PBGen: Partial Binarization of Deconvolution-Based Generators for Edge Intelligence

Jinglan Liu\textsuperscript{1}, Jiaxin Zhang\textsuperscript{2}, Yukun Ding\textsuperscript{1}, Xiaowei Xu\textsuperscript{1}, Meng Jiang\textsuperscript{1}, and Yiyu Shi\textsuperscript{1}

\textsuperscript{1} University of Notre Dame  
\{jliu16, yding5, xxu8, mjiang2, yshi4\}@nd.edu  
\textsuperscript{2} University of Science and Technology of China  
zjx219@mail.ustc.edu.cn

Abstract. This work explores the binarization of the deconvolution-based generator in a GAN for memory saving and speedup of image construction. Our study suggests that different from convolutional neural networks (including the discriminator) where all layers can be binarized, only some of the layers in the generator can be binarized without significant performance loss. Supported by theoretical analysis and verified by experiments, a direct metric based on the dimension of deconvolution operations is established, which can be used to quickly decide which layers in the generator can be binarized. Our results also indicate that both the generator and the discriminator should be binarized simultaneously for balanced competition and better performance. Experimental results based on CelebA suggest that directly applying state-of-the-art binarization techniques to all the layers of the generator will lead to 2.83× performance loss measured by sliced Wasserstein distance compared with the original generator, while applying them to selected layers only can yield up to 25.81× saving in memory consumption, and 1.96× and 1.32× speedup in inference and training respectively with little performance loss.

Keywords: generative adversarial network, binarization, deconvolution

1 Introduction

In recent years, generative adversarial networks (GANs), which are spin-offs from conventional convolutional neural networks (CNNs), have attracted much attention in the fields of reinforcement learning, unsupervised learning and also semi-supervised learning \cite{123}. Some promising applications based on GANs include images reconstruction with super-resolution, art creation and image-to-image translation \cite{4}, many of which can run on mobile devices (edge computing). For example, one potential application of GANs allow videos to be broadcast in low resolution and then reconstructed to ultra-high resolution by end users \cite{5} as shown in Fig. 1.
However, the resources required by GANs to perform computations in real-time may not be easily accommodated by mobile devices. For example, constructing an image of 64×64 resolution with deep convolutional generative adversarial network (DCGAN) requires 86.6 MB of memory, most of which is used for the generator. The memory goes up to 620.8 MB for 1024×1024 resolution, and up to about 800 MB for the popular 4K video with resolution of 3840×2160. On the other hand, one of the state-of-the-art mobile processors, A11 in the newest iPhone X, provides only 3 GB RAM, most of which must be occupied by the operating system and its peripherals. As a result, developers must restrict neural network models to just a few megabytes to avoid crash. The memory budget gets even tighter when it comes to mobile devices of smaller form factor such as Apple Watch series 3, which only has 786 MB RAM.

The same problem has been well known for conventional CNNs, and various solutions have been proposed via redesigning the algorithms and/or computation structures. Among them, quantization until binary is one of the most popular techniques as it fits hardware implementation well with high efficiency. Its success on CNNs has been demonstrated by multiple works, where memory consumption is deeply compressed although sometimes the performance cannot be preserved.

Compression techniques can be readily applied to discriminator networks in GANs, which are nothing different from conventional CNNs. It may be alluring to also apply it to binarize generators, especially the deconvolution-based ones as the computation process looks similar. However, instead of distilling local information from a global map as in convolution operations, deconvolution attempts to construct the global map by local information. This difference can lead to significantly different binarization results, as will be discussed in Section 3. Accordingly, a scheme tailored to deconvolution-based generators is warranted.

In this paper, we show through theoretical analysis that under certain conditions, binarizing a deconvolution layer may cause significant performance loss, which also happens in compression of CNNs. Since there is no explanation for this phenomenon to the best of the authors’ knowledge, an intuitive guess is that not all layers can be binarized together while preserving performance. Thus, some layers need to stay in the format of floating point for performance, while others can be binarized without affecting performance. To quickly decide whether a layer can be binarized, supported by theoretical analysis and verified by experiments, a simple yet effective metric based on the dimension of deconvolution
operations is established. Based on this metric, we can make use of existing compression techniques to binarize the generator of GANs with little performance loss. We then propose the scheme of partial binarization of deconvolution-based generators (PBGen) under the guide of the metric.

Furthermore, we find that only binarizing the generator and leaving discriminator network unchanged will introduce unbalanced competition and performance degradation. Thus, both networks should be binarized at the same time. Experimental results based on CelebA suggest that directly applying state-of-the-art binarization techniques to all the layers of the generator will lead to 2.83× performance loss measured by sliced Wasserstein distance compared with the original generator, while applying them to selected layers only can yield up to 25.81× saving in memory consumption, and 1.96× and 1.32× speedup in inference and training respectively with little performance loss.

2 Related Works and Background

2.1 CNN Compression

Compression techniques for CNNs mainly consist of pruning, quantization, re-structure and other approximations based on mathematical matrix manipulations [10,18,19]. The main idea of the pruning method in [14] is to “prune” connections with smaller weights out so that both synapses and neurons are possible to be removed from the original structure. This can work well with traditional CNNs and reduce the number of parameters of AlexNet by a factor of nine [14]. Re-structure methods modify network structures for compression, such as changing functions or block order in layers [19,20].

In this work, we focus on the quantization technique. Quantization aims to use fewer bits to present values of weights or even inputs. It has been used to accelerate CNNs in various works at different levels [21,22] including ternary quantization [11,23] and iterative quantization [24], with small loss. In [10], the authors proposed to determine weight sharing after a network is fully trained, so that the shared weights approximate the original network. From a fully trained model, weights are clustered and replaced by the centroids of clusters. During retraining, the summation of the gradients in same groups are used for the fine-tuning of the centroids. Through such quantization, it is reported to be able to compress AlexNet up by around 8% before significant accuracy loss occurs. If the compression rate goes beyond that, the accuracy will deteriorate rapidly.

A number of recent works [13,16,20,25,26,27] pushed it further by using binarization to compress CNNs, where only a single bit is used to represent values. Training networks with weights and activations constrained to ±1 was firstly proposed in [26]. Through transforming 32-bit floating point weight values to binary representation, CNNs with binary weights and activations are about 32× smaller. In addition, when weight values are binary, convolutions can be estimated by only addition and subtraction without multiplication, which can achieve around 2.0× speedup. However, the method introduces significant performance loss. To
alleviate the problem, [20] proposed Binary-Weight-Network, where all weight values are binarized with an additional continuous scaling factor for each output channel. We will base our discussion on this weight binarization, which is one of the state-of-the-art binarization methods.

However, none of the existing works explored the compression of generators in GANs, where deconvolution replaces convolution as the major operation. Note that while there is a recent work that uses the term of “binary generative adversarial networks” [28], it is not about the binarization of GANs. In that work, only the inputs of the generator are restricted to binary code to meet the specific application requirement. All parameters inside the networks and the training images are not quantized.

2.2 GAN

GAN was developed by [29] as a framework to train a generative model by an adversarial process. In a GAN, a discriminative network (discriminator) learns to distinguish whether a given instance is real or fake, and a generative network (generator) learns to generate realistic instances to confuse the discriminator.

Originally, the discriminator and the generator of a GAN are both multilayer perceptrons. Researchers have since proposed many variants of it. For example, DCGAN transformed multilayer perceptrons to deep convolutional networks for better performance. Specifically, the generator is composed by four deconvolutional layers. GANs with such a convolutional/deconvolutional structure have also been successfully used to synthesize plausible visual interpretations of given text [30] and to learn interpretable and disentangled representation from images in an unsupervised way [31]. Wasserstein generative adversarial networks (WGAN) [32] and least squares generative adversarial networks (LSGAN) [33] have been proposed with different loss functions to achieve more stable performance, yet they both employed the deconvolution operations too.

3 Analysis on Power of Representation

In this section, to decide whether a layer can be binarized, we analyze the power of a deconvolution layer to represent any given mapping between the input and the output, and how such power will affect the performance after binarization. We will show that the performance loss of a layer is related to the dimension of the deconvolution, and develop a metric called the degree of redundancy to indicate the loss. Finally, based on the analysis, several inferences are deduced at the end of this section, which should lead to effective and efficient binarization.

In the discussion below, we ignore batch normalization as well as activation operations and focus on the deconvolution operation in a layer, as only the weights in that operation are binarized. The deconvolution process can be transformed to equivalent matrix multiplication. Let $D^I$ ($\in \mathcal{R}^{c_i \times h_i \times w_i}$, where $c_i$, $h_i$, and $w_i$ are number of channels, height and width of the input respectively) be the input matrix, and $D^O$ ($\in \mathcal{R}^{c_o \times h_o \times w_o}$, where $c_o$, $h_o$ and $w_o$ are the number
of channels, height and width of the output respectively) be the output matrix. Denote \( \mathbf{K} \in \mathbb{R}^{c_i \times c_o \times h_k \times w_k} \), where \( h_k \) and \( w_k \) are the height and width of a kernel in the weight matrix) as the weight matrix to be deconvoluted with \( \mathbf{D}^I \). Padding is ignored in the discussion, since it will not effect the results.

For the deconvolution operation, the local regions in the output can be stretched out into columns, by which we can cast \( \mathbf{D}^O \) to \( \mathbf{D}^{Od} \in \mathbb{R}^{s_i \times r_o} \), where \( s_i = h_i w_i \), \( r_o = c_o h_k w_k \). Similarly, \( \mathbf{D}^{Id} \in \mathbb{R}^{s_i \times c_i} \) can be restructured from \( \mathbf{D}^I \), and \( \mathbf{K}^d \in \mathbb{R}^{c_i \times r_o} \) can be restructured from \( \mathbf{K} \). Please refer to [34] for details about the transform. Then, the deconvolution operation can be compactly written as

\[
\mathbf{D}^{Od} = \mathbf{D}^{Id} \ast \mathbf{K}^d,
\]

(1)

where \( \ast \) denotes matrix multiplication. \( \mathbf{D}^{Id} \) and \( \mathbf{D}^{Od} \) are the matrices containing pixels for an image or an intermediate feature map. During the training process, we adjust the values of \( \mathbf{K}^d \) to construct a desired mapping between \( \mathbf{D}^{Id} \) and \( \mathbf{D}^{Od} \).

We use \((\cdot)_j\) to denote the \( j \)-th column of a matrix. Then (1) can be decomposed column-wise as

\[
\mathbf{D}^{Od}_j = \mathbf{D}^{Id} \ast \mathbf{K}^d_j, \quad 1 \leq j \leq r_o,
\]

(2)

where \( \mathbf{K}^d_j \in \mathbb{R}^{c_i} \) and \( \mathbf{D}^{Od}_j \in \mathbb{R}^{s_i} \).

Now we analyze a mapping between an arbitrary input \( \mathbf{D}^{Id} \) and an arbitrary output \( \mathbf{D}^{Od}_j \). From [2], when the weights are continuously selected, all vectors that can be expressed by the right hand expression is a subspace \( \Omega \) spanned by the columns of \( \mathbf{D}^{Id} \), the dimension of which is \( c_i \). Here we have assumed without loss of generality that \( \mathbf{D}^{Id} \) has full column rank. When \( c_i < s_i \), which is the dimension of the output space \( \Phi \) where \( \mathbf{D}^{Od}_j \) lies, \( \Omega \) is of lower dimension than \( \Phi \), and accordingly, \( \mathbf{D}^{Od}_j \) can either be uniquely expressed as a linear combination of the columns in \( \mathbf{D}^{Id} \) if it lies in \( \Omega \) (i.e. a unique \( \mathbf{K}^d_j \) exists), or cannot be expressed if it is not (i.e. no such \( \mathbf{K}^d_j \) exists). When \( c_i = s_i \), \( \Omega \) and \( \Phi \) are equivalent, and any \( \mathbf{D}^{Od}_j \) can be uniquely expressed as a linear combination of the columns in \( \mathbf{D}^{Id} \). When \( c_i > s_i \), \( \Omega \) and \( \Phi \) are still equivalent, but any \( \mathbf{D}^{Od}_j \) can be expressed as an infinite number of different linear combinations of the columns in \( \mathbf{D}^{Id} \). In fact, the coefficients \( \mathbf{K}^d_j \) of these combinations lie in a \((c_i - s_i)\)-dimensional sub-space \( \Psi \).

The binarization imposes a constraint on the possible values of the elements in \( \mathbf{K}^d_j \). Only finite number of combinations are possible. If \( c_i \leq s_i \), then at least one of these combinations has to be proportional to the unique \( \mathbf{K}^d_j \) that yields the desired \( \mathbf{D}^{Od}_j \) to preserve performance. If \( c_i > s_i \), then one of these combinations needs to lie in the subspace \( \Psi \) to preserve performance. Apparently, the larger the dimension of \( \Psi \) is, the more likely this will happen, and the less the performance loss is. A detailed math analysis is straightforward to illustrate this, and is omitted here in the interest of space. Accordingly, we define the dimension of
\( \Psi, c_i - s_i \), as the degree of redundancy in the rest of the paper. Note that when this metric is negative, it reflects that \( \Omega \) is of lower dimension than \( \Phi \) and thus this deconvolution layer is more vulnerable to binarization errors. In general, a higher degree of redundancy should give lower binarization error.

We will use a small numerical example to partially validate the above discussion. We construct a deconvolution layer and vary its degree of redundancy by adjusting the \( c_i \) in it. For each degree of redundancy we calculate the minimum average Euclidean distance between the original output and the output produced by binarized weights, which reflects the error introduced through the binarization process. The minimum average Euclidean distance is obtained by enumerating all the possible combinations of those binary weights. The results are depicted in Fig. 2. From the figure we can see that the error decreases with the increase of degree of redundancy, which matches our conjecture.

![Fig. 2. Binarization error v.s. degree of redundancy for a deconvolution layer](image1.png)

![Fig. 3. Degree of redundancy v.s. layer number for DCGAN. The intermediate feature maps at the output of each layer as well as the final output are also presented](image2.png)

For generators in most state-of-the-art GAN models [633], we find that the degree of redundancy reduces with the increase in depth, eventually dropping below zero. Such a decrease reflects the fact that more details are generated at the output of a layer as its depth grows, as can also be seen in Fig. 3. These details are highly correlated, and reduce the subspace needed to cover them.

Based on our analysis, several inferences can be deduced to guide the binarization:

- With the degree of redundancy, taking advantage of existing binarization methods becomes reasonable and feasible. Binarizing layers with higher degree of redundancy will lead to lower performance loss after binarization, while layers with negative degree of redundancy should be kept un-binarized to avoid excessive performance loss.
- According to the chain rule of probability in directed graphs, the output of every layer is only dependent on its direct input. Therefore, the binarizability of each layer can be superposed. If a layer can be binarized alone, it can be binarized with other such layers.
When binarizing several deconvolution layers together, the layer with the least degree of redundancy may be the bottleneck of the generator’s performance.

As a result, only shallower layer(s) of a generator can be binarized together to preserve its performance, because of the degree of redundancy trend in it. This leads to PBGen. Besides, such analysis may also explain why binarization can be applied in almost all convolution layers: distilling local information from a global map leads to positive degree of redundancy.

**Fig. 4.** Structure of the generators in DCGAN with dimension of each layer labeled. Deconvolutional layers are denoted as “CONV” (figure credit: [6])

### 4 Experiments

#### 4.1 DCGAN and Different Binarization Settings

DCGAN will serve as a vehicle to verify the inferences deducted from the theoretical analysis in Section 3. We will explore how to best binarize it with preserved performance. Specifically, we use the TensorFlow [35] implementation of DCGAN on GitHub [36]. The structure of its generator is illustrated in Fig. 4. The computed degree of redundancy in each layer in the generator is shown in Fig. 3 and qualitatively summarized in Table 1. The degree of redundancy in the last layer drops to -896. According to the inferences before, we can expect that since the degree of redundancy decreases as the depth increases, binarizing shallower layers and keeping the deeper layers in the format of floating point will help preserve the performance.

**Table 1.** Degree of redundancy in each deconvolution layer of the generator in DCGAN

| Layer number | Label in Fig. 4 | Degree of redundancy |
|--------------|-----------------|----------------------|
| 1            | CONV 1          | 1008                 |
| 2            | CONV 2          | 448                  |
| 3            | CONV 3          | 0                    |
| 4            | CONV 4          | -896                 |

Since there are four deconvolution layers in total in the generator, each layer can be either binarized or not. For verification, we have conducted experiments on all $2^4 = 16$ different settings, but only the eight representative ones are discussed for clarity and space, and others will lead us to the same conclusion.
Those eight different representative settings are summarized in Table 2 for clear-
ness. In this table, the “Setting” column labels each setting. “Layer(s) binarized”
indicates which layer(s) are binarized in the generator. The “Discriminator bi-
larized” column tells whether the discriminator is binarized or not. “Y” means
yes, while “N” means no. This column is introduced to verify an observation in
our experiments to be discussed later.

| Setting | Layer(s) binarized | Discriminator binarized |
|---------|--------------------|------------------------|
| A       | None               | N                      |
| B       | 1                  | N                      |
| C       | 2                  | N                      |
| D       | 3                  | N                      |
| E       | 4                  | N                      |
| F       | 1,2,3              | N                      |
| G       | 1,2,3,4            | N                      |
| H       | 1,2,3              | Y                      |

Setting G will serve as the baseline model for performance after binarization,
because it adopts the compression techniques based on CNNs directly without
considering the degree of redundancy. The binarization method we adopt
is Binary-Weight-Network from [20], which is one of the state-of-the-art com-
pression techniques [37]. On the other hand, Setting A serves as the baseline
model when considering the memory saving, speedup as well as performance dif-
ference before and after binarization, because it represents the original DCGAN
in floating point representation. It is considered as one common GAN structure
providing good performance.

4.2 Dataset and Metrics

CelebA [38] is used as the dataset for our experiments, because it is a popular
and verified dataset for different GAN structures. DCGAN, WGAN, LSGAN
and many other GAN structures are tested on it [39]. As every image in CelebA
contains only one face, it is much easier to tell the quality of the generated
images.

Traditionally the quality of the generated images is identified by observation.
However, qualitatively evaluation is always a hard problem. According to the in-
depth analysis of commonly used criteria in [40], good performance in a single or
extrapolated metric from average log-likelihood, Parzen window estimates, and
visual fidelity of samples does not directly translate to good performance of a
GAN. On the other hand, the log-likelihood score proposed in [41] only estimates
a lower bound instead of the actual performance.

Very recently, [7] proposed an efficient metric, which we will use in our experi-
ments, and showed that it is superior to MS-SSIM [42], which is a commonly used
metric. It calculates the sliced Wasserstein distance (SWD) between the training samples and the generated images under different resolutions. In particular, the SWD from the lower resolution patches indicates similarity in holistic image structures, while the finer-level patches encode information about pixel-level attributes. In this work, the max resolution is 64×64. Thus, according to [7], we will use three different resolutions to evaluate the performance: 16×16, 32×32, and 64×64. For all different resolutions, small SWD indicates that the distributions of the patches are similar, which means that a generator with smaller SWD is expected to produce images more similar to the images from the training samples in both appearance and variation.

4.3 Experimental Results

In this section, we will present experimental results that verify our inferences in Section 3, along with some additional observations about the competition between the generator and the discriminator. The images generated by the original GAN (setting A), in which all weights of each deconvolution layer are in the form of floating point, are displayed in Fig. 5(a). The images generated by the binarized DCGAN without considering the degree of redundancy are displayed in Fig. 5(g). These are our two baseline models.

Qualitative Comparison of Single-Layer Binarization

We start our experiments by comparing the images generated by binarizing a single layer in the generator of DCGAN. The results are shown in Figures 5(b) - 5(e), which are
generated by PBGen’s under setting B - setting E respectively. In other words, those PBGen’s utilize binary weights to the first, the second, the third, and the last deconvolution layer respectively. The degree of redundancy of each layer is shown in Fig. 3. From the generated figures we can then see that Fig. 5(b) generates the highest quality of images, similar to the original ones in Fig. 5(a). Images in Fig. 5(c) are slightly inferior to those in Fig. 5(b) but better than those in Fig. 5(d). Fig. 5(e) has no meaningful images at all. These observations are in accordance with our inferences in Section 3: the performance loss when binarizing a layer is decided by its degree of redundancy, and a layer with negative degree of redundancy should not be binarized.

**Quantitative Comparison of Single-Layer Binarization** We further quantitatively compute the SWD values with 16×16, 32×32 and 64×64 resolutions for the four different settings B, C, D, and E. Their relationship with the degree of redundancy of the binarized layer is plotted in Fig. 6. From the figure, two things are clear: first, regardless of resolution, a negative degree of redundancy (setting E) results in a more than 5× increase in SWD compared with other settings with non-negative degree of redundancy (settings B, C, and D). Second, for all the three resolutions, SWD decreases almost linearly with the increase of the degree of redundancy when it is non-negative. This confirms that our degree of redundancy can capture the impact of binarization not only on the holistic structure but also on the pixel-level fine details, and as such, is indeed a good indicator to quickly judge whether a layer can be binarized.

We also report the SWD averaged over different resolutions (16×16, 32×32, 64×64) in Table 3 where the result for the original GAN (setting A) is also reported. From the table we can draw similar conclusions, that binarizing second layer (setting C) increases the average SWD by 2.3% compared with the original GAN (setting A), while binarizing third and fourth layer (settings D and E) further increases it by 52.3% and 913.6%, respectively.
Table 3. Average SWD under different settings

| Setting | A | B | C | D | E |
|---------|---|---|---|---|---|
| Average SWD ($\times 10^{-3}$) | 44 | 38 | 45 | 67 | 449 |

It is interesting to note that the average SWD achieved by binarizing the first layer (setting B) is 13.6% smaller than that from the original DCGAN (setting A). To further check this, we plot the SWD v.s. resolution for these two settings in Fig. 7. From the figure we can see that the SWD from setting B is always smaller than that from setting A across all three resolutions. This shows that setting B can achieve better similarity as well as detailed attributes. Such an improvement is probably due to the regularization effect, and similar effect has been observed in the compression of CNNs [25].

Validation of Superposition of Binarizability We now explore experiments to verify our inference that all layers that can be binarized alone can be binarized together. The images generated by setting F in Fig. 5(f), where the first three layers in the generator are binarized together, show no significant difference from those in Figures 5(a) 5(d). Binarizing any two layers from the first three layers (not shown here) will lead to the same result. On the other hand, setting G does not generate any meaningful output (Fig. 5(g)), as the last layer, which cannot be binarized alone, is binarized together with the first three layers. Binarizing any of the first three layers as well as the last layer (not shown here) will produce meaningless results too. Setting G follows the state-of-the-art binarization for CNNs directly without considering the degree of redundancy. That is, with the assistance of the degree of redundancy, we can figure out that at most the first three deconvolution layers can be binarized with small loss on performance in the generator (Setting F). Nevertheless, directly adopting the existing binarization method will lead to excess degradation in performance and cannot provide any hint to improve (Setting G).

Moreover, the average SWD for setting F is 0.067, the same as setting D. Further looking at the SWD values under different resolutions for the two different settings as shown in Fig. 8, it is clear that the two curves are very close. This validates our last inference, that when multiple layers are binarized together, the

![Fig. 8. SWD v.s. resolutions under Setting D and Setting F](image-url)
layer with least degree of redundancy is the bottleneck, which decides the overall performance of the network.

**Compression Saving** Table 4 summarizes the speedup during training and inference as well as the memory reduction for PBGen compared with the original generator in DCGAN, which is the baseline model when considering the memory saving and speedup. PBGen under Setting F can achieve $25.81\times$ memory saving as well as $1.96\times$ and $1.32\times$ speedup during inference and training respectively with little performance loss. For both the original generator and PBGen, during the training process the floating point representation of all weights need to be used for backward propagation and update [20]. As such, the speedup mainly comes from faster forward propagation with binarized weights. Compression savings are also shown in Table 4. They are practical savings calculated according to our strategy.

**Table 4.** Training and inference speedup as well as memory reduction for PBGen

| Generator model                | Speedup Inference | Speedup Training | Memory cost  |
|-------------------------------|-------------------|------------------|--------------|
| Original generator from DCGAN (Setting A) | 1.0× | 1.0× | 1.0× |
| PBGen (Setting F)             | 1.96× | 1.32× | $1/25.81\times$ |

**Unbalanced Competition** So far, our discussion has focused on the binarization of the generator in a GAN only, as the discriminator takes the same form as conventional CNNs. However, since competition between generator and discriminator is the key of GANs, would a binarized generator still compete well with a full discriminator? The loss values for the discriminator network and PBGen under Setting F are depicted in Fig. 9 where x-axis indicates the number of epochs and y-axis is the loss value. The images generated from different number of epochs are also exhibited aside. From the figure we can see that during the initial stage, distorted faces are generated. As the competition is initiated, image quality improves. But very quickly, the competition vanishes, and the generated images stop improving. However, when we binarize the discriminator at the same time (Setting H), the competition continues to improve image quality, as can be seen in Fig. 10.

We further plot the loss values of the discriminator and the generator of the original DCGAN (Setting A), and the results are shown in Fig. 11. It is very similar to Fig. 10 except that the competition is initiated earlier, which is due to the stronger representation power of both the generator and the discriminator before binarization. These figures confirm that the quick disappearance of competition is mainly due to the unbalanced generator and discriminator, which should be avoided.
Fig. 9. Loss values of the original discriminator and PBGen under Setting F along epochs.

Fig. 10. Loss values of binarized discriminator and PBGen under Setting H along epochs.

Fig. 11. Loss values of the discriminator and the generator in original DCGAN under Setting A along epochs.
We now explore the quality of the images generated from balanced competition using Setting H. The images generated are shown in Fig. 5(h), the quality of which is apparently better than the rest in Fig. 5. To further confirm this quantitatively, we compute the average SWD values of those images, which is 0.034 in average. This is even smaller than any average SWD values listed in Table 3, which shows that the images are of better quality, even compared with the original DCGAN.

Summary To summarize the discussion and comparisons in this section, we plot the SWD v.s. resolution curves for all the 8 settings in Fig. 12. It allows a complete view of how these different settings compare in terms of similarity as a whole and fine details. From the figure we can see that setting H gives the best similarity as a whole, while setting C yields the finest detailed attributes.

![Fig. 12. SWD v.s. resolutions under all different settings](image)

Consequently, utilizing the degree of redundancy as a tool, we can efficiently find out eligible layers that can be binarized and based on their superposition, a final binarization strategy can be decided. It cannot guarantee an optimal result but does decrease the search space for the final solution from $O(2^n)$ to $O(n)$ or less, where $n$ is the number of layers, because testing on all combinations of binarization strategy is not necessary and we only need to binarize every single layer with high degree of redundancy to decide the final strategy. Since our theoretical analysis and experiments are based on deconvolutional layers, we believe this method can work for other deconvolution based generators beyond DCGAN.

5 Conclusion

Compression techniques have been widely studied for convolutional neural networks, but directly adopting them to all layers will fail deconvolution-based generator in generative adversarial networks based on our observation. We propose and validate that the performance of deconvolution-based generator can be preserved when applying binarization to carefully selected layers (PBGen). To
accelerate the process deciding whether a layer can be binarized or not, the degree of redundancy is proposed based on theoretical analysis and further verified by experiments. Under the guide of this metric, search space for optimal binarization strategy is decreased from $O(2^n)$ to $O(n)$ where $n$ is the number of layers in the generator. PBGen for DCGAN can yield up to $25.81\times$ saving in memory consumption with $1.96\times$ and $1.32\times$ speedup in inference and training respectively with little performance loss measured by sliced Wasserstein distance score. Besides, we also demonstrate that both generator and discriminator should be binarized at the same time for a balanced competition and better performance.

References

1. Finn, C., Goodfellow, I., Levine, S.: Unsupervised learning for physical interaction through video prediction. In: Advances in Neural Information Processing Systems. (2016) 64–72
2. Pfau, D., Vinyals, O.: Connecting generative adversarial networks and actor-critic methods. arXiv preprint arXiv:1610.01945 (2016)
3. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems. (2016) 2234–2242
4. Goodfellow, I.: Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160 (2016)
5. Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al.: Photo-realistic single image super-resolution using a generative adversarial network. arXiv preprint arXiv:1609.04802 (2016)
6. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
7. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
8. Inc., A.: Apple Inc. (2017)
9. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: International Conference on Machine Learning. (2015) 2285–2294
10. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
11. Li, F., Zhang, B., Liu, B.: Ternary weight networks. arXiv preprint arXiv:1605.04711 (2016)
12. Zhang, S., Du, Z., Zhang, L., Lan, H., Liu, S., Li, L., Guo, Q., Chen, T., Chen, Y.: Cambricon-x: An accelerator for sparse neural networks. In: Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on, IEEE (2016) 1–12
13. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: Training neural networks with low precision weights and activations. arXiv preprint arXiv:1609.07061 (2016)
14. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. (2015) 1135–1143

15. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. arXiv preprint arXiv:1704.04760 (2017)

16. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)

17. Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R.: Deconvolutional networks. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE (2010) 2528–2535

18. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866 (2014)

19. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)

20. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision, Springer (2016) 525–542

21. Judd, P., Albericio, J., Hetherington, T., Aamodt, T.M., Moshovos, A.: Stripes: Bit-serial deep neural network computing. In: Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on, IEEE (2016) 1–12

22. Miyashita, D., Lee, E.H., Murmann, B.: Convolutional neural networks using logarithmic data representation. arXiv preprint arXiv:1603.01025 (2016)

23. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. arXiv preprint arXiv:1612.01064 (2016)

24. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: Towards lossless cnns with low-precision weights. arXiv preprint arXiv:1702.03044 (2017)

25. Cai, Z., He, X., Sun, J., Vasconcelos, N.: Deep learning with low precision by half-wave gaussian quantization. arXiv preprint arXiv:1702.00953 (2017)

26. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems. (2015) 3123–3131

27. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)

28. Song, J.: Binary generative adversarial networks for image retrieval. arXiv preprint arXiv:1708.04150 (2017)

29. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. (2014) 2672–2680

30. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396 (2016)

31. Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P.: Info-gan: Interpretable representation learning by information maximizing generative adversarial nets. In: Advances in Neural Information Processing Systems. (2016) 2172–2180

32. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: International Conference on Machine Learning. (2017) 214–223
33. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Smolley, S.P.: Least squares generative adversarial networks. arXiv preprint ArXiv:1611.04076 (2016)
34. cs231n Course Materials: Implementation as Matrix Multiplication (2017)
35. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
36. Kim, T.: DCGAN-tensorflow (2017)
37. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.: Efficient processing of deep neural networks: A tutorial and survey. arXiv preprint arXiv:1703.09039 (2017)
38. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of International Conference on Computer Vision (ICCV). (2015)
39. Cha, J.: tf.gans-comparision (2017)
40. Theis, L., van den Oord, A., Bethge, M.: Anote on the evaluation of generative models. stat 1050 (2016) 24
41. Wu, Y., Burda, Y., Salakhutdinov, R., Grosse, R.: On the quantitative analysis of decoder-based generative models. arXiv preprint arXiv:1611.04273 (2016)
42. Odena, A., Olah, C., Shlens, J.: Conditional image synthesis with auxiliary classifier gans. arXiv preprint arXiv:1610.09585 (2016)