PROGRESSIVE GROWING OF GANs FOR IMPROVED QUALITY, STABILITY, AND VARIATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We describe a new training methodology for generative adversarial networks. The key idea is to grow both the generator and discriminator progressively: starting from a low resolution, we add new layers that model increasingly fine details as training progresses. This both speeds the training up and greatly stabilizes it, allowing us to produce images of unprecedented quality, e.g., CELEBA images at $1024^2$. We also propose a simple way to increase the variation in generated images, and achieve a record inception score of 8.80 in unsupervised CIFAR10. Additionally, we describe several implementation details that are important for discouraging unhealthy competition between the generator and discriminator. Finally, we suggest a new metric for evaluating GAN results, both in terms of image quality and variation. As an additional contribution, we construct a higher-quality version of the CELEBA dataset.

1 INTRODUCTION

Generative methods that produce novel samples from high-dimensional data distributions, such as images, are finding widespread use, for example in speech synthesis (van den Oord et al., 2016a), image-to-image translation (Zhu et al., 2017; Liu et al., 2017), and image inpainting (Iizuka et al., 2017). Currently the most prominent approaches are autoregressive models (van den Oord et al., 2016b;c), variational autoencoders (VAE) (Kingma & Welling, 2014), and generative adversarial networks (GAN) (Goodfellow et al., 2014). Currently they all have significant strengths and weaknesses. Autoregressive models – such as PixelCNN – produce sharp images but are slow to evaluate and do not have a latent representation as they directly model the conditional distribution over pixels, potentially limiting their applicability. VAEs are easy to train but tend to produce blurry results due to restrictions in the model, although recent work is improving this (Kingma et al., 2016). GANs produce sharp images, albeit only in fairly small resolutions and with somewhat limited variation, and the training continues to be unstable despite recent progress (Salimans et al., 2016; Gulrajani et al., 2017; Berthelot et al., 2017; Kodali et al., 2017). Hybrid methods combine various strengths of the three, but so far lag behind GANs in image quality (Makhzani & Frey, 2017; Ulyanov et al., 2017; Dumoulin et al., 2016).

Typically, a GAN consists of two networks: generator and discriminator (aka critic). The generator produces a sample, e.g., an image, from a latent code, and the distribution of these images should ideally be indistinguishable from the training distribution. Since it is generally infeasible to engineer a function that tells whether that is the case, a discriminator network is trained to do the assessment, and since networks are differentiable, we also get a gradient we can use to steer both networks to the right direction. Typically, the generator is of main interest – the discriminator is an adaptive loss function that gets discarded once the generator has been trained.

There are multiple potential problems with this formulation. When we measure the distance between the training distribution and the generated distribution, the gradients can point to more or less random directions if the distributions do not have substantial overlap, i.e., are too easy to tell apart (Arjovsky & Bottou, 2017). Originally, Jensen-Shannon divergence was used as a distance metric (Goodfellow et al., 2014), and recently that formulation has been improved (Hjelm et al., 2017) and a number of more stable alternatives have been proposed, including least squares (Mao et al., 2016b), absolute deviation with margin (Zhao et al., 2017), and Wasserstein distance (Arjovsky et al., 2017; Gulrajani et al., 2017). Our contributions are largely orthogonal to this ongoing discussion, and we primarily use the improved Wasserstein loss, but also experiment with least-squares loss.
The generation of high-resolution images is difficult because higher resolution makes it easier to tell the generated images apart from training images (Odena et al., 2017), thus drastically amplifying the gradient problem. Large resolutions also necessitate using smaller minibatches due to memory constraints, further compromising training stability. Our key insight is that we can grow both the generator and discriminator progressively, starting from easier low-resolution images, and add new layers that introduce higher-resolution details as the training progresses. This greatly speeds up training and improves stability in high resolutions, as we will discuss in Section 2.

The GAN formulation does not explicitly require the entire training data distribution to be represented by the resulting generative model. The conventional wisdom has been that there is a tradeoff between image quality and variation, but that view has been recently challenged (Odena et al., 2017). The degree of preserved variation is currently receiving attention and various methods have been suggested for measuring it, including inception score (Salimans et al., 2016), multi-scale structural similarity (MS-SSIM) (Odena et al., 2017; Wang et al., 2003), birthday paradox (Arora & Zhang, 2017), and explicit tests for the number of discrete modes discovered (Metz et al., 2016). We will describe our method for encouraging variation in Section 3, and propose a new metric for evaluating the quality and variation in Section 5.

Section 4.1 discusses a subtle modification to the initialization of networks, leading to a more balanced learning speed for different layers. Furthermore, we observe that mode collapses traditionally plaguing GANs tend to happen very quickly, over the course of a dozen minibatches. Commonly they start when the discriminator overshoots, leading to exaggerated gradients, and an unhealthy competition follows where the signal magnitudes escalate in both networks. We propose a mechanism to stop the generator from participating in such escalation, overcoming the issue (Section 4.2).

We evaluate our contributions using the CELEBA, LSUN, CIFAR10 datasets. We improve the best published inception score for CIFAR10. Since the datasets commonly used in benchmarking generative methods are limited to a fairly low resolution, we have also created a higher quality version of the CELEBA dataset that allows experimentation with output resolutions up to $1024 \times 1024$ pixels. We are making an effort to publish this dataset. Our full implementation is available at https://github.com/tkarras/progressive_growing_of_gans, trained networks can be found from https://drive.google.com/open?id=0B4qLcYyJmizOZHFULTdYc051X0U along with result images, and a supplementary video illustrating the datasets, additional results, and latent space interpolations is at https://youtu.be/X0xxPcy5Gr4.

## 2 Progressive Growing of GANs

Our primary contribution is a training methodology for GANs where we start with low-resolution images, and then progressively increase the resolution by adding layers to the networks as visualized in Figure 1. This incremental nature allows the training to first discover large-scale structure of the image distribution and then shift attention to increasingly finer scale detail, instead of having to learn all scales simultaneously.

We use generator and discriminator networks that are mirror images of each other and always grow in synchrony. All existing layers in both networks remain trainable throughout the training process. When new layers are added to the networks, we fade them in smoothly, as illustrated in Figure 2. This avoids sudden shocks to the already well-trained, smaller-resolution layers. Appendix A describes structure of the generator and discriminator in detail, along with other training parameters.

We observe that the progressive training has several benefits. Early on, the generation of smaller images is substantially more stable because there is less class information and fewer modes (Odena et al., 2017). By increasing the resolution little by little we are continuously asking a much simpler question compared to the end goal of discovering a mapping from latent vectors to e.g. $1024^2$ images. This approach has conceptual similarly to recent work by Chen & Koltun (2017). In practice it stabilizes the training sufficiently for us to reliably synthesize megapixel-scale images using WGAN-GP loss (Gulrajani et al., 2017) and even LSGAN loss (Mao et al., 2016b).

Another benefit is the reduced training time. With progressively growing GANs most of the iterations are done at lower resolutions, and comparable result quality is often obtained up to 2–6 times faster, depending on the final output resolution.
Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of $4 \times 4$ pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. On the right we show six example images generated using progressive growing at $1024 \times 1024$.

Figure 2: When doubling the resolution of the generator (G) and discriminator (D) we fade in the new layers smoothly. This example illustrates the transition from $16 \times 16$ images (a) to $32 \times 32$ images (c). During the transition (b) we treat the layers that operate on the higher resolution like a residual block, whose weight $\alpha$ increases linearly from 0 to 1. Here $2 \times$ and $0.5 \times$ refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The `toRGB` represents a layer that projects feature vectors to RGB colors and `fromRGB` does the reverse; both use $1 \times 1$ convolutions. When training the discriminator, we feed in real images that are downsampled to match the current resolution of the network. During a resolution transition, we interpolate between two resolutions of the real images, similarly to how the generator output combines two resolutions.

The idea of growing GANs progressively is related to curriculum GANs (Anonymous), where the idea is to attach multiple discriminators that operate on different spatial resolutions to a single generator, and furthermore adjust the balance between resolutions as a function of training time. That work in turn is motivated by Durugkar et al. (2016) who use one generator and multiple discriminators concurrently, and Ghosh et al. (2017) who do the opposite with multiple generators and one discriminator. In contrast to early work on adaptively growing networks, e.g., growing neural gas (Fritzke, 1995) and neuro evolution of augmenting topologies (Stanley & Miikkulainen, 2002) that grow networks greedily, we simply defer the introduction of pre-configured layers. In that sense our approach resembles layer-wise training of autoencoders (Bengio et al., 2007).
3 INCREASING VARIATION USING MINIBATCH STANDARD DEVIATION

GANs have a tendency to capture only a subset of the variation found in training data, and Salimans et al. (2016) suggest “minibatch discrimination” as solution. They compute feature statistics not only from individual images but also across the minibatch, thus encouraging the minibatches of generated and training images to show similar statistics. This is implemented by adding a minibatch layer towards the end of the discriminator, where the layer learns a large tensor that projects the input activation to an array of statistics. A separate set of statistics is produced for each example in a minibatch and it is concatenated to the layer’s output, so that the discriminator can use the statistics internally. We simplify this approach drastically while also improving the variation.

Our simplified solution has neither learnable parameters nor new hyperparameters. We first compute the standard deviation for each feature in each spatial location over the minibatch. We then average these estimates over all features and spatial locations to arrive at a single value. We replicate the value and concatenate it to all spatial locations and over the minibatch, yielding one additional (constant) feature map. This layer could be inserted anywhere in the network, but we have found it best to insert it towards the end (see Appendix A.1 for details). We experimented with a richer set of statistics, but were not able to improve the variation further.

Alternative solutions to the variation problem include unrolling the discriminator (Metz et al., 2016) to regularize its updates, and a “repelling regularizer” (Zhao et al., 2017) that adds a new loss term to the generator, trying to encourage it to orthogonalize the feature vectors in a minibatch. The multiple generators of Ghosh et al. (2017) also serve a similar goal. We acknowledge that these solutions may increase the variation even more than our solution – or possibly be orthogonal to it – but leave a detailed comparison to a later time.

4 NORMALIZATION IN GENERATOR AND DISCRIMINATOR

GANs are prone to the escalation of signal magnitudes as a result of unhealthy competition between the two networks. Most if not all earlier solutions discourage this by using a variant of batch normalization (Ioffe & Szegedy, 2015; Salimans & Kingma, 2016; Ba et al., 2016) in the generator, and often also in the discriminator. These normalization methods were originally introduced to eliminate covariate shift. However, we have not observed that to be an issue in GANs, and thus believe that the actual need in GANs is constraining signal magnitudes and competition. We use a different approach that consists of two ingredients, neither of which include learnable parameters.

4.1 EQUALIZED LEARNING RATE

We deviate from the current trend of careful weight initialization, and instead use a trivial \( \mathcal{N}(0, 1) \) initialization and then explicitly scale the weights at runtime. To be precise, we set \( \hat{w}_i = w_i / c \), where \( w_i \) are the weights and \( c \) is the per-layer normalization constant from He’s initializer (He et al., 2015). The benefit of doing this dynamically instead of during initialization is somewhat subtle, and relates to the scale-invariance in commonly used adaptive stochastic gradient descent methods such as RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2015). These methods normalize a gradient update by its estimated standard deviation, thus making the update independent of the scale of the parameter. As a result, if some parameters have a larger dynamic range than others, they will take longer to adjust. This is a scenario modern initializers cause, and thus it is possible that a learning rate is both too large and too small at the same time. Our approach ensures that the dynamic range, and thus the learning speed, is the same for all weights.

4.2 PIXELWISE FEATURE VECTOR NORMALIZATION IN GENERATOR

To disallow the scenario where the magnitudes in the generator and discriminator spiral out of control as a result of competition, we normalize the feature vector in each pixel to unit length in the generator after each convolutional layer. We do this using a variant of “local response normalization” (Krizhevsky et al., 2012), configured as \( b_{x,y} = a_{x,y} / \sqrt{N \sum_{j=0}^{N-1} (a_{j,x,y})^2 + \epsilon} \), where \( \epsilon = 10^{-8} \). \( N \) is the number of feature maps, and \( a_{x,y} \) and \( b_{x,y} \) are the original and normalized feature vector in pixel \((x, y)\), respectively. We find it surprising that this heavy-handed constraint does not seem to harm the generator in any way, and indeed with most datasets it does not change the results much, but it prevents the escalation of signal magnitudes very effectively when needed.
5 Multi-scale statistical similarity for assessing GAN results

In order to compare the results of one GAN to another, one needs to investigate a large number of images, which can be tedious, difficult, and subjective. Thus it is desirable to rely on automated methods that compute some indicative metric from large image collections. We noticed that existing methods such as MS-SSIM (Odena et al., 2017) find large-scale mode collapses reliably but fail to react to smaller effects such as loss of variation in colors or textures, and they also do not directly assess image quality in terms of similarity to the training set.

We build on the intuition that a successful generator will produce samples whose local image structure is similar to the training set over all scales. We propose to study this by considering the multi-scale statistical similarity between distributions of local image patches drawn from Laplacian pyramid (Burt & Adelson, 1987) representations of generated and target images, starting at a low-pass scale.

We assess image quality in terms of similarity to the training set.

We will first use the sliced Wasserstein distance (SWD) and multi-scale structural similarity (MS-SSIM) (Odena et al., 2017) to evaluate the importance our individual contributions, and also perceptually validate the metrics themselves. We will do this by building on top of a previous state-of-the-art loss function (WGAN-GP) and training configuration (Gulrajani et al., 2017) in an unsupervised setting using CELEBA (Liu et al., 2015) and LSUN bedroom (Yu et al., 2015) datasets in 128×128 resolution. CELEBA is particularly well suited for such comparison because the images contain noticeable artifacts (aliasing, compression, blur) that are difficult for the generator to reproduce faithfully. In this test we amplify the differences between training configurations by choosing a relatively low-capacity network structure (Appendix A.2) and terminating the training once the discriminator has been shown a total of 10M real images. As such the results are not fully converged.

Table 1 lists the numerical values for SWD and MS-SSIM in several training configurations, where our individual contributions are cumulatively enabled one by one on top of the baseline (Gulrajani et al., 2017). The MS-SSIM numbers were averaged from 10000 pairs of generated images, and SWD was calculated as described in Section 5. Generated CELEBA images from these configurations are shown in Figure 3. Due to space constraints, the figure shows only a small number of examples for each row of the table, but a significantly broader set is available in Appendix H.
Table 1: Sliced Wasserstein distance (SWD) between the generated and training images (Section 5) and multi-scale structural similarity (MS-SSIM) among the generated images for several training setups at 128 × 128. For SWD, each column represents one level of the Laplacian pyramid, and the last gives an average of the four distances.

<table>
<thead>
<tr>
<th>Training configuration</th>
<th>CELEB-A</th>
<th>LSUN BEDROOM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sliced Wasserstein distance × 10^3</td>
<td>MS-SSIM</td>
</tr>
<tr>
<td>(a) Gulrajani et al. (2017)</td>
<td>12.99 7.79 7.82 8.73 9.28</td>
<td>0.2854</td>
</tr>
<tr>
<td>(b) + Progressive growing</td>
<td>6.42 2.64 3.78 6.06 4.28</td>
<td>0.2838</td>
</tr>
<tr>
<td>(c) + Small minibatch</td>
<td>75.42 41.33 41.62 26.57 46.23</td>
<td>0.4065</td>
</tr>
<tr>
<td>(d) + Revised training parameters</td>
<td>9.20 6.55 4.71 11.84 8.07</td>
<td>0.3027</td>
</tr>
<tr>
<td>(e*) + Minibatch discrimination</td>
<td>10.76 6.28 6.04 16.29 9.84</td>
<td>0.3057</td>
</tr>
<tr>
<td>(e) Minibatch stddev</td>
<td>13.94 5.67 2.83 5.71 7.04</td>
<td>0.2950</td>
</tr>
<tr>
<td>(f) + Equalized learning rate</td>
<td>4.42 3.28 2.32 7.52 4.39</td>
<td>0.2902</td>
</tr>
<tr>
<td>(g) + Pixelwise normalization</td>
<td>4.06 3.04 2.02 5.13 3.56</td>
<td>0.2845</td>
</tr>
<tr>
<td>(h) Converged</td>
<td>2.95 2.38 1.98 5.16 3.12</td>
<td>0.2880</td>
</tr>
</tbody>
</table>

Figure 3: (a) – (g) CELEBA examples corresponding to rows in Table 1. These are intentionally non-converged. (h) Our converged result. Notice that some images show aliasing and some are not sharp – this is a flaw of the dataset, which the model learns to replicate faithfully.

Putative, a good evaluation metric should reward plausible images that exhibit plenty of variation in colors, textures, and viewpoints. However, this is not captured by MS-SSIM: We can immediately see that configuration (h) generates significantly better images than configuration (a), but MS-SSIM remains approximately unchanged because it measures only the variation between outputs, not similarity to the training set. SWD, on the other hand, does indicate a clear improvement. The first training configuration (a) corresponds to Gulrajani et al. (2017), featuring batch normalization in the generator, layer normalization in the discriminator, and minibatch size of 64. (b) enables progressive growing of the networks, which results in sharper and more believable output images. SWD correctly finds the distribution of generated images to be more similar to the training set. Our primary goal is to enable high output resolutions, and this requires reducing the size of minibatches in order to stay within the available memory budget. We illustrate the ensuing challenges in (c) where we decrease the minibatch size from 64 to 16. The generated images are unnatural, which is clearly visible in both metrics. In (d), we stabilize the training process by adjusting the hyperparameters as well as by removing batch normalization and layer normalization. As an intermediate test (e*), we enable minibatch discrimination (Salimans et al., 2016), which somewhat surprisingly fails to improve any of the metrics, including MS-SSIM that measures output variation. In contrast, our minibatch standard deviation (e) improves the average SWD scores and images. We then enable our remaining contributions in (f) and (g), leading to an overall improvement in SWD and subjective visual quality. Finally, in (h) we use a non-crippled network and longer training – we feel the quality of the generated images is at least comparable to the best published results so far.

6.2 CONVERGENCE AND TRAINING SPEED

Figure 4 illustrates the effect of progressive growing in terms of the SWD metric and raw image throughput. The first two plots correspond to the training configuration of Gulrajani et al. (2017) without and with progressive growing. We observe that the progressive variant offers two main benefits: it converges to a considerably better optimum and also reduces the total training time by about a factor of two. The improved convergence is explained by an implicit form of curriculum learning that is imposed by the gradually increasing network capacity. Without progressive growing, all layers of the generator and discriminator are tasked with simultaneously finding succinct interme-
Figure 4: Effect of progressive growing on training speed and convergence. The timings were measured on a single-GPU setup using NVIDIA Tesla P100. (a) Statistical similarity with respect to wall clock time for Gulrajani et al. (2017) using CELEBA at $128 \times 128$ resolution. Each graph represents sliced Wasserstein distance on one level of the Laplacian pyramid, and the vertical line indicates the point where we stop the training in Table 1. (b) Same graph with progressive growing enabled. The dashed vertical lines indicate points where we double the resolution of G and D. (c) Effect of progressive growing on the raw training speed in $1024 \times 1024$ resolution.

The speedup from progressive growing increases as the output resolution grows. Figure 4(c) shows training progress, measured in number of real images shown to the discriminator, as a function of training time when the training progresses all the way to $1024^2$ resolution. We see that progressive growing gains a significant head start because the networks are shallow and quick to evaluate at the beginning. Once the full resolution is reached, the image throughput is equal between the two methods. The plot shows that the progressive variant reaches approximately 6.4 million images in 96 hours, whereas it can be extrapolated that the non-progressive variant would take about 520 hours to reach the same point. In this case, the progressive growing offers roughly a $5.4 \times$ speedup.

6.3 High-resolution image generation using CELEBA-HQ dataset

To meaningfully demonstrate our results at high output resolutions, we need a sufficiently varied high-quality dataset. However, virtually all publicly available datasets previously used in GAN literature are limited to relatively low resolutions ranging from $32^2$ to $480^2$. To this end, we created a high-quality version of the CELEBA dataset consisting of 30000 of the images at $1024 \times 1024$ resolution. We refer to Appendix C for further details about the generation of this dataset.

Our contributions allow us to deal with high output resolutions in a robust and efficient fashion. Figure 5 shows selected $1024 \times 1024$ images produced by our network. While megapixel GAN results have been shown before in another dataset (Marchesi, 2017), our results are vastly more varied and of higher perceptual quality. Please refer to Appendix F for a larger set of result images as well as the nearest neighbors found from the training data. The accompanying video shows latent space interpolations and visualizes the progressive training. The interpolation works so that we first randomize a latent code for each frame (512 components sampled individually from $\mathcal{N}(0, 1)$), then we blur the latents across time with a Gaussian ($\sigma = 45$ frames @ 60Hz), and finally normalize each vector to lie on a unit hypersphere.

We trained the network on a single NVIDIA Tesla P100 GPU for 20 days, after which we no longer observed qualitative differences between the results of consecutive training iterations. Our imple-
Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations. On the right, two images from an earlier megapixel GAN by Marchesi (2017) show limited detail and variation.


Figure 6: Visual quality comparison in LSUN BEDROOM; pictures copied from the cited articles.

mentation used an adaptive minibatch size depending on the current output resolution so that the available memory budget was optimally utilized.

In order to demonstrate that our contributions are largely orthogonal to the choice of a loss function, we have also trained the same network using LSGAN loss instead of WGAN-GP loss. Figure 1 shows six examples of 1024² images produced using our method using LSGAN. Further details of this setup are given in Appendix B.

6.4 LSUN RESULTS

Figure 6 shows a purely visual comparison between our solution and earlier results in LSUN BEDROOM. Figure 7 gives selected examples from seven very different LSUN categories at 256². A larger, non-curated set of results from all 30 LSUN categories is available in Appendix G, and the video demonstrates interpolations. We are not aware of earlier results in most of these categories, and while some categories work better than others, we feel that the overall quality is high.

6.5 CIFAR10 INCEPTION SCORES

The best inception scores for CIFAR10 (10 categories of 32 × 32 RGB images) we are aware of are 7.90 for unsupervised and 8.87 for label conditioned setups (Grinblat et al., 2017). The large
difference between the two numbers is primarily caused by “ghosts” that necessarily appear between classes in the unsupervised setting, while label conditioning can remove many such transitions.

When all of our contributions are enabled, we get 8.80 in the unsupervised setting. Appendix D shows a representative set of result images along with a more comprehensive list of results from earlier methods. The network and training setup were the same as for CELEBA, progression limited to $32 \times 32$ of course. The only customization was to the WGAN-GP’s regularization term $\mathbb{E}_{\hat{x} \sim \hat{P}}[(\|
abla_{\hat{x}} D(\hat{x})\|_2 - \gamma)^2]/\gamma^2)$. Gulrajani et al. (2017) used $\gamma = 1.0$, which corresponds to 1-Lipschitz, but we noticed that it is in fact significantly better to prefer fast transitions ($\gamma = 750$) to minimize the ghosts. We have not tried this trick with other datasets yet.

7 DISCUSSION

While the quality of our results is generally high compared to earlier work on GANs, and the training is stable in large resolutions, there is a long way to true photorealism. Semantic sensibility and understanding dataset-dependent constraints, such as certain objects being straight rather than curved, leaves a lot to be desired. There is also room for improvement in the micro-structure of the images. That said, we feel that convincing realism may now be within reach, especially in CELEBA-HQ.

REFERENCES


Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. *CoRR*, abs/1606.00704, 2016.


Table 2 shows the network architectures of full-resolution generator and discriminator that we use with a layer-specific constant at runtime as described in Section 4.1. We inject the across-minibatch weights according to the normal distribution with unit variance. However, we scale the weights either network, but we perform pixelwise normalization of the feature vectors after each Conv in the next 3-layer block during 600k images, etc. stabilize the networks for 600k images, fade have shown the discriminator 600k real images in total. We then alternate between two phases: fade in the first 3-layer block during the next 600k images, stabilize the networks for 600k images, fade in the next 3-layer block during 600k images, etc.

Our latent vectors correspond to random points on a 512-dimensional hypersphere. We use leaky ReLU with leakiness 0.2 in all layers of both networks, except for the last layer that uses linear activation. We do not employ batch normalization, layer normalization, or weight normalization in either network, but we perform pixelwise normalization of the feature vectors after each Conv 3 × 3 layer in the generator as described in Section 4.2. We initialize all bias parameters to zero and all weights according to the normal distribution with unit variance. However, we scale the weights with a layer-specific constant at runtime as described in Section 4.1. We inject the across-minibatch standard deviation as an additional feature map at 4 × 4 resolution toward the end of the discriminator as described in Section 3. The upsampling and downsampling operations in Table 2 correspond to 2 × 2 element replication and average pooling, respectively.

We train the networks using Adam (Kingma & Ba, 2015) with $\alpha = 0.001, \beta_1 = 0, \beta_2 = 0.99$, and $\epsilon = 10^{-8}$. We do not use any learning rate decay or rampdown, but for visualizing generator output at any given point during the training, we use an exponential running average for the weights of the generator with decay 0.999. We use a minibatch size 16 for resolutions $4^2$-$128^2$ and gradually then decrease the size according to $256^2 \rightarrow 14, 512^2 \rightarrow 6$, and $1024^2 \rightarrow 3$ to avoid exceeding the available memory budget. We use the WGAN-GP loss, but unlike Gulrajani et al. (2017), we alternate between optimizing the generator and discriminator on a per-minibatch basis, i.e., we set $n_{critic} = 1$. Additionally, we introduce a fourth term into the discriminator loss with an extremely

### Table 2: Generator and discriminator that we use with CELEBA-HQ to generate 1024×1024 images.

<table>
<thead>
<tr>
<th>Generator</th>
<th>Act.</th>
<th>Output shape</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latent vector</td>
<td>$-128 \times 1 \times 1$</td>
<td>$-1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>Conv 4 × 4</td>
<td>LReLU</td>
<td>$512 \times 4 \times 4$</td>
<td>$4.2M$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$512 \times 4 \times 4$</td>
<td>$2.4M$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-512 \times 8 \times 8$</td>
<td>$2.4M$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$512 \times 8 \times 8$</td>
<td>$2.4M$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$512 \times 8 \times 8$</td>
<td>$2.4M$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-512 \times 16 \times 16$</td>
<td>$2.4M$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$512 \times 16 \times 16$</td>
<td>$2.4M$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$512 \times 16 \times 16$</td>
<td>$2.4M$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-512 \times 32 \times 32$</td>
<td>$2.4M$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$512 \times 32 \times 32$</td>
<td>$2.4M$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$512 \times 32 \times 32$</td>
<td>$2.4M$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-256 \times 64 \times 64$</td>
<td>$1.2M$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$256 \times 64 \times 64$</td>
<td>$1.2M$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$256 \times 64 \times 64$</td>
<td>$590k$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-256 \times 128 \times 128$</td>
<td>$295k$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$128 \times 128 \times 128$</td>
<td>$148k$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$128 \times 128 \times 128$</td>
<td>$148k$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-64 \times 256 \times 256$</td>
<td>$74k$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$64 \times 256 \times 256$</td>
<td>$37k$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$64 \times 256 \times 256$</td>
<td>$37k$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-32 \times 512 \times 512$</td>
<td>$18k$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$32 \times 512 \times 512$</td>
<td>$18k$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$32 \times 512 \times 512$</td>
<td>$9.2k$</td>
</tr>
<tr>
<td>Upsample</td>
<td>$-16 \times 1024 \times 1024$</td>
<td>$4.6k$</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$16 \times 1024 \times 1024$</td>
<td>$4.6k$</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>LReLU</td>
<td>$16 \times 1024 \times 1024$</td>
<td>$2.3k$</td>
</tr>
<tr>
<td>Conv 1 × 1</td>
<td>linear</td>
<td>$3 \times 1024 \times 1024$</td>
<td>$51$</td>
</tr>
<tr>
<td><strong>Total trainable parameters</strong></td>
<td></td>
<td></td>
<td><strong>23.1M</strong></td>
</tr>
</tbody>
</table>

A NETWORK STRUCTURE AND TRAINING CONFIGURATION

#### A.1 1024 × 1024 NETWORKS USED FOR CELEBA-HQ

Table 2 shows the network architectures of full-resolution generator and discriminator that we use with CELEBA-HQ dataset. Both networks consist mainly of replicated 3-layer blocks that we introduce one by one during the course of the training. The last Conv $1 \times 1$ layer of the generator corresponds to the toRGB block in Figure 2, and the first Conv $1 \times 1$ layer of the discriminator similarly corresponds to fromRGB. We start with $4 \times 4$ resolution and train the networks until we have shown the discriminator 600k real images in total. We then alternate between two phases: fade in the first 3-layer block during the next 600k images, stabilize the networks for 600k images, fade in the next 3-layer block during 600k images, etc.

We train the networks using Adam (Kingma & Ba, 2015) with $\alpha = 0.001, \beta_1 = 0, \beta_2 = 0.99$, and $\epsilon = 10^{-8}$. We do not use any learning rate decay or rampdown, but for visualizing generator output at any given point during the training, we use an exponential running average for the weights of the generator with decay 0.999. We use a minibatch size 16 for resolutions $4^2$-$128^2$ and gradually then decrease the size according to $256^2 \rightarrow 14, 512^2 \rightarrow 6$, and $1024^2 \rightarrow 3$ to avoid exceeding the available memory budget. We use the WGAN-GP loss, but unlike Gulrajani et al. (2017), we alternate between optimizing the generator and discriminator on a per-minibatch basis, i.e., we set $n_{critic} = 1$. Additionally, we introduce a fourth term into the discriminator loss with an extremely
small weight to keep the output values from drifting too far away from zero. To be precise, we set $L' = L + \epsilon_{\text{drift}} \mathbb{E}_{x \in P_r}[D(x)^2]$, where $\epsilon_{\text{drift}} = 0.001$.

### A.2 Other Networks

Whenever we need to operate on a spatial resolution lower than $1024 \times 1024$, we do that by leaving out an appropriate number copies of the replicated 3-layer block in both networks. We also train each resolution for a total 800k images instead of 600k, and also fade in new layers for the duration of 800k images.

Furthermore, Section 6.1 uses a slightly lower-capacity version, where we halve the number of feature maps in Conv $3 \times 3$ layers at the $16 \times 16$ resolution, and divide by 4 in the subsequent resolutions. This leaves 32 feature maps to the last Conv $3 \times 3$ layers.

For the “Gulrajani et al. (2017)” case in Table 1, we follow their training configuration as closely as possible. In particular, we set $\alpha = 0.0001$, $\beta_2 = 0.9$, $n_{\text{critic}} = 5$, $\epsilon_{\text{drift}} = 0$, and minibatch size 64. We disable progressive resolution, minibatch stddev, as well as weight scaling at runtime, and initialize all weights using He’s initializer (He et al., 2015). Furthermore, we modify the generator by replacing LReLU with ReLU, linear activation with tanh in the last layer, and pixelwise normalization with batch normalization. In the discriminator, we add layer normalization to all Conv $3 \times 3$ and Conv $4 \times 4$ layers. For the latent vectors, we use 128 components sampled independently from the normal distribution.

### B Least-Squares GAN (LSGAN) at 1024 × 1024

We find that LSGAN is generally a less stable loss function that WGAN-GP, and it also has a tendency to lose some of the variation towards the end of long runs. Thus we prefer WGAN-GP, but have also produced high-resolution images by building on top of LSGAN, for example the $1024^2$ images in Figure 1 are LSGAN-based.

On top of the techniques described in Sections 2–4, we need one additional hack with LSGAN that prevents the training from spiraling out of control when the dataset is too easy for the discriminator, and the discriminator gradients are at risk of becoming meaningless as a result. We adaptively increase the magnitude of multiplicative Gaussian noise in discriminator as a function of the discriminator's output. The noise is applied to the input of each Conv $3 \times 3$ and Conv $4 \times 4$ layer. There is a long history of adding noise to the discriminator, and it is generally detrimental for the image quality (Arjovsky et al., 2017) and ideally one would never have to do that, which according to our tests is the case for WGAN-GP (Gulrajani et al., 2017). The magnitude of noise is determined as $0.2 \max(0, \hat{d}_t - 0.5)^2$, where $\hat{d}_t = 0.1d + 0.9\hat{d}_{t-1}$ is an exponential moving average of the discriminator output $d$. The motivation behind this hack is that LSGAN is seriously unstable when $d$ approaches (or exceeds) 1.0.

### C Celeba-HQ Dataset

In this section we describe the process we used to create the high-quality version of the Celeba dataset, consisting of 30000 images in $1024 \times 1024$ resolution. As a starting point, we took the collection of in-the-wild images included as a part of the original Celeba dataset. These images are extremely varied in terms of resolution and visual quality, ranging all the way from $43 \times 55$ to $6732 \times 8984$. Some of them show crowds of several people whereas others focus on the face of a single person – often only a part of the face. Thus, we found it necessary to apply several image processing steps to ensure consistent quality and to center the images on the facial region.

Our processing pipeline is illustrated in Figure 8. To improve the overall image quality, we preprocess each JPEG image using two pre-trained neural networks: a convolutional autoencoder trained to remove JPEG artifacts in natural images, similar in structure to the proposed by Mao et al. (2016a), and an adversarially-trained 4x super-resolution network (Korobchenko & Foco, 2017) similar to Ledig et al. (2016). To handle cases where the facial region extends outside the image, we employ padding and filtering to extend the dimensions of the image as illustrated in Fig.8(c–d).
(a) (b) (c) (d) (e) (f)

Figure 8: Creating the CELEBA-HQ dataset. We start with a JPEG image (a) from the CelebA in-the-wild dataset. We improve the visual quality (b,top) through JPEG artifact removal (b,middle) and 4x super-resolution (b,bottom). We then extend the image through mirror padding (c) and Gaussian filtering (d) to produce a visually pleasing depth-of-field effect. Finally, we use the facial landmark locations to select an appropriate crop region (e) and perform high-quality resampling to obtain the final image at $1024 \times 1024$ resolution (f).

then select an oriented crop rectangle based on the facial landmark annotations as follows:

\[
\begin{align*}
    x' &= e_1 - e_0 \\
    y' &= \frac{1}{2} (e_0 + e_1) - \frac{1}{2} (m_0 + m_1) \\
    c &= \frac{1}{2} (e_0 + e_1) - 0.1 \cdot y' \\
    s &= \max (4.0 \cdot |x'|, 3.6 \cdot |y'|) \\
    x &= \text{Normalize} \left( x' - \text{Rotate90}(y') \right) \\
    y &= \text{Rotate90}(x)
\end{align*}
\]

$e_0$, $e_1$, $m_0$, and $m_1$ represent the 2D pixel locations of the two eye landmarks and two mouth landmarks, respectively, $c$ and $s$ indicate the center and size of the desired crop rectangle, and $x$ and $y$ indicate its orientation. We constructed the above formulas empirically to ensure that the crop rectangle stays consistent in cases where the face is viewed from different angles. Once we have calculated the crop rectangle, we transform the rectangle to $4096 \times 4096$ pixels using bilinear filtering, and then scale it to $1024 \times 1024$ resolution using a box filter.

We perform the above processing for all 202599 images in the dataset, analyze the resulting $1024 \times 1024$ images further to estimate the final image quality, sort the images accordingly, and discard all but the best 30000 images. We use a frequency-based quality metric that favors images whose power spectrum contains a broad range of frequencies and is approximately radially symmetric. This penalizes blurry images as well as images that have conspicuous directional features due to, e.g., visible halftoning patterns. We selected the cutoff point of 30000 images as a practical sweet spot between variation and image quality, because it appeared to yield the best results.

D CIFAR10 RESULTS

<table>
<thead>
<tr>
<th>Method</th>
<th>Inception score</th>
<th>Method</th>
<th>Inception score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALI</td>
<td>5.34 ± 0.05</td>
<td>DCGAN</td>
<td>6.58</td>
</tr>
<tr>
<td>GMAN</td>
<td>6.00 ± 0.19</td>
<td>Improved GAN</td>
<td>8.09 ± 0.07</td>
</tr>
<tr>
<td>Improved GAN</td>
<td>6.86 ± 0.06</td>
<td>AC-GAN</td>
<td>8.25 ± 0.07</td>
</tr>
<tr>
<td>CEGAN-Ent-VI</td>
<td>7.07 ± 0.07</td>
<td>SGAN</td>
<td>8.59 ± 0.12</td>
</tr>
<tr>
<td>LR-AGN</td>
<td>7.17 ± 0.17</td>
<td>WGAN-GP†</td>
<td>8.67 ± 0.14</td>
</tr>
<tr>
<td>DFM</td>
<td>7.72 ± 0.13</td>
<td>Splitting GAN†</td>
<td>8.87 ± 0.09</td>
</tr>
<tr>
<td>WGAN-GP†</td>
<td>7.86 ± 0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Splitting GAN†</td>
<td>7.90 ± 0.09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our</td>
<td><strong>8.80 ± 0.05</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: CIFAR10 inception scores, higher is better. † uses “dequantization” augmentation.
Figure 9: CIFAR10 images generated using a network that was trained unsupervised (no label conditioning), and achieves a record 8.80 inception score.

Figure 9 shows non-curated images generated in the unsupervised setting, and Table 3 compares against prior art in terms of inception scores.

It is worth mentioning that some earlier methods augment the training dataset with low-intensity noise (“dequantization” in the official WGAN-GP implementation). We noticed that this improves the scores by \( \sim 0.1 \). In general, augmentations seem to be very powerful in this test, e.g., mirroring the training set lifts the scores by as much as 1.0. However, we did not use any kind of augmentation in our reported numbers.

## E MNIST-1K DISCRETE MODE TEST WITH CRIPPLED DISCRIMINATOR

<table>
<thead>
<tr>
<th>Arch</th>
<th>GAN</th>
<th>+ unrolling</th>
<th>WGAN-GP</th>
<th>+ our norm</th>
<th>+ mb stddev</th>
<th>+ progression</th>
<th>cudaGAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>K/4</td>
<td>#</td>
<td>30.6 ± 20.7</td>
<td>37.2 ± 20.7</td>
<td>60.1 ± 10.1</td>
<td>856.7 ± 50.4</td>
<td>881.3 ± 30.2</td>
<td>843.1 ± 31.8</td>
</tr>
<tr>
<td>KL</td>
<td>5.99 ± 0.06</td>
<td>4.64 ± 0.46</td>
<td>1.97 ± 0.29</td>
<td>1.19 ± 0.19</td>
<td>1.09 ± 0.10</td>
<td>1.05 ± 0.09</td>
<td>1.69 ± 0.19</td>
</tr>
<tr>
<td>K/2</td>
<td>#</td>
<td>62.9 ± 140.9</td>
<td>81.3 ± 39.5</td>
<td>77.4 ± 146.1</td>
<td>886.6 ± 88.5</td>
<td>918.4 ± 80.2</td>
<td>919.8 ± 35.1</td>
</tr>
<tr>
<td>KL</td>
<td>2.58 ± 0.75</td>
<td>1.43 ± 0.12</td>
<td>1.35 ± 0.55</td>
<td>0.98 ± 0.33</td>
<td>0.89 ± 0.21</td>
<td>0.82 ± 0.13</td>
<td>1.06 ± 0.10</td>
</tr>
</tbody>
</table>

Table 4: Results for MNIST discrete mode test using two tiny discriminators (K/4, K/2) defined by Metz et al. (2016). The number of covered modes (#) and KL divergence from a uniform distribution are given as an average \( \pm \) standard deviation over 8 random initializations. Higher is better for the number of modes, and lower is better for KL divergence.

Metz et al. (2016) describe a setup where a generator synthesizes MNIST digits simultaneously to 3 color channels, the digits are classified using a pre-trained classifier (0.4% error rate in our case), and concatenated to form a number in \([0, 999]\). They generate a total of 25,600 images and count how many of the discrete modes are covered. They also compute KL divergence as KL(histogram \| uniform). Modern GAN implementations can trivially cover all modes at very low divergence (0.05 in our case), and thus Metz et al. specify a fairly low-capacity generator and two severely crippled discriminators (“K/2” has \( \sim 2000 \) params and “K/4” only about \( \sim 500 \)) to tease out differences between training methodologies. Both of these networks use batch normalization.

As shown in Table 4, using WGAN-GP loss with the networks specified by Metz et al. covers much more modes than the original GAN loss, and even more than the unrolled original GAN with the smaller (K/4) discriminator. The KL divergence, which is arguably a more accurate metric than the raw count, acts even more favorably.

\[ \text{https://github.com/igul222/improved_wgan_training/} \]
Replacing batch normalization with our normalization improves the result considerably, while also slightly reducing the capacity of the discriminators. The addition of a minibatch stddev layer further improves the scores, while restoring the discriminator capacity to within 0.5% of the original. Progression does not help with these tiny images, but does not hurt either. In concurrent work, curriculum GANs (cuGAN) (Anonymous) find even more modes with “K/2”, but at higher KL divergence, and our results are better with the smaller discriminator (K/4).

**F  ADDITIONAL CELEBA-HQ RESULTS**

Figure 10 shows the nearest neighbors found for our generated images (Figure 5). Figure 11 gives additional generated examples from CELEBA-HQ.

**G  LSUN RESULTS**

Figures 12–17 show non-curated images generated from all 30 LSUN categories. A separate network was trained for each category using identical parameters. All categories were trained using 100k images, except for BEDROOM and DOG that used all the available data. Since 100k images is a very limited amount of training data for most categories, we enabled mirror augmentation in these tests (but not for BEDROOM or DOG).

**H  ADDITIONAL IMAGES FOR TABLE 1**

Figure 18 shows larger collections of images corresponding to the non-converged setups in Table 1. The training time was intentionally limited to make the differences between various methods more visible.
Figure 11: Additional 1024 × 1024 images generated using the CELEBA-HQ dataset. See the accompanying video for latent space interpolations.
Figure 12: Example images generated at $256 \times 256$ from LSUN categories.
Figure 13: Example images generated at $256 \times 256$ from LSUN categories.
Figure 14: Example images generated at 256 × 256 from LSUN categories.
Figure 15: Example images generated at $256 \times 256$ from LSUN categories.
Figure 16: Example images generated at 256 × 256 from LSUN categories.
Figure 17: Example images generated at $256 \times 256$ from LSUN categories.
Figure 18: A larger set of generated images corresponding to the non-converged setups in Table 1.