Tantawy *et al. Journal of Engineering and Applied Science* (2021) 68:47
https://doi.org/10.1186/s44147-021-00045-5

Journal of Engineering and Applied Science

REVIEWS Open Access

A survey on GAN acceleration using memory compression techniques

Dina Tantawy[1,2*], Mohamed Zahran[2] and Amr Wassal[1]

*Correspondence:
dina.tantawy@eng.cu.edu.eg
[1]Computer Engineering Department, Faculty of Engineering, Cairo University, Cairo, Egypt
[2]Courant institution of Mathematical Sciences, NewYork Univeristy, NewYork, USA

Abstract

Since its invention, generative adversarial networks (GANs) have shown outstanding results in many applications. GANs are powerful, yet resource-hungry deep learning models. The main difference between GANs and ordinary deep learning models is the nature of their output and training instability. For example, GANs output can be a whole image versus other models detecting objects or classifying images. Thus, the architecture and numeric precision of the network affect the quality and speed of the solution. Hence, accelerating GANs is pivotal. Data transfer is considered the main source of energy consumption, that is why memory compression is a very efficient technique to accelerate and optimize GANs. Two main types of memory compression exist: lossless and lossy ones. Lossless compression techniques are general among all models; thus, we will focus in this paper on lossy techniques. Lossy compression techniques are further classified into (a) pruning, (b) knowledge distillation, (c) low-rank factorization, (d) lowering numeric precision, and (e) encoding. In this paper, we survey lossy compression techniques for CNN-based GANs. Our findings showed the superiority of knowledge distillation over pruning alone and the gaps in the research field that needs to be explored like encoding and different combination of compression techniques.

Keywords: Survey, Generative adversarial networks, Compression, Optimization, Acceleration, Knowledge distillation, Pruning, Quantization

Introduction

Nowadays, deep learning (DL) applications are getting unprecedented popularity. Many deep learning applications are used daily such as face recognition, voice recognition, weather predictions, and image super-resolution. Companies and researchers alike compete to present more applications each day and enhance existing ones. They also compete to make them affordable and usable by everyone.

Deep generative models have been on a rise as well. GAN or generative adversarial network is one of the most famous generative models [1]. It consists of at least two networks competing against each other. It has been used in many applications like speech synthesis [2], text-to-image translation [3, 4], image-to-image translation [5], image super resolution [6], music generation [7], and videos synthesis [8].

Having one network in deep learning is very computationally intensive, thus having two networks or more is even worse. Additionally, GAN training is susceptible to divergence or mode collapses. GAN training instability adds an extra layer of complexity. Moreover, there is an increasing need for running generators of GANs on embedded devices which have limited resources and power. Thus, current techniques and accelerators need to be revisited and adapted to serve the challenges of GANs.

Accelerating GAN goals are power efficiency, speed, and solution quality. In real world, it is hard to improve everything, a price must be paid according to the no free-lunch theorem. Depending on the application, the importance of one goal over the other will vary and thus the optimization technique as well. Acceleration (optimization) techniques target three main categories:

- Memory compression: this category uses compression techniques to minimize memory requirement while preserving solution quality which in turn saves energy usage.
- Computation optimization: this category uses mathematical transformation and circuit optimization to decrease the number of mathematical operations or cycles needed alongside optimizing their needed power and increase their speed.
- Dataflow optimization: this category uses mapping, scheduling, and reordering data and/or operations to maximize data reuse and minimize ineffectual operations[1]. Those optimizations will save energy and enhance throughput.

One bottleneck of running GANs is the huge cost endured by data transfer to/from accelerating-chip memory followed by the computation cost and eventually the overhead of data transfer between different units. Thus, main goal of our presented review is to investigate the memory compression techniques because of its huge cost.

We divided memory compression techniques to lossy techniques and lossless ones. Lossless techniques are generic one, that's why we will focus on lossy techniques. In our taxonomy, we divided the lossy techniques to five categories: (a) pruning, (b) knowledge distillation, (c) low-rank factorization, (d) lowering numeric precision, and (e) encoding. Those five techniques are explained thoroughly in this survey, focusing only on methods applied to GANs.

Several survey papers exist about deep learning compression like [9–12]. However, those papers discuss general deep learning algorithm not targeting issues specific to GANs. While some techniques mentioned in the previous surveys might apply to GANs, GANs propose more challenges and opportunities. First, generative networks are sensitive to number representation. In other words, not all quantization techniques used for general DL models would be efficient for use with GANs. Second, GAN training suffers from instability, which makes normal compression and pruning techniques inefficient. Finally, GAN output resolution is large and correlated as opposed to normal classification or regression problems whose output are very small.

Another related work is the work done by Gou et al. [13]. Although this work "uses" GAN to perform distillation, it does not consider GAN themselves for compression. While the work by Wang and Yoon [14] mentioned briefly the impact of distillation on image translation tasks using GANs, its focus was using GAN to perform distillation to other models.

[1] Multiply by zero is considered an ineffectual operation as we already know the result without multiplying

Our work will focus on CNN-based GANs although it applies to other types of GANs as well. Computational optimization and dataflow optimization are defferred to future work. Our work is complementary to other survey papers mentioned above as it highlights special issues facing GANs specifically which are not well covered in other surveys.

This paper has the following contributions:

- To our best knowledge, this is the first paper to survey GAN compression.
- Providing a taxonomy for GAN optimization.
- Summarizing recent research work in accelerating GANs.
- Providing open research questions for accelerating GANs.

The remainder of this paper is organized as follows: the "Memory compression techniques for GANs" section presents a brief background on how GANs work followed by reviewing different efforts to optimize GANs using compression techniques and eventually, it provides open research questions that need to be further studied. Finally, the "Conclusion" section concludes the work.

Memory compression techniques for GANs

In this section will start by background explanation for GANs and its performance metrics in the "Background" section, followed by reviewing latest work on lossy compression techniques in the "Memory compression techniques". It will explain each technique and show the latest work using them on GAN generators. In the last section "Results and discussion", we will provide summary of findings and open research questions that need to be further studied.

Background

Generative Adversarial Network (GAN) is a type of generative model that uses Deep Learning (DL) techniques to generate data. As mentioned earlier, GAN is more of a way of training different smaller models to compete against each other than being a newly devised model.

CNN-based GANs can have many different architectures like DCGAN [15] and Pix2Pix [16]. Although different CNN-based GAN models have different architectures, most of the models share the same underlying concept of having competitive networks and using transpose convolution or upsampling layers. In this section, we will use DCGAN to explain the idea of GANs in general due to its simplicity and wide use.

Structure: GAN model consists of at least two networks[2]: generator and discriminator in an organization like Fig. 1. In training, the generator takes an input noise (z) and generates data $G(z)$ that looks more like a real data not a synthesized (fake) one, while the discriminator tries to be better at discriminating the generated (fake) data from the real ones x. Thus, the goal of the discriminator is opposite to that of the generator. That's why it is called "adversarial" as they are competing against each other. The training ends when (i) the discriminator cannot enhance its accuracy anymore and (ii) the evaluation metrics are satisfactory. Sometimes the training does not converge, and measures and limits are used to halt the training process and restart it.

[2] Some applications like style-transfer requires more than one GAN and more than two networks.

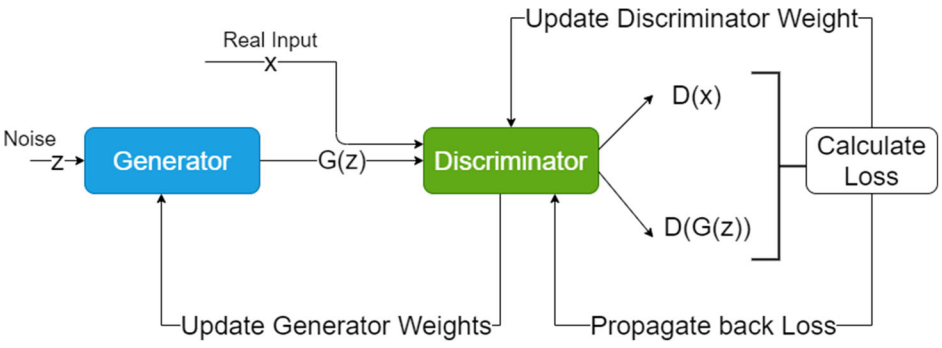

Fig. 1 Example of GAN general organization

The optimization problem can be represented as the following equation

$$\begin{aligned}\mathcal{L}_{\text{gan}} &= \min_G \max_D V(D,G) \\ &= E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z))] \end{aligned} \tag{1}$$

where G represents the generator, D represents the discriminator, $p_{\text{data}}(x)$ represents the original data distribution, and $p_z(z)$ represents the noise input distribution. The first part of the equation $E_{x\sim p_{\text{data}}(x)}[logD(x)]$ represents the probability that the discriminator classifies real data as real. The second part of the equation $E_{z\sim p_z(z)}[\log(1-D(G(z))]$ represents the ability of the discriminator to classify fake data as fake. The discriminator tries to maximize those two parts of the equation, that's why we need $\max_D V(D,G)$. On the other hand, the generator tries to fool the discriminator to detect fake images as real, thus it tries to minimize the second part of the equation, hence adding the $\min_G V(D,G)$. Combining the two network optimizations, we get the above equation.

A discriminator network is an ordinary CNN, or LSTM, or any other deep-learning classification/regression model. In contrast to the discriminator that outputs only a decision or a prediction, a generator network generates the data itself (e.g., an image, music notes, and animated character). Thus, the output of the generator network is larger than its input. Figure 2 shows the generator network in DCGAN. Like most generator networks, the input is a noise vector or initial image that gets expanded and reshaped to a bigger size. This expansion and reshaping are performed to train the convolution to act as a transposed convolution.

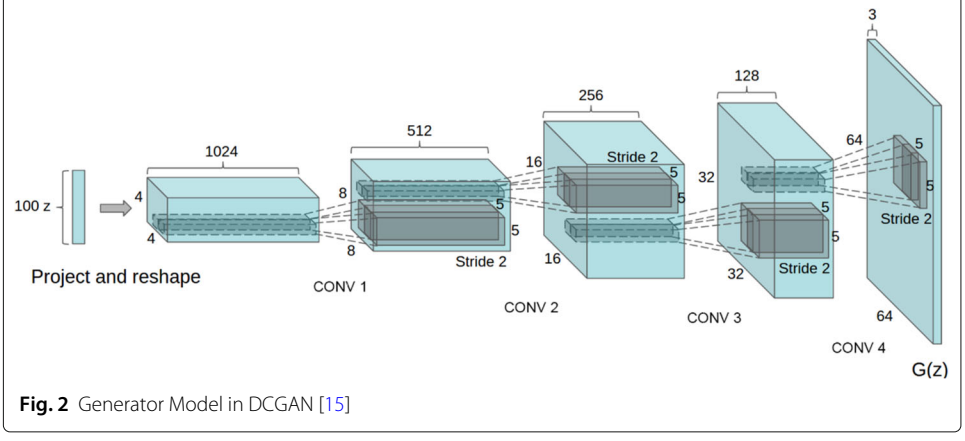

Fig. 2 Generator Model in DCGAN [15]

Table 1 Functional metrics (scores) summary

Name	Enhancement direction
Inception score (**IS**)	Higher is better
Fréchet inception distance (**FID**)	Lower is better
Peak signal-to-noise ratio[a] (**PSNR**)	Higher is better
Mean pixel accuracy[a] (**mPA**)	Higher is better
Intersection over union[a] (**IoU**)	Higher is better

[a] Maximum value is 1 and requires ground truth

Evaluation metrics: GANs have many evaluation metrics. we can split metrics into two types: (1) functional metrics and (2) performance metrics. Functional metrics measure how good the result of a GAN towards the target functionality. We can also name them as "quality" metrics or scores. Most commonly used function metrics are inception score (IS), Fréchet inception distance (FID), peak signal-to-noise ratio (PSNR), mean pixel accuracy (mPA), and intersection over union (IoU) . Both IS and FID measure the quality of the generated outputs and their diversity. More information about how they are calculated can be found in the following work [17, 18]. The higher the IS, the better the diversity in the generated data. On the other hand, the lower the FID, the better image quality is produced. PSNR is also used especially in blending images or creating a super-resolution image (the higher the better). mPA measures the mean difference in percentage between the generated image and the ground truth. IoU measures how close the generated image to the "real image" (the higher the better, maximum =1). Functional metrics are summarized in Table 1.

Performance metrics measures the efficiency of the model optimization. The efficiency of a model refers to its speed, power, and used area. These metrics depends on both software model and hardware architecture and used technology. Focusing on compression techniques, our main metrics is the compression ratio, number of MACS (multiply-accumulate operations) and throughput as listed in Table 2.

Memory compression techniques

What is memory compression? Any DL Model needs three items to reside in the memory: (1) model architecture (control flow or computational graph), (2) model parameters (weights and biases), and (3) inputs (activations). Compressing memory means minimizing any/all the above three components.

What are the advantages of using memory compression techniques? It improves storage space required to store model leading to cross-layer optimization by allowing several layers to fit into the chip memory at once or even storing the whole model on embedded system. It also decreases off-chip data-transfer leading to higher throughput and lower power usage.

Table 2 Performance metrics summary

Name	Enhancement direction
Compression ratio (**CR**)	Higher is better
No. multiply-accumulate ops. (**#MACs**)	Lower is better
Throughput (**output/s**)	Higher is better

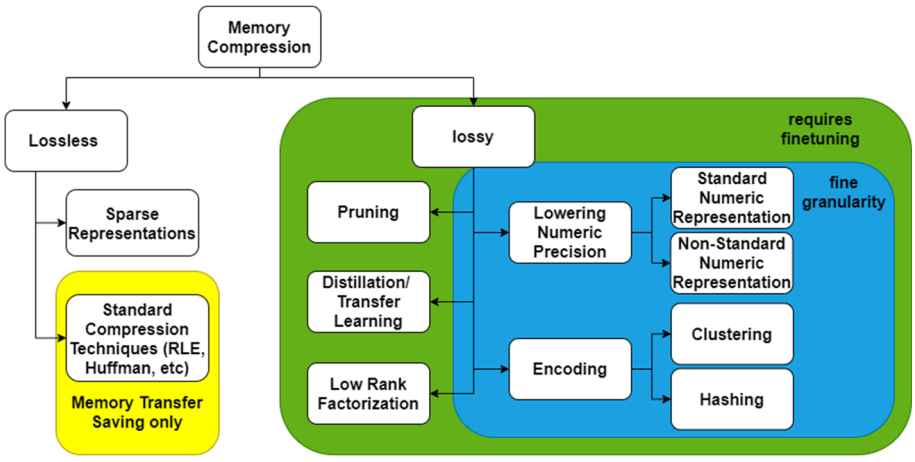

Fig. 3 Classification of memory compression techniques

What is the classification of memory compression techniques? Memory Compression can be done using several techniques as shown in Fig. 3. Compression techniques are classified into two main categories: (a) lossless compression techniques and (b) lossy ones. As the name stated, lossless compression techniques does not impact the accuracy of the computation, it just optimizes the needed storage and hence memory transfer cost is reduced. Lossless compression techniques can be further classified into sparse representation which work on sparse matrices such as compressed sparse column format (CSC), compressed sparse row format (CSR), etc. [19, 20] and standard compression techniques like Huffman [21], run length encoding(RLE) [22], etc which depends on the statistical distribution of the data. In both branches of lossless techniques, the actual data is preserved. However, lossless compression techniques require extra software and/or hardware support to revert compression or apply computation on sparse matrices. On the other hand, lossy compression techniques introduce losses. That is why finetuning or retraining the compressed model is advised with lossy techniques such as pruning, distillation, low-rank factorization, lowering numeric precision, and encoding.

We also classify the techniques according to their granularity, a coarse granularity would consider compressing the model architecture, while a finer granularity would consider compressing data used by the model (both weights and activation). Combining several techniques together is always an option, but a careful eye needs to watch the quality of the solution.

Lossless compression techniques are general ones that can be used with different applications and are orthogonal to lossy ones, thus why we choose to focus on lossy techniques in the rest of this section.

Pruning

It is the process of eliminating parts of the network model to make it smaller without much loss of results quality. Pruning is usually done after the model is trained and then the model is finetuned to adjust the remaining weights as seen in Fig. 4.

Pruning is defined by 4 main decisions as seen in Fig. 5. The first decision is the pruning criteria or in other words "how to choose the part to be pruned." Criteria could be totally random using trial and error or based on the selected element norm using some certain

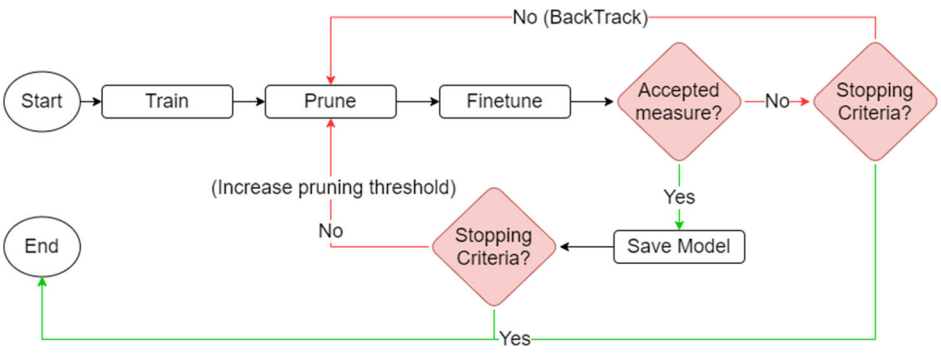

Fig. 4 Pruning process flowchart

threshold a.k.a. norm-threshold [23]. A similarity threshold also could be used when there are two activations (feature maps) very similar to each other, hence, one of them can be removed as it introduces no new information [24]. Finally, evolutionary algorithms like genetic algorithm can be used to choose the elements to be pruned [25].

The second decision is the pruning granularity. The pruning could be unstructured which means eliminating individual elements from weight/bias matrices. This kind of pruning leaves the network structure as it is, but it converts its weight matrices to sparse matrices which could be further compressed [23]. On the other hand, structured pruning eliminates components from the network leaving it slimmer (less channels or filters) or shorter (less layers) [24, 25].

The third decision is the application time or "when to apply pruning". As mentioned previously, most work apply pruning as post-processing step; however, lately, several techniques are introduced to apply pruning while training as will be explained later in this section [24]. The post-processing pruning can further be classified to gradual or iterative pruning where the pruning starts with a small threshold and increases the threshold

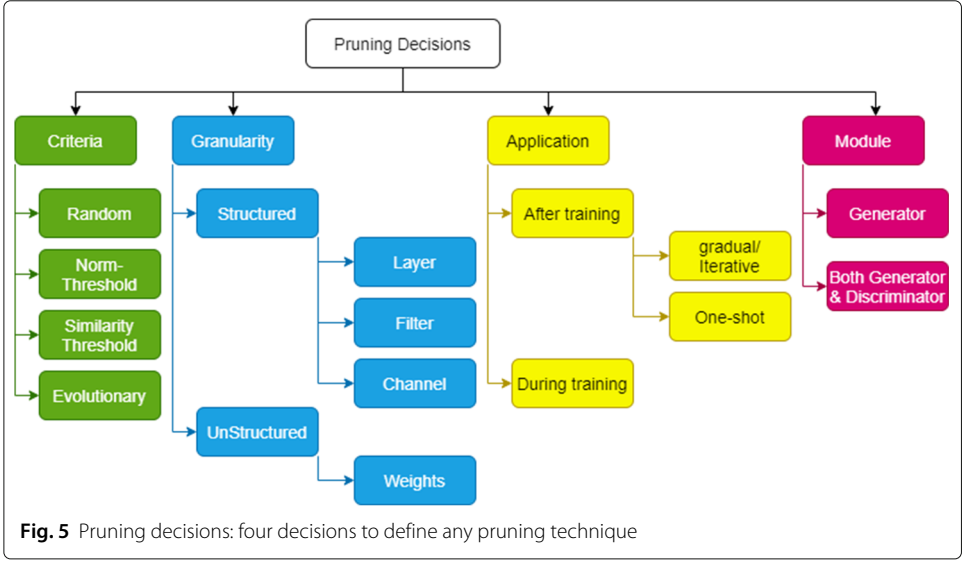

Fig. 5 Pruning decisions: four decisions to define any pruning technique

gradually to avoid accuracy loss [23]. On the contrary, the one-shot pruning is manually pruning several networks and finetuning all of them at once to get the best pruned network [23].

The final decision is the module or "which module should be pruned, the generator module or both the generator and discriminator modules." Pruning the generator part of the network is usually the target, however due to instability of training and hence finetuning, some works suggested to prune the discriminator as well to avoid overpowering the pruned generator while finetuning [23].

In a study made by Chong and Jeff [23], they showed that the quality of results represented in FID scores [18] degraded significantly using thresholding pruning as seen in Fig. 6. Thresholding pruning is eliminating the element undertest if it is below a certain threshold. The element could be a single weight, a filter, or a channel. They implemented several pruning techniques on StarGAN [5]. They implemented iterative pruning after training [23], iterative pruning during training [23], pruning both generator and discriminator [23] and one-shot pruning after training [23]. It worth noting that pruning during training resulted in a lower quality than pruning after training. This indicates that the model fails to converge. They also presented other non-pruning techniques which we will explore later.

Instead of using thresholding in pruning, Shu et al. [25] proposed using evolutionary algorithm like genetic algorithm (GA) to compress the cyclic networks like CYCLEGAN. The idea is to represent the generator as a bitstream where each bit corresponds to a filter if the bit = 0, then the filter is pruned. The GA fitness function is a function in three criteria: (a) the size of the network, (b) the compression distance, and (c) the cycle loss. The compression distance is the mean square error between the discriminator output for compressed and uncompressed generators. Whereas the cycle loss is a special loss in training paired images as explained in [26]. GA achieved a compression ratio between 3.54× and 5.7× compression ratio on CYCLEGAN. GA pruning achieved 0.542 mean pixel accuracy compared to 0.218 using thresholding in pruning. It also achieved a better FID score

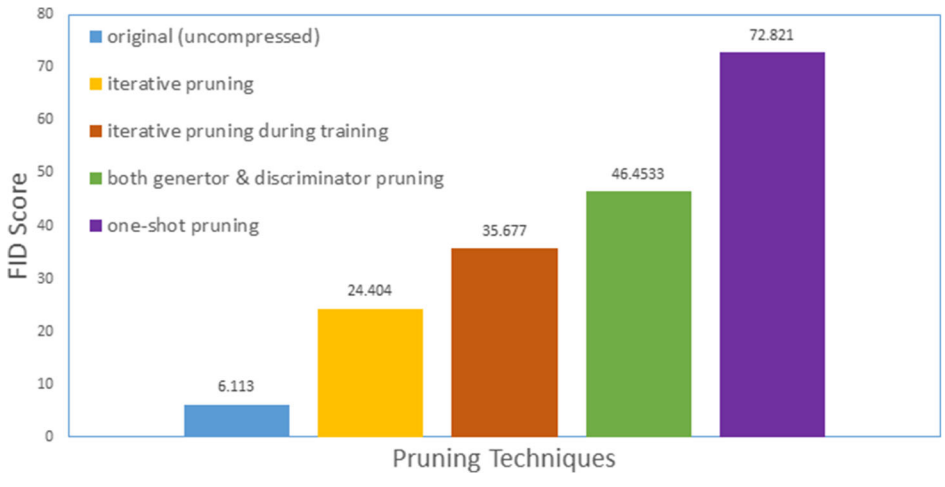

Fig. 6 FID scores of different pruning techniques for StarGAN. This shows the failure of pruning techniques to preserve the required quality compared to the original non-pruned version of the algorithm

by average of 30 points compared to thresholding pruning while only degrading FID than the original none-pruned with 8.5 points (calculated using 4 datasets on CYCLEGAN).

Song et al. presented overlapping pruning with training instead of the ordinary train-prune-finetune approach [24]. In their work, the authors adopted the train-expand-prune approach. He started by training a small network (called seed network), then progressively expanded it by adding more width (filters) to the network. Then similar feature maps are pruned and the whole network is fine tuned. They scored 1.25× less flops than baseline GAN.

Although many combinations of pruning techniques have not been explored in pruning as seen in the summary (Table 3), the current results indicate that pruning alone is not enough and there is a significant loss in quality as shown in Fig. 6. This failure is attributed to the following reasons: (1) the high resolution of the generator output compared to discriminator models makes it more sensitive to noise, (2) The generator evaluation metrics are more subjective than objective, and (3) the training of GANs is unstable and care should be taken to avoid discriminator over-powering the generator. To overcome those challenges, a more general approach called knowledge distillation is used where pruning is usually a part of it.

Knowledge distillation

It is the transfer of knowledge acquired by the uncompressed generator (called teacher model) to a smaller model (called student model). To apply knowledge distillation, we need to define 4 main components as explained in Fig. 7.

First component is the teacher model. While the straight-forward approach suggests that we should have a pretrained one that we are trying to compress, some works trained an overly large model from scratch to give them more flexibility in finding the most optimal student model.

The second component is the reconstruction of the student model. In its simplest form, pruning is used to generate the student model from the teacher model. Student models can also be constructed using Network Architecture Search (NAS). Or even it can be progressively constructed using sub-constructs from the teacher model.

The third component is the training architecture or building blocks. This component is concerned with which components from the teacher model will be included in the training and whether to construct a complete student GAN (both generator and discriminator) or just construct a student generator.

Table 3 Summary of GAN pruning work

Work/aspect	Criteria	Granularity	Application	Module
[23](e)	Threshold	Unstructured	After training	Generator
[23](f)	Threshold	Unstructured	During training	Generator
[23](n)	Threshold	Unstructured	After training	Generator
[23](d)	Threshold	Unstructured	After training	Both
[25]	Evolutionary	Structured	After training	Generator
[24]	Similarity	Structured	During training	Generator

(e) Iterative pruning after training
(f) Iterative pruning during training
(n) Pruning both generator and discriminator
(d) One-shot pruning after training

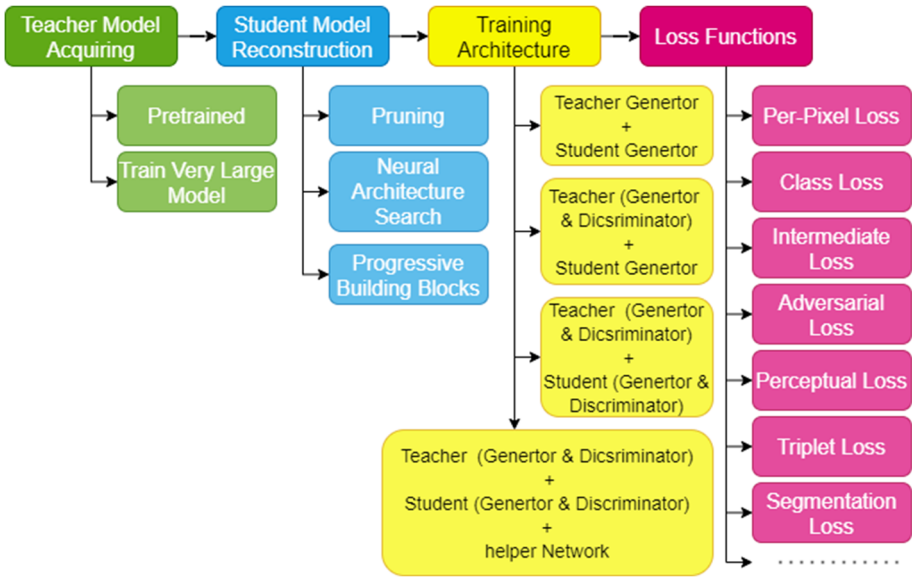

Fig. 7 Knowledge distillation components

The last component is the loss function. The loss function determines how good and how fast the student will learn from the teacher. A lot of loss function has been introduced that will be explained below.

In [27], Aguinaldo et al. used knowledge distillation to train a student generator to even beat the teacher generator. They used pretrained teacher generator and discriminator as seen in Fig. 8. They devised a reconstruction loss (Eq. 3) as a joint function between gan loss (Eq. 1) and per-pixel loss (Eq. 2). First, they provided the same input to both student and teacher generators. Second, the output of both generators are fed into the teacher discriminator. Third, they calculated the reconstruction loss as in (Eq. 3) to train the compressed generator.

$$\mathcal{L}_{\text{per\_pixel}} = \text{loss}(G_{\text{student}}(z), G_{\text{teacher}}(z)) \tag{2}$$

$$\mathcal{L}_{\text{recon}} = \mathcal{L}_{\text{gan}} + \lambda \mathcal{L}_{\text{per\_pixel}} \tag{3}$$

where z is the noise used as input to generators, $\mathcal{L}_{\text{gan}}$ is the adversarial gan loss from (Eq. 1), and λ is a weighting parameter between both losses. The authors used a very large teacher to guide the small network leading 1669× compression ratio while retaining 83% of the teacher's inception score on MNIST. However, the produced images were

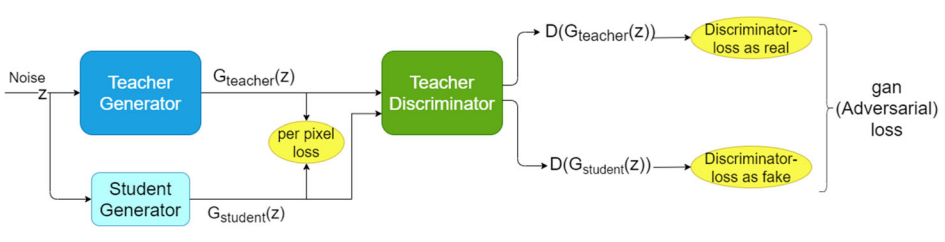

Fig. 8 Training architecture for the work [27] showing per-pixel loss and gan-loss (a.k.a. adversarial loss)

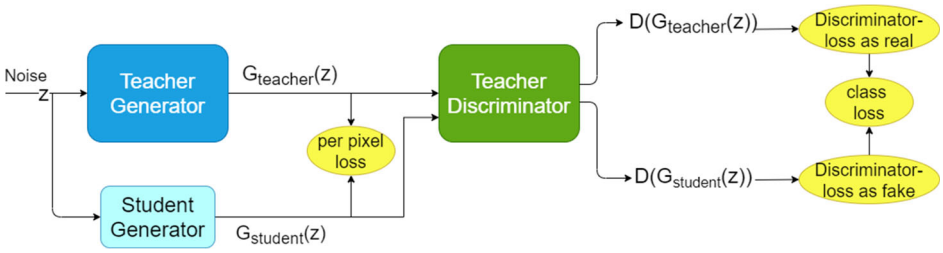

Fig. 9 Training architecture for the work [23] showing per-pixel loss and class-loss

very blurred at this compression rate. Yet what makes this work standout is that the distilled networks using this method always beat the trained-from scratch networks of the same size.

In [23], Chong and Jeff also used a pretrained generator and discriminator to train a student generator as shown in Fig. 9. They measured the loss as follows: first they measured the per-pixel loss as the loss between the two generators as in (Eq. 2). Per-pixel loss is usually the mean square error loss, but it could be anything else. Additionally, they measured class-loss as the loss between the discriminator output of the student and teacher generators as seen in Eq. 4. The total loss is the weighted sum of the previous two losses as shown in (Eq. 5).

$$\mathcal{L}_{\text{class\_loss}} = \text{loss}(D_{\text{teacher}}(G_{\text{student}}(z)), D_{\text{teacher}(G_{\text{teacher}}(z))}) \tag{4}$$

$$\mathcal{L}_{\text{total}_1} = \mathcal{L}_{\text{per\_pixel}} + \lambda \mathcal{L}_{\text{class\_loss}} \tag{5}$$

Similarly, In [28], they used $\mathcal{L}_{\text{gan}}$ function using the teacher discriminator. Additionally, they added another loss term representing intermediate distillation loss. Intermediate distillation loss is the loss between two corresponding inner-layers outputs as in (Eq. 6). If the data is paired, then they calculate $\mathcal{L}_{\text{per\_pixel}}$ between student generated image and the paired image, else they use the output of the teacher generator to calculate $\mathcal{L}_{\text{per\_pixel}}$. The training architecture is shown in Fig. 10 while the final loss function is seen in (Eq. 7).

$$\mathcal{L}_{\text{interdist}} = \sum_{t=1}^{T} ||f_t(G_{\text{teacher}_t}(x)) - G_{\text{student}_t}(x)||_2 \tag{6}$$

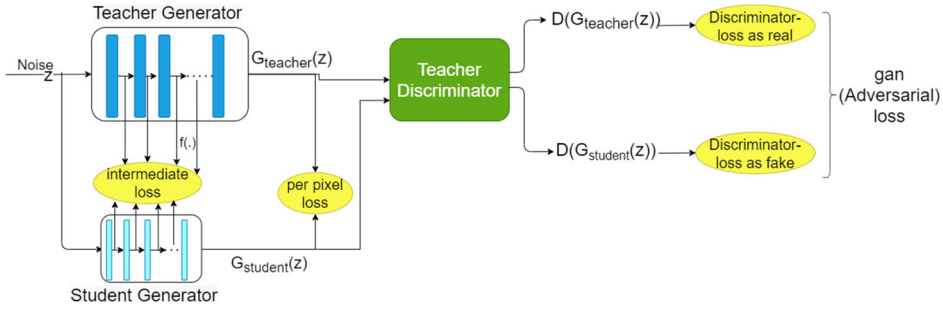

Fig. 10 Training architecture for the work [28, 30] showing Intermediate distillation loss, per-pixel loss, and GAN loss

where f_t is the mapping function between teacher inner layer and the corresponding student layer to adjust the size. t is the layer number.

$$\mathcal{L} = \mathcal{L}_{\text{gan}} + \lambda_{\text{per\_pixel}}\mathcal{L}_{\text{per\_pixel}} + \lambda_{\text{interdist}}\mathcal{L}_{\text{interdist}} \tag{7}$$

where different λ are used to weight the different loss functions.

To construct the student model, they used a neural architecture search (NAS). To avoid the long running time of NAS, they used one shot learning to train a variable number of networks at once. The idea of one-for-all network training (OFA) also called one-shot learning in training is best explained in [29]. They reached a compression ratio between $4\times$ and $33\times$ on various datasets and networks.

In contrast to the previous work, the work in [30] suggested to use more constrained search for the student model instead of unconstrained NAS. They built DART, an AutoML-like framework for GANs to perform differential search [31] for the operators at each layer and layer width . They constrained the first and last layers to be like famous models' architecture like Cycle-GAN in style-transfer tasks and ESRGAN [32] in super-resolution tasks.

In the work made by Qing et al., they reconstructed the teacher network to be a large supernet for image-to-image translation [33]. Thus, pruning and distilling such a large network would lead to a more efficient student network. The student network is made by pruning channels of the teacher generator. They used aggregated 4 loss functions. The training architecture is shown in Fig. 11. First, they used intermediate distillation using kernel alignment function to map the corresponding intermediate layer sizes. Second, they used perceptual loss, which is the loss between intermediate features in the discriminator between real(teacher) and student images as seen in (Eq. 8).

$$\mathcal{L}_{\text{perc}} = \sum_{t=1}^{T} \frac{1}{N_t} ||D_t(x) - D_t(G_{\text{student}}(z))||_1 \tag{8}$$

where x is real image or teacher image$G_{\text{teacher}}(z)$, and T is the number of layers and N is the total number of elements in each layer.

Third, they used gan adversarial loss as all other GANs. Eventually, they used the cyclic-loss since it is image-to-image translation task. The cycle loss is the loss of converting

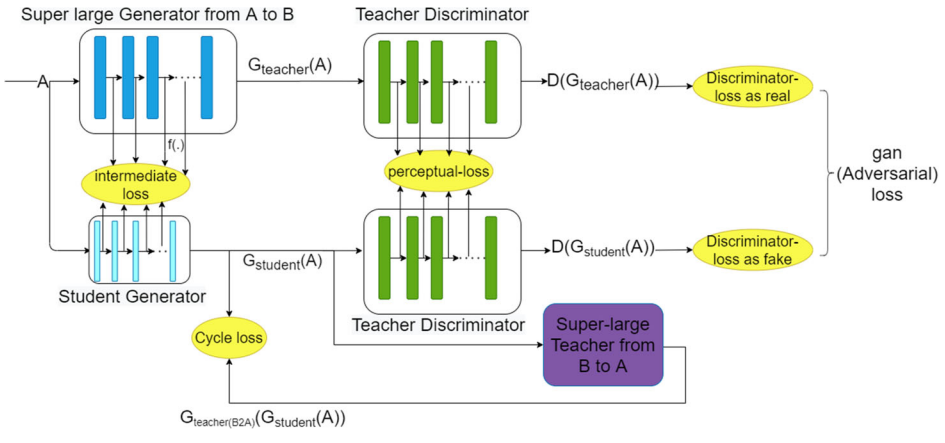

Fig. 11 Training architecture for the work [33] showing intermediate distillation loss, perceptual-loss, and gan-loss

from domain A to B then back to A'. The difference between A and A' is the cycle loss. This reconstruction method optimized the time required to construct student network by $10,000\times$ compared to unconstrained NAS with better FID scores.

Another work by Chen et al. distilled both generator and discriminator. They used two discriminators instead of one as shown in Fig. 12. The distillation of the generator required two loss functions: perceptual-loss, per-pixel loss and adversarial-loss. To distill the discriminator, they used the adversarial loss, intermediate distillation loss on discriminator and introduced another loss term, which is triple loss. The idea of this loss term is to consider that the distance between teacher generated and student generated images will be smaller than the distance between real images and student generated images, thus they added an extra parameter to weight the two differences in the loss function.

In [35], the authors combined intermediate distillation loss, perceptual loss and per-pixel loss as shown in Fig. 13. They also used a content aware approach to enhance distillation. This is done by detecting the areas of interest using an auxiliary segmentation network (content-aware network) and mask the corresponding images before calculating loss leading to $10\times$ to $11.5\times$ compression ratios compared to several original GANs with FIDs $\sim$7.5 for FFHQ dataset (original FID 2.7$\sim$4.5). Zhang et al. also combined intermediate distillation loss from discriminator and generator in their work presented PKDGAN [36]. However, it is only applied on novelty detection, so a further work needs to compare it with other GAN methods.

In a work done by Haotoa et al., they presented a unified framework called ganslimming to stack several memory compression techniques together [37]. They used distillation to compress the generator. The student generator is automatically and adaptively generated from the teacher generator by channel-pruning and quantization. They used the normalization layer scale parameter to guide the pruning during training and adapted the adversarial-loss, per-pixel loss, and the added normalization scale parameter as the combined loss function.

Table 4 summarizes the above-mentioned works and how their four main components are selected. While Fig. 14 shows the difference performance of the above techniques

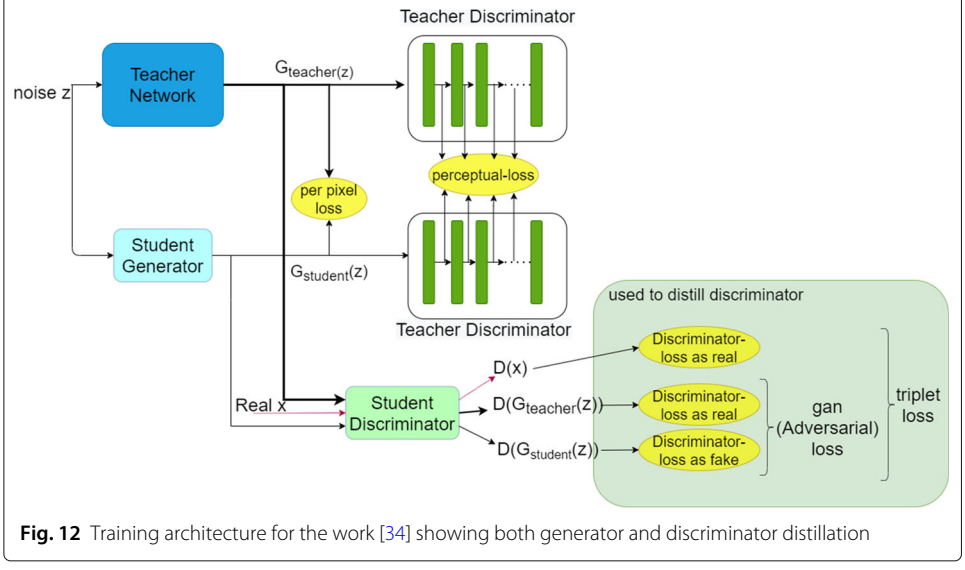

Fig. 12 Training architecture for the work [34] showing both generator and discriminator distillation

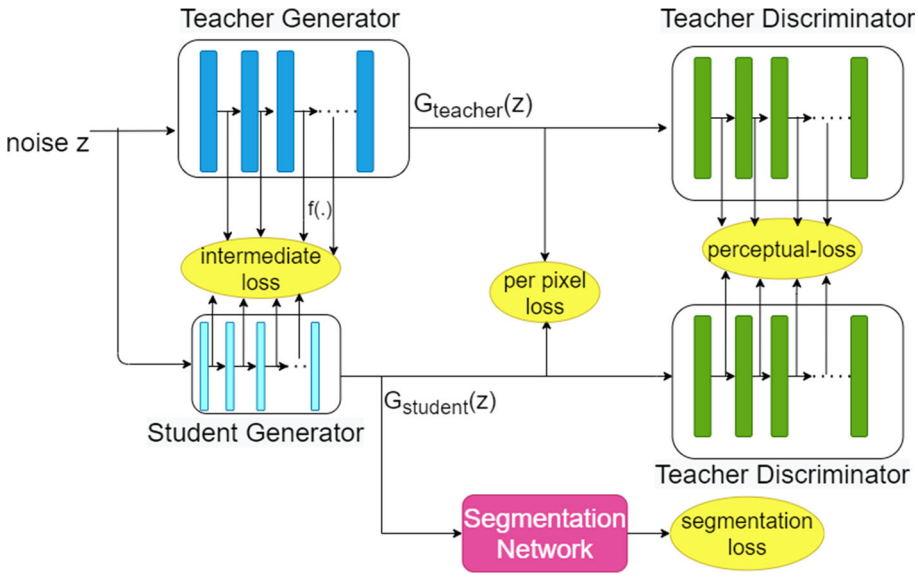

Fig. 13 Training architecture for the work [35] showing both generator and discriminator distillation and using segmentation network to allow content-aware distillation

Table 4 Summary of GAN distillation works

Work/component	Teacher model	Student recons.	Training architecture	Loss function
[23]	Pretrained	Pruning	2G+ Teacher D	Per-pixel Per-class
[27]	Pretrained	Pruning	2G+ Teacher D	Per-pixel Adversarial loss
[28]	Pretrained	NAS	2G+ Teacher D	Per-pixel Adversarial loss Intermediate loss
[30]	Pretrained	NAS (DART)	2G+ Teacher D	Per-pixel Adversarial loss Intermediate loss
[33]	Super-large	Pruning	2G+ Teacher D	Perceptual loss Adversarial loss Intermediate loss Cycle-loss [26]
[34]	Pretrained	Pruning	2G+ 2D	Per-pixel Perceptual loss Adversarial loss Triplet-loss
[35]	Pretrained	Pruning	2G + D + Segmentation Network	Per-pixel Perceptual loss Intermediate loss Segmentation-loss
[37]	Pretrained	Pruning	2G+ Teacher D	Per-pixel Adversarial loss Normalization-loss[38]

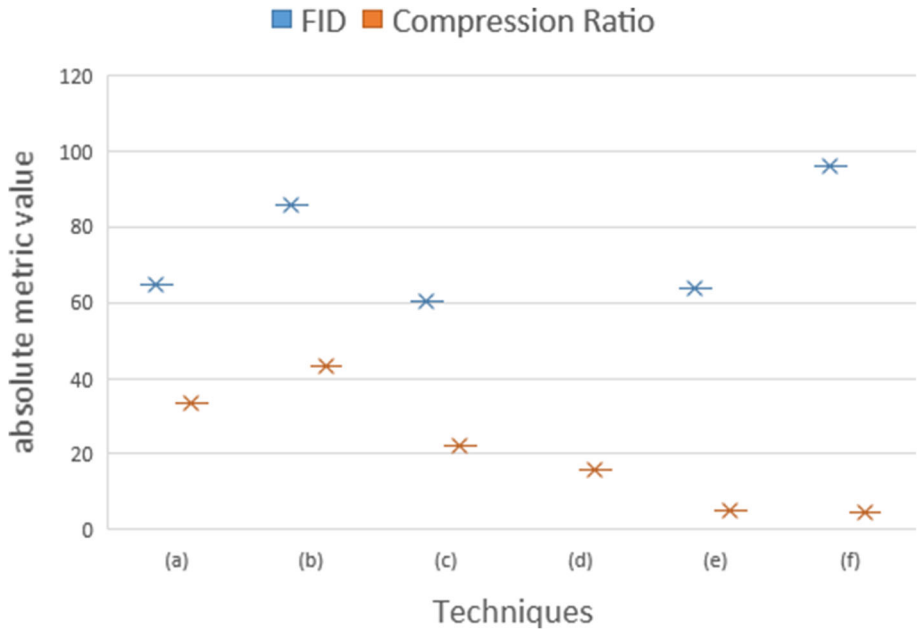

Fig. 14 Comparison between different distillation and pruning techniques using cyclegan model and horse2zebra dataset. **a**, **b**, **c**, **d**, **e**, and **f** represents the works in [25, 28, 30, 33, 34, 37], respectively. (f) [25] is a pruning technique, while all the others are distillation. In **e**, we only reported the distillation results and omitted the quantization effect for a fair comparison. Work **a**, **c** had stronger teacher than the rest. In **c**, we estimated the compression ratio as the ratio between the number mac operations

with respect to each other and with a sample from pruning techniques as well. For a fair comparison, we used only techniques with Cyclegan and horse2zebra dataset. The axes names in Fig. 14a, b, c, d, e, and f represent the following works [28], [30], [33], [34], [37], [25], respectively. Chen et al. [34] did not report the FID score in his paper, that is why it is omitted from the graph. In works [28, 33], their teacher model FID score was better than the others. This justifies why they have much better FID score because they had a better teacher. However, Wang et al. in [37] managed to score very close to them despite starting from a weaker teacher although its compression ratio is one of the lowest compared to others. It worth noting that the pruning technique done by [25] has the worst compression ratio and the worst FID score, which is consistent with our conclusion in pruning section, that pruning alone is not enough.

Low-rank factorization

Low-rank factorization is very famous in optimizing both storage and computation for sparse matrices [39]. Low-rank factorization is a lossy technique and introduces some loss in contrast to other lossless sparse representations. It uses a matrix (filter) of a lower rank (hence less storage and less computation) to perform the operation. Low-rank factorization is a very wide field that is applicable to many applications not just GANs. A good reference is in [40, 41]. Most computational engines like Tensorflow [42], Pytorch [43] make computational optimization seamlessly, which explains why very few works reported for using them.

Wu et al. used low-rank factorization in his work PDGAN [44]. Despite approximation, PDGAN achieved close score to other state-of-the-art methods in recommendation systems. Unfortunetly computational and compression performance are not reported.

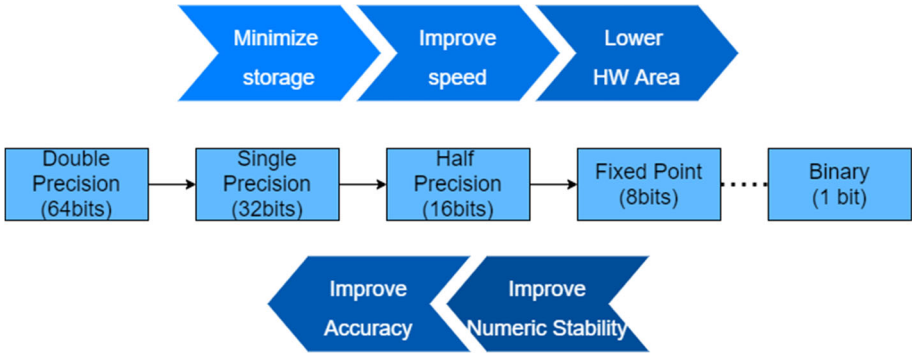

Fig. 15 Impact of precision on performance metrics

Lowering numeric precision

As the name stated, it is using less bits to represent a number. Lowering numeric precision is not unique to Machine Learning. As the precision increases the accuracy and numerical stability increases. On the other hand, as the precision decreases, the speed, memory footprint and hardware area get improved as shown in Fig. 15. Despite that, the relation is not linear, and it differs according to the application.

As mentioned earlier, GAN generators are more sensitive to precision due to the resolution of the output. Thus, in this section, we will show the impact of such optimization on generators and explore different numeric formats. Changing numeric precision can be done by using standard formats like single precision (float32), half precision (float16) or fixed-point representation which can take many forms depending on the place of the fixed-point. Those standard format has more adoption in hardware since they are "standard." On the contrary, Non-standard formats use out-of-the-box ideas such as Bfloat[3] and FlexPoint. Those out-of-the-box formats need dedicated hardware support to be used.

Lowering numeric precision to fixed-point integer is usually called quantization in literature whenever the quantization follows the affine mapping as in Eq. 9. The advantage of using affine transformation is that the multiplication and addition can be carried out without the need to revert the mapping [45]. Other types of quantization requiring reverting back the conversion before calculations is considered as a type of encoding.

$$\text{Quantized\_number} = \frac{\text{Full\_precision} - \text{Zero\_point}}{\text{Scale\_factor}} \tag{9}$$

To perform quantization, we need to define the necessary decisions shown in Fig. 16. First decision is the general numeric format; whether the number format is float-like or integer-like. Float-like format is (sign, exponent, mantissa) format. The integer-like format like fixed point format requires (integer, scale factor, zero-point) format.

Second decision is the quantization function. Determining the quantization function determines how the full-precision number is converted to the quantized one as shown in Eq. 9. The quantization could be uniform or could give a higher priority to certain range depending on the data distribution using log or tanh function.

[3] Although Bfloat is not IEEE standardized, however, Google TPU support for it makes it common to use and supported by different optimized deep learning libraries.

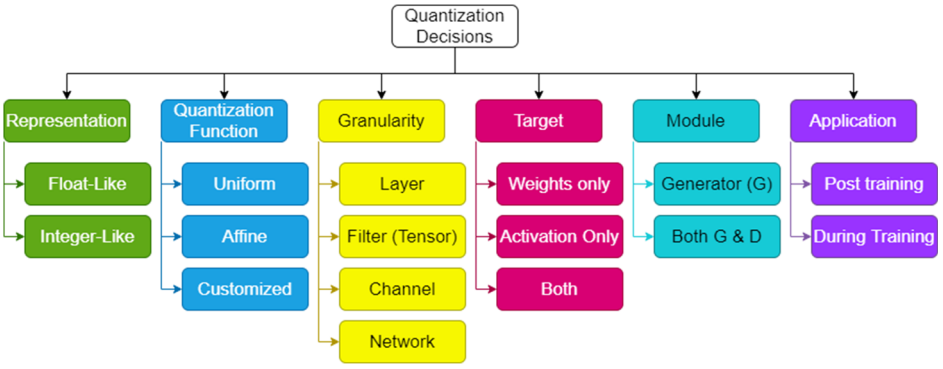

Fig. 16 Quantization decisions

Third to Fifth decisions are about what part of the network to apply quantization to. Starting by the granularity whether the same quantization should be applied to all layers, or each layer should have different quantization. The quantization could be per-tensor, per-channel, per-layer, or per-network. The fourth decision is whether to quantize weights, activations, or both. If only one type of data is quantized, then the saving will be in storage only not in calculation as it must cast back to the largest of the two formats. And the fifth one is whether to quantize both networks or the generator only.

Last decision is when to apply the quantization. Post-training quantization requires finetuning while during training or called "Training-aware quantization" does not need any additional finetuning.

Wang introduced a method to quantize GAN called QGAN [46]. In his study, he showed that normal quantization methods like (uniform, log, tanh) are not sufficient for a stable and convergent GAN in a very low-bit quantization. Moreover, generator (G) and discriminator (D) sensitivity to quantization is different. Though eventually, only the generator is needed, but finetuning it needs a balanced discriminator. Thus, quantizing both discriminator and generator leads to a convergent finetuning. This leads to proposing multi-precision quantization for G and D separately from each other. Then he used the expectation-maximization (EM) algorithm to find the optimal zero_point and scale_factor parameters from Eq. 9. Quantized weight is calculated from the Eq. 10. The EM algorithm tries to minimize the mean square error between the non-quantized weight and the quantized one.

$$W_q = \text{scale\_factor} * \text{round}(\text{quantized\_number}) + \text{zero\_point} \tag{10}$$

where scale_factor, quantized_number, zero_point are same parameters from Eq. 9. W_q is the quantized weight, round(quantized_number) is the integer part of the fixed-point number that will be used in the calculation.

QGAN has been applied to several GANs like DCGAN [15], WGAN [47], and LSGAN [48]. With a quantization to 1–4 bits, QGAN achieved a compression ratio from 8× up to 32× with a small loss in inception score.

A study made by Deng et al. using a PATCH-GAN-like generator to reconstruct face images showed that as the number of bits decreases, the peak signal-to-noise ratio (PSNR) gets worse while the memory footprint improves, which is not surprising [49]. In their study for quantized GAN for mobiles (QMGAN), they found that 32-bit representation

(single precision) has ∼2.5× improvement in PSNR over the 1-bit quantized (binary) network. However, the 1-bit quantized network has better memory size by 35× over single precision. They tried different values for quantization, what worth noting is that the PSNR of 32-bit is almost the same as the PSNR of 6-bits while the 6-bit has around 5.4× memory size improvement.

Haotao et al. continued their work on ganslimming [37] by applying both quantization and knowledge distillation. He used the uniform quantization on both activations and weights on the generator model. They unified the quantization on all layers so that it would be HW friendly. They performed 4×∼8× compression ratios on style-transfer problems with a very competitive result.

In [50], the authors of ApGAN used memory compression on a ReRam accelerator [4]. They quantized all weights and activations to 1-bit (sign-bit)which simplified MACs to just ANDing followed by ORing. This technique decreases the weights storage size. However, that comes at a cost of accuracy and speed of convergence. An experiment done by [50] compared fully-binarized DCGAN to full-precision, after 20 epochs the loss of binarized version is 3x the loss of full precision. For that reason, instead of binarizing all layers, they used variable layers quantization based on the data redundancy measure. Data redundancy measure is defined as $(c_i - h_i * w_i)$ where i is the layer number c_i is the number of channels of layer input i, and h_i, w_i are the input height and width respectively of the ith layer. A negative redundancy measure indicates a high sensitivity for quantization error; thus, it is not recommended to quantize such layers. Other layers with high redundancy measure are binarized by taking the sign bit of the weight and the average weight is considered the scale factor. Multiplication computations turns into just sign manipulation operation followed by scaling.

In [51], Rakin et al. proposed TGAN, a GAN that ternarize weights to {-1,0,1}. The ternarization depends on the sign of the weight like ApGAN. They applied the quantization schema on both generator and discriminator in the forward path and used the backward path to update the scale factor. They achieved on average ∼ 85% of IS of the full-precision network.

A work by Köster et al. proposed using non-standard numeric format called Flex-point [52]. The new format makes one shared exponent for each tensor; thus, the tensor operations are handled as if they were fixed-point operations, and an extra circuit is needed to manage exponent which is faster than the floating-point operation with different exponent for each number. Flex-point format stores the tensor as 16-bit mantissa for each element and one shared 5-bits exponent for the whole tensor. In contrast to floating point, the exponent is shared across tensor elements, and different from fixed point, the exponent is updated automatically every time a tensor is written. By implementing several networks using, the flex-point format, the FID score of those networks was comparable to the same networks implemented using float32 and better than the ones implemented with fixed-point or float16. Flex-point has the advantage of supporting training and applying all techniques of floating-point with even faster calculation given their newly Flex-point format.

Table 5 summarizes the design decisions taken by different works. Because each work uses a different network it is hard to make a fair comparison using metrics like IS or

[4] ReRam accelerator is processing in memory using analog crossbar circuit

Table 5 Summary of quantization works

Decision/work	Rep.	Func.	Granularity	Target	Module	App.
QGAN [46]	Int	EM	Network	W	G and D	Post
QMGAN [49]	Int	Uni	Network	W	G	Post
ApGAN [50]$^{+}$	Int	Uni	Layer	W	G and D	During
TGAN [51]$^{+}$	Int	Uni	Layer	W	G and D	During
Flexpoint [52]$^{+}$	Float	Cust.	Tensor	A and W	G and D	During
GANslim [37]	Int	Uni	Network	A and W	G	Post

Int integer-like, *Uni* uniform, *A* activation, + customized accelerator, *Cust* customized, *W* weight

FID. Figure 17 shows the number of bits that each quantization method reported for best results.

Encoding

It is a form of lossy compression used to minimize the data transfer using fewer bits. The data transferred to the chip is not the real data, but an index or key used to get the real data from a preloaded codebook or using a predefined hash function.

This technique has been extensively used in deep learning like in [53–57] . However, to the best of our knowledge, we found no work applied to GANs and it is one of the opportunities to seek.

Results and discussion

In this section, we will discuss the open opportunities and challenges for optimizing GANs. No doubt that minimizing the memory footprint enhances storage, speed, and power efficiency of running GANs, however there are some areas such as "encoding" that is not explored at all in GANs.

Another opportunity exists in optimizing GANs is to fill in the gaps and mix between different compression techniques. As seen from Tables 3, 4, and 5, a lot of combinations

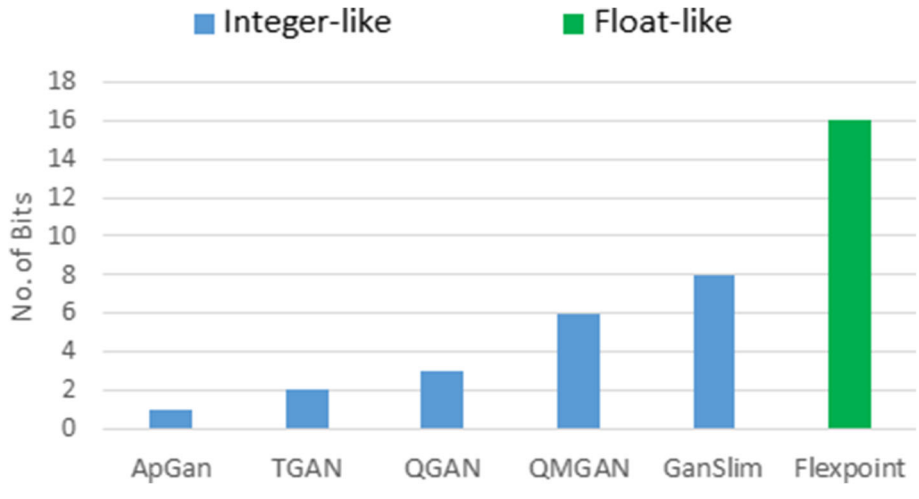

Fig. 17 No. of quantized bits that resulted in the best performance for the GAN undertest. Each technique is tested on different GAN. The works QGAN [46], QMGAN [49], GanSlim [37], Flexpoint [52], and TGAN [51] are fully quantized. While ApGan [50] has mixed quantized and non-quantized layers. Also, FlexPoint, ApGAN and TGAN quantization is used during training

can still be explored like using similarity-pruning with knowledge distillation. In addition, combining losses introduced in knowledge distillation with quantization needs more exploration.

A challenge that exists with all the mentioned optimization techniques is how to unify measures (both qualitative and quantitative) and benchmark between several methods. With no unified measures, benchmarks, and platforms, we can hardly evaluate techniques compared to each other.

While compression is very plausible, some techniques are not hardware friendly though they give high compression ratio and accuracy. An opportunity exists in enhancing indexing methods in accelerators or building a cache like system to support clustering or hashing-based quantization. Also, the support for processing compressed elements or quantized element should be considered without uncompressing them.

Optimizing GANs is not limited to GAN compression, on the other opposite, compression is just one technique. Other optimization methods like computational optimization and dataflow optimization are still open areas for exploration.

We can summarize the possible future work as follows:

- Unify the design metrics between different designs and provide an evaluation study using several dataset.
- Explore the missing areas like encoding in GANs.
- Study systems with combinations of optimization techniques.
- Study the impact of optimization on different platform (FPGA, ASIC, RERAMs, GPUS, etc.)
- Explore computational optimization.
- Explore dataflow optimization.

Conclusion

In this paper, we surveyed the lossy compression techniques used to optimize GANs. GANs differ from other DL models because of training instability and the high output resolution. Those challenges make GANs optimization more challenging.

We divided lossy compression techniques to five categories: (a) pruning, (b) knowledge distillation, (c) low-rank factorization, (d) lowering numeric precision, and (e) encoding. Those techniques can be overlapped with each other to produce more robust model after compression.

By surveying several works, it is found that pruning alone is not enough to preserve the model quality, thus why knowledge distillation is combined with pruning for better results. Knowledge distillation is a very powerful technique to handle the training instability as it simplifies the problem from satisifying a general quality criteria to mimicing the teacher model.

Lowering numeric precision is a commonly used technique as can be called "quantization." It has a wide varity of options depending on the hardware support available. Processing in memory is best used with binary and trenary data. Finetuning is required with lowering numeric precision.

Low-rank factorization and encoding techniques are not well explored in the domain of GANs which needs further investigation. Additionally, the combination of several techniques needs to be explored as well.

Our contributions can be summarized in being the first paper dedicated to survey GAN compression techniques, providing a taxonomy for the compression techniques, summarizing recent work in this area and providing open research areas for exploration in accelerating GANs.

Abbreviations
GANs: Generative adversarial networks; DL: Deep learning; CNN: Convolution neural network; IS: Inception score; FID: Fréchet inception distance; PSNR: Peak signal-to-noise ratio; mPA: Mean pixel accuracy; IoU: Intersection over union; MACs: Multiply-accumulate operations; CR: Compression ratio; G: Generator; D: Discriminator

Acknowledgements
I would like to thank Eman Hosam and Mayada Hadhoud for their valuable contributions in proof reading, language editing, and article organization.

Authors' contributions
This review article is made during internship in NYU by Dina Tantawy under the supervision of Prof. Mohamed Zahran from NYU and Prof. Amr Wassal from Cairo University. The authors read and approved the final manuscript.

Funding
No funding was provided for this work.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable as it has no human or animal participants.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests beyond their current affliations.

Received: 3 September 2021 Accepted: 11 November 2021
Published online: 19 December 2021

References

1. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in Neural Information Processing Systems. pp 2672–2680
2. Bollepalli B, Juvela L, Alku P (2019) Generative adversarial network-based glottal waveform model for statistical parametric speech synthesis. arXiv preprint arXiv:1903.05955
3. Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X, Metaxas D (2017) Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. Proc IEEE Int Conf Comp Vision:5907–5915
4. Zhang Z, Xie Y, Yang L (2018) Photographic text-to-image synthesis with a hierarchically-nested adversarial network. Proc IEEE Conf Comput Vis Pattern Recognit:6199–6208
5. Choi Y, Choi M, Kim M, Ha J-W, Kim S, Choo J (2018) Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. Proc IEEE Conf Comput Visi Pattern Recognit:8789–8797
6. Wang T-C, Liu M-Y, Zhu J-Y, Tao A, Kautz J, Catanzaro B (2018) High-resolution image synthesis and semantic manipulation with conditional gans Vol. Proc IEEE Conf Comput Visi Pattern Recognit. pp 8798–8807
7. Engel J, Agrawal K, Chen S, Gulrajani I, Donahue C, Roberts A (2019) Gansynth: Adversarial neural audio synthesis. arXiv preprint arXiv:1902.08710
8. Clark A, Donahue J, Simonyan K (2019) Efficient video generation on complex datasets. arXiv preprint arXiv:1907.06571
9. Cheng Y, Wang D, Zhou P, Zhang T (2017) A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282
10. Cheng Y, Wang D, Zhou P, Zhang T (2018) Model compression and acceleration for deep neural networks: The principles, progress, and challenges. IEEE Signal Proc Mag 35(1):126–136
11. Cheng J, Wang P, Li G, Hu Q, Lu H (2018) Recent advances in efficient computation of deep convolutional neural networks. Front Inform Technol Electron Eng 19:64–77
12. Choudhary T, Mishra V, Goswami A, Sarangapani J (2020) A comprehensive survey on model compression and acceleration. Artif Intell Rev 53(7):5113–5155
13. Gou J, Yu B, Maybank S, Tao D (2021) Knowledge distillation: A survey. Int J Comput Vis 129(6):1789–1819
14. Wang L, Yoon K-J (2021) Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. IEEE Trans pattern Anal Mach Intell PP
15. Gao F, Yang Y, Wang J, Sun J, Yang E, Zhou H (2018) A deep convolutional generative adversarial networks (dcgans)-based semi-supervised method for object recognition in synthetic aperture radar (sar) images. Remote Sens 10(6):846

16. Isola P, Zhu J-Y, Zhou T, Efros A (2017) Image-to-image translation with conditional adversarial networks. Proc IEEE Conf Comput Vis Pattern Recognit:1125–1134
17. Salimans T, Goodfellow I, Zaremba W, Cheung V, Radford A, Chen X (2016) Improved techniques for training gans. Adv Neural Inf Process Syst:2234–2242
18. Heusel M, Ramsauer H, Unterthiner T, Nessler B, Hochreiter S (2017) Gans trained by a two time-scale update rule converge to a local nash equilibrium. Adv Neural Inf Process Syst:6626–6637
19. Vuduc R (2003) Automatic Performance Tuning of Sparse Matrix Kernels. University of California, Berkeley
20. Gilbert J, Moler C, Schreiber R (1992) Sparse matrices in matlab: Design and implementation. SIAM J Matrix Anal Appl 13(1):333–356
21. Knuth D (1985) Dynamic huffman coding. J Algorithms 6(2):163–180
22. Bradley S (1969) Optimizing a scheme for run length encoding. Proce IEEE 57(1):108–109
23. Yu C, Pool J (2020) Self-supervised gan compression. arXiv:2007.01491v2
24. Song X, Chen Y, Feng Z-H, Hu G, Yu D-J, Wu X-J (2020) Sp-gan: Self-growing and pruning generative adversarial networks. IEEE Trans Neural Netw Learn Syst 32(6):2458–2469
25. Shu H, Wang Y, Jia X, Han K, Chen H, Xu C, Tian Q, Xu C (2019) Co-evolutionary compression for unpaired image translation. Proc IEEE/CVF Int Conf Comput Vis:3235–3244
26. Zhu J-Y, Park T, Isola P, Efros A (2017) Unpaired image-to-image translation using cycle-consistent adversarial networks. Proc IEEE Int Conf Comput Vis:2223–2232
27. Aguinaldo A, Chiang P-Y, Gain A, Patil A, Pearson K, Feizi S (2019) Compressing gans using knowledge distillation. arXiv preprint arXiv:1902.00159
28. Li M, Lin J, Ding Y, Liu Z, Zhu J-Y, Han S (2020) Gan compression: Efficient architectures for interactive conditional gans. Proc IEEE/CVF Conf Comput Vis Pattern Recog:5284–5294
29. Cai H, Gan C, Wang T, Zhang Z, Han S (2019) Once-for-all: Train one network and specialize it for efficient deployment. arXiv preprint arXiv:1908.09791
30. Fu Y, Chen W, Wang H, Li H, Lin Y, Wang Z (2020) Autogan-distiller: Searching to compress generative adversarial networks. arXiv preprint arXiv:2006.08198
31. Liu H, Simonyan K, Yang Y (2018) Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055
32. Wang X, Yu K, Wu S, Gu J, Liu Y, Dong C, Qiao Y, Change Loy C (2018) Esrgan: Enhanced super-resolution generative adversarial networks. Proc Eur Conf Comput Vis (ECCV) Workshops
33. Jin Q, Ren J, Woodford O, Wang J, Yuan G, Wang Y, Tulyakov S (2021) Teachers do more than teach: Compressing image-to-image models. arXiv preprint arXiv:2103.03467
34. Chen H, Wang Y, Shu H, Wen C, Xu C, Shi B, Xu C, Xu C (2020) Distilling portable generative adversarial networks for image translation. Proc AAAI Conf Artif Intell 34:3585–3592
35. Liu Y, Shu Z, Li Y, Lin Z, Perazzi F, Kung S (2021) Content-aware gan compression. arXiv preprint arXiv:2104.02244
36. Zhang Z, Chen S, Sun L (2020) P-kdgan: Progressive knowledge distillation with gans for one-class novelty detection. arXiv preprint arXiv:2007.06963
37. Wang H, Gui S, Yang H, Liu J, Wang Z (2020) Gan slimming: All-in-one gan compression by a unified optimization framework. In: European Conference on Computer Vision. Springer, Cham. pp 54–73
38. Liu Z, Li J, Shen Z, Huang G, Yan S, Zhang C (2017) Learning efficient convolutional networks through network slimming. Proc IEEE Int Conf Comput Vis:2736–2744
39. Jaderberg M, Vedaldi A, Zisserman A (2014) Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866
40. Swaminathan S, Garg D, Kannan R, Andres F (2020) Sparse low rank factorization for deep neural network compression. Neurocomputing 398:185–196
41. Bouwmans T, Aybat N, Zahzah E-h (2016). CRC Press
42. Abadi M, Agarwal A, Barham P, et. al. (2015) TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/
43. Paszke A, Gross S, Massa F, et.al. (2019) Pytorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d♪ Alché-Buc F, Fox E, Garnett R (eds). Advances in Neural Information Processing Systems 32. Curran Associates, Inc., Vancouver. pp 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
44. Wu Q, Liu Y, Miao C, Zhao B, Zhao Y, Guan L (2019) Pd-gan: Adversarial learning for personalized diversity-promoting recommendation. IJCAI 19:3870–3876
45. Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proc IEEE Conf Comput Vis Pattern Recog:2704–2713
46. Wang P (2019) "QGAN: Quantized Generative Adversarial Networks". "arXiv preprint"
47. Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V, Courville A (2017) Improved training of wasserstein gans. arXiv preprint arXiv:1704.00028
48. Mao X, Li Q, Xie H, Lau R, Wang Z, Paul Smolley S (2017) Least squares generative adversarial networks. Proc IEEE Int Conf Comput Vis:2794–2802
49. Deng A, Looi W, Tsun A (2019) Quantized GANs for Mobile Image Reconstruction
50. Roohi A, Sheikhfaal S, Angizi S, Fan D, DeMara R (2019) Apgan: Approximate gan for robust low energy learning from imprecise components. IEEE Trans Comput 69(3):349–360
51. Rakin A, Angizi S, He Z, Fan D (2018) Pim-tgan: A processing-in-memory accelerator for ternary generative adversarial networks. In: 2018 IEEE 36th International Conference on Computer Design (ICCD). IEEE, Orlando. pp 266–273
52. Köster U, Webb T, Wang X, Nassar M, Bansal A, Constable W, Elibol O, Gray S, Hall S, Hornof L, et al (2017) Flexpoint: An adaptive numerical format for efficient training of deep neural networks. Adv Neural Inf Process Syst:1742–1752
53. Han S, Mao H, Dally W (2015) A deep neural network compression pipeline: Pruning, quantization, huffman encoding. arXiv preprint arXiv:1510.00149 10
54. Ullrich K, Meeds E, Welling M (2017) Soft weight-sharing for neural network compression. arXiv preprint arXiv:1702.04008

55. Chen W, Wilson J, Tyree S, Weinberger K, Chen Y (2015) Compressing neural networks with the hashing trick. Int Conf Mach Learning:2285–2294
56. Zhu J, Qian Z, Tsui C-Y (2017) Bhnn: A memory-efficient accelerator for compressing deep neural networks with blocked hashing techniques. In: 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, Japan. pp 690–695
57. Eban E, Movshovitz-Attias Y, Wu H, Sandler M, Poon A, Idelbayev Y, Carreira-Perpinan M (2019) Structured multi-hashing for model compression. arXiv preprint arXiv:1911.11177

Publisher's Note