This WACV 2022 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

PPCD-GAN: Progressive Pruning and Class-Aware Distillation for Large-Scale Conditional GANs Compression

Duc Minh Vo[1] vmduc@nlab.ci.i.u-tokyo.ac.jp
Akihiro Sugimoto[2] sugimoto@nii.ac.jp
Hideki Nakayama[1] nakayama@ci.i.u-tokyo.ac.jp

[1] The University of Tokyo, Japan
[2] National Institute of Informatics, Japan

Abstract

*We push forward neural network compression research by exploiting a novel challenging task of large-scale conditional generative adversarial networks (GANs) compression. To this end, we propose a gradually shrinking GAN (**PPCD-GAN**) by introducing progressive pruning residual block (PP-Res) and class-aware distillation. The PP-Res is an extension of the conventional residual block where each convolutional layer is followed by a learnable mask layer to progressively prune network parameters as training proceeds. The class-aware distillation, on the other hand, enhances the stability of training by transferring immense knowledge from a well-trained teacher model through instructive attention maps. We train the pruning and distillation processes simultaneously on a well-known GAN architecture in an end-to-end manner. After training, all redundant parameters as well as the mask layers are discarded, yielding a lighter network while retaining the performance. We comprehensively illustrate, on ImageNet* 128×128 *dataset, PPCD-GAN reduces up to 5.2*$\times$ *(81%) parameters against state-of-the-arts while keeping better performance.*

1. Introduction

Scaling-up network parameters in GANs [1, 2, 3] has brought breakthroughs in large-scale conditional image generation. Nevertheless, such scaled-up models [1, 2, 3] often require enormously expensive computational cost and memory, limiting their adoption to usage under real scenarios such as mobile devices. In this paper, we aim to reduce the size of a conditional GAN model in terms of the number of parameters while attaining performance comparably to SOTAs [1, 2, 3]. We state this novel problem as *large-scale conditional GANs compression*, which can be regarded as a new branch of neural network compression research [4]. Different from image-to-image GANs compression [5, 6], our targeting class-conditional GANs have to deal with hundreds of classes at the same time. This novel problem undoubtedly bridges the gap between academia and industry, yet it has not been well explored in the literature.

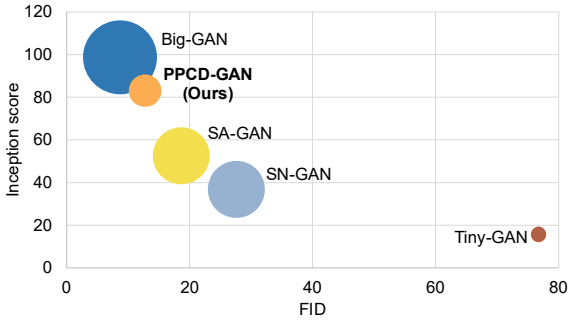

Figure 1: Overall comparison on the number of parameters, Inception score (IS), and FID against Big-GAN [1], SA-GAN [2], SN-GAN [3], and Tiny-GAN [7]. PPCD-GAN is smaller than the others while achieving comparable IS and FID. The size of each circle reflects the number of parameters.

Neural network compression is long-standing research and most of its active works [4, 8, 9, 10, 11, 12, 13] rely on the two-step approach where the pruning step is followed by the re-training step. The former step often directly removes network weights by employing pre-set criteria such as norm regularization [4, 13] or weight magnitude [8, 10], producing a compressed model. The latter one aims at restoring the performance of the compressed model with helps from fine-tuning or knowledge distillation [14, 15]. Although showing great potential in recognition and classification tasks, previous works cannot be straightforwardly extended to GANs compression [5, 16] due to two inevitable reasons. On one hand, the GANs model structure requires more parameters and more complicated computation than those of the typical convolutional neural networks (CNNs) model. As a result, the GANs model is vulnerable to pre-set criteria, leading to drastically dropping its performance as addressed in [5, 16]. On the other hand, training GANs is notably unstable, and thus re-training the compressed model brings new challenges. The above facts reveal difficulties in

GANs compression.

Very recently, Fu et al. [5] and Li et al. [6] proposed a method for image-to-image GANs compression where neural architecture search (NAS) [17] on a trained GAN is used. Li et al. [6] incorporated knowledge distillation [14] to stablize the training. The usage of NAS, however, requires a high computational cost due to a large search space. Moreover, using NAS independently of the training process may break the underlying distribution of GANs because NAS itself does not know the distribution of a trained model. To overcome these issues, the pruning and training processes are required to perform simultaneously.

Inspired by the above observations, we approach to our problem by unifying the pruning and distillation processes. To this end, we introduce progressive pruning residual block (PP-Res) and class-aware distillation. The PP-Res is mostly identical to the conventional residual block [18] except for that we add a learnable mask layer with its own parameters on top of each convolutional layer, leading to gradual removal of redundant weights as training proceeds. All unnecessary weights and masks are safely removed after training without any risk. The class-aware distillation, on the other hand, acquires knowledge from a pre-trained model (i.e., Big-GAN [1]) through attention maps [15], resulting in stability of training. As Big-GAN [1] suffers from class leakage, we employ the class conditional normalization before computing the attention maps. We simultaneously train the pruning (i.e., PP-Res) and distillation (i.e., class-aware distillation) processes in the end-to-end manner. Our contributions are two-fold:

- We address a novel problem of large-scale conditional GANs compression. While a compact model for conditional GANs was very recently proposed [7], it manually changes the architecture of Big-GAN [1] significantly, so that it successfully works on only homogeneous classes of a dataset (i.e., ImageNet [19] 398 animal classes). In contrast, to best of our knowledge, PPCD-GAN is the first GANs compression framework that fully explores a large-scale dataset.
- Our proposed *PPCD-GAN* unifies pruning and distillation for the first time on the large-scale conditional GANs compression problem.

Comprehensive experiments on ImageNet 128×128 dataset [19] show the efficacy of PPCD-GAN against SOTAs (see Fig. 1 for comparison).

2. Related Work

Large-scale conditional image generation. Only a few methods [3, 2, 1] successfully handled this problem. Miyato et al. [3] used the label projection layer (i.e., SN-GAN) in the discriminator for better real/fake classification. Zhang et al. [2] incorporated a self-attention mechanism into GANs (i.e., SA-GAN), which allows the details of the image are generated by using cues from all feature locations. Brock et al. [1] introduced Big-GAN, which is built upon SA-GAN [2] by scaling up parameters and the batch size in training. These methods require enormously expensive computational cost and memory. For instance, SN-GAN [3] and SA-GAN [2] models need $\sim$40M parameters whereas Big-GAN [1] requires even more ($\sim$70M).

Very recently, Chang et al. [7] proposed Tiny-GAN with less parameters than [3, 2, 1]. Tiny-GAN is a manually designed network based on the Big-GAN [1] architecture and replaces the standard convolutional layer with depth-wise separable convolution. Moreover, it employs the generated images obtained by pre-trained Big-GAN [1] as the real images in training, which thus allows to distill black-box knowledge from Big-GAN [1]. Despite such efforts, Tiny-GAN works properly only on a small sub-set of the large-scale ImageNet. Different from Tiny-GAN [7], our method directly removes redundant parameters without significantly changing a well-known architecture. Furthermore, we distill instructive knowledge from Big-GAN [1] through the attention maps, resulting in successfully handling the ImageNet in full-scale.

CNNs and GANs compression. CNNs compression consists of three types [20]: matrix decomposition, network quantization, and pruning. Among them, the pruning approach actively receives more attentions [4, 9, 10, 11, 12, 13]. The current mainstream of pruning methods focuses on connection pruning [4, 10] and structure pruning [9, 10, 11, 12, 13]. The connection pruning methods [4, 10] eliminate unnecessary connections using heuristics or optimization processes, which heavily depends on the hardware specification [20]. The structure pruning methods [9, 11, 12, 13] are, on the other hand, hardware-friendly and aim at reducing unnecessary channels or layers. For instance, Li et al. [13] employ ℓ_1 norm to remove redundant channels and then fine-tune the pruned network to recover its performance. Our method belongs to the structure pruning type where we remove superfluous channels. However, we tackle the GANs compression problem where the network is more vulnerable than the typical CNNs [16, 5].

Some image-to-image translation GANs compression methods [16, 5, 6] have been proposed recently. Shu et al. [16] proposed a co-evolution to regularize pruning on CycleGAN. However, their method cannot be extended to other GANs architecture since it is bound to cycle-consistency loss and pre-trained discriminator. Fu et al. [5] and Li et al. [6] concurrently employed NAS [17] in GANs compression. Li et al. [6] further incorporated knowledge distillation [14] for training stability. However, NAS is not effective in a broad search space and using NAS in pruning separated from training may lead to the corruption of a

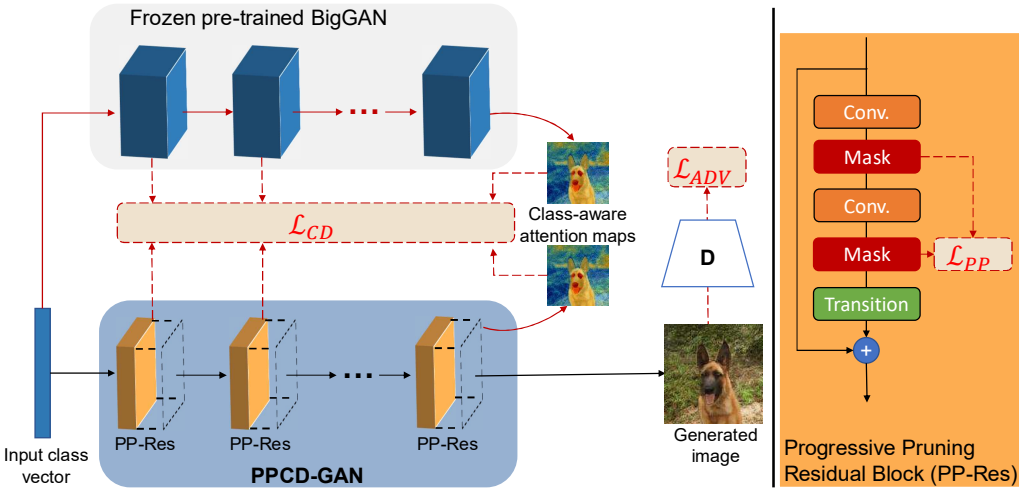

Figure 2: Overall framework of our proposed method (left) and progressive pruning residual block (right). For simplicity, we mainly illustrate important layers in the progressive pruning residual block. The red parts are removed after training.

pre-trained generator's underlying distribution.

Very recently, Kang et al. [20] proposed to train a differentiable mask for each output channel (in a convolutional layer) to softly discard unnecessary channels without extra re-training. Their differentiable mask is estimated based on the batch normalization and ReLU layers. Inspired by [20], we also employ a learnable mask to remove useless channels gradually. Yet, our mask has its own parameters, holding the advantage of the class conditional normalization.

Knowledge distillation. This is widely used to transfer knowledge from a well-trained teacher model to a (compact) student model in order to enhance the performance of the student model [14, 15]. Existing methods [6, 7] showed the advantages of knowledge distillation in GANs compression. We thus employ knowledge distillation for better training our network. Different from the methods [6, 7], we follow Zaroruyko et al. [15] to use attention maps as instructive guidance. This is because together with the class conditional normalization, the attention maps enable us to learn useful class-aware information from the teacher model.

3. Proposed PPCD-GAN

3.1. Design of PPCD-GAN

We designed PPCD-GAN in the context of unification of pruning and distillation. We propose a tailored framework for efficient GANs, inspired by the student—teacher framework [14], because neither NAS [5, 6] nor significant changes in a network architecture [7] are applicable in compressing GANs. Unlike [14], we aim to gradually prune network parameters as training proceeds while stabilizing the training with help of knowledge on distillation. Therefore, our framework requires two conditions: (i) a robust pre-trained teacher model, and (ii) the architecture of the teacher model roughly agrees with that of the student model. We remark that both the models should have the capability of handling large-scale conditional image generation. Here, as an example, we choose SA-GAN [2] as our student model whereas Big-GAN [1] as the teacher model because Big-GAN [1] is an extension of SA-GAN [2] and achieves leader-board IS and FID. Note that our framework can be applied to other GANs architectures as long as the above requirements are satisfied.

Residual blocks [18] are essential ingredients of SA-GAN [2] and their parameters are a tremendous burden to the network parameters. Accordingly, we pay our attention to reducing the parameters of the residual blocks, pruning the output channels (i.e., the number of convolutional filters) of convolutional layers. Needless to say, directly cutting off several output channels is not effective [5, 6]. We thus follow the idea proposed in [20] to progressively prune convolutional layers along with training. In our case, because one residual block is connected to its previous block and/or the next one, pruning a residual block may cause channel inconsistency among different residual blocks. To resolve this inconsistency, we introduce a transition layer on top of the residual block. The transition layer ensures the channel consistency and efficiently works to avoid an explosion of the network parameters in general.

Since training GANs is notably unstable, we distill immense knowledge from a teacher model to settle down the training process. We thus introduce the class-aware distillation to better transfer prior knowledge from Big-GAN to our network. Fig. 2 (left) shows the PPCD-GAN framework.

Like SA-GAN [2], the generator of PPCD-GAN receives a pair of a noise vector and a class condition vector as its inputs. The discriminator, on the other hand, receives a pair of an image and its corresponding class condition vector

where the image can be either real or fake.

3.2. Progressive pruning residual block

Our progressive pruning residual block is based on the convention residual block [18], but has two distinctions. First, the *learnable mask layer* enables the network to gradually remove channels during training while retaining the benefit of class conditional normalization (the key idea of conditional GANs). Second, the *transition layer* maintains the channel consistency across multiple residual blocks.

Learnable mask layer. Inspired by [20], we place our learnable mask layer before the ReLU layer because negative activations are zeroed out by the ReLU layer. This realizes network parameters removal with low risk. Although [20] estimates the mask value based on the parameters of the batch normalization layer using a pre-defined constant, we build a mask layer with its own parameters. This is because our model employs the conditional batch normalization where the class condition vector (obtained through an embeddings layer) is used to distinguish various classes. We enforce the parameters of the batch normalization layer towards a pre-defined constant, so that the condition vectors fall inside a constant range. As a result, the model may ignore the contribution of the class condition. We, in what follows, introduce the design of the learnable mask layer and its loss term.

Let n be output channels (i.e., the number of filters) of a convolutional layer. We devise a mask $\mathbf{m} = \{m_1, m_2, ..., m_n\}$ ($m_i \in [0, 1]$) from its learnable parameters $\mathbf{W} = \{w_1, w_2, ..., w_n\}$. Each value m_i is estimated based on its corresponding parameter w_i. To jointly optimize the mask layer and other layers through a simple gradient-based optimization, we employ a popular differentiable function (i.e., *sigmoid*) with a constant δ for continuous relaxation as:

$$m_i = \frac{1}{1 + \exp(-\delta w_i)}. \tag{1}$$

Like [20], we impose sparse regularization ℓ_{PP} in training the mask layer to progressively enforce its each value towards zero:

$$\ell_{\text{PP}} = \sum_{i=1}^{n} |w_i + 1|. \tag{2}$$

Our learnable mask is multiplied with the outputs of its corresponding convolutional layer. As training proceeds, the values of the mask gradually become zeros, meaning that the gradient to its corresponding convolutional layer decreases to zeros. Consequently, the mask hides redundant weights (in a convolutional layer) during training. Therefore, we can safely remove unnecessary weights when testing without any risk. Note that m_i is expected to become 0 or 1 after training, but the training fact shows that the expectation is not always satisfied.

We employ a "binarization trick" to enforce m_i to be 0 or 1. Letting c be the number of elements of $\{m_1, m_2, ..., m_n\}$ whose value is less than or equal to a pivot (we choose 0.005 in our experiments), we define a $ratio = \frac{c}{n}$. We note that the above pivot can be any sufficiently small value between 0 and 1 which is independent of training settings. If $ratio$ is greater than a (given) compression ratio threshold α, we set all m_i's greater than 0.005 to 1, otherwise 0. We then stop update the mask. In this way, our learnable mask fills the gap between the continuous space in training and the discrete space in testing. Our "binarization trick" can be summarized as follows:

$$m_i^* = \begin{cases} m_i & \text{if } ratio \leq \alpha \\ 1 & \text{if } m_i > 0.005 \text{ and } ratio > \alpha \\ 0 & \text{if } m_i \leq 0.005 \text{ and } ratio > \alpha \end{cases}. \tag{3}$$

Transition layer. When we prune a residual block, channel inconsistency among different residual blocks inevitably happens because two consecutive blocks are connected through residual connection [18]. We thus propose a simple yet effective solution by adding the 1×1 convolution layer, called transition layer, before executing residual connection. The usage of the transition layer is insightful because of two reasons. First, it does not significantly affect the feature maps from the previous layer since the 1×1 convolution layer is widely used to increase/decrease output channels. Second, its number of parameters is small enough not to cause the explosion of the network parameters.

Leveraging the learnable mask layer and the transition layer, we propose a progressive pruning residual block (PP-Res) to replace the commonly used residual block in SA-GAN [2]. More precisely, we incorporate the learnable mask layer on top of each convolutional layer and the transition layer on top of each residual block. In the end, each residual block consists of two mask layers and one transition layer (Fig. 2 right). Extending the learnable mask layer and the transition layer to multiple residual blocks is straightforward. It is worth noting that our PP-Res does not significantly change the architecture of SA-GAN [2].

3.3. Class-aware distillation

To stabilize the training process, we transfer immense knowledge from a teacher model to our PPCD-GAN through the distillation process [14]. Though our teacher model (i.e., Big-GAN [1]) is powerful, we potentially have two issues to solve. First, since the number of the output channels of Big-GAN is much larger than that of PPCD-GAN, we may fail in directly computing the norm distance (e.g., ℓ_1). Second, Big-GAN (sometimes) suffers from class leakage, which may affect the distilled knowledge. A simple way is to adopt the learnable remapping [6]. That solution, however, is not good in our case because it computes

loss on a set of consecutive feature maps under the assumption on the set being a portion of the compressed (pruned) model. We thus follow Zaroruyko et al. [15] to use attention maps. This is because the attention maps contain more instructive information. More precisely, we extract two attention maps from the output features of the residual block (in Big-GAN [1]) and the output features of PP-Res (in our model). Next, we compute the distance between the two attention maps. To avoid class leakage, we apply the class conditional normalization on output features of Big-GAN before computing the attention maps. Our class-aware distillation is rational because summing along all feature maps ignores the contributions of removed channels, and gives, in return, more informative feedback to remaining ones.

Let $\mathbf{O}^{\mathrm{T}}$ and $\mathbf{O}^{\mathrm{S}}$ be the (normalized) features of Big-GAN with the size of $C^{\mathrm{T}} \times H \times W$ and the features of PPCD-GAN with the size of $C^{\mathrm{S}} \times H \times W$ (C^{S} is smaller than C^{T}). Following [15], we compute the attention map of Big-GAN by $F^{\mathrm{T}} = \sum_{i=1}^{C^{\mathrm{T}}} |\mathbf{O}^{\mathrm{T}}(i,:,:)|^2$ (using Python notation) and that of PPCD-GAN by $F^{\mathrm{S}} = \sum_{i=1}^{C^{\mathrm{S}}} |\mathbf{O}^{\mathrm{S}}(i,:,:)|^2$. We then formulate our class-aware distillation as:

$$\ell_{\mathrm{CD}} = \left\| \frac{F^{\mathrm{T}}}{\|F^{\mathrm{T}}\|_2} - \frac{F^{\mathrm{S}}}{\|F^{\mathrm{S}}\|_2} \right\|_2 . \tag{4}$$

3.4. Loss function

PPCD-GAN replaces all residual blocks of SA-GAN [2] with PP-Res(es). Therefore, Eq. (2) is naturally aggregated to the progressive pruning loss:

$$\mathcal{L}_{\mathrm{PP}} = \frac{1}{2N} \sum_{k=1}^{2N} \ell_{\mathrm{PP}_k}, \tag{5}$$

where N is the number of PP-Res(es). N is set to 5 in experiments.

We execute class-aware distillation (Eq. (4)) at some blocks. We thus define the class-aware distillation loss as:

$$\mathcal{L}_{\mathrm{CD}} = \frac{1}{L} \sum_{k=1}^{L} \ell_{\mathrm{CD}_k}, \tag{6}$$

where L is the number of (residual) blocks to which we execute class-aware distillation. In experiments, we set $L = 4$ because we execute the distillation from the second to the fifth (residual) blocks.

We also employ adversarial loss:

$$\begin{aligned} \mathcal{L}_{\mathrm{ADV}} = \min_{G^{\mathrm{S}}} \max_{D^{\mathrm{S}}} & \mathbb{E}_{(I_{\mathrm{real}}, cls) \sim p_{\mathrm{data}}} \left[\log(D^{\mathrm{S}}(I_{\mathrm{real}}, cls)) \right] \\ & + \mathbb{E}_{cls \sim p_{\mathrm{data}}, z \sim p_z} \left[\log(1 - D^{\mathrm{S}}(G^{\mathrm{S}}(z, cls), cls)) \right], \end{aligned} \tag{7}$$

where G^{S} and D^{S} are the generator and discriminator; I_{real} is a real image and cls is the class condition sampled from all the training dataset p_{data}. Similarly, z is the noise vector sampled from Gaussian distribution p_z. $\mathbb{E}(\cdot)$ denotes expectation over either p_{data} or p_z.

The total loss is defined as:

$$\mathcal{L} = 0.01 \times \mathcal{L}_{\mathrm{PP}} + \mathcal{L}_{\mathrm{CD}} + \mathcal{L}_{\mathrm{ADV}}. \tag{8}$$

We set an empirically determined small hyperparameter for $\mathcal{L}_{\mathrm{PP}}$ to prevent quick convergence of mask layers. We minimize Eq. (8) in the end-to-end manner. The details of training procedure can be found in the supplementary.

4. Experiments

4.1. Compared methods and evaluation metrics

Dataset and compared methods. We principally evaluated PPCD-GAN on a challenging large-scale ImageNet dataset [19]. We note that although CIFAR and/or JFT-300 might be considered to be evaluated, CIFAR is insufficient in scale (100 classes with size of 32×32) and JFT-300 is an undisclosed and internal dataset of Google.

We employ SA-GAN [2] as the baseline. We compare PPCD-GAN with Big-GAN [1], SN-GAN [3], and Tiny-GAN [7]. For Big-GAN [1], we employ pre-trained model released by the authors [21]. We train SA-GAN [2] on the re-implementation provided by Big-GAN's authors [21] since it produces better images. For SN-GAN [3] and Tiny-GAN [7], we trained authors' provided codes (SN-GAN [22], Tiny-GAN [23]) on ImageNet. All networks are trained with their original settings.

Taking into account the ability of the compared models and our computing resource, all the models are designed to generate images with the size of $3 \times 128 \times 128$.

Evaluation metrics. We evaluate the overall quality of generated images using Inception score (IS) [24], Fréchet Inception Distance (FID) [25], and diversity score (LPIPS) [26] (implemented in [27, 28, 29]). We also report the number of parameters (#Par.) and MACs [30] to intuitively compare the compression rates. We compute #Par. and MACs using torchprofile tool [31]. To compare practical speed of all compared methods, we measure the wall clock generating time on GPU (NVIDIA GTX3090) and CPU (AMD Ryzen 9 3900X) with the batch size of 64.

4.2. Implementation and training details

PPCD-GAN was implemented using PyTorch [32]. The network was trained for 100 epochs using Adam optimizer [33]. We set the initial learning rate of 0.1, which was dropped by a factor of 10 at the 30th, 60th, and 90th epochs. In our experiments, the network was with the batch size of 128, and 2 gradient accumulations (equivalent to the batch size of 256, refer [21] for more details). Empirically, we set $\delta = 10^3$ (Eq. (1)) and $\alpha = 0.7$ (Eq. (3)). All the networks were trained from scratch on a PC with GTX3090×2

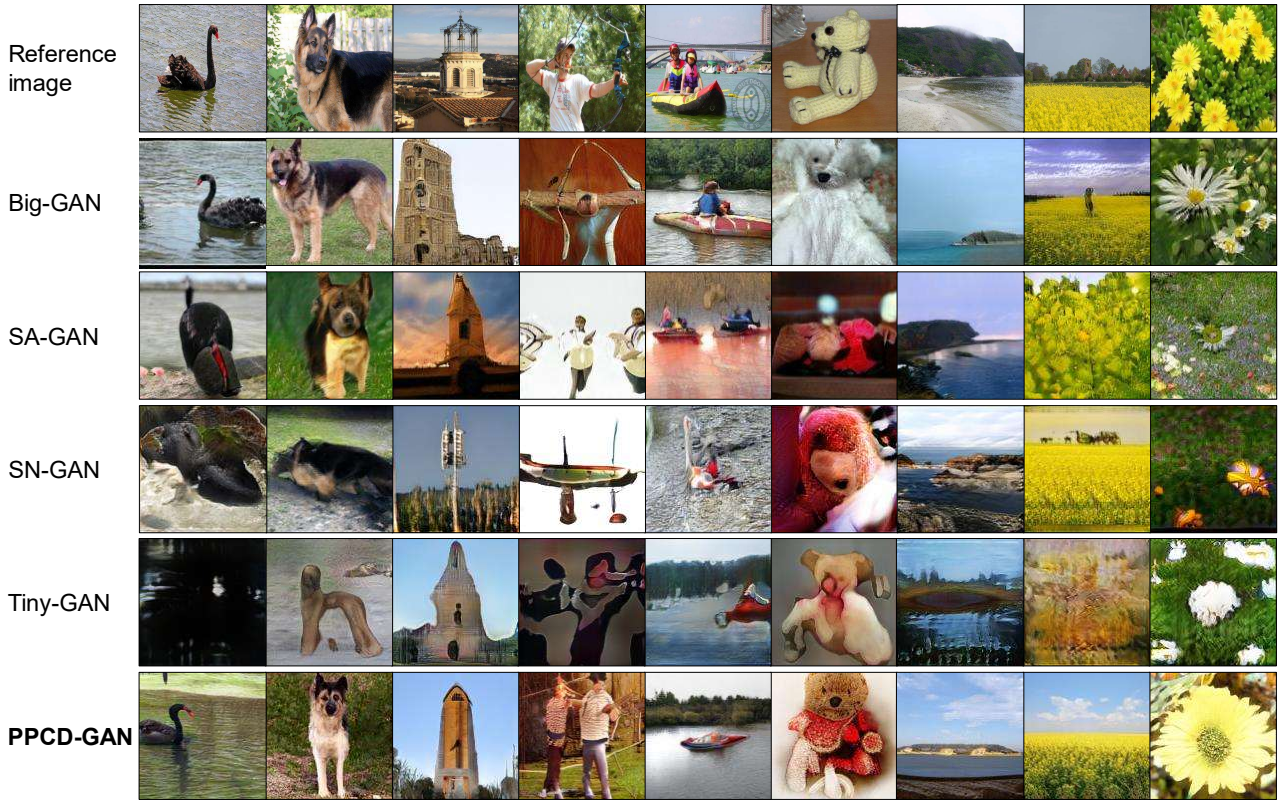

Figure 3: Visual comparison of PPCD-GAN against Big-GAN [1], SA-GAN [2], SN-GAN [3], and Tiny-GAN [7]. From left to right, conditional classes are black swan, German shepherd, bell cote, bow, canoe, teddy, promontory, rapeseed, and daisy.

where the parameters of the teacher model (i.e., Big-GAN) were fixed. It took about 10 days to train each model.

4.3. Comparison with state-of-the-arts

Qualitative evaluation. Fig. 3 shows randomly selected examples of generated images obtained by PPCD-GAN and SOTAs [1, 2, 3, 7]. Since each method receives a different noise vector as its input (along with the class vector), the generated images become different. We thus run each method 10 times for a fair comparison, and picked up the best generated images based on the Inception score. We see that the visual impressions of generated images by our method are on a par with those by Big-GAN [1], and much better than those by the other methods (SA-GAN [2], SN-GAN [3], and Tiny-GAN [7]). SA-GAN [2] is able to generate reasonable images, but the generated background is sometimes not clear (see the 2nd and 5th columns, for examples). The generated images by SN-GAN [3] regarding natural scenes are reasonable (see the 7th and 8th columns). However, we observe low quality in generating objects with more details. For Tiny-GAN [23], the generated images are incredibly miserable. Interestingly, we visually observe that Big-GAN [1] generates images with class leakage (see the images in the 6th and 8th columns) while PPCD-GAN is not the case. We attribute this advance to our usage of the class conditional normalization in the distillation process.

Quantitative evaluation. First, we evaluated the overall quality of generated images (Table 1, the second block). We see that PPCD-GAN significantly outperforms SA-GAN [2], SN-GAN [3], and Tiny-GAN [7] on IS and FID while achieving comparable levels with Big-GAN [1]. This is because PPCD-GAN learns the immense knowledge from Big-GAN [1] thanks to our class-aware distillation. Moreover, PPCD-GAN, SA-GAN [2], and SN-GAN [3] are all able to achieve diversity (see LPIPS scores; we report the average over all the classes) whereas Big-GAN [1] and Tiny-GAN [7] do less diversity. This less diversity comes from their usage of a "truncation trick" which aims at the trade-off between variety and fidelity. Note that LPIPS of each model is computed independently of the others; although the noise vector varies for each model, LPIPS always holds true. We may conclude that PPCD-GAN is more effective in perceptive visuals and diversity than the others.

Next, we evaluated the complexity of each model using #Par. and MACs (Table 1, the third block). We observe that PPCD-GAN significantly reduces the complexity. More precisely, when compared with Big-GAN [1], SA-GAN [2], and SN-GAN [3], PPCD-GAN reduces up

Table 1: Quantitative comparison against other conditional GANs methods (the best in blue; the second best in red).

Models	Quality			Complexity		Runtime (second/batch)	
	IS ↑	FID ↓	LPIPS ↑	#Par. ↓	MACs ↓	GPU ↓	CPU ↓
Big-GAN [1]	98.8	8.70	0.58	70.4M (1.0×)	21.3B (1.0×)	2.42	4.59
SA-GAN [2]	52.52	18.65	0.63	42M (1.7×)	9.6B (2.2×)	1.63	2.65
SN-GAN [3]	36.80	27.62	0.62	42M (1.7×)	9.1B (2.2×)	1.47	2.35
Tiny-GAN [7]	15.66	76.74	0.40	3.1M (22.7×)	0.44B (48.4×)	0.45	3.62
PPCD-GAN ($\alpha = 0.7$)	83.13	12.76	0.62	13.6M (5.2×)	1.6B (13.3×)	1.19	2.05

to 5.2× (81%) parameters (#Par.) and 13.3× (92%) MACs. Although Tiny-GAN [7] is more compact than our model, its IS, FID, and LPIPS are much poorer as seen above.

The out-performances of PPCD-GAN hold true when we compute the runtime (second per the batch of 64 images) on GPU and CPU for all the methods. Indeed, the fourth block in Table 1 confirms that PPCD-GAN is (almost) faster than the other models. We see that Tiny-GAN [7] does not work properly on the CPU mode because it employs the depth-wise separable convolution, which is not optimized on CPU.

We also compare our method with other GANs compression methods. Distill+NAS [6] indicates the model (SA-GAN [2]) is first trained with knowledge distillation (i.e., learnable remapping [6]) and then compressed using NAS [17]. Distill only [34] indicates the model is trained with knowledge distillation [14] (we use the settings in the original paper [34]). Table 2 shows that other GANs compression methods [6, 34] perform far worse than our method in both overall quality and diversity of generated images. Distill only [34] is faster than ours, but its quality and diversity are worst. This reveals that previous works [6, 34] struggle in handling our novel task properly. In contrast, our method successfully completes the task thanks to the proposal of simultaneously pruning and distillation. Since Distill + NAS and Distill cannot produce reasonable results, we show their generated images in the supplementary.

Finally, to clearly illustrate the advance of PPCD-GAN against CNNs compression methods, we employed ℓ_1 regularization [13] to prune the network parameters of Big-GAN [1] and SA-GAN [2] under various compression ratio thresholds (i.e., $\alpha = 0.5, 0.6, 0.7, 0.8$), and then fine-tune the compressed model. After that, for each compressed model, we generated images 10 times and computed the average of FID over the 10 times (Fig. 4). We see that increasing the compression ratio leads to quickly degrading FIDs on Big-GAN [1] and SA-GAN [2]. In contrast, we see that PPCD-GAN sustainably generates high-quality images up to $\alpha = 0.7$ even under different compression ratios while it fails in the case of $\alpha = 0.8$. It is obvious that the compression ratio of 0.8 makes the model size too small to handle a huge of classes properly. These observations confirm much better compression capability of PPCD-GAN than the typical CNNs compression methods for GANs compression.

Table 2: Quantitative comparison against other GANs compression methods (the best in blue; the second best in red).

Models	IS ↑	FID ↓	LPIPS ↑	#Par. ↓	GPU ↓
Distill + NAS [6]	6.98	217	0.13	37M	1.41
Distill only [34]	3.34	272	0.09	3.5M	0.46
PPCD-GAN ($\alpha = 0.7$)	83.13	12.76	0.62	13.6M	1.19

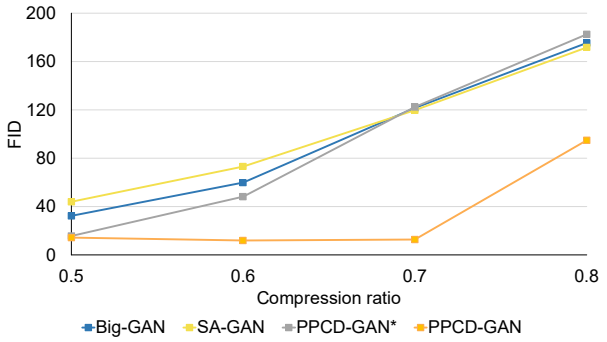

Figure 4: FID under various compression ratio thresholds ($\alpha = 0.5, 0.6, 0.7, 0.8$).

Table 3: Comparison of ablation models. We used $\alpha = 0.5$ for all the models except for PPCD-GAN w/o $\mathcal{L}_{\text{PP}}$ (the best in blue; the second best in red).

Models	IS ↑	FID ↓	#Par. ↓
PPCD-GAN (complete)	81.47	14.26	25M
PPCD-GAN w/o $\mathcal{L}_{\text{PP}}$	82.29	13.02	42M
PPCD-GAN w/o $\mathcal{L}_{\text{CD}}$	53.75	18.23	25M
.....			
PPCD-GAN*	78.65	15.76	25M

Through these various quantitative comparisons, we conclude that PPCD-GAN is superior to the others in both quality of generated images and complexity of the model.

4.4. Detailed analysis

Ablation study. We evaluated the necessity of employing PP-Res and class-aware distillation, see first three rows of Table 3. PPCD-GAN w/o $\mathcal{L}_{\text{PP}}$ denotes the model dropping PP-Res (i.e., we trained SA-GAN [2] with class-aware distillation), and PPCD-GAN w/o $\mathcal{L}_{\text{CD}}$ denotes the model dropping class-aware distillation. All the ablation models

Figure 5: Example results by the ablation models. From left to right, conditional classes are brambling, jellyfish, ladybug, bearskin, catamaran, and pizza.

Figure 6: Interpolated results by changing class condition while fixing noise vector.

in this experiment are trained with $\alpha = 0.5$ for fair comparison since the model not using class-aware distillation (i.e., PPCD-GAN w/o $\mathcal{L}_{\mathrm{CD}}$) does not work precisely with other compression ratios. PPCD-GAN w/o $\mathcal{L}_{\mathrm{PP}}$ has no pruning (i.e., is not affected by α). We see that the ablation models suffer from the quality–complexity trade-off. In particular, PPCD-GAN w/o $\mathcal{L}_{\mathrm{PP}}$ generates images with better IS and FID while PPCD-GAN w/o $\mathcal{L}_{\mathrm{CD}}$ is better in size of the model. We conclude that the PP-Res and the class-aware distillation together bring gain the performance of our complete model. Indeed, we can further compress PPCD-GAN (complete) (up to $\alpha = 0.7$, Table 1). It is worth noting that all the ablation models generated plausible images (Fig. 5).

Joint training. We evaluated the plausibility of jointly training the pruning and the class-aware distillation processes. PPCD-GAN* denotes the two-step training model where we first pruned the model and then executed class-aware distillation. The last row in Table 3 shows the performances of PPCD-GAN* (with $\alpha = 0.5$). We see that although PPCD-GAN* (with $\alpha = 0.5$) achieves comparable results with PPCD-GAN (complete), $\alpha = 0.7$ cannot be used for PPCD-GAN* (see the ablation study above). To further conduct fine-grained study on the limitation of PPCD-GAN*, we changed compression ratio α by 0.1 from 0.5 to 0.8, see Fig. 4 (gray line). We see that different from PPCD-GAN (complete), PPCD-GAN* degrades FIDs quickly and drastically as α increases. We thus conclude that training both the processes jointly is plausible. We note that training the two processes reversely is not applicable because pruning a trained model is risky as aforementioned.

Interpolation. We demonstrate the interpolation between

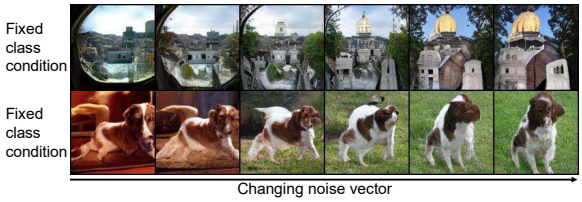

Figure 7: Interpolated results by fixing class condition while changing noise vector.

Figure 8: Failure examples of generated images.

the noise vector and the class condition vector in Figs 6 and 7. In Fig. 6, we use a fixed noise vector while changing the class condition vector. We observe that the viewpoint semantics are maintained consistently regardless of class. In contrast, when we generate images with variant noise vectors and fixed class condition vector (Fig. 7), the viewpoint as well as quantity are changed while the objects are unchanged. These observations suggest that PPCD-GAN preserves the generalization ability even it is smaller in size.

Failure cases. Though our method generates high-quality images, we find some failure cases when visually observing many generated images. Some are generated with incomplete background (i.e., blue zone) (Fig. 8). Meanwhile, the main objects are apparent, leading to their not affecting the quantitative scores (IS, FID, and LPIPS). Note that we observe that such failed images are rare. One possible reason is that since PPCD-GAN pays more attention to "important" parameters concerning main objects, parameters concerning the background may be primarily removed during training. Detailed investigations are left for future work.

5. Conclusion

This paper presents PPCD-GAN for large-scale conditional GANs compression by introducing progressive pruning residual block (PP-Res) and class-aware distillation. The PP-Res consists of the learnable mask layer and the transition layer, allowing us to progressively prune redundant network parameters during training. Class-aware distillation stabilizes training through transferring immense knowledge from a teacher model to our model via attention maps. We simultaneously train the pruning and distillation processes in the end-to-end fashion. Our experiments on ImageNet show that PPCD-GAN is notably efficient in both parameter reduction and speed while achieving comparable image quality against state-of-the-arts.

Acknowledgement. This work was supported by Institute of AI and Beyond of the University of Tokyo, and JSPS KAKENHI Grant Number JP19H04166.

References

[1] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *ICLR*, 2019. 1, 2, 3, 4, 5, 6, 7

[2] H. Zhang, I. J. Goodfellow, D. N. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *ICML*, 2019. 1, 2, 3, 4, 5, 6, 7

[3] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *ICLR*, 2018. 1, 2, 5, 6, 7

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016. 1, 2

[5] Y. Fu, W. Chen, H. Wang, H. Li, Y. Lin, and Z. Wang, "Autogan-distiller: Searching to compress generative adversarial networks," in *ICML*, 2020. 1, 2, 3

[6] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "Gan compression: Efficient architectures for interactive conditional gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 1, 2, 3, 4, 7

[7] T.-Y. Chang and C.-J. Lu, "Tinygan: Distilling biggan for conditional image generation," in *ACCV*, 2020. 1, 2, 3, 5, 6, 7

[8] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015. 1

[9] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *ICCV*, 2017. 1, 2

[10] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2019. 1, 2

[11] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *CVPR*, 2020. 1, 2

[12] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," 2020. 1, 2

[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *ICLR*, 2017. 1, 2, 7

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. 1, 2, 3, 4, 7

[15] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," in *ICLR*, 2017. 1, 2, 3, 5

[16] H. Shu, Y. Wang, X. Jia, K. Han, H. Chen, C. Xu, Q. Tian, and C. Xu, "Co-evolutionary compression for unpaired image translation," in *ICCV*, 2019. 1, 2

[17] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, 2019. 2, 7

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016. 2, 3, 4

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, 2015. 2, 5

[20] M. Kang and B. Han, "Operation-aware soft channel pruning using differentiable masks," in *ICML*, 2020. 2, 3, 4

[21] https://github.com/ajbrock/BigGAN-PyTorch. 5

[22] https://github.com/pfnet-research/sngan_projection. 5

[23] https://github.com/terarachang/ACCV_TinyGAN. 5, 6

[24] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training gans," in *NIPS*, 2016. 5

[25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *NIPS*, 2017. 5

[26] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *CVPR*, 2018. 5

[27] https://github.com/openai/improved-gan/tree/master/inception_score. 5

[28] https://github.com/bioinf-jku/TTUR. 5

[29] https://github.com/richzhang/PerceptualSimilarity. 5

[30] P. D. P. Jebashini, R. Uma and H. K. Wye, "A survey and comparative analysis of multiply-accumulate (mac) block for digital signal processing application on asic and fpga," *Journal of Applied Sciences*, 2015. 5

[31] https://github.com/zhijian-liu/torchprofile. 5

[32] https://pytorch.org/. 5

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015. 5

[34] A. Aguinaldo, P.-Y. Chiang, A. Gain, A. D. Patil, K. Pearson, and S. Feizi, "Compressing gans using knowledge distillation," *ArXiv*, vol. abs/1902.00159, 2019. 7