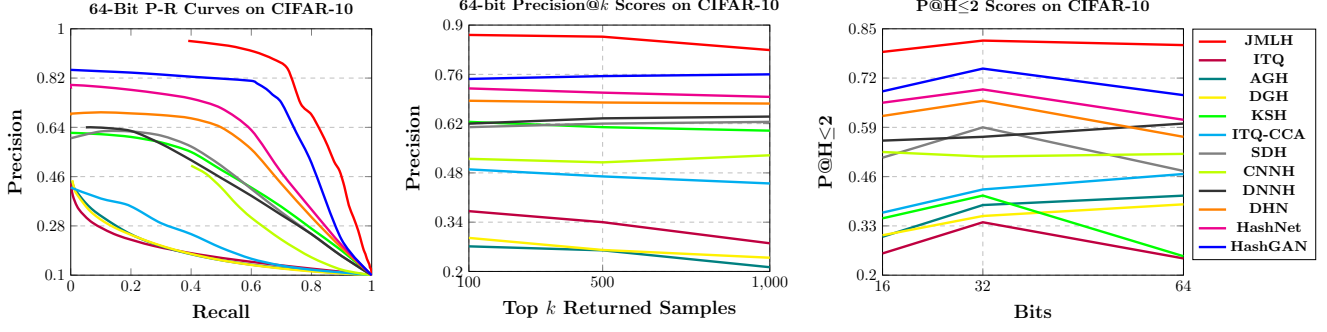
Figure 3. **Left:** 64-bit P-R curves on CIFAR-10 [22]. **Middle:** 64-bit precision of top k returned samples on CIFAR-10 [22]. **Right:** Precision within Hamming radius of 2 scores on CIFAR-10 [22].

4.3. Ablation Study

In this subsection, we evaluate different components in terms of formulating a simple deep hashing model, and empirically show which one is of importance for good performance.

4.3.1 Baselines

JMLH-Cont. We firstly look at the influence of quantization. By dropping the binary stochastic neuron and employing the `sigmoid` activation on the code bottleneck B, a regular deep neural classifier is built. The regularization term is kept, and is subsequently analyzed by other baselines.

JMLH-QR. The KL term of Eq. (3) is replaced by the **quantization regularizer** between the activated binary codes B and their real-valued counterparts before the stochastic neurons.

JMLH-NR. The regularizer is deprecated in this baseline, and the whole learning objective is formulated by the classification cross-entropy.

JMLH-VAE. We replace the classifier $\pi(\cdot)$ with a decoder, and use the $L2$ reconstruction error instead of classification loss during training. Therefore, the model collapses to an unsupervised Variational Auto-Encoder (VAE) [21], with a negative Evidence Lower-BOund (ELBO) of

$$\frac{1}{n} \sum_{\mathbf{x}} \mathbb{E}_{q(\mathbf{b}|\mathbf{x})}[-\log q(\mathbf{x}|\mathbf{b})] + \mathrm{KL}\left(q(\mathbf{b}|\mathbf{x})||p(\mathbf{b})\right). \quad (10)$$

For the simplicity of training, the encoder and decoder for this baseline are both implemented with a two-layer neural networks and are fed by AlexNet [23] `fc_7` features.

4.3.2 Results and Analysis

The mAP results of the above-mentioned baselines are shown in Table 3. Since JMLH-VAE is an unsupervised model, its performance is relatively lower than the others.

Table 3. mAP@all results by using different variants of the proposed JMLH on CIFAR-10.

	Baseline	16 bits	32 bits	64 bits
1	JMLH-Cont	0.616	0.628	0.659
2	JMLH-QR	0.778	0.827	0.835
3	JMLH-NR	0.729	0.725	0.736
4	JMLH-VAE	0.423	0.435	0.441
5	**JMLH (full model)**	**0.805**	**0.841**	**0.837**

We experience a 20% performance drop when using the continuous relaxation during training, *i.e.*, JMLH-Cont. As discussed in Section 3, the binary constraints are essential for models like JMLH as it directly influences the classifier's observation. Without regularization, JMLH-NR struggles in the training-test generalization. Though not competing our full model, JMLH-QR still performs closely to GreedyHash [40], as the learning objectives are similar. The difference between JMLH-QR and GreedyHash [40] lies in the stochasticity of gradient estimation. Both ST [3] and distributional derivative [12] work for this case as long as the binary constraints are not violated. Hence, a proper learning objective becomes more important.

4.4. More Results

4.4.1 Hyper-Parameter

The regularization penalty of JMLH is scaled by a hyper-parameter λ. By default, it is set to $\lambda = 0.1$ for the overall best performance. The impact of λ is illustrated in Figure 4 (a). The performance drops quickly when λ goes larger, which actually reflects the penalty of the mutual information between data X and codes B, *i.e.*, $I(X, B)$. A large value of λ over-regularizes the model by decorrelating X with B, making the produced codes less-informative.

4.4.2 Towards Model Simplicity

One key claim of this paper is to build a simple deep hashing model. Training JMLH is non-trivial and effi-
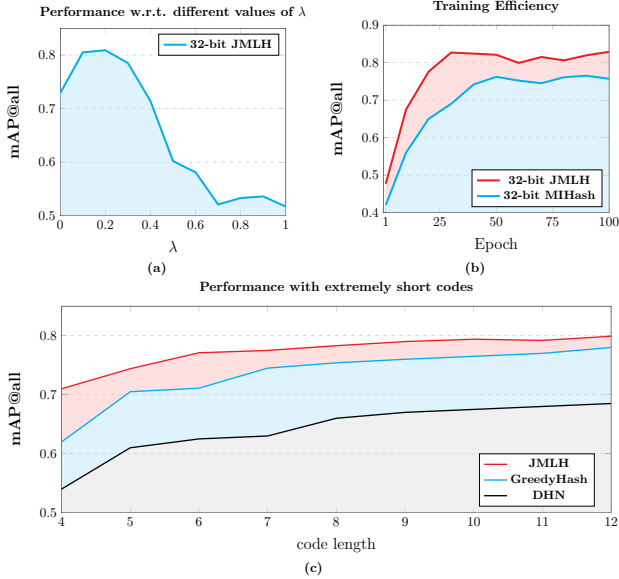
Figure 4. **(a)** mAP@all results of 32-bit JMLH on CIFAR-10 [22] with different values of λ. **(b)** Training efficiency of JMLH and MIHash [4] on CIFAR-10 [22]. **(c)** Encoding performance comparison with extremely short code length on CIFAR-10 [22].

Figure 5. **(a)** 32-bit JMLH t-SNE [30] visualization on CIFAR-10 [22]. **(b)** Examples of top-10 retrieved candidates of 32-bit JMLH on CIFAR-10 [22].

cient. Our classification likelihood learning objective provides a straightforward way to convey data semantics to the encoder. We show training efficiency comparison between JMLH and MIHash [4] in Figure 4 (b). It can be observed that JMLH converges more quickly to the best performance than MIHash [4] with a margin of ~10 epochs. Although MIHash [4] requires no auxiliary networks, its histogram-based learning objective introduces complex positive-negative data pairing and histogram binning. All these factors make the training of MIHash [4] indirect, resulting in relatively slower convergence rate than JMLH. Note that the performance of MIHash is slightly lower than the one reported in [4], as it was previously trained with VGG [38] features and we reproduce the results with the AlexNet [23] backbone for fair comparison.

The whole parameter size of JMLH for all experiments conducted in this section is slightly smaller than AlexNet [23], as we have a relatively narrow fully-connecting bottleneck in the middle. Compared with the models that involve end-to-end generative networks [15, 5], this is believed to be a light one.

4.4.3 Extremely Short Codes

Following [40], we also explore the minimal size of codes to represent data semantics. The experiments are conducted by setting the code length to $m = 4, 5, ..., 11, 12$, and the corresponding results are shown in Figure 4 (c). We can see that, compared with GreedyHash [40] and DHN [48],
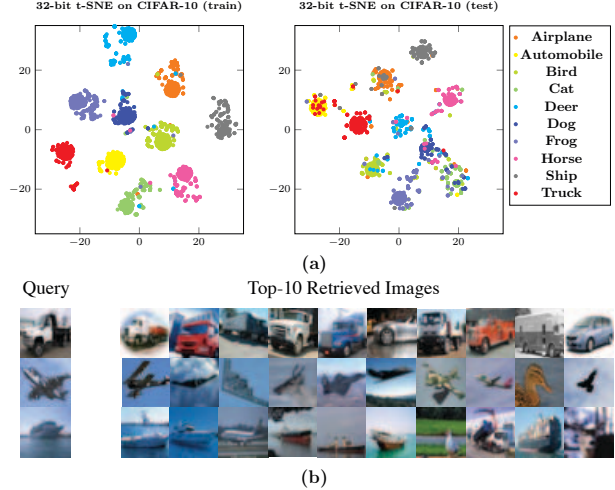
JMLH obtains better performance even when the encoding length is very short. The entropy-preserving regularization term plays the key role here since the maximum number of concepts that the code space can cover is limited.

4.4.4 Visualization Results

The t-SNE [30] visualization of 32-bit JMLH on CIFAR-10 [22] is shown in Figure 5 (a). Even though the proposed model is simple both in terms of network structure and learning objective, the resulting binary codes are still clearly scattered in the feature space according to their semantic meanings. We further provide several image retrieval examples where the top-10 retrieved candidates are shown together with the query image in Figure 5 (b). Obviously, JMLH successfully finds related images in the top of the retrieval list. Here we only show the 32-bit results to keep the content concise.

5. Conclusion

In this paper, we proposed a simple but effective deep hashing model called JMLH. Our model shaped a conventional deep neural network with a single likelihood maximization learning objective. A differentiable binary bottleneck was plugged in, making the whole network end-to-end trainable using SGD. JMLH was linked to the information bottleneck methods, which aimed at learning maximally representative features for a given task. We showed that, when applying proper binary-preserving gradient estimators and suitable regularization terms, a single classification model could generate high-quality hash codes for similarity search, outperforming state-of-the-art models.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 5

[2] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. In *International Conference on Learning Representations (ICLR)*, 2016. 2, 3, 4, 5

[3] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 5, 7

[4] F. Cakir, K. He, S. Adel Bargal, and S. Sclaroff. Mihash: Online hashing with mutual information. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 5, 6, 8

[5] Y. Cao, B. Liu, M. Long, and J. Wang. Hashgan: Deep learning to hash with pair conditional wasserstein gan. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1, 5, 6, 8

[6] Y. Cao, M. Long, B. Liu, and J. Wang. Deep cauchy hashing for hamming space retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 5

[7] Z. Cao, M. Long, J. Wang, and P. S. Yu. Hashnet: Deep learning to hash by continuation. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1

[8] Z. Cao, M. Long, J. Wang, and P. S. Yu. Hashnet: Deep learning to hash by continuation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 6

[9] J. Chen, J. Qin, L. Liu, F. Zhu, F. Shen, J. Xie, and L. Shao. Deep sketch-shape hashing with segmented 3d stochastic viewing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 5

[10] J. Chen, Y. Wang, J. Qin, L. Liu, and L. Shao. Fast person re-identification via cross-camera semantic binary transformation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 4

[11] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: a real-world web image database from national university of singapore. In *ACM International Conference on Image and Video Retrieval (CIVR)*, 2009. 2, 5

[12] B. Dai, R. Guo, S. Kumar, N. He, and L. Song. Stochastic generative hashing. In *International Conference on Machine Learning (ICML)*, 2017. 3, 5, 7

[13] T.-T. Do, A.-D. Doan, and N.-M. Cheung. Learning to hash with binary deep neural network. In *European Conference on Computer Vision (ECCV)*, 2016. 1, 4

[14] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 5

[15] K. Ghasedi Dizaji, F. Zheng, N. Sadoughi, Y. Yang, C. Deng, and H. Huang. Unsupervised deep generative adversarial hashing network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1, 5, 6, 8

[16] 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. 6

[17] 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. 6

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems (NIPS)*, 2014. 5

[19] H. Jain, J. Zepeda, P. Perez, and R. Gribonval. Subic: A supervised, structured binary code for image search. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 4, 5

[20] D. Kingma and J. Ba. Adam: A method for acm symposium on theory of computing (stoc)hastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 4, 5

[21] D. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014. 3, 5, 7

[22] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 2, 5, 6, 7, 8

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3, 5, 6, 7, 8

[24] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 5, 6

[25] L. Liu, F. Shen, Y. Shen, X. Liu, and L. Shao. Deep sketch hashing: Fast free-hand sketch-based image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 5

[26] W. Liu, C. Mu, S. Kumar, and S.-F. Chang. Discrete graph hashing. In *Advances in Neural Information Processing Systems (NIPS)*, 2014. 6

[27] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 6

[28] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *International Conference on Machine Learning (ICML)*, 2011. 5, 6

[29] Z. Liu, J. Qin, A. Li, Y. Wang, and L. Van Gool. Adversarial binary coding for efficient person re-identification. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2019. 1

[30] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. 8

[31] J. Qin, L. Liu, L. Shao, B. Ni, C. Chen, F. Shen, and Y. Wang. Binary coding for partial action analysis with limited observation ratios. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 4

[32] J. Qin, L. Liu, M. Yu, Y. Wang, and L. Shao. Fast action retrieval from videos via feature disaggregation. *Computer Vision and Image Understanding*, 156:104–116, 2017. 4

[33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 2, 5, 6

[34] F. Shen, C. Shen, W. Liu, and H. Tao Shen. Supervised discrete hashing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 4, 6

[35] Y. Shen, L. Liu, and L. Shao. Unsupervised binary representation learning with deep variational networks. *International Journal of Computer Vision, DOI: 10.1007/s11263-019-01166-4*. 5

[36] Y. Shen, l. Liu, L. Shao, and J. Song. Deep binaries: encoding semantic-rich cues for efficient textual-visual cross retrieval. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 5

[37] Y. Shen, L. Liu, F. Shen, and L. Shao. Zero-shot sketch-image hashing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 5

[38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference in Learning Representations (ICLR)*, 2015. 8

[39] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, and H. T. Shen. Binary generative adversarial networks for image retrieval. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. 1, 5

[40] S. Su, C. Zhang, K. Han, and Y. Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *Advances in Neural Information Processing Systems*, 2018. 5, 6, 7, 8

[41] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. In *Annual Allerton Conference on Communication, Control, and Computing*, 1999. 2, 4

[42] E. Ustinova and V. Lempitsky. Learning deep embeddings with histogram loss. In *Advances in Neural Information Processing Systems (NIPS)*. 2016. 5

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

[44] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI Conference on Artificial Intelligence (AAAI*, 2014. 4, 5, 6

[45] Y. Yang, Y. Luo, W. Chen, F. Shen, J. Shao, and H. T. Shen. Zero-shot hashing via transferring supervised knowledge. In *ACM international conference on Multimedia (MM)*, 2016. 1, 4

[46] X. Yuan, L. Ren, J. Lu, and J. Zhou. Relaxation-free deep hashing via policy gradient. In *The European Conference on Computer Vision (ECCV)*, September 2018. 5, 6

[47] 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

[48] H. Zhu, M. Long, J. Wang, and Y. Cao. Deep hashing network for efficient similarity retrieval. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016. 1, 6, 8

[49] B. Zhuang, G. Lin, C. Shen, and I. Reid. Fast training of triplet-based deep binary embedding networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[50] M. Zieba, P. Semberecki, T. El-Gaaly, and T. Trzcinski. Bin-gan: Learning compact binary descriptors with a regularized gan. In *Advances in Neural Information Processing Systems (NIPS)*, 2018. 1, 5