IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 February 2023, accepted 14 March 2023, date of publication 20 March 2023, date of current version 29 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3259104

RESEARCH ARTICLE

RelaHash: Deep Hashing With Relative Position

PHAM VU THAI MINH[1], NGUYEN DONG DUC VIET[1], NGO TUNG SON[1,2], BUI NGOC ANH[1], AND JAFREEZAL JAAFAR[2], (Senior Member, IEEE)

[1]Information and Communication Technology Department, FPT University, Hanoi 100000, Vietnam
[2]Department of Computer and Information Sciences, Universiti Teknologi Petronas, Seri Iskandar 32610, Malaysia

Corresponding author: Ngo Tung Son (sonnt69@fe.edu.vn)

ABSTRACT Deep hashing has been widely used as a solution to encoding binary hash code for approximating nearest neighbor problem. It has been showing superior performance in terms of its ability to index high-level features by learning compact binary code. Many recent state-of-the-art deep hashing methods often use multiple loss terms at once, thus introducing optimization difficulty and may result in sub-optimal hash codes. OrthoHash was proposed to replace those losses with just a single loss function. However, the quantization error minimization problem in OrthoHash is still not addressed effectively. In this paper, we take one step further - propose a single-loss model that can effectively minimize the quantization error without explicit loss terms. Specifically, we introduce a new way to measure the similarity between the relaxed codes with centroids, called relative similarity. The relative similarity is the similarity between the relative position representation of continuous codes and the normalized centroids. The resulting model outperforms many state-of-the-art deep hashing models on popular benchmark datasets.

INDEX TERMS Image retrieval, quantization, supervised deep hashing, neural network, convolutional neural network.

I. INTRODUCTION

In the big data era, there is an increasing demand for an efficient way to quickly retrieve images from a set of query images in giant image databases. Many image retrieval algorithms for approximating nearest neighbors have been proposed, ranging from Tree [2], [3], [4], Ranking [5], to hashing techniques [6], [7]. Hashing techniques have empirically shown their advantages compared to other methods in terms of time and space complexity. Hashing is a mapping function that generates compact binary codes to represent the images (indexing process) in order to store and retrieve images from the database efficiently later (retrieval process), as illustrated in Figure 1. Comparing images by their binary hash codes in the Hamming space is very efficient because it only requires a logical bit-wise XOR operation, followed by a *popcount*. Combined with hash code searching algorithms [8], this can significantly reduce computational time to search for images in large high-dimensional databases. The goal of learning to hash is to learn a hash function that can preserve the semantic difference between data points. Images with similar semantics should be assigned to similar hash codes, while distinct images should be assigned to very different hash codes. Learning to hash can also leverage deep learning to become deep hashing, which shows much more performance than conventional methods. With the rise of deep learning [9], many powerful yet efficient neuron network architectures have been proposed [10], [11], [12], [13], [14] and rapidly incorporated into deep hashing [15], [16], [17], [18], [19]. As a result, deep hashing can encode compact binary codes representing complex, high-level features.

In order to quantize the Euclidean space into the discrete Hamming space, deep hashing methods often use a quantization layer. Quantization error is the amount of information loss caused by this quantization process. Many deep hashing methods often include quantization error minimization objectives (Figure 2) by explicitly adding some penalty terms into the loss function [20], [21], [22], [23]. These penalty terms often aim to reduce the Euclidean distance (Figure 3a) between these two representations. However, this generally makes the training process harder and can lead to sub-optimal solutions. Recently many efforts have aimed to reduce the total number of loss terms added to the loss function [1], [24].

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Shorif Uddin.

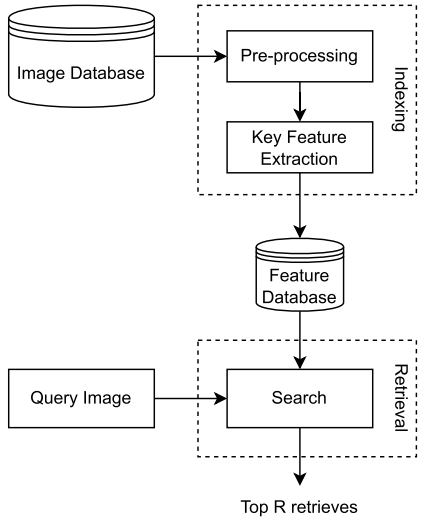
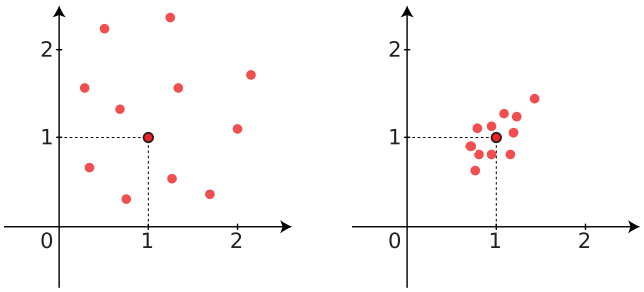
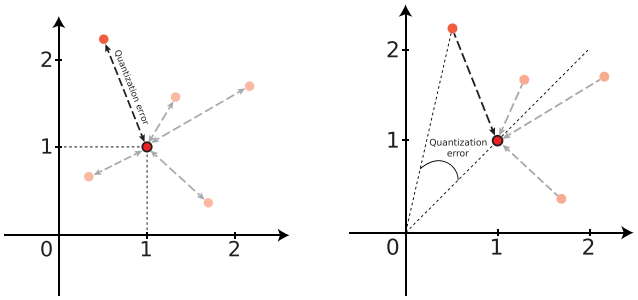


FIGURE 1. Overview of the image indexing and retrieval framework. Images in the database get pre-processed, and their key features are extracted into an indexed feature database (indexing process). In order to retrieve the top R retrieved items (retrieval process), query images representing users' intentions will search through this indexed feature database.



(a) High quantization error (b) Low quantization error

FIGURE 2. The illustration of high quantization error (a) and low quantization error (b) of the learned hash code in the Euclidean space.



(a) Euclidean distance (b) Angular distance

FIGURE 3. The visualization of how quantization error is measured. Quantization error is the amount of information loss caused by the quantization layer, i.e., the difference between the continuous hash code in the Euclidean space and its discrete representation in the Hamming space. This difference can be measured by the Euclidean distance (a) or angular distance (b).

OrthoHash [1] proposed directly merging the quantization error optimization objective into the main loss function by only minimizing the angle instead (Figure 3b). Nevertheless, we find that the quantization error can be further optimized to achieve even better performance.

This paper presents a new approach to address the issue of quantization error in a unified loss manner using a novel concept called "relative similarity". A new layer called the Relative Transform layer is introduced and combined with the scaled dot-product similarity, leading to a model that can learn hash codes with low quantization error without the need for additional loss terms. We have conducted extensive experiments to illustrate the effectiveness of our model compared to many recent state-of-the-art methods. Our model outperforms many current state-of-the-art methods on single-label datasets while having a competitive performance on multi-label datasets. Concretely, our model RelaHash generally achieves new state-of-the-art performance on CIFAR-10 [25] and ImageNet100 [26] datasets. Interestingly, we also have some considerable improvements, about 2.6%-11.5% on ImageNet100 16-bits settings compared to other methods.

In this paper, our **main contributions** are as follows:

1) We introduce a new transformation layer that implicitly embeds the minimizing quantization error objective into the softmax loss without the need for additional loss terms.
2) We propose a novel method to measure similarity between points in space, called Relative Similarity.
3) We demonstrate the effectiveness and robustness of our model - RelaHash - compared to recent state-of-the-art methods on several popular benchmark datasets and establish new state-of-the-art.

Our work in this paper is organized as follows: We review related work on deep hashing in Section II. Then, we present the detail of our proposed method in Section III. The experimental results and some visual analysis of our approach are provided in Section IV. Finally, we draw some conclusions in Section V.

II. RELATED WORK

Hashing is divided into data-dependent and data-independent methods. Data-independent hashing methods [6], [27] are to design hash functions that do not rely on underlying data. On the other hand, data-dependent hashing methods are learning to hash based on data. With the advancement of deep learning [9], learning to hash begins to leverage the deep learning power, to become deep hashing [8], [28], [29]. Deep hashing can be unsupervised or supervised, whereas supervised methods have empirically shown much better performance over unsupervised ones because they can derive semantic similarity from labeled data. HashGAN [16] uses labels to obtain pairwise similarity information for augmenting the training data. These labels also allow DFPH [30] to extract multi-level visual and semantic information contained in images. Besides, DOH [31] utilizes labels to learn local and global semantics using two subnetworks, whereas [32] makes use of multi-label to learn instance-aware representations split into groups, each group represents a category.

Additionally, HashFormer [17] directly optimizes retrieval accuracy by using labels to compute its average precision loss. Based on how similarity between data points is measured, supervised deep hashing methods can be further categorized into pairwise [16], [22], [33], triplet-wise [34], [35], and pointwise [1], [36], [37] methods. Among them, pairwise and triplet-wise methods are prone to imbalance data, have a high learning time complexity of at least an order of $O(N^2)$, and are hard to cover the entire data distribution. Thus, pointwise methods are generally better and suitable for real-world application [1], [37]. Learning in a pointwise manner with a set of target centroids has increasingly shown great performance compared to other methods. Data points that have mutual labels are converged to the same hash centroids. CSQ [37] uses fixed centroids generated by the Hadamard matrix, while DPN [36] uses hinge-like polarized loss to converge to randomly generated target vectors. OrthoHash [1] also proposed Maximum Hamming Distance algorithm that can heuristically find optimal centroids [38]. Our method takes a similar approach to learning hash code with pre-defined centroids in the Hamming space.

Several factors contribute to the quality of learned hash codes, such as vanishing gradient [20], [33], code-balance [24], [39], and quantization error [24].

Hashing is a binary optimization problem, which is proved to be NP-hard. Learning to hash is to learn a mapping from a continuous distribution to a discrete distribution. Continuous values will be quantized into discrete values through a quantization layer - where the sign function is often used. The gradient of the sign function, however, is ill-defined. Its gradient is 0 for all non-zero values, which makes it impossible for the network to learn through backpropagation with gradient descent. Existing methods often find ways to get around this problem, such as by softening the discrete constraints [33] or by using the straight-through estimator layer [40] to custom the backpropagation flow [20], [39]. These methods involve sophisticated ways to modify the computational graphs, resulting in complicated optimization. Our method avoids these complicated problems by using a different approach that does not need to involve the non-differentiable sign function at training time.

Quantization error is the amount of information lost when relaxed hash codes pass through the quantization layer. This error occurs when using a discrete distribution to approximate a continuous distribution. Minimizing this error often leads to improvement in retrieval performance. Existing methods usually penalize this error by introducing additional loss terms such as the Minkowski distance between the relaxed and the discrete hash codes [20], [21], [22], [23]. HSWD [24] proposed to minimize this error by penalizing the Wasserstein distance between these relaxed codes with a uniform discrete distribution. HHF [41] tries to balance between metric loss and quantization loss by designing an inflection point. By adding another loss term, we have to make an extra effort to balance these losses, making the training process more difficult and possibly producing suboptimal hash codes [20]. This paper proposes a method to reduce quantization errors naturally.

Some point-wise methods such as CSQ [37] and DPN [36] use pre-transformations on learning hash code before comparing them with centroids. CSQ uses the tanh function before computing BCE, whereas DPN employs a ternary assignment before going to the Polarization Loss to gain better performance. Our method uses Relative Transform as a proxy to learn the continuous codes with low quantization constraints. The obtained Relative Position representations from this function are adjusted in the continuous space by taking into account the total squared norm of each instance, which results in these representations being pulled toward the center. These adjustments will update the continuous codes appropriately through backpropagation.

The use of angle-based techniques to measure similarity between data points to learn hash codes has recently shown promising results [1], [42], [43]. These methods can utilize previous work in metric learning problems about margin loss, such as A-Softmax [44], L-Softmax [45], and Large Margin Cosine Loss [46], to further minimize intra-class distances while maximizing inter-class distances at the same time. OrthoHash [1] has found a way to integrate the quantization error into the main loss by maximizing cosine similarity between data points and centroids, resulting in a single-loss model that allows end-to-end training. Our method inherits all these advantages since we closely follow these approaches.

III. RelaHash: DEEP HASHING WITH RELATIVE POSITION

This section begins by mathematically defining the hashing problem and formulating our model. We then present our new formula to measure similarity and apply it to the supervised deep hashing problem. After that, we discuss the optimization process, including the derivatives and time complexity of the transformation function.

Here are the mathematical definitions of our supervised deep hashing problem. Let $\boldsymbol{\mathcal{X}} = \{\mathbf{x}_n\}_{n=1}^N$ be N data points and $\boldsymbol{\mathcal{Y}} = \{\mathbf{y}_n \in \{0,1\}^M\}^N$ as their corresponding label of M classes. Our learning goal is a non-linear hash function f that maps from $\boldsymbol{\mathcal{X}}$ to K-bits binary codes, as follows:

$$\boldsymbol{\mathcal{H}} = f(\boldsymbol{\mathcal{X}}) = sgn(\boldsymbol{\mathcal{Z}}) \tag{1}$$

where the function is given by:

$$f : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{H}} = \{\mathbf{h}_n \in \{-1,1\}^K\}_{n=1}^N \tag{2}$$

where $sgn(z)$ is the element-wise sign function, return 1 if $z \geq 0$ and -1 otherwise, and $\boldsymbol{\mathcal{Z}} = \{\mathbf{z}_n \in \mathbb{R}^K\}_{n=1}^N$ is the relaxed codes computed by the backbone $\phi(\cdot)$ following by a latent layer.

$$\boldsymbol{\mathcal{Z}} = \phi(\boldsymbol{\mathcal{X}})\boldsymbol{W} \tag{3}$$

where $\phi(\boldsymbol{\mathcal{X}}) \in \mathbb{R}^d$ is the output of the backbone ϕ, and $\boldsymbol{W} \in \mathbb{R}^{d\times K}$ denotes the weights of the latent layer. We define a set of M centroids $\boldsymbol{\mathcal{C}} = \{\mathbf{c}_i \in \{-1,1\}^K\}_{i=1}^M$, where each centroid

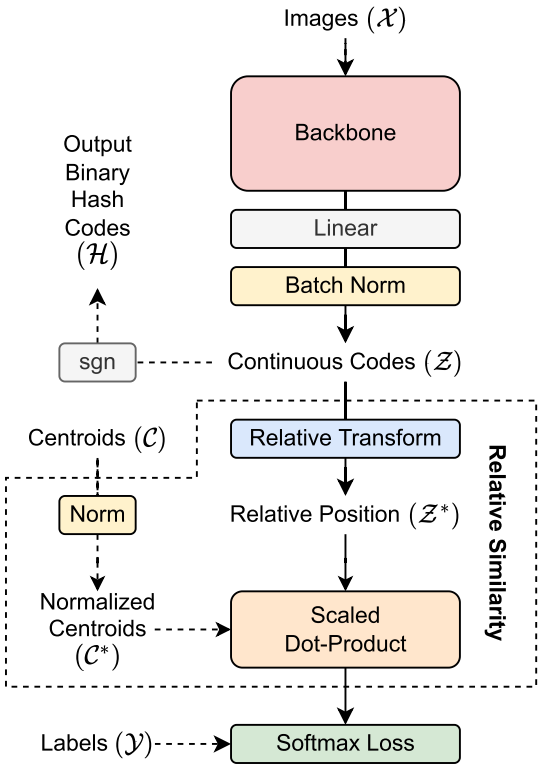


FIGURE 4. Overview of RelaHash's architecture. First, the backbone ϕ computes deep representation features of images, followed by a fully-connected hash layer W and a batch norm layer to compute K-bit relaxed hash codes. These continuous codes $\mathcal{Z}$ can go through the quantization layer (*sgn* function) to get output binary codes. To train these continuous codes $\mathcal{Z}$, we calculate their Relative Similarity with centroids $\mathcal{C}$ and then use their corresponding labels $\mathcal{Y}$ to compute Softmax Loss. See section III for details. *The dashed arrows indicate where backpropagation is dropped.*

represents the semantic position of a class in the Hamming space.

A. FORMULATE MODEL

Our model uses the following architecture as shown in Figure 4. We use the backbone $\phi(\cdot)$ as a feature extractor for images. The extracted feature will go through a fully connected hash layer (latent layer), followed by a batch norm layer. The batch norm layer allows the model to achieve code balancing while not modifying the computational graph [1]. The resulting representation is the continuous hash codes, i.e. relaxed codes $\mathcal{Z} = \{\mathbf{z}_n \in \mathbb{R}^K\}_{n=1}^N$, which can be applied $sgn(\cdot)$ function to get the corresponding binary hash codes. In order to update hash codes $\mathcal{Z}$, we compute the scaled dot-product similarity between the Relative Position representation of $\mathcal{Z}$ with the normalized centroids $\mathcal{C}^*$. By maximizing this similarity through a softmax loss (cross-entropy), we can achieve a new representation of $\mathcal{Z}$ in the Euclidean space where quantization error and hamming distance to their assigned centroids are minimized. We use Maximum Hamming Distance algorithm [38] to generate fixed centroids $\mathcal{C}$ in the Hamming space.

B. RELATIVE SIMILARITY

1) RELATIVE POSITION

Given a set of column vectors $\mathcal{Z} = \{\mathbf{z}_n \in \mathbb{R}^K\}_{n=1}^B$, where B denotes the batch size, and K denotes the number of bits. The relative position representation of $\mathcal{Z}$ is defined as:

$$\text{RelativeTransform}(\mathcal{Z}) = \sqrt{B \times K}\,\frac{\mathcal{Z} - \mu_{\text{batch}}}{\|\mathcal{Z} - \mu_{\text{batch}}\|_F} \tag{4}$$

in which $\|\cdot\|_F$ is the Frobenius norm. μ_{batch} is a single real-valued number representing the mean along the bits of the expectation of the batch, which is obtained as:

$$\mu_{\text{batch}} := \frac{1}{K}\sum_{i=1}^{K}\mathbb{E}[\mathcal{Z}]_i \tag{5}$$

This new representation of $\mathcal{Z}$ embeds the quantization error learning objective while still preserving the relative relationship between the column vectors $\mathbf{z}_n \in \mathcal{Z}$ in both the Euclidean space and Hamming space.

2) SCALED DOT-PRODUCT SIMILARITY WITH RELATIVE POSITION

We then compute the similarity between each vector $\mathbf{z}_n^* \in \mathcal{Z}^* := \mathit{RelativeTransform}(\mathcal{Z})$ and the normalized centroids $\mathbf{c}_i^* = \frac{\mathbf{c}_i}{\|\mathbf{c}_i\|} \in \mathcal{C}^*$ whose $\mathbf{c}_i \in \mathcal{C}$ via scaled dot product operation. The matrix of output can be computed as follow:

$$\text{sim}(\mathcal{Z}, \mathcal{C}) = \alpha \mathcal{Z}^{*\mathsf{T}}\mathcal{C}^* \tag{6}$$

in which α is the hyperparameter controlling the scale of the dot product operation. We aim to maximize the similarity between each vector $\mathbf{z}_n^*$ and the centroids $\mathbf{c}_i^*$ representing its ground-truth label. This can be accomplished by using their associated labels with softmax loss (cross-entropy loss). Since the term $\sqrt{B \times K}$ in equation (4) is constant, this can be merged with hyperparameter α. Geometrically, the dot product between a vector $\mathbf{z}_n^*$ and a centroid $\mathbf{c}_i^*$ can be interpreted as:

$$\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_i^* = \left\|\mathbf{z}_n^*\right\| \left\|\mathbf{c}_i^*\right\| \cos\theta_{\mathbf{z}_n^*,\mathbf{c}_i^*} \tag{7}$$

In which $\|\cdot\|$ is the Euclidian norm, $\left\|\mathbf{c}_i^*\right\| = 1$ since $\mathbf{c}_i^*$ is the normalized vector, and $\theta_{\mathbf{z}_n^*,\mathbf{c}_i^*}$ is the angle between $\mathbf{z}_n^*$ and $\mathbf{c}_i^*$. Since our equation is proportional to the cosine of $\theta_{\mathbf{z}_n^*,\mathbf{c}_i^*}$, maximizing this product also means minimizing the angle $\theta_{\mathbf{z}_n^*,\mathbf{c}_i^*}$, which reduces quantization error and enhances the discriminativeness of hash code [1]. For the norm of vector $\mathbf{z}_n^*$, we let the network decide its own best value.

3) HASHING WITH RELATIVE SIMILARITY

Since we are using softmax loss to maximize inter-class distance in the Hamming space while minimizing intra-class variance, we can leverage the large cosine margin loss [46]. Therefore, our loss function is as follows:

$$L = -\frac{1}{N}\sum_{n=1}^{N}\log\frac{e^{s\left(\text{sim}\left(\mathbf{z}_n^*,\mathbf{c}_{y_n}^*\right)-m\right)}}{e^{s\left(\text{sim}\left(\mathbf{z}_n^*,\mathbf{c}_{y_n}^*\right)-m\right)} + \sum_{i=1,i\neq y_n}^{M} e^{s\cdot\text{sim}\left(\mathbf{z}_n^*,\mathbf{c}_i^*\right)}}$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\log\frac{e^{s\left(\alpha\cdot\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_{y_n}^{*}-m\right)}}{e^{s\left(\alpha\cdot\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_{y_n}^{*}-m\right)}+\sum_{i=1,i\neq y_n}^{M}e^{s\cdot\alpha\cdot\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_i^{*}}} \tag{8}$$

where m is the cosine margin, and s is the scale factor. As a further enhancement to formula (8), we decided to scale margin m by α, resulting in the following new loss function:

$$\begin{aligned} L &= -\frac{1}{N}\sum_{n=1}^{N}\log\frac{e^{s\cdot\alpha\left(\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_{y_n}^{*}-m\right)}}{e^{s\cdot\alpha\left(\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_{y_n}^{*}-m\right)}+\sum_{i=1,i\neq y_n}^{M}e^{s\cdot\alpha\cdot\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_i^{*}}} \\ &= -\frac{1}{N}\sum_{n=1}^{N}\log\frac{e^{\beta\left(\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_{y_n}^{*}-m\right)}}{e^{\beta\left(\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_{y_n}^{*}-m\right)}+\sum_{i=1,i\neq y_n}^{M}e^{\beta\cdot\mathbf{z}_n^{*\mathsf{T}}\mathbf{c}_i^{*}}} \end{aligned} \tag{9}$$

where $\beta = s\cdot\alpha$ is the hyperparameter acts similar role compared to s and α. By minimizing this loss, we are able to train a robust deep hashing network that generates accurate binary hash codes.

C. OPTIMIZATION PROCESS

Let Θ denotes the parameters of the network, which is comprised of the backbone ϕ and the latent layer $\boldsymbol{W}$. The learning rate of the backbone ϕ is lowered by ten times to avoid overfitting. First, we preprocess the training set $\boldsymbol{\mathcal{X}}$, and initialize centroids $\boldsymbol{\mathcal{C}}$ by using the MaxHD algorithm [38]. These centroids then remain fixed and will not be learned. In fact, learning centroids may even harm the model's performance [36], [37]. In the forward propagation of the training stage, we calculate $\boldsymbol{\mathcal{Z}}^*$ and $\boldsymbol{\mathcal{C}}^*$, then calculate the total loss. Backward propagation is performed by calculating the gradient of network parameters and updating them using Adam optimizer [47]. The overall optimization process is summarized in Algorithm 1. In step 8 of Algorithm 1, we obtain the gradient of $\boldsymbol{\mathcal{Z}}^*$ as follows:

$$\begin{aligned} dL &= \frac{\partial L}{\partial\left(\boldsymbol{\mathcal{Z}}^{*\mathsf{T}}\boldsymbol{\mathcal{C}}^{*}\right)} : d\boldsymbol{\mathcal{Z}}^{*\mathsf{T}}\boldsymbol{\mathcal{C}}^{*} = \frac{\partial L}{\partial\left(\boldsymbol{\mathcal{Z}}^{*\mathsf{T}}\boldsymbol{\mathcal{C}}^{*}\right)}\boldsymbol{\mathcal{C}}^{*\mathsf{T}} : d\boldsymbol{\mathcal{Z}}^{*\mathsf{T}} \\ &= \boldsymbol{\mathcal{C}}^{*}\left(\frac{\partial L}{\partial\left(\boldsymbol{\mathcal{Z}}^{*\mathsf{T}}\boldsymbol{\mathcal{C}}^{*}\right)}\right)^{\mathsf{T}} : d\boldsymbol{\mathcal{Z}}^{*} \\ \Leftrightarrow \frac{\partial L}{\partial\boldsymbol{\mathcal{Z}}^{*}} &= \boldsymbol{\mathcal{C}}^{*}\left(\frac{\partial L}{\partial\left(\boldsymbol{\mathcal{Z}}^{*\mathsf{T}}\boldsymbol{\mathcal{C}}^{*}\right)}\right)^{\mathsf{T}} \end{aligned} \tag{10}$$

where (:) is the Frobenius inner product. To calculate $\frac{\partial L}{\partial\boldsymbol{\mathcal{Z}}}$ in step 9 of Algorithm 1, we need to calculate $\frac{\partial\boldsymbol{\mathcal{Z}}^*}{\partial\boldsymbol{\mathcal{Z}}}$.

1) THE DERIVATIVE OF THE RELATIVE TRANSFORM FUNCTION

We now need to analyze the derivative of the Relative Transform function. We set the following:

$$\boldsymbol{\mathcal{Z}} - \mu_{\text{batch}} = \boldsymbol{\mathcal{A}} \in \mathbb{R}^{K\times B}$$

The derivative of $\boldsymbol{\mathcal{Z}}^*$ with respect to $\boldsymbol{\mathcal{Z}}$ is obtained by using the chain rule:

$$\frac{\partial\boldsymbol{\mathcal{Z}}^{*}}{\partial\boldsymbol{\mathcal{Z}}} = \frac{\partial\boldsymbol{\mathcal{Z}}^{*}}{\partial\boldsymbol{\mathcal{A}}}\frac{\partial\boldsymbol{\mathcal{A}}}{\partial\boldsymbol{\mathcal{Z}}} \tag{11}$$

Algorithm 1 RelaHash

1: Prepare training set $\boldsymbol{\mathcal{X}}$.
2: Initialize centroids $\boldsymbol{\mathcal{C}}$.
3: **repeat**
{*Forward propagation* :}
4: - Calculate $\boldsymbol{\mathcal{Z}}$ by ϕ and W
5: - $\boldsymbol{\mathcal{Z}}^* \leftarrow$ RelativeTransform($\boldsymbol{\mathcal{Z}}$)
6: - Calculate $\boldsymbol{\mathcal{C}}^*$ by $\boldsymbol{\mathcal{C}}$ {normalize centroids}
7: - Calculate loss L by $\boldsymbol{\mathcal{Z}}^*$ and $\boldsymbol{\mathcal{C}}^*$
{*Backward propagation* :}
8: - Calculate $\frac{\partial L}{\partial\boldsymbol{\mathcal{Z}}^*}$
9: - Calculate $\frac{\partial L}{\partial\boldsymbol{\mathcal{Z}}} = \frac{\partial L}{\partial\boldsymbol{\mathcal{Z}}^*}\frac{\partial\boldsymbol{\mathcal{Z}}^*}{\partial\boldsymbol{\mathcal{Z}}}$
10: - Calculate $\frac{\partial L}{\partial\Theta} = \frac{\partial L}{\partial\boldsymbol{\mathcal{Z}}}\frac{\partial\boldsymbol{\mathcal{Z}}}{\partial\Theta}$
11: - Update network's parameters
12: **until** convergence

We have to compute the derivative of $\boldsymbol{\mathcal{Z}}^*$ with respect to $\boldsymbol{\mathcal{A}}$, then calculate the derivative of $\boldsymbol{\mathcal{A}}$ with respect to $\boldsymbol{\mathcal{Z}}$. First, we differentiate the Frobenius norm:

$$\begin{aligned} \gamma &= \|\boldsymbol{\mathcal{A}}\|_F \\ \gamma^2 &= \|\boldsymbol{\mathcal{A}}\|_F^2 = \boldsymbol{\mathcal{A}} : \boldsymbol{\mathcal{A}} \\ 2\gamma\, d\gamma &= 2\boldsymbol{\mathcal{A}} : d\boldsymbol{\mathcal{A}} \\ d\gamma &= \gamma^{-1}\boldsymbol{\mathcal{A}} : d\boldsymbol{\mathcal{A}} \end{aligned} \tag{12}$$

Following this, we differentiate $\boldsymbol{\mathcal{Z}}^*$

$$\begin{aligned} \boldsymbol{\mathcal{Z}}^{*} &= \frac{1}{\|\boldsymbol{\mathcal{A}}\|_F}\boldsymbol{\mathcal{A}} = \gamma^{-1}\boldsymbol{\mathcal{A}} \\ d\boldsymbol{\mathcal{Z}}^{*} &= \gamma^{-1}d\boldsymbol{\mathcal{A}} - \gamma^{-2}\boldsymbol{\mathcal{A}}d\gamma \end{aligned} \tag{13}$$

We define an identity tensor $\mathcal{E}$ using Kronecker deltas δ:

$$\mathcal{E}_{ijkl} = \delta_{ik}\delta_{jl} = \begin{cases} 1 & \text{if} \quad i = k \quad \text{and} \quad j = l \\ 0 & \text{otherwise} \end{cases}$$

Substitute $d\gamma$ into equation (13), we have:

$$\begin{aligned} d\boldsymbol{\mathcal{Z}}^{*} &= \gamma^{-1}\mathcal{E} : d\boldsymbol{\mathcal{A}} - \gamma^{-2}\boldsymbol{\mathcal{A}}\left(\gamma^{-1}\boldsymbol{\mathcal{A}} : d\boldsymbol{\mathcal{A}}\right) \\ &= \gamma^{-1}\left[\mathcal{E} - (\gamma^{-1}\boldsymbol{\mathcal{A}})\otimes\left(\gamma^{-1}\boldsymbol{\mathcal{A}}\right)\right] : d\boldsymbol{\mathcal{A}} \\ &= \gamma^{-1}\left(\mathcal{E} - \boldsymbol{\mathcal{Z}}^{*}\otimes\boldsymbol{\mathcal{Z}}^{*}\right) : d\boldsymbol{\mathcal{A}} \\ &= \|\boldsymbol{\mathcal{A}}\|_F^{-1}\left(\mathcal{E} - \boldsymbol{\mathcal{Z}}^{*}\otimes\boldsymbol{\mathcal{Z}}^{*}\right) : d\boldsymbol{\mathcal{A}} \\ \Leftrightarrow \frac{\partial\boldsymbol{\mathcal{Z}}^{*}}{\partial\boldsymbol{\mathcal{A}}} &= \|\boldsymbol{\mathcal{A}}\|_F^{-1}\left(\mathcal{E} - \boldsymbol{\mathcal{Z}}^{*}\otimes\boldsymbol{\mathcal{Z}}^{*}\right) \end{aligned} \tag{14}$$

where $\otimes$ is the tensor product. Meanwhile, the derivative of $\boldsymbol{\mathcal{A}}$ with respect to $\boldsymbol{\mathcal{Z}}$ can be computed as follows:

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{Z}} - \mu_{\text{batch}} = \boldsymbol{\mathcal{Z}} - \frac{1}{B\times K}\sum_{i=1}^{K}\sum_{j=1}^{B}\boldsymbol{\mathcal{Z}}_{ij}$$

TABLE 1. Performance comparison for 4 different bits on various benchmark datasets. Results on CIFAR-10 are run by us, while the results of other methods on ImageNet100 and NUS-WIDE are obtained from [1] to make them directly comparable. The highest value of each column is shown in **bold**. Point-wise, pair-wise, and triplet-wise methods are indicated by the superscripts [1], [2], and [3].

Methods	**CIFAR-10** (mAP@all)				**ImageNet100** (mAP@1K)				**NUS-WIDE** (mAP@5K)			
	16	32	64	128	16	32	64	128	16	32	64	128
HashNet[2] [33]	0.817	0.818	0.820	0.825	0.343	0.480	0.573	0.612	0.814	0.831	0.842	0.847
DTSH[3] [34]	0.789	0.804	0.816	0.818	0.442	0.528	0.581	0.612	**0.816**	**0.836**	**0.851**	**0.862**
GreedyHash[1] [20]	0.805	0.829	0.836	0.834	0.570	0.639	0.659	0.659	0.771	0.797	0.815	0.832
JMLH[1] [48]	0.803	0.821	0.838	0.829	0.517	0.621	0.662	0.678	0.791	0.825	0.836	0.843
DPN[1] [36]	0.770	0.797	0.821	0.833	0.592	0.670	0.703	0.714	0.783	0.818	0.838	0.842
CSQ[1] [37]	0.794	0.803	0.812	0.815	0.586	0.666	0.693	0.700	0.797	0.824	0.835	0.839
OrthoHash[1] [1]	0.777	0.814	0.832	0.836	0.606	0.679	0.711	0.717	0.804	0.836	0.850	0.856
RelaHash[1]	**0.820**	**0.835**	**0.852**	**0.854**	**0.632**	**0.684**	**0.713**	**0.727**	0.811	0.830	0.843	0.850

$$\Leftrightarrow \frac{\partial \mathcal{A}}{\partial \mathcal{Z}} = \mathcal{E} - (B \times K)^{-1} J \tag{15}$$

where J is a tensor of ones. From equation (11), (14), and (15) we can derive the following:

$$\begin{aligned}
&\frac{\partial \mathcal{Z}^*}{\partial \mathcal{Z}} \\
&= \frac{\partial \mathcal{Z}^*}{\partial \mathcal{A}} \frac{\partial \mathcal{A}}{\partial \mathcal{Z}} \\
&= \left[\|\mathcal{A}\|_F^{-1} \left(\mathcal{E} - \mathcal{Z}^* \otimes \mathcal{Z}^* \right) \right] \cdot \left(\mathcal{E} - (B \times K)^{-1} J \right) \\
&= \left[(B \times K)^{-1} \left(\mathcal{Z}^* \sum_{k=1}^{K} \sum_{b=1}^{B} \mathcal{Z}^*_{kb} - J_{K \times B} \right) \otimes J_{K \times B} \right. \\
&\quad \left. - \mathcal{Z}^* \otimes \mathcal{Z}^* + \mathcal{E} \right] \|\mathcal{Z} - \mu_{\text{batch}}\|_F^{-1}
\end{aligned} \tag{16}$$

The equation (16) above is the derivative of $\mathcal{Z}^*$ with respect to $\mathcal{Z}$. By using the chain rule combined with the derivative of the total loss with respect to $\mathcal{Z}^*$, we can compute the derivative of the continuous hash code as $\frac{\partial L}{\partial \mathcal{Z}} = \frac{\partial L}{\partial \mathcal{Z}^*} \frac{\partial \mathcal{Z}^*}{\partial \mathcal{Z}}$. Following this, backpropagation is used to calculate the derivative of Θ.

2) TIME COMPLEXITY

This subsection investigates the computational complexity of the Relative Similarity function during forward propagation and backward propagation.

a: FORWARD PROPAGATION

After obtaining $\mathcal{Z}$ from backbone ϕ, calculating Relative Position $\mathcal{Z}^*$ in equation (4) takes $O(KB)$ since calculating μ_{batch}, $\mathcal{A}$, and γ each takes $O(KB)$. Additionally, it takes $O(KM)$ to compute normalized centroids $\mathcal{C}^*$ from $\mathcal{C}$. For equation (6), calculating the scaled dot-product between $\mathcal{Z}^* \in \mathbb{R}^{K \times B}$ and $\mathcal{C}^* \in \mathbb{R}^{K \times M}$ takes $O(MKB)$. As a result, the forward propagation of Relative Similarity has a time complexity of $O(KB+KM+MKB) = O(K(B+M+BM)) = O(KBM)$.

b: BACKWARD PROPAGATION

According to equation (10), $\frac{\partial L}{\partial \mathcal{Z}^*}$ takes $O(KMB)$ to calculate. Furthermore, equation (16) requires $O(B^2K^2)$ to calculate $\frac{\partial \mathcal{Z}^*}{\partial \mathcal{Z}}$. In step 9 of Algorithm 1, we can computationally calculate $\frac{\partial L}{\partial \mathcal{Z}}$ as follows:

$$\frac{\partial L}{\partial \mathcal{Z}_{ij}} = \sum_{k=1}^{K} \sum_{b=1}^{B} \frac{\partial L}{\partial \mathcal{Z}^*_{kb}} \frac{\partial \mathcal{Z}^*_{kb}}{\partial \mathcal{Z}_{ij}} \tag{17}$$

Based on the above equation, calculating $\frac{\partial L}{\partial \mathcal{Z}}$ takes $O(B^2K^2)$. Since $\mathcal{C}$ is fixed during training, we do not need to compute its gradient. As a result, the backward propagation has a time complexity of $O(KMB + B^2K^2 + B^2K^2) = O(KB(M + KB))$. Generally, the time complexity of Relative Similarity is $O(KBM)$ during forward propagation and $O(KB(M + KB))$ during backward propagation.

IV. EXPERIMENTS

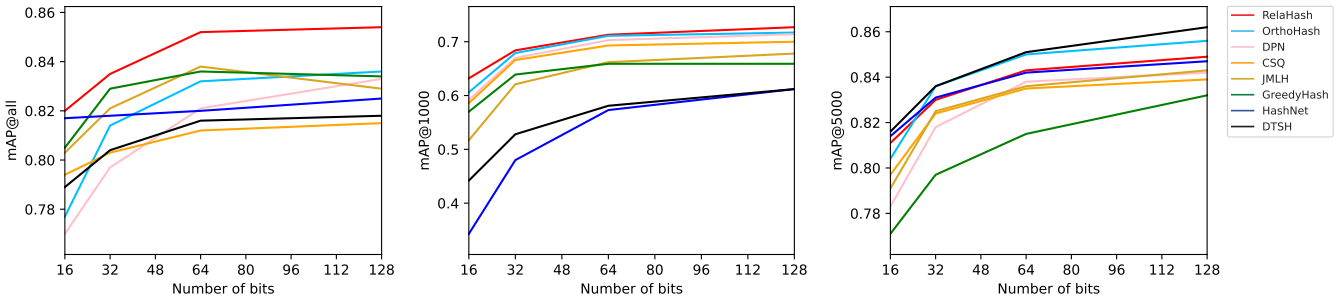
This section begins by comparing RelaHash's performance with existing state-of-the-art models on image retrieval tasks using many popular benchmark datasets. Afterward, we provide the ablation study of many hyperparameters. We then compare RelaHash with other methods from a variety of perspectives. Finally, we visualize the learned hash code and the actual retrieved results from our method.

A. SETUP

1) DATASETS

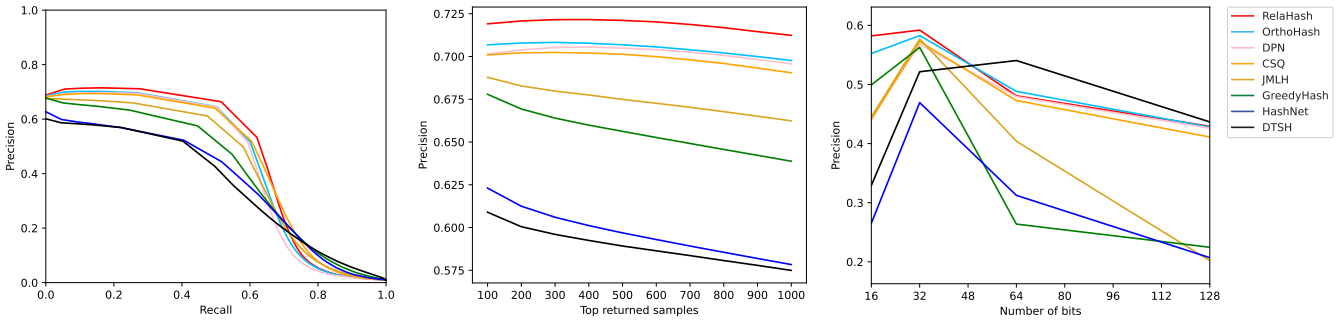
CIFAR-10 [25] is a single-label dataset, cosisting of 60K 32×32 colored images grouped into 10 categories. We follow prior settings [20], [36], [42], [48], [49], and randomly select 100 images from each class as the query set. The remaining 59000 images are used as the database. We randomly sample 500 images from each class from the database for training.

ImageNet100 is also a single-label dataset that contains only 100 classes from the ImageNet dataset [26]. Following prior settings [1], [33], [36], [37], [48], we use all the training set as the database, all the validation set as the queries, and



(a) CIFAR-10 (b) ImageNet100 (c) NUS-WIDE

FIGURE 5. Performance comparision on various benchmark datasets under mAP@R w.r.t. different bits.



(a) PR curve @64bits (b) P@N @64bits (c) P@H≤2

FIGURE 6. Experimental results of RelaHash and other methods on ImageNet100 under three evaluation metrics.

randomly sample 100 images in each class from the database for training.

NUS-WIDE [50] is a multi-label dataset containing 269,648 images divided into 81 concepts. Following prior settings [1], [33], [36], [37], [48], [51], we use the most 21 frequent concepts, select 100 images in each class as the queries, and the remaining images are used as the database. We randomly sample 500 images from each class from the database for training. Because NUS-WIDE is a multi-label dataset, we use label smoothing to generate labels for softmax loss, as in [1].

2) BASELINES

To demonstrate our model's performance, we compare our model with the seven most recent well-known state-of-the-art hashing methods, including one pair-wise method [33], one triplet-wise method [34], and five point-wise methods [1], [20], [36], [37], [48].

3) EVALUATION METRIC

Following prior works in deep hashing [42], [37], [21], we evaluate retrieval performance based on the mean of Average Precision (AP) per R retrieved items (mAP@R), Precision-Recall curve (PR curve), Precision curve w.r.t. different numbers of retrieved items (P@N), and Precision curve within Hamming distance 2 (P@H≤2). Specifically, mAP@R is calculated by averaging Average Precision across all classes.

$$\text{mAP@R} = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{18}$$

4) EXPERIMENTAL SETTINGS

We follow the training settings as in [1], as we use pre-trained AlexNet [10] as the backbone, Adam optimizer [47] with the same learning rate of 0.0001, and train for 100 epochs for all methods. For our method, the β hyperparameter and batch size B are obtained by grid search.

On **CIFAR-10**, we ran all results on NVIDIA Tesla T4 GPU provided by Google Colab with their default hyperparameter settings. Results of other methods on **ImageNet100** and **NUS-WIDE** are taken from [1] to make them directly comparable. Our results on **ImageNet100** are run on a single NVIDIA GeForce RTX 2070 SUPER GPU, and our results on **NUS-WIDE** are run on a single NVIDIA GeForce RTX 3060 GPU.

B. RESULTS

Table 1 shows retrieval performance in terms of mAP@R. On single-label datasets (CIFAR-10 and ImageNet100), our method performed the best with significant improvements compared to recent state-of-the-art methods. On CIFAR-10 dataset, our method achieved about 0.3%-4.5% improvements in mAP@R compared to others. On ImageNet100 dataset, pair-wise and triplet-wise methods such as

TABLE 2. The performance in terms of mAP@R of various β values for 4 different bits on CIFAR-10 (16 batch size) and ImageNet100 (64 batch size) datasets ($m = 0.5$). The highest mAP@R of each column is shown in **bold**. The second highest mAP@R of each column is underlined.

β	**CIFAR10** (mAP@all)				**ImageNet100** (mAP@1K)			
	16	32	64	128	16	32	64	128
$\beta = 5$	0.8050	0.8216	0.8347	0.8397	0.5787	0.6648	0.7033	0.7152
$\beta = 10$	0.7978	0.8201	0.8338	0.8396	0.5849	0.6661	0.7027	0.7184
$\beta = 25$	0.8024	0.8189	0.8374	0.8406	0.5882	0.6681	0.7053	0.7200
$\beta = 50$	0.8045	0.8285	0.8392	0.8424	0.5950	0.6719	0.7072	0.7224
$\beta = 80$	0.8095	**0.8313**	0.8385	0.8423	0.6040	0.6721	0.7079	0.7247
$\beta = 100$	**0.8110**	0.8295	0.8380	0.8409	0.6077	0.6723	0.7101	**0.7256**
$\beta = 200$	0.8092	0.8256	**0.8401**	0.8425	0.6183	0.6791	0.7083	0.7228
$\beta = 250$	0.8045	0.8296	0.8379	**0.8426**	0.6252	0.6808	**0.7122**	0.7078
$\beta = 300$	0.8072	0.8252	0.8397	0.8408	0.6280	0.6811	0.7068	0.7200
$\beta = 500$	0.8048	0.8263	0.8373	0.8409	**0.6304**	**0.6838**	0.7088	0.7167

HashNet [33] and DTSH [34] face the issue of imbalanced data, so RelaHash has a considerable improvement from 11.5% on 128-bits to 19%-28.9% on 16-bits settings. Compared with other point-wise methods, RelaHash has improvements of about 2.6%-11.5% on 16-bits settings, 0.5%-6.3% on 32-bits settings, 0.2%-5.4% on 64-bits settings and 1%-6.8% on 128-bits settings. On NUS-WIDE - a multi-label dataset, our method is comparable to current state-of-the-art point-wise methods. We have a slight boost over OrthoHash [1] on 16-bits settings while still having competitive performance on other bits. In the multi-label dataset, the data point is pulled towards multiple hash centroids simultaneously, making it more difficult for RelaHash to find the optimal hash codes. As the number of dimensions increases, the discrete hash codes can more accurately represent the continuous output, so quantization error reduction becomes less effective. In fact, trying to reduce the quantization error too much in multi-label settings can even result in overfitting. mAP@R w.r.t. different numbers of K-bits on these datasets are shown in Figure 5. Figure 6 shows retrieval performance in Precision-Recall curves (PR curve), Precision curves w.r.t. different numbers of retrieved items (P@N), and Precision curves within Hamming distance 2 (P@H$\leq$2) of compared methods on ImageNet100 with 64-bits settings. It can be observed from the graph that our method has better general retrieval performance over other methods. We thus conclude that naturally reducing quantization error on single-label datasets can lead to better performance.

C. ABLATION STUDY

This subsection examines the impact of hyperparameters β, m, and batch size B on the performance of the method. All results in this subsection are run on NVIDIA Tesla T4 GPU provided by Google Colab.

1) EFFECT OF MARGIN m

For a better understanding of how margin m affects model performance, we train our model with Adam optimizer for 100 epochs on CIFAR-10 with 64-bits settings, set $\beta = 50$, and use batch size of 16. Various margins were tested between 0.0 and 1.0. The results are presented in Figure 7. It can be observed from the graph that performance degrades when the m value passes through 0.2. Due to the cosine constraint [46], when m is too large, the feature space begins to vanish and the model fails to converge.

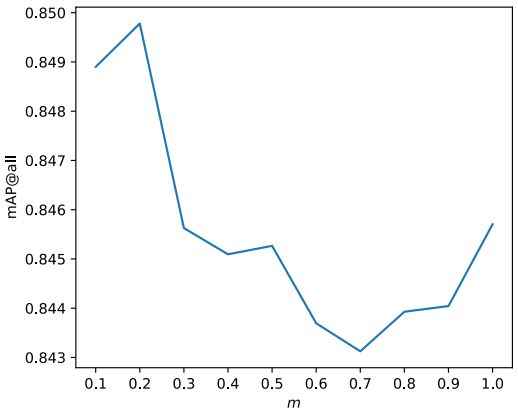


FIGURE 7. The performance analysis of hyperparameter margin m on CIFAR-10 datasets with 64-bits settings ($\beta = 50$).

2) EFFECT OF SCALE β

Various experiments with different settings were conducted in order to find the optimal parameter β and investigate how it affects our model's performance.

In the first experiment, we try different β values, starting from 0.5 with the increasing power of 2. This time we also train our model using Adam optimizer for 100 epochs on CIFAR-10 dataset with 64-bits settings, set $m = 0.5$, and using batch size of 16. The results are shown in Figure 8.

We conducted another experiment - testing β with different values in order to find how optimal β varies between bits and datasets. On both CIFAR-10 and ImageNet100, we used the Adam optimizer and stopped training when the best mAP@R

TABLE 3. The performance in terms of mAP@R of various batch size (B) values for 4 different bits on CIFAR-10 ($\beta = 80$) and ImageNet100 ($\beta = 100$) datasets. ($m = 0.5$) The highest mAP@R of each column is shown in **bold**. The second highest mAP@R of each column is underlined.

Batch size (B)	**CIFAR10** (mAP@all)				**ImageNet100** (mAP@1K)			
	16	32	64	128	16	32	64	128
$B = 16$	**0.8095**	**0.8313**	**0.8385**	**0.8423**	**0.6195**	**0.6785**	0.7060	0.7178
$B = 32$	0.7887	0.8189	0.8294	0.8402	0.6161	0.6750	0.7049	0.7216
$B = 64$	0.7801	0.8159	0.8288	0.8398	0.6077	0.6723	0.7101	0.7256
$B = 128$	0.7726	0.8111	0.8279	0.8372	0.6025	0.6746	**0.7105**	**0.7270**

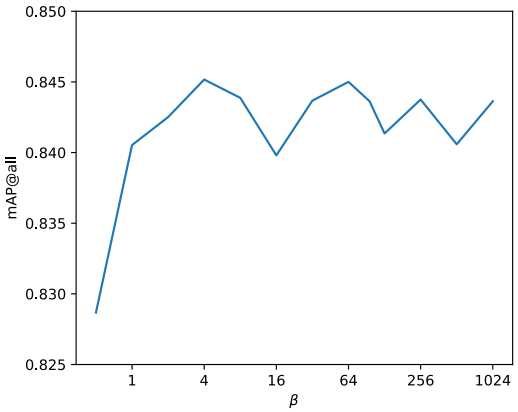


FIGURE 8. The performance analysis of hyperparameter β on CIFAR-10 datasets with 64-bits settings ($m = 0.5$).

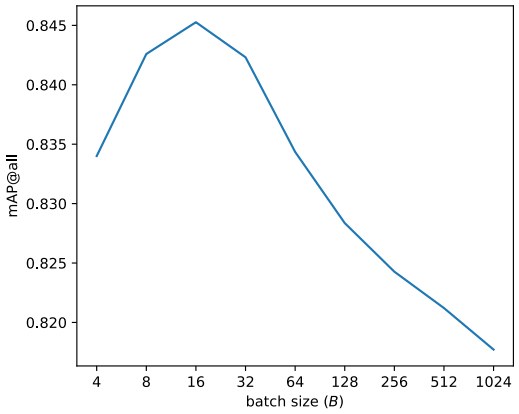


FIGURE 9. The performance analysis of batch size B on CIFAR-10 datasets with 64-bits settings ($m = 0.5$).

ceased to increase after 6 epochs - which often occurred after 12-25 epochs. A batch size of 16 was used on CIFAR-10, while a batch size of 64 was used on ImageNet100. A value of 0.5 is set for the hyperparameter m. The results are summarized in Table 2.

We can observe from Table 2 and Figure 8, the model's performance varies around 0.1%-0.5% with different choices of the β hyperparameter. As the number of bits increases, the variance in performance decreases.

3) EFFECT OF BATCH SIZE

The number of batch size B is also critical to RelaHash's performance. We experimented with various settings in order to determine the optimal batch size for each bit and dataset. For the first experiment, we set $\beta = 50$ and train our model using Adam optimizer for 100 epochs on CIFAR-10 with 64-bits settings to try different batch sizes, starting from 4 with the increasing power of 2. The results are presented in Figure 9.

Another experiment was conducted with different batch sizes of 16, 32, 64, and 128 for each bit and dataset setting. On both CIFAR-10 and ImageNet100, we used the Adam optimizer and stopped training when the best mAP@R ceased to increase after 6 epochs - which often occurred after 12-25 epochs. On CIFAR-10 we set $\beta = 80$, while on ImageNet100 we set $\beta = 100$. A value of 0.5 is set for the hyperparameter m. The obtained results are summarized in Table 3.

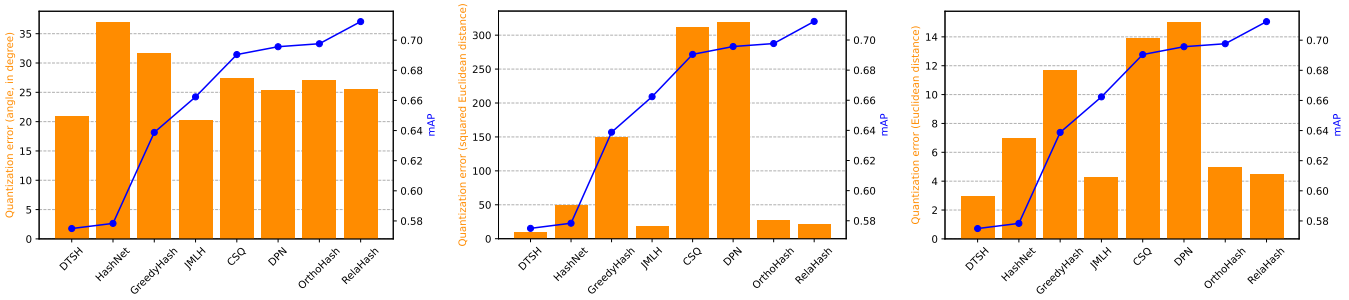
We can observe from Table 3 and Figure 9, the model's performance varies around 1%-2% with different choices of batch size. CIFAR-10 has an optimal batch size of 16 among all K-bits settings, whereas ImageNet100 has optimal batch sizes proportional to the number of bits.

D. FURTHER ANALYSIS

This subsection analyzes the performance of RelaHash compared to other methods in terms of quantization error, orthogonality, and separability. Furthermore, we also compare RelaHash's performance when LMCL is used as the same loss in other point-wise methods.

1) QUANTIZATION ERROR

Quantization error is the amount of information loss when the relaxed hash codes $\mathcal{Z}$ pass through the quantization layer, i.e., the sign function. Retrieval performance can be improved by minimizing this quantization error. Specifically, the process of mapping the continuous hash code $\mathbf{z}_n \in \mathbb{R}^K$ in the Euclidean space into the discrete binary hash code $\mathbf{h}_n \in \{-1, 1\}^K$ in the Hamming space introduces quantization error. This quantization error can be measured either by Euclidean or Angle distance, as illustrated in Figure 3. Precisely, we measure the quantization error of the database by Euclidean distance (19), squared Euclidean distance (20), and the angle $\theta_{\mathbf{z}_n\mathbf{h}_n}$ between the continuous hash code $\mathbf{z}_n$ and



(a) Angular distance $\theta_{\mathbf{z}_n \mathbf{h}_n}$ @64bits (b) squared Euclidean distance @64bits (c) Euclidean distance @64bits

FIGURE 10. Analysis of quantization error on ImageNet100 with 64-bits settings.

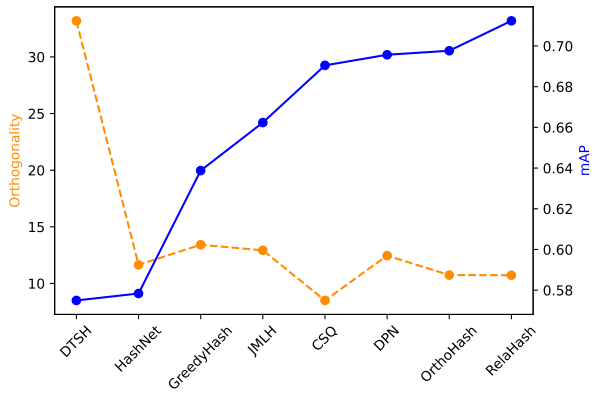


FIGURE 11. Orthogonality analysis with 64-bits ImageNet100.

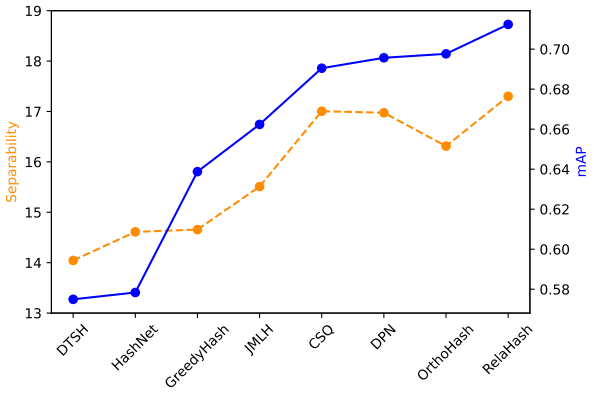


FIGURE 12. Separability analysis with 64-bits ImageNet100.

the discrete hash code $\mathbf{h}_n$ (21).

$$\frac{1}{N}\sum_{n=1}^{N}\|\mathbf{z}_n - \mathbf{h}_n\| \tag{19}$$

$$\frac{1}{N}\sum_{n=1}^{N}\|\mathbf{z}_n - \mathbf{h}_n\|^2 \tag{20}$$

$$\frac{1}{N}\sum_{n=1}^{N}\deg\left(\cos^{-1}\left(\frac{\mathbf{z}_n \cdot \mathbf{h}_n}{\|\mathbf{z}_n\|\,\|\mathbf{h}_n\|}\right)\right) \tag{21}$$

in which N is the size of the database. As depicted in Figure 10, our approach exhibits a relatively low quantization error compared to other techniques. It can be noticed that while both DTSH and JMLH have low quantization errors, they perform poorly in terms of mAP. Our method, on the other hand, has a lower quantization error compared to the majority of other methods, including OrthoHash. These findings indicate that our method is effective in reducing quantization error.

2) ORTHOGONALITY

The orthogonality of hash codes can be measured by computing the center of data points' hash codes belonging to each class in the Hamming space ($\mathcal{H}_{\text{center}}$). As a result, the orthogonality can be calculated as follows:

$$\left\|\frac{1}{K}\mathcal{H}_{\text{center}}\mathcal{H}_{\text{center}}^{\mathsf{T}} - \mathbb{I}\right\|_F \tag{22}$$

where K is the number of bits, and $\mathbb{I}$ is the identity matrix. Better performance can be achieved by lowering orthogonality. According to Figure 11, our method has relatively low orthogonality compared with other methods. The high orthogonality of DTSH indicates that its learned hash code has some redundant correlations. In contrast, CSQ has low orthogonality, meaning that each bit learned is orthogonal to the others. This is due to CSQ's use of the Haddamard matrix in generating centroids, which ensures equal distances between each centroid. As a result, we can conclude that the hash codes generated by our method have a low pairwise correlation between any two bits.

3) SEPARABILITY

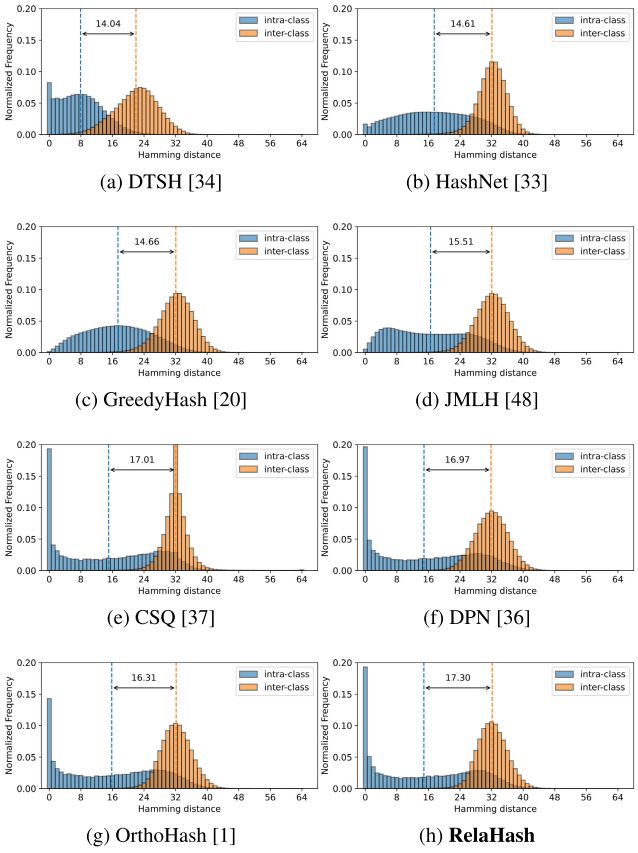
In order to get an insight into inter-class distances ($\Delta_{\text{inter-class}}$) versus intra-class distances ($\Delta_{\text{intra-class}}$) between data points, we take a look at the separability of the learned hash codes in the Hamming space $\left(\mathcal{H} \subset \{-1, 1\}^K\right)$. A histogram of intra-class and inter-class distances between compared methods is shown in Figure 13. The difference in the expectation of these two distributions is the separability in the Hamming space, expressed as follows:

$$\mathbb{E}\left[\Delta_{\text{inter-class}}\right] - \mathbb{E}\left[\Delta_{\text{intra-class}}\right] \tag{23}$$

where $\mathbb{E}[\cdot]$ is the expectation of the distances between data points in the Hamming space. The separability metric indicates how separative learned clusters are (Figure 15).

TABLE 4. Performance comparison using LMCL [46] for all pointwise methods on CIFAR-10 and ImageNet100 datasets with different K-bits settings. The highest value of each column is shown in **bold**.

Methods	CIFAR-10 (mAP@all)				ImageNet100 (mAP@1K)			
	16	32	64	128	16	32	64	128
GreedyHash [20]	0.755	0.757	0.765	0.746	0.529	0.569	0.568	0.563
JMLH [48]	0.773	0.790	0.803	0.812	0.545	0.615	0.647	0.671
OrthoHash [1]	0.774	0.801	0.810	0.807	**0.607**	**0.672**	0.697	0.705
RelaHash	**0.803**	**0.821**	**0.841**	**0.844**	0.585	0.668	**0.704**	**0.721**



(a) DTSH [34] (b) HashNet [33] (c) GreedyHash [20] (d) JMLH [48] (e) CSQ [37] (f) DPN [36] (g) OrthoHash [1] (h) **RelaHash**

FIGURE 13. Histogram of intra-class and inter-class Hamming distances with 64bits ImageNet100. The arrow annotation is the separability in Hamming distances.

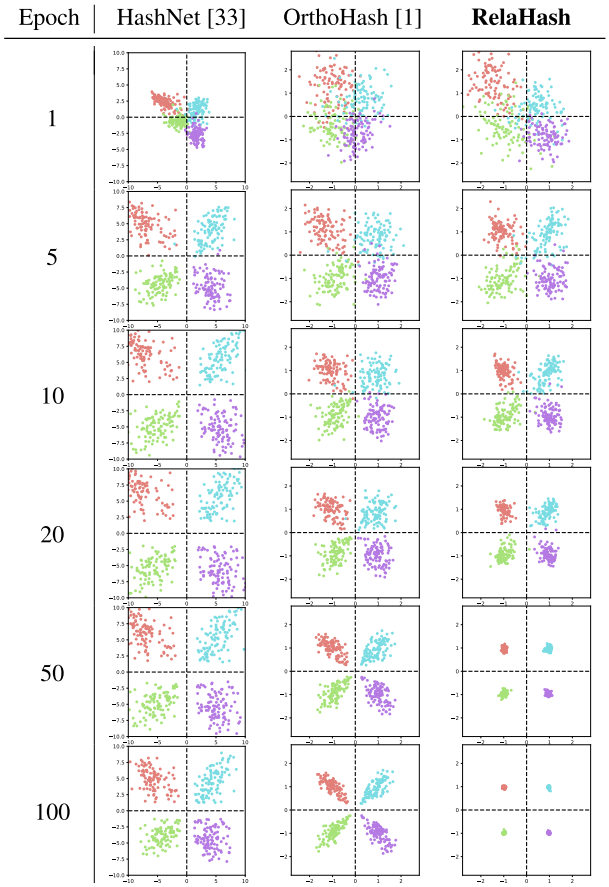
Separability increases when clusters are more separable, resulting in better retrieval performance. It can be observed that CSQ has the smallest variance in inter-class Hamming distances because its generated centroids have equal distances of $\frac{K}{2}$ from each other. The separability of RelaHash and other methods is displayed in Figure 12. As shown in the figure, our method has the highest separability, leading to the best retrieval results. Therefore, we can conclude that our method is effective in separating clusters in high-dimensional space.

4) COMPARISON USING LARGE MARGIN COSINE LOSS

This subsection compares various point-wise methods by using LMCL [46] as the same loss function. Specifically, we set $m = 0.2$ and $s = \sqrt{K}$ with the same batch size of 64, using Adam optimizer and train for 100 epochs for all methods. Results are shown in Figure 14 and summarized in table 4.

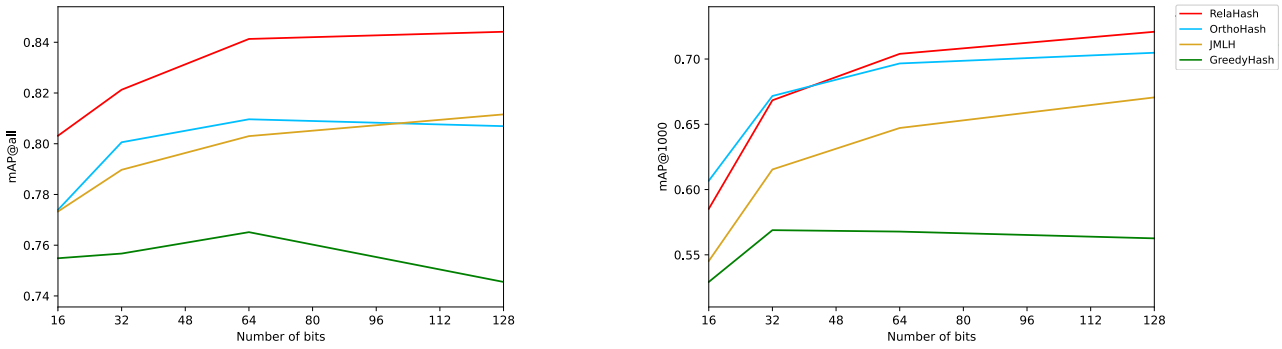
Based on the results, we are able to demonstrate that our method is still superior to other point-wise methods even when using LMCL as the loss function. On ImageNet100, our method also yields great performance when only behind OrthoHash in 16 and 32 bits settings.

TABLE 5. The visualization of continuous hash codes training process with 4 classes and 2-bits settings.



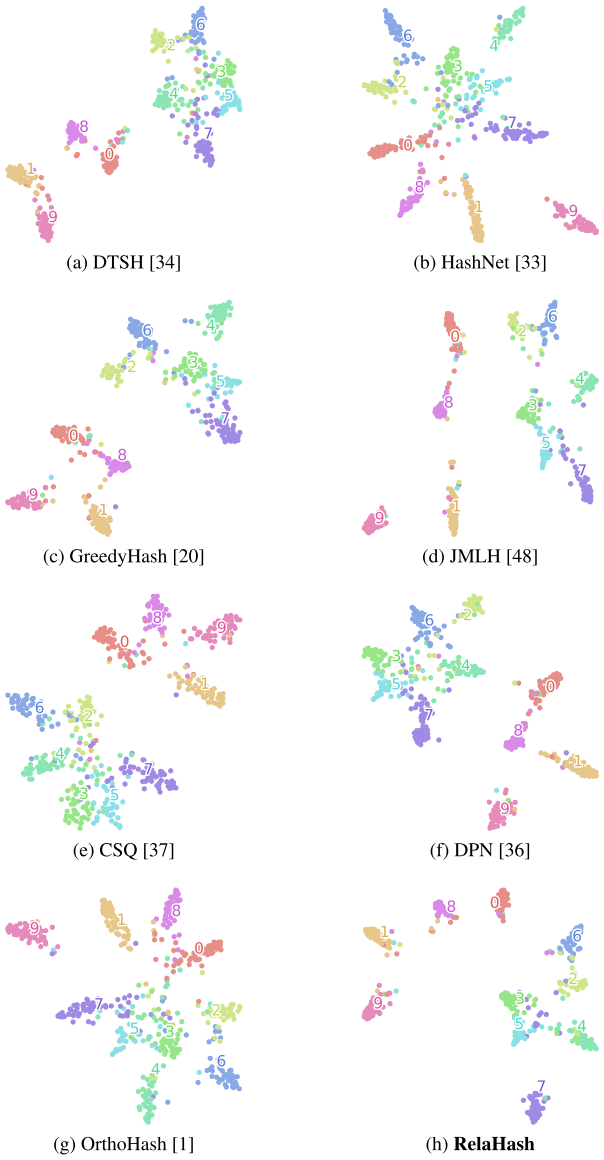
E. VISUALIZATION

For a better understanding of how RelaHash learns, we visualize the process of training their continuous hash codes in



(a) CIFAR-10 (b) ImageNet100

FIGURE 14. Performance comparision using LMCL [46] for all pointwise methods on CIFAR-10 and ImageNet100 datasets under mAP@R w.r.t. different bits.

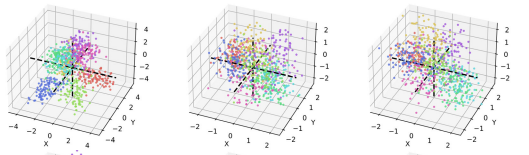
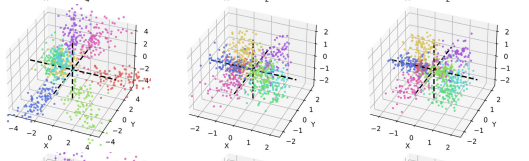
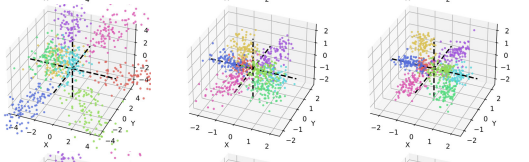
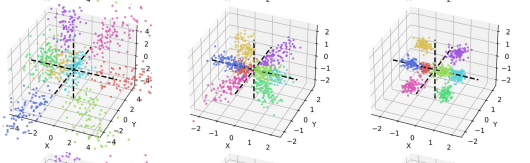
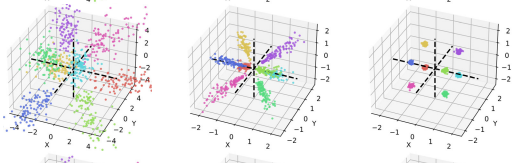
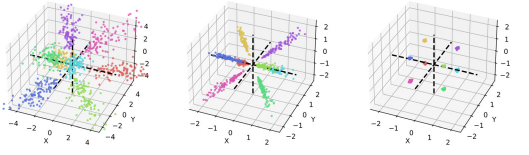


(a) DTSH [34] (b) HashNet [33]
(c) GreedyHash [20] (d) JMLH [48]
(e) CSQ [37] (f) DPN [36]
(g) OrthoHash [1] (h) **RelaHash**

FIGURE 15. The t-SNE visualization of learned hash codes $\mathcal{Z}$ on CIFAR-10.

Euclidean space and their t-SNE representation. Furthermore, we compare our retrieval results with other retrieval methods on ImageNet100 datasets.

TABLE 6. The visualization of continuous hash codes training process with 8 classes and 3-bits settings.

Epoch	HashNet [33]	OrthoHash [1]	**RelaHash**
1			
5			
10			
20			
50			
100			

1) LOW-DIMENSIONAL HASH CODES VISUALIZATION

a: 2-DIMENSIONAL HASH CODES

We train deep hashing methods using the first 4 classes of CIFAR-10 [25] dataset on 2-bits settings. The continuous (relaxed) hash codes $\mathcal{Z}$ before the quantization layer are visualized in Table 5. The learned hash codes of our method are pulled closer to their respective centroids

FIGURE 16. Examples of Top-10 retrieved results on ImageNet100.

$\mathcal{C} = \left\{\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\-1\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}, \begin{pmatrix}-1\\-1\end{pmatrix}\right\}$. Due to this, our method has lower quantization error than others while having low intra-class and high inter-class distances.

b: 3-DIMENSIONAL HASH CODES

We train deep hashing methods using 8 classes taken from CIFAR-10 [25] dataset on 3-bits settings. The continuous (relaxed) hash codes $\mathcal{Z}$ before the quantization layer are visualized in Table 6. In comparison to 2-bits settings, our method can achieve similar performance.

2) HIGH-DIMENSIONAL HASH CODES VISUALIZATION

To gain insight into high-dimensional hash codes, we used t-SNE [52] to visualize the learned continuous hash codes $\mathcal{Z}$ on the CIFAR-10 [25] dataset with 64-bits settings. As seen in Figure 15, the hash codes learned by our model exhibit better cluster separation than other methods. Data points with the same label are pulled closer together, while those with different labels are pushed away. These visualizations reinforce the conclusion in subsection IV-D3 about separability in the high-dimensional space.

3) RETRIEVED RESULTS

We show retrieval results of HashNet [33], OrthoHash [1], and our RelaHash on ImageNet100 [26] with various query images from easy to difficult, as in Figure 16. Comparing RelaHash to other methods, our proposed method appears to return more relevant results. One side note is that the ImageNet100 database and training set only contain 100 classes randomly sampled from the ImageNet-1K dataset [26], making it difficult to include many unknown objects in the database. All methods successfully return relevant results from the database on images whose class label is in the training set. As the query image becomes more challenging, such as when the class definition is vague, the label does not exist in the training set, or only a few relevant samples exist in the database, RelaHash begins to demonstrate its effectiveness.

V. CONCLUSION & FUTURE WORK

In this paper, we proposed a new approach to reducing quantization error without the need for explicit quantization loss terms. We demonstrated that low quantization error can be achieved by using an intermediate representation called Relative Position to learn compact hash codes. Our approach is robust to imbalanced datasets and has the advantage of using margin losses. Unlike conventional approaches, our method does not require explicit quantization loss terms or involve the quantization layer at training time, allowing for end-to-end training of the network. Comprehensive experiments confirmed that our method outperforms state-of-the-art methods on single-label datasets while having a competitive performance on multi-label dataset. We discovered that our method can bring adverse effects when it tries to further lower quantization error in multi-label contexts. In our study, we found that our method is effective in reducing quantization error, lowering pairwise correlation between any two bits, and separating clusters in the high-dimensional space. This work takes the field of deep hashing one step further in achieving end-to-end training without additional explicit constraints on the final loss.

In the future, we will work on finding better hash centroids, determining a systematic way to select the hyperparameter β, and applying the method to unsupervised deep hashing.

The code is available at https://github.com/thaiminhpv/RelaHash

REFERENCES

[1] J. T. Hoe, K. W. Ng, T. Zhang, C. S. Chan, Y.-Z. Song, and T. Xiang, "One loss for all: Deep hashing with a single cosine similarity based learning objective," in *Proc. Adv. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates, vol. 34, 2021, pp. 24286–24298.

[2] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proc. VLDB*, 1997, pp. 426–435.

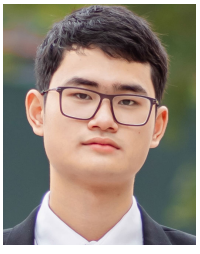
[3] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *SIGMOD Rec.*, vol. 14, no. 2, pp. 47–57, Jun. 1984.

[4] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[5] B. Xu, J. Bu, C. Chen, D. Cai, X. He, W. Liu, and J. Luo, "Efficient manifold ranking for image retrieval," in *Proc. 34th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*. New York, NY, USA: Association for Computing Machinery, 2011, pp. 525–534.

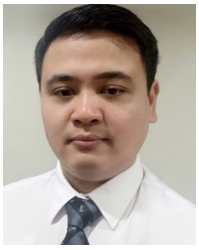
[6] M. Datar, N. Immorlica, P. Indyk, and S. V. Mirrokni, "Locality-sensitive hashing scheme based on P-stable distributions," in *Proc. Twentieth Annu. Symp. Comput. Geometry (SCG)*. New York, NY, USA: Association for Computing Machinery, 2004, pp. 253–262.

[7] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proc. IEEE ICCV*, Kyoto, Oct. 2009, pp. 2130–2137.
[8] J. Wang, T. Zhang, J. Song, N. Sebe, and H. Tao Shen, "A survey on learning to hash," 2016, *arXiv:1606.00185*.
[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, Feb. 2015.
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25. Red Hook, NY, USA: Curran Associates, 2012.
[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2014.
[12] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," 2019, *arXiv:1905.11946*.
[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.
[14] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted Windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Montreal, QC, Canada, Oct. 2021, pp. 9992–10002.
[15] A. Krizhevsky and G. Hinton, "Using very deep autoencoders for content-based image retrieval," in *Proc. ESANN*, Jan. 2011, p. 2.
[16] Y. Cao, B. Liu, M. Long, and J. Wang, "HashGAN: Deep learning to hash with pair conditional Wasserstein GAN," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1287–1296.
[17] T. Li, Z. Zhang, L. Pei, and Y. Gan, "HashFormer: Vision transformer based deep hashing for image retrieval," *IEEE Signal Process. Lett.*, vol. 29, pp. 827–831, 2022.
[18] Y. Chen, S. Zhang, F. Liu, Z. Chang, M. Ye, and Z. Qi, "TransHash: Transformer-based Hamming hashing for efficient image retrieval," in *Proc. Int. Conf. Multimedia Retr.* New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 127–136.
[19] L. Peng, J. Qian, C. Wang, B. Liu, and Y. Dong, "Swin transformer-based supervised hashing," *Int. J. Speech Technol.*, Jan. 2023.
[20] S. Su, C. Zhang, K. Han, and Y. Tian, "Greedy hash: Towards fast optimization for accurate hash coding in CNN," in *Advances in Neural Information Processing Systems*, vol. 31. Red Hook, NY, USA: Curran Associates, 2018.
[21] Y. Cao, M. Long, B. Liu, and J. Wang, "Deep Cauchy hashing for Hamming space retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 1229–1237.
[22] W.-J. Li, S. Wang, and W.-C. Kang, "Feature learning based deep supervised hashing with pairwise labels," in *Proc. 25th Int. Joint Conf. Artif. Intell. (IJCAI)*. New York, NY, USA: AAAI Press, 2016, pp. 1711–1717.
[23] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 2064–2072.
[24] K. D. Doan, P. Yang, and P. Li, "One loss for quantization: Deep hashing with discrete Wasserstein distributional matching," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 9437–9447.
[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, vol. 1, no. 4.
[26] 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. (CVPR)*, Jun. 2009, pp. 248–255.
[27] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. 25th Int. Conf. Very Large Data Bases (VLDB)*. San Francisco, CA, USA: Morgan Kaufmann, 1999, pp. 518–529.
[28] X. Luo, H. Wang, D. Wu, C. Chen, M. Deng, J. Huang, and X.-S. Hua, "A survey on deep hashing methods," 2020, *arXiv:2003.03369*.
[29] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to hash for indexing big data—A survey," 2015, *arXiv:1509.05472*.
[30] A. Redaoui and K. Belloulata, "Deep feature pyramid hashing for efficient image retrieval," *Information*, vol. 14, no. 1, p. 6, Dec. 2022.
[31] L. Jin, X. Shu, K. Li, Z. Li, G.-J. Qi, and J. Tang, "Deep ordinal hashing with spatial attention," *IEEE Trans. Image Process.*, vol. 28, no. 5, pp. 2173–2186, May 2019.
[32] H. Lai, P. Yan, X. Shu, Y. Wei, and S. Yan, "Instance-aware hashing for multi-label image retrieval," *IEEE Trans. Image Process.*, vol. 25, no. 6, pp. 2469–2479, Jun. 2016.
[33] Z. Cao, M. Long, J. Wang, and P. S. Yu, "HashNet: Deep learning to hash by continuation," 2017, *arXiv:1702.00758*.
[34] X. Wang, Y. Shi, and K. M. Kitani, "Deep supervised hashing with triplet labels," 2016, *arXiv:1612.03900*.
[35] B. Liu, Y. Cao, M. Long, J. Wang, and J. Wang, "Deep triplet quantization," 2019, *arXiv:1902.00153*.
[36] L. Fan, K. W. Ng, C. Ju, T. Zhang, and C. S. Chan, "Deep polarized network for supervised learning of accurate binary hashing codes," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Yokohama, Japan, Jul. 2020, pp. 825–831.
[37] L. Yuan, "Central similarity quantization for efficient image and video retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Seattle, WA, USA, Jun. 2020, pp. 3080–3089.
[38] J. T. Hoe, K. W. Ng, C. S. Chan, Y.-Z. Song, T. Zhang, and T. Xiang. *Supplementary Material for One Loss for All: Deep Hashing With a Single Cosine Similarity Based Learning Objective*. Accessed: Feb. 26, 2023. [Online]. Available: https://openreview.net/attachment?id=2pJZSVcSZz&name=supplementary_material
[39] Y. Li and J. van Gemert, "Deep unsupervised image hashing by maximizing bit entropy," 2020, *arXiv:2012.12334*.
[40] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.
[41] C. Xu, Z. Chai, Z. Xu, H. Li, Q. Zuo, L. Yang, and C. Yuan, "HHF: hashing-guided Hinge function for deep hashing retrieval," *IEEE Trans. Multimedia*, early access, Nov. 16, 2022, doi: 10.1109/TMM.2022.3222598.
[42] C. Zhou, L.-M. Po, Y. F. W. Yuen, K. W. Cheung, X. Xu, K. W. Lau, Y. Zhao, M. Liu, and H. W. P. Wong, "Angular deep supervised hashing for image retrieval," *IEEE Access*, vol. 7, pp. 127521–127532, 2019.
[43] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik, "Angular quantization-based binary codes for fast similarity search," in *Advances in Neural Information Processing Systems*, vol. 25, F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012.
[44] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "SphereFace: Deep hypersphere embedding for face recognition," 2017, *arXiv:1704.08063*.
[45] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks," in *Proc. 33rd Int. Conf. Mach. Learn.*, New York, NY, USA, vol. 48, 2016, pp. 507–516.
[46] H. Wang, "CosFace: Large margin cosine loss for deep face recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 5265–5274.
[47] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, Dec. 2014, pp. 1–9.
[48] Y. Shen, J. Qin, J. Chen, L. Liu, F. Zhu, and Z. Shen, "Embarrassingly simple binary representation learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Seoul, South Korea, Oct. 2019, pp. 2883–2892.
[49] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," 2015, *arXiv:1504.03410*.
[50] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "NUS-WIDE: A real-world web image database from National University of Singapore," in *Proc. CIVR*. New York, NY, USA: Association for Computing Machinery, Jul. 2009, pp. 1–9.
[51] H.-F. Yang, K. Lin, and C.-S. Chen, "Supervised learning of semantics-preserving hash via deep convolutional neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 2, pp. 437–451, Feb. 2018.
[52] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.



PHAM VU THAI MINH was born in Hanoi, Vietnam. He is currently pursuing the B.Sc. degree in IT with ICT Department, FPT University, Hanoi. From 2022 to 2023, he was a Research Assistant with the SAP-LAB Innovation Laboratory, FPT University. Since 2023, he has been a Data Scientist with Coc Coc. His current research interests include computer vision, deep learning, image/video indexing and retrieval, and image processing.

NGUYEN DONG DUC VIET was born in Hai Duong, Vietnam. He is currently pursuing the B.Sc. degree in IT with ICT Department, FPT University, Hanoi, Vietnam. Since 2019, he has been a Research Assistant with the SAPLAB Innovation Laboratory, FPT University. From 2021 to 2022, he was with FPT Software, as a Full-Stack Engineer. His current research interests include computer vision, image/video processing, object detection, and anomaly detection.



NGO TUNG SON received the bachelor's degree in computing from the Hanoi University of Science and Technology and the master's degree in computer science from La Rochelle University, France. He is currently pursuing the Ph.D. degree in information technology with Universiti Teknologi Petronas (UTP), Malaysia. From 2008 to 2014, he was a software engineer in some local and international companies in the software development industry. From 2014 to 2016, he joined the Panasonic Research and Development Center, Hanoi, Vietnam, as a Senior Research and Development Engineer. Since 2016, he has been with FPT University, as a Lecturer in information technology. He was the Team Leader of the Intelligent and Adaptive System with SAP-LAB Innovation, FPT University, in 2019. His research interests include artificial intelligence, adaptive and intelligent systems, and decision support systems. He has published more than 40 research articles in his research fields.

BUI NGOC ANH received the bachelor's and master's degrees in computer science from the Hanoi University of Technology. Since 2003, he has been worked on various projects in the software industry. With nearly 20 years of experience in the software industry, he is one of the experts with FPT Technology Corporation, Vietnam. In 2011, he started his work as an IT Lecturer with FPT University, Vietnam. Since 2019, he has been the Head of Computing Fundamentals Department, FPT University, where he is currently the Leader of the SAP-LAB FPT Laboratory. His research interests include big data, data mining, computer vision, and software engineering.

JAFREEZAL JAAFAR (Senior Member, IEEE) received the B.Sc. degree in computer science from Universiti Teknologi Malaysia, in 1998, the M.App.Sc. degree in IT from RMIT University, Australia, in 2002, and the Ph.D. degree from the University of Edinburgh, U.K., in 2009. He joined the Department of Computer and Information Sciences, Universiti Teknologi Petronas (UTP), Malaysia, in 1999, where he was appointed as the Head, from 2012 to 2016. He was the former Head of the Center for Research in Data Science, from 2017 to 2020, which focuses on the implementation of machine learning and predictive analytic in oil and gas (O&G) industry. He is currently an Associate Professor and the Dean of the Faculty of Science and Information Technology, UTP. He is also active in conducting professional training in AI and big data analytic for the public and industry. His main research interests include AI, machine learning, and data analytics, with over 100 technical publications. He was a member of the Academy Science Malaysia SIG in machine learning, a Senior Member, and an Executive Committee Member of IEEE CS Malaysia Chapter, from 2016 to 2018, and MyAIS, from 2017 to 2019. Based on his experience and expertise, he has been awarded as the Professional Technology (P.Tech.) by the Malaysia Board of Technology.