SIP (2021), vol. 10, e17, page 1 of 14 © The Author(s), 2021. Published by Cambridge University Press in association with Asia Pacific Signal and Information Processing Association. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution, and reproduction in any medium, provided the original work is properly cited.
doi:10.1017/ATSIP.2021.15

ORIGINAL PAPER

TGHop: an explainable, efficient, and lightweight method for texture generation

XUEJING LEI, GANNING ZHAO, KAITAI ZHANG AND C.-C. JAY KUO

An explainable, efficient, and lightweight method for texture generation, called TGHop (an acronym of Texture Generation PixelHop), is proposed in this work. Although synthesis of visually pleasant texture can be achieved by deep neural networks, the associated models are large in size, difficult to explain in theory, and computationally expensive in training. In contrast, TGHop is small in its model size, mathematically transparent, efficient in training and inference, and able to generate high-quality texture. Given an exemplary texture, TGHop first crops many sample patches out of it to form a collection of sample patches called the source. Then, it analyzes pixel statistics of samples from the source and obtains a sequence of fine-to-coarse subspaces for these patches by using the PixelHop++ framework. To generate texture patches with TGHop, we begin with the coarsest subspace, which is called the core, and attempt to generate samples in each subspace by following the distribution of real samples. Finally, texture patches are stitched to form texture images of a large size. It is demonstrated by experimental results that TGHop can generate texture images of superior quality with a small model size and at a fast speed.

Keywords: Texture generation, Texture synthesis, Generative model, Successive subspace modeling

Received 1 July 2021; Revised 13 September 2021

I. INTRODUCTION

Automatic generation of visually pleasant texture that resembles exemplary texture has been studied for several decades since it is of theoretical interest in texture analysis and modeling. Research in texture generation benefits texture analysis and modeling research [1–7] by providing a perspective to understand the regularity and randomness of textures. Texture generation finds broad applications in computer graphics and computer vision, including rendering textures for 3D objects [5, 8], image or video super-resolution [9], etc.

Early works of texture generation generates textures in pixel space. Based on exemplary input, texture can be generated pixel-by-pixel [10–12] or patch-by-patch [13–16], starting from a small unit and gradually growing to a larger image. These methods, however, suffer from slow generation time [11, 14] or limited diversity of generated textures [13, 15, 17]. Later works transform texture images to a feature space with kernels and exploit the statistical correlation of features for texture generation. Commonly used kernels include the Gabor filters [18] and the steerable pyramid filter banks [19]. This idea is still being actively studied with the resurgence of neural networks. Various deep learning (DL) models, including Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs), yieldvisually pleasing results in texture generation. Compared to traditional methods, DL-based methods [20–26] learn weights and biases through end-to-end optimization. Nevertheless, these models are usually large in model size, difficult to explain in theory, and computationally expensive in training. It is desired to develop a new generation method that is small in model size, mathematically transparent, efficient in training and inference, and able to offer high-quality textures at the same time. Along this line, we propose the TGHop (Texture Generation PixelHop) method in this work.

TGHop consists of four steps. First, given an exemplary texture, TGHop crops numerous sample patches out of it to form a collection of sample patches called the source. Second, it analyzes pixel statistics of samples from the source and obtains a sequence of fine-to-coarse subspaces for these patches by using the PixelHop++ framework [27]. Third, to generate realistic texture patches, it begins with generating samples in the coarsest subspace, which is called the core, by matching the distribution of real and generated samples, and attempts to generate spatial pixels given spectral coefficients from coarse to fine subspaces. Last, texture patches are stitched to form texture images of a larger size. Extensive experiments are conducted to show that TGHop can generate texture images of superior quality with a small model size, at a fast speed, and in an explainable way.

It is worthwhile to point out that this work is an extended version of our previous work in [28], where a method called

Ming Hsieh Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089, USA

Corresponding author:
Xuejing Lei
Email: xuejing@usc.edu

NITES was presented. Two works share the same core idea, but this work provides a more systematic study on texture synthesis task. In particular, a spatial Principal Component Analysis (PCA) transform is included in TGHop. This addition improves the quality of generated textures and reduces the model size of TGHop as compared with NITES. Furthermore, more experimental results are given to support our claim on efficiency (i.e., a faster computational speed) and lightweight (i.e., a smaller model size).

The rest of the paper is organized as follows. Related work is reviewed in Section II. A high-level idea of successive subspace analysis and generation is described in Section III. The TGHop method is detailed in Section IV. Experimental results are shown in Section V. Finally, concluding remarks and future research directions are given in Section VI.

II. RELATED WORK

A) Early work on texture generation

Texture generation (or synthesis) has been a long-standing problem of great interest. The methods for it can be categorized into two types. The first type generates one pixel or one patch at a time and grows synthesized texture from small to large regions. Pixel-based method synthesizes a center pixel conditioned on its neighboring pixels. Efros and Leung [11] proposed to synthesize a pixel by randomly choosing from the pixels that have similar neighborhood as the query pixel. Patch-based methods [13–16] usually achieve higher quality than pixel-based methods [10–12]. They suffer from two problems. First, searching the whole space to find a matched patch is slow [11, 14]. Second, the methods [13, 15, 17] that stitching small patches to form a larger image sustain limited diversity of generated patches, though they are capable of producing high-quality textures at a fast speed. A certain pattern may repeat several times in these generated textures without sufficient variations due to lack of understanding the perceptual properties of texture images. The second type addresses this problem by analyzing textures in feature spaces rather than pixel space. A texture image is first transformed to a feature space with kernels. Then, statistics in the feature space, such as histograms [18] and handcrafted summary [19], is analyzed and exploited for texture generation. For the transform, some pre-defined filters such as Gabor filters [18] or steerable pyramid filter banks [19] were adopted in early days. The design of these filters, however, heavily relies on human expertise and lack adaptivity. With the recent advances of deep neural networks, filters from a pre-trained networks such as VGG provide a powerful transformation for analyzing texture images and their statistics [20, 24].

B) DL-based texture generation

DL-based methods often employ a texture loss function that computes the statistics of the features. Fixing the weights of a pre-trained network, the method in [20] applies the Gram matrix as the statistical measurement and iteratively optimizes an initial white-noise input image through backpropagation. The method in [24] computes feature covariances of white-noise image and texture image, and matches them through whitening and coloring. The method in [29] trained a generator network using a loss based on the same statistics as [20] inside a pre-trained descriptor network. Its training is sensitive to hyper-parameters choice varying with different textures. These three methods utilized a VGG-19 network pre-trained on the Imagenet to extract features. The method in [25] abandons the deep VGG network but adopts only one convolutional layer with random filter weights. Although these methods can generate visually pleasant textures, the iterative optimization process (i.e. backpropagation) is computationally expensive. There is a lot of follow-ups to [20] using the Gram matrix as the statistics measurement such as incorporating other optimization terms [21, 22] and improving inference speed [23, 26]. However, there is a price to pay. The former aggravates the computational burden while the latter increases the training time. Another problem of these methods lies in the difficulty of explaining the usage of a pre-trained network. The methods in [20, 24] develop upon a VGG-19 network pre-trained on the Imagenet dataset. The Imagenet dataset is designed for understanding the semantic meaning of a large number of natural images. Textures, however, mainly contain low-level image characteristics. Although shallow layers (such as conv_1) of VGG are known to capture low-level characteristics of images, generating texture only with shallow layers does not give a good results in [20]. It is hard to justify whether the VGG feature contains redundancy for textures or ignores some texture-specific information. Lack of explainability also raises the challenge of inspecting the methods when unexpected generation results occurred. There are some advances in explainable DL research. Visual Analytics systems are utilized for in-depth model understanding and diagnoses [30]. Algorithm unrolling technique was developed to help connect neural networks and iterative algorithms [31]. Although they are beneficial tools to alleviate the issue, more work needs to be done to thoroughly understand the mechanism of networks. Thus, these drawbacks motivate us to design a method that is efficient, lightweight, and dedicated to texture.

C) Successive subspace learning

To reduce the computational burden in training and inference of DL-based methods, we adopt spatial-spectral representations for texture images based on the successive subspace learning (SSL) framework [32–34]. To implement SSL, PixelHop [35] and PixelHop++ [27] architectures have been developed. PixelHop consists of multi-stage Saab transforms in cascade. PixelHop++ is an improved version of PixelHop by replacing the Saab transform with the channel-wise (c/w) Saab transform, exploiting weak correlations among spectral channels. Both the Saab transform and the c/w Saab transform are data-driven transforms, which are variants of the PCA transform. PixelHop++

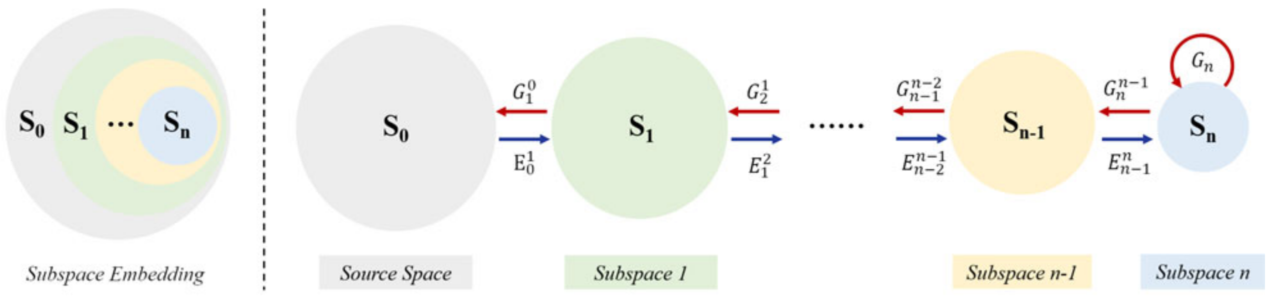

Fig. 1. Illustration of successive subspace analysis and generation, where a sequence of subspace $S_1, \dots, S_n$ is constructed from source space, S_0, through a successive process indicated by blue arrows while red arrows indicate the successive subspace generation process.

offers powerful hierarchical representations and plays a key role of dimension reduction in TGHop. SSL-based solutions have been proposed to tackle quite a few problems, including [36–45]. In this work, we present an SSL-based texture image generation method. The main idea of the method is demonstrated in the next section.

III. SUCCESSIVE SUBSPACE ANALYSIS AND GENERATION

In this section, we explain the main idea behind the TGHop method, successive subspace analysis, and generation, as illustrated in Fig. 1. Consider an input signal space denoted by $\tilde{S}_0$, and a sequence of subspaces denoted by $\tilde{S}_1, \dots, \tilde{S}_n$. Their dimensions are denoted by $\tilde{D}_0, \tilde{D}_1, \dots, \tilde{D}_n$. They are related with each other by the constraint that any element in $\tilde{S}_{i+1}$ is formed by an affine combination of elements in $\tilde{S}_i$, where $i = 0, \dots, n-1$.

An affine transform can be converted to a linear transform by augmenting vector $\tilde{\mathbf{a}}$ in $\tilde{S}_i$ via $\mathbf{a} = (\tilde{\mathbf{a}}^T, 1)^T$. We use S_i to denote the augmented space of $\tilde{S}_i$ and $D_i = \tilde{D}_i + 1$. Then, we have the following relationship

$$S_n \subset S_{n-1} \subset \cdots \subset S_1 \subset S_0, \tag{1}$$

and

$$D_n < D_{n-1} < \cdots < D_1 < D_0. \tag{2}$$

We use texture analysis and generation as an example to explain this pipeline. To generate homogeneous texture, we collect a number of texture patches cropped out of exemplary texture as the input set. Suppose that each texture patch has three RGB color channels, and a spatial resolution $P \times P$. The input set then has a dimension of $3P^2$ and its augmented space S_0 has a dimension of $D_0 = 3P^2 + 1$. If $P = 32$, we have $D_0 = 3073$ which is too high to find an effective generation model directly.

To address this challenge, we build a sequence of subspaces $S_0, S_1, \dots, S_n$ with decreasing dimensions. We call S_0 and S_n the "source" space and the "core" subspace, respectively. We need to find an effective subspace S_{i+1} from S_i, and such an analysis model is denoted by E_i^{i+1}. Proper subspace analysis is important since it determines how to decompose an input space into the direct sum of two subspaces in the forward analysis path. Although we choose one of the two for further processing and discard the other one, we need to record the relationship of the two decomposed subspaces so that they are well-separated in the reverse generation path. This forward process is called fine-to-coarse analysis.

In the reverse path, we begin with the generation of samples in S_n by studying its own statistics. This is accomplished by generation model G_n. The process is called core sample generation. Then, conditioned on a generated sample in S_{i+1}, we generate a new sample in S_i through a generation model denoted by G_{i+1}^i. This process is called coarse-to-fine generation. In Fig. 1, we use blue and red arrows to indicate analysis and generation, respectively. This idea can be implemented as a non-parametric method since we can choose subspaces $S_1, \dots, S_n$, flexibly in a feedforward manner. One specific design is elaborated in the next section.

IV. TGHOP METHOD

The TGHop method is proposed in this section. An overview of the TGHop method is given in Section A). Next, the forward fine-to-coarse analysis based on the two-stage c/w Saab transforms is discussed in Section B). Afterwards, sample generation in the core is elaborated in Section C). Finally, the reverse coarse-to-fine pipeline is detailed in Section D).

A) System overview

An overview of the TGHop method is given in Fig. 2. The exemplary color texture image has a spatial resolution of 256×256 and three RGB channels. We would like to generate multiple texture images that are visually similar to the exemplary one. By randomly cropping patches of size 32×32 out of the source image, we obtain a collection of texture patches serving as the input to TGHop. The dimension of these patches is $32 \times 32 \times 3 = 3072$. Their augmented vectors form source space S_0. The TGHop system is designed to generate texture patches of the same size that are visually similar to samples in S_0. This is feasible if we can capture both global and local patterns of these samples. There are two paths in Fig. 2. The blue arrows go from left to right, denoting the fine-to-coarse analysis process. The red arrows go from right to left, denoting the coarse-to-fine generation process. We can generate as many texture patches as desired using this procedure. In order to generate a texture

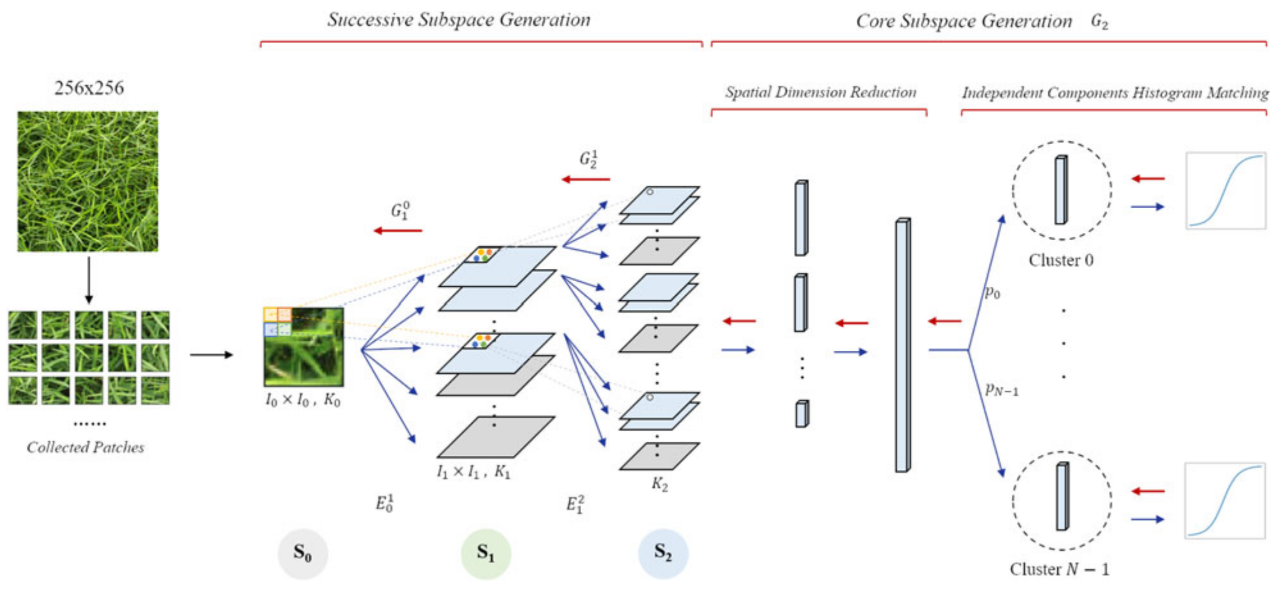

Fig. 2. An overview of the proposed TGHop method. A number of patches are collected from the exemplary texture image, forming source space S_0. Subspaces S_1 and S_2 are constructed through analysis models E_0^1 and E_1^2. Input filter window sizes to Hop-1 and Hop-2 are denoted as I_0 and I_1. Selected channel numbers of Hop-1 and Hop-2 are denoted as K_1 and K_2. A block of size $I_i \times I_i$ of K_i channels in space/subspace S_i is converted to the same spatial location of K_{i+1} channels in subspace S_{i+1}. Red arrows indicate the generation process beginning from core sample generation followed by coarse-to-fine generation. The model for core sample generation is denoted as G_2 and the models for coarse-to-fine generation are denoted as G_2^1 and G_1^0.

image of a larger size, we perform image quilting [13] based on synthesized patches.

B) Fine-to-coarse analysis

The global structure of an image (or an image patch) can be well characterized by spectral analysis, yet it is limited in capturing local detail such as boundaries between regions. Joint spatial-spectral representations offer an ideal solution to the description of both global shape and local detail information. Analysis model E_0^1 finds a proper subspace, S_1, in S_0 while analysis model E_1^2 finds a proper subspace, S_2, in S_1. As shown in Fig. 2, TGHop applies two-stage transforms. They correspond to E_0^1 and E_1^2, respectively. Specifically, we can apply the c/w Saab transform in each stage to conduct the analysis. In the following, we provide a brief review on the Saab transform [34] and the c/w Saab transform [27].

We partition each input patch into non-overlapping blocks, each of which has a spatial resolution of $I_0 \times I_0$ with K_0 channels. We flatten 3D tensors into 1D vectors, and decompose each vector into the sum of one Direct Current (DC) and multiple Alternating Current (AC) spectral components. The DC filter is a all-ones filter weighted by a constant. AC filters are obtained by applying the PCA to DC-removed residual tensor. By setting $I_0 = 2$ and $K_0 = 3$, we have a tensor block of dimension $2 \times 2 \times 3 = 12$. Filter responses of PCA can be positive or negative. There is a sign confusion problem [32, 33] if both of them are allowed to enter the transform in the next stage. To avoid sign confusion, a constant bias term is added to all filter responses to ensure that all responses become positive, leading to the name of the "subspace approximation with adjusted bias (Saab)" transform. The Saab transform is a data-driven transform, which is significantly different from traditional transforms (e.g. Fourier and wavelet transforms) which are data independent. We partition AC channels into two low- and high-frequency bands. The energy of high-frequency channels (shaded by gray color in Fig. 2) is low and they are discarded for dimension reduction without affecting the performance much. The energy of low-frequency channels (shaded by blue color in Fig. 2) is higher. For a tensor of dimension 12, we have one DC and 11 AC components. Typically, we select $K_1 = 6$ to 10 leading AC components and discard the rest. Thus, after E_0^1, one 12-D tensor becomes a K_1-D vector, which is illustrated by dots in subspace S_1. The K_1-D response vectors are fed into the next stage for another transform.

The channel-wise (c/w) Saab transform [27] exploits the weak correlation property between channels so that the Saab transform can be applied to each channel separately (see the middle part of Fig. 2). The c/w Saab transform offers an improved version of the standard Saab transform with a smaller model size.

One typical setting used in our experiments is shown below.

- Dimension of the input patch ($\tilde{D}_0$): $32 \times 32 \times 3 = 3072$;
- Dimension of subspace $\tilde{S}_1$ ($\tilde{D}_1$): $16 \times 16 \times 10 = 2560$ (by keeping 10 channels in Hop-1);
- Dimension of subspace $\tilde{S}_2$ ($\tilde{D}_2$): $8 \times 8 \times 27 = 1728$ (by keeping 27 channels in Hop-2).

Note that the ratio between $\tilde{D}_1$ and $\tilde{D}_0$ is 83.3% while that between $\tilde{D}_2$ and $\tilde{D}_1$ is 67.5%. We are able to reduce the dimension of the source space to that of the core subspace by a factor of 56.3%. In the reverse path indicated by red arrows, we need to develop a multi-stage generation

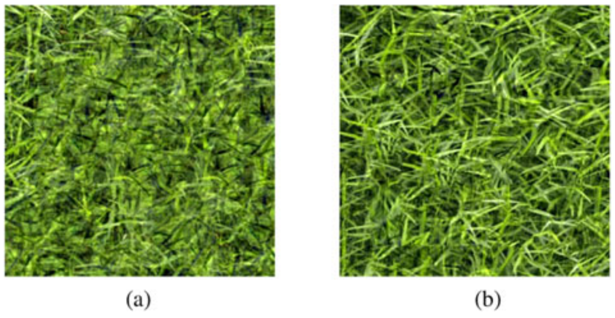

Fig. 3. Generated *grass* texture image with and without spatial dimension reduction (SDR). (a) Without SDR, (b) with SDR.

process. It should also be emphasized that users can flexibly choose channel numbers in Hop-1 and Hop-2. Thus, TGHop is a non-parametric method.

The first-stage Saab transform provides the spectral information on the nearest neighborhood, which is the first hop of the center pixel. By generalizing from one to multiple hops, we can capture the information in the short-, mid-, and long-range neighborhoods. This is analogous to increasingly larger receptive fields in deeper layers of CNNs. However, filter weights in CNNs are learned from end-to-end optimization via backpropagation while weights of the Saab filters in different hops are determined by a sequence of PCAs in a feedforward unsupervised manner.

C) Core sample generation

In the generation path, we begin with sample generation in core S_n which is denoted by G_n. In the current design, $n = 2$. We first characterize the sample statistics in the core, S_2. After two-stage c/w Saab transforms, the sample dimension in S_2 is less than 2000. Each sample contains K_2 channels of spatial dimension 8×8. Since there exist correlations between spatial responses in each channel, PCA is adopted for further Spatial Dimension Reduction (SDR). We discard PCA components whose variances are lower than threshold γ. The same threshold applies to all channels. SDR can help reduce the model size and improve the quality of generated textures. For example, we compare a generated *grass* texture with and without SDR in Fig. 3. The quality with SDR significantly improves.

After SDR, we flatten the PCA responses of each channel and concatenate them into a 1D vector denoted by $\mathbf{z}$. It is a sample in S_2. To simplify the distribution characterization of a high-dimensional random vector, we group training samples into clusters and transform random vectors in each cluster into a set of independent random variables. We adopt the K-Means clustering algorithm to cluster training samples into N clusters, which are denoted by $\{C_i\}$, $i = 0, \ldots, N-1$. Rather than modeling probability $P(\mathbf{z})$ directly, we model condition probability $P(\mathbf{z} \mid \mathbf{z} \in C_i)$ with a fixed cluster index. The probability, $P(\mathbf{z})$, can be written as

$$P(\mathbf{z}) = \sum_{i=0}^{N-1} P(\mathbf{z} \mid \mathbf{z} \in C_i) \cdot P(\mathbf{z} \in C_i), \qquad (3)$$

where $P(\mathbf{z} \in C_i)$ is the percentage of data points in cluster C_i. It is abbreviated as p_i, $i = 0, \ldots, N-1$ (see the right part of Fig. 2).

Typically, a set of independent Gaussian random variables is used for image generation. To do the same, we convert a collection of correlated random vectors into a set of independent Gaussian random variables. To achieve this objective, we transform random vector $\mathbf{z}$ in cluster C_i into a set of independent random variables through independent component analysis (ICA), where non-Gaussianity serves as an indicator of statistical independence. ICA finds applications in noise reduction [46], face recognition [47], and image infusion [48]. Our implementation is detailed below.

(i) Apply PCA to $\mathbf{z}$ in cluster C_i for dimension reduction and data whitening.
(ii) Apply FastICA [49], which is conceptually simple, computationally efficient, and robust to outliers, to the PCA output.
(iii) Compute the cumulative density function (CDF) of each ICA component of random vector $\mathbf{z}$ in each cluster based on its histogram of training samples.
(iv) Match the CDF in Step 3 with the CDF of a Gaussian random variable (see the right part of Fig. 2), where the inverse CDF is obtained by resampling between bins with linear interpolation. To reduce the model size, we quantize N-dimensional CDFs, which have N bins, with vector quantization and store the codebook of quantized CDFs.

We encode p_i in Eq. (3) using the length of a segment in $[0, 1]$. All segments are concatenated in order to build the unit interval. The segment index is the cluster index. These segments are called the interval representation as shown in Fig. 4. To draw a sample from subspace S_2, we use the uniform random number generator to select a random number from interval $[0, 1]$. This random number indicates the cluster index on the interval representation.

To generate a new sample in S_2, we perform the following steps:

(i) Select a random number from the uniform random number generator to determine the cluster index.
(ii) Draw a set of samples independently from the Gaussian distribution.

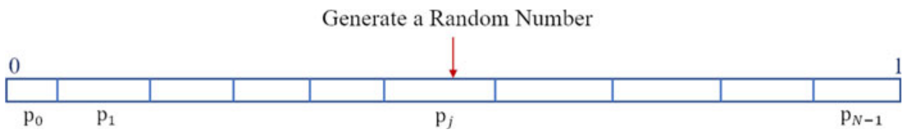

Fig. 4. Illustration of the interval representation, where the length of a segment in the unit interval represents the probability of a cluster, p_i. A random number is generated in the unit interval to indicate the cluster index.

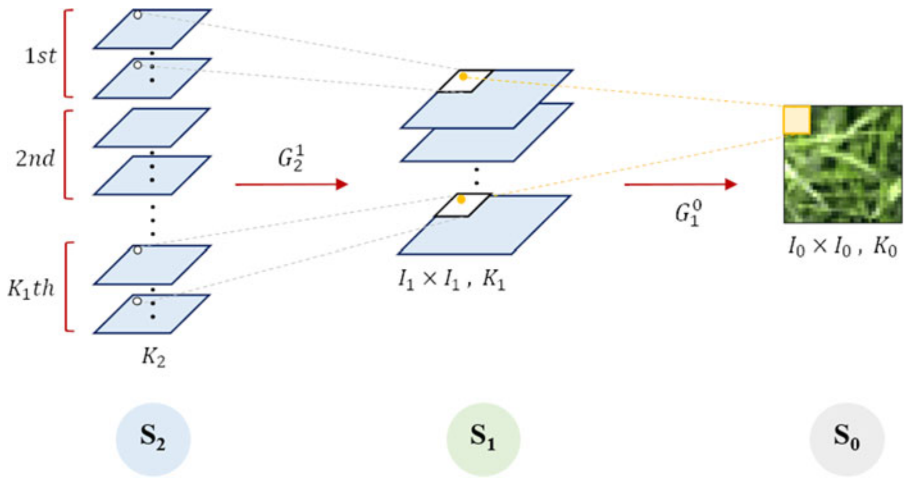

Fig. 5. Illustration of the generation process.

(iii) Match histograms of the generated Gaussian samples with the inverse CDFs in the chosen cluster.
(iv) Repeat Steps 1–3 if the generated sample of Step 3 has no value larger than a pre-set threshold.
(v) Perform the inverse transform of ICA and the inverse transform of PCA.
(vi) Reshape the 1D vector into a 3D tensor and this tensor is the generated sample in S_2.

The above procedure is named Independent Components Histogram Matching (ICHM). To conclude, there are two main modules in core sample generation: spatial dimension reduction (SDR) and independent components histogram matching as shown in Fig. 2.

D) Coarse-to-fine generation

In this section, we examine generation model G_{i+1}^i, whose role is to generate a sample in S_i given a sample in S_{i+1}. Analysis model, E_i^{i+1}, transforms S_i to S_{i+1} through the c/w Saab transform in the forward path. In the reverse path, we perform the inverse c/w Saab transform on generated samples in S_{i+1} to S_i. We take generation model G_2^1 as an example to explain the generation process from S_2 and to S_1. A generated sample in S_2 can be partitioned into K_1 groups as shown in the left part of Fig. 5. Each group of channels is composed of one DC channel and several low-frequency AC channels. The kth group of channels in S_2, whose number is denoted by $K_2^{(k)}$, is derived from the kth channel in S_1. We apply the inverse c/w Saab transform to each group individually. The inverse c/w Saab transform converts the tensor at the same spatial location across $K_2^{(k)}$ channels (represented by white dots in Fig. 5) in S_2 into a block of size $I_i \times I_i$ (represented by the white parallelogram in Fig. 5) in S_1, using the DC and AC components obtained in the fine-to-coarse analysis. After the inverse c/w Saab transform, the Saab coefficients in S_1 form a generated sample in S_1. The same procedure is repeated between S_1 and S_0.

Examples of several generated textures in core S_2, intermediate subspace S_1, and source S_0 are shown in Fig. 6. The DC channels generated in the core offer gray-scale low-resolution patterns of a generated sample. More local details are added gradually from S_2 to S_1 and from S_1 to S_0.

V. EXPERIMENTS

A) Experimental setup

The following hyper parameters (see Fig. 2) are used in our experiments.

- Input filter window size to Hop-1: $I_0 = 2$,
- Input filter window size to Hop-2: $I_1 = 2$,
- Selected channel numbers in Hop-1 (K_1): $6 \sim 10$,
- Selected channel numbers in Hop-2 (K_2): $20 \sim 30$.

The window size of the analysis filter is the same as the generation window size. All windows are non-overlapping with each other. The actual channel numbers K_1 and K_2 are texture-dependent. That is, we examine the energy distribution based on the PCA eigenvalues and choose the knee point where the energy becomes flat.

B) An example: brick wall generation

We show generated *brick_wall* patches of size 32×32 and 64×64 in Figs 7(a) and 7(c). We performed two-stage c/w Saab transforms on 32×32 patches and three-stage

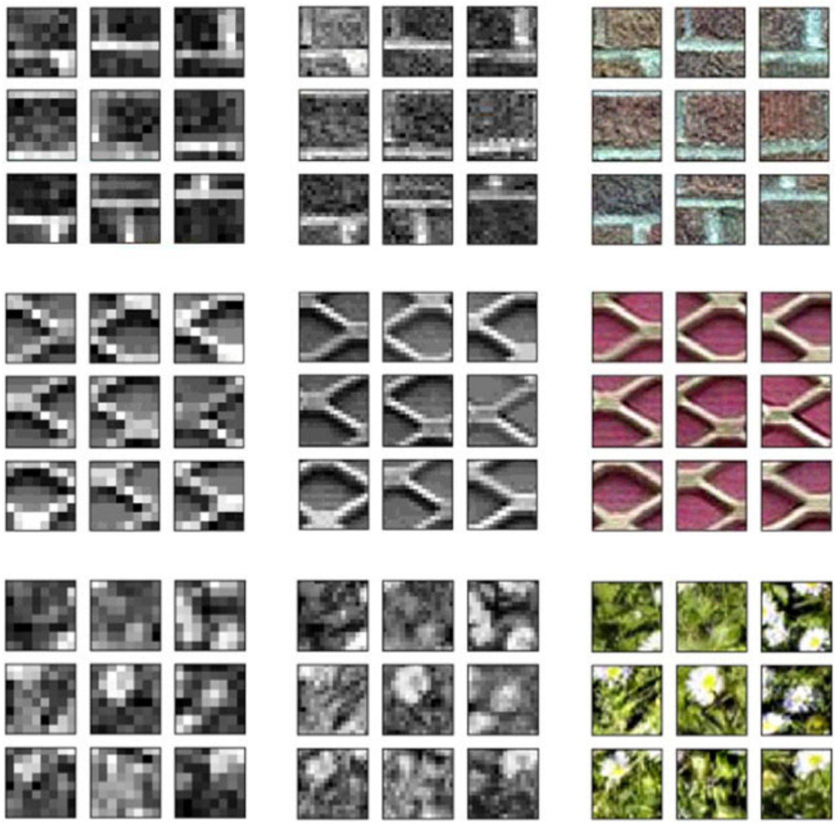

Fig. 6. Examples of generated DC maps in core S_2 (first column), generated samples in subspace S_1 (second co-lumn), and the ultimate generated textures in source S_0 (third column).

c/w Saab transforms on 64 × 64 patches, whose core subspace dimensions are 1728 and 4032, respectively. Patches in these figures were synthesized by running the TGHop method in 100rounds. Randomness in each round primarily comes from two factors: (1) random cluster selection, and (2) random seed vector generation.

Generated patches retain the basic shape of bricks and the diversity of brick texture. We observe some unseen patterns generated by TGHop, which are highlighted by red squared boxes in Figs 7(a) and 7(c). As compared with generated 32 × 32 patches, generated 64 × 64 patches were sometimes blurry (e.g., the one in the upper right corner) due to a higher source dimension.

As a non-parametric model, TGHop can choose multiple settings under the same pipeline. For example, it can select different channel numbers in $\tilde{S}_1$ and $\tilde{S}_2$ to derive different generation results. Four settings are listed in Table 1. The corresponding generation results are shown in Fig. 8. Dimensions decrease faster from (a) to (d). The quality of generated results becomes poorer due to smaller dimensions of the core subspace, $\tilde{S}_2$, and the intermediate subspace, $\tilde{S}_1$.

To generate larger texture images, we first generate 5000 texture patches and perform image quilting [13] with them. The results after quilting are shown in Figs 7(b) and 7(d). All eight images are of the same size, i.e., 256 × 256. They are obtained using different initial patches in the image quilting process. By comparing the two sets of stitched images, the global structure of the brick wall is better preserved using larger patches (i.e. of size 64 × 64) while its local detail is a little bit blurry sometimes.

C) Performance benchmarking with DL-based methods

C.1 Visual quality comparison

The quality of generated texture is usually evaluated by human eyes. Subjective user study is not a common choice for texture synthesis because different people have different standards to judge the quality of generated texture images. Evaluation metrics such as Inception Score [50] or Fréchet Inception Distance [51] are proposed for natural image generation. These metrics however demand sufficient number of samples to evaluate the distributions of generated images and real images, which are not suitable for texture synthesis containing only one image for reference. The value of loss function was used to measure texture quality or diversity for CNN-based methods [23, 26]. A lower loss however does not guarantee better generation quality. Since TGHop dose not have a loss function, we show generated results of two DL-based methods and TGHop side by side in Fig. 9 for 10 input texture images collected from [19, 20, 25] or the Internet. The benchmarking DL methods were proposed by Gatys et al. [20] and Ustyuzhaninov *et al.* [25]. By running their codes, we show their results in the second and third columns of Fig. 9, respectively, for comparison. These

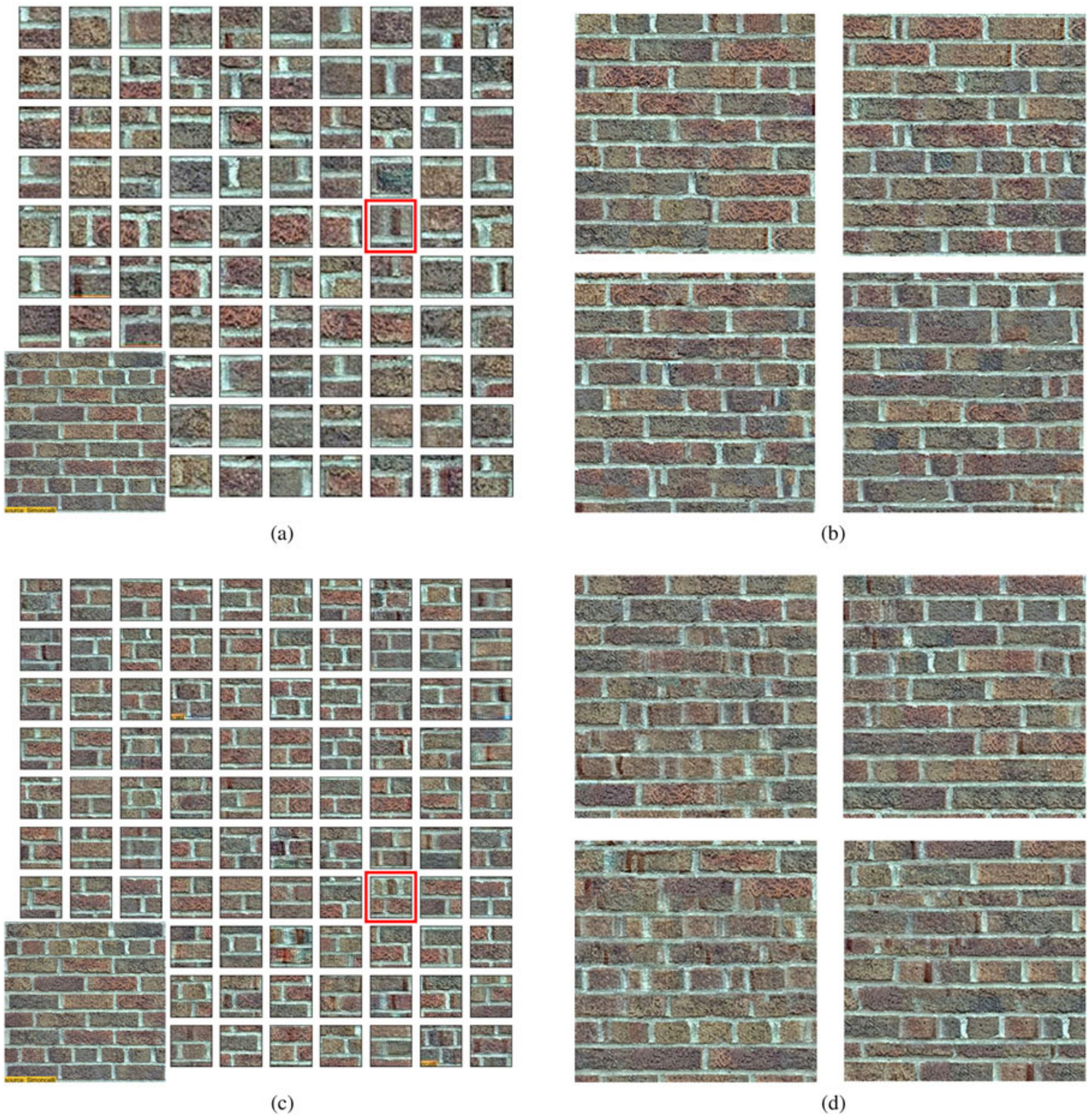

Fig. 7. Examples of generated *brick_wall* texture patches and stitched images of larger sizes, where the image in the bottom-left corner is the exemplary texture image and the patches highlighted by red squared boxes are unseen patterns. (a) Synthesized 32×32 patches. (b) Stitched images with 32×32 patches. (c) Synthesized 64×64 patches. (d) Stitched images with 64×64 patches.

results are obtained by default iteration numbers; namely, 2000 in [20] and 4000 in [20]. The results of TGHop are shown in the last three columns. The left two columns are obtained without SDR in two different runs while the last column is obtained with SDR. There is little quality degradation after dimension reduction of S_2 with SDR. For *meshed* and *cloud* textures, the brown fog artifact in [20, 25] is apparent. In contrast, it does not exist in TGHop. More generated images using TGHop are given in Fig. 10. As shown in Figs 9 and 10, TGHop can generate high-quality and visually pleasant texture images.

C.2 Comparison of generation time

We compare the generation time of different methods in Table 2. All experiments were conducted on the same machine composed of 12 CPUs (Intel Core i7-5930K CPU at 3.50GHz) and 1 GPU (GeForce GTX TITAN X). GPU was needed in two DL-based methods but not in TGHop. We set the iteration number to 1000 for [20] and 100 for [25]. TGHop generated 10K 32×32 patches for texture quilting. For all three methods, we show the time needed in generating one image of size 256×256 in Table 2, TGHop generates one texture image in 291.25 s while Gatys' method

Table 1. The settings of four generation models.

Setting	$\tilde{D}_0$	$\tilde{D}_1$	$\tilde{D}_2$
a	3072	2560	2048
b	3072	1536	768
c	3072	1280	512
d	3072	768	192

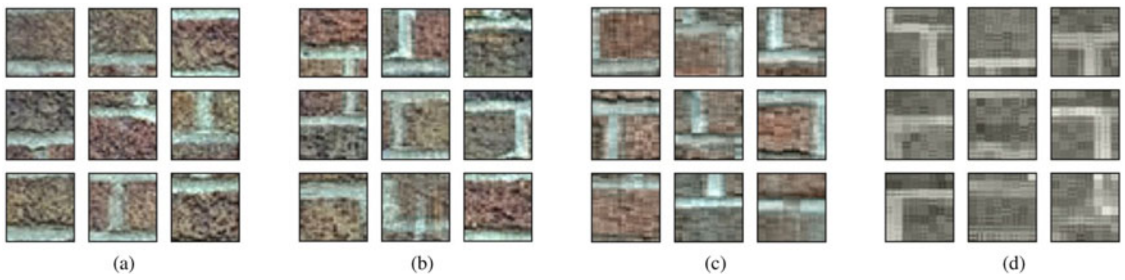

Fig. 8. Generated patches using different settings, where the numbers below the figure indicates the dimensions of S_0, S_1, and S_2, respectively. (a) $3072 \rightarrow 2560 \rightarrow 2048$. (b) $3072 \rightarrow 1536 \rightarrow 768$. (c) $3072 \rightarrow 1280 \rightarrow 512$. (d) $3072 \rightarrow 768 \rightarrow 192$.

Table 2. Comparison of time needed to generate one texture image.

Methods	Time (s)	Factor
Ustyuzhaninov *et al.* [25]	949.64	4.62×
Gatys *et al.* [20]	513.98	2.50×
TGHop with analysis overhead	291.25	1.42×
TGHop w/o analysis overhead	205.50	1×

Table 3. The time of three processes in our method.

Processes	Time (s)
Analysis (forward path)	85.75
Generation (reverse path)	197.42
Quilting	8.08

and Ustyuzhaninov's method demand 513.98 and 949.64 s, respectively. TGHop is significantly faster.

We break down the generation time of TGHop into three parts: (1) successive subspace analysis (i.e. the forward path), (2) core and successive subspace generation (i.e. the reverse path), and (3) the quilting process. The time required for each part is shown in Table 3. They demand 85.75, 197.42, and 8.08 s, respectively. To generate multiple images from the same exemplary texture, we run the first part only once, which will be shared by all generated texture images, and the second and third parts multiple times (i.e. one run for a new image). As a result, we can view the first part as a common overhead and count the last two parts as the time for single texture image generation. This is equal to 205.5 s. The two DL benchmarks do not have such a breakdown and need to go through the whole pipeline to generate one new texture image.

D) Comparison of model sizes

The model size is measured by the number of parameters. The size of TGHop is calculated below.

- Two-stage c/w Saab Transforms
  The forward analysis path and the reverse generation path share the same two-stage c/w Saab transforms. For an input RGB patch, the input tensor of size $2 \times 2 \times 3 = 12$ is transformed into a K_1-D tensor in the first-stage transform, leading a filter size of $12K_1$ plus one shared bias. For each of K_1 channels, the input tensor of size 2×2 is transformed into a tensor in the second stage transform. The sum of the output dimensions is K_2. The total parameter number for all K_1 channels is $4K_1K_2$ plus K_1 biases. Thus, the total number of parameters in the two-stage transforms is $13K_1 + 4K_1K_2 + 1$.
- Core Sample Generation
  Sample generation in the core contains two modules: SDR and ICHM. For the first module, SDR is implemented by K_2 PCA transforms, where the input of size $8 \times 8 = 64$ and the output is a K_{r_i} dimensional vector, yielding the size of each PCA transformation matrix to be $64 \times K_{r_i}$. The total number of parameters is $64 \times \sum_{i=1}^{K_2} K_{r_i} = 64D_r$, where D_r is the dimension of the concatenated output vector after SDR. For the second module, it has three components:
  - Interval representation $p_0, \ldots, p_{N-1}N$ parameters are needed for N cluster.
  - Transform matrices of FastICAIf the input vector is of dimension D_r and the output dimension of FastICA is K_{c_i} for the ith cluster, $i = 1, \ldots, N$, the total parameter number of all transform matrices is D_rF, where $F = \sum_{i=1}^{N} K_{c_i}$ is the number of CDFs.
  - Codebook size of quantized CDFsThe codebook contains the index, the maximum and the minimum values for each CDF. Furthermore, we have W clusters of CDF, where all CDFs in each cluster share the same bin structure of 256 bins. As a result, the total parameter number is $3F + 256W$.

  By adding all of the above together, the total parameter number in core sample generation is $64D_r + N + (D_r + 3)F + 256W$.

The above equations are summarized and an example is given in Table 4 under the experiment setting of $N = 50$, $K_1 = 9$, $K_2 = 22$, $D_r = 909$, $F = 2518$, and $W = 200$. The model size of TGHop is 2.4M. For comparison, the model sizes of [20] and [25] are 0.852 and 2.055M, respectively. A great majority of TGHop model parameters comes from ICHM(ii). Further model size reduction without sacrificing generated texture quality is an interesting extension of this work.

As compared with [28], SDR is a new module introduced in this work. It helps remove correlations of spatial responses to reduce the model size. We examined the impact of using different threshold γ in SDR on texture generation quality and model size with *brick_wall* texture. The same threshold is adopted for all channels to select PCA

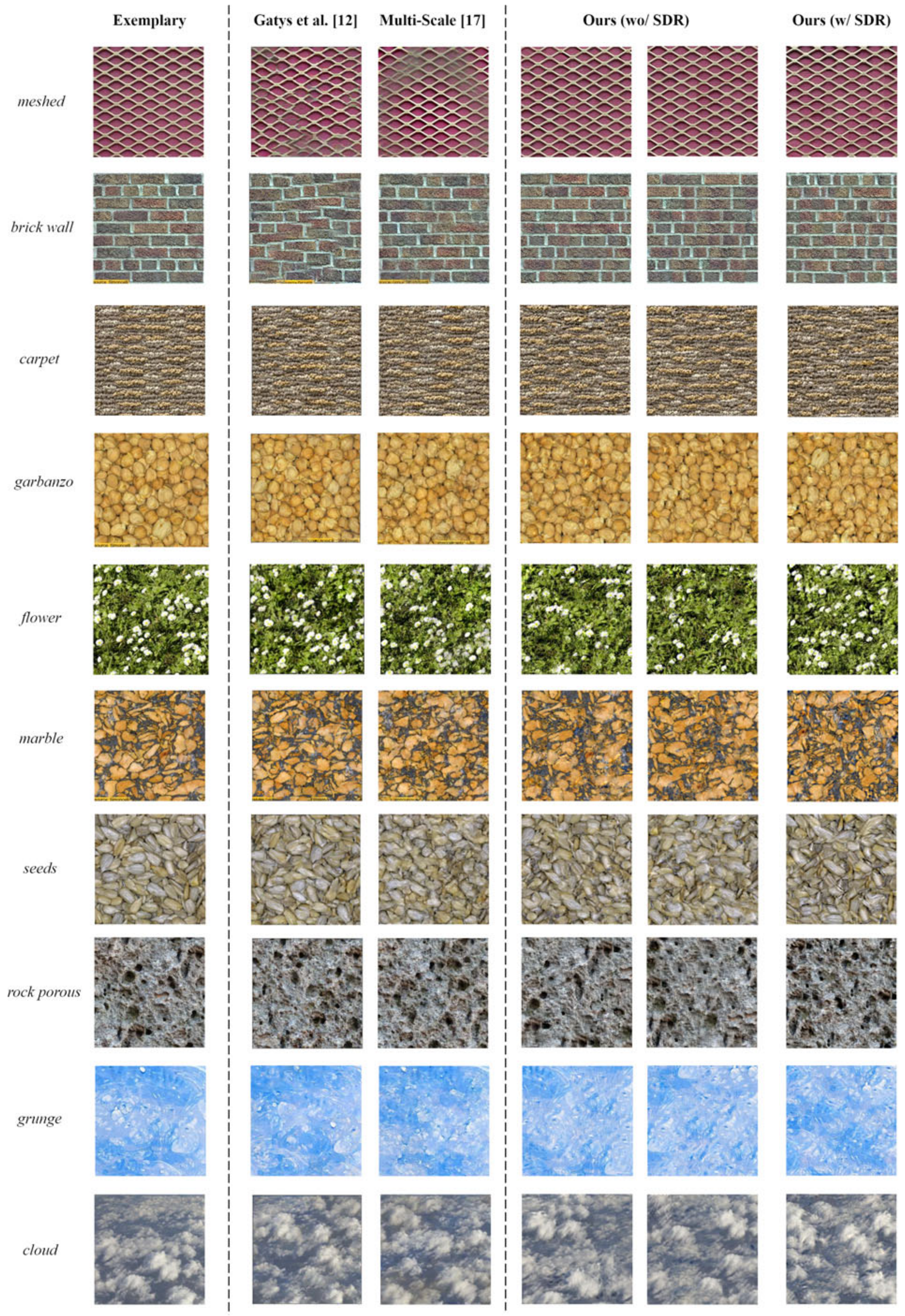

Fig. 9. Comparison of texture images generated by two DL-based methods and TGHop (from left to right): exemplary texture images, texture images generated by [20], by [25], two examples by TGHop without spatial dimension reduction (SDR) and one example by TGHop with SDR.

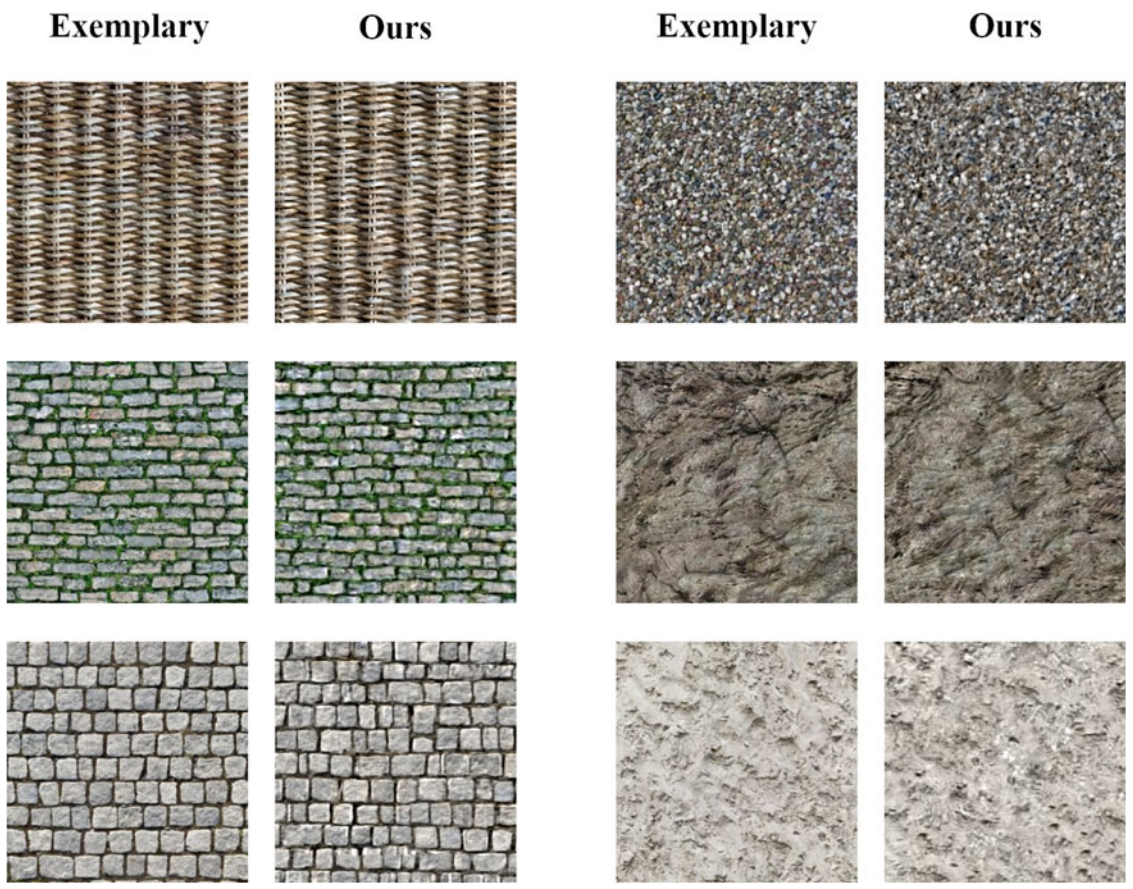

Fig. 10. More texture images generated by TGHop.

Table 4. The number of parameters of TGHop, under the setting of $\gamma = 0.01$, $N = 50$ $K_1 = 9$, $K_2 = 22$, $D_r = 909$, $F = 2,518$ and $W = 200$.

Module	Equation	Num. of param.
Transform - stage 1	$12K_1 + 1$	109
Transform - stage 2	$4K_1K_2 + K_1$	801
Core - SDR	$64D_r$	58 176
Core - ICHM(i)	N	50
Core - ICHM(ii)	FD_r	2 288 862
Core - ICHM(iii)	$3F + 256W$	58 754
Total		**2 406 752**

Table 5. The reduced dimension, D_r, and the model size as a function of threshold γ used in SDR.

γ	D_r	Number of parameters
0	1408	3.72M
0.0005	1226	3.26M
0.005	1030	2.74M
0.01	909	2.41M
0.02	718	1.88M
0.03	553	1.43M
0.04	399	1.00M
0.05	289	0.69M
0.1	102	0.19M

components. The dimension of reduced space, D_r, and the cluster number, N, are both controlled by threshold γ, used in SDR. $\gamma = 0$ represents all 64 PCA components are kept in SDR. We can vary the value of γ to get a different cluster number and the associated model size. The larger the value of γ, the smaller D_r and N and, thus, the smaller the model size as shown in Table 5. The computation given in Table 4 is under the setting of $\gamma = 0.01$.

A proper cluster number is important since too many clusters lead to larger model sizes while too few clusters result in bad generation quality. To give an example, consider the *brick_wall* texture image of size 256×256, where the dimension of S_2 is $8 \times 8 \times 22 = 1408$ with $K_2 = 22$. We extract 12 769 patches of size 32×32 (with stride 2) from this image. We conduct experiments with $N =$ 50, 80, 110, and 200 clusters and show generated patches in Fig. 11. As shown in (a), 50 clusters were too few and we see the artifact of over-saturation in generated patches. By increasing N from 50 to 80, the artifact still exists but is less apparent in (b). The quality improves furthermore when $N = 100$ as shown in (c). We see little quality improvement when N goes from 100 to 200. Furthermore, patches generated using different thresholds γ are shown in Fig. 12. We see little quality degradation from (a) to (f) while the dimension is reduced from 1408 to 553. Image blur shows up from (g) to (i), indicating that some details were discarded along with the corresponding PCA components.

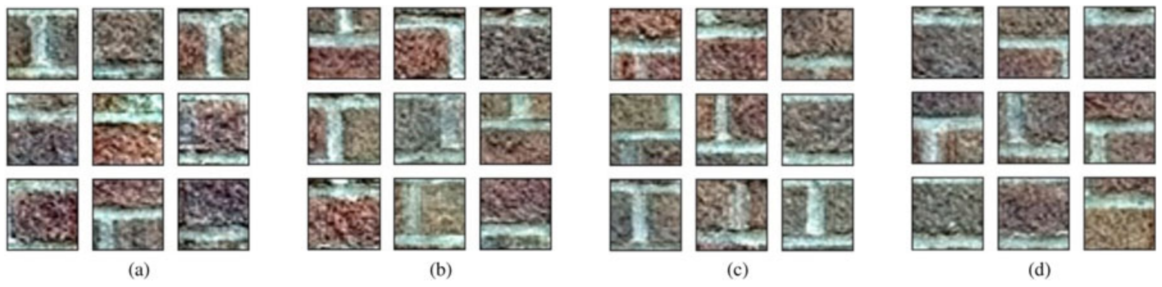

Fig. 11. Generated *brick_wall* patches using different cluster numbers in independent component histogram matching. (a) 50 clusters. (b) 80 clusters. (c) 110 clusters. (d) 200 clusters.

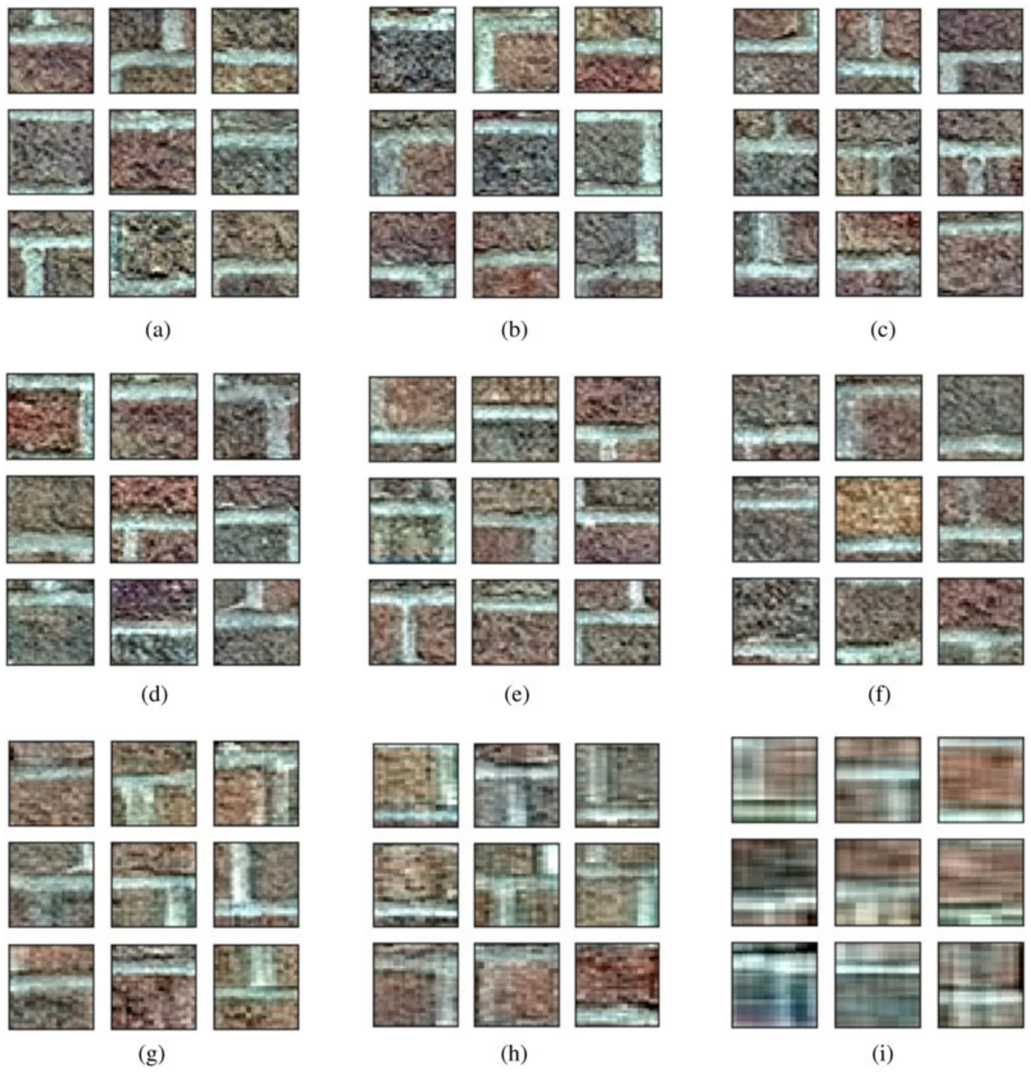

Fig. 12. Generated *brick_wall* patches using different threshold γ values in SDR. (a) $\gamma = 0$. (b) $\gamma = 0.0005$. (c) $\gamma = 0.005$. (d) $\gamma = 0.01$. (e) $\gamma = 0.02$. (f) $\gamma = 0.03$. (g) $\gamma = 0.04$. (h) $\gamma = 0.05$. (i) $\gamma = 0.1$.

VI. CONCLUSION AND FUTURE WORK

An explainable, efficient, and lightweight texture generation method, called TGHop, was proposed in this work. Texture can be effectively analyzed using the multi-stage c/w Saab transforms and expressed in the form of joint spatial-spectral representations. The distribution of sample texture patches was carefully studied so that we can generate samples in the core. Based on generated core samples, we can go through the reverse path to increase its spatial dimension. Finally, patches can be stitched to form texture images of a larger size. It was demonstrated by experimental results that TGHop can generate texture images of superior quality with a small model size and at a fast speed.

Future research can be extended in several directions. Controlling the growth of dimensions of intermediate subspaces in the generation process appears to be important. Is it beneficial to introduce more intermediate subspaces between the source and the core? Can we apply the same model for the generation of other images such as human faces, digits, scenes, and objects? Is it possible to generalize the framework to image inpainting? How does our generation model compare to GANs? These are all open and interesting questions for further investigation.

ACKNOWLEDGEMENTS

This research was supported by a gift grant from Mediatek. Computation for the work was supported by the University of Southern California's Center for High Performance Computing (hpc.usc.edu).

REFERENCES

[1] Tuceryan, M.; Jain, A.K.: Texture Analysis, *Handbook of Pattern Recognition and Computer Vision.* Singapore, World Scientific, 1993, 235–276.

[2] Chang, T.; Kuo, C.-C.J.: Texture analysis and classification with tree-structured wavelet transform. *IEEE Trans. Image Process.*, **2** (4) (1993), 429–441.

[3] Arivazhagan, S.; Ganesan, L.: Texture classification using wavelet transform. *Pattern Recognit. Lett.*, **24** (9–10) (2003), 1513–1521.

[4] Zhu, S.C.; Wu, Y.; Mumford, D.: Filters, random fields and maximum entropy (frame): towards a unified theory for texture modeling. *Int. J. Comput. Vis.*, **27** (2) (1998), 107–126.

[5] Wang, J.; Zhong, Y.; Li, Y.; Zhang, C.; Wei, Y.: Re-identification supervised texture generation, in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition.* IEEE, Long Beach, CA, USA, 2019, 11846–11856.

[6] Zhang, K.; Chen, H.-S.; Wang, Y.; Ji, X.; Kuo, C.-C.J.: Texture analysis via hierarchical spatial-spectral correlation (HSSC), in *2019 IEEE Int. Conf. on Image Processing (ICIP).* IEEE, Taipei, Taiwan, 2019, 4419–4423.

[7] Zhang, K.; Chen, H.-S.; Zhang, X.; Wang, Y.; Kuo, C.-C.J.: A data-centric approach to unsupervised texture segmentation using principle representative patterns, in *ICASSP 2019–2019 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP).* IEEE, Brighton, UK, 2019, 1912–1916.

[8] Oechsle, M.; Mescheder, L.; Niemeyer, M.; Strauss, T.; Geiger, A.: Texture fields: learning texture representations in function space, in *Proc. of the IEEE/CVF Int. Conf. on Computer Vision.* IEEE, Seoul, Korea, 2019, 4531–4540.

[9] Hsu, C.-C.; Kang, L.-W.; Lin, C.-W.: Temporally coherent superresolution of textured video via dynamic texture synthesis. *IEEE Trans. Image Process.*, **24** (3) (2015), 919–931.

[10] De Bonet, J.S.: Multiresolution sampling procedure for analysis and synthesis of texture images, in *Proc. of the 24th Annual Conf. on Computer Graphics and Interactive Techniques.* ACM Press, New York, NY, USA, 1997, 361–368.

[11] Efros, A.A.; Leung, T.K.: Texture synthesis by non-parametric sampling, in *Proc. of the Seventh IEEE Int. Conf. on Computer Vision*, vol. 2. IEEE, Kerkyra, Greece, 1999, 1033–1038.

[12] Wei, L.-Y.; Levoy, M.: Fast texture synthesis using tree-structured vector quantization, in *Proc. of the 27th Annual Conf. on Computer Graphics and Interactive Techniques.* ACM Press, New York, NY, USA, 2000, 479–488.

[13] Efros, A.A.; Freeman, W.T.: Image quilting for texture synthesis and transfer, in *Proc. of the 28th Annual Conf. on Computer Graphics and Interactive Techniques.* ACM Press, New York, NY, USA, 2001, 341–346.

[14] Liang, L.; Liu, C.; Xu, Y.-Q.; Guo, B.; Shum, H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Trans. Graphics (ToG)*, **20** (3) (2001), 127–150.

[15] Cohen, M.F.; Shade, J.; Hiller, S.; Deussen, O.: Wang tiles for image and texture generation. *ACM Trans. Graphics (TOG)*, **22** (3) (2003), 287–294.

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

[17] Kwatra, V.; Essa, I.; Bobick, A.; Kwatra, N.: Texture optimization for example-based synthesis, in *ACM SIGGRAPH 2005 Papers.* Association for Computing Machinery, New York, NY, USA, 2005, 795–802.

[18] Heeger, D.J.; Bergen, J.R.: Pyramid-based texture analysis/synthesis, in *Proc. of the 22nd Annual Conf. on Computer Graphics and Interactive Techniques.* Association for Computing Machinery, New York, NY, USA, 1995, 229–238.

[19] Portilla, J.; Simoncelli, E.P.: A parametric texture model based on joint statistics of complex wavelet coefficients. *Int. J. Comput. Vis.*, **40** (1) (2000), 49–70.

[20] Gatys, L.; Ecker, A.S.; Bethge, M.: Texture synthesis using convolutional neural networks, in *Advances in Neural Information Processing Systems.* MIT Press, Cambridge, MA, USA, 2015, 262–270.

[21] Liu, G.; Gousseau, Y.; Xia, G.-S.: Texture synthesis through convolutional neural networks and spectrum constraints, in *2016 23rd Int. Conf. on Pattern Recognition (ICPR).* IEEE, Cancun, Mexico, 2016, 3234–3239.

[22] Risser, E.; Wilmot, P.; Barnes, C.: Stable and controllable neural texture synthesis and style transfer using histogram losses, *arXiv preprint arXiv:1701.08893*, 2017.

[23] Li, Y.; Fang, C.; Yang, J.; Wang, Z.; Lu, X.; Yang, M.-H.: Diversified texture synthesis with feed-forward networks, in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition.* IEEE, Honolulu, HI, USA, 2017, 3920–3928.

[24] Li, Y.; Fang, C.; Yang, J.; Wang, Z.; Lu, X.; Yang, M.-H.: Universal style transfer via feature transforms, in *Advances in Neural Information Processing Systems.* Curran Associates Inc., Red Hook, NY, USA, 2017, 386–396.

[25] Ustyuzhaninov, I.; Brendel, W.; Gatys, L.A.; Bethge, M.: What does it take to generate natural textures?. Presented at *ICLR*, 2017. [Online] Available: https://openreview.net/forum?id=BJhZeLsxx

[26] Shi, W.; Qiao, Y.: Fast texture synthesis via pseudo optimizer, in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition.* IEEE, Virtual, 2020, 5498–5507.

[27] Chen, Y.; Rouhsedaghat, M.; You, S.; Rao, R.; Kuo, C.-C.J.: Pixelhop++: a small successive-subspace-learning-based (ssl-based) model for image classification, *arXiv preprint arXiv:2002.03141*, 2020.

[28] Lei, X.; Zhao, G.; Kuo, C.-C.J.: Nites: a non-parametric interpretable texture synthesis method, in *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conf. (APSIPA ASC).* IEEE, Virtual, 2020, 1698–1706.

[29] Ulyanov, D.; Lebedev, V.; Vedaldi, A.; Lempitsky, V.S.: Texture networks: feed-forward synthesis of textures and stylized images, in Proc. of the 33rd Int. Conf. on Int. Conf. on Machine Learning, vol. 48. JMLR.org, New York, NY, USA, 2016, 1349–1357.

[30] Choo, J.; Liu, S.: Visual analytics for explainable deep learning. *IEEE Comput. Graph. Appl.*, **38** (4) (2018), 84–92.

[31] Monga, V.; Li, Y.; Eldar, Y.C.: Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Process. Mag.*, **38** (2) (2021), 18–44.

[32] Kuo, C.-C.J.: Understanding convolutional neural networks with a mathematical model. *J. Vis. Commun. Image Represent.*, **41**, (2016), 406–413.

[33] Kuo, C.-C.J.: The CNN as a guided multilayer recos transform [lecture notes]. *IEEE Signal Process. Mag.*, **34** (3) (2017), 81–89.

[34] Kuo, C.-C.J.; Zhang, M.; Li, S.; Duan, J.; Chen, Y.: Interpretable convolutional neural networks via feedforward design. *J. Vis. Commun. Image Represent*, **60** (2019), 346–359.

[35] Chen, Y.; Kuo, C.-C.J.: Pixelhop: A successive subspace learning (ssl) method for object recognition. *J. Vis. Commun. Image Represent.* **70** (2020), 102749.

[36] Chen, H.-S.; Rouhsedaghat, M.; Ghani, H.; Hu, S.; You, S.; Kuo, C.-C.J.: Defakehop: a light-weight high-performance deepfake detector, in *2021 IEEE Int. Conf. on Multimedia and Expo (ICME)*. IEEE, Virtual, 2021, 1–6.

[37] Zhang, K.; Wang, B.; Wang, W.; Sohrab, F.; Gabbouj, M.; Kuo, C.-C.J.: Anomalyhop: an ssl-based image anomaly localization method, *arXiv preprint arXiv:2105.03797*, 2021.

[38] Kadam, P.; Zhang, M.; Liu, S.; Kuo, C.-C.J.: R-pointhop: a green, accurate and unsupervised point cloud registration method, *arXiv preprint arXiv:2103.08129*, 2021.

[39] Liu, X.; Xing, F.; Yang, C.; Kuo, C.-C.J.; Babu, S.; Fakhri, G.E.; Jenkins, T.; Woo, J.: Voxelhop: successive subspace learning for ALS disease classification using structural MRI, *arXiv preprint arXiv:2101.05131*, 2021.

[40] Zhang, M.; Wang, Y.; Kadam, P.; Liu, S.; Kuo, C.-C.J.: Pointhop++: a lightweight learning model on point sets for 3D classification, in *2020 IEEE Int. Conf. on Image Processing (ICIP)*. IEEE, Virtual, 2020, 3319–3323.

[41] Zhang, M.; You, H.; Kadam, P.; Liu, S.; Kuo, C.-C.J.: Pointhop: an explainable machine learning method for point cloud classification. *IEEE Trans. Multimedia,* **22** *(7)* (2020), 1744–1755.

[42] Zhang, M.; Kadam, P.; Liu, S.; Kuo, C.-C.J.: Unsupervised feedforward feature (UFF) learning for point cloud classification and segmentation, in *2020 IEEE Int. Conf. on Visual Communications and Image Processing (VCIP)*. IEEE, Virtual, 2020, 144–147.

[43] Kadam, P.; Zhang, M.; Liu, S.; Kuo, C.-C.J.: Unsupervised point cloud registration via salient points analysis (SPA), in *2020 IEEE Int. Conf. on Visual Communications and Image Processing (VCIP)*. IEEE, Virtual, 2020, 5–8.

[44] Rouhsedaghat, M.; Wang, Y.; Ge, X.; Hu, S.; You, S.; Kuo, C.-C.J.: Facehop: a light-weight low-resolution face gender classification method, *arXiv preprint arXiv:2007.09510*, 2020.

[45] Rouhsedaghat, M.; Monajatipoor, M.; Azizi, Z.; Kuo, C.-C.J.: Successive subspace learning: an overview, *arXiv preprint arXiv:2103.00121*, 2021.

[46] Hyvärinen, A.; Hoyer, P.O.; Oja, E.: Sparse code shrinkage: denoising by nonlinear maximum likelihood estimation, in *Advances in Neural Information Processing Systems.* MIT Press, Cambridge, MA, USA, 1999, 473–479.

[47] Bartlett, M.S.; Movellan, J.R.; Sejnowski, T.J.: Face recognition by independent component analysis. *IEEE Trans. Neural Netw.*, **13** (6) (2002), 1450–1464.

[48] Mitianoudis, N.; Stathaki, T.: Pixel-based and region-based image fusion schemes using ICA bases. *Inf. Fusion*, **8** (2) (2007), 131–142.

[49] Hyvärinen, A.; Oja, E.: Independent component analysis: algorithms and applications. *Neural Netw.*, **13** (4–5) (2000), 411–430.

[50] Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; Chen, X.: Improved techniques for training gans, in *Advances in Neural Information Processing Systems.* Curran Associates Inc., Red Hook, NY, USA, 2016, 2234–2242.

[51] 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, in *Advances in Neural Information Processing Systems.* Curran Associates Inc., Red Hook, NY, USA, 2017, 6626–6637.

Xuejing Lei received the B.E. degree in Automation from Xi'an Jiaotong University, Xi'an, China, in 2016, and the M.S. degree in Electrical Engineering from the University of Southern California, Los Angeles, USA, in 2018. She is currently pursuing the Ph.D. degree in Electrical and Computer Engineering from the University of Southern California, Los Angeles, USA. Her research interests lie in machine learning and computer vision including video segmentation and tracking, image generative modeling, and 3D object reconstruction.

Ganning Zhao received her B.E. degree in Automation from Guangdong University of Technology, Guangzhou, China, in 2019. She is currently pursuing the Ph.D. degree in Electrical Engineering from the University of Southern California, Los Angeles, USA. Her research interests lie in machine learning and computer vision including image generative modeling and 3D point cloud.

Kaitai Zhang received the B.S. degree in Physics from Fudan University, Shanghai, China, in 2016, and the Ph.D. degree in Electrical and Computer Engineering from the University of Southern California (USC), Los Angeles, USA, in 2021. He was fortunate enough to join the Media Communication Lab at USC and have Prof. C.-C. Jay Kuo as his Ph.D. advisor, and his research interests include image processing, computer vision, and machine learning.

C.-C. Jay Kuo (F'99) received the B.S. degree in Electrical Engineering from the National Taiwan University, Taipei, Taiwan, in 1980, and the M.S. and Ph.D. degrees in Electrical Engineering from the Massachusetts Institute of Technology, Cambridge, in 1985 and 1987, respectively. He is currently the Director of the Multimedia Communications Laboratory and a Distinguished Professor of electrical engineering and computer science at the University of Southern California, Los Angeles. His research interests include digital image/video analysis and modeling, multimedia data compression, communication and networking, and biological signal/image processing. He is a co-author of 310 journal papers, 970 conference papers, 30 patents, and 14 books. Dr. Kuo is a Fellow of the American Association for the Advancement of Science (AAAS) and The International Society for Optical Engineers (SPIE).