Supplemental Material: Analyzing and Improving the Image Quality of StyleGAN

Tero Karras
NVIDIA
tkarras@nvidia.com

Samuli Laine
NVIDIA
slaine@nvidia.com

Miika Aittala
NVIDIA
jlehtinen@nvidia.com

Janne Hellsten
NVIDIA
jhellsten@nvidia.com

Jaakko Lehtinen
NVIDIA and Aalto University
jlehtinen@nvidia.com

Timo Aila
NVIDIA
taila@nvidia.com

A. Image quality

We include several large images that illustrate various aspects related to image quality. Figure 1 shows hand-picked examples illustrating the quality and diversity achievable using our method in FFHQ, while Figure 2 shows uncurated results for all datasets mentioned in the paper.

Figures 3 and 4 demonstrate cases where FID and P&R give non-intuitive results, but PPL seems to be more in line with human judgement.

We also include images relating to StyleGAN artifacts. Figure 5 shows a rare case where the blob artifact fails to appear in StyleGAN activations, leading to a seriously broken image. Figure 6 visualizes the activations inside Table 1 of the paper configurations A and F. It is evident that progressive growing leads to higher-frequency content in the intermediate layers, compromising shift invariance of the network. We hypothesize that this causes the observed uneven location preference for details when progressive growing is used.

B. Implementation details

We implemented our techniques on top of the official TensorFlow implementation of StyleGAN[1] corresponding to configuration A in Table 1 of the paper. We kept most of the details unchanged, including the dimensionality of $\mathcal{Z}$ and $\mathcal{W}$ (512), mapping network architecture (8 fully connected layers, $100\times$ lower learning rate), equalized learning rate for all trainable parameters [5], leaky ReLU activation with $\alpha = 0.2$, bilinear filtering [13] in all up/downsampling layers [6], minibatch standard deviation layer at the end of the discriminator [5], exponential moving average of generator weights [5], style mixing regularization [6], non-saturating logistic loss [4] with R_1 regularization [9], Adam optimizer [7] with the same hyperparameters ($\beta_1 = 0, \beta_2 = 0.99, \epsilon = 10^{-8}, \text{minibatch} = 32$), and training datasets [6, 12]. We performed all training runs on NVIDIA DGX-1 with 8 Tesla V100 GPUs using TensorFlow 1.14.0 and cuDNN 7.4.2.

Generator redesign In configurations B–F we replace the original StyleGAN generator with our revised architecture. In addition to the changes highlighted in Section 2 of the paper, we initialize components of the constant input c_1 using $\mathcal{N}(0, 1)$ and simplify the noise broadcast operations to use a single shared scaling factor for all feature maps. Similar to Karras et al. [6], we initialize all weights using $\mathcal{N}(0, 1)$ and all biases and noise scaling factors to zero, except for the biases of the affine transformation layers, which we initialize to one. We employ weight modulation and demodulation in all convolution layers, except for the output layers (tRGB in Figure 7 of the paper) where we leave out the demodulation. With 1024^2 output resolution, the generator contains a total of 18 affine transformation layers where the first one corresponds to 4^2 resolution, the next two correspond to 8^2, and so forth.

Weight demodulation Considering the practical implementation of Equations 1 and 3 of the paper, it is important to note that the resulting set of weights will be different for each sample in a minibatch, which rules out direct implementation using standard convolution primitives. Instead, we choose to employ *grouped convolutions* [8] that were originally proposed as a way to reduce computational costs by dividing the input feature maps into multiple independent groups, each with their own dedicated set of weights. We implement Equations 1 and 3 of the paper by temporarily reshaping the weights and activations so that each convolution sees one sample with N groups — instead of N

[1] https://github.com/NVlabs/stylegan

Figure 1. Four hand-picked examples illustrating the image quality and diversity achievable using StylegGAN2 (config F).

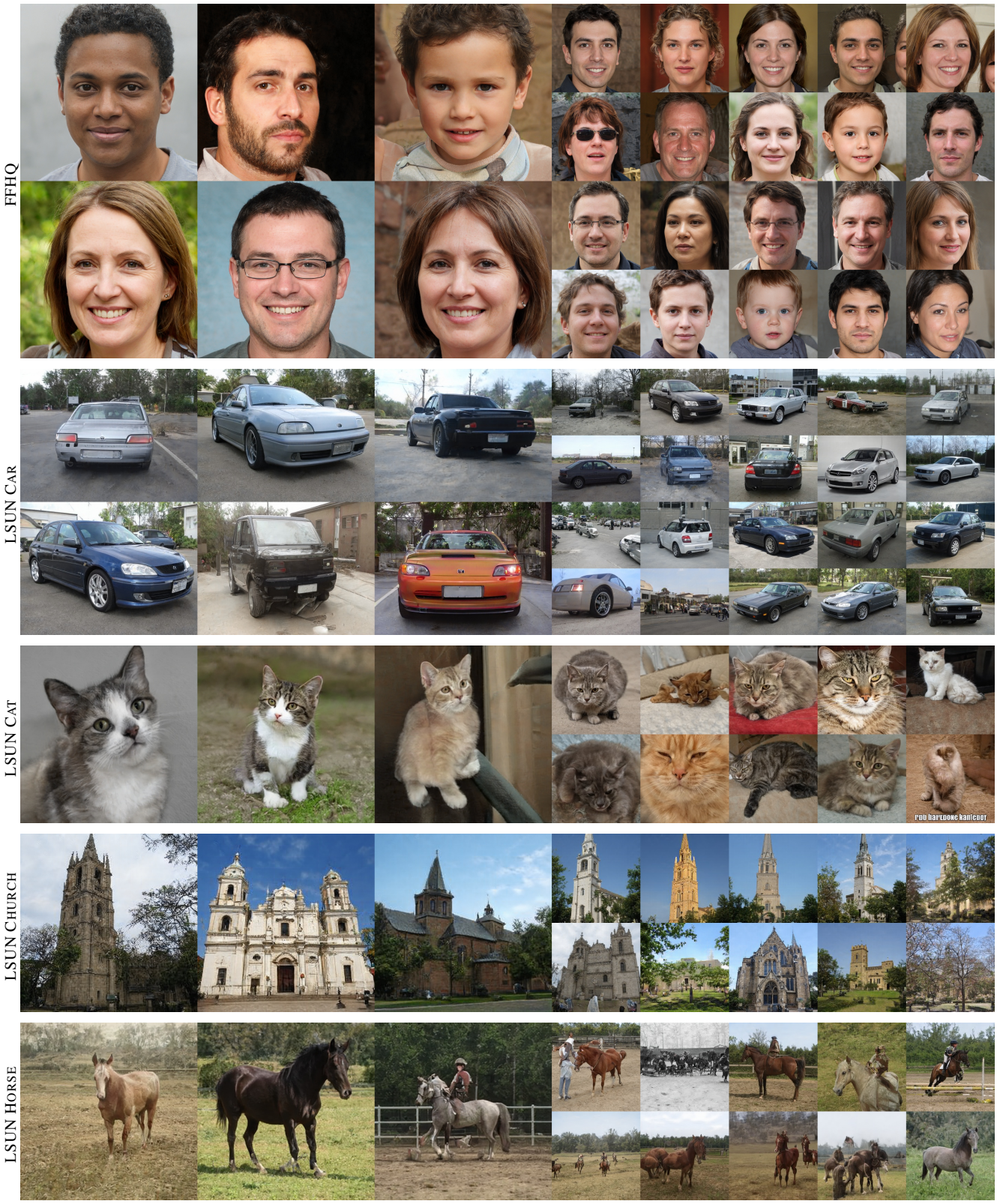
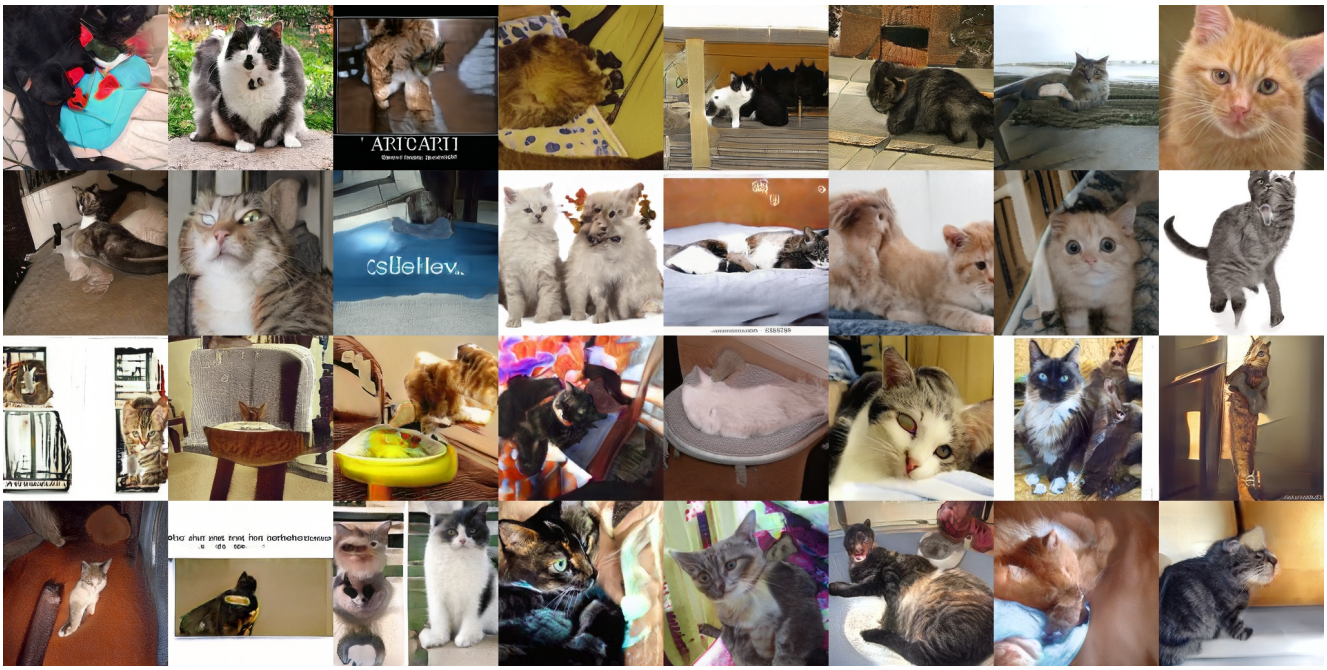


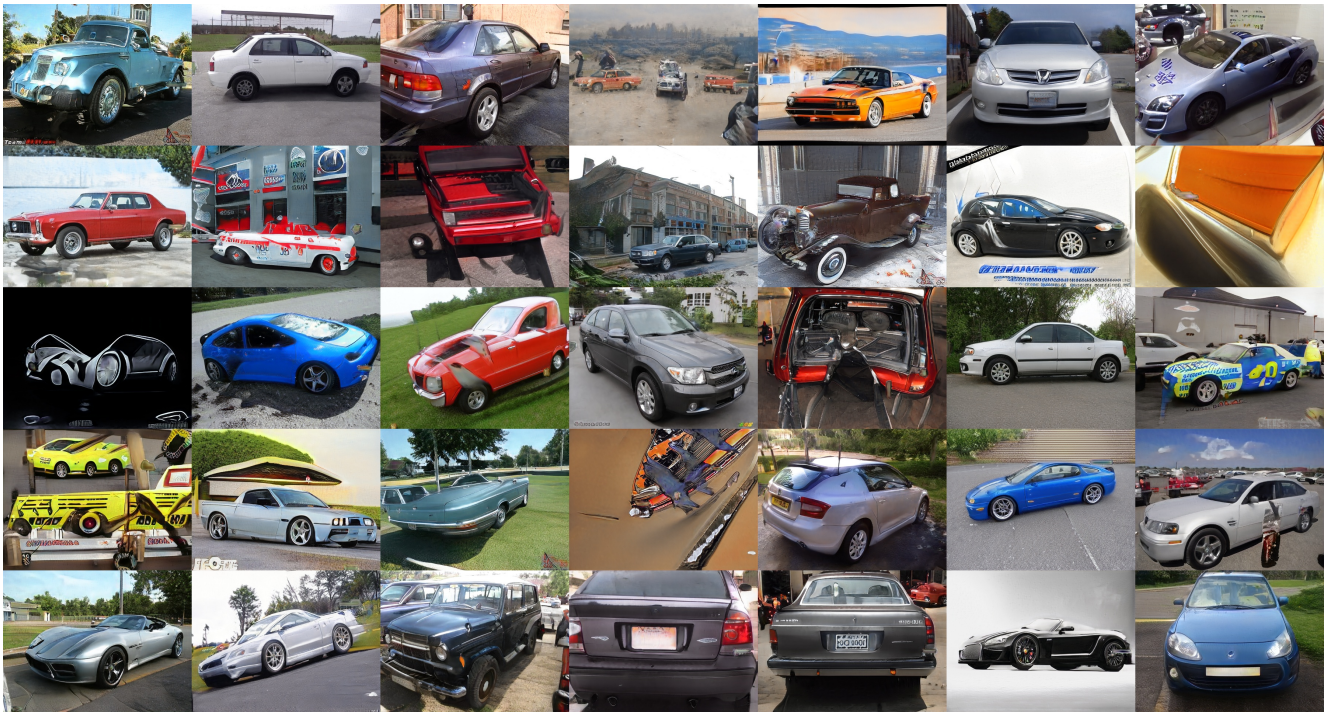
Figure 2. Uncurated results for each dataset used in Tables 1 and 3 of the paper. The images correspond to random outputs produced by our generator (config F), with truncation applied at all resolutions using $\psi = 0.5$ [6].



Model 1: FID = 8.53, P = 0.64, R = 0.28, PPL = 924

Model 2: FID = 8.53, P = 0.62, R = 0.29, PPL = 387

Figure 3. Uncurated examples from two generative models trained on LSUN CAT without truncation. FID, precision, and recall are similar for models 1 and 2, even though the latter produces cat-shaped objects more often. Perceptual path length (PPL) indicates a clear preference for model 2. Model 1 corresponds to configuration A in Table 3 of the paper, and model 2 is an early training snapshot of configuration F.

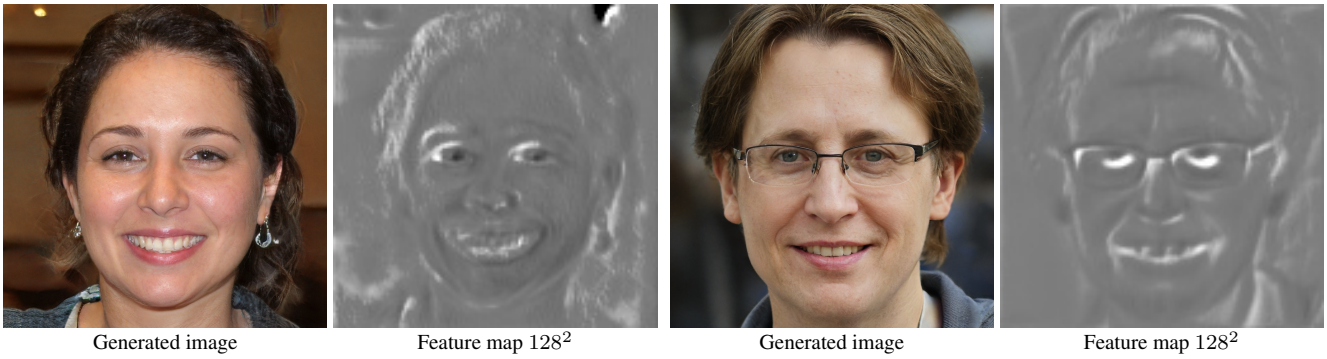


Model 1: FID = 3.27, P = 0.70, R = 0.44, PPL = 1485

Model 2: FID = 3.27, P = 0.67, R = 0.48, PPL = 437

Figure 4. Uncurated examples from two generative models trained on LSUN CAR without truncation. FID, precision, and recall are similar for models 1 and 2, even though the latter produces car-shaped objects more often. Perceptual path length (PPL) indicates a clear preference for model 2. Model 1 corresponds to configuration A in Table 3 of the paper, and model 2 is an early training snapshot of configuration F.

Figure 5. An example of the importance of the droplet artifact in StyleGAN generator. We compare two generated images, one successful and one severely corrupted. The corresponding feature maps were normalized to the viewable dynamic range using instance normalization. For the top image, the droplet artifact starts forming in 64^2 resolution, is clearly visible in 128^2, and increasingly dominates the feature maps in higher resolutions. For the bottom image, 64^2 is qualitatively similar to the top row, but the droplet does not materialize in 128^2. Consequently, the facial features are stronger in the normalized feature map. This leads to an overshoot in 256^2, followed by multiple spurious droplets forming in subsequent resolutions. Based on our experience, it is rare that the droplet is missing from StyleGAN images, and indeed the generator fully relies on its existence.



(a) Progressive growing (config A)

(b) Without progressive growing (config F)

Figure 6. Progressive growing leads to significantly higher frequency content in the intermediate layers. This compromises shift-invariance of the network and makes it harder to localize features precisely in the higher-resolution layers.

samples with one group. This approach is highly efficient because the reshaping operations do not actually modify the contents of the weight and activation tensors.

Lazy regularization In configurations C–F we employ lazy regularization (Section 3.1 of the paper) by evaluating the regularization terms (R_1 and path length) in a separate regularization pass that we execute once every k training iterations. We share the internal state of the Adam optimizer between the main loss and the regularization terms, so that the optimizer first sees gradients from the main loss for k iterations, followed by gradients from the regularization terms for one iteration. To compensate for the fact that we now perform $k + 1$ training iterations instead of k, we adjust the optimizer hyperparameters $\lambda' = c{\cdot}\lambda$, $\beta_1' = (\beta_1)^c$, and $\beta_2' = (\beta_2)^c$, where $c = k/(k + 1)$. We also multiply the regularization term by k to balance the overall magnitude of its gradients. We use $k = 16$ for the discriminator and $k = 8$ for the generator.

Path length regularization Configurations D–F include our new path length regularizer (Section 3.2 of the paper).

We initialize the target scale a to zero and track it on a per-GPU basis as the exponential moving average of $\left\|\mathbf{J}_{\mathbf{w}}^T\mathbf{y}\right\|_2$ using decay coefficient $\beta_{\text{pl}} = 0.99$. We weight our regularization term by

$$\gamma_{\text{pl}} = \frac{\ln 2}{r^2(\ln r - \ln 2)}, \tag{1}$$

where r specifies the output resolution (e.g. $r = 1024$). We have found these parameter choices to work reliably across all configurations and datasets. To ensure that our regularizer interacts correctly with style mixing regularization, we compute it as an average of all individual layers of the synthesis network. Appendix C provides detailed analysis of the effects of our regularizer on the mapping between $\mathcal{W}$ and image space.

Progressive growing In configurations A–D we use progressive growing with the same parameters as Karras et al. [6] (start at 8^2 resolution and learning rate $\lambda = 10^{-3}$, train for 600k images per resolution, fade in next resolution for 600k images, increase learning rate gradually by $3\times$). In configurations E–F we disable progressive growing and set the learning rate to a fixed value $\lambda = 2 \cdot 10^{-3}$, which we found to provide the best results. In addition, we use output skips in the generator and residual connections in the discriminator as detailed in Section 4.1 of the paper.

Dataset-specific tuning Similar to Karras et al. [6], we augment the FFHQ dataset with horizontal flips to effectively increase the number of training images from 70k to 140k, and we do not perform any augmentation for the LSUN datasets. We have found that the optimal choices for the training length and R_1 regularization weight γ tend to vary considerably between datasets and configurations. We use $\gamma = 10$ for all training runs except for configuration E in Table 1 of the paper, as well as LSUN CHURCH and LSUN HORSE in Table 3 of the paper, where we use $\gamma = 100$. It is possible that further tuning of γ could provide additional benefits.

Performance optimizations We profiled our training runs extensively and found that — in our case — the default primitives for image filtering, up/downsampling, bias addition, and leaky ReLU had surprisingly high overheads in terms of training time and GPU memory footprint. This motivated us to optimize these operations using hand-written CUDA kernels. We implemented filtered up/downsampling as a single fused operation, and bias and activation as another one. In configuration E at 1024^2 resolution, our optimizations improved the overall training time by about 30% and memory footprint by about 20%.

C. Effects of path length regularization

The path length regularizer described in Section 3.2 of the paper is of the form:

$$\mathcal{L}_{\text{pl}} = \mathbb{E}_{\mathbf{w}}\mathbb{E}_{\mathbf{y}}\left(\left\|\mathbf{J}_{\mathbf{w}}^T\mathbf{y}\right\|_2 - a\right)^2, \tag{2}$$

where $\mathbf{y} \in \mathbb{R}^M$ is a unit normal distributed random variable in the space of generated images (of dimension $M = 3wh$, namely the RGB image dimensions), $\mathbf{J}_{\mathbf{w}} \in \mathbb{R}^{M\times L}$ is the Jacobian matrix of the generator function $g : \mathbb{R}^L \mapsto \mathbb{R}^M$ at a latent space point $\mathbf{w} \in \mathbb{R}^L$, and $a \in \mathbb{R}$ is a global value that expresses the desired scale of the gradients.

C.1. Effect on pointwise Jacobians

The value of this prior is minimized when the inner expectation over $\mathbf{y}$ is minimized at every latent space point $\mathbf{w}$ separately. In this subsection, we show that the inner expectation is (approximately) minimized when the Jacobian matrix $\mathbf{J}_{\mathbf{w}}$ is orthogonal, up to a global scaling factor. The general strategy is to use the well-known fact that, in high dimensions L, the density of a unit normal distribution is concentrated on a spherical shell of radius $\sqrt{L}$. The inner expectation is then minimized when the matrix $\mathbf{J}_{\mathbf{w}}^T$ scales the function under expectation to have its minima at this radius. This is achieved by any orthogonal matrix (with suitable global scale that is the same at every $\mathbf{w}$).

We begin by considering the inner expectation

$$\mathcal{L}_{\mathbf{w}} := \mathbb{E}_{\mathbf{y}}\left(\left\|\mathbf{J}_{\mathbf{w}}^T\mathbf{y}\right\|_2 - a\right)^2.$$

We first note that the radial symmetry of the distribution of $\mathbf{y}$, as well as of the l_2 norm, allows us to focus on diagonal matrices only. This is seen using the Singular Value Decomposition $\mathbf{J}_{\mathbf{w}}^T = \mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{L\times L}$ and $\mathbf{V} \in \mathbb{R}^{M\times M}$ are orthogonal matrices, and $\tilde{\mathbf{\Sigma}} = [\mathbf{\Sigma}\ \mathbf{0}]$ is a horizontal concatenation of a diagonal matrix $\mathbf{\Sigma} \in \mathbb{R}^{L\times L}$ and a zero matrix $\mathbf{0} \in \mathbb{R}^{L\times(M-L)}$ [3]. Because rotating a unit normal random variable by an orthogonal matrix leaves the distribution unchanged, and rotating a vector leaves its norm unchanged, the expression simplifies to

$$\begin{aligned}\mathcal{L}_{\mathbf{w}} &= \mathbb{E}_{\mathbf{y}}\left(\left\|\mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^T\mathbf{y}\right\|_2 - a\right)^2 \\ &= \mathbb{E}_{\mathbf{y}}\left(\left\|\tilde{\mathbf{\Sigma}}\mathbf{y}\right\|_2 - a\right)^2.\end{aligned}$$

Furthermore, the zero matrix in $\tilde{\mathbf{\Sigma}}$ drops the dimensions of $\mathbf{y}$ beyond L, effectively marginalizing its distribution over those dimensions. The marginalized distribution is again a unit normal distribution over the remaining L dimensions. We are then left to consider the minimization of the expression

$$\mathcal{L}_{\mathbf{w}} = \mathbb{E}_{\tilde{\mathbf{y}}}\left(\left\|\mathbf{\Sigma}\tilde{\mathbf{y}}\right\|_2 - a\right)^2,$$

over diagonal square matrices $\mathbf{\Sigma} \in \mathbb{R}^{L\times L}$, where $\tilde{\mathbf{y}}$ is unit normal distributed in dimension L. To summarize, all matrices $\mathbf{J}_{\mathbf{w}}^T$ that share the same singular values with $\mathbf{\Sigma}$ produce the same value for the original loss.

Next, we show that this expression is minimized when the diagonal matrix $\mathbf{\Sigma}$ has a specific identical value at every diagonal entry, i.e., it is a constant multiple of an identity matrix. We first write the expectation as an integral over the probability density of $\tilde{\mathbf{y}}$:

$$\begin{aligned}\mathcal{L}_{\mathbf{w}} &= \int \left(\|\mathbf{\Sigma}\tilde{\mathbf{y}}\|_2 - a\right)^2 p_{\tilde{\mathbf{y}}}(\tilde{\mathbf{y}})\, \mathrm{d}\tilde{\mathbf{y}} \\ &= (2\pi)^{-\frac{L}{2}} \int \left(\|\mathbf{\Sigma}\tilde{\mathbf{y}}\|_2 - a\right)^2 \exp\left(-\frac{\tilde{\mathbf{y}}^T\tilde{\mathbf{y}}}{2}\right)\, \mathrm{d}\tilde{\mathbf{y}}\end{aligned}$$

Observing the radially symmetric form of the density, we change into a polar coordinates $\tilde{\mathbf{y}} = r\phi$, where $r \in \mathbb{R}_+$ is the distance from origin, and $\phi \in \mathbb{S}^{L-1}$ is a unit vector, i.e., a point on the $L-1$-dimensional unit sphere. This change of variables introduces a Jacobian factor r^{L-1}:

$$\tilde{\mathcal{L}}_{\mathbf{w}} = (2\pi)^{-\frac{L}{2}} \int_{\mathbb{S}}\int_0^{\infty} \left(r\,\|\mathbf{\Sigma}\phi\|_2 - a\right)^2 r^{L-1} \exp\left(-\frac{r^2}{2}\right)\, \mathrm{d}r\, \mathrm{d}\phi$$

The probability density $(2\pi)^{-L/2} r^{L-1} \exp\left(-\frac{r^2}{2}\right)$ is then an L-dimensional unit normal density expressed in polar coordinates, dependent only on the radius and not on the angle. A standard argument by Taylor approximation shows that when L is high, for any ϕ the density is well approximated by density $(2\pi e/L)^{-L/2} \exp\left(-\frac{1}{2}(r-\mu)^2/\sigma^2\right)$, which is a (unnormalized) one-dimensional normal density in r, centered at $\mu = \sqrt{L}$ of standard deviation $\sigma = 1/\sqrt{2}$ [1]. In other words, the density of the L-dimensional unit normal distribution is concentrated on a shell of radius $\sqrt{L}$. Substituting this density into the integral, the loss becomes approximately

$$\mathcal{L}_{\mathbf{w}} \approx (2\pi e/L)^{-L/2} \int_{\mathbb{S}}\int_0^{\infty} \left(r\,\|\mathbf{\Sigma}\phi\|_2 - a\right)^2 \exp\left(-\frac{\left(r-\sqrt{L}\right)^2}{2\sigma^2}\right)\, \mathrm{d}r\, \mathrm{d}\phi, \quad (3)$$

where the approximation becomes exact in the limit of infinite dimension L.

To minimize this loss, we set $\mathbf{\Sigma}$ such that the function $\left(r\,\|\mathbf{\Sigma}\phi\|_2 - a\right)^2$ obtains minimal values on the spherical shell of radius $\sqrt{L}$. This is achieved by $\mathbf{\Sigma} = \frac{a}{\sqrt{L}}\mathbf{I}$, whereby the function becomes constant in ϕ and the expression reduces to

$$\mathcal{L}_{\mathbf{w}} \approx (2\pi e/L)^{-L/2} \mathcal{A}(\mathbb{S}) a^2 L^{-1} \int_0^{\infty} \left(r-\sqrt{L}\right)^2 \exp\left(-\frac{\left(r-\sqrt{L}\right)^2}{2\sigma^2}\right)\, \mathrm{d}r,$$

where $\mathcal{A}(\mathbb{S})$ is the surface area of the unit sphere (and like the other constant factors, irrelevant for minimization). Note that the zero of the parabola $(r-\sqrt{L})^2$ coincides with the maximum of the probability density, and therefore this choice of Σ minimizes the inner integral in Eq. 3 separately for every ϕ.

In summary, we have shown that — assuming a high dimensionality L of the latent space — the value of the path length prior (Eq. 2) is minimized when all singular values of the Jacobian matrix of the generator are equal to a global constant, at every latent space point $\mathbf{w}$, i.e., they are orthogonal up to a globally constant scale.

While in theory a merely scales the values of the mapping without changing its properties and could be set to a fixed value (e.g., 1), in practice it does affect the dynamics of the training. If the imposed scale does not match the scale induced by the random initialization of the network, the training spends its critical early steps in pushing the weights towards the required overall magnitudes, rather than enforcing the actual objective of interest. This may degrade the internal state of the network weights and lead to sub-optimal performance in later training. Empirically we find that setting a fixed scale reduces the consistency of the training results across training runs and datasets. Instead, we set a dynamically based on a running average of the existing scale of the Jacobians, namely $a \approx \mathbb{E}_{\mathbf{w},\mathbf{y}}\left(\left\|\mathbf{J}_w^T\mathbf{y}\right\|_2\right)$. With this choice the prior targets the scale of the local Jacobians towards whatever global average already exists, rather than forcing a specific global average. This also eliminates the need to measure the appropriate scale of the Jacobians explicitly, as is done by Odena et al. [11] who consider a related conditioning prior.

Figure 7 shows empirically measured magnitudes of singular values of the Jacobian matrix for networks trained with and without path length regularization. While orthogonality is not reached, the eigenvalues of the regularized network are closer to one another, implying better conditioning, with the strength of the effect correlated with the PPL metric (Table 1 of the paper).

C.2. Effect on global properties of generator mapping

In the previous subsection, we found that the prior encourages the Jacobians of the generator mapping to be everywhere orthogonal. While Figure 7 shows that the map-

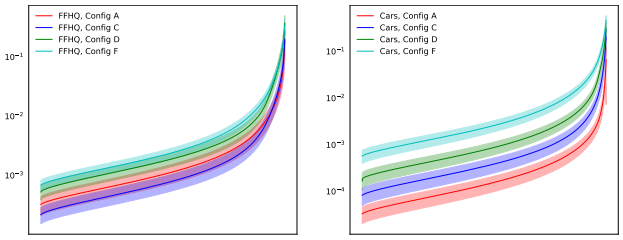


Figure 7. The mean and standard deviation of the magnitudes of sorted singular values of the Jacobian matrix evaluated at random latent space points $\mathbf{w}$, with largest eigenvalue normalized to 1. In both datasets, path length regularization (Config D) and novel architecture (Config F) exhibit better conditioning; notably, the effect is more pronounced in the Cars dataset that contains much more variability, and where path length regularization has a relatively stronger effect on the PPL metric (Table 1 of the paper).

ping does not satisfy this constraint exactly in practice, it is instructive to consider what global properties the constraint implies for mappings that do. Without loss of generality, we assume unit global scale for the matrices to simplify the presentation.

The key property is that that a mapping $g : \mathbb{R}^L \mapsto \mathbb{R}^M$ with everywhere orthogonal Jacobians preserves the lengths of curves. To see this, let $u : [t_0, t_1] \mapsto \mathbb{R}^L$ parametrize a curve in the latent space. Mapping the curve through the generator g, we obtain a curve $\tilde{u} = g \circ u$ in the space of images. Its arc length is

$$L = \int_{t_0}^{t_1} |\tilde{u}'(t)| \, \mathrm{d}t, \tag{4}$$

where prime denotes derivative with respect to t. By chain rule, this equals

$$L = \int_{t_0}^{t_1} |J_g(u(t))u'(t)| \, \mathrm{d}t, \tag{5}$$

where $J_g \in \mathbb{R}^{L \times M}$ is the Jacobian matrix of g evaluated at $u(t)$. By our assumption, the Jacobian is orthogonal, and consequently it leaves the 2-norm of the vector $u'(t)$ unaffected:

$$L = \int_{t_0}^{t_1} |u'(t)| \, \mathrm{d}t. \tag{6}$$

This is the length of the curve u in the latent space, prior to mapping with g. Hence, the lengths of u and $\tilde{u}$ are equal, and so g preserves the length of any curve.

In the language of differential geometry, g isometrically embeds the Euclidean latent space $\mathbb{R}^L$ into a submanifold $\mathcal{M}$ in $\mathbb{R}^M$ — e.g., the manifold of images representing faces, embedded within the space of all possible RGB images. A consequence of isometry is that straight line segments in the latent space are mapped to geodesics, or shortest paths, on the image manifold: a straight line v that connects two latent space points cannot be made any shorter, so neither can there be a shorter on-manifold image-space path between the corresponding images than $g \circ v$. For example, a geodesic on the manifold of face images is a continuous morph between two faces that incurs the minimum total amount of change (as measured by l_2 difference in RGB space) when one sums up the image difference in each step of the morph.

Isometry is not achieved in practice, as demonstrated in empirical experiments in the previous subsection. The full loss function of the training is a combination of potentially conflicting criteria, and it is not clear if a genuinely isometric mapping would be capable of expressing the image manifold of interest. Nevertheless, a pressure to make the mapping as isometric as possible has desirable consequences. In particular, it discourages unnecessary "detours": in a non-constrained generator mapping, a latent space interpolation between two similar images may pass through any number of distant images in RGB space. With regularization, the mapping is encouraged to place distant images in different regions of the latent space, so as to obtain short image paths between any two endpoints.

D. Projection method details

Given a target image $\boldsymbol{x}$, we seek to find the corresponding $\mathbf{w} \in \mathcal{W}$ and per-layer noise maps denoted $\boldsymbol{n}_i \in \mathbb{R}^{r_i \times r_i}$ where i is the layer index and r_i denotes the resolution of the ith noise map. The baseline StyleGAN generator in 1024×1024 resolution has 18 noise inputs, i.e., two for each resolution from 4×4 to 1024×1024 pixels. Our improved architecture has one fewer noise input because we do not add noise to the learned 4×4 constant (Figure 1 of the paper).

Before optimization, we compute $\boldsymbol{\mu}_{\mathbf{w}} = \mathbb{E}_{\mathbf{z}} \, f(\mathbf{z})$ by running 10 000 random latent codes $\mathbf{z}$ through the mapping network f. We also approximate the scale of $\mathcal{W}$ by computing $\sigma_{\mathbf{w}}^2 = \mathbb{E}_{\mathbf{z}} \, \|f(\mathbf{z}) - \boldsymbol{\mu}_{\mathbf{w}}\|_2^2$, i.e., the average square Euclidean distance to the center.

At the beginning of optimization, we initialize $\mathbf{w} = \boldsymbol{\mu}_{\mathbf{w}}$ and $\boldsymbol{n}_i = \mathcal{N}(\mathbf{0}, \mathbf{I})$ for all i. The trainable parameters are the components of $\mathbf{w}$ as well as all components in all noise maps $\boldsymbol{n}_i$. The optimization is run for 1000 iterations using Adam optimizer [7] with default parameters. Maximum learning rate is $\lambda_{max} = 0.1$, and it is ramped up from zero linearly during the first 50 iterations and ramped down to zero using a cosine schedule during the last 250 iterations. In the first three quarters of the optimization we add Gaussian noise to $\mathbf{w}$ when evaluating the loss function as $\tilde{\mathbf{w}} = \mathbf{w} + \mathcal{N}(0, 0.05 \, \sigma_{\mathbf{w}} t^2)$, where t goes from one to zero during the first 750 iterations. This adds stochasticity to the optimization and stabilizes finding of the global optimum.

Given that we are explicitly optimizing the noise maps, we must be careful to avoid the optimization from sneaking actual signal into them. Thus we include several noise

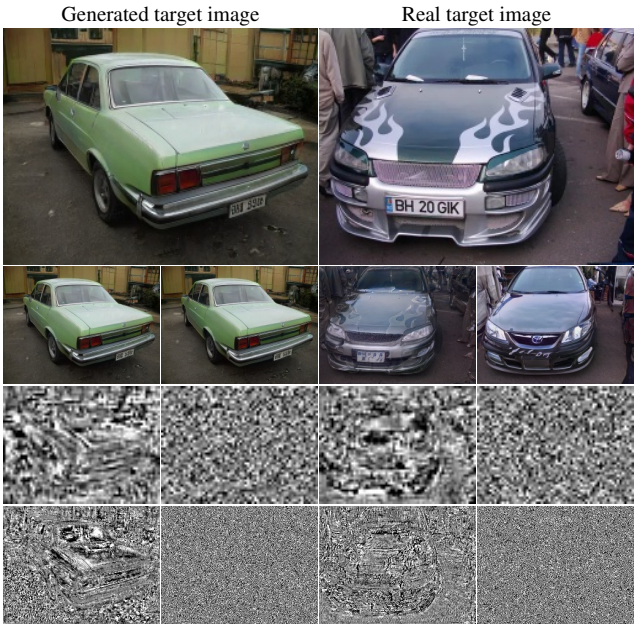


Figure 8. Effect of noise regularization in latent-space projection where we also optimize the contents of the noise inputs of the synthesis network. Top to bottom: target image, re-synthesized image, contents of two noise maps at different resolutions. When regularization is turned off in this test, we only normalize the noise maps to zero mean and unit variance, which leads the optimization to sneak signal into the noise maps. Enabling the noise regularization prevents this. The model used here corresponds to configuration F in Table 1 of the paper.

map regularization terms in our loss function, in addition to an image quality term. The image quality term is the LPIPS [14] distance between target image $\boldsymbol{x}$ and the synthesized image: $L_{image} = D_{\text{LPIPS}}[\boldsymbol{x}, g(\tilde{\mathbf{w}}, \boldsymbol{n}_0, \boldsymbol{n}_1, \ldots)]$. For increased performance and stability, we downsample both images to 256×256 resolution before computing the LPIPS distance. Regularization of the noise maps is performed on multiple resolution scales. For this purpose, we form for each noise map greater than 8×8 in size a pyramid down to 8×8 resolution by averaging 2×2 pixel neighborhoods and multiplying by 2 at each step to retain the expected unit variance. These downsampled noise maps are used for regularization only and have no part in synthesis.

Let us denote the original noise maps by $\boldsymbol{n}_{i,0} = \boldsymbol{n}_i$ and the downsampled versions by $\boldsymbol{n}_{i,j>0}$. Similarly, let $r_{i,j}$ be the resolution of an original ($j = 0$) or downsampled ($j > 0$) noise map so that $r_{i,j+1} = r_{i,j}/2$. The regularization term for noise map $\boldsymbol{n}_{i,j}$ is then

$$L_{i,j} = \left(\frac{1}{r_{i,j}^2} \cdot \sum_{x,y} \boldsymbol{n}_{i,j}(x,y) \cdot \boldsymbol{n}_{i,j}(x-1,y) \right)^2 + \left(\frac{1}{r_{i,j}^2} \cdot \sum_{x,y} \boldsymbol{n}_{i,j}(x,y) \cdot \boldsymbol{n}_{i,j}(x,y-1) \right)^2,$$

where the noise map is considered to wrap at the edges. The regularization term is thus sum of squares of the resolution-normalized autocorrelation coefficients at one pixel shifts horizontally and vertically, which should be zero for a normally distributed signal. The overall loss term is then $L_{total} = L_{image} + \alpha \sum_{i,j} L_{i,j}$. In all our tests, we have used noise regularization weight $\alpha = 10^5$. In addition, we renormalize all noise maps to zero mean and unit variance after each optimization step. Figure 8 illustrates the effect of noise regularization on the resulting noise maps.

	SN-G	SN-D	Demod	P.reg	FID ↓	PPL ↓	Pre. ↑	Rec. ↑
1	–	–	✓	✓	**2.83**	145.0	0.689	**0.492**
2	–	✓	✓	✓	2.98	131.4	0.700	0.469
3	✓	✓	✓	✓	3.40	**130.9**	**0.720**	0.435
4	✓	✓	–	✓	3.38	162.6	0.705	0.468
5	✓	✓	–	–	3.33	394.9	0.705	0.463
6	✓	–	–	✓	3.36	217.1	0.695	0.464
7	✓	–	–	–	3.22	394.4	0.692	0.489

Table 1. Effect of spectral normalization with FFHQ at 1024^2. The first row corresponds to StyleGAN2, i.e., config F in Table 1 of the paper. In the subsequent rows, we enable spectral normalization in the generator (*SN-G*) and in the discriminator (*SN-D*). We also test the training without weight demodulation (*Demod*) and path length regularization (*P.reg*). All of these configurations are highly detrimental to FID, as well as to Recall. ↑ indicates that higher is better, and ↓ that lower is better.

E. Results with spectral normalization

Since spectral normalization (SN) is widely used in GANs [10], we investigated its effect on StyleGAN2. Table 1 gives the results for a variety of configurations where spectral normalization is enabled in addition to our techniques (weight demodulation, path length regularization) or instead of them.

Interestingly, adding spectral normalization to our generator is almost a no-op. On an implementation level, SN scales the weight tensor of each layer with a scalar value $1/\sigma(w)$. The effect of such scaling, however, is overridden by Equation 3 of the paper for the main convolutional layers as well as the affine transformation layers. Thus, the only thing that SN adds on top of weight demodulation is through its effect on the `tRGB` layers.

When we enable spectral normalization in the discriminator, FID is slightly compromised. Enabling it in the generator as well leads to significantly worse results, even

Item	GPU years (Volta)	Electricity (MWh)
Initial exploration	20.25	58.94
Paper exploration	13.71	31.49
FFHQ config F	0.23	0.68
Other runs in paper	7.20	16.77
Backup runs left out	4.73	12.08
Video, figures, etc.	0.31	0.82
Public release	4.62	10.82
Total	51.05	131.61

Table 2. Computational effort expenditure and electricity consumption data for this project. The unit for computation is GPU-years on a single NVIDIA V100 GPU — it would have taken approximately 51 years to execute this project using a single GPU. See the text for additional details about the computation and energy consumption estimates. *Initial exploration* includes all training runs after the release of StyleGAN [6] that affected our decision to start this project. *Paper exploration* includes all training runs that were done specifically for this project, but were not intended to be used in the paper as-is. *FFHQ config F* refers to the training of the final network. This is approximately the cost of training the network for another dataset without hyperparameter tuning. *Other runs in paper* covers the training of all other networks shown in the paper. *Backup runs left out* includes the training of various networks that could potentially have been shown in the paper, but were ultimately left out to keep the exposition more focused. *Video, figures, etc.* includes computation that was spent on producing the images and graphs in the paper, as well as on the result video. *Public release* covers testing, benchmarking, and large-scale image dumps related to the public release.

though its effect is isolated to the `tRGB` layers. Leaving SN enabled, but disabling a subset of our contributions does not improve the situation. Thus we conclude that StyleGAN2 gives better results without spectral normalization.

F. Energy consumption

Computation is a core resource in any machine learning project: its availability and cost, as well as the associated energy consumption, are key factors in both choosing research directions and practical adoption. We provide a detailed breakdown for our entire project in Table 2 in terms of both GPU time and electricity consumption.

We report expended computational effort as single-GPU years (Volta class GPU). We used a varying number of NVIDIA DGX-1s for different stages of the project, and converted each run to single-GPU equivalents by simply scaling by the number of GPUs used.

The entire project consumed approximately 131.61 megawatt hours (MWh) of electricity. We followed the Green500 power measurements guidelines [2] as follows. For each job, we logged the exact duration, number of GPUs used, and which of our two separate compute clusters the job was executed on. We then measured the actual power draw of an 8-GPU DGX-1 when it was training FFHQ config F. A separate estimate was obtained for the two clusters because they use different DGX-1 SKUs. The vast majority of our training runs used 8 GPUs, and for the rest we approximated the power draw by scaling linearly with $n/8$, where n is the number of GPUs.

Approximately half of the total energy was spent on early exploration and forming ideas. Then subsequently a quarter was spent on refining those ideas in more targeted experiments, and finally a quarter on producing this paper and preparing the public release of code, trained models, and large sets of images. Training a single FFHQ network (config F) took approximately 0.68 MWh (0.5% of the total project expenditure). This is the cost that one would pay when training the network from scratch, possibly using a different dataset. In short, vast majority of the electricity used went into shaping the ideas, testing hypotheses, and hyperparameter tuning. We did not use automated tools for finding hyperparameters or optimizing network architectures.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 8

[2] R. Ge, X. Feng, H. Pyla, K. Cameron, and W. Feng. Power measurement tutorial for the Green500 list. https://www.top500.org/green500/resources/tutorials/, Accessed March 1, 2020. 11

[3] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013. 7

[4] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *NIPS*, 2014. 1

[5] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. 1

[6] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proc. CVPR*, 2018. 1, 3, 7, 11

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 1, 9

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105. 2012. 1

[9] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? *CoRR*, abs/1801.04406, 2018. 1

[10] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018. 10

[11] Augustus Odena, Jacob Buckman, Catherine Olsson, Tom B. Brown, Christopher Olah, Colin Raffel, and Ian Goodfellow. Is generator conditioning causally related to GAN performance? *CoRR*, abs/1802.08768, 2018. 8

[12] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015. 1

[13] Richard Zhang. Making convolutional networks shift-invariant again. In *Proc. ICML*, 2019. 1

[14] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018. 10