Analyzing and Improving the Image Quality of StyleGAN

Tero Karras NVIDIA
Samuli Laine NVIDIA
Miika Aittala NVIDIA
Janne Hellsten NVIDIA
Jaakko Lehtinen NVIDIA and Aalto University
Timo Aila NVIDIA

Abstract

The style-based GAN architecture (StyleGAN) yields state-of-the-art results in data-driven unconditional generative image modeling. We expose and analyze several of its characteristic artifacts, and propose changes in both model architecture and training methods to address them. In particular, we redesign the generator normalization, revisit progressive growing, and regularize the generator to encourage good conditioning in the mapping from latent codes to images. In addition to improving image quality, this path length regularizer yields the additional benefit that the generator becomes significantly easier to invert. This makes it possible to reliably attribute a generated image to a particular network. We furthermore visualize how well the generator utilizes its output resolution, and identify a capacity problem, motivating us to train larger models for additional quality improvements. Overall, our improved model redefines the state of the art in unconditional image modeling, both in terms of existing distribution quality metrics as well as perceived image quality.

1. Introduction

The resolution and quality of images produced by generative methods, especially generative adversarial networks (GAN) [16], are improving rapidly [23, 31, 5]. The current state-of-the-art method for high-resolution image synthesis is StyleGAN [24], which has been shown to work reliably on a variety of datasets. Our work focuses on fixing its characteristic artifacts and improving the result quality further.

The distinguishing feature of StyleGAN [24] is its unconventional generator architecture. Instead of feeding the input latent code $\mathbf{z} \in \mathcal{Z}$ only to the beginning of a the network, the *mapping network* f first transforms it to an intermediate latent code $\mathbf{w} \in \mathcal{W}$. Affine transforms then produce *styles* that control the layers of the *synthesis network* g via adaptive instance normalization (AdaIN) [21, 9, 13, 8]. Additionally, stochastic variation is facilitated by providing additional random noise maps to the synthesis network. It has been demonstrated [24, 38] that this design allows the intermediate latent space $\mathcal{W}$ to be much less entangled than the input latent space $\mathcal{Z}$. In this paper, we focus all analysis solely on $\mathcal{W}$, as it is the relevant latent space from the synthesis network's point of view.

Many observers have noticed characteristic artifacts in images generated by StyleGAN [3]. We identify two causes for these artifacts, and describe changes in architecture and training methods that eliminate them. First, we investigate the origin of common blob-like artifacts, and find that the generator creates them to circumvent a design flaw in its architecture. In Section 2, we redesign the normalization used in the generator, which removes the artifacts. Second, we analyze artifacts related to progressive growing [23] that has been highly successful in stabilizing high-resolution GAN training. We propose an alternative design that achieves the same goal — training starts by focusing on low-resolution images and then progressively shifts focus to higher and higher resolutions — without changing the network topology during training. This new design also allows us to reason about the effective resolution of the generated images, which turns out to be lower than expected, motivating a capacity increase (Section 4).

Quantitative analysis of the quality of images produced using generative methods continues to be a challenging topic. Fréchet inception distance (FID) [20] measures differences in the density of two distributions in the high-dimensional feature space of an InceptionV3 classifier [39]. Precision and Recall (P&R) [36, 27] provide additional visibility by explicitly quantifying the percentage of generated images that are similar to training data and the percentage of training data that can be generated, respectively. We use these metrics to quantify the improvements.

Both FID and P&R are based on classifier networks that have recently been shown to focus on textures rather than shapes [12], and consequently, the metrics do not accurately capture all aspects of image quality. We observe that the perceptual path length (PPL) metric [24], originally introduced as a method for estimating the quality of latent space

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

interpolations, correlates with consistency and stability of shapes. Based on this, we regularize the synthesis network to favor smooth mappings (Section 3) and achieve a clear improvement in quality. To counter its computational expense, we also propose executing all regularizations less frequently, observing that this can be done without compromising effectiveness.

Finally, we find that projection of images to the latent space $\mathcal{W}$ works significantly better with the new, path-length regularized StyleGAN2 generator than with the original StyleGAN. This makes it easier to attribute a generated image to its source (Section 5).

Our implementation and trained models are available at `https://github.com/NVlabs/stylegan2`

2. Removing normalization artifacts

We begin by observing that most images generated by StyleGAN exhibit characteristic blob-shaped artifacts that resemble water droplets. As shown in Figure 1, even when the droplet may not be obvious in the final image, it is present in the intermediate feature maps of the generator.[1] The anomaly starts to appear around 64×64 resolution, is present in all feature maps, and becomes progressively stronger at higher resolutions. The existence of such a consistent artifact is puzzling, as the discriminator should be able to detect it.

We pinpoint the problem to the AdaIN operation that normalizes the mean and variance of each feature map separately, thereby potentially destroying any information found in the magnitudes of the features relative to each other. We hypothesize that the droplet artifact is a result of the generator intentionally sneaking signal strength information past instance normalization: by creating a strong, localized spike that dominates the statistics, the generator can effectively scale the signal as it likes elsewhere. Our hypothesis is supported by the finding that when the normalization step is removed from the generator, as detailed below, the droplet artifacts disappear completely.

[1] In rare cases (perhaps 0.1% of images) the droplet is missing, leading to severely corrupted images. See Appendix A for details.

2.1. Generator architecture revisited

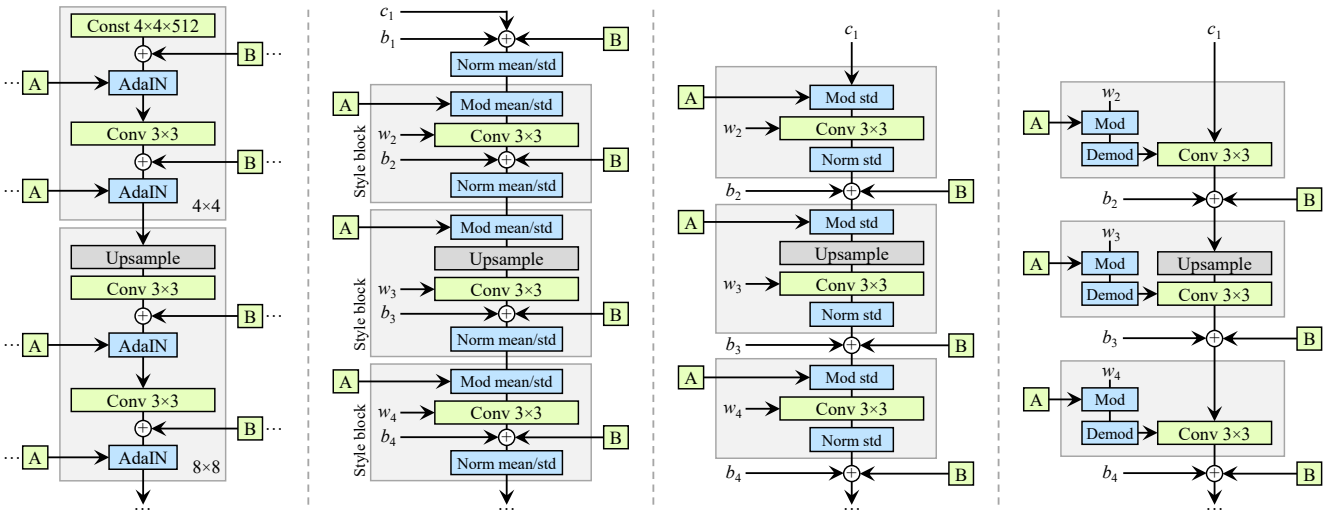
We will first revise several details of the StyleGAN generator to better facilitate our redesigned normalization. These changes have either a neutral or small positive effect on their own in terms of quality metrics.

Figure 2a shows the original StyleGAN synthesis network g [24], and in Figure 2b we expand the diagram to full detail by showing the weights and biases and breaking the AdaIN operation to its two constituent parts: normalization and modulation. This allows us to re-draw the conceptual gray boxes so that each box indicates the part of the network where one style is active (i.e., "style block"). Interestingly, the original StyleGAN applies bias and noise within the style block, causing their relative impact to be inversely proportional to the current style's magnitudes. We observe that more predictable results are obtained by moving these operations outside the style block, where they operate on normalized data. Furthermore, we notice that after this change it is sufficient for the normalization and modulation to operate on the standard deviation alone (i.e., the mean is not needed). The application of bias, noise, and normalization to the constant input can also be safely removed without observable drawbacks. This variant is shown in Figure 2c, and serves as a starting point for our redesigned normalization.

2.2. Instance normalization revisited

One of the main strengths of StyleGAN is the ability to control the generated images via *style mixing*, i.e., by feeding a different latent $\mathbf{w}$ to different layers at inference time. In practice, style modulation may amplify certain feature maps by an order of magnitude or more. For style mixing to work, we must explicitly counteract this amplification on a per-sample basis — otherwise the subsequent layers would not be able to operate on the data in a meaningful way.

If we were willing to sacrifice scale-specific controls (see video), we could simply remove the normalization, thus removing the artifacts and also improving FID slightly [27]. We will now propose a better alternative that removes the artifacts while retaining full controllability. The main idea is to base normalization on the *expected* statistics of the incoming feature maps, but without explicit forcing.



(a) StyleGAN (b) StyleGAN (detailed) (c) Revised architecture (d) Weight demodulation

Figure 2. We redesign the architecture of the StyleGAN synthesis network. (a) The original StyleGAN, where A denotes a learned affine transform from $\mathcal{W}$ that produces a style and B is a noise broadcast operation. (b) The same diagram with full detail. Here we have broken the AdaIN to explicit normalization followed by modulation, both operating on the mean and standard deviation per feature map. We have also annotated the learned weights (w), biases (b), and constant input (c), and redrawn the gray boxes so that one style is active per box. The activation function (leaky ReLU) is always applied right after adding the bias. (c) We make several changes to the original architecture that are justified in the main text. We remove some redundant operations at the beginning, move the addition of b and B to be outside active area of a style, and adjust only the standard deviation per feature map. (d) The revised architecture enables us to replace instance normalization with a "demodulation" operation, which we apply to the weights associated with each convolution layer.

Recall that a style block in Figure 2c consists of modulation, convolution, and normalization. Let us start by considering the effect of a modulation followed by a convolution. The modulation scales each input feature map of the convolution based on the incoming style, which can alternatively be implemented by scaling the convolution weights:

$$w'_{ijk} = s_i \cdot w_{ijk}, \tag{1}$$

where w and w' are the original and modulated weights, respectively, s_i is the scale corresponding to the ith input feature map, and j and k enumerate the output feature maps and spatial footprint of the convolution, respectively.

Now, the purpose of instance normalization is to essentially remove the effect of s from the statistics of the convolution's output feature maps. We observe that this goal can be achieved more directly. Let us assume that the input activations are i.i.d. random variables with unit standard deviation. After modulation and convolution, the output activations have standard deviation of

$$\sigma_j = \sqrt{\sum_{i,k} {w'_{ijk}}^2}, \tag{2}$$

i.e., the outputs are scaled by the L_2 norm of the corresponding weights. The subsequent normalization aims to restore the outputs back to unit standard deviation. Based on Equation 2, this is achieved if we scale ("demodulate") each output feature map j by $1/\sigma_j$. Alternatively, we can again bake this into the convolution weights:

$$w''_{ijk} = w'_{ijk} \Big/ \sqrt{\sum_{i,k} {w'_{ijk}}^2 + \epsilon}, \tag{3}$$

where ϵ is a small constant to avoid numerical issues.

We have now baked the entire style block to a single convolution layer whose weights are adjusted based on s using Equations 1 and 3 (Figure 2d). Compared to instance normalization, our demodulation technique is weaker because it is based on statistical assumptions about the signal instead of actual contents of the feature maps. Similar statistical analysis has been extensively used in modern network initializers [14, 19], but we are not aware of it being previously used as a replacement for data-dependent normalization. Our demodulation is also related to weight normalization [37] that performs the same calculation as a part of reparameterizing the weight tensor. Prior work has identified weight normalization as beneficial in the context of GAN training [43].

Our new design removes the characteristic artifacts (Figure 3) while retaining full controllability, as demonstrated in the accompanying video. FID remains largely unaffected (Table 1, rows A, B), but there is a notable shift from precision to recall. We argue that this is generally desirable, since recall can be traded into precision via truncation, whereas

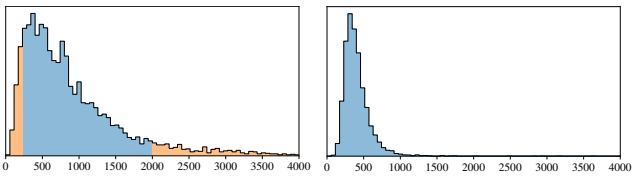
Configuration	FFHQ, 1024×1024				LSUN Car, 512×384			
	FID ↓	Path length ↓	Precision ↑	Recall ↑	FID ↓	Path length ↓	Precision ↑	Recall ↑
A Baseline StyleGAN [24]	4.40	212.1	**0.721**	0.399	3.27	1484.5	**0.701**	0.435
B + Weight demodulation	4.39	175.4	0.702	0.425	3.04	862.4	0.685	0.488
C + Lazy regularization	4.38	158.0	0.719	0.427	2.83	981.6	0.688	0.493
D + Path length regularization	4.34	**122.5**	0.715	0.418	3.43	651.2	0.697	0.452
E + No growing, new G & D arch.	3.31	124.5	0.705	0.449	3.19	471.2	0.690	0.454
F + Large networks (StyleGAN2)	**2.84**	145.0	0.689	**0.492**	**2.32**	**415.5**	0.678	**0.514**
Config A with large networks	3.98	199.2	0.716	0.422	–	–	–	–

Table 1. Main results. For each training run, we selected the training snapshot with the lowest FID. We computed each metric 10 times with different random seeds and report their average. *Path length* corresponds to the PPL metric, computed based on path endpoints in $\mathcal{W}$ [24], without the central crop used by Karras et al. [24]. The FFHQ dataset contains 70k images, and the discriminator saw 25M images during training. For LSUN CAR the numbers were 893k and 57M. ↑ indicates that higher is better, and ↓ that lower is better.

Figure 3. Replacing normalization with demodulation removes the characteristic artifacts from images and activations.

(a) Low PPL scores (b) High PPL scores

Figure 4. Connection between perceptual path length and image quality using baseline StyleGAN (config A) with LSUN CAT. (a) Random examples with low PPL ($\leq 10^{\text{th}}$ percentile). (b) Examples with high PPL ($\geq 90^{\text{th}}$ percentile). There is a clear correlation between PPL scores and semantic consistency of the images.



(a) StyleGAN (config A) (b) StyleGAN2 (config F)

Figure 5. (a) Distribution of PPL scores of individual images generated using baseline StyleGAN (config A) with LSUN CAT (FID = 8.53, PPL = 924). The percentile ranges corresponding to Figure 4 are highlighted in orange. (b) StyleGAN2 (config F) improves the PPL distribution considerably (showing a snapshot with the same FID = 8.53, PPL = 387).

the opposite is not true [27]. In practice our design can be implemented efficiently using grouped convolutions, as detailed in Appendix B. To avoid having to account for the activation function in Equation 3, we scale our activation functions so that they retain the expected signal variance.

3. Image quality and generator smoothness

While GAN metrics such as FID or Precision and Recall (P&R) successfully capture many aspects of the generator, they continue to have somewhat of a blind spot for image quality. For an example, refer to Figures 13 and 14 that contrast generators with identical FID and P&R scores but markedly different overall quality.[2]

We observe a correlation between perceived image quality and perceptual path length (PPL) [24], a metric that was originally introduced for quantifying the smoothness of the mapping from a latent space to the output image by measuring average LPIPS distances [50] between generated images under small perturbations in latent space. Again consulting Figures 13 and 14, a smaller PPL (smoother generator mapping) appears to correlate with higher overall image qual-

[2]We believe that the key to the apparent inconsistency lies in the particular choice of feature space rather than the foundations of FID or P&R. It was recently discovered that classifiers trained using ImageNet [35] tend to base their decisions much more on texture than shape [12], while humans strongly focus on shape [28]. This is relevant in our context because FID and P&R use high-level features from InceptionV3 [39] and VGG-16 [39], respectively, which were trained in this way and are thus expected to be biased towards texture detection. As such, images with, e.g., strong cat textures may appear more similar to each other than a human observer would agree, thus partially compromising density-based metrics (FID) and manifold coverage metrics (P&R).

ity, whereas other metrics are blind to the change. Figure 4 examines this correlation more closely through per-image PPL scores on LSUN CAT, computed by sampling the latent space around $\mathbf{w} \sim f(\mathbf{z})$. Low scores are indeed indicative of high-quality images, and vice versa. Figure 5a shows the corresponding histogram and reveals the long tail of the distribution. The overall PPL for the model is simply the expected value of these per-image PPL scores. We always compute PPL for the entire image, as opposed to Karras et al. [24] who use a smaller central crop.

It is not immediately obvious why a low PPL should correlate with image quality. We hypothesize that during training, as the discriminator penalizes broken images, the most direct way for the generator to improve is to effectively stretch the region of latent space that yields good images. This would lead to the low-quality images being squeezed into small latent space regions of rapid change. While this improves the average output quality in the short term, the accumulating distortions impair the training dynamics and consequently the final image quality.

Clearly, we cannot simply encourage minimal PPL since that would guide the generator toward a degenerate solution with zero recall. Instead, we will describe a new regularizer that aims for a smoother generator mapping without this drawback. As the resulting regularization term is somewhat expensive to compute, we first describe a general optimization that applies to any regularization technique.

3.1. Lazy regularization

Typically the main loss function (e.g., logistic loss [16]) and regularization terms (e.g., R_1 [30]) are written as a single expression and are thus optimized simultaneously. We observe that the regularization terms can be computed less frequently than the main loss function, thus greatly diminishing their computational cost and the overall memory usage. Table 1, row C shows that no harm is caused when R_1 regularization is performed only once every 16 minibatches, and we adopt the same strategy for our new regularizer as well. Appendix B gives implementation details.

3.2. Path length regularization

We would like to encourage that a fixed-size step in $\mathcal{W}$ results in a non-zero, fixed-magnitude change in the image. We can measure the deviation from this ideal empirically by stepping into random directions in the image space and observing the corresponding $\mathbf{w}$ gradients. These gradients should have close to an equal length regardless of $\mathbf{w}$ or the image-space direction, indicating that the mapping from the latent space to image space is well-conditioned [33].

At a single $\mathbf{w} \in \mathcal{W}$, the local metric scaling properties of the generator mapping $g(\mathbf{w}) : \mathcal{W} \mapsto \mathcal{Y}$ are captured by the Jacobian matrix $\mathbf{J}_\mathbf{w} = \partial g(\mathbf{w})/\partial \mathbf{w}$. Motivated by the desire to preserve the expected lengths of vectors regardless of the direction, we formulate our regularizer as

$$\mathbb{E}_{\mathbf{w},\mathbf{y}\sim\mathcal{N}(0,\mathbf{I})} \left(\left\| \mathbf{J}_\mathbf{w}^T \mathbf{y} \right\|_2 - a \right)^2, \tag{4}$$

where $\mathbf{y}$ are random images with normally distributed pixel intensities, and $\mathbf{w} \sim f(\mathbf{z})$, where $\mathbf{z}$ are normally distributed. We show in Appendix C that, in high dimensions, this prior is minimized when $\mathbf{J}_\mathbf{w}$ is orthogonal (up to a global scale) at any $\mathbf{w}$. An orthogonal matrix preserves lengths and introduces no squeezing along any dimension.

To avoid explicit computation of the Jacobian matrix, we use the identity $\mathbf{J}_\mathbf{w}^T \mathbf{y} = \nabla_\mathbf{w}(g(\mathbf{w}) \cdot \mathbf{y})$, which is efficiently computable using standard backpropagation [6]. The constant a is set dynamically during optimization as the long-running exponential moving average of the lengths $\|\mathbf{J}_\mathbf{w}^T \mathbf{y}\|_2$, allowing the optimization to find a suitable global scale by itself.

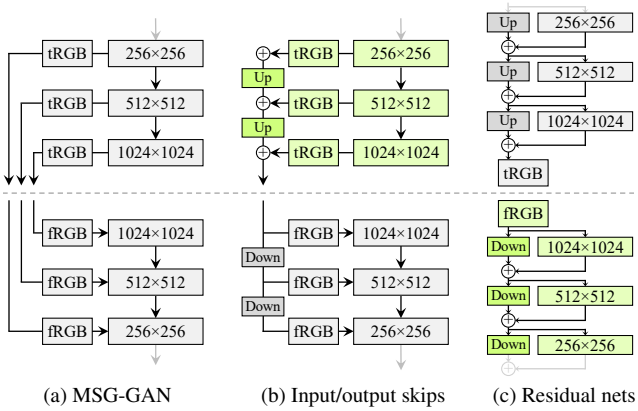
Our regularizer is closely related to the Jacobian clamping regularizer presented by Odena et al. [33]. Practical differences include that we compute the products $\mathbf{J}_\mathbf{w}^T \mathbf{y}$ analytically whereas they use finite differences for estimating $\mathbf{J}_\mathbf{w}\boldsymbol{\delta}$ with $\mathcal{Z} \ni \boldsymbol{\delta} \sim \mathcal{N}(0, \mathbf{I})$. It should be noted that spectral normalization [31] of the generator [46] only constrains the largest singular value, posing no constraints on the others and hence not necessarily leading to better conditioning. We find that enabling spectral normalization in addition to our contributions — or instead of them — invariably compromises FID, as detailed in Appendix E.

In practice, we notice that path length regularization leads to more reliable and consistently behaving models, making architecture exploration easier. We also observe that the smoother generator is significantly easier to invert (Section 5). Figure 5b shows that path length regularization clearly tightens the distribution of per-image PPL scores, without pushing the mode to zero. However, Table 1, row D points toward a tradeoff between FID and PPL in datasets that are less structured than FFHQ.

4. Progressive growing revisited

Progressive growing [23] has been very successful in stabilizing high-resolution image synthesis, but it causes its own characteristic artifacts. The key issue is that the progressively grown generator appears to have a strong location preference for details; the accompanying video shows that when features like teeth or eyes should move smoothly over the image, they may instead remain stuck in place before jumping to the next preferred location. Figure 6 shows a related artifact. We believe the problem is that in progressive growing each resolution serves momentarily as the output resolution, forcing it to generate maximal frequency details, which then leads to the trained network to have excessively high frequencies in the intermediate layers, compromising shift invariance [49]. Appendix A shows an example. These

Figure 6. Progressive growing leads to "phase" artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.



(a) MSG-GAN (b) Input/output skips (c) Residual nets

Figure 7. Three generator (above the dashed line) and discriminator architectures. Up and Down denote bilinear up and downsampling, respectively. In residual networks these also include 1×1 convolutions to adjust the number of feature maps. tRGB and fRGB convert between RGB and high-dimensional per-pixel data. Architectures used in configs E and F are shown in green.

issues prompt us to search for an alternative formulation that would retain the benefits of progressive growing without the drawbacks.

4.1. Alternative network architectures

While StyleGAN uses simple feedforward designs in the generator (synthesis network) and discriminator, there is a vast body of work dedicated to the study of better network architectures. Skip connections [34, 22], residual networks [18, 17, 31], and hierarchical methods [7, 47, 48] have proven highly successful also in the context of generative methods. As such, we decided to re-evaluate the network design of StyleGAN and search for an architecture that produces high-quality images without progressive growing.

Figure 7a shows MSG-GAN [22], which connects the matching resolutions of the generator and discriminator using multiple skip connections. The MSG-GAN generator is modified to output a mipmap [42] instead of an image, and a similar representation is computed for each real image as well. In Figure 7b we simplify this design by upsampling and summing the contributions of RGB outputs corresponding to different resolutions. In the discriminator, we similarly provide the downsampled image to each resolution block of the discriminator. We use bilinear filtering in all up and downsampling operations. In Figure 7c we further modify the design to use residual connections.[3] This design is similar to LAPGAN [7] without the per-resolution discriminators employed by Denton et al.

FFHQ	D original		D input skips		D residual	
	FID	PPL	FID	PPL	FID	PPL
G original	4.32	265	4.18	235	3.58	269
G output skips	4.33	169	3.77	127	**3.31**	**125**
G residual	4.35	203	3.96	229	3.79	243

LSUN Car	D original		D input skips		D residual	
	FID	PPL	FID	PPL	FID	PPL
G original	3.75	905	3.23	758	3.25	802
G output skips	3.77	544	3.86	**316**	3.19	471
G residual	3.93	981	3.40	667	**2.66**	645

Table 2. Comparison of generator and discriminator architectures without progressive growing. The combination of generator with output skips and residual discriminator corresponds to configuration E in the main result table.

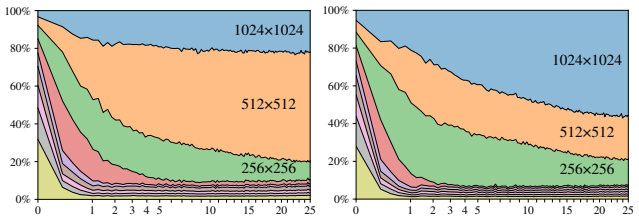
Table 2 compares three generator and three discriminator architectures: original feedforward networks as used in StyleGAN, skip connections, and residual networks, all trained without progressive growing. FID and PPL are provided for each of the 9 combinations. We can see two broad trends: skip connections in the generator drastically improve PPL in all configurations, and a residual discriminator network is clearly beneficial for FID. The latter is perhaps not surprising since the structure of discriminator resembles classifiers where residual architectures are known to be helpful. However, a residual architecture was harmful in the generator — the lone exception was FID in LSUN CAR when both networks were residual.

For the rest of the paper we use a skip generator and a residual discriminator, without progressive growing. This corresponds to configuration E in Table 1, and it significantly improves FID and PPL.

4.2. Resolution usage

The key aspect of progressive growing, which we would like to preserve, is that the generator will initially focus on low-resolution features and then slowly shift its attention to finer details. The architectures in Figure 7 make it possible for the generator to first output low resolution images that are not affected by the higher-resolution layers in a significant way, and later shift the focus to the higher-resolution

[3] In residual network architectures, the addition of two paths leads to a doubling of signal variance, which we cancel by multiplying with $1/\sqrt{2}$. This is crucial for our networks, whereas in classification resnets [18] the issue is typically hidden by batch normalization.



(a) StyleGAN-sized (config E) (b) Large networks (config F)

Figure 8. Contribution of each resolution to the output of the generator as a function of training time. The vertical axis shows a breakdown of the relative standard deviations of different resolutions, and the horizontal axis corresponds to training progress, measured in millions of training images shown to the discriminator. We can see that in the beginning the network focuses on low-resolution images and progressively shifts its focus on larger resolutions as training progresses. In (a) the generator basically outputs a 512^2 image with some minor sharpening for 1024^2, while in (b) the larger network focuses more on the high-resolution details.

layers as the training proceeds. Since this is not enforced in any way, the generator will do it only if it is beneficial. To analyze the behavior in practice, we need to quantify how strongly the generator relies on particular resolutions over the course of training.

Since the skip generator (Figure 7b) forms the image by explicitly summing RGB values from multiple resolutions, we can estimate the relative importance of the corresponding layers by measuring how much they contribute to the final image. In Figure 8a, we plot the standard deviation of the pixel values produced by each tRGB layer as a function of training time. We calculate the standard deviations over 1024 random samples of $\mathbf{w}$ and normalize the values so that they sum to 100%.

At the start of training, we can see that the new skip generator behaves similar to progressive growing — now achieved without changing the network topology. It would thus be reasonable to expect the highest resolution to dominate towards the end of the training. The plot, however, shows that this fails to happen in practice, which indicates that the generator may not be able to "fully utilize" the target resolution. To verify this, we inspected the generated images manually and noticed that they generally lack some of the pixel-level detail that is present in the training data — the images could be described as being sharpened versions of 512^2 images instead of true 1024^2 images.

This leads us to hypothesize that there is a capacity problem in our networks, which we test by doubling the number of feature maps in the highest-resolution layers of both networks.[4] This brings the behavior more in line with expectations: Figure 8b shows a significant increase in the contribution of the highest-resolution layers, and Table 1, row F shows that FID and Recall improve markedly. The last row shows that baseline StyleGAN also benefits from additional capacity, but its quality remains far below StyleGAN2.

[4] We double the number of feature maps in resolutions 64^2–1024^2 while keeping other parts of the networks unchanged. This increases the total number of trainable parameters in the generator by 22% (25M → 30M) and in the discriminator by 21% (24M → 29M).

Dataset	Resolution	StyleGAN (A)		StyleGAN2 (F)	
		FID	PPL	FID	PPL
LSUN CAR	512×384	3.27	1485	**2.32**	**416**
LSUN CAT	256×256	8.53	924	**6.93**	**439**
LSUN CHURCH	256×256	4.21	742	**3.86**	**342**
LSUN HORSE	256×256	3.83	1405	**3.43**	**338**

Table 3. Improvement in LSUN datasets measured using FID and PPL. We trained CAR for 57M images, CAT for 88M, CHURCH for 48M, and HORSE for 100M images.

Table 3 compares StyleGAN and StyleGAN2 in four LSUN categories, again showing clear improvements in FID and significant advances in PPL. It is possible that further increases in the size could provide additional benefits.

5. Projection of images to latent space

Inverting the synthesis network g is an interesting problem that has many applications. Manipulating a given image in the latent feature space requires finding a matching latent code $\mathbf{w}$ for it first. Previous research [1, 10] suggests that instead of finding a common latent code $\mathbf{w}$, the results improve if a separate $\mathbf{w}$ is chosen for each layer of the generator. The same approach was used in an early encoder implementation [32]. While extending the latent space in this fashion finds a closer match to a given image, it also enables projecting arbitrary images that should have no latent representation. Instead, we concentrate on finding latent codes in the original, unextended latent space, as these correspond to images that the generator could have produced.

Our projection method differs from previous methods in two ways. First, we add ramped-down noise to the latent code during optimization in order to explore the latent space more comprehensively. Second, we also optimize the stochastic noise inputs of the StyleGAN generator, regularizing them to ensure they do not end up carrying coherent signal. The regularization is based on enforcing the autocorrelation coefficients of the noise maps to match those of unit Gaussian noise over multiple scales. Details of our projection method can be found in Appendix D.

5.1. Attribution of generated images

Detection of manipulated or generated images is a very important task. At present, classifier-based methods can quite reliably detect generated images, regardless of their exact origin [29, 45, 40, 51, 41]. However, given the rapid pace of progress in generative methods, this may not be a lasting situation. Besides general detection of fake images, we may also consider a more limited form of the problem:

StyleGAN — generated images | StyleGAN2 — generated images | StyleGAN2 — real images

Figure 9. Example images and their projected and re-synthesized counterparts. For each configuration, top row shows the target images and bottom row shows the synthesis of the corresponding projected latent vector and noise inputs. With the baseline StyleGAN, projection often finds a reasonably close match for generated images, but especially the backgrounds differ from the originals. The images generated using StyleGAN2 can be projected almost perfectly back into generator inputs, while projected real images (from the training set) show clear differences to the originals, as expected. All tests were done using the same projection method and hyperparameters.

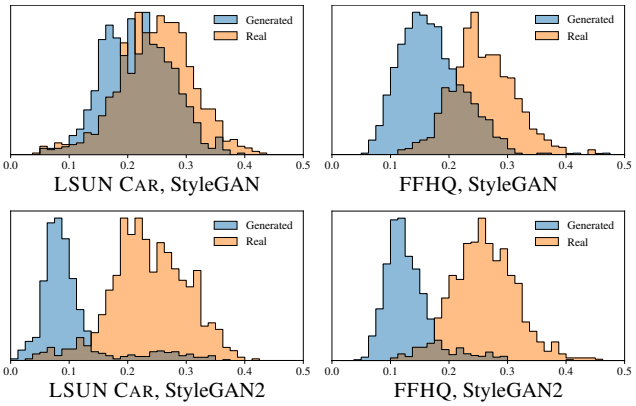


Figure 10. LPIPS distance histograms between original and projected images for generated (blue) and real images (orange). Despite the higher image quality of our improved generator, it is much easier to project the generated images into its latent space $\mathcal{W}$. The same projection method was used in all cases.

being able to attribute a fake image to its specific source [2]. With StyleGAN, this amounts to checking if there exists a $\mathbf{w} \in \mathcal{W}$ that re-synthesis the image in question.

We measure how well the projection succeeds by computing the LPIPS [50] distance between original and re-synthesized image as $D_{\text{LPIPS}}[\boldsymbol{x}, g(\tilde{g}^{-1}(\boldsymbol{x}))]$, where $\boldsymbol{x}$ is the image being analyzed and $\tilde{g}^{-1}$ denotes the approximate projection operation. Figure 10 shows histograms of these distances for LSUN CAR and FFHQ datasets using the original StyleGAN and StyleGAN2, and Figure 9 shows example projections. The images generated using StyleGAN2 can be projected into $\mathcal{W}$ so well that they can be almost unambiguously attributed to the generating network. However, with the original StyleGAN, even though it should technically be possible to find a matching latent code, it appears that the mapping from $\mathcal{W}$ to images is too complex for this to succeed reliably in practice. We find it encouraging that StyleGAN2 makes source attribution easier even though the image quality has improved significantly.

6. Conclusions and future work

We have identified and fixed several image quality issues in StyleGAN, improving the quality further and considerably advancing the state of the art in several datasets. In some cases the improvements are more clearly seen in motion, as demonstrated in the accompanying video. Appendix A includes further examples of results obtainable using our method. Despite the improved quality, StyleGAN2 makes it easier to attribute a generated image to its source.

Training performance has also improved. At 1024^2 resolution, the original StyleGAN (config A in Table 1) trains at 37 images per second on NVIDIA DGX-1 with 8 Tesla V100 GPUs, while our config E trains 40% faster at 61 img/s. Most of the speedup comes from simplified dataflow due to weight demodulation, lazy regularization, and code optimizations. StyleGAN2 (config F, larger networks) trains at 31 img/s, and is thus only slightly more expensive to train than original StyleGAN. Its total training time was 9 days for FFHQ and 13 days for LSUN CAR.

The entire project, including all exploration, consumed 132 MWh of electricity, of which 0.68 MWh went into training the final FFHQ model. In total, we used about 51 single-GPU years of computation (Volta class GPU). A more detailed discussion is available in Appendix F.

In the future, it could be fruitful to study further improvements to the path length regularization, e.g., by replacing the pixel-space L_2 distance with a data-driven feature-space metric. Considering the practical deployment of GANs, we feel that it will be important to find new ways to reduce the training data requirements. This is especially crucial in applications where it is infeasible to acquire tens of thousands of training samples, and with datasets that include a lot of intrinsic variation.

Acknowledgements We thank Ming-Yu Liu for an early review, Timo Viitanen for help with the public release, David Luebke for in-depth discussions and helpful comments, and Tero Kuosmanen for technical support with the compute infrastructure.

References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2StyleGAN: How to embed images into the StyleGAN latent space? In *ICCV*, 2019. 7

[2] Michael Albright and Scott McCloskey. Source generator attribution via inversion. In *CVPR Workshops*, 2019. 8

[3] Carl Bergstrom and Jevin West. Which face is real? http://www.whichfaceisreal.com/learn.html, Accessed November 15, 2019. 1

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 17

[5] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018. 1

[6] Yann N. Dauphin, Harm de Vries, and Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015. 5

[7] Emily L. Denton, Soumith Chintala, Arthur Szlam, and Robert Fergus. Deep generative image models using a Laplacian pyramid of adversarial networks. *CoRR*, abs/1506.05751, 2015. 6

[8] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations. 1

[9] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *CoRR*, abs/1610.07629, 2016. 1

[10] Aviv Gabbay and Yedid Hoshen. Style generator inversion for image enhancement and animation. *CoRR*, abs/1906.11880, 2019. 7

[11] 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. 21

[12] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *CoRR*, abs/1811.12231, 2018. 1, 4

[13] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *CoRR*, abs/1705.06830, 2017. 1

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. 3

[15] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013. 17

[16] 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, 11

[17] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of Wasserstein GANs. *CoRR*, abs/1704.00028, 2017. 6

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 6

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *CoRR*, abs/1502.01852, 2015. 3

[20] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Proc. NIPS*, pages 6626–6637, 2017. 1

[21] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017. 1

[22] Animesh Karnewar and Oliver Wang. MSG-GAN: multi-scale gradients for generative adversarial networks. In *Proc. CVPR*, 2020. 6

[23] 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, 5, 11

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proc. CVPR*, 2018. 1, 2, 4, 5, 11, 13, 16, 20

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 11, 19

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105. 2012. 11

[27] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Proc. NeurIPS*, 2019. 1, 2, 4

[28] Barbara Landau, Linda B. Smith, and Susan S. Jones. The importance of shape in early lexical learning. *Cognitive Development*, 3(3), 1988. 4

[29] Haodong Li, Han Chen, Bin Li, and Shunquan Tan. Can forensic detectors identify GAN generated images? In *Proc. Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2018. 7

[30] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? *CoRR*, abs/1801.04406, 2018. 5, 11

[31] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018. 1, 5, 6, 20

[32] Dmitry Nikitko. StyleGAN – Encoder for official TensorFlow implementation. https://github.com/Puzer/stylegan-encoder/, 2019. 7

[33] 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. 5, 18

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015. 6

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet large scale visual recognition challenge. In *Proc. CVPR*, 2015. 4

[36] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. *CoRR*, abs/1806.00035, 2018. 1

[37] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016. 3

[38] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of GANs for semantic face editing. *CoRR*, abs/1907.10786, 2019. 1

[39] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 4

[40] Run Wang, Lei Ma, Felix Juefei-Xu, Xiaofei Xie, Jian Wang, and Yang Liu. FakeSpotter: A simple baseline for spotting AI-synthesized fake faces. *CoRR*, abs/1909.06122, 2019. 7

[41] Sheng-Yu Wang, Oliver Wang, Richard Zhang, Andrew Owens, and Alexei A. Efros. CNN-generated images are surprisingly easy to spot... for now. *CoRR*, abs/1912.11035, 2019. 7

[42] Lance Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, 1983. 6

[43] Sitao Xiang and Hao Li. On the effects of batch and weight normalization in generative adversarial networks. *CoRR*, abs/1704.03971, 2017. 3

[44] 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. 11

[45] Ning Yu, Larry Davis, and Mario Fritz. Attributing fake images to GANs: Analyzing fingerprints in generated images. *CoRR*, abs/1811.08180, 2018. 7

[46] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *CoRR*, abs/1805.08318, 2018. 5

[47] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris N. Metaxas. Stack-GAN: text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017. 6

[48] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N. Metaxas. Stack-GAN++: realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1710.10916, 2017. 6

[49] Richard Zhang. Making convolutional networks shift-invariant again. In *Proc. ICML*, 2019. 5, 11

[50] 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. 4, 8, 19

[51] Xu Zhang, Svebor Karaman, and Shih-Fu Chang. Detecting and simulating artifacts in GAN fake images. *CoRR*, abs/1907.06515, 2019. 7

A. Image quality

We include several large images that illustrate various aspects related to image quality. Figure 11 shows hand-picked examples illustrating the quality and diversity achievable using our method in FFHQ, while Figure 12 shows uncurated results for all datasets mentioned in the paper.

Figures 13 and 14 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 15 shows a rare case where the blob artifact fails to appear in StyleGAN activations, leading to a seriously broken image. Figure 16 visualizes the activations inside Table 1 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[5] corresponding to configuration A in Table 1. 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 [23], leaky ReLU activation with $\alpha = 0.2$, bilinear filtering [49] in all up/downsampling layers [24], minibatch standard deviation layer at the end of the discriminator [23], exponential moving average of generator weights [23], style mixing regularization [24], nonsaturating logistic loss [16] with R_1 regularization [30], Adam optimizer [25] with the same hyperparameters ($\beta_1 = 0, \beta_2 = 0.99, \epsilon = 10^{-8}, \text{minibatch} = 32$), and training datasets [24, 44]. 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, 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. [24], 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) 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.

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

Weight demodulation Considering the practical implementation of Equations 1 and 3, 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* [26] 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 by temporarily reshaping the weights and activations so that each convolution sees one sample with N groups — instead of N 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) 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). 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

$$\gamma_{\text{pl}} = \frac{\ln 2}{r^2(\ln r - \ln 2)}, \tag{5}$$

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.

Figure 11. Four hand-picked examples illustrating the image quality and diversity achievable using StylegGAN2 (config F).

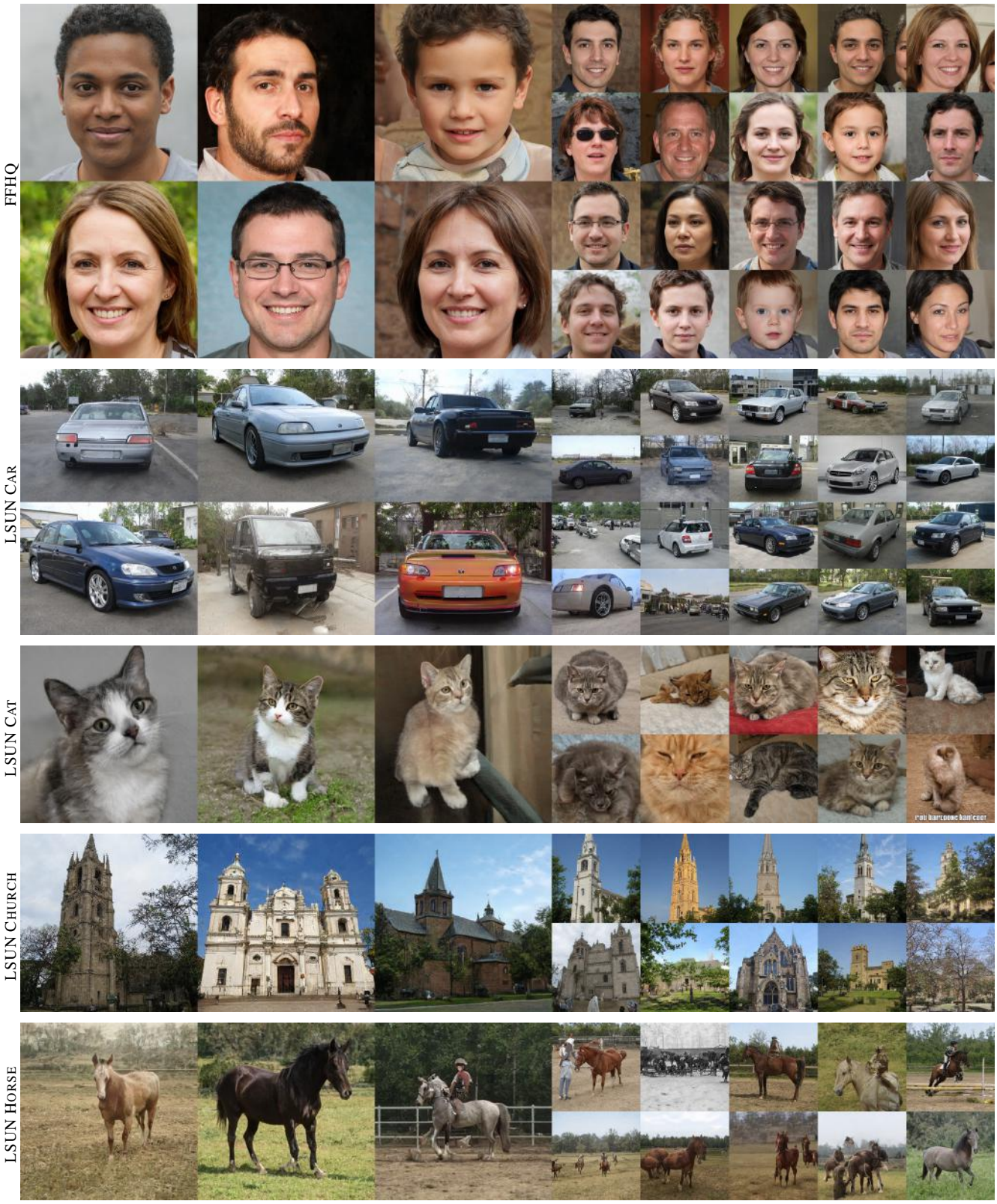
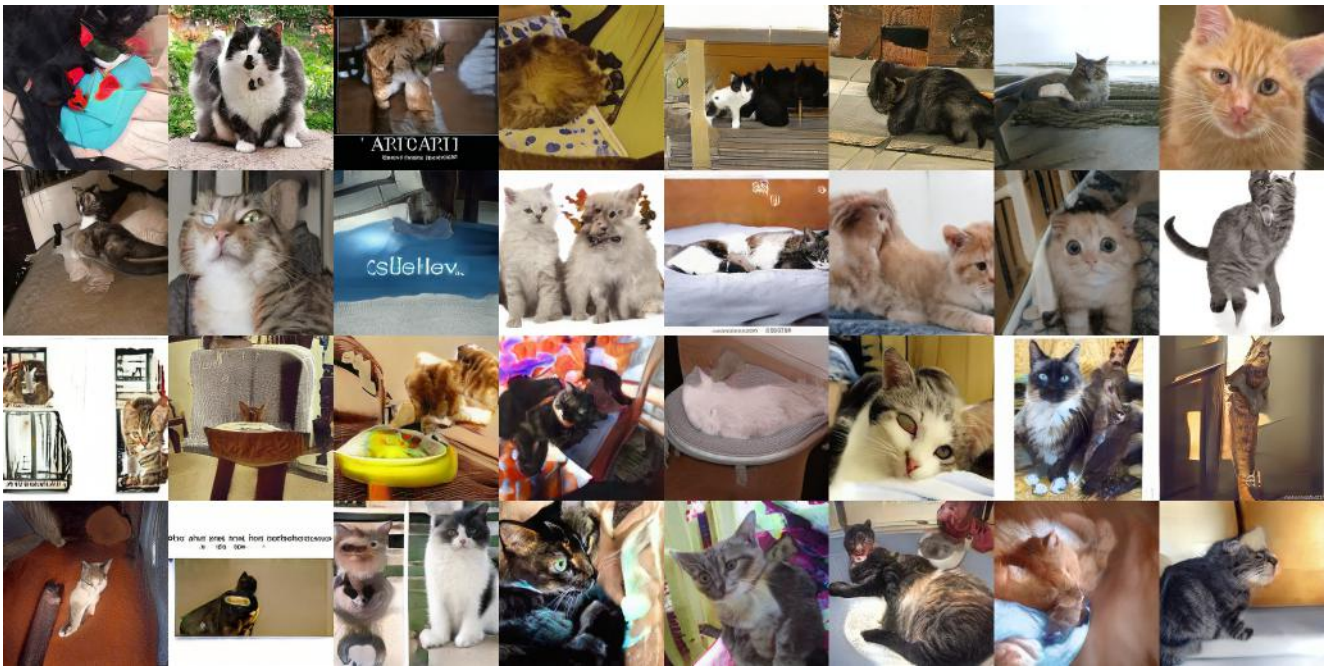
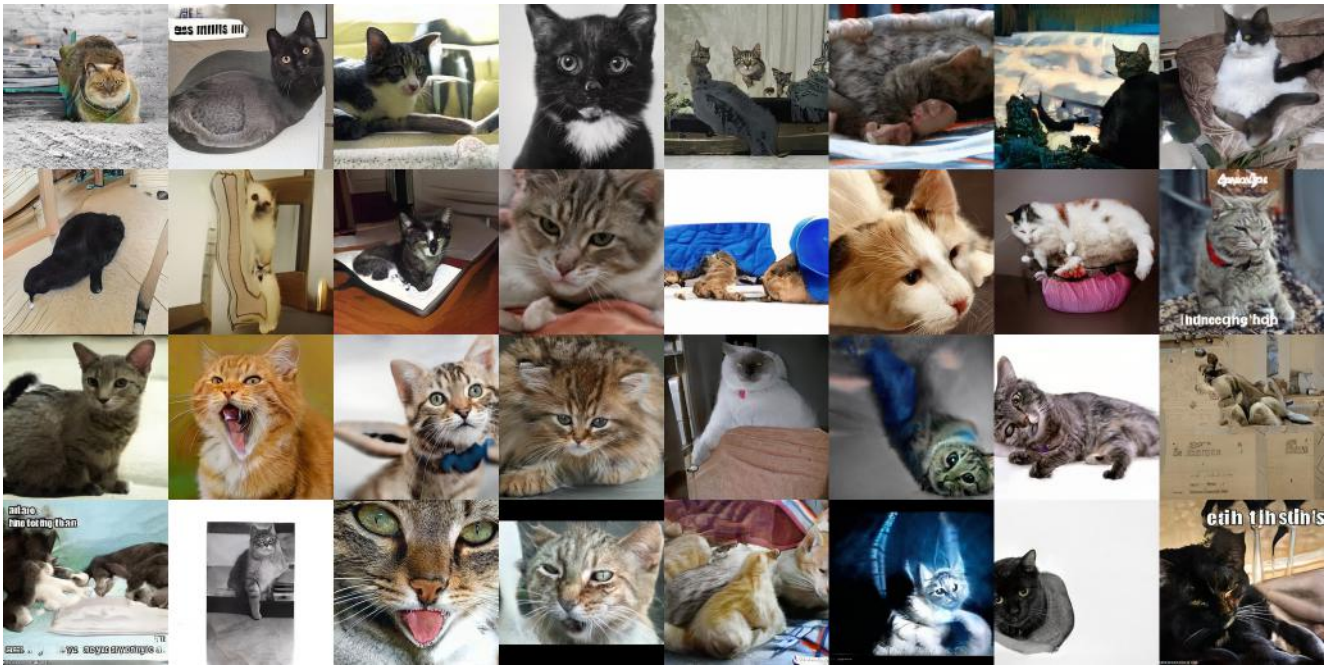


Figure 12. Uncurated results for each dataset used in Tables 1 and 3. The images correspond to random outputs produced by our generator (config F), with truncation applied at all resolutions using $\psi = 0.5$ [24].

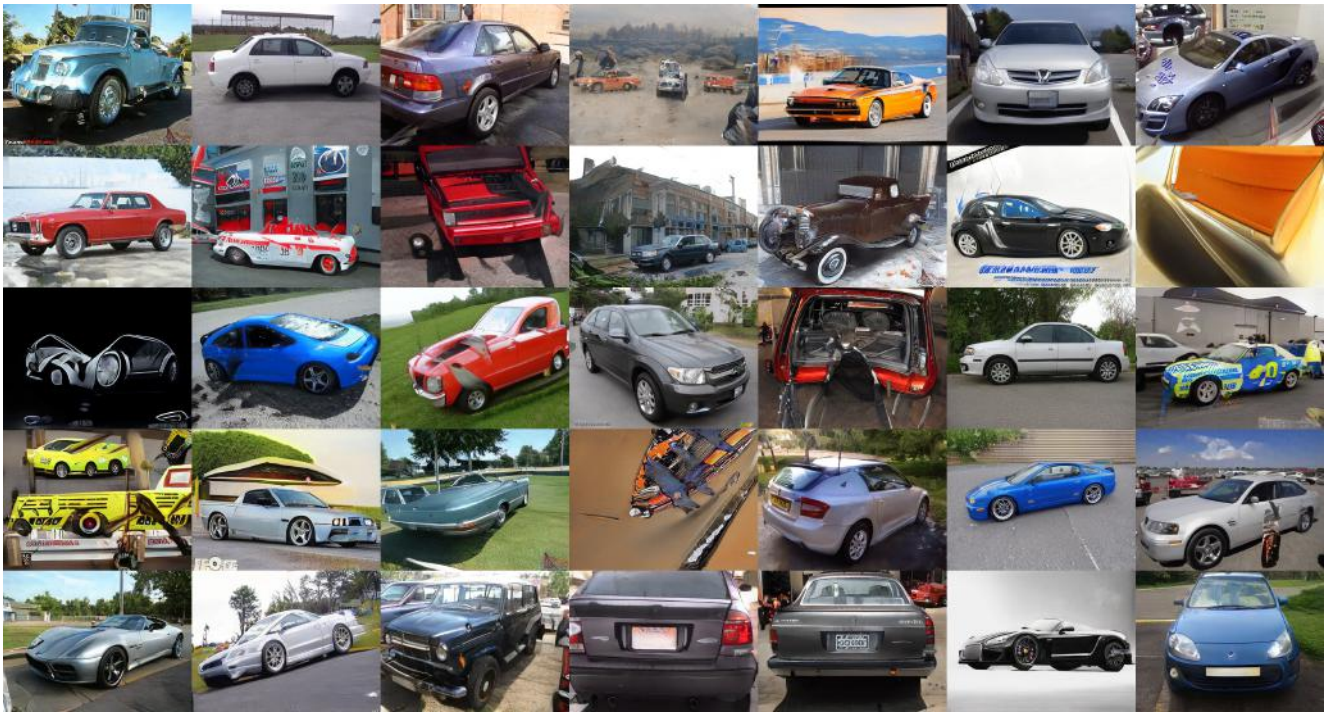


Model 1: FID = 8.53, P = 0.64, R = 0.28, PPL = 924



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

Figure 13. 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, 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 14. 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, and model 2 is an early training snapshot of configuration F.

Figure 15. 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 16. 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.

Progressive growing In configurations A–D we use progressive growing with the same parameters as Karras et al. [24] (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.

Dataset-specific tuning Similar to Karras et al. [24], 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, as well as LSUN CHURCH and LSUN HORSE in Table 3, 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 is of the form:

$$\mathcal{L}_{\text{pl}} = \mathbb{E}_{\mathbf{w}}\mathbb{E}_{\mathbf{y}}\left(\left\|\mathbf{J}_{\mathbf{w}}^T\mathbf{y}\right\|_2 - a\right)^2, \tag{6}$$

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 $\mathbf{0} \in \mathbb{R}^{L\times(M-L)}$ [15]. 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 $\tilde{\mathbf{y}}$:

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

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}:

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

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 $\sigma = 1/\sqrt{2}$ [4]. 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

$$\mathcal{L}_{\mathbf{w}} \approx (2\pi e/L)^{-L/2} \int_{\mathbb{S}} \int_{0}^{\infty} \left(r \left\|\mathbf{\Sigma}\phi\right\|_2 - a\right)^2 \exp\left(-\frac{\left(r-\sqrt{L}\right)^2}{2\sigma^2}\right) \mathrm{d}r \, \mathrm{d}\phi, \quad (7)$$

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

To minimize this loss, we set $\mathbf{\Sigma}$ such that the function $\left(r \left\|\mathbf{\Sigma}\phi\right\|_2 - a\right)^2$ 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. 7 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. 6) 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. [33] who consider a related conditioning prior.

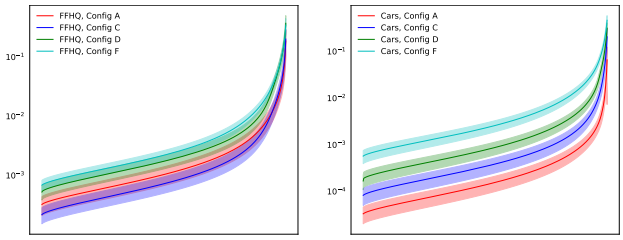


Figure 17. The mean and standard deviation of the magnitudes of sorted singular values of the Jacobian matrix evaluated at random latent space points $\mathbf{w}$, with largest eigenvalue normalized to 1. In both datasets, path length regularization (Config D) and novel architecture (Config F) exhibit 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).

Figure 17 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).

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 17 shows that the mapping 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

$$L = \int_{t_0}^{t_1} |\tilde{u}'(t)| \, \mathrm{d}t, \quad (8)$$

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

$$L = \int_{t_0}^{t_1} |J_g(u(t))u'(t)| \, \mathrm{d}t, \quad (9)$$

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:

$$L = \int_{t_0}^{t_1} |u'(t)| \, \mathrm{d}t. \tag{10}$$

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 2).

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 [25] 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 map regularization terms in our loss function, in addition to an image quality term. The image quality term is the LPIPS [50] distance between target image $\boldsymbol{x}$ and the synthesized image: $L_{image} = D_{\mathrm{LPIPS}}[\boldsymbol{x}, g(\tilde{\mathbf{w}}, \boldsymbol{n}_0, \boldsymbol{n}_1, \ldots)]$. 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

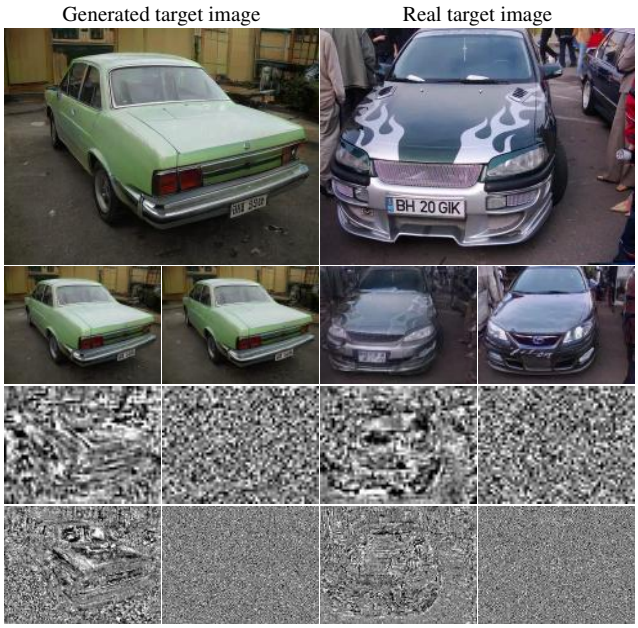


Figure 18. Effect of noise regularization in latent-space projection where we also optimize the contents of the noise inputs of the synthesis network. Top to bottom: target image, re-synthesized image, contents of two noise maps at different resolutions. When regularization is turned off in this test, 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.

	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 4. Effect of spectral normalization with FFHQ at 1024^2. The first row corresponds to StyleGAN2, i.e., config F in Table 1. 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.

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

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

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 18 illustrates the effect of noise regularization on the resulting noise maps.

E. Results with spectral normalization

Since spectral normalization (SN) is widely used in GANs [31], we investigated its effect on StyleGAN2. Table 4 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.

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 5. 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 [24] 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.

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 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 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 5 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 [11] 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.