Contrastive Generative Adversarial Networks

Minguk Kang  
Jaesik Park

Graduate School of Artificial Intelligence  
POSTECH

{mgkang, jaesik.park}@postech.ac.kr

Abstract

Conditional image synthesis is the task to generate high-fidelity diverse images using class label information. Although many studies have shown realistic results, there is room for improvement if the number of classes increases. In this paper, we propose a novel conditional contrastive loss to maximize a lower bound on mutual information between samples from the same class. Our framework, called Contrastive Generative Adversarial Networks (ContraGAN), learns to synthesize images using class information and data-to-data relations of training examples. The discriminator in ContraGAN discriminates the authenticity of given samples and maximizes the mutual information between embeddings of real images from the same class. Simultaneously, the generator attempts to synthesize images to fool the discriminator and to maximize the mutual information of fake images from the same class prior. The experimental results show that ContraGAN is robust to network architecture selection and outperforms state-of-the-art models by 3.7% and 11.2% on CIFAR10 and Tiny ImageNet datasets, respectively, without any data augmentation. For the fair comparison, we re-implement the nine state-of-the-art approaches to test various methods under the same condition. The software package that can re-produce all experiments is available at https://github.com/POSTECH-CVLab/PyTorch-StudioGAN.

1 Introduction

Generative Adversarial Networks (GAN) [1] have introduced a new paradigm for realistic data generation. The following approaches have shown impressive improvements in unconditional image synthesis tasks [2, 3, 4, 5, 6, 7, 8, 9]. The studies on non-convexity of objective landscapes [10, 11, 12] and gradient vanishing problems [3, 11, 13, 14] emphasize the instability of the adversarial dynamics. Therefore, many approaches have tried to stabilize the training procedure by adopting well-behaved objectives [3, 13, 15] and regularization techniques [4, 7, 16]. In particular, spectral normalization [4] with the projection discriminator [17] makes the first success in generating images of ImageNet dataset [18]. SAGAN [5] shows using spectral normalization on both the generator and discriminator can alleviate training instability of GANs. BigGAN [6] dramatically advances the quality of generated images by scaling up the number of network parameters and batch size.

On this journey, conditioning class information for the generator and discriminator turns out to be the secret behind the realistic image synthesis [17, 19, 20]. ACGAN [19] validates this direction by training a softmax classifier along with the discriminator. cGAN [17] utilizes a projection discriminator with probabilistic model assumptions. Especially, cGAN shows surprising image synthesis results and becomes the basic model adopted by SNResGAN [4], SAGAN [7], BigGAN [6], CRGAN [7], and LOGAN [9]. However, the cGAN does not take data-to-data relationships into account to discriminate the classes of given images. The ACGAN is known to be unstable when the number of classes increases [17, 19].

Preprint. Under review.
Figure 1: Illustrative figures visualize the metric learning losses (a,b,c) and conditional GANs (d,e,f). The objective of all losses is to collect samples if they have the same label but keep them away otherwise. The color indicates the class label, and the shape represents the role. (Square) an embedding of an image. (Diamond) an augmented embedding. (Circle) a reference. Each loss is applied to the reference. (Star) the embedding of a class label. (Triangle) the one-hot encoding of a class label. The thicknesses of red and blue lines represents the strength of pull and push force, respectively. Compared to ACGAN and cGAN, our loss is inspired by XT-Xent to consider data-to-data relationships and to infer full information without data augmentation.

In this paper, we propose a new conditional generative adversarial network framework. It is called Contrastive Generative Adversarial Networks (ContraGAN). To design ContraGAN, we formulate a lower bound on mutual information of image features and propose to use a novel class conditional contrastive loss (2C loss). We show the 2C loss can maximize the lower bound on mutual information between latent image features from the same class. Unlike previous class conditional approaches, such as ACGAN and cGAN, ContraGAN pulls for the embeddings of images from the discriminator to be close to each other when the class label is the same, but it pushes far away otherwise. In this manner, the discriminator can capture not only class information but also data-to-data relations between samples in the same batch. An illustrative figure of 2C loss is shown in Fig. 1f.

We performed conditional images synthesis experiments on CIFAR10 [21] and Tiny ImageNet [22] datasets using various backbone network architectures, such as SNDCGAN [4], SNResGAN [16, 4], and BigGAN [6] that are equipped with spectral normalization. Through exhaustive experiments, we demonstrate that the proposed ContraGAN improves the state-of-the-art-models by 3.7% in CIFAR10 and by 11.2% in Tiny ImageNet in terms of Frechet Inception Distance (FID) [23]. Since ContraGAN can learn plentiful data-to-data relations from large batches, it can reduce FID significantly without hard negative and positive mining. In ablation study, we show that our ContraGAN can benefit from consistency regularization [7] that uses data augmentations.

In summary, the contributions of our work are as follows:

• We introduce a novel, Contrastive Generative Adversarial Network (ContraGAN), that learns data-to-data relationships and improves the state-of-the-art-results by 3.7% and 11.2% on CIFAR10 and Tiny ImageNet datasets, respectively.
• We formulate one of the lower bounds on mutual information between images from the same class label. Based on this, we propose a novel conditional contrastive loss (2C loss) for conditional image synthesis.
• Our approach shows the superior synthesis result without data augmentations for consistency regularization. In addition, ContraGAN can give improved image generation results on Tiny ImageNet dataset with consistency regularization.
• We carefully implement the nine state-of-the-art-approaches [2, 3, 4, 5, 6, 7, 16, 17, 19] for the fair comparison. Due to standardized modules, our implementation of the prior arts achieves even better performances than FID scores reported in the original papers.

2 Background

2.1 Generative Adversarial Networks

Generative adversarial networks (GAN) [1] are implicit generative models that use a generator and a discriminator to synthesize realistic images. While the discriminator (D) should distinguish whether
Figure 2: Schematics of the discriminators of three conditional GANs. (a) ACGAN [19] has an auxiliary classifier to help the generator to synthesize well-classifiable images. (b) cGAN [17] improves the ACGAN by adding the inner product of an embedded image and the corresponding class embedding. (c) Our approach extends cGAN with conditional contrastive loss (2C loss) between embedded images and the actual label embedding. ContraGAN considers multiple positive and negative pairs in the same batch, as shown in Fig 1f. In the similar way, ContraGAN utilizes 2C loss to update the generator as well.

the given images are synthesized or not, the generator ($G$) tries to fool the discriminator by generating realistic images from noise vectors. The objective of the adversarial training is in the following:

$$
\min_G \max_D E_{x \sim p_{\text{real}}(x)} [\log (D(x))] + E_{z \sim p(z)} [\log (1 - D(G(z)))]
$$

where $p_{\text{real}}(x)$ is the real data distribution, and $p_{z}(z)$ is a predefined prior distribution, typically multivariate gaussian. Since the dynamics between the generator and discriminator is unstable and hard to achieve the Nash equilibrium [27], there are many objective functions [3, 13, 15, 28] and regularization techniques [4, 7, 16, 29] to help networks to converge in a proper equilibrium.

2.2 Conditional GAN

To synthesize realistic images, one of the widely used strategies is utilizing class label information. Early approaches in this category are conditional variational auto-encoder (CVAE) [30] and conditional generative adversarial networks [31]. These approaches concatenate input image vectors with the corresponding labels to manipulate the class conditional latent features of images. Since DCGAN [2] demonstrated high-resolution image synthesis, GANs utilizing class label information has shown advanced performances [6, 7, 9, 8].

The most common approach for the conditional image synthesis is to inject label information into the generator and discriminator of GANs. ACGAN attaches an auxiliary classifier on the top of convolution layers in the discriminator to distinguish the classes between images. An illustration of ACGAN is shown in Fig. 2a. cGAN points out that ACGAN is likely to generate classifiable images, but the images are not diverse due to a mode collapse problem. cGAN proposes a projection discriminator to relive the issue. However, these approaches do not explicitly consider data-to-data relations in the training phase (see Fig. 1e). Besides, the recent study by Wu et al. [9] discovers that the conditional GAN model [6] still suffers from a mode dropping phenomenon.

In this paper, we propose a new conditional GAN framework that explicitly pulls and pushes the image embeddings in a batch using a conditional contrastive loss. Our approach can generate more realistic and diverse images than ACGAN [19] and cGAN [17].

3 Method

This section introduces the proposed ContraGAN. First, we introduce a proposition about the lower bound on mutual information between any images from the same image class. We show that maximizing the lower bound can be an objective for conditional GANs (Sec. 3.1). Then, we bridge maximizing the lower bound with metric learning objectives and introduce a new conditional contrastive loss (2C loss) that can maximize the formulated lower bound on mutual information.
We put the proof of Proposition 1 in the supplement. Proposition 1 indicates that the lower bound $t$ where the scalar value $(Sec. 3.2). Lastly, we combine the mutual information maximization with GAN frameworks to generate more realistic images using data-to-data relations and class label information (Sec. 3.3).

3.1 Maximizing Lower bound on Mutual Information

The motivation of our approach is based on the view regarding the mutual information between images. The mutual information between two random variables is minimum when the random variables are independent of each other. If the two random variables are closely related, they will have a substantial value of the mutual information. In practice, we can think that images may have common latent features if they can be categorized in the same class. From this idea, we can deduce that encoder features of two images with the same class label should have a substantial value of the mutual information. Any network that can produce such encoder features is desirable for the conditional image synthesis.

Understanding a lower bound on mutual information between images with the same class label is essential to make such a desirable framework. It is because we can determine an additive objective for GAN pipeline to control the mutual information between images depending on the label. Our approach is inspired by Tian et al. [32]’s work. They propose a way to maximize the lower bound on mutual information between a teacher and a student network for knowledge distillation [32]. While Tian et al.’s work handles the mutual information between different embeddings from the same image, our work concentrates on the mutual information of different images with the same label. Thus, we can expand our lower bound on mutual information and introduce the following Proposition:

**Proposition 1.** Let $x$ and $y$ be a random variable vector of images and a random variable of corresponding labels, respectively. Using this, we can sample tuples $(x_i, y_i)$ and $(x_j, y_j)$ from the joint distribution $p_{real}(x, y)$. Let $f$ be a indicator function: $f(y_i, y_j) = 1_{y_i = y_j}$. Then, $\mathbb{E}_{p(x_i, x_j | f=1)}[\log(p(f = 1 | x_i, x_j))] - \log p(f = 1 | p(f = m))$ is one of the lower bounds on mutual information between $x_i$ and $x_j$.

We put the proof of Proposition 1 in the supplement. Proposition 1 indicates that the lower bound on mutual information is determined by the expectation of log posterior $p(f = 1 | x_i, x_j)$, because $\log\frac{p(f = 1)}{p(f = m)}$ is constant if the number of images per class is the same. In other words, if there is an encoder that makes embeddings of images, and if there is a classifier that takes two image features and correctly determines whether two images belong to the same image class, that would be equivalent to maximizing the lower bound on mutual information.

3.2 Conditional Contrastive Loss

Intuitively, the message from Proposition 1 is quite similar to the fundamental goal for metric learning [26, 24, 33, 34, 35, 36, 37]. Therefore, our approach is to add metric learning objectives in the discriminator and generator to explicitly control distances between embedded image features depending on the labels. Several metric learning losses, such as contrastive loss [33], triplet loss [24], quadruplet loss [34], and N-pair loss [35] could be adopted for our model. However, it is known that mining informative triplets and quadruplets require higher training complexity, and poor tuples make the training time longer. While the proxy-based losses [25, 36, 37] relieves mining complexity using trainable embedding vectors, such losses do not explicitly account data-to-data relationships [38].

Before introducing the proposed 2C loss, we bring XT-Xent loss [26] to express our idea better. Let $X = \{x_1, ..., x_m\}$, where $x \in \mathbb{R}^W \times H$ be a randomly sampled minibatch of training images and $y = \{y_1, ..., y_m\}$, where $y \in \mathbb{R}$ be the collection of corresponding class labels. Then, we define a deep neural network encoder $S(x) \in \mathbb{R}^k$ and a projection layer that embeds onto a new unit hypersphere $h : \mathbb{R}^k \rightarrow S^d$. Then, we can map the data space to the hypersphere using the composition of $l = h(S(\cdot))$. XT-Xent loss conducts random data augmentations $T$ on the training data $X$ and denote it as $A = \{x_1, T(x_1), ..., x_m, T(x_m)\} = \{a_1, ..., a_{2m}\}$. Using the above, we can formulate the XT-Xent loss as follows:

$$\ell(a_i, a_j; t) = -\log \left( \frac{\exp(l(a_i) \cdot l(a_j)/t)}{\sum_{k=1}^{2m} \exp(l(a_i) \cdot l(a_k)/t)} \right),$$

where the scalar value $t$ is called temperature to control push and pull force. In this work, we use the part of the discriminator network before the fully connected layers $(D_{\phi})$ as the encoder network $(S)$.
With the proposed 2C loss, we describe the framework, called ContraGAN and introduce the
Algorithm 1

\textbf{Input:} Learning rate: α₁, α₂, Adam hyperparameters [39]: β₁, β₂, Batch size: m. Temperature: t.
# of discriminator iterations per single generator iteration: nₙₐᵣ. Contrastive coefficient: λ.
Parameters of the generator, the discriminator, and the projection layer: (θ, φ, ϕ).

\textbf{Output:} Optimized (θ, φ, ϕ).

1: Initialize (φ, θ, ϕ).
2: for \{1, ..., # of training iterations\} do
3: \hspace{1em} for \{1, ..., nₙₐᵣ\} do
4: \hspace{2em} Sample \{(xᵢ, yᵢ\text{real})\}_{i=1}^m \sim p\text{real}(x, y)
5: \hspace{2em} Sample \{zᵢ\}_{i=1}^m \sim p(z) and \{yᵢ\}_{i=1}^m \sim p(y)
6: \hspace{2em} \mathcal{L}_C^{\text{real}} \leftarrow -\frac{1}{m} \sum_{i=1}^m \log(\ell_{2C}(xᵢ, yᵢ; t)) \hspace{1em} \triangleright Eq. (8) with real images.
7: \hspace{2em} \mathcal{L}_D \leftarrow -\frac{1}{m} \sum_{i=1}^m \{D_θ(G_θ(zᵢ, yᵢ)) - D_θ(xᵢ)\} + λ\mathcal{L}_C^{\text{real}}
8: \hspace{2em} φ \leftarrow \text{Adam}(\mathcal{L}_D, α₁, β₁, β₂)
9: \hspace{1em} end for
10: Sample \{zᵢ\}_{i=1}^m \sim p(z) and \{yᵢ\}_{i=1}^m \sim p(y)
11: \hspace{2em} \mathcal{L}_C^{\text{fake}} \leftarrow -\frac{1}{m} \sum_{i=1}^m \log(\ell_{2C}(G_θ(zᵢ, yᵢ), yᵢ; t)) \hspace{1em} \triangleright Eq. (8) with fake images.
12: \hspace{2em} \mathcal{L}_G \leftarrow -\frac{1}{m} \sum_{i=1}^m \{D_θ(G_θ(zᵢ, yᵢ))\} + λ\mathcal{L}_C^{\text{fake}}
13: \hspace{2em} θ \leftarrow \text{Adam}(\mathcal{L}_G, α₂, β₁, β₂)
14: \hspace{1em} end for

end for

and use multi-layer perceptrons parameterized by ϕ as the projection layer (h). As a result, we can map the data space to the unit hyper sphere using \( l = h(D_ϕ(\cdot)) \).

However, Eq. (6) requires proper data augmentations and spends much time on propagation and backpropagation. To release the issues, we propose to use the embeddings of class labels instead of explicit data augmentations. With a class embedding function \( e : y \in \mathbb{Y} \rightarrow \mathbb{R}^d \), Eq. (6) can be formulated as follows:

\[
\ell(xᵢ, yᵢ; t) = -\log\left( \frac{\exp(l(xᵢ)ᵀe(yᵢ\text{}/t))}{\exp(l(xᵢ)ᵀe(yᵢ)/t) + \sum_{k=1}^m 1_{k \neq i} \cdot \exp(l(xᵢ)ᵀl(x_k)/t)} \right). \tag{7}
\]

Eq. 7 pulls a reference sample \( xᵢ \) nearer the class embedding \( e(yᵢ) \) and pushes the others. This scheme may push negative samples which have the same label as \( yᵢ \). Therefore, we make an exception by adding cosine similarities of such negative samples in the numerator of Eq. 7. The final loss function is as follows:

\[
\ell_{2C}(xᵢ, yᵢ; t) = -\log\left( \frac{\exp(l(xᵢ)ᵀe(yᵢ)/t) + \sum_{k=1}^m 1_{y_k = yᵢ} \cdot \exp(l(xᵢ)ᵀl(x_k)/t)}{\exp(l(xᵢ)ᵀe(yᵢ)/t) + \sum_{k=1}^m 1_{k \neq i} \cdot \exp(l(xᵢ)ᵀl(x_k)/t)} \right). \tag{8}
\]

Eq. (8) is the proposed conditional contrastive loss (2C loss). 2C loss follows the lesson from Proposition 1 – it minimizes distances between the embeddings of images with the same label while maximizing the others. 2C loss explicitly considers data-to-data relationships \( l(xᵢ)ᵀl(x_k) \) and the data-to-class relationship \( l(xᵢ)ᵀe(yᵢ) \) without comprehensive mining of the training dataset and augmentations.

3.3 Contrastive Generative Adversarial Networks

With the proposed 2C loss, we describe the framework, called ContraGAN and introduce the training procedure. Like the typical training procedure, ContraGAN has a discriminator training step and a generator training step that compute an adversarial loss. With this foundation, ContraGAN additionally calculates 2C loss using a set of real or fake images. Algorithm 1 shows the training procedure of the proposed ContraGAN. A notable aspect is that 2C loss is computed using \( m \) real images in the discriminator training step and generated images in the generator training step.

In this manner, the discriminator updates itself by maximizing the lower bound on mutual information of real images from the same class. By forcing the embeddings to be dependent via 2C loss, the discriminator can learn useful representations of real images. Similarly, the generator exploits knowledge of the discriminator, such as intra-class characteristics and higher-order representations of the real images, to generate more realistic images.
4 Experiments

4.1 Datasets and the evaluation metric

We perform experiments with relatively small datasets, such as CIFAR10 [21] and Tiny ImageNet [22], to focus on analyzing the proposed idea and other approaches with various configurations.

**CIFAR10** [21] is the widely used benchmark dataset in many image synthesis works [4, 6, 7, 8, 9, 17, 19] and contains $32 \times 32$ pix. color images of 10 different classes that makes 60,000 images in total. It is divided into 50,000 images for training and 10,000 images for testing.

**Tiny ImageNet** [22] provides 120,000 color images in total. Image size is $64 \times 64$ pix., and the dataset consists of 200 categories. Each category has 600 images divided into 500, 50, and 50 samples for training, validation, and testing. Tiny ImageNet has $10 \times$ smaller number of images per class than CIFAR10, but it provides $20 \times$ larger number of classes than CIFAR10. Compared to CIFAR10, Tiny ImageNet is selected to test a more challenging scenario – the number of images per class is not plentiful, and the network needs to learn more categories.

**Frechet Inception Distance (FID)** is the evaluation metric for the experiments in this paper. The FID proposed by Heusel et al. [40] calculates Wasserstein-2 distance [41] between the features obtained from real images and generated images using Inception-V3 network [42]. Since FID is distance between two distributions, lower FID indicates better results.

4.2 Software

There are various approaches that report strong FID scores, but it is not easy to reproduce the results because detailed specifications for training or ways to measure the results are not clearly stated. For instance, FID could be different depending on the choice of the reference image dataset (test or validation sets could be used). Besides, FIDs and performances of prior work could not be consistent, depending on the TensorFlow version [43]. Therefore, we re-implement the nine state-of-the-art conditional GANs [2, 3, 4, 5, 6, 7, 16, 17, 19] to validate the proposed ContraGAN under the same condition. Our implementation carefully follows principal concepts and available specifications. Experimental results show that results from our implementation are superior to the numbers in the original papers [4, 6]. We hope that our implementation would relieve efforts to compare various GAN pipelines. The software will be released to the public.

4.3 Experimental setup

To conduct reliable assessments, all experiments that use CIFAR10 dataset are performed three times with different random seeds, and we report end-time performance using means and standard deviations of FIDs. Experiments using Tiny ImageNet are executed once and reported the best performance during the training. We calculate FID using CIFAR10’s 10,000 test images and the same amount of generated images. We compute FID using Tiny ImageNet’s 10,000 validation images and the same amount of generated images. All FID values reported in our paper are calculated using the PyTorch FID implementation [44].

Since spectral normalization [4] has become the essential element in modern GAN training, we use Hinge loss [15] and apply spectral normalization on all architectures used in our experiments. We adopt modern architectures used in the papers: SNDCGAN [2, 4], SNResGAN [16], and BigGAN [6], and all details about the architectures are described in the supplement. Since ACGAN concatenates random noise vectors with class labels, using the original ACGAN implementation may produce unfair results. Therefore, we unify the protocol of conditioning the generator in all experiments and use the conditional batch normalization [45, 46, 17], which is the way adopted by the original cGAN.

Before conducting the main experiments, we investigate the performance change according to the type of projection layer $h$ in Eq. (7) and batch size. Although Chen et al. [26] reports that the higher-dimensional projection layer and larger batch size make better test accuracy, we found that the linear projection with batch size 64 for CIFAR10 and 1,024 for Tiny ImageNet performs the best. We select 512 dimensions for CIFAR10 and 768 dimensions for Tiny ImageNet generation experiments. We do a grid search to find a proper temperature ($t$) used in Eq. 8 and experimentally found that the temperature of 1 gives the best results. Detailed hyperparameter settings used in our experiments are described in the supplement.
Table 1: Experiments using CIFAR10 and Tiny ImageNet dataset. Using three backbone architectures (SNDCGAN, SNResGAN, and BigGAN), we test three approaches using different class information conditioning (ACGAN, cGAN, and ours). Mean±variance of FID is reported, and lower is better.

| Dataset         | Backbone       | Method for class information conditioning | ACGAN [19] | cGAN [17] | Our ContraGAN |
|-----------------|----------------|-------------------------------------------|------------|----------|---------------|
| CIFAR10 [21]    | SNDCGAN [2, 4] | 21.439±0.581                              | 19.524±0.249 | 18.788±0.571 |
|                 | SNResGAN [16]  | 11.588±0.093                              | **11.025±0.177** | 11.334±0.126 |
|                 | BigGAN [6]     | 10.697±0.129                              | 10.739±0.016 | **10.597±0.273** |
| Tiny ImageNet [22] | BigGAN [6] | 221.381                                    | 40.981     | **32.094** |

Table 2: Comparison with state-of-the-art GAN models. We mark ‘*’ to FID values reported in the original papers [6, 7]. The other FID values are obtained from our implementation.

| Dataset         | SNResGAN [4] | SAGAN [5] | BigGAN [6] | Our ContraGAN | Improvement |
|-----------------|---------------|-----------|------------|---------------|-------------|
| CIFAR10 [21]    | *17.5         | 17.283    | *14.73/10.722 | **10.322**    | *+29.9%/+3.7% |
| Tiny ImageNet [22] | 46.969       | 42.558    | 34.090     | **30.286**    | +11.2%      |

4.4 Evaluation results

Comparison with other conditional GANs. We compare ContraGAN with ACGAN [19] and cGAN [17], since these approaches are representative ones using class information conditioning. As shown in Table 1, our approach shows favorable performances in CIFAR10, but our approach exhibits larger variations. Examples of generated images is shown in Fig. 3 (left). Experiment with Tiny ImageNet indicates that our ContraGAN is more effective when the target dataset is in the higher-dimensional space and has large inter-class variations.

Comparison with state-of-the-art models. We compare our method with SNResGAN [4], SAGAN [5], and BigGAN [6]. All of these approaches adopt cGAN [17] for class information conditioning. We conduct all experiments on Tiny ImageNet dataset using the hyperparameter setting used in SAGAN [5] with BigGAN as the backbone architecture and report the best FID values during training. For fair comparison, we use our implementation of BigGAN to produce better FID than the original papers [6, 7].

If we consider the most recent works that are under-review or just got accepted, CRGAN [7], ICRGAN [8], and LOGAN [9] can generate more realistic images than the BigGAN. Compared to such approaches, we show that our framework outperforms BigGAN by just adopting the proposed 2C loss. CRGAN, and ICRGAN conduct explicit data augmentations during the training, which require additional gradient calculations for the backpropagation. LOGAN needs one more feedforward and backpropagation processes for latent optimization and takes about two times longer for training than standard GANs.

As a result, we identify how our ContraGAN performs without explicit data augmentations or latent optimization. Table 2 quantitatively shows ContraGAN can synthesize images better than other state-of-the-art GAN models under the same conditions. Compared to BigGAN, ContraGAN improves the performances by 3.7% on CIFAR10 and 11.2% on Tiny ImageNet. If we use the reported number in BigGAN paper [6], the improvement is 29.9% on CIFAR10.

4.5 Ablation study

We study how ContraGAN can be improved further with a large batch size and data augmentations. We use cGAN with BigGAN architecture on Tiny imageNet for this study. We use consistency regularization (CR) [7] to identify our ContraGAN can benefit from regularization that uses data augmentations. Also, to identify 2C loss is not only computationally cheap but also effective to train GANs, we replace the class embeddings with augmented positive images (APS). APS is widely used in the self-supervised contrastive learning work [26, 47]. Table 3 shows the experiment settings, FID, and time per iteration. We indicate the number of parameters as Param. and denote three ablations – (the 2C loss, augmented positive samples (APS), and consistency regularization (CR)) as Reg.
Figure 3: Examples of generated images using the proposed ContraGAN. (left) CIFAR10 [21], FID: 10.322, (right) Tiny ImageNet [22], FID: 27.018. Images in each row belong to the same class.

**Large batch size.** (A, C) and (E, H) show that ContraGAN can benefit from large batch size.

**Effect of the proposed 2C loss.** (A, E) and (C, H) show that the proposed 2C loss significantly reduces FID scores of the vanilla networks (A, C) by 21.6% and 11.2%, respectively.

**Comparison with APS.** From the experiments (E, F), we can see that the 2C loss performs better than 2C loss + APS, despite the latter takes about 12.9% more time. We speculate this is because each class embedding can become the representatives of the class, and it serves as the anchor that pulls corresponding images. Without the class embeddings, images in a minibatch are collected depending on a sampling state, and this may lead to training instability.

**Comparison with CR.** (A, E, G) and (C, H, I) show that vanilla + 2C loss + CR can reduce FID of either the results from vanilla networks (A, C) and vanilla + 2C loss (E, H). Note that the synergy is only observable if CR is used with 2C loss, and vanilla + 2C loss + CR beats vanilla + CR (B, D) with a large margin. Results are shown in Fig. 3 (right).

| ID  | (A) | (B) | (C) | (D) | (E) | (F) | (G) | (H) | (I) |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Batch | 256 | 256 | 1024 | 1024 | 256 | 256 | 256 | 1024 | 1024 |
| Param. | 48.1 | 48.1 | 48.1 | 48.1 | 49.0 | 49.0 | 49.0 | 49.0 | 49.0 |
| Reg. | - - ✓ | - - ✓ | - - ✓ | - - ✓ | ✓ ✓ ✓ | ✓ ✓ ✓ | ✓ ✓ ✓ | ✓ ✓ ✓ | ✓ ✓ ✓ |
| FID | 40.981 | 36.434 | 34.090 | 38.231 | 32.094 | 33.151 | 28.631 | 30.286 | 27.018 |
| Time | 0.901 | 1.093 | 3.586 | 4.448 | 0.967 | 1.110 | 1.121 | 3.807 | 4.611 |

**5 Conclusion**

In our paper, we formulate a lower bound on mutual information between images categorized to the same class. Using this, we present a new Contrastive Generative Adversarial Networks (ContraGAN) that maximizes the lower bound on mutual information using a new conditional contrastive loss (2C loss). Unlike previous losses used in conditional GANs, the 2C loss considers not only data-to-class but also data-to-data relationships. Under the same condition, we demonstrated that our ContraGAN achieved state-of-the-art performances on conditional image synthesis on CIFAR10 and Tiny ImageNet datasets. Also, we identified ContraGAN performs even better when consistency regularization is applied. As future work, we would like to explore that the advanced regularization techniques [8, 9] and generator conditional way [20] can improve our framework further. Also, we would like to conduct a large-scale conditional image synthesis experiment using ImageNet dataset [18].
References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Advances in Neural Information Processing Systems (NeurIPS), pages 2672–2680, 2014.

[2] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv 1511.06434, 2016.

[3] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. arXiv preprint arXiv 1701.07875, 2017.

[4] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2018.

[5] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-Attention Generative Adversarial Networks. In Proceedings of the International Conference on Machine Learning (ICML), pages 7354–7363, 2019.

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

[7] Han Zhang, Zizhao Zhang, Augustus Odena, and Honglak Lee. Consistency Regularization for Generative Adversarial Networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2020.

[8] Zhengli Zhao, Sameer Singh, Honglak Lee, Zizhao Zhang, Augustus Odena, and Han Zhang. Improved Consistency Regularization for GANs. arXiv preprint arXiv 2002.04724, 2020.

[9] Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy P. Lillicrap. LOGAN: Latent Optimisation for Generative Adversarial Networks. arXiv preprint arXiv 1912.00953, 2019.

[10] Naveen Kodali, James Hays, Jacob D. Abernethy, and Zsolt Kira. On Convergence and Stability of GANs. arXiv preprint arXiv 1705.07215, 2018.

[11] Jerry Li, Aleksandar Madry, John Peebles, and Ludwig Schmidt. On the Limitations of First-Order Approximation in GAN Dynamics. In Proceedings of the International Conference on Machine Learning (ICML), 2018.

[12] Vaishnavh Nagarajan and J. Zico Kolter. Gradient descent GAN optimization is locally stable. In Advances in Neural Information Processing Systems (NeurIPS), pages 5585–5595, 2017.

[13] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhixiang Wang, and Stephen Paul Smolley. Least Squares Generative Adversarial Networks. In Proceedings of the International Conference on Computer Vision (ICCV), pages 2813–2821, 2017.

[14] Martín Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[15] Jae Hyun Lim and Jong Chul Ye. Geometric GAN. arXiv preprint arXiv 1705.02894, 2017.

[16] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved Training of Wasserstein GANs. In Advances in Neural Information Processing Systems (NeurIPS), pages 5767–5777, 2017.

[17] Takeru Miyato and Masanori Koyama. eGANs with Projection Discriminator. In Proceedings of the International Conference on Learning Representations (ICLR), 2018.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), pages 248–255, 2009.

[19] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional Image Synthesis with Auxiliary Classifier GANs. In Proceedings of the International Conference on Machine Learning (ICML), pages 2642–2651, 2017.

[20] Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening and Coloring Batch Transform for GANs. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

[21] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. PhD thesis, University of Toronto, 2012.

[22] Johnson et al. Tiny ImageNet Visual Recognition Challenge. https://tiny-imagenet.herokuapp.com.

[23] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Two time-scale update rule for training GANs. https://github.com/bioinf-jku/TTUR, 2018.
[24] Elad Hoffer and Nir Ailon. Deep Metric Learning Using Triplet Network. In SIMBAD, 2015.

[25] Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, and Saurabh Singh. No Fuss Distance Metric Learning Using Proxies. In Proceedings of the International Conference on Computer Vision (ICCV), pages 360–368, 2017.

[26] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. arXiv preprint arXiv 2002.05709, 2020.

[27] John Nash. Non-Cooperative Games. Annals of mathematics, pages 286–295, 1951.

[28] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization. In Advances in Neural Information Processing Systems (NeurIPS), pages 271–279, 2016.

[29] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. Neural Photo Editing with Introspective Adversarial Networks. In Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[30] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In Advances in Neural Information Processing Systems (NeurIPS), pages 3483–3491, 2015.

[31] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. arXiv preprint arXiv 1411.1784, 2014.

[32] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive Representation Distillation. In Proceedings of the International Conference on Learning Representations (ICLR), 2020.

[33] Nicolás Aziere and Sinisa Todorovic. Ensemble Deep Manifold Similarity Learning Using Hard Proxies. In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), pages 7291–7299, 2019.

[34] Marc T. Law, Nicolas Thome, and Matthieu Cord. Quadruplet-Wise Image Similarity Learning. In Proceedings of the International Conference on Computer Vision (ICCV), pages 249–256, 2013.

[35] Kihyuk Sohn. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In Advances in Neural Information Processing Systems (NeurIPS), pages 1857–1865, 2016.

[36] 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 Advances in Neural Information Processing Systems (NeurIPS), pages 6626–6637, 2017.

[37] Leonid Nisonovich Vaserstein. Markov processes over denumerable products of spaces, describing large systems of automata. Problemy Peredachi Informatsii, 5(3):64–72, 1969.

[38] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), pages 2818–2826, 2016.

[39] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

[40] mseitzer. A Port of Fréchet Inception Distance (FID score) to PyTorch. https://github.com/mseitzer/pytorch-fid, 2018.
[46] Harm de Vries, Florian Strub, Jeremie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C Courville. Modulating early visual processing by language. In Advances in Neural Information Processing Systems (NeurIPS), pages 6594–6604, 2017.

[47] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised Contrastive Learning. arXiv preprint arXiv 2004.11362, 2020.

[48] Christopher M Bishop. Pattern recognition and machine learning. springer, 2006.

[49] Claude E Shannon. A mathematical theory of communication. Bell system technical journal, 27(3):379–423, 1948.

[50] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the International Conference on Machine Learning (ICML), pages 448–456, 2015.

[51] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In Proceedings of the International Conference on Learning Representations (ICLR), 2010.

[52] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. arXiv preprint arXiv 1505.00853, 2015.

[53] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. arXiv preprint arXiv 1312.4400, 2014.

[54] David Warde-Farley and Yoshua Bengio. Improving Generative Adversarial Networks with Denoising Feature Matching. In Proceedings of the International Conference on Learning Representations (ICLR), 2017.

[55] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. arXiv preprint arXiv 1710.10196, 2018.

[56] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. Which Training Methods for GANs do actually Converge? In Proceedings of the International Conference on Machine Learning (ICML), 2018.

[57] Yasin Yazıcı, Chuan-Sheng Foo, Stefan Winkler, Kim-Hui Yap, Georgios Piliouras, and Vijay Chandrasekhar. The Unusual Effectiveness of Averaging in GAN Training. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andrea Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems (NeurIPS), pages 8024–8035, 2019.
Appendices

A Proof of the Proposition 1

Proposition 1. Let \(x\) and \(y\) be a random variable vector of images and a random variable of corresponding labels, respectively. Using this, we can sample tuples \((x_i, y_i)\) and \((x_j, y_j)\) from the joint distribution \(p_{\text{real}}(x, y)\). Let \(f\) be a indicator function: \(f(y_i, y_j) = 1_{y_i = y_j}\). Then, 
\[
\mathbb{E}_{p(x_i, x_j|f=1)}[\log(p(f = 1|x_i, x_j))] - \log\frac{\mathbb{E}[f=1]}{p(f=0)}
\]
is one of the lower bounds on mutual information between \(x_i\) and \(x_j\).

Proof. Suppose \(p(f = 1|x_i, x_j)\) is a posterior distribution that expresses probability about whether the given images are came from the same category or not. By Bayes’ theorem [48], we can expand the posterior distribution as follows:
\[
p(f = 1|x_i, x_j) = \frac{p(x_i, x_j|f = 1)p(f = 1)}{\sum_{c=0}^{1} p(x_i, x_j|f = c)p(f = c)}.
\] (A1)

By taking log both sides, we can develop Eq. (A1)
\[
\log(p(f = 1|x_i, x_j)) = \log\left(\frac{p(x_i, x_j|f = 1)p(f = 1)}{\sum_{c=0}^{1} p(x_i, x_j|f = c)p(f = c)}\right)
= -\log\left(\sum_{c=0}^{1} p(x_i, x_j|f = c)p(f = c)\right)
= -\log\left(1 + \frac{p(x_i, x_j|f = 0)p(f = 0)}{p(x_i, x_j|f = 1)p(f = 1)}\right).
\] (A2)

Since \(-\log(x)\) is a decreasing function, \(-\log(1 + x) \leq -\log(x)\) is valid. Given this, we can develop Eq. (A2) as follows:
\[
\log(p(f = 1|x_i, x_j)) \leq -\log\left(\frac{p(x_i, x_j|f = 0)p(f = 0)}{p(x_i, x_j|f = 1)p(f = 1)}\right)
= -\log\left(\frac{p(x_i, x_j|f = 0)}{p(x_i, x_j|f = 1)}\right) - \log\left(\frac{p(f = 0)}{p(f = 1)}\right)
= \log\left(\frac{p(x_i, x_j|f = 1)}{p(x_i, x_j|f = 0)}\right) + \log\left(\frac{p(f = 1)}{p(f = 0)}\right).
\] (A3)

In Eq. A3, we replace \(p(x_i, x_j|f = 0)\) with \(p(x_i)p(x_j)\). It is based on the assumption that two random variables of different labels should not share latent features, and \(x_i, x_j\) are independent. Therefore we can rewrite Eq. (A3) as follows:
\[
\log(p(f = 1|x_i, x_j)) \leq \log\left(\frac{p(x_i, x_j|f = 1)}{p(x_i)p(x_j)}\right) + \log\left(\frac{p(f = 1)}{p(f = 0)}\right),
\] (A4)

where \(a = \log(\frac{p(f=1)}{p(f=0)})\) is a constant if the number of images per class is the same. If we take expectation w.r.t the distribution \(p(x_i, x_j|f = 1)\), the non-constant term on the right-side in Eq. (A4) becomes mutual information \(\mathbb{I}[\cdot]\) [49] as follows:
\[
\mathbb{E}_{p(x_i, x_j|f=1)}[\log(p(f = 1|x_i, x_j))] \leq \mathbb{I}(x_i, x_j) + a.
\] (A5)

To conclude, we can derive one of the lower bounds on mutual information \(\mathbb{I}(x_i, x_j)\),
\[
\mathbb{I}(x_i, x_j) \geq \mathbb{E}_{p(x_i, x_j|f=1)}[\log(p(f = 1|x_i, x_j))] - a.
\]

B Network Architectures

Since DCGAN [2] showed astonishing image generation ability, several generator and discriminator architectures have been proposed to stabilize and enhance the generation quality. Representatively, Miyato et al. [4] have used modified version of DCGAN [2] and ResNet-style GAN [16] architectures.
with spectral normalization (We abbreviate it SNDCGAN and SNResGAN, respectively). Brock et al. [6] have expanded the capacity of SNResGAN with shared embeddings and skip connections from the noise vector (BigGAN). As a result, we tested the aforementioned frameworks to validate the proposed approach. To provide details of the main experiments in our paper, we introduce the network architectures in this section.

We start by defining some notations: \( m \) is a batch size, \( \text{FC}(\text{in_features}, \text{out_features}) \) is a fully connected layer, \( \text{CONV}(\text{in_channels}, \text{out_channels}, \text{kernel_size}, \text{strides}) \) is a convolutional layer, \( \text{DECONV}(\text{in_channels}, \text{out_channels}, \text{kernel_size}, \text{strides}) \) is a deconvolutional layer, \( \text{BN} \) is a batch normalization [50], \( \text{C} \text{BN} \) is a conditional batch normalization [45, 46, 17], \( \text{ReLU}, \text{LRU}, \text{and TANH} \) indicate ReLU [51], Leaky ReLU [52], and hyperbolic tangent functions, respectively. \( \text{GENBLOCK}(\text{in channels}, \text{out channels}, \text{upsampling}) \) is a generator block used in [16, 4], \( \text{BIGGENBLOCK}(\text{in channels}, \text{out channels}, \text{upsampling}, \text{z split dim}, \text{shared dim}) \) is a modified version of the \( \text{GENBLOCK} \) used in [6], \( \text{DISBLOCK}(\text{in channels}, \text{out channels}, \text{downsampling}) \) is a discriminator block used in [6], \( \text{SELFATTENTION} \) is a self-attention layer used in [5], \( \text{NORMALIZE} \) is a normalize operation to project given embeddings onto a unit hypersphere, and \( \text{GSP} \) is a global sum pooling layer [53]. For more details about the \( \text{GENBLOCK}, \text{BIGGENBLOCK}, \text{DISBLOCK}, \) and the \( \text{SELF-ATTENTION} \) layer, please refer to the papers [4, 5, 6] or the code of our PyTorch implementation (ContraGAN/models/.py).

| Layer            | Input          | Output          | Operation               |
|------------------|----------------|-----------------|-------------------------|
| Input Layer      | \((m, 128)\)   | \((m, 8192)\)   | \(\text{FC}(128, 8192)\) |
| Reshape Layer    | \((m, 8192)\)  | \((m, 4, 4, 512)\) | \(\text{RESHAPE}\)      |
| Hidden Layer     | \((m, 4, 4, 512)\) | \((m, 8, 8, 256)\) | \(\text{DECONV}(512, 256, 4, 2), \text{CBN}, \text{LReLU}\) |
| Hidden Layer     | \((m, 8, 8, 256)\) | \((m, 16, 16, 128)\) | \(\text{DECONV}(256, 128, 4, 2), \text{CBN}, \text{LReLU}\) |
| Hidden Layer     | \((m, 16, 16, 128)\) | \((m, 32, 32, 64)\) | \(\text{DECONV}(128, 64, 4, 2), \text{CBN}, \text{LReLU}\) |
| Hidden Layer     | \((m, 32, 32, 64)\) | \((m, 32, 32, 3)\) | \(\text{CONV}(64, 3, 3, 1)\) |
| Output Layer     | \((m, 32, 32, 3)\) | \((m, 32, 32, 3)\) | \(\text{TANH}\) |

| Layer            | Input          | Output          | Operation               |
|------------------|----------------|-----------------|-------------------------|
| Input Layer      | \((m, 32, 32, 3)\) | \((m, 32, 32, 64)\) | \(\text{CONV}(3, 64, 3, 1), \text{LReLU}\) |
| Hidden Layer     | \((m, 32, 32, 64)\) | \((m, 16, 16, 64)\) | \(\text{CONV}(64, 64, 4, 2), \text{LReLU}\) |
| Hidden Layer     | \((m, 16, 16, 64)\) | \((m, 16, 16, 128)\) | \(\text{CONV}(64, 128, 3, 1), \text{LReLU}\) |
| Hidden Layer     | \((m, 16, 16, 128)\) | \((m, 8, 8, 128)\) | \(\text{CONV}(128, 128, 4, 2), \text{LReLU}\) |
| Hidden Layer     | \((m, 8, 8, 128)\) | \((m, 8, 8, 256)\) | \(\text{CONV}(128, 256, 3, 1), \text{LReLU}\) |
| Hidden Layer     | \((m, 8, 8, 256)\) | \((m, 4, 4, 256)\) | \(\text{CONV}(256, 256, 4, 2), \text{LReLU}\) |
| Hidden Layer     | \((m, 4, 4, 256)\) | \((m, 4, 4, 512)\) | \(\text{CONV}(256, 512, 3, 1), \text{LReLU}\) |
| Hidden Layer     | \((m, 4, 4, 512)\) | \((m, 512)\) | \(\text{GSP}\) |
| Output Layer     | \((m, 512)\) | \((m, 1)\) | \(\text{FC}(512, 1)\) |
Table A3: Generator of SNResGAN [4] used for CIFAR10 [21] image synthesis.

| Layer          | Input         | Output        | Operation                  |
|----------------|---------------|---------------|----------------------------|
| Input Layer    | (m, 128)      | (m, 4096)     | FC(128, 4096)              |
| Reshape Layer  | (m, 4096)     | (m, 4, 4, 256)| RESHAPE                   |
| Hidden Layer   | (m, 4, 4, 256)| (m, 8, 8, 256)| GEN_BLOCK(256, 256, True)  |
| Hidden Layer   | (m, 8, 8, 256)| (m, 16, 16, 256)| GEN_BLOCK(256, 256, True) |
| Hidden Layer   | (m, 16, 16, 256)| (m, 32, 32, 256)| GEN_BLOCK(256, 256, True) |
| Hidden Layer   | (m, 32, 32, 256)| (m, 32, 32, 3  | BN, RELU, CONV(256, 3, 3, 1) |
| Output Layer   | (m, 32, 32, 3)| (m, 32, 32, 3)| TANH                       |

Table A4: Discriminator of SNResGAN [4] used for CIFAR10 [21] image synthesis.

| Layer          | Input         | Output        | Operation                  |
|----------------|---------------|---------------|----------------------------|
| Input Layer    | (m, 32, 32, 3)| (m, 16, 16, 128)| DIS_BLOCK(3, 128, True)   |
| Hidden Layer   | (m, 16, 16, 128)| (m, 8, 8, 128)| DIS_BLOCK(128, 128, True) |
| Hidden Layer   | (m, 8, 8, 128)| (m, 8, 8, 128)| DIS_BLOCK(128, 128, False) |
| Hidden Layer   | (m, 8, 8, 128)| (m, 8, 8, 128)| DIS_BLOCK(128, 128, False), RELU GSP |
| Output Layer   | (m, 128)      | (m, 1)        | FC(128, 1)                 |

Table A5: Generator of BigGAN [6] used for CIFAR10 [21] image synthesis.

| Layer          | Input         | Output        | Operation                  |
|----------------|---------------|---------------|----------------------------|
| Input Layer    | (m, 20)       | (m, 6144)     | FC(20, 6144)               |
| Reshape Layer  | (m, 6144)     | (m, 4, 4, 384)| RESHAPE                   |
| Hidden Layer   | (m, 4, 4, 384)| (m, 8, 8, 384)| BIGG_BLOCK(384, 384, True, 20, 128) |
| Hidden Layer   | (m, 8, 8, 384)| (m, 16, 16, 384)| BIGG_BLOCK(384, 384, True, 20, 128) |
| Hidden Layer   | (m, 16, 16, 384)| (m, 16, 16, 384)| SELF-ATTENTION |
| Hidden Layer   | (m, 16, 16, 384)| (m, 32, 32, 384)| BIGG_BLOCK(384, 384, True, 20, 128) |
| Hidden Layer   | (m, 32, 32, 384)| (m, 32, 32, 3)| BN, RELU, CONV(384, 3, 3, 1) |
| Output Layer   | (m, 32, 32, 3)| (m, 32, 32, 3)| TANH                       |
Table A6: Discriminator of BigGAN [6] used for CIFAR10 [21] image synthesis.

| Layer     | Input            | Output          | Operation                              |
|-----------|------------------|-----------------|----------------------------------------|
| Input Layer | (m, 32, 32, 3)   | (m, 16, 16, 192)| DIS_BLOCK(3, 192, True)                |
| Hidden Layer | (m, 16, 16, 192) | (m, 16, 16, 192)| SELF-ATTENTION                         |
| Hidden Layer | (m, 8, 8, 192)   | (m, 8, 8, 192)  | DIS_BLOCK(192, 192, False)             |
| Hidden Layer | (m, 8, 8, 192)   | (m, 8, 8, 192)  | DIS_BLOCK(192, 192, False)             |
| Hidden Layer | (m, 8, 8, 192)   | (m, 192)        | RELU, GSP                              |
| Output Layer | (m, 192)        | (m, 1)          | FC(192, 1)                             |

Table A7: Generator of BigGAN [6] for Tiny ImageNet [22] image synthesis.

| Layer     | Input            | Output          | Operation                              |
|-----------|------------------|-----------------|----------------------------------------|
| Input Layer | (m,20)           | (m,20480)       | FC(20, 20480)                          |
| Reshape Layer | (m,20480)   | (m,4,4,1280)    | RESHAPE                                |
| Hidden Layer | (m,4, 4, 1280)  | (m,8, 8, 640)   | BIGG_BLOCK(1280, 640, True, 20, 128)   |
| Hidden Layer | (m,8, 8, 640)   | (m,16, 16, 320) | BIGG_BLOCK(640, 320, True, 20, 128)    |
| Hidden Layer | (m,16, 16, 320) | (m,32, 32, 160) | BIGG_BLOCK(320, 160, True, 20, 128)    |
| Hidden Layer | (m,32, 32, 160) | (m,32, 32, 160) | SELF-ATTENTION                         |
| Hidden Layer | (m,32, 32, 160) | (m,64, 64, 80)  | BIGG_BLOCK(160, 80, True, 20, 128)     |
| Hidden Layer | (m,64, 64, 80)  | (m,64, 64, 3)   | BN, RELU, CONV(80,3, 3, 1)             |
| Output Layer | (m,32, 32, 3)   | (m,32, 32, 3)   | TANH                                   |

Table A8: Discriminator of BigGAN [6] for Tiny ImageNet [22] image synthesis.

| Layer     | Input            | Output          | Operation                              |
|-----------|------------------|-----------------|----------------------------------------|
| Input Layer | (m, 64, 64, 3)  | (m, 32, 32, 80) | DIS_BLOCK(3, 80, True)                 |
| Hidden Layer | (m, 32, 32, 80) | (m, 32, 32, 80) | SELF-ATTENTION                         |
| Hidden Layer | (m, 16, 16, 160)| (m, 8, 8, 320)  | DIS_BLOCK(160, 320, True)              |
| Hidden Layer | (m, 8, 8, 320)  | (m, 4, 4, 640)  | DIS_BLOCK(320, 640, True)              |
| Hidden Layer | (m, 4, 4, 640)  | (m, 4, 4, 1280) | DIS_BLOCK(640, 1280, False)            |
| Hidden Layer | (m, 4, 4, 1280) | (m, 1280)       | RELU, GSP                              |
| Output Layer | (m, 1280)       | (m, 1)          | FC(128, 1)                             |
C Hyperparameter Setup

Table A9: Hyperparameter values used for experiments. Settings (B, C, E) and (F) are the settings used in [54, 2, 7] and [5], respectively. We conduct experiments with CIFAR10 [21] using setting (A, B, C, D, E) and with Tiny ImageNet [22] using setting (F).

| Setting | $\alpha_1$ | $\alpha_2$ | $\beta_1$ | $\beta_2$ | $n_{\text{dis}}$ |
|---------|------------|------------|------------|------------|-----------------|
| A       | 0.0001     | 0.0001     | 0.5        | 0.999      | 2               |
| B       | 0.0001     | 0.0001     | 0.5        | 0.999      | 1               |
| C       | 0.0002     | 0.0002     | 0.5        | 0.999      | 1               |
| D       | 0.0002     | 0.0002     | 0.5        | 0.999      | 2               |
| E       | 0.0002     | 0.0002     | 0.5        | 0.999      | 5               |
| F       | 0.0004     | 0.0001     | 0.0        | 0.999      | 1               |

Choosing a proper hyperparameter setup is crucial to train GANs. In this paper, we conduct experiments using six settings with Adam optimizer [39]. $\alpha_1$ and $\alpha_2$ are learning rates of the discriminator and generator. $\beta_1$ and $\beta_2$ are hyperparameters of Adam optimizer to control exponential decay rates of moving averages. $n_{\text{dis}}$ is the number of discriminator iterations per single generator iteration. For the Contrastive coefficient $\lambda$ (see Algorithm 1), the value is fixed at 1.0 for fair comparison with [19, 17]. In all experiments, we use the temperature $t = 1.0$. Experiments over temperature are displayed in Fig. A1. Besides, we apply moving averages of the generator’s weights used in [55, 56, 57] after 20,000 generator iterations with the decay rate of 0.9999. The settings (B, C, E) are known to give satisfactory performances on CIFAR10 [21] in previous papers [54, 2, 7]. Since Heusel et al. [40] and Zhang et al. [5] have shown that two time-scale update (TTUR) can converge to a stationary local Nash equilibrium [27], we adopt the hyperparameter setup used in [5] (setting F) to generate realistic images on Tiny ImageNet [22] dataset.

Figure A1: Change of FID values as the temperature increases. Experiments are executed three times, and the means and standard deviations are represented by the blue dots and solid lines, respectively.

Experimental setup used for Table 1 in the main paper: Experiments on CIFAR10 dataset are performed three times with different random seeds using the settings (A, B, C, D, E) with the batch size of 64. We stop training GANs with SNDCGAN, SNResGAN, and BigGAN architectures after 200k, 100k, 80k generator updates, respectively. Also, we report performances of the hyperparameter settings that showed the lowest FID values by mean. Experiments on Tiny ImageNet dataset is conducted once until 100k generator updates using the setting (F) with the batch size of 256 and BigGAN architecture (see Table A7 and Table A8). The hyperparameter settings: C, D, E, show the best performance in SNDCGAN [4], SNResGAN [4], and BigGAN [6], respectively. We reason that as the capacity of the model increases, training GANs becomes more difficult; thus, it requires more discriminator updates. Moreover, we experimentally identify that updating discriminator more times does not always produce better performance, but it might be related to the model capacity.

Experimental setup used for Table 2 in the main paper: FID values on CIFAR10 dataset are reported using the setting (E) with the batch size of 64. Also, the experiments on the Tiny ImageNet
are conducted using the setting (F) with the batch size of 1024. All other settings not noticed here are the same as the experimental setup for Table 1 above.

Experimental setup used for Table 3 in the main paper: All ablation results are reported using the setting (F), and models with consistency regularization (CR) [7] are trained with the coefficient of 10.0. We use an Intel(R) Xeon(R) Silver 4114 CPU, four NVIDIA Geforce RTX 2080 Ti GPUs, and PyTorch DataParallel library to measure time per iteration. All other settings not noticed here are the same as the experimental settings used for Table 1.

D Nonlinear Projection and Batch Size

![Figure A2](image1.png)

(a)

![Figure A2](image2.png)

(b)

Figure A2: (a) FID values of ContraGANs with different projection layers and embedding dimensionalities. (b) the change in FID value as the batch size increases. The experiments (a) and (b) are conducted using the setting (D).

We study the effect of a projection layer $h : \mathbb{R}^k \rightarrow \mathbb{S}^d$ that is introduced in Sec. 3.2. We change types of the layer (linear vs. nonlinear) and increase the dimensionality of projected embeddings, $d$ on CIFAR10 dataset. Fig. A2a shows the overview of FID values. All experiments are conducted using 3 different architectures: SNDCGAN, SNResGAN, and BigGAN that are equipped with spectral normalization. We also run the experiments using three different random seeds and do not apply moving averages of the generator’s weights. SNDCGAN with the liner projection layer projects latent feature onto the 1024 dimensional space. This configuration shows higher FID than the nonlinear counterpart, but ContraGANs with a nonlinear projection layer generally give lower FIDs. Although GANs are known to need careful hyperparameter selection, our ContraGAN not seems to be sensitive to the type and dimensionality of the projection layer.

Figure A2b shows the change in FID value as the batch size increases. Experiments conducted by Brock et al. [6] have demonstrated that increasing the batch size enhances image generation performance on ImageNet dataset [18]. However, as shown in Fig. A2b, optimal batch sizes for CIFAR10 and Tiny ImageNet are 64 and 1024, respectively. Based on these results, we can deduce that increasing batch size does not always give the best synthesis results. We presume that this phenomenon is related to the number of classes used for the training.

E Number of Classes used for Training

In this chapter, we quantitatively show that ContraGAN can generate more diverse images than the others when the number of classes for training increases. To do this, we make subsets of Tiny ImageNet that contain only 25%, 50%, and 75% of the total classes. For evaluation, we train ACGAN [19], cGAN [17], and ContraGAN (Ours) using the setting (F) with the batch size of 256. Lastly, we compute the inception embedding statistics of the subsets to calculate FID values. As shown in Fig. A3, ContraGAN performs better than the others, while FID values of cGAN and ACGAN increase. We conjecture that it is because one-hot vectors of ACGAN are not flexible than class embeddings; thus, the optimization for the adversarial game becomes harder (see Fig. 1d,
Besides, the experiment shows that considering data-to-data relations is a good strategy for the conditional image synthesis task.

Figure A3: Change of log(FID) values as the number of classes increases. The proposed ContraGAN benefits from larger number of classes. Experiments are performed using Tiny ImageNet dataset.

### FID Implementations

FID is a widely used metric to evaluate the performance of a GAN model. Since calculating FID requires a pre-trained inception-V3 network [42], many implementations use Tensorflow [43] or PyTorch [58] libraries. Among them, the TensorFlow implementation [23] for FID measurement is widely used. We use the PyTorch implementation for FID measurement [44], instead. In this section, we show that the PyTorch based FID implementation [44] used in our work provides almost the same results with the TensorFlow implementation. The results are summarized in Table A10.

**Table A10: Comparison of TensorFlow and PyTorch FID implementations.**

| FID implementation | ContraGAN CIFAR10 | ContraGAN Tiny ImageNet |
|--------------------|-------------------|-------------------------|
| TensorFlow         | 10.308            | 26.924                  |
| PyTorch            | 10.304            | 27.131                  |

### Qualitative Results

This section presents images generated by various conditional image synthesis frameworks. Figure A4, A5, and A6 show the synthesized images using CIFAR10 dataset. Figure A7, A8, A9, and A10 show the synthesized images using Tiny ImageNet dataset. As shown in Fig. A9, our approach can achieve favorable FID compared to the other baseline approaches. With consistency regularization, the results of our approach can be improved as shown in Fig. A10.
Figure A4: Examples generated by ACGAN [19] trained on CIFAR10 dataset [21] (FID=11.111).

Figure A5: Examples generated by cGAN [17] on CIFAR10 dataset [21] (FID=10.933).

Figure A6: Examples generated by ContraGAN (Ours) on CIFAR10 dataset [21] (FID=10.188).
Figure A7: Examples generated by cGAN [17] on Tiny ImageNet dataset [22] (FID=34.090).
Figure A8: Examples generated by cGAN [17] with consistency regularization [7] on Tiny ImageNet dataset [22] (FID=38.231).
Figure A9: Examples generated by ContraGAN (Ours) on Tiny ImageNet dataset [22] (FID=30.286).
Figure A10: Examples generated by ContraGAN (Ours) with consistency regularization [7] on Tiny ImageNet dataset [22] (FID=27.018).