# Non-Stationary Texture Synthesis by Adversarial Expansion

YANG ZHOU*, Shenzhen University and Huazhong University of Science & Technology
ZHEN ZHU and XIANG BAI, Huazhong University of Science and Technology
DANI LISCHINSKI, The Hebrew University of Jerusalem
DANIEL COHEN-OR, Shenzhen University and Tel Aviv University
HUI HUANG†, Shenzhen University

Fig. 1. Examples of two extremely challenging non-stationary textures (middle column), synthesized by our method (left and right). Note that our method succeeds in reproducing and extending the global structure and trends present in the input exemplars.

The real world exhibits an abundance of non-stationary textures. Examples include textures with large scale structures, as well as spatially variant and inhomogeneous textures. While existing example-based texture synthesis methods can cope well with stationary textures, non-stationary textures still pose a considerable challenge, which remains unresolved. In this paper, we propose a new approach for example-based non-stationary texture synthesis. Our approach uses a generative adversarial network (GAN), trained to double the spatial extent of texture blocks extracted from a specific texture exemplar. Once trained, the fully convolutional generator is able to expand the size of the entire exemplar, as well as of any of its sub-blocks. We demonstrate that this conceptually simple approach is highly effective for capturing large scale structures, as well as other non-stationary attributes of the input exemplar. As a result, it can cope with challenging textures, which, to our knowledge, no other existing method can handle.

CCS Concepts: • **Computing methodologies → Appearance and texture representations**; *Image manipulation*; *Texturing*;

Additional Key Words and Phrases: Example-based texture synthesis, non-stationary textures, generative adversarial networks

**ACM Reference Format:**
Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2018. Non-Stationary Texture Synthesis by Adversarial Expansion. *ACM Trans. Graph.* 37, 4, Article 49 (August 2018), 13 pages. https://doi.org/10.1145/3197517.3201286

*Yang Zhou and Zhen Zhu are joint first authors
†Corresponding author: Hui Huang (hhzhiyan@gmail.com)

Authors' addresses: Yang Zhou, Shenzhen University, Huazhong University of Science & Technology; Zhen Zhu; Xiang Bai, Huazhong University of Science and Technology; Dani Lischinski, The Hebrew University of Jerusalem; Daniel Cohen-Or, Shenzhen University, Tel Aviv University; Hui Huang, College of Computer Science & Software Engineering, Shenzhen University.


0730-0301/2018/8-ART49 $15.00
https://doi.org/10.1145/3197517.3201286

## 1 INTRODUCTION

Example-based texture synthesis has been an active area of research for over two decades. However, despite excellent results for many classes of textures, example-based synthesis of significantly non-stationary textures remains an open problem. Examples of non-stationary textures include textures with large-scale irregular structures, or ones that exhibit spatial variance in various attributes, such as color, local orientation, and local scale. Inhomogeneous textures, such as weathered surfaces are another challenging example of non-stationarity. Several challenging non-stationary examples are shown in Figures 1 and 2.

During the previous decade, stitching-based [Efros and Freeman 2001; Kwatra et al. 2003] and optimization-based [Kwatra et al. 2005; Wexler et al. 2007] approaches have established themselves as highly effective for example-based texture synthesis. More recently, deep learning based approaches for texture synthesis have begun to gain popularity. However, Figure 7 demonstrates that none of the existing state-of-the-art methods are able to successfully cope with significantly non-stationary input exemplars. Depending on the assumptions of each method, the results are either stationary or periodic, failing to mimic the large-scale structure and spatially variant attributes of the exemplars.

Fig. 2. Four non-stationary textures (middle) and our synthesis results (left and right). Again, the global structure present in the input exemplars is preserved and extended in our results. For example, in the leaf texture, additional veins appear, whose orientation and spacing is consistent with the input.

The fundamental goal of example-based texture synthesis is to generate a texture, usually larger than the input, that faithfully captures all the visual characteristics of the exemplar, yet is neither identical to it, nor exhibits obvious unnatural looking artifacts. Given this goal, a major challenge of non-stationary texture synthesis lies in preserving the large-scale structures present in the exemplar. Consider, for example, the nearly co-centric wood rings in the right example in Figure 1; reproducing this structure is essential for maintaining the visual similarity of the outcome to the input, and preserving the natural appearance of wood. Additionally, it is crucial to reproduce the manner in which local patterns vary across the spatial domain, such as the changes in scale in the left example in Figure 1. These requirements are challenging for existing methods, most of which operate by enforcing similarity of local patterns and/or of certain global statistics to those of the exemplar.

In this work, we propose a new method for example-based synthesis of non-stationary textures, which uses a generative adversarial network (GAN) for this purpose. Conceptually, our approach is, in fact, extremely simple. The goal of the generator network is to learn how to *expand* (double the spatial extent) an arbitrary texture block cropped from the exemplar, such that expanded result is visually similar to a containing exemplar block of the appropriate size. The visual similarity between the expanded block and an actual containing block is assessed using a discriminator network. The discriminator is trained (in parallel to the generator) to distinguish between actual larger blocks from the exemplar and those produced by the generator. This self-supervised adversarial training takes place for each specific texture exemplar. Once trained, the fully convolutional generator may be used to generate extended textures up to double the original exemplar's size, that visually closely resemble the exemplar. Even larger textures may be synthesized by feeding the generator with its own output.

Our approach also supports texture transfer: when a generator trained using a certain texture exemplar is fed with a pattern taken from another image or texture, the resulting synthesized texture follows the large scale structure from the input pattern.

At first glance, our approach may resemble deep super-resolution approaches, such as SRGAN [Ledig et al. 2016]. Note, however, that super-resolution aims to enhance (sharpen) the *already existing* content of an image patch or block. In contrast, our approach learns to *inject* new content! This is evident in the examples of our results shown in Figures 1 and 2: all these results exhibit more elements (feathers, wood rings, leaf veins, tiles, etc.) than present in the input exemplar. Unlike in super-resolution, the size and spacing of the elements remains similar to the input, but additional elements are added without obvious repetition of the original ones.

In summary, through a variety of results and comparisons, we show that using a conceptually simple adversarial training strategy, we are able to cope with an unprecedented array of highly non-stationary textures, which to our knowledge none of the currently existing methods are able to handle.

## 2 RELATED WORK

We begin with a brief review of classical example-based texture synthesis methods, followed by a more detailed discussion of recent deep learning based approaches. In either category, the existing methods are unable to cope with highly inhomogeneous textures, or textures that exhibit large scale or global structures.

### 2.1 Classical approaches

Example-based texture synthesis has been extensively researched for over twenty years, and we refer the reader to Wei et al. [2009] for a comprehensive survey. The most effective approaches have been non-parametric methods, which include pixel-based methods [Efros and Leung 1999; Wei and Levoy 2000], stitching-based methods [Efros and Freeman 2001; Kwatra et al. 2003], optimization-based methods [Kwatra et al. 2005; Wexler et al. 2007], and appearance-space texture synthesis [Lefebvre and Hoppe 2006].

Image melding [Darabi et al. 2012] unifies and generalizes patch-based synthesis and texture optimization, while Kaspar et al. [2015] describe a self-tuning texture optimization approach, which uses image melding with automatically generated and weighted guidance channels. These guidance channels are designed to help reproduce the middle-scale structures present in the texture exemplar. However, as demonstrated in Figure 7, this state-of-the-art classical approach is unable to capture and reproduce the large-scale or global structure that may be present in the exemplar.

In general, while classical non-parametric methods are typically able to reproduce small scale structure, they assume a stationary Markov Random Field (MRF) model, making it difficult for them to cope with highly inhomogeneous textures, which violate this assumption. Thus, control of large scale structure and inhomogeneity has typically required user-provided or automatically generated guidance maps (e.g., [Hertzmann et al. 2001; Rosenberger et al. 2009; Zhang et al. 2003; Zhou et al. 2017]). We are not aware of *any* classical example-based texture synthesis method capable of automatically coping with challenging non-stationary exemplars, such as the ones shown in Figures 1 and 2.

Certain classes of global structures can be handled by classical texture synthesis approaches. For example, Liu et al. [2004] analyze near-regular textures and explicitly model their geometric and photometric deviations from a regular tiling. In contrast, our approach does not make any assumptions regarding the structure, nor does it attempt to analyze it. Yet, with the same deep architecture and training strategy, we are also able to synthesize regular and near-regular textures, as demonstrated in Figure 3.

### 2.2 Deep Learning based approaches

Gatys et al. [2015a] were, to our knowledge, the first to use a deep neural network for example-based texture synthesis. They characterize an input texture by a collection of Gram matrices, each defined by inner products between feature channels at a certain convolution layer of a pre-trained image classification network (in their case VGG-19 [Simonyan and Zisserman 2014]). An image input to the network is then iteratively optimized (using back-propagation) so as to minimize a loss function defined as a sum of weighted differences between its Gram matrices and those of the original exemplar.

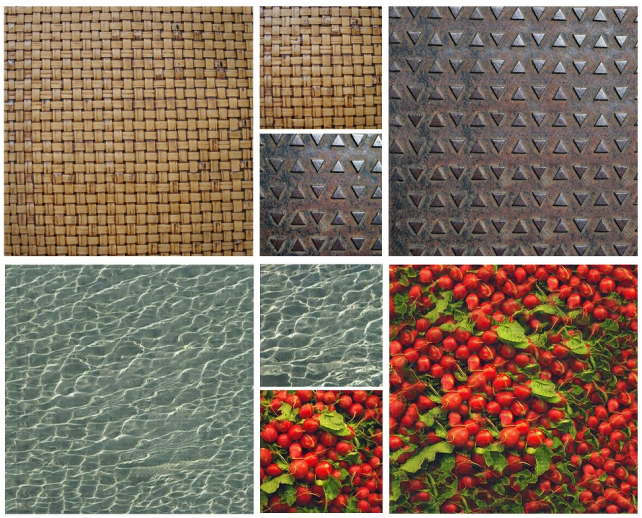

Fig. 3. Our method can also be successfully applied to stationary, regular, near-regular, or stochastic textures.

The loss function of Gatys et al. [2015a], often referred to as *Gram loss* or *style loss* (in the context of neural style transfer [Gatys et al. 2015b]), is unable to capture well regularities and larger structures in the texture. Sendik and Cohen-Or [2017] address this deficiency by introducing *structural energy*, based on deep inter-feature correlations. This approach considerably improves synthesis quality for textures with regular structure, but still can not deal with non-stationary structures.

Gatys et al. [2015b] extend their Gram-based texture synthesis approach to perform artistic style transfer. To achieve this, a *content loss* term is added to the Gram-based style loss. This approach still requires iterative optimization to obtain each result. Ulyanov et al. [2016] and Johnson et al. [2016] both propose a fast implementation of Gatys et al.'s texture synthesis and style transfer using a single feed-forward pass through a network trained for a specific texture (style). The idea is to move the computational burden to the learning stage: a generator network is trained by using a pre-trained *descriptor network* (also referred to as *loss network*) based on VGG-19 in place of a loss function. The quality of the results is comparable to Gatys et al., while the synthesis itself is extremely fast, once the network has been trained. In Figure 7 we compare our results to those of Ulyanov et al. (TextureNets), which can also be viewed as a comparison with Gatys et al. [2015b] and Johnson et al. [2016].

Several works have utilized Generative Adversarial Networks (GANs) to perform texture synthesis and style transfer.

Li and Wand [2016] introduce Markovian Generative Adversarial Networks (MGANs). Rather than capturing style with global statistics, their generator network is trained using a discriminator which examines statistics of Markovian neural patches, i.e., local patches on feature maps extracted by a pre-trained VGG network, thereby imposing a Markov Random Field prior. As in other style transfer approaches, explicit layout constraints may also be imposed via a "content" image provided as additional input.

Jetchev et al. [2016] also utilize GANs for texture synthesis, where texture patches are generated from random noise, and patches of the same size as the generated output are randomly selected from the exemplar as the ground truth for adversarial training. However, their method failed to produce high quality textures consistently. Bergmann et al. [2017] extend this approach by introducing a periodic function into the input noise, which enables synthesizing periodic textures with high quality. However, the approach, referred to as PSGAN, is limited to periodic textures and thus is not applicable to most real-world textures, as demonstrated in Figure 7.

Isola et al. [2016] demonstrate the effectiveness of GANs for a variety of image-to-image translation tasks. Zhu et al. [2017] introduce CycleGANs, where the translation network can be trained with unpaired training data. In these tasks, the input and output differ in appearance, but correspond to different renderings of the same underlying structure. This is not the case in our approach, where the goal is to extend the global structure of the exemplar. We do so by introducing new instances of local patterns, which are similar, but not identical, to those present in the exemplar.

## 3 OUR APPROACH

We begin this section with an overview of our approach, followed by a more detailed explanation of the network architectures used and the training procedure.

Our approach is very simple conceptually: given that our ultimate goal is to generate larger instances that perceptually resemble a smaller input texture exemplar, the main idea is to teach a fully convolutional generator network how to do just that. The approach is depicted by the diagram in Figure 4. More specifically, given a $k \times k$ *source block* $S$ cropped from the input exemplar, the generator must learn to produce a $2k \times 2k$ output, which is perceptually similar to an enclosing *target block* $T$ of the latter size. Note that this training procedure is self-supervised: the ground truth extended texture blocks are taken directly from the input texture. Since the generator is a fully-convolutional network, once it has been trained, we can apply it onto the entire input exemplar, or a sufficiently large portion thereof, to generate a texture that is larger than the input (up to double its size).

It is well known that pixel-based metrics, such as $L_1$ or $L_2$ are not well suited for assessing the perceptual differences between images. This is even more true when the goal is to compare different instances of the same texture, which are the output of texture synthesis algorithms. On the other hand, recent work has shown the effectiveness of adversarial training and GANs for a variety of image synthesis tasks [Isola et al. 2017; Ledig et al. 2016; Zhu et al. 2017], including texture synthesis [Bergmann et al. 2017; Li and Wand 2016]. Thus, we also adopt an adversarial training approach to train our generator. In other words, our generator $G$ is trained alongside with a discriminator $D$ [Goodfellow et al. 2014]. The discriminator $D$ is trained to classify whether a $2k \times 2k$ texture block is real (a crop from the input exemplar) or fake (synthesized by $G$).

In our approach, a dedicated GAN must be trained for each input exemplar, which takes considerable computational resources. But once the fully-convolutional generator has been trained, large texture blocks may be synthesized from smaller ones in a single

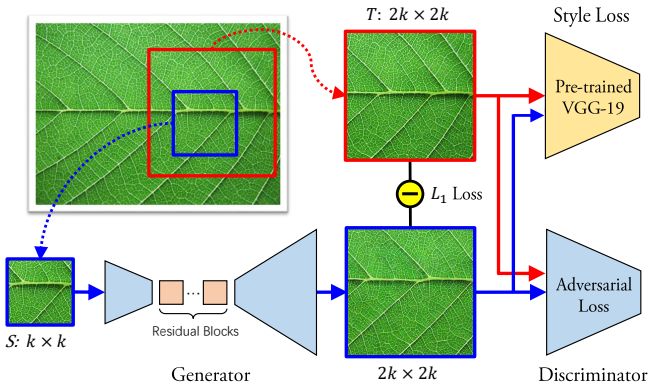


Fig. 4. Method overview. The generator learns to expand $k \times k$ texture blocks into $2k \times 2k$ ones using a combination of adversarial loss, $L_1$ loss and style loss.

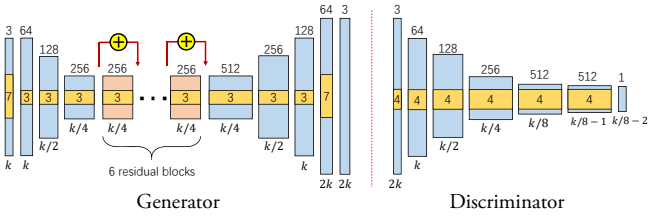


Fig. 5. Architecture of our generator and discriminator. The number of feature channels and the spatial resolution of feature maps are respectively specified on the top of and under each block, while the kernel sizes are specified in the central yellow regions.

forward pass through the network, which is extremely fast when the network runs on the GPU. The size of the $k \times k$ source blocks that we use during the training stage should be chosen large enough to capture the non-stationary behavior across the input exemplar. On the other hand, it should be small enough relative to the size of the exemplar, so that we can extract a sufficient number of different $2k \times 2k$ target blocks to train the network. In our current implementation we set $k = 128$, and our exemplars are typically of size $600 \times 400$.

### Network architecture

As explained earlier, we would like to model the generator as a fully-convolutional deep neural network. Using a fully-convolutional network allows us to apply the generator to arbitrary-sized inputs at test time, and reduces the number of parameters, compared to networks with fully connected layers. Network depth is important both for the expressive power of the generator, and for increasing the receptive field of the network's neurons. Since our goal is to capture large-scale non-stationary behavior across the source texture block, at some level of the network the receptive field should approach the size of the source block. This may be effectively achieved by introducing a chain of residual blocks [He et al. 2016].

A generator architecture that satisfies our requirements was, in fact, already proposed by Johnson et al. [2016], who demonstrated

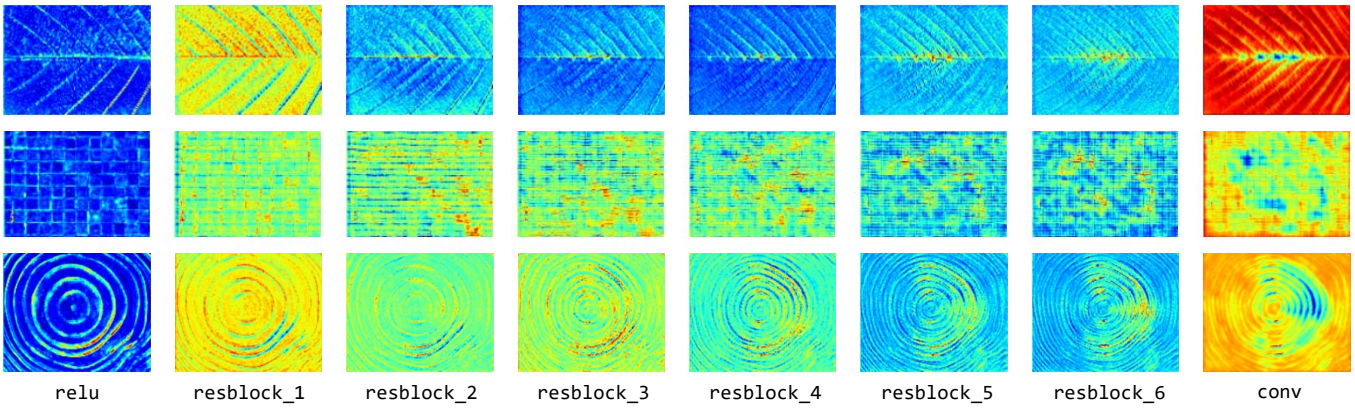


Fig. 6. Visualization of feature maps output by the middle part of our generator. Besides the intermediate results of residual blocks (from `resblock_1` to `resblock_6`) we also visualize the final output of encoding stage (`relu`), and the feature map output by the first convolution layer of the decoder (conv). While the leaf (top), bricks (middle) and wood ring (bottom) textures have very different large-scale structures, it may be observed that all of the new structures emerge in the course of the residual block chain. The creation of new structures is typically complete before the end of the chain, as evidenced by the similarity between the `resblock_5` and `resblock_6` feature maps.

its effectiveness for neural style transfer and for super-resolution. The same generator was later successfully used by Zhu et al. [2017] for a variety of unpaired image-to-image translation tasks. Neural style transfer is closely related to texture synthesis; thus, we adopt a similar architecture for our generator.

The architecture of the generator is shown in the diagram in Figure 5. The network consists of three convolution layers, two of which use stride-2 convolutions that reduce the spatial dimensions of the input. These three layers are followed by a sequence of six residual blocks [He et al. 2016]. The receptive field of the neurons at the end of the residual chain is $109 \times 109$, i.e., close to the size of our training source blocks. From this point onward, we first double the number of channels, after which the spatial dimensions are doubled three times via strided deconvolution layers, yielding twice the original spatial resolution. Finally, the multi-channel result of this process is combined back into three image channels. Similarly to previous work we use batch normalization after each convolution, except the last one.

Figure 6 visualizes the feature maps output by the residual blocks of our generator. Through this visualization, we can gain a better understanding of how the generator works. The different activation maps after the downsampling stages (`relu`) reveal that they encode details at various scales and orientations. No new large scale structures appear to be present yet. The situation is different by the end of the residual chain (`resblock_6`), where we can see that the number of the large scale structures (leaf veins, bricks or wood rings) has roughly doubled. Thus, the residual blocks appear to be responsible for introducing new large scale structures. This makes a lot of sense, since each residual block is capable of spatially transforming its input (via its two convolution layers), and adding the transformed result to its input. It appears that a chain of such blocks is capable of learning which structures, among those present in the chain's input, should be replicated, and how the resulting replicas should be spatially transformed before they are recombined with the original pattern. For example, for the leaf texture, it is capable of learning that the leaf vein structures should be shifted horizontally after replication, while for the wood rings texture it learns to shift the replicated rings radially. In either case, the amount of large scale structure is roughly doubled. However, when a generator trained on a certain texture is applied to an input consisting of completely different structures, these structures are not replicated, as demonstrated by the results in Figure 15.

While Johnson et al. [2016] employ a loss network, which is used to compute the style loss and content loss functions of Gatys et al. [2015b], we require a loss function with more sensitivity to spatial arrangement of texture elements and their spatially variant appearance. Thus, we adopt the PatchGAN discriminator [Isola et al. 2017; Ledig et al. 2016; Li and Wand 2016; Zhu et al. 2017] instead. The discriminator architecture is shown in Figure 5 (bottom right). This fully-convolutional network halves the spatial resolution of the input four times, while doubling the number of channels. The neurons at the sixth layer may be viewed as texture descriptors of length 512, representing overlapping texture patches of size $142 \times 142$ in the input. Each of these 512-dimensional descriptors is then projected into a scalar (using a $1 \times 1$ convolution, followed by a sigmoid), and the resulting 2D pattern is classified as real or fake using binary cross-entropy.

## Training procedure

Our training process follows the one outlined in the pioneering work of Goodfellow et al. [2014]: we repeatedly alternate between performing a single training iteration on the discriminator $D$, and a single training iteration on the generator $G$. In each training iteration, we randomly select one $256 \times 256$ target block $T$ from the exemplar to serve as the ground truth, as well as a random $128 \times 128$ source block $S$, contained in $T$, which is fed as input to the generator. For maximum utilization of the available data we choose to not set aside a validation or a test set. Nevertheless, our results show that the network is able to plausibly expand unseen input texture blocks that are different in both size and content from

those encountered during training, and it is easy to see that it does not merely memorize patterns. It is also capable of texture transfer, as demonstrated in Figure 15.

In addition to the standard adversarial loss function [Goodfellow et al. 2014] $\mathcal{L}_{adv}$, we use two additional loss terms: $L_1$ loss $\mathcal{L}_{L_1}$ and style loss $\mathcal{L}_{style}$ [Gatys et al. 2015a]:

$$\mathcal{L}_{total} = \mathcal{L}_{adv} + \lambda_1 \mathcal{L}_{L_1} + \lambda_2 \mathcal{L}_{style}, \tag{1}$$

where $\lambda_1 = 100$ and $\lambda_2 = 1$. As we shall demonstrate in our ablation study in Section 4.3, the adversarial loss appears to be the main workhorse, while the other two terms help stabilize the training and slightly reduce artifacts.

Following Gatys et al. [2015a], we compute the style loss using a pre-trained (on ImageNet) VGG-19 model, and compute Gram matrices for the ReLU activated feature maps output by the `relu1_1`, `relu2_1`, `relu3_1`, `relu4_1`, and `relu5_1` layers. The weights used to sum up the corresponding Gram losses are set to 0.244, 0.061, 0.015, 0.004, and 0.004, respectively. More specifically, they are given by 1000/(64 x 64), 1000/(128 x 128), 1000/(256 x 256), 1000/(512 x 512), and 1000/(512 x 512).

We choose Adam [Kingma and Ba 2014] as our optimization method with momentum set to 0.5, and train our models for up to 100,000 iterations. Learning rate is set to 0.0002 initially and kept unchanged for the first 50,000 iterations. Then, the learning rate linearly decays to zero over the remaining 50,000 iterations. Weights of convolutional layers are initialized from a Gaussian distribution with mean 0 and standard deviation 0.02. We train and test all our models on an NVIDIA Titan Xp GPU with 12GB of GPU memory.

## 4 RESULTS

Our approach was implemented using PyTorch, building on publicly available existing implementations of its various components. Generators were trained for a variety of input exemplars of sizes around 600×400 pixels. Training our GAN on an exemplar of this size takes about 5 hours for 100,000 iterations on a PC equipped with a NVIDIA Titan Xp GPU with 12GB memory. In many cases the results no longer improve after around 36,000 iterations (under 2 hours). Our implementation, as well as our trained models and other supplementary materials, are all available on the project page[1].

Once the generator has been trained it takes only 4–5 milliseconds to double the size of a 600×400 texture, since this requires only a single feed-forward pass through the generator.

A few of our synthesis results from challenging non-stationary texture exemplars exhibiting irregular large-scale structures and inhomogeneities are shown in Figures 1 and 2. In all of these examples, the global structure present in the input exemplars is successfully captured and extended by our method. Of course, our method is also applicable to more stationary textures as well, including textures with regular, near-regular, or stochastic structures. Four examples of our results on such textures are shown in Figure 3. Results for additional textures are included in the supplementary material.

[1] http://vcc.szu.edu.cn/research/2018/TexSyn

### 4.1 Comparison

Figure 7 compares our results with those produced by a number of state-of-the-art methods. The first column shows the input exemplars, which include both non-stationary and stationary textures. Our results are shown in the second column. The third column shows results produced by self-tuning texture optimization [Kaspar et al. 2015], which is a representative of classical optimization-based texture synthesis methods. The next four columns show results produced by several recent deep learning based approaches: TextureNets by Ulyanov et al. [2016], a feed-forward version of the method proposed by Gatys et al. [2015a]; DeepCor by Sendik and Cohen-Or [2017] improves upon Gatys et al.'s approach by introducing a deep correlations loss that enables better handling of large scale regular structures; MGANs of Li and Wand [2016], the first texture synthesis method to use adversarial training, employing a discriminator that examines statistics of local patches; and PSGAN of Bergmann et al. [2017], which learns to convert periodic noise into texture patches sampled from the exemplar.

These comparisons demonstrate that our approach is able to handle large-scale non-stationarity much better than existing methods, while for stationary or homogeneous textures, we produce comparable results to the state-of-the-art approaches. Additional comparison results are contained in our supplementary materials.

In terms of computation times, the self-tuning method [Kaspar et al. 2015] takes about 20 minutes per result; the deep learning based methods take between 1 hour of training per exemplar with TextureNets [Ulyanov et al. 2016], to 12 hours of training an PSGAN [Bergmann et al. 2017], and up to 8 hours for each result using Deep Correlations [Sendik and Cohen-Or 2017]. Thus, while the training time of our method is much slower than the time it takes to synthesize a single texture with a classical method, it is far from being the slowest among the deep-learning based methods.

### 4.2 Diversification

It is important for a texture synthesis algorithm to be able to produce a diverse collection of results from a single input exemplar. Since our method does not generate textures from a random seed or noise, we have explored a number of alternatives for diversifying the output. The simplest approach is to simply feed different subwindows of the exemplar as input to be expanded by our generator. Since the appearance across non-stationary exemplars varies greatly, cropping and expanding different windows may result in quite different results. This is demonstrated in Figure 8, which shows two different 512×512 synthesis results for each exemplar, obtained by taking two random 256×256 crops as input.

For exemplars with a more stochastic and stationary nature, without a clear global structure, it is also possible to diversify the results by reshuffling or perturbing the source texture. Specifically, for sufficiently stationary textures, we have been able to produce a wide variety of synthesis results by reshuffling the exemplar's content. Figure 9 shows three exemplars, each of which was split into 4×4 tiles, which were randomly reshuffled each time before feeding into the generator to yield different results. We have also experimented with adding Perlin noise to both stationary and non-stationary exemplars. We found that the changes among different results generated

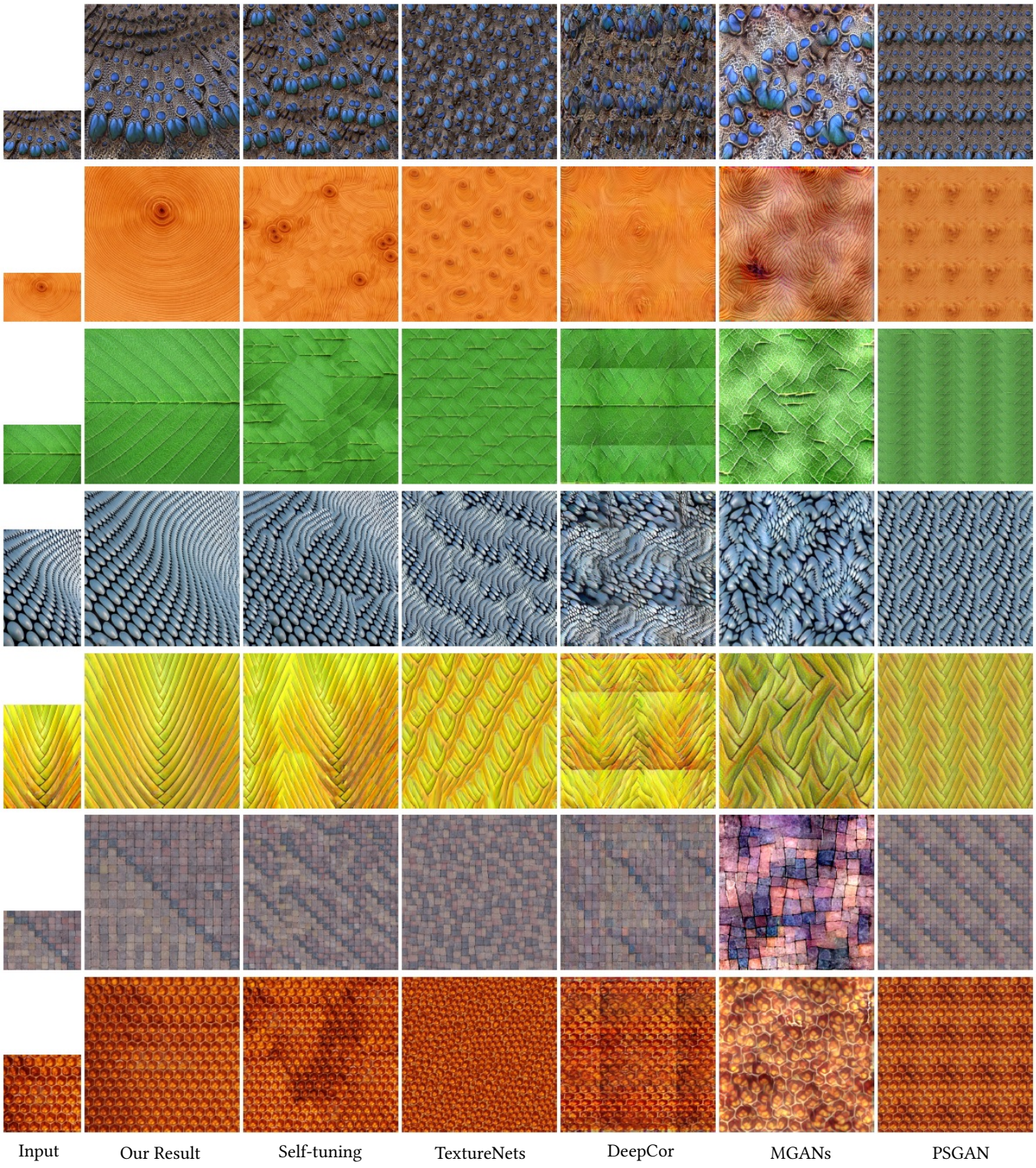


Fig. 7. Comparisons to several state-of-the-art texture synthesis methods. For each texture, the results from left to right are respectively produced by our method, Self-tuning of Kaspar et al.[2015], TextureNets of Ulyanov et al.[2016], DeepCor of Sendik and Cohen-Or [2017], MGANs of Li and Wand[2016], and PSGAN of Bergmann et al. [2017].

Fig. 8. Diversification by cropping. For each source texture (left in each triplet), we randomly crop two 256×256 sub-regions from the source texture after training, to generate different expansion results on size 512×512.

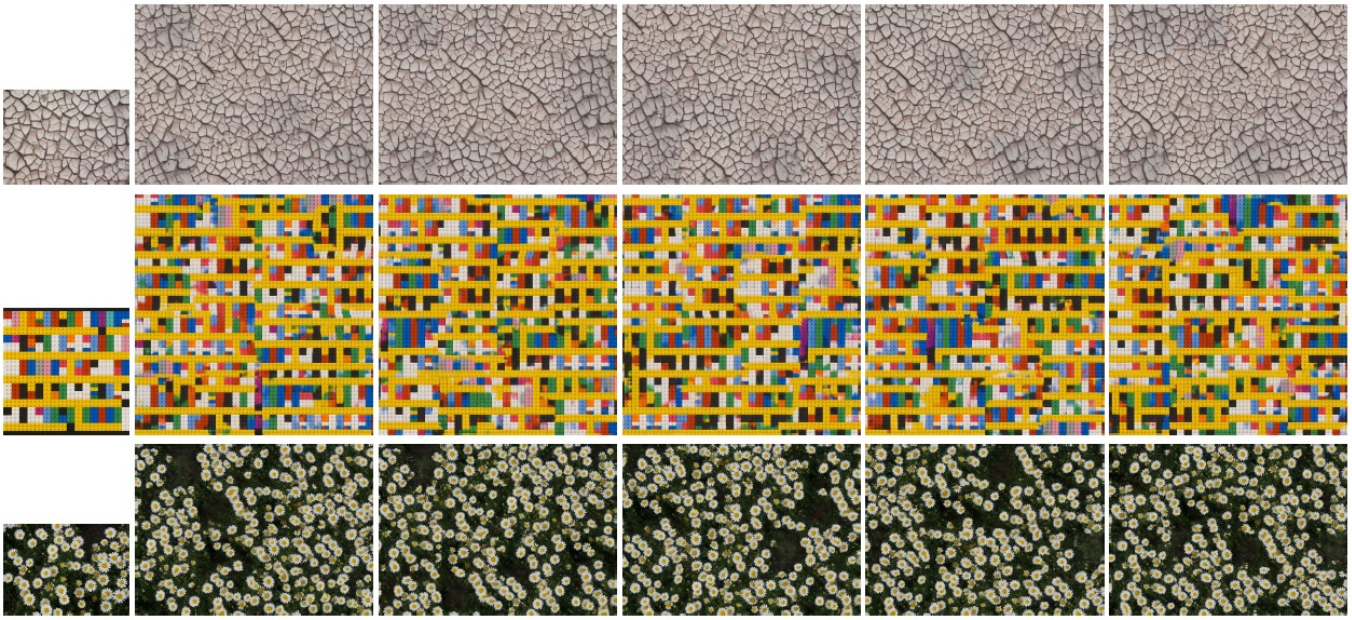

Fig. 9. Diversification by tile shuffling. The exemplar used to train the generator (leftmost column) is divided into tiles, which are randomly reshuffled before feeding into the generator, yielding different results.

in this manner are more moderate, and are best presented using animated sequences; we include a number of such animations in our supplementary materials.

### 4.3 Self evaluation

*Ablation study.* Figure 10 shows the results of an ablation study that we carried out in order to verify the importance of the various loss terms in Equation 1. We first train the generator with the adversarial loss switched off, i.e., without adversarial training. In this case, the generator fails to properly expand the input texture: no new large scale structures are introduced in the leaf example, and the smaller scale structures are not reproduced faithfully. Next, we turn on adversarial training and experiment with different combinations of the other two loss terms, including: adversarial loss only, adversarial and $L_1$ loss, adversarial and style loss, and the combination of all three terms. The visual differences between results achieved using these different combinations are quite subtle. Clearly, the adversarial loss plays a *key role* in our approach, as it alone already produces good results. Nevertheless, some noise and artifacts are present, which are reduced by adding the $L_1$ loss. However, this also causes

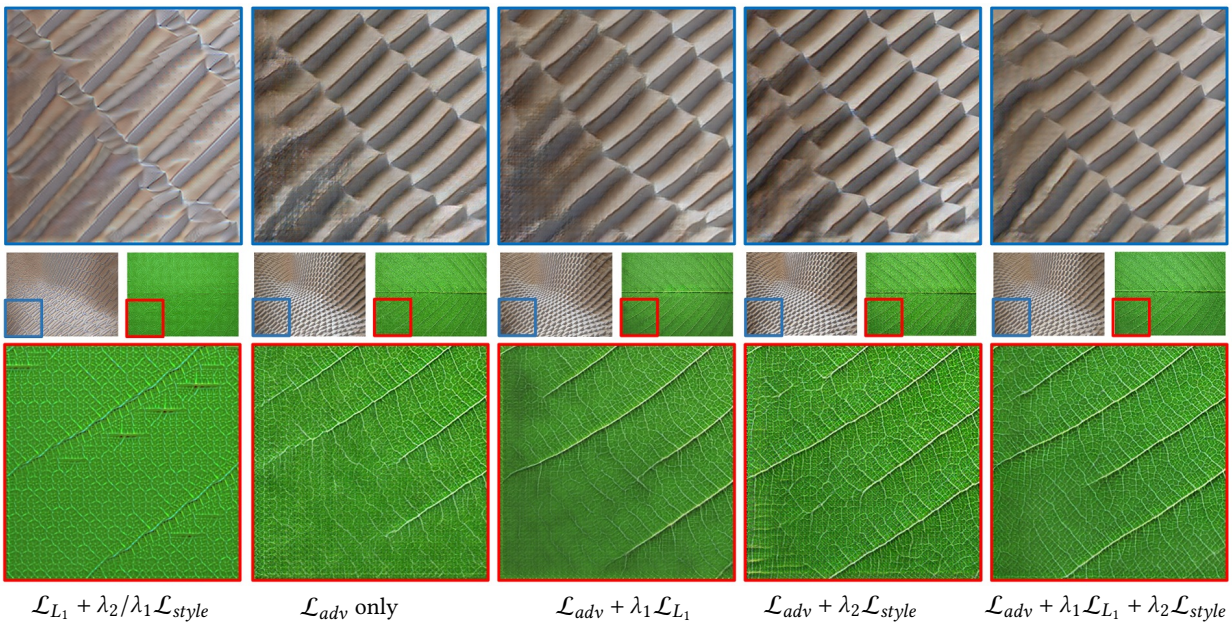


Fig. 10. Ablation study on two textures shown in Figure 2. The leftmost column shows the expansion results without adversarial training, and the remaining columns show results of using different combinations of loss terms with adversarial loss switched on. The full results of adversarial expansion are shown in the middle row, while the top & bottom rows zoom into the blue and red framed windows indicated in the middle row. For high-resolution full image results, please refer to our supplementary materials.

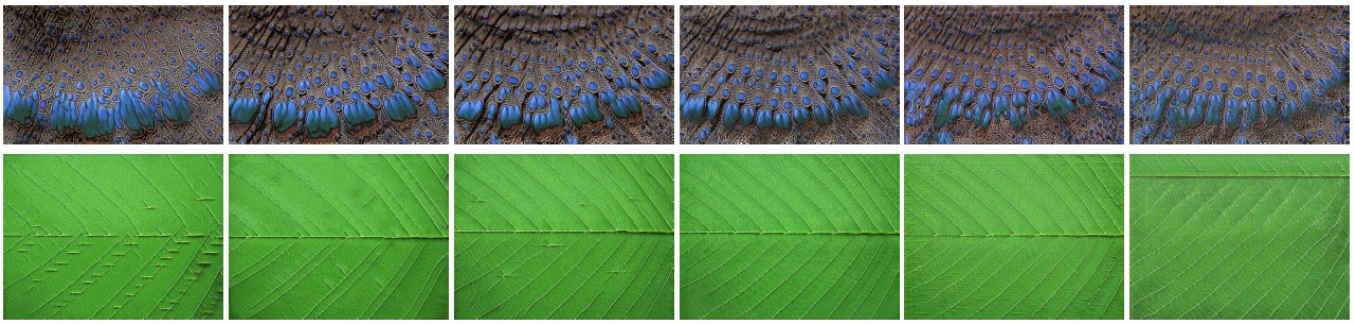


Fig. 11. Comparison of PatchGANs with different receptive fields, ranging from 16×16 to 574×574, corresponding to 3 to 8 convolutional layers in the discriminator. Adding layers increases the receptive field (i.e., the patch size) of PatchGAN, which makes it possible for the discriminator to examine larger structures. However, as may be seen above, very large patch sizes seem to cause the discriminator to pay less attention to local details. We use a patch size of 142×142 in our results.

oversmoothing of local details in some areas. In contrast, style loss enhances details, but at the same time introduces artifacts into the structures and causes some color distortions. The combination of all three terms, yields the best results, in our experience.

*Discriminator patch size.* The PatchGAN discriminator used in our approach is fully convolutional. Thus, it can be adjusted to examine texture patches of different sizes by changing the number of its resolution-reducing convolutional levels. We experimented with PatchGANs of six different sizes (ranging from 16 to 574). Results for two textures are shown in Figure 11. Our results on these and other textures consistently indicate that the best texture expansions are obtained using a 142×142 PatchGAN.

*Synthesis stability.* Kaspar et al. [2015] proposed an interesting *stress test* to evaluate the stability of a synthesis algorithm, which consists of feeding an algorithm with its own output as the input exemplar. Since our approach doubles the size of its input at every stage, we conducted a modified version of this test, where after each synthesis result is obtained, we randomly crop from the result a block of the same size as the original input and feed it back to our method. Note that we keep applying the same generator, without

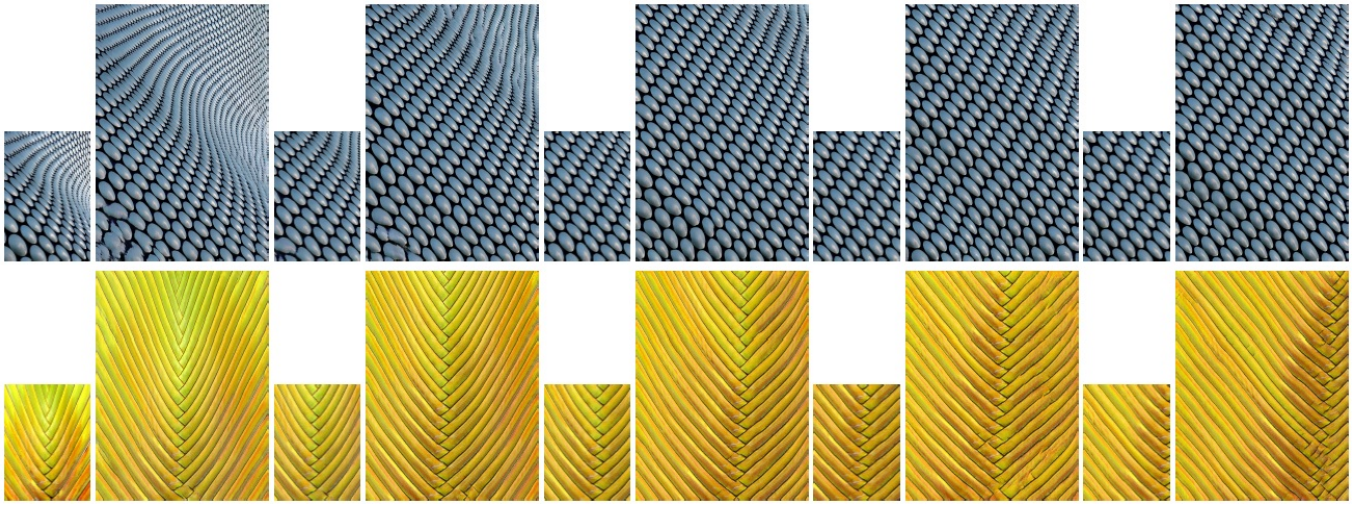

Fig. 12. Stress test #1. Given a source texture (leftmost column), we double its size using our method. Then we randomly crop a region of the same size as the source texture from the expansion result, and expand it again without any further training. The above crop-expansion cycle is repeated 4 times. We can see that the final result (rightmost column) is still very sharp and natural looking, attesting to the stability of our method.

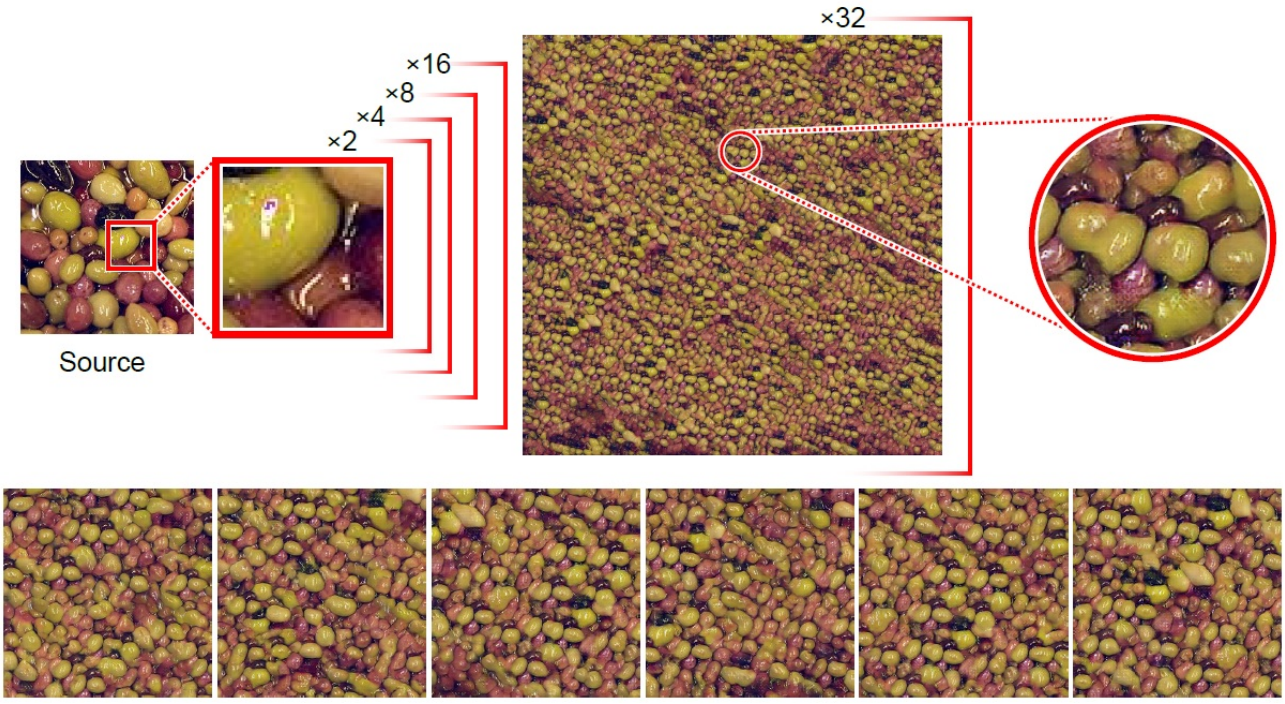


Fig. 13. Extreme expansion. Having trained a generator on the source exemplar (left), we feed it with a small cropped texture block (64×64 pixels), and feed the expanded result back into the generator. Five such cycles produce a 2048×2048 result. Six different crops from this result are shown in the bottom row.

any re-training or fine-tuning. Figure 12 shows the results of five synthesis generations on two textures. Obviously, since in this process we essentially zoom-in on a portion of the original texture, the global structure changes accordingly. However, it may be seen that the smaller scale texture elements remain sharp and faithful to their shapes in the original exemplar.

*Extreme expansion.* Given that our method can expand the source texture up to twice its size, by repeating the expansion one can synthesize very large results. Figure 14 shows the result of expanding

Fig. 14. Expansion of the wood ring texture to a size four times larger than the exemplar by repeating the expansion twice, resulting in a 2400×1504 texture. The synthesis adds additional wood rings while preserving their local appearance, as well as their global radial structure.

the wood rings exemplar by a factor of four (by expanding once more the result shown in Figure 1 using the same trained model). The result successfully maintains the radial structure of the wood rings. Figure 13 shows a more extreme expansion result, where starting from a 64×64 patch, it is expanded to x32 of its original size via five expansion cycles. All of the cycles use the same model trained on the original exemplar. Two additional multi-cycle expansion examples can be seen in our supplementary materials.

### 4.4 Texture Transfer

Texture transfer is a process where a given example texture is applied to another image, guided by various properties of the latter. Early work [Efros and Freeman 2001; Hertzmann et al. 2001] performed texture transfer based on the brightness of the target image. Artistic style transfer [Gatys et al. 2015b] may be viewed as texture transfer guided by a more sophisticated analysis of the target's content. Our approach, may be used without any modification to produce synthesized textures that follow the large scale structure of a *guiding image*. This is achieved simply by feeding the guiding image as input to a trained generator. A collection of texture transfer results is shown in Figure 15. The resolution of these results is twice that of the guiding images. In this case, however, no new large scale structures are produced, since the generator was not trained to extend the structures present in the guidance image. Since our generator is not trained to extract high-level semantic information from the input image, we find that this approach is not well suited for artistic style transfer. However, Figure 15 demonstrates its usefulness for synthesis of textures that follow a certain large-scale pattern.

## 5 SUMMARY

We have presented an example-based texture synthesis method capable of expanding an exemplar texture, while faithfully preserving the global structures therein. This is achieved by training a generative adversarial network, whose generator learns how to expand small subwindows of the exemplar to the larger texture windows containing them. A variety of results demonstrate that, through such adversarial training, the generator is able to faithfully reproduce local patterns, as well as their global arrangements. Although a dedicated generator must be trained for each exemplar, once it is trained, synthesis is extremely fast, requiring only a single feed-forward pass through the generator network. The trained model is stable enough for repeated application, enabling generating diverse results of different sizes.

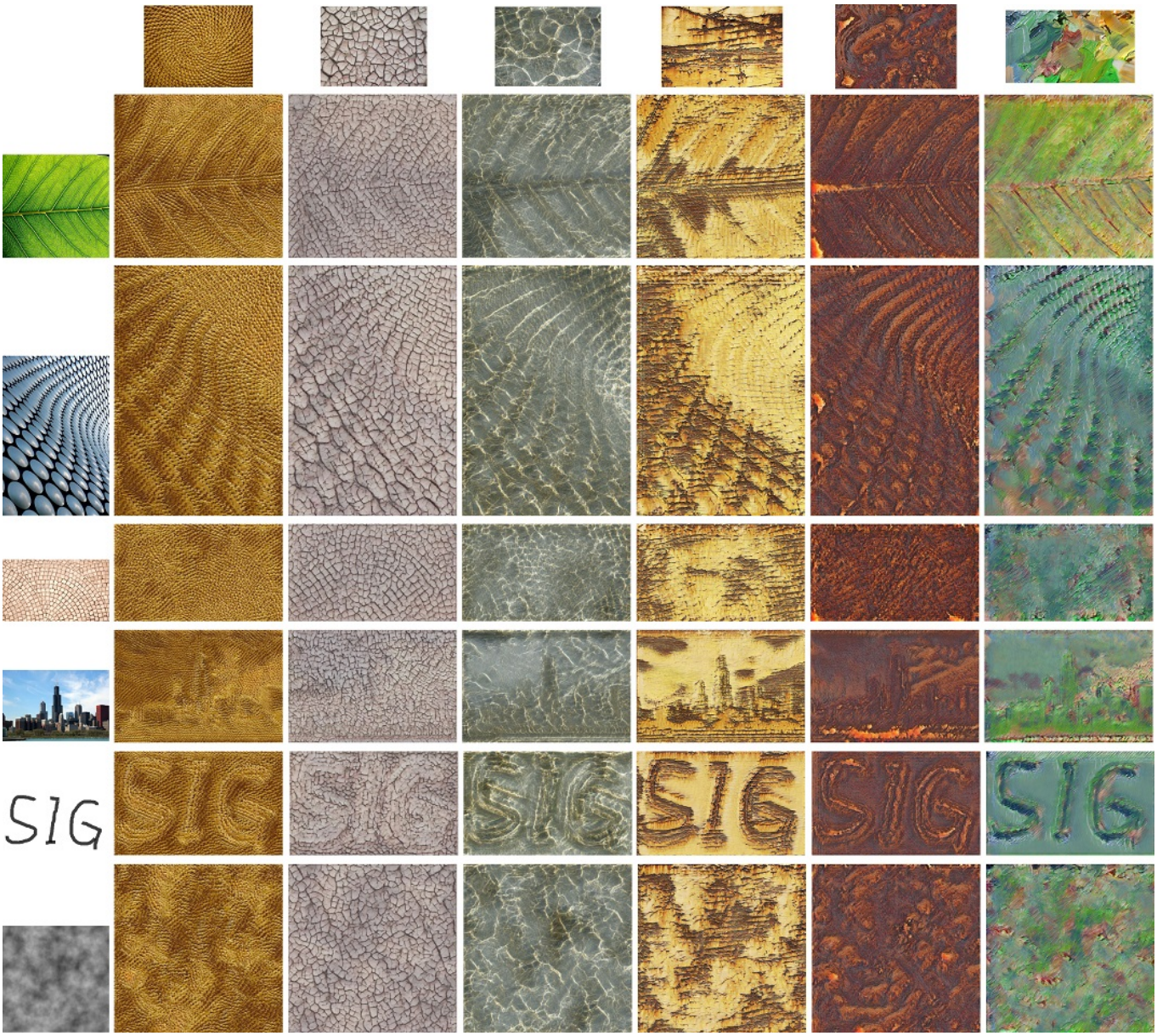


Fig. 15. Texture transfer. By feeding generators trained using the texture exemplars in the top row with guiding images in the leftmost column we synthesize textures that adapt to the large scale structures present in the guiding images. Note that we can even input a simple user sketch or pure random noise (Perlin noise) and generate satisfactory results as shown in the last two rows.

Training time is a limitation of our approach, although it is faster than previous GAN-based synthesis approaches. It would be useful to find a reliable stopping criterion for the training: at the moment, we train our models for 100,000 iterations, although in many cases the results no longer improve after 36,000 iterations or so.

In terms of result quality, artifacts tend to emerge in the vicinity of borders and corners, as may be seen in Figure 16. This may be attributed to fewer training examples in these areas, and possibly also related to the padding performed by the convolution layers.

Figure 17 shows two failure cases of our method. These failures may still be attributed to limited training examples. For example, for the stone tiles texture, all the tiles are quite large and distinct. So is the singularity at the center of the sunflower texture. In general, if the generator has not seen enough examples of a particular large scale structure or pattern during training, it cannot be expected to correctly reproduce and/or extend such structures during test time. The network does not learn some kind of a high-level representation of the texture; it only learns how to extend commonly occurring

patterns. In the future, we would like to address this issue. It might be facilitated by training on multiple textures of the same class. With richer data we may possibly train a more powerful model for generalized texture synthesis tasks.

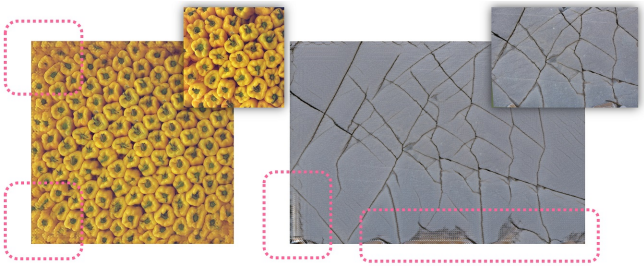

Fig. 16. Artifacts in the border and corner regions.

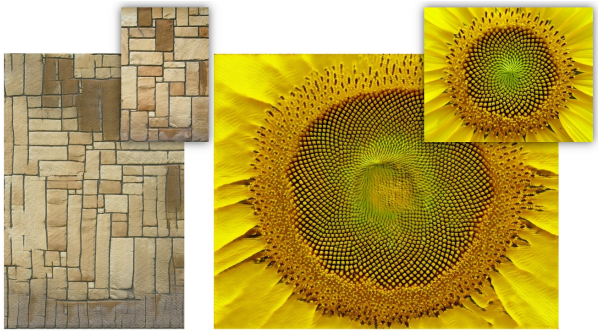

Fig. 17. Failure cases of our method. For the stone tiles texture, our method failed to learn its large scale structure (left). While for the sunflower, our method failed to reproduce the singularity at the center (right).

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. This work was supported in part by NSFC (61522213, 61761146002, 61602461, 6171101466), 973 Program (2015CB352501), Guangdong Science and Technology Program (2015A030312015), the Israel Science Foundation (2366/16), the ISF-NSFC Joint Research Program (2217/15, 2472/17) and the Shenzhen Innovation Program (KQJSCX20170727101233642, JCYJ20151015151249564).

## REFERENCES

Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. 2017. Learning Texture Manifolds with the Periodic Spatial GAN. *CoRR* abs/1705.06566 (2017).

Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B Goldman, and Pradeep Sen. 2012. Image Melding: Combining Inconsistent Images using Patch-based Synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH 2012)* 31, 4 (2012).

Alexei A. Efros and William T. Freeman. 2001. Image quilting for texture synthesis and transfer. In *Proc. SIGGRAPH 2001*. 341–346.

Alexei A. Efros and Thomas K. Leung. 1999. Texture Synthesis by Non-Parametric Sampling. *Proc. ICCV '99* 2 (1999), 1033–1038.

L.A. Gatys, A.S. Ecker, and M. Bethge. 2015a. Texture Synthesis Using Convolutional Neural Networks. *Advances in Neural Information Processing Systems* 28 (May 2015), 262–270. http://arxiv.org/abs/1505.07376

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015b. A Neural Algorithm of Artistic Style. *CoRR* abs/1508.06576 (Aug 2015). http://arxiv.org/abs/1508.06576

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. *CoRR* abs/1406.2661 (June 2014). https://arxiv.org/abs/1406.2661

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proc. CVPR 2016*.

Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. 2001. Image Analogies. *Proc. SIGGRAPH 2001* (August 2001), 327–340.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2016. Image-to-Image Translation with Conditional Adversarial Networks. *CoRR* abs/1611.07004 (2016). arXiv:1611.07004 http://arxiv.org/abs/1611.07004

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. In *Proc. CVPR 2017*.

Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. 2016. Texture Synthesis with Spatial Generative Adversarial Networks. *CoRR* abs/1611.08207 (2016).

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *Proc. ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.), Vol. Part II. 694–711.

A. Kaspar, B. Neubert, D. Lischinski, M. Pauly, and J. Kopf. 2015. Self tuning texture optimization. *Computer Graphics Forum* 34, 2 (May 2015).

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. 2005. Texture optimization for example-based synthesis. *ACM Trans. Graph.* 24, 3 (Proc. SIGGRAPH 2005) (2005), 795–802.

Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. 2003. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.* 22, 3 (Proc. SIGGRAPH 2003) (2003), 277–286.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2016. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *CoRR* abs/1609.04802 (2016). arXiv:1609.04802 http://arxiv.org/abs/1609.04802

Sylvain Lefebvre and Hugues Hoppe. 2006. Appearance-space texture synthesis. *ACM Transactions on Graphics* 25, 3 (Proc. SIGGRAPH 2006) (2006), 541–548.

Chuan Li and Michael Wand. 2016. Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks. In *Proc. ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.), Vol. Part III. 702–716.

Yanxi Liu, Web-Chieh Lin, and James H. Hays. 2004. Near-Regular Texture Analysis and Manipulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)* 23, 3 (2004).

Amir Rosenberger, Daniel Cohen-Or, and Dani Lischinski. 2009. Layered Shape Synthesis: Automatic Generation of Control Maps for Non-Stationary Textures. *ACM Trans. Graph* 28, 5 (Dec. 2009), 107:1–9.

Omry Sendik and Daniel Cohen-Or. 2017. Deep Correlations for Texture Synthesis. *ACM Trans. Graph.* 36, 5, Article 161 (July 2017), 15 pages. https://doi.org/10.1145/3015461

Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S. Lempitsky. 2016. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. *CoRR* abs/1603.03417 (2016). arXiv:1603.03417 http://arxiv.org/abs/1603.03417

Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. 2009. State of the Art in Example-based Texture Synthesis. In *Eurographics 2009 State of The Art Reports*. Eurographics.

Li-Yi Wei and Marc Levoy. 2000. Fast texture synthesis using tree-structured vector quantization. *Proc. SIGGRAPH 2000* (2000), 479–488.

Y. Wexler, E. Shechtman, and M. Irani. 2007. Space-time completion of video. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 29, 3 (2007), 463–476.

Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph.* 22, 3 (Proc. SIGGRAPH 2003) (2003), 295–302.

Yang Zhou, Huajie Shi, Dani Lischinski, Minglun Gong, Johannes Kopf, and Hui Huang. 2017. Analysis and Controlled Synthesis of Inhomogeneous Textures. *Computer Graphics Forum (Proc. Eurographics 2017)* 36, 2 (2017).

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *CoRR* abs/1703.10593 (2017). arXiv:1703.10593 http://arxiv.org/abs/1703.10593