Evaluating Post-Training Compression in GANs using Locality-Sensitive Hashing

Gonçalo Mordido
Hasso Plattner Institute
Potsdam, Germany
goncalo.mordido@hpi.de

Haojin Yang
Hasso Plattner Institute
Potsdam, Germany
haojin.yang@hpi.de

Christoph Meinel
Hasso Plattner Institute
Potsdam, Germany
christoph.meinel@hpi.de

Abstract

The analysis of the compression effects in generative adversarial networks (GANs) after training, i.e., without any fine-tuning, remains an unstudied, albeit important, topic with the increasing trend of their computation and memory requirements. While existing works discuss the difficulty of compressing GANs during training, requiring novel methods designed with the instability of GANs training in mind, we show that existing compression methods (namely clipping and quantization) may be directly applied to compress GANs post-training, without any additional changes. High compression levels may distort the generated set, likely leading to an increase of outliers that may negatively affect the overall assessment of existing k-nearest-neighbor (KNN) based metrics. We propose two new precision and recall metrics based on locality-sensitive hashing (LSH), which, on top of increasing the outlier robustness, decrease the complexity of assessing an evaluation sample against n reference samples from $O(n)$ to $O(\log(n))$, if using LSH and KNN, and to $O(1)$, if only applying LSH. We show that low-bit compression of several pre-trained GANs on multiple datasets induces a trade-off between precision and recall, retaining sample quality while sacrificing sample diversity.

1. Introduction

Existing compression techniques have been shown to negatively affect Generative Adversarial Networks (GANs [11]) during training [31, 6, 35, 36, 19], often leading to low sample quality and/or low sample diversity (commonly referred to as mode collapse). Yu and Pool [36] suggested the high-entropy in the generator ($G$) input and output as well as the inherent training instability of GANs [26, 27, 30] as the main causes for this ineffectiveness.

In this work, we study the effects of post-training compression in pre-trained GANs, without the need for any retraining fine-tuning. This is important when there is a lack of computational power or time to fine-tune a compressed model, as well as inaccessibility to the training data or training details used to train the original model. Our main goals are two-fold: (1) analyze if current compression techniques, such as linear quantization, pruning [24], splitting [38] and clipping [2], may be successful applied to compress GANs in a post-training regime, and (2) separately assess the detriment in terms of quality and diversity of the generated set.

Objective evaluation metrics that separately assess the sample quality and sample diversity in terms of precision and recall, respectively, have been recently proposed [28, 18, 22, 23]. Common methods use k-nearest neighbors (KNN) to obtain a topological representation, i.e., approximated manifolds, of both the reference and evaluation set, consisting of real and generated samples. Although simple to implement, KNN-based metrics have linear complexity and are sensitive to outliers in both the reference and evaluation sets, which may compromise the overall assessment.

In our use-case, it is especially important to deal with outliers since high compression rates may distort the generated set. To this end, we propose to approximate the KNN manifold by using locality-sensitive hashing (LSH). In particular, we use random hyperplanes to divide the data space into different regions. Then, we only calculate KNN on points that are in the same region. The computation may be improved at the cost of a poorer approximation by not applying KNN at all and considering entire regions as manifolds. Since outliers are likely to be isolated in a certain region, they are less likely to corrupt the evaluation process (see Figure 1).

On another note, we improve the computational complexity of assessing an evaluation sample against $n$ reference samples of current KNN-based metrics from $O(n)$ to $O(\log(n))$, when applying KNN on top of LSH, and to $O(1)$, if only using LSH. Improving the complexity in order of $n$ is of extreme importance since increasing $n$ is likely to lead to a more accurate assessment in all evaluation methods.

We find that existing compression techniques, such as clipping and quantization, may be successfully applied to pre-trained GANs, providing a trade-off between precision and recall: while sample quality is mostly retained, sample diversity is majorly affected.
2. Related work

Several recent methods propose to alleviate the issues of compressing GANs during training. Shu et al. [31] used evolutionary algorithms to perform channel pruning and construct a compressed $G$. Aguinaldo et al. [1] used knowledge distillation [13] to train a compressed student $G$ based on a larger teacher $G$ trained with the discriminator ($D$) feedback. Chen et al. [6] added a student $D$ to the distilling framework allowing an increase of student $G$'s compression levels. Koratana et al. [17] used learned intermediate representation training (LIT) to learn only a student $G$, without updating a pre-trained teacher $G$. Wang et al. [35] proposed a new compression scheme based on Expectation-Maximization (EM) algorithms to compress both $G$ and $D$ during training. Liu et al. [20] suggested quantizing more layers in $D$ than $G$ to stabilize training. Yu et al. [36] proposed a self-supervised compression mechanism that combines the feedback from a pre-trained $D$ as well as the original $G$ to train a pruned $G$. Wang et al. [34] combined model distillation, channel pruning, and quantization on top of the original minimax objective to compress GANs in an end-to-end manner. Li et al. [19] proposed a general framework to compress generators in conditional GANs by combining knowledge distillation and neural architecture search [10].

One big shortcoming of previous works is in the evaluation of the generated samples using a single-valued metric, namely FID [12] and Inception Score [29]. This is sub-optimal since no separate assessment of the sample quality and sample diversity was performed. Several precision and recall metrics have been recently proposed [28, 18, 22, 23]. In this work, we show that common single-valued metrics, such as FID and KID [4] tend to correlate more with recall than precision, biasing against models with high sample quality but low sample diversity. This reinforces the importance of using double-valued metrics that separately assess the quality and diversity of the generated set. Hence, we propose two new evaluation metrics that evaluate precision and recall with LSH and KNN. Our metrics are outlier-aware and more efficient than current KNN-based approaches.

To the best of our knowledge, we are the first work to study the effects of post-training compression in GANs, which enable the compression of pre-trained GANs without the need of additional training. Since we do not have to deal with the training instability of GANs, we show that current quantization methods may be directly applied to compress a pre-trained $G$ to low bit-width.

Figure 1: Evaluation methods. Considering a set of real and generated points (a), we split the data space into several regions by generating random hyperplanes (e). Each region is characterized by which side it lies regarding each hyperplane. In our LSH metric, we consider points inside the same region, i.e. manifold, to calculate precision and recall (b, f). In our LSH + KNN metric, we consider the hyperspheres of nearest points (K=1 in this example) within each region as the approximated manifold (c, g). Existing KNN approaches [18], consider all points to generate hyperspheres (d, h), which may have undesirable effects in dealing with outliers (h). Our methods help mitigate this by likely inserting outliers in isolated regions (f, g).
3. Locality-sensitive hashing (LSH)

As briefly mentioned, using solely KNN [18] to evaluate a generated set has several shortcomings. The more meaningful of which is the inability to deal with outliers in the reference and evaluation sets, which may greatly distort the approximation of manifolds and lead to an inaccurate assessment of precision and/or recall. The aforementioned distortion in the approximated manifolds may also occur with sparse data, where points are sparsely distributed and manifolds may cover most of the data space. Lastly, KNN-based approaches have linear complexity on the number of reference samples, which is sub-optimal and expensive at evaluation time.

One common way of improving the efficiency of KNN is to either reduce the data dimensionality or only compare each sample to a subset of samples. In this work, we focus on the latter. Several algorithms may be used to achieve this. Specifically, K-D trees [3] are a viable alternative for low-dimensional, continuous data, with the downside of possibly missing nearest neighbors. Inverted lists are commonly used for high-dimensional, discrete data, providing an efficient alternative without the risk of not detecting nearest neighbors.

However, neither K-D trees nor inverted lists are good alternatives in our use-case due to the nature of our data. As initially proposed by Kynkäänniemi et al. [18], each image is passed as input to a pre-trained VGG-16 model [32], being represented as the embeddings of the second fully connected layer. Hence, our data is high-dimensional and continuous, handled high-dimensional, continuous (or discrete) data, being the most suited alternative. Similar to K-D trees, LSH may miss nearest neighbors, but this may act as a prevention mechanism against outliers, as previously mentioned.

In particular, LSH considers only the subset of points that are within the same region as an evaluation point. The goal of LSH is then to generate identical hashes for points that are nearby, storing that information efficiently in a hash table. A simple, yet effective way of generating each key is by using random projections [33], as described next.

3.1. Hash generation with random projections

To simulate the random projections, we generate $H$ random hyperplanes, with each hyperplane consisting of a random vector $h \sim \mathcal{N}_d(0, 1)$, where $d$ is the data dimension, and a random variable $b \sim U(0, 1)$ in the form of:

$$h_0 x_0 + \ldots + h_{d-1} x_{d-1} + b = 0. \quad (1)$$

Each bit of a $d$-dimensional point $\phi$’s key is the sign of the random projection with each random hyperplane, obtained by a dot product:

$$\text{hash}_{h,b}(\phi) = \begin{cases} 1, & \text{if } \text{sign}(\phi \cdot h + b) \geq 0 \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

In other words, we are checking on which "side" of a random hyperplane the point $\phi$ lies on. The final $\phi$’s key consists of $H$ bits, obtained by comparing $\phi$ with all $H$ random hyperplanes:

$$\text{key}_H(\phi) = \text{hash}_{h_0,b_0}(\phi) \ldots \text{hash}_{H-1,b_{H-1}}(\phi), \quad (3)$$

where $h_0, \ldots, h_H$ and $b_0, \ldots, b_H$, are independently generated and each bit/hash is computed in parallel.

We store this information in an initialized hash table $HT_{\Phi}$, which is populated by iterating through a set $\Phi$:

$$\sum_{\phi \in \Phi} HT_{\Phi}[\text{key}_H(\phi)].\text{append}(\phi). \quad (4)$$

3.2. Assessing the quality and diversity of a set

We propose two novel metrics to evaluate the quality and diversity of a generated set in terms of precision (P) and recall (R), respectively. The simplest of which solely uses LSH (Section 3.2.1), calculating the ratio of evaluation samples with at least one reference sample in their region. Moreover, we propose an additional metric that uses LSH and KNN (Section 3.2.2), calculating the ratio of evaluation samples inside at least one reference sample’s hypersphere in their region.

3.2.1 LSH metric

Consider the binary function $f_{LSH}(\text{key},HT_{\Phi})$, which returns if a key is in the list of keys of a populated hash table $HT_{\Phi}$:

$$f_{LSH}(\text{key},HT_{\Phi}) = \begin{cases} 1, & \text{if key } \in HT_{\Phi} \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

Given two sets $\Phi_a$ and $\Phi_b$, we assess quality in terms of precision:

$$P_{LSH}(\Phi_a, \Phi_b) = \frac{1}{|\Phi_b|} \sum_{\phi_b \in \Phi_b} f_{LSH}(\text{key}_H(\phi_b),HT_{\Phi_a}). \quad (6)$$

and diversity in terms of recall:

$$R_{LSH}(\Phi_a, \Phi_b) = \frac{1}{|\Phi_a|} \sum_{\phi_a \in \Phi_a} f_{LSH}(\text{key}_H(\phi_a),HT_{\Phi_b}). \quad (7)$$
3.2.2 LSH + KNN metric

To ease method comparisons, we follow Kynkäänniemi et al. [18]’s formulations and describe a binary function that indicates if a given point $\phi$ is located within the volume formed by all the hyperspheres of each point $\phi' \in \Phi$ and its $k$-nearest point in the set $\Phi$, returned by $\text{NN}_k(\phi', \Phi)$:

$$f_{\text{LSH+KNN}}(\phi, \Phi) = \begin{cases} 1, & \text{if } \|\phi - \phi'\|_2 \leq \|\phi - \text{NN}_k(\phi', \Phi)\|_2 \text{ for at least one } \phi' \in \Phi \\ 0, & \text{otherwise.} \end{cases}$$

(8)

Note that, if the number of neighbors is equal or higher than the number of samples in a given region, i.e. $k \geq |\Phi|$, then the farthest point from $\phi'$ in its region is retrieved by $\text{NN}_k(\phi', \Phi)$.

Given two sets $\Phi_a$ and $\Phi_b$, we assess quality in terms of precision:

$$P_{\text{LSH+KNN}}(\Phi_a, \Phi_b) = \frac{1}{|\Phi_b|} \sum_{\phi_b \in \Phi_b} f_{\text{LSH+KNN}}(\phi_b, \text{HT}_{\Phi_a}(\text{key}_H(\phi_b))),$$

(9)

and diversity in terms of recall:

$$R_{\text{LSH+KNN}}(\Phi_a, \Phi_b) = \frac{1}{|\Phi_a|} \sum_{\phi_a \in \Phi_a} f_{\text{LSH+KNN}}(\phi_a, \text{HT}_{\Phi_b}(\text{key}_H(\phi_a))).$$

(10)

3.3. Assessing the realism of individual samples

Kynkäänniemi et al. [18] further proposed a realism score ($RS$) to calculate the quality of a given sample. Such metric can be modified and be applied to our LSH + KNN metric by:

$$RS(\phi_a, \Phi_b) = \max_{\phi_b} \left\{ \frac{\|\phi_b - \text{NN}_k(\phi_b, \text{HT}_{\Phi_b}(\text{key}_H(\phi_b)))\|_2}{\|\phi_a - \phi_b\|_2} \right\}.$$  

(11)

Once again, the KNN is performed only with samples in $\Phi_b$ that are in the same region as $\phi_b$. More concretely, $RS \geq 1$ means that $\phi_a$ is inside the hypersphere of at least one $\phi_b$ in its region.

Notably, Kynkäänniemi et al. [18] tried to mitigate the aforementioned outlier problem and computed this individual score by discarding half the samples in $\Phi_b$ with the largest radii. Even though this was a step in the right direction, no samples were discarded on their calculation of precision and recall when evaluating a sample set.

| Evaluation method | Computational cost | Complexity in $n$ |
|-------------------|-------------------|------------------|
| LSH (ours)        | $d \times H$      | $O(1)$           |
| LSH + KNN (ours)  | $d \times H + d \times n/2^H$ | $O(\log(n)), H \sim \log(n)$ |
| KNN [18]          | $d \times n$      | $O(n)$           |

Table 1: Comparison of the different methods in assessing a $d$-dimensional evaluation sample against $n$ reference samples. $H$ denotes the number of random hyperplanes.

3.4. Complexity analysis

Calculating the of a $d$-dimensional point costs $d \times H$, since $H$ dot products are required, i.e. one for each random hyperplane. By generating $H$ random hyperplanes, we are splitting the data space into $\approx 2^H$ regions. Hence, on average, we will have $n/2^H$ points inside each region. Moreover, the comparison cost inside each region is, on average, $d \times n/2^H$. Note that, if we set $H \sim \log(n)$, we reduce the comparison cost to $d$, becoming constant in terms of $n$. Table 1 summarizes the complexities in order of $n$ of the different evaluation methods. Figure 2 shows how the methods scale on practical hardware.

4. Compression techniques

Wang et al. [35] studied the effects of linear quantization schemes while training GANs. In our work, we study different compression techniques other than quantization, such as pruning, splitting, and clipping while compressing GANs post-training and without any retraining or fine-tuning. Zhao et al. [38] performed similar studies for image classification networks. Here, we extrapolate such studies

1. https://github.com/cornell-zhang/dnn-quant-ocs
to the GANs domain, compressing the weights of several pre-trained generators on multiple datasets.

4.1. Quantization (Q)

Our linear quantization scheme (LQ) simply maps the floating-point weights to their closest discrete value covering their symmetric dynamic range:

$$LQ(x) = \frac{x(2^{b-1} - 1)}{\max(|x|)} \times \frac{\max(|x|)}{2^{b-1} - 1},$$

(12)

where b represents the number of bits. This is considered one of the most straightforward quantization schemes. Since it uses the full dynamic range of the original weights, LQ may be sensitive to outlier weights, which may hurt the final weight approximation.

4.2. Pruning and quantization (P + Q)

Mordido et al. [24, 25] leveraged Monte Carlo techniques to perform pruning and quantization in one algorithm called Monte Carlo Quantization (MCQ). In sum, they use the probability density function (PDF) of the floating-point weight distribution to generate a cumulative density function (CDF). Then, they perform importance sampling on the CDF using N uniformly distributed samples with the assigned discrete value for each weight being the number of times it got hit during the sampling process. Weights that are not hit, may then be pruned at the end.

The bit-width and pruning levels are controlled by N, i.e. the amount of sampling performed. Hence, more sampling likely leads to a better approximation of the original weights at the cost of higher bit-width and fewer pruned weights.

4.3. Splitting and quantization (S + Q)

Outlier Channel Splitting (OCS) [38] proposed to improve the approximation of the original weights by duplicating outlier weights and halving their values. Concretely, outlier weights are moved closer to the center of the distribution, reducing the dynamic range and improving the approximation of the original weights when applying linear quantization after the splitting process. Hence, this method introduces a new trade-off between quantization loss and final network size. Based on the reported results in their original paper, we used an expand ratio of 0.01 in our experiments.

4.4. Clipping and quantization (C + Q)

Analytical Clipping for Integer Quantization (ACIQ) [2] clips the original weight distribution by approximating the optimal clip threshold that minimizes the mean-squared-error between the continuous and discrete weight distributions. To achieve this, it uses the statistics of a Gaussian or a Laplacian distribution, depending on which one is closer to the original weight distribution. Due to its clipping nature, ACIQ improves the final approximation by increasing outlier distortion. Inspired by Zhao et al. [38], we applied linear quantization on top of clipping in our experiments.

5. Experiments

We experimented compressing GANs pre-trained on several datasets: Flickr-Faces-HQ (FFHQ [15]), a high-quality image dataset of human faces, Animal Faces-HQ (AFHQ [8]), a high-quality image dataset of cats, dogs, and wild life faces, and ImageNet [9], the large-scale dataset organized according to the WordNet hierarchy. We used different resolutions for each dataset: 1024x1024 and 256x256 for FFHQ, 512x512 for AFHQ, and 128x128 for ImageNet.

Moreover, we tested compressing the weights of different generators of popular GANs. Namely, PA-GAN [37] applies progressive augmentation in D’s input during training to reduce overfitting. zCR-GAN [39] uses latent consistency regularization by augmenting G’s input with additional small-magnitude noise, stimulating diversity in the generated set. SN-GAN [21] applies spectral normalization to D’s weights to stabilize training. SS-GAN [7] adds an auxiliary rotation loss for the G and D, combining adversarial training and self-supervision. StyleGAN2 [16] improves its predecessor StyleGAN [15] by revamping G’s normalization, progressive growing and regularization. ADA [14] introduces adaptive discriminator augmentation to promote learning with limited data. BigGAN [5] applies orthogonal regularization to G to train class-conditional GANs with large-scale data while controlling the trade-off between sample quality and diversity. All of the used pre-trained generators are publicly available ²³⁴.

To alleviate possible negative compression effects, we do not compress the parameters in the last residual block of each network. However, these layers only hold a small percentage (≈ 4 to 6%) of the total number of parameters, depending on the network architecture and dataset resolution. We use k = 3, as originally proposed by Kynkäänniemi et al. [18], and \( H = \lfloor \log(n) \rfloor \), as discussed in Section 3.4, in all of our experiments.

5.1. Mean images

We will first study the effects that the different compression schemes, i.e. quantization, pruning, splitting, and clipping, have on the different models and datasets. Overall, in low-bit settings (2 to 4 bits), combining clipping and quantization is better than the alternatives, enabling the highly compressed models to still generate realistic samples (see Figures 3 and 4). This suggests a high tolerance to outlier weight distortion in G induced by clipping.

² https://tfhub.dev/deepmind/biggan-128/2
³ https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada/pretrained/
⁴ https://nvlabs-fi-cdn.nvidia.com/stylegan2/networks/
settings (5 to 6 bits), all of the studied compression techniques are able to successfully compress the different models, with none of them showing no apparent loss of performance at 6 bits. Mean images for the different models on FFHQ 256x256 share similar conclusions and are provided in the appendix.

One interesting observation in the mean generated images when applying clipping and quantization (C+Q) at low bit-widths, is that they still resemble faces both on the FFHQ and AFHQ datasets. This suggests that the generated set may be losing diversity, due to the average face being different than the 32-bit baselines, but still maintaining some of its original quality. We will now take a deeper diving into analyzing the effects of clipping and quantization. (Results for the rest compression techniques are in the appendix.)

5.2. Generated images

We may also analyze how the pre-trained GANs trained on FFHQ 256x256 differ from one another by visualizing their mean images (Figure 5). Looking at the mean 32-bit image of the different models, we observe that all models successfully learn how to generated faces with similar attributes. However, at 2 bits, we start seeing differences between the mean images, which different models generating different types of face attributes, such as elder (Figure 5e), longer hair (Figures 5a and 5f), darker hair (Figure 5b and 5d), and male (Figure 5c).
2 bits 32 bits 2 bits 32 bits 2 bits 32 bits

(a) AFHQ Cat.  (b) AFHQ Dog.  (c) AFHQ Wild.

Figure 6: ADA on AFHQ 512x512 (C + Q).

2 bits 32 bits 2 bits 32 bits 2 bits 32 bits

(a) ADA.  (b) StyleGAN2.  (c) zCR-GAN.

(d) SN-GAN.  (e) PA-GAN.  (f) SS-GAN.

Figure 7: FFHQ 256x256 (C + Q).

32 bits 6 bits 5 bits 4 bits 3 bits 2 bits

Figure 8: BigGAN on ImageNet 128x128 (C + Q).

seeds (transformations at higher bits are in the appendix). At 2 bits, we notice a general lack of diversity in the generated set, however, all models are still able to produce recognizable samples retaining some of their original quality.

Figure 8 shows generated images from BigGAN on ImageNet. As we increase the compression levels, objects tend to get increasingly distorted while starting to share similar textures. (Generated images of all models using the other compression techniques can be found in the appendix.)

5.3. Evaluation metrics

In terms of single-valued metrics, we evaluate our compressed models with FID [12] and KID [4] which are commonly used to evaluate GANs. In particular, we use KID to evaluate our AFHQ models, since it has been shown to be less biased than FID at evaluating small datasets [14]. We evaluate FID for the models trained on the rest of the datasets.

Separate notions of quality and diversity allow us to determine if they are affected differently with compression. Hence, we test all models using our metrics, i.e. LSH and LSH + KNN, as well as the original KNN-based precision and recall metric proposed by Kynkänneni et al. [18].

Results of all models compressed to 4 to 2 bits are shown in Tables 2 and 3. Most 4 to 3-bit compressed models are able to produce samples with similar quality as the 32-bit baseline, showing higher precision scores across the board. On the other hand, recall rarely improves, indicating a loss of sample diversity. Similarly, FID and KID always deteriorate with compression and do not reflect the quality retention in the generated samples from compressed models.

5.4. Correlations with precision and recall

The degradation of the FID and KID scores with increased compression levels incentivized us to study their individual correlations with precision and recall. To increase the number of values when calculating the Pearson
and Spearman correlations, we use the scores of compression from 6 to 2 bits for each metric. Whereas Pearson directly uses the score values to evaluate their correlation in terms of linear relationships, Spearman uses the ranks of the scores, assessing their correlation in terms of monotonic relationships. Correlation values closer to $-1$ are ideal both for Pearson and Spearman since an inverse trend between FID/KID scores and precision/recall scores is expected, i.e. as one increases, the other should decrease, and vice-versa.

Figures 9 and 10 show the correlation results for FID and KID, respectively. We observe that the absolute Pearson correlations are high both for precision and recall. However, absolute Spearman correlations are high for recall but show close to no correlation for precision, in most cases. This suggests that FID and KID scores reflect more sample diversity than sample quality, being biased against high quality but low diversity models, i.e. mode collapsed.

5.5. Pareto frontiers

If we look at quality or diversity dominant GANs, do they become more balanced after compression? To try to answer this question we took a look at how compressed StyleGAN2 compares against other StyleGAN2 configuration variants originally proposed by Karras et al. [16]. Their discussed variants consist in modifications to the original StyleGAN [15] by applying weight demodulation (config-b), lazy regularization (config-c), path length regularization (config-d), new generator and discriminator architectures (config-e), and larger networks, i.e. StyleGAN2. Compared to the other variants, StyleGAN2 improves recall at the cost of sacrificing precision when evaluating with KNN.

In Figure 11, we observe that compression in StyleGAN2 improves the precision and recall trade-off relative to the rest of the variants, improving the previous Pareto frontier and achieving higher precision than the other configurations at similar recall levels. This suggests that compression may be used to balance out quality or diversity-dominant GANs.

6. Conclusion

With this work, we showed post-training compression in GANs is a viable way of balancing the trade-off between sample quality and sample diversity, on top of reducing the generator’s size. In particular, clipping and linear quantization techniques retain the quality of the generated set at the cost of some diversity, which has been an active research topic in current GANs research. Hence, depending on the use-case, compression may act as a simple, yet effective measure to balance existing GANs post-training, i.e. without requiring much time and computation. In the future, we plan to further optimize GANs inference by also compressing activations using the studied compression techniques.
References

[1] Angeline Aguinaldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. Compressing gans using knowledge distillation. arXiv preprint arXiv:1902.00159, 2019.

[2] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32, pages 7950–7958. Curran Associates, Inc., 2019.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9):509–517, 1975.

[4] Mikołaj Bińkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In International Conference on Learning Representations, 2018.

[5] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In International Conference on Learning Representations, 2019.

[6] Hanting Chen, Yunhe Wang, Han Shu, Changyuan Wen, Chunqing Xu, Boxin Shi, Chao Xu, and Chang Xu. Distilling portable generative adversarial networks for image translation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 3585–3592, 2020.

[7] Ting Chen, Xiaohua Zhai, Marvin Ritter, Mario Lucic, and Neil Houlsby. Self-supervised gans via auxiliary rotation loss. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12154–12163, 2019.

[8] Yunjey Choi, Youngjung Uh, Jaesoon Yoo, and Jung-Woo Ha. StarGAN v2: Diverse image synthesis for multiple domains. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2020.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

[10] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. J. Mach. Learn. Res., 20(55):1–21, 2019.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014.

[12] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30, pages 6626–6637. Curran Associates, Inc., 2017.

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[14] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 12104–12114. Curran Associates, Inc., 2020.

[15] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4396–4405, 2019.

[16] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In Proc. CVPR, 2020.

[17] Animesh Koratana, Daniel Kang, Peter Bailis, and Matei Zaharia. LIT: Learned intermediate representation training for model compression. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 3509–3518. PMLR, 09–15 Jun 2019.

[18] Tuomas Kynkäläniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32, pages 3927–3936. Curran Associates, Inc., 2019.

[19] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. Gan compression: Efficient architectures for interactive conditional gans. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.

[20] Jinglan Liu, Jiaxin Zhang, Yukun Ding, Xiaowei Xu, Meng Jiang, and Yiyu Shi. Binarizing weights wisely for edge intelligence: Guide for partial binarycation of deconvolution-based generators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020.

[21] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks with limited data. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 12104–12114. Curran Associates, Inc., 2020.

[22] Gonçalo Mordido and Christoph Meinel. Mark-evaluate: Assessing language generation using population estimation methods. In Proceedings of the 28th International Conference on Computational Linguistics, pages 1963–1977, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics.

[23] Gonçalo Mordido, Julian Niedermeier, and Christoph Meinel. Assessing image and text generation with topological analysis and fuzzy logic. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), pages 2013–2022, January 2021.

[24] Gonçalo Mordido, Matthijs Van Keirsbilck, and Alexander Keller. Instant quantization of neural networks using monte carlo methods. arXiv preprint arXiv:1905.12253, 2019.
[25] Goncalo Mordido, Matthijs Van Keirsbilck, and Alexander Keller. Monte carlo gradient quantization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pages 718–719, 2020. 5

[26] Goncalo Mordido, Haojin Yang, and Christoph Meinel. Dropout-gan: Learning from a dynamic ensemble of discriminators. arXiv preprint arXiv:1807.11346, 2018. 1

[27] Gonçalo Mordido, Haojin Yang, and Christoph Meinel. microbatchgan: Stimulating diversity with multi-adversarial discrimination. In The IEEE Winter Conference on Applications of Computer Vision, pages 3061–3070, 2020. 1

[28] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31, pages 5228–5237. Curran Associates, Inc., 2018. 1, 2

[29] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 29, pages 2234–2242. Curran Associates, Inc., 2016. 2

[30] Jonathan Sauder, Ting Hu, Xiaoyin Che, Goncalo Mordido, Haojin Yang, and Christoph Meinel. Best student forcing: A simple training mechanism in adversarial language generation. In Proceedings of the 12th Language Resources and Evaluation Conference, pages 4680–4688, Marseille, France, May 2020. European Language Resources Association. 1

[31] Han Shu, Yunhe Wang, Xu Jia, Kai Han, Hanting Chen, Chunjing Xu, Qi Tian, and Chang Xu. Co-evolutionary compression for unpaired image translation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 3235–3244, 2019. 1, 2

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014. 3

[33] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. IEEE Signal processing magazine, 25(2):128–131, 2008. 3

[34] Haotao Wang, Shupeng Gui, Haichuan Yang, Ji Liu, and Zhangyang Wang. Gan slimming: All-in-one gan compression by a unified optimization framework. In European Conference on Computer Vision, 2020. 2

[35] Peiqi Wang, Dongsheng Wang, Yu Ji, Xinfeng Xie, Haoxuan Song, XuXin Liu, Yongqiang Lyu, and Yuan Xie. Qgan: Quantized generative adversarial networks. arXiv preprint arXiv:1901.08263, 2019. 1, 2, 4

[36] Chong Yu and Jeff Pool. Self-supervised gan compression. arXiv preprint arXiv:2007.01491, 2020. 1, 2

[37] Dan Zhang and Anna Khoreva. Progressive augmentation of gans. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc., 2019. 5

[38] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In International conference on machine learning, pages 7543–7552. PMLR, 2019. 1, 4, 5

[39] Zhengli Zhao, Sameer Singh, Honglak Lee, Zizhao Zhang, Augustus Odena, and Han Zhang. Improved consistency regularization for gans. arXiv preprint arXiv:2002.04724, 2020. 5
A. Algorithm

Pseudo-code for the proposed metrics (LSH and LSH+KNN) is presented in Algorithm 1.

B. Realism scores

Figure 12 shows the mean realism scores of each ImageNet class using our LSH+KNN metric. We evaluated the images generated by the compressed models against the images generated by the 32-bit BigGAN (left) as well as the images from the real set (right), as usual. When assessing the generated sets of the compressed and uncompressed models, we see a decline in the realism scores, as expected. When comparing the compressed models with the real data, LSH+KNN shows that realism scores increase until the 2-bit compression mark, and decline at 2 bits. This goes in accordance with our findings in the paper.

Figure 12: BigGAN’s mean class realism scores on ImageNet 128x128.

C. Mean and generated images

Additional generated images of the studied networks and datasets are presented in Figures 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, and 33. The respective mean 10k images are presented in Figures 14, 16, 18, 20, 22, 24, 26, 28, 30, and 32.
Algorithm 1: Precision and recall metrics using locality-sensitive hashing.

**Input:** Set of real and generated images \((X_r, X_g)\), feature network \(F\) retrieving \(d\)-dimensional embeddings.

**function** \( \text{Precision-Recall}(X_r, X_g, F, k) : \)

\[
\begin{align*}
\Phi_r & \leftarrow F(X_r) \\
\Phi_g & \leftarrow F(X_g)
\end{align*}
\]

// Generate random hyperplanes

\[
H \leftarrow \log(|\Phi_r| + |\Phi_g|)
\]

\[
h \leftarrow \text{random.normal}(H, D)
\]

\[
b \leftarrow \text{random.uniform}(H)
\]

// Initialize hash-tables

\[
HT_r \leftarrow \text{Hash-Table}(\Phi_r, h, b)
\]

\[
HT_g \leftarrow \text{Hash-Table}(\Phi_g, h, b)
\]

// Evaluate with LSH

\[
\text{precision}_{LSH} \leftarrow \text{LSH}(HT_r, HT_g, k)
\]

\[
\text{recall}_{LSH} \leftarrow \text{LSH}(HT_g, HT_r, k)
\]

// Evaluate with LSH+KNN

\[
\text{precision}_{LSH+KNN} \leftarrow \text{LSH+KNN}(HT_r, HT_g, k)
\]

\[
\text{recall}_{LSH+KNN} \leftarrow \text{LSH+KNN}(HT_g, HT_r, k)
\]

**return** \( \text{precision}_{LSH}, \text{recall}_{LSH}, \text{precision}_{LSH+KNN}, \text{recall}_{LSH+KNN} \)

**function** \( \text{Hash-Table}(\theta, h, b) : \)

\[
\begin{align*}
\text{for } \phi & \in \theta \\
\text{key} & \leftarrow \text{dot}(\phi, h) + b \geq 0.\text{int}() \\
\text{if } \text{key} & \notin HT_\theta \text{ then} \\
HT_\theta[\text{key}] & = [\phi] \\
\text{else} \\
HT_\theta[\text{key}] & .\text{append}(\phi)
\end{align*}
\]

**return** \( HT_\theta \)

**function** \( \text{LSH}(HT_a, HT_b) : \)

\[
\begin{align*}
n & \leftarrow 0 \\
\text{for } \phi & \in HT_a.values \text{ do} \\
\text{if } HT_a.key(\phi) & \in HT_b \text{ then} \\
\text{ } & n \leftarrow n + 1
\end{align*}
\]

**return** \( n/|HT_a.values| \)

**function** \( \text{LSH+KNN}(HT_a, HT_b, k) : \)

\[
\begin{align*}
\text{for } \phi & \in HT_a.values \text{ do} \\
d & \leftarrow \{\|\phi - \phi'\|_2 \text{ for all } \phi' \in HT_a[HT_a.key(\phi)]\} \\
r_\phi & \leftarrow \text{min}_{k+1}(d)
\end{align*}
\]

\[
n \leftarrow 0 \\
\text{for } \phi & \in HT_b.values \text{ do} \\
\text{if } \|\phi - \phi'\|_2 \leq r_\phi \text{ for any } \phi' \in HT_a[HT_b.key(\phi)] \text{ then} \\
\text{ } & n \leftarrow n + 1
\end{align*}
\]

**return** \( n/|HT_b.values| \)
Figure 13: StyleGAN2 on FFHQ 1024x1024.
| 6 bits | 32 bits | 5 bits | 32 bits | 4 bits | 32 bits | 3 bits | 32 bits | 2 bits | 32 bits |
|---|---|---|---|---|---|---|---|---|---|

Figure 14: StyleGAN2 mean 10k images on FFHQ 1024x1024.
Figure 15: ADA on FFHQ 256x256.
Figure 16: ADA mean 10k images on FFHQ 256x256.
Figure 17: StyleGAN2 on FFHQ 256x256.
Figure 18: StyleGAN2 mean 10k images on FFHQ 256x256.
Figure 19: zCR-GAN on FFHQ 256x256.
Figure 20: zCR-GAN mean 10k images on FFHQ 256x256.
Figure 21: SN-GAN on FFHQ 256x256.
| 6 bits | 32 bits | 5 bits | 32 bits | 4 bits | 32 bits | 3 bits | 32 bits | 2 bits | 32 bits |
|--------|---------|--------|---------|--------|---------|--------|---------|--------|---------|

Figure 22: SN-GAN mean 10k images on FFHQ 256x256.
Figure 23: PA-GAN on FFHQ 256x256.
Figure 24: PA-GAN mean 10k images on FFHQ 256x256.
Figure 25: SS-GAN on FFHQ 256x256.
Figure 26: SS-GAN mean 10k images on FFHQ 256x256.
Figure 27: ADA on AFHQ Cat 512x512.
| 6 bits | 32 bits | 5 bits | 32 bits | 4 bits | 32 bits | 3 bits | 32 bits | 2 bits | 32 bits |
|--------|---------|--------|---------|--------|---------|--------|---------|--------|---------|
| ![Image](image1.png) | ![Image](image2.png) | ![Image](image3.png) | ![Image](image4.png) | ![Image](image5.png) | ![Image](image6.png) | ![Image](image7.png) | ![Image](image8.png) | ![Image](image9.png) | ![Image](image10.png) |

Figure 28: ADA mean 10k images on AFHQ Cat 512x512.
Figure 29: ADA on AFHQ Dog 512x512.
Figure 30: ADA mean 10k images on AFHQ Dog 512x512.
Figure 31: ADA on AFHQ Wild 512x512.
Figure 32: ADA mean 10k images on AFHQ Wild 512x512.
Figure 33: BigGAN on ImageNet 128x128.