Deep Supervised Hashing with Spherical Embedding

Deep Supervised Hashing with Spherical Embedding

Stanislav Pidhorskyi[1], Quinn Jones[1], Saeid Motiian[2], Donald Adjeroh[1], and Gianfranco Doretto[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26508, USA
{stpidhorskyi, qjones1, daadjeroh, gidoretto}@mix.wvu.edu
[2] Adobe Applied Research, San Francisco, CA 94103, USA
motiian@adobe.com

Abstract. Deep hashing approaches are widely applied to approximate nearest neighbor search for large-scale image retrieval. We propose Spherical Deep Supervised Hashing (SDSH), a new supervised deep hashing approach to learn compact binary codes. The goal of SDSH is to go beyond learning similarity preserving codes, by encouraging them to also be balanced and to maximize the mean average precision. This is enabled by advocating the use of a different relaxation method, allowing the learning of a spherical embedding, which overcomes the challenge of maintaining the learning problem well-posed without the need to add extra binarizing priors. This allows the formulation of a general triplet loss framework, with the introduction of the spring loss for learning balanced codes, and of the ability to learn an embedding quantization that maximizes the mean average precision. Extensive experiments demonstrate that the approach compares favorably with the state-of-the-art while providing significant performance increase at more compact code sizes.

1 Introduction

Indexing and searching large-scale image databases leverage heavily hashing based approximate nearest neighbor search technology. The goal of hashing is to map high dimensional data, such as images, into compact codes in a way that visually, or semantically similar images are mapped into similar codes, according to the Hamming distance. Given a query image, hierarchically structured based methods can then be used for the rapid retrieval of the neighbors within a certain distance from the query [1].

Recent data-dependent methods for hashing images (as opposed to data-independent methods [12]) leverage deep neural networks due to their ability to integrate the image feature representation learning with the objective of the hashing task [9, 21, 45, 42, 3], leading to a superior efficiency and compactness of the codes. However, so far the focus of these approaches has been on designing architectures only with the similarity preserving goal of mapping similar images to similar codes. On the other hand, hashing methods based on hand-crafted image features [28], have improved performance also by requiring codes to have certain properties, for example, to be balanced, uncorrelated, or to be obtained with a small quantization error [44, 39, 36, 13, 30, 26, 25].

Balanced codes are such that bits partition data in equal portions [44, 39]. It is a desirable property because it increases the variance of bits since they approach 50%

chance of being one or zero. Also code bits that are uncorrelated, or even better independent, increase their information content. In addition, learning hash functions generally require solving intractable optimization problems that are made tractable with the continuous relaxation of the codomain of the function. This means that quantization is required to discretize the continuous embedding into codes, and controlling the quantization error has been shown to improve performance [13].

In this work, we propose a deep supervised hashing approach that goes beyond similarity preserving, and that aims at learning hashing functions that map onto codes with good quality, by encouraging them to be *balanced*, and to *maximize the mean average precision*. We do so by advocating the use of a different continuous relaxation strategy that has several advantages, incto overcome the challenge of maintaining the learning problem well-posed, without the addition of costly priors that typically encourage a binary output, and that have led deep hashing approaches to focus, so far, only on similarity preserving codes. This approach allows learning a hashing function composed of a *spherical embedding* followed by an optimal *quantization*. Specifically, the embedding is based on a convolutional neural network, learned with a triplet loss framework [34], that has the advantage of being more general, because it allows the use of different triplet losses, and it allows introducing a new *triplet spring loss* that aims at learning balanced codes. Moreover, the loss framework is rotation invariant in the embedded space, which allows optimizing the quantization for a rotation that provides the highest mean average precision. We call the resulting model *Spherical Deep Supervised Hashing (SDSH)*, and provides state-of-the-art performance on standard benchmarks, including a significantly greater performance at more compact code dimensions.

2 Related Work

The existing variety of data-dependent, learning based hashing methods can be categorized into unsupervised and supervised methods [40].

Unsupervised methods [44, 13, 27, 20, 19, 15, 11, 16] use unlabeled data to learn a hashing function that preserves some metric distance between data points. This work instead falls into the supervised category, which tries to improve the quality of hashing by leveraging label information to learn compact codes. Supervised methods can be divided into those which use off-the-shelf visual features versus those that leverage deep networks. Representative examples of non-deep methods include Minimal Loss Hashing (MLH) [31], Supervised Hashing with Kernels (KSH) [26] and Latent Factor Hashing (LFH) [46].

The initial attempts to utilize deep learning [18, 37, 38, 41] for hashing included CNNH [45] and DNNH [21]. Deep Hashing Network (DHN) [50] and Deep Supervised Hashing (DSH) [24] extend DNNH by performing a continuous relaxation of the intractable discrete optimization by introducing a quantization error prior which is controlled by a quantization loss. DHN uses a cross entropy loss to link the pairwise Hamming distances with the pairwise similarity labels, while DSH uses max-margin loss. Deep Cauchy Hashing (DCH) [2] improves DHN by utilizing Cauchy distribution. Deep Pairwise-Supervised Hashing (DPSH) [23] uses pairwise similarity labels and a loss-function similar to LFH, which maximizes the log-likelihood of the pairwise

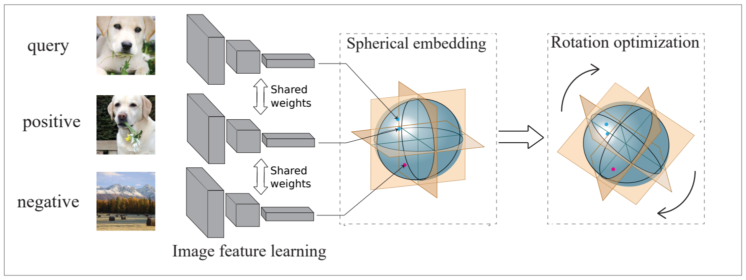

Fig. 1. Overview. Overview of the approach, highlighting the stages of the hash embedding learning, and the optimal quantization.

similarities. The log-likelihood is modeled as a function of the Hamming distance between the corresponding data points. Deep Triplet-Supervised Hashing (DTSH) [42] extends DPSH by using triplet label information.

The learning problem of deep hashing methods (DHN, DSH, DPSH, DTSH, etc.) turns out to be NP-complete, due to the discrete nature of the codomain of the hash function being sought, which is the Hamming space. The workaround is to relax the native space into the continuous Euclidean counterpart. This makes the original learning problem ill-posed, and a regularizing prior becomes necessary, which is often chosen to encourage the sought mapping to produce a nearly binary output. While needed, such prior complicates the training and might lead to performance reduction. Discrepancy Minimizing Deep Hashing (DMDH) [7] suggests an alternating optimization approach with a series expansion of the objective. Our work instead, leverages a different relaxation, which has the advantage of maintaining a well-posed learning problem, without the need for extra priors. Besides the obvious computational advantage, the framework allows to identify a class of triplet losses, and to define new ones, tailored to seeking good hash functions.

Another line of work, like Deep Quantization Network (DQN) [4], avoid the use of relaxation by incorporating quantization methods into the approach [10, 48, 43]. DQN performs a joint learning of image representations and a product quantization for generating compact binary codes. Deep Visual-Semantic Quantization (DVSQ) [3] extends DQN by adding semantic knowledge extracted from the label space. In this way, hash codes are directly optimized. While being an interesting direction, it significantly increases the complexity of the learning process. Indeed, that might be one of the contributing factors that make our approach comparing favorably against those.

Finally, our approach also considers the quantization problem. Indeed, the proposed relaxation suggests learning a spherical embedding, which is an equivalence class of solutions, because the loss turns out to be rotation invariant with respect to the embedded spherical space. This allows picking, as a solution, a representative of the class that will affect the quantization of the spherical embedding in such a way that it directly maximizes the mean average precision. This is different from previous approaches, and it is different also from approaches like Iterative Quantization (ITQ) [13], which is unsupervised, and it aims at minimizing the quantization error. Our comparison with ITQ shows that linking the quantization directly to the retrieval metric leads to better solutions.

3 Problem Overview

Given a training set of N images $\mathcal{I} = \{I_1, \cdots, I_N\}$, with labels $\mathcal{Y} = \{y_1, \cdots, y_N\}$, we are interested in learning a *hash function* h that maps an image I onto a compact binary code $\mathbf{b} = h(I) \in \{+1, -1\}^B$ of length B. The typical approach based on deep learning assumes that given three images I_i, I_j, I_k, with labels y_i, y_j, y_k, such that $y_i = y_j$, and $y_i \neq y_k$, then the hash function should be such that the corresponding binary codes $\mathbf{b}_i$ and $\mathbf{b}_j$ should be close, while $\mathbf{b}_i$ and $\mathbf{b}_k$ should be far away in the Hamming space. If $d_H(\cdot, \cdot)$ indicates the Hamming distance, this means that $d_H(\mathbf{b}_i, \mathbf{b}_j)$ should be as small as possible, while $d_H(\mathbf{b}_i, \mathbf{b}_k)$ should be as large as possible.

In addition to that, ideally, we would want to encourage hash codes to be maximally informative, where bits are independent and balanced, and to ultimately maximize the mean average precision (mAP). These aspects have been of secondary importance thus far, because deep learning approaches have to use binarizing priors to regularize loss functions that are already computationally intensive to optimize.

We overcome the major hurdle of the binarizing prior by advocating the use of a different relaxation method, which does not require additional priors, and learns a *spherical embedding*. This modeling choice has ripple effects. Besides simplifying the learning by eliminating the binarizing prior, it enables a unified formulation of a class of triplet losses [34] that are *rotation invariant*, and it allows to introduce one, which we name *spring loss*, that encourages balanced hash codes. In addition, the rotation invariance allows us to look for a rotation of the embedding hypersphere that leads to its optimal *quantization* for producing hash codes that directly maximize the retrieval mAP. This two-stage approach is depicted in Figure 1.

4 Hash Function Learning

The desired hash function should ensure that the code $\mathbf{b}_i$ of image I_i would be closer to all other codes $\mathbf{b}_j$ of I_j because $y_i = y_j$, than it would be to any code $\mathbf{b}_k$ of I_k since $y_i \neq y_k$. Therefore, if $\mathcal{T} = \{(i, j, k) | y_i = y_j \neq y_k\}$ is the set of the allowed triplet labels, then we certainly desire this condition to be verified

$$d_H(\mathbf{b}_i, \mathbf{b}_j) < d_H(\mathbf{b}_i, \mathbf{b}_k) \quad \forall (i, j, k) \in \mathcal{T} \; . \tag{1}$$

A loss function that aims at achieving condition (1) could simply be written as

$$L(h) = \sum_{(i,j,k) \in \mathcal{T}} \ell(d_H(\mathbf{b}_i, \mathbf{b}_j) - d_H(\mathbf{b}_i, \mathbf{b}_k)) \tag{2}$$

where $\ell(\cdot) : \mathbb{R} \to [0, +\infty)$ is the cost for a triplet that does not satisfy (1). Equation (2) is a more general version of the well known triplet loss [34].

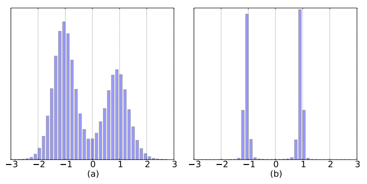

Fig. 2. Embedding distribution. Bimodal distribution of the hash embedding components (a) early on during training, and (b) at advanced training stage, when modes are separated.

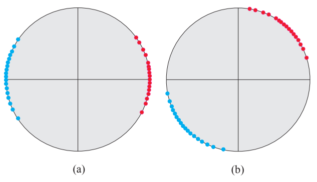

Fig. 3. Quantization. (a) Distribution of hash embeddings on the unit circle for two classes. The sign quantization assigns different hash codes to samples in the same class. (b) Rotated distribution of hash embeddings. The sign quantization assigns same hash codes to samples in the same class, thus increasing the mAP.

As pointed out in [42], it is easy to realize that

$$d_H(\mathbf{b}_i, \mathbf{b}_j) - d_H(\mathbf{b}_i, \mathbf{b}_k) = \frac{1}{2}\mathbf{b}_i^\top \mathbf{b}_k - \frac{1}{2}\mathbf{b}_i^\top \mathbf{b}_j \; . \tag{3}$$

In particular, the Hamming space where codes are defined, and through which depends the estimation of the hash function h, makes the optimization of (2) intractable [23]. Therefore, the typical approach is to relax the domain of $\mathbf{b}$ from the Hamming space to the continuous space $\mathbb{R}^B$ [23, 42]. However, this method has severe drawbacks. The first and most important one is that optimizing (2) becomes an ill-posed problem, with trivial solutions corresponding to pulling infinitely apart relaxed codes with label mismatch. This forces the introduction of a regularizing prior to the loss, which typically is designed to encourage the relaxed code $\tilde{\mathbf{b}}$ to be also “as binary as possible”, or in other words, to stay close to one of the vertices of the Hamming space.

Computationally, adding a prior is a major setback because it increases the number of hyperparameters at best, with all the consequences. In addition, if we look at the distribution of the values of the components of $\tilde{\mathbf{b}}$, we have observed experimentally that as the two main modes around $+1$ and -1 become separated, the corresponding hash codes, obtained simply by taking $\mathbf{b} = \operatorname{sgn}(\tilde{\mathbf{b}})$, stop changing during the training procedure. See Figure 2. This “locking” behavior might prevent from learning a hash function that could potentially be more efficient if it still had room to adjust the outputs. Finally, we note that (3) does not hold in the relaxed space, meaning that

$$\left\| \frac{\tilde{\mathbf{b}}_i - \tilde{\mathbf{b}}_j}{2} \right\|^2 - \left\| \frac{\tilde{\mathbf{b}}_i - \tilde{\mathbf{b}}_k}{2} \right\|^2 \neq \frac{1}{2}\tilde{\mathbf{b}}_i^\top \tilde{\mathbf{b}}_k - \frac{1}{2}\tilde{\mathbf{b}}_i^\top \tilde{\mathbf{b}}_j \; . \tag{4}$$

even though there are approaches that rely on the left-hand-side of (4) being approximately equal to the right-hand-side [42].

The following section addresses the drawbacks outlined above by advocating the use of a different relaxation for hash function learning.

5 Spherical Embedding

The hash function learning problem can be summarized as learning a function $\tilde{h}$ such that $h(I) = \mathrm{sgn}[\tilde{h}(I)]$, where $\tilde{h}$ optimizes a relaxed version of (2). Differently from previous hashing work, we propose to use a relaxation where $\tilde{h}$ is a *spherical embedding*, meaning that we constrain the output $\mathbf{s} = \tilde{h}(I)$ to be defined on the $(B-1)$-dimensional unit sphere. This means that, using the previous notation, $\mathbf{s} \doteq \tilde{\mathbf{b}}/\|\tilde{\mathbf{b}}\|$, and the meaning of Equations (1), (2), and (3) remain valid by simply substituting $\mathbf{b}$ with $\mathbf{s}$, and $d_H(\mathbf{b_i}, \mathbf{b_j})$ with $\|(\mathbf{s}_i - \mathbf{s}_j)/2\|^2$. Therefore, we advocate the end-to-end learning of a function $\tilde{h}$, given by minimizing the loss

$$L(\tilde{h}) = \sum_{(i,j,k)\in\mathcal{T}} \ell(\mathbf{s}_i^\top \mathbf{s}_k - \mathbf{s}_i^\top \mathbf{s}_j) \ . \tag{5}$$

This approach, also used in [34] to regularize the spreading of the embedding, is leveraged here to address the limitations of the continuous relaxation described in Section 4. Indeed, a spherical embedding makes the optimization of the relaxed version of (2) (which is (5)) a well-posed problem, and this was the main reason why previous works required a regularizing prior. In addition, previous priors encouraged the embedding space to be "as binary as possible" by moving $\tilde{\mathbf{b}}$ closer to a vertex of the Hamming cube, without direct evidence that this was producing better hash codes. On the other hand, we observed this practice to encourage a "locking" behavior, wich we do not have with the spherical embedding because there are no forces pushing towards the Hamming cube, and $\mathbf{s}$ is free to move on the unit sphere. Moreover, if $\tilde{h}$ is an optimal spherical embedding, so is $R\tilde{h}$, where R is a rotation matrix, since it still minimizes (5). Therefore, since (5) is *rotation invariant*, $\tilde{h}$ is found modulo a rotation, which can be estimated at a later stage to optimize other hash code properties (Section 6).

In Section 7 we design different triplet loss functions $\ell(\cdot)$, leading to different spherical embeddings. In practice, the spherical embedding comprises a convolutional neural network with a number of layers aiming at learning visual features from images, followed by fully connected layers with an output dimension equal to the number of bits B of the hash code. We adopt the VGG-F architecture [5], and replace the last fully connected layer, but other architectures can also be used [18, 14].

6 Quantization

Given an image I, its hash code is computed with a *sign quantization* as $\mathbf{b} = \mathrm{sgn}(\tilde{h}(I))$. Since $\tilde{h}$ minimizes (5), we observed that also a rotated version $R\tilde{h}$ does, so it is important to analyze the difference between the two solutions. Figure 3(a) shows a case where the spherical embeddings of two classes along the unit circle are such that the sign quantization assigns different hash codes to samples of the same class. Therefore, a rotation R could be applied to the embeddings as in Figure 3(b), where the sign quantization would now produce the expected results.

We propose to use the extra degrees of freedom due to the *rotation invariance* of (5) for learning a rotation matrix R that finds the quantization that produces the best hash function. We do so by estimating the rotation R that maximizes the mean average precision (mAP), which is the metric that we value the most for retrieval

$$\hat{R} = \arg\max_{R} \text{mAP}(R) \ . \tag{6}$$

Since mAP$(\cdot)$ is not a smooth function, and has zero gradient almost everywhere, (6) is not easy to optimize, even with derivative-free methods. On the other hand, we found that a standard random search optimization with a linear annealing schedule allows to achieve good results quickly. At the i-th iteration we apply a random perturbation $Q^{(i)}$ to the current rotation matrix $R^{(i)}$ to obtain the update $R^{(i+1)} = Q^{(i)}R^{(i)}$, which we retain if it improves the mAP. Since the perturbation should be random, uniform, and with a controllable magnitude, we generate it by setting $Q^{(i)} = P^{(i)}E(\theta){P^{(i)}}^\top$, where $P^{(i)}$ is a random unitary matrix, generated with a simplified approach based on the SVD decomposition of a matrix with normally sampled elements [29, 32]. $E(\theta)$ instead represents a rotation by θ on the plane identified by the first two basis vectors. The angle θ defines the perturbation magnitude and varies linearly with the iteration number, starting with $\theta_0 = 1.0$ down to 0 when the maximum number of iterations is reached, which was 800 in our experiments. Algorithm 1 summarizes the steps. We compute the mAP with a C++ implementation, where we take 1000 samples from the training set as queries and 16000 samples as database, or a smaller number if the training set is smaller. With a PC workstation with CPU Core i7 5820K 3.30GHz the running time for one iteration update is around 0.4s, keeping the time for the random search optimization very small, if compared with the time for training the deep network of the spherical embedding.

We now note that if R is an optimal solution, by swapping two columns of R we obtain a new solution, corresponding to swapping two bits in all hash codes. Since there are $B!$ of these kind of changes, it means that the order of growth of the solution space is $O(B!)$. Therefore, as B increases, estimating R according to (6) becomes less important because the likelihood that a random R is not far from an optimal solution has increased accordingly. The experimental section supports this observation.

7 Triplet Spherical Loss

In this section we give three examples of *rotation invariant* triplet loss ℓ that can be used in (5), namely the *margin loss*, the *label likelihood loss*, and the new *spring loss*. To shorten the notation, we define the quantity $d_{i,j,k} \doteq \mathbf{s}_i^\top \mathbf{s}_k - \mathbf{s}_i^\top \mathbf{s}_j$.

7.1 Margin Loss

The first loss that we consider is well known, and stems from requiring condition (1) to be verified with a certain margin α, in combination with using the standard hinge loss. This translates into the following *margin loss*

$$\ell(d_{i,j,k}) = \max\{0, d_{i,j,k} + \alpha\} \ . \tag{7}$$

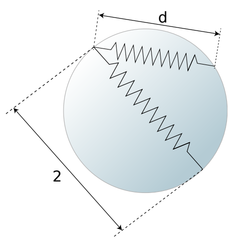

Fig. 4. Spring loss. Unit sphere where two points with different class labels are pulled apart by an elastic force proportional to the displacement $2-d$, while constrained to remain on the sphere.

7.2 Label Likelihood Loss

The second loss has been originally proposed in [42], where it was used with the Hamming space relaxed into the continuous space $\mathbb{R}^B$ for learning a hashing function. Here we extend it for learning a spherical embedding. The loss is derived from a probabilistic formulation of the likelihood of the triplet labels $\mathcal{T}$, where triplets that verify condition (1) by bigger margins have a bigger likelihood, and where the margin parameter α can affect the speed of the training process. This *label likelihood loss*, adapted to our framework becomes

$$\ell(d_{i,j,k}) = d_{i,j,k} + \alpha + \log(1 + e^{-d_{i,j,k}-\alpha}) \; . \tag{8}$$

7.3 Spring Loss

While both the margin loss and the label likelihood loss produce remarkable results, none of them make explicit efforts towards clustering samples in the spherical embedding space, according to their classes, and in a way that classes cover the sphere in a spatially uniform manner. This last property is very important because if we assume an equal number of samples per class, for each bit b, balanced codes satisfy the property

$$\sum_{i=1}^{N} h_b(I_i) = 0, \quad b = 1, \cdots, B \; . \tag{9}$$

Therefore, we note that condition (9) is satisfied whenever the spherical embedding distributes the classes uniformly on the unit sphere, thus producing balanced codes, where code bits have higher variance and are more informative. This has motivated the design of the loss that we introduce.

Let us consider two points $\mathbf{s}_i$ and $\mathbf{s}_k$ on the $(B-1)$-dimensional sphere of unit radius, and let us assume that a spring is connecting them. The Euclidean distance $d = \|\mathbf{s}_i - \mathbf{s}_k\| \doteq \sqrt{d_{i,k}}$, between the points varies in the range $[0, 2]$. At distance 2, we consider the spring unstretched, while at distance $d < 2$, the spring will have accumulated an elastic potential energy proportional to $(2-d)^2$. See Figure 4. This suggests that we could train the hash embedding by minimizing (5), with $\ell(d_{i,k}) = (2 - \sqrt{d_{i,k}})^2$,

```
Result: Returns the optimal matrix $R$ according to a random search
$R^{(0)} = I$;
$i = 0$ ;
while $i <$ number of iterations do
    // f(i) is a linear annealing schedule to update the
       rotation magnitude
    $\theta^{(i)} \leftarrow f(i)$; $E^{(i)} \leftarrow E(\theta^{(i)})$; $P^{(i)} \leftarrow$ random unitary matrix ;
    $Q^{(i)} \leftarrow P^{(i)} E^{(i)} {P^{(i)}}^{\top}$; $R' \leftarrow Q^{(i)} R^{(i)}$;
    if $mAP(R') > mAP(R^{(i)})$ then
        $R^{(i+1)} \leftarrow R'$;
    else
        $R^{(i+1)} \leftarrow R^{(i)}$;
    end
    $i \leftarrow i + 1$;
end
return $R^i$
```

Algorithm 1: Random search for the optimal rotation R.

where we would limit the summation to the pairs $(i,k) \in \mathcal{Q} = \{(i,j)|y_i \neq y_j\}$. In this way, the training would aim at minimizing the total elastic potential energy of the system of springs. The effect is that samples from different classes would be mapped on the sphere, but as far apart as possible. We note that minimizing this loss is equivalent to solving a first order linear approximation to the Thomson's Problem [35], which concerns the determination of the minimum electrostatic potential energy configuration of N electrons, constrained to the surface of a unit sphere. The solution has been rigorously identified in only a handful of cases, which mostly correspond to spatial configurations forming regular polyhedrons, which we have observed also in our simulations using the spring loss described before, as it can be seen in Figure 5. If we were to perform a Voronoi tessellation on the sphere based on the class centroids, we clearly would obtain a pretty uniform partition of the sphere, which is our main goal. For comparison, the right-most image in Figure 5 is obtained with the margin loss, which stops pulling apart query and negative samples once they are more far apart than the margin. This leads to a less uniform distribution of the classes.

We have experimented with the loss described above and indeed, it provides fairly good results. However, we argue that it can be improved because it does not explicitly pull closer samples that belong to the same class, besides pulling apart samples with different labels. We address that issue with the *triplet spring loss*, which we define as follows

$$\ell(d_{i,j,k}) = (2 - \sqrt{2 - d_{i,j,k}})^2 \ . \tag{10}$$

Note that $d_{i,j,k}$ varies in the range $[-2,2]$, thus the square root varies in the range $[0,2]$, and the loss varies in the range $[0,4]$. The loss is minimized when $d_{i,j,k}$ approaches -2. Since $d_{i,j,k}$ is proportional to $\|\mathbf{s}_i - \mathbf{s}_j\|^2 - \|\mathbf{s}_i - \mathbf{s}_k\|^2$, convergence is approached by maximally pulling closer $\mathbf{s}_i$ and $\mathbf{s}_j$, while maximally pushing apart $\mathbf{s}_i$ and $\mathbf{s}_k$. Note that, differently than before, now even when $\mathbf{s}_i$ and $\mathbf{s}_k$ have reached a distance of 2, the loss still works to pull $\mathbf{s}_i$ and $\mathbf{s}_j$ closer.

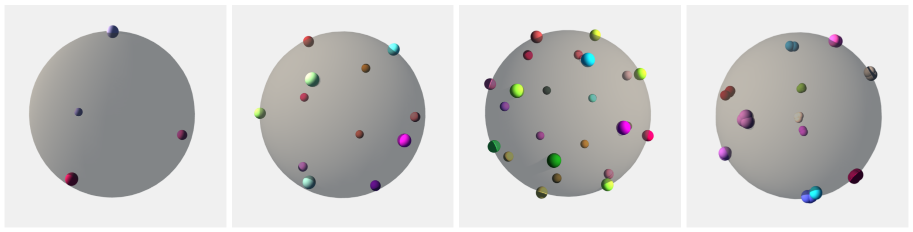

Fig. 5. Regular polyhedrons. Three left-most images: Three unit spheres with $5 \times n$ points at minimum elastic potential, where n is the number of classes. From left to right n is equal to 4, 12, 24. The 5 points per class coincide with the class centroid at equilibrium. Right-most image: For $n = 12$, the points at minimum margin loss do not reach a uniform distribution on the sphere.

8 Experiments

We tested our approach on the most relevant datasets for deep hashing applications, namely *CIFAR-10* [17], *NUS_WIDE* [8], and we also tested on *MNIST* [22] to compare with some older methods. For each experimental setting, we report the average mAP score over 5 runs for comparison against the previous works for hashes of size as low as 4 and up to 48 bits.

8.1 Experimental setup

Similar to other deep hashing methods we use raw image pixels as input. Following [42, 49, 23], for the spherical embedding we adopt VGG-F [6] pre-trained on ImageNet. We replace the last layer with our own, initialized with normal distribution. The output layer doesn't have activation function and the number of outputs matches the needed number of bits - B. The input layer of VGG-F is 224x224, so we crop and resize images of the *NUS_WIDE* dataset and upsample images of the *CIFAR-10* dataset to match the input size.

CIFAR-10: CIFAR-10 [17] is a publicly available dataset of small images of size 32x32 which have each been labeled to one of ten classes. Each class is represented by 6,000 images for a total of 60,000 available samples. In terms of evaluation in the CIFAR domain, two images are counted as relevant to each other if their labels match. In order for our experiments to be comparable to as many works as possible, including [42] and [3], we use two different experimental settings, which are labeled “Full” and “Reduced”.

“Full” setting: For this setting, 1,000 images are first selected randomly from each class of the dataset to make up the test images. Which, by extension, results in 10,000 query images. The remaining 50,000 images are used as the database images and as the images used in training.

“Reduced” setting: For this setting, 100 images are selected randomly from each class for use as 1,000 total test images. From the remaining 59,000 samples we randomly sample 500 images per category to form the reduced training set with only 5,000 images. The database is composed of all 59,000 samples which were not selected for testing.

Table 1. Mean Average Precision (MAP) Results for Different Number of Bits of CIFAR-10: In the case of DTSH and DVSQ we have filled some additional results which were not presented by the original papers by using the authors' respective released source code to replicate their experiments such that we may compare with them across more hash sizes.

	CIFAR-10 Full setting: Number of Bits							CIFAR-10 Reduced setting: Number of Bits						
Method	4	8	12	16	24	32	48	4	8	12	16	24	32	48
DQN[4]	-	-	-	-	-	-	-	-	-	0.554	-	0.558	0.564	0.580
DSH[24]	-	-	0.616	-	0.651	0.661	0.676	-	-	-	-	-	-	-
DPSH[23]	-	-	0.763	-	0.781	0.795	0.807	-	-	0.763	-	0.727	0.744	0.757
DMDH [7]	-	-	-	-	-	-	-	-	-	-	0.704	-	0.719	0.732
DTSH[42]	-	0.814	0.859	0.915	0.923	0.925	0.926	-	0.641	0.710	0.723	0.750	0.765	0.774
DVSQ[3]	-	0.839	-	0.839	0.843	0.840	0.842	-	**0.715**	-	0.727	0.730	0.733	0.764
BL[33]				0.870							-			
SDSH-ML	-	0.839	0.882	0.886	0.939	0.880	0.878	-	0.657	0.712	0.756	0.747	0.765	0.764
SDSH-LL	0.481	0.763	0.854	**0.942**	**0.945**	**0.944**	**0.947**	0.407	0.673	**0.757**	0.782	0.799	**0.815**	**0.822**
SDSH-S	**0.755**	**0.911**	**0.939**	0.938	0.939	0.939	0.934	**0.569**	0.697	0.723	**0.783**	**0.801**	0.810	0.813

The mAP for CIFAR-10, full and reduced setting, is computed based on all samples from the database set. We have used for training purposes a system with only one NVIDIA Titan X GPU, in this configuration training takes about four hours for the Cifar Full setting.

NUS_WIDE: NUS_WIDE [8] is another publicly available dataset, but unlike CIFAR-10 each sample image is multi-labeled from a set of 81 possible labels across all 269,643. This is reduced slightly, as [42] and [3] have also done in their experiments, by first removing every image which does not have any of the 21 most common labels associated with it. This is done as many of the less common labels have very few samples associated with them, but prepared in this way each of the 21 labels are represented by at least 5,000 samples. In terms of evaluation in the NUS_WIDE domain two images are counted as relevant to each other if any of their labels match. Note, that despite the usage of samples associated with the 21 most frequent labels, all 81 labels are used for determining similarity between samples. To compare with previous work we use three different settings, labeled "Full", "Reduced A", and "Reduced B".

"Full" setting: For this setting, 100 samples from each of the 21 most frequent labels are reserved for the test set. And the remaining images are used both as the database and as the training set. The mAP is computed based on the top 50000 returned neighbors.

"Reduced A" setting: For this setting, the 2100 test samples are selected as in the Full setting. From the remaining samples, 500 were sampled from the 21 most frequent labels to compose the training set. The remaining were used for the database. The mAP is computed based on top 5000 returned neighbors.

"Reduced B" setting: For this setting, the training set was chosen by sampling the available images uniformly 5,000 times. From the remaining samples the training set was uniformly sampled 10,000 times. All of the remaining samples from these two operations were used as the database. The mAP is computed based on top 5000 returned neighbors.

8.2 Results

Our method is abbreviated as SDSH, and the combination with the margin loss, label likelihood loss, and spring loss are indicated as SDSH-ML, SDSH-LL, and SDSH-S

Table 2. Mean Average Precision (MAP) Results for Different Number of Bits on NUS_WIDE

	Number of Bits			
Method	16	24	32	48
DTSH[42]	0.756	0.776	0.785	0.799
DPSH[23]	0.715	0.722	0.736	0.741
SDSH-ML	0.794	0.800	0.797	0.805
SDSH-LL	0.452	0.808	0.810	0.812
SDSH-S	**0.812**	**0.817**	**0.821**	**0.821**

(a) Full settings

	Number of Bits					
Method	8	12	16	24	32	48
DTSH[42]	-	0.773	-	**0.808**	**0.812**	**0.824**
SDSH-ML	0.758	0.770	0.784	0.798	0.802	0.810
SDSH-LL	0.751	0.780	0.792	0.805	0.810	0.817
SDSH-S	**0.774**	**0.789**	**0.796**	0.807	**0.812**	0.820

(b) Reduced A settings

	Number of Bits			
Method	8	16	24	32
DVSQ[3]	**0.780**	**0.790**	**0.792**	**0.797**
DMDH [7]	-	0.751	-	0.781
SDSH-ML	0.739	0.771	0.785	0.791
SDSH-LL	0.750	0.771	0.782	0.789
SDSH-S	0.755	0.783	0.786	0.790

(c) Reduced B settings

Table 3. Mean Average Precision (MAP) Results for Different Number of Bits of MNIST

	Number of Bits			
Method	16	24	32	48
CNNH[45]	0.957	0.963	0.956	0.960
CNNH+[45]	0.969	0.975	0.971	0.975
DSCH[47]	0.965	0.966	0.972	0.975
DRSCH[47]	0.969	0.974	0.979	0.979
SDSH-ML	0.993	**0.995**	0.994	0.995
SDSH-LL	**0.994**	0.994	**0.995**	**0.996**
SDSH-S	**0.994**	**0.995**	**0.995**	0.995

respectively. All reported results are from our publicly available implementation[3]. The average mAP scores for all methods on CIFAR-10 are listed in Table 1, which includes also the baseline (BL) classification accuracy. As pointed out in [33], the BL value can be interpreted as mAP attainable by a supervised system retrieving samples, one class at a time, with classes ranked according to the class probability of the query. Therefore, a retrieval approach should surpass the BL threshold to be effective. Table 1 shows that the proposed SDSH-S is above BL starting from 8-bits, and it always outperforms the state-of-the-art methods. In full setting and $B = 8$, and 12, SDSH-S shows large improvements, and outperforms DVSQ by 7.2% and 5.7%. SDSH-LL performs slightly better than SDSH-S for $B > 12$, but has worse performance for lower number of bits. For reduced setting, SDSH-S and SDSH-LL perform about the same, and always outperforms the state-of-the-art methods with an exception for the 8-bit case. Figure 6 presents the data of Table 1 in the form of plots.

Figure 7 shows the comparison of the precision-recall curves of SDSH-S with those produced by two state-of-the-art approaches, namely DTSH [42] and DVSQ [3], highlighting the promising performance of the proposed approach.

Tables 3a, 3b, and 3c show average mAP scores on NUS_WIDE for Full, Reduced A, and Reduced B settings respectively. Unfortunately, the protocol of this experiment does not allow to compare against the BL value. Although [33] describes two additional protocols, since they are suitable for tasks other than supervised hashing, for comparison with the state-of-the-art, here we follow protocols that have been in use by the widest majority of the literature. In particular, SDSH-S, SDSH-LL, and SDSH-ML always outperform the state-of-the-art methods on Full setting except for SDSH-LL at 8-bit case, where it converged poorly. On full setting, SDSH-S outperforms other type of losses for all bit numbers. On Reduced A setting, SDSH-S outperforms the state-of-the-art methods for $B < 24$ and for higher B it stays about the same as DTSH and slightly

[3] https://github.com/podgorskiy/SDSH

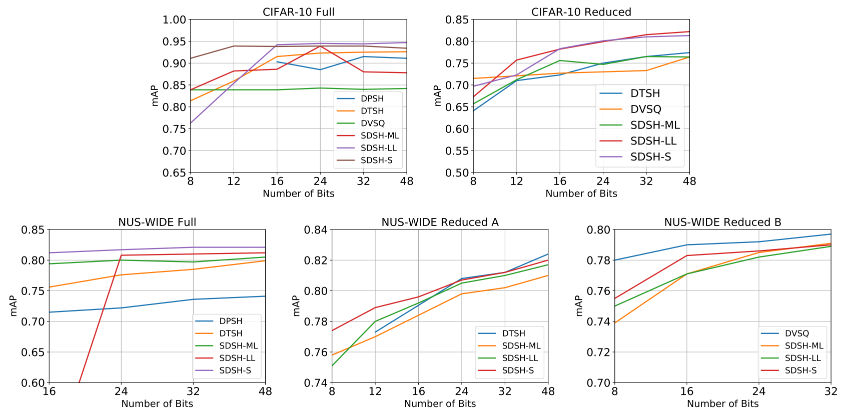

Fig. 6. Average mAP scores. Comparison of mAP values w.r.t. bit number for our method (SDSH-ML, SDSH-LL, SDSH-S) with DPSH [23], DTSH[42] and DVSQ[3].

below for $B = 48$. On Reduced B setting our method is outperformed by DVSQ. It is important to note that NUS_WIDE has 81 labels but only 500 samples from the 21 most frequent labels are used for training. Therefore, even though the training set for NUS-WIDE's reduced setting is still about twice as large as the training set for CIFAR-10's reduced setting, the ratio of samples per label for CIFAR is 500, while the ratio for this NUS_WIDE setting is on average 129.6 per label. Therefore, for NUS_WIDE reduced setting, the network is more prone to overfitting. As for CIFAR-10, the bottom row of Figure 6 presents the data of Tables 3a, 3b, and 3c in the form of plots.

In Table 3 we show a comparison between CNNH, CNNH+ [45], and DSCH, DRSCH [47] on the MNIST [22] dataset, noticing that all the three losses provide a performance increase.

8.3 Ablation Study

The proposed approach requires two steps for learning a hash function. The first step learns a spherical embedding that identifies an equivalence class of solutions because of the rotation invariance of the loss, and then a rotation needs to be identified to pick a representative of the equivalence class. Here we analyze what happens if use the identity as rotation, versus using the proposed method, versus using an off-the-shelve method like ITQ [13]. The summary of the results is shown in Figure 9, where the three approaches have been applied to SDSH-S, which has been tested on CIFAR-10, and NUS_WIDE. The first observation is that estimating the optimal rotation becomes more important at lower number of bits. As we suggested in Section 6, when B grows, the solution space grows significantly, so a random solution is more likely to do well. In addition, we note that ITQ tends to underperform our approach, and often decreases the performance of the identity solution. We note that ITQ differs from the proposed approach at least in two important aspects. First, ITQ is an unsupervised method, whereas our random search leverages the label information. In addition, ITQ aims at minimiz-

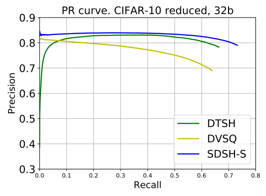

Fig. 7. Precision-Recall Curves Comparison of P-R curves from our method, DVSQ [3] and DTSH [42] on CIFAR-10 reduced, top 5000 samples @ 32 bits.

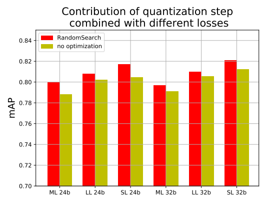

Fig. 8. Effect of quantization step. Contribution of quantization step for different losses on NUS-WIDE Full @ 24 bits, 32bits.

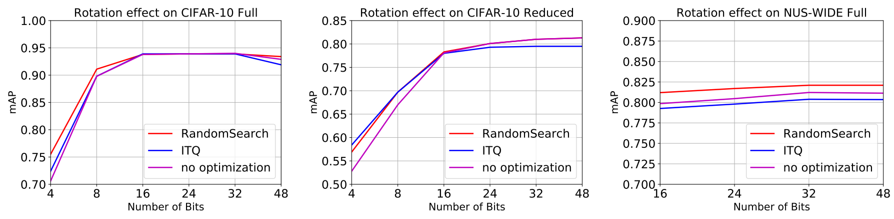

Fig. 9. Effect of learning rotation Comparison of mAP values for a range of bit number for three scenarios: ITQ [13] and random search optimization and no rotation optimization.

ing the quantization error, whereas the proposed method looks for the best quantization that maximizes the mAP. Finally, Figure 8 shows the performance improvement that adds rotation optimisation for different loss types, highlighting, as expected, that each of them can benefit from that step.

9 Conclusions

We have introduced SDSH, a novel deep hashing method that moves beyond the sole goal of similarity preserving, and explicitly learns a hashing function that produces quality codes. This is achieved by leveraging a different relaxation method that eliminates the need for regularizing priors, and it enables the design of loss functions for learning balanced codes, and it allows to optimize the quantized hash function to maximize the mAP. Extensive experiments on three standard benchmark datasets demonstrated the strength of the approach. In particular, addressing the issue of quality in deep hashing approaches has revealed to be valuable, because the performance has increased particularly for more compact codes, which is very important for building efficient retrieval systems.

Acknowledgments

This material is based upon work supported in part by the Center for Identification Technology Research and the National Science Foundation under Grant No. 1650474.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: IEEE FOCS. pp. 459–468 (2006)
2. Cao, Y., Long, M., Liu, B., Wang, J., KLiss, M.: Deep cauchy hashing for hamming space retrieval. In: CVPR. pp. 1229–1237 (2018)
3. Cao, Y., Long, M., Wang, J., Liu, S.: Deep visual-semantic quantization for efficient image retrieval. In: CVPR (2017)
4. Cao, Y., Long, M., Wang, J., Zhu, H., Wen, Q.: Deep quantization network for efficient image retrieval. In: AAAI. pp. 3457–3463 (2016)
5. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: Delving deep into convolutional nets. In: BMVC (2014)
6. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: Delving deep into convolutional nets. arXiv preprint arXiv:1405.3531 (2014)
7. Chen, Z., Yuan, X., Lu, J., Tian, Q., Zhou, J.: Deep hashing via discrepancy minimization. In: CVPR. pp. 6838–6847 (2018)
8. Chua, T.S., Tang, J., Hong, R., Li, H., Luo, Z., Zheng, Y.: Nus-wide: A real-world web image database from national university of singapore. In: ACM CIVR. pp. 48:1–48:9 (2009)
9. Erin Liong, V., Lu, J., Wang, G., Moulin, P., Zhou, J.: Deep hashing for compact binary codes learning. In: CVPR. pp. 2475–2483 (2015)
10. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization. IEEE TPAMI **36**(4), 744–755 (2014)
11. Ghasedi Dizaji, K., Zheng, F., Sadoughi, N., Yang, Y., Deng, C., Huang, H.: Unsupervised deep generative adversarial hashing network. In: CVPR. pp. 3664–3673 (2018)
12. Gionis, A., Indyk, P., Motwani, R., et al.: Similarity search in high dimensions via hashing. In: VLDB. vol. 99, pp. 518–529 (1999)
13. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE TPAMI **35**(12), 2916–2929 (2013)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016)
15. He, K., Wen, F., Sun, J.: K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In: CVPR. pp. 2938–2945 (2013)
16. Heo, J., Lee, Y., He, J., Chang, S., Yoon, S.: Spherical hashing: Binary code embedding with hyperspheres. IEEE Transactions on Pattern Analysis and Machine Intelligence **37**(11), 2304–2316 (2015). https://doi.org/10.1109/TPAMI.2015.2408363
17. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. pp. 1097–1105 (2012)
19. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. In: NIPS. pp. 1042–1050 (2009)
20. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: ICCV. pp. 2130–2137. IEEE (2009)
21. Lai, H., Pan, Y., Liu, Y., Yan, S.: Simultaneous feature learning and hash coding with deep neural networks. In: CVPR. pp. 3270–3278 (2015)
22. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
23. Li, W.J., Wang, S., Kang, W.C.: Feature learning based deep supervised hashing with pairwise labels. arXiv preprint arXiv:1511.03855 (2015)

24. Liu, H., Wang, R., Shan, S., Chen, X.: Deep supervised hashing for fast image retrieval. In: CVPR. pp. 2064–2072 (2016)
25. Liu, W., Mu, C., Kumar, S., Chang, S.F.: Discrete graph hashing. In: NIPS. pp. 3419–3427 (2014)
26. Liu, W., Wang, J., Ji, R., Jiang, Y.G., Chang, S.F.: Supervised hashing with kernels. In: CVPR. pp. 2074–2081. IEEE (2012)
27. Liu, W., Wang, J., Kumar, S., Chang, S.F.: Hashing with graphs. In: ICML. pp. 1–8 (2011)
28. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. IJCV **60**(2), 91–110 (2004)
29. Mishkin, D., Matas, J.L.: All you need is a good init. CoRR **abs/1511.06422** (2015)
30. Norouzi, M., Blei, D.M., Salakhutdinov, R.R.: Hamming distance metric learning. In: NIPS (2012)
31. Norouzi, M., Blei, D.M.: Minimal loss hashing for compact binary codes. In: ICML. pp. 353–360 (2011)
32. Ozols, M.: How to generate a random unitary matrix (2009)
33. Sablayrolles, A., Douze, M., Usunier, N., Jgou, H.: How should we evaluate supervised hashing? In: ICASSP. pp. 1732–1736 (2017)
34. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: CVPR. pp. 815–823 (2015)
35. Schwartz, R.E.: The five-electron case of thomsons problem. Experimental Mathematics **22**(2), 157–186 (2013)
36. Strecha, C., Bronstein, A., Bronstein, M., Fua, P.: Ldahash: Improved matching with smaller descriptors. IEEE TPAMI **34**(1), 66–78 (2012)
37. Szegedy, C., Toshev, A., Erhan, D.: Deep neural networks for object detection. In: NIPS. pp. 2553–2561 (2013)
38. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: CVPR. pp. 1701–1708 (2014)
39. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for scalable image retrieval. In: CVPR. pp. 3424–3431 (2010)
40. Wang, J., Zhang, T., Sebe, N., Shen, H.T., et al.: A survey on learning to hash. IEEE TPAMI **40**(4), 769–790 (2018)
41. Wang, N., Yeung, D.Y.: Learning a deep compact image representation for visual tracking. In: NIPS. pp. 809–817 (2013)
42. Wang, X., Shi, Y., Kitani, K.M.: Deep supervised hashing with triplet labels. ACCV (2016)
43. Wang, X., Zhang, T., Qi, G.J., Tang, J., Wang, J.: Supervised quantization for similarity search. In: CVPR. pp. 2018–2026 (2016)
44. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS. pp. 1753–1760 (2009)
45. Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S.: Supervised hashing for image retrieval via image representation learning. In: AAAI. vol. 1, pp. 2156–2162 (2014)
46. Zhang, P., Zhang, W., Li, W.J., Guo, M.: Supervised hashing with latent factor models. In: ACM SIGIR. pp. 173–182 (2014)
47. Zhang, R., Lin, L., Zhang, R., Zuo, W., Zhang, L.: Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. IEEE TIP **24**(12), 4766–4779 (2015)
48. Zhang, T., Du, C., Wang, J.: Composite quantization for approximate nearest neighbor search. In: ICML. pp. 838–846. No. 2 (2014)
49. Zhao, F., Huang, Y., Wang, L., Tan, T.: Deep semantic ranking based hashing for multi-label image retrieval. In: CVPR. pp. 1556–1564 (2015)
50. Zhu, H., Long, M., Wang, J., Cao, Y.: Deep hashing network for efficient similarity retrieval. In: AAAI. pp. 2415–2421 (2016)