Co-occurrence Based Texture Synthesis

Anna Darzi\textsuperscript{1}, Itai Lang\textsuperscript{1}, Ashutosh Taklikar\textsuperscript{1}, Hadar Averbuch-Elor\textsuperscript{2}, and Shai Avidan\textsuperscript{1}

\textsuperscript{1} Tel Aviv University, Tel Aviv 6998701, Israel \
\{annadarzi, itailang, ashutosht\}@mail.tau.ac.il, avidan@eng.tau.ac.il \
\textsuperscript{2} Cornell Tech, Cornell University, New York NY 10044, United States
hadarelor@cornell.edu

Fig. 1. Given a texture exemplar (left), co-occurrence statistics are collected for different texture crops. Using these co-occurrences, our method can synthesize a variety of novel texture samples with desired local properties (right), similar to those of the corresponding crops from the texture exemplar.

Abstract. We model local texture patterns using the co-occurrence statistics of pixel values. We then train a generative adversarial network, conditioned on co-occurrence statistics, to synthesize new textures from the co-occurrence statistics and a random noise seed. Co-occurrences have long been used to measure similarity between textures. That is, two textures are considered similar if their corresponding co-occurrence matrices are similar. By the same token, we show that multiple textures generated from the same co-occurrence matrix are similar to each other. This gives rise to a new texture synthesis algorithm.

We show that co-occurrences offer a stable, intuitive and interpretable latent representation for texture synthesis. Our technique can be used to generate a smooth texture morph between two textures, by interpolating between their corresponding co-occurrence matrices. We further show an interactive texture tool that allows a user to adjust local characteristics of the synthesized texture image using the co-occurrence values directly.
1 Introduction

Methods for texture synthesis have been investigated extensively for decades. Most of these methods boil down to two fundamental questions. How to represent texture? and how to synthesize a new texture image, given the texture representation?

The seminal work of Heeger and Bergen [13] represents texture using various image statistics. They synthesized texture by starting with a noise image and iteratively matching its statistics to that of a target texture image.

Non-parametric approaches [7,32,6] model texture as a collection of patches. They synthesize texture by starting with a noisy seed and sampling the input texture image accordingly.

Texture synthesis in the Deep Learning era [9] follows the approach of [13]. Instead of matching image statistics, they match the statistics of various layers of a deep neural network. This approach was later improved by follow up papers [31,17,29].

Alternatively, one can use generative adversarial networks (GANs) [10] to synthesize textures that resemble the input exemplar [15,2]. This line of work lets the discriminator implicitly take the role of texture representation, while the generator is tasked with the goal of synthesizing the new texture.

In this work we use co-occurrence matrices, that were first introduced in the context of texture analysis [12], to represent texture. The key technical contribution of our work is a conditional GAN (cGAN) that synthesizes textures from a given co-occurrence matrix. That is, our algorithm takes an input texture and generates variations that match its local co-occurrence properties.

We synthesize texture using a cGAN that takes as input a co-occurrence matrix and a random seed vector. To modify the generated texture we can either edit the input co-occurrence matrix, or use it in conjunction with different random seed vectors (see Figure 2). We find that co-occurrence strikes a balance between non-parametric models that are difficult to control and understand, and fully parametric models that are not expressive enough.

Our co-occurrence based texture synthesis engine is utilized in a number of applications. First, we show that fixing a co-occurrence matrix and using various random seeds generates textures that are variations of the input texture. If, on the other hand, we fix the noise vector and change the co-occurrence matrix, the underlying properties of the texture image change accordingly. This can be used, for example, to interpolate between two texture regions by interpolating between their corresponding co-occurrence matrices, resulting in a dynamic texture morph (see horizontal axis in Figure 2).

In another use-case, we show how to control the synthesized texture by editing the co-occurrence input. Since the co-occurrence matrix has a clear interpretation, we can manually modify its entries to modify the appearance of the resulting texture.
Fig. 2. Our technique synthesizes texture given a co-occurrence matrix and a random seed vector. Above we illustrate textures generated from interpolating between two co-occurrence vectors (across the columns) and two random seed vectors (across the rows).

2 Related Work

There are different ways to model texture. Heeger and Bergen [13] represented textures as histograms of different levels of a steerable pyramid, while De Bonet [3] represented texture as the conditional probability of pixel values at multiple scales of the image pyramid. New texture is synthesized by matching the statistics of a noise image to that of the input texture. Portilla and Simoncelli [27] continued this line of research, using a set of statistical texture properties. None of these methods used co-occurrence statistics for texture synthesis.

Stitching-based methods [6,22,21] assume a non-parametric model of texture. In this case a new texture is generated by sampling patches from the input texture image. This sampling procedure can lead to deteriorated results and one way to fix that is to use an objective function that forces the distribution of patches in the output texture to match that of the input texture [33,30]. These methods were proved to be very effective in synthesizing plausible textures.

Some methods to interpolate between textures and not necessarily synthesize new texture have been proposed. For example, Matusik et al. [26] capture the structure of the induced space by a simplicial complex where vertices of the simplices represent input textures. Interpolating between vertices corresponds to interpolating between textures. Rabin et al. [28] interpolate between textures by averaging discrete probability distributions that are treated as a barycenter over the Wasserstein space. Soheil et al. [5] use the screened Poisson equation
Deep learning for texture synthesis by Gatys et al. [9] follows the approach of Heeger and Bergen [13]. Instead of matching the histograms the image pyramid, they match the Gram matrix of different features maps of the texture image, where the Gram matrix measures the correlation between features at selected layers of a neural network. This approach was later improved by subsequent works [31,29]. These methods look at the pair-wise relationships between features, which is similar to what we do. However, the Gram matrix measures the correlation of deep features, whereas we use the co-occurrence statistics of pixel values.

Alternatively, one can use a generative adversarial network (GAN) to synthesize textures that resemble the input exemplar. Li and Wand [23] used a GAN combined with a Markov Random Field to synthesize texture images from neural patches. Liu et al. [25] improved the method of Gatys et al. [9] by adding constraints on the Fourier spectrum of the synthesized image, and Li et al. [24] use a feed-forward network to synthesize diversified texture images. Zhou et al. [36] use GANs to spatially expand texture exemplars, extending non-stationary structures. Frühstück et al. [8] synthesize large textures using a pre-trained generator, which can produce images at higher resolutions.

Texture synthesis using GANs is also used for texture interpolation. Jetchev et al. [15] suggested using a spatial GAN (SGAN), where the input noise to the generator is a spatial tensor rather than a vector. This method was later extended to the periodic spatial GAN (PSGAN), proposed by Bergmann et al. [2]. Interpolating between latent vectors within the latent tensor results in a spatial interpolation in the generated texture image. These works focus on spatial interpolation, and learn the input texture’s structure as a whole. We focus on representing the appearance of different local regions of the texture in a controllable manner.

In their texture mixing work, Yu et al. [35] also performed texture interpolation in latent space, training their model on generating both single and mixed textures. While they generate interpolated textures, their method also relies on post-processing the edges between them, while we generate samples of texture based on just a latent co-occurrence matrix and noise, without post-processing.

Co-occurrences were introduced by Julesz [18] who conjectured that two distinct textures can be spontaneously discriminated based on their second order statistics (i.e., co-occurrences). This conjecture is not necessarily true because it is possible to construct distinct images that have the same first (histograms) and second order (co-occurrence) statistics. It was later shown by Yellott [34] that images with the same third-order statistics are indistinguishable. In practice, co-occurrences have long been used for texture analysis [12,14], but not for texture synthesis, as we propose in this work.
3 Method

We use a conditional generative adversarial network (cGAN) to synthesize textures with spatially varying co-occurrence statistics. Before diving into the details, let us fix notations first. A texture patch (i.e., a 64×64 pixels region) is represented by a local co-occurrence matrix. A texture crop (i.e., a 128×128 pixels region) is represented by a collection of local co-occurrence matrices, organized as a co-occurrence tensor. We train our cGAN on texture crops, because crops capture the interaction of neighboring co-occurrences. This allows the generator to learn how to synthesize texture that fits spatially varying co-occurrence statistics. Once the cGAN is trained, we can feed the generator with a co-occurrence tensor and a random seed to synthesize images of arbitrary size. We can do that because both the discriminator and generator are fully convolutional.

An overview of our cGAN architecture is shown in Figure 4. It is based on the one proposed by Bergmann et al. [2]. In what follows, we explain how the co-occurrence statistics are collected, followed by a description of our cGAN and how we use it to generate new texture images.

We use the co-occurrence matrix to represent the local appearance of patches in the texture image. For a given patch, we calculate the co-occurrence matrix $M$ of the joint appearance statistics of pixel values [16,20].

The size of the co-occurrence matrix scales quadratically with the number of pixel values and it is therefore not feasible to work directly with RGB values. To circumvent that we quantize the color space to a small number of clusters. The pixel values of the input texture image are first grouped into $k$ clusters using standard $k$-means. This results in a $k \times k$ co-occurrence matrix.

Let $(\tau_a, \tau_b)$ denote two cluster centers. Then $M(\tau_a, \tau_b)$ is given by:

$$M(\tau_a, \tau_b) = \frac{1}{Z} \sum_{p,q} \exp \left( -\frac{d(p,q)^2}{2\sigma^2} \right) K(I_p, \tau_a) K(I_q, \tau_b),$$

(1)

where $I_p$ is the pixel value at location $p$, $d(p,q)$ is the Euclidean distance between pixel locations $p$ and $q$, $\sigma$ is a user specified parameter, and $Z$ is a normalizing factor designed to ensure that the elements of $M$ sum up to 1.

$K$ is a soft assignment kernel function that decays exponentially with the distance of the pixel value from the cluster center:

$$K(I_p, \tau_l) = \exp \left( -\sum_i \frac{(I^i_p - \tau^i_l)^2}{(\sigma^i_l)^2} \right),$$

(2)

where $i$ runs over the RGB color channels and $\sigma^i_l$ is the standard deviation of color channel $i$ of cluster $l$.

The contribution of a pixel value pair to the co-occurrence statistics decays with their Euclidean distance in the images plane. In practice, we do not sum over all pixel pairs $(p,q)$ in the image, but rather consider only pixels $q$ within a window around $p$. An illustrative image and its corresponding co-occurrence matrix is given in Figure 3.
We collect the co-occurrence statistics of an image crop $x$ of spatial dimensions $h \times w$, according to Equations 1 and 2. We row-stack each of these matrices to construct a co-occurrence volume of size $h \times w \times k^2$. This volume is then downsampled spatially by a factor of $s$, to obtain a co-occurrence tensor $C$. This downsampled volume is meant to allow more variability for the texture synthesis process, implicitly making every spatial position in $C$ a description of the co-occurrence of a certain receptive field in $x$.

### 3.1 Texture Synthesis

We denote a real texture crop as $x_r$, and its co-occurrence tensor $C_r$. The random noise seed tensor is denoted as $z$, where its entries are drawn from a normal distribution $\mathcal{N}(0, 1)$. In order to balance the influence of the co-occurrence and the noise on the output of the generator, we also normalise $C_r$ to have zero mean and unit variance.

The concatenated co-occurrence and noise tensor are given as input to the generator. The output of the generator $G$ is a synthesized crop marked as $x_g = G(C_r, z)$. The corresponding downsampled co-occurrence of $x_g$ is marked $C_g$.

The input to the discriminator $D$ is a texture crop that is either a real crop from the texture image ($x_r$) or a synthetic one from the generator ($x_g$). In addition to the input texture crop, the original co-occurrence tensor $C_r$ is also provided to the discriminator. In case of a real input texture, this is its corresponding co-occurrence tensor. For a synthetic input, the same co-occurrence condition given to the generator is used at the discriminator.

We emphasise that both the generator and discriminator are conditioned on the co-occurrence tensor. With this cGAN architecture, the discriminator teaches the generator to generate texture samples that preserve local texture properties, as captured by the co-occurrence tensor. In order to further guide
the generator to respect the co-occurrence condition when synthesizing texture, we compute the co-occurrence statistics of the generated texture and demand consistency with the corresponding generator’s input.

We train the network with Wasserstein GAN with gradient penalty (WGAN-GP) objective function, proposed by Gulrajani et al. [11]. In addition, we add a novel consistency loss on the co-occurrence of the generated texture and the input condition.

The generator is optimized according the the following loss function:

$$L(G) = -\mathbb{E}_{x_{g} \sim P_{g}}[D(G(z, C_{r}), C_{r})] + \lambda_{M} |M_{g} - M_{r}|_{1},$$

where $D$ is the discriminator network, and $\cdot |_{1}$ is $L_{1}$ loss.

The discriminator is subject to the following objective:

$$L(D) = \mathbb{E}_{x_{r} \sim P_{r}}[D(x_{r}, C_{r})] - \mathbb{E}_{x_{g} \sim P_{g}}[D(x_{g}, C_{r})] + \lambda_{p} \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x}, C_{r})\|_{2} - 1)^{2}],$$

where $\hat{x}$ is a randomly sampled as a liner interpolation between real and synthetic textures $\hat{x} = u x_{r} + (1 - u)x_{g}$, with a uniform interpolation coefficient $u \sim U[0, 1]$.

The training is done with batches of texture samples and their corresponding co-occurrence tensors. When training the discriminator, the co-occurrences of the real texture samples are used to generate the synthetic samples. For the generator training, only the co-occurrences of the batch are used.

**Fig. 4. Method Overview.** The generator receives as input a co-occurrence tensor $C_{r}$, corresponding to a real texture crop $x_{r}$. This tensor is concatenated to a random noise tensor $z$. The generator outputs a synthetic texture sample $x_{g}$. The discriminator's input alternates between real and synthetic texture samples. It also receives the co-occurrence tensor $C_{r}$. In addition, the co-occurrence of the synthesized texture is required to be consistent with the input statistics to the generator. By this architecture, the generator learns to synthesize samples with desired local texture properties, as captured by the co-occurrence tensor.
4 Evaluation

4.1 Experimental Details

The texture images that we use are taken from the Describable Textures Dataset [4], texture images from the work of Bellini et al. [1] and several texture images we found online. For all texture images we keep $k$ either equal to 2, 4 or 8 clusters. We collect the co-occurrence statistics for each pixel. We take a patch of $65 \times 65$ around that pixel, and calculate the co-occurrence statistics using a window of size $51 \times 51$ around each pixel in that patch and set $\sigma^2$ in Equation 1 to 51. So, for each pixel we have $k \times k$ co-occurrence matrix, which is reshaped to $k^2$, and for an image of $w \times h \times 3$, we have a co-occurrence volume of $w \times h \times k^2$.

The dataset for a texture image contains $N = 2,000$ crops of $n \times n$ pixels. As our work aims at extracting and analyzing local properties, we use $n = 128$. The down sampling factor of the co-occurrence volume is $s = 32$. Thus, the spatial dimensions of the condition co-occurrence tensor are $4 \times 4$.

The architecture of the generator and discriminator is fully convolutional, thus larger textures can be synthesized at inference time. The architecture is based on that of PSGAN [2]. The generator in our case is a stack of 5 convolutional layers, each having an upsampling factor of 2 and filter of size $5 \times 5$, stride of 1 and a padding of 2, with ReLU activation and batch normalization.

The discriminator too is a stack of 5 convolutional layers, with filter size of $5 \times 5$, stride of 2 and padding of 2, with sigmoid activation. The stride here is 2 to bring down the upscaled spatial dimensions by the generator back to original input size. After the activation of the third layer, we concatenate the co-occurrence volume in the channel dimension, to help the discriminator condition on the co-occurrence as well (see Figure 4).

For each texture image, we train our method for 120 epochs using Adam optimizer with momentum of 0.5. The learning rate is set to 0.0002, and the gradient penalty weight is $\lambda = 1$. The training time is about 7 hours using a NVIDIA Tesla V100 GPU. Our code is publicly available at https://github.com/Ashu7397/cooc_texture.

4.2 Evaluation Tests

Fidelity and Diversity. Texture synthesis algorithms must balance the contradicting requirements of fidelity and diversity. Fidelity refers to how similar the synthesized texture is to the input texture. Diversity expresses the difference in the resulting texture, with respect to the input texture. We maintain fidelity by keeping the co-occurrences fixed and achieve diversity by varying the noise seed.

Figure 1 shows that the generated samples resemble their corresponding texture crop. Utilizing the adversarial loss during training can be viewed as an implicit demand of this property, while the the co-occurrence loss requires is more explicitly. Additional fidelity and diversity results are presented in the supplementary.
We demonstrate the smoothness of the latent space by interpolating in two axes: between co-occurrence tensors and between noise tensors. As shown in Figure 2, interpolation between two different co-occurrences results in a smooth traverse between different texture characteristics. On the other hand, for a given co-occurrences tensor, interpolation between different noise tensors smoothly transitions between texture samples with similar properties. In addition, any intermediate result of the interpolation yields a plausible texture result. This suggests that the generator learned a smooth texture manifold, with respect to both co-occurrence and noise.

**Nearest Neighbors in the Training Set.** To verify that our network is truly generating novel, unseen texture samples and not simply memorizing seen examples, we examine the nearest neighbors in the training set. To do so, we generate textures using unseen co-occurrences from the test set and search for their nearest neighbors in the training set, in terms of $L_1$ of RGB values. For comparison, we also compute the co-occurrence of the generated samples and look for nearest neighbor, in $L_1$ sense, in the co-occurrence space.

We demonstrate the results of this experiment in Figure 5. The spatial arrangement of nearest neighbors in RGB space resembles that of the synthesized texture, yet they are not identical. Nearest neighbors in the co-occurrence space may have different spatial arrangement. In any case, the synthesized texture samples differ from their nearest neighbors in both the co-occurrence and RGB measures.

**Fig. 5. Novelty of generated texture samples.** Left: Generated textures using co-occurrences from the test set. For each generated sample, we find its nearest neighbors in the training set, measured in terms of RGB values (middle) and co-occurrence (right). As the figure illustrates, the generated textures are different from their nearest neighbors in the training set.
Stability of Synthesized Textures. We use a texture degradation test \[19,29\] to measure the stability of our algorithm. Specifically, we do the following: Given a co-occurrence tensor from the train set, we generate a texture sample, compute its co-occurrence tensor, and feed it back to the generator (with the same noise vector) to generate another sample. This process is repeated several times.

Figure 6 shows representative results of our stability test. As the figure illustrates, the appearance of the synthesized textures remains roughly the same. We attribute this stable behaviour to the use of the co-occurrence as a condition for the texture generation process.

To quantitatively evaluate the stability of our method, we repeated this test for the entire test set of different texture images. Following each looping iteration, we measure the \(L_1\) difference between the input co-occurrence and that of the generated sample and average over all the examples in the set. For 10 looping iterations the average \(L_1\) measure remains within a 2\% range with respect to the average \(L_1\) measure of the first iteration. This was the case on all the examined textures, indicating that the co-occurrence of the generated samples is indeed stable.

![Fig. 6. Stability test. Textures were generated in a loop. In the first iteration, co-occurrences are taken from the test set. In the next, co-occurrence of the generated texture is used. The original crop from the texture exemplar is on the left and the generated sequence for eight iterations is on the right. It can be noted that the textures remain stable over the iterations.](image)

Limitations. The quality of our results degrade when texture elements are larger than the crop size we use (i.e., 128 × 128 pixels). The method also fails to capture texture characterized by global structure. We show examples of these kinds of textures in Figure 7.
Fig. 7. Limitation example. When the input texture includes elements of scale larger than our crop size of 128 × 128, they are not preserved in the generated samples.

5 Applications

We demonstrate our co-occurrence based texture synthesis in the context of several applications. These include a novel method for generating a dynamic texture morph from a single texture image, texture tiling and interactive texture editing.

5.1 Texture Morphing

We generate a dynamic texture morph by interpolating and extrapolating between randomly selected co-occurrence tensors of various local texture regions. The generated image sequence, obtained by conditioning on the smooth co-occurrence sequence and a fixed random noise seed, exhibits a unique temporal behavior. In Figure 8, we illustrate a few representative frames along the sequence.

Simply interpolating between the textures in RGB space will not do. This yields a smooth transition between pixel values but does not create the same dynamic morphing effect. Instead, the interpolated images are merely a blend of the two source samples (see Figure 9). Extrapolation in RGB space clearly fails.

A more advanced alternative is to use a nearest-neighbor scheme, where we interpolate the co-occurrence tensors of the source and target samples, and use it to retrieve the most similar texture (in co-occurrence sense) from the training set. This limits the resulting sequence to examples in the training set, whereas we can generate never-seen-before samples. On top of that, the smoothness of the resulting sequence depends on the sample density of the training set. In practice, we have a limited set, which inevitably leads to a non-smooth sequence.

5.2 Synthesizing larger texture images

Using a cGAN for training allows us to seamlessly synthesize texture images of arbitrary size (see Figure 10). The top example shows how we collect a co-
**Fig. 8. Texture interpolation and extrapolation.** Examples of interpolated and extrapolated textures, generated between two sample co-occurrence vectors (corresponding to the generated images marked in green). As the figure illustrates, the extrapolated examples extend the smooth interpolation sequence, magnifying the differences in the local texture properties.

**Fig. 9.** Generating dynamic sequences with alternative techniques. Above, we illustrate the results obtained using a naive interpolation between RGB values of the samples marked in green (top) and a nearest neighbor retrieval of the smooth co-occurrence sequence (bottom). For comparison purposes, we use the samples illustrated in Figure 8. As illustrated above, temporal sequences between texture samples cannot be easily obtained without our method.
occurrence tensor from one image and use it to synthesize an image of a different size. The bottom two examples show how we can fine tune our control by spatially concatenating the co-occurrences of several texture crops and feeding them to the generator. This gives a coherent result, with smooth transition between the local properties of the synthesized crops. For comparison, simply tiling multiple synthesized texture crops creates noticeable seams. In the supplementary we provide additional results of synthesizing large textures with control.

![Fig. 10. Large texture synthesis.](image)

(Top) We collect a co-occurrence tensor from the input image (left) and use it to synthesize an image of a different size (right). (Bottom) Synthesizing multiple texture crops and tiling them together controls the local properties of texture at the cost of noticeable seams (left example in each pair). The generator, however, is able to create a single large texture image that respects local statistics while avoiding the noticeable seams (right example in each pair).

### 5.3 Interactive Texture Editing

We edit texture by editing its co-occurrence matrix. It is possible since co-occurrences have a clear and intuitive interpretation. To edit a texture, we present the user with a co-occurrence matrix and let her modify it. Once the user modified bin $M(\tau_i, \tau_j)$ in the co-occurrence matrix, we re-normalize the matrix to sum to 1. Then, we generate a texture image with the modified co-occurrence.
Figure 11 shows results obtained using our interactive editing system. As can be seen, we can alter the appearance of the synthesized texture locally. Additional texture editing results can be seen in the supplemental material.

Fig. 11. Interactive co-occurrence editing. A user can adjust a given co-occurrence matrix and its corresponding generated image. In our interactive system, users can select a bin in the matrix to modify. In the middle row we illustrate representative samples conditioned on co-occurrences, obtained by multiplying a selected bin with an increasing factor. The top row and bottom row correspond to editing the co-occurrence of the top left bottom right regions, respectively.

6 Conclusion

We proposed a co-occurrence based texture synthesis method. Local texture properties are captured by a co-occurrence tensor. While the computation of the co-occurrence for a given texture crop is deterministic, the opposite direction is not. Meaning, different texture crops can have similar co-occurrence statistics. Our cGAN managed to learn a mapping form the co-occurrence space back to the image space. It generates new texture samples with the desired local properties.

We further show how to interpolate between different textures by interpolating between their corresponding co-occurrences. This can be used to generate a temporally morphing texture from a static exemplar. In addition, we let the user edit textures by editing their corresponding co-occurrence matrix.

We think that co-occurrences strikes the right balance between non-parametric methods that are difficult to control and manipulate and parametric methods that might have limited expressive power. Our experiments demonstrate the capabilities of this approach.
References

1. Bellini, R., Kleiman, Y., Cohen-Or, D.: Time-varying weathering in texture space. ACM Transactions on Graphics (TOG) 35(4), 141 (2016) 8
2. Bergmann, U., Jetchev, N., Vollgraf, R.: Learning texture manifolds with the periodic spatial gan. arXiv preprint arXiv:1705.06566 (2017) 2, 4, 5, 8
3. Bonet, J.S.D.: Multiresolution sampling procedure for analysis and synthesis of texture images. In: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997. pp. 361–368 (1997). https://doi.org/10.1145/258734.258882, https://doi.org/10.1145/258734.258882 3
4. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., Vedaldi, A.: Describing textures in the wild. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014) 8
5. Darabi, S., Shechtman, E., Barnes, C., Goldman, D.B., Sen, P.: Image Melding: Combining inconsistent images using patch-based synthesis. ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2012) 31(4), 82:1–82:10 (2012) 3
6. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 341–346. ACM (2001) 2, 3
7. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: IEEE International Conference on Computer Vision. pp. 1033–1038. Corfu, Greece (September 1999) 2
8. Frühstück, A., Alhashim, I., Wonka, P.: Tilegan: synthesis of large-scale non-homogeneous textures. ACM Transactions on Graphics (TOG) 38(4), 1–11 (2019) 4
9. Gatys, L., Ecker, A.S., Bethge, M.: Texture synthesis using convolutional neural networks. In: Advances in neural information processing systems. pp. 262–270 (2015) 2, 4
10. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial nets. In: NIPS. pp. 2672–2680 (2014) 2
11. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: Proceesings of the Annual Conference on Neural Information Processing Systems (NIPS) (2017) 7
12. Haralick, R.M., Shanmugam, K.S., Dinstein, I.: Textural features for image classification. IEEE Trans. Systems, Man, and Cybernetics 3(6), 610–621 (1973) 2, 4
13. Heeger, D.J., Bergen, J.R.: Pyramid-based texture analysis/synthesis. In: Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques. pp. 229–238. SIGGRAPH ’95, ACM, New York, NY, USA (1995). https://doi.org/10.1145/218380.218446, http://doi.acm.org/10.1145/218380.218446 2, 3, 4
14. Isola, P., Zoran, D., Krishnan, D., Adelson, E.H.: Crisp boundary detection using pointwise mutual information. In: Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part III. pp. 799–814 (2014) 4
15. Jetchev, N., Bergmann, U., Vollgraf, R.: Texture synthesis with spatial generative adversarial networks. Workshop on Adversarial Training, NIPS (2016) 2, 4
16. Jevnisek, R.J., Avidan, S.: Co-occurrence filter. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3184–3192 (2017)

17. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II. pp. 694–711 (2016)

18. Julesz, B.: Visual pattern discrimination. Information Theory, IRE Transactions on 8, 84 – 92 (03 1962). https://doi.org/10.1109/TIT.1962.1057698

19. Kaspar, I., Neubert, B., Lischinski, D., Pauly, M., Kopf, J.: Self tuning texture optimization. Computer Graphics Forum 34(2), 349–359 (2015)

20. Kat, R., Jevnisek, R., Avidan, S.: Matching pixels using co-occurrence statistics. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1751–1759 (2018)

21. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. ACM Transactions on Graphics (ToG) 24(3), 795–802 (2005)

22. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: image and video synthesis using graph cuts. ACM Transactions on Graphics (ToG) 22(3), 277–286 (2003)

23. Li, C., Wand, M.: Precomputed real-time texture synthesis with markovian generative adversarial networks. In: European Conference on Computer Vision. pp. 702–716. Springer (2016)

24. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Diversified texture synthesis with feed-forward networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 266–274 (07 2017). https://doi.org/10.1109/CVPR.2017.36

25. Liu, G., Gousseau, Y., Xia, G.S.: Texture synthesis through convolutional neural networks and spectrum constraints. In: 23rd International Conference on Pattern Recognition (ICPR). pp. 3234–3239. IEEE (2016)

26. Matusik, W., Zwicker, M., Durand, F.: Texture design using a simplicial complex of morphable textures. ACM Trans. Graph. 24(3), 787–794 (Jul 2005). https://doi.org/10.1145/1073204.1073262, http://doi.acm.org/10.1145/1073204.1073262

27. Portilla, J., Simoncelli, E.P.: A parametric texture model based on joint statistics of complex wavelet coefficients. International journal of computer vision 40(1), 49–70 (2000)

28. Rabin, J., Peyré, G., Delon, J., Bernot, M.: Wasserstein barycenter and its application to texture mixing. In: Bruckstein, A.M., ter Haar Romeny, B.M., Bronstein, A.M., Bronstein, M.M. (eds.) Scale Space and Variational Methods in Computer Vision. pp. 435–446. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

29. Sendik, O., Cohen-Or, D.: Deep correlations for texture synthesis. ACM Transactions on Graphics (TOG) 36(5), 161 (2017)

30. Simakov, D., Caspi, Y., Shechtman, E., Irani, M.: Summarizing visual data using bidirectional similarity. In: 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA (2008)

31. Ulyanov, D., Lebedev, V., Vedaldi, A., Lempitsky, V.S.: Texture networks: Feed-forward synthesis of textures and stylized images. In: Proceedings of the International Conference on Machine Learning (ICML). pp. 1349–1357 (2016)
32. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. pp. 479–488. ACM Press/Addison-Wesley Publishing Co. (2000) 2

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

34. Yellott, J.I.: Implications of triple correlation uniqueness for texture statistics and the julesz conjecture. J. Opt. Soc. Am. A 10(5), 777–793 (May 1993). https://doi.org/10.1364/JOSAA.10.000777, http://josaa.osa.org/abstract.cfm?URI=josaa-10-5-777 4

35. Yu, N., Barnes, C., Shechtman, E., Amirghodsi, S., Lukac, M.: Texture mixer: A network for controllable synthesis and interpolation of texture. arXiv preprint arXiv:1901.03447 (2019) 4, 18, 22, 23, 24

36. Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D., Huang, H.: Non-stationary texture synthesis by adversarial expansion. ACM Transactions on Graphics (TOG) 37(4) (2018) 4
Supplementary

In the following sections we provide additional results of our texture synthesis method, and compare it to the recent Texture Mixer work [35].

A  Fidelity and Diversity

Figure 12 presents additional results, demonstrating the fidelity and diversity of the generated texture samples. See Section 4.2 in the manuscript for more details.

B  Co-occurrence Editing

In Section 5.3 in the main body, we showed how to manipulate texture appearance by editing the input co-occurrence condition to the generator. Here we present additional examples of this application in Figure 13.

C  Controllable Large Texture Synthesis

Our texture synthesis algorithm is fully convolutional and thus can generate arbitrary large textures. In addition, since it is conditioned on co-occurrence, we have control over the local appearance of the synthesized texture.

In order to demonstrate this ability, we take two crops of 128 × 128 pixels with different properties form a texture exemplar. Then, we tile their corresponding co-occurrence tensors in a desired spatial arrangement, and run it through generator. This way obtain a texture with desired appearance.

Figure 14 shows synthesized textures of size 2816 × 1408 pixels. The fidelity to the co-occurrence condition enables us to depict the number 2020. The diversity property of our algorithm enables to produce these large size textures with a plausible look.

D  Comparison to Texture Mixer

A recent work, called Texture Mixer [35], proposed a method for controllable texture synthesis and interpolation. We compare our approach to theirs in three aspects: fidelity and diversity, stability and interpolation between texture regions with different properties. We use their publicly available model, that was trained on earth textures. Results are presented in Figures 15, 16 and 17. As the figures show, our method tends to have better fidelity to the input texture than Texture Mixer, it is more stable and the interpolations with our method are smoother.
Fig. 12. Examples of generated texture crops from different co-occurrence tensors and noise vectors. Note that all these conditional co-occurrence were taken from the test set, unseen during training. The synthesized texture samples from the same co-occurrence condition and different noise seeds differ from each other, while respecting the properties of the corresponding crop from the texture image.
Fig. 13. **Additional example of co-occurrence editing.** In this example, we increase the co-occurrence of the black color with itself. The synthesized texture before editing appears in the center of the middle row (marked in green). The edit of the top left and bottom right regions of the texture is shown from the center to the left and right, respectively. The corresponding co-occurrence matrices are at the top and bottom rows. The edited bin is marked in red. Darker color in the matrix represents higher value. It can be seen that presence of the black color is increased locally, at the desired regions. It is done gracefully in a plausible behaviour of the texture - an increase of black honey cones for the top example and a larger dark tile for the bottom example.
Fig. 14. Generating large textures with control. Textures of size $2816 \times 1408$ pixels, produced by our method. We can control the local appearance of the texture, by constructing the co-occurrence condition to the generator network. For these examples, we used co-occurrences from only two $128 \times 128$ crops of the original textures!
Fig. 15. Fidelity and diversity comparison. For each texture, the input crop is on the left column. Synthesized samples with different noise seeds for Texture Mixer [35] and for our method appear on the middle and right columns, respectively. While both methods generate diverse results, our method seems to better respect the properties of the input texture crop.
Fig. 16. Stability comparison. For each texture, the input crop is on the left. A sequence of texture samples generated iteratively with Texture Mixer [35] (top row) and with our method (bottom row) is shown on the right. While Texture Mixer suffers from sever artifacts along the iterations, our texture synthesis method remains very stable.
Fig. 17. Interpolation comparison. For each texture, the top, middle and bottom rows correspond to $\alpha$-blending, interpolation with Texture Mixer [35] and interpolation with our method. The interpolation stripes are of size $1024 \times 128$ pixels. Our interpolations tend to be smoother and less repetitive than those of Texture Mixer.