Supplementary materials for: Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space

S6. Markov chain Monte Carlo methods and derivation of MALA-approx

Assume a distribution $p(x)$ that we wish to produce samples from. For certain distributions with amenable structure it may be possible to write down directly an independent and identically distributed (IID) sampler, but in general this can be difficult. In such cases where IID samplers are not readily available, we may instead resort to Markov Chain Monte Carlo (MCMC) methods for sampling. Complete discussions of this topic fill books [25, 4]. Here we briefly review the background that led to the sampler we propose.

In cases where evaluation of $p(x)$ is possible, we can write down the Metropolis-Hastings (hereafter: *MH*) sampler for $p(x)$ [35, 18]. It requires a choice of proposal distribution $q(x'|x)$; for simplicity we consider (and later use) a simple Gaussian proposal distribution. Starting with an x_0 from some initial distribution, the sampler takes steps according to a transition operator defined by the below routine, with $N(0, \sigma^2)$ shorthand for a sample from that Gaussian proposal distribution:

1. $x_{t+1} = x_t + N(0, \sigma^2)$
2. $\alpha = p(x_{t+1})/p(x_t)$
3. if $\alpha < 1$, reject sample x_{t+1} with probability $1 - \alpha$ by setting $x_{t+1} = x_t$, else keep x_{t+1}

In theory, sufficiently many steps of this simple sampling rule produce samples for any computable $p(x)$, but in practice it has two problems: it mixes slowly, because steps are small and uncorrelated in time, and it requires us to be able to compute $p(x)$ to calculate α, which is often not possible. A Metropolis-adjusted Langevin algorithm (hereafter: *MALA*) [46, 45] addresses the first problem. This sampler follows a slightly modified procedure:

1. $x_{t+1} = x_t + \sigma^2/2 \nabla \log p(x_t) + N(0, \sigma^2)$
2. $\alpha = f(x_t, x_{t+1}, p(x_{t+1}), p(x_t))$
3. if $\alpha < 1$, reject sample x_{t+1} with probability $1 - \alpha$ by setting $x_{t+1} = x_t$, else keep x_{t+1}

where $f(\cdot)$ is the slightly more complex calculation of α, with the notable property that as the step size goes to 0, $f(\cdot) \to 1$. This sampler preferentially steps in the direction of higher probability, which allows it to spend less time rejecting low probability proposals, but it still requires computation of $p(x)$ to calculate α.

The stochastic gradient Langevin dynamics (SGLD) method [61, 52] was proposed to sidestep this troublesome requirement by generating probability proposals that are based on a small subset of the data only: by using stochastic gradient descent plus noise, by skipping the accept-reject step, and by using decreasing step sizes. Inspired by SGLD, we define an approximate sampler by assuming small step size and doing away with the reject step (by accepting every sample). The idea is that the stochasticity of SGD itself introduces an implicit noise: although the resulting update does not produce asymptotically unbiased samples, it does if we also anneal the step size (or, equivalently, gradually increase the minibatch size).

While an accept ratio of 1 is only approached in the limit as the step size goes to zero, in practice we empirically observe that this approximation produces reasonable samples even for moderate step sizes. This approximation leads to a sampler defined by the simple update rule:

$$x_{t+1} = x_t + \sigma^2/2 \nabla \log p(x_t) + N(0, \sigma^2) \qquad (12)$$

As explained below, we propose to decouple the two step sizes for each of the above two terms after x_t, with two independent scaling factors to allow independently tuning each (ϵ_{12} and ϵ_3 in Eq. 13). This variant makes sense when we consider that the stochasticity of SGD itself introduces more noise, breaking the direct link between the amount of noise injected and the step size under Langevin dynamics.

We note that $p(x) \propto \exp(-\text{Energy}(x))$, $\nabla \log p(x_t)$ is just the gradient of the energy (because the partition function does not depend on x) and that the scaling factor ($\sigma^2/2$ in the above equation) can be partially absorbed when changing the *temperature* associated with energy, since temperature is just a multiplicative scaling factor in the energy. Changing that link between the two terms is thus equivalent to changing temperature because the incorrect scale factor can be absorbed in the energy as a change

	mixes	uses accept/ reject step and requires $p(x)$	update rule (not including accept/reject step)
MH	slowly	yes	$x_{t+1} = x_t + N(0, \sigma^2)$
MALA	ok	yes	$x_{t+1} = x_t + 1/2\sigma\nabla \log p(x_t) + N(0, \sigma^2)$
MALA-approx	ok	no	$x_{t+1} = x_t + \epsilon_{12}\nabla \log p(x_t) + N(0, \epsilon_3^2)$

Table S1: Samplers properties assuming Gaussian proposal distributions. Samples are drawn via MALA-approx in this paper.

in the temperature. Since we do not control directly the amount of noise (some of which is now produced by the stochasticity of SGD itself), it is better to "manually" control the trade-off by introducing an extra hyperparameter. Doing so also may help to compensate for the fact that the SGD noise is not perfectly normal, which introduces a bias in the Markov chain. By manually controlling both the step size and the normal noise, we can thus find a good trade-off between variance (or temperature level, which would blur the distribution) and bias (which makes us sample from a slightly different distribution). In our experience, such decoupling has helped find better tradeoffs between sample diversity and quality, perhaps compensating for idiosyncrasies of sampling without a reject step. We call this sampler *MALA-approx*:

$$x_{t+1} = x_t + \epsilon_{12}\nabla \log p(x_t) + N(0, \epsilon_3^2) \tag{13}$$

Table S1 summarizes the samplers and their properties.

S7. Probabilistic interpretation of previous models (continued)

In this paper, we consider four main representative approaches in light of the framework:

1. Activation maximization with no priors [38, 51, 11]
2. Activation maximization with a Gaussian prior [48, 64]
3. Activation maximization with hand-designed priors [48, 64, 40, 60, 39, 38, 34]
4. Sampling in the latent space of a generator network [2, 63, 67, 6, 37, 17]

Here we discuss the first three and refer readers to the main text (Sec. 2.2) for the fourth approach.

Activation maximization with no priors. From Eq. 5, if we set $(\epsilon_1, \epsilon_2, \epsilon_3) = (0, 1, 0)$, we obtain a sampler that follows the class gradient directly without contributions from a $p(x)$ term or the addition of noise. In a high-dimensional space, this results in adversarial or fooling images [51, 38]. We can also interpret the sampling procedure in [51, 38] as a sampler with non-zero ϵ_1 but with a $p(x)$ such that $\frac{\partial \log p(x)}{\partial x} = 0$; in other words, a uniform $p(x)$ where all images are equally likely.

Activation maximization with a Gaussian prior. To combat the fooling problem [38], several works have used L_2 decay, which can be thought of as a simple Gaussian prior over images [48, 64, 60]. From Eq. 5, if we define a Gaussian $p(x)$ centered at the origin (assume the mean image has been subtracted) and set $(\epsilon_1, \epsilon_2, \epsilon_3) = (\lambda, 1, 0)$, pulling Gaussian constants into λ, we obtain the following noiseless update rule:

$$x_{t+1} = (1 - \lambda)x_t + \frac{\partial \log p(y = y_c|x_t)}{\partial x_t} \tag{14}$$

The first term decays the current image slightly toward the origin, as appropriate under a Gaussian image prior, and the second term pulls the image toward higher probability regions for the chosen class. Here, the second term is computed as the derivative of the log of a softmax unit in the output layer of the classification network, which is trained to model $p(y|x)$. If we let l_i be the logit outputs of a classification network, where i indexes over the classes, then the softmax outputs are given by $s_i = \exp(l_i)/\sum_j \exp(l_j)$, and the value $p(y = y_c|x_t)$ is modeled by the softmax unit s_c.

Note that the second term is similar, but not identical, to the gradient of logit term used by [48, 64, 34]. There are three variants of computing this class gradient term: 1) derivative of logit; 2) derivative of softmax; and 3) derivative of log of softmax. Previously mentioned papers empirically reported that derivative of the logit unit l_i produces better visualizations than the derivative of the softmax unit s_i (Table S2a vs. b), but this observation had not been fully justified [48]. In light of our probablistic interpretation (discussed in Sec. 2.1), we consider activation maximization as performing noisy gradient descent to minimize the energy function $E(x, y)$:

$$\begin{aligned} E(x, y) &= -log(p(x, y)) \\ &= -log(p(x)p(y|x)) \\ &= -(log(p(x)) + log(p(y|x))) \end{aligned} \tag{15}$$

To sample from the joint model $p(x, y)$, we follow the energy gradient:

a. Derivative of logit. Has worked well in practice [37, 11] but not quite the right term to maximize under the sampler framework set out in this paper.	$\frac{\partial l_i}{\partial x}$
b. Derivative of softmax. Previously avoided due to poor performance [48, 64], but poor performance may have been due to ill-conditioned optimization rather than the inclusion of logits from other classes. In particular, the term goes to 0 as s_i goes zero.	$\frac{\partial s_i}{\partial x} = s_i \left(\frac{\partial l_i}{\partial x} - \sum_j s_j \frac{\partial l_j}{\partial x} \right)$
c. Derivative of log of softmax. Correct term under the sampler framework set out in this paper. Well-behaved under optimization, perhaps due to the $\partial l_i / \partial x$ term untouched by the s_i multiplier.	$\frac{\partial \log s_i}{\partial x} = \frac{\partial \log p(y = y_i \vert x_t)}{\partial x} = \frac{\partial l_i}{\partial x} - \frac{\partial}{\partial x} \log \sum_j \exp(l_j)$

Table S2: A comparison of derivatives for use in activation maximization experiments. The first has most commonly been used, the second has worked in the past but with some difficulty, but the third is correct under the sampler framework set out in this paper. We perform experiments in this paper with the third variant.

$$\frac{\partial E(x,y)}{\partial x} = -\left(\frac{\partial log(p(x))}{\partial x} + \frac{\partial log(p(y|x))}{\partial x} \right) \quad (16)$$

which derives the class gradient term that matches that in our framework (Eq. 14, second term). In addition, recall that the classification network is trained to model $p(y|x)$ via softmax, thus the class gradient variant (the derivative of log of softmax) is the most theoretically justifiable in light of our interpretation. We summarize all three variants in Table S2. In overall, we found the proposed class gradient term a) theoretically justifiable under the probabilistic interpretation, and b) working well empirically, and thus suggest it for future activation maximization studies.

Activation maximization with hand-designed priors. In an effort to outdo the simple Gaussian prior, many works have proposed more creative, hand-designed image priors such as Gaussian blur [64], total variation [34], jitter [36], and data-driven patch priors [59]. These priors effectively serve as a simple $p(x)$ component. Those that cannot be explicitly expressed in the mathematical $p(x)$ form (e.g. jitter [36] and center-biased regularization [40]) can be written as a general regularization function $r(.)$ as in [64], in which case the noiseless update becomes:

$$x_{t+1} = r(x_t) + \frac{\partial \log p(y = y_c|x_t)}{\partial x_t} \quad (17)$$

Note that all methods considered in this section are noiseless and therefore produce samples showing diversity only by starting the optimization process at different initial conditions. The effect is that samples tend to converge to a single mode or a small number of modes [11, 40].

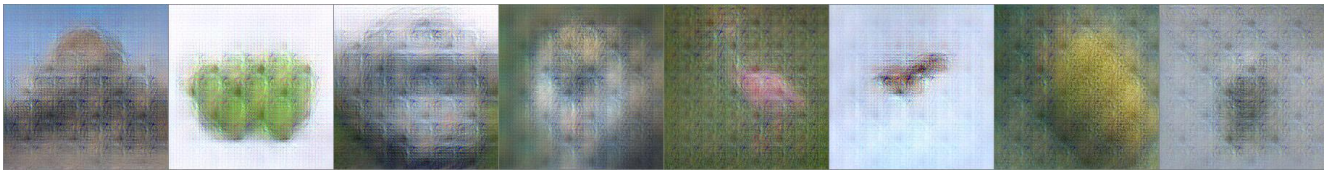
S8. Comparing feature matching losses

The addition of *feature matching* losses (i.e. the differences between a real image and a generated image not in pixel space, but in a feature space, such as a high-level code in a deep neural network) to the training cost has been shown to substantially improve the quality of samples produced by generator networks, e.g. by producing sharper and more realistic images [9, 28, 22].

Dosovitskiy & Brox [9] used the feature matching loss measured in the pool5 layer code space of AlexNet deep neural network (DNN) [26] trained to classify 1000-class ImageNet images [7]. Here, we empirically compare several feature matching losses computed in different layers of the AlexNet DNN. Specifically, we follow the training procedure in Dosovitskiy & Brox [9], and train 3 generator networks, each with a different feature matching loss computed in different layers from the pretrained AlexNet DNN: a) pool5, b) fc6 and c) both pool5 and fc6 losses. We empirically found that matching the pool5 features leads to the best image quality (Fig. S8), and chose the generator with this loss for the main experiments in the paper.

(a) Real images

(b) Joint PPGN-h ($L_{img} + L_{h_1} + L_h + L_{GAN}$)



(c) L_{GAN} removed ($L_{img} + L_{h_1} + L_h$)

(d) L_{h_1} removed: $L_{img} + L_h + L_{GAN}$

(e) L_h removed: $L_{img} + L_{h_1} + L_{GAN}$

Figure S8: A comparison of images produced by different generators G, each trained with a different loss combination (below each image). L_{img}, L_{h_1}, and L_h are L_2 reconstruction losses respectively in the pixel (x), pool5 feature (h_1) and fc6 feature (h) space. G is trained to map $h \rightarrow x$, i.e. reconstructing images from fc6 features. In the Joint PPGN-h treatment (Sec. 3.4), G is trained with a combination of 4 losses (panel b). Here, we perform an ablation study on this loss combination to understand the effect of each loss, and find a combination that produces the best image quality. We found that removing the GAN loss yields blurry results (panel c). The Noiseless Joint PPGN-h variant (Sec. 3.5) is trained with the loss combination that produces the best image quality (panel e). Compared to pool5, fc6 feature matching loss often produce the worse image quality because it is effectively encouraging generated images to match the high-level abstract statistics of real images instead of low-level statistics [16]. Our result is in consistent with Dosovitskiy & Brox [9].

S9. Training details

S9.1. Common training framework

We use the Caffe framework [21] to train the networks. All networks are trained with the Adam optimizer [23] with momentum $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\gamma = 0.5$, and an initial learning rate of 0.0002 following [9]. The batch size is 64. To stabilize the GAN training, we follow heuristic rules based on the ratio of the discriminator loss over generator loss $r = loss_D/loss_G$ and pause the training of the generator or discriminator if one of them is winning too much. In most cases, the heuristics are a) pause training D if $r < 0.1$; b) pause training G if $r > 10$. We did not find BatchNorm [20] helpful in further stabilizing the training as found in Radford et al. [43]. We have not experimented with all of the techniques discussed in Salimans et al. [47], some of which could further improve the results.

S9.2. Training PPGN-x

We train a DAE for images and incorporate it to the sampling procedure as a $p(x)$ prior to avoid fooling examples [37]. The DAE is a 4-layer convolutional network that encodes an image to the layer conv1 of AlexNet [26] and decodes it back to images with 3 upconvolutional layers. We add an amount of Gaussian noise $\sim N(0, \sigma^2)$ with $\sigma = 25.6$ to images during training. The network is trained via the common training framework described in Sec. S9.1 for $25,000$ mini-batch iterations. We use L_2 regularization of 0.0004.

S9.3. Training PPGN-h

For the PPGN-h variant, we train two separate networks: a generator G (that maps codes h to images x) and a prior $p(h)$. G is trained via the same procedure described in Sec. S9.4. We model $p(h)$ via a multi-layer perceptron DAE with 7 hidden layers of size: $4096 - 2048 - 1024 - 500 - 1024 - 2048 - 4096$. We experimented with larger networks but found this to work the best. We sweep across different amounts of Gaussian noise $N(0, \sigma^2)$, and empirically chose $\sigma = 1$ (i.e. $\sim$10% of the mean fc6 feature activation). The network is trained via the common training framework described in Sec. S9.1 for $100,000$ mini-batch iterations. We use L_2 regularization of 0.001.

S9.4. Training Noiseless Joint PPGN-h

Here we describe the training details of the generator network G used in the main experiments in Sections 3.3, 3.5, 3.4. The training procedure follows closely the framework by Dosovitskiy & Brox [9].

The purpose is to train a generator network G to reconstruct images from an abstract, high-level feature code space of an encoder network E—here, the first fully connected layer (fc6) of an AlexNet DNN [26] pre-trained to perform image classification on the ImageNet dataset [7] (Fig. S9a) We train G as a decoder for the encoder E, which is kept frozen. In other words, $E + G$ form an image autoencoder (Fig. S9b).

Training losses. G is trained with 3 different losses as in Dosovitskiy & Brox [9], namely, an adversarial loss L_{GAN}, an image reconstruction loss L_{img}, and a feature matching loss L_{h_1} measured in the spatial layer pool5 (Fig. S9b):

$$L_G = L_{img} + L_{h_1} + L_{GAN} \tag{18}$$

L_{img} and L_{h_1} are L_2 reconstruction losses in their respective space of images x and h_1 (pool5) codes :

$$L_{img} = ||\hat{x} - x||^2 \tag{19}$$

$$L_{h_1} = ||\hat{h_1} - h_1||^2 \tag{20}$$

The adversarial loss for G (which intuitively maximizes the chance D makes mistakes) follows the original GAN paper [14]:

$$L_{GAN} = -\sum_i log(D_\rho(G_\theta(h_i))) \tag{21}$$

where x_i is a training image, and $h_i = E(x_i)$ is a code. As in Goodfellow et al. [14], D tries to tell apart real and fake images, and is trained with the adversarial loss as follows:

$$L_D = -\sum_i log(D_\rho(x_i)) + log(1 - D_\rho(G_\theta(h_i))) \tag{22}$$

Architecture. G, an upconvolutional (also "deconvolutional") network [10] with 9 upconvolutional and 3 fully connected layers. D is a regular convolutional network for image classification with a similar architecture to AlexNet [26] with 5 convolutional layers followed by 3 fully connected layers, and 2 outputs (for "real" and "fake" classes).

The networks are trained via the common training framework described in Sec. S9.1 for 10^6 mini-batch iterations. We use L_2 regularization of 0.0004.

Specifics of DGN-AM reproduction. Note that while the original set of parameters in Nguyen et al. [37] (including a small number of iterations, an L_2 decay on code h, and a step size decay) produces high-quality images, it does not allow for a long sampling chain, traveling from one mode to another. For comparisons with other models within our framework, we sample from DGN-AM with $(\epsilon_1, \epsilon_2, \epsilon_3) = (0, 1, 10^{-17})$, which is slightly different from $(\lambda, 1, 0)$ in Eq. 10, but produces qualitatively the same result.

S9.5. Training Joint PPGN-h

Via the same existing network structures from DGN-AM [37], we train the generator G differently by treating the entire model as being composed of 3 interleaved DAEs: one for h, h_1, and x respectively (see Fig. S9c). Specifically, we add Gaussian noise to these variables during training, and by incorporating three corresponding L_2 reconstruction losses (see Fig. S9c). Adding noise to an AE can be considered as a form of regularization that encourages an autoencoder to extract more useful features [57]. Thus, here, we hypothesize that adding a small amount of noise to h_1 and x might slightly improve the result. In addition, the benefits of adding noise to h and training the pair G and E as a DAE for h are two fold: 1) it allows us to formally estimate the quantity $\partial log p(h)/\partial h$ (see Eq. 6) following a previous mathematical justification from Alain & Bengio [1]; 2) it allows us to sample with a larger noise level, which might improve the mixing speed.

We add noise to h during training, and train G with a L_2 reconstruction loss for h:

$$L_h = ||\hat{h} - h||^2 \tag{23}$$

Thus, generator network G is trained with 4 losses in total:

$$L_G = L_{img} + L_h + L_{h_1} + L_{GAN} \tag{24}$$

Three losses L_{img}, L_{h_1}, and L_{GAN} remain the same as in the training of Noiseless Joint PPGN-h (Sec. S9.4). Network architectures and other common training details remain the same as described in Sec. S9.4.

The amount of Gaussian noise $N(0, \sigma^2)$ added to the 3 different variables x, h_1, and h are respectively $\sigma = \{1, 4, 1\}$ which are $\sim$1% of the mean pixel values and $\sim$10% of the mean activations respectively in pool5 and fc6 space computed from the training set. We experimented with larger noise levels, but were not able to train the model successfully as large amounts of noise resulted in training instability. We also tried training without noise for x, i.e. treating the model as being composed of 2 DAEs instead of 3, but did not obtain qualitatively better results.

Note that while we did not experiment in this paper, jointly training both the generator G and the encoder E via their respective maximum likelihood training algorithms is possible. Also, Xie et al. [62] has proposed a training regime that alternatively updates these two networks. That cooperative training scheme indeeds yields a generator that synthesizes impressive results for multiple image datasets [62].

S10. Inpainting

We first randomly mask out a 100×100 patch of a real 227×227 image x_{real} (Fig. 7a). The patch size is chosen following Pathak et al. [42]. We perform the same update rule as in Eq. 11 (conditioning on a class, e.g. "volcano"), but with an additional step updating image x during the forward pass:

$$x = M \odot x + (1 - M) \odot x_{real} \tag{25}$$

where M is the binary mask for the corrupted patch, $(1 - M) \odot x_{real}$ is the uncorrupted area of the real image, and $\odot$ denotes the Hadamard (elementwise) product. Intuitively, we clamp the observed parts of the synthesized image and then sample only the unobserved portion in each pass. The DAE $p(h)$ model and the image classification network $p(y|h)$ model see progressively refined versions of the final, filled in image. This approach tends to fill in semantically correct content, but it often fails to match the local details of the surrounding context (Fig. 7b, the predicted pixels often do not transition smoothly to the surrounding context). An explanation is that we are sampling in the fully-connected fc6 feature space, which mostly encodes information of the global structure of objects instead of local details [64].

To encourage the synthesized image to match the context of the real image, we can add an extra condition in pixel space in the form of an additional term to the update rule in Eq. 5 to update h in the direction of minimizing the cost: $||(1 - M) \odot x_{real} - (1 - M) \odot x||_2^2$. This helps the filled-in pixels match the surrounding context better (Fig. 7 b vs. c). Compared to the Context-Aware Fill feature in Photoshop CS6, which is based on the PatchMatch technique [3], our method often performs worse in matching the local features of the surrounding context, but can fill in semantic objects better in many cases (Fig. 7, bird & bell pepper). More inpainting results are provided in the Fig. S24.

S11. PPGN-x: DAE model of $p(x)$

We investigate the effectiveness of using a DAE to model $p(x)$ directly (Fig. 3a). This DAE is a 4-layer convolutional network trained on unlabeled images from ImageNet. We sweep across different noise amounts for training the DAE and empirically find that a noise level of 20% of the pixel value range, corresponding to $\epsilon_3 = 25.6$, produces the best results. Full training and architecture details are provided in Sec. S9.2.

We sample from this chain following Eq. 7 with $(\epsilon_1, \epsilon_2, \epsilon_3) = (1, 10^5, 25.6)$[5] and show samples in Figs. S13a & S14a. PPGN-x exhibits two expected problems: first, it models the data distribution poorly, evidenced by the images becoming blurry over time. Second, the chain mixes slowly, changing only slightly in hundreds of steps.

[5] The ϵ_1 and ϵ_3 correspond to the noise level used while training the DAE, and the ϵ_2 value is chosen manually to produce the best samples.

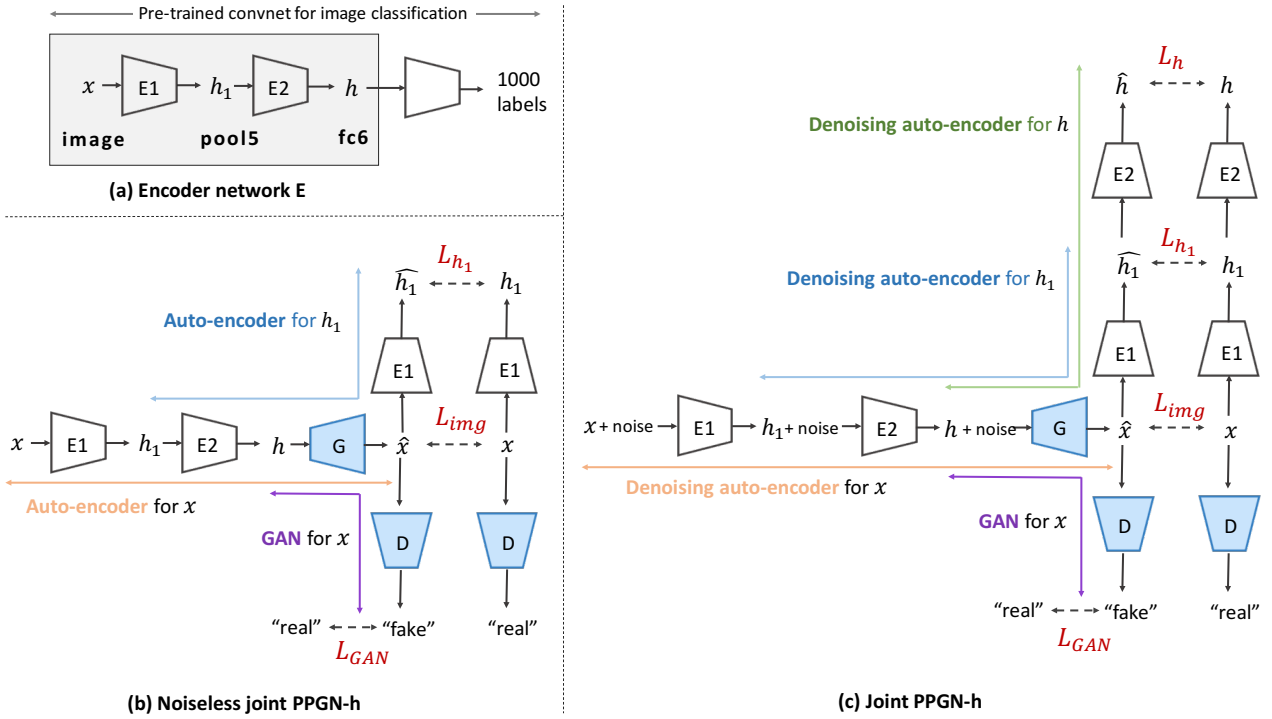


Figure S9: In this paper, we propose a class of models called PPGNs that are composed of 1) a generator network G that is trained to draw a wide range of image types, and 2) a replaceable "condition" network C that tells G what to draw (Fig. 3). Panel (b) and (c) show the components involved in the training of the generator network G for two main PPGN variants experimented in this paper. Only shaded components (G and D) are being trained while others are kept frozen. **b)** For the Noiseless Joint PPGN-h variant (Sec. 3.5), we train a generator G to reconstruct images x from compressed features h produced by a pre-trained encoder network E. Specifically, h and h_1 are, respectively, features extracted at layer fc6 and pool5 of AlexNet [26] trained to classify ImageNet images (a). G is trained with 3 losses: an image reconstruction loss L_{img}, a feature matching loss [9] L_{h_1} and an adversarial loss [14] L_{GAN}. As in Goodfellow et al. [14], D is trained to tell apart real and fake images. This PPGN variant produces the best image quality and thus used for the main experiments in this paper (Sec. 4). After G is trained, we sample from this model following an iterative sampling procedure described in Sec. 3.5. **c)** For the Joint PPGN-h variant (Sec. 3.4), we train the entire model as being composed of 3 interleaved DAEs respectively for x, h_1 and h. In other words, we add noise to each of these variables and train the corresponding AE with a L_2 reconstruction loss. The loss for D remains the same as in (a), while the loss for G is now composed of 4 components: $L = L_{img} + L_{h_1} + L_h + L_{GAN}$. The sampling procedure for this PPGN variant is provided in Sec. 3.4. See Sec. S9 for more training and architecture details of the two PPGN variants.

Note that, instead of training the above DAE, one can also form an x-DAE by combining a pair of separately trained encoder E and a generator G into a composition $E(G(.))$. We also experiment with this model and call it Joint PPGN-x. The details of network E and G and how they can be combined are described in Sec. 3.4 (Joint PPGN-h). For sampling, we sample in the image space, similarly to the PPGN-x in this section. We found that Joint PPGN-x model performs better than PPGN-x, but worse than Joint PPGN-h (data not shown).

S12. Why PPGNs produce high-quality images

One practical question is why Joint PPGN-h produces high-quality images at a high resolution for 1000-class ImageNet more successfully than other existing latent variable models [41, 47, 43]. We can consider this question from two perspectives.

First, from the perspective of the training loss, G is trained with the combination of three losses (Fig. S9b), which may be a beneficial approach to model $p(x)$. The GAN [14] loss, which is the gradient of $\log(1 - D(x))$, that is used to train G pushes each reconstruction $G(h)$ toward a mode of real images $p_{\text{data}}(x)$ and away from the current reconstruction distribution. This can be seen by noting

that the Bayes optimal D is $p_{\text{data}}(x)/(p_{\text{data}}(x) + p_{\text{model}}(x))$ [14]. Since $x \sim G(h)$ is already near a mode of $p_{\text{model}}(x)$, the net effect is to push $G(h)$ towards one of the modes of p_{data}, thus making the reconstructions sharper and more plausible. If one uses only the GAN objective and no reconstruction objectives (L_2 losses in the pixel or feature space), G may bring the sample far from the original x, possibly collapsing several modes of x into fewer modes. This is the typical, known "missing-mode" behavior of GANs [47, 14] that arises in part because GANs minimize the Jensen-Shannon divergence rather than Kullback-Leibler divergence between p_{data} and p_{model}, leading to an over-memorization of modes [53]. The reconstruction losses are important to combat this missing mode problem and may also serve to enable better convergence of the feature space autoencoder to the distribution it models, which is necessary in order to make the h-space reconstruction properly estimate $\partial \log p(h)/\partial h$ [1].

Second, from the perspective of the learned $h \rightarrow x$ mapping, we train the G parameters of the $E + G$ pair of networks as an x-AE, mapping $x \rightarrow h \rightarrow x$ (see Fig. S9b). In this configuration, as in VAEs [24] and regular DAEs [57], the one-to-one mapping helps prevent the typical latent $\rightarrow$ input missing mode collapse that occurs in GANs, where some input images are not representable using any code [14, 47]. However, unlike in VAEs and DAEs, where the latent distribution is learned in a purely unsupervised manner, we leverage the labeled ImageNet data to train E in a supervised manner that yields a distribution of features h that we hypothesize to be semantically meaningful and useful for building a generative image model. To further understand the effectiveness of using deep, supervised features, it might be interesting future work to train PPGNs with other feature distributions h such as random features or shallow features (e.g. produced by PCA).

Model	Image size	Inception accuracy	Inception score	MS-SSIM	Percent of classes
Real ImageNet images	256×256	76.1%	210.4 ± 4.6	0.10 ± 0.06	999 / 1000
AC-GAN [41]	128×128	10.1%	N/A	N/A	847 / 1000
PPGN	256×256	59.6%	60.6 ± 1.6	0.23 ± 0.11	829 / 1000
PPGN samples resized to 128×128	128×128	54.8%	47.7 ± 1.0	0.25 ± 0.11	770 / 1000

Table S3: A comparison between real ImageNet validation set images, AC-GAN [41] samples, PPGN samples and their resized 128×128 versions. Following the literature, we report Inception scores [47] (higher is better) and Inception accuracies [41] (higher is better) to evaluate sample quality, and MS-SSIM score [41] (lower is better), which measures sample diversity within each class. As in Odena et al. [41], the last column ("Percent of classes", higher is better) shows the number of classes that are more diverse (by MS-SSIM metric) than the least diverse class in ImageNet. Overall, PPGN samples are of substantially higher quality quality than AC-GAN samples (by Inception accuracy, i.e. PPGN samples are far more recognizable by the Google Inception network [50] than AC-GAN samples). Their diversity scores are similar (last column, $847/1000$ vs. $829/1000$). However, by all 4 metrics, PPGN samples have substantially lower diversity and quality than real images. This result aligns with our qualitative observations in Figs. S25 & S10.

Row 2: Note that we chose to compare with AC-GAN [41] because, this model is also class-conditional and, to the best of our knowledge, it produces the previous highest resolution ImageNet images (128×128) in the literature.

Row 3: For comparison with ImageNet 256×256 images, the spatial dimension of the samples from the generator G is 256×256 and we did not crop it to 227×227 as done in other experiments in the paper.

Row 4: Although imperfect, we resized PPGN 256×256 samples down to 128×128 (last row) for comparison with AC-GAN.

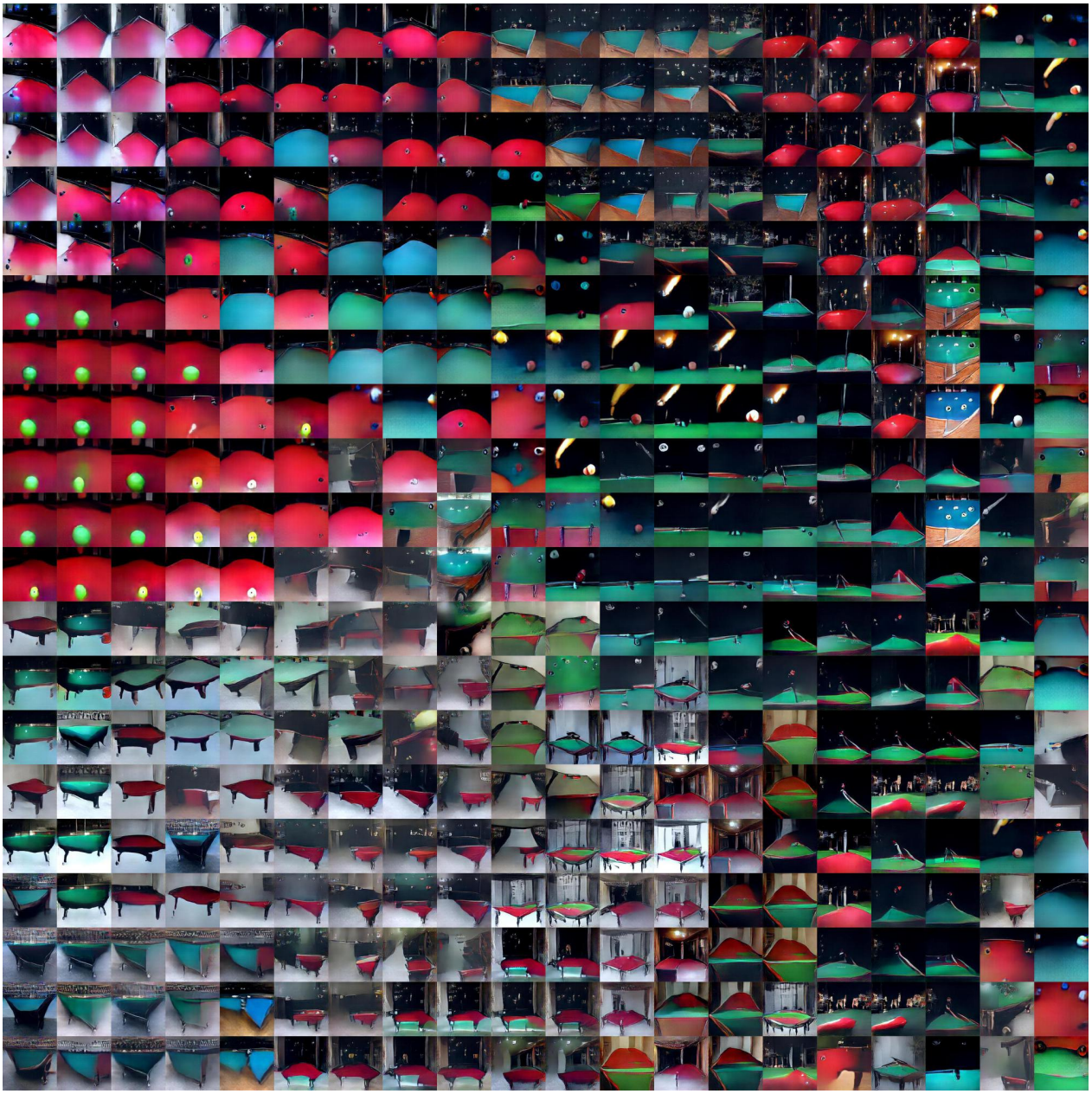
Figure S10: (a) The 9 training set images that most highly activate a given class output neuron (e.g. fire engine). (b) DGN-AM [37] synthesizes high-quality images, but they often converge to the mode of high-activating images (the top-9 mode). (c) 9 training set images randomly picked from the same class. (d) Our new method PPGN produces samples with better quality and substantially larger diversity than DGN-AM, thus better representing the diversity of images from the class.

(a) Samples produced by PPGN visualized in a grid t-SNE [56]

.

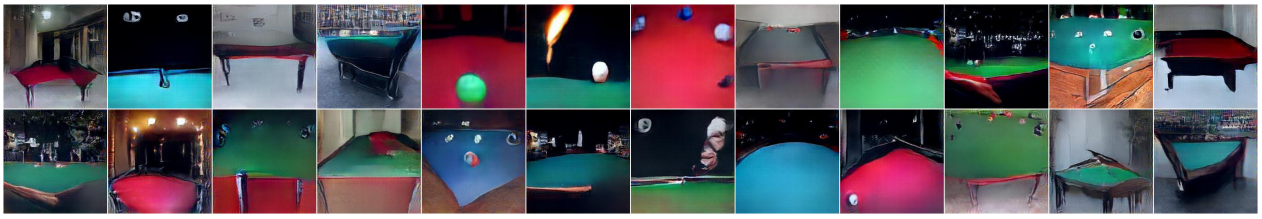
(b) Samples hand-picked from (a) to showcase the diversity and quality of images produced by PPGN.

Figure S11: We qualitatively evaluate sample diversity by running 10 sampling chains (conditioned on class "volcano"), each for 200 steps, to produce 2000 samples, and filtering out samples with class probability of less than 0.97. From the remaining, we randomly pick 400 samples and plot them in a grid t-SNE [56] (top panel). From those, we chose a selection to highlight the quality and diversity of the samples (bottom panel). There is a tremendous amount of detail in each image and diversity across images. Samples include dormant volcanos and active volcanoes with smoke plumes of different colors from white to black to fiery orange. Some have two peaks and others one, and underneath are scree, green forests, or glaciers (complete with crevasses). The sky changes from different shades of mid-day blue through different sunsets to pitch black night.



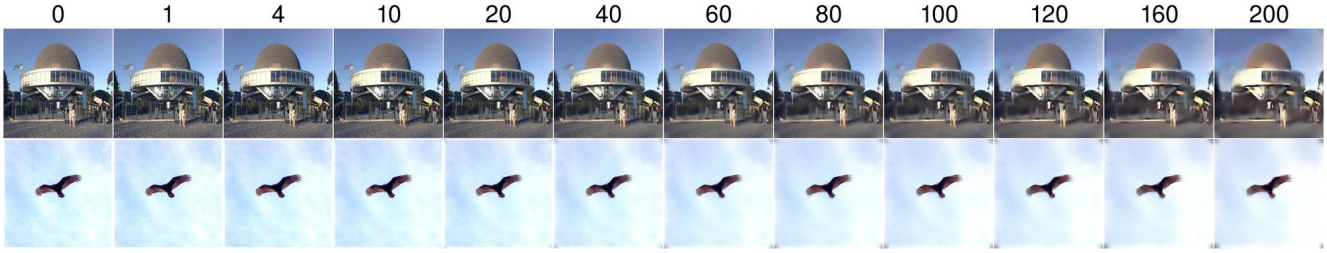
(a) Samples produced by PPGN visualized in a grid t-SNE [56]

.



(b) Samples hand-picked from (a) to showcase the diversity and quality of images produced by PPGN.

Figure S12: The figures are selected and plotted in the same way as Fig. S11, but here for the "pool table" class. Once again, we observe a high degree of both image quality and diversity. Different felt colors (green, blue, and red), lighting conditions, camera angles, and interior designs are apparent.



(a) PPGN-x with a DAE model of $p(x)$

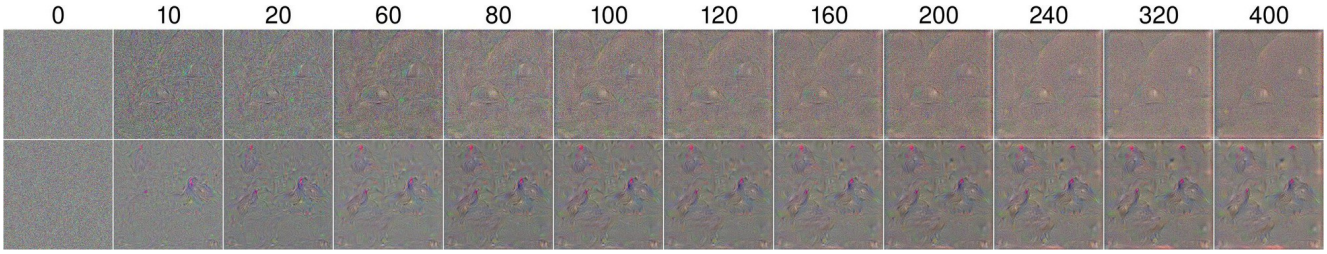
(b) DGN-AM [37] (which has a hand-designed Gaussian $p(h)$ prior)

(c) PPGN-h: Generator and multi-layer perceptron DAE model of $p(h)$

(d) Joint PPGN-h: joint Generator and DAE

(e) Noiseless Joint PPGN-h: joint Generator and AE

Figure S13: A comparison of samples generated from a single sampling chain (starting from a real image on the left) across different models. Each panel shows two sampling chains for that model: one conditioned on the "planetarium" class and the other conditioned on the "kite" (a type of bird) class. The iteration number of the sampling chain is shown on top. (a) The sampling chain in the image space mixes poorly. (b) The sampling chain from DGN-AM [37] (in the h code space with a hand-designed Gaussian $p(h)$ prior) produces better images, but still mixes poorly, as evidenced by similar samples over many iterations. (c) To improve sampling, we tried swapping in a $p(h)$ model represented by a 7-layer DAE for h. However, the sampling chain does not mix faster or produce better samples. (d) We experimented with a better way to model $p(h)$, i.e. modeling h via x. We treat the generator G and encoder E as an autoencoder for h and call this treatment "Noiseless Joint PPGN-h" (see Sec. 3.5). This is also our best model that we use for experiments in Sec. 4. This substantially improves the mixing speed and sample quality. (e) We train the entire model as being composed of 3 DAEs and sample from it by adding noise to the image, fc6 and pool5 variables. The chain mixes slightly faster compared to (d), but generates slightly worse samples.

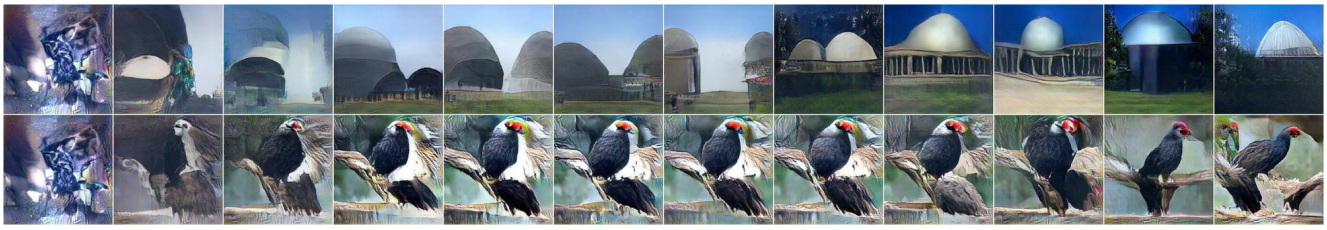


(a) PPGN-x with a DAE model of $p(x)$

(b) DGN-AM [37] (which has a hand-designed Gaussian $p(h)$ prior)

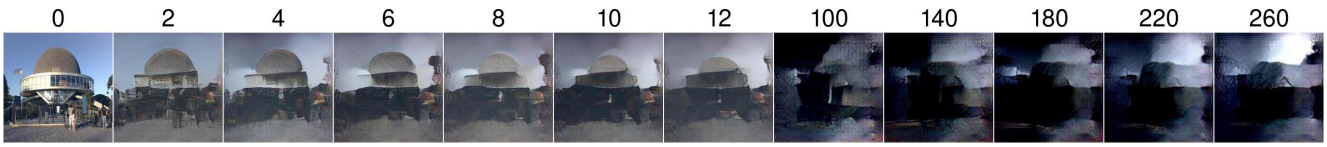
(c) PPGN-h: Generator and a multi-layer perceptron DAE model of $p(h)$

(d) Joint PPGN-h: joint Generator and DAE



(e) Noiseless Joint PPGN-h: joint Generator and AE

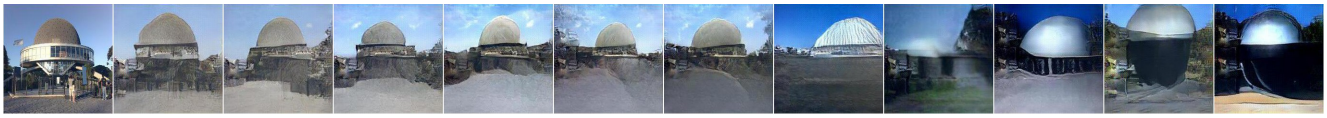
Figure S14: Same as Fig. S13, but starting from a random code h (which when pushed through generator network G produces the leftmost images) except for (a) which starts from random images as the sampling operates directly in the pixel space. All of our qualitative conclusions are the same as for Fig. S13. Note that the samples in (b) appear slightly worse than the images reported in Nguyen et al. [37]. The reason is that in the new framework introduced in this paper we perform an infinitely long sampling chain at a constant learning rate to travel from one mode to another in the space. In contrast, the set of parameters (including the number of iterations, an L_2 decay on code h, and a learning rate decay) in Nguyen et al. [37] is carefully tuned for the best image quality, but does not allow for a long sampling chain (Fig. 2).



(a) Very large noise ($\epsilon_3 = 10^{-1}$)

(b) Large noise ($\epsilon_3 = 10^{-5}$)

(c) Medium noise ($\epsilon_3 = 10^{-9}$)

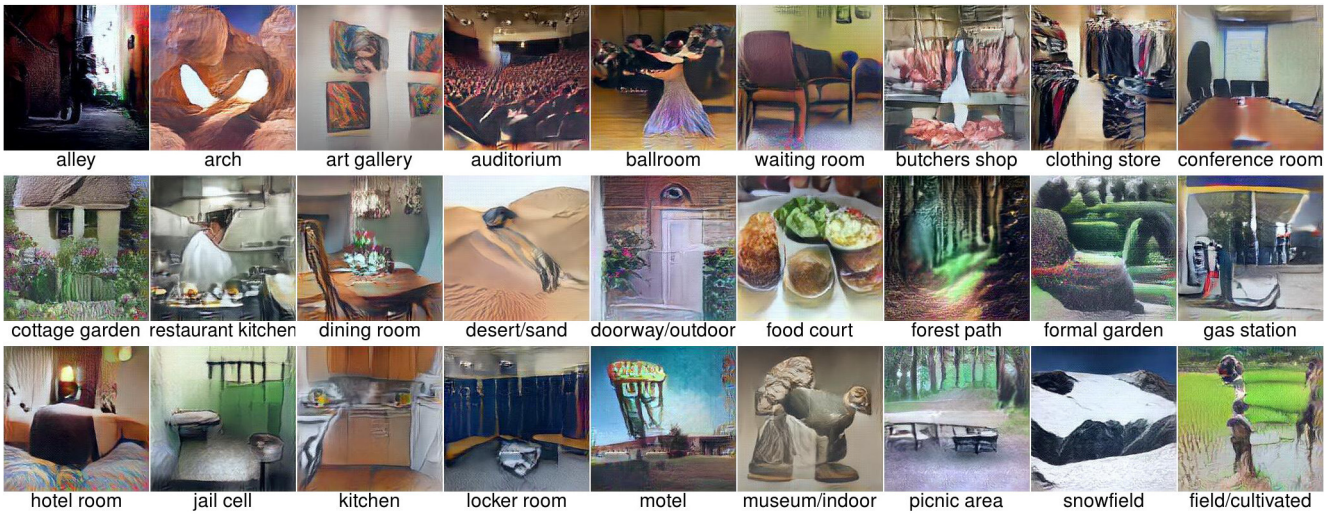


(d) Small noise ($\epsilon_3 = 10^{-13}$)

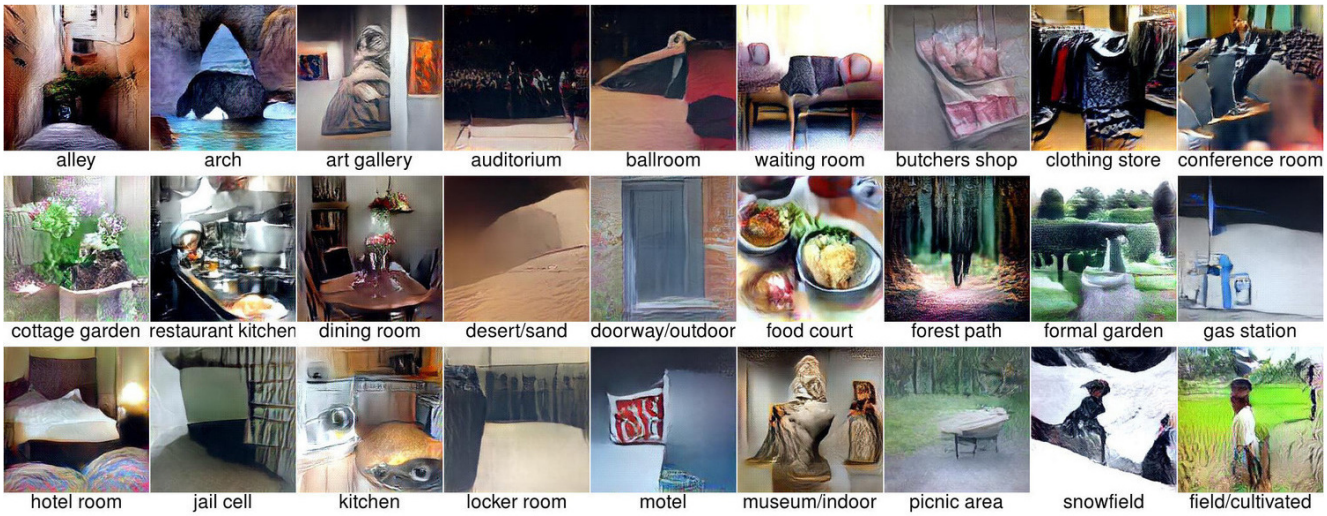
(e) Infinitesimal noise ($\epsilon_3 = 10^{-17}$)

Figure S15: Sampling chains with the *noiseless* PPGN model starting from the code of a real image (*left*) and conditioning on class "planetarium" i.e. $(\epsilon_1, \epsilon_2) = (1, 10^{-5})$ for different noise levels ϵ_3. The sampling step numbers are shown on top. Samples are better with a tiny amount of noise (e) than with larger noise levels (a,b,c & d), so we chose that as our default noise level for all sampling experiments with the Noiseless Joint PPGN-h variant (Sec. 3.5). These results suggest that a certain amount of noise added to the DAE during training might help the chain mix faster, and thus partly motivated our experiment in Sec. 3.4.

Figure S16: The default generator network G in our experiments (used in Sections 3.3 & 3.5) was trained to reconstruct images from compressed fc6 features extracted from AlexNet classification network [26] with three different losses: adversarial loss [14], feature matching loss [9], and image reconstruction loss (more training details are in Sec. S9.4). Here, we test how robust G is to Gaussian noise added to an input code h of a real image. We sweep across different levels of Gaussian noise $N(0, \sigma^2)$ with $\sigma = \{1\%, 10\%, 20\%, 30\%, 40\%\}$ of the mean fc6 activation computed by the activations of validation set images. We observed that G is robust to even a large amount of noise up to $\sigma = 20\%$ despite being trained without explicit regularizations (i.e. with noise [57] or a contractive penalty [44]).



(a) Samples produced by the DGN-AM method [37]



(b) Samples produced by PPGN (the new model proposed in this paper)

Figure S17: A comparison of images produced by the DGN-AM method [37] (top) and the new PPGN method we introduce in this paper (bottom). Both methods synthesize images conditioned on classes of scene images that the generator was never trained on. Specifically, the condition model $p(y|x)$ is AlexNet trained to classify 205 categories of scene images from the MIT Places dataset [65], while the prior model $p(x)$ is trained to generate ImageNet images. Despite having a strong, learned prior (represented by a DAE trained on ImageNet images), the PPGN (like DGN-AM) produces high-quality images for an unseen dataset.

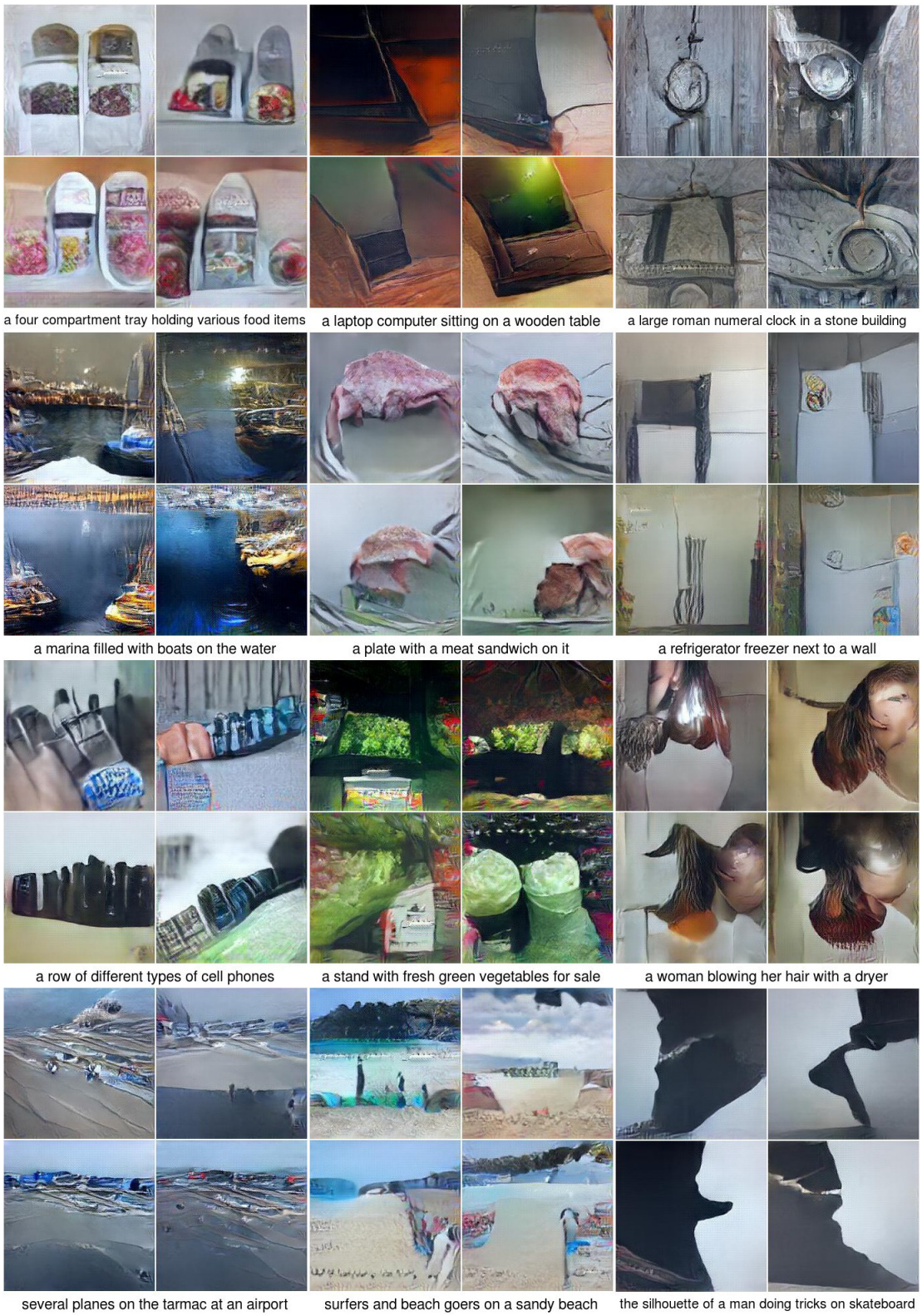


Figure S18: The model can be given a text description of an image and asked to generate the described image. Technically, that involves the same PPGN model, but conditioning on a caption instead of a class. Here the condition network is the LRCN image captioning model from Donahue et al. [8], which can generate reasonable captions for images. For each caption, we show 4 images synthesized by starting from random initializations. Note that it reasonably draws "tarmac", "silhouette" or "woman" although these are not categories in the ImageNet dataset [7].

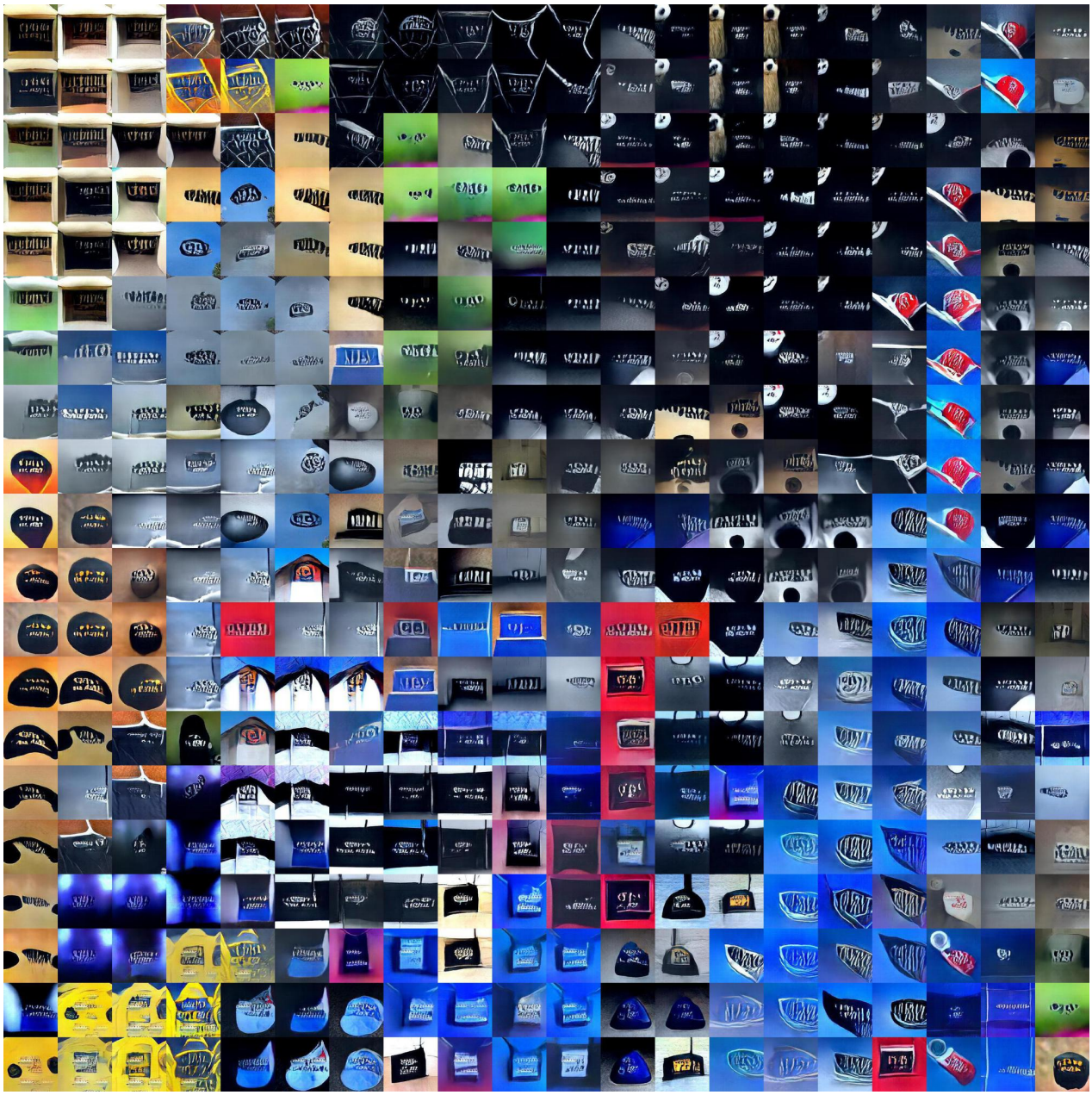


Figure S19: PPGNs have the ability to perform 'multifaceted feature visualization,' meaning they can generate the set of inputs that activate a given hidden neuron, which improves our ability to understand what type of features that neuron has learned to detect [40, 64]. To demonstrate that capability, instead of conditioning on a class from the dataset, here we generate images conditioned on a hidden neuron previously identified as detecting text [64]: neuron number 243 in layer conv5 of the AlexNet DNN. We run 10 sampling chains, each for 200 steps, to produce 2000 samples, and filtering out samples with a softmax probability (taken over all depth columns at the same spatial location) of less than 0.97. From the remaining, we randomly pick 400 samples and plot them in a grid t-SNE [56]. These images can be interpreted as the preferred stimuli for this text detector unit [37]. The diversity of samples is substantially improved vs. previous methods [40, 37, 64] uncovering different facets that the neuron has learned to detect. In other words, while previous methods produced one type of sample per neuron [64, 37], or lower quality samples with greater diversity [40], PPGNs produce a diversity of high-quality samples, and thus represent the state of the art in multifaceted feature visualization.

Figure S20: This figure shows the same thing as Fig. S19, except in this case for a hidden neuron previously identified to be a face detector [64] neuron (number 196) in layer conv5 of the AlexNet DNN. One can see different types of faces that the neuron has learned to detect, including everything from dog faces (top row) to masks (left columns), and human faces from different angles, against different backgrounds, with and without hats, and with different shirt colors. Interestingly, we see that certain types of houses with windows that resemble eye sockets also activate this neuron (center left). This demonstrates the value of PPGNs as tools to identify unexpected facets, which aids in understanding the network, predicting some failure cases, and providing hints for how the network may be improved.

(a) Snail

(b) Studio couch

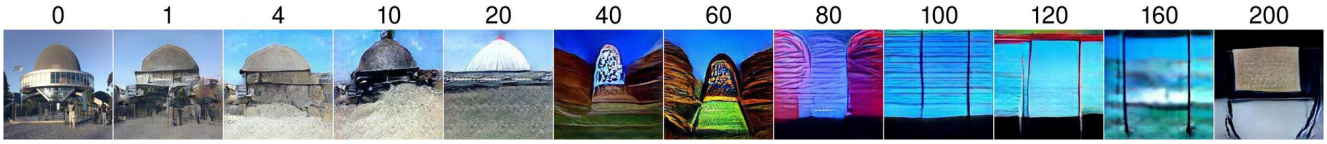
(c) Harvester

(d) Pomegranate

(e) Tape player

Figure S21: To evaluate the diversity of images within a class, here we show randomly chosen images from 5 different classes (a class label shown below each panel). Each image is the last sample produced from a 200-iteration sampling chain starting from a random initialization. The PPGN model is described in Sec. 3.5. We chose this method because it is simple, intuitive and straightforward to interpret and compare to other image generative models that do not require MCMC sampling. Another method to produce samples is to run a long sampling chain and record images that are produced at every sampling step; however, as done in Fig. S11, that method would require additional processing (including heuristic filtering and clustering) to obtain a set of different images because a well-known issue with MCMC sampling is that mixing is slow i.e. subsequent samples are often correlated. Note that one could obtain a larger diversity by running each sampling chain with a different set of parameters (ϵ multipliers in Eq. 5); however, here we use the same set of parameters as previously reported in Sec. 3.5 for simplicity and reproducibility.

Figure S22: For a fair evaluation of the image quality produced by PPGN, here we show one randomly chosen sample for each of 120 random ImageNet classes (neither cherry-picked). Each image shown is the last sample produced from a 200-iteration sampling chain starting from a random initialization. The PPGN model is described in Sec. 3.5.



(a) $\epsilon_1 = 10^{-1}$

(b) $\epsilon_1 = 10^{-3}$

(c) $\epsilon_1 = 10^{-5}$ (Noiseless Joint PPGN-h)

(d) $\epsilon_1 = 10^{-7}$

(e) $\epsilon_1 = 10^{-11}$

(f) $\epsilon_1 = 0$ (no contribution from the prior)

Figure S23: To evaluate the effect of the prior term ϵ_1 in the sampling, here we sweep across different values of this multiplier. We sample from the Noiseless Joint PPGN-h model (Sec. 3.5) starting from the code of a real image (*left*) and conditioning on class "planetarium" with a fixed amount of noise i.e. $(\epsilon_2, \epsilon_3) = (1, 10^{-17})$ for different ϵ_1 values. The sampling step numbers are shown on top. Without the learned prior $p(h)$ (f), we arrive at the DGN-AM treatment results where the chain does not mix at all (the same result as in Fig. S13b). Increasing ϵ_1 up to a small value (c-e) results in a chain that mixes faster, from one planetarium to another. When the contribution of the prior is too high which overwrites the class gradients, yielding a chain that mixes from one mode of generic images to another (a). We empirically chose $\epsilon_1 = 10^{-5}$ as the default value for the Noiseless Joint PPGN-h experiments in this paper as it produces the best image quality and diversity for many classes.

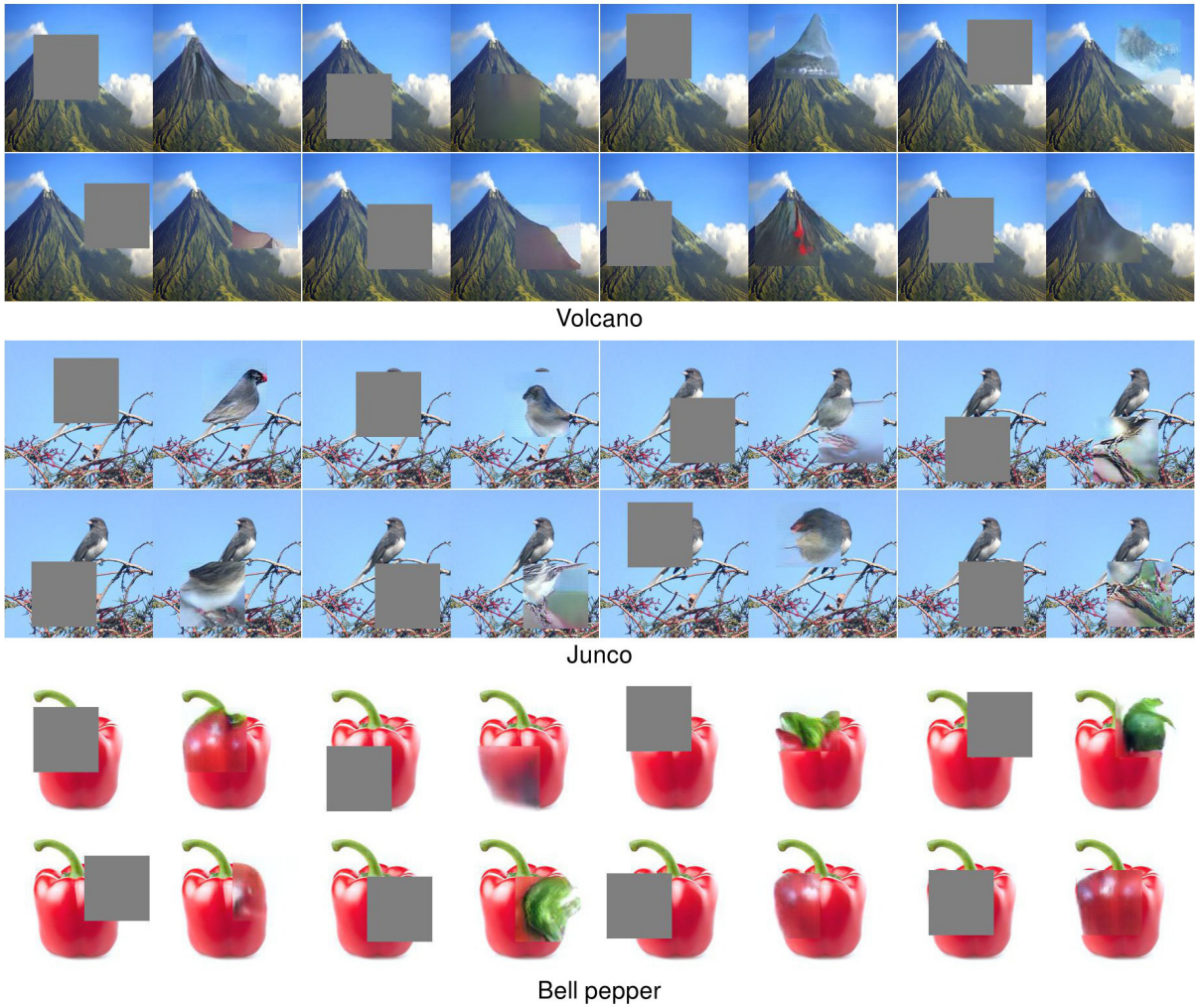
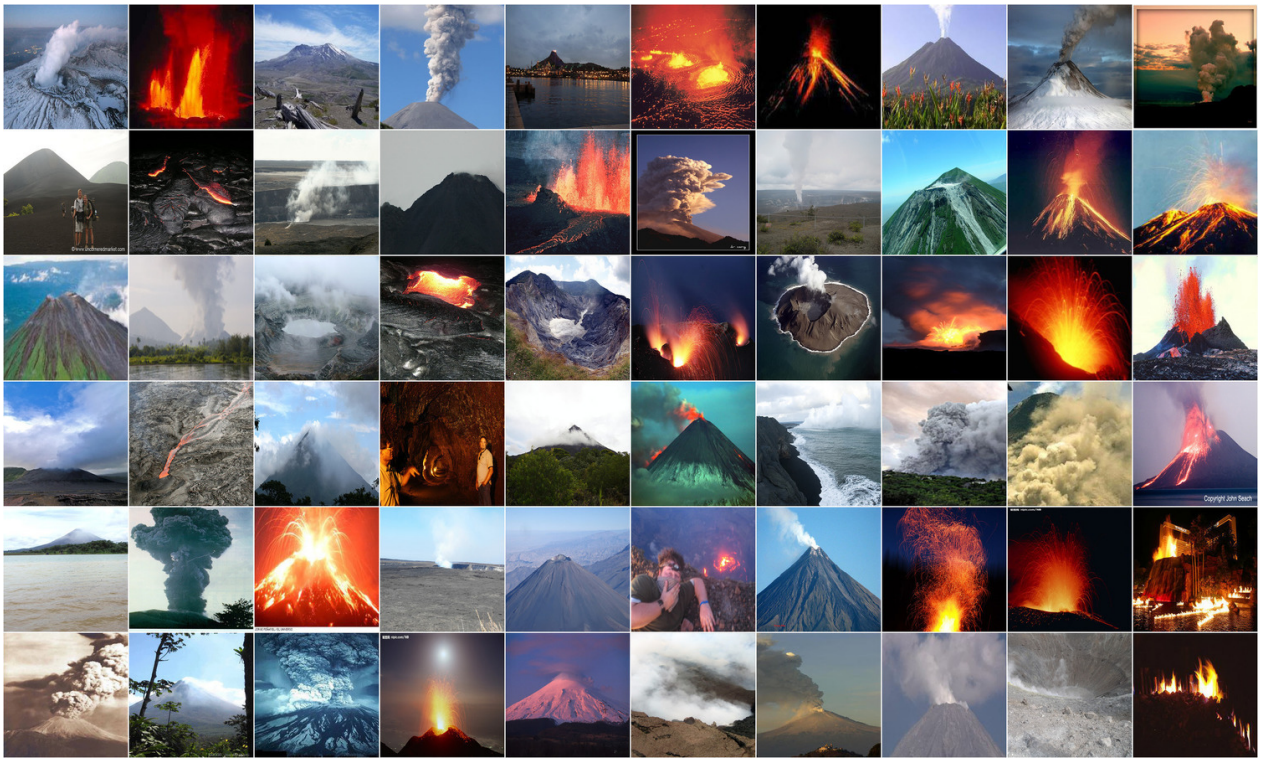


Figure S24: To test the ability of PPGNs to perform "inpainting", we randomly mask out a 100×100 patch in a real image, and perform class-conditional image sampling via PPGN-context (described in Sec. 4.2) to fill in missing pixels. In addition to conditioning on a specific class (here, "volcano", "junco" and "bell pepper" respectively), we put an additional constraint on the code h that it has to produce an image that matches the context region. PPGN-context performs semantically well in many cases. However, sometimes it does not match the local features of the surrounding regions. The result shows that the class-conditional image model does understand the semantics of images.



(a) 60 training set images randomly taken from the “volcano” class.

.

(b) 60 PPGN samples randomly selected from 2000 samples, which are produced from 10 200-step sampling chains.

Figure S25: To evaluate how well PPGN samples represent the training set images, we compare 60 real images (top) vs. 60 PPGN generated images (bottom). All images are randomly selected. While the set of generated images are high-quality, they have a much lower diversity compared to the set of real images. This observation aligns with our quantitative measure in Table S3.