Seeing What a GAN Cannot Generate

David Bau[1,2], Jun-Yan Zhu[1], Jonas Wulff[1], William Peebles[1]
Hendrik Strobelt[2], Bolei Zhou[3], Antonio Torralba[1,2]
[1]MIT CSAIL, [2]MIT-IBM Watson AI Lab, [3]The Chinese University of Hong Kong

Abstract

Despite the success of Generative Adversarial Networks (GANs), mode collapse remains a serious issue during GAN training. To date, little work has focused on understanding and quantifying which modes have been dropped by a model. In this work, we visualize mode collapse at both the distribution level and the instance level. First, we deploy a semantic segmentation network to compare the distribution of segmented objects in the generated images with the target distribution in the training set. Differences in statistics reveal object classes that are omitted by a GAN. Second, given the identified omitted object classes, we visualize the GAN's omissions directly. In particular, we compare specific differences between individual photos and their approximate inversions by a GAN. To this end, we relax the problem of inversion and solve the tractable problem of inverting a GAN layer instead of the entire generator. Finally, we use this framework to analyze several recent GANs trained on multiple datasets and identify their typical failure cases.

1. Introduction

The remarkable ability of a Generative Adversarial Network (GAN) to synthesize realistic images leads us to ask: How can we know what a GAN is *unable* to generate? Mode-dropping or mode collapse, where a GAN omits portions of the target distribution, is seen as one of the biggest challenges for GANs [14, 24], yet current analysis tools provide little insight into this phenomenon in state-of-the-art GANs.

Our paper aims to provide detailed insights about dropped modes. Our goal is not to measure GAN quality using a single number: existing metrics such as Inception scores [34] and Fréchet Inception Distance [17] focus on that problem. While those numbers measure *how far* the generated and target distributions are from each other, we instead seek to understand *what* is different between real and fake images. Existing literature typically answers the latter question by sampling generated outputs, but such samples only visualize what a GAN is *capable* of doing. We address the complementary problem: we want to see what a GAN *cannot* generate.
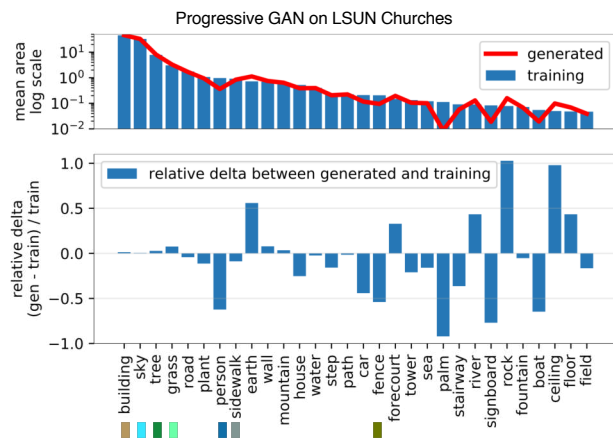
(a) generated vs training object segmentation statistics
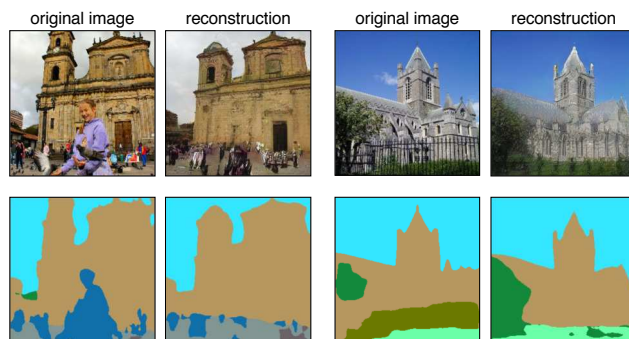
(b) real images vs. reconstructions

Figure 1. Seeing what a GAN cannot generate: (a) We compare the distribution of object segmentations in the training set of LSUN churches [47] to the distribution in the generated results: objects such as people, cars, and fences are dropped by the generator. (b) We compare pairs of a real image and its reconstruction in which individual instances of a person and a fence cannot be generated. In each block, we show a real photograph (top-left), a generated reconstruction (top-right), and segmentation maps for both (bottom).

In particular, we wish to know: Does a GAN deviate from the target distribution by ignoring difficult images altogether? Or are there specific, semantically meaningful parts and objects that a GAN decides not to learn about? And if so, how can we detect and visualize these missing concepts that a GAN does not generate?

Image generation methods are typically tested on images of faces, objects, or scenes. Among these, scenes are an especially fertile test domain as each image can be parsed into clear semantic components by segmenting the scene into objects. Therefore, we propose to directly understand mode dropping by analyzing a scene generator at two levels: the distribution level and instance level.

First, we characterize omissions in the distribution as a whole, using *Generated Image Segmentation Statistics*: we segment both generated and ground truth images and compare the distributions of segmented object classes. For example, Figure 1a shows that in a church GAN model, object classes such as people, cars, and fences appear on fewer pixels of the generated distribution as compared to the training distribution.

Second, once omitted object classes are identified, we want to visualize specific examples of failure cases. To do so, we must find image instances where the GAN *should* generate an object class but does not. We find such cases using a new reconstruction method called *Layer Inversion* which relaxes reconstruction to a tractable problem. Instead of inverting the entire GAN, our method inverts a layer of the generator. Unlike existing methods to invert a small generator [51, 8], our method allows us to create reconstructions for complex, state-of-the-art GANs. Deviations between the original image and its reconstruction reveal image features and objects that the generator cannot draw faithfully.

We apply our framework to analyze several recent GANs trained on different scene datasets. Surprisingly, we find that dropped object classes are not distorted or rendered in a low quality or as noise. Instead, they are simply not rendered at all, *as if the object was not part of the scene*. For example, in Figure 1b, we observe that large human figures are skipped entirely, and the parallel lines in a fence are also omitted. Thus a GAN can ignore classes that are too hard, while at the same time producing outputs of high average visual quality. Code, data, and additional information are available at **ganseeing.csail.mit.edu**.

2. Related work

Generative Adversarial Networks [15] have enabled many computer vision and graphics applications such as generation [7, 21, 22], image and video manipulation [19, 20, 30, 35, 39, 41, 52], object recognition [6, 42], and text-to-image translation [33, 45, 49]. One important issue in this emerging topic is how to evaluate and compare different methods [40, 43]. For example, many evaluation metrics have been proposed to evaluate unconditional GANs such as Inception score [34], Fréchet Inception Distance [17], and Wasserstein Sliced Distance [21]. Though the above metrics can quantify different aspects of model performance, they cannot explain what visual content the models fail to synthesize. Our goal here is *not* to introduce a metric. In-

stead, we aim to provide explanations of a common failure case of GANs: mode collapse. Our error diagnosis tools complement existing single-number metrics and can provide additional insights into the model's limitations.

Network inversion. Prior work has found that inversions of GAN generators are useful for photo manipulation [2, 8, 31, 51] and unsupervised feature learning [10, 12]. Later work found that DCGAN left-inverses can be computed to high precision [25, 46], and that inversions of a GAN for glyphs can reveal specific strokes that the generator is unable to generate [9]. While previous work [51] has investigated inversion of 5-layer DCGAN generators, we find that when moving to a 15-layer Progressive GAN, high-quality inversions are much more difficult to obtain. In our work, we develop a layer-wise inversion method that is more effective for these large-scale GANs. We apply a classic layer-wise training approach [5, 18] to the problem of training an encoder and further introduce layer-wise image-specific optimization. Our work is also loosely related to inversion methods for understanding CNN features and classifiers [11, 27, 28, 29]. However, we focus on understanding generative models rather than classifiers.

Understanding and visualizing networks. Most prior work on network visualization concerns discriminative classifiers [1, 3, 23, 26, 37, 38, 48, 50]. GANs have been visualized by examining the discriminator [32] and the semantics of internal features [4]. Different from recent work [4] that aims to understand what a GAN has learned, our work provides a complementary perspective and focuses on what semantic concepts a GAN fails to capture.

3. Method

Our goal is to visualize and understand the semantic concepts that a GAN generator cannot generate, in both the entire distribution and in each image instance. We will proceed in two steps. First, we measure *Generated Image Segmentation Statistics* by segmenting both generated and target images and identifying types of objects that a generator omits when compared to the distribution of real images. Second, we visualize how the dropped object classes are omitted for individual images by finding real images that contain the omitted classes and projecting them to their best reconstruction given an intermediate layer of the generator. We call the second step *Layer Inversion*.

3.1. Quantifying distribution-level mode collapse

The systematic errors of a GAN can be analyzed by exploiting the hierarchical structure of a scene image. Each scene has a natural decomposition into objects, so we can estimate deviations from the true distribution of scenes by estimating deviations of constituent object statistics. For example, a
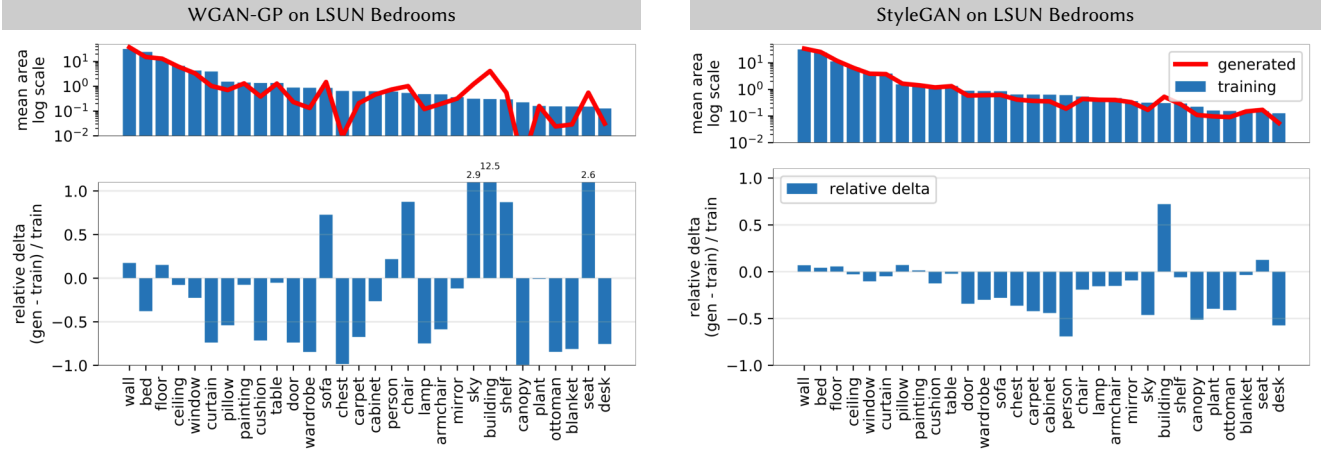
Figure 2. Using Generated Image Segmentation Statistics to understand the different behavior of the two models trained on LSUN bedrooms [47]. The histograms reveal that WGAN-GP [16] (left) deviates from the true distribution much more than StyleGAN [22] (right), identifying segmentation classes that are generated too little and others that are generated too much. For example, WGAN-GP does not generate enough pixels containing beds, curtains, or cushions compared to the true distribution of bedroom images, while StyleGAN correctly matches these statistics. StyleGAN is still not perfect, however, and does not generate enough doors, wardrobes, or people. Numbers above bars indicate clipped values beyond the range of the chart.

GAN that renders bedrooms should also render some amount of curtains. If the curtain statistics depart from what we see in true images, we will know we can look at curtains to see a specific flaw in the GAN.

To implement this idea, we segment all the images using the Unified Perceptual Parsing network [44], which labels each pixel of an image with one of 336 object classes. Over a sample of images, we measure the total area in pixels for each object class and collect mean and covariance statistics for all segmented object classes. We sample these statistics over a large set of generated images as well as training set images. We call the statistics over all object segmentations *Generated Image Segmentation Statistics*.

Figure 2 visualizes mean statistics for two networks. In each graph, the mean segmentation frequency for each generated object class is compared to that seen in the true distribution. Since most classes do not appear on most images, we focus on the most common classes by sorting classes by descending frequency. The comparisons can reveal many specific differences between recent state-of-the-art models. Both analyzed models are trained on the same image distribution (LSUN bedrooms [47]), but WGAN-GP [16] departs from the true distribution much more than StyleGAN [22].

It is also possible to summarize statistical differences in segmentation in a single number. To do this, we define the Fréchet Segmentation Distance (FSD), which is an interpretable analog to the popular Fréchet Inception Distance (FID) metric [17]: $\text{FSD} \equiv ||\mu_g - \mu_t||^2 + \text{Tr}(\Sigma_g + \Sigma_t - 2(\Sigma_g \Sigma_t)^{1/2})$. In our FSD formula, μ_t is the mean pixel count for each object class over a sample of training images, and Σ_t is the covariance of these pixel counts. Similarly, μ_g and Σ_g reflect segmentation statistics for the generative

model. In our experiments, we compare statistics between 10,000 generated samples and 10,000 natural images.

Generated Image Segmentation Statistics measure the entire distribution: for example, they reveal when a generator omits a particular object class. However, they do not single out specific images where an object should have been generated but was not. To gain further insight, we need a method to visualize omissions of the generator for each image.

3.2. Quantifying instance-level mode collapse

To address the above issue, we compare image pairs $(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}$ is a real image that contains a particular object class dropped by a GAN generator G, and $\mathbf{x}'$ is a projection onto the space of all images that can be generated by a layer of the GAN model.

Defining a tractable inversion problem. In the ideal case, we would like to find an image that can be perfectly synthesized by the generator G and stay close to the real image $\mathbf{x}$. Formally, we seek $\mathbf{x}' = G(\mathbf{z}^*)$, where $\mathbf{z}^* = \arg\min_{\mathbf{z}} \ell(G(\mathbf{z}), \mathbf{x})$ and ℓ is a distance metric in image feature space. Unfortunately, as shown in Section 4.4, previous methods [10, 51] fail to solve this full inversion problem for recent generators due to the large number of layers in G. Therefore, we instead solve a tractable subproblem of full inversion. We decompose the generator G into layers

$$G = G_f(g_n(\cdots((g_1(\mathbf{z}))))), \tag{1}$$

where $g_1, ..., g_n$ are several early layers of the generator, and G_f groups all the later layers of the G together.

Any image that can be generated by G can also be generated by G_f. That is, if we denote by $\text{range}(G)$ the
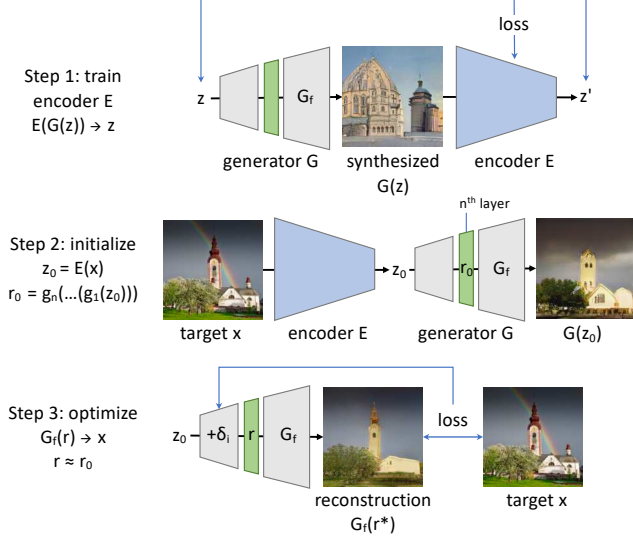
Figure 3. Overview of our layer inversion method. First, we train a network E to invert G; this is used to obtain an initial guess of the latent $\mathbf{z}_0 = E(\mathbf{x})$ and its intermediate representation $\mathbf{r}_0 = g_n(\cdots(g_1(\mathbf{z}_0)))$. Then $\mathbf{r}_0$ is used to initialize a search for $\mathbf{r}^*$ to obtain a reconstruction $\mathbf{x}' = G_f(\mathbf{r}^*)$ close to the target $\mathbf{x}$.

set of all images that can be output by G, then we have $\mathrm{range}(G) \subset \mathrm{range}(G_f)$. That implies, conversely, that any image that cannot be generated by G_f cannot be generated by G either. Therefore any omissions we can identify in $\mathrm{range}(G_f)$ will also be omissions of $\mathrm{range}(G)$.

Thus for layer inversion, we visualize omissions by solving the easier problem of inverting the later layers G_f:

$$\mathbf{x}' = G_f(\mathbf{r}^*), \qquad (2)$$
$$\text{where } \mathbf{r}^* = \arg\min_{\mathbf{r}} \ell(G_f(\mathbf{r}), \mathbf{x}).$$

Although we ultimately seek an intermediate representation $\mathbf{r}$, it will be helpful to begin with an estimated $\mathbf{z}$: an initial guess for $\mathbf{z}$ helps us regularize our search to favor values of $\mathbf{r}$ that are more likely to be generated by a $\mathbf{z}$. Therefore, we solve the inversion problem in two steps: First we construct a neural network E that approximately inverts the entire G and computes an estimate $\mathbf{z}_0 = E(\mathbf{x})$. Subsequently we solve an optimization problem to identify an intermediate representation $\mathbf{r}^* \approx \mathbf{r}_0 = g_n(\cdots(g_1(\mathbf{z}_0)))$ that generates a reconstructed image $G_f(\mathbf{r}^*)$ to closely recover $\mathbf{x}$. Figure 3 illustrates our layer inversion method.

Layer-wise network inversion. A deep network can be trained more easily by pre-training individual layers on smaller problems [18]. Therefore, to learn the inverting neural network E, we also proceed layer-wise. For each layer $g_i \in \{g_1, ..., g_n, G_f\}$, we train a small network e_i to approximately invert g_i. That is, defining $\mathbf{r}_i = g_i(\mathbf{r}_{i-1})$, our goal is to learn a network e_i that approximates the computation $\mathbf{r}_{i-1} \approx e_i(\mathbf{r}_i)$. We also want the predictions of the

network e_i to well preserve the output of the layer g_i, so we want $\mathbf{r}_i \approx g_i(e_i(\mathbf{r}_i))$. We train e_i to minimize both left- and right-inversion losses:

$$\mathcal{L}_{\mathrm{L}} \equiv \mathbb{E}_{\mathbf{z}}[||\mathbf{r}_{i-1} - e(g_i(\mathbf{r}_{i-1}))||_1]$$
$$\mathcal{L}_{\mathrm{R}} \equiv \mathbb{E}_{\mathbf{z}}[||\mathbf{r}_i - g_i(e(\mathbf{r}_i))||_1]$$
$$e_i = \arg\min_e \quad \mathcal{L}_{\mathrm{L}} + \lambda_{\mathrm{R}}\mathcal{L}_{\mathrm{R}}, \qquad (3)$$

To focus on training near the manifold of representations produced by the generator, we sample $\mathbf{z}$ and then use the layers g_i to compute samples of $\mathbf{r}_{i-1}$ and $\mathbf{r}_i$, so $\mathbf{r}_{i-1} = g_{i-1}(\cdots g_1(\mathbf{z})\cdots)$. Here $||\cdot||_1$ denotes an L1 loss, and we set $\lambda_R = 0.01$ to emphasize the reconstruction of $\mathbf{r}_{i-1}$.

Once all the layers are inverted, we can compose an inversion network for all of G:

$$E^* = e_1(e_2(\cdots(e_n(e_f(\mathbf{x}))))). \qquad (4)$$

The results can be further improved by jointly fine-tuning this composed network E^* to invert G as a whole. We denote this fine-tuned result as E.

Layer-wise image optimization. As described at the beginning of Section 3.2, inverting the entire G is difficult: G is non-convex, and optimizations over $\mathbf{z}$ are quickly trapped in local minima. Therefore, after obtaining a decent initial guess for $\mathbf{z}$, we turn our attention to the more relaxed optimization problem of inverting the layers G_f; that is, starting from $\mathbf{r}_0 = g_n(\cdots(g_1(\mathbf{z}_0)))$, we seek an intermediate representation $\mathbf{r}^*$ that generates a reconstructed image $G_f(\mathbf{r}^*)$ to closely recover $\mathbf{x}$.

To regularize our search to favor $\mathbf{r}$ that are close to the representations computed by the early layers of the generator, we search for $\mathbf{r}$ that can be computed by making small perturbations of the early layers of the generator:

$$\mathbf{z}_0 \equiv E(\mathbf{x})$$
$$\mathbf{r} \equiv \delta_n + g_n(\cdots(\delta_2 + g_2(\delta_1 + g_1(\mathbf{z}_0))))$$
$$\mathbf{r}^* = \arg\min_{\mathbf{r}} \left(\ell(\mathbf{x}, G_f(\mathbf{r})) + \lambda_{\mathrm{reg}}\sum_i ||\delta_i||^2 \right). \quad (5)$$

That is, we begin with the guess $\mathbf{z}_0$ given by the neural network E, and then we learn small perturbations of each layer before the n-th layer, to obtain an $\mathbf{r}$ that reconstructs the image $\mathbf{x}$ well. For ℓ we sum image pixel loss and VGG perceptual loss [36], similar to existing reconstruction methods [11, 51]. The hyper-parameter λ_{reg} determines the balance between image reconstruction loss and the regularization of $\mathbf{r}$. We set $\lambda_{\mathrm{reg}} = 1$ in our experiments.

4. Results

Implementation details. We analyze three recent models: WGAN-GP [16], Progressive GAN [21], and StyleGAN [22],
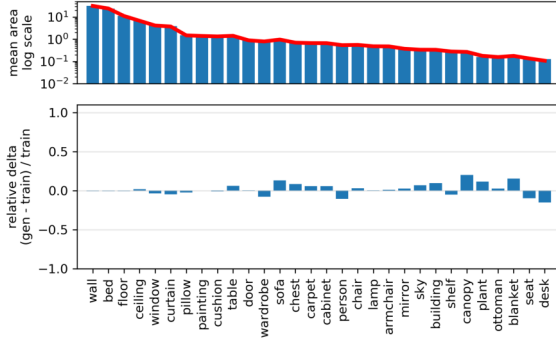
Figure 4. Sensitivity test for Generated Image Segmentation Statistics. This plot compares two different random samples of $10,000$ images from the LSUN bedroom dataset. An infinite-sized sample would show no differences; the observed differences reveal the small measurement noise introduced by the finite sampling process.

trained on LSUN bedroom images [47]. In addition, for Progressive GAN we analyze a model trained to generate LSUN church images. To segment images, we use the Unified Perceptual Parsing network [44], which labels each pixel of an image with one of 336 object classes. Segmentation statistics are computed over samples of 10,000 images.

4.1. Generated Image Segmentation Statistics

We first examine whether segmentation statistics correctly reflect the output quality of models across architectures. Figure 2 and Figure 5 show Generated Image Segmentation Statistics for WGAN-GP [16], StyleGAN [22], and Progressive GAN [21] trained on LSUN bedrooms [47]. The histograms reveal that, for a variety of segmented object classes, StyleGAN matches the true distribution of those objects better than Progressive GAN, while WGAN-GP matches least closely. The differences can be summarized using Fréchet Segmentation Distance (Table 1), confirming that better models match the segmented statistics better overall.

Model	FSD
WGAN-GP [16] bedrooms (Figure 2)	428.4
ProGAN [21] bedrooms (Figure 5)	85.2
StyleGAN [22] bedrooms (Figure 2)	22.6

Table 1. FSD summarizes Generated Image Segmentation Statistics

4.2. Sensitivity test

Figure 4 illustrates the sensitivity of measuring Generated Image Segmentation Statistics over a finite sample of $10,000$ images. Instead of comparing a GAN to the true distribution, we compare two different randomly chosen subsamples of the LSUN bedroom data set to each other. A perfect test with infinite sample sizes would show no difference; the small differences shown reflect the sensitivity of the test and are due to sampling error.
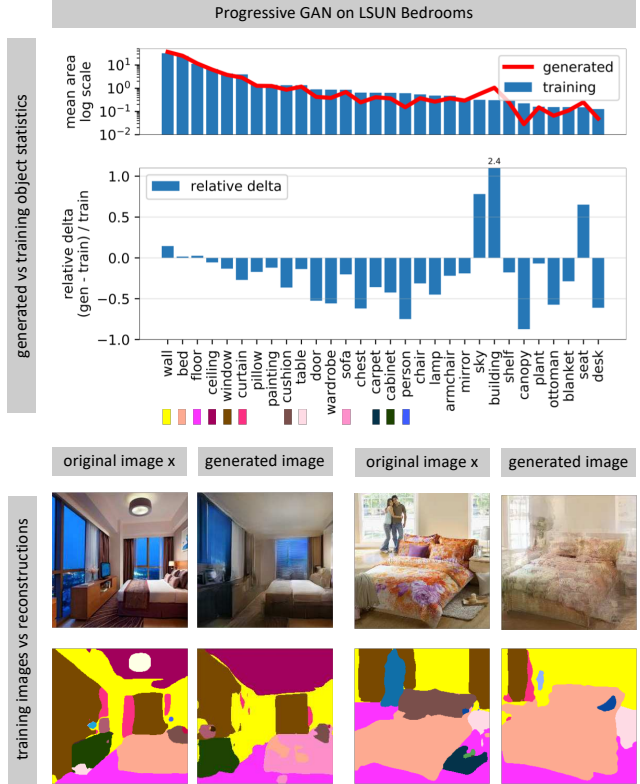

Figure 5. A visualization of the omissions of a bedroom generator; a Progressive GAN for LSUN bedrooms is tested. On top, a comparison of object distributions shows that many classes of objects are left out by the generator, including people, cushions, carpets, lamps, and several types of furniture. On the bottom, photographs are shown with their reconstructions $G(E(\mathbf{x}))$, along with segmentations. These examples directly reveal many object classes omitted by the bedroom generator.

4.3. Identifying dropped modes

Figure 1 and Figure 5 show the results of applying our method to analyze the generated segmentation statistics for Progressive GAN models of churches and bedrooms. Both the histograms and the instance visualizations provide insight into the limitations of the generators.

The histograms reveal that the generators partially skip difficult subtasks. For example, neither model renders as many people as appear in the target distribution. We use inversion to create reconstructions of natural images that include many pixels of people or other under-represented objects. Figure 1 and Figure 5 each shows two examples on the bottom. Our inversion method reveals the way in which the models fail. The gaps are not due to low-quality rendering of those object classes, but due to the wholesale omission of these classes. For example, large human figures and certain classes of objects are not included.
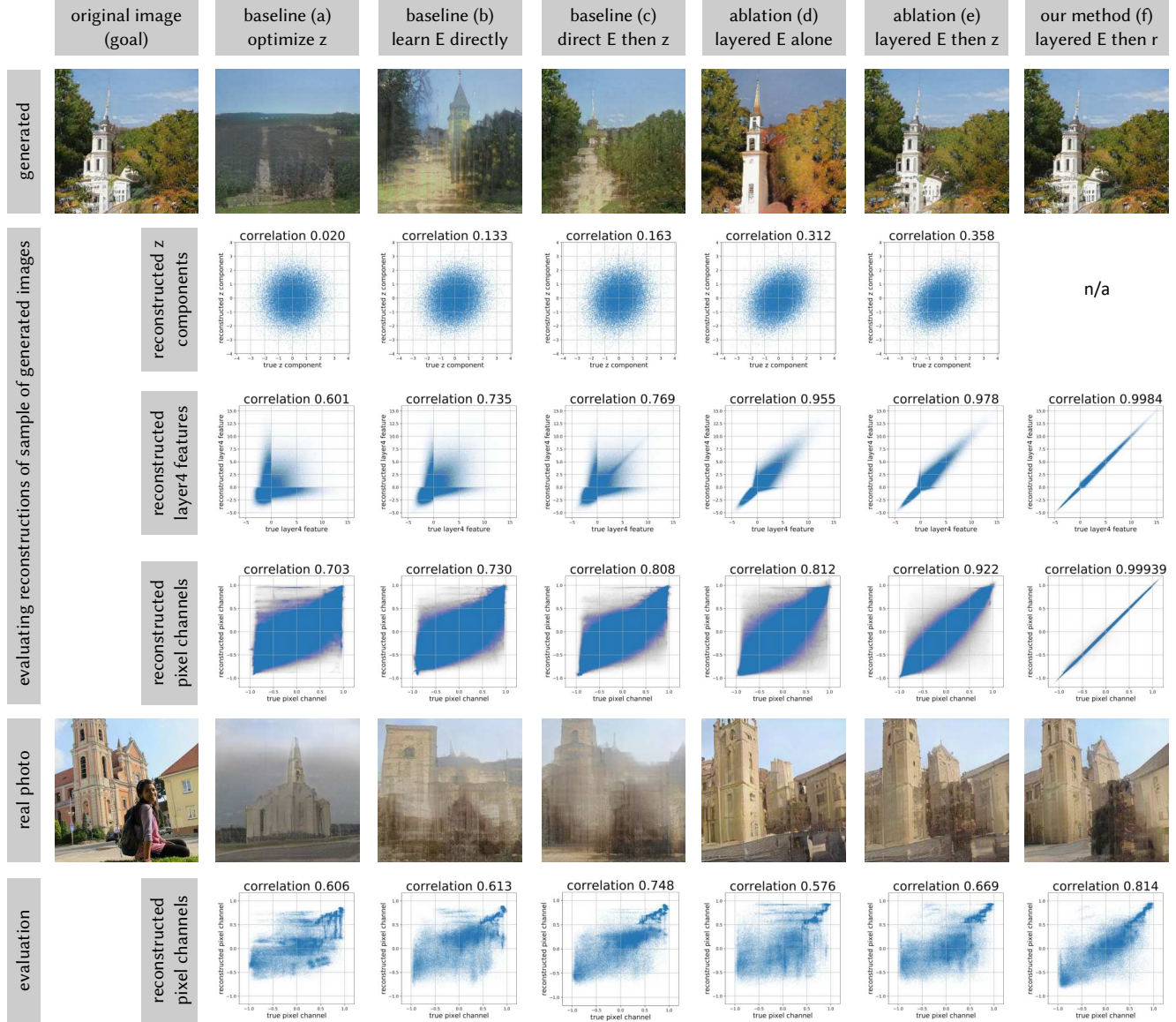
Figure 6. Comparison of methods to invert the generator of Progressive GAN trained to generate LSUN church images. Each method is described; (a) (b) and (c) are baselines, and (d), (e), and (f) are variants of our method. The first four rows show behavior given GAN-generated images as input. In the scatter plots, every point plots a reconstructed component versus its true value, with a point for every RGB pixel channel or every dimension of a representation. Reconstruction accuracy is shown as mean correlation over all dimensions for **z**, `layer4`, and image pixels, based on a sample of 100 images. Our method (f) achieves nearly perfect reconstructions of GAN-generated images. In the bottom rows, we apply each of the methods on a natural image.

4.4. Layer-wise inversion vs other methods

We compare our layer-wise inversion method to several previous approaches; we also benchmark it against ablations of key components of the method.

The first three columns of Figure 6 compare our method to prior inversion methods. We test each method on a sample of 100 images produced by the generator G, where the ground truth **z** is known, and the reconstruction of an example image is shown. In this case an ideal inversion should be able to

perfectly reconstruct $\mathbf{x}' = \mathbf{x}$. In addition, a reconstruction of a real input image is shown at the bottom. While there is no ground truth latent and representation for this image, the qualitative comparisons are informative.

(a) Direct optimization of z. Smaller generators such as 5-layer DCGAN [32] can be inverted by applying gradient descent on **z** to minimize reconstruction loss [51]. In column (a), we test this method on a 15-layer Progressive GAN and find that neither **z** nor **x** can be constructed accurately.
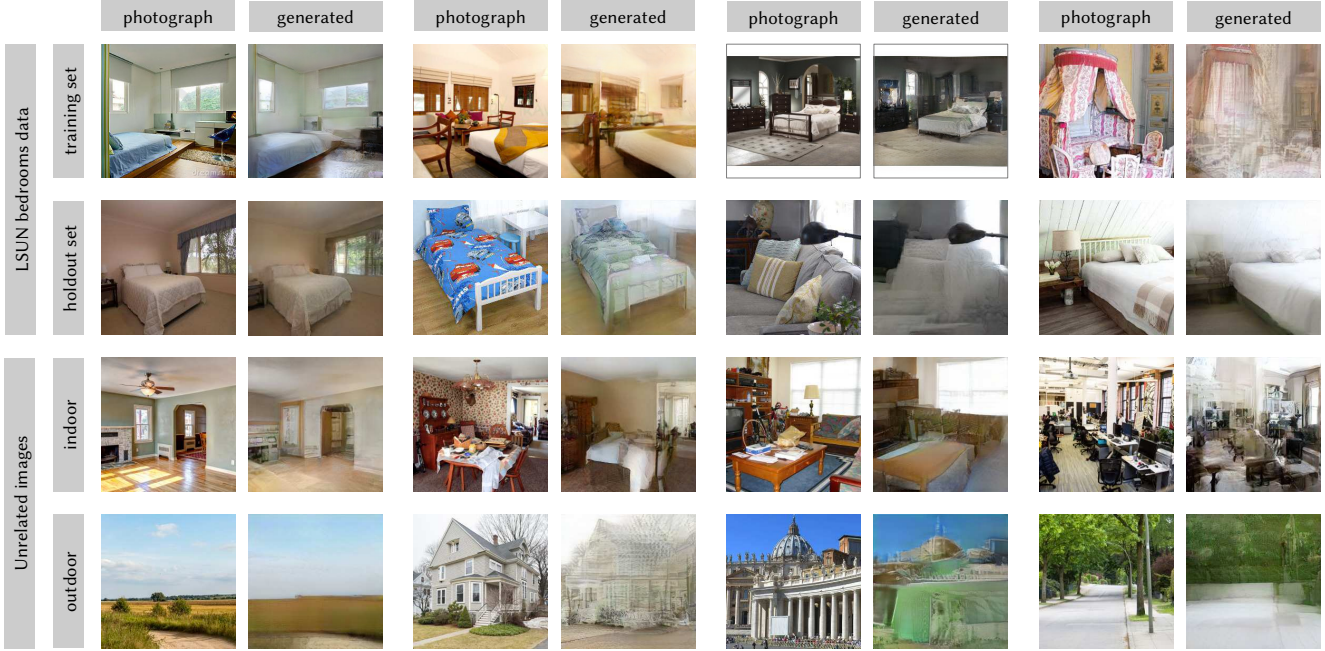
Figure 7. Inverting layers of a Progressive GAN bedroom generator. From top to bottom: uncurated reconstructions of photographs from the LSUN training set, the holdout set, and unrelated (non-bedroom) photographs, both indoor and outdoor.

(b): Direct learning of E**.** Another natural solution [10, 51] is to learn a deep network E that inverts G directly, without the complexity of layer-wise decomposition. Here, we learn an inversion network with the same parameters and architecture as the network E used in our method, but train it end-to-end by directly minimizing expected reconstruction losses over generated images, rather than learning it by layers. The method does benefit from the power of a deep network to learn generalized rules [13], and the results are marginally better than the direct optimization of $\mathbf{z}$. However, both qualitative and quantitative results remain poor.

(c): Optimization of z after initializing with $E(\mathbf{x})$**.** This is the full method used in [51]. By initializing method (a) using an initial guess from method (b), results can be improved slightly. For smaller generators, this method performs better than method (a) and (b). However, when applied to a Progressive GAN, the reconstructions are far from satisfactory.

Ablation experiments. The last three columns of Figure 6 compare our full method (f) to two ablations of our method.

(d): Layer-wise network inversion only. We can simply use the layer-wise-trained inversion network E as the full inverse, and simply use the initial guess $\mathbf{z}_0 = E(\mathbf{x})$, setting $\mathbf{x}' = G(\mathbf{z}_0)$. This fast method requires only a single forward pass through the inverter network E. The results are better than the baseline methods but far short of our full method.

Nevertheless, despite the inaccuracy of the latent code $\mathbf{z}_0$, the intermediate layer features are highly correlated with their true values; this method achieves 95.5% correlation

versus the true $\mathbf{r}_4$. Furthermore, the qualitative results show that when reconstructing real images, this method obtains more realistic results despite being noticeably different from the target image.

(e): Inverting G **without relaxation to** G_f**.** We can improve the initial guess $\mathbf{z}_0 = E(\mathbf{x})$ by directly optimizing $\mathbf{z}$ to minimize the same image reconstruction loss. This marginally improves upon $\mathbf{z}_0$. However, the reconstructed images and the input images still differ signficantly, and the recovery of $\mathbf{z}$ remains poor. Although the qualitative results are good, the remaining error means that we cannot know if any reconstruction errors are due to failures of G to generate an image, or if those reconstruction errors are merely due to the inaccuracy of the inversion method.

(f): Our full method. By relaxing the problem and regularizing optimization of $\mathbf{r}$ rather than $\mathbf{z}$, our method achieves nearly perfect reconstructions of both intermediate representations and pixels. Denote the full method as $\mathbf{r}^* = E_f(\mathbf{x})$.

The high precision of E_f within the range of G means that, when we observe large differences between $\mathbf{x}$ and $G_f(E_f(\mathbf{x}))$, they are unlikely to be a failure of E_f. This indicates that G_f cannot render $\mathbf{x}$, which means that G cannot either. Thus our ability to solve the relaxed inversion problem with an accuracy above 99% gives us a reliable tool to visualize samples that reveal what G cannot do.

Note that the purpose of E_f is to show dropped modes, not positive capabilities. The range of G_f upper-bounds the range of G, so the reconstruction $G_f(E_f(\mathbf{x}))$ could be better
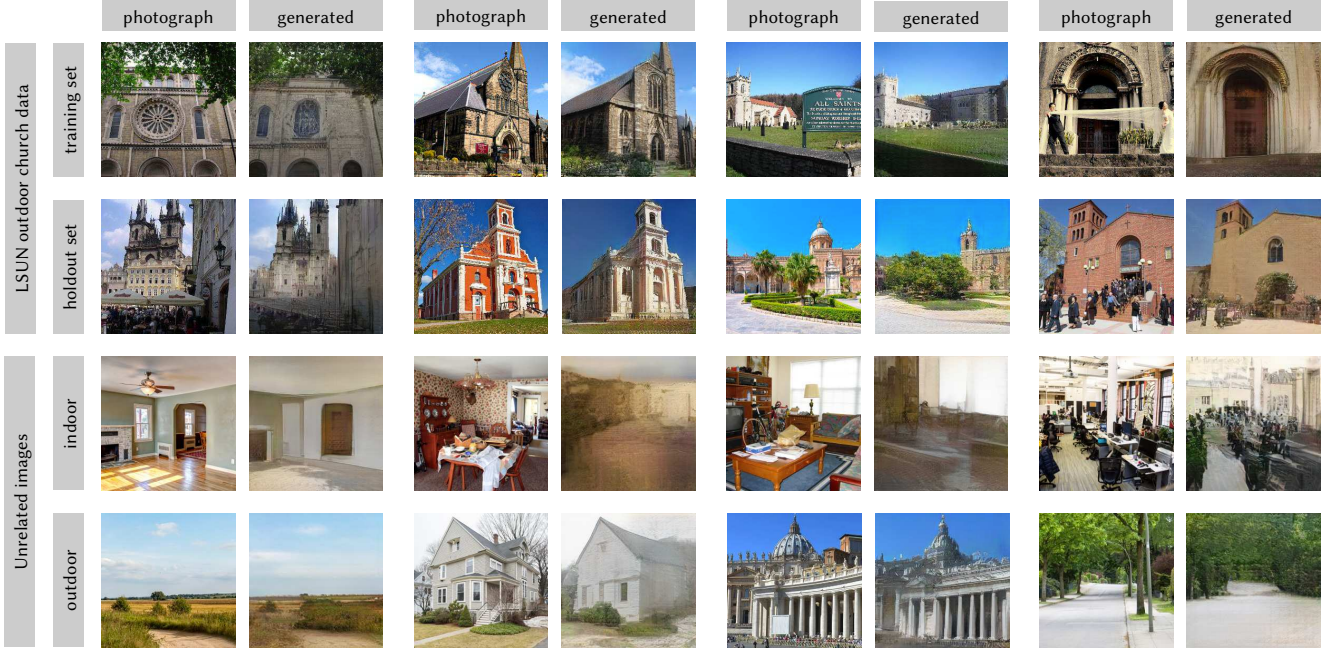
Figure 8. Inverting layers of a Progressive GAN outdoor church generator. From top to bottom: uncurated reconstructions of photographs from the LSUN training set, the holdout set, and unrelated (non-church) photographs, both indoor and outdoor.

than what the full network G is capable of. For a more complete picture, methods (d) and (e) can be additionally used as lower-bounds: those methods do not prove images are outside G's range, but they can reveal positive capabilities of G because they construct generated samples in range(G).

4.5. Layer-wise inversion across domains

Next, we apply the inversion tool to test the ability of generators to synthesize images outside their training sets. Figure 7 shows qualitative results of applying method (f) to invert and reconstruct natural photographs of different scenes using a Progressive GAN trained to generate LSUN bedrooms. Reconstructions from the LSUN training and LSUN holdout sets are shown; these are compared to newly collected unrelated (non-bedroom) images taken both indoors and outdoors. Objects that disappear from the reconstructions reveal visual concepts that cannot be represented by the model. Some indoor non-bedroom images are rendered in a bedroom style: for example, a dining room table with a white tablecloth is rendered to resemble a bed with a white bed sheet. As expected, outdoor images are not reconstructed well.

Figure 8 shows similar qualitative results using a Progressive GAN for LSUN outdoor church images. Interestingly, some architectural styles are dropped even in cases where large-scale geometry is preserved. The same set of unrelated (non-church) images as shown in Figure 7 are shown. When using the church model, the indoor reconstructions exhibit lower quality and are rendered to resemble outdoor scenes; the reconstructions of outdoor images recover more details.

5. Discussion

We have proposed a way to measure and visualize mode-dropping in state-of-the-art generative models. Generated Image Segmentation Statistics can compare the quality of different models and architectures, and provide insights into the semantic differences of their output spaces. Layer inversions allow us to further probe the range of the generators using natural photographs, revealing specific objects and styles that cannot be represented. By comparing labeled distributions with one another, and by comparing natural photos with imperfect reconstructions, we can identify specific objects, parts, and styles that a generator cannot produce.

The methods we propose here constitute a first step towards analyzing and understanding the latent space of a GAN and point to further questions. Why does a GAN decide to ignore classes that are more frequent than others in the target distribution (e.g. "person" vs. "fountain" in Figure 1)? How can we encourage a GAN to learn about a concept without skewing the training set? What is the impact of architectural choices? Finding ways to exploit and address the mode-dropping phenomena identified by our methods are questions for future work.

Acknowledgements

References

[1] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7), 2015. 2

[2] David Bau, Hendrik Strobelt, William Peebles, Jonas Wulff, Bolei Zhou, Jun-Yan Zhu, and Antonio Torralba. Semantic photo manipulation with a generative image prior. *SIGGRAPH*, 2019. 2

[3] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *CVPR*, 2017. 2

[4] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Zhou Bolei, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. In *ICLR*, 2019. 2

[5] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007. 2

[6] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017. 2

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *ICLR*, 2019. 2

[8] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. In *ICLR*, 2017. 2

[9] Antonia Creswell and Anil Anthony Bharath. Inverting the generator of a generative adversarial network. *IEEE transactions on neural networks and learning systems*, 2018. 2

[10] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *ICLR*, 2017. 2, 3, 7

[11] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. In *CVPR*, 2016. 2, 4

[12] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. In *ICLR*, 2017. 2

[13] Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, 2014. 7

[14] Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016. 1

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014. 2

[16] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NIPS*, 2017. 3, 4, 5

[17] 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 *NIPS*, 2017. 1, 2, 3

[18] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. 2, 4

[19] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. *ECCV*, 2018. 2

[20] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 2

[21] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018. 2, 4, 5

[22] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019. 2, 3, 4, 5

[23] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. *arXiv preprint arXiv:1711.00867*, 2017. 2

[24] Ke Li and Jitendra Malik. On the implicit assumptions of gans. *arXiv preprint arXiv:1811.12402*, 2018. 1

[25] Zachary C Lipton and Subarna Tripathi. Precise recovery of latent vectors from generative adversarial networks. *arXiv preprint arXiv:1702.04782*, 2017. 2

[26] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, 2017. 2

[27] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015. 2

[28] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017. 2

[29] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018. 2

[30] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 2

[31] Irad Peleg and Lior Wolf. Structured gans. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 719–728. IEEE, 2018. 2

[32] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 2, 6

[33] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *ICML*, 2016. 2

[34] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016. 1, 2

[35] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In *CVPR*, 2017. 2

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 4

[37] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017. 2

[38] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. 2

[39] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017. 2

[40] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *ICLR*, 2016. 2

[41] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *NIPS*, 2018. 2

[42] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *CVPR*, 2017. 2

[43] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. In *ICLR*, 2017. 2

[44] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *ECCV*, 2018. 3, 5

[45] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *CVPR*, 2018. 2

[46] Raymond A Yeh, Chen Chen, Teck Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *CVPR*, 2017. 2

[47] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015. 1, 3, 5

[48] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014. 2

[49] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, 2017. 2

[50] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014. 2

[51] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In *ECCV*, 2016. 2, 3, 4, 6, 7

[52] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017. 2