

---

# Texture Synthesis with Spatial Generative Adversarial Networks

---

Nikolay Jetchev\*

Urs Bergmann\*

Roland Vollgraf

Zalando Research

{nikolay.jetchev,urs.bergmann,roland.vollgraf}@zalando.de

## 1 Abstract

Generative adversarial networks (GANs) [7] are a recent approach to train generative models of data, which have been shown to work particularly well on image data. In the current paper we introduce a new model for texture synthesis based on GAN learning. By extending the input noise distribution space from a single vector to a whole spatial tensor, we create an architecture with properties well suited to the task of texture synthesis, which we call spatial GAN (SGAN). To our knowledge, this is the first successful completely data-driven texture synthesis method based on GANs.

Our method has the following features which make it a state of the art algorithm for texture synthesis: high image quality of the generated textures, very high scalability w.r.t. the output texture size, fast real-time forward generation, the ability to fuse multiple diverse source images in complex textures. To illustrate these capabilities we present multiple experiments with different classes of texture images and use cases. We also discuss some limitations of our method with respect to the types of texture images it can synthesize, and compare it to other neural techniques for texture generation.

## 2 Introduction

### 2.1 Background: texture synthesis

A texture can be defined as an image containing repeating patterns with some amount of randomness. More formally, a texture is a realization of a stationary ergodic stochastic process [6]. The goal of visual texture analysis is to infer a generating process from an example texture, which then allows to generate arbitrarily many new samples of that texture - hence performing texture synthesis. Success in that task is judged primarily by visual quality and closeness to the original texture as estimated by human observers, but also by other criteria which may be application specific [17], e.g. the speed of analysis and synthesis, ability to generate diverse textures of arbitrary size, the ability to create smoothly morphing textures.

Approaches to do that fall in two broad categories. Non-parametric techniques resample pixels [3] or whole patches [2] from example textures, effectively randomizing the input texture in ways that preserve its visual perceptual properties. They can produce high quality images, but have two drawbacks: (i) they do not "learn" any models of the textures of interest but just reorder the input texture using local similarity, and (ii) they can be time-consuming when large textures should be synthesized because of all the search routines involved. There are methods to accelerate example-based techniques [17], but this requires complicated algorithms.

---

\*These authors contributed equally to this work.

Our source code is available at [https://github.com/zalandoresearch/spatial\\_gan](https://github.com/zalandoresearch/spatial_gan)



Figure 1: Learning a texture from a satellite image of Barcelona of size 1371x647 pixels. We visualize a 647x647 pixel subset of the training and generated images, and for comparison draw the 125 pixel SGAN5 receptive field (yellow box, top-left corner of the left image). Our adversarial approach to texture synthesis SGAN (with 5 layers) generates a city texture of higher visual quality (e.g. clearly visible city streets) than the output of the method of Gatys [5].

The second category of texture synthesis methods is based on matching statistical properties or descriptors of images. Texture synthesis is then equivalent to finding an image with similar descriptors, usually by solving an optimization problem in the space of image pixels. The work of Portilla and Simoncelli [12] is a notable example of this approach, which yields very good image quality for some textures. Carefully designed descriptors over spatial locations, orientations, and scales are used to represent statistics over target textures.

Gatys et al. [5] present a more data driven parametric approach to allow generation of high quality textures over a variety of natural images. Using filter correlations in different layers of the convolutional networks – trained discriminatively on large natural image collections – results in a powerful technique that nicely captures expressive image statistics. However, creating a single output texture requires solving an optimization problem with iterative backpropagation, which is costly – in time and memory.

Recent papers [16, 9] deal with that problem and train feed-forward convolutional networks in order to speed up the texture synthesis approach of [5]. Instead of doing the costly optimization of the output image pixels, they utilize powerful deep learning networks that are trained to produce images minimizing the loss. A separate network is trained for each texture of interest and can then quickly create an image with the desired statistics in one forward pass.

A generative approach to texture synthesis [15] uses a recurrent neural network to learn the pixel probabilities and statistical dependencies of natural images. They obtain good texture quality on many image types, but their method is computationally expensive and this makes it less practical for texture generation in cases where size and speed matter.

## 2.2 An adversarial approach to texture synthesis

We will present a novel class of generative parametric models for texture synthesis, using a fully convolutional architecture trained employing an adversarial criterion.

As introduced in [7], GANs train a generative model  $G$  that captures the data distribution and a discriminator  $D$  that attempts to separate generated from training data. Radford et al. [13] improved the GAN architecture by using deep convolutional layers with (fractional) stride [14] and batch normalization [8]. Overall, GANs are powerful enough to generate natural looking images of high quality (but low pixel resolution) that can confuse even human observers [4].

However, in GANs the size of the output image (e.g. 64x64 pixels) is hard coded into the network architecture. This is a limitation for texture synthesis, where much larger textures and multiple sizes may be required. Laplacian pyramids have been used to generate images of increasing size [4], gradually adding more details to the images with stacked conditional GANs. However, that technique

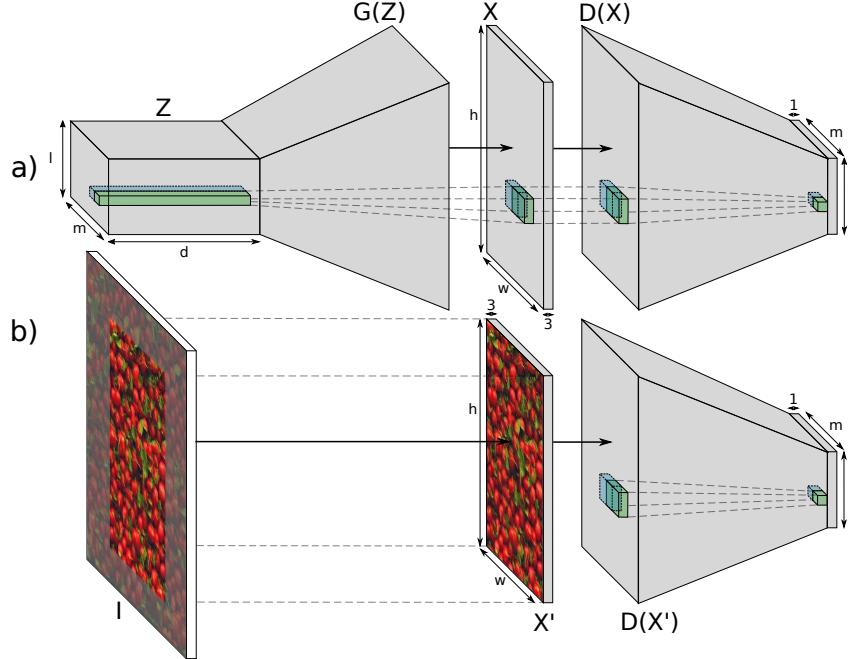


Figure 2: Spatial GAN model (SGAN): a generator  $G$  transforms a spatial noise array  $Z \in \mathbb{R}^{l \times m \times d}$  into an RGB image  $X \in \mathbb{R}^{h \times w \times 3}$  via a stack of fractionally strided convolution layers. The discriminator  $D$  is fed either a generated image ( $X$  - case (a)) or a rectangular patch extracted from an image  $I$  of a database ( $X'$  - case (b)). It uses a stack of convolutional layers to output a 2D field of probabilities for fake/real ( $X$  vs.  $X'$ ) data. The detailed architecture of both generator and discriminator (the “funnels”) is akin to [13], but varies in having exclusively convolutional layers and a potentially different number of hidden layers. The subnetwork that projects a single vector of the array  $Z$ , i.e.  $z_{\lambda\mu} \in \mathbb{R}^d$ , to the generated output image is equivalent to a standard GAN (see e.g. the green blocks in the figure). However, non-overlapping vectors have overlapping projective fields in the output. In this view, SGAN is a convolutional roll-out of GANs.

is still limited in the output image sizes it can handle because it needs to train GAN models with increasing complexity for each scale in the image pyramid. The scale levels must also be specified in advance, so the method cannot create output of arbitrary size.

In our work we will input images of textures (possibly of high pixel resolution) as the data distribution that the GAN must learn. However, we will modify the DCGAN architecture [13] to allow for scalability and the ability to create any desired output texture size by employing a purely convolutional architecture without any fully connected layers. The SGAN architecture is especially well suited for texture synthesis of arbitrary large size, a novel application of adversarial methods.

In the experiments in Section 4 we will examine these points in detail. In the next section we describe the spatial GAN architecture.

### 3 The SGAN Model

The key idea behind Generative Adversarial Networks [7] is to simultaneously learn a generator network  $G$  and a discriminator network  $D$ . The task of  $G(z)$  is to map a randomly sampled vector  $z \in \mathbb{R}^d$  from a prior distribution  $p_z(z)$  to a sample  $X \in \mathbb{R}^{h \times w \times 3}$  in the image data space. The discriminator  $D(X)$  outputs a scalar representing the probability that  $X$  is from real training data and not from the generator  $G$ . Learning is motivated from game theory: the generator  $G$  tries to fool the discriminator into classifying generated data as real one, while the discriminator tries to discriminate real from generated data. As both  $G$  and  $D$  adapt over time,  $G$  generates data that gets close to the input data distribution.

The SGAN generalizes the generator  $G(Z)$  to map a tensor  $Z \in \mathbb{R}^{l \times m \times d}$  to an image  $X \in \mathbb{R}^{h \times w \times 3}$ , see Figure 2. We call  $l$  and  $m$  the spatial dimensions and  $d$  the number of channels. Like in GANs,  $Z$  is sampled from a (simple) prior distribution:  $Z \sim p_Z(Z)$ . We restricted our experiments to having each slice of  $Z$  at position  $\lambda$  and  $\mu$ , i.e.  $z_{\lambda\mu} \in \mathbb{R}^d$ , independently sampled from  $p_Z(z)$ , where  $\lambda, \mu \in \mathbb{N}$  with  $1 \leq \lambda \leq l$  and  $1 \leq \mu \leq m$ . Note that the architecture of the GAN puts a constraint on the dimensions  $l, m, h, w$  – if we have a network with  $k$  convolution layers with stride  $\frac{1}{2}$  and same zero padding (see e.g. [1]) then  $\frac{h}{l} = \frac{w}{m} = 2^k$ .

Similarly to the way we extended the generator, the discriminator  $D$  maps to a two-dimensional field of size  $l \times m$  containing probabilities that indicate if an input  $X$  (or  $X'$ ) is real or generated. In order to apply the SGAN to a target texture  $I$ , we need to use  $I$  to define the true data distribution  $p_{\text{data}}$ . To this end we extract rectangular patches  $X' \in \mathbb{R}^{h \times w \times 3}$  from the image  $I$  at random positions. We chose  $X'$  to be of the same size as the samples of the generator  $X = G(Z)$  – otherwise GAN training failed in our experiments. For the same reason we chose symmetric architectures for  $G$  and  $D$ , i.e.  $D(G(Z))$  has the same spatial dimensions as  $Z$ .

Both the generator  $G(Z)$  and the discriminator  $D(X)$  are derived from the architecture of [13]. In contrast to the original architecture however, spatial GANs forgo any fully connected layers – the networks are purely convolutional. This allows for manipulation of the spatial dimensions (i.e.  $l$  and  $m$ ) without any changes in the weights. Hence, a network trained to generate small images is able to generate a much larger image during deployment, which matches the local statistics of the training data.

We optimize the discriminator (and the generator) simultaneously over all spatial dimensions:

$$\begin{aligned} \min_G \max_D V(D, G) = & \frac{1}{lm} \sum_{\lambda=1}^l \sum_{\mu=1}^m \mathbb{E}_{Z \sim p_Z(Z)} [\log (1 - D_{\lambda\mu}(G(Z)))] \\ & + \frac{1}{lm} \sum_{\lambda=1}^l \sum_{\mu=1}^m \mathbb{E}_{X' \sim p_{\text{data}}(X)} [\log D_{\lambda\mu}(X')] \end{aligned} \quad (1)$$

In this formula the first row corresponds to Figure 2(a), and the second row to Figure 2(b). In practice, it is helpful to apply the trick of [7] and minimize  $-\log(D(G(Z)))$  instead of  $\log(1 - D(G(Z)))$ .

Note that the model describes a stochastic process over the image space. In particular, as the generator  $G$  is purely convolutional and each  $z_{\lambda\mu}$  is identically distributed and independent of its location  $\lambda$  and  $\mu$ , the generated data is translation-invariant. Hence the process is *stationary* with respect to translations.

To show that the process is also strong mixing, we first need to define the projective field (PF) of a spatial patch  $\hat{Z}$  of  $Z$  as the smallest patch  $\hat{X}$  of the image  $X$  which contains all affected pixels of  $X = G(Z)$  under all possible changes of  $\hat{Z}$ . In full analogy, we refer to the receptive field (RF) of a patch in  $D(X)$  as the corresponding minimal patch in  $X$  which affects it. Assume then two non-overlapping patches from the generated data,  $A$  and  $B$ . Additionally, take their respective projective fields  $Z_A, Z_B$  to be non-overlapping – this can be always achieved as projective fields are finite, but the array  $Z$  can be made arbitrarily large. The generated data in  $A$  and  $B$  is then independently generated. The process is hence *strong mixing* (with the length scale of the projective field), which implies it is also *ergodic*.

## 4 Experiments

### 4.1 Architectural details and speed

For the following experiments, we used an architecture close to the DCGAN setup [13]: convolutional layers with stride  $\frac{1}{2}$  in the generator, convolutional layers with stride 2 in the discriminator, kernels of size 5x5 and zero padding. We used a uniform distribution for  $p_Z(z)$  with support in  $[-1, 1]$ . Depending on the size and structure of the texture to be learned, we used networks of different complexity, see Table 1. We used networks with identical depths in  $G$  and  $D$ . The sizes of filter banks of  $D$  were chosen to be in reverse to those of  $G$ , yielding more channels for smaller spatial

	SGAN4	SGAN5	SGAN6
$Z$ channel number $d$	20	50	100
$r = h/l = w/m =$	16	32	64
PF/RF size	61	125	253
generator $G$ filters	256-128-64	512-256-128-64	1024-512-256-128-64
discriminator $D$ filters	64-128-256	64-128-256-512	64-128-256-512-1024

Table 1: Details of the SGAN architecture with 4,5 or 6 layers. The calculation of the projective and receptive field (PF and RF) sizes is examined in detail in Appendix I.

	SGAN4	SGAN5	TextureNet	Gatys
256x256 px	.005s	.006s	0.020s	10s
512x512 px	.013s	.019s	-	-
1024x1024 px	.047s	.07s	-	-
2048x2048 px	.178s	.269s	-	-

Table 2: Time required for generating an output texture of certain size. The SGAN architecture is faster than both TextureNet [16] and Gatys [5]. The time costs per calculated pixel scale sublinearly.

representations. We denote with SGAN $x$  that we have  $x$  layers depth in  $G$  and  $D$ . We applied batch normalization on all layers, except the output layer of  $G$ , the input and output layers of  $D$ . All network weights were initialized as 0-mean Gaussians with  $\sigma = 0.02$ .

We tried different sizes for the image patches  $X$ . Note that the spatial dimensions of  $Z$  and  $X$  are dependent,  $h = rl$  and  $w = rm$ . Both setting  $h = w = 640$  or  $l = m = 4$  and adjusting for the respective depending variables worked similarly well, despite different relative impact of the zero padded boundaries.

Our code was implemented in Theano and tested on an Nvidia Tesla K80 GPU. The texture generation speeds of a trained SGAN with different architectures and image sizes are shown in Table 2. Generation with  $G$  is very fast, as is expected for a single forward pass. Forward pass generation in TextureNet [16] is significantly slower (20ms, according to their publication) than SGAN (5ms) for the 256 pixel resolution, despite the fact that TextureNet uses fewer filters (8 to 40 channels per convolution layer) than we do (64 to 512 filters). The simpler SGAN architecture avoids the multiple scales and join operations of TextureNet, rendering it more computationally efficient. As expected, the method of Gatys [5] is orders of magnitude slower due to the iterative optimization required.

There are initial time costs for training the SGAN on the target textures. For optimization we used ADAM [10] with parameters as in [13] and 32 samples per minibatch. For the simple textures from Section 4.2.1, subjective assessment indicates that generated images are close to their final quality after roughly 10 minutes of training. The more complex textures of Sections 4.2.2 required around 30 minutes of training.

Training times TextureNet [16] requires a few hours per texture. We could not compare this directly with SGAN on the same machine and textures, but we assume that SGAN trains more efficiently than TextureNet because of its simpler architecture.

The exact time and number of iterations required for training SGANs depends on the structure of the target texture. A general problem in GAN training is that it is often required to monitor the results and stop training – as occasionally overtraining may lead to degeneracy or image quality degradation.

## 4.2 Examples of generated textures

### 4.2.1 Single small image

A common way to measure quality of texture synthesis is to visually evaluate the generated samples. A setup with small input textures allows a direct comparison between SGAN and the method of Gatys [5]. For these examples, we used an SGAN with 4 layers. Figure 3 shows our results for textures coming from a stationary and ergodic process. The textures of radishes and stones (top rows)

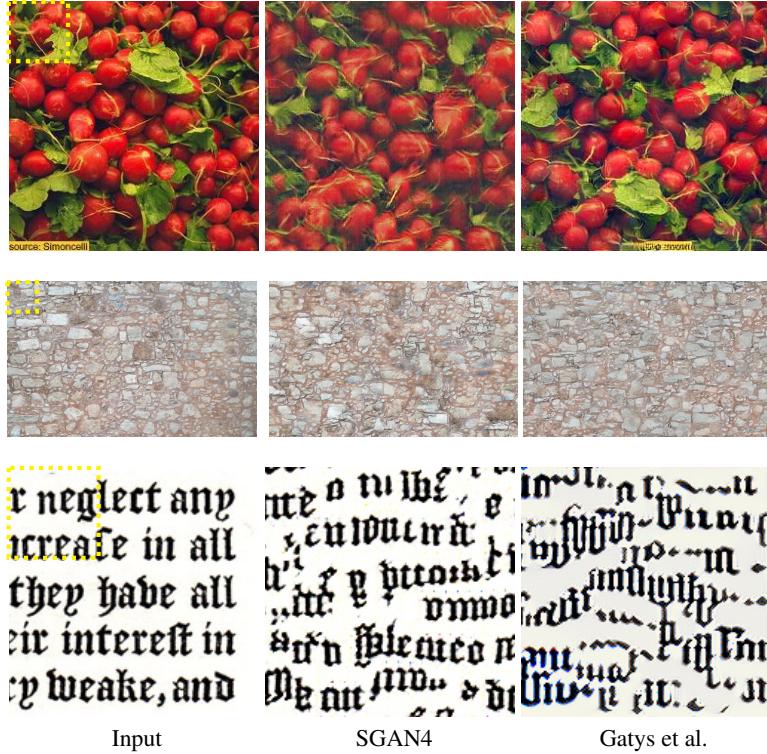


Figure 3: Texture synthesis of small texture images of different size - the results of SGAN are comparable to Gatys [5]. In the top left corner of the images we indicate the receptive field of size 61 pixels of SGAN4 to visually illustrate their size relative to the texture image sizes.

are also mixing, while the letters (bottom row) are mixing horizontally, but not vertically. The texture synthesis of both SGAN and Gatys fail to preserve the row regularity of the source image.

#### 4.2.2 Single large image

Satellite images provide interesting examples of macroscopic structures with texture-like properties. City blocks in particular resemble realizations from a stationary stochastic process: different city blocks have similar visual properties (color, size). Mixing occurs on a characteristic length scale given by the major streets.

Figure 1 shows that our method works better than [5] on satellite images, here concretely a single image of Barcelona.<sup>2</sup> SGAN creates a city-like structure, whereas Gatys’ method generates less structure and detail. We interpret this in the following way: the SGAN is trained specifically on the input image and utilizes all its model power to learn the statistics of this particular texture. Gatys [5] relies on pretrained filters learned on the ImageNet dataset, which generalize well to a large set of images (but apparently not well to satellite imagery) and fails to model salient features of this city image, in particular the street grid orientation.

To indicate the superior quality of SGAN for that texture the spatial auto-correlation (AC) of the original and synthesized textures from Figure 1 are shown on Figure 4. We calculate the AC on whole images of size 1371x647 pixels. The AC of the original and the SGAN5 generation are similar to one another and show clearly the directions of the street grid. In contrast, the AC of Gatys’ texture looks more isotropic than the original, indicating loss of visually important information.

Figure 5 illustrates the effects of different network depths on the SGAN generated outputs. More layers and larger receptive fields as in SGAN6 allow larger structures to be learned and longer streets emerge, i.e., there is less mixing and more regularity at a given scale.

<sup>2</sup>[www.google.com/maps/@41.4033724,2.1551133,292m/data=!3m!1e3](http://www.google.com/maps/@41.4033724,2.1551133,292m/data=!3m!1e3)

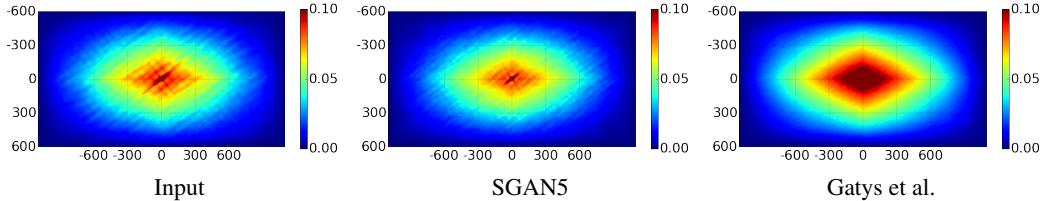


Figure 4: Spatial autocorrelation (AC) of the Barcelona city example from Figure 1: the preferred directions of the city streets are clearly visible in the centre of the AC of the input texture. The AC of the SGAN texture reflects the structure much better than the result of Gatys et al.

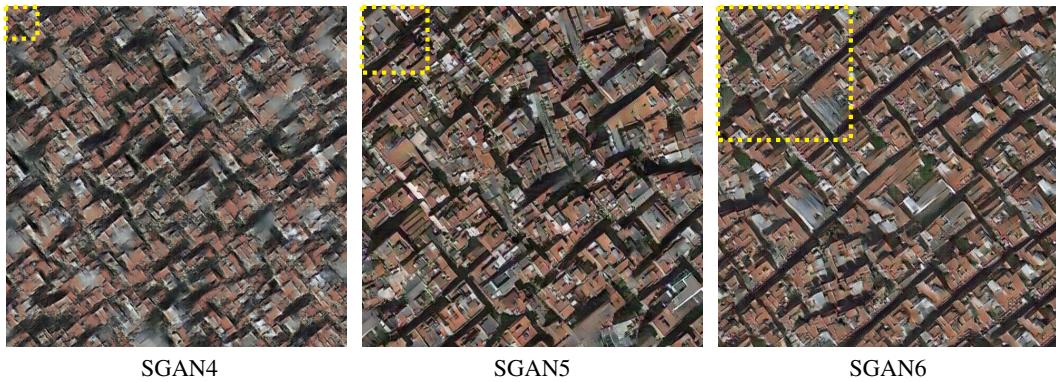


Figure 5: SGAN with  $G, D$  layers of depth 4,5,6 changes the resulting output texture (cropped to size 647x647 pixels). Bigger models exhibit less mixing and generate longer streets.

#### 4.2.3 Composite textures from multiple images

The GAN approach to texture synthesis can combine multiple example texture images in a natural way. We experimented with the flowers dataset<sup>3</sup> containing 8189 images of various flowers, see Figure 6 (a) for examples. We resized each image to 160 pixels in the  $h$ -dimension, while rescaling the  $w$ -dimension to preserve the aspect ratio of the original image. Then we trained an SGAN with 5 layers; each minibatch for training contained 128x128 pixel patches extracted at random positions from randomly selected input flower images. This is an example of a non-ergodic stochastic process since the input images are quite different from one another.

A sample from the generated composite texture is shown in Figure 6 (b). The algorithm generates a variety of natural looking flowers but cannot blend them smoothly since it was trained on a dataset of single flower images. Still, the final result looks aesthetic and such fusion of large image datasets for texture learning has great potential for photo mosaic applications.

In another experiment we learned a texture representing the 5 satellite images shown on Figure 6 (c). The input images depict areas in the Old City of Amsterdam with different prevailing orientations. Figure 6 (d) demonstrates how the generated texture has orientations from all 5 inputs. Although we did not use any data augmentation for training, the angled segments join smoothly and the model learns spatial transitions between the different input textures. Overall, the Amsterdam city segments come from a more ergodic process than the flowers example, but less ergodic than the Barcelona example.

GAN based methods can fuse several images naturally, as they are generative models that capture the statistics of the image patches they're trained with. In contrast, methods with specified image statistical descriptors [12, 5] generate textures that match a single target image closely in these descriptors. Extending these methods to several images is not straight-forward.

<sup>3</sup>[www.robots.ox.ac.uk/~vgg/data/flowers/](http://www.robots.ox.ac.uk/~vgg/data/flowers/)

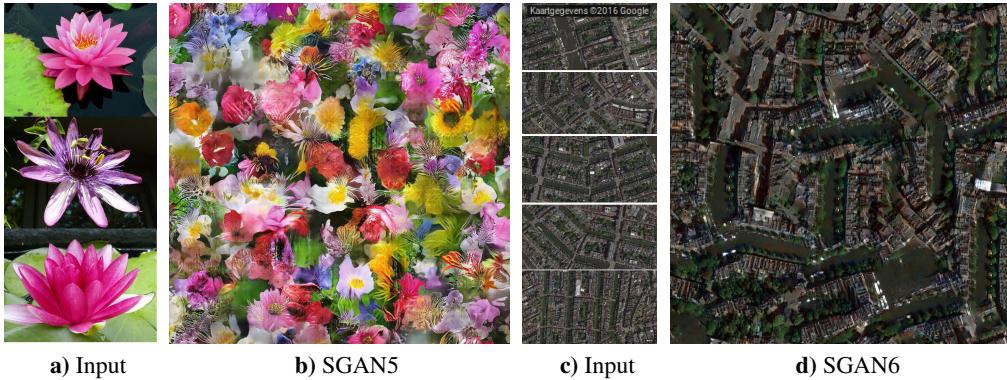


Figure 6: Learning textures from multiple images. (a) and (c) show the training data textures rescaled to fit the screen. (b) and (d) show examples of generated 1000x1000 pixel composite textures.

### 4.3 Extension using properties of the spatial GAN

The spatial dimensions of  $Z$  are locally independent – output image pixels depend only on a subset of the input noise tensor  $Z$ . This property of the SGAN allows two practical tricks for creation of output textures with special properties. Below we illustrate these tricks briefly, see Appendix I for details.

#### 4.3.1 Seamless textures

Seamless textures are important in computer graphics, because arbitrarily large surfaces can be covered by them. Suppose we want to synthesize a seamless texture of desired size  $\hat{h}, \hat{w}$  and generating it would require  $\hat{l}, \hat{m}$  spatial dimensions of  $Z$  for a given SGAN model. Let  $r = \hat{h}/\hat{l} = \hat{w}/\hat{m}$  be the ratio of the spatial dimensions of  $G(Z)$  to  $Z$ . For notation we use Python slicing notation, where  $Z_{[-i,4:]}$  indicates  $i$  indices before the end of the array in the 1st dimension, the ‘4:’ indicates all elements but the first 4 along the second dimension, and all elements along the last, not explicitly indexed, dimension. We should sample a slightly bigger noise tensor  $Z \in \mathbb{R}^{(\hat{l}+4) \times (\hat{m}+4) \times d}$  and set its edges to repeat:  $Z_{[:, -4:]} = Z_{[:, :4]}$  and  $Z_{[-4:, :]} = Z_{[:4, :]}$ . Then we can calculate  $G(Z)$  and crop  $2r$  pixels from each border, resulting in an image  $I = G(Z)_{[2r:-2r, 2r:-2r]}$  of size  $\hat{h}, \hat{w}$  that can be tiled in a rectangular grid as shown on Figure 7.

#### 4.3.2 Memory efficient generation

In addition, we can use the SGAN generator to create textures in a memory efficient way by splitting the calculation of  $I = G(Z)$  into independent chunks that use less memory. This approach allows for straightforward parallelization. A potential application would be real-time 3D engines, where the SGANs could produce the currently visible parts of arbitrary large textures.

Suppose again that we have  $Z \in \mathbb{R}^{\hat{l} \times \hat{m} \times d}$ . Let us split  $Z$  in two along dimension  $\hat{l}$ . We can call the generator twice, producing  $I^1 = G(Z_{[:\hat{l}/2+2:]})$  and  $I^2 = G(Z_{[\hat{l}/2-2:]})$ , with each call using approximately half the memory than a call to the whole  $G(Z)$ . To create the desired large output, we concatenate the two partially generated images, cropping their edges:  $I = [I^1_{[:-2r]}, I^2_{[2r:]}]$ . With this approach, the only limitation is the number of pixels that can be stored, while the memory footprint in the GPU is constant. Appendix I has precise analysis of the procedure.

## 5 Discussion

Our SGAN synthesizes textures by learning to generate locally consistent image patches, thus making use of the repeating structures present in most textures. The mixing length scale of the generation depends on the projective field sizes. Choosing the best architecture depends on the specific texture image. This holds also for the algorithm of Gatys et al. [5], where the parametric expressiveness of the model depends on the set of network layers used for the descriptive statistics calculation.



Figure 7: A 320x320 pixels texture, tiled 2 times vertically and 6 times horizontally. It is straightforward to create seamless textures in the SGAN framework by enforcing periodic boundary conditions on  $Z$ .

Rather than using handcrafted features or other priors to specify the desired properties of the output, the adversarial framework learns the relevant statistics only from information contained in the training texture. In contrast, parametric methods specify statistical descriptors *a priori*, before seeing the image  $I$ . This includes models using the wavelet transform [12] or the properties of the filters of a neural network trained on a large image dataset [5]. Such models can generalize to many textures, but are not universal – sometimes it is better to train a generative model for a single texture, as our examples from Section 4.2.2 show.

[16] is an interesting case because it describes a generative model that takes as inputs noise vectors and produces images with desired statistics. This is analogous to the generator  $G$  in the GAN. However, features extracted from pre-trained discriminative networks (as in [5]) play the role of a discriminator function, in contrast to learned discriminators in adversarial frameworks.

The examples in Sections 4.2.1 and 4.2.2 show that our method deals well with texture images, corresponding to realizations of stationary ergodic stochastic processes that mix. It is not possible to learn statistical dependencies that exceed the projective field size with SGANs.

Synthesizing regular non-mixing textures (e.g. a chess grid pattern) is problematic for some methods [11]. The SGAN cannot learn such cases - distant pixels in the output image are independent from one another because of the strong mixing property of the SGAN generation, as discussed in Section 3. For example, the model learns basic letter shapes from the text image in Figure 3 (bottom row), but fails to align the "letters" into globally straight rows of text. The same example shows that the approach of Gatys [5] has similar problems synthesizing regular textures. In contrast, the parametric method of [12] and non-parametric instance-based methods work well with regular patterns.

To summarize the capabilities of our method:

- real-time generation of high quality textures with a single forward pass
- generation of images of any desired size
- processing time requirements scale linearly with the number of output pixels
- combination of separate source images into complex textures
- seamless texture tiles

As next steps, we would like to examine modifications allowing the learning of images with longer spatial correlations and strong regularity. Conditional GAN [13] architectures can help with that, as well as allow for more precise control over the generated content. Conditioning can be used to train a single network for a variety of textures simultaneously, and then blend them in novel textures.

In future work, we plan to examine further applications of the SGAN such as style transfer, mosaic rendering, image completion, in-painting and modeling 3D data. In particular, 3D data offers intriguing possibilities in combination with texture modeling – e.g. in machine generated fashion designs. The SGAN approach can also be applied to audio data represented as time series, and we can experiment with novel methods for audio waveform generation.

## Acknowledgements

We would like to thank Christian Bracher, Sebastian Heinz and Calvin Seward for their valuable feedback on the manuscript.

## References

- [1] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285*, 2016.
- [2] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH, 2001.
- [3] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the International Conference on Computer Vision*, 1999.
- [4] Arthur Szlam Emily Denton, Soumith Chintala and Rob Fergus. Deep generative image models using a Laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems 28*, 2015.
- [5] Leon Gatys, Alexander Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, 2015.
- [6] G. Georgiadis, A. Chiuso, and S. Soatto. Texture compression. In *Data Compression Conference*, March 2013.
- [7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, 2014.
- [8] 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*, 2015.
- [9] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.
- [10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [11] Yanxi Liu, Wen-Chieh Lin, and James Hays. Near-regular texture analysis and manipulation. In *ACM SIGGRAPH Papers*, 2004.
- [12] Javier Portilla and Eero P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vision*, 40(1), October 2000.
- [13] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.
- [14] Jost T. Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.
- [15] Lucas Theis and Matthias Bethge. Generative image modeling using spatial LSTMs. In *Advances in Neural Information Processing Systems 28*, 2015.
- [16] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *International Conference on Machine Learning*, 2016.
- [17] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the Art in Example-based Texture Synthesis. In M. Pauly and G. Greiner, editors, *Eurographics 2009 - State of the Art Reports*. The Eurographics Association, 2009.

## Appendix I

We examine in detail the projective fields (PF) and which inputs from  $Z$  map to which output pixels in  $G(Z)$ . Let  $[a, b)$  indicate a range starting at index  $a$  inclusive and ending at index  $b$  exclusive, which we will also call left and right border. For convenience of notation, we will write only 1D indices for the square PFs, e.g.  $[0, 1)$  will refer to the square field  $Z_{[1,1]}$  in python slicing notation. For simplicity, we express the formulas valid only for the architecture we usually used for SGAN, 5x5 kernels and  $k$  convolutional layers with stride  $\frac{1}{2}$ .

We start by examining the recursive relation between input and output of a fractionally strided convolutional layer.

**Proposition 1** *An input  $[a, b)$  has as its PF an output  $[a', b')$  after applying one convolutional layer. It holds that  $a' = 2a - 2$  and  $b' = 2b + 1$ .*

This relation holds because of the way we implement a convolutional layer with stride  $\frac{1}{2}$  in Theano, just like DCGAN [13] does. Note that  $b' - a' = 2(b - a) + 3$  which is exactly relationship 13 from [1] between input and output sizes of a transposed convolution.

We can rewrite the recursive relations as a function of the initial size and count of layers  $k$ :

**Proposition 2** *An input  $[a, b)$  has as its PF an output  $[a', b')$  after  $k$  convolutional layers. It holds that  $a' = a2^k - 2^{k+1} + 2$  and  $b' = b2^k + 2^k - 1$ .*

In particular, we get the PF size of a single  $z_{\lambda\mu}$  for  $b = a + 1$  as  $PF(k) = 2^{k+2} - 3$ , which we denote in Table 1 as PF/RF.

With these relations, we can show why we can split without any loss of information the calculation of a big array  $Z$  in two smaller volumes as in Section 4.3.2.

**Proposition 3** *Let  $G(Z) = I$  with  $Z \in \mathbb{R}^{\hat{l} \times \hat{m} \times d}$ . We can define  $I^1 = G(Z_{[\hat{l}/2+2:]})$  and  $I^2 = G(Z_{[\hat{l}/2-2:]})$ . Then it holds that  $I = [I^1_{[-2r]}, I^2_{[2r:]}]$  where  $r = 2^k$ .*

Since we split only on one of the spatial dimensions, it is enough to reason only for intervals in that dimension and not the whole 2D field. Image  $I^1_{[-2r]}$  in the Python slicing notation is the same as image  $I^1_{[:r\hat{l}/2]}$ , and its rightmost pixel has index  $x_r = 2^k\hat{l}/2 - 1$ . The PF of  $Z_{[\hat{l}/2+2:]}$  has a left border  $(\hat{l}/2 + 2)2^k - 2^{k+1} + 2 = 2^{k-1}\hat{l} + 2 > x_r$ . The calculation of the pixel  $x_r$  is not influenced by any further elements from  $Z$ , i.e. for any  $I^{1+\delta} = G(Z_{[\hat{l}/2+2+\delta:]})$ ,  $\delta \geq 0$  it holds that  $I^1_{[:r\hat{l}/2]} = I^{1+\delta}_{[:r\hat{l}/2]}$ . This would mean that  $I^1_{[:r\hat{l}/2]}$  is exactly equal to  $I_{[:r\hat{l}/2]}$ , the left half of the desired image  $I$ . By a similar argument, the left-most pixel of  $I^2_{[2r:]}$  is not influenced by  $Z_{[\hat{l}/2-2:]}$ , and  $I^2_{[2r:]}$  is equal to  $I_{[r\hat{l}/2-:]}^2$ , the right half of the desired image  $I$ . This proves that  $I = [I^1_{[-2r]}, I^2_{[2r:]}]$ .

This proof shows why an overlap of 2 spatial dimensions of  $Z$  is sufficient for splitting. Is it also necessary? The answer is yes, since  $Z_{[\hat{l}/2+2:]}$  is the smallest set required to calculate  $I^1$ : the PF of  $Z_{[\hat{l}/2+1:]}$  has left border  $(\hat{l}/2 + 1)2^k - 2^{k+1} + 2 = 2^{k-1}\hat{l} - 2^k + 2 < x_r$  and right border  $(\hat{l}/2 + 2)2^k + 2^k - 1 > x_r$ , so the pixel  $x_r$  is inside the PF of  $Z_{[\hat{l}/2+1:]}$ .

A similar proof can be made for the seamless, i.e. periodic, texture case from Section 4.3.1, and we sketch it here. Making the  $Z$  periodic in its spacial dimensions makes the output  $I$  periodic as well. As in the previous case, we need for each border an overlap of 2, hence we need to make 4 elements along each dimension to be identical, i.e. we set  $Z_{[:, -4:]} = Z_{[:, :4]}$  and  $Z_{[-4:, :]}$  =  $Z_{[:4, :]}$ . More of the periodic structure in  $Z$  is not needed as it would be redundant. The output image is  $I = G(Z)_{[2r:-2r, 2r:-2r]}$ . It is easily shown that the leftmost and rightmost pixels  $I_{[0,0]}$  and  $I_{[0,-1]}$  fit together as required for a seamless texture. These pixels use information from elements of  $Z$  that are equal numerically,  $Z_{[:4,:4]} = Z_{[:4,-4:]}$ . The relative positions of the pixel  $I_{[0,0]}$  inside the PF of the volume  $Z_{[:4,:4]}$  is offset exactly by 1 pixel compared to the position of  $I_{[0,-1]}$  inside the PF of  $Z_{[:4,-4:]}$ .