# Omni-GAN and Omni-INR-GAN
# Supplementary Material

## Table of Contents



## A. Derivation from Unified Loss to Omni-loss.

### A.1. Derivation of Omni-loss

The unified loss [18] is defined as

$$\begin{aligned}\mathcal{L}_{\text{uni}} &= \log\left[1+\sum_{s_i^{(n)}\in\mathbb{S}_{\text{neg}}}\sum_{s_j^{(p)}\in\mathbb{S}_{\text{pos}}} e^{\left(\gamma\left(s_i^{(n)}-s_j^{(p)}+m\right)\right)}\right] \\ &= \log\left[1+\sum_{s_i^{(n)}\in\mathbb{S}_{\text{neg}}} e^{\left(\gamma\left(s_i^{(n)}+m\right)\right)}\sum_{s_j^{(p)}\in\mathbb{S}_{\text{pos}}} e^{\left(\gamma\left(-s_j^{(p)}\right)\right)}\right],\end{aligned} \tag{1}$$

where $\gamma$ stands for a scale factor, and $m$ for a margin between positive and negative scores. $\mathbb{S}_{\text{pos}} = \{s_1^{(p)}, \cdots, s_K^{(p)}\}$ and $\mathbb{S}_{\text{neg}} = \{s_1^{(n)}, \cdots, s_L^{(n)}\}$ denote positive score set and negative score set, respectively. Eq. (1) aims to maximize $s^{(p)}$ and to minimize $s^{(n)}$.

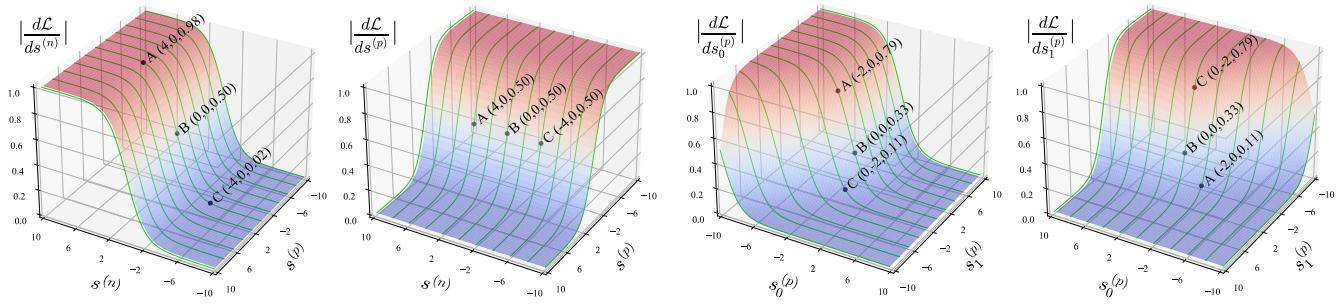
The omni-loss is defined as

$$\mathcal{L}_{\text{omni}}\left(\boldsymbol{x}, \boldsymbol{y}\right) = \log\left(1+\sum_{i\in\mathbb{I}_{\text{neg}}} e^{s_i(\boldsymbol{x})}\right) + \log\left(1+\sum_{j\in\mathbb{I}_{\text{pos}}} e^{-s_j(\boldsymbol{x})}\right), \tag{2}$$

where $\mathbb{I}_{\text{neg}}$ is a set consisting of indexes of negative scores ($|\mathbb{I}_{\text{neg}}| = L$), and $\mathbb{I}_{\text{pos}}$ consists of indexes of positive scores ($|\mathbb{I}_{\text{pos}}| = K$).

Eq. (2) is a special case of Eq. (1), which has been proved by [17]. For the convenience of readers in the English community, we provide our proof here. Let $\gamma$ be 1 and $m$ be 0, then

$$\begin{aligned}\mathcal{L}_{\text{uni}} &= \log\left[1+\sum_{s_i^{(n)}\in\mathbb{S}_{\text{neg}}} e^{s_i^{(n)}}\sum_{s_j^{(p)}\in\mathbb{S}_{\text{pos}}} e^{-s_j^{(p)}}\right] \\ &= \log\left[1+e^{\log\left(\sum_{s_i^{(n)}\in\mathbb{S}_{\text{neg}}} e^{s_i^{(n)}}\sum_{s_j^{(p)}\in\mathbb{S}_{\text{pos}}} e^{-s_j^{(p)}}\right)}\right] \\ &= \text{softplus}\left[\log\left(\sum_{s_i^{(n)}\in\mathbb{S}_{\text{neg}}} e^{s_i^{(n)}}\sum_{s_j^{(p)}\in\mathbb{S}_{\text{pos}}} e^{-s_j^{(p)}}\right)\right] \\ &= \text{softplus}\left[\log\left(\sum_{s_i^{(n)}\in\mathbb{S}_{\text{neg}}}\sum_{s_j^{(p)}\in\mathbb{S}_{\text{pos}}} e^{s_i^{(n)}-s_j^{(p)}}\right)\right] \\ &\approx \left[\log\left(\sum_{s_i^{(n)}\in\mathbb{S}_{\text{neg}}}\sum_{s_j^{(p)}\in\mathbb{S}_{\text{pos}}} e^{s_i^{(n)}-s_j^{(p)}}\right)\right]_+,\end{aligned} \tag{3}$$




(a) (b)

Figure 1: Gradients of the omni-loss. (a) Gradients w.r.t. $s^{(n)}$ and $s^{(p)}$ are independent. (b) Gradients w.r.t. $s_k^{(p)}$, $\{k = 0, 1, \dots\}$, are automatically balanced. Please see the text in Sec. A.2 for details. This figure is inspired by [18].

where $[\cdot]_+$ means $\max(\cdot, 0)$.

According to $\log \sum_{i=1}^{n} e^{x_i} \approx \max(x_1, x_2, \dots, x_n)$, we get

$$\mathcal{L}_{\text{uni}} \approx \left[ \max_{s_i^{(n)} \in \mathbb{S}_{\text{neg}}, s_j^{(p)} \in \mathbb{S}_{\text{pos}}} s_i^{(n)} - s_j^{(p)} \right]_+, \tag{4}$$

where minimizing Eq. (4) makes the smallest $s_j^{(p)}$ greater than the largest $s_i^{(n)}$.

Let $\mathbb{S}_{\text{pos}}^{(1)} = \{0\}$ and $\mathbb{S}_{\text{neg}}^{(1)} = \{s_1^{(n)}, \cdots, s_L^{(n)}\}$. According to Eq. (3), we get

$$\begin{aligned} \mathcal{L}_{\text{uni}}^{(1)} &= \log \left[ 1 + \sum_{s_i^{(n)} \in \mathbb{S}_{\text{neg}}^{(1)}} e^{s_i^{(n)}} \sum_{s_j^{(p)} \in \{0\}} e^{-s_j^{(p)}} \right] \\ &= \log \left[ 1 + \sum_{s_i^{(n)} \in \mathbb{S}_{\text{neg}}^{(1)}} e^{s_i^{(n)}} e^{0} \right] \\ &= \log \left[ 1 + \sum_{s_i^{(n)} \in \mathbb{S}_{\text{neg}}^{(1)}} e^{s_i^{(n)}} \right], \end{aligned} \tag{5}$$

where from Eq. (4) we know that minimizing Eq. (5) makes $s_i^{(n)}$ less than 0.

Let $\mathbb{S}_{\text{pos}}^{(2)} = \{s_1^{(p)}, \cdots, s_K^{(p)}\}$ and $\mathbb{S}_{\text{neg}}^{(2)} = \{0\}$. According to Eq. (3), we get

$$\begin{aligned} \mathcal{L}_{\text{uni}}^{(2)} &= \log \left[ 1 + \sum_{s_i^{(n)} \in \{0\}} e^{s_i^{(n)}} \sum_{s_j^{(p)} \in \mathbb{S}_{\text{pos}}^{(2)}} e^{-s_j^{(p)}} \right] \\ &= \log \left[ 1 + e^{0} \sum_{s_j^{(p)} \in \mathbb{S}_{\text{pos}}^{(2)}} e^{-s_j^{(p)}} \right] \\ &= \log \left[ 1 + \sum_{s_j^{(p)} \in \mathbb{S}_{\text{pos}}^{(2)}} e^{-s_j^{(p)}} \right] \end{aligned} \tag{6}$$

where minimizing Eq. (6) makes $s_j^{(p)}$ greater than 0.

Adding Eq. (5) and Eq. (6), we get

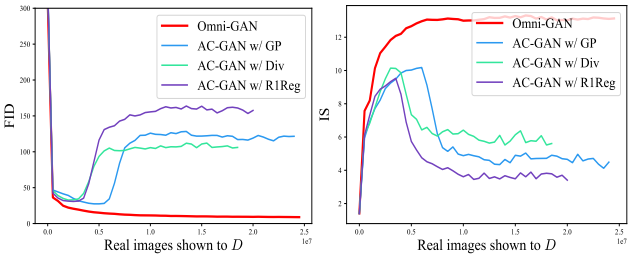
$$\mathcal{L}_{\text{omni}} = \log \left[ 1 + \sum_{s_i^{(n)} \in \mathbb{S}_{\text{neg}}^{(1)}} e^{s_i^{(n)}} \right] + \log \left[ 1 + \sum_{s_j^{(p)} \in \mathbb{S}_{\text{pos}}^{(2)}} e^{-s_j^{(p)}} \right], \tag{7}$$

where minimizing Eq. (7) makes $s_i^{(n)}$ less than 0 and $s_j^{(p)}$ greater than 0. We finish the derivation.

## A.2. Gradient Analysis

The gradients of omni-loss have two properties: on one hand, the gradients w.r.t. $s^{(n)}$ and $s^{(p)}$ are independent; on the other hand, the gradients w.r.t. $s_k^{(p)}$ (or $s_k^{(n)}$), $\{k = 0, 1, \dots\}$, are automatically balanced. To illustrate these properties, we visualize the gradients of omni-loss. Fig. 1a shows a case that only contains one $s^{(n)}$ and one $s^{(p)}$. $A$, $B$, and $C$ have the same $s^{(p)}$, which is 0, but different $s^{(n)}$ (*i.e.*, $4, 0, -4$, respectively). As a result, the gradients w.r.t. $s^{(p)}$ at these three points are the same (*i.e.*, 0.5). Nevertheless, the gradients w.r.t. $s^{(n)}$ at these three points are different. For example, the gradient w.r.t. $s^{(n)}$ at $A$ is largest (equal to 0.98). The reason for this is that the objective of omni-loss is to minimize $s^{(n)}$. Thus the larger the $s^{(n)}$, the larger the gradient w.r.t. $s^{(n)}$.

In Fig. 1b, we show the ability of omni-loss to automatically balance gradients. We consider a case with only two positive labels, namely $s_0^{(p)}$ and $s_1^{(p)}$. We can observe that for $A$, its $s_0^{(p)}$ is smaller than $s_1^{(p)}$ (*i.e.*, -2 vs. 0). As a result, the gradients w.r.t. $s_0^{(p)}$ is larger than that w.r.t. $s_1^{(p)}$ (*i.e.*, 0.79 vs. 0.11), meaning that the omni-loss try to increase $s_0^{(p)}$ with higher superiority. A similar analysis applies to $C$ as well. For $B$, since $s_0^{(p)}$ and $s_1^{(p)}$ are equal, the gradients of them are also equal (0.33).




(a) FID on CIFAR100 (b) IS on CIFAR100

Figure 2: FID and IS on CIFAR100. We test three gradient penalty methods (*i.e.*, WGAN-GP, WGAN-div, and R1 regularization), none of which can alleviate the collapse issue of AC-GAN.

## B. Gradient Penalty for Classification-based cGANs

We investigate whether gradient penalty will alleviate early collapse. We chose AC-GAN [11], the currently widely known classification-based cGAN, as the testbed, and evaluate three gradient penalty methods: WGAN-GP [4], WGAN-div [8], and R1 regularization [23]. Because cGANs are more likely to collapse when the number of categories is large, we evaluate them on CIFAR100 instead of CIFAR10. As shown in Fig. 2, none of the three gradient penalty methods can prevent AC-GAN from collapsing. We emphasize that computing gradient penalties will introduce additional computational overhead during GAN's training, which is very unfriendly to large-scale datasets such as ImageNet. However, weight decay effectively alleviates the collapse problem without adding any additional training overhead.

## C. Improved AC-GAN (ImAC-GAN)

Auxiliary classifier GAN (AC-GAN) [11] uses an auxiliary classifier to enhance the standard GAN model. Its objective function consists of two parts: the GAN loss, $\mathcal{L}_{\text{GAN}}$, and the classification loss, $\mathcal{L}_{\text{cls}}$:

$$\begin{aligned}\mathcal{L}_{\text{GAN}} =& \mathbb{E}\left[\log P\left(\mathrm{g}=\text{real} \mid \boldsymbol{x}_{\text{real}}\right)\right]+ \\ & \mathbb{E}\left[\log P\left(\mathrm{g}=\text{fake} \mid \boldsymbol{x}_{\text{fake}}\right)\right],\end{aligned} \tag{8}$$

$$\mathcal{L}_{\text{cls}} = \mathbb{E}\left[\log P\left(\mathrm{g}=c \mid \boldsymbol{x}_{\text{real}}\right)\right]+\mathbb{E}\left[\log P\left(\mathrm{g}=c \mid \boldsymbol{x}_{\text{fake}}\right)\right], \tag{9}$$

where g is a random variable denoting the class label and $c$ is the ground truth label of $\boldsymbol{x}$. $\boldsymbol{x}_{\text{real}}$ and $\boldsymbol{x}_{\text{fake}}$ represent a real image and a generated image respectively. The discriminator $D$ of AC-GAN is trained to maximize $\mathcal{L}_{\text{GAN}}+\mathcal{L}_{\text{cls}}$, and the generator is trained to maximize $\mathcal{L}_{\text{cls}}-\mathcal{L}_{\text{GAN}}$.

The discriminator loss of AC-GAN is not optimal. We give a slightly modified version below. Suppose the dataset owns $C$ categories, then the discriminator is trained to maximize

$$\begin{aligned}\mathcal{L}_D =& \mathcal{L}_{\text{GAN}}+ \\ & \mathbb{E}\left[\log P\left(\mathrm{g}=c \mid \boldsymbol{x}_{\text{real}}\right)\right]+\mathbb{E}\left[\log P\left(\mathrm{g}=C \mid \boldsymbol{x}_{\text{fake}}\right)\right],\end{aligned} \tag{10}$$

where $c \in \{0,1,\ldots,C-1\}$ is the ground truth class label of $\boldsymbol{x}_{\text{real}}$, and $\mathrm{g}=C$ means that $\boldsymbol{x}_{\text{fake}}$ belongs to the fake class. To sum up, we use an additional class to represent the generated image. In practice, this is achieved by setting the dimension of the fully connected layer of the auxiliary classification layer to be $C+1$ rather than $C$.

The objective function of the generator is consistent with that of the original AC-GAN, *i.e.*, maximizing

$$\mathcal{L}_G = -\mathcal{L}_{\text{GAN}}+\mathbb{E}\left[\log P\left(\mathrm{g}=c_{\text{fake}} \mid \boldsymbol{x}_{\text{fake}}\right)\right], \tag{11}$$

where $c_{\text{fake}}$ is the class label used by the generator to generate $\boldsymbol{x}_{\text{fake}}$.

We name this improved version of AC-GAN ImAC-GAN. As shown in the paper, ImAC-GAN is comparable to Omni-GAN, both of which achieve superior performance compared to projection-based cGANs. However, because ImAC-GAN uses cross-entropy as the loss function for classification, it can only handle the case where the sample has a positive label. Omni-GAN uses omni-loss, essentially a multi-label classification loss, which naturally supports handling samples with one positive label or multiple positive labels. We will give an example of generating images with multiple positive labels in Sec. E.

## D. Technical Details of Omni-INR-GAN

Learning image prior model is helpful for image restoration and manipulation, such as denoising, inpainting, and harmonizing. Deep generative prior (DGP) [12] showed the potential of employing the generator prior captured by a pre-trained GAN model (*i.e.*, a BigGAN model trained on a large-scale image dataset, ImageNet). However, BigGAN can only output images with a fixed aspect ratio, limiting the practical application of DGP. To make the pre-trained GAN model more flexible for downstream tasks, we propose a new GAN named Omni-INR-GAN, which can output images with any aspect ratio and any resolution.

Images are usually represented by a set of pixels with fixed resolution. A popular method named implicit neural representation (INR) is prevalent in the 3D field [13, 9, 2]. Recently, people introduced the INR method to 2D images [1, 16]. As shown in Fig. 3 (a), the INR of an image directly maps $(x, y)$ coordinates to image's RGB pixel values. Since the coordinates are continuous, once we get the INR of an image, we can get images of arbitrary resolutions by sampling different numbers of coordinates.

Inspired by the local implicit image function (LIIF) [1], we use INR to enhance Omni-GAN, with the goal of en-

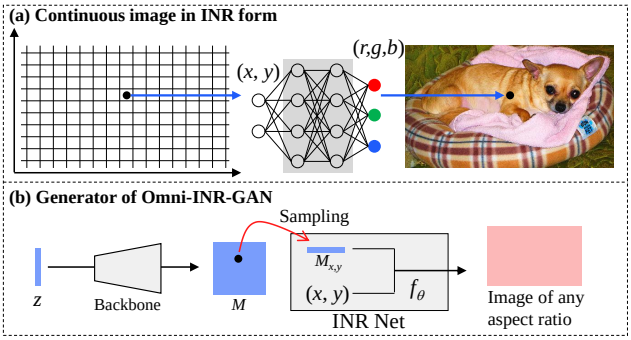



Figure 3: (a) An example of an image represented in INR form. A fully connected network maps coordinates $(x, y)$ to pixel values $(r, g, b)$. (b) Using an INR network to enhance the generator so that the generator can output images with any resolution and any aspect ratio.

abling the generator to output images with any aspect ratios and any resolution. We name our method Omni-INR-GAN. As shown in Fig. 3 (b), we keep the backbone of the generator network unchanged and employ an INR network for the output layer. Let $\boldsymbol{M} \in \mathbb{R}^{C\times H\times W}$ represent the output feature map of the backbone, $f_\theta$ be the implicit neural function. Then the RGB signal at $(x, y)$ coordinate is given by $s = f_\theta(\boldsymbol{M}_{x,y}, x, y)$, where $\boldsymbol{M}_{x,y}$ stands for the feature vector at $(x, y)$. Note that since $x$ and $y$ can be any real numbers, $\boldsymbol{M}_{x,y}$ may not exist in $\boldsymbol{M}$. In such a case, we adopt the bilinear interpolation of the four feature vectors near $(x, y)$ as the feature at $(x, y)$.

Omni-INR-GAN can generate images with any aspect ratio, so as to be more friendly to downstream tasks like image restoration and manipulation. After trained on the large-scale dataset ImageNet, Omni-INR-GAN can be combined with DGP to do restoration tasks. Omni-INR-GAN eliminates cropping operations before image restoration, making it possible to repair the entire image directly. Since the generator has seen considerable natural images, utilizing the generator prior can facilitate downstream tasks significantly.

## E. An Example of Multi-label Discriminator

Omni-loss is essentially a multi-label classification loss and naturally supports classification with multiple positive labels. To verify the ability of Omni-GAN for generating samples with multiple positive labels, we construct a mixed dataset containing images of digits from two distinct domains, namely MNIST [7] of handwritten digits and SVHN [10] of house numbers. Some example images from the datasets are shown in Fig. 4. In this setting, the discriminator needs to predict three attributes, class (recognizing digits), domain, and reality.

Let us take images of MNIST as an example, and show how to set the loss for the discriminator. As for SVHN, the case is analogous. Suppose $\boldsymbol{x}_{\text{real}}$ is an image sampled from MNIST, its multi-label vector is given by

$$\boldsymbol{y}_{\text{real}} = [\underbrace{-1,\ldots,1_{\text{gt}},\ldots,-1}_{\text{class}}, \underbrace{1_{\text{mnist}},-1}_{\text{domain}}, \underbrace{1_{\text{real}},-1}_{\text{reality}}], \quad (12)$$

where $-1$ means the corresponding score belongs to the negative set, and 1 to the positive set. As can be seen, $\boldsymbol{y}_{\text{real}}$ possesses three positive labels. The multi-label vector for $\boldsymbol{x}_{\text{fake}}$ is then given by

$$\boldsymbol{y}_{\text{fake}} = [\underbrace{-1,\ldots,-1,\ldots,-1}_{\text{class}}, \underbrace{-1,-1}_{\text{domain}}, \underbrace{-1,1_{\text{fake}}}_{\text{reality}}], \quad (13)$$

which is a one-hot vector with the last element being 1. The discriminator loss is given by

$$\begin{aligned}\mathcal{L}_D =& \mathbb{E}_{\boldsymbol{x}_{\text{real}}\sim p_{\text{d}}}\left[\mathcal{L}_{\text{omni}}\left(\boldsymbol{x}_{\text{real}}, \boldsymbol{y}_{\text{real}}\right)\right] \\ &+ \mathbb{E}_{\boldsymbol{x}_{\text{fake}}\sim p_{\text{g}}}\left[\mathcal{L}_{\text{omni}}\left(\boldsymbol{x}_{\text{fake}}, \boldsymbol{y}_{\text{fake}}\right)\right].\end{aligned} \quad (14)$$

For generator, its goal is to cheat the discriminator. The multi-label vector for $\boldsymbol{x}_{\text{fake}}$ is given by

$$\boldsymbol{y}^{(\text{G})}_{\text{fake}} = [\underbrace{-1,\ldots,1_{\text{G}},\ldots,-1}_{\text{class}}, \underbrace{1_{\text{mnist}},-1}_{\text{domain}}, \underbrace{1_{\text{real}},-1}_{\text{reality}}], \quad (15)$$

where $1_{\text{G}}$ is 1 if its index in the vector is equal to the label adopted by the generator to generate $\boldsymbol{x}_{\text{fake}}$, otherwise $-1$. The generator loss is given by.

$$\mathcal{L}_G = \mathbb{E}_{\boldsymbol{x}_{\text{fake}}\sim p_{\text{g}}}\left[\mathcal{L}_{\text{omni}}\left(\boldsymbol{x}_{\text{fake}}, \boldsymbol{y}^{(\text{G})}_{\text{fake}}\right)\right]. \quad (16)$$

We experimentally found that this multi-label discriminator can instruct the generator to generate images from different domains. Some generated images are shown in Fig. 5. We must emphasize that this is only a preliminary experiment to verify the function of the multi-label discriminator. We look forward to applying the multi-label discriminator to other tasks in the future, such as translation between images in different domains, domain adaptation, *etc*.

## F. Additional Results on CIFAR

### F.1. Over-fitting of the Discriminator

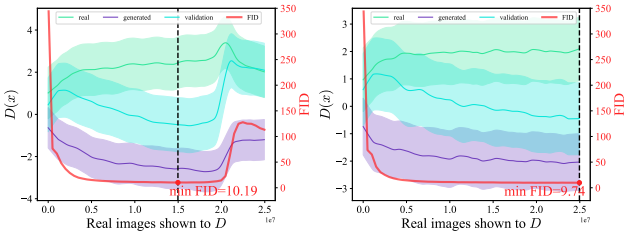
Karras *et al*. [6] found that the discriminator overfits the training dataset, which will lead to incorrect gradients provided to the generator. Thus the training diverges. To verify that the collapse of the projection-based cGAN is due to the over-fitting of the discriminator, we plotted the scalar output of the discriminator, $D(\boldsymbol{x})$, over the course of training. We utilized the test set of CIFAR100 containing $10,000$ images as the verification set, which was not used in the training.

(a) MNIST (b) SVHN

Figure 4: Real images sampled from the dataset.

(a) MNIST (b) SVHN

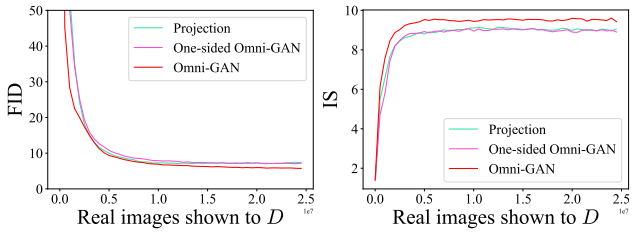
Figure 5: Images generated by a generator which is guided by a multi-label discriminator.




(a) (b)

Figure 6: The raw logits of $D(\boldsymbol{x})$ and the corresponding FID score of a projection-based cGAN are plotted in the same figure. The black dashed line indicates where the minimum FID is reached. (a) training without weight decay. (b) training with weight decay. The figures of $D(\boldsymbol{x})$ are inspired by [6].

As shown in Fig. 6a, obviously, as training progresses, the $D(\boldsymbol{x})$ of the validation set tends to that of the generated images, substantiating that the discriminator overfits the training data. We also plotted the FID curve in the same figure. We can see that the training commences diverging when showing about 20M real images (*i.e.*, around 400 epoch) to the discriminator. The best FID is obtained when approximately 15M real images are shown to the discriminator.




(a) (b)

Figure 7: One-sided Omni-GAN is on par with projection-based BigGAN on CIFAR10, proving that one-sided Omni-GAN indeed belongs to projection-based cGANs. Both of them are inferior to the classification-based cGAN, Omni-GAN.

In Fig. 6b, we show the $D(\boldsymbol{x})$ and FID after applying weight decay to the projection-based discriminator. We can find that although the discriminator still overfits the training data, the training dose not collapse during the whole training process (the minimum FID, 9.74, is reached at the end of the training).

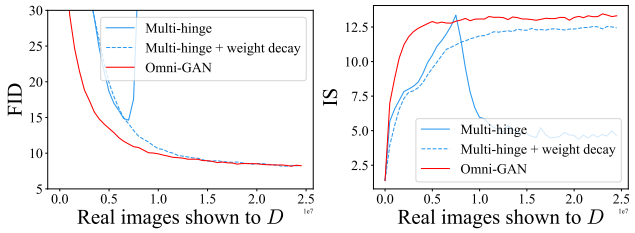
## F.2. Comparison of One-sided Omni-GAN and Projection-based GAN on CIFAR10

We provide the results of one-sided Omni-GAN and projection-based BigGAN on CIFAR10. As shown in Fig. 7, one-sided Omni-GAN is comparable to the projection-based BigGAN in terms of both FID and IS. This proves that one-sided Omni-GAN indeed belongs to projection-based cGANs. Both one-sided Omni-GAN and projection-based BigGAN are inferior to Omni-GAN. Because the only difference between one-sided Omni-GAN and Omni-GAN is whether the supervision is fully utilized, we conclude that the superiority of Omni-GAN lies in the full use of supervision.

## F.3. Comparison with Multi-hinge GAN

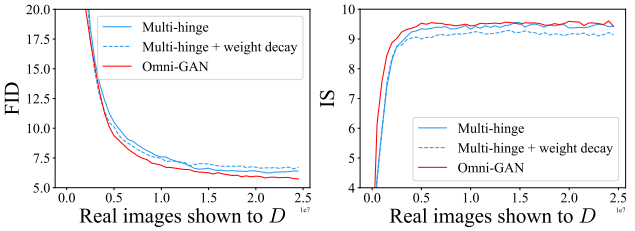
Multi-hinge GAN belongs to classification-based cGANs, and also suffers from the early collapse issue. We study whether weight decay is effective for Multi-hinge GAN. As shown in Fig. 8, original Multi-hinge GAN suffers a severe early collapse issue. After equipped with weight decay, Multi-hinge GAN enjoys a safe optimization and its FID is even comparable to that of Omni-GAN. However, its IS is worse than that of Omni-GAN.

Multi-hinge GAN combined with weight decay does not always perform well. The results on CIFAR10 are shown in Fig. 9. Weight decay deteriorates Multi-hinge GAN in terms of both FID and IS. However, Omni-GAN outperforms Multi-hinge GAN. In addition, omni-loss is more flexible than multi-hinge loss. It supports implementing a




(a) (b)

Figure 8: FID and IS on CIFAR100. Weight decay can eliminate the early collapse problem of Multi-hinge GAN on CIFAR100.




(a) (b)

Figure 9: FID and IS on CIFAR10. Weight decay deteriorates Multi-hinge GAN.

multi-label discriminator. As a result, we suggest first considering using Omni-GAN when choosing cGANs.

### F.4. Comparison with Other Regularization Methods

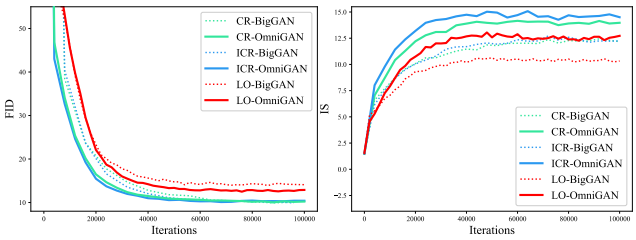



Figure 10: Results on CIFAR100. OmniGAN can be easily integrated with off-the-shelf methods and shows a consistent and clear improvement, especially for the IS. wd: weight decay.

It is very easy to combine Omni-GAN with other regularization methods, such as CR [26], ICR [27] and LOGAN [24]. As shown in Fig. 10, when combined with different methods, Omni-GAN is consistently better than BigGAN on CIFAR100. Note that unlike other experiments which are based on BigGAN [1], the implementations of CR, ICR, and LOGAN are based on StudioGAN [2]. FID and IS are computed using 10K test and 10K generated Images.

Although CR and ICR are effective regularization methods, they introduce additional computational overhead. Table 1 shows the properties of different regularization methods. Compared with other regularization, weight decay has negligible computational overhead. Moreover, weight decay is easily implemented by directly adjusting the optimizer rather than defining a new term in the loss function, thus it is computationally cheap.

| Regularization | GP[20] | R1Reg[38] | CR [68] | ICR | LO | DataAug | wd |
|---|---|---|---|---|---|---|---|
| Additional forward/backward | ✔ | | ✔ | ✔ | ✔ | | |
| Second-order gradient | ✔ | ✔ | | | | | |
| More iterations | | | | | | ✔ | |

| Method | BigGAN (baseline) | OmniGAN w/ wd | CR-BigGAN | ICR-BigGAN | LO-BigGAN |
|---|---|---|---|---|---|
| Time (seconds/1k iterations) | 643 | 669 (+4.0%) | 811 (+26.1%) | 1429 (+122.2%) | 5470 (+750.6%) |

Table 1: Comparison of regularizations. The training time is evaluated on CIFAR100. We refer to weight decay (wd) as a 'moderate' regularization in that it increases negligible training overhead compared to other regularizations.

### F.5. Applying Weight Decay to the Generator

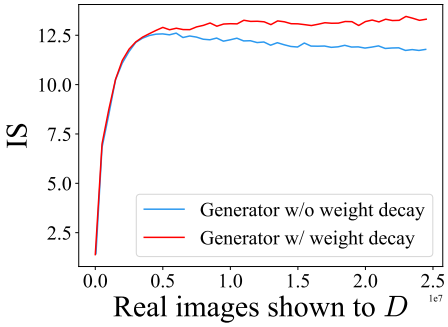



Figure 11: Applying weight decay to the generator. Applying weight decay to the discriminator helps alleviate the collapse issue, but the IS gradually decreases as the training progresses. Applying weight decay to the generator simultaneously solves this problem. Experiments are conducted on CIFAR100.

We found empirically that applying weight decay also to the generator can make training more stable. As shown in Fig. 11, although only applying weight decay to the discriminator can avoid the risk of collapse earlier, the IS has a trend of gradually decreasing as the training progresses. Fortunately, applying weight decay (set to be 0.001 in our most experiments) to the generator can solve this problem. This phenomenon seems to indicate that the generator is also at a risk of over-fitting.

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

[2] https://github.com/POSTECH-CVLab/PyTorch-StudioGAN

### F.6. How to Set the Weight Decay?

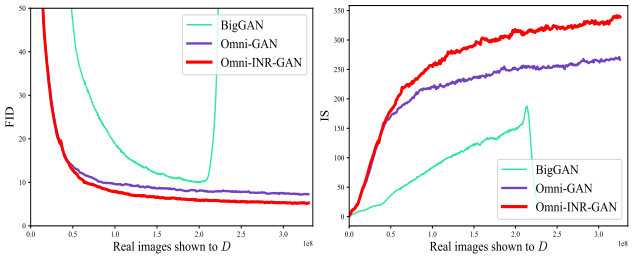
We did a grid search for the weight decay on CIFAR and found that its value is related to the size of the training dataset. For CIFAR100, there are only 500 images per class, and the weight decay is set to be 0.0005. For CIFAR10, there are 5000 images per class, and the weight decay is set to be 0.0001. For ImageNet, it is a large dataset with a considerable number of training data (approximate 1.2M). The weight decay is set to 0.00001. The conclusion is that the smaller the dataset, the higher the risk of overfitting for the discriminator. Then weight decay should be larger.

### F.7. Mathematical Explanation for Mode Collapse

During the paper review, a reviewer was interested in the mathematical analysis of why weight decay contribute to stabilize the training of cGANs. We provide a possible direction for the mathematical analysis. FAR-GAN [19] provides a mathematical analysis on mode collapse of GANs. In their opinion, there is an unbalanced generation and a vicious circle issue during GAN's training. Inspired by them, we can similarly analyze why weight decay can alleviate the training collapse of classification-based GANs. We will provide the analysis in the arXiv version of the paper once we have completed it.

## G. Additional Results on ImageNet

We provide convergence curves on ImageNet $256 \times 256$. As shown in Fig. 12, both Omni-GAN and Omni-INR-GAN converge faster than BigGAN, proving the effectiveness of combining strong supervision and weight decay. Omni-INR-GAN clearly outperforms Omni-GAN, showing its significant potential for future applications. In Fig. 13, we show the tradeoff curve of these methods using the truncation trick on ImageNet $256 \times 256$. Omni-INR-GAN is consistently superior to Omni-GAN and BigGAN.




(a) FID on ImageNet $256 \times 256$ (b) IS on ImageNet $256 \times 256$

Figure 12: FID and IS on ImgeNet $256 \times 256$. Omni-GAN and Omni-INR-GAN converge faster than the projection-based BigGAN. Omni-INR-GAN clearly outperforms Omni-GAN, showing its significant potential for future applications.

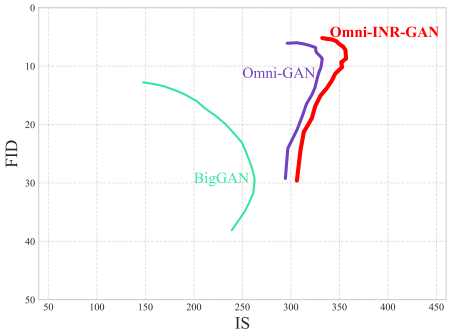



Figure 13: Tradeoff curves using truncation trick on ImageNet $256 \times 256$. We show truncation values from $\sigma = 0.05$ to $\sigma = 1$ with step being 0.05. Omni-INR-GAN outperforms Omni-GAN and BigGAN.

## H. Application to Image-to-Image Translation

Omni-GAN can be used for image-to-image translation tasks. We verify the effectiveness of Omni-GAN on semantic image synthesis [21, 15]. In particular, we replace the GAN loss of SPADE [14] with Omni-GAN's loss, and keep other hyper-parameters unchanged. The discriminator is a fully convolutional network, which is widely adopted by image-to-image translation tasks [14, 5, 22]. As shown in Fig. 14, the discriminator takes images as input and outputs feature maps with the number of channels being $C+2$. $C$ represents the number of classes which is analogous to that of the semantic segmentation task. 2 indicates there are two extra feature maps representing to what extent the input image is real or fake. We adopt nearest neighbor downsampling to downsample the label map to the same resolution as the output feature maps of the discriminator. Then we use the downsampled label map as the ground truth label, and apply a per-pixel omni-loss to the output feature maps of the discriminator.

We use Cityscapes dataset [3] as a testbed, and train models on the training set with size of $2,975$. The images is resized to $256 \times 512$. Models are evaluated by the mIoU of the generated images on the test set with 500 images. We use a pre-trained DRN-D-105 [25] as the segmentation model for the sake of evaluation. As shown in Table 2, Omni-GAN improves the mIoU score of SPADE from 62.21 to 65.07, substantiating that the synthesized images possess more semantic information. We believe that the improvement comes from the improved ability of the discriminator in distinguishing different classes, so that the generator receives better guidance and thus produces images with richer semantic information.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| SPADE [14] | *road* | *sidewalk* | *building* | *wall* | *fence* | *pole* | *traffic light* | *traffic sign* | *vegetation* | *terrain* |
| | 97.44 | 79.89 | 87.86 | 50.57 | 47.21 | 35.90 | 38.97 | 44.67 | 88.15 | 66.14 |
| | *sky* | *person* | *rider* | *car* | *truck* | *bus* | *train* | *motorcycle* | *bicycle* | mIoU |
| | 91.61 | 62.27 | 38.67 | 88.68 | 64.96 | 70.17 | 41.42 | 28.58 | 58.86 | 62.21 |
| + Omni-GAN | *road* | *sidewalk* | *building* | *wall* | *fence* | *pole* | *traffic light* | *traffic sign* | *vegetation* | *terrain* |
| | 97.57 | 81.62 | 88.58 | 53.39 | 50.47 | 35.88 | 41.08 | 46.75 | 89.31 | 67.00 |
| | *sky* | *person* | *rider* | *car* | *truck* | *bus* | *train* | *motorcycle* | *bicycle* | mIoU |
| | 92.14 | 63.97 | 41.99 | 89.91 | 71.06 | 74.21 | 56.16 | 33.99 | 61.23 | **65.07** |

Table 2: Semantic image synthesis using SPADE. Replacing the GAN used by SPADE with Omni-GAN can improve the quality of synthesized images.

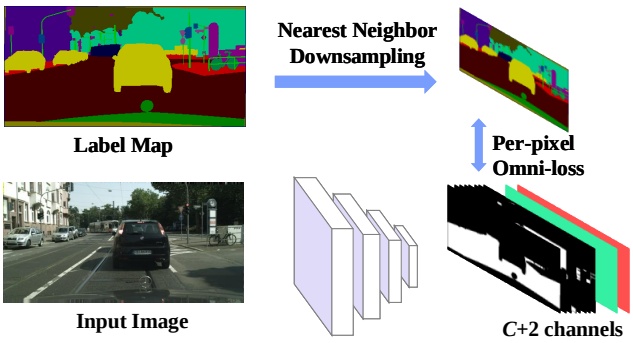



Figure 14: Combine omni-loss with a fully convolutional discriminator whose outputs are feature maps. In the figure, the green and red feature maps represent scores that the input images are real and fake, respectively. Omni-loss is applied to the output feature maps pixel-by-pixel.

# I. Application to Downstream Tasks

## I.1. Colorization and Super-resolution

Deep generative prior (DGP) [12] showed the potential of employing image prior captured by a pre-trained GAN model. Our colorization and super-resolution schemes are based on DGP. We first introduce the preliminary knowledge of DGP.

Suppose $\boldsymbol{x}$ is a natural image and $\phi$ is a degradation function, *e.g.*, gray transform for colorization and downsampling for super-resolution. Then $\hat{\boldsymbol{x}} = \phi(\boldsymbol{x})$ represents the degraded image, *i.e.*, a partial observation of the original image, $\boldsymbol{x}$. The goal of image restoration is to recover $\boldsymbol{x}$ from $\hat{\boldsymbol{x}}$ with the help of some statistical image prior of $\boldsymbol{x}$. DGP proposes employing the image prior stored in a pre-trained GAN's generator. The objective is defined as

$$\boldsymbol{\theta}^*, \boldsymbol{z}^* = \arg\min_{\boldsymbol{\theta},\boldsymbol{z}} \mathcal{L}(\hat{\boldsymbol{x}}, \phi(G(\boldsymbol{z};\boldsymbol{\theta}))), \tag{17}$$

where $\boldsymbol{z}$ is a noise vector. $G$ represents the generator in GAN and is parameterized by $\boldsymbol{\theta}$. $\mathcal{L}$ is a discriminator-based distance metric: $\mathcal{L}(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i\in\mathcal{I}} \|D(\mathbf{x}_1, i), D(\mathbf{x}_2, i)\|_1$. $D$ is the discriminator coupled with $G$. $\mathcal{I}$ is a index set for feature maps of different blocks of $D$. Note that both $G$ and $D$ have been trained on a large-scale natural image dataset. DGP employs the prior of $G$ by fine-tuning $\boldsymbol{\theta}$ and $\boldsymbol{z}$. After fine-tuning, we get the restored image $\boldsymbol{x}^* = G(\boldsymbol{z}^*;\boldsymbol{\theta}^*)$.

Although DGP has achieved noteworthy results in image restoration and manipulation, it has limitations due to the inflexibility of the pre-trained GAN model. For example, if DGP adopts a $128 \times 128$ BigGAN model, DGP must first crop the original image into image patch of size $128 \times 128$ before restoration, restricting its practical application. However, because Omni-INR-GAN can output images of any resolution, combining it with DGP can directly restore the original image.

We use Omni-INR-GAN pre-trained on ImageNet $256 \times 256$ for colorization and super-resolution. Eq. (17) is the objective. For colorization, $\hat{\boldsymbol{x}}$ is a grayscale image, and for super-resolution, $\hat{\boldsymbol{x}}$ is a low-resolution image. We resize the input image's short edge to 256 and keep the aspect ratio of the image unchanged. After fine-tuning, $\boldsymbol{x}^* = G(\boldsymbol{z}^*;\boldsymbol{\theta}^*)$ is the restored image. Because $G(\boldsymbol{z}^*;\boldsymbol{\theta}^*)$ represents $\boldsymbol{x}^*$ in the INR form, we can get the restored image at any resolution through $G(\boldsymbol{z}^*;\boldsymbol{\theta}^*)$. Therefore, Omni-INR-GAN is more friendly to downstream tasks.

## I.2. Reconstruction

We compare pre-trained GAN models for image reconstruction tasks. Specifically, we finetune the parameters of the generator to make it reconstruct given images. Note that we do not use mse or L1 loss, because these loss functions make it easy for the generator to overfit the given image, as long as the training iterations are enough. Instead, we only use the discriminator feature loss, because it has been proven to be very effective for utilizing the prior of the generator. For the dataset, we use 1k images sampled from the ImageNet validation set, which is the same as DGP's choice. Note that these data have not been used in GAN's training. We adopt the progressive reconstruction strategy of DGP [12], and finetune each GAN model for the same number of iterations.

## J. Implementation Details

We adopt BigGAN architectures of $128 \times 128$ and $256 \times 256$ in our experiments. Table 3, 4, 5, and 6 show the architectural details. Each experiment is conducted on eight v100 GPUs. Training Omni-GAN on ImageNet $128 \times 128$ and $256 \times 256$ took 25 days and 60 days, respectively. Training Omni-INR-GAN on ImageNet $128 \times 128$ and $256 \times 256$ took 27 days and 87 days, respectively. No collapse occurred during the entire training process. We have found experimentally that classification-based cGANs cannot set a large batch size like projection-based BigGAN. For all experiments of Omni-GAN and Omni-INR-GAN, the batch size is set to 256. We adopt the ADAM optimizer in all experiments, with betas being 0 and 0.999. The learning rates of the generator and discriminator are set to 0.0001 and 0.0004, respectively.

For $128 \times 128$ experiments, the generator and discriminator use non-local block at $64 \times 64$ resolution. The generator is updated once every time the discriminator is updated. The weight decay of the generator and discriminator are set to 0.001 and 0.00001, respectively. For Omni-INR-GAN, we removed the non-local block at $64 \times 64$ resolution of the discriminator. Because when Omni-INR-GAN is used for downstream tasks, the input image of the discriminator may be of any size, so the middle layer of the discriminator may not output $64 \times 64$ resolution features. Moreover, although Omni-INR-GAN can generate images of any resolution, we did not adopt a multi-scale training strategy. We found that multi-scale training led to training collapse. We think that a possible reason is that multi-scale training enhances the discriminator, resulting in the ability of the generator and the discriminator to be out of balance. Thus we only generate $128 \times 128$ images during training, and the real images are also resized to $128 \times 128$.

For $256 \times 256$ experiments, the weight decay of the generator and discriminator are set to 0.0001 and 0.00001, respectively. The generator is updated once every time the discriminator is updated twice. We have found experimentally that this will make training more stable. The generator and discriminator use non-local block at $64 \times 64$ resolution rather than $128 \times 128$ due to limited GPU memory. For Omni-INR-GAN, in order to support downstream tasks friendly, we do not use non-local block in the discriminator. Moreover, due to GPU memory limitation, we reduce the batch size to 128 and accumulate the gradient twice to approximate the gradient when the batch size is 256. We did not adopt a multi-scale training strategy. Only $256 \times 256$ images are generated during training, and the real images are also resized to $256 \times 256$.

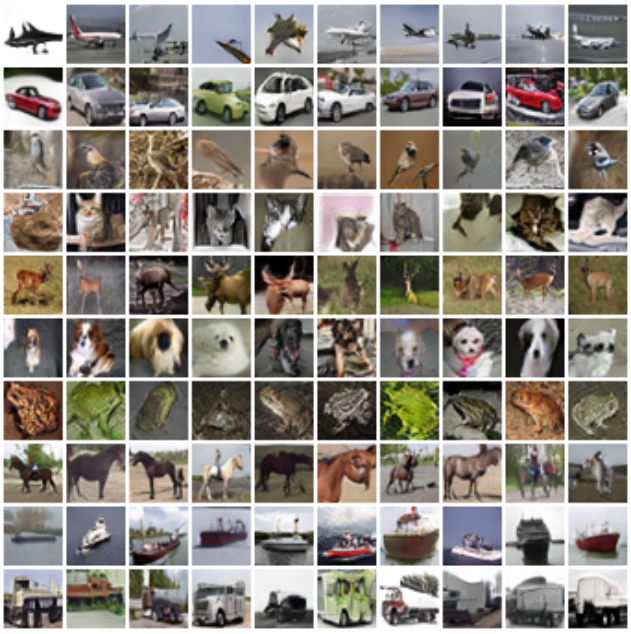


Figure 15: Randomly generated image by Omni-GAN for CIFAR10

## K. Additional Results

### K.1. Generated Images on CIFAR

In Fig. 15 and 16, we show generated images from Omni-GAN on CIFAR10, CIFAR100 respectively. Due to limited space, we only show images of some categories on CIFAR100.

### K.2. Generated Images on ImageNet

Omni-INR-GAN inherently supports generating images of arbitrary resolution. We adopt the Omni-INR-GAN $256 \times 256$ model to generate some images with different resolutions, *e.g.*, Fig. 17, 18, etc.

### K.3. Results of Semantic Image Synthesis

In Fig. 36, we show several results of Omni-GAN as well as those of SPADE for semantic image synthesis. The label maps and the ground truth images are from the first ten items in the test set of Cityscapes dataset, without cherry-picking.

| $\boldsymbol{z} \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$, embed$(y) \in \mathbb{R}^{128}$ |
|---|
| Linear $20 \rightarrow 4 \times 4 \times 16ch$ |
| ResBlock up $16ch \rightarrow 16ch$ |
| ResBlock up $16ch \rightarrow 8ch$ |
| ResBlock up $8ch \rightarrow 4ch$ |
| ResBlock up $4ch \rightarrow 2ch$ |
| Non-local Block ($64 \times 64$) |
| ResBlock up $2ch \rightarrow ch$ |
| BN, ReLU, $3 \times 3$ Conv $ch \rightarrow 3$ |
| Tanh |

(a) Generator

| RGB image $\boldsymbol{x} \in \mathbb{R}^{128 \times 128 \times 3}$ |
|---|
| ResBlock down $3 \rightarrow ch$ |
| Non-local Block ($64 \times 64$) |
| ResBlock down $ch \rightarrow 2ch$ |
| ResBlock down $2ch \rightarrow 4ch$ |
| ResBlock down $4ch \rightarrow 8ch$ |
| ResBlock down $8ch \rightarrow 16ch$ |
| ResBlock $16ch \rightarrow 16ch$ |
| ReLU, global sum pooling |
| Linear $\rightarrow 1002$ |

(b) Discriminator

Table 3: Omni-GAN architecture on ImageNet $128 \times 128$. $ch$ is set to be 96.

| $\boldsymbol{z} \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$, embed$(y) \in \mathbb{R}^{128}$ |
|---|
| Linear $17 \rightarrow 4 \times 4 \times 16ch$ |
| ResBlock up $16ch \rightarrow 16ch$ |
| ResBlock up $16ch \rightarrow 8ch$ |
| ResBlock up $8ch \rightarrow 8ch$ |
| ResBlock up $8ch \rightarrow 4ch$ |
| Non-local Block ($64 \times 64$) |
| ResBlock up $4ch \rightarrow 2ch$ |
| ResBlock up $2ch \rightarrow ch$ |
| BN, ReLU, $3 \times 3$ Conv $ch \rightarrow 3$ |
| Tanh |

(a) Generator

| RGB image $\boldsymbol{x} \in \mathbb{R}^{256 \times 256 \times 3}$ |
|---|
| ResBlock down $3 \rightarrow ch$ |
| ResBlock down $ch \rightarrow 2ch$ |
| Non-local Block ($64 \times 64$) |
| ResBlock down $2ch \rightarrow 4ch$ |
| ResBlock down $4ch \rightarrow 8ch$ |
| ResBlock down $8ch \rightarrow 8ch$ |
| ResBlock down $8ch \rightarrow 16ch$ |
| ResBlock $16ch \rightarrow 16ch$ |
| ReLU, global sum pooling |
| Linear $\rightarrow 1002$ |

(b) Discriminator

Table 4: Omni-GAN architecture on Imagenet $256 \times 256$. $ch$ is set to be 96.

| $\boldsymbol{z} \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$, embed$(y) \in \mathbb{R}^{128}$ |
|---|
| Linear $20 \rightarrow 4 \times 4 \times 16ch$ |
| ResBlock up $16ch \rightarrow 16ch$ |
| ResBlock up $16ch \rightarrow 8ch$ |
| ResBlock up $8ch \rightarrow 4ch$ |
| ResBlock up $4ch \rightarrow 2ch$ |
| Non-local Block ($64 \times 64$) |
| ResBlock up $2ch \rightarrow ch$ |
| Unfold(kernel_size=3) $ch \rightarrow 9ch$ |
| Grid_sample$(x, y)$, Concat feature and $(x, y)$ |
| Linear, Relu $9ch + 2 \rightarrow ch$ |
| Linear, Relu $ch \rightarrow ch$ |
| Linear $ch \rightarrow 3$ |
| Tanh |

(a) Generator

| RGB image $\boldsymbol{x} \in \mathbb{R}^{128 \times 128 \times 3}$ |
|---|
| ResBlock down $3 \rightarrow ch$ |
| ResBlock down $ch \rightarrow 2ch$ |
| ResBlock down $2ch \rightarrow 4ch$ |
| ResBlock down $4ch \rightarrow 8ch$ |
| ResBlock down $8ch \rightarrow 16ch$ |
| ResBlock $16ch \rightarrow 16ch$ |
| ReLU, global sum pooling |
| Linear $\rightarrow 1002$ |

(b) Discriminator

Table 5: Omni-INR-GAN architecture on ImageNet $128 \times 128$. $ch$ is set to be 96.

| $\boldsymbol{z} \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$, embed$(y) \in \mathbb{R}^{128}$ |
|---|
| Linear $17 \rightarrow 4 \times 4 \times 16ch$ |
| ResBlock up $16ch \rightarrow 16ch$ |
| ResBlock up $16ch \rightarrow 8ch$ |
| ResBlock up $8ch \rightarrow 8ch$ |
| ResBlock up $8ch \rightarrow 4ch$ |
| Non-local Block ($64 \times 64$) |
| ResBlock up $4ch \rightarrow 2ch$ |
| ResBlock up $2ch \rightarrow ch$ |
| Unfold(kernel_size=3) $ch \rightarrow 9ch$ |
| Grid_sample$(x, y)$, Concat feature and $(x, y)$ |
| Linear, Relu $9ch + 2 \rightarrow ch$ |
| Linear, Relu $ch \rightarrow ch$ |
| Linear $ch \rightarrow 3$ |
| Tanh |

(a) Generator

| RGB image $\boldsymbol{x} \in \mathbb{R}^{256 \times 256 \times 3}$ |
|---|
| ResBlock down $3 \rightarrow ch$ |
| ResBlock down $ch \rightarrow 2ch$ |
| ResBlock down $2ch \rightarrow 4ch$ |
| ResBlock down $4ch \rightarrow 8ch$ |
| ResBlock down $8ch \rightarrow 8ch$ |
| ResBlock down $8ch \rightarrow 16ch$ |
| ResBlock $16ch \rightarrow 16ch$ |
| ReLU, global sum pooling |
| Linear $\rightarrow 1002$ |

(b) Discriminator

Table 6: Omni-INR-GAN architecture on Imagenet $256 \times 256$. $ch$ is set to be 96.

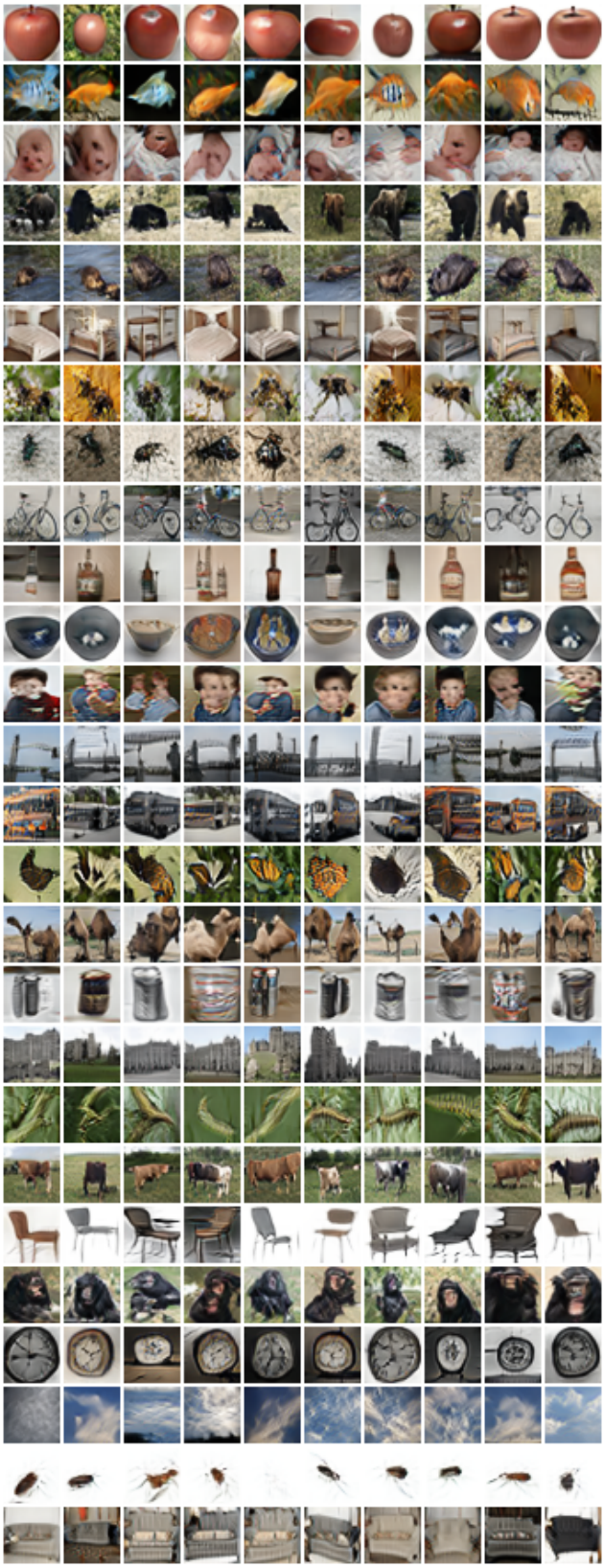


Figure 16: Randomly generated image by Omni-GAN for CIFAR100

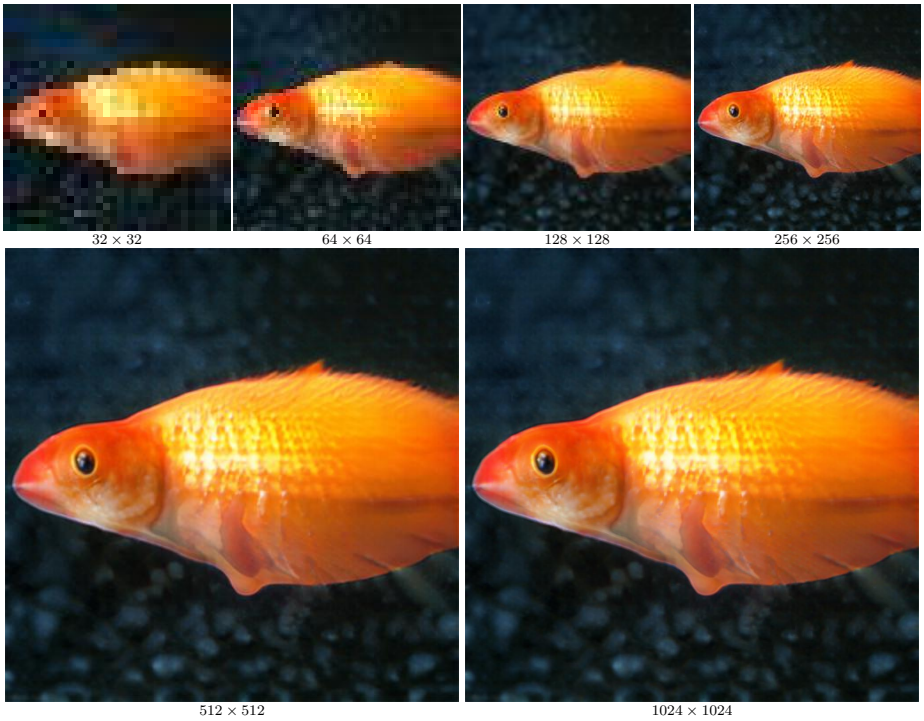



Figure 17: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 18: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

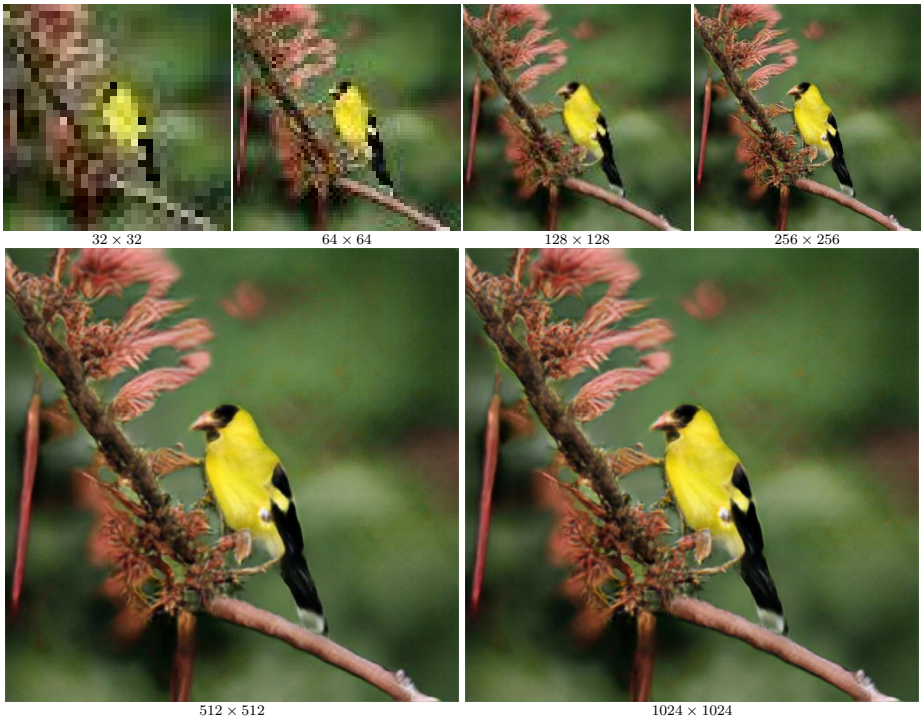



Figure 19: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

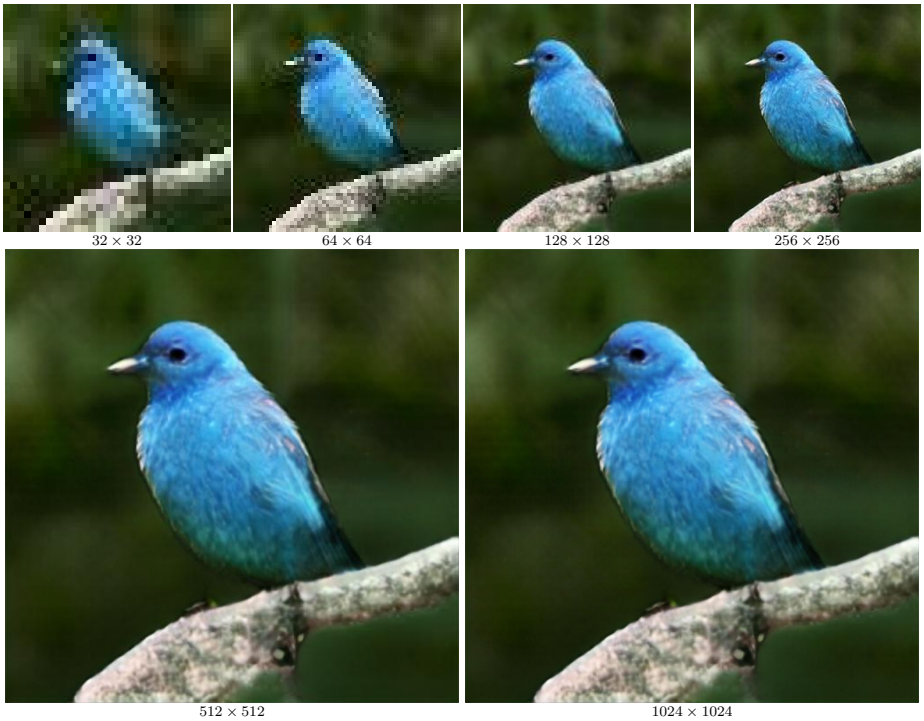



Figure 20: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 21: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

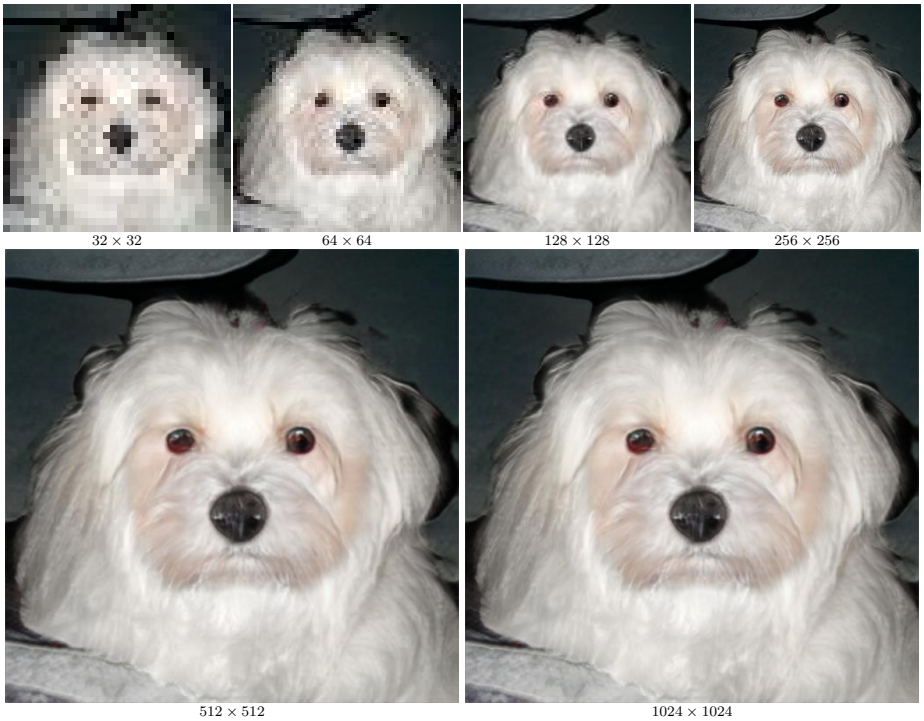



Figure 22: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

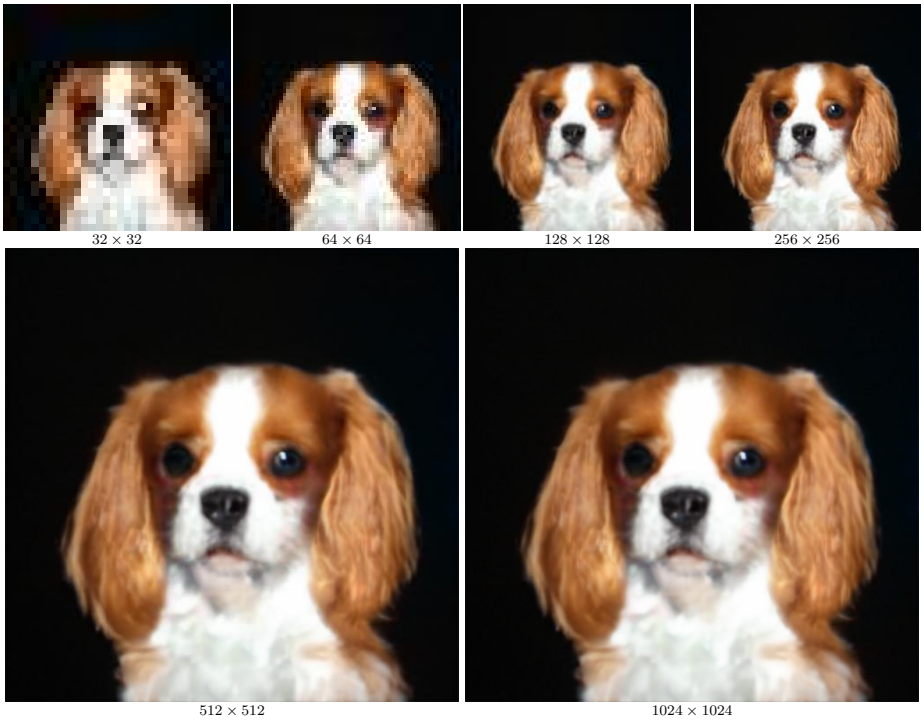



Figure 23: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 24: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 25: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

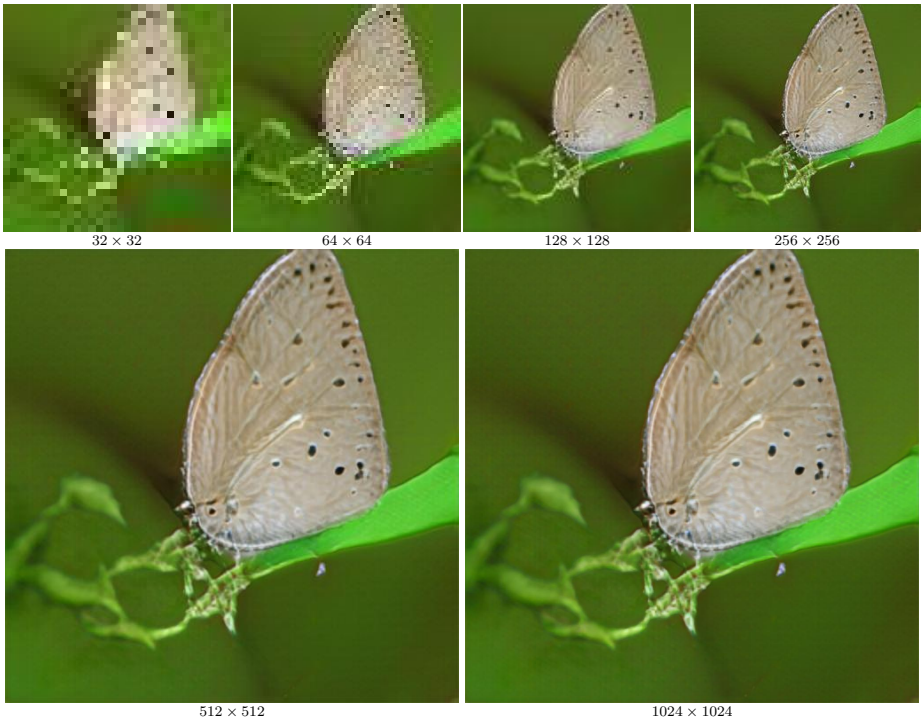



Figure 26: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

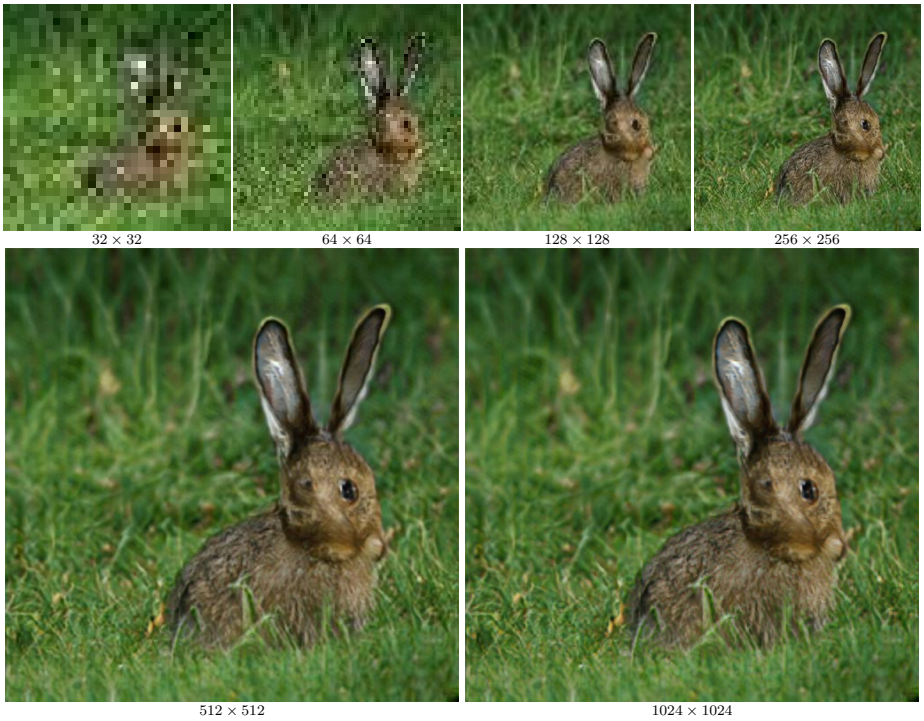



Figure 27: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 28: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 29: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 30: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

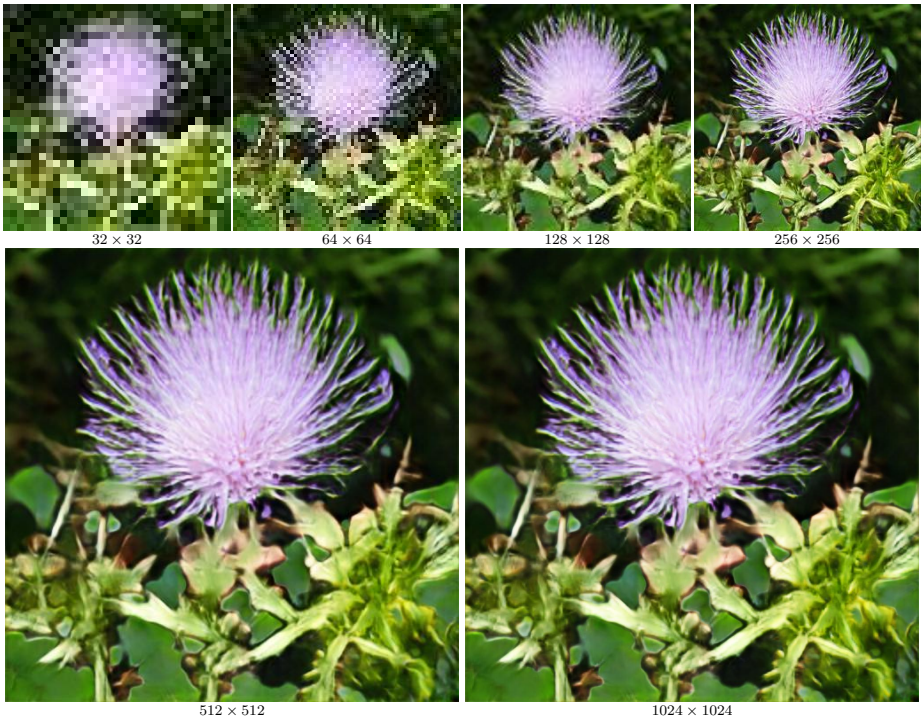



Figure 31: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

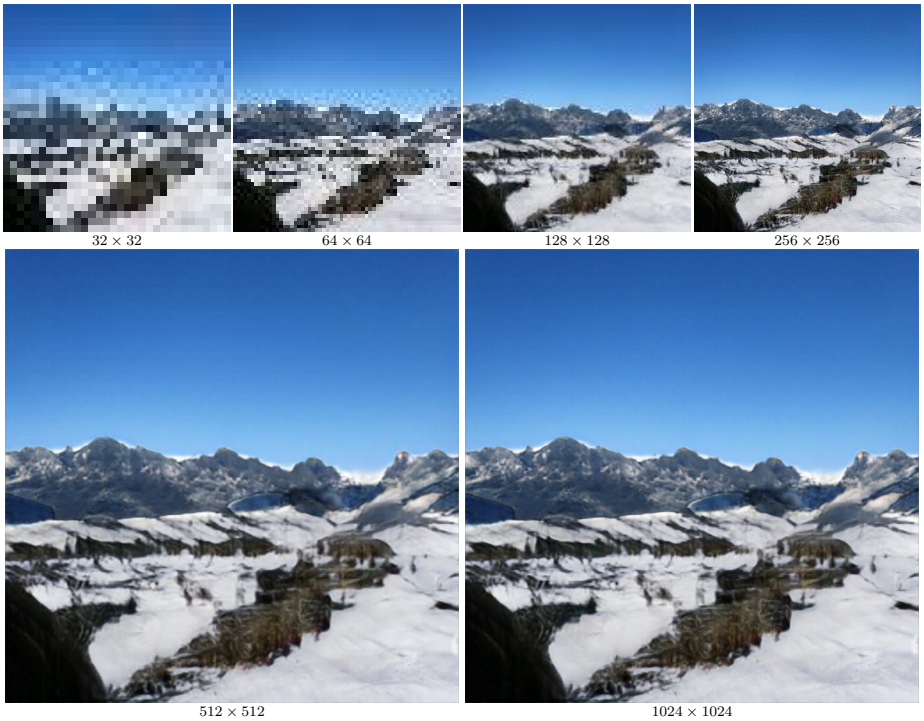



Figure 32: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

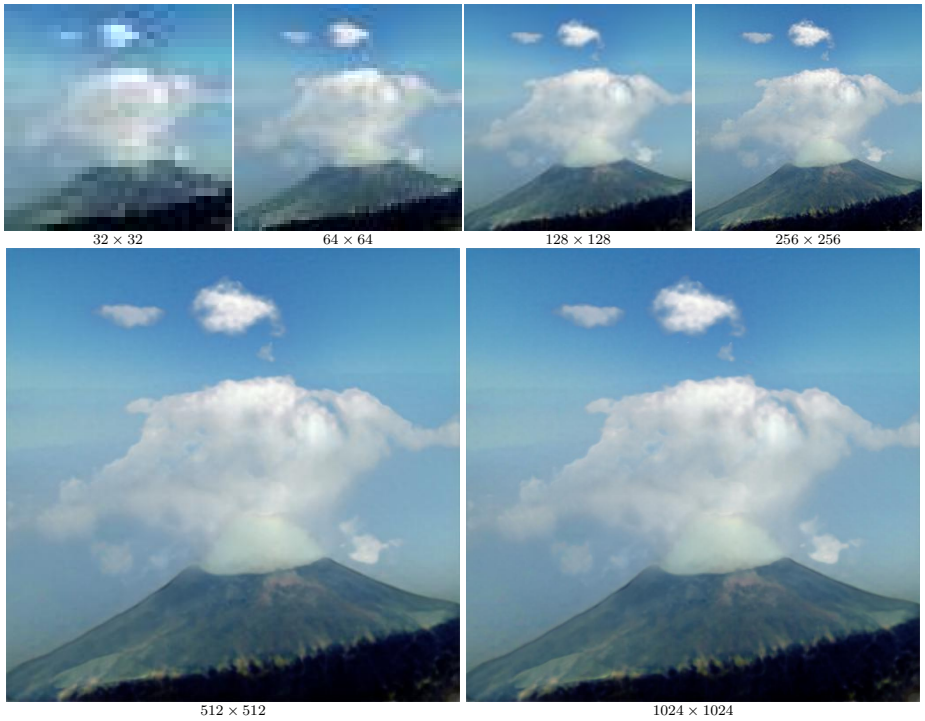



Figure 33: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.


Figure 34: Samples generated by our Omni-INR-GAN $256 \times 256$ model. Omni-INR-GAN has the ability to generate images of any resolution.

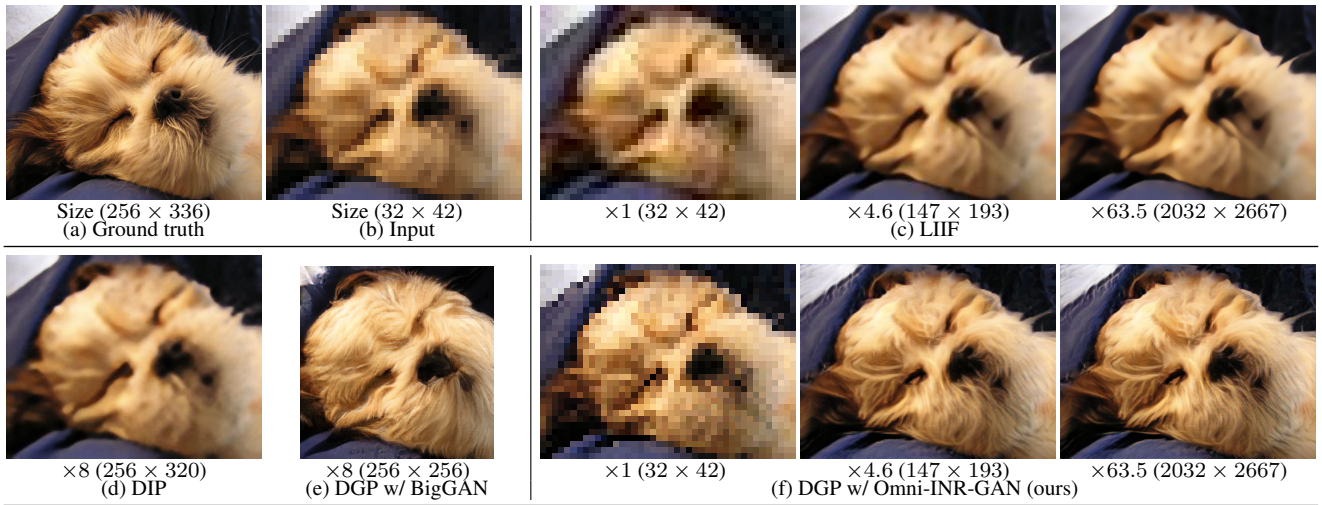



Figure 35: Super-resolution using Omni-INR-GAN's prior, at any scale ($\times$1-$\times$60+). (b) input image with low resolution. (c) LIIF [1] can extrapolate the input image to any scale, but it cannot add semantic details, so the result is still blurred. (d) DIP [20] also failed because the input image resolution is too low. (e) DGP [12] with BigGAN must crop the input and upsamples the cropped patch to a fixed size, which is inflexible. (f) Omni-INR-GAN has the ability to upsample the input image to any scale and also adds rich semantic details.

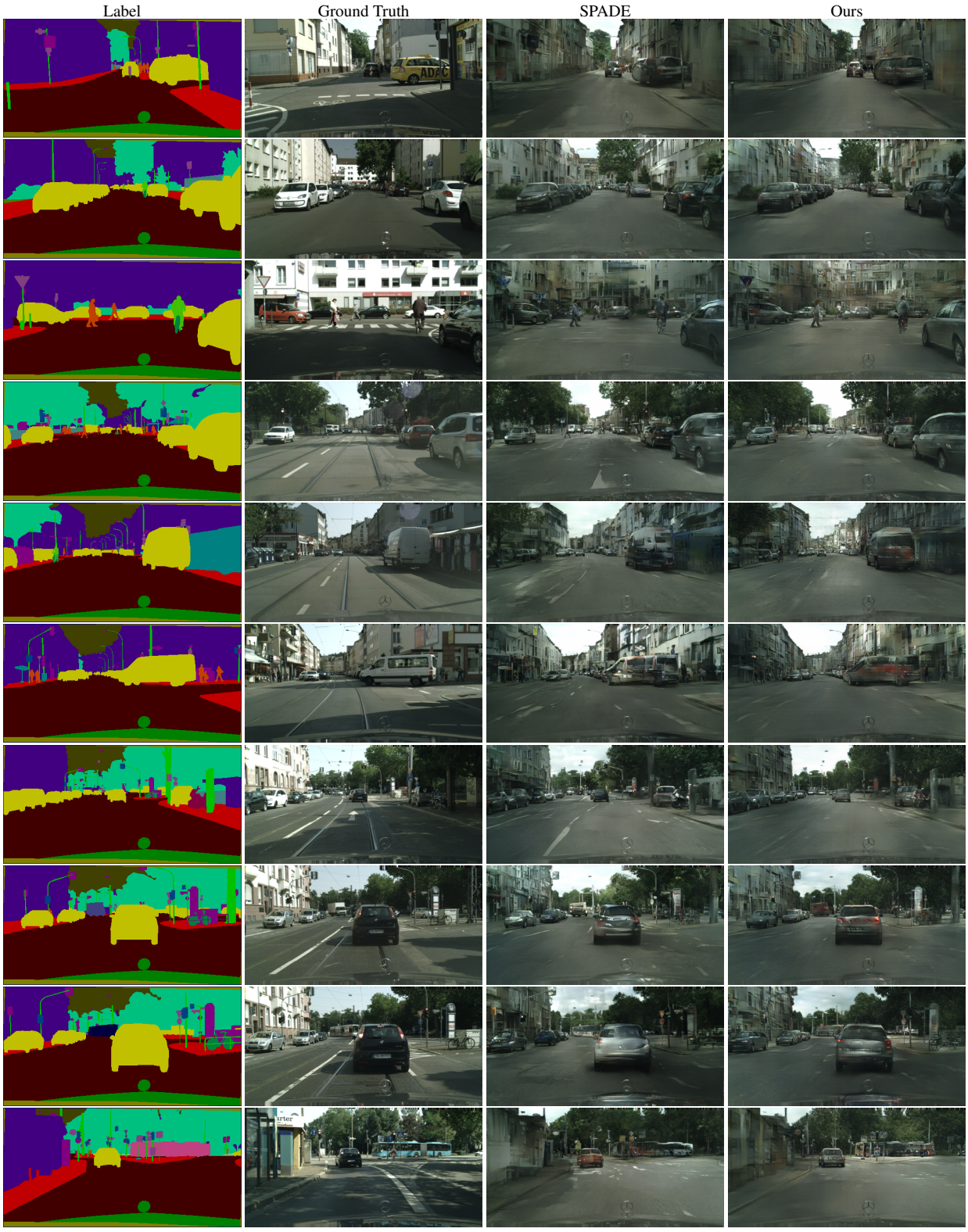



Figure 36: Results of semantic image synthesis on Cityscapes.

## References


[1] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning Continuous Image Representation with Local Implicit Image Function. *arXiv:2012.09161 [cs]*, 2020. 3, 22

[2] Zhiqin Chen and Hao Zhang. Learning Implicit Fields for Generative Shape Modeling. *arXiv:1812.02822 [cs]*, 2019. 3

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *CVPR*, 2016. 7

[4] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NeurIPS*, 2017. 3

[5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *CVPR*, 2017. 7

[6] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training Generative Adversarial Networks with Limited Data. *arXiv:2006.06676 [cs, stat]*, 2020. 4, 5

[7] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998. 4

[8] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually Converge? In *ICML*, 2018. 3

[9] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy Networks: Learning 3D Reconstruction in Function Space. *arXiv:1812.03828 [cs]*, 2019. 3

[10] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011. 4

[11] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional Image Synthesis With Auxiliary Classifier GANs. *arXiv:1610.09585 [cs, stat]*, 2017. 3

[12] Xingang Pan, Xiaohang Zhan, Bo Dai, Dahua Lin, Chen Change Loy, and Ping Luo. Exploiting Deep Generative Prior for Versatile Image Restoration and Manipulation. In *ECCV*, 2020. 3, 8, 22

[13] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. *arXiv:1901.05103 [cs]*, 2019. 3

[14] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic Image Synthesis with Spatially-Adaptive Normalization. In *CVPR*, 2019. 7, 8

[15] Xiaojuan Qi, Qifeng Chen, Jiaya Jia, and Vladlen Koltun. Semi-parametric Image Synthesis. In *CVPR*, 2018. 7

[16] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial Generation of Continuous Images. *arXiv:2011.12026 [cs]*, 2020. 3

[17] Jianlin Su. Extending Cross-Entropy of Softmax to Multi-Label Classification. https://kexue.fm/archives/7359, 2020. 1

[18] Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang, and Yichen Wei. Circle Loss: A Unified Perspective of Pair Similarity Optimization. In *CVPR*, 2020. 1, 2

[19] Song Tao and Jia Wang. Alleviation of Gradient Exploding in GANs: Fake Can Be Real. In *CVPR*, 2020. 7

[20] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep Image Prior. *arXiv:1711.10925 [cs, stat]*, 2018. 22

[21] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-Video Synthesis. In *NeurIPS*, 2018. 7

[22] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. In *CVPR*, 2018. 7

[23] Jiqing Wu, Zhiwu Huang, Janine Thoma, Dinesh Acharya, and Luc Van Gool. Wasserstein divergence for gans. In *ECCV*, 2018. 3

[24] Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy Lillicrap. LOGAN: Latent Optimisation for Generative Adversarial Networks. *arXiv:1912.00953 [cs, stat]*, 2019. 6

[25] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated Residual Networks. In *CVPR*, Honolulu, HI, 2017. 7

[26] Han Zhang, Zizhao Zhang, Augustus Odena, and Honglak Lee. Consistency Regularization for Generative Adversarial Networks. In *ICLR*, 2020. 6

[27] Zhengli Zhao, Sameer Singh, Honglak Lee, Zizhao Zhang, Augustus Odena, and Han Zhang. Improved Consistency Regularization for GANs. In *AAAI*, 2021. 6