Diversified Texture Synthesis with Feed-forward Networks

Yijun Li[1], Chen Fang[2], Jimei Yang[2], Zhaowen Wang[2], Xin Lu[2], and Ming-Hsuan Yang[1]
[1]University of California, Merced [2]Adobe Research
{yli62,mhyang}@ucmerced.edu {cfang,jimyang,zhawang,xinl}@adobe.com

Abstract

Recent progresses on deep discriminative and generative modeling have shown promising results on texture synthesis. However, existing feed-forward based methods trade off generality for efficiency, which suffer from many issues, such as shortage of generality (i.e., build one network per texture), lack of diversity (i.e., always produce visually identical output) and suboptimality (i.e., generate less satisfying visual effects). In this work, we focus on solving these issues for improved texture synthesis. We propose a deep generative feed-forward network which enables efficient synthesis of multiple textures within one single network and meaningful interpolation between them. Meanwhile, a suite of important techniques are introduced to achieve better convergence and diversity. With extensive experiments, we demonstrate the effectiveness of the proposed model and techniques for synthesizing a large number of textures and show its applications with the stylization.

1. Introduction

Impressive neural style transfer results by Gatys et al. [14] have recently regained great interests from computer vision, graphics and machine learning communities for the classic problem of texture synthesis [10, 9, 35]. Considering the expensive optimization process in [14], a few attempts have been made to develop feed-forward networks to efficiently synthesize a texture image or a stylized image [19, 32]. However, these methods often suffer from many issues, including shortage of generality (i.e., build one network per texture), lack of diversity (i.e., always produce visually identical output) and suboptimality (i.e., generate less satisfying visual effects).

In this paper, we propose a deep generative network for synthesizing diverse outputs of multiple textures in a single network. Our network architecture, inspired by [28, 7], takes a noise vector and a selection unit as input to generate texture images using up-convolutions. The selection unit is a one-hot vector where each bit represents a texture type and provides users with a control signal to switch between different types of textures to synthesize. More importantly, such a multi-texture synthesis network facilitates generating new textures by interpolating with the selection units. Meanwhile, the noise vector is intended to drive the network to generate diverse samples even from a single exemplar texture image.

However, learning such a network is a challenging task. First, different types of textures have quite different statistical characteristics, which are partially reflected by the varied magnitudes of texture losses (i.e., the Gram matrix-based losses introduced in [13, 14] to measure style similarity) across different feature layers. Second, the convergence rates for fitting different textures are inherently different due to their drastic visual difference and semantic gaps. As a result, the overall difficulty of learning such a network is determined by the variation among the textures and the complexity of individual textures. Third, the network often encounters the "explain-away" effect that the noise vector is marginalized out and thus fails to influence the network output. Specifically, the network is not able to generate diverse samples of a given texture, and it often means overfitting to a particular instance.

In this work, we propose a suite of effective techniques to help the network generate diverse outputs of higher quality. We first improve the Gram matrix loss by subtracting the feature mean, so that the newly designed loss is more stable in scale and adds stability to the learning. Second, in order to empower the network with the ability of generating diverse samples and further prevent overfitting, we aim to correlate the output samples with the input noise vector. Specifically, we introduce a diversity loss that penalizes the feature similarities of different samples in a mini-batch. Third, we show that a suitable training strategy is critical for the network to converge better and faster. Thus we devise an incremental learning algorithm that expose new training textures sequentially to the learner network. We start from learning to synthesize one texture and only incorporate next new unseen texture into the learning when the previous texture can be well generated. As such, the network gradually learns to synthesize new textures while retaining the ability to generate all previously seen textures.

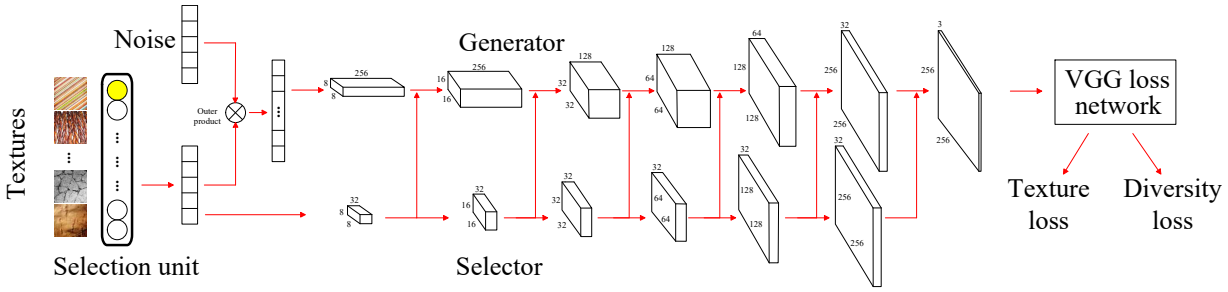


Figure 1. Architecture of the proposed multi-texture synthesis network. It consists of a generator and a selector network.

The contributions of this work are threefold:

- We propose a generative network to synthesize multiple textures in a user-controllable manner.
- A diversity loss is introduced to prevent the network from being trapped in a degraded solution and more importantly it allows the network to generate diverse texture samples.
- The incremental learning algorithm is demonstrated to be effective at training the network to synthesize results of better quality.

2. Related Work

Traditional synthesis models. Texture synthesis methods are broadly categorized as non-parametric or parametric. Parametric methods [17, 27] for texture synthesis aim to represent textures through proper statistical models, with the assumption that two images can be visually similar when certain image statistics match well [20]. The synthesis procedure starts from a random noise image and gradually coerces it to have the same relevant statistics as the given example. The statistical measurement is either based on marginal filter response histograms [3, 17] at different scales or more complicated joint responses [27]. However, exploiting proper image statistics is challenging for parametric models especially when synthesizing structured textures.

Alternatively, non-parametric models [10, 9, 22, 34] focus on growing a new image from an initial seed and regard the given texture example as a source pool to constantly sample similar pixels or patches. This is also the basis of earlier texture transfer algorithms [9, 23, 1, 18]. Despite its simplicity, these approaches can be slow and subject to non-uniform pattern distribution. More importantly, these methods aim at growing a perfect image instead of building rich models to understand textures.

Synthesis with neural nets. The success of deep CNNs in discriminative tasks [21, 29] has attracted much attention for image generation. Images can be reconstructed by inverting features [26, 6, 5], synthesized by matching features, or even generated from noise [16, 28, 4]. Synthesis with neural nets is essentially a parametric approach, where intermediate network outputs provide rich and effective image statistics. Gatys et al. [13] propose that two textures are perceptually similar if their features extracted by a pre-trained CNN-based classifier share similar statistics. Based on this, a noise map is gradually optimized to a desired output that matches the texture example in the CNN feature space.

Subsequent methods [19, 32] accelerate this optimization procedure by formulating the generation as learning a feed-forward network. These methods train a feed-forward network by minimizing the differences between statistics of the ground truth and the generated image. In particular, image statistics was measured by intermediate outputs of a pre-trained network. Further improvements are made by other methods that follow either optimization based [24, 11, 12, 15] or feed-forward based [25, 33] framework. However, these methods are limited by the unnecessary requirement of training one network per texture. Our framework also belongs to the feed-forward category but synthesizes diverse results for multiple textures in one single network.

A concurrent related method recently proposed by Dumoulin et al. [8] handles multi-style transfer in one network by specializing scaling and shifting parameters after normalization to each specific texture. Our work differs from [8] mainly in two aspects. First, we employ a different approach in representing textures. We represent textures as bits in a one-hot selection unit and as a continuous embedding vector within the network. Second, we propose diversity loss and incremental training scheme in order to achieve better convergence and output diverse results. Moreover, we demonstrate the effectiveness of our method on a much larger set of textures (e.g., 300) whereas [8] develops a network for 32 textures.

3. Proposed Algorithm

We show the network architecture of the proposed model in Figure 1. The texture synthesis network (bottom part)

has two inputs, a noise vector and a selection unit, while the upper part in blue dash line boxes are modules added for extending our model to style transfer. The noise vector is randomly sampled from a uniform distribution, and the selection unit is a one-hot vector, where each bit represents a texture in the given texture set. The network consists of two streams: the generator and the selector. The generator is responsible for synthesis and the selector is guiding the generator towards the target texture, conditioned on the activated bit in the selection unit.

Given M target textures, we first map the M dimensional selection unit to a lower dimensional selection embedding. Then we compute the outer product of the noise vector and selection embedding. After the outer-product operation, we reshape the result as a bunch of 1×1 maps and then use the *SpatialFullConvolution* layer to convolve them to a larger spatial representation with numerous feature maps. After a series of nearest-neighbor upsampling followed by convolutional operations, this representation is converted to a $256 \times 256 \times 3$ pixel image. On the selector stream, it starts with a spatial projection of the embedding, which is then consecutively upsampled to be a series of feature maps which are concatenated with those feature maps in the generator, in order to offer guidance (from coarse to fine) at each scale.

Finally, the output of the generator is fed into a fixed pre-trained loss network to match the correlation statistics of the target texture using the visual features extracted at different layers of the loss network. We use the 19-layer VGG [31] model as the loss network.

3.1. Loss function

We employ two loss functions, i.e., texture loss and diversity loss. The texture loss is computed between the synthesized result and the given texture to ensure that these two images share similar statistics and are perceptually similar. The diversity loss is computed between outputs of the same texture (i.e., same input at selection unit) driven by different input noise vectors. The goal is to prevent the generator network from being trapped to a single degraded solution and to encourage the model to generate diversified results with large variations.

Texture loss. Similar to existing methods [13, 19, 32], the texture loss is based on the Gram matrix (G) difference of the feature maps in different layers of the loss network as

$$L_{texture} = \|G_{gt} - G_{output}\|_1 \ , \quad G_{ij} = \sum_k F_{ik} F_{jk} \ , \quad (1)$$

where each entry G_{ij} in the Gram matrix is defined as the inner product of F_{ik} and F_{jk}, which are vectorized activations of the ith (and jth) filter at position k in the current layer of the loss network. We use the activations at the

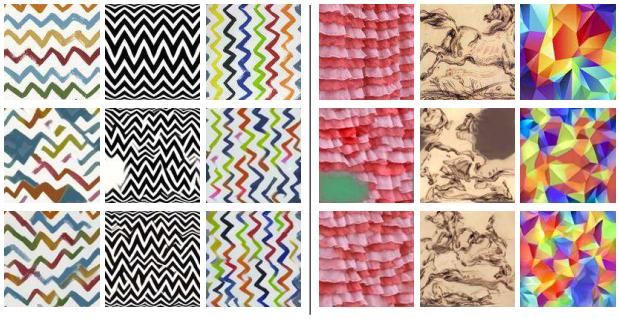


Figure 2. Comparisons between using G and mean subtracted $\overline{G}$. We show results of two 3-texture networks (left and right). Top: original textures, Middle: synthesized results using G based texture loss, Bottom: synthesized results using $\overline{G}$ based texture loss.

$conv1\_1$, $conv2\_1$, $conv3\_1$, $conv4\_1$ and $conv5\_1$ layer of the VGG model.

The Gram matrix based texture loss has been shown demonstrated to effective for single texture synthesis. However, for the purpose of multiple textures synthesis, we empirically find that the original texture loss (defined as Eq. 1) poses difficulty for the network to distinguish between textures and thus fails to synthesize them well. In the middle row of Figure 2, we show a few examples of textures generated using the original texture loss in two experiments of synthesizing 3 textures with one network. Note the obvious artifacts and color mixing problems in the synthesized results. We attribute this problem to the large discrepancy in scale of the Gram matrices of different textures.

Motivated by this observation, we modify the original Gram matrix computation by subtracting the mean before calculating the inner product between two activations:

$$\overline{G}_{ij} = \sum_k (F_{ik} - \overline{F})(F_{jk} - \overline{F}) \ , \quad (2)$$

where $\overline{F}$ is defined as the mean of all activations in the current layer of the loss network and the rest of terms remain the same with those in the definition of the Gram matrix (1). Without re-centering the activations, we notice that during training the values of losses and gradients from different textures vary drastically, which suggests that the network is biased to learn the scale of Gram matrix, i.e., $\overline{F}$, instead of discriminating between them. In the bottom row of Figure 2, we provide the same textures synthesized with the re-centered Gram matrix, which clearly shows improvements compared to the middle row.

Diversity loss. As mentioned above, one of the issues with existing feed-forward methods is being easily trapped to a degraded solution where it always outputs that are visually identical (sometimes with less satisfying repetitive patterns) [30]. When trained only with the texture loss, the pro-

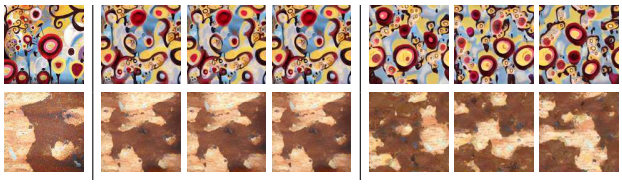


Figure 3. Comparisons between without and with the diversity loss. Left: original textures, Middle: outputs (w/o diversity) under there different noise inputs, Right: outputs (w/ diversity) under the same set of different noise inputs.

posed network has the same issue. We show several examples in the middle panel of Figure 3. The results under different noise input are nearly identical with subtle and unnoticeable difference in pixel values. This is expected because the texture loss is designed to ensure all synthesized results to have the similar style with the given texture, but does not enforce diversity among outputs. In other words, each synthesized result is not correlated with the input noise.

In order to correlate the output with input noise, we design a diversity loss which explicitly measures the variation in visual appearance between the generated results under the same texture but different input noise. Assume that there are N input samples in a batch at each feed-forward pass, the generator will then emit N outputs $\{P_1, P_2, ..., P_N\}$. Our diversity loss measures the visual difference between any pair of outputs P_i and P_j using visual features. Let $\{Q_1, Q_2, ..., Q_N\}$ be a random reordering of $\{P_1, P_2, ..., P_N\}$, satisfying that $P_i \neq Q_i$. In order to encourage the diversity in a higher level rather than lower level such as pixel shift, the diversity loss is computed between feature maps at the $conv4\_2$ layer of the loss network Φ as follows:

$$L_{diversity} = \frac{1}{N}\sum_{i=1}^{N} \|\Phi(P_i) - \Phi(Q_i)\|_1, \qquad (3)$$

The results generated by our method with this diversity loss are shown in the right panel of Figure 3. While being perceptually similar, the results from our method contain rich variations. Similar observations are found in [30] which also encourages diversity in generative model training by enlarging the distance among all samples within a batch on intermediate layers features, while our method achieves this with the diversity loss.

The final loss function of our model is a combination of the texture loss and the diversity loss as shown in (4). As the goal is to minimize the texture loss and maximize the diversity loss, we use the coefficients $\alpha = 1, \beta = -1$ in our experiments.

$$L = \alpha L_{texture} + \beta L_{diversity}, \qquad (4)$$

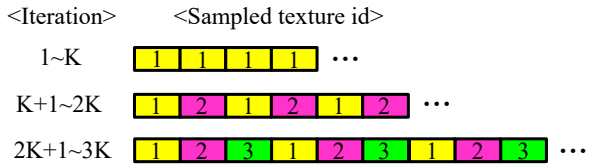


Figure 4. Incremental training strategy. Each block represents an iteration and the number in it is the sampled texture id for this iteration (also the bit we set as 1 in the selection unit). We use $K = 1000$ in the experiments.

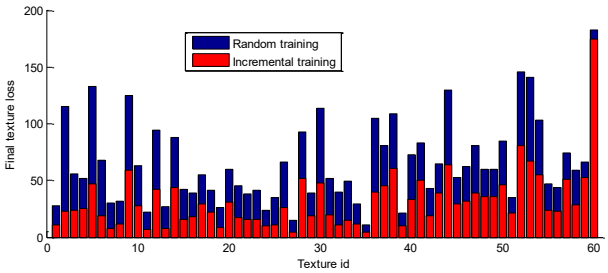


Figure 5. Comparisons of the final texture loss when converged between the random and incremental training on 60-texture synthesis.

3.2. Incremental training

We discuss the training process for the proposed network with focus on how to sample a target texture among a set of predefined texture set. More specifically, we address the issue whether samples should randomly selected or in certain order in order to generate diversified textures. Once a target texture is selected, we set the corresponding bit in the selection unit as 1 and the corresponding texture is used to compute the texture loss.

Empirically we find that the random sampling strategy typically yields inferior results and it becomes difficult to further push down texture losses for all texture images after certain number of iterations. We train a 60-texture network as an example and show the converged results (10 out of 60) with random sampling in the middle row of Figure 6. The artifacts are clearly visible. Major patterns of each texture are captured, however the geometry is not well preserved (e.g., hexagons in the first texture), and colors are not well matched (i.e., mixing with colors from other textures).

We attribute this issue to the constant drastic change in the learning objective caused by random sampling within a diverse set of target textures. In other words, although the network gains improvement toward a sampled texture at every iteration, the improvement is likely to be overwhelmed in the following iterations, where different textures are optimized. As a consequence, the learning becomes less effective and eventually gets stuck to a bad local optimum.

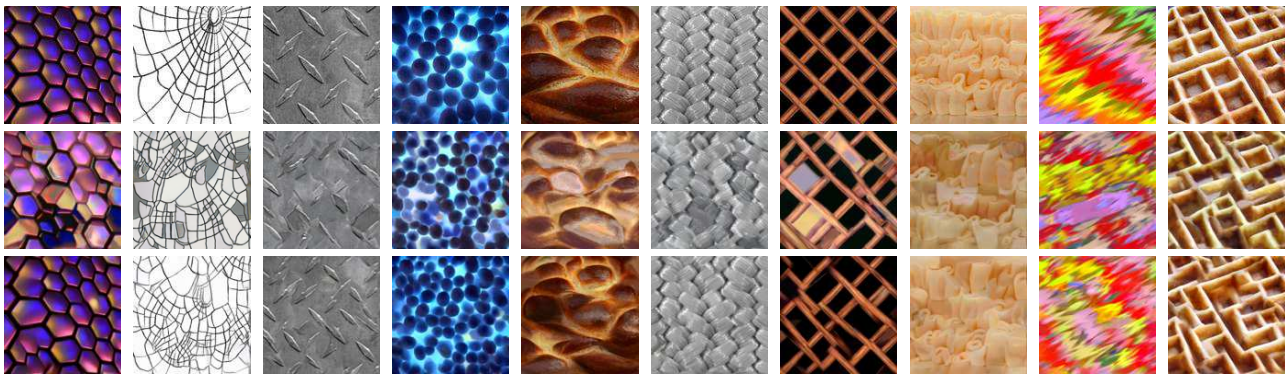
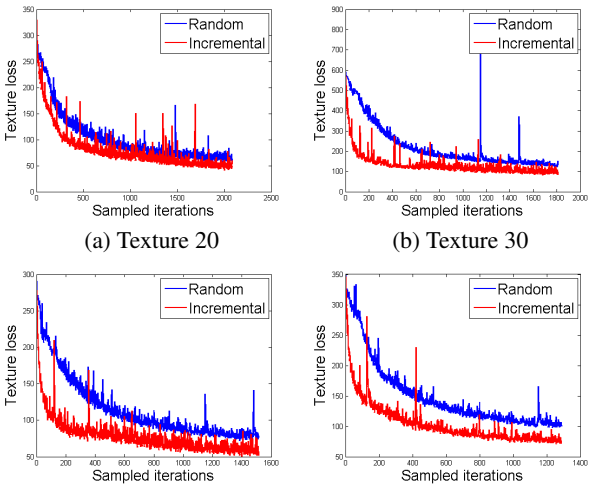


Figure 6. Comparisons between random training and incremental training. Top: original textures, Middle: synthesis with random training, Bottom: synthesis with incremental training. The model is handling 60 textures in Figure 5 and we show the synthesized results of 10 textures here.

Therefore we propose an incremental training strategy to help the learning to be more effective. Overall our incremental training strategy can be seen as a form of curriculum learning. There are two aspects in training the proposed network incrementally. First, we do not teach the network to learn new tasks *before* existing the network learns existing ones well. That is, we start from learning one texture and gradually introduce new textures when the network can synthesize previous textures well. Second, we ensure that the network does not *forget* what is already learned. Namely, we make sure that all the target textures fed to the network so far will still be sampled in future iterations, so that the network "remembers" how to synthesize them.

Specifically, in the first K iterations, we keep setting the 1st bit of the selection unit as 1 to let the network fully focus on synthesizing Texture 1. In the next K iterations, Texture 2 is involved and we sample the bit from 1 to 2 in turn. We repeat the same process to the other textures. We illustrate this procedure in Figure 4. After all the textures are introduced to the network, we switch to the random sampling strategy until the training process converges. In Figure 5 and 6, we show the comparison of both the final texture loss and synthesized visual results between the random and incremental training strategies in the 60-texture network experiment. Clearly the incremental training scheme leads to better convergence quantitatively and qualitatively.

Interestingly, we observe that the network learns new textures faster as it sees more textures in later training stages. To demonstrate that, we record the texture losses for each texture when it is sampled and show four examples in Figure 7 when training the 60-texture network. Take Texture 20 (Figure 7(a)) as an example, the network learned with incremental training quickly reaches lower losses compare to the one with random sampling strategy. We hypothesize that the network benefits from the shared knowledge learned from Texture 1-19. This conjecture is supported by later introduced textures (Figure 7(b-d)) where incremental training gets relatively faster at convergence as it learns more textures.



(a) Texture 20 (b) Texture 30

(c) Texture 40 (d) Texture 50

Figure 7. Comparisons between the random and incremental training on a single texture during a 60-texture network training. Note that some sudden drastic changes on the loss curve appear when a new texture is firstly involved which causes a short-term oscillation in the network.

4. Experimental Results

In this section, we present extensive experimental results to demonstrate the effectiveness of our algorithm. We experiment with synthesizing a large number of textures using a single network and then show that our model is able to generate diverse outputs and create new textures by linear interpolation.

4.1. Multi-texture synthesis

In addition to the 60-texture network trained for illustration purpose in Section 3, we experiment with a larger

Figure 8. Synthesized results of a 300-texture network. In each panel, Left: original texture, Right: synthesized result. We show results of 20 (out of 300) textures as examples here. For each texture, we only show one synthesized result.

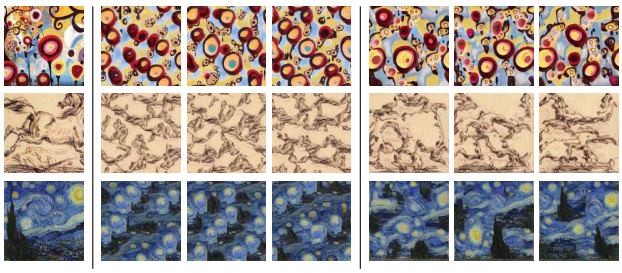


Figure 9. Comparisons of diverse synthesized results between the TextureNet [32] (middle) and our model (right).

300-texture network to further validate the robustness and scalability of our model. We map the 300-dimensional selection unit to a 128-dimensional embedding and use a 5-dimensional noise vector. The network is trained with both texture and diversity loss under the incremental training strategy. Texture images used in our experiments are from the Describable Textures Dataset (DTD) [2]. Figure 8 shows the synthesized results of 20 textures.

4.2. Diversity

By sampling different noise in the noise vector, our network can generate diverse synthesized results for each texture. Existing single-texture networks [32] can also generate diversity to a certain extent. However, the diversity is still limited because their network is trained with the texture loss only. The diversity in [32] is mainly enforced by injecting multiple noise maps at different scales (from 8×8 to 256×256). Without explicit constraints to push diversity, such a huge variation will be reduced or absorbed by the network, which still leads to limited diversity in outputs. We compare the diverse outputs of our model and [32] in Figure 9. Note that the common diagonal layout is shared across different results of [32], which causes unsatisfying visual experience. In contrast, our method achieves diversity in a more natural and flexible manner. With the diversity loss, our model enables diverse outputs with low dimensional noise input, which gives us the ability to generate continuous transition between those outputs.

4.3. Interpolation

Equipped with a selection unit and a learned M-texture network, we can interpolate between bits at test time to create *new* textures or generate smooth transitions between textures. We show two examples of interpolation with our previously trained 300-texture network in Figure 10. For example in the top row of Figure 10, we start from Texture 20 and drive the synthesis towards Texture 19. This is carried out by gradually decreasing the weight in the 20th bit and increasing the weight in the 19th bit with the rest bits all set as zero. Such a smooth transition indicates that our generation can go along a continuous space.

The method in [14] is also able to synthesize the interpolated result of two textures. In [14], if we denote G_1 and G_2 as the Gram matrix of two textures, the interpolated texture is generated by matching some intermediate Gram matrix $a \times G_1 + (1 - a) \times G_2$ through optimization (e.g., a=0.5). We show the interpolation comparison between [14] and our method in Figure 11. It is observed that the results by [14] are simply overlaid by two textures while our method generates new textural effects.

4.4. Extension to multi-style transfer

We extend the idea of multi-texture synthesis to the multi-style transfer for image stylization. Given a style image and a content image, image stylization aims at synthesizing an image that preserves the global content while

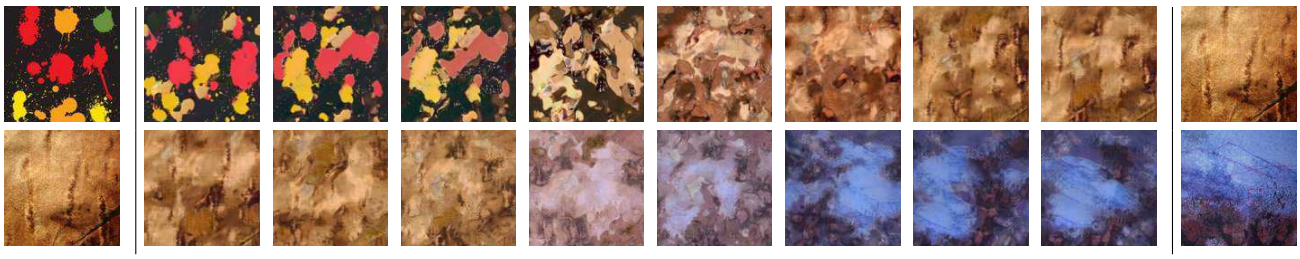


Figure 10. Texture interpolation (or transition) with the 300-texture network. Top: Texture 20 to Texture 19, Bottom: Texture 19 to Texture 12. Images in the leftmost and rightmost are original textures.

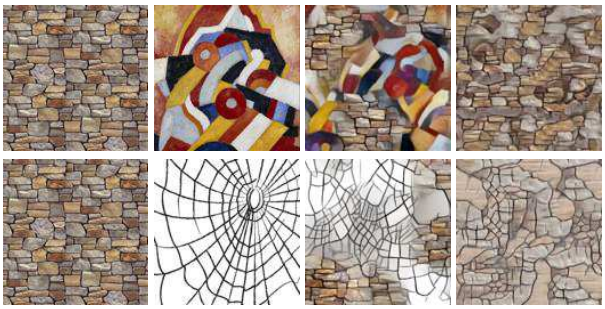


Figure 11. Interpolation comparison between [14] and our method.

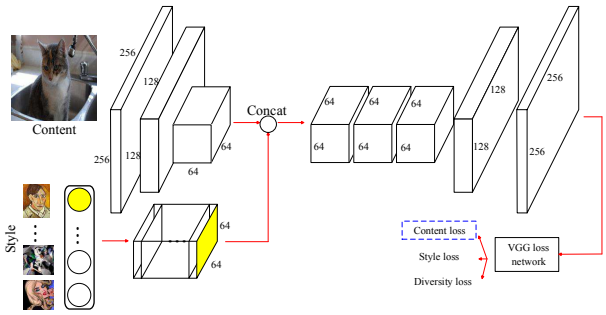


Figure 12. Architecture of the multi-style transfer network.

transferring the colors and local structures from the style image. For presentation clarity, we will use the term *style* instead of the *texture*.

The network architecture is shown in Figure 12. We use an autoencoder network similar to [19] and incorporate our idea of introducing a selection unit to handle the transferring of different styles. More specifically, for each bit in the selection unit, we generate a corresponding noise map (e.g., from the uniform distribution) and concatenate these maps with the encoded features from the content, which are then decoded to the transferred result. When one style is selected, only the noise map that corresponds to it is randomly initialized while other noise maps are set to zero. The content loss is computed as the feature differences between the transferred result and the content at the $conv4\_2$ layer of the VGG model as in [14]. The style loss and diversity loss are defined in the same way as those in texture synthesis. We train a 1000-style transfer network and show the transferred results in Figure 13. Note that as our multi-transfer model is fully convolutional, it is able to handle the content image of arbitrary sizes.

In addition, we compare our multi-style transfer model with existing methods in Figure 15. We adjust the style weight such that all methods have similar transferring effects. It clearly shows that our multi-style transfer model achieves improved or comparable results.

With the selection unit, we interpolate between styles by adjusting the weights of different bits in the selection unit and generate the style interpolation (or transition) results in Figure 14. Specifically, if we denote s_1 and s_2 as the bit value of two styles and N_1 and N_2 as the corresponding noise map, the interpolation is generated by feeding $s_1 \times N_1 + s_2 \times N_2$ as the selection input.

Diverse transfer results are shown in Figure 16. Different from the case of texture synthesis, the global structure of images is constrained by the demand of preserving content. Therefore the diversity is exhibited at local visual structures. Notice the slight but meaningful differences among these outputs.

5. Discussion

Selector network. In our model, we introduce a selector network (Figure 1) in order to drive the network towards synthesizing the desired texture only. The selector injects guidance to the generator at every upsampling scale and helps the model distinguish different textures better during the synthesis. We show an example of training a 60-texture network w/o and w/ the selector network in Figure 17. We present the loss curves of two textures as examples, which clearly shows that with the selector, the network training achieves better convergence.

Embedding. Starting with a one-hot selection unit to represent each texture per bit, we first map it to a lower dimensional embedding and aim at learning a better representation of given textures. In our presented 60-texture and 300-

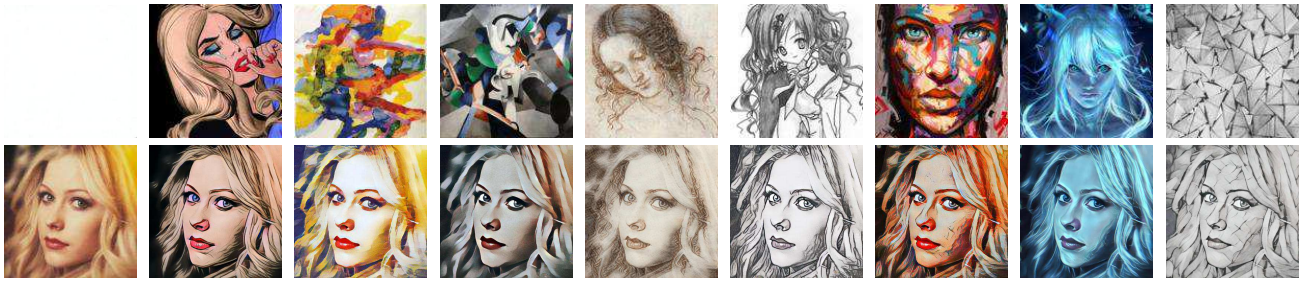


Figure 13. Transferred results of a 1000-style transfer network. We show results of 8 (out of 1000) styles as examples. Top: style images, Leftmost: content image, Bottom: transferred results.

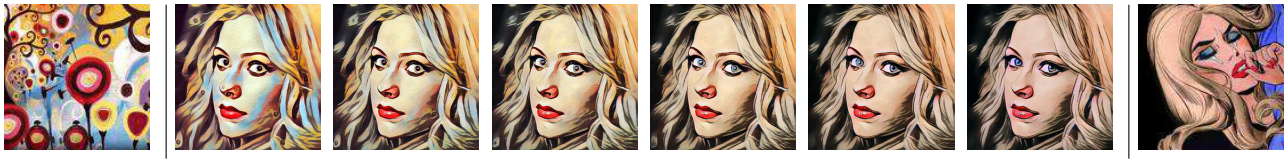


Figure 14. Style interpolation (or transition) with the multi-style transfer network. Images in the leftmost and rightmost are original styles. The content image used in the middle is from Figure 13.

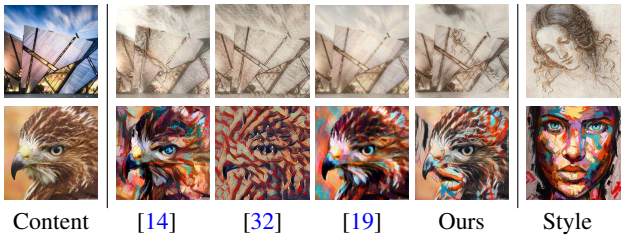
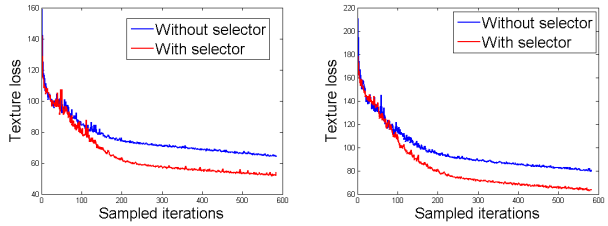


Figure 15. Comparison of style transfer results between existing methods and ours.

Figure 16. Diverse transferred results of our multi-style transfer network. Left: content images, Middle: diverse transferred results, Right: style images. Note the difference in the *beak* and *sky*.

texture model, we map the 60-D and 300-D selection unit to a 32-D and 128-D embedding respectively. Our results show that the embedding can still distinguish different textures for synthesis, which indicates that the original one hot representation is redundant. In addition, as we have shown that new textures can be created through interpolation in a feed-forward way, it poses an open question that whether we can find the coefficients in a backward way to represent a given new texture as a weighted combination of learned embeddings. We leave this as a future direction to pursue.



(a) Texture 20 (b) Texture 40

Figure 17. Comparisons of the loss curve between our framework without and with the selector network.

6. Conclusion

In this paper, we focus on synthesizing multiple textures in one single network. Given M textures, we propose a deep generative feed-forward network which can synthesize diverse results for each texture. In order to train a deep network for multi-texture synthesis, we introduce the diversity loss and propose an incremental leaning scheme. The diversity loss helps the network to synthesize diverse textures with the same input, and the incremental learning scheme helps effective and efficient training process. Experimental results demonstrate the effectiveness of model, which generates comparable results compared to existing single-texture networks but greatly reduces the model size. We also show the extension of our multi-texture synthesis model to multi-style transfer for image stylization.

Acknowledgment. This work is supported in part by the NSF CAREER Grant #1149783, gifts from Adobe and Nvidia.

References

[1] N. Ashikhmin. Fast texture transfer. *IEEE Computer Graphics and Applications*, 23(4):38–43, 2003. 2

[2] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *CVPR*, 2014. 6

[3] J. S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH*, 1997. 2

[4] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a Laplacian pyramid of adversarial networks. In *NIPS*, 2015. 2

[5] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. In *NIPS*, 2016. 2

[6] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *CVPR*, 2016. 2

[7] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015. 1

[8] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *arXiv preprint arXiv:1610.07629*, 2016. 2

[9] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH*, 2001. 1, 2

[10] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999. 1, 2

[11] O. Frigo, N. Sabater, J. Delon, and P. Hellier. Split and match: Example-based adaptive patch sampling for unsupervised style transfer. In *CVPR*, 2016. 2

[12] L. A. Gatys, M. Bethge, A. Hertzmann, and E. Shechtman. Preserving color in neural artistic style transfer. *arXiv preprint arXiv:1606.05897*, 2016. 2

[13] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. In *NIPS*, 2015. 1, 2, 3

[14] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016. 1, 6, 7, 8

[15] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman. Controlling perceptual factors in neural style transfer. *arXiv preprint arXiv:1611.07865*, 2016. 2

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 2

[17] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH*, 1995. 2

[18] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *SIGGRAPH*, 2001. 2

[19] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 1, 2, 3, 7, 8

[20] B. Julesz. Visual pattern discrimination. *IRE transactions on Information Theory*, 8(2):84–92, 1962. 2

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 2

[22] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: image and video synthesis using graph cuts. In *SIGGRAPH*, 2003. 2

[23] H. Lee, S. Seo, S. Ryoo, and K. Yoon. Directional texture transfer. In *NPAR*, 2010. 2

[24] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *CVPR*, 2016. 2

[25] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *ECCV*, 2016. 2

[26] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015. 2

[27] J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *IJCV*, 40(1):49–70, 2000. 2

[28] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 1, 2

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 2

[30] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. In *NIPS*, 2016. 3, 4

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3

[32] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016. 1, 2, 3, 6, 8

[33] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 2

[34] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH*, 2000. 2

[35] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH*, 2001. 1