NITES: A Non-Parametric Interpretable Texture Synthesis Method

Xuejing Lei, Ganning Zhao and C.-C. Jay Kuo
University of Southern California, USA
E-mail: {xuejing, ganningz, jckuo}@usc.edu

***Abstract*—A non-parametric interpretable texture synthesis method, called NITES, is proposed in this work. Although automatic synthesis of visually pleasant texture can currently be achieved by deep neural networks, the associated generation models are mathematically intractable and their training demands higher computational cost. NITES offers a new texture synthesis solution to address these shortcomings. NITES is mathematically transparent and efficient in training and inference. The input is a single exemplary texture image. The NITES method crops out patches from the input and analyzes the statistical properties of these texture patches to obtain their joint spatial-spectral representations. Then, the probabilistic distributions of samples in the joint spatial-spectral spaces are characterized. Finally, numerous texture images that are visually similar to the exemplary texture image can be generated automatically. Experimental results are provided to show the superior quality of generated texture images and efficiency of the proposed NITES method in terms of both training and inference time.**

I. Introduction

Automatic synthesis of visually pleasant texture that resembles exemplary texture finds applications in computer graphics. Texture synthesis has been studied for several decades since it is also of theoretical interest in texture analysis and modeling. Texture can be synthesized pixel-by-pixel [1], [2], [3] or patch-by-patch [4], [5], [6], [7], [8] based on an exemplary pattern. For the pixel-based synthesis, a pixel conditioned on its squared neighbor was synthesized using the conditional probability and estimated by a statistical method in [2]. Generally, patch-based texture synthesis yields higher quality than pixel-based texture synthesis. Yet, searching the whole image for patch-based synthesis is extremely slow [2], [5]. To speed up the process, small patches of the exemplary texture can be stitched together to form a larger region [4], [6], [9]. Although these methods can produce texture of higher quality, the diversity of produced textures is limited. Besides texture synthesis in the spatial domain, texture images from the spatial domain can be transformed to the spectral domain with certain filters (or kernels), thus exploiting the statistical correlation of filter responses for texture synthesis. Commonly used kernels include the Gabor filters [10] and the steerable pyramid filter banks [11].

We have witnessed amazing quality improvement of synthesized texture over the last five to six years due to the resurgence of neural networks. Texture synthesis based on deep learning (DL), such as Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs), yield visually pleasing results. DL-based methods learn transform kernels from numerous training data [12], [13], [14], [15], [16], [17], [18] through end-to-end optimization. However, these methods have two main shortcomings: 1) a lack of mathematical transparency and 2) a higher training and inference complexity. To address these drawbacks, we investigate a non-parametric and interpretable texture synthesis method, called NITES.

NITES consists of three steps. First, it analyzes the exemplary texture to obtain its joint spatial-spectral representations. Second, the probabilistic distributions of training samples in the joint spatial-spectral spaces are characterized. Finally, new texture images are generated by mimicking probabilities of source texture images. In particular, we adopt a data-driven transform, known as the channel-wise (c/w) Saab transform [19], which provides a powerful representation in the joint spatial-spectral space. The c/w Saab transform is derived from the successive subspace learning (SSL) theory [20], [21], [22], [23]. We will show that NITES can generate high quality texture at lower complexity.

The rest of the paper is organized as follows. The framework of successive subspace embedding and generation is described in Sec. II. The NITES method is proposed in Sec. III. Experimental results are shown in Sec. IV. Concluding remarks are given in Sec. V.

II. Successive Subspace Embedding and Generation

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

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 embedding relationship

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

and

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

This concept is illustrated in Fig. 1.

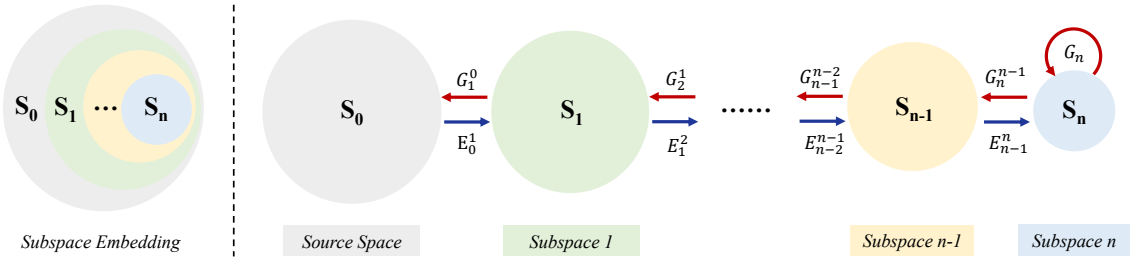

Fig. 1. Illustration of the concept of successive subspace embedding and generation. A sequence of subspace $S_1, \ldots, S_n$ is successively embedded into source space S_0 through subspace embedding processes indicated by blue arrows. Red arrows indicate subspace generation processes.

We use texture embedding and generation as an example to explain this concept. 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 $N \times N$. The input set then has a dimension of $3N^2$ and its augmented space S_0 has a dimension of $D_0 = 3N^2 + 1$. If $N = 32$, 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 embedded subspaces $S_0, S_1, \cdots, S_n$ with decreasing dimensions. We call S_0 and S_n the "source" space and the "core" subspace, respectively. We need to find an effective embedded subspace S_{i+1} from S_i, and such an embedding model is denoted by E_i^{i+1}. Proper subspace embedding is important since it determines how to decompose an input space into the direct sum of two subspaces in the forward embedding path. Although we choose one of the two for further processing, we also need to record the probabilistic relationship of the two decomposed subspaces so that samples of diversity and fidelity can be generated in the reverse generation path. This forward embedding process is called Successive Subspace Embedding (SSE).

In the reverse generation 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 named Core Subspace Generation (CSG). 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 Successive Subspace Generation (SSG). In Fig. 1, we use blue and red arrows to indicate a sequence of subspace embedding and generation processes, respectively. This idea can be implemented as a non-parametric method since we can choose subspaces $S_1, \cdots, S_n$, flexibly in a feedforward manner. One specific design is elaborated in the next section.

III. Proposed NITES Method

The proposed NITES method is presented in this section. To begin with, we provide a system overview of the NITES method in Sec. III-A. Next, we discuss the Successive Subspace Embedding (SSE) scheme based on the c/w Saab transform in Sec. III-B. After that, we examine the problem of Core Subspace Generation (CSG) in Sec. III-C. Finally, we describe the Successive Subspace Generation (SSG) process in Sec. III-D.

A. System Overview

An overview of the NITES method is given in Fig. 2. The exemplary color texture image has a spatial resolution of 256×256 and three RGB channels. We are interested in generating multiple texture images that are visually similar to the exemplary texture. By randomly cropping texture patches of size 32×32 out of the source image, we obtain a collection of texture samples, which serves as the input to the NITES system. The dimension of these patches is $32 \times 32 \times 3 = 3072$. Their augmented vectors form source space S_0. The NITES 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 successive subspace embedding process. The red arrows go from right to left, denoting the successive subspace generation process. We can generate as many texture patches as desired using this procedure. In order to generate a texture image of a larger size, we perform image quilting [4] based on synthesized patches.

B. Successive Subspace Embedding (SSE)

The global structure of an image (or an image patch) can be well characterized by spectral analysis, yet it is weak 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. Embedding model E_0^1 finds a proper subspace, S_1, in S_0 while embedding model E_1^2 finds a proper subspace, S_2, in S_1. As shown in Fig. 2, NITES 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 achieve the Successive Subspace Embedding (SSE) task. In the following, we provide a brief review on the Saab transform [22] and the c/w Saab transform [19].

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.

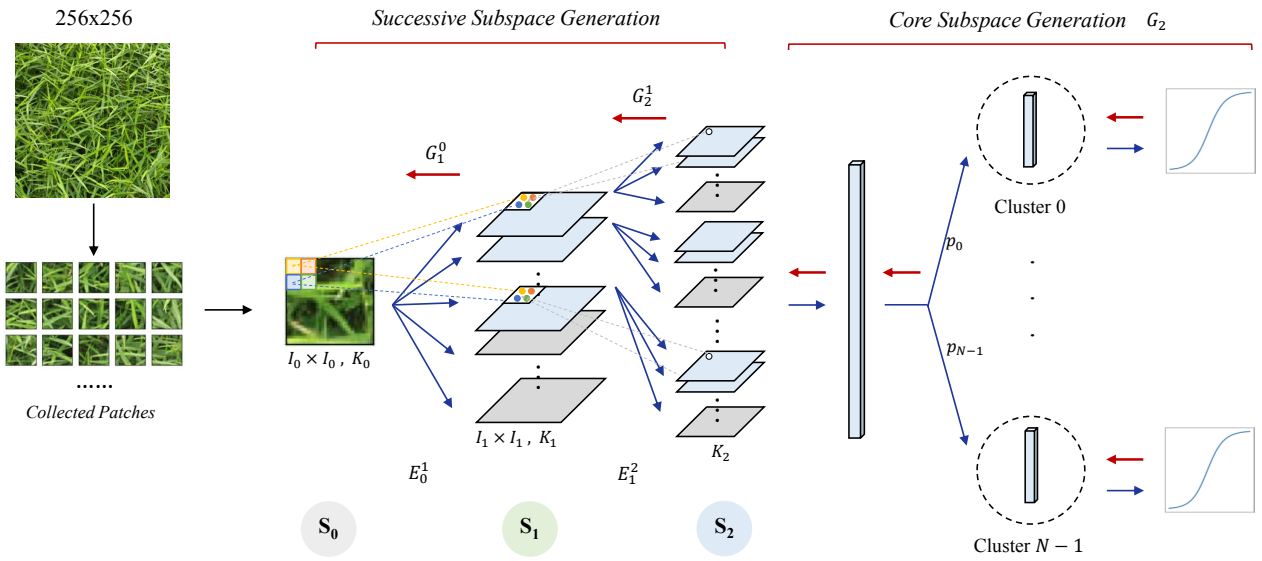

Fig. 2. An overview of the proposed NITES method. A number of patches are collected from the exemplary texture image, forming source space S_0. Subspace S_1 and S_2 are constructed through embedding model E_0^1 and E_1^2. Input filter window sizes to Hop-1 and Hop-2 are denoted as I_0 and I_1. Selected 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 subspace generation followed by successive subspace generation. The model for core subspace generation is denoted as G_2 and the models for successive subspace generation are denoted as G_2^1 and G_1^0.

The DC filter is a all-ones filter weighted by a constant. AC filters are obtained by applying the principal component analysis (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 [20], [21] 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 date-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 12D tensor becomes 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 [19] 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 process. It should also be emphasized that users can flexibly choose channel numbers in Hop-1 and Hop-2. Thus, NITES 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 Subspace Generation (CSG)

We begin with the generation of samples in S_n, which is accomplished by generation model G_n, in the reverse generation path. In the current case, $n = 2$. We need to characterize the sample statistics in core subspace S_2 for the purpose of synthesis. The statistics of a large texture image can be complicated. We leverage the semi-periodic property of texture so as to focus on patches of size 32×32, whose statistics can be analyzed more easily.

After the two-stage c/w Saab transform, the dimension of S_2 is typically less than 2000. S_2 is in form of c/w Saab coefficients. We flatten these coefficients into a 1D vector, denoted by $\mathbf{z}$, which is a random vector in S_2. To simplify the distribution characterization of a high-dimensional random vector, we cluster training samples into clusters and transform random vectors in each cluster into a set of independent random variables. This is inspired by the divide-and-conquer principle; namely, breaking down a difficult problem into several easier sub-problems and solving them individually.

We adopt a hierarchical K-Means clustering algorithm [24] to cluster the training samples of $\mathbf{z}$ into N clusters, which are denoted by $\{C_i\}$, $i = 0, \cdots, N-1$. 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

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

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 need to 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 [25], face recognition [26], and image infusion [27].

Our implementation is detailed below.

1) Apply PCA to $\mathbf{z}$ in cluster C_i for dimension reduction and data whitening.
2) Apply FastICA [28], which is conceptually simple, computationally efficient and robust to outliers, to the PCA output.
3) 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.
4) 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 (VQ) and store the codebook of quantized CDFs.

We encode $P(\mathbf{z} \in C_i)$ in Eq. (3) to be the length of a segment in the unit interval, $[0, 1]$. 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. 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:

1) Select a random number from the uniform random number generator to determine the cluster index.
2) Draw a set of samples independently from the Gaussian distribution.
3) Match histograms of the generated Gaussian samples with the inverse CDFs in the chosen cluster.
4) Repeat Steps 1-3 if the generated sample of Step 3 has no value larger than a pre-set threshold.
5) Perform the inverse transform of ICA and the inverse transform of PCA.
6) Reshape the 1D vector into a 3D tensor and this tensor is the generated sample in S_2.

D. Successive Subspace Generation (SSG)

In this section, we examine generation model G_{i+1}^i, whose role is to draw a sample in S_i given a sample in S_{i+1}. The CSG process can generate a sample that captures the global structure of an image patch but lacks in local detail such as boundaries between regions. Embedding model, E_i^{i+1}, provides a good description of local detail, which is obtained by the c/w Saab transform. Thus, in the generation process, we conduct the inverse c/w Saab transform on the generated sample in S_{i+1}.

It is worth noting that both Saab and c/w Saab transforms have the forward and the inverse transforms. If no high-frequency channels are removed, they are lossless transforms. If some high-frequency channels are removed, they become lossy transforms. In the current case, we use lossy transforms for dimension reduction in the successive subspace embedding process and lossy inverse transforms in the successive subspace generation process. The inverse c/w Saab transform is a deterministic one once the forward transform is decided. It can be used for signal reconstruction.

In the following, we take generation model G_2^1 from S_2 and to S_1 as an example to explain the generation process. A generated sample in S_2 can be partitioned into K_1 groups as shown in the left part of Fig. 3. 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. 3) in S_2 into a block of size $I_i \times I_i$ (represented by the white parallelogram in Fig. 3)

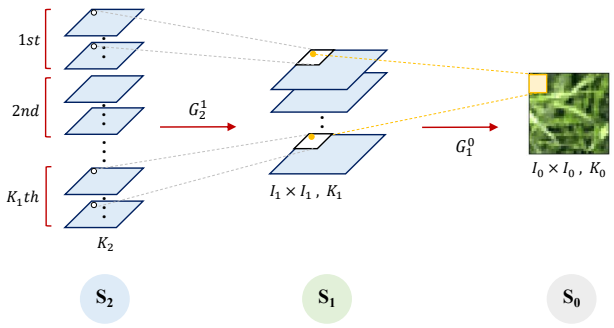

Fig. 3. Illustration of successive subspace generation process. K_2 channels in subspace S_2 are cut into K_1 groups. G_2^1 converts the same spatial location of each group of channels to a block of size $I_1 \times I_1$ at each of K_1 channels in space/subspace S_1. G_1^0 converts the same spatial location of K_1 channels to a block of size $I_1 \times I_1$ of three RGB channels.

in S_1, using the DC and AC components obtained in the successive subspace embedding process. 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 subspace S_0, intermediate subspace S_1 and source space S_0 are shown in Fig. 4, where we display the generated DC channel of group 1 (the first channel in Fig. 3) in S_2 and S_1 and the generated texture patch in S_0. These DC channels offer gray-scale low-resolution patterns of a generated sample. NITES can generate samples containing more local detail gradually.

IV. 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 embedding 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 eigenvalue plot and choose the knee point where the energy becomes nearly flat.

B. An Example: Brick Wall Texture Generation

We show generated *brick_wall* texture patches of size 32×32 and 64×64 in Figs. 5(a) and (c). We performed two-stage c/w Saab transforms on 32×32 patches and three-stage c/w Saab transforms on 64×64 patches, whose core subspace dimensions are equal to 1728 and 4032, respectively. Patches in these figures were synthesized by running the NITES method in one hundred rounds. 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 NITES, which are highlighted by red squared boxes in Figs. 5 (a) and (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 generation model, NITES freely chooses 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 I. The corresponding generation results are shown in Fig. 6. 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, $\tilde{S}_1$.

TABLE I
The settings of our four generation model.

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

To generate larger texture images, we first generate 5,000 texture patches and perform image quilting [4] with them. The results after quilting are shown in Figs. 5 (b) and (d). All eight images are of the same size, 256×256. They are obtained using different initial patches for 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 CNN-based Methods

Visual Quality Comparison. The quality of synthesized texture is usually evaluated by human eyes. A diversity loss function was proposed to measure texture diversity for CNN-based methods [18], [15]. Yet, NITES dose not have a loss function. Thus, we show synthesis results of two CNN methods and NITES side by side in Fig. 7. Exemplary texture images are collected from those in [12], [17], [11] or the Internet for illustration purposes. The two benchmarking CNN methods were proposed by Gatys *et al.* [12] and Ustyuzhaninov *et al.* [17]. We ran the codes provided by them, and show their results in the second and third columns of Fig. 7, respectively. We used the default setting of the iteration number, which is 2000 in [12] and 4000 in [12] for visualization. Two results generated by NITES are shown in the last two columns. They were obtained in two different runs. For texture *meshed*, we see the brown fog artifact in [12], [17], which is visually apparent. However, it does not show up in our generation. As demonstrated by these examples, NITES can generate high quality and visually pleasant texture images.

Comparison of Texture Generation Time. It is worthwhile to compare the synthesis time of different texture image generation methods. The results are shown in Table II. All

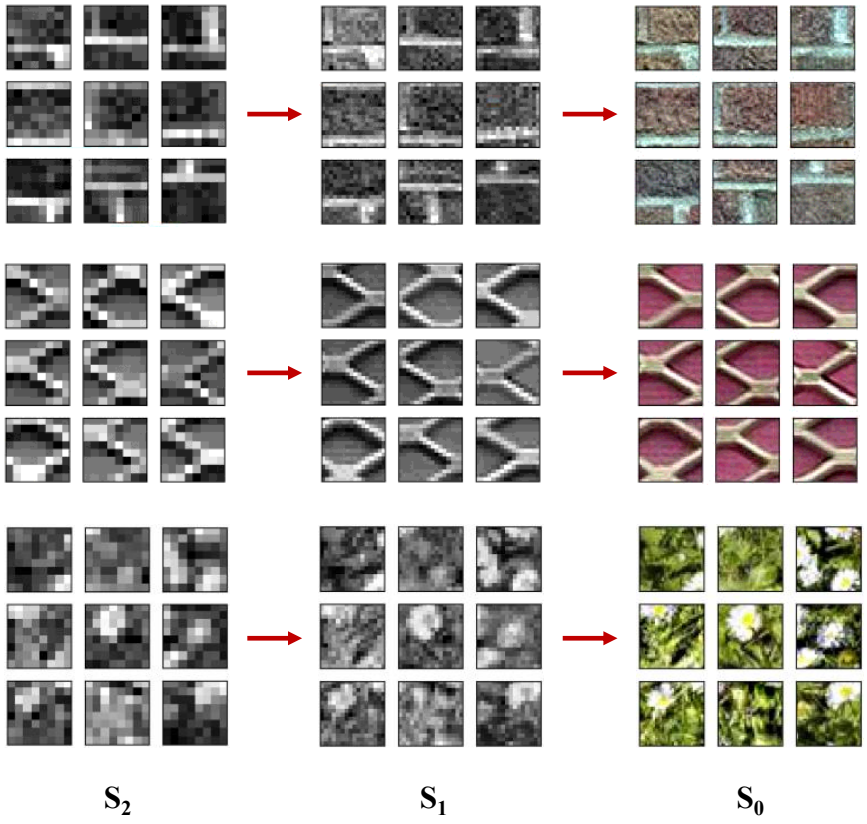

Fig. 4. Examples of several generated textures in core subspace S_0, intermediate subspace S_1 and source space S_0.

experiments were conducted in the same machine composed of 12 CPUs (Intel Core i7-5930K CPU at 3.50GHz) and 1 GPU (GeForce GTX TITAN X). No GPU was needed in our model but it was needed in [12], [17]. We set the iteration number to 1000 for [12] and 100 for [17]. NITES generated 5,000 32×32 texture patches, 81 of which were stitched into the final texture image. For all three methods, we measured the time needed in generating one image of size 256×256. As shown in Table II, NITES generates one texture image in 213.04 seconds while Gatys' method and Ustyuzhaninov's method demand 513.98 and 949.64 seconds, respectively. NITES is significantly faster.

TABLE II
COMPARISON OF TIME NEEDED TO GENERATE ONE TEXTURE IMAGE.

Methods	Time (seconds)
Gatys et al. [12]	513.98
Multi-Scale [17]	949.64
NITES (Ours)	213.04

Breakdown of NITES' Generation Time. We can break down the texture generation time of NITES into three parts: 1) successive subspace embedding (i.e., the forward embedding path), 2) core and successive subspace generations (i.e., the reverse generation path) and 3) the quilting process. The time required for each part is shown in Table III. Three parts demand 24.24, 108.03 and 8.08 seconds, respectively. To generate multiple texture images from the same exemplary texture, we need to run the first part once but the second and third parts multiple times (one run for one new synthesis). Thus, it is fair to focus on the last two parts only for single texture image generation, which is equal to 116 seconds. In contrast, the two benchmarking CNN methods do not have such a breakdown. They need to go through the whole pipeline to generate one new texture image.

TABLE III
THE TIME OF THREE PROCESSES IN OUR METHOD.

Process	Time (seconds)
Forward Embedding	24.24
Reverse Generation	108.03
Quilting	8.08

V. CONCLUSION AND FUTURE WORK

A non-parametric interpretable texture synthesis (NITES) method was proposed based on a new texture analysis and synthesis framework in this study. Texture can be analyzed and represented effectively using the multi-stage c/w Saab

(a) Synthesized 32x32 Patches

(b) Stitched using 32x32 Patches

(c) Synthesized 64x64 Patches

(d) Stitched using 64x64 Patches

Fig. 5. 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.

transforms that offer a sequence of joint spatial-spectral representations. The sample distribution in the core subspace was carefully studied, which allows us to build a core subspace generation model. Furthermore, a successive subspace generation model was developed to build a higher-dimensional subspace based on a lower-dimensional subspace. As a result, new texture samples can be generated by mimicking probabilities and/or conditional probabilities of source texture patches. Extensive experimental results were conducted to demonstrate the power of the proposed NITES method. It can generate visually pleasant texture images effectively, including some unseen patterns.

Future research should be extended in several directions. Controlling the growth of subspace dimensions in the generation process appears to be an important objective. 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 types of 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.

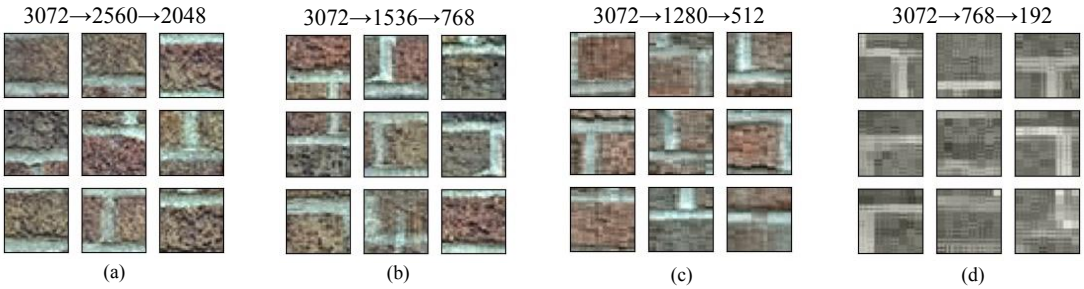

Fig. 6. Generated patches using different subspace settings of our non-parametric generation model. The numbers above the figure indicates the dimensions of S_0, S_1 and S_2, respectively.

ACKNOWLEDGMENT

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] J. S. De Bonet, "Multiresolution sampling procedure for analysis and synthesis of texture images," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 361–368, 1997.

[2] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1033–1038, IEEE, 1999.

[3] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 479–488, 2000.

[4] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 341–346, 2001.

[5] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics (ToG)*, vol. 20, no. 3, pp. 127–150, 2001.

[6] M. F. Cohen, J. Shade, S. Hiller, and O. Deussen, "Wang tiles for image and texture generation," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3, pp. 287–294, 2003.

[7] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: image and video synthesis using graph cuts," *ACM Transactions on Graphics (ToG)*, vol. 22, no. 3, pp. 277–286, 2003.

[8] Q. Wu and Y. Yu, "Feature matching and deformation for texture synthesis," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 364–367, 2004.

[9] V. Kwatra, I. Essa, A. Bobick, and N. Kwatra, "Texture optimization for example-based synthesis," in *ACM SIGGRAPH 2005 Papers*, pp. 795–802, 2005.

[10] D. J. Heeger and J. R. Bergen, "Pyramid-based texture analysis/synthesis," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 229–238, 1995.

[11] J. Portilla and E. P. Simoncelli, "A parametric texture model based on joint statistics of complex wavelet coefficients," *International journal of computer vision*, vol. 40, no. 1, pp. 49–70, 2000.

[12] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in neural information processing systems*, pp. 262–270, 2015.

[13] G. Liu, Y. Gousseau, and G.-S. Xia, "Texture synthesis through convolutional neural networks and spectrum constraints," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 3234–3239, IEEE, 2016.

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

[15] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Diversified texture synthesis with feed-forward networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3920–3928, 2017.

[16] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Universal style transfer via feature transforms," in *Advances in neural information processing systems*, pp. 386–396, 2017.

[17] I. Ustyuzhaninov, W. Brendel, L. A. Gatys, and M. Bethge, "What does it take to generate natural textures?," in *ICLR (Poster)*, 2017.

[18] W. Shi and Y. Qiao, "Fast texture synthesis via pseudo optimizer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5498–5507, 2020.

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

[20] C.-C. J. Kuo, "Understanding convolutional neural networks with a mathematical model," *Journal of Visual Communication and Image Representation*, vol. 41, pp. 406–413, 2016.

[21] C.-C. J. Kuo, "The cnn as a guided multilayer recos transform [lecture notes]," *IEEE signal processing magazine*, vol. 34, no. 3, pp. 81–89, 2017.

[22] C.-C. J. Kuo, M. Zhang, S. Li, J. Duan, and Y. Chen, "Interpretable convolutional neural networks via feedforward design," *Journal of Visual Communication and Image Representation*, 2019.

[23] Y. Chen and C.-C. J. Kuo, "Pixelhop: A successive subspace learning (ssl) method for object recognition," *Journal of Visual Communication and Image Representation*, p. 102749, 2020.

[24] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, pp. 2161–2168, Ieee, 2006.

[25] A. Hyvärinen, P. O. Hoyer, and E. Oja, "Sparse code shrinkage: Denoising by nonlinear maximum likelihood estimation," in *Advances in Neural Information Processing Systems*, pp. 473–479, 1999.

[26] M. S. Bartlett, J. R. Movellan, and T. J. Sejnowski, "Face recognition by independent component analysis," *IEEE Transactions on neural networks*, vol. 13, no. 6, pp. 1450–1464, 2002.

[27] N. Mitianoudis and T. Stathaki, "Pixel-based and region-based image fusion schemes using ica bases," *Information fusion*, vol. 8, no. 2, pp. 131–142, 2007.

[28] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

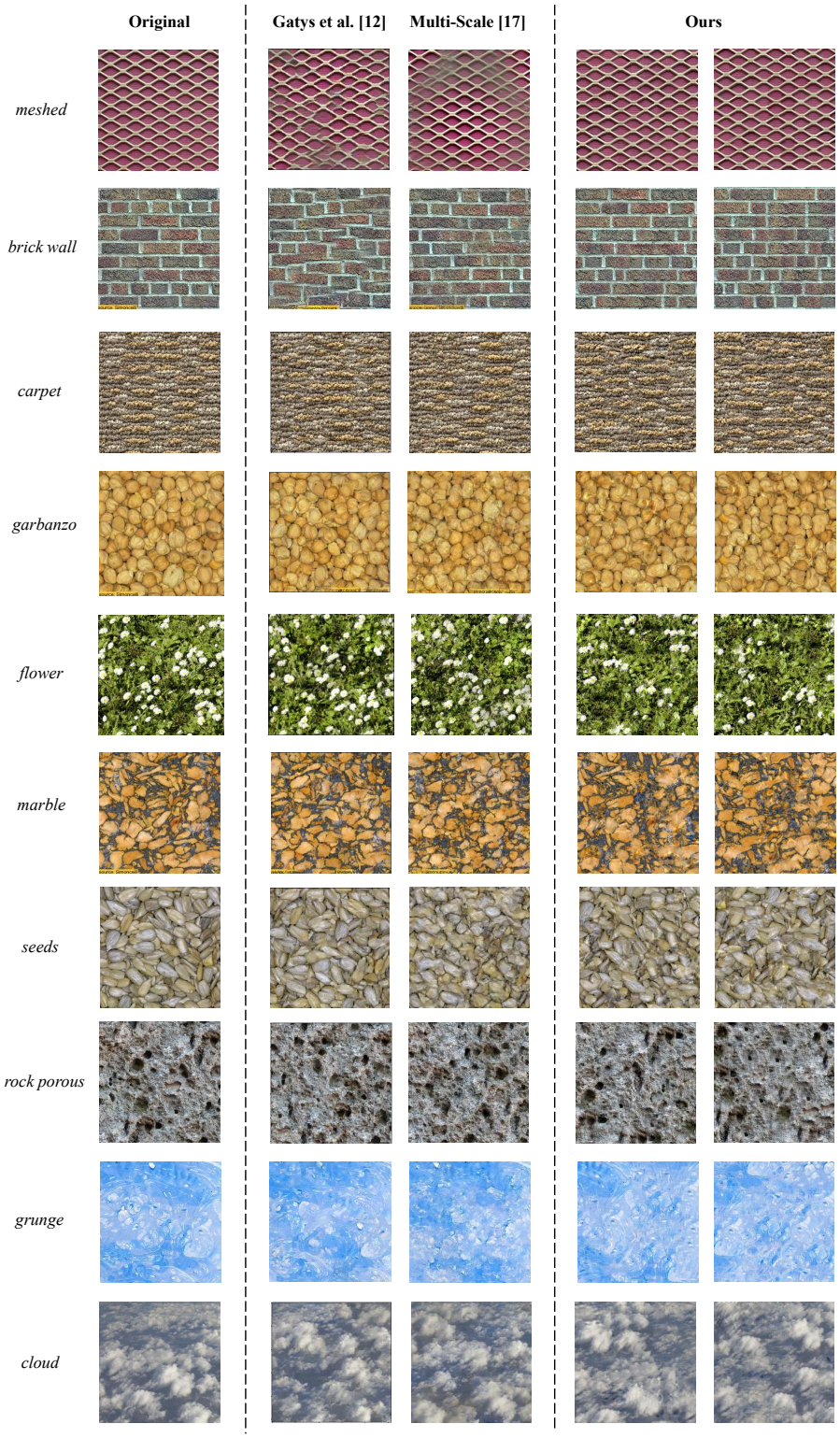

Fig. 7. Comparison of synthesized texture images using two CNN-based methods and NITES (from left to right): exemplary texture images, texture images generated by [12] texture images generated by [17] and two texture images generated by NITES.