Positional Encoding as Spatial Inductive Bias in GANs

Supplementary Material

A. Padding Encodes Spatial Bias for Convolutional Generators

In Sec. A.1 and Sec. A.2, we will provide the detailed deviation for Sec. 3.1 and Sec. 3.2, respectively. In addition, the effects of the padding mode will also be analyzed with an interesting example in Sec. A.2. We show that the spatial consistent nonlinear activation function cannot influence the weak stationarity in Sec. A.3. Section A.4 shows the bias term will not influence the weak stationarity in the convolutional feature map. Finally, we discuss zero padding from another view by presenting the effects of zero padding in different convolutional layers.

A.1. Detailed Proof for Weak Stationarity

Following the idea in Sec. 3.1, we will first consider the effects of translation invariance in the padding-free convolutional generators and give some preliminaries in the stochastic process. Then, with zero padding, the expectation ($\mathbb{E}$) and autocorrelation function (R) for the features shows how padding leaks spatial information.

In this work, we mainly focus on the standard convolution layer with the nonlinear activation layer and take the commonly used LeakyReLU (Eq. (1)) function as an example. The batch normalization and instance normalization are spatial identical operation and can be merged into convolutional operation [6, 5]. Therefore, we will neglect these layers in the following proof.

$$LeakyReLU(y) = \begin{cases} y, & y \geq 0 \\ \gamma y, & otherwise \end{cases}. \tag{1}$$

Taking the spatial noise map ($X_{\vec{\mathbf{i}}} \overset{i.i.d.}{\sim} \mathcal{N}(0,1)$) as input, the expectation of the first convolutional feature map ($\mathbb{E}(y^{(1)}_{\vec{\mathbf{i}}})$) is:

$$\begin{aligned} \mathbb{E}(y^{(1)}_{\vec{\mathbf{i}}}) &= \sum_k w^{(1)}_k \int_{-\infty}^{+\infty} x_k p(x_k) dx_k + b^{(1)} \\ &= \sum_k w^{(1)}_k \mathbb{E}(x_k) + b^{(1)} \qquad (2) \\ &= b^{(1)}. \qquad (3) \end{aligned}$$

Here, as we assume the input is sampled from a zero-expectation distribution, the final results in Eq. (3) shows the expectation is only related to the bias parameters in convolutional layers. However, this is not the general formulation for the expectation of the feature map in the convolutional generators.

After adopting a LeakyReLU function (g) and the next convolutional layer, we will obtain the general formulation of $\mathbb{E}(y_{\vec{\mathbf{i}}})$:

$$\begin{aligned} \mathbb{E}(y^{(2)}_{\vec{\mathbf{i}}}) &= \sum_k w_k \int_{-\infty}^{+\infty} g(y^{(1)}_k) p(y^{(1)}_k) dy^{(1)}_k + b^{(2)} \\ &= \sum_k w^{(2)}_k \int_{-\infty}^{0} \gamma y^{(1)}_k p(y^{(1)}_k) dy^{(1)}_k + \sum_k w^{(2)}_k \int_{0}^{+\infty} y^{(1)}_k p(y^{(1)}_k) dy^{(1)}_k + b^{(2)} \\ &= \sum_k w^{(2)}_k \cdot (\gamma \mathbb{C}_1 + \mathbb{C}_2) + b^{(2)}, \qquad (4) \end{aligned}$$

where $\mathbb{C}_1, \mathbb{C}_2$ are constants from the piecewise-defined finite integration. Therefore, the translation-invariant convolutional layer results in a spatial consistent expectation of the feature map. The expectation is a linear combination of the parameters in the convolution kernel, which is irrelevant to the positions.

$R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})$ portrays the relationship between two spatial locations. Here, taking $X_{\vec{\mathbf{i}}} \overset{i.i.d.}{\sim} \mathcal{N}(0,1)$, we directly analyze the feature map after the convolutional operation. The bias item is removed in this part since we do not care about the spatial

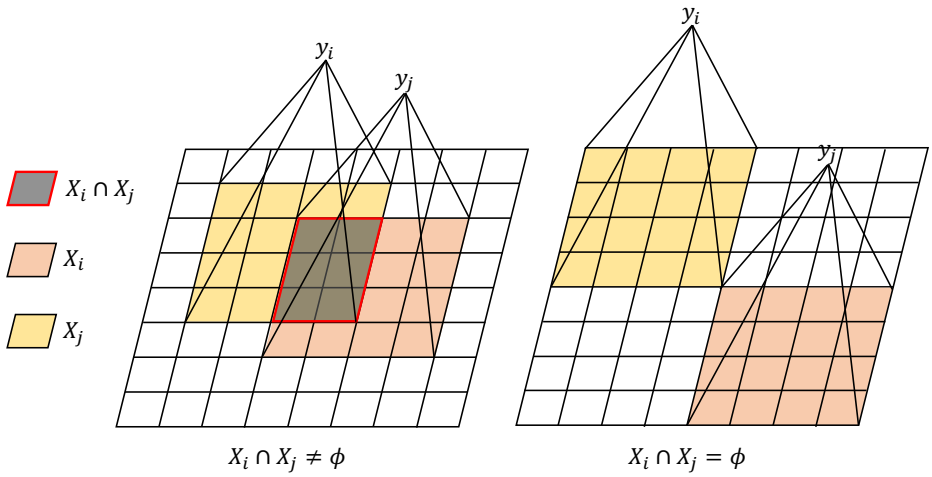


Figure 1: An illustration for the two cases of calculating the autocorrelation function in the convolutional feature map.

consistent addition component in autocorrelation analysis. However, as shown in Fig. 1, we should consider two cases of whether the input patch regions $(X_{\vec{\mathbf{i}}}, X_{\vec{\mathbf{j}}})$ are seperated. If $X_{\vec{\mathbf{i}}}, X_{\vec{\mathbf{j}}}$ are separated, it is trivial to calculate$R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})$:

$$\begin{aligned} R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) &= \mathbb{E}(y_{\vec{\mathbf{i}}} y_{\vec{\mathbf{j}}}) \\ &= \mathbb{E}[(\sum_k w_k x_k)(\sum_t w_t x_t)] \\ &= \sum_{k,t} w_k w_t \mathbb{E}(x_k)\mathbb{E}(x_t) \\ &= 0 \end{aligned} \tag{5}$$

Once there lies an intersection between the two input features, the autocorrelation function $R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{i}}})$ should be:

$$\begin{aligned} R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) &= \mathbb{E}(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) \\ &= \mathbb{E}[(\sum_k w_k x_k)(\sum_t w_t x_t)] \\ &= \mathbb{E}[\sum_{x_l \in X_i \cap X_j} w_{k_l} w_{t_l} x_l^2] + \mathbb{E}[\sum_{p \neq q} w_{k_p} w_{t_q} x_p x_q] \qquad (6) \\ &= \sum_{x_l \in X_i \cap X_j} w_{k_l} w_{t_l} \mathbb{E}(x_l^2) \qquad (7) \\ &= R(\vec{\mathbf{i}} - \vec{\mathbf{j}}), \qquad (8) \end{aligned}$$

where $p \neq q$ indicates that x_p and x_q are not the same feature variable. As shown in Eq. (6), the formulation can be split into two parts of the intersection part and the separated part. The spatial independence causes the separated part to be zero. The intersection part is related to the variance of the random variables and the parameters in the convolutional kernel. As the input X is spatial identical, $E(x_l^2)$ must be consistent among the spatial space. Thus, Eq. (7) is only related the intersection mode of $X_{\vec{\mathbf{i}}} \cap X_{\vec{\mathbf{i}}}$, which is determined by $\vec{\mathbf{i}} - \vec{\mathbf{j}}$. In summary, after fusing Eq. (5) and Eq. (7), the autocorrelation function of a feature after convolution operation should be:

$$\begin{aligned} R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) &= R(\vec{\mathbf{i}} - \vec{\mathbf{j}}) \\ &= \sum_{x_l \in X_{\vec{\mathbf{i}}} \cap X_{\vec{\mathbf{j}}}} w_{k_l} w_{t_l} \mathbb{E}(x_l^2). \end{aligned} \tag{9}$$

Here, a large offest vector $(\vec{\mathbf{i}} - \vec{\mathbf{j}})$ will result in no intersection between two input feature maps. Then, there will not exist any element (x_l) belonging to $X_{\vec{\mathbf{i}}} \cap X_{\vec{\mathbf{j}}}$ and the autocorrelation function will be zero as Eq. (5).

A.2. How Padding Serves as Spatial Bias

Taking the zero padding into consideration, the linear combination of the parameters in the convolutional kernels in Eq. (4) and Eq. (9) will be influenced:

$$\begin{aligned}\mathbb{E}(y_{\vec{\mathbf{i}}}^{(2)}) &= \sum_k w_k \int_{-\infty}^{+\infty} g(y_k^{(1)}) p(y_k^{(1)}) dy_k^{(1)} \cdot \mathbb{1}(y_k^{(1)} \notin Pad) + b^{(2)} \\ &= \sum_k w_k^{(2)} (\gamma \mathbb{C}_1 + \mathbb{C}_2) \mathbb{1}(y_k^{(1)} \notin Pad) + b^{(2)}, \end{aligned} \tag{10}$$

$$R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) = \sum_{x_l \in X_{\vec{\mathbf{i}}} \cap X_{\vec{\mathbf{j}}}} w_{k_l} w_{t_l} \mathbb{E}(x_l^2) \cdot \mathbb{1}(x_l \notin Pad) \tag{11}$$

$$\neq R(\vec{\mathbf{i}} - \vec{\mathbf{j}}), \tag{12}$$

where the indicator function $\mathbb{1}(x_i \notin Pad)$ determines whether current input belongs to zero padding regions.

Here, we further investigate the effects of padding modes on the spatial information leak. The effects of the padding mode can also be clarified from the view of the expectation and autocorrelation function. Reflection padding will be taken as an example in the following analysis. In sucn padding mode, the boundary variables are copied to the padding regions. When analyzing the expectation ($\mathbb{E}(y_{\vec{\mathbf{i}}})$, we find the linear combination will not be influenced as Eq. (10). The expectation with reflection padding mode is the same as the padding-free formulation in Eq. (4). However, the autocorrelation function ($R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{i}}})$) is influenced by the reflected padding:

$$\begin{aligned} R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) &= \mathbb{E}[(\sum_k w_k x_k)(\sum_t w_t x_t)] \\ &= \mathbb{E}[\sum_{x_l \in X_{\vec{\mathbf{i}}} \cap X_{\vec{\mathbf{j}}}} w_{k_l} w_{t_l} x_l^2] + \mathbb{E}[\sum_{p \neq q} w_{k_p} w_{t_q} x_p x_q] \qquad (13) \\ &= \sum_{x_l \in X_{\vec{\mathbf{i}}} \cap X_{\vec{\mathbf{j}}}} w_{k_l} w_{t_l} \mathbb{E}(x_l^2) + \sum_{p \neq q} w_{k_p} w_{t_q} \mathbb{E}(x_p^2) \mathbb{1}(x_p \in Pad, x_p = x_q) \qquad (14) \\ &\neq R(\vec{\mathbf{i}} - \vec{\mathbf{j}}), \qquad (15) \end{aligned}$$

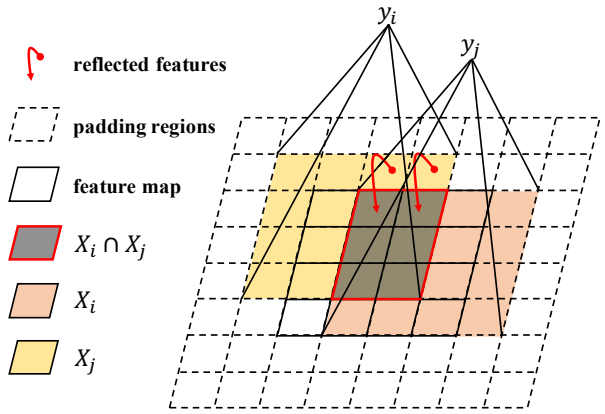


Figure 2: Illustration for refelcted padding.

where $\mathbb{1}(x_p \in Pad, x_p = x_q)$ indicates the whether x_p belongs to the padding regions and x_q is the corresponding boundary variable copied by x_p. As shown in Fig. 2, the red arrow presents the case in $\mathbb{1}(x_p \in Pad, x_p = x_q)$. Unlike the zero padding, the reflection padding can introduce extra relations in the non-intersection regions. As shown in Eq. (14), though the reflected padding does not change the expectation distribution, it adds extra items related to the padding in the autocorrelation function ($R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})$). Therefore, the reflection padding can inject implicit positional embedding by only changing the distribution of the autocorrelation function, leading to a location-aware autocorrelation function. Other padding modes like 'Circular Padding' in PyTorch can also be analyzed from the view of the expectation and autocorrelation function similarly with Eq. (14). In 'Circular Padding', the padded variable are borrowed from the other side of the feature map. Intuitively, the padded feature variables from remote areas can be assumed to be independent to the boundary feature variables. Thus, such padding mode cannot encode spatial information for the convolutional networks.

A.3. Effects of Nonlinear Activation Function on Autocorrelation Function

Intuitively, as a spatial identical operation, the nonlinear activation function changed the range of each feature, while does not influence the weak stationarity. Here, we adopt the LeakyReLU function in Eq. (9) to verify it will not change the weak

stationarity.

$$
\begin{aligned}
R(g(y_{\vec{\mathbf{i}}}), g(y_{\vec{\mathbf{j}}})) &= \mathbb{E}(g(y_{\vec{\mathbf{i}}})g(y_{\vec{\mathbf{j}}})) \\
&= \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} g(y_{\vec{\mathbf{i}}})g(y_{\vec{\mathbf{j}}})p(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})dy_{\vec{\mathbf{i}}}dy_{\vec{\mathbf{j}}} \\
&= \int_{-\infty}^{0}\int_{-\infty}^{0} \gamma^2 y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}p(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})dy_{\vec{\mathbf{i}}}dy_{\vec{\mathbf{j}}} + \int_{-\infty}^{0}\int_{0}^{+\infty} \gamma y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}p(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})dy_{\vec{\mathbf{i}}}dy_{\vec{\mathbf{j}}} \\
&\quad + \int_{0}^{+\infty}\int_{-\infty}^{0} \gamma y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}p(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})dy_{\vec{\mathbf{i}}}dy_{\vec{\mathbf{j}}} + \int_{0}^{+\infty}\int_{0}^{+\infty} y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}p(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})dy_{\vec{\mathbf{i}}}dy_{\vec{\mathbf{j}}} \qquad (16) \\
&= F_1(\vec{\mathbf{i}} - \vec{\mathbf{j}}) + F_2(\vec{\mathbf{i}} - \vec{\mathbf{j}}) + F_3(\vec{\mathbf{i}} - \vec{\mathbf{j}}) + F_4(\vec{\mathbf{i}} - \vec{\mathbf{j}}) \qquad (17) \\
&= R(\vec{\mathbf{i}} - \vec{\mathbf{j}}), \qquad (18)
\end{aligned}
$$

where $F_i(\vec{\mathbf{i}} - \vec{\mathbf{j}})$ denotes the function related to the offet vector $\vec{\mathbf{i}} - \vec{\mathbf{j}}$. In Eq. (16), the manipulation item $(y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}p(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})dy_{\vec{\mathbf{i}}}dy_{\vec{\mathbf{j}}})$ can be expanded similarly with Eq. (6) and then the final value of such item will be determined by the offet vector. Therefore, the spatial identical nonlinear activation function like LeakyReLU cannot change the weak stationarity.

A.4. Bias Term in Autocorrelation Function

In Sec. A.1 and Sec. A.2, we do not condider the bias term in the analysis of the autocorrelation function. Intuitively, the spatially identical addition operation cannot introduce an extra relationship between feature variables. Here, we further analyze the effects of the bias term on $R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}})$ in Eq. (9):

$$
\begin{aligned}
R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) &= \mathbb{E}((y_{\vec{\mathbf{i}}} + b)(y_{\vec{\mathbf{j}}} + b)) \\
&= \mathbb{E}(y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}} + b \cdot (y_{\vec{\mathbf{i}}} + y_{\vec{\mathbf{j}}}) + b^2) \\
&= \mathbb{E}(y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}) + b \cdot (\mathbb{E}(y_{\vec{\mathbf{i}}}) + \mathbb{E}(y_{\vec{\mathbf{j}}})) + b^2 \qquad (19) \\
&= \mathbb{E}(y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}) + b^2 \qquad (20)
\end{aligned}
$$

The extra b^2 term is not related to the absolute positional information. Thus, the spatially identical addition operation cannot influence the weak stationarity in the convolutional feature map. Besides, as shown in Eq. (19), the weak stationarity indeed implies the property of spatially consistent expectation in Eq. (4). Equation (9) reveals that if there is not any intersection between input features, the output features are independent. After adopting the bias term, such independence is also kept:

$$
\begin{aligned}
R(y_{\vec{\mathbf{i}}}, y_{\vec{\mathbf{j}}}) &= \mathbb{E}((y_{\vec{\mathbf{i}}} + b)(y_{\vec{\mathbf{j}}} + b)) \\
&= \mathbb{E}(y_{\vec{\mathbf{i}}}y_{\vec{\mathbf{j}}}) + b^2 \\
&= b^2 = \mathbb{E}(y_{\vec{\mathbf{i}}} + b)\mathbb{E}(y_{\vec{\mathbf{j}}} + b). \qquad (21)
\end{aligned}
$$

A.5. Locally Asymmetric and Symmetric Zero Padding

In this section, we will clarify that it is the asymmetric zero padding that introduces the implicit spatial bias. Firstly, the asymmetric zero padding denotes the cases shown in Fig. 3(a), where the convolutional kernel covers the locally asymmetric zero padding at corners. Indeed, zero padding adopted in the convolutional layers is always locally asymmetric, from the view of the effective receptive field. However, in the transposed convolution layer, Fig. 3(b) presents a case in which the effective zero padding on the input feature is locally symmetric. As for the transposed convolutional layer for $2\times$ upsampling in generators, the commonly used setting also brings locally asymmetric zero padding, as illustrated in Fig. 3(c). StyleGAN2 adopts the transposed convolutional layer with the same padding mode as Fig. 3(c). As analyzed in Sec. 3.2, the locally symmetric zero padding cannot introduce any spatial bias, because the convolutional kernel will meet the same padding patterns anywhere during its sliding over the feature map. On the contrary, Fig. 3(a) and Fig. 3(c) present the asymmetric zero padding pattern at corners which encodes an implicit spatial anchor for the convolutional generators.

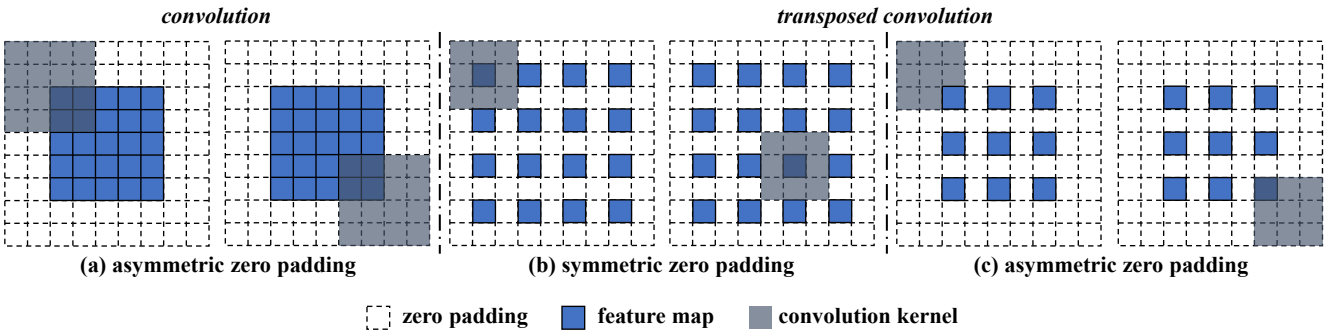


Figure 3: Illustration for asymmetric and symmetric zero padding in the convolution and transposed convolution.

B. StyleGAN2

B.1. Implementation Details

We directly follow the original setting detailed in [5] to train the StyleGAN2 model mentioned in our study. Generally, the channel multiplier is set to two (C2), which indicates the 'Large Network' proposed in the original StyleGAN2. As for the evaluation metric, we compute each metric three times with different random seeds and report their average. The FID is calculated with 50,000 real images while the P&R is computed with 10,000 real images.

B.2. Padding Effects on StyleGAN2

Section 3.3 describes the experiments on StyleGAN2 for investigating the padding effects. The motivation of the experiments is to adopt spatially identical input in StyleGAN2 to show the behavior of the convolutional generator. In StyleGAN2, there lie three input signals, *i.e.*, style latent code, spatial noise map, and the learnable constant input. Intuitively, the style latent code is originally identical in spatial space while the spatial noise map is sampled from spatially *i.i.d.* noise distribution. Thus, we only need to modify the learnable constant input at the start of the generator to guarantee that the input signals are spatially identical. In this experiment, we directly fill in the learned constant input with an identical value. Figure 4 presents more results sampled from the standard StyleGAN2 and padding-free StyleGAN2 with different identical input values.

As discussed in Sec. 3.1, if the generator is translation-invariant, the spatially identical input signal should have resulted in spatially identical colors or patterns in the output images. However, the standard StyleGAN2 with a fully-convolutional generator in Fig. 4 presents a biased spatial structure in which the borders are fixed to a frozen pattern. Once the padding is removed from the generator, the spatially identical colors or patterns will cover the whole canvas, as shown in Fig. 4. This experiment does not rely on any assumption on the convolutional weights like [1]. Thus, we believe that it is a more reasonable and convenient way to show the padding effects on the convolutional generator.

C. PGGAN and DCGAN

C.1. Implementation Details

We follow the original training configuration in the training of PGGAN. In addition to the experiments on the cropped CelebA dataset, we also present the results on the LSUN Bedroom dataset in Tab. 1. As for DCGAN, the baseline architecture used in our experiments only contains upsampling layers and convolutional layers, because the zero padding in the transposed convolution layer cannot be removed easily. Meanwhile, we adopt WGAN-GP [3] in DCGAN to improve the generation quality of the baseline model. Namely, a much stronger baseline model with DCGAN architecture is used in this study.

Combine with SPE. In the experiments on PGGAN and DCGAN, we combine the padding-free generator with SPE to further demonstrate that removing padding directly causes the lack of spatial information. Since SPE can be constructed with any channel dimension, we combine such flexible explicit positional encoding with the padding-free PGGAN and DCGAN. SPE is directly added with the first input feature map ahead of the convolutional generator. To be specified, the first 4×4 feature map is combined with SPE. Then, such a feature map containing explicit positional information will be adopted as the input of the following convolutional generator.

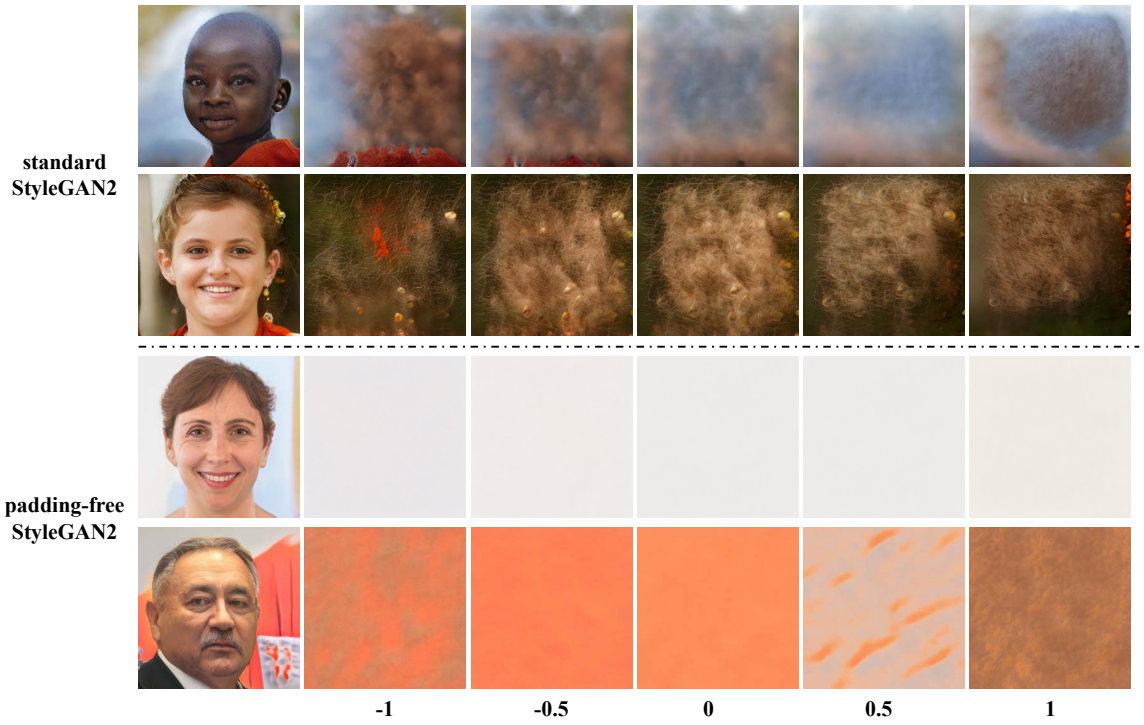


Figure 4: More results for standard StyleGAN2 (above the dotted line) and padding-free StyleGAN2 (under the dotted line). The first column is sampled with the original leaned constant input. The other five columns are sampled with different identical values (from left to right: -1, -0.5, 0, 0.5, 1) filling in the learned constant input at the start of the convolutional generator.

Table 1: Multi-level Sliced Wasserstein Distance (SWD) [4] between the synthesized and training images for different training configurations at 128×128. Each column in SWD represents one level of Laplacian pyramid [2], and the last one offers an average of the three distances. $\downarrow$ indicates lower is better.

Training configuration	CELEBA				LSUN BEDROOM			
	Sliced Wasserstein distance $\times 10^3 \downarrow$				Sliced Wasserstein distance $\times 10^3 \downarrow$			
	128	64	32	Avg.	128	64	32	Avg.
(a) PGGAN	**3.162**	**4.285**	**5.000**	**4.149**	7.473	5.197	5.214	5.962
(b) + Remove padding	11.169	6.945	7.488	8.534	13.619	10.043	6.581	10.081
(c) + SPE at head, w/o padding	4.555	6.164	6.365	5.694	**6.588**	**4.437**	**5.090**	**5.371**

C.2. More Results and Analyses

Table 1 furthter shows the results in LSUN Bedroom dataset with PGGAN. More qualitative results are presented in Fig. 5, Fig. 6, and Fig. 7. In Tab. 1(c), the closer distance in the LSUN Bedroom dataset further demonstrates the impact of the spatial inductive bias to the convolutional generator.

D. MS-PIE

D.1. Implementation Details and Other Results

In this study, we demonstrate the effectiveness of our MS-PIE in the state-of-the-art 256^2 StyleGAN2 model that is originally designed for 256×256 image generation. Three scales will be adopted the multi-scale training strategy while the sampling probability is fixed to $[0.5, 0.25, 0.25]$. When we replace the learnable constant input with the Cartesian grid, the input feature will only contain two channels. Other training details strictly follow the original configuration in the StyleGAN2, as mentioned in Sec. B. Finally, we show the quantitative results at 384^2 scale in Tab.2.

(a) PGGAN (b) + Remove Padding (c) + SPE at head, w/o padding

Figure 5: Sampled images from various PGGANs trained on cropped CelebA. (a), (b), and (c) indicate the different training configurations in Tab. 1.

(a) PGGAN (b) + Remove Padding (c) + SPE at head, w/o padding

Figure 6: Sampled images from various PGGANs trained on LSUN Bedroom. (a), (b), and (c) indicate the different training configurations in Tab. 1. (**Best viewed with zoom in.**)

(a) DCGAN (b) + Remove Padding (c) + SPE at head, w/o padding

Figure 7: Sampled images from various DCGANs trained on LSUN Bedroom. (a), (b), and (c) indicate the different training configurations in Tab. 4 (in the main paper). (**Best viewed with zoom in.**)

D.2. MS-PIE in Higher Resolutions

With MS-PIE, the original 256^2 StyleGAN2 containing six upsampling blocks can also achieve high-quality image generation in higher resolutions. However, we have to admit that our MS-PIE will cause additional GPU memory cost. This is because current deep learning frameworks, like PyTorch, always store tensors in continuous memory blocks, which will further accelerate the computational speed. When we switch to a different generation scale from the previous training iteration, we observe that the GPU memory that saves data for the last iteration will not be released or used for the current iteration. Thus, during training, the total GPU memory cost is not the cost for the largest scale. The peak of the GPU memory cost may be the sum of the cost for the three different scales. We have tested MS-PIE on Nvidia Tesla V100 with 32G GPU memory and found the highest resolution in which we can train our model under the limitation of GPU memory.

Table 2: Results in 384^2 scale for our MS-PIE with 256^2 StyleGAN2 in FFHQ dataset. The precision and recall are calculated at the same scale as the Fréchet inception distance (FID).'C2' indicates the channel multiplier in the generator is two.

Training configuration		Resize	FID@384↓ 20M	FID@384↓ 25M	Precision (%)↑	Recall (%)↑
MS-PIE w/ padding	(c) Leanable constant input	Interp	4.24	4.01	**74.23**	54.27
	(d) Cartesian spatial grid	Interp	4.30	4.13	74.11	54.23
	(e) SPE-interp	Interp	4.27	4.08	74.07	55.09
	(f) SPE-expand	Expand	**4.20**	**3.89**	73.52	**56.02**
	(g) SPE-expand-C1	Expand	4.23	4.02	74.01	55.39
MS-PIE w/o padding	(h) Learnable constant input	Interp	5.46	5.15	72.53	53.15
	(i) Cartesian spatial grid	Interp	**5.24**	**5.01**	72.13	**53.91**
	(j) SPE-interp	Interp	5.53	5.21	**72.56**	52.17
	(k) SPE-expand	Expand	5.51	5.23	71.85	52.27

Figure 8: Sampled multi-scale images from 256^2 StyleGAN2 with MS-PIE. We adopt three scales of 896^2, 512^2 and 256^2 resolutions. The larger resolution is shown with larger size.

For 256^2 StyleGAN2 with a channel multiplier of two, 896×896 is the highest resolution in which we can achieve a competitive FID of 4.10. We adopt three different scales of 256^2, 512^2, and 896^2 and the synthesized images are shown in Fig 8. Reducing the channel multiplier to one, we can train the lite StyleGAN2 model with 256^2, 512^2, and 1024^2 scales in MS-PIE. Even if only containing six lite convolutional blocks, as shown in Fig. 9, the lite generator can achieve compelling generation quality with an FID of 6.24.

D.3. Image Manipulation

To further verify the effectiveness of our MS-PIE in image editing, we customize a manipulation algorithm for 512×512 image manipulation with a single 256^2 StyleGAN2. Based on the best training configuration in Tab. 5(f) in Sec. 4.3, we will demonstrate its effectiveness. Achieving image manipulation in higher resolution with the 256^2 backbone is not trivial. We have tried the closed-form factorization method [7] in our generators but found that it cannot manage to control the image style.

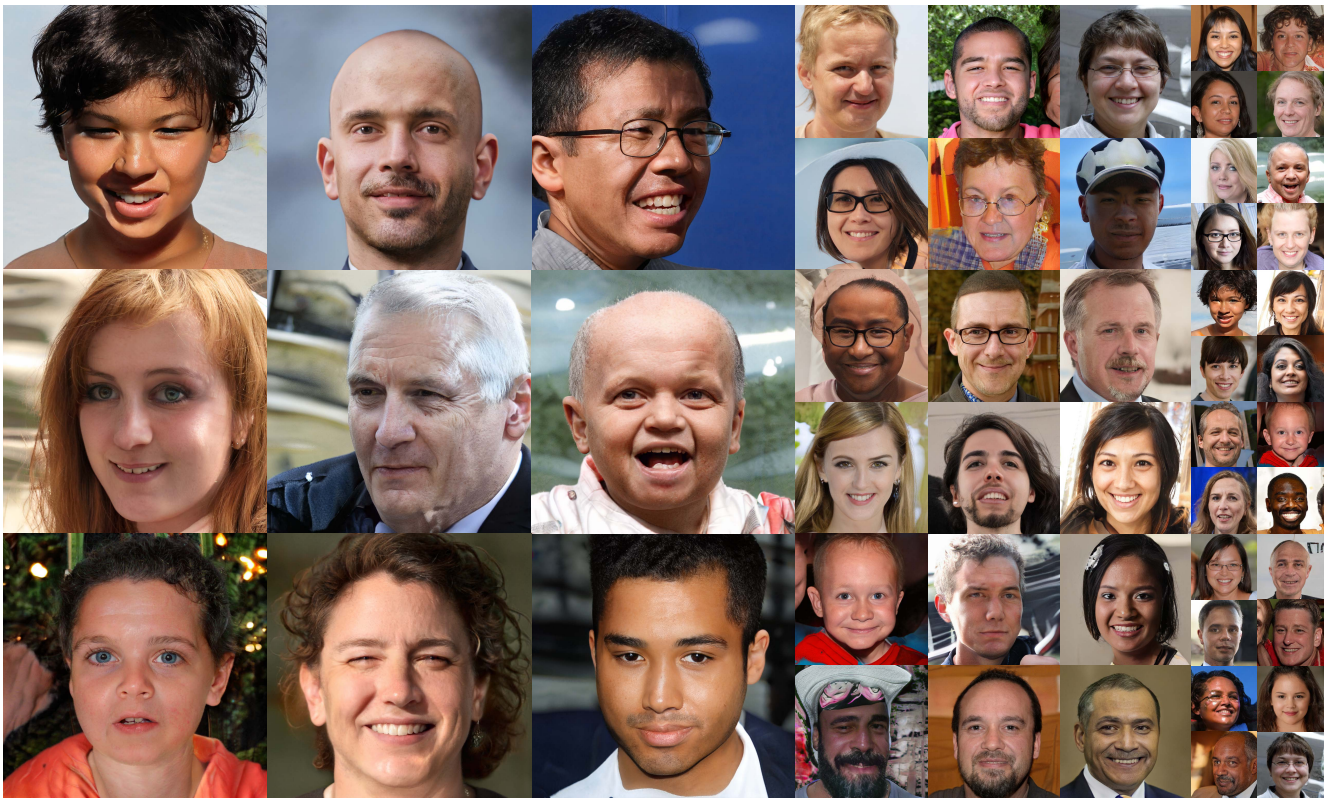


Figure 9: Sampled images from MSStyleGAN with 1024^2, 512^2 and 256^2 resolutions. The larger resolution is shown at larger size.

After analyzing the results and improving the closed-form factorization method, a more convenient and effective manipulation algorithm is proposed for the generators trained in MS-PIE.

Firstly, unlike the original method, we directly concatenate all of the weights in the style encoding layer from the upsampling convolutional blocks and compute the eigenvector for the large weight matrix. In our experiments, we found that such eigenvectors from the large weight matrix can control the attributes of the output. Furthermore, different eigenvectors tend to control different attributes separately. As shown in Fig. 10, the third eigenvector only controls the gender attribute while the 10-th eigenvector only controls the pose attribute.

The eigenvectors are computed from the large weight matrix containing weights from all of the convolutional blocks. Such eigenvectors can be applied to each convolutional block or only several blocks. For example, we can ***globally*** adopt the third eigenvector in each convolutional block. However, as shown in Fig. 10, this eigenvector will not have any influence on other attributes, *e.g.*, pose, and mouth shape. The 15-th eigenvector controls the lighting of the output and the 8-th and the 9-th convolutional block are much sensitive to this eigenvector. As shown in Fig. 10, the lighting attribute can be well controlled by ***locally*** applying the 15-th eigenvector in the 8-th and 9-th convolutional block. Once ***globally*** applying the 15-th eigenvector in each convolutional block, we observe that other attributes, like the color of the glass, will be influenced by the lighting.

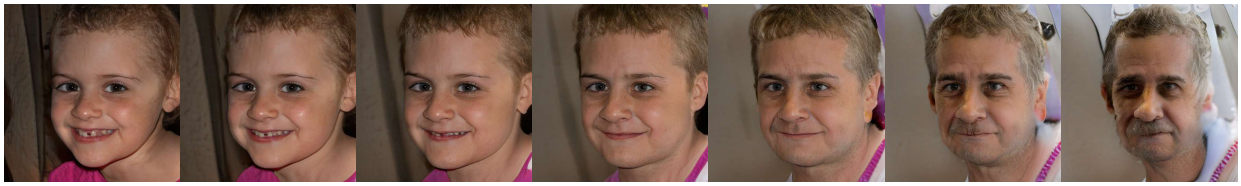
E. SinGAN

E.1. Implementation Details

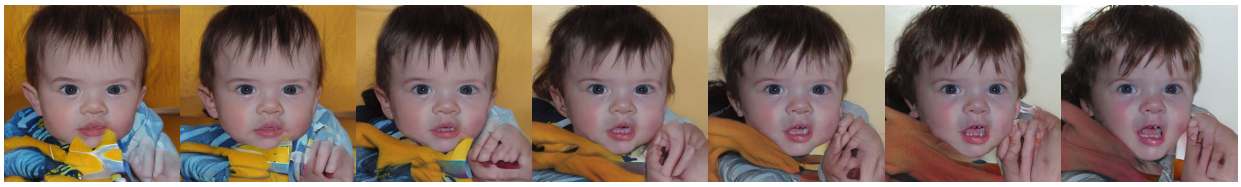
In SinGAN, we strictly follow the original training configuration to study the padding effects and the impacts of spatial inductive bias. For all of the training samples, we adopt the minimum size of 25 at the start stage while the final size of the final stage is set according to the original scale of the training image. As for padding removal, we carefully discard the padding from the input noise feature and the input images at each stage. In the experiments on the Cartesian grid, we adopt the Cartesian grid with two channels as the input positional encoding. Without other specifications, the number of total channels in sinusoidal positional encoding is set to 8. We use 16 channels in the sample of '*Bohemian Rhapsody*'.

(a) Gender

(b) Pose



(c) Age



(d) Mouth Shape (close --- open)

(e) Cosplay style

(f) Lighting (Locally)

(f) Lighting (Globally)

Figure 10: Image manipulation in 512×512 resolution with a 256^2 StyleGAN2 trained in MS-PIE.

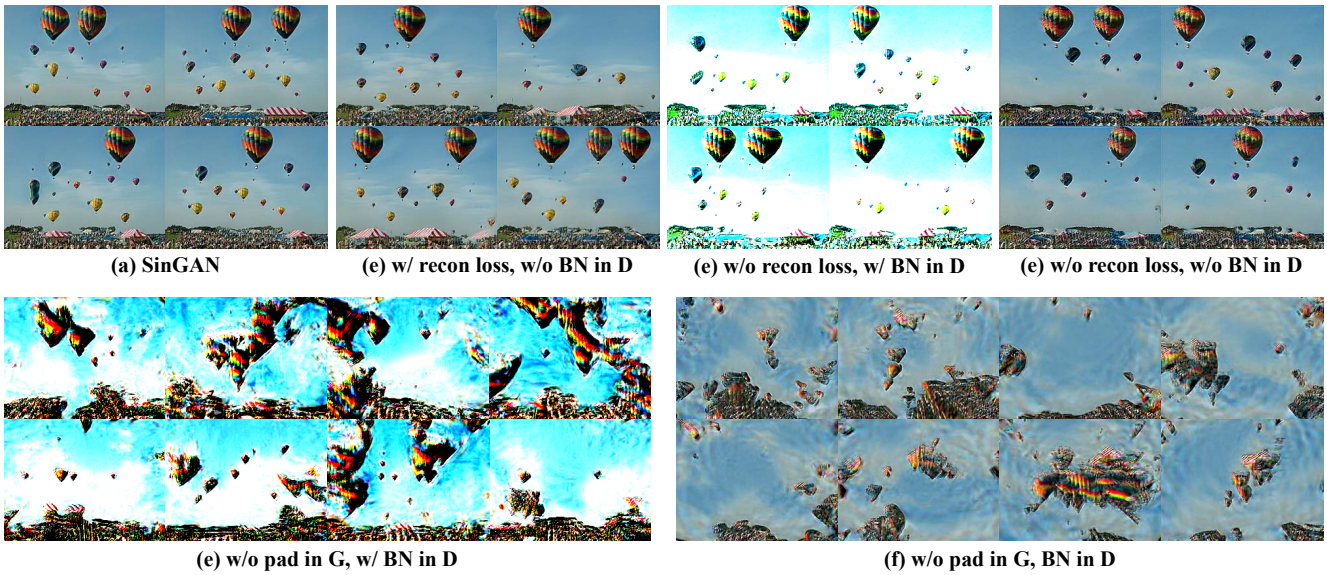


Figure 11: The effects of adopting batch normalization in discriminator and the reconstruction loss on SinGAN.

In the experiments with padding-free SinGAN, we modify the architecture of the discriminator by removing the batch normalization layer. This is because the channel-wise normalization layer destroys the learned relationship between the different channels, which brings a significant color shift in the output image. In Fig. 11, we presents an ablation study about the color shift phenomenon. Firstly, as shown in Fig. 11(b), removing the batch normalization layer in the discriminator will not influence the generated spatial structure, despite marginal negative effects on the texture quality. Once discarding the reconstruction loss in SinGAN, the generator performs a significant color shift with unreasonable brightness in the results in Fig. 11(c). Meanwhile, we observe that the spatial structure can be retained without any reconstruction supervision. Furthermore, based on the model without reconstruction loss, we remove the batch normalization layer in the discriminator. Surprisingly, the results in Fig. 11(d) recover the original brightness, indicating that the reconstruction loss indeed plays a role in correcting the color shift brought by adopting batch normalization in the discriminator. Finally, in Fig. 11(e) and Fig. 11(f), we show the results from padding-free SinGAN trained with different discriminators, *i.e.*, with batch normalization and without batch normalization. With a padding-free generator, adopting batch normalization in the discriminator only brings the color shift in results while the spatial structure cannot be captured. Thus, removing batch normalization in the discriminator does not influence the generated spatial structure that we care about in this work.

E.2. More Qualitative Results from SinGAN

Figure 12 presents more comparison for the effects of adopting different spatial inductive bias on SinGAN. The first case with different nuts verifies that the Cartesian grid can naturally keep the global structure retained, *e.g.*, the regular arrangements as the same as the original image. However, such spatial inductive bias is not suitable to perform multi-scale synthesis in the last case with a car. It is better to choose the sinusoidal positional encoding for a faithful detailed structure and realistic patch recurrence. On the contrary, without a clear or balanced spatial bias over the whole image space, the standard SinGAN cannot manage to perform high-quality internal sampling.

References

[1] Bilal Alsallakh, Narine Kokhlikyan, Vivek Miglani, Jun Yuan, and Orion Reblitz-Richardson. Mind the pad–cnns can develop blind spots. *arXiv preprint arXiv:2010.02178*, 2020. 5

[2] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. 6

[3] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, 2017. 5

[4] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018. 6

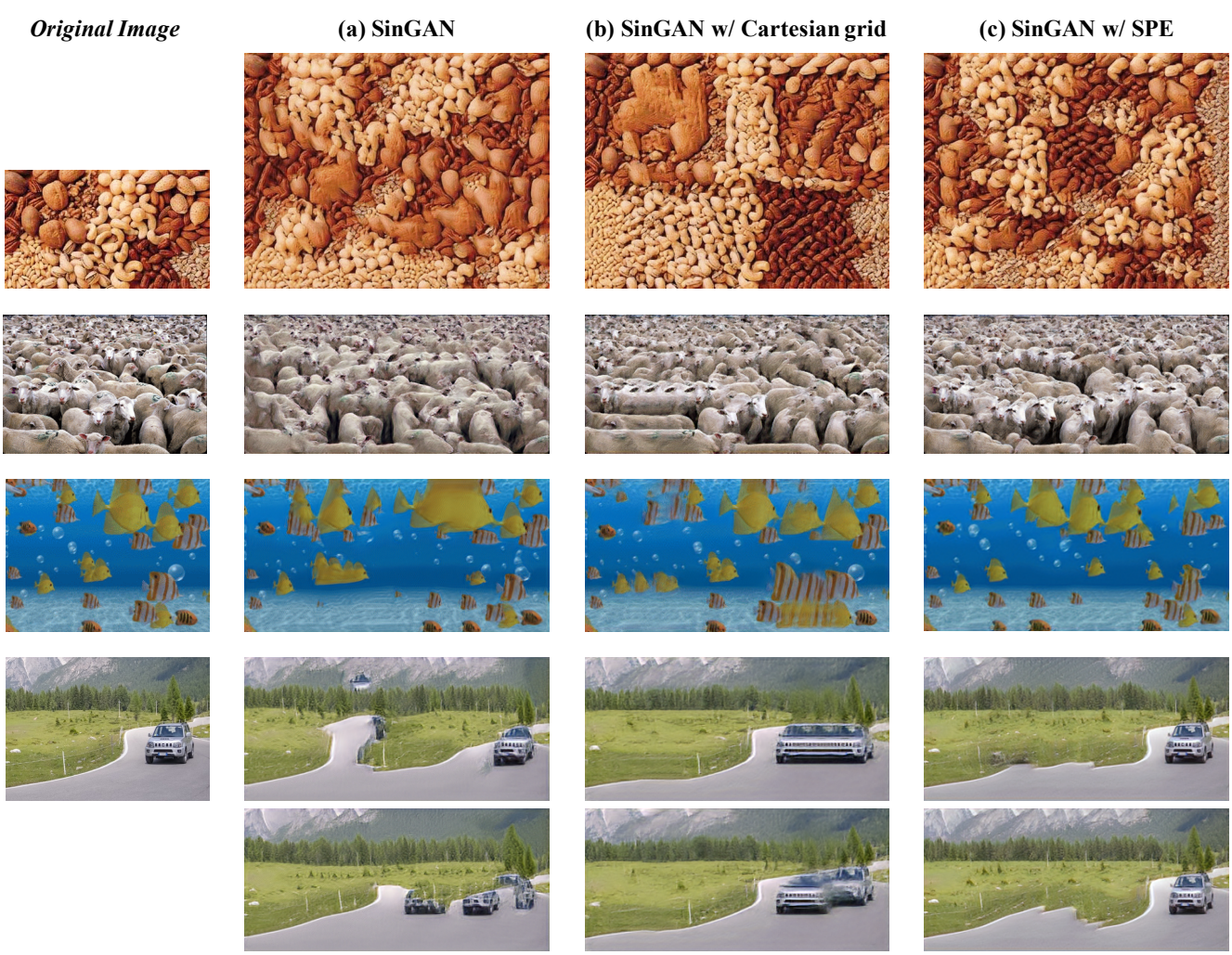


Figure 12: More results for SinGAN with different positional encodings.

[5] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 1, 5

[6] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019. 1

[7] Yujun Shen and Bolei Zhou. Closed-form factorization of latent semantics in gans. *arXiv preprint arXiv:2007.06600*, 2020. 8