Deep Unsupervised Image Hashing by Maximizing Bit Entropy

Yunqiang Li and Jan van Gemert
Vision Lab, Delft University of Technology, Netherlands
{y.li-19, j.c.vangemert}@tudelft.nl

Abstract

Unsupervised hashing is important for indexing huge image or video collections without having expensive annotations available. Hashing aims to learn short binary codes for compact storage and efficient semantic retrieval. We propose an unsupervised deep hashing layer called Bi-half Net that maximizes entropy of the binary codes. Entropy is maximal when both possible values of the bit are uniformly (half-half) distributed. To maximize bit entropy, we do not add a term to the loss function as this is difficult to optimize and tune. Instead, we design a new parameter-free network layer to explicitly force continuous image features to approximate the optimal half-half bit distribution. This layer is shown to minimize a penalized term of the Wasserstein distance between the learned continuous image features and the optimal half-half bit distribution. Experimental results on the image datasets Flickr25k, Nus-wide, Cifar-10, Mscoco, Mnist and the video datasets Ucf-101 and Hmdb-51 show that our approach leads to compact codes and compares favorably to the current state-of-the-art.

1 Introduction

Semantically similar images or videos can be found by comparing their output features in the last layer of a deep network. Such features are typically around 1,000 continuous floating point values (He et al. 2016), which is already too slow and large for moderately sized datasets of a few million samples. Speed and storage are greatly improved by replacing the continuous features with just a small number of bits. Unsupervised hashing aims to learn compact binary codes that preserves semantic similarity without making use of any annotated label supervision and is thus of great practical importance for indexing huge visual collections.

In this paper, as illustrated in Fig. 1, we see the transition from a continuous variable to a discrete binary variable as a lossy communication channel. The capacity of a hash bit as measured by the entropy is maximized when it is half-half distributed: Half of the images are encoded with -1 and the other half of the images is encoded with $+1$. We minimize the information loss in the hash channel by forcing the continuous variable to be half-half distributed. Other methods have optimized entropy by adding an additional term to the

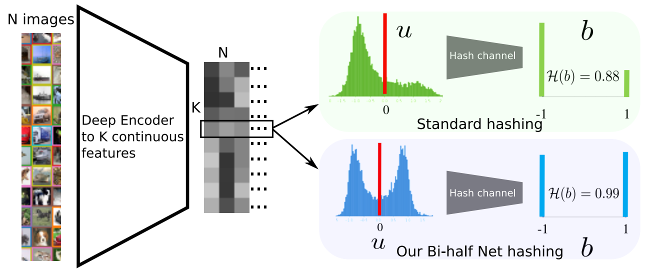


Figure 1: Hashing compresses N images to K bits per image. K continuous features are learned, and thresholded to K binary values. We see the continuous to binary transition as a lossy communication channel –a hash channel– between a single continuous value u to a single discrete binary value b. The green histograms (standard hashing) and blue histograms (our approach) show how a single feature is distributed over the N images. Instead of adding an additional loss term, we design a Bi-half layer to explicitly maximizes the bit capacity in the hash channel, leading to more informative hash codes, as measured by the entropy $\mathcal{H}$ of the bits over the images which leads to improved hashing accuracy.

loss (Erin Liong et al. 2015; Liu et al. 2014; Shen et al. 2018; Weiss, Torralba, and Fergus 2008; Xu et al. 2013) which adds an additional hyper-parameter to tune and is difficult to optimize. Instead, we propose Bi-half: A new parameter-free network layer which is shown to minimize the optimal transport cost as measured by the Wasserstein distance. We here explicitly design a new layer to maximize the bit capacity in the hash channel, leading to compact and informative hash codes which yield excellent hashing similarity accuracy.

We have the following contributions.

- A simple, parameter-free, bi-half layer to maximize hash channel information capacity;
- A Wasserstein distance is minimized end-to-end to align continuous features with the optimal discrete distribution;
- We study 2 alternatives to maximizing bit entropy using an additional term in the loss;
- We show state-of-the-art results for unsupervised hashing on 5 image datasets, and 2 video datasets, and make our code available[1].
- ;

[1]https://github.com/liyunqianggyn/Deep-Unsupervised-Image-Hashing

2 Related Work

Amount of supervision. Hashing methods can be grouped into data-independent hashing methods and data-dependent hashing methods. Data-independent hashing methods (Datar and Indyk 2004; Gionis, Indyk, and Motwani 2000; Kulis and Grauman 2009; Kulis, Jain, and Grauman 2009; Mu and Yan 2010; Raginsky 2009) design hash function independent of a dataset. In contrast, data-dependent hashing methods can exploit the data distribution. As such, with the availability of labeled training data, supervised hashing methods (Chang 2012; Lai et al. 2015b; Lin et al. 2014a; Raziperchikolaei and Carreira-Perpiñán 2016; Shen et al. 2015; Zhang et al. 2014) learn hash codes by optimizing class labels. Particularly successful supervised image hashing methods use deep learning (Cao et al. 2018; 2017; Lai et al. 2015b; Li et al. 2017; 2019; Liu et al. 2016; 2017; Yuan et al. 2018; Chen et al. 2018) to learn feature representations and binary codes. Supervised methods work well, yet rely on data annotations done by humans, which are expensive or difficult to obtain. Unsupervised hashing methods (Gong and Lazebnik 2011; He, Wen, and Sun 2013; Jiang and Li 2015; Kong and Li 2012; Liu et al. 2014; 2011; Weiss, Torralba, and Fergus 2008) skip this problem, as they do not rely on annotation labels. Recent unsupervised hashing methods rely on deep learning for representation learning (Dai et al. 2017; Yang et al. 2019; Shen et al. 2018; Ghasedi Dizaji et al. 2018; Lin et al. 2016; Erin Liong et al. 2015). We follow these works and focus on the unsupervised setting.

Quantization from continuous to discrete values. The typical approach for deep learning hashing is to optimize a continuous output and in the last step quantize the continuous values to discrete values. The current approach (Chen, Cheung, and Wang 2018; Lu, Liong, and Zhou 2017; Su et al. 2018) is to apply a sign function, where all negative values are set to -1 and all positive values are set to $+1$. We argue that the sign function is not information efficient. For example, we set the continuous features of one dimension for 4 images to be $[0.2, 0.8, 1.5, 3]$. Passing them through the sign function will binarize them all to same value $+1$ and thus the bit has no discriminative information for these 4 images. In this paper we focus on this loss of information and learn to discretize based on maximum bit capacity over images.

Obtaining gradients for binary codes. A major challenge of learning hash codes with deep nets is that the desired discrete hash output codes have no continuous derivatives and cannot be directly optimized by gradient descent. By the continuous relaxation (Cao et al. 2018; Jiang and Li 2017; Liu et al. 2016; Zhao et al. 2015), a continuous space is optimized instead and the continuous values are quantized to binary codes. Such methods are approximations as they do not optimize the binary codes directly. The continuation based hashing methods (Cao et al. 2017; Lai et al. 2015a) gradually approximate the non-smooth *sign* function with *sigmoid* or *tanh*, but unfortunately comes with the drawback that such relaxation inevitably becomes more non-smooth during training which slows down convergence, making it difficult to optimize. To overcome these problems, a recent simple and efficient method called greedy hash (Su et al. 2018), uses the sign function in the forward pass to directly optimize binary codes. The optimization is done with the straight-through estimator (Bengio, Léonard, and Courville 2013) which after quantization computes gradients by simply ignoring the quantization function during training. This optimization is simple and works well in practice. Yet, it ignores bit information capacity and thus may lead to redundant codes. In this work we use the same straight-through estimator to obtain gradients for binary codes while focusing on maximizing bit information capacity to obtain compact and discriminative hash codes.

Information theory in hashing. Many popular unsupervised feature learning methods (Belghazi et al. 2018; Chen et al. 2016; Jolliffe 2002) are based on information theory to find good features. In hash learning, some methods (Erin Liong et al. 2015; Liu et al. 2014; Shen et al. 2018; Weiss, Torralba, and Fergus 2008; Xu et al. 2013) proposed to add an additional term in the loss function to encourage each bit to have a 50% chance of being one or zero, to maximize bit entropy. It is, however, difficult to balance the added loss term with other terms in the loss, which requires careful hyper-parameter tuning of how much to weight each term in the loss. Instead of adding an additional loss term and an additional hyper-parameter, we design a new network layer without any additional parameters to explicitly force continuous image features to approximate the optimal half-half bit distribution. Some non-deep learning approaches (Jegou, Douze, and Schmid 2008; Zhang et al. 2010) directly threshold the learned feature vectors at their median point, which have shown excellent performance. Yet, it is a suboptimal solution under deep learning scenario since the median point should be dynamically adapted to random sample statistic computed over each minibatch. We are inspired by their works, and aim to generalize such ideas to an end-to-end deep learnable setting. We cast it into an optimal transport problem and directly quantize the continuous features into half-half distributed binary codes by minimizing the Wasserstein distance between the continuous distribution and a prior half-half distribution.

3 Approach

This paper we maximize the hash channel capacity to design a parameter-free bi-half coding layer. We will first introduce some notations. Let $\mathbf{X} = \{\mathbf{x_i}\}_{i=1}^{N}$ denote N training images. The images are encoded to K compact binary codes $\mathbf{B} \in \{1, -1\}^{N \times K}$, which also denotes the output of our hash coding layer. $\mathbf{U} \in \mathbb{R}^{N \times K}$ would be expressed as the continuous feature representations in the last layer of a standard neural network, *e.g.*, an encoder, which serves as the input to our hash coding layer.

3.1 Maximizing hash channel capacity

We see the transition from a continuous variable U to a binary code variable B as a lossy communication channel. Per channel, the *maximum* transmitted information from continuous variable U to binary variable B, known as channel capacity (Cover and Thomas 2012; Shannon 1948), is:

$$C = \max_{p(u)} I(U; B), \tag{1}$$

where the maximum is taken over all possible input distributions $p(u)$ and $I(U;B)$ denotes mutual information between variable U and binary variable B. We aim to maximize the channel capacity. To maximize channel capacity C we first rewrite the mutual information term $I(U;B)$ in Eq. (1) in terms of entropy:

$$I(U;B) = \mathcal{H}(B) - \mathcal{H}(B|U), \tag{2}$$

where $\mathcal{H}(B)$ and $\mathcal{H}(B|U)$ denote entropy and conditional entropy respectively. Thus, maximizing channel capacity C in Eq. (1) is equivalent to maximizing the entropy $\mathcal{H}(B)$ of B and minimizing the conditional entropy $\mathcal{H}(B|U)$.

The entropy $\mathcal{H}(B)$ in Eq. (2) should be maximized. Since B is a discrete binary variable, its entropy is maximized when it is half-half distributed:

$$p(B=+1) = p(B=-1) = \frac{1}{2}. \tag{3}$$

The conditional entropy $\mathcal{H}(B|U)$ in Eq. (2) should be minimized. Give a certain continuous value u, the transmission probability $p_u(\text{pos})$ is defined as how probable a $+1$ binary output value is and the transmission probability $p_u(\text{neg})$ is defined as how probable the binary value -1 is. These are probabilities and thus are non-negative and sum to one as $p_u(\text{pos}) + p_u(\text{neg}) = 1$ and $0 \leq p_u(\text{pos}), p_u(\text{neg}) \leq 1$. Then the conditional entropy is computed as:

$$\begin{aligned}\mathcal{H}(B|U) &= \int_{u\in\mathcal{U}} p(u)\mathcal{H}(B|U=u)du \\ &= -\int_{u\in\mathcal{U}} p(u)\Big(p_u(\text{pos})\log p_u(\text{pos}) \\ &\qquad + p_u(\text{neg})\log p_u(\text{neg})\Big)du,\end{aligned} \tag{4}$$

which is between 0 and 1, and Eq. (4) is thus minimized for setting the $\mathcal{H}(B|U=u)$ to 0, *i.e.*: $-\int_{u\in\mathcal{U}} p(u)0\ du = 0$. This minimum is obtained when either $p_u(\text{pos}) = 1$ or $p_u(\text{neg}) = 1$, which means that there is no stochasticity for a certain continuous value u, and its binary value is deterministically transmitted.

In the following, we maximize the entropy of binary variables by encouraging the continuous feature distribution $p(u)$ to align with the ideal half-half distributed distribution $p(b)$ in Eq. (3). To minimize Eq. (4), we first start with a non-deterministic transmission probability during training, but since we train to align $p(u)$ with the half-half distribution of $+1$ and -1, this allows us at test time to simply use the sign function as a deterministic function for quantization to guarantee minimizing Eq. (4).

3.2 Bi-half layer for quantization

To align the continuous feature distribution with the ideal prior half-half distributed distribution from Eq. (3) we use Optimal Transport (OT) (Villani 2003). Optimal Transport aims to find a minimal cost plan for moving one unit of mass from one location $\mathbf{x}$ to one other location $\mathbf{y}$ between two probability distributions $\mathbb{P}_r$ and $\mathbb{P}_g$. When $\mathbb{P}_r$ and $\mathbb{P}_g$ are only accessible through discrete samples, the corresponding optimal transport cost can be defined as:

$$\pi_0 = \min_{\pi\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \langle \pi, \mathbf{D}\rangle_F\,, \tag{5}$$

where $\Pi(\mathbb{P}_r,\mathbb{P}_g)$ is the space of joint probability measures with marginals $\mathbb{P}_r$ and $\mathbb{P}_g$, and π is the general probabilistic coupling that indicates how much mass is transported to push distribution $\mathbb{P}_r$ towards distribution $\mathbb{P}_g$. The $\langle .,. \rangle_F$ denotes the Frobenius dot product, and $\mathbf{D} \geq 0$ is the cost function matrix whose element $D(i,j) = d(\mathbf{x},\mathbf{y})$ denotes the non-negative cost to move a probability mass from location $\mathbf{x}$ to location $\mathbf{y}$. When the cost is defined as a distance, OT is referred to as a Wasserstein distance. Specifically, if $d(\mathbf{x},\mathbf{y})$ is the squared Euclidean distance, it is the Earth mover's distance, which is also known as the 1-Wasserstein distance. We optimize the 1-Wasserstein distance because it is flexible and easy to bound.

With a randomly sampled mini-batch of M samples, the corresponding empirical distributions of the continuous variable U and binary variable B, P_u and P_b, can be written as:

$$P_u = \sum_{i=1}^{M} p_i \delta_{u_i}, \quad P_b = \sum_{j=1}^{2} q_j \delta_{b_j}, \tag{6}$$

where $\delta_{\mathbf{x}}$ is the Dirac function at location $\mathbf{x}$. The p_i and q_j are the probability mass associated to the corresponding location u_i and b_j, where the total mass is one, *i.e.*: $\sum_{i=1}^{M} p_i = 1$ and $\sum_{j=1}^{2} q_j = 1$. Particularly, a binary variable only has two locations b_1 and b_2, with the corresponding mass q_1 and q_2.

For the ideal prior half-half distribution in Eq. (3), the probability mass q_1 at location b_1 is equal to the probability mass q_2 at location b_2 that is $q_1 = q_2 = \frac{1}{2}$. The hash coding strategy is to find the optimal transport coupling π_0 by minimizing the 1-Wasserstein distance $W_1(P_u, P_b)$:

$$\pi_0 = \min_{\pi\in\Pi(P_u,P_b)} \sum_i \sum_j \pi_{ij}(u_i - b_j)^2, \tag{7}$$

where $\Pi(P_u, P_b)$ is the set of all joint probability distributions π_{ij}, *i.e.* all probabilistic couplings, with marginals P_u and P_b, respectively.

By optimizing Eq. (7), we find an optimal transport plan $\pi_0 \in \Pi(P_u, P_b)$ for one hash bit to quantize the encoded features into half-half distributed binary codes. For a single hash bit in M samples, with a continuous feature vector $\mathbf{u} \in \mathbb{R}^M$, we first simply sort the elements of $\mathbf{u}$ over all mini-batch images, and then assign the top half elements of sorted $\mathbf{u}$ to $+1$ and assign the remaining elements to -1, that is:

$$\mathbf{b} = \pi_0(\mathbf{u}) = \begin{cases} +1, & \text{top half of sorted } \mathbf{u} \\ -1, & \text{otherwise} \end{cases}. \tag{8}$$

We implement above equation as a new simple hash coding layer, dubbed ***bi-half layer*** shown in Fig.2, to quantize the continuous feature into half-half distributed binary code for each hash channel. The proposed bi-half layer can be easily embedded into current deep architectures to automatically generate higher quality binary codes. During training, the transmission stochasticity introduced by random small batches as shown in Eq. (4) can also improve the model generalization capability as the same effect of denoising.

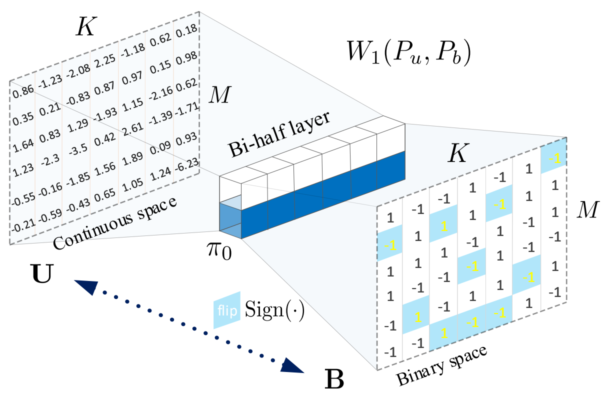


Figure 2: The proposed bi-half layer. M is the mini-batch size and K is the feature dimensions. A bi-half layer (middle part in white and blue) is used to quantize continuous features in $\mathbf{U}$ into binary codes in $\mathbf{B}$ via minimizing $W_1(P_u, P_b)$ in Eq.(7). The assignment strategy is the optimal probabilistic coupling π_0. For each bit, *i.e.*per column of $\mathbf{U}$, we first rank its elements and then the top half elements is assigned to $+1$ and the remaining half elements to -1. In contrast, the commonly used sign function directly during training assigns the continuous features to their nearest binary codes which minimizes the Euclidean distance. The blue boxes indicate where our method differs from the sign function as the code in that position should flip.

Optimization. The discrete binary codes $\mathbf{B}$ have no continuous derivatives and cannot be directly optimized by gradient descent. Fortunately, some recent works on binarized neural networks (BNNs) have explored to use a proxy derivative approximated by straight through estimator (STE) (Bengio, Léonard, and Courville 2013) to avoid the vanishing gradients. We use the same straight-through estimator to obtain the gradients. Specifically, we expect $\mathbf{U}$ and $\mathbf{B}$ have the same update states in backward pass to match the forward goal.

Given time-step t, the current states are denoted as $\mathbf{U}_t$ and $\mathbf{B}_t$. In time-step $t+1$, we force their update states to be same that $\mathbf{U}_{t+1} = \mathbf{B}_{t+1}$. Considering the simplest SGD algorithm, we have $\mathbf{U}_{t+1} = \mathbf{U}_t - lr * \frac{\partial \mathcal{L}}{\partial \mathbf{U}_t}$ and $\mathbf{B}_{t+1} = \mathbf{B}_t - lr * \frac{\partial \mathcal{L}}{\partial \mathbf{B}_t}$ with learning rate lr and loss function $\mathcal{L}$ where $\mathcal{L}$ can be any loss function you need to use, *e.g.* reconstruction loss, cross entropy loss and so on, then the gradient of $\mathbf{U}_t$ is computed as $\frac{\partial \mathcal{L}}{\partial \mathbf{U}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{B}_t} + \gamma(\mathbf{U}_t - \mathbf{B}_t)$ with $\gamma = \frac{1}{lr}$. The forward pass and backward pass are concluded as:

$$\begin{aligned} \text{Forward:} \quad & \mathbf{B} = \pi_0\left(\mathbf{U}\right), \\ \text{Backward:} \quad & \frac{\partial \mathcal{L}}{\partial \mathbf{U}} = \frac{\partial \mathcal{L}}{\partial \mathbf{B}} + \gamma(\mathbf{U} - \mathbf{B}). \end{aligned} \tag{9}$$

In forward pass, the continuous feature is optimally quantized to half-half distributed binary codes. In backward pass, the proposed proxy derivative can automatically encourage the continuous feature distribution to align with the ideal half-half distributed distribution.

4 Experiments

Datasets. • *Flickr25k* (Huiskes and Lew 2008) contains 25k images categorized into 24 classes. Each image is annotated with at least one label. Following (Yang et al. 2019), 2,000 random images are queries and from the remaining images 5,000 random images are training set.

• *Nus-wide* (Chua et al. 2009) has around 270k images with 81 classes. To fairly compare with other methods, we consider two versions. Nus-wide(I), following (Shen et al. 2018), uses the 21 most frequent classes for evaluation. Per class, 100 random images form the query set and the remaining images form the retrieval database and training set. Nus-wide(II), following (Yang et al. 2019), uses the 10 most popular classes where 5,000 random images form the test set, and the remaining images are the retrieval set. From the retrieval set, 10,500 images are randomly selected as the training set.

• *Cifar-10* (Krizhevsky and Hinton 2009) consists of 60k color images categorized into 10 classes. In the literature there are also two experimental settings. In Cifar-10(I), following (Su et al. 2018), 1k images per class (10k images in total) form the test query set, and the remaining 50k images are used for training. For Cifar-10(II), following (Yang et al. 2019), randomly selects 1,000 images per class as queries and 500 as training images, and the retrieval set has all images except for the query set.

• *Mscoco* (Lin et al. 2014b) is a dataset for multiple tasks. We use the pruned set as (Cao et al. 2017) with 12,2218 images from 80 categories. We randomly select 5,000 images as queries with the rest used as database, from which 10,000 images are chosen for training.

• *Mnist* (LeCun et al. 1998) contains 70k gray-scale 28×28 images of hand written digits from “0” to “9” across 10 classes. 1,000 images per class are randomly selected as queries and the remaining images as training set and database.

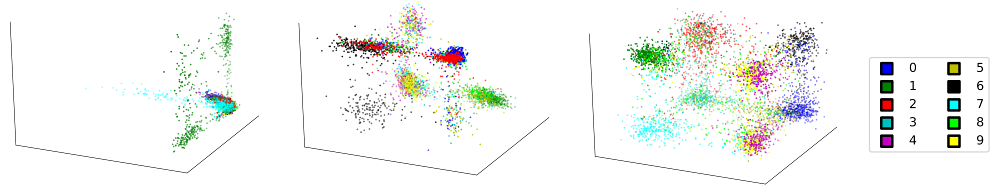
• *Ucf-101* (Soomro, Zamir, and Shah 2012) contains 13,320 action instances from 101 human action classes. All the videos are downloaded from YouTube. The average duration per video is about 7 seconds.

• *Hmdb-51* (Kuehne et al. 2011) includes 6,766 videos from 51 human action categories. The average duration of each video is about 3 seconds. For both Ucf-101 and Hmdb-51 datasets, we use the provided split 1, where per class 30% of the videos are used for testing and the rest 70% for training and retrieval.

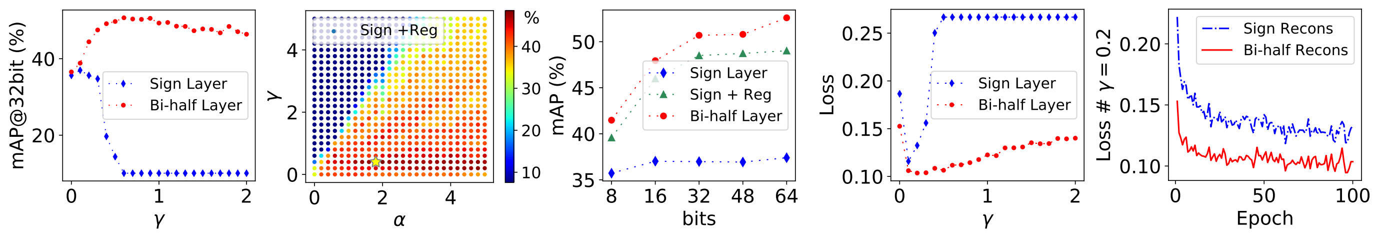
Implementation details. Our code is available online[2].

• *Image setup.* For Mnist image dataset, we train an AutoEncoder from scratch. The details will be described in the corresponding subsection. For other image datasets, an ImageNet pre-trained VGG-16 (Simonyan and Zisserman 2015) is used as our backbone where following (Su et al. 2018; Shen et al. 2018; Yang et al. 2019) an additional fc layer is used for dimensionality reduction. Our bi-half layer is appended to generate the binary codes. During training, we use Stochastic Gradient Descent(SGD) as the optimizer with a momentum of 0.9 and a weight decay of 5×10^{-4} and a

[2]https://github.com/liyunqianggyn/Deep-Unsupervised-Image-Hashing



(a) Sign Layer (b) Sign + Reg (c) Bi-half Layer



(d) Retrieval Results (e) Reconstruction Loss

Figure 3: Train an AutoEncoder from scratch on Mnist dataset. The top row (a, b, c) visualizes the continuous feature distributions before binarization over different methods by training the network with 3 hash bits. (d) shows the corresponding retrieval results. We compare bi-half layer with sign layer and sign+reg. In specific, sign+reg uses an additional entropy regularization term to optimize entropy, while it is hard to balance the added term. (e) shows the reconstruction loss curves for sign layer and bi-half layer. Generating informative binary codes in latent space can help to do reconstruction.

batch size of 32. In all experiments, the initial learning rate is set as 0.0001 and we divide the learning rate by 10 when the loss stop decreasing. The hyper-parameters γ is tuned by cross-validation on training set and set as $\gamma = 3 \times \frac{1}{N \cdot K}$.

• *Video setup.* Two 3D CNNs pre-trained on kinetics (Kay et al. 2017), ResNet-34 (Hara, Kataoka, and Satoh 2017) and ResNet-101 (Hara, Kataoka, and Satoh 2018), are used as backbones where we append bi-half layer to replace the last fc layer. Following the setting of (Hara, Kataoka, and Satoh 2018), we use SGD as optimizer with a momentum of 0.9 and a weight decay of 0.001. The learning rate starts from 0.1, and is divided by 10 after the validation loss saturates.

Evaluation metrics. We adopt semantic similarity (by labels) as evaluation ground truth, which is widely used in the unsupervised hashing literature, for instance, AGH (Liu et al. 2011), SADH (Shen et al. 2018) and DeepBit (Lin et al. 2016). Specifically, for multi-label datasets Flickr25k, Nus-wide and Mscoco, the true neighbors are defined based on whether two images share at least one common label. We measure the performance of compared methods based on the standard evaluation metrics: Mean Average Precision (mAP), Precision-Recall curves (PR) and TopN-precision curves with top N returned samples. In our experiments, N is set to 5,000.

4.1 Training an AutoEncoder from scratch

Our bi-half layer can embedded into current deep architectures to learn binary codes from scratch. We train an AutoEncoder with a deep encoder and decoder on Mnist datasets where encoder and decoder consist of two fc layers. We append our bi-half layer after encoder to generate binary code. The reconstruction loss is used as cost function. **We compare to using the sign layer and to adding an additional entropy regularization term in the loss**. For this baseline, as in (Shen et al. 2018; Erin Liong et al. 2015), we use $\mathbf{B}^\intercal\mathbf{1}$ as regularization term balanced with the *BCE* reconstruction loss through a hyper-parameter α.

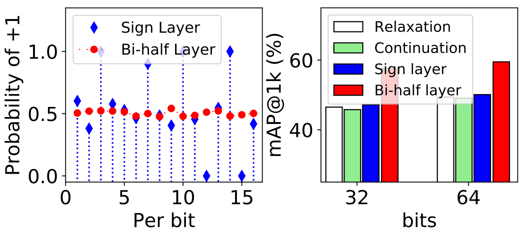
In the top row of Fig. 3 we train the network with 3 bits and visualize the distributions of the continuous feature $\mathbf{U}$ over 5,000 images. We observe that the features learned by sign layer are seriously tangled with each other. With binarization, most images will be scattered to same binary vertex and thus some bits have no discriminative information. By adding an entropy regularization term, the feature tanglement can be mitigated, but it is suboptimal solution which requires careful hyper-parameter tuning. The proposed bi-half layer can learn evenly distributed features.

Fig. 3 (d) shows the retrieval performance where the left two subfigures show the effect of hyper-parameters γ in Eq. (9) and α of term $\mathbf{B}^\intercal\mathbf{1}$. with code length 32 and the right one presents the mAP over different code lengths. Tuning the parameters can effectively improve the performance. For Sign+Reg method, it is a suboptimal solution in optimizing information entropy in comparison with bi-half layer, which can be further demonstrated in the right subfigure of Fig. 3 (d). The reconstruction loss for sign layer and bi-half layer is shown in Fig. 3 (e). We observe that generating informative binary codes in latent space can effectively minimize the reconstruction loss.

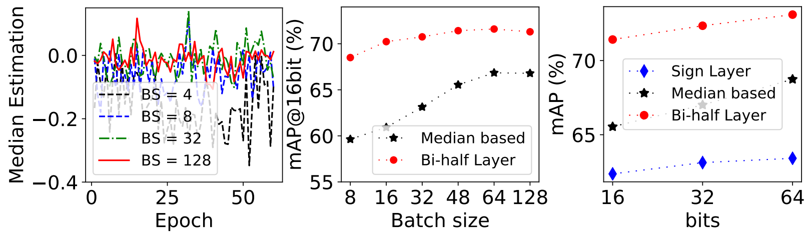
4.2 Empirical analysis

For the pre-trained models, we follow the unsupervised setting in (Su et al. 2018) and use $||cos(\mathbf{a}_1, \mathbf{a}_2) - cos(\mathbf{b}_1, \mathbf{b}_2)||_2^2$ as cost function to minimize the difference on the cosine distance relationships, where $\mathbf{a}$ means the continuous feature extracted from the last layer of a pre-trained network of one sample while $\mathbf{b}$ means the corresponding binary code.

How are the continuous features distributed? In Fig. 5 we train the network on Cifar-10(I) with 4 bits and visualize the histogram distributions of each dimension in the continuous encoded feature $\mathbf{U}$ over all images. The sign layer (Cao et al. 2017; Su et al. 2018) does not match an ideal half-half distribution whereas our bi-half method does a better approximation.



(a) left two subfigures on Cifar-10

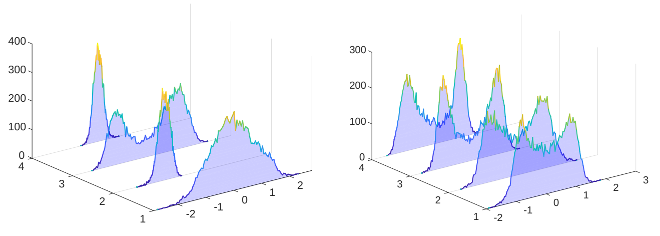


(b) right three subfigures on Flickr25k

Figure 4: Empirical analysis on Cifar-10 and Flickr25k datasets. (a) Our bi-half layer can generate informative hash bits and outperforms other coding methods; (b) the alternative median based method performs worse than bi-half layer.

Method	Cifar-10(I)			Nus-wide(I)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
DeepBit	19.40	24.90	27.70	39.22	40.32	42.06
SAH	41.80	45.60	47.40	–	–	–
SADH	–	–	–	60.14	57.99	56.33
HashGAN	44.70	46.30	48.10	–	–	–
GreedyHash*	44.80	47.20	50.10	55.49	57.47	60.93
Ours	**56.10**	**57.60**	**59.50**	**65.12**	**66.31**	**67.26**

Table 1: mAP@1000 results on Cifar-10(I) and mAP@All results on Nus-wide(I). The * denotes that we run the experiments with the released code by the authors.



Sign layer Bi-half layer

Figure 5: Comparing the distribution of continuous feature $\mathbf{U}$ for training 4 bits with a sign layer (left) versus our bi-half layer (right) over all images in Cifar-10. The y-axis shows each of the 4 bit dimensions; the x-axis shows the continuous values in $\mathbf{U}$; the z-axis presents how many images contain such a continuous value (binned). In contrast to the sign layer, our bi-half method approximates the ideal half-half distribution.

How are individual hashing bits distributed? In the left subfigure of Fig. 4 (a) we show the per-bit probability of code $+1$ over all images for 16 bits. Cifar-10(I) dataset is used to generate hash codes. The sign layer gives a non-uniformly distribution, and even for some bits the probability is completely zero or completely one: Those bits never change their value in the entire dataset and can thus safely be discarded. In contrast, our bi-half method approximates a uniform distribution, making good use of full bit capacity.

Other hash coding strategies. The right subfigure of Fig. 4 (a) shows the comparison between our bi-half coding method and three other hash coding strategies: continuous relaxation layer (Cao et al. 2018; Liu et al. 2016) ($\mathbf{B} \rightarrow \mathbf{U}$), smoothed continuation layer (Cao et al. 2017; Lai et al. 2015a) ($\mathbf{B} \rightarrow \tanh(\beta\mathbf{U})$) and sign layer (Su et al. 2018) ($\mathrm{sign}(\mathbf{U})$), respectively. Both 32 and 64 bits are used to generate hash codes on the Cifar-10(I) dataset. From the results, we see that the sign layer method slightly outperforms the other two coding methods which is consistent with (Su et al. 2018). This may be because the sign layer method can effectively keep the discrete constraint in comparison with other two methods. Our bi-half method outperforms other methods for both code sizes.

An alternative variant of bi-half layer. An alternative variant of the bi-half layer is to learn a translation term t added to the sign function $\mathrm{sign}(\mathbf{u}+t)$ for each hash bit to get half-half distributed binary codes. We estimate the median statistic over mini-batches to implement this idea. Specifically, we keep an exponential moving average (EMA) of median points over each mini-batch which is used during inference. We conduct the comparison on Flickr25k dataset in Fig. 4 (b). The left subfigure of Fig. 4 (b) shows how the EMA estimation of median changes with the training epochs over different batch sizes. We adopt the linear learning rate scaling rule (Goyal et al. 2017; Krizhevsky 2014) to adapt to batch size. We note that smaller batch size makes the estimation value unstable. The middle subfigure conducts a comparison between bi-half layer method and median translation method over different batch sizes on using 16 bits. Increasing batch size can significantly improve the performance for median based method. Due to memory limitations, unfortunately, it is difficult to use very large batch sizes. The left subfigure of Fig. 4 (b) shows the comparison with greedy hash (sign layer) and the median-based method with code length 16, 32 and 64. As expected, adding median term increases the sign layer baseline and bi-half layer significantly outperforms median-based approach.

4.3 Comparison with state-of-the art

We compare our method with previous unsupervised hashing methods, including seven shallow unsupervised hashing methods, *i.e.* LSH (Datar and Indyk 2004), SH (Weiss, Torralba, and Fergus 2008), PCAH, ITQ (Gong and Lazebnik 2011), SGH (Jiang and Li 2015), and eight deep unsupervised hashing methods, *i.e.* DeepBit (Lin et al. 2016), SGH (Dai et al. 2017), SSDH (Yang et al. 2018), DistillHash (Yang et al. 2019), SAH (Do et al. 2017), HashGAN (Ghasedi Dizaji et al. 2018), SADH (Shen et al. 2018), and GreedyHash (Su et al. 2018). To have a fair comparison, we adopt the deep features for all shallow architecture-based baseline methods. For GreedyHash, we run the experiments with the released code by the authors. For other methods, the results are taken from the related literatures.

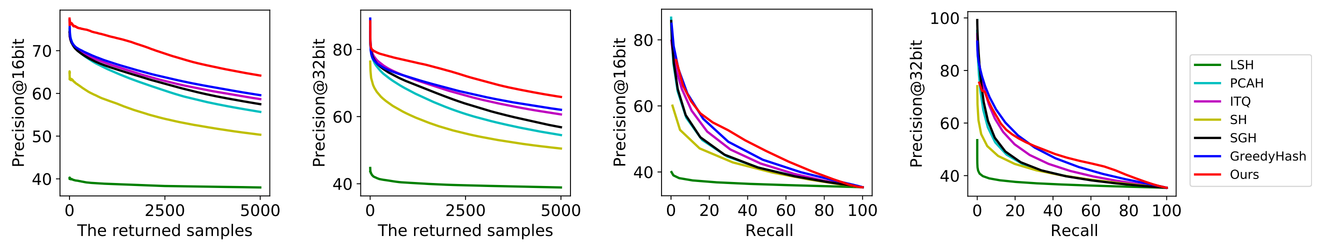


Figure 6: Top N precision and precision-recall curves on Mscoco. The proposed bi-half layer performs best.

Method	Flickr25k			Nus-wide(II)			Cifar-10(II)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH + VGG (Datar and Indyk 2004)	58.31	58.85	59.33	43.24	44.11	44.33	13.19	15.80	16.73
SH + VGG (Weiss, Torralba, and Fergus 2008)	59.19	59.23	60.16	44.58	45.37	49.26	16.05	15.83	15.09
ITQ + VGG (Gong and Lazebnik 2011)	61.92	63.18	63.46	52.83	53.23	53.19	19.42	20.86	21.51
DeepBit (Lin et al. 2016)	59.34	59.33	61.99	45.42	46.25	47.62	22.04	24.10	25.21
SGH (Dai et al. 2017)	61.62	62.83	62.53	49.36	48.29	48.65	17.95	18.27	18.89
SSDH (Yang et al. 2018)	66.21	67.33	67.32	62.31	62.94	63.21	25.68	25.60	25.87
DistillHash (Yang et al. 2019)	69.64	70.56	70.75	66.67	67.52	67.69	28.44	28.53	28.67
GreedyHash* (Su et al. 2018)	62.36	63.12	63.41	51.39	55.80	59.27	28.71	31.72	35.47
Ours	**71.42**	**72.35**	**73.10**	**67.12**	**68.05**	**68.21**	**42.87**	**43.29**	**44.13**

Table 2: mAP@All for various methods on three Flickr25k, Nus-wide(II) and Cifar-10(II) datasets. Our method with 16 bits outperforms others that use 64 bits.

Table 1 shows the mAP@1000 results on Cifar-10(I) and mAP@All results on Nus-wide(I) over three different hash code sizes 16, 32 and 64. The compared greedy hash (Su et al. 2018) method which directly uses the sign function as hash coding layer outperforms everything except our method for all code sizes. Greedy hash (Su et al. 2018) is effective to solve the vanishing gradient problem and maintain the discrete constraint in hash learning, but it cannot maximize hash bit capacity. In contrast, our method does maximize hash bit capacity and clearly outperforms all other methods on this two datasets.

In Table 2, we present the mAP results on three datasets Flickr25k, Nuswide(II), and Cifar-10(II), with hash code length varying from 16 to 64. The experiments are conducted with the same setting as in the compared methods. We do best for all hash bits sizes for all three datasets.

In Fig. 6, we conduct experiments on more challenging Mscoco dataset. The left two subfigures present the TopN-precision curves with code lengths 16 and 32. Consistent with mAP results, we can observe that our method performs best. Both mAP and TopN-precision curves are Hamming ranking based metrics where our method can achieve superior performance. Moreover, we plot the precision-recall curves for all methods with hash bit lengths of 16 and 32 in the right two subfigures Fig. 6 to illustrate the hash lookup results. From the results, we can again observe that our method consistently achieves the best results among all approaches, which further demonstrates the superiority of our proposed method.

Hashing is about compact storage and fast retrieval, thus we analyze using fewer bits in Table 1, Table 2 and Fig. 6. Only for Nus-wide(II) we perform on par while in all other datasets our method using 16 bits clearly outperforms other methods using 64 bits. This shows a 3 times reduction in storage and speed while even improving accuracy.

Backbone	Method	Ucf-101			Hmdb-51		
		16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
ResNet-34	GreedyHash*	45.49	57.24	64.77	30.32	37.55	40.53
	Ours	50.83	60.30	65.89	34.21	38.67	41.74
ResNet-101	GreedyHash*	39.29	58.35	67.23	27.60	39.96	42.07
	Ours	**59.30**	**66.13**	**68.47**	**36.68**	**41.48**	**43.03**

Table 3: mAP@100 results on two video datasets using kinetics pre-trained 3D ResNet-34 and 3D ResNet-101. The * denotes we run the experiments with the released code.

Video Retrieval Results: In Table 3, we present the mAP@100 results for Ucf-101 and Hmdb-51 datasets with code length 16, 32 and 64. For both datasets and both ResNet models our bi-half method consistently outperforms the sign layer method (Su et al. 2018) over all hash bit length, especially for short bits. In hashing, fewer bits is essential to save storage and compute.

5 Conclusion

We propose a new parameter-free Bi-half Net for unsupervised hashing learning by optimizing bit entropy. Our Bi-half layer has no hyper-parameters and compares favorably to minimizing bit entropy with an additional hyper-parameter in the loss. The designed bi-half layer can be easily embedded into current deep architectures, such as AutoEncoders, to automatically generate higher quality binary codes. The proposed proxy derivative in backward pass can effectively encourage the continuous feature distribution to align with the ideal half-half distributed distribution. One limitation is that the independence between different bits is not considered, which will be investigated in future work. Experiments on 7 datasets show state of the art results. We often outperform other hashing methods that use 64 bits where we need only 16 bits.

References

Belghazi, M. I.; Baratin, A.; Rajeswar, S.; Ozair, S.; Bengio, Y.; Courville, A.; and Hjelm, R. D. 2018. Mine: mutual information neural estimation. *ICML.*

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *Technical Report*.

Cao, Z.; Long, M.; Wang, J.; and Yu, P. S. 2017. Hashnet: Deep learning to hash by continuation. In *ICCV*.

Cao, Y.; Long, M.; Bin, L.; and Wang, J. 2018. Deep cauchy hashing for hamming space retrieval. In *CVPR*.

Chang, S. F. 2012. Supervised hashing with kernels. In *CVPR*.

Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; and Abbeel, P. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2172–2180.

Chen, Z.; Yuan, X.; Lu, J.; Tian, Q.; and Zhou, J. 2018. Deep hashing via discrepancy minimization. In *CVPR*, 6838–6847.

Chen, J.; Cheung, W. K.; and Wang, A. 2018. Learning deep unsupervised binary codes for image retrieval. In *IJCAI*.

Chua, T.-S.; Tang, J.; Hong, R.; Li, H.; Luo, Z.; and Zheng, Y. 2009. Nus-wide: a real-world web image database from national university of singapore. In *CIVR*.

Cover, T. M., and Thomas, J. A. 2012. *Elements of information theory*. John Wiley & Sons.

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

Datar, M., and Indyk, P. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the ACM Symposium on Computational Geometry*, 253–262.

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

Erin Liong, V.; Lu, J.; Wang, G.; Moulin, P.; and Zhou, J. 2015. Deep hashing for compact binary codes learning. In *CVPR*, 2475–2483.

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

Gionis, A.; Indyk, P.; and Motwani, R. 2000. Similarity search in high dimensions via hashing. In *Proceedings of International Conference on Very Large Databases*, 518–529.

Gong, Y., and Lazebnik, S. 2011. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*.

Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

Hara, K.; Kataoka, H.; and Satoh, Y. 2017. Learning spatio-temporal features with 3d residual networks for action recognition. In *ICCV*, 3154–3160.

Hara, K.; Kataoka, H.; and Satoh, Y. 2018. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *CVPR*, 6546–6555.

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

He, K.; Wen, F.; and Sun, J. 2013. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *CVPR*.

Huiskes, M. J., and Lew, M. S. 2008. The mir flickr retrieval evaluation. In *ACM Multimedia*, 39–43.

Jegou, H.; Douze, M.; and Schmid, C. 2008. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, 304–317. Springer.

Jiang, Q.-Y., and Li, W.-J. 2015. Scalable graph hashing with feature transformation. In *IJCAI*.

Jiang, Q.-Y., and Li, W.-J. 2017. Deep cross-modal hashing. In *CVPR*.

Jolliffe, I. T. 2002. Principal component analysis. *Journal of Marketing Research* 87(100):513.

Kay, W.; Carreira, J.; Simonyan, K.; Zhang, B.; Hillier, C.; Vijayanarasimhan, S.; Viola, F.; Green, T.; Back, T.; Natsev, P.; et al. 2017. The kinetics human action video dataset. *CORR*.

Kong, W., and Li, W. J. 2012. Isotropic hashing. In *NIPS*.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

Kuehne, H.; Jhuang, H.; Garrote, E.; Poggio, T.; and Serre, T. 2011. Hmdb: a large video database for human motion recognition. In *ICCV*, 2556–2563.

Kulis, B., and Grauman, K. 2009. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*.

Kulis, B.; Jain, P.; and Grauman, K. 2009. Fast similarity search for learned metrics. *IEEE TPAMI* 31(12):2143.

Lai, H.; Pan, Y.; Liu, Y.; and Yan, S. 2015a. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*.

Lai, H.; Pan, Y.; Liu, Y.; and Yan, S. 2015b. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*.

LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.; et al. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

Li, Q.; Sun, Z.; He, R.; and Tan, T. 2017. Deep supervised discrete hashing. In *NIPS*.

Li, Y.; Pei, W.; van Gemert, J.; et al. 2019. Push for quantization: Deep fisher hashing. In *BMVC*.

Lin, G.; Shen, C.; Shi, Q.; Hengel, A. V. D.; and Suter, D. 2014a. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*.

Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014b. Microsoft coco: Common objects in context. In *European conference on computer vision*, 740–755. Springer.

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

Liu, W.; Wang, J.; Kumar, S.; and Chang, S.-F. 2011. Hashing with graphs. In *ICML*.

Liu, W.; Kumar, S.; Kumar, S.; and Chang, S. F. 2014. Discrete graph hashing. In *NIPS*.

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

Liu, H.; Wang, R.; Shan, S.; and Chen, X. 2017. Learning multifunctional binary codes for both category and attribute oriented retrieval tasks. In *CVPR*.

Lu, J.; Liong, V. E.; and Zhou, J. 2017. Deep hashing for scalable image search. *IEEE TIP* 26(5):2352–2367.

Mu, Y., and Yan, S. 2010. Non-metric locality-sensitive hashing. In *AAAI*.

Raginsky, M. 2009. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*.

Raziperchikolaei, R., and Carreira-Perpiñán, M. A. 2016. Optimizing affinity-based binary hashing using auxiliary coordinates. In *NIPS*.

Shannon, C. E. 1948. A mathematical theory of communication. *Bell system technical journal* 27(3):379–423.

Shen, F.; Shen, C.; Liu, W.; and Shen, H. T. 2015. Supervised discrete hashing. In *CVPR*.

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

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. *ICLR*.

Soomro, K.; Zamir, A. R.; and Shah, M. 2012. Ucf101: A dataset of 101 human actions classes from videos in the wild. *CORR*.

Su, S.; Zhang, C.; Han, K.; and Tian, Y. 2018. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *NIPS*. 798–807.

Villani, C. 2003. *Topics in optimal transportation*. Number 58. American Mathematical Soc.

Weiss, Y.; Torralba, A.; and Fergus, R. 2008. Spectral hashing. In *NIPS*.

Xu, B.; Bu, J.; Lin, Y.; Chen, C.; He, X.; and Cai, D. 2013. Harmonious hashing. In *IJCAI*.

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

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

Yuan, X.; Ren, L.; Lu, J.; and Zhou, J. 2018. Relaxation-free deep hashing via policy gradient. In *ECCV*, 134–150.

Zhang, D.; Wang, J.; Cai, D.; and Lu, J. 2010. Self-taught hashing for fast similarity search. In *ACM SIGIR*, 18–25.

Zhang, P.; Zhang, W.; Li, W. J.; and Guo, M. 2014. Supervised hashing with latent factor models. In *SIGIR*.

Zhao, F.; Huang, Y.; Wang, L.; and Tan, T. 2015. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*.