TU Delft

Delft University of Technology

Unified Binary Generative Adversarial Network for Image Retrieval and Compression

Song, Jingkuan; He, Tao; Gao, Lianli; Xu, Xing; Hanjalic, Alan; Shen, Heng Tao

DOI
10.1007/s11263-020-01305-2

Publication date
2020

Document Version
Accepted author manuscript

Published in
International Journal of Computer Vision

Citation (APA)
Song, J., He, T., Gao, L., Xu, X., Hanjalic, A., & Shen, H. T. (2020). Unified Binary Generative Adversarial Network for Image Retrieval and Compression. *International Journal of Computer Vision*, *128*(8-9), 2243-2264. https://doi.org/10.1007/s11263-020-01305-2

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Noname manuscript No.
(will be inserted by the editor)

Unified Binary Generative Adversarial Network for Image Retrieval and Compression

Jingkuan Song · Tao He · Lianli Gao · Xing Xu · Alan Hanjalic · Heng Tao Shen

Received: date / Accepted: date

Abstract Binary codes have often been deployed to facilitate large-scale retrieval tasks, but not that often for image compression. In this paper, we propose a unified framework, BGAN+, that restricts the input noise variable of generative adversarial networks (GAN) to be binary and conditioned on the features of each input image, and simultaneously learns two binary representations per image: one for image retrieval and the other serving as image compression. Compared to related methods that attempt to learn a single binary code serving both purposes, we demonstrate that choosing for two codes leads to more effective representations due to less concessions needed when balancing the requirements. The added value of using a unified framework compared to two separate frameworks lies in the synergy in data representation that is beneficial for both learning processes. When devising this framework, we also address another challenge in learning binary codes, namely that of learning supervision. While the most striking successes in image retrieval using binary codes have mostly involved discriminative models requiring labels, the proposed BGAN+ framework learns the binary codes in an unsupervised fashion, yet more effectively than the state-of-the-art supervised approaches. The proposed BGAN+ framework is evaluated on three benchmark datasets for image retrieval and two datasets on image compression. The experimental results show that BGAN+ outperforms the existing retrieval methods with significant margins and achieves promising performance for image compression, especially for low bit rates.

J. Song, L. Gao, X. Xu and H. Shen
Center for Future Media and School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan, China, 611731.
E-mail: jingkuan.song@gmail.com, lianli.gao, xing.xu @uestc.edu.cn, shenhengtao@hotmail.com

T. He,
Monash University, Clayton VIC 3800.
E-mail: tao.he@monash.edu

A. Hanjalic,
Delft University of Technology, Netherlands.
E-mail: a.hanjalic@tudelft.nl

Keywords Binary Codes · Image Retrieval · Image Compression · Generative Adversarial Network

1 Introduction

Image retrieval and compression have both been extensively studied, however mostly as two disjointed research topics due to their distinct key techniques and applications. Image retrieval makes use of the representation of visual content to identify relevant images, and image compression searches for ways to achieve efficient image representation to lower the cost of storage and transmission. In this paper we investigate the possibility to address both challenges using a unified framework. This possibility offers itself in the form of binary codes, or *hashes*.

In the context of image retrieval, binary codes have been deployed for approximate nearest-neighbor (ANN) search, which has proven itself as a tractable alternative for the nearest-neighbor search (NN) on large image collections. ANN search is more practical and can achieve orders of magnitude in speed-up compared to exact NN search [26,72]. Recently, learning-based hashing methods [72,25,38,62,58] 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 $\mathbf{x} \in \mathbb{R}^{D\times 1}$, $\mathbf{h}(.)$ are the hash functions, and $\mathbf{b}$ is a binary vector with code length L.

According to whether labels are leveraged when learning a hashing function, we roughly divide the hashing methods into two categories, supervised and unsupervised. An unsupervised method is aimed at preserving similarity properties of the original data points in the binary codes. Typical techniques preserving the similarity include pairwise similarity [74,46], and multi-wise similarity [51,71] or implicit preservation, which means that we do not need to calculate the explicit similarity between the inputs and the compact codes [23,27]. Unsupervised hashing methods show many practical problems, such as how to construct the pairwise data points and how to measure and model different aspects of similarity in training data. Aiming at resolving the problems of unsupervised methods, supervised hashing methods [38, 14,63] have been well studied in recent years. While they usually significantly outperform unsupervised methods, 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 [76,40,11,20,68], 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 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 [40]. Another issue is a non-smooth sign function used to generate the binary codes, which, despite several ideas being proposed to tackle it [36, 6, 11], still makes the standard back-propagation infeasible.

The research field of image compression has already developed over many decades. The key challenge here is to find a pair of well-matched encoder and decoder. The encoder is used to transform the original large discrete data into low dimensional codes with minimal entropy [57], while the decoder acts as a translator which decodes the compressed codes into new data that should be identical as the original. In fact, the compression system is heavily associated with the probabilistic structure of the original data so the solution is similar to modeling a probabilistic source. In practice, since the compression codes always have finite entropy, we can not avoid the constructed errors. In this context, lossy compression problem has been studied for many years and generally, we must trade off two costs: the loss from the quantization (distortion) and the entropy of the discretized representation (rate). To be specific, low compression rate results in high entropy loss and high distortion directly leads to low-quality constructed data. However, joint learning of rate and distortion is difficult. [12] has demonstrated that it is intractable to optimize vector quantization without others constraints. [75] utilized linear projection to transform the original data into a continuous-valued image representation, and then independently quantized its elements and finally encoded the discrete representation in a lossless fashion. The widespread compression technique, JPEG [67] deploys cosine transform on each pixel block, while another popular technique, JPEG 2000 [53], applies a multi-scale orthogonal wavelet decomposition of the original data. Recently, several attempts have been made to deploy deep learning for compression [3], [56], [64], [65].

While the insights gained by the compression methods based on deep learning can be deployed to develop a compression method based on binary codes, no such method has been proposed yet. We believe developing such a method would be useful, in view of a high effectiveness of binary codes for retrieval. While, ideally, one could try to find a binary code that is usable for both tasks, our preliminary study has shown that optimizing a hash-function learning from both perspectives requires the learning algorithms to make too many concessions towards one of the objectives, making either retrieval or compression less effective than the common state-of-the-art. However, we hypothesize that we can come far in unifying the two binary-code learning processes. In this way, we can produce two different codes that are individually optimized for their different purposes, but in a way that the two learning procedures optimally benefit from each other in terms of learning efficiency and effectiveness.

In view of the analysis above, we propose in this paper a binary generative adversarial network (BGAN+) to convert images to binary codes for both image retrieval and compression in a multi-task learning fashion [8]. To the best of our knowledge, this is the first attempt to generate binary codes for compression. In addition, we take the challenge of learning these codes in an unsupervised way in order not to need to rely on typically scarce training data. Finally, we also propose several solutions to address the gradient vanishing problem caused by *sign* function in the hash-learning process. We validate the binary codes generated by our proposed BGAN+ framework

using extensive experiments addressing image retrieval and compression. The results show that our BGAN+ outperforms the existing retrieval methods with significant margins and achieves competitive performance for image compression, especially for low bit rates.

The remainder of this paper is organized as follows. We first review the related work in Sec. 2. Then, we provide the details of our proposed model in Sec. 3, followed by the experimental results in Sec. 5. Sec. 6 concludes the paper.

2 Related Work

In this section, we briefly review the related work, and then specifically the work on hashing for image retrieval, image generation and image compression. Regarding image retrieval using binary codes, supervised methods generally use information to learn hashing codes in three different formats: point-wise, pair-wise and rank orders. Representative point-wise hashing methods include CCA-ITQ [16], supervised discrete hashing (SDH) [59] and deep hashing [41]. Pair-wise hashing can best be illustrated by the methods, such as SPLH [69], which utilizes sequential projection learning strategy to generate efficient hashing codes, and KSH [44], which uses kernel function to learn hashing function and which outperforms other supervised methods, the fast supervised hashing [38] and two-step hashing (TSH) [39]. At the same time, many other methods based on deep learning have been developed, like the convolutional neural network hashing [77], in which it is proposed to automatically learn convolutional image representation instead of the previous work using hand draft features as input. Furthermore, DPSH [36] directly combines two independent tasks, learning image representation and hashing function, into a deep end-to-end network. The representative rank-label methods include column generation hashing [37], ranking-based supervised hashing [70], discretely semantic rank orders (DSeRH) [43] and ranking preserving hashing [79]. In our work, we use pair-wise similarity as the hashing-learning strategy, but unlike previous work, we do not use the ground truth labels to construct pair-wise labels. Instead, we adopt two ways, via hand-crafted feature and deep feature, to create the similarity matrix. In this sense, our proposed method can be treated as the unsupervised method.

Regarding the research on image generation, generative adversarial networks (GAN) [17] has played a critical role recently. GAN usually consists of two networks, a generator and a discriminator network, which are involved in a min-max optimization game. There, the discriminator acting as an adversary to the generator is used to judge whether the generated image from the generator is real or fake, that is, whether it matches the criteria imposed by the input image or not. This is why the goal of the generator is to generate images that resemble the input image in the best possible way so it can 'fool' the discriminator. Theoretically, when the discriminator cannot distinguish the source of the image (original or from the generator), we can consider the overall GAN optimization as converged. Recently, a vast number of image generation methods based on GAN have been explored [33], [34].

Lossy image compression has been widely used for data storage and transmission. JPEG [67] and JPEG 2000 [53] are two commonly used methods of lossy com-

pression for digital images. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality. After that, more sophisticated compression methods have been proposed, e.g., WebP [18], JPEG 420, Better Portable Graphics (BPG) [4]. Recently, with the wide application of deep learning, there are numerous novel compression methods based on CNN or Recurrent Neural Network (RNN) [66,35,3]. In [66], Toderici *et al.* proposed a deep RNN network, which can provide variable compression rates during deployment, and introduced a new hybrid of GRU and ResNet. In [35], Li *et al.* explored a content-aware compression method based on the convolutional network, which can generate an importance map of the image content and optimize the compression quality. It can also retain as much detail as possible and in the low bit rate their method outperforms JPEG and JPEG 2000. [2] proposed multi-stage progressive encoders, whose structure resembles a bottleneck, like VAE [30]. [3] proposed an image compression framework, consisting of a nonlinear encoding transformation, a uniform quantizer, and a nonlinear decoding transformation, which only contains three convolutional layers. With the great performance achieved in [16], the residual block has been proved to be a remarkably efficient way in the aspect of reducing information loss due to deep layers network [2]. Firstly, the residual block allows the deeper layers to know how to utilize information which could not be generated by the previous stage. Secondly, these connections reduce the distance of the path that information travels, which brings better joint optimization. In [1], Agustsson *et al.* proposed to learn compressible representations using deep architectures, which can be trained end-to-end. In [64] Theis *et al.* utilized the derivative of a smooth approximation to replace the derivative in the backward pass of back-propagation and optimized the autoencoder network. Outstanding performance was reported.

3 Proposed Framework

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 $\mathbf{B}^c$ such that: (a) the original image can be reconstructed from $\mathbf{B}^c$, (b) similar images can be retrieved using $\mathbf{B}$, and (c) both $\mathbf{B}^c$ and $\mathbf{B}$ can be computed directly without relaxation.

We illustrate our proposed BGAN+ framework by the scheme in Fig. 1. The framework consists of two components: 1) a binary generative adversarial *compression* network (BGANc), and 2) a binary generative adversarial *retrieval* network (BGANr). Both parts learn their binary codes in an unsupervised fashion. In the BGANc part, $\mathbf{B}^c$ is learned through the interplay between a generator and a discriminator. Specifically, the generator takes an image as input and represents it by a binary code. Then, this code is decoded to reconstruct the image, which enters the verification process in the discriminator to be compared with the original image. The BGANr part learns binary code $\mathbf{B}$ by also taking into account the visual neighborhood structure of the image in order to make sure that the proper notion of image similarity propagates into the similarity of the learned binary codes for retrieval. The two learning processes are coupled by the output of the shared encoder. In this way,

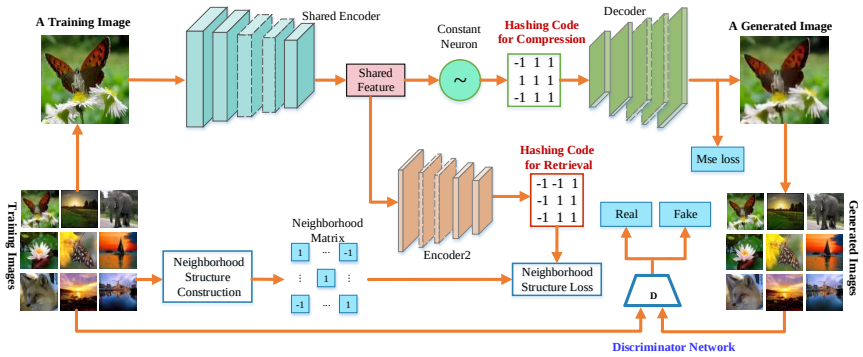


Fig. 1 An overview of our proposed BGAN+ framework for simultaneously learning binary codes for image retrieval and compression. Our framework contains two major networks, i.e., image compression network and image retrieval network. For the image compression network (BGANc), there are four key components: (1) a shared encoder, for learning low-level image representations, (2) a constant neuron layer, for learning the binary codes for image compression, (3) a decoder, to reconstruct the original images, and (4) a discriminator, to distinguish between real and reconstructed images. For the image retrieval network (BGANr), there are three key components: (1) a shared encoder, (2) encoder2, for learning high-level image representations, and (3) a hashing layer, for learning the binary codes for image retrieval. As a pre-processing step, we construct the neighborhood structure of the training images.

the criteria used to learn binary code in one part of the framework help in learning the binary code in the other part. Although we learn two separate codes for two different purpose, we hypothesize that this synergetic effect is beneficial for both learning processes and justifies their integration into a single framework, as opposed to creating two binary codes using separate frameworks. In addition, through shared modules, both codes are learned in a more efficient manner than if they are learned independently. Related to the latter, for the learning of the parameters, we train the entire framework at once by a joint training strategy. In the remainder of this section, we provide detailed information about the our proposed BGAN+ framework.

3.1 Binary Generative Adversarial Compression Network (BGANc)

The binary codes $\mathbf{B}^c$ learned from $\mathbf{I} = \{\mathbf{I}_i\}_{i=1}^N$ by BGANc have the task to represent an image such that it can be reconstructed as well as possible back to its original state. We model this goal by the following expression:

$$\ell(\mathbf{I}) = \min_f ||\mathbf{I} - f(\mathbf{I})|| \tag{2}$$

where f denotes the transformation function from the original image $\mathbf{I}_i$ into the reconstructed image $\mathbf{I}^R$. The transformation function consists of the elements of the shared encoder, the proposed constant neuron and the decoder. We explain these components in more detail in subsections 3.1.1, 3.1.2 and 3.1.3, respectively. Then, in subsection 3.1.4. we come back with an elaborate version of the above expression, taking into account the realizations of the three components.

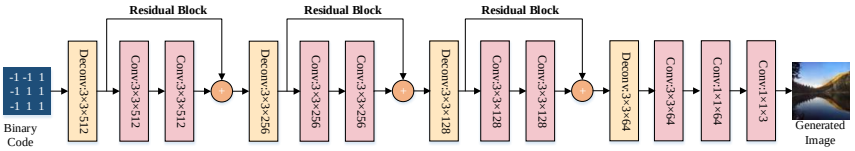


Fig. 2 Configuration of the decoder. $\mathbf{B}_i^c \in \mathbb{R}^{\frac{W}{16} \times \frac{H}{16} \times C}$ is the input code, where C controls the bit-rate, and $\mathbf{I}_i^R \in \mathbb{R}^{W \times H \times 3}$ is the output image. The input image $\mathbf{I}_i$ is firstly compressed to $\mathbf{B}_i^c$. The decoder reconstructs an image $\mathbf{I}_i^R$ from $\mathbf{B}_i^c$ using several Deconv and Conv layers, and ensures that the final output image $\mathbf{I}_i^R$ has the same size as the original image $\mathbf{I}_i$. Skip connection works as the shortcut in residual network.

3.1.1 Shared Encoder

VGG network [60] utilizing an architecture with 3×3 convolution filters is able to achieve good performance for large scale image classification with 19 weight layers. In this paper, we build our shared encoder with the first five convolution layers of VGG, with the details illustrated in Tab. 3.1.4. Following the architecture of [34,54], we avoid two max-pooling operations throughout the shared encoder to allow our network to learn its own spatial downsampling. Specifically, we set the stride of the first four convolutional layers as 2, and thus each convolutional layer has the size (i.e., width and height) of the input feature map. The stride of the last convolutional layer is set as 1, which can be considered as a fully convolutional layer. Given an image $\mathbf{I}_i$ with the size of $W \times H$ (i.e., W as width and H as height), we can obtain C feature map with the size of $\frac{W}{16} \times \frac{H}{16} \times 3$ through our shared encoder, where C denotes the number of feature map channels.

3.1.2 Constant Neuron

To compress an image into a hash code and then reconstruct the image from the generated hash code, the issue related to binary constraints becomes relevant. The problem of binary constraints has been addressed by relaxing the constraints from $\{0,1\}$ [74] or by adopting alternating optimization strategies [16,15]. However, they either cause a large optimality gap between non-relaxed and relaxed objectives or substantially weaken the model flexibility, respectively. As a result, in [9], Dai *et al.* proposed to define a function $\mathbf{h}$ to re-parameterize Bernoulli distribution over the binary variables to avoid directly working with binary variables. $\mathbf{h}$ is referred to as stochastic neuron:

$$\mathbf{h}(z) = \begin{cases} 1 & if\ z \geq \xi \\ 0 & if\ z < \xi \end{cases} \tag{3}$$

where $\xi \sim \mu(0,1)$. Inspired by the stochastic neuron, in this paper, we propose a *constant neuron*, which is defined as:

$$\mathbf{h}(z) = \begin{cases} 1 & if\ z \geq 0.5 \\ 0 & if\ z < 0.5 \end{cases} \tag{4}$$

Since $\mathbf{h}$ is not smooth, it is still difficult to apply stochastic gradient descent to calculate the gradient of the parameters. To solve this problem, we firstly define $\mathbf{W}_e, \mathbf{b}_e$

as the parameters of our shared encoder. As a result, the intermediate compressed hash codes can be formulated as:

$$\mathbf{B}^c = \mathbf{h}(E_n(\mathbf{I}; \mathbf{W}_e, \mathbf{b}_e, \delta)) \tag{5}$$

where δ is the active function of the convolutional layers and E_n is our encoding function. Then, we set the active function of the last convolution layer of shared encoder as *sigmoid* and the other four layers are set as *ReLu*. Finally, we define our constant neuron as:

$$\mathbf{h}(\mathbf{I}; \Phi) = \frac{sign(E_n(\mathbf{I}; \mathbf{W}_e, \mathbf{b}_e, \delta) - 0.5) + 1}{2} \tag{6}$$

Unfortunately, the *sign* function is non-smooth and gradient of *sign* is zero. Following [19], we use distributional derivative to estimate the stochastic gradient by:

$$\nabla_{\Phi}\mathbf{h}(\mathbf{I}; \Phi) = \nabla_{\ell}(\mathbf{I}; \Theta) E_n(\mathbf{I}; \Phi, \delta) \bullet (1 - E_n(\mathbf{x}; \Phi, \delta))\mathbf{I}^T \tag{7}$$

where $\bullet$ denotes a point-wise product, and where $\Phi = \{\mathbf{W}_e, \mathbf{b}_e\}$ and $\nabla_{\ell}(\mathbf{I}; \Theta)$ is the gradient from our objective function. More specifically, we utilize chain rules to calculate it. To conduct optimization, we leverage standard stochastic gradient descent algorithm to optimize $\mathbf{B}^c$ by following [48,29].

3.1.3 Decoder

The decoder of BGANc takes the output of the constant neuron as input to reconstruct the original image. Therefore, the input for the decoder is $\mathbf{B}_i^c$ and the output is an image $\mathbf{I}_i^R$. Transferring such low dimensional features $\mathbf{B}_i^c$ to a high dimensional feature is a challenging task. In previous work, most auto-encoding systems [33,30] use a fully connected layer as the first layer of a decoder for transforming a low dimensional feature into a high dimensional feature by a non-linear projection, but this substantially reduces the model flexibility: the input size must be fixed. However, cropping or wrapping an image into a fixed size can lead to a loss of image information [21]. More importantly, in reality, we need to provide an efficient approach to compress images with an arbitrary size. Previous work [16,34,2] demonstrated that residual blocks have a significant effect on reducing the information loss as the network deepening. Inspired by this observation, we design our decoder network by integrating deconvolutional layers [52], residual blocks with fully convolutional layers, for efficiently reconstructing images from binary codes. Specifically, the decoder consists of four deconvolutional layers (i.e., setting as $3 \times 3 \times 512$, $3 \times 3 \times 256$, $3 \times 3 \times 128$ and $3 \times 3 \times 64$) and three residual blocks, each with two convolutional layers, followed by a convolutional layer ($3 \times 3 \times 64$ and two fully convolutional layers ($1 \times 1 \times 64$, and $1 \times 1 \times 3$). The structure of our proposed decoder is shown in Fig. 2.

3.1.4 BGANc Optimization Objective

Based on the realizations of the three BGANc components as explained above, we can now define the expression Eq.(2) more concretely. What we minimize in Eq.(2) is actually the loss of reconstructing the input image. The definition of the corresponding loss function as the optimization objective is critical for the performance of our generator network. In this subsection, we define two loss functions that contribute to the optimization objective of the BGANc network.

Content Loss The first component is the *content loss*, which directly measures the deviation of the reconstructed image from the original. While both l_1 loss and l_2 loss are applied for image generation task and previous work [34] has proven that l_1 loss performs better than l_2 loss, thus we define our content loss function as below:

$$\ell_{D_e} = \min_{\Omega} \sum_{i=1}^{N} \|\mathbf{I}_i - D_e(\mathbf{B}_i^c; \Omega)\| \tag{8}$$

where D_e denotes the decoding operation, Ω denotes the parameters of decoder and $\mathbf{B}^c$ is seen as the compressed hashing codes generated by our encoder. Furthermore, we can rewrite Eq. 8 as the following pixel-wise l_1 loss:

$$\ell_{D_e}(\mathbf{I}; \Omega) = \min_{\Omega} \frac{1}{WH} \sum_{i=1}^{N} \sum_{p=1}^{W} \sum_{q=1}^{H} \left\| I_{i,p,q} - I^R_{i,p,q} \right\| \tag{9}$$

Obviously, Eq. 9 is continuous and can be directly optimized by the stochastic gradient descent algorithm.

Adversarial Loss In order to make the optimization of BGANc more robust, we also look from another perspective at the quality of the reconstructed image. Following the approach by Goodfellow et al. [17], we define a "Discriminator" network **D** in such a way that it is optimized using criteria that are conflicting to those of **G**. **G** is the "Generator" network (i.e., the decoder D_c shown in Fig. 1). In this way, **D** can act as adversary to **G** in the overall min-max optimization process. The goal of this optimization is to improve **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 **G** manages to generate the images so well to "fool" **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 **D** assessing the original (**I**) and reconstructed ($\mathbf{I}^R$) image, we can formally define the min-max game resulting in the optimal system parameters as follows:

$$\ell_A(\mathbf{I}_i; \Phi, \Omega, \Theta) = \min_{\Phi,\Omega} \max_{\Theta} \log(D(\mathbf{I}_i)) + \log(1 - D(\mathbf{I}_i^R)) \tag{10}$$

where Φ, Ω are, respectively, the parameters of the encoder and decoder network, and Θ is the vector of the parameters of the discriminator.

Here we follow the architecture design summarized by Radford et al. [55]. We use ReLU activation and avoid max-pooling throughout the network. It contains 4

Table 1 The architecture for feature extraction

Layer	Size of Filter	Filters	Others
conv1_1	3x3	64	Stride=1,padding=1,relu
conv1_2	3x3	64	Stride=2,padding=1,relu
conv2_1	3x3	128	Stride=1,padding=1,relu
conv2_2	3x3	128	Stride=2,padding=1,relu
conv3_1	3x3	256	Stride=1,padding=1,relu
conv3_2	3x3	256	Stride=1,padding=1,relu
conv3_3	3x3	256	Stride=1,padding=1,relu
Max pooling	2x2		2
conv4_1	3x3	512	Stride=1,padding=1,relu
conv4_2	3x3	512	Stride=1,padding=1,relu
conv4_3	3x3	512	Stride=1,padding=1,relu
Max pooling	2x2		2
conv5_1	3x3	512	Stride=1,padding=1,relu
conv5_2	3x3	512	Stride=1,padding=1,relu
conv5_3	3x3	512	Stride=1,padding=1,relu
Max pooling	2x2		2
FC6	None	4096	relu
FC7	None	4096	relu

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 and 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.

We can formulate the objective function of compression network as the weighted sum of the pixel-wise l_1 loss and the adversarial loss as:

$$\ell_C = \ell_{D_e} + \lambda \ell_A \tag{11}$$

where λ is the weighted factor to balance the impact of pixel-wise loss and adversarial loss.

3.2 Binary Generative Adversarial Retrieval Network (BGANr)

The task of our retrieval network BGANr is to generate a hash code $\mathbf{B}_i$ from an image $\mathbf{I}_i$. The structure of BGANr consists of two parts: shared encoder, the encoder2 and the hashing layer (see Fig. 1).

3.2.1 Encoder2

Specifically, our BGANr is based on the VGG network and the specific configuration is defined in Tab. 3.1.4. Theoretically, we can directly use $\mathbf{B}^c$ to retrieve images. However, it is unlikely to acquire excellent results due to the reason that compression network only encodes low-level information without high-level semantic information. To conduct an efficient search, hash code $\mathbf{B}$ must encode both low-level and high-level semantic information. As a result, we design our BGANr by sharing the encoder of $\mathbf{G}$ to extract better low-level information.

Algorithm 1 Construction of neighborhood structure

Input: Images $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$, the number of neighbors K1, the number of neighbors K2 for the neighbors expansion;
Output: Neighborhood matrix $\mathbf{S} = \{s_{ij}\}$;

```
First ranking: Use cosine similarity to generate the index of K1-NN of each image L_1, L_2, ..., L_N;
Neighborhood expansion:
for j=1,...,N do
    Initialize num ← 0;
    for j=1,...N do
        num_j ← the size of L_i ∩ L_j;
    end for
    Sort num by descending order and keep the top K2 {L_j};
    Set new L'_i ← union of the top K2 {L_j};
end for
for j=1,...,N do
    Construct S with new L'_i based on Eq. 14;
end for
return S;
```

3.2.2 Construction of Neighborhood Structure

Moreover, for the training of the system, we first conduct the neighborhood structure of images and then train the network. Neighborhood structure is beneficial to exploiting the manifold structure of the training data, and can improve the performance of image retrieval [72]. Next, we introduce our approach to construct a similarity matrix by an unsupervised method.

In our unsupervised approach, we propose to exploit the neighborhood structure of the images in feature space as information source steering the process of hash learning. Specifically, we propose a method based on the K-Nearest Neighbor (KNN) concept to create a neighborhood matrix of $\mathbf{S}$. We use two types of features to construct $\mathbf{S}$: non-deep features and deep features. For non-deep features, we use the hand-crafted features provided with the dataset. For deep features, we extract 2,048-dimensional features from the pool5-layer based on [22]. This results in the set $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ where $\mathbf{x}_i$ is the feature vector of image $\mathbf{I}_i$.

For the representation of the neighboring structure, our task is to construct a matrix $\boldsymbol{S} = \{s_{ij}\}_{i,j=1}^{N}$, whose elements indicate the similarity ($s_{ij} = 1$) or dissimilarity ($s_{ij} = -1$) of any two images i and j in terms of their features $\mathbf{x}_i$ and $\mathbf{x}_j$.

We compare images using cosine similarity of the feature vectors. For each image, we select $K1$ images with the highest cosine similarity as its neighbors. Then we can construct an initial similarity matrix $\mathbf{S}_1$:

$$(S_1)_{ij} = \begin{cases} 1, \text{ if } \mathbf{x}_j \text{ is K1-NN of } \mathbf{x}_i \\ -1, \text{ otherwise} \end{cases} \tag{12}$$

The precision curve (evaluated using the labels) in Fig. 3 indicates the quality of the constructed neighborhood for different values of $K1$. Due to rapidly decreasing precision with increasing $K1$, creating a large-enough neighborhood by simply increasing $K1$ is not the best option. In order to find a better approach, we borrow ideas from the domain of graph modeling. In an undirected graph, if a node v is connected

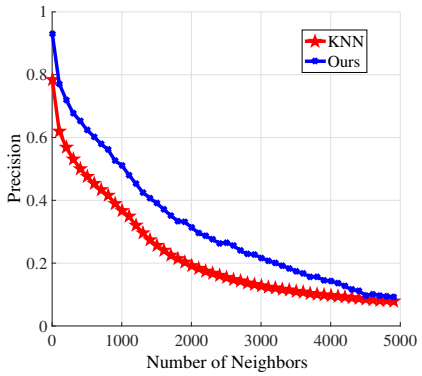


Fig. 3 Precision of constructed labels on cifar-10 dataset with different K, and different methods (deep features are used).

to a node u and if u is connected to a node w, we can infer that v is also connected to w. Inspired by this, if we treat every training image as a node in an undirected graph, we can expand the neighborhood of an image i by exploring the neighbors of its neighbors. Specifically, if $\mathbf{x}_i$ connects to $\mathbf{x}_j$ and $\mathbf{x}_j$ connects to $\mathbf{x}_k$, we can infer that $\mathbf{x}_i$ has the potential to be also connected to $\mathbf{x}_k$.

In view of the above, we use the initial similarity matrix $\mathbf{S}_1$ to expand the neighborhood structure. Specifically, based on $\mathbf{S}_1$, we calculate the similarity of two images by comparing the corresponding columns in $\mathbf{S}_1$ using the expression $\frac{1}{||(\mathbf{S}_1)_i-(\mathbf{S}_1)_j||^2}$. Then we again construct a ranked list of $K2$ neighbors, based on which we generate the second similarity matrix $\mathbf{S}_2$ as:

$$(\mathbf{S}_2)_{ij} = \begin{cases} 1, \text{ if } \mathbf{x}_j \text{ is K2-NN of } \mathbf{x}_i \\ -1, \text{ otherwise} \end{cases} \tag{13}$$

Finally, we construct the neighborhood structure by combining the direct and indirect similarities from the two matrices together. This results in the final similarity matrix $\mathbf{S}$:

$$S_{ij} = \begin{cases} 1, \text{ if } (\mathbf{S}_1)_{ij} = 1 \text{ or } \mathbf{x}_j \text{ is a K1-NN of } \mathbf{x}_i\text{'s K2-NN} \\ -1, \text{ otherwise} \end{cases} \tag{14}$$

The whole algorithm is shown in Alg. 1. We note here that we could have also omitted this preprocessing step and construct the neighborhood structure directly during the learning of our neural network. We found, however, that the construction of neighborhood structure is time-consuming, and that updating of this structure based on the updating of image features in each epoch does not have a significant impact on the performance. Therefore, we chose to obtain this neighborhood structure as described above and fix it for the rest of the process.

Table 2 mAP on CIFAR-10 using different optimization methods.

	mAP		
Methods	24-bit	32-bit	48-bit
two-step solution	0.344	0.362	0.373
$\text{sign}(z) = \lim_{\beta \to +\infty} \tanh(\beta z)$	**0.387**	**0.398**	**0.413**
$\text{sign}(z) = \lim_{\beta \to +\infty} app(\beta z)$	0.363	0.371	0.389

3.2.3 Neighborhood Structure Loss

The last section describes how to construct the similarity matrix and in this section we will present our objective function to preserve pair-wise similarity into hashing codes. Like [72], we define our neighbor loss as below:

$$\ell_N(x;\Lambda,\Phi) = \min_{\Lambda,\Phi} \sum_{s_{ij} \subset \mathbf{S}} \frac{1}{2} \left(\frac{1}{L} b_i^T b_j - s_{ij} \right)^2 \tag{15}$$

where Λ denotes the parameters of the retrieval network and $\mathbf{S}$ is constructed by Algorithm 1 and $s_{ij} \in \mathbf{S}$. Unfortunately, in Equation 15 b_i is discrete, whose gradient is zero for all nonzero inputs and leads to disable the back propagation to train the deep network. A wide range of works have proposed many novel methods to solve this problem. [40,78] proposed to use an approximate solution to relax the binary codes, however, which certainly a large quantization error. Therefore, relaxation the binary code is not efficient way to solve the discrete hashing problem.

In order to address this problem of optimizing binary codes with non-smooth *sign* activation, we acquire the inspiration from recent works [6,59]. These studies mainly focus on how to convert the difficult optimization problems into several easily optimized subproblems by changing the smoothness of the original function. Especially, we can gradually reduce the degree of the smoothness of function, which results in a sequence of subproblem optimizations converging to the original optimization problem. Following this idea, if we figure out the similar or approximate smooth function with $sign(.)$, and then gradually make the smooth function non-smooth during the training process, and finally, the results will converge to the desired target.

Motivated by this, we define a function $app(.)$ to approximate $sign(.)$:

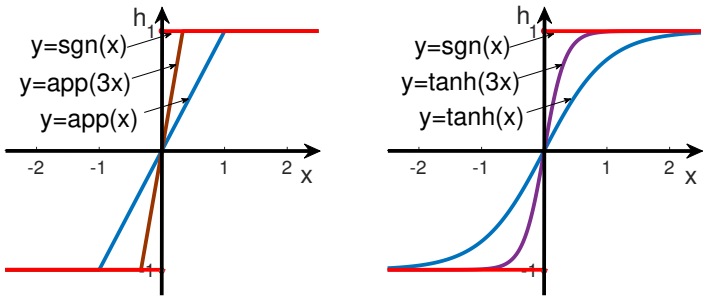
$$app(z) = \begin{cases} +1, & if\ z > 1 \\ z, & if\ 1 \geq z \geq -1 \\ -1, & if\ z < -1 \end{cases} \tag{16}$$

Obviously, there is a relationship between $sign(.)$ and $app(.)$ function:

$$sign(z) = \lim_{\beta \to \infty} app(\beta z) \tag{17}$$

In Fig. 4, we illustrate how $app(.)$ function approximates the original $sign(.)$. In addition, we also introduce an alternative $tanh(.)$:

$$\text{sign}(z) = \lim_{\beta \to +\infty} \tanh(\beta z) \tag{18}$$



(a) tanh(.) (b) app(.)

Fig. 4 Illustrative process of how app(.) and tanh(.) approximate sign(.)

Algorithm 2 Leaning parameters

Input: Images $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$
Initialize $\{\Phi, \Omega, \Theta\}$ randomly and Λ with the pre-trained model in [60].
for j=1,...,t **do**
  Sample x_i uniformly from $\{x_i\}_{i=1}^N$.
  Compute the stochastic gradient $\nabla \ell_C$ in 11.
    Update decoder parameters as
      $\Omega_{i+1} = \Omega_i - \tau_i \nabla_\Omega \ell_C$
  Compute the stochastic gradient $\nabla_\Phi \mathbf{h}(x;\Phi)$ in 7.
    Update encoder parameters as
      $\Phi_{i+1} = \Phi_i - \tau_i \nabla_\Phi \mathbf{h}(x;\Phi)$
  Compute the stochastic gradient ∇_{ℓ_A} in 10.
    Update discriminator parameters as
      $\Theta_{i+1} = \Theta_i - \tau_i \nabla_\Theta \ell_A(x;\Theta)$
  Compute the stochastic gradient ∇_{ℓ_N} in 19.
    Update encoder2 parameters as
      $\Lambda_{i+1} = \Lambda_i - \tau_i \nabla_\Lambda \ell_N(x;\Lambda)$
end for

Therefore, as for Equation 15, we can optimize $\mathbf{Z}$ instead of directly modeling the neighbors structure loss on the binary codes $\mathbf{B}$. Then Equation 15 can be reformulated as:

$$\ell_N(x;\Lambda,\Phi) = \frac{1}{2} \min_{\Lambda,\Phi} \sum_{s_{ij} \subset \mathbf{S}} \left(\frac{1}{L} z_i^T z_j - s_{ij} \right)^2 + \alpha \|\mathbf{Z} - \mathbf{B}\|^2 \tag{19}$$

where α is the hyper-parameter to balance the terms.

4 Learning

Using the loss functions in Eq.9, Eq.10 and Eq.19, 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 constant neuron layer to embed

the real-valued features into binary codes:

$$\mathbf{B}_i^c = h(E_n(\mathbf{I}_i; \Phi)) \tag{20}$$

where $\mathbf{I}_i$ is an input image, Φ is the parameter of the encoder, E_n is the encoder operation and h is the constant neuron layer. Then, $\mathbf{B}_i^c$ is the input for a generator (decoder) D_e to reconstruct an image $\mathbf{I}^R$:

$$\mathbf{I}_i^R = D_e(\mathbf{B}_i^c; \Omega) \tag{21}$$

where Ω stands for the parameters of the generator (decoder) D_e. Finally, a discriminator $\mathbf{D}$ assigns the probability

$$p = \mathbf{D}(\mathbf{I}_i^R; \Theta) \tag{22}$$

that $\mathbf{I}_i^R$ is an actual training sample and probability $1-p$ that $\mathbf{I}_i^R$ is generated by our model ${\mathbf{I}_\mathrm{i}}^R = D_e(\mathbf{B}_i^c; \Omega)$. Θ stands for the parameters of the discriminator $\mathbf{G}$.

Similarly, for the retrieval network, given an image $\mathbf{I}_i$, we can obtain its binary codes by:

$$\mathbf{B}_i = sign\,(E_{n2}(E_n(\mathbf{I}_i; \Phi); \Lambda)) \tag{23}$$

where E_{n2} is encoder2, and Λ represents its parameters.

As shown in Fig. 1, the compression network and retrieval network shared the top 5 convolutional layers, which allow us to train the entire network through a joint training strategy. In our network, we have parameters of $\{\Phi, \Omega, \Theta, \Lambda\}$ to learn. All parameters can be learned by back-propagation (BP). In particular, we randomly initialize $\{\Phi, \Omega, \Theta\}$ and use the pre-trained model [60] on ImageNet to initialize Λ. During each iteration, we sample a mini-batch of the images from the training data and use forward propagation to obtain the value of $\{\ell_C, \ell_N\}$, and then apply BP rules to update the associated parameters. The updated formulation is bellow:

$$\begin{aligned}
&\Omega_{i+1} \leftarrow \Omega_i - \tau_i \nabla_\Omega \ell_C \\
&\Phi_{i+1} \leftarrow \Phi_i - \tau_i \nabla_\Phi \mathbf{h}(x; \Phi) \\
&\Theta_{i+1} \leftarrow \Theta_i - \tau_i \nabla_\Theta \ell_A(x; \Theta) \\
&\{\Lambda_{i+1}, \Phi_{i+1}\} \leftarrow \{\Lambda_i, \Phi_{i+1}\} - \tau_i \nabla_{\Lambda,\Phi} \ell_N(x; \Lambda, \Phi)
\end{aligned} \tag{24}$$

when ∇ denotes the gradient and τ_i is the learning rate. The learning steps are shown in Alg. 2.

For the retrieval hashing codes, we set the $\beta = 1$ at the beginning. For each stage, after the retrieval network converges, we enlarge β for the next stage and use the parameters converging in the last stage to initialize the current stage parameters. By involving $app(\beta z)$ with $\beta \approx \infty$, the retrieval network obtains the same results as using $sign(z)$, which can learn exact binary hash codes as we desire. In the experiment, when we increase β to 10, the network can converge to the expected degree. In addition, we set $\lambda = 0.1$.

5 Experiments

We evaluate our BGAN+ on the task of large-scale image retrieval and image compression. Firstly, we compare BGAN+ with the state of the art methods both in image retrieval and compression. Secondly, we conduct an ablation study to evaluate the effect of each major component.

5.1 Datasets and Settings

To evaluate our method, we conduct our experiments on six public datasets: The Oxford 17 Category Flower [49], Stanford Dogs-120 [28], CIFAR-10 [31], Flickr-25K [47], NUS-WIDE [7], and Kodak [13]. Specifically, the first five datasets are used for retrieval, and NUS-WIDE and Kodak are used for image compression.

The Oxford 17 Category Flower dataset [49] contains 17 categories and each class consists of 80 images, resulting in a total of 1,360 images of flowers.

Stanford Dogs-120 [28] dataset consists of 20,580 images in 120 mutually categories. Each class contains about 150 images of dogs.

CIFAR-10 dataset consists of $60,000$ labeled tiny colored image (32×32). It is a single labeled dataset. Each image has a unique class label belonging to one of the 10 classes.

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

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 [80] and use the subset of 195,834 images from the 21 most frequent concepts, where each concept consists of at least 5,000 images.

Kodak contains 25 uncompressed color images of size 768×512. They are used as a standard test suite for compression testing.

In terms of retrieval, we split the Oxford Flower-17 into the training (40 images per class), validation (20 images per class), and test (20 images per class) sets. The Stanford Dogs-120 is divided into two parts: the train set (100 images per class) and test set (totally 8580 images for all categories). 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.

In terms of compression, we randomly select $21,000$ images ($1,000$ per class) to train our compression network. After the training, we apply the trained model to evaluate the performance on two testing datasets: 1) randomly select $10,000$ from NUS-WIDE dataset as the first testing dataset and 2) the Kodak dataset.

5.1.1 Evaluation Metric

For retrieval Task, the 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

Table 3 mAP results for fine-grained image retrieval using different number of bits on Oxford Flower-17 and Stanford Dogs-120. Note that our method is unsupervised while FashH is a supervised method.

	Oxford Flower-17				Stanford Dogs-120			
Bits	12	24	32	48	12	24	32	48
SH [74]	0.589	0.589	0.588	0.587	0.008	0.008	0.008	0.008
ITQ-CCA [16]	0.585	0.587	0.587	0.586	0.008	0.008	0.008	0.008
SDH [59]	0.108	0.140	0.117	0.145	0.009	0.018	0.090	0.037
KSH [44]	0.243	0.501	0.253	0.355	0.014	0.123	0.136	0.193
FastH [32]	0.402	0.524	0.528	0.536	0.044	**0.223**	**0.364**	**0.393**
DQN [5]	0.476	0.537	0.562	0.573	0.009	0.013	0.035	0.053
DSH [42]	0.566	0.614	0.637	0.680	0.012	0.012	0.012	0.012
BGAN+	**0.706**	**0.712**	**0.735**	**0.738**	**0.163**	0.192	0.215	0.235

value obtained for the set of top-k results, and this value is then averaged over all the queries. 2) **Precision**: We further use the precision-recall curve and precision@K to evaluate the precision of retrieved images.

For compression task, we use MS-SSIM [73] to test the quality of the image. The higher MS-SSIM means better quality.

5.1.2 Compared Methods

For retrieval Task, 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 [16], spectral hashing (SH) [24], Locality Sensitive Hashing (LSH) [10] , Spherical Hashing [24]), and two unsupervised deep hashing methods (DeepBit [40] and Deep Hashing (DH) [41]). To make a fair comparison, we also apply the non-deep hashing methods on deep features extracted by the VGG network (VGG-fc7 [60]).

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, named as (BGAN+_s). Therefore, we also compare with some supervised hashing methods, e.g., iterative quantization hashing (ITQ-CCA) [16], KSH [45], minimal loss hashing (MLH) [50], CNNH [76] and Deep Hashing Network (DHN) [81]. For compression task, we compare with four widely used image compression approaches: JPEG [67], JPEG 2000 [53], Theis *et al.* [64] and JPEG 420.

5.1.3 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: CIFAR-10, NUS-WIDE, and Flickr-25K. By default, we set $\lambda = 0.1$ and the learning rate as 0.001.

Table 4 mAP for different unsupervised hashing methods using a different number of bits on two image datasets. The first four methods are non-deep hashing methods, and the second the four methods are based on deep networks.

	Cifar-10				NUS-WIDE			
Bits	12	24	32	48	12	24	32	48
ITQ [16]	0.162	0.169	0.172	0.175	0.452	0.468	0.472	0.477
SH [74]	0.131	0.135	0.133	0.130	0.433	0.426	0.426	0.423
LSH [10]	0.121	0.126	0.120	0.120	0.403	0.421	0.426	0.441
Spherical [24])	0.138	0.141	0.146	0.150	0.413	0.413	0.424	0.431
ITQ+VGG	0.196	0.246	**0.289**	**0.301**	0.435	0.435	0.548	0.435
SH+VGG	0.174	0.205	0.220	0.232	0.433	0.426	0.426	0.423
LSH+VGG	0.101	0.128	0.132	0.169	0.401	0.442	0.480	0.471
Spherical+VGG	**0.212**	**0.247**	0.256	0.281	**0.549**	**0.614**	**0.653**	**0.678**
DeepBit [40]	0.185	0.218	0.248	0.263	0.383	0.401	0.403	0.412
DH [41]	0.160	0.164	0.166	0.168	0.422	0.448	0.480	0.493
BGAN_non [61]	0.361	0.369	0.375	0.395	0.518	0.541	0.545	0.568
BGAN [61]	0.401	0.512	0.531	0.558	0.675	0.690	0.714	0.728
BGAN+_non	0.375	0.387	0.398	0.413	0.544	0.552	0.561	0.579
BGAN+	**0.531**	**0.543**	**0.564**	**0.586**	**0.682**	**0.719**	**0.723**	**0.736**

Table 5 mAP for different unsupervised hashing methods using a different number of bits on Flickr. The first four methods are non-deep hashing methods, and the second four methods are based on deep networks.

	Flickr			
Bits	12	24	32	48
ITQ [16]	0.544	0.555	0.560	0.570
SH [74]	0.531	0.533	0.531	0.529
LSH [10]	0.499	0.513	0.521	0.548
Spherical [24])	**0.569**	**0.559**	**0.583**	**0.572**
ITQ+VGG	0.553	0.548	0.545	0.560
SH+VGG	0.550	0.544	0.541	0.545
LSH+VGG	0.543	0.549	0.555	0.551
Spherical+VGG	0.552	0.547	0.546	0.545
DeepBit [40]	0.501	0.505	0.511	0.513
DH [41]	0.553	0.548	0.543	0.556
BGAN_non [61]	0.591	0.601	0.607	0.626
BGAN [61]	0.683	0.702	0.703	0.703
BGAN+_non	0.599	0.612	0.618	0.636
BGAN+	**0.715**	**0.719**	**0.723**	**0.736**

5.2 Results on Image Retrieval

5.2.1 The Effect of Binary Optimization

As discussed above, both BGAN and 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). The previous study on BGAN has verified our argument that the two-step solution is sub-optimal, and binary optimization can achieve better performance. In this section, we further study the effect of binary codes optimization on the performance of hash codes to

verify the robustness of our binary optimization approach. The performance results of BGAN+ on the CIFAR-10 are shown in Tab. 2. As shown in Tab. 2, our binary optimization can improve the performance of the learned binary codes. Specifically, the first app solution (Eq. 17) outperforms two-step solution by 1.9%, 0.9%, and 1.6% for 24, 32, and 48-bit hash codes, while the second solution $tanh$ (Eq. 18) improves it by 4.3%, 3.6%, and 4.0%. This verifies our argument on BGAN+ that two-step solution is sub-optimal, and binary optimization can achieve better performance. These experimental results show the robustness of our proposed binary optimization approach.

5.2.2 Comparison with State-of-the-art methods for Fine-grained Image Retrieval

In this section, we compare our BGAN+with state-of-the-art methods for fine-grained image retrieval on two datasets. The mAP results are shown in Tab. 3.

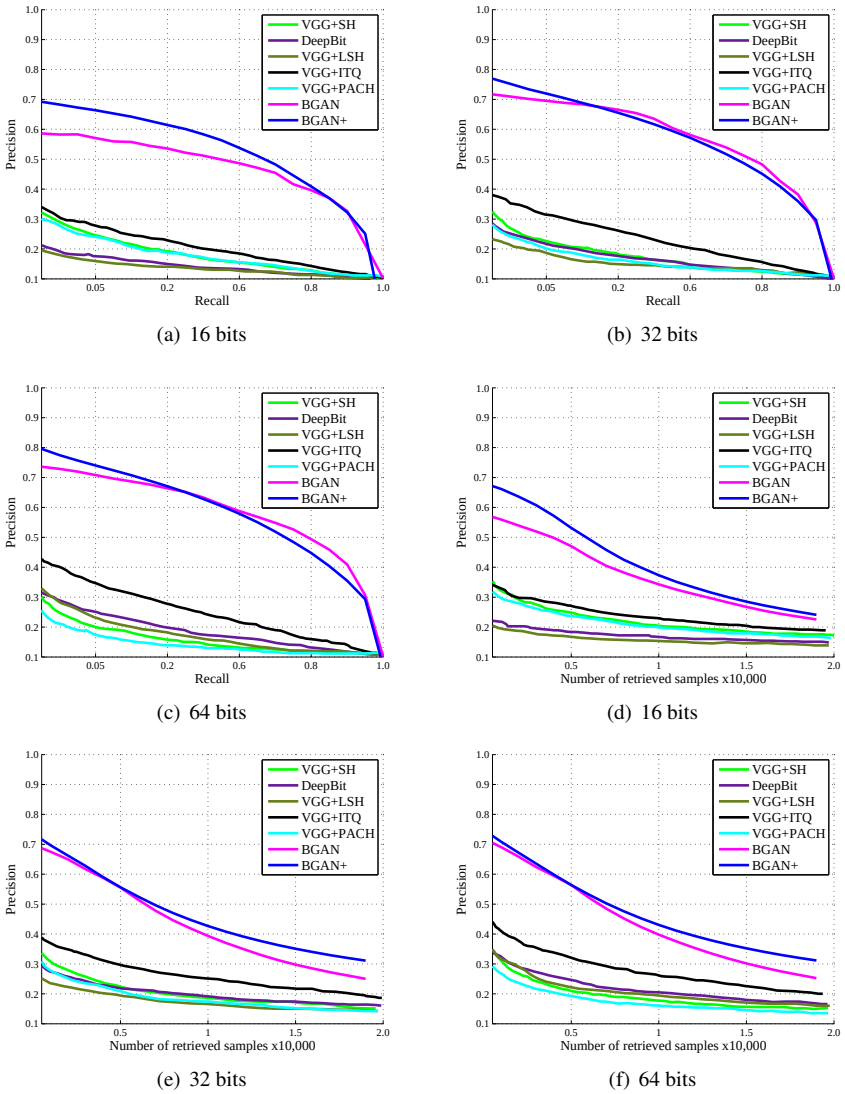
It shows that our method (BGAN+) significantly outperforms the other unsupervised hashing methods (SH and ITQ+CCA) in both datasets. In Oxford 17 Category Flower dataset, BGAN+ outperforms the best counterpart (SH) by 11.7%, 12.3%, 14.7% and 15.2% for 12, 24, 32 and 48 bits, respectively. On the other hand, both SH and ITQ+CCA have unsatisfactory performance in Stanford Dogs-120 dataset. Their mAP is 0.008 for different bits, which is almost random. This indicates that hashing methods for general image retrieval may not work well on the task of fine-grained image retrieval. Compared with several supervised hashing methods, e.g., SDH [59], KSH [44], FastH [32], DQN [5], DQN [5], our BGAN+, as an unsupervised method, achieves even better performance in Oxford 17 Category Flower dataset. BGAN+ outperforms the best counter-part (DSH) by 14.0%, 9.8%, 9.8% and 5.8% for 12, 24, 32 and 48 bits, respectively. However, in the Stanford Dogs-120 dataset, FastH has better performance in general, and it is better than BGAN+ by 3.1%, 14.9% and 15.8% for 24, 32 and 48 bits. Nevertheless, FastH is a supervised hashing method while our BGAN+ is unsupervised.

5.2.3 Comparison with State-of-the-art methods for General Image Retrieval

In this section, we evaluate our hashing method performance on three datasets. The mAP results are shown in Tab. 4 and Tab. 5 and Precision-Recall curves are shown in Fig. 5. From Tab. 4 and 5, we can obtain following conclusions:

First, our method (BGAN+) significantly outperforms the other deep or non-deep hashing methods in all datasets. In CIFAR-10, the improvement of BGAN+ over other methods is more significant, compared with NUS-WIDE and Flickr dataset. Specifically, it outperforms the best counter-part (Spherical+VGG) by 31.9%, 29.6%, 30.8% and 30.5% for 12, 24, 32 and 48-bit 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 method by 13.3%, 10.5%, 7.0% and 5.8% in the NUS-WIDE dataset, and 14.6%, 16.0%, 14.0% and 16.4% in Flicker dataset.

Second, comparing with BGAN (or BGAN+), the performance of BGAN_non (or BGAN+_non) is worse. This indicates that the similarity graph plays an important



(a) 16 bits (b) 32 bits

(c) 64 bits (d) 16 bits

(e) 32 bits (f) 64 bits

Fig. 5 Precision for different unsupervised hashing methods using different number of bits on CIFAR-10 dataset.

role in the learning of hashing codes, and the non-deep features are not as good as deep features.

Third, from Tab. 4 and 5, we observe that Spherical+VGG is a strong competitor in terms of mAP. On the other hand, the performance of deep hashing methods (i.e., DeepBit and DH) is not superior. A possible reason is that the deep hashing methods use only 3 fully connected layers to extract the features, which is not powerful.

Table 6 mAP for different supervised hashing methods using different number of bits on CIFAR-10

	CIFAR-10			
Method	12 bits	24 bits	32 bits	48 bits
ITQ-CCA [16]	0.435	0.435	0.435	0.435
KSH [45]	0.556	0.572	0.581	0.588
MLH [50]	0.500	0.514	0.520	0.522
DNNH [32]	0.674	0.697	0.713	0.715
CNNH [76]	0.611	0.618	0.625	0.608
DHN [81]	**0.708**	**0.735**	**0.748**	**0.758**
BGAN_s	0.866	0.874	0.876	0.877
BGAN+_s	**0.884**	**0.889**	**0.892**	**0.894**

Fourth, when we run the non-deep hashing method on deep features, the performance is usually improved compared with the hand-craft features. The performance gap is larger in CIFAR-10 and NUS-WIDE datasets than in Flicker dataset.

Fifth, with the increase of hash code length, the performance of most hashing method 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.

Sixth, compared with BGAN (or BGAN_non), BGAN+ (or BGAN+_non) achieves better performance. In particular, the increase of BGAN+ of 12-bit on the CIFAR-10 dataset is 13.0%. In addition, BGAN+ improves BGAN by 3.2%, 1.7%, 2% and 3.3% of 12, 24, 32 and 48-bit on the Flick dataset.

From Fig. 5, we have the following observations. In terms of the precision-recall curve, the results indicate that BGAN and BGAN+ significantly outperform existing approaches. In general, BGAN+ performs better than BGAN, especially when the hash code is 16-bit. In addition, the bottom row of Fig. 5 shows the precision curves when we set a different number of retrieved samples (times of $10,000$) and then train the model with 16, 32 and 64-bit, separately. When code length is 16-bit, BGAN+ achieves better performance. When code length is set as 32 or 64-bit and the number of retrieved samples is $5,000$, BGAN and BGAN+ obtain the same value of precision. In addition, when the number of retrieved samples gradually increases from $5,000$, the gap between BGAN and BGAN+ increases gradually.

We also compared with supervised hashing methods, and present the mAP results on CIFAR-10 dataset in Tab. 6. It is observed that our BGAN+ reaches the highest mAP scores across hashing code length ranging from 12-bit to 48-bit. Compared with the best deep supervised hashing method DHN, BGAN+ has an increase of 17.6%, 15.0%, 14.4% and 13.6% over 12, 24, 32 and 48-bit. In generally, BGAN+ increases the BGAN by around 2.0% and the improvement is contributed by our compression network. 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 a large room to be improved for the task of supervised hashing.

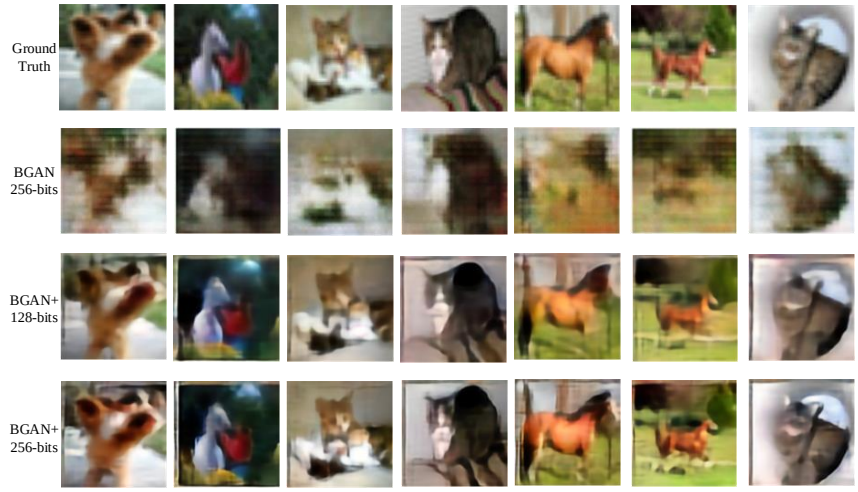


Fig. 6 Reconstructed images on CIFAR-10 using binary codes.

Table 7 MS-SSIM on NUS-WIDE at different bit-rate.

	MS-SSIM		
Methods	0.15 bit/px	0.25 bit/px	0.5 bit/px
JPEG [67]	0.875	0.894	0.922
JPEG 2000 [53]	0.925	0.937	0.945
BGAN	**0.927**	**0.939**	**0.948**

Table 8 MS-SSIM on Kodak at different bit-rate.

	MS-SSIM		
Methods	0.15 bit/px	0.25 bit/px	0.5 bit/px
JPEG [67]	0.802	0.844	0.945
JPEG 2000 [53]	0.903	0.922	**0.951**
Theis *et al.* [64]	0.901	0.920	0.948
JPEG 420	0.824	0.891	0.950
BGAN	**0.906**	**0.924**	0.949

5.3 Results on Image Compression

JPEG is an image compression standard approach, while JPEG 200 is an improvement on JPEG. They are both widely used for image compression. To evaluate the performance of our compression network, we use NUS-WIDE dataset to train our compression model and then evaluate it on two datasets: NUS-WIDE and Kodak dataset.

The experimental results obtained from the NUS-WIDE dataset are shown in Tab. 7, which demonstrates that our BGAN+ obtains the best performance in terms of MS-SSIM. Compared with JPEG, BGAN+ has an increase of 5.2%, 4.5% and 2.6% for

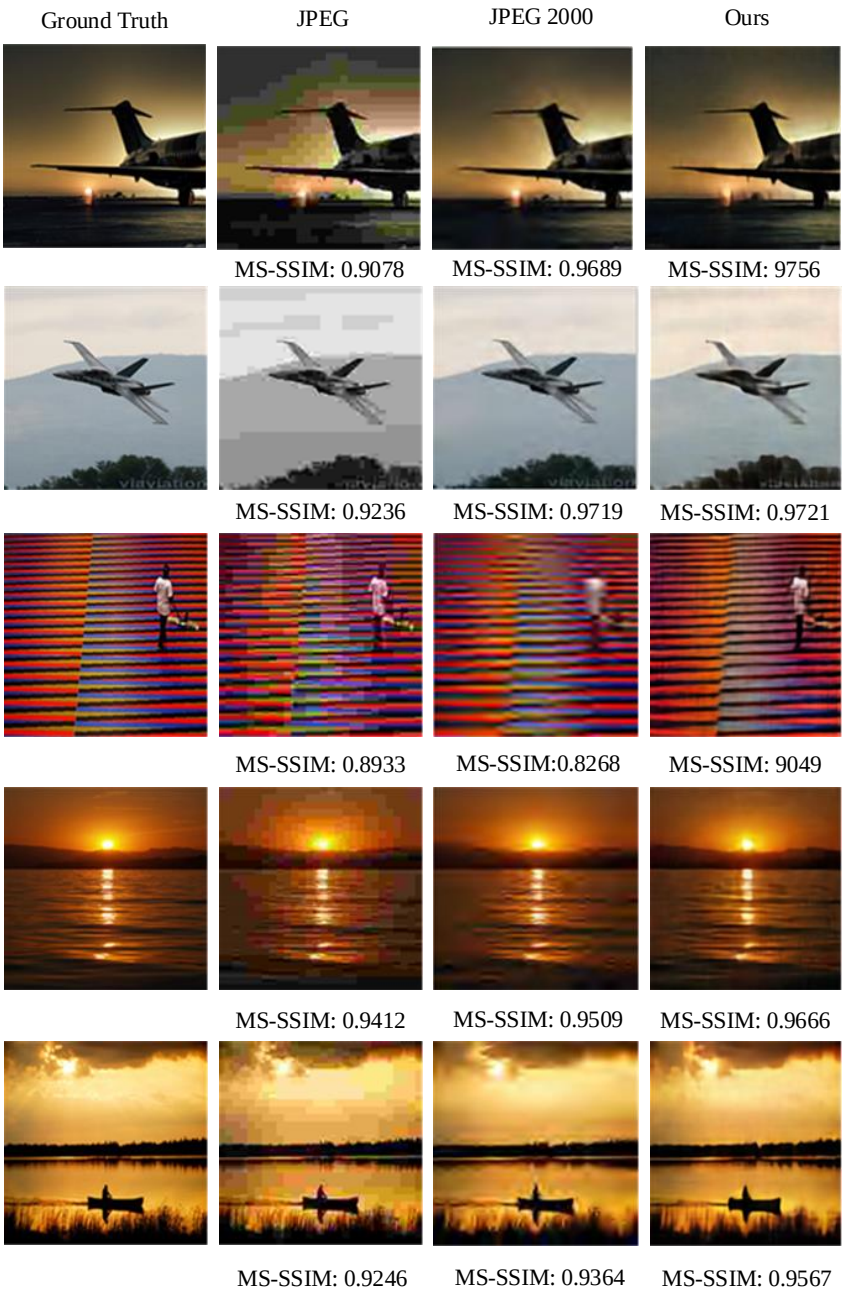


Fig. 7 Samples from NUS-WIDE dataset for visualization (0.15 bit/px)

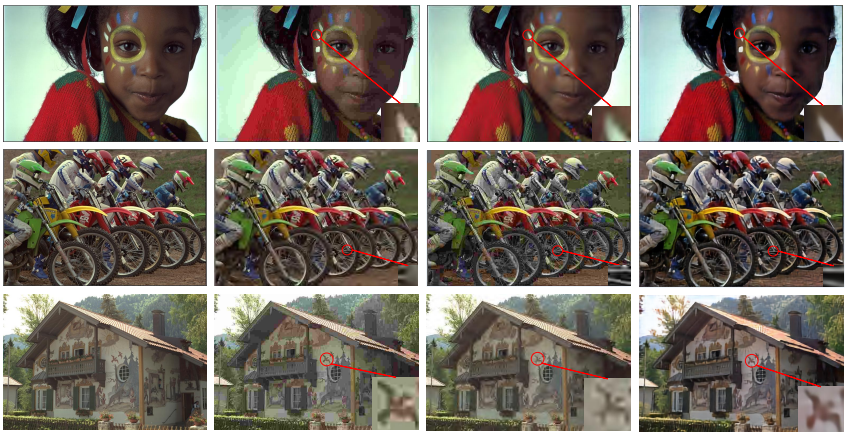


Fig. 8 Samples from Kodak dataset for visualization. Ground truth, JPEG, JPEG 2000 and Ours BGAN+ are from left to right (0.15 bit/pix).

0.15, 0.25 and 0.5 bit/px. Specifically, the improvement gap becomes narrow with the increase of bit-rate. The lower the bit-rate is, the harder the compression operation is. Furthermore, our BGAN+ performs slightly better than JPEG2000.

In order to test the robustness of our compression network, we further run the trained model on the image compression benchmark dataset Kodak and the experimental results are shown in Tab. 8. From Tab. 8, we can see that our BGAN+ performs the best, and JPEG 2000 goes the second. Compared with JPEG, BGAN+ performs better with the increase of 10.4%, 8.0% and 0.4% for 0.15, 0.25 and 0.5 bit/px, respectively. However, for the 0.5 bit/px, our method is slightly outperformed by JPEG 2000 and JPEG 420 by 0.2% and 0.1% respectively. Generally speaking, our method is more competitive at a lower bit rate because the reconstructed image is generated from the binary codes, which are highly compact codes.

To evaluate the ability of image reconstruction using BGAN+ and to compare with the previous BGAN proposed in [61], we demonstrate some qualitative results on CIFAR-10 dataset in Fig. 6. From Fig. 6, we can see that the images reconstructed from BGAN with 256-bit hash code are blurry compared with the ground-truth images. Compared with the images reconstructed from BGAN with 256-bit, BGAN+ can generate a higher quality of images with only 128-bit hash code. This indicates the effectiveness of our BGAN+ for image compression. With longer hash code (i.e., 256-bit), it can produce even better quality images, which is as good as the ground-truth images from the human visual aspect.

In addition, more visual examples are provided in Fig. 7 and Fig. 8. All the images are randomly selected from the NUS-WIDE and Kodak dataset respectively. In Fig. 7, each image is compressed by JPEG, JPEG 2000 and our BGAN+ and their corresponding MS-SSIM values are provided. The higher the MS-SSIM is the better the compression results are. All the examples indicate that our BGAN+ performs the best. While for some examples (i.e., the third row), JPEG performs better than JPEG 2000. In terms of human visual visualization, in generally the images generated by

Table 9 The mAP of BGAN+ on CIFAR-10 using different combinations of components (Without jointly learning with compression net and with it).

	mAP		
Components	24-bit	32-bit	48-bit
STL-BGAN+	0.511	0.533	0.567
MTL-BGAN+	**0.543**	**0.564**	**0.586**

Table 10 MS-SSIM on NUS-WIDE at different bit-rate.

	MS-SSIM		
Methods	0.15 bit/px	0.25 bit/px	0.5 bit/px
STL-BGAN	0.914	0.929	0.932
MTL-BGAN+	**0.927**	**0.939**	**0.948**

the JPEG is blurring, while BGAN+ and JPEG provide images with high-resolution images. Fig. 8 shows that our BGAN+ can reconstruct images with photo-realistic details.

5.4 Evaluation of Individual Component

To verify the effects of individual components (i.e., retrieval network and compression network) in our framework and show each of them contributes to the performance boost, we evaluate two variants of our approaches. Instead of using multi-task learning (MTL), we assume tasks are independent and learn retrieval network and compression network separately. The resulting retrieval model (ℓ_N) is acquired based on single task learning (STL) by utilizing ℓ_N loss only to train the retrieval network. In this way, STL trains its retrieval model separately without sharing the first five conv layers with compression network. For the retrieval task, the experiments are conducted on the CIFAR-10 dataset and the experimental results are shown in Tab. 9. The results by STL is worse than by MTL with a decrease of 3.2%, 3.1% and 1.9% over 24, 32 and 48-bit, respectively.

In the second experiment conducted on the NUS-WIDE dataset, we compare BGAN+ trained by MTL with BGAN+ trained by STL for compressing images and the results are shown in Tab. 10. The results show that MTL-BGAN+ outperforms STL-BGAN+ by 1.3%, 1.0% and 1.6% for 0.15, 0.25 and 0.5 bit/px respectively. These two experiments indicate that multi-task learning framework using the retrieval and compression network is beneficial for both image retrieval and compression tasks. This is due to the reason that learning-related tasks simultaneously can successfully exploit shared features among tasks and increase the discriminative ability of the learned models. As we can see from the experimental results, our method is able to simultaneously generate binary codes for image retrieval and compression.

Furthermore, we also test the efficiency of the constant neuron layer and conduct three different experiments as showing in Tab 11 and Tab 12. To be specific, we test the quality of reconstructed images from real value vectors and binary codes, which

is optimized by the two steps relaxation optimizing strategy. From Tab 11, we can see that our results from binary codes does not degrade much compared with the real value compression codes. Obviously, in the condition of the same dimension, BGAN+ just declines 0.017, 0.013 and 0.014 on four dimensions 7k, 12k and 25k, respectively. It is worth noting that those results are based on the same compression dimension but not the same compression rate. It is reasonable that the compression strategy of binary codes is worse than the real value because the binary codes lose more information compared to the real value with the same dimension. Tab 12 shows the results from the real value vector under the same compression rate. Here, we should note that the dimension of compression codes depends on the size of the input image, and set the input image size as 224×224. However, when based on the same compression rate, our method shows a significant superiority to the real value codes and improves the compression performance about 0.126, 0.115 and 0.092 on the 0.15, 0.25 and 0.5 bix/px. The last experiment is to evaluate different strategies to generate binary codes. From Tab 12, we can see that our constant neuron has a hug benefit to the binary learning compared to the two steps relaxation method and has about 0.283, 0.27 and 0.254 improvement on the three compression rates. The reason for this scenario is that two steps approximate strategy can lead to huge quantization errors. To sum up, through the three experiments, we can gain the following conclusions: 1) Compared with the reconstruction from the real value vector, BGAN+ has the comparable performance but less storage cost. 2) The constant neuron layer can directly optimize binary codes and avoid large quantization errors, which is the key unit to ensure the high quality reconstructed image and low storage space.

Table 11 MS-SSIM on NUS-WIDE based on different dimensions with/without the layer of constant neuron.

	MS-SSIM		
Methods	7k	12k	25k
real value	**0.943**	**0.950**	**0.962**
BGAN+	0.926	0.937	0.948

Table 12 MS-SSIM on NUS-WIDE under different compression rate by different reconstruction strategies.

	MS-SSIM		
Methods	0.15 bit/px	0.25 bit/px	0.5 bit/px
real value	0.801	0.824	0.856
two steps	0.643	0.669	0.695
BGAN+	**0.927**	**0.939**	**0.948**

6 Conclusion

In this paper, we propose a unified binary generative adversarial networks (BGAN+) to simultaneously convert images to binary codes for both image compression and retrieval. in a multi-task fashion and 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 two binary representations per image: one for image retrieval and one for image compression. To equip the binary representation with the ability of accurate image retrieval and compression, we design a novel loss function. We also propose several solutions to address the gradient vanishing problem caused by *sign* function. Extensive experiments are conducted for image retrieval and compression. The results show that our BGAN+ outperforms the existing retrieval methods with significant margins and achieves competitive performance for image compression, especially for low bitrates. And the multi-task strategy is beneficial for both tasks. As far as we know, this is the first work of using binary codes for image compression.

Acknowledgment

This work is supported by the Fundamental Research Funds for the Central Universities (Grant No. ZYGX2014J063, No. ZYGX2014Z007) and the National Natural Science Foundation of China (Grant No. 61502080, No. 61632007, No. 61602049).

References

1. Agustsson, E., Mentzer, F., Tschannen, M., Cavigelli, L., Timofte, R., Benini, L., Gool, L.V.: Soft-to-hard vector quantization for end-to-end learned compression of images and neural networks. CoRR **abs/1704.00648** (2017)
2. Baig, M.H., Koltun, V., Torresani, L.: Learning to inpaint for image compression. In: NIPS, pp. 1246–1255 (2017)
3. Ballé, J., Laparra, V., Simoncelli, E.P.: End-to-end optimized image compression. CoRR **abs/1611.01704** (2016)
4. Bellard, M.: Bpg image format (http://bellard.org/bpg/) (Accessed:2017-01-30. 1, 2)
5. Cao, Y., Long, M., Wang, J., Zhu, H., Wen, Q.: Deep quantization network for efficient image retrieval. In: AAAI, pp. 3457–3463 (2016)
6. Cao, Z., Long, M., Wang, J., Yu, P.S.: Hashnet: Deep learning to hash by continuation. In: ICCV, pp. 5609–5618 (2017)
7. 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: Proceedings of the ACM international conference on image and video retrieval, p. 48. ACM (2009)
8. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: ICML, pp. 160–167. ACM (2008)
9. Dai, B., Guo, R., Kumar, S., He, N., Song, L.: Stochastic generative hashing. In: ICML, pp. 913–922 (2017)
10. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Symposium on Computational Geometry (2004)
11. Do, T.T., Doan, A.D., Cheung, N.M.: Learning to hash with binary deep neural network. In: ECCV, pp. 219–234. Springer (2016)
12. Farvardin, N.: Review of 'vector quantization and signal compression' (gersho, a., and gray, r.m.; 1992). IEEE Trans. Information Theory **40**(1), 287 (1994)

13. Franzen, R.: Kodak lossless true color image suite. source: http://r0k. us/graphics/kodak **4** (1999)
14. Ge, T., He, K., Sun, J.: Graph cuts for supervised binary coding. In: ECCV, pp. 250–264 (2014)
15. Gong, Y., Kumar, S., Verma, V., Lazebnik, S.: Angular quantization-based binary codes for fast similarity search. In: NIPS, pp. 1205–1213 (2012)
16. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(12), 2916–2929 (2013)
17. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: NIPS, pp. 2672–2680 (2014)
18. Google: Webp: Compression techniques (http://developers.google.com/speed/webp/docs/compression). (Accessed:2017-01-30. 1, 2, 5)
19. Grubb, G.: Distributions and operators, vol. 252. Springer Science & Business Media (2008)
20. Gu, Y., Ma, C., Yang, J.: Supervised recurrent hashing for large scale video retrieval. In: ACM Multimedia, pp. 272–276. ACM (2016)
21. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Trans. Pattern Anal. Mach. Intell. **37**(9), 1904–1916 (2015)
22. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR, pp. 770–778 (2016)
23. Heo, J., Lee, Y., He, J., Chang, S., Yoon, S.: Spherical hashing: Binary code embedding with hyperspheres. IEEE Trans. Pattern Anal. Mach. Intell. **37**(11), 2304–2316 (2015)
24. Heo, J., Lee, Y., He, J., Chang, S., Yoon, S.: Spherical hashing: Binary code embedding with hyperspheres. IEEE Trans. Pattern Anal. Mach. Intell. **37**(11), 2304–2316 (2015)
25. Irie, G., Li, Z., Wu, X., Chang, S.: Locally linear hashing for extracting non-linear manifolds. In: CVPR, pp. 2123–2130 (2014)
26. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. PAMI **33**(1), 117–128 (2011)
27. Jin, Z., Hu, Y., Lin, Y., Zhang, D., Lin, S., Cai, D., Li, X.: Complementary projection hashing. In: ICCV, pp. 257–264 (2013)
28. Khosla, A., Jayadevaprakash, N., Yao, B., Fei-Fei, L.: Novel dataset for fine-grained image categorization. In: First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition. Colorado Springs, CO (2011)
29. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
30. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. CoRR **abs/1312.6114** (2013)
31. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
32. Lai, H., Pan, Y., Liu, Y., Yan, S.: Simultaneous feature learning and hash coding with deep neural networks. In: CVPR, pp. 3270–3278 (2015)
33. Larsen, A.B.L., Sønderby, S.K., Larochelle, H., Winther, O.: Autoencoding beyond pixels using a learned similarity metric. In: ICML, pp. 1558–1566 (2016)
34. Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A.P., Tejani, A., Totz, J., Wang, Z., Shi, W.: Photo-realistic single image super-resolution using a generative adversarial network. In: CVPR, pp. 105–114 (2017)
35. Li, M., Zuo, W., Gu, S., Zhao, D., Zhang, D.: Learning convolutional networks for content-weighted image compression. In: CVPR (2018)
36. Li, W., Wang, S., Kang, W.: Feature learning based deep supervised hashing with pairwise labels. In: IJCAI, pp. 1711–1717 (2016)
37. Li, X., Lin, G., Shen, C., van den Hengel, A., Dick, A.R.: Learning hash functions using column generation. In: ICML, pp. 142–150 (2013)
38. Lin, G., Shen, C., Shi, Q., van den Hengel, A., Suter, D.: Fast supervised hashing with decision trees for high-dimensional data. In: CVPR, pp. 1971–1978 (2014)
39. Lin, G., Shen, C., Suter, D., Van Den Hengel, A.: A general two-step approach to learning-based hashing. In: ICCV, pp. 2552–2559. IEEE (2013)
40. Lin, K., Lu, J., Chen, C., Zhou, J.: Learning compact binary descriptors with unsupervised deep neural networks. In: CVPR, pp. 1183–1192 (2016)
41. Liong, V.E., Lu, J., Wang, G., Moulin, P., Zhou, J.: Deep hashing for compact binary codes learning. In: CVPR, pp. 2475–2483 (2015)
42. Liu, H., Wang, R., Shan, S., Chen, X.: Deep supervised hashing for fast image retrieval. In: CVPR, pp. 2064–2072 (2016)
43. Liu, L., Shao, L., Shen, F., Yu, M.: Discretely coding semantic rank orders for supervised image hashing. In: CVPR, pp. 5140–5149 (2017)

44. Liu, W., Wang, J., Ji, R., Jiang, Y., Chang, S.: Supervised hashing with kernels. In: CVPR, pp. 2074–2081 (2012)
45. Liu, W., Wang, J., Ji, R., Jiang, Y.G., Chang, S.F.: Supervised hashing with kernels. In: CVPR, pp. 2074–2081 (2012)
46. Liu, X., He, J., Deng, C., Lang, B.: Collaborative hashing. In: CVPR, pp. 2147–2154 (2014)
47. Mark J. Huiskes, B.T., Lew, M.S.: New trends and ideas in visual concept detection: The mir flickr retrieval evaluation initiative. In: MIR '10: Proceedings of the 2010 ACM International Conference on Multimedia Information Retrieval, pp. 527–536 (2010)
48. Nemirovski, A., Juditsky, A., Lan, G., Shapiro, A.: Robust stochastic approximation approach to stochastic programming. SIAM Journal on Optimization **19**(4), 1574–1609 (2009)
49. Nilsback, M., Zisserman, A.: A visual vocabulary for flower classification. In: CVPR, pp. 1447–1454 (2006)
50. Norouzi, M., Blei, D.M.: Minimal loss hashing for compact binary codes. In: ICML, pp. 353–360 (2011)
51. Norouzi, M., Fleet, D.J.: Cartesian k-means. In: CVPR, pp. 3017–3024 (2013)
52. Odena, A., Dumoulin, V., Olah, C.: Deconvolution and checkerboard artifacts. Distill **1**(10), e3 (2016)
53. Rabbani, M., Joshi, R.L.: An overview of the JPEG 2000 still image compression standard. Sig. Proc.: Image Comm. **17**(1), 3–48 (2002)
54. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. CoRR **abs/1511.06434** (2015)
55. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
56. Rippel, O., Bourdev, L.D.: Real-time adaptive image compression. In: ICML, pp. 2922–2930 (2017)
57. Shannon, C.E.: A mathematical theory of communication. Mobile Computing and Communications Review **5**(1), 3–55 (2001)
58. Shen, F., Mu, Y., Yang, Y., Liu, W., Liu, L., Song, J., Shen, H.T.: Classification by retrieval: Binarizing data and classifiers. In: SIGIR, pp. 595–604 (2017)
59. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. In: CVPR, pp. 37–45 (2015)
60. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014)
61. Song, J., He, T., Gao, L., Xu, X., Hanjalic, A., Shen, H.T.: Binary generative adversarial networks for image retrieval. In: AAAI (2018)
62. Song, J., Yang, Y., Yang, Y., Huang, Z., Shen, H.T.: Inter-media hashing for large-scale retrieval from heterogeneous data sources. In: SIGMOD, pp. 785–796 (2013)
63. Strecha, C., Bronstein, A.M., Bronstein, M.M., Fua, P.: Ldahash: Improved matching with smaller descriptors. IEEE Trans. Pattern Anal. Mach. Intell. **34**(1), 66–78 (2012)
64. Theis, L., Shi, W., Cunningham, A., Huszr, F.: Lossy image compression with compressive autoencoders. In: ICLR (2017)
65. Toderici, G., O'Malley, S.M., Hwang, S.J., Vincent, D., Minnen, D., Baluja, S., Covell, M., Sukthankar, R.: Variable rate image compression with recurrent neural networks. arXiv preprint arXiv:1511.06085 (2015)
66. Toderici, G., Vincent, D., Johnston, N., Hwang, S.J., Minnen, D., Shor, J., Covell, M.: Full resolution image compression with recurrent neural networks. CoRR **abs/1608.05148** (2016)
67. Wallace, G.K.: The JPEG still picture compression standard. Commun. ACM **34**(4), 30–44 (1991)
68. Wang, B., Yang, Y., Xu, X., Hanjalic, A., Shen, H.T.: Adversarial cross-modal retrieval. In: ACM Multimedia, pp. 272–276 (2017)
69. Wang, J., Kumar, S., Chang, S.: Sequential projection learning for hashing with compact codes. In: ICML, pp. 1127–1134 (2010)
70. Wang, J., Liu, W., Sun, A.X., Jiang, Y.: Learning hash codes with listwise supervision. In: ICCV, pp. 3032–3039 (2013)
71. Wang, J., Wang, J., Yu, N., Li, S.: Order preserving hashing for approximate nearest neighbor search. In: ACM Multimedia, pp. 133–142. ACM (2013)
72. Wang, J., Zhang, T., Song, J., Sebe, N., Shen, H.T.: A survey on learning to hash. IEEE Trans. Pattern Anal. Mach. Intell. **40**(4), 769–790 (2018)
73. Wang, Z., Simoncelli, E.P., Bovik, A.C.: Multiscale structural similarity for image quality assessment. In: Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on, vol. 2, pp. 1398–1402 (2003)
74. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS, pp. 1753–1760 (2008)

75. Wintz, P.A.: Transform picture coding. Proceedings of the IEEE **60**(7), 809–820 (1972)
76. Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S.: Supervised hashing for image retrieval via image representation learning. In: AAAI, vol. 1, p. 2 (2014)
77. Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S.: Supervised hashing for image retrieval via image representation learning. In: AAAI, pp. 2156–2162 (2014)
78. Zhang, P., Zhang, W., Li, W., Guo, M.: Supervised hashing with latent factor models. In: SIGIR, pp. 173–182 (2014)
79. Zhao, F., Huang, Y., Wang, L., Tan, T.: Deep semantic ranking based hashing for multi-label image retrieval. In: CVPR, pp. 1556–1564. IEEE (2015)
80. Zhu, H., Long, M., Wang, J., Cao, Y.: Deep hashing network for efficient similarity retrieval. In: AAAI, pp. 2415–2421 (2016)
81. Zhu, H., Long, M., Wang, J., Cao, Y.: Deep hashing network for efficient similarity retrieval. In: AAAI, pp. 2415–2421 (2016)