Modeling Cardinality in Image Hashing

Dayong Tian, Chen Gong, *Member, IEEE*, Maoguo Gong, *Senior Member, IEEE*, Yiwen Wei, *Member, IEEE*, and Xiaoxuan Feng

***Abstract*—Cardinality constraint, namely, constraining the number of nonzero outputs of models, has been widely used in structural learning. It can be used for modeling the dependencies between multidimensional labels. In hashing, the final outputs are also binary codes, which are similar to multidimensional labels. It has been validated that estimating how many 1's in a multidimensional label vector is easier than directly predicting which elements are 1 and estimating cardinality as a prior step will improve the classification performance. Hence, in this article, we incorporate cardinality constraint into the unsupervised image hashing problem. The proposed model is divided into two steps: 1) estimating the cardinalities of hashing codes and 2) then estimating which bits are 1. Unlike multidimensional labels that are known and fixed in the training phase, the hashing codes are generally learned through an iterative method and, therefore, their cardinalities are unknown and not fixed during the learning procedure. We use a neural network as a cardinality predictor and its parameters are jointly learned with the hashing code generator, which is an autoencoder in our model. The experiments demonstrate the efficiency of our proposed method.**

***Index Terms*—Approximate nearest neighbors searching, binary embedding, cardinality, image hashing.**

I. Introduction

IMAGE hashing maps the original images to binary hashing codes to boost the efficiency of approximate nearest-neighbor searching. With the dramatically increasing number of images online, hashing methods have attracted lots of research interest.

Manuscript received December 13, 2020; revised April 1, 2021; accepted June 10, 2021. This work was supported in part by the Key Laboratory Foundation of Information Perception and Systems for Public Security of MIIT (Nanjing University of Science and Technology) under Grant 202006; in part by the National Science Foundation of China under Grant 61806166 and Grant 61801362; in part by the Natural Science Foundation of Shaanxi Province under Grant 2020JQ-197; and in part by the Fundamental Research Funds for the Central Universities, NPU, under Grant G2018KY0303. This article was recommended by Associate Editor D. Tao. *(Corresponding author: Dayong Tian.)*

Dayong Tian and Xiaoxuan Feng are with the School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710071, China (e-mail: dayong.tian@nwpu.edu.cn).

Chen Gong is with the PCA Laboratory, Key Laboratory of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education, School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 20094, China.

Maoguo Gong is with the Key Laboratory of Intelligent Perception and Image Understanding, Ministry of Education, Xidian University, Xi'an 71000, China.

Yiwen Wei is with the School of Physics and Optoelectronic Engineering, Xidian University, Xi'an 71000, China.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCYB.2021.3089879.

Digital Object Identifier 10.1109/TCYB.2021.3089879

Generally, there are two kinds of categorizations for image hashing methods. They can be categorized to deep and shallow methods based on whether deep neural networks are incorporated. Due to the effectiveness on feature extraction of deep neural networks, the deep models are generally superior to shallows ones. On the other hand, based on whether annotations or labels are available, they can be categorized to supervised, unsupervised, and semisupervised methods. Supervised methods benefit from the annotations and usually achieve better performances. However, manual annotation is too laborious especially for large-scale datasets.

In this article, we are interested in unsupervised deep image hashing. We investigate image hashing in a different view. We treat hashing codes as dynamic labels during the iterative learning procedure and, hence, in each iteration, hashing codes can be treated as a multilabel classification problem. Thanks to the development in deep structured prediction, we can incorporate the advanced methods to our hashing method. Specifically, we model cardinality [4] in hashing.

Cardinality refers to the number of nonzero elements of a vector. Applying cardinality constraint in hashing is to constrain the number of 1's of a binary hashing code. There are two main advantages to model cardinality in hashing. First, similar to cardinality constraints in multilabel classification, predicting which elements are nonzero will be easier after predicting how many nonzero elements there are [4]. Hence, we can use a two-step scheme to generate hashing codes to improve the performance, that is: 1) predicting the cardinality of hashing codes of a sample and 2) then predict which bits are 1. Second, imposing cardinality constraints on two hashing codes is equivalent to imposing an upper and a lower bounds on their Hamming distance. If two hashing codes are of similar cardinalities, they are more likely true neighbors and the possible Hamming distances between their hashing codes are in a narrow interval. That is, cardinality constraints help filter out some local optima of hashing's object function. Let two binary vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ be of cardinalities z_i and z_j. It can be proven that the Hamming distance between $\mathbf{b}_i$ and $\mathbf{b}_j$ varies within the interval $[|z_i - z_j|, \min(z_i + z_j, 2l - z_i - z_j)]$, where l is the length of $\mathbf{b}_i$ and $\mathbf{b}_j$ (Appendix A). From this view, our two-step scheme can also be treated as a coarse-to-fine scheme. Estimating the cardinality is an interval estimation for hashing codes, which is a coarse estimation. The second step is a point estimation, which generates hashing codes.

The overall flowchart of our method is shown in Fig. 1. In the training phase, we train a convolutional autoencoder to generate codes in real for images. The hashing codes are generated by maximizing the covariance between real codes

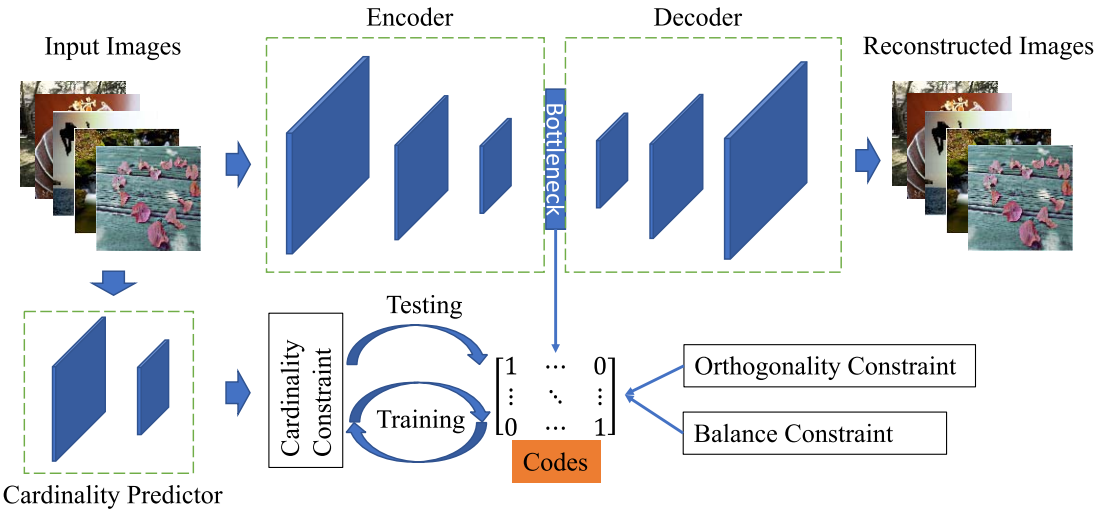


Fig. 1. Flowchart of our proposed method. In the training phase, the autoencoder generates the real codes for input images. The orthogonality, balance, and cardinality constraints are used to generate hashing codes. In the testing phase, the cardinalities of queries are first predicted and then the hashing codes are generated by an encoder with the cardinality constraint.

and hashing codes. Besides, the classical balance and orthogonality constraints [35] are used to regularize the code matrix. To model cardinality, we train a separate convolutional neural network (CNN) to predict the cardinality of the corresponding hashing code of an input image. Note that the labels for training the cardinality predictor, that is, cardinalities of hashing codes, are varying during the training process. The cardinality predictor maps images to cardinalities of their hashing codes. On the other hand, the outputs of the cardinality predictor are used to "supervise" the learning of hashing codes. Since similar images should have similar hashing codes and hence, similar cardinalities, the outputs of the cardinality predictor should be similar for similar images. In the case that hashing codes of two similar images are too dissimilar, outputs of the cardinality predictor will try to make their hashing codes have similar cardinalities and hence, constrain the bounds of their Hamming distances. In the testing phase, the cardinality of a query image' hashing code is first predicted. Then, the real codes are generated by the convolutional autoencoder. Finally, the hashing code is generated based on the real codes and cardinality.

The remainder of this article is organized as follows. In Section II, related works are briefly reviewed. The preliminaries are given in Section III. The formulation of our models is shown in Section IV. In Section VI, we report the experimental results of our method. The conclusive remarks are given in Section VII.

II. Related Works

Based on whether label information is available, hashing methods can be categorized into supervised [23], unsupervised, and semisupervised ones [33]. The unsupervised methods can be further divided into data-dependent and data-independent categories.

The locality-sensitive hashing (LSH) [2] method is one of the most popular data-independent hashing methods. To improve the performance of LSH, cosine similarity [5] and kernel similarity [16] can be adopted. Due to the lack of data information, these methods are generally inferior to data-dependent ones.

Data-dependent methods learn hashing functions on training data. It aims to maximize the correlation between structures of training data and hashing codes. Most data-dependent hashing methods relax the binary constraints on hashing codes to avoid an NP-hard problem. The intermediate is a code matrix in real and hashing codes are generated by thresholding it. Spectral hashing (SH) [35], one of the earliest data-dependent methods, models the hashing problem as minimizing the average Hamming distance between similar neighbors. SH generates hashing codes by solving the relaxing mathematical problem to circumvent the computation of pairwise distances in the entire dataset, that is, the affinity matrix, which is required to compute the average Hamming distance.

Anchor graph hashing (AGH) [20] approximates the true affinity matrix by a highly sparse affinity matrix using anchor points. Discrete graph hashing (DGH) [19] follows this idea and incorporates the orthogonality constraint of a code matrix in the training procedure.

Iterative quantization (ITQ) [38] rotates the principal components to minimize the quantization errors. There are some variants of ITQ based on principal component analysis (PCA) [13], [14] and linear discriminant analysis [30]. Unlike ITQ that precomputes the principal components, which are projections of original data on principal component coefficients, neighborhood discriminant hashing (NDH) [31] computes the projections during the optimization procedure.

In general, linear dimension reduction methods, such as PCA, are inferior to nonlinear embeddings. Inductive manifold hashing (IMH) [26], [27] learns a nonlinear manifold on a small subset and inductively inserts the remaining data. Since the orthogonality and balance constraints were proposed in [35], they have never been simultaneously fulfilled. Theoretically, DGH can generate an orthogonal and balanced code matrix by setting a parameter to infinity.

However, it is impractical to optimize an object function with infinite parameters, so the authors preset the parameter to a positive constant. Most methods focus on the orthogonality of the real intermediates and hope that quantizing the intermediates will not break the orthogonality. Nevertheless, Tian *et al.* [32] demonstrated that quantization will only preserve the orthogonality of intermediates in some extremely ideal cases.

Matrix factorization is another promising technique for hashing. Ding *et al.* [9] used collective matrix factorization for multimodal hashing. Lu *et al.* [22] used matrix decomposition to extract latent semantic features for generating discriminative binary codes. Zhu *et al.* [41] proposed a topic hypergraph hashing method by exploiting auxiliary texts around images. Liu *et al.* [18] noticed the sparsity of data structure and proposed an adaptively sparse matrix factorization for hashing. Most methods try to solve a continuous optimization problem by relaxing the binary constraint. Wang *et al.* [34] transformed the discrete optimization problem to a continuous one to circumvent quantization errors. He *et al.* [10] proposed a bidirectional discrete matrix factorization hashing method, which simultaneously learns codes from data and recovers data from codes.

Besides the above-mentioned shallow hashing methods, deep neural networks have been used in hashing models. Deep transfer hashing (DTH) [39] substitutes the principal coefficients and orthogonal rotation matrix in ITQ with a deep neural network. Deep binary descriptors (DeepBit) [17] uses VGGNet [29] to extract the features of images and learns the hashing codes with a combined object function of quantization loss, balanced regularization, and rotation invariant objective. Stochastic generative hashing (SGH) [7] learns hashing codes by the minimum description length principle so as to maximally compress the dataset as well as regenerate outputs from the codes. Liu *et al.* generated pseudolabels for a self-taught hashing algorithm. Semantic structure-based unsupervised hashing (SSDH) [36] uses two half Gaussian distributions to estimate pairwise cosine distances of data points and assign any two data points with obviously smaller distance as semantically similar pair. A pairwise loss function to preserve this semantic structure is used to train the neural network. DistillHash [37] learns confidence similarity signals first to supervise the subsequent hashing code generating. Lu *et al.* [21] integrated the quantization process and ranking process into a unified architecture. Shen *et al.* [28] found that graphs built from original data introduce biased prior knowledge of data relevance and therefore, they proposed a twin-bottleneck autoencoder to trace the code-driven similarity graph. Graph convolutional network hashing [40] introduces an intuitive asymmetric graph convolutional layer to avoid using affinity graph as the only learning guidance.

III. Preliminaries

We consider the setting of embedding the data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ to a hashing code matrix $\mathbf{B} \in \{0, 1\}^{n\times l}$, where n is the number of data points, d is the dimension of data, and l is the code length. We use the lower case $\mathbf{x}_i$ and $\mathbf{b}_i$ to represent the ith row of $\mathbf{X}$ and $\mathbf{B}$, that is, the ith data point and its corresponding hashing code, respectively.

Our model can be formulated as minimization of an energy function

$$\begin{aligned}\mathbf{B} = \arg\min_{\mathbf{B}} \ & E(\mathbf{X}, \mathbf{B}) \\ \text{s.t.} \ & \mathbf{B} \in \{0, 1\}.\end{aligned} \tag{1}$$

We decompose (1) into three components

$$E(\mathbf{X}, \mathbf{B}) = h(\mathbf{X}, \mathbf{B}) + p(\mathbf{B}) + \sum_{i=1}^{n} c_{g(\mathbf{x}_i)}(\mathbf{b}_i) \tag{2}$$

where $h(\mathbf{X}, \mathbf{B})$ is the hashing function that embeds an input data matrix $\mathbf{X}$ to the binary hashing code matrix $\mathbf{B}$, $p(\mathbf{B})$ is a function modeling the prior knowledge for $\mathbf{B}$, $z = g(\mathbf{x}_i)$ is the cardinality for the ith data point, and $c_{g(\mathbf{x}_i)}(\mathbf{b}_i)$ is the cardinality potential, which constrains $\mathbf{b}_i$ to be of cardinality $g(\mathbf{x}_i)$.

Although our formulation is inspired by that proposed in [4] and somewhat similar to its formulation, they are completely different in meanings and hence, their optimization procedures are also different. In [4], the formulation is given by

$$\hat{\mathbf{y}}_i = \arg\max_{\mathbf{y}} s(\mathbf{x}_i, \mathbf{y}_i) \tag{3}$$

where $\mathbf{y}_i$ is the label vector for $\mathbf{x}_i$ and $s(\mathbf{x}_i, \mathbf{y}_i)$ is a score function. Then, the authors decompose $s(\mathbf{x}_i, \mathbf{y}_i)$ as follows:

$$s(\mathbf{x}_i, \mathbf{y}_i) = \sum_{j} s_i\big(\mathbf{x}_i, y_{ij}\big) + s_p(\mathbf{y}_i) + s_z(\mathbf{y}_i) \tag{4}$$

where y_{ij} is the jth element of label vector $\mathbf{y}_i$, s_i aims to measure the score of assigning $\mathbf{y}_i$ to $\mathbf{x}_i$, s_g is a global potential, which is modeled by a simple neural network, and s_z is similar to our modeling of cardinality constraints except that their constraints are applied on labels while our constraints are applied on hashing codes.

The key difference of label $\mathbf{y}_i$ and $\mathbf{b}_i$ is in the training phase. $\mathbf{y}_i$ is known and hence, it is a supervised learning, while $\mathbf{b}_i$ is unknown and hence, our method is unsupervised. As $\mathbf{y}_i$ is known in the training phase, the global potential s_p can be learned. s_p act as a prior knowledge on the distribution of $\mathbf{y}_i$. However, we must manually design the prior knowledge $p(\mathbf{B})$ in our model. The cardinality of hashing code $\mathbf{b}_i$ is varying during the optimization procedure in the training phase while the cardinality of label $\mathbf{y}_i$ is a fixed number in the training phase.

IV. Formulation

In this section, we will explain our models in detail. There are three main components of our models. The hashing component is modeled by a deep convolutional autoencoder. The bottleneck of the autoencoder is used as real codes for images. For an unsupervised hashing model, we incorporate some prior knowledge on hashing codes as another component of our model, that is, the famous orthogonality and balance constraints [35]. Finally, we model cardinality as a component in our model.

TABLE I
STRUCTURE OF THE AUTOENCODER

Encoder				
Layer	Output Dim.	Kernel	Batch Norm.	Activation
Input	(3,64,64)	NA	NA	NA
Conv2d	(64,32,32)	(4,4)	True	ReLU
Conv2d	(128,16,16)	(4,4)	True	ReLU
Conv2d	(256,8,8)	(4,4)	True	ReLU
Conv2d	(512,4,4)	(4,4)	True	ReLU
Linear	8192	NA	NA	ReLU
Linear	1024	NA	NA	ReLU
Linear	l	NA	NA	NA
Decoder				
Layer	Output Dim.	Kernel	Batch Norm.	Activation
Linear	1024	NA	NA	ReLU
Linear	8192	NA	NA	ReLU
ConvTranspose2d	(256,8,8)	(4,4)	True	ReLU
ConvTranspose2d	(128,16,16)	(4,4)	True	ReLU
ConvTranspose2d	(64,32,32)	(4,4)	True	ReLU
ConvTranspose2d	(3,64,64)	(4,4)	True	tanh

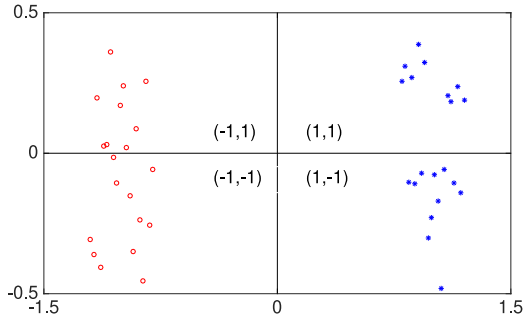


Fig. 2. Drawback of directly quantizing bottleneck outputs. The data of the same class are quantized into two different hashing codes. ITQ rotates the data to make data of the same class locate in the same quadrant.

A. Hashing Function

We use a convolutional autoencoder to embed the images into real vectors. The structure of the autoencoder is given in Table I. Quantizing the output of the bottleneck of the autoencoder is an intuitive and widely used way to generate binary codes. As illustrated in Fig. 2, such quantization may greatly break the structure of learned codes.

There are several ways to solve this problem. For example, ITQ [38] reduces the dimension of the zero-centered data matrix by PCA and rotates the principal components using an orthogonal matrix.

In this article, we try to directly maximize the covariance between $\mathbf{v}_i$ and $\mathbf{b}_i$, where $\mathbf{v}_i$ is the output of the bottleneck of the autoencoder for $\mathbf{x}_i$. Mathematically, our hashing model can be formulated as

$$\begin{aligned} \arg\max_{\mathbf{B}} \ & tr\Big(\big(\mathbf{V}-\bar{\mathbf{V}}\big)\big(\mathbf{B}-\bar{\mathbf{B}}\big)^\top\Big) \\ \text{s.t.} \ & \mathbf{B}\in\{0,1\}^{n\times c} \end{aligned} \tag{5}$$

where $\bar{\mathbf{V}}$ and $\bar{\mathbf{B}}$ are the column means of $\mathbf{V}$ and $\mathbf{B}$, respectively. $\mathbf{v}_i$ is the ith row of $\mathbf{V}$. The object function of the autoencoder is

$$\arg\min_{\Theta} \|\mathbf{X}-f(\mathbf{X};\Theta)\|_F^2 \tag{6}$$

where $f(\mathbf{X};\Theta)$ is the function of the autoencoder and Θ is the set of its weights and biases. We subtract (6) by (5) to formulate h

$$h(\mathbf{X},\mathbf{B}) = \|\mathbf{X}-f(\mathbf{X};\Theta)\|_F^2 - \lambda_1 tr\Big(\big(\mathbf{V}-\bar{\mathbf{V}}\big)\big(\mathbf{B}-\bar{\mathbf{B}}\big)^\top\Big) \tag{7}$$

where λ_1 is a positive constant tuned in the experiments. The binary constraints on $\mathbf{B}$ will be also applied on the optimization problem (2). In Section V, we will show how to minimize (2).

B. Prior Knowledge on Hashing Codes

For unsupervised image hashing, orthogonality and balance constraints [35] are widely used as prior knowledge on hashing codes. The orthogonality constrains the code matrix to be orthogonal, that is

$$\mathbf{B}'^\top\mathbf{B}' = n\mathbf{I} \tag{8}$$

where $\mathbf{I}$ is a $d\times d$ identity matrix and $\mathbf{B}' = (\mathbf{B}-0.5)\times 2$. The balance constraints require each column of $\mathbf{B}'$ to have the same number of -1 and 1, or equivalently the same number of 0 and 1 in $\mathbf{B}$, that is,

$$\mathbf{1}^{1\times n}\mathbf{B}' = \mathbf{0}^{1\times l} \tag{9}$$

or equivalently

$$\mathbf{1}^{1\times n}\mathbf{B} = \frac{n}{2}\times\mathbf{1}^{1\times l}. \tag{10}$$

For simplicity, when the dimensions of $\mathbf{1}$, $\mathbf{0}$, and $\mathbf{I}$ can be determined by the context, we use them without explicitly declaring their dimensions. For example, given $\mathbf{X}$ is an $n\times p$ matrix, in $\mathbf{1}^\top\mathbf{X}$, $\mathbf{1}$ is a column vector of length n. Vectors are defaulted to be column vectors in this article.

Due to binary constraints on $\mathbf{B}$, these two constraints are generally intractable. Inspired by [19], we define a set $\Omega = \{\mathbf{Y}\in\mathbb{R}^{n\times l} | \mathbf{1}^\top\mathbf{Y} = \mathbf{0}, \mathbf{Y}^\top\mathbf{Y} = n\mathbf{I}\}$. Let Π_Ω denotes an approximate projection operator, which computes an approximation of a Euclidean projection onto Ω in a differentiable way. $\Pi_\Omega(\mathbf{B}')$ approximately solve the following problem:

$$\begin{aligned} \min_{\mathbf{Y}} \ & \big\|\mathbf{B}'-\mathbf{Y}\big\|_F^2 \\ \text{s.t.} \ & \mathbf{Y}\in\Omega \end{aligned} \tag{11}$$

or equivalently

$$\begin{aligned} \min_{\mathbf{Y}} \ & \big\|\mathbf{B}'-\mathbf{Y}\big\|_F^2 \\ \text{s.t.} \ & \mathbf{1}^\top\mathbf{Y} = \mathbf{0}, \mathbf{Y}^\top\mathbf{Y} = n\mathbf{I}. \end{aligned} \tag{12}$$

The formulation of $p(\mathbf{B})$ in (2) is $p(\mathbf{B}) = \|2(\mathbf{B}-0.5)-\mathbf{Y}\|_F^2$ and the constraint $\mathbf{Y}\in\Omega$ should be applied on (2).

C. Learning With Cardinality Potentials

The cardinality potential consists of two steps. First, we learn a cardinality predictor $z = g(\mathbf{x}_i)$ to estimate the cardinality of $\mathbf{b}_i$ corresponding to an input $\mathbf{x}_i$. Second, we learn the hashing codes satisfying the cardinality constraint.

The cardinality predictor $z = g(\mathbf{x}_i)$ is modeled as a CNN (Table II). The fully connected neural network is used on the top of convolutional layers to generate l-D outputs. One-hot vector is used to represent the cardinality, that is, the kth element is 1 and other elements are 0 when the cardinality is k. The cardinality prediction is equivalent to classify the input

TABLE II
STRUCTURE OF CARDINALITY PREDICTOR

Layer	Output Dim.	Kernel	Batch Norm.	Activation
Input	(3,64,64)	NA	NA	NA
Conv2d	(32,32,32)	4	True	ReLU
Conv2d	(64,16,16)	4	True	ReLU
Conv2d	(128,8,8)	4	True	ReLU
Conv2d	(256,4,4)	4	True	ReLU
Linear	(4096,1024)	NA	NA	ReLU
Linear	(1024,l)	NA	NA	Softmax

images into l categories according to their hashing codes' cardinalities. Hence, cross-entropy can be used as the object function.

In the second step, we learn the hashing codes based on the cardinality. First, we define the potential s_z as

$$s_z(\mathbf{b}_i) = \begin{cases} 0, & \text{if } \mathbf{b}_i^\top \mathbf{1} = z \\ +\infty, & \text{otherwise.} \end{cases} \tag{13}$$

Let us define the differetiable approximate Euclidean projection of $\mathbf{b}_i$ on a set $\mathcal{Z} = \{\mathbf{u}_i | \forall i, u_{ij} \in [0,1], \sum_j u_{ij} = z\}$ as $\Pi_{\mathcal{Z}}(\mathbf{b}_i)$. $\Pi_{\mathcal{Z}}$ approximately solves the following problem:

$$\begin{aligned} \min_{\mathbf{u}_i, \Phi} \ & \|\mathbf{b}_i - \mathbf{u}_i\|_F^2 \\ \text{s.t.} \ & \sum_j u_{ij} = z, 0 \le u_{ij} \le 1 \quad \forall j \in 1, \ldots, l \end{aligned} \tag{14}$$

where Φ is the parameter set of the cardinality predictor. The formulation of $c_{g(\mathbf{x}_i)}(\mathbf{b}_i)$ in (2) is $c_{g(\mathbf{x}_i)}(\mathbf{b}_i) = \|\mathbf{b}_i - \mathbf{u}_i\|_F^2$ and the constraint $\mathbf{u}_i \in \mathcal{Z}$ should be applied on (2).

D. Overall Formulation

The overall formulation of our model can be written as

$$\begin{aligned} \arg\min_{\mathbf{B}, \Theta, \Phi, \mathbf{Y}, \{\mathbf{u}_i\}} \ & E = \|\mathbf{X} - f(\mathbf{X}; \Theta)\|_F^2 \\ & - \lambda_1 tr\Big(\big(\mathbf{V} - \bar{\mathbf{V}}\big)\big(\mathbf{B} - \bar{\mathbf{B}}\big)^\top\Big) \\ & + \lambda_2 \|2(\mathbf{B} - 0.5) - \mathbf{Y}\|_F^2 + \lambda_3 \sum_{i=1}^{n} \|\mathbf{b}_i - \mathbf{u}_i\|_F^2 \\ \text{s.t.} \ & \mathbf{B} \in \{0,1\}^{n \times l}, \mathbf{1}^\top \mathbf{Y} = \mathbf{0}, \mathbf{Y}^\top \mathbf{Y} = n\mathbf{I} \\ & \sum_j u_{ij} = z, 0 \le u_{ij} \le 1 \quad \forall j \in 1, \ldots, l \end{aligned} \tag{15}$$

where λ_1, λ_2, and λ_3 are all positive constants empirically tuned in the experiments. $\mathbf{V}$ is the output of the bottleneck of the autoencoder. It is not an independent variable in (15). It can be completely determined by the network parameters Θ. Φ is the parameter set of cardinality predictor.

V. OPTIMIZATION

In this section, we are going to show our algorithm to minimize the constrained problem (15). We use an alternative minimization procedure to minimize (15). To handle the constraints, we project the code matrix onto the boundaries of the constraints.

A. Update Θ and Φ

The gradient of E in (15) with respect to Θ can be easily computed by the deep learning library, such as PyTorch [24] used in our implementation. $\partial E / \partial \Theta$ is only related to the first two terms in (15). The updating of Φ is also implemented by PyTorch.

The initialization of $\mathbf{B}$ is important in our method. We tried random initialization for $\mathbf{B}$. It is difficult to converge to a good result. For updating Θ, $\mathbf{B}$ acts like labels. A random $\mathbf{B}$ brings too much erroneous information, which misleads the autoencoder. We found that when the autoencoder is trained in advance, $\mathbf{X}$ is somewhat well represented by $\mathbf{V}$ and in the later iterations, $\mathbf{V}$ will instruct $\mathbf{B}$ to a good position. This trick is easy to implement. Train the autoencoder for several epochs in advance without considering $\mathbf{B}$ and then quantize the output of bottleneck to initialize $\mathbf{B}$.

B. Update $\mathbf{Y}$

Updating $\mathbf{Y}$ is to solve the following problem:

$$\begin{aligned} \arg\min_{\mathbf{Y}} \ & \big\|\mathbf{B}' - \mathbf{Y}\big\|_F^2 \\ \text{s.t.} \ & \mathbf{Y}^\top \mathbf{Y} = \mathbf{I}, \mathbf{1}^\top \mathbf{Y} = \mathbf{0}. \end{aligned} \tag{16}$$

Note that $\mathbf{B}' = 2(\mathbf{B} - 0.5)$. We solve (16) by two subproblems. First, we project $\mathbf{B}'$ on set $\Omega_1 = \{\mathbf{Y} | \mathbf{1}^\top \mathbf{Y} = 0\}$. Then, we project the resulting projection in the first step on set $\Omega_2 = \{\mathbf{Y} | \mathbf{Y}^\top \mathbf{Y} = \mathbf{I}\}$.

It can be proven that $\Pi_{\Omega_1}(\mathbf{B}') = \mathbf{B}' - \mathbf{1}(\mathbf{1}^\top / n)\mathbf{B}'$ is the solution of the first subproblem (Appendix B). The projection on Ω_2 is equivalent to the orthogonal Procrustes problem, which has a analytic solution [25]. Using singular value decomposition (SVD), we obtain

$$\Pi_{\Omega_2}\big(\mathbf{B}'\big) = \mathbf{L}\mathbf{R}^\top \tag{17}$$

where the columns of $\mathbf{L}$ and $\mathbf{R}$ are the left and right singular vectors of $\mathbf{B}'$. The solution of projecting $\mathbf{B}'$ on the intersection of $\Omega = \Omega_1 \cap \Omega_2$ can be calculated by Dykstra's algorithm [3] (Appendix C).

C. Update $\mathbf{u}_i$

Updating $\mathbf{u}_i$ is to solve the following problem:

$$\begin{aligned} \arg\min_{\mathbf{u}_i} \ & \|\mathbf{b}_i - \mathbf{u}_i\|_F^2 \\ \text{s.t.} \ & \sum_j u_{ij} = z, 0 \le u_{ij} \le 1 \quad \forall j \in 1, \ldots, l. \end{aligned} \tag{18}$$

Following [4], we represent $\mathcal{Z}$ as an intersection of $\mathcal{Z}_1 = \{\mathbf{u}_i | \forall i, u_{ij} \le 1\}$ and $\mathcal{Z}_2 = \{\mathbf{u}_i | \forall i, u_{ij} \ge 0, \sum_j u_{ij} = z\}$, that is, $\mathcal{Z} = \mathcal{Z}_1 \cap \mathcal{Z}_2$.

The projection on $\mathcal{Z}_1$ can be obtained by $\Pi_{\mathcal{Z}_1}(\mathbf{b}_i) = \min(\mathbf{b}_i, 1)$. The projection on $\mathcal{Z}_2$ can be obtained by Algorithm 1. Again, Dykstra's algorithm is used to calculate the projection on the intersection $\mathcal{Z} = \mathcal{Z}_1 \cap \mathcal{Z}_2$.

The gradients of the cardinality prediction CNN are calculated by the deep learning library, that is, PyTorch in our implementation.

Algorithm 1 Algorithm for Projection on $\mathcal{Z}_1$

Input: vector $\mathbf{b}_i \in \mathbb{R}^l$, cardinality z.
Output: $\max(\mathbf{b}_i - \theta, 0)$.
1: Sort $\mathbf{b}_i$ into $\mu : \mu_1 \geq \cdots \geq \mu_l$.
2: Compute $\bar{\mu}$ the cumulative sum of μ.
3: Let α be a vector of indices $1, \ldots, l$.
4: $\delta = \text{Softsign}(\mu \circ \alpha - (\bar{\mu} - z))$.
5: $\rho = \text{Softmax}(\delta \circ \alpha)$.
6: $\theta = \frac{1}{\alpha^\top \rho}(\bar{\mu}^\top \rho - z)$.

Algorithm 2 Algorithm to Minimize (15)

Input: $\mathbf{X}$, c, λ_1, λ_2 and λ_3.
Output: $\mathbf{B}$.
1: Train the autoencoder (Table I) for several epochs in advance.
2: **repeat**
3: Update Θ and Φ.
4: Project $\mathbf{B}'$ on Ω_1 by $\mathbf{B}' - \frac{\mathbf{1}\mathbf{1}^\top}{n}\mathbf{B}'$.
5: Project $\mathbf{B}'$ on Ω_2 by Eq. (17)
6: Calculate the projection of $\mathbf{B}$ on $\Omega_1 \cap \Omega_2$ by Dykstra's algorithm.
7: Project $\mathbf{B}$ on $\mathcal{Z}_1$ by $\min(\mathbf{b}_i, 1)$.
8: Project $\mathbf{B}$ on $\mathcal{Z}_2$ by **Algorithm 1**.
9: Calcualte the projection of $\mathbf{B}$ on $\mathcal{Z}_1 \cap \mathcal{Z}_2$ by Dykstra's algorithm.
10: Update $\mathbf{B}$ by Eq. (20).
11: **until** Convergence

D. Update B

We update $\mathbf{B}$ by two steps. First, relax the binary constraint and update $\mathbf{B}$. Then, threshold $\mathbf{B}$ at 0.5 and calculate binary $\mathbf{B}$. $\mathbf{B}$ can be updated by setting the partial derivative of (15) with respect to $\mathbf{B}$ to 0, that is

$$\frac{\partial E}{\partial \mathbf{B}} = -\lambda_1(\mathbf{V} - \bar{\mathbf{V}})\left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top\right) + 2\lambda_2(2(\mathbf{B} - 0.5) - \mathbf{Y}) + \lambda_3(\mathbf{B} - \mathbf{U}) = 0 \quad (19)$$

where the ith row of $\mathbf{U}$ is $\mathbf{u}_i$. From (19), we can derive that

$$\mathbf{B} = \frac{1}{4\lambda_2 + \lambda_3}\left(\lambda_1(\mathbf{V} - \bar{\mathbf{V}})\left(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top\right) + 2\lambda_2(1 + \mathbf{Y}) + \lambda_3\mathbf{U}\right). \quad (20)$$

E. Overall Optimization Procedure

The overall optimization procedure to minimize (15) is shown in Algorithm 2. The parameter setting and implementation details are given in Section VI-D. In Algorithm 2, the autoencoder is trained in advance for five epochs. The termination criterion is the maximum iteration, which is set as 20 epochs.

F. Handling Queries

For a new query, the hashing code is generated by solving problem (18). In this case, $\mathbf{b}_i$ is the output of the bottleneck layer of the autoencoder for the ith query. z is the output of cardinality predictors. From (18), we can obtain $\mathbf{u}_i$ in real. Then, we thresholds $\mathbf{u}_i$ at 0.5 to generate hashing codes for the new query.

VI. Experimental Results

We evaluate our method on three widely used benchmark datasets: 1) *CIFAR10* [15]; 2) *MIRFlickr* [11]; and 3) *NUSWIDE* [6]. Two kinds of experiments: 1) *hashing lookup* and 2) *Hamming ranking*, were conducted. Mean average precision (mAP) and F-measure are used as metrics.

A. Datasets

CIFAR10 consists of 60 000 images in ten classes. In each class, we randomly select 1000 images as queries and the remaining 5000 images as the retrieval set. Five hundred images are randomly selected from the retrieval set as the training set. That is, we train models with $500 \times 10 = 5000$ images and use $1000 \times 10 = 10\,000$ images to query images in the retrieval set of $(6000 - 1000) \times 10 = 50\,000$ images. The ground-truth neighbors for a query are defined as those in the same category.

MIRFlickr contains 25 000 images. Each image is annotated with at least one of the 24 unique labels. Two thousand images are randomly selected as queries. The remaining 23 000 images are used as retrieval set from which 5000 images are randomly selected as the training set. Ground-truth neighbors for a query are defined as those sharing at least one label.

NUS-WISE contains 269 648 images. The ground truth of 81 concepts is provided for the entire dataset. Ten most common concepts are selected for labels and hence, 186 577 images are left. Five thousand images are randomly selected as queries and the remaining 181 577 images are used as a retrieval set from which 10 500 images are randomly selected as the training set. Ground-truth neighbors for a query are defined as those sharing at least one label.

B. Baselines

The proposed method is compared with three shallow unsupervised hashing models: 1) LSH [2]; 2) ITQ [38]; and 3) SH [35], and four deep unsupervised hashing models: 1) DeepBit [17]; 2) SGH [7]; 3) SSDH [36]; and 4) DistillHash [37]. All the shallow compared models are implemented with MATLAB. We directly use the codes of DeepBit provided by the authors,[1] which is implemented in Caffe [12]. The source codes in TensorFlow [1] of SGH,[2] SSDH,[3] and DistillHash[4] are also provided by the authors. We implement our method with PyTorch.

C. Evaluation

Hamming ranking and *hash lookup* are widely used experiments for evaluating the retrieval performance of hashing

[1] https://github.com/kevinlin311tw/cvpr16-deepbit
[2] https://github.com/doubling/Stochastic_Generative_Hashing
[3] https://github.com/yangerkun/IJCAI2018_SSDH
[4] https://tongliang-liu.github.io/code.html

TABLE III
MAP RESULTS ON WIKI, MIRFLICKR, AND NUS-WIDE DATASETS

Methods	CIFAR10				MIRFlickr				NUS-WIDE			
	16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits
LSH	0.1319	0.1580	0.1673	0.1794	0.5831	0.5885	0.5933	0.6014	0.4324	0.4411	0.4433	0.4816
SH	0.1605	0.1583	0.1509	0.1538	0.5919	0.5923	0.6016	0.6213	0.4458	0.4537	0.4926	0.5000
ITQ	0.1942	0.2086	0.2151	0.2188	0.6192	0.6318	0.6346	0.6477	0.5283	0.5323	0.5319	0.5424
DeepBit	0.2204	0.2410	0.2521	0.2530	0.5934	0.5933	0.6199	0.6349	0.4542	0.4625	0.4762	0.4923
SGH	0.1795	0.1827	0.1889	0.1904	0.6162	0.6283	0.6253	0.6206	0.4936	0.4829	0.4865	0.4975
SSDH	0.2568	0.2560	0.2587	0.2601	0.6621	0.6733	0.6732	0.6771	0.6231	0.6294	0.6321	0.6485
DistillHash	0.2844	0.2853	0.2867	0.2895	0.6964	0.7056	0.7075	0.6995	0.6667	0.6752	0.6769	0.6747
ours	**0.2956**	**0.2960**	**0.2964**	**0.2970**	**0.7122**	**0.7123**	**0.7168**	**0.7193**	**0.6831**	**0.6910**	**0.6917**	**0.6932**

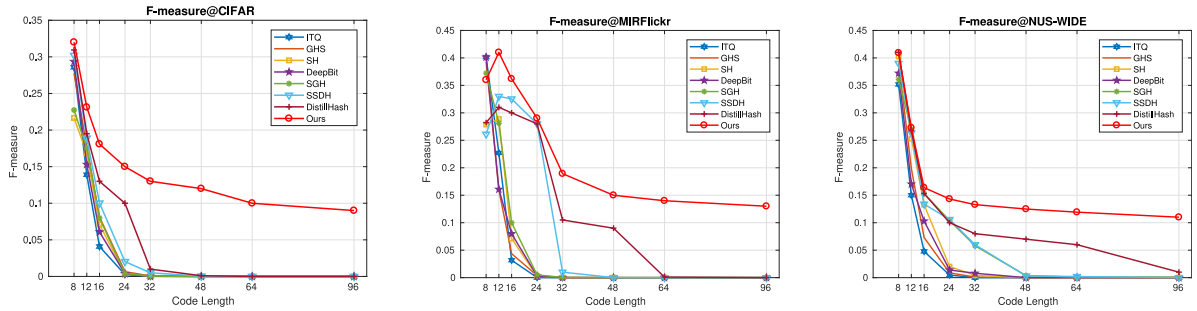


Fig. 3. Results of hash lookup experiments.

TABLE IV
MAP RESULTS OF THREE VERSIONS OF THE PROPOSED METHOD

Methods	CIFAR10				MIRFlickr				NUS-WIDE			
	16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits	16 bits	32 bits	64 bits	128 bits
Version 0	**0.2956**	**0.2960**	**0.2964**	**0.2970**	**0.7122**	**0.7123**	**0.7168**	**0.7193**	**0.6831**	**0.6910**	**0.6917**	**0.6932**
Version 1	0.2719	0.2745	0.2750	0.2718	0.6894	0.6970	0.7022	0.7030	0.6515	0.6639	0.6772	0.6814
Version 2	0.2663	0.2657	0.2672	0.2671	0.6631	0.6705	0.6804	0.6818	0.6439	0.6715	0.6828	0.6856

methods. The Hamming ranking is evaluated by mAP. The average precision (AP) is defined as

$$\mathrm{AP} = \frac{1}{n}\sum_{r=1}^{R} P(r)\delta(r) \tag{21}$$

where n is the number of retrieved items, R is the radius of the Hamming distance, $P(r)$ is the precision of the top r retrieved images, and $\delta(r) = 1$ if the rth retrieved image is true neighbor, otherwise, $\delta(r) = 0$. mAP is the mean of APs for all queries. The larger the mAP, the better the performance.

Hash lookup is evaluated by F-measure. The Hamming radius is set to 2, that is, the hashing codes whose distances to the query are less than 2 are retrieved to calculate F-measure. F-measure is defined as

$$\frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}. \tag{22}$$

D. Implementation Details

We empirically set $\lambda_1 = 1$, $\lambda_2 = 0.1$, and $\lambda_3 = 1$ for all our experiments. The maximal iteration of Dykstra's algorithm is set to 2. We use stochastic gradient descent with momentum 0.9 to update the parameters of the autoencoder and the cardinality prediction network. The learning rate is set to 0.0001. The minibatch size is set to 32 for our model.

For a fair comparison, 4096-dimensional features are extracted by the last fully connected layer of VGG16 [29] network for all shallow methods. The weights of VGG16 are pretrained on ImageNet [8]. All images are resized to 244×244 for VGG16 and 64×64 for the two neural networks used in our model.

E. Results and Discussion

The mAP results are given in Table III. The experiments were done with code length 16, 32, 64, and 128 bits. With features extracted by deep neural networks, the shallow hashing methods are comparable to some deep ones, such as DeepBit and SGH. The shallow methods benefit from the pretrained weights of VGG16 on ImageNet, which is a considerably large compared to datasets used in our experiments. These deep hashing methods learn weights directly from these relatively small datasets, making them difficult to extract better features.

From Table III, we can see that the proposed method achieves the best performance in hash ranking experiments. Furthermore, due to the incorporation of an orthogonality constraint, our method does not have reduced performance on 128-bit experiments compared to 64-bit ones. However, DistillHash, SSDH, and DeepBit experience a performance reduce on 128-bit experiments, because they do not consider any decorrelation among bits.

The hash lookup results measured by F-measure are illustrated in Fig. 3. Our method achieved better performance on most experiments, except for the 8-bit experiments on the MIRFlickr dataset. A possible reason is that cardinality or orthogonality constraints have too large impact on the final codes and hence, the precision is low in this case. Lower λ_1, λ_2, and λ_3 may lead to better results. However, we aim at a robust hashing methods, so we hope there are little efforts on parameter tuning. Because the precision does not increase much for all methods as the code length increases, it is safe to conclude that the higher F-measure of our method is due to the high recall on long-bit experiments.

As the Hamming radius is fixed at 2, generally, the longer the code length, the less the retrived images. With few retrived images, both of precision and recall may approximate 0 for the compared methods. The proposed method handles this problem by incorporating cardinality and orthogonality constraints. The efficiency of orthogonality constraints on long-bit experiments has been well studied, while the effects of cardinality constraint have not been revealed yet. The benefits of cardinality constraint on long-bit experiments are intuitive. Cardinality narrows the interval of Hamming distances of hashing codes of similar data points, which can improve the recall values.

F. Ablation Studies

To examine the effect of cardinality in hashing, we implement two variants of our proposed method.

1) We remove the cardinality component in the testing phase (Version 1). That is, the cardinality predictor is still learned jointly with the autoencoder. However, in the testing phase, we directly quantize the outputs of bottleneck of the autoencoder to generate hashing codes.
2) The cardinality component are removed completely (Version 2). The mAP results are given in Table IV, where "Version 0" denotes the original version of our proposed method.

Without cardinality, "Version 2" achieves the worst results. "Version 1" is a little superior to Version 2, because the cardinality is used in the training phase. Although the cardinality of new queries is not predicted in Version 1, the cardinality constraint affects the parameters of neural networks during the learning procedure, and the hashing codes for training data are generated under it. Hence, the generated codes are a little better than those of Version 2.

VII. Conclusion

In this article, we proposed an unsupervised image hashing model by incorporating cardinality. The proposed model consists of three components. The hashing component is modeled by a deep convolutional autoencoder. The outputs of the bottleneck of the autoencoder are used as the real codes. Another component focusing on prior knowledge of hashing codes is comprised of orthogonality and balance constraints. The cardinality component is modeled by a deep CNN to predict the cardinalities of hashing codes. In the training phase, the cardinality predictor forces similar images to have similar cardinalities, which limits the bounds of the Hamming distances among the corresponding hashing codes of images. In the testing phase, the cardinality of a query is first estimated, then real code of the query is generated by the encoder, and finally, the hashing code is generated by real codes with the cardinality constraint. The experiments on three widely used benchmarks, that is: 1) CIFAR10; 2) MIRFlickr; and 3) NUSWIDE, validate the superiority of our method against the compared state-of-the-art unsupervised hashing methods.

Appendix A
Cardinality and the Bounds of Hamming Distance

Let the cardinality of binary vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ be z_i and z_j, respectively. The code length of $\mathbf{b}_i$ and $\mathbf{b}_j$ is l. Without losing generality, we assume $z_i \leq z_j$.

1) $z_i + z_j \leq l$: The maximal Hamming distance between $\mathbf{b}_i$ and $\mathbf{b}_j$ is $z_i + z_j$. That is, all the bits of 1 of $\mathbf{b}_i$ and $\mathbf{b}_j$ are not overlapping. The minimal Hamming distance is $z_j - z_i$, $\sum_l \mathbf{b}_i \& \mathbf{b}_j = z_i$, where & is the bitwise logical AND, and logical FALSE and TRUE are quantized as 0 and 1, respectively.
2) $z_i + z_j \geq l$: The maximal Hamming distance is $l - z_i + l - z_j = 2l - z_i - z_j$. In this case, there are always some bits of 1 that are overlapping. Since $l - z_i + l - z_j \leq l$, we use bits of 0 to calculate the Hamming distance. The minimal Hamming distance is $(l - z_i) - (l - z_j) = z_j - z_i$.

Putting these two cases together, we obtain the interval for Hamming distance of $\mathbf{b}_i$ and $\mathbf{b}_j$ is $[|z_i - z_j|, \min(z_i + z_j, 2l - z_i - z_j)]$.

Appendix B
Proof of Projecting $\mathbf{B}$ on Ω_1

To prove $\Pi_{\Omega_1}(\mathbf{B}') = \mathbf{B}' - (1/n)\mathbf{1}\mathbf{1}^\top\mathbf{B}'$, we calculate the minimizer of the following problem:

$$\begin{aligned} \arg\min_{\mathbf{Y}} \ & \left\|\mathbf{B}' - \mathbf{Y}\right\|_F^2 \\ \text{s.t.} \ & \mathbf{1}^\top\mathbf{Y} = \mathbf{0}. \end{aligned} \tag{23}$$

Equation (23) is equivalent to the following problem:

$$\arg\min_{\mathbf{Y}} E = \frac{1}{2}\left\|\mathbf{B}' - \mathbf{Y}\right\|_F^2 + \mathbf{1}^\top\mathbf{Y}\boldsymbol{\lambda}. \tag{24}$$

Taking the partial derivative of E with respect to $\mathbf{Y}$, we obtain

$$\frac{\partial E}{\partial \mathbf{Y}} = \left(\mathbf{Y} - \mathbf{B}'\right) + \mathbf{1}\boldsymbol{\lambda}. \tag{25}$$

By setting $(\partial E/\partial \mathbf{Y}) = 0$, we obtain

$$\mathbf{Y} = \mathbf{B} - \mathbf{1}\boldsymbol{\lambda}^\top. \tag{26}$$

According to $\mathbf{1}^\top\mathbf{Y} = 0$, we can deduce that $\boldsymbol{\lambda}^\top = (\mathbf{1}^\top/n)\mathbf{B}$. Hence, $\Pi_{\Omega_1}(\mathbf{B}') = \mathbf{B}' - (1/n)\mathbf{1}\mathbf{1}^\top\mathbf{B}'$. ■

Appendix C
Dykstra's Algorithm

Dykstra's algorithm converges to the Euclidean projection onto the intersection of convex sets $\mathcal{A}$ and $\mathbf{B}$. For each step $r \in$

$[R]$ where R is the maximal iteration, the following sequence is computed:

$$\begin{aligned}\tilde{\mathbf{y}}^{(r)} &= \Pi_{\mathcal{A}}\left(\mathbf{y}^{(r)} + p^{(r)}\right)\\ \mathbf{p}^{(r+1)} &= \mathbf{y}^{(r)} + \mathbf{p}^{(r)} - \tilde{\mathbf{y}}^{(r)}\\ \mathbf{y}^{(r+1)} &= \Pi_{\mathcal{B}}\left(\tilde{\mathbf{y}}^{(r)} + \mathbf{q}^{(r)}\right)\\ \mathbf{q}^{(r+1)} &= \tilde{\mathbf{y}}^{(r)} + \mathbf{q}^{(r)} - \mathbf{y}^{(r+1)}\end{aligned}$$

, where $\mathbf{p}^{(0)} = \mathbf{q}^{(0)} = 0$.

REFERENCES

[1] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, [Online]. Available: arXiv:1603.04467.

[2] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, Jan. 2008.

[3] J. P. Boyle and R. L. Dykstra, "A method for finding projections onto the intersection of convex sets in hilbert spaces," in *Advances in Order Restricted Statistical Inference*. New York, NY, USA: Springer, 1986, pp. 28–47.

[4] N. Brukhim and A. Globerson, "Predict and constrain: Modeling cardinality in deep structured prediction," in *Proc. 35th Int. Conf. Mach. Learn.*, Jul. 2018, pp. 659–667.

[5] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. ACM Symp. Theory Comput.*, 2002, pp. 380–388.

[6] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng, "NUS-WIDE: A real-world Web image database from National University of Singapore," in *Proc. ACM Conf. Image Video Retrieval*, 2009, p. 48.

[7] B. Dai, R. Guo, S. Kumar, N. He, and L. Song, "Stochastic generative hashing," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 913–922.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[9] G. Ding, Y. Guo, and J. Zhou, "Collective matrix factorization hashing for multimodal data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 2083–2090.

[10] S. He *et al.*, "Bidirectional discrete matrix factorization hashing for image search," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 4157–4168, Sep. 2020.

[11] M. J. Huiskes and M. S. Lew, "The MIR flickr retrieval evaluation," in *Proc. ACM Int. Conf. Multimedia Inf. Retrieval*, 2008, pp. 39–43.

[12] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," 2014. [Online]. Available: arXiv:1408.5093.

[13] W. Jun, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2393–2406, Dec. 2012.

[14] W. Kong and W.-J. Li, "Isotropic hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1646–1654.

[15] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[16] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 6, pp. 1092–1104, Jun. 2012.

[17] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 1183–1192.

[18] H. Liu, X. Li, S. Zhang, and Q. Tian, "Adaptive hashing with sparse matrix factorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 4318–4329, Oct. 2020.

[19] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3419–3427.

[20] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. Int. Conf. Mach. Learn.*, 2011, pp. 1–8.

[21] X. Lu, Y. Chen, and X. Li, "Discrete deep hashing with ranking optimization for image retrieval," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 2052–2063, Jun. 2020.

[22] X. Lu, X. Zheng, and X. Li, "Latent semantic minimal hashing for image retrieval," *IEEE Trans. Image Process.*, vol. 26, no. 1, pp. 355–368, Jan. 2017.

[23] C. Ma, I. W. Tsang, F. Shen, and C. Liu, "Error correcting input and output hashing," *IEEE Trans. Cybern.*, vol. 49, no. 3, pp. 781–791, Mar. 2019.

[24] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Red Hook, NY, USA: Curran Assoc., 2019, pp. 8024–8035.

[25] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, vol. 31, pp. 1–10, Mar. 1966.

[26] F. Shen, C. Shen, Q. Shi, A. van den Hengel, Z. Tang, and H. T. Shen, "Hashing on nonlinear manifolds," *IEEE Trans. Image Process.*, vol. 24, no. 6, pp. 1839–1851, Jun. 2015.

[27] F. Shen, C. Shen, Q. Shi, A. V. D. Hengel, and Z. Tang, "Inductive hashing on manifolds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 1562–1569.

[28] Y. Shen *et al.*, "Auto-encoding twin-bottleneck hashing," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 2815–2824.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.

[30] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua, "LDAHash: Improved matching with smaller descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 1, pp. 66–78, Jan. 2012.

[31] J. Tang, Z. Li, M. Wang, and R. Zhao, "Neighborhood discriminant hashing for large-scale image retrieval," *IEEE Trans. Image Process.*, vol. 24, no. 9, pp. 2827–2840, Sep. 2015.

[32] D. Tian, Y. Wei, and D. Zhou, "Learning decorrelated hashing codes with label relaxation for multimodal retrieval," *IEEE Access*, vol. 8, pp. 79260–79272, 2020.

[33] D. Tian, D. Zhou, M. Gong, and Y. Wei, "Interval type-2 fuzzy logic for semisupervised multimodal hashing," *IEEE Trans. Cybern.*, early access, Apr. 3, 2019, doi: 10.1109/TCYB.2019.2906658.

[34] S. Wang, C. Li, and H.-L. Shen, "Equivalent continuous formulation of general hashing problem," *IEEE Trans. Cybern.*, early access, Jan. 30, 2019, doi: 10.1109/TCYB.2019.2894020.

[35] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 1753–1760.

[36] E. Yang, C. Deng, T. Liu, W. Liu, and D. Tao, "Semantic structure-based unsupervised deep hashing," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 1064–1070.

[37] E. Yang, T. Liu, C. Deng, W. Liu, and D. Tao, "DistillHash: Unsupervised deep hashing by distilling data pairs," 2019. [Online]. Available: arXiv:1905.03465.

[38] G. Yunchao and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 817–824.

[39] J. T. Zhou, H. Zhao, X. Peng, M. Fang, Z. Qin, and R. S. M. Goh, "Transfer hashing: From shallow to deep," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 6191–6201, Dec. 2018.

[40] X. Zhou *et al.*, "Graph convolutional network hashing," *IEEE Trans. Cybern.*, vol. 50, no. 4, pp. 1460–1472, Apr. 2020.

[41] L. Zhu, J. Shen, L. Xie, and Z. Cheng, "Unsupervised topic hypergraph hashing for efficient mobile image retrieval," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3941–3954, Nov. 2017.

Dayong Tian received the B.S. and M.E. degrees from Xidian University, Xi'an, China, in 2010 and 2014, respectively, and the Ph.D. degree from the University of Technology at Sydney, Sydney, NSW, Australia, in 2017.

He is currently an Assistant Professor with the School of Electronics and Information, Northwestern Polytechnical University, Xi'an. His research interests include computer vision and machine learning, and, in particular, image restoration, image retrieval, and face recognition.

Chen Gong (Member, IEEE) received the B.E. degree from the East China University of Science and Technology, Shanghai, China, in 2010, and the dual Ph.D. degrees from Shanghai Jiao Tong University (SJTU), Shanghai, and the University of Technology Sydney, Ultimo, NSW, Australia, in 2016 and 2017, respectively.

He is currently a Full Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. He has published over 70 technical articles at prominent journals and conferences, such as the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON IMAGE PROCESSING, the IEEE TRANSACTIONS ON CYBERNETICS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, *ACM Transactions on Intelligent Systems and Technology*, NeurIPS, CVPR, the Association for the Advancement of Artificial Intelligence, International Joint Conference on Artificial Intelligence, and ICDM. His research interests include machine learning, data mining, and learning-based vision problems.

Dr. Gong was a recipient of the Excellent Doctoral Dissertation Award by Shanghai Jiao Tong University and the Chinese Association for Artificial Intelligence. He serves as a SPC/PC member for several top-tier conferences, such as ICML, NeurIPS, AAAI, IJCAI, ICDM, and AISTATS. He was also enrolled by the Young Elite Scientists Sponsorship Program of Jiangsu Province and the China Association for Science and Technology. He also serves as a reviewer for over 20 international journals, such as *Artificial Intelligence*, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, and IEEE TRANSACTIONS ON IMAGE PROCESSING.

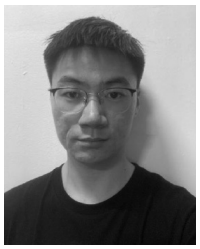
Maoguo Gong (Senior Member, IEEE) received the B.S. degree (First Class Hons.) in electronic engineering and the Ph.D. degree in electronic science and technology from Xidian University, Xi'an, China, in 2003 and 2009, respectively.

Since 2006, he has been a Teacher with Xidian University. In 2008 and 2010, he was promoted as an Associate Professor and as a Full Professor, respectively, both with exceptional admission. His research interests are in the area of computational intelligence with applications in optimization, learning, data mining, and image understanding.

Dr. Gong received the Prestigious National Program for the support of Top-Notch Young Professionals from the Central Organization Department of China, the Excellent Young Scientist Foundation from the National Natural Science Foundation of China, and the New Century Excellent Talent in University from the Ministry of Education of China. He is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.

Yiwen Wei (Member, IEEE) received the Ph.D. degree in radio science with the School of Physics and Optoelectronic Engineering, Xidian University, Xi'an, China, in 2016.

From 2016 to 2018, she worked as a Research Scientist with Temasek Laboratories, National University of Singapore, Singapore. She is currently an Assistant Professor with the School of Physics and Optoelectronic Engineering Science, Xidian University. Her research interests include electromagnetic-wave propagation and scattering in complex systems, computational electromagnetics, remote sensing, parameters retrieval, and, in particular, applying machine learning methods on complex electromagnetic problems.



Xiaoxuan Feng received the B.S. degree from the School of Electronic Information, Northwestern Polytechnical University, Xi'an, China, in 2019, where he is currently pursuing the M.E. degree in avionics comprehensive technology.

His research interests include computer vision and machine learning, and, in particular, image restoration and image change detection.