Scraping Textures from Natural Images for Synthesis and Editing

Xueting Li[1], Xiaolong Wang[2],
Ming-Hsuan Yang[1], Alexei A. Efros[3], and Sifei Liu[4]

[1] UC Merced [2] UC San Diego [3] UC Berkeley [4] NVIDIA

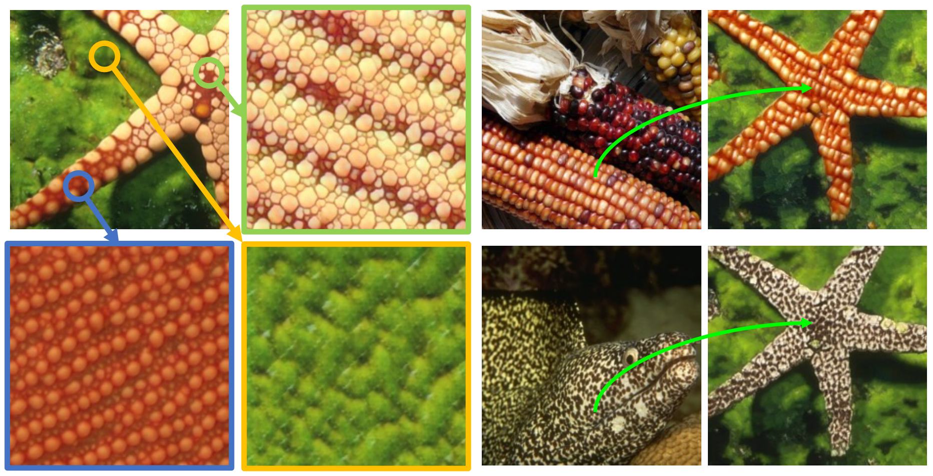

Fig. 1: **Texture synthesis and editing.** (a) Scraping textures from a natural image (upper left) to synthesize texture images. (b) Scraping textures from one image (left) to transfer onto another image (right).

Abstract. Existing texture synthesis methods focus on generating large texture images given a small texture sample. But such samples are typically assumed to be highly curated: rectangular, clean, and stationary. This paper aims to scrape textures directly from natural images of everyday objects and scenes, build texture models, and employ them for texture synthesis, texture editing, etc. The key idea is to jointly learn image grouping and texture modeling. The image grouping module discovers clean texture segments, each of which is represented as a texture code and a parametric sine wave by the texture modeling module. By enforcing the model to reconstruct the input image from the texture codes and sine waves, our model can be learned via self-supervision on a set of cluttered natural images, without requiring any form of annotation or clean texture images. We show that the learned texture features capture many natural and man-made textures in real images, and can be applied to tasks like texture synthesis, texture editing and texture swapping.

Keywords: Texture Synthesis; Segmentation and Grouping

Project page: https://sunshineatnoon.github.io/texture-from-image/.

1 Introduction

Texture synthesis aims at generating a texture image of arbitrary size given a small texture example. Existing texture synthesis approaches [13,30,58,65,42] rely on carefully curated texture datasets [6,8,10], where each image is rectangular and only includes a single texture pattern. Collecting such datasets requires tedious human effort. At the same time, most natural images already contain abundant textures found on most objects and materials (e.g., the starfish, the corn, and the snake in Fig. 1). A question ensues – can we *scrape* textures directly from natural images, i.e., learn texture models in an unsupervised fashion?

This is a highly challenging task since: a) a natural image usually includes multiple different texture regions that cannot be readily used as supervision for texture feature learning, and b) the shapes of these texture regions are difficult to discover without segmentation annotation. This is why existing texture synthesis methods [13,12,16,17,33,37,65,58] require clean, rectangular texture patches as training data. Yet, even with clean texture patches, modeling and synthesizing diverse real world texture patterns is non-trivial. Contemporary methods mainly have two ways to synthesize texture given texture patches: a) encoding texture statistics in spatial features and resorting to transpose convolution for synthesis [45,42] or b) map a Gaussian distribution to a texture manifold [58,37,65]. However,the former method can only synthesize new texture image of limited size while the latter method fails to capture structural textures (e.g., the grid pattern in Fig. 3) due to a lack of structure in the Gaussian noise input. Furthermore, the limited representation capacity of neural networks may preclude the model to generalize to unseen images and produces inferior texture synthesis results.

To resolve these issues, we introduce a texture encoder that decomposes an image into a small number of clean texture segments and encode the texture pattern in each segment with a relatively low-dimensional vector texture code. We then use a decoder to reconstruct the input image given these texture codes. Our key insight is, by limiting the number of texture codes, we create an information bottleneck. In order to reconstruct the input image from this information bottleneck, our model is forced to group pixels of the same texture together without any segmentation annotation. Overview of our method is shown in Fig. 2.

Furthermore, to faithfully model the more structured textures in natural images, we introduce a parametric sine wave for each texture segment, besides the vector texture code. The periodicity of this sine wave allows our model to capture both regular (e.g., the corn in Fig. 3) and irregular (e.g., the zebra in Fig. 3) texture patterns, as well as enables arbitrary-sized texture synthesis during inference. To efficiently generalize our model to unseen images, we first train the model on a natural image dataset such that the encoder can learn grouping prior from divergent samples. We then generalize the model to unseen images with test-time adaptation [35,38,57].

During inference, our model can be readily applied for arbitrary-sized texture synthesis, editing and swapping as shown in Fig. 1. Besides, it also produces texture segments that often adhere nicely to object boundaries as shown in Fig. 6.

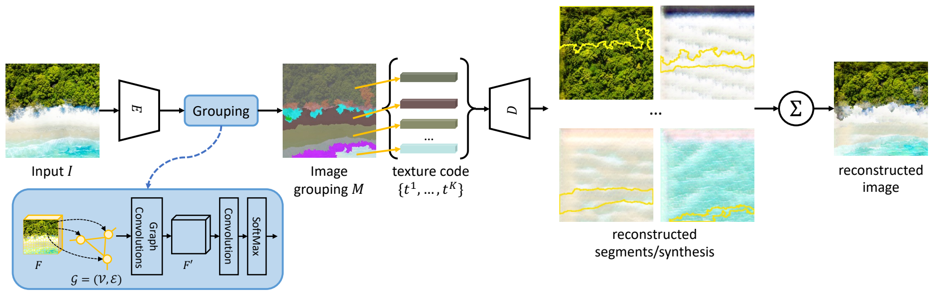

Fig. 2: **Overview.** Given a natural image I, the grouping module (blue box) produces a grouping mask M. For each group, we reconstruct the corresponding texture segment (shown inside the yellow contour) and extend it through texture synthesis. The reconstructed image is composited from all reconstructed texture segments and shown in the last column. Detailed illustration can be found in the supplementary.

The key contribution of our work is a method for jointly learning image grouping and texture modeling from natural images using an autoencoder formulation. The rest of the paper will discuss prior work, describe our method in detail, and present results.

2 Related Work

Texture synthesis aims to generate a texture image conditioned on a given texture exemplar, while image grouping attempts to cluster pixels into segments based on their appearance. Since our work learns the two tasks jointly, we discuss the related work in both domains.

2.1 Texture Synthesis

Scraping Textures from Natural Images. Most existing texture synthesis methods require a rectangular and clean texture patch as exemplar. As a result, large amount of efforts have been made to collect and clean texture datasets [6,8,10]. A few methods tried to overcome this limitation and learn texture synthesis directly using natural images by making strong assumptions of the training images. For instance, Bergmann et al. [5] proposed PSGAN to encode few texture patterns in a natural image by learning a model with small patches cropped from an ultra-resolution image. However, the strong assumption in PSGAN – small patches cropped from ultra-resolution image only include one texture pattern easily breaks for normal sized natural images. Closest to our work, Rosenberger et al. [52] propose to decompose an image into different texture layers and synthesize larger texture images layer by layer. However, their model is only applicable to a very specific type of natural textures, where grouping process can simply be carried out by clustering on raw pixels.

Texture Synthesis from Well-cropped Patches. Most existing texture synthesis methods learn from well-cropped texture patches. Existing methods can roughly be categorized into non-parametric [13,12,60] and optimization-based [17,29,58,65,42,5,63].

The non-parametric methods sequentially search and copy a pixel [13] or a patch [12] from the given texture exemplar that best fits for the current location. By bypassing the challenging task of analysing and modeling the complex structure and statistics of texture patterns, the algorithm produces realistic results that resemble the given exemplar. However, the sequential texture growing process can be unstable – it may start to produce "garbage" texture should it fail to find a suitable patch fitting the current location. It also takes a long time for high-resolution texture image synthesis without a specially designed mechanisms for efficient searching [60,4].

One line of optimization-based methods [22,50,62,33,30,17,54,23] try to map a prior distribution (e.g., a Gaussian noise) to a texture manifold by repeatedly modifying an initial noise image until it presents the same pattern as the given texture exemplar. For instance, the seminal work from Gatys et al. [17] starts with a random noise image and iteratively updates it by comparing the Gram matrices of the output and the exemplar. However, these methods suffer from the slow optimization process that has to be carried out for each given exemplar. They are also weaker at modeling structured textures than non-parametric methods due to a lack of structure in the noise image.

Another line of optimization-based works learn deep neural networks to speed up the synthesis process. They either encode the statistics of a single texture [29,58,68,55], or a limited number of textures [36,2,5,16,27,65] in a model while failing to generalize to unseen textures. Recently, two works [45,42] successfully produced a single model to synthesize unseen textures. They represent texture patterns by spatial feature maps and either pose texture synthesis as image upsampling in the Fourier domain or formulate texture expansion as transposed convolutions. However, all these texture synthesis methods require well-cropped texture patches as exemplar images and can only expand the texture to a limited size.

2.2 Perceptual Grouping and Image Segmentation

Perceptual grouping refers to the principles, first outlined by the Gestaltists [61] by which the visual system groups elements of the visual world into higher-order perceptual units (i.e. image regions). In computer vision, the area of image segmentation has been focused on operationalizing these principles, but despite some early successes, e.g. the classic Normalized Cuts algorithms [56], this problem still remains largely unsolved. Indeed, most subsequent methods gave up on partitioning an image into large, perceptual segments and instead focused on over-segmenting an image into superpixels [51], based on either hand-crafted low-level features such as color [1,14,15], texture and brightness [51,56] or deep features learned by neural networks [26,64,59]. The produced superpixels are compact and preserve sufficient local image structure that can be utilized for downstream tasks. Some later image segmentation approaches focus on producing larger segments by merging the pre-computed superpixels [51,25,49,44,39,40]. For example, the MCG method [49] produces region proposals by aligning segmentation from the normalized cuts algorithm [56] at different resolutions. The DeepGrouping method [39] learns a hierarchical graph neural network [53] and utilizes the features from the low- to high-level for texture, material, part, and object segmentation, respectively. In this work, we learn image grouping to produce texture segments that we then use as supervision for texture model learning. Our model learns to group pre-computed superpixels [1] into large texture segments without using any segmentation or grouping annotation as supervision.

2.3 Shape and Appearance Disentanglement

Disentangling an image into shape and appearance has been extensively studied in the past decades. Classical methods [19,20] such as the Primal Sketch [19] decomposes an image into "sketchable" and "non-sketchable" components, which essentially capture the structure and texture of the image respectively. More recent works [47,69,34] disentangle an image into structure and texture represented as deep features. The structure component is either represented as a spatial feature map [47] or approximated by a set of primitives [69,34]. The texture is encoded as a vector feature [47] or simply RGB values [34] or pre-defined stroke patterns [69]. Our model explicitly decomposes an image into structure represented as image grouping and texture captured by vector texture codes and sine waves.

3 Method

We propose a texture-based image auto-encoder that decomposes a natural image into a set of clean texture segments, each of which is represented by a 256-dimension texture code and a parametric sine wave. Our key idea is to create a information bottleneck with low-dimensional representations, so that when they are used to reconstruct the input images, the model is encouraged to: (i) cluster pixels belonging to the same texture, and (ii) learn compact texture representation for each segment.

In the following, we first provide a high-level overview of the proposed framework in Section 3.1, then, we describe the encoder that is responsible for texture grouping and representation learning in Section 3.2. In Section 3.3, we introduce the decoder that reconstructs the input image from the compact texture representations. We describe how to employ our model for texture synthesis in Section 3.4. Finally, the self-supervised training objectives are discussed in Section 3.5.

3.1 Overview

The framework of the proposed algorithm is shown in Fig.2. Given an input image $I \in \mathcal{R}^{3\times H\times W}$, the encoder E maps it to a per-pixel feature map $F \in \mathcal{R}^{C_F\times H\times W}$. The grouping module (blue box in Fig. 2) takes the feature map F as input and produces a grouping map $M \in \mathcal{R}^{K\times H\times W}$ that assigns each pixel to one of K groups. Each group represents a texture pattern in the image and is encoded by a vector texture code $t^k \in \mathcal{R}^{C_T}$ and a learnable sine wave $S^k, k = 1, \ldots, K$. For each texture segment, the decoder D is tasked to simultaneously learn to reconstruct it and synthesize a similar texture image.

3.2 Texture Encoder

The texture encoder aims to discover a set of clean texture segments from a highly cluttered natural image and encode their texture patterns for texture synthesis. To this end, we propose to build a graph neural network on top of superpixels [51,1,64,26] to group superpixels of the same texture together. The rationale of using superpixels instead of pixels is that superpixels are compact, consistent with boundaries between different texture regions while preserving sufficient receptive field for feature learning. Meanwhile, a graph neural network allows nearby superpixels to share or differentiate features depending on their similarities and yields features fit for the final grouping mask computation. We note that an alternative solution to obtain texture segments is by directly utilizing readily available semantic segmentation masks obtained by manual annotation [67,41,3] or pre-trained models [43,7]. However, these masks are defined based on semantic prior, where each region may contain several different texture patterns, e.g., a human face includes hair, skin, eyes, etc. Our encoder further encodes the texture pattern in each segment compactly using a texture code and a parametric sine wave. We introduce the details of the image grouping and texture modeling module in the following.

Image Grouping. The detailed structure of the grouping module is illustrated in the blue box in Fig. 2. First, we introduce a CNN backbone E to produce per-pixel feature vectors F, which is then input to the grouping module that contains the GNN. Each vertex $v \in \mathcal{V}$ in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a superpixel, whose feature is computed as the averaged F of all pixels within the superpixel. The weight of each edge $e \in \mathcal{E}$ connecting two superpixels in the graph is computed as the cosine similarity of two adjacent superpixel features. Given such a graph, we first employ two graph convolution layers [32] to propagate information from each superpixel to its neighboring superpixels. Then a per-pixel feature map $F' \in \mathcal{R}^{C_F \times H \times W}$ is recovered by setting the feature of each pixel as the feature of the superpixel it belongs to. Finally, a convolution layer followed by a SoftMax layer is applied to group all pixels into K segments. The predicted grouping mask $M \in \mathcal{R}^{K \times H \times W}$ decomposes the input image into K different groups, each includes a unique texture pattern and is captured by a vector texture code and a parametric sine wave discussed next.

Texture Code Computation. Given the per-pixel feature $F' \in \mathcal{R}^{C_F \times H \times W}$ and the grouping mask $M \in \mathcal{R}^{K \times H \times W}$ from the grouping module discussed above, we first compute the features $F_K \in \mathcal{R}^{K \times C_F}$ for all groups as $F_K = flatten(M) \times flatten(F')^T$, where $flatten(M) \in \mathcal{R}^{K \times (HW)}$ and $flatten(F') \in \mathcal{R}^{C_F \times (HW)}$ are the flattened version of M and F' respectively. The kth row of F_K thus stores the feature of group k. The texture code t^k for each group is computed by two linear layers, taking ${F_K}^k$ as input.

Parametric Sine Wave Computation. Besides the texture code, we introduce a parametric sine wave to improve texture modeling. Our key insight is that the periodic sine wave well fits for the repetitive nature of textures, especially for regular patterns that resemble strong periodicity. Inspired by PSGAN [5], we formulate the parametric sine wave with learnable frequency P^k as:

$$S^k = \sin(P^k C + \Phi^k), k = 1, \ldots, K \tag{1}$$

where d is a hyperparameter indicating the channel number of the sine wave, $C \in \mathcal{R}^{2\times(HW)}$ is a standard 2D coordinate grid, and Φ^k is a shift randomly sampled from $[0, 2\pi)$ to mimic random positional shift of texture segments. Intuitively, the frequency parameter P^k determines the repeating period of a texture and thus is predicted by applying two linear layers on the statistic texture code t^k.

3.3 Decoding Texture Codes and Sine Waves into RGB Images

Given the texture code and sine wave of each segment, we adopt a decoder to reconstruct the input image by taking the sine wave composition of different texture segments as input and modulating the feature at each layer by the texture code). Specifically, given the feature map F_l from layer l of the decoder, a pixel $F_l(i, j)$ assigned to group k is modulated by its corresponding texture code t^k as:

$$\gamma^k \frac{F_l(i, j) - \mu^k}{\sigma^k} + \beta^k \tag{2}$$

where the modulation parameters γ^k and β^k are projected from the texture code t^k by linear layers and μ^k, σ^k are the mean and standard deviation in each channel of F_l.

3.4 Texture Synthesis

Next, we task our decoder for texture synthesis to enhance texture feature learning. Specifically, we randomly choose a texture segment $q \in \{1, \ldots, K\}$ from the K groups and synthesize a corresponding texture image resembling the chosen texture segment. To this end, we feed the corresponding sine wave S^q into the decoder and modulate all pixels in the feature maps at each decoder layer using the texture code t^q. As a result, the decoder synthesizes a texture image that presents the same pattern resembling the chosen texture group q. More importantly, thanks to the inherent periodicity of the sine wave, our model can synthesize an arbitrary-sized texture image by simply extending the input sine wave during inference.

However, the segments produced by the grouping module may be noisy (e.g., a texture segment that includes more than one texture pattern), especially at the early training stage. To clean up the texture segments and provide better supervision for texture features learning, we split each group into spatially connected regions [26]. The texture segment used as the supervision for synthesis is then chosen from these spatially connected regions. This is based on the intuition that spatially connected regions are more likely to be clean texture segments with a single texture pattern, introducing less noise for texture features learning.

3.5 Objectives

Reconstruction Objectives. For each texture segment, we apply the L1, the VGG-based perceptual [29,66] loss and the focal frequency loss [28] between the input and the reconstructed segment.

Texture Synthesis Objectives. Since there is no direct supervision for the regions outside the given texture segment, we apply the Gram matrix matching objective [17,18,42] and a patch discriminator [47]. The former ensures the synthesized texture match the statistics of the texture segment while the latter encourages realistic texture synthesis.

As discussed in Section 1, texture segments in natural images often have irregular shapes. Thus we utilize the grouping mask produced by the grouping module to mask out irrelevant regions in both objectives. Specifically, when computing the gram matrix of a texture segment, we only take the pixels within the texture segment to calculate the correlation matrix. A detailed mathematical formulation can be found in the supplementary material.

For the patch discriminator, we randomly crop ten patches from the corresponding texture segment in the input image as real data and ten patches from the synthesized texture image as fake data. To ensure patches cropped from the synthesized texture images are comparable with those cropped from input images, we fill irrelevant areas in the former with corresponding pixels from the latter. The objective of the patch discriminator is:

$$L_{PGAN} = \mathbb{E}_{I,\hat{I}_t}[-\log(D_p(\text{crops}(\hat{I}_t) \cdot m + \text{crops}(I) \cdot (1-m), \text{crops}(I)))] \tag{3}$$

where D_p is the patch discriminator [47], crops$(\cdot)$ is the operation that randomly crops patches from an image and m is the mask patch from M_t corresponding to the cropped patches from the input image (i.e., crops(I)).

3.6 Generalization to Unseen Textures

To generalize our model to an unseen image, we first train our model on a small image dataset and then fine-tune it on a single image. Importantly, we make the following differences during test-time tuning compared to the main network training: a) we do not include any form of data augmentation, i.e., the training image is cropped once and fixed. b) we sample one texture code and synthesize two images from it. One image using a sine wave with zero offset and the other using a random offset. c) we apply reconstruction loss on the former synthesis and gram matrix matching and patch GAN to the later. d) we use Meanshift [9] to allow each image to have different group numbers. The rationale of these differences is that our model has to reconstruct the given texture segment well before extending it. By fixing the offset and the training image, we allow the model to achieve such a goal.

4 Experimental Results

4.1 Experimental Settings

Implementation Details Our encoder is composed of multiple Residual blocks [21] and two graph convolution layers. The decoder is the same as the generator in [47]. We cluster pixels in each image into $K = 10$ groups for the main network training if not otherwise specified. Empirically we find that progressive training stabilizes the process and yields the best performance. Specifically, we train the model without the grouping module for 100 epochs and fine-tune with the grouping module for another 100 epochs. The texture synthesis objective is then added, and the entire pipeline is jointly trained until convergence. For each unseen image, we fine-tune the model for 5000 iterations with the same objectives introduced in Section 3.5.We use the Adam optimizer [31] with an initial learning rate of 5×10^{-5}. More details of the network architecture and optimization hyperparameters can be found in the supplementary material. The proposed model is implemented in PyTorch [48].

Datasets We use the BSDS500 dataset [46,3] as our training dataset. Compared to other datasets such as COCO [41] or the ImageNet [11], the BSDS500 dataset provides larger and richer texture regions that are more fit for our texture synthesis task. For texture synthesis evaluation, we compose a dataset of 104 texture-abundant images not included in the training dataset from the ImageNet dataset [11]. Although our main focus is on texture synthesis, we validate the learned grouping masks by evaluating the image grouping module on the testing dataset from the BSDS500 dataset [46,3], which includes 200 testing images with grouping annotations. We note that these grouping annotations maybe semantic and include multiple textures in a single segment.

Baselines We quantitatively evaluate our model on the task of texture synthesis and compare with the WCT [37], DeepTexture [17], PSGAN [5], and Image Quilting [12]. None of the baseline models can generalize to unseen texture images, nor can they synthesize texture directly from natural images or even irregular-shaped texture segments. Thus, to ensure best performance of the baselines, we provide texture patches that are as clean as possible for the baseline model training and train a separate baseline model for each testing image. As shown in Fig. 3 (e), for each target texture segment, we crop a 100×100 texture patch at the center of the texture segment to serve as the training exemplar for the baseline methods. The cropped texture segment maximally suppresses irrelevant textures. We emphasize that the proposed method does not require clean texture patches for training, thus avoids such a clean process.

4.2 Texture Synthesis, Swapping and Editing

Texture Synthesis We present texture synthesis results as well as comparisons with baseline methods in Fig. 3. Compared to the WCT and DeepTexture method, our model is able to capture both regular (e.g., the grid in the first row of Fig. 3) and irregular texture patterns thanks to the parametric sine wave. The texture images produced by our method are also more realistic and uniform compared to the DeepTexture method, which naively repeats the texture region. Furthermore, by introducing the grouping module, our method is able to synthesize texture images from a texture segment. Instead, both the Image Quilting [13] and PSGAN [5] method require clean texture patches as exemplar and cannot automatically ignore other irrelevant textures in the exemplar. Arbitrary-sized texture synthesis by our method can be found in the supplementary material.

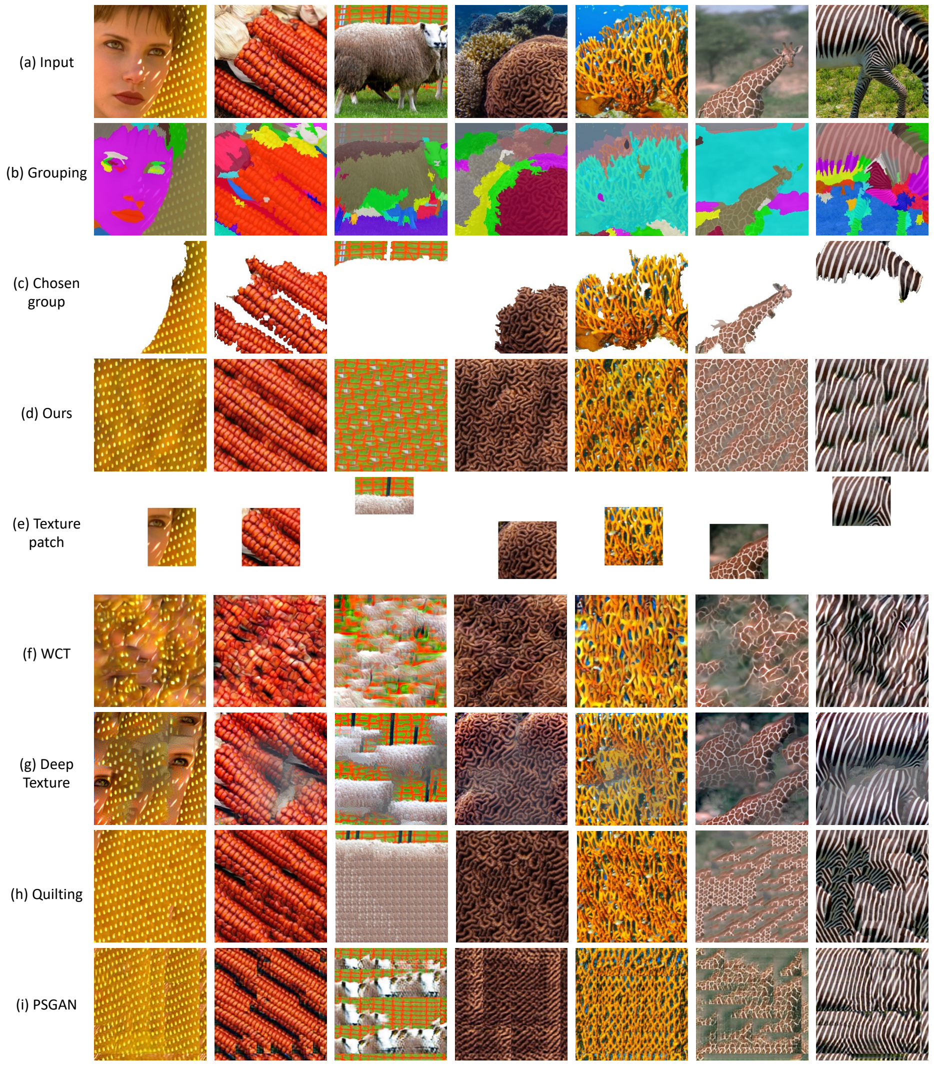

Fig. 3: **Texture synthesis.** (a) Input image. (b) Grouping mask predicted by our encoder. (c) A randomly chosen texture segment. (d) Texture synthesis by our method from the chosen texture segment in (c). (e) A patch cropped at the center of the chosen texture segment that serves as texture exemplar for the baselines models. (f)∼(i) Texture synthesis results by WCT [37], DeepTexture [17], Image quilting [12] and PSGAN [5].

Quantitative Evaluation on Texture Synthesis We use the Fréchet Inception Distance (FID) [24,55,47] and the Perceptual Similarity Distance (LPIPS) [66] to quantitatively evaluate how realistic the generated texture images are.

Since we only have access to texture segments of irregular shapes, we compare patches cropped from the texture segments in input images and synthesized texture images similarly as [42]. Specifically, we synthesize three different texture

Metric	c-FID	c-FID (mask)	c-LPIPS	c-LPIPS (mask)
WCT [37]	149.47	82.65	0.3576	0.3547
DeepTexture [58]	151.91	107.34	0.3930	0.3825
Quilting [13]	76.80	63.64	0.3519	0.3568
PSGAN [5]	128.80	83.32	0.3810	0.3697
Ours	72.36	60.91	0.2838	0.3193

Table 1: **Quantitative evaluation of texture synthesis results.** All metrics are the lower the better.

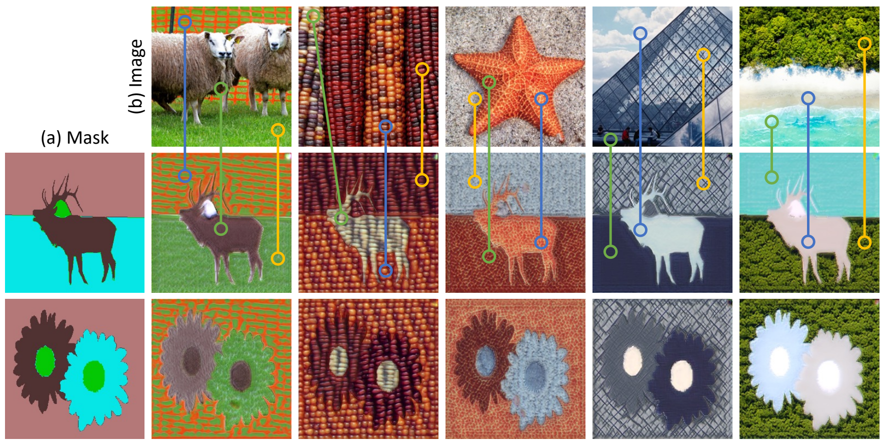

Fig. 4: **Texture editing.** Given the mask drawn by a user in (a) and a reference image in (b), we fill each region in the mask with textures from the reference image. The lines show the corresponding texture specified by the user that fills each region in the mask.

images from the texture segments of each testing image. We then randomly crop ten 64×64 patches from the corresponding texture segment in the input image defined by the grouping segment mask and ten corresponding patches from the synthesized texture image. The patch-based FID score (denoted as c-FID) as well as LPIPS score (denoted as c-LPIPS) are computed between patches cropped from the input image and synthesized texture images. Note that we use a dimension of 2048 for c-FID as opposite in [42] since we have sufficient number of cropped patches for evaluation. Furthermore, since the cropped patches in the input images may include irrelevant textures, we also mask the irrelevant textures in both the input image patches and the synthesized texture patches using our grouping masks. The FID and LPIPS metric of this setting are shown in the third and fifth column of Table 1. As shown in Table 1, the proposed method achieves lower c-FID and c-LPIPS score, indicating that it can synthesize more realistic textures from natural images.

Texture Editing We further apply the learned model to texture editing and show results in Fig. 4. In this task, a user draws a mask (Fig. 4(a)) and chooses the texture segments in the input image (Fig. 4(b)) to provide texture codes. A texture feature map is composed by filling each region in the mask with the texture code of the chosen segment. The decoder combines the composed texture

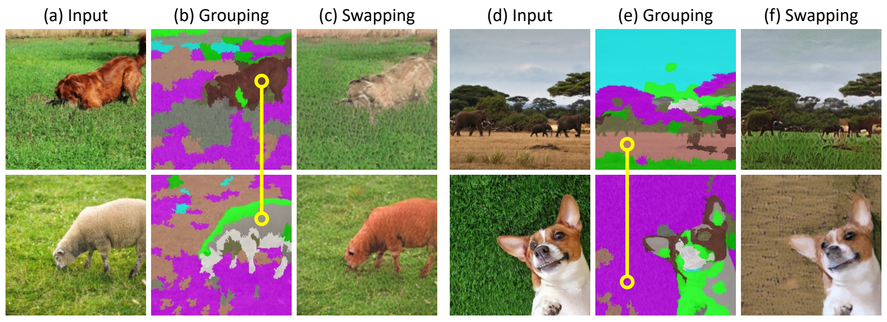

Fig. 5: **Texture swapping.** Given a pair of images, our model predicts their groupings in (b)(e). We swap the texture codes of segments in each image chosen by a user. The yellow line indicates the swapped regions in the two images.

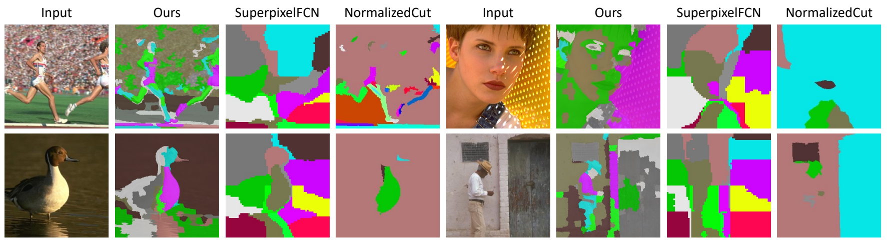

Fig. 6: **Image Grouping.** We show image grouping results by our method, the supervsied SuperpixelFCN [64] and the unsupervised Normalized Cuts [56] method visually.

feature map with random noise to produce the final editing image. As shown in Fig. 4, each texture can be successfully transferred and realistically synthesized onto the corresponding region in the mask, demonstrating the flexibility of the learned texture codes in controlled texture editing tasks.

Texture Swapping We also show the application of texture swapping in Fig. 5. Given a pair of images, we first predict their grouping masks as shown in Fig. 5(b)(e). A user defines which segments to be swapped out in each image (yellow lines in Fig. 5(b)(e)), the swapping is then carried out by swapping the texture codes of corresponding regions. The final swapped image (Fig. 5(c)(f)) is obtained by feeding the spatial code together with the swapped texture codes to the decoder.

4.3 Image Grouping Evaluation

Although our work focuses on texture synthesis, we validate the quality of the predicted segment boundary by comparing it with ground truth grouping annotations using the Boundary Recall (BR) and Boundary Precision (BP) metric. Additionally, we measure the upper bound performance of any segmentation model on top of our predicted segments using the Achievable Segmentation Accuracy (ASA) metric. The definition and detailed explanation of all these three metrics can be found in [26]. We compare the proposed algorithm against a supervised superpixel method [64] trained with ground truth grouping annotations

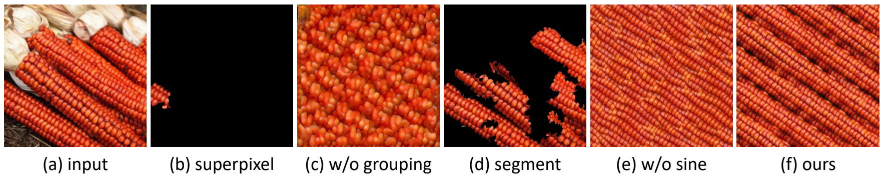

Fig. 7: **Ablation study.** (a) Input image. (b) A superpixel predicted by the SLIC method [1]. (c) Texture synthesis result by the baseline model using the superpixel in (b). (d) A texture segment predicted by our grouping model. (e) Texture synthesis by a baseline model learned with noise instead of sine waves. (f) Texture synthesis of our full model.

and the unsupervised Normalized Cuts [56] method. As discussed in Section 4, our model predicts $K = 10$ segments for each image. For fair comparisons, we adjust the hyper-parameters (i.e., the input image size in [64] and the merging threshold in [56]) such that these baselines produce a similar amount of groups (i.e., around ten groups). Table 2 shows the performance comparison and averaged group numbers by different methods. Fig. 6 demonstrates the predicted grouping by each method visually. Both the proposed method and the Normalized Cuts [56] approach do not require ground truth grouping annotations. Yet, the proposed method achieves better boundary recall and serves better as a segmentation pre-processing step as it has a higher ASA score. On the other hand, the Normalized Cuts method aggressively merges superpixels when setting group number small (i.e., setting lower merging threshold), as shown in Fig. 6(d). This is also why the Normalized Cuts method has higher BP since it misses many boundaries in the image. Although self-supervisedly learned, our method achieves a comparable BP score with fewer groups than the supervised superpixel method [64]. The superpixel method achieves a higher ASA score since it has access to the ground truth grouping annotations during model training.

Metric	BR ↑	BP ↑	ASA ↑	Groups
SuperpixelFCN [64]	0.35	0.25	0.83	16
NormalizedCut [56]	0.3	0.38	0.58	14
Ours	0.67	0.27	0.68	10

Table 2: **Image grouping evaluation on the BSDS500 [46,3] dataset**

4.4 Ablation Studies

Image Grouping. To validate the contribution of image grouping in texture features learning, we introduce a baseline model that takes the SLIC superpixels [1] as texture segments instead of learning image grouping together with texture modeling. We show qualitative comparison with the full model in Fig. 7 (c) and (f). The baseline model is limited by the small receptive field of superpixels and cannot fully capture the texture in each segment (e.g., the grid and corn kernel in Fig. 7). While our method is able to synthesize realistic textures that resemble the chosen texture segment.

Sine Wave. The inherent periodicity of the sine wave helps our model to synthesize regular pattern such as the grid shown in the first row of Fig. 7. On the contrary, if we feed a noise instead of a sine wave into our decoder, the baseline model fails to capture stationary pattern as shown in Fig. 7 (e).

5 Conclusion

In this work, we propose a framework to scrape textures from natural images by decomposing an image into a set of texture segments and represent each texture segment with a vector texture code. By reconstructing the input image from these texture codes, our model learns to group pixels of the same texture without any supervision. Through encoding each texture segment by the texture code, along with a parametric sine wave, we can learn texture representations directly from cluttered images without requiring any well-cropped texture patches. We demonstrate that these texture representations can be used for texture synthesis and editing, and the emergent image segments are often consistent with object boundaries. While our paper demonstrates some promising results, plenty of work remains in both texture modeling as well as image segmentation. The lesson of our work is that these two tasks would likely benefit from being considered jointly.

References

1. Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Süsstrunk, S.: Slic superpixels compared to state-of-the-art superpixel methods. TPAMI (2012) 4, 5, 6, 13
2. Alanov, A., Kochurov, M., Volkhonskiy, D., Yashkov, D., Burnaev, E., Vetrov, D.: User-controllable multi-texture synthesis with generative adversarial networks. arXiv preprint arXiv:1904.04751 (2019) 4
3. Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. TPAMI (2011) 6, 9, 13
4. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: PatchMatch: A randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics (Proc. SIGGRAPH) **28**(3) (Aug 2009) 4
5. Bergmann, U., Jetchev, N., Vollgraf, R.: Learning texture manifolds with the periodic spatial gan. arXiv preprint arXiv:1705.06566 (2017) 3, 4, 7, 9, 10, 11
6. Brodatz, P.: Textures: a photographic album for artists and designers. New York: Dover Pub. (1966) 2, 3
7. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. TPAMI (2017) 6
8. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., , Vedaldi, A.: Describing textures in the wild. In: CVPR (2014) 2, 3
9. Comaniciu, D., Meer, P.: Mean shift: A robust approach toward feature space analysis. TPAMI (2002) 8
10. Dai, D., Riemenschneider, H., Van Gool, L.: The synthesizability of texture examples. In: CVPR (2014) 2, 3

11. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009) 9
12. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. SIGGRAPH (2001) 2, 3, 4, 9, 10
13. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: ICCV (1999) 2, 3, 4, 9, 11
14. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. IJCV (2004) 4
15. Fiedler, M., Alpers, A.: Power-slic: Diagram-based superpixel generation. arXiv preprint arXiv:2012.11772 (2020) 4
16. Frühstück, A., Alhashim, I., Wonka, P.: Tilegan: synthesis of large-scale non-homogeneous textures. TOG (2019) 2, 4
17. Gatys, L., Ecker, A.S., Bethge, M.: Texture synthesis using convolutional neural networks. NeurIPS (2015) 2, 3, 4, 8, 9, 10
18. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: CVPR (2016) 8
19. Guo, C.e., Zhu, S.C., Wu, Y.N.: Primal sketch: Integrating structure and texture. Computer Vision and Image Understanding (2007) 5
20. Han, F., Zhu, S.C.: Bottom-up/top-down image parsing with attribute grammar. TPAMI (2008) 5
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016) 8
22. 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 (1995) 4
23. Heitz, E., Vanhoey, K., Chambon, T., Belcour, L.: A sliced wasserstein loss for neural texture synthesis. In: CVPR (2021) 4
24. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. NeurIPS (2017) 10
25. Hoiem, D., Efros, A.A., Hebert, M.: Geometric context from a single image. In: Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1. vol. 1, pp. 654–661. IEEE (2005) 4
26. Jampani, V., Sun, D., Liu, M.Y., Yang, M.H., Kautz, J.: Superpixel sampling networks. In: ECCV (2018) 4, 6, 7, 12
27. Jetchev, N., Bergmann, U., Vollgraf, R.: Texture synthesis with spatial generative adversarial networks. arXiv preprint arXiv:1611.08207 (2016) 4
28. Jiang, L., Dai, B., Wu, W., Loy, C.C.: Focal frequency loss for image reconstruction and synthesis. In: ICCV (2021) 7
29. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: ECCV (2016) 3, 4, 7
30. Kaspar, A., Neubert, B., Lischinski, D., Pauly, M., Kopf, J.: Self tuning texture optimization. In: Computer Graphics Forum (2015) 2, 4
31. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014) 9
32. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016) 6
33. Kwatra, V., Essa, I., Bobick, A., Kwatra, N.: Texture optimization for example-based synthesis. In: SIGGRAPH (2005) 2, 4
34. Li, T.M., Lukáč, M., Gharbi, M., Ragan-Kelley, J.: Differentiable vector graphics rasterization for editing and learning. TOG (2020) 5

35. Li, X., Liu, S., De Mello, S., Kim, K., Wang, X., Yang, M.H., Kautz, J.: Online adaptation for consistent mesh reconstruction in the wild. NeurIPS (2020) 2
36. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Diversified texture synthesis with feed-forward networks. In: CVPR (2017) 4
37. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Universal style transfer via feature transforms. In: NeurIPS (2017) 2, 9, 10, 11
38. Li, Y., Hao, M., Di, Z., Gundavarapu, N.B., Wang, X.: Test-time personalization with a transformer for human pose estimation. NeurIPS (2021) 2
39. Li, Z., Bao, W., Zheng, J., Xu, C.: Deep grouping model for unified perceptual parsing. In: CVPR (2020) 4
40. Lin, Q., Zhong, W., Lu, J.: Deep superpixel cut for unsupervised image segmentation. In: ICPR. pp. 8870–8876 (2021) 4
41. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: ECCV (2014) 6, 9
42. Liu, G., Taori, R., Wang, T.C., Yu, Z., Liu, S., Reda, F.A., Sapra, K., Tao, A., Catanzaro, B.: Transposer: Universal texture synthesis using feature maps as transposed convolution filter. arXiv preprint arXiv:2007.07243 (2020) 2, 3, 4, 8, 10, 11
43. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR (2015) 6
44. Malisiewicz, T., Efros, A.A.: Improving spatial support for objects via multiple segmentations. In: British Machine Vision Conference (BMVC) (September 2007) 4
45. Mardani, M., Liu, G., Dundar, A., Liu, S., Tao, A., Catanzaro, B.: Neural ffts for universal texture image synthesis. NeurIPS (2020) 2, 4
46. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: ICCV (2001) 9, 13
47. Park, T., Zhu, J.Y., Wang, O., Lu, J., Shechtman, E., Efros, A.A., Zhang, R.: Swapping autoencoder for deep image manipulation. In: NeurIPS (2020) 5, 8, 10
48. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. NeurIPS (2019) 9
49. Pont-Tuset, J., Arbelaez, P., Barron, J.T., Marques, F., Malik, J.: Multiscale combinatorial grouping for image segmentation and object proposal generation. TPAMI (2016) 4
50. Portilla, J., Simoncelli, E.P.: A parametric texture model based on joint statistics of complex wavelet coefficients. IJCV (2000) 4
51. Ren, X., Malik, J.: Learning a classification model for segmentation. In: ICCV (2003) 4, 6
52. Rosenberger, A., Cohen-Or, D., Lischinski, D.: Layered shape synthesis: automatic generation of control maps for non-stationary textures. ACM Transactions on Graphics (TOG) (2009) 3
53. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks (2008) 4
54. Sendik, O., Cohen-Or, D.: Deep correlations for texture synthesis. TOG (2017) 4
55. Shaham, T.R., Dekel, T., Michaeli, T.: Singan: Learning a generative model from a single natural image. In: ICCV (2019) 4, 10
56. Shi, J., Malik, J.: Normalized cuts and image segmentation. TPAMI (2000) 4, 12, 13
57. Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A., Hardt, M.: Test-time training with self-supervision for generalization under distribution shifts. In: ICML (2020) 2

58. Ulyanov, D., Lebedev, V., Vedaldi, A., Lempitsky, V.S.: Texture networks: Feed-forward synthesis of textures and stylized images. In: ICML (2016) 2, 3, 4, 11
59. Wang, Y., Wei, Y., Qian, X., Zhu, L., Yang, Y.: Ainet: Association implantation for superpixel segmentation. arXiv preprint arXiv:2101.10696 (2021) 4
60. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: SIGGRAPH (2000) 3, 4
61. Wertheimer, M.: Laws of organization in perceptual forms. Psycologische Forschung **4** (1923) 4
62. Wu, Q., Yu, Y.: Feature matching and deformation for texture synthesis. TOG (2004) 4
63. Wu, Z., Lin, D., Tang, X.: Deep markov random field for image modeling. In: ECCV (2016) 3
64. Yang, F., Sun, Q., Jin, H., Zhou, Z.: Superpixel segmentation with fully convolutional networks. CVPR (2020) 4, 6, 12, 13
65. Yu, N., Barnes, C., Shechtman, E., Amirghodsi, S., Lukac, M.: Texture mixer: A network for controllable synthesis and interpolation of texture. In: CVPR (2019) 2, 3, 4
66. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR (2018) 7, 10
67. Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Scene parsing through ade20k dataset. In: CVPR (2017) 6
68. Zhou, Y., Zhu, Z., Bai, X., Lischinski, D., Cohen-Or, D., Huang, H.: Non-stationary texture synthesis by adversarial expansion. arXiv preprint arXiv:1805.04487 (2018) 4
69. Zou, Z., Shi, T., Qiu, S., Yuan, Y., Shi, Z.: Stylized neural painting. In: CVPR (2021) 5