The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Binary Generative Adversarial Networks for Image Retrieval

Jingkuan Song,[1] Tao He,[1] Lianli Gao,[1] Xing Xu,[1] Alan Hanjalic,[2] Heng Tao Shen[1*]
[1]Center for Future Media and School of Computer Science and Engineering,
University of Electronic Science and Technology of China.
[2]Delft University of Technology, Netherlands. {jingkuan.song, hetaoconquer}@gmail.com,
{lianli.gao,xing.xu}@uestc.edu.cn, a.hanjalic@tudelft.nl, shenhengtao@hotmail.com

Abstract

The most striking successes in image retrieval using deep hashing have mostly involved discriminative models, which require labels. In this paper, we use binary generative adversarial networks (BGAN) to embed images to binary codes in an unsupervised way. By restricting the input noise variable of generative adversarial networks (GAN) to be binary and conditioned on the features of each input image, BGAN can simultaneously learn a binary representation per image, and generate an image plausibly similar to the original one. In the proposed framework, we address two main problems: 1) how to directly generate binary codes without relaxation? 2) how to equip the binary representation with the ability of accurate image retrieval? We resolve these problems by proposing new sign-activation strategy and a loss function steering the learning process, which consists of new models for adversarial loss, a content loss, and a neighborhood structure loss. Experimental results on standard datasets (CIFAR-10, NUSWIDE, and Flickr) demonstrate that our BGAN significantly outperforms existing hashing methods by up to 107% in terms of mAP (See Table 2)[1].

1 Introduction

With the rapidly increasing amount of images, similarity search in large image collections has been actively pursued in a number of domains, including computer vision, information retrieval and pattern recognition (Shakhnarovich, Darrell, and Indyk 2008; Wang et al. 2017b). However, exact nearest-neighbor (NN) search is often intractable because of the size of dataset and the high dimensionality of images. Instead, approximate nearest-neighbor (ANN) search is more practical and can achieve orders of magnitude in speed-up compared to exact NN search (Shakhnarovich, Darrell, and Indyk 2008).

Recently, learning-based hashing methods (Wang et al. 2017b; Irie et al. 2014; Lin et al. 2014; Song et al. 2013; Shen et al. 2017) have become the mainstream for scalable image retrieval due to their compact binary representation and efficient Hamming distance calculation. Such approaches embed data points to compact binary codes by hash functions, which can be generally expressed as:

$$\mathbf{b} = \mathbf{h}(\mathbf{x}) \in \{0,1\}^L \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^{M\times 1}$, $\mathbf{h}(.)$ are the hash functions, and $\mathbf{b}$ is a binary vector with code length L.

Hashing methods can be generally categorized as being unsupervised or supervised. The unsupervised learning of a hash function is usually based on the criterion of preserving important properties of the training data points in the original space. Typical approaches target pairwise similarity preservation (i.e., the similarity/distance of binary codes should be consistent with that of the original data points) (Weiss, Torralba, and Fergus 2008; Liu et al. 2014; Irie et al. 2014; Song et al. 2017; Liu et al. 2013), multi-wise similarity preservation (i.e., the similarity orders over more than two items computed from the input space and the coding space should be preserved) (Norouzi and Fleet 2011), or implicit similarity preservation (i.e., pursuing effective space partitioning without explicitly evaluating the relation between the distances/similarities in the input and coding spaces), (Heo et al. 2015; Jin et al. 2013). A fundamental limitation of a hashing method geared to preserve a particular image property is that its performance may degrade when it is applied to a context where a different property is relevant.

Supervised hashing is designed to generate binary codes based on predefined labels (Lin et al. 2014; Strecha et al. 2012; Zhang et al. 2016). For example, Strecha *et al.* (Strecha et al. 2012) developed a supervised hashing method, which maximizes the between-class Hamming distance and minimizes the within-class Hamming distance. (Lin et al. 2014) proposed to learn the hash codes such to approximate the pairwise label similarity. Supervised hashing methods usually significantly outperform unsupervised methods. However, the information that can be used for supervision is also typically scarce.

More recently, deep learning has been introduced in the development of hashing algorithms (Xia et al. 2014; Lin et al. 2016; Do, Doan, and Cheung 2016; Gu, Ma, and Yang 2016; Wang et al. 2017a; Gao et al. 2017; Song et al. 2018), leading to a new generation of *deep hashing* algorithms. Due to powerful feature representation, remarkable image retrieval performance has been reported using the hashes obtained in this way. However, a number of

[*]Heng Tao Shen and Lianli Gao are the corresponding authors.

[1]Our code: https://github.com/htconquer/BGAN

open issues have still remained open. The most successful deep hashing methods are usually supervised and require labels. The labels are, however, scarce and subjective. Unsupervised approaches, on the other hand, cannot take full advantages of the current deep learning models, and thus yield unsatisfactory performance (Lin et al. 2016). Another issue is a non-smooth sign function used to generate the binary codes, which, despite several ideas being proposed to tackle it (Li, Wang, and Kang 2015; Cao et al. 2017; Do, Doan, and Cheung 2016), still makes the standard back-propagation infeasible. Cai *et. al.* (Cai 2016) studied the evaluation of ANNS problem and confirmed serious drawbacks in the evaluation of most of the existing hashing methods.

To address the above issues, we propose an unsupervised hashing method that deploys a generative adversarial network (GAN) (Reed et al. 2016). GAN has proven effective to generate synthetic data similar to the training data from a latent space. Therefore, if we restrict the input noise variable of generative adversarial networks (GAN) to be binary and conditioned on the features of each input image, we can learn a binary representation for each image and generate a plausibly similar image to the original one simultaneously. Feeding the generated images through a "discriminator" that verifies them with respect to the training images removes the need for supervision and the relevant hash can be learned in an unsupervised fashion. We refer to this proposed architecture as *binary GAN* (BGAN). For the BGAN learning process, we design a novel loss function to equip the binary representation with the ability of accurate image retrieval, beyond vivid image generation. Furthermore, inspired by recent studies on continuation methods (Allgower and Georg 2012), we propose two equivalent realizations of the sign function, and design an optimization strategy whose solution is equivalent to the non-smooth sign function.

2 Binary Adversarial Networks

Given N images, $\mathbf{I} = \{\mathbf{I}_i\}_{i=1}^N$ without labels, our goal is to learn their compact binary codes $\mathbf{B}$ and reconstructed images $\mathbf{I}^R$ for the original images such that: (a) the binary codes can reconstruct the image content, and (b) the binary codes could be computed directly without relaxation.

We illustrate our proposed BGAN architecture by the scheme in Fig. 1. The scheme shows how hash codes are learned in an unsupervised fashion through the interplay between, on the one side, the image generation process (the generator module) taking the generated hash code (the hash layer) as input, and, on the other side, the verification process (the discriminator module) where the images from the generator are compared to the original training images. For training of the system, we first construct the neighborhood structure of images and then train the neural network underlying the encoder, generator and discriminator. In the remainder of this section, we describe the process of constructing the neighborhood structure, the network architecture, our loss function and learning of parameters.

System Architecture

As shown in Figure 1, our BGAN consists of four components: encoder, hashing, generator and discriminator. We describe each of them in detail in the remainder of this section.

Encoder For feature extraction, we use a structure similar to VGG19 (Szegedy et al. 2015). We use 5 groups of convolution layers and 5 max convolution-pooling layers. Similar to (Szegedy et al. 2015), we use 64, 128, 256, 512, 512 filters in the 5 groups of convolutional layers, respectively.

Hashing A binary hash code is learned directly, by converting the L-dimensional representation $\mathbf{z}$ learned from the last fully-connected layer FC7, which is continuous in nature, to a binary hash code $\mathbf{b}$ taking values of either $+1$ or -1. This binarization process can only be performed by taking the sign function $\mathbf{b} = sgn(\mathbf{z})$ as the activation function on top of the hash layer.

$$\mathbf{b} = \operatorname{sgn}(\mathbf{z}) = \begin{cases} +1, & if\ \mathbf{z} \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2}$$

Unfortunately, as the sign function is non-smooth and non-convex, its gradient is zero for all nonzero inputs, and is ill-defined at zero, which makes the standard back-propagation infeasible for training deep networks. This is known as the vanishing gradient problem, which has been a key difficulty in training deep neural networks via back-propagation. Approximate solutions (Zhang et al. 2014; Kang, Li, and Zhou 2016) that relax the binary constraints are not a good alternative as they lead to a large quantization error and therefore to a suboptimal solution (Zhang et al. 2014).

In order to tackle this challenge of optimizing deep networks with non-smooth sign activation, we draw inspiration from recent studies on continuation methods (Allgower and Georg 2012; Cao et al. 2017). These studies address a complex optimization problem by smoothing the original function, turning it into a different problem that is easier to optimize. By gradually reducing the amount of smoothing during training, this results in a sequence of optimization problems converging to the original optimization problem. Following this approach, if we find an approximate smooth function of $sgn(.)$, and then gradually make the smoothed objective function non-smooth as the training proceeds, the final solution should converge to the desired optimization target.

Motivated by the continuation methods, we define a function $app(.)$ to approximate $sgn(.)$:

$$app(z) = \begin{cases} +1, & if\ \mathbf{z} \geq 1 \\ z, & if\ 1 \geq \mathbf{z} \geq -1 \\ -1, & if\ \mathbf{z} \leq -1 \end{cases} \tag{3}$$

We notice that there exists a key relationship between the sign function and the app function in the concept of limit:

$$\operatorname{sgn}(\mathbf{z}) = \lim_{\beta \to +\infty} app\,(\beta \mathbf{z}) \tag{4}$$

An illustration of the process through which $app(.)$ approximates $sgn(.)$ is given in Fig. 2. The figure also shows the same process for an alternative to $app(.)$: $tanh(.)$:

$$\operatorname{sgn}(\mathbf{z}) = \lim_{\beta \to +\infty} \tanh(\beta \mathbf{z}) \tag{5}$$

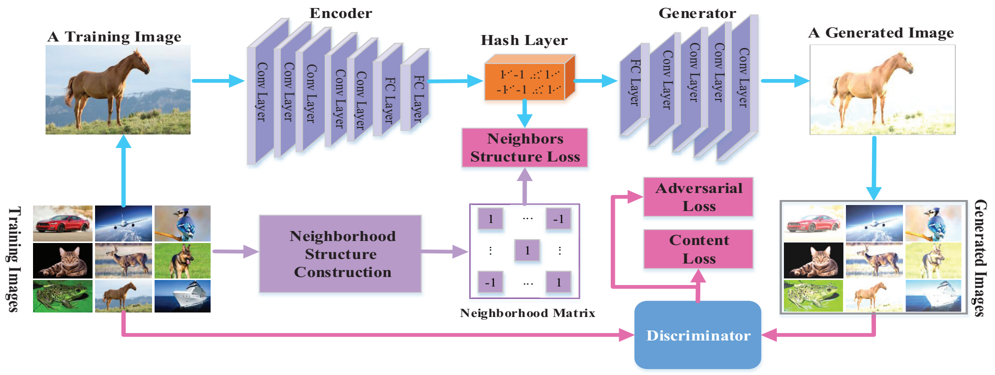


Figure 1: The proposed framework for BGAN, which is comprised of four key components: (1) an encoder, for learning image representations, (2) a hashing layer, for embedding the L-dimensional representation into L-bit binary hash code, (3) a decoder, to reconstruct the original images, and (4) a discriminator, for distinguishing real and reconstructed images. As a pre-processing step, we construct the neighborhood structure of the training images.

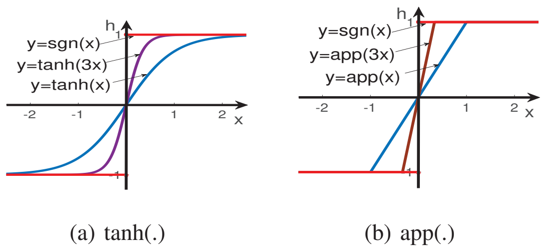


Figure 2: An illustration of the process through which $app(.)$ approximates $sgn(.)$

Generator and Discriminator The task of our BGAN "Generator" network $\mathbf{G}$ is to generate an image based on the hash code $\mathbf{b}$. First, we let $\mathbf{b}$ serve as the input of the top fully-connected layer with the size $8 \times 8 \times 256$. Then, we use four deconvolutional layers with the size of kernels $5 \times 5, 5 \times 5, 5 \times 5, 1 \times 1$, and the number of kernels $256, 128, 32, 3$, which is followed by batch-normalization layers and eLU as the activation function.

Following the approach by Goodfellow et al. (Goodfellow et al. 2014), we define a "Discriminator" network $\mathbf{D}$ in such a way that it is optimized using criteria that are conflicting to those of $\mathbf{G}$. In this way, $\mathbf{D}$ can act as adversary to $\mathbf{G}$ in the overall min-max optimization process. The goal of this optimization is to improve $\mathbf{G}$ such to be able to generate the images as well as possible. The process being adversarial to image generation is the process of trying to distinguish between the original and reconstructed images. If $\mathbf{G}$ manages to generate the images so well to "fool" $\mathbf{D}$, then it "wins" the min-max game and the overall GAN optimization has converged. In view of this, given a model of the image classifier $\mathbf{D}$ assessing the original ($\mathbf{I}$) and reconstructed ($\mathbf{I}^R$) image, we can formally define the min-max game resulting in the optimal system parameters as follows:

$$\min_{\theta_{\mathbf{G}}} \max_{\theta_{\mathbf{D}}} \log\left(\mathbf{D}\left(\mathbf{I}\right)\right) + \log\left(1 - \mathbf{D}\left(\mathbf{I}^R\right)\right) \tag{6}$$

Here we follow the architecture design summarized by Radford et al. (Radford, Metz, and Chintala 2015), use eLU activation and avoid max-pooling throughout the network. It contains 4 convolutional layers with an increasing number of 5×5 filter kernels (32, 128, 256, and 512). Strided convolutions are used to reduce the image resolution each time the number of features is doubled. The resulting 512 feature maps are followed by a dense layer with the size of 1024 and a final sigmoid activation function to obtain a probability for sample classification.

Loss Function

The definition of the loss function ℓ is critical for the performance of our network as it steers the overall optimization of the min-max game. In the following subsections, we explain our realization of this loss function.

Content Loss The content loss models the loss in the quality of reconstructed images. While auto-encoding is commonly modeled based on the Mean Squared Error, we follow (Larsen et al. 2016; Ledig et al. 2017) and design a loss function that assesses our hashing solution with respect to perceptually relevant characteristics. The most widely used pixel-wise MSE loss is calculated as:

$$\ell_{MSE} = \frac{1}{WH} \sum_{i=1}^{W} \sum_{j=1}^{H} \left(I_{ij} - I_{ij}^R\right)^2 \tag{7}$$

However, as pointed out in (Ledig et al. 2017), solutions of MSE optimization problems often lack high-frequency content, which results in perceptually unsatisfying solutions with overly smooth textures. Also, element-wise reconstruction errors are not robust for images and other signals with invariance.

Therefore, instead of only relying on pixel-wise losses we build on the ideas of Ledig et. al. (Ledig et al. 2017) and use a loss function that closely resembles perceptual similarity. Specifically, we define the VGG loss based on the eLU activation layers of the last convolutional layer of our discriminator network **D**. With ϕ we indicate the feature map obtained by the last convolution (after activation) and then define the VGG loss as the Euclidean distance between the feature representations of a reconstructed image I^R and the original image I:

$$\ell_{Perceptual} = \frac{1}{WH}\sum_{i=1}^{W}\sum_{j=1}^{H}\left(\phi\left(\mathbf{I}_{ij}\right) - \phi\left(\mathbf{I}_{ij}^{R}\right)\right)^2 \tag{8}$$

Here, W and H represent the dimensions of the respective feature maps. Based on the above, we can now define the content loss as:

$$\ell_C = \ell_{MSE} + \ell_{Perceptual}. \tag{9}$$

Adversarial Loss The adversarial loss models the loss due to misclassification of images, as done by **D**, in the original and reconstructed ones. This loss strengthens the power of image reconstruction, because a reconstructed image similar to its original image may not look like a real image. Using the notations already explained in the context of Eq.8, we model this loss as:

$$\ell_A = \log\left(\mathbf{D}\left(\mathbf{I}\right)\right) + \log\left(1 - \mathbf{D}\left(\mathbf{I}^R\right)\right) \tag{10}$$

Neighborhood Structure Loss Using the previously defined loss functions ℓ_C and ℓ_A, we can learn a binary code of an image which can reconstruct the original image. However, we cannot guarantee that similar images have close binary codes. The neighborhood structure loss models the loss in the similarity structure in data, as revealed in the set of neighbors obtained for an image by applying the hash code of that image. We first construct a similarity graph **S** of the images using their features. Given the binary codes $\mathbf{B} = \{\mathbf{b}_i\}_{i=1}^{N}$ for all the images of the length L, and using the similarity matrix **S** as the reference for similarity relations between images, we define the neighborhood structure loss as follows:

$$\ell_N = \frac{1}{2}\sum_{s_{ij}\in S}\left(\frac{1}{L}{\mathbf{b}_i}^T\mathbf{b}_j - s_{ij}\right)^2 \tag{11}$$

The goal of optimizing for this loss function is clearly to bring the binary codes of similar images as close to each other as possible. Instead of calculating the neighborhood structure loss directly on the binary codes **B**, we use **Z** (defined in Eq.2). Then Eq.11 can be reformulated as:

$$\ell_N = \frac{1}{2}\sum_{s_{ij}\in S}\left(\frac{1}{L}{\mathbf{z}_i}^T\mathbf{z}_j - s_{ij}\right)^2 + \|\mathbf{B} - \mathbf{Z}\|_F^2 \tag{12}$$

where $\|\mathbf{B} - \mathbf{Z}\|_F^2$ imposes the learned binary codes **B** to be close to the L-dimensional representation **z** learned from the last fully-connected layer FC7.

We formulate the loss function as the weighted sum of a *neighborhood structure loss*, *content loss* and *adversarial loss* as:

$$\ell = \ell_N + \lambda_1\ell_C + \lambda_2\ell_A \tag{13}$$

Learning

Using the loss function in Eq.13, we train our network. The forward propagation is as follows. First, we use a deep convolutional network as the encoder to extract the features and then use the hash layer to embed the real-valued features into binary codes:

$$\mathbf{b}_i = sgn(\mathbf{W}_h^T\varphi(\mathbf{I}_i;\theta)) \tag{14}$$

where $\mathbf{I}_i$ is an input image, θ is the parameter of the encoder and $\mathbf{W}_h$ stands for the parameter of generating **Z**. Then, $\mathbf{b}_i$ is the input for a generator **G** to reconstruct an image $\mathbf{I}^R$:

$$\mathbf{I}_i^R = \phi(\mathbf{b}_i;\pi) \tag{15}$$

where π stands for the parameters of the generator **G**. Finally, a discriminator **D** assigns the probability

$$p = \mathbf{D}(\mathbf{I}_i^R;\psi) \tag{16}$$

that $\mathbf{I}_i^R$ is an actual training sample and probability $1-p$ that $\mathbf{I}_i^R$ is generated by our model $\mathbf{I}_i^R = \phi(\mathbf{b}_i;\pi)$.

In the network, we have parameters of θ, π, ψ and $\mathbf{W}_h$ to learn. We use back-propagation (BP) for learning and stochastic gradient descent (SGD) to minimize the loss. In particular, we initialize network parameters and use forward propagation to obtain the value of each loss (ℓ_N, ℓ_C, ℓ_A). In each iteration, we sample a mini-batch of images from the training set, and then update each parameter:

$$\theta \leftarrow \theta - \tau\nabla_\theta\left(\ell_N + \ell_C\right), \tag{17}$$

$$\pi \leftarrow \pi - \tau\nabla_\pi\left(\ell_C + \ell_A\right), \tag{18}$$

$$\psi \leftarrow \psi + \tau\nabla_\psi\ell_A, \tag{19}$$

$$\mathbf{W}_h \leftarrow \mathbf{W}_h - \tau\nabla_{\mathbf{W}_h}\left(\ell_N + \ell_C\right) \tag{20}$$

where τ is the learning rate. We train our network until it converges.

For the hashing layer, we start training BGAN with $\beta_t = 1$. For each stage t, after BGAN converges, we increase β_t and train (i.e., fine-tune) BGAN by setting the converged network parameters as the initialization for training the BGAN in the next stage. For β_t towards ∞, the network will converge to BGAN with $sgn(\mathbf{z})$ as activation function, which can generate the desired binary codes. Using $\beta_t = 10$ we can already achieve fast convergence for training BGAN.

3 Experiments

We evaluate our BGAN on the task of large-scale image retrieval. Specifically, the experiments are designed to study the following research questions of our algorithm:

RQ1: How does each component of our algorithm affect the performance?

RQ2: Do the binary codes computed directly without relaxation improve the performance of the relaxed resolution?

RQ3: Does the performance of BGAN significantly outperform the state-of-the-art hashing algorithms?

RQ4: What is the efficiency of BGAN?

Settings

Datasets We conduct empirical evaluation on three public benchmark datasets, CIFAR-10, NUS-WIDE, and Flickr.

CIFAR-10 labeled subsets of the 80 million tiny images dataset, which consists of 60,000 32×32 color images in 10 classes, with 6,000 images per class.

NUS-WIDE is a web image dataset containing 269,648 images downloaded from Flickr. Tagging ground-truth for 81 semantic concepts is provided for evaluation. We follow the settings in (Zhu et al. 2016) and use the subset of 195,834 images from the 21 most frequent concepts, where each concept consists of at least 5,000 images.

Flickr is a collection of about 25,000 images from Flickr, where each image is labeled with one of the 38 concepts. We resize images of this subset into 256×256.

In NUS-WIDE and CIFAR-10, we randomly select 100 images per class as the test query set, and 1,000 images per class as the training set. In Flickr, we randomly select 1,000 images as the test query set, and 4,000 images for training.

Evaluation Metric Hamming ranking is used as the search protocol to evaluate our proposed approaches, and two indicators are reported. 1) Mean Average Precision (**mAP**): For a single query, Average Precision (AP) is the average of the precision value obtained for the set of top-k results, and this value is then averaged over all the queries.2) **Precision**: We further use precision-recall curve and precision@K to evaluate the precision of retrieved images.

We compare our BGAN with other state-of-the-art hashing algorithms. Specifically, we compare with four non-deep hashing methods (iterative quantization (ITQ) hashing (Gong et al. 2013), spectral hashing (SH) (Weiss, Torralba, and Fergus 2008), Locality Sensitive Hashing (LSH) (Datar et al. 2004), PCAH (Wang, Kumar, and Chang 2012), Spherical Hashing (Heo et al. 2015)), and two unsupervised deep hashing methods (DeepBit (Lin et al. 2016) and Deep Hashing (DH) (Erin Liong et al. 2015)).

To make a fair comparison, we also apply the non-deep hashing methods on deep features extracted by the VGG network (VGG-fc7 (Szegedy et al. 2015)). For non-deep hashing algorithms, we use the features provided with the dataset.

By constructing the neighborhood structure using the labels, our method can be easily modified as a supervised hashing method. Therefore, we also compare with some supervised hashing methods, e.g., iterative quantization hashing (ITQ-CCA) (Gong et al. 2013), KSH (Liu et al. 2012), minimal loss hashing (MLH) (Norouzi and Fleet 2011), DNNH (Lai et al. 2015), CNNH (Xia et al. 2014) and Deep Hashing Network (DHN) (Zhu et al. 2016).

Implementation Details When constructing the neighborhood structure, we use two different types of features: non-deep features provided with the dataset, and 2,048-dimensional deep features extracted using ResNet. We denote them as **BGAN_non** and **BGAN** respectively. The average number of the neighbors for each image is $400, 1021, 1168$ for the three datasets. By default, we set $\lambda_1 = 0.1$ and $\lambda_2 = 0.1$. We set the mini-batch size as 256, and the learning rate as 0.01.

Table 1: The mAP of BGAN_non on CIFAR-10 using different combinations of components. The similarity graph **S** is constructed using the non-deep features.

	mAP		
Components	24-bit	32-bit	48-bit
ℓ_C	0.131	0.137	0.146
ℓ_A	0.124	0.136	0.144
ℓ_N	0.342	0.351	0.372
$\ell_N + \ell_A$	0.357	0.366	0.371
$\ell_N + \ell_{MSE}$	0.353	0.361	0.361
$\ell_C + \ell_A$	0.145	0.156	0.163
$\ell_N + \ell_C + \ell_A$	**0.369**	**0.375**	**0.395**

Component Analysis (RQ1)

Our loss function consists of three major components: neighborhood structure loss (ℓ_N), content loss (ℓ_C) and adversarial loss (ℓ_A). In this subsection, we study the effect of each component on the performance. Due to the space limit, we only report the results on the CIFAR-10 dataset in Table 1. As analyzed in Sec.2, using the loss function of ℓ_C can already learn the binary codes of images. ℓ_A is the further help the reconstruction quality, and ℓ_N is to ensure that similar images have similar binary codes.

By using ℓ_C only, our network is similar to an autoencoder, and the retrieval performance is far from satisfactory. The mAP is only 0.131, 0.137, and 0.146 on CIFAR-10 dataset, which is consistent with another autoencoder based method DH (Erin Liong et al. 2015). This is because even the binary codes can reconstruct the images well, they cannot guarantee similar images to have similar binary codes. Similarly, by using ℓ_A only, the performance is unsatisfactory. On the other hand, by using ℓ_N only, the retrieval performance reaches 0.341, 0.351 and 0.372 for 24, 32 and 48 bits on CIFAR-10 dataset. However, the reconstructed images are not close to the original images. By using the combination of ℓ_N and ℓ_C, or the combination of ℓ_N and ℓ_A, the mAP can be further increased by around 0.01 compared with using ℓ_N only. This verifies the effectiveness of ℓ_A and ℓ_C. As expected, using the combination of ℓ_A and ℓ_C still cannot lead to a good retrieval performance.

The best performance is achieved when we use the combination of the three components: $\ell_N + \ell_C + \ell_A$. Compared with using ℓ_N only, the performance is improved by 2.7%, 2.4% and 2.3% for 24, 32, and 48-bit hash codes. From the above analysis, we can conclude that all these three components contribute to the great performance of our BGAN.

Effect of Binary Optimization (RQ2)

As discussed above, BGAN can learn binary hash codes directly while previous hashing methods first learn continuous representations and then generate hash codes using a sign function (denoted as two-step solution). In this subsection, we study the effect of direct binary codes optimization on the performance of hash codes, and the results are shown in Table 3. As shown in Table 3, our binary optimization can improve the performance of the learned binary codes. Specifically, the first solution (Eq.4) outperforms two-step

Table 2: mAP for different unsupervised hashing methods using different number of bits on three image datasets

Method	CIFAR-10				NUS-WIDE				Flickr			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
ITQ (Gong et al. 2013)	0.162	0.169	0.172	0.175	0.452	0.468	0.472	0.477	0.544	0.555	0.560	0.570
SH (Weiss, Torralba, and Fergus 2008)	0.131	0.135	0.133	0.130	0.433	0.426	0.426	0.423	0.531	0.533	0.531	0.529
LSH (Datar et al. 2004)	0.121	0.126	0.120	0.120	0.403	0.421	0.426	0.441	0.499	0.513	0.521	0.548
Spherical (Heo et al. 2015))	0.138	0.141	0.146	0.150	0.413	0.413	0.424	0.431	**0.569**	**0.559**	**0.583**	**0.572**
ITQ+VGG	0.196	0.246	**0.289**	**0.301**	0.435	0.435	0.548	0.435	0.553	0.548	0.545	0.560
SH+VGG	0.174	0.205	0.220	0.232	0.433	0.426	0.426	0.423	0.550	0.544	0.541	0.545
LSH+VGG	0.101	0.128	0.132	0.169	0.401	0.442	0.480	0.471	0.543	0.549	0.555	0.551
Spherical+VGG	**0.212**	**0.247**	0.256	0.281	**0.549**	**0.614**	**0.653**	**0.678**	0.552	0.547	0.546	0.545
DeepBit (Lin et al. 2016)	0.185	0.218	0.248	0.263	0.383	0.401	0.403	0.412	0.501	0.505	0.511	0.513
DH (Erin Liong et al. 2015)	0.160	0.164	0.166	0.168	0.422	0.448	0.480	0.493	0.553	0.548	0.543	0.556
BGAN_non	**0.361**	**0.369**	**0.375**	**0.395**	**0.518**	**0.541**	**0.545**	**0.568**	**0.591**	**0.601**	**0.607**	**0.626**
BGAN	**0.401**	**0.512**	**0.531**	**0.558**	**0.675**	**0.690**	**0.714**	**0.728**	**0.683**	**0.702**	**0.703**	**0.703**

solution by 2.3%, 1.7%, and 2.6% for 24, 32, and 48-bit hash codes, while the second solution (Eq.5) improves it by 3.2%, 2.0%, and 3.4%. This verifies our argument that two-step solution is sub-optimal, and binary optimization can achieve a better performance.

Table 3: The mAP of BGAN_non on CIFAR-10 using different binary optimization methods.

	mAP		
Methods	24-bit	32-bit	48-bit
two-step solution	0.337	0.355	0.361
$\text{sgn}\,(z) = \lim_{\beta \to +\infty} \tanh\,(\beta z)$	**0.369**	**0.375**	**0.395**
$\text{sgn}\,(z) = \lim_{\beta \to +\infty} app\,(\beta z)$	0.360	0.372	0.387

Compare with the State-of-the-art Algorithms (RQ3)

In this subsection, we compare our BGAN with different unsupervised hashing methods on three datasets. The results on mAP are shown in Table 2, and the precision is shown in Fig. 3. From Table 2 and Fig. 3, we have the following observations:
1) Our method (BGAN) significantly outperforms the other deep or non-deep hashing methods in all datasets. In CIFAR-10, the improvement of BGAN over the other methods is more significant, compared with that in NUS-WIDE and Flickr datasets. Specifically, it outperforms the best counterpart (Spherical+VGG) by 18.9%, 26.5%, 27.5% and 27.7% for 12, 24, 32 and 48-bit hash codes. One possible reason is that CIFAR-10 contains simple images, and the constructed neighborhood structure is more accurate than in the other two datasets. BGAN improves the state-of-the-art by 12.6%, 7.6%, 6.1% and 5.0% in NUS-WIDE dataset, and 11.4%, 14.3%, 12.0% and 13.1% in Flickr dataset.
2) Comparing to BGAN, the performance of our BGAN_non is worse. This indicates that the similarity graph plays an important role in the learning of hashing codes, and the non-deep features are not as good as deep features.
3) From Table 2, we observe that Spherical+VGG is a strong competitor in terms of mAP. On the other hand, the performance of deep hashing methods (DeepBit (Lin et al. 2016) and DH (Erin Liong et al. 2015)) is not superior. A possible reason is that the deep hashing methods use only 3 full connected layers to extract the features, which is not very powerful.

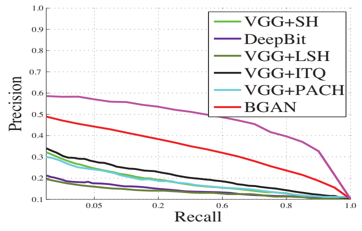
Table 4: mAP for different supervised hashing methods using different number of bits on three image datasets

Method	CIFAR-10			
	12 bits	24 bits	32 bits	48 bits
ITQ-CCA (Gong et al. 2013)	0.435	0.435	0.435	0.435
KSH (Liu et al. 2012)	0.556	0.572	0.581	0.588
MLH (Norouzi and Fleet 2011)	0.500	0.514	0.520	0.522
DNNH (Lai et al. 2015)	0.674	0.697	0.713	0.715
CNNH(Xia et al. 2014)	0.611	0.618	0.625	0.608
DHN (Zhu et al. 2016)	**0.708**	**0.735**	**0.748**	**0.758**
BGAN_s	**0.866**	**0.874**	**0.876**	**0.877**

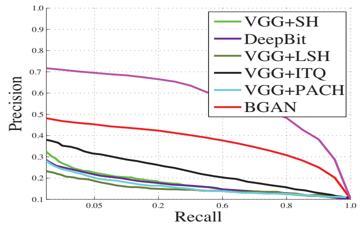
4) When we run the non-deep hashing methods on deep features, the performance is usually improved compared with the hand-crafted features. The performance gap is larger in CIFAR-10 and NUS-WIDE datasets than in Flickr dataset.

5) With the increase of code length, the performance of most hashing methods is improved accordingly. More specifically, the mAP improvements using deep features are generally more significant than that of non-deep features in CIFAR-10 dataset and NUS-WIDE dataset. An exception is SH, which has no improvement with the increase of code length.

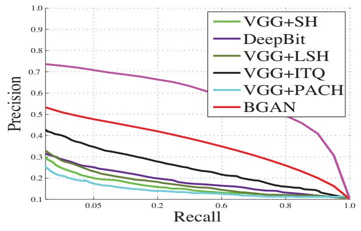
We also compared with supervised hashing methods, and present the mAP results on CIFAR-10 dataset in Table 5. It is obvious that our BGAN_s outperforms the state-of-the-art deep and non-deep supervised hashing algorithms by a large margin, which are 15.8%, 13.9%, 12.8% and 11.9% for 12, 24, 32, and 48-bits hash codes. This indicates that the performance improvement of BGAN is not only due to the constructed neighborhood structure, but also the other components. However, our method is mainly designed for unsupervised learning of hashing codes, and it has large room to be improved for the task of supervised hashing.



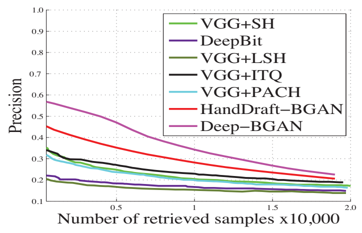
(a) 16 bits



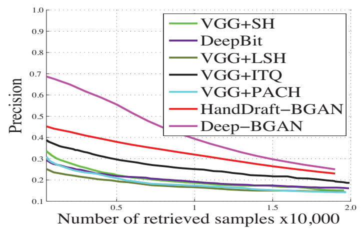
(b) 32 bits



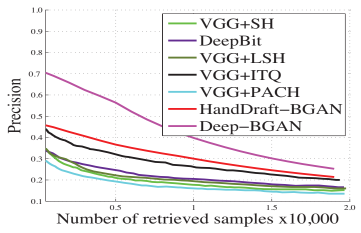
(c) 64 bits



(d) 16 bits



(e) 32 bits



(f) 64 bits

Figure 3: Precision for different unsupervised hashing methods using different number of bits on CIFAR-10 dataset.

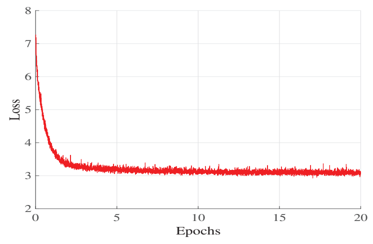


Figure 4: The convergence of BGAN in CIFAR-10 dataset.

The Study of Efficiency (RQ4)

In this subsection, we study the efficiency of our algorithm. First, we study the convergence of our BGAN in CIFAR-10 dataset, and the results are shown in Fig. 4. It can be seen that our method converges after a few epochs, which shows the efficiency of our solution.

We also report the training and testing time in CIFAR-10 dataset of our algorithm, and compare it with DH (Erin Liong et al. 2015) in Table 5. Since our BGAN has more parameters, it takes longer time for training and testing. However, BGAN is still fast in generating the hash codes for a test image.

Reconstruction Results

To evaluate the ability of image reconstruction using BGAN, we show some qualitative results on CIFAR-10 dataset in Fig. 5. The top row are the reconstructed results using random input, the second row are the reconstructed results from BGAN, and the last row are the original images. We can see that BGAN can reconstruct images which are similar to the original images.

Table 5: The training and testing time for BGAN and DH.

Methods	Training time	Testing time
DH (Erin Liong et al. 2015)	1 hour	0.5 ms
BGAN	5 hours	3 ms

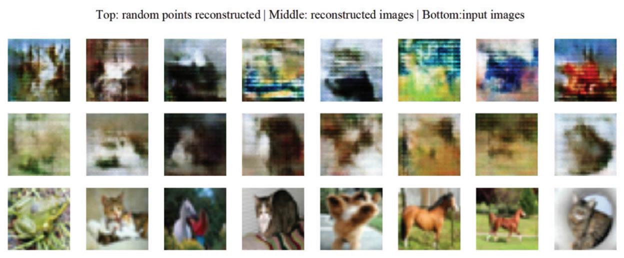

Top: random points reconstructed | Middle: reconstructed images | Bottom:input images

Figure 5: Image reconstruction using binary codes.

4 Conclusion

In this work, we propose an unsupervised hashing which addresses two central problems remaining largely unsolved for image hashing: 1) how to directly generate binary codes without relaxation? 2) how to equip the binary representation with the ability of accurate image retrieval, beyond vivid image generation? First, we propose two equivalent but smoothed activation functions, and design a learning strategy whose solution converges to the results of sign activation. Second, we propose a loss function which consists of an adversarial loss, a content loss, and a neighborhood structure loss. Experimental results show that our BGAN doubled the performance of the state-of-the-art in CIFAR-10 dataset, and also significantly outperformed the competitors on NUSWIDE, and Flickr datasets. In the future, it is necessary to improve the reconstruction accuracy of BGAN.

5 Acknowledgments

This work is supported by the Fundamental Research Funds for the Central Universities (Grant No. ZYGX2016J085), the National Natural Science Foundation of China (Grant No. 61772116, No. 61502080, No. 61632007) and the 111 Project (Grant No. B17008).

References

Allgower, E. L., and Georg, K. 2012. *Numerical continuation methods: an introduction*, volume 13. Springer Science & Business Media.

Cai, D. 2016. A revisit of hashing algorithms for approximate nearest neighbor search. *CoRR* abs/1612.07545.

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

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*.

Do, T.-T.; Doan, A.-D.; and Cheung, N.-M. 2016. Learning to hash with binary deep neural network. In *ECCV*, 219–234. Springer.

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

Gao, L.; Guo, Z.; Zhang, H.; Xu, X.; and Shen, H. T. 2017. Video captioning with attention-based LSTM and semantic consistency. *IEEE Trans. Multimedia* 19(9):2045–2055.

Gong, Y.; Lazebnik, S.; Gordo, A.; and Perronnin, F. 2013. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. PAMI* 35(12):2916–2929.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *NIPS*, 2672–2680.

Gu, Y.; Ma, C.; and Yang, J. 2016. Supervised recurrent hashing for large scale video retrieval. In *ACM Multimedia*, 272–276. ACM.

Heo, J.; Lee, Y.; He, J.; Chang, S.; and Yoon, S. 2015. Spherical hashing: Binary code embedding with hyperspheres. *IEEE Trans. Pattern Anal. Mach. Intell.* 37(11):2304–2316.

Irie, G.; Li, Z.; Wu, X.; and Chang, S. 2014. Locally linear hashing for extracting non-linear manifolds. In *CVPR*, 2123–2130.

Jin, Z.; Hu, Y.; Lin, Y.; Zhang, D.; Lin, S.; Cai, D.; and Li, X. 2013. Complementary projection hashing. In *ICCV*, 257–264.

Kang, W.-C.; Li, W.-J.; and Zhou, Z.-H. 2016. Column sampling based discrete supervised hashing. In *AAAI*, 1230–1236.

Lai, H.; Pan, Y.; Liu, Y.; and Yan, S. 2015. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, 3270–3278.

Larsen, A. B. L.; Sønderby, S. K.; Larochelle, H.; and Winther, O. 2016. Autoencoding beyond pixels using a learned similarity metric. In *ICML*, 1558–1566.

Ledig, C.; Theis, L.; Huszar, F.; Caballero, J.; Cunningham, A.; Acosta, A.; Aitken, A. P.; Tejani, A.; Totz, J.; Wang, Z.; and Shi, W. 2017. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 105–114.

Li, W. J.; Wang, S.; and Kang, W. C. 2015. Feature learning based deep supervised hashing with pairwise labels. *Computer Science*.

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

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

Liu, W.; Wang, J.; Ji, R.; Jiang, Y.; and Chang, S. 2012. Supervised hashing with kernels. In *CVPR*.

Liu, X.; He, J.; Lang, B.; and Chang, S. 2013. Hash bit selection: A unified solution for selection problems in hashing. In *CVPR*, 1570–1577.

Liu, X.; He, J.; Deng, C.; and Lang, B. 2014. Collaborative hashing. In *CVPR*.

Norouzi, M., and Fleet, D. J. 2011. Minimal loss hashing for compact binary codes. In *ICML*.

Radford, A.; Metz, L.; and Chintala, S. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Reed, S.; Akata, Z.; Yan, X.; Logeswaran, L.; Schiele, B.; and Lee, H. 2016. Generative adversarial text to image synthesis.

Shakhnarovich, G.; Darrell, T.; and Indyk, P. 2008. Nearest-neighbor methods in learning and vision. *IEEE Transactions on Neural Networks* 19(2):377.

Shen, F.; Mu, Y.; Yang, Y.; Liu, W.; Liu, L.; Song, J.; and Shen, H. T. 2017. Classification by retrieval: Binarizing data and classifiers. In *SIGIR*, 595–604.

Song, J.; Yang, Y.; Yang, Y.; Huang, Z.; and Shen, H. T. 2013. Inter-media hashing for large-scale retrieval from heterogeneous data sources. In *SIGMOD*, 785–796.

Song, J.; Gao, L.; Liu, L.; Zhu, X.; and Sebe, N. 2017. Quantization-based hashing: a general framework for scalable image and video retrieval. *Pattern Recognition*.

Song, J.; He, T.; Gao, L.; Xu, X.; and Shen, H. T. 2018. Deep region hashing for efficient large-scale instance search from images. In *AAAI*.

Strecha, C.; Bronstein, A. M.; Bronstein, M. M.; and Fua, P. 2012. Ldahash: Improved matching with smaller descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.* 34(1):66–78.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*, 1–9.

Wang, B.; Yang, Y.; Xu, X.; Hanjalic, A.; and Shen, H. T. 2017a. Adversarial cross-modal retrieval. In *ACM Multimedia*, 272–276.

Wang, J.; Zhang, T.; Song, J.; Sebe, N.; Shen, H. T.; et al. 2017b. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wang, J.; Kumar, S.; and Chang, S. 2012. Semi-supervised hashing for large-scale search. *IEEE Trans. PAMI* 34(12):2393–2406.

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

Xia, R.; Pan, Y.; Lai, H.; Liu, C.; and Yan, S. 2014. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2156–2162.

Zhang, P.; Zhang, W.; Li, W.-J.; and Guo, M. 2014. Supervised hashing with latent factor models. In *SIGIR*, 173–182.

Zhang, H.; Zhao, N.; Shang, X.; Luan, H.; and Chua, T. 2016. Discrete image hashing using large weakly annotated photo collections. In *AAAI*, 3669–3675.

Zhu, H.; Long, M.; Wang, J.; and Cao, Y. 2016. Deep hashing network for efficient similarity retrieval. In *AAAI*, 2415–2421.