Compositional Pattern Producing GAN

Luke Metz
Google Brain
lmetz@google.com

Ishaan Gulrajani
Google Brain
igul@google.com

Introduction In this work, we explore generative models of images as tools for creative work. Typically generative models have the goal of faithfully approximating a data distribution. In the creative setting, we also want the generating process to be manually *controllable* in ways that let us achieve visually interesting images, even if this makes them less similar to the original data.

Generative Adversarial Networks (GANs) are a class of methods capable of generating photorealistic images [7, 16, 4]. They posit a generative model and use a discriminator network to distinguish generated samples from real data. The generator uses gradients from the discriminator to learn how to generate realistic data. Besides generating photorealistic images, GANs have the useful property that they permit any generator structure which allows sampling, which we leverage in this work.

Typically, the generator of a GAN transforms a latent vector z into an image at a fixed resolution using strided transposed convolution [14]. There has been work on generators which output images at multiple resolutions by generating at a low resolution and progressively upsampling [6, 3, 16]. In this work, we propose an alternate generator parameterization built on Compositional Pattern Producing Networks (CPPN) [15] which is capable of directly generating images at any resolution, including resolutions other than the ones it was trained at.

A CPPN can be seen as a neural network that takes input spatial coordinates x, y and outputs a color at that location. To generate an image, we use a rendering process similar to how cameras capture images: we project a grid of pixels over the field to be captured and compute the integral of the area enclosed by each pixel. In this work, we are interested in modeling a distribution of images rather than a single image, so we condition our CPPN on a latent variable z. This work is an extension of ideas from [1, 2] where both VAE and GAN CPPN models were explored. We extend their work with recent GAN innovations and show further applications of the technique.

Using CPPNs for image generation in this way has a number of benefits. First, these models are able to generate the same image at arbitrary resolutions because the resolution is entirely a property of the rendering process and not the model. Second, we are no longer bound to generating an image in its entirety; instead, we can render arbitrary crops of an image (again at arbitrary resolution). Finally, parameterizing the generator in this way enables easy customization of the generated images: we observe that the choice of network architecture in CPPNs has interpretable effects on properties of the resulting images when rendered at higher resolutions.

Most prior GAN work tries to learn a generator which approximates a data distribution. This produces realistic images, but our goal is to produce images that are visually interesting, sometimes in ways that are unrealistic. To achieve this, we simultaneously train the generator to match multiple (potentially conflicting) image distributions at different scales. This way, we can control the structure of our generated images independently at each scale.

Effect of different generator parameterizations To demonstrate the effect of the generator parameterization on the generated images, we train multiple different generated architectures and present images in Figure 1. We see clear differences in the high-resolution images but not the low-resolution ones.

Multi-resolution training Next, we explore constraints from the discriminator. From the ImageNet [5] dataset, we construct two datasets which we train the generator to match simultaneously: the full images resized to 32×32, and 32×32 crops of 64×4-resized images. The generator must learn to generate realistic structure at both scale levels to succeed. We present results in Figure 2.

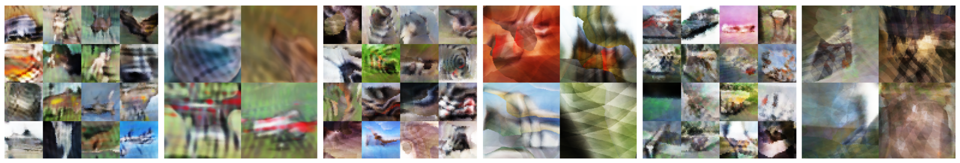

Figure 1: Models trained on 32x32 renderings of CIFAR-10 [12] images. Shown are the native resolution as well as rendering at 128x128. Each block consists of a different projection activation function used. Left: Sine waves. Center: Triangle waves. Right: Square waves. Sine waves yield smoother, higher resolution images whereas square and triangle yield harder edges. Images rendered at training resolution look similar as they are minimizing the same adversarial divergence.

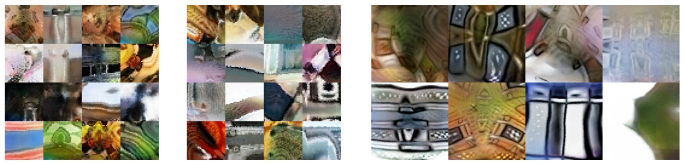

Figure 2: Left: 32×32 rendering matching the distribution from ImageNet samples. Center: 32×32 renderings matching crops taken from a 64×64 ImageNet samples. These samples represent higher detail textures. Right: 128×128 renderings that merge the global structure and the more fine grain texture information.

Training against different distributions at different scales In addition to matching the same distribution at different resolutions, we can also match very different distributions at different scales to guide the output. First, we match the full canvas, a box of size $(1, 1)$, to 32×32 CIFAR-10 images. Second, we match crops of size $(0.1, 0.1)$, to an artificial dataset consisting of randomly generated circles. The generator is shared and minimizes a weighted sum of these two divergences. In this test, the generator is also augmented with higher frequency sin and cos features squared to make learning circular features easier. The results are in figure 3. We observe circular patterns in the high resolution images.

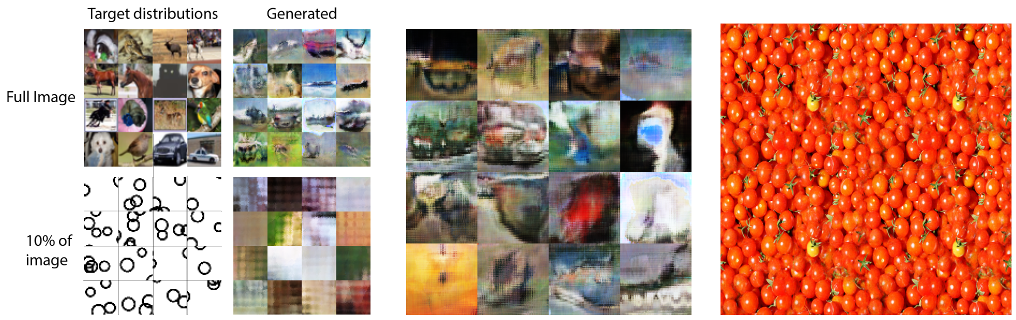

Figure 3: Left-Top: Training images and samples rendered at 32×32 from full image crops. Generated images are smooth and globally consistent. Left-Bottom: Training images for texture distribution and 32×32 renders of generated patches. While still dissimilar, the generated samples contain circular patterns that more closely match the target texture distribution. Center: 128×128 renders. These renders show the global structure as well as how the circular patterns are integrated into this structure to create an interesting resulting image. Right: template from which we picked patches of examples. Right: A texture generated from a picture of tomatoes tiled 2×2 times.

Texture synthesis A common task in compute graphics is texture synthesis: given an image, produce a tileable image which resembles the original image in any local region. Enforcing this constraint is easy given the parameterization of our CPPN network. We can simply use as input x, y positions mapped through a periodic function. Taking this approach and training on arbitrary crops lets us generate a tile-able textures. Results are in figure 3.

1 Acknowledgements

We would like to thank David Berthelot, Brian Cheung, Jascha Sohl-Dickstein, Douglas Eck, David Ha, and the rest of the Brain team, as well as Alec Radford, for providing useful feedback on this work. Finally, we would like to thank Pluma [13] for creating the image of the cherry tomatoes which we used as training data for our texture synthesis experiment.

References

[1] Generating large images from latent vectors. `http://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/`. Accessed: 2017-11-17.

[2] Generating large images from latent vectors - part two. `http://blog.otoro.net/2016/06/02/generating-large-images-from-latent-vectors-part-two/`. Accessed: 2017-10-26.

[3] Work in progress: Portraits of imaginary people. `https://mtyka.github.io/machine/learning/2017/06/06/highres-gan-faces.html`.

[4] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a Laplacian pyramid of adversarial networks. *NIPS*, 2015.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[8] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[9] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *CoRR*, abs/1702.03275, 2017.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[12] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[13] Pluma. File cherry tomatoesjpg. `https://commons.wikimedia.org/wiki/File:Cherry_tomatoes.jpg`. Accessed: 2017-10-31.

[14] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[15] Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

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

Appendix

A Method

Much like in standard GAN training, we attempt to minimize an objective defined as a min max game. A discriminator network estimates some divergence between two distributions and a generator minimizes this estimate. In this work, the distributions in question are images formed by a grid of pixels and we minimize the WGAN-GP objective from [8]. We had difficulty training our model with the standard GAN objective proposed in [7]. In the next two sections we define the architecture of our generator and discriminator.

A.1 CPPN Generator

We define a CPPN function C that takes as input two coordinates x, y and a latent vector z and outputs an RGB color: $C(x, y, z) :: (\mathcal{R}, \mathcal{R}, \mathcal{R}^{N_z}) \to \mathcal{R}^3$. For simplicity, we assume $x, y \in (0, 1)$. The generated image G^R for a given resolution R is defined pixel-wise by a "rendering" function:

$$G^R_{ij}(z) = \int_{\frac{i}{R}}^{\frac{i+1}{R}} \int_{\frac{j}{R}}^{\frac{j+1}{R}} C(x, y, z) dx dy \tag{1}$$

A similar expression can be written for sampling arbitrary image crops. We approximate the integrals with Monte Carlo estimates using a small number of samples for each pixel.

The form of C is similar to an MLP, but in addition to the activation functions typically used in neural networks we also use trigonometric functions (optionally with learned frequencies), differentiable texture lookups, x^2, and $\text{abs}(x)$. This parameterization yields an interpretability of sorts. For example, using square or triangle waves in the generator biases it towards geometric imagery with sharp edges, and using high-frequency periodic nonlinearities creates images with high-frequency patterns.

We find that careful normalization in this model is critical for successful training. Straightforward use of batch normalization [10] is problematic because during training, the inputs x, y, z to our generator within a given minibatch are not i.i.d. (for example, rendering a 32×32 image requires evaluating the generator 1024 times with the same z value in a single minibatch). We use batch renormalization [9] which resolves this problem and makes optimization easier.

A.2 Discriminator

We use discriminators based on DCGAN [14] but replace batch normalization with layer normalization as in [8]. GANs generators are typically trained to approximate a single data distribution, but in this work we use multiple discriminators (trained on different, sometimes incompatible, distributions) to independently control the structure of our generated images at different scales. For example, we can enforce that our generator generates images which look like natural scenes at low resolution but look like a pattern of circles at high resolutions. To do this, we train one discriminator on natural scenes and large-scale generations and another on circle patterns and small-scale generations and train the generator to minimize both discriminators' value functions.

B Optimization challenges

We found WGAN-GP [8] and Adam [11] made stable training of our model substantially easier. Additionally, when rendering we must approximate an integral over color in a pixel. When this pixel itself contains high frequency texture, these estimates will not only be wrong, but the gradients will be high variance, which makes learning hard.

Due to these optimization issues, we are unable to reach the same visual quality of existing convolution transposed models even after millions of steps of training. We believe it is possible to reach this level of performance but modifications to the underlying architectures will be needed.