Fast Texture Synthesis via Pseudo Optimizer

Wu Shi Yu Qiao
ShenZhen Key Lab of Computer Vision and Pattern Recognition,
SIAT-SenseTime Joint Lab, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
SIAT Branch, Shenzhen Institute of Artificial Intelligence and Robotics for Society
wu.shi@siat.ac.cn yu.qiao@siat.ac.cn

Abstract

Texture synthesis using deep neural networks can generate high quality and diversified textures. However, it usually requires a heavy optimization process. The following works accelerate the process by using feed-forward networks, but at the cost of scalability. diversity or quality. We propose a new efficient method that aims to simulate the optimization process while retains most of the properties. Our method takes a noise image and the gradients from a descriptor network as inputs, and synthesis a refined image with respect to the target image. The proposed method can synthesize images with better quality and diversity than the other fast synthesis methods do. Moreover, our method trained on a large scale dataset can generalize to synthesize unseen textures.

1. Introduction

The pioneering work of Gatys *et al*. [5] can generate high-quality and diversified texture images given an example image. Their key idea is to match the statistics of the synthesized image with the reference image by using an iterative optimization process. The reference statistics are extracted by a descriptive network that is trained on recognition tasks. This process requires numerous steps to modify the synthesized image based on the gradient information from the descriptive network until it is close to the reference statistics enough. It typically takes several minutes to synthesize an image of a moderate size even with a modern GPU.

Several following works [21, 13, 10, 14] have been proposed to accelerate the inference process, but at the cost of scalability, diversity or quality. [21] trains a feed-forward network to synthesize a single target texture by imposing the texture loss [5] on the synthesized image. The main shortcoming is that the trained network cannot generate new textures except for the one used in training. [13] designs a deep neural network with conditional labels to synthesize multiple textures in a user-controlled manner. This method greatly extends the scalability of the synthesis network. However, this method cannot generate novel textures in an online manner like [5]. [10] proposes to use Adaptive Instance Normalization to do fast arbitrary style transfer. In this method, the activations of the input image are normalized and then scaled and shifted by the statistics of the reference image. Thus, the reference image can be inserted at the inference time adaptively. It is shown to generalize well in the task of style transfer. WCT [14] further extends the linear transformation to a combination of whitening and coloring transforms. This method computes the covariance matrices from the features and uses the SVD decomposition to compute the transform matrix. This is not suitable for computing on GPU. We notice that the common idea of their methods is to integrate the information to change from the input image to the reference image with the transform network.

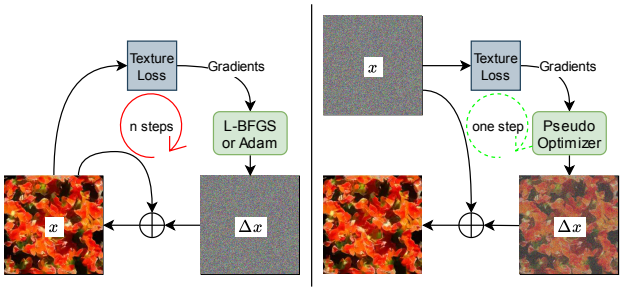

Figure 1. We propose a feed-forward framework, *Pseudo Optimizer*, to simulate the optimization process of [5] in one step.

We incorporate the idea of fast feed-forward methods into the optimization-based method and propose a novel fast texture synthesis framework named *Pseudo Optimizer* (PO). In our framework, the optimization process of [5] is reduced to a prediction problem by training a feed-forward network to map the gradient information to the optimal solution (Figure 3). We make a careful investigation into the

computation path of the optimization process and conclude that the process can be implemented by a feed-forward network and some arithmetic operations depending on the choice of the optimizer. Unfolding the iterative optimization algorithms into feed-forward neural networks has been employed in the area of compressive sensing [7, 24]. As neural networks are known to be good universal approximators [20], we replace the backward part of the optimization process by a learnable network and train the network to approximate the iterative optimization process. Our method has the following advantages:

- **Efficient**. The numerous iterative optimization steps are replaced by a single forward pass of the proposed network. The inference time is reduced from several minutes to < 0.1 second per image. The network is fully convolutional and can generate images with arbitrary size after training.

- **Adaptive**. Our method is designed to be self-supervised and can learn from large-scale datasets. The target texture is integrated in an online manner like Gatys *et al.*'s method during inference phase.

- **Diverse**. An incremental learning mechanism is designed to make it difficult for the model to memorize each target images and thus the model produces diversified results. We further propose a progressive architecture to encourage the locality of synthesized results.

We conduct extensive experiments to validate the effectiveness of our framework. Comparing with the slow optimization-based method, the proposed network can generate visually pleasing results in near real-time. The qualitative and quantitive results show that our method outperforms the other fast texture synthesis methods in the aspects of quality, diversity, and scalability. The code is available at https://github.com/swift-n-brutal/syntex.

2. Related Work

Traditional methods Texture synthesis is defined as a problem of sampling from a probability distribution in many traditional methods. Heeger and Bergen [9] refine a random noise image to match the histograms of filter response in an image pyramid. Simoncelli and Portilla [17] use the first and second order statistics of wavelet coefficients to model textures. The major drawback of these methods is the limited expressive power. Zhu *et al*. [27] build a texture model based on a Markov Random Field over the response of filters and synthesize textures using Gibbs sampling. Efros and Leung [4] propose a sequential model by synthesizing one pixel at a time based on synthesized pixels and a Markov Random Field model. [3] improves the previous method via a process called image quilting. While these methods can produce good results, their inference time is notoriously long due to the difficulty of convergence or the sequential synthesis procedure.

Optimization-based methods Recently Gatys *et al*. [5] propose a successful texture synthesis method by matching the statistics of the synthesized image with the reference image. Starting from a white noise image, they gradually refine the image using an iterative optimization method until it matches with the reference statistic closely enough. Following the same idea, [1] incorporates long-range consistency statistics to the objective function and generates textures with certain spatial structures. [19] matches the statistics across scales of a Gaussian pyramid to synthesize superior high-resolution textures. [23] uses histogram losses to synthesize texture and makes the optimization process more stable. citesendik2017deep introduces a structural energy to capture the self-similarities and regularities. All of these methods have a common shortcoming that it requires a heavy optimization process which is both time and memory consuming.

Efficient methods Several following methods [21, 13, 10, 14, 25] aim to accelerate the inference process of [5]. [21] imposes the texture loss [5] on the output of a feed-forward network. The network is trained to map from a set of noise images of different sizes to the texture image. [25] proposes an adversarial expansion approach to synthesize a non-stationary texture image. The drawback is that the trained network can only synthesize textures similar to the one used during training. [13] designs a deep neural network with conditional labels to synthesize multiple textures in a user-controlled manner. They propose the selection unit and the incremental learning algorithm to greatly extend the scalability of the synthesis network. Moreover, they introduce a diversity loss to prevent the network from mode-collapse [6]. However, their method cannot generate novel textures outside of the training dataset. In the area of style transfer which is closely related to texture synthesis, [10] proposes to use Adaptive Instance Normalization to do fast arbitrary style transfer. It inspires a lot of adaptive methods for style and domain transfer [11, 15]. WCT [14] further extends the linear transformation to a combination of whitening and coloring transforms on the covariance matrices. The common idea of their methods is to integrate the information to change from the input image to the reference image with the transform network. To make our framework adaptive as well, we employ the per-layer gradients from the texture loss as the information for transform.

Unfolding optimization algorithm Unfolding iterative optimization algorithms into feed-forward neural networks

has been applied in the areas of compressive sensing [7, 24], and image processing [16]. An advantage of these network-based methods is that they do not require an iterative process. This effectively reduces the time needed comparing with their optimization-based counterparts. As neural networks are known to be good universal approximators [20], we replace the backward part of Gatys *et al.*'s method by a learnable network and train the network to approximate the iterative optimization process.

3. Method

Texture synthesis aims to infer a generative model from an example texture, which can then synthesize new images similar to the given texture. The optimization-based method can generate high-quality and diverse images but requires a long inference time. We propose to reduce the optimization process into a prediction problem by training a feed-forward network mapping the gradients of the objective function to the optimal solution. In this way, the optimization can be done in a single forward pass. The logical flow is shown as follows. We first recap the optimization-based method using CNN in Sec. 3.1, and then analyze the feasibility of unfolding the optimization loop into a feed-forward network in Sec. 3.2. In Sec. 3.3, we delicately design a new feed-forward network named Pseudo Optimizer (PO) to simulate the optimization process. The network takes gradient information from the texture loss for a *single* target image as input and predicts the modification direction to the input image. In Sec. 3.4, we introduce an adaptive extension (AdaPO) to the PO network by simply training it on a set of *different* target images. We explain the reasonableness of "adaptive" by illustrating its relations to AdaIN and WCT. In Sec. 3.5, we propose a progressive model (ProPO) consisting of multi-stage refinements, which is a serial of AdaPO networks. Each sub-network has *independent parameters* and *different objectives*. The progressive architecture improves the quality and diversity of results.

3.1. Texture Synthesis using CNN

The authors of [5] reduce texture synthesis to the problem of sampling from the set of images that match the spatial summary statistics of the example texture image. They use the VGG19 network [18], a convolutional neural network trained on object classification, to extract a set of powerful descriptive feature activations: $\{F^{(l)} \in \mathbb{R}^{N^{(l)} \times C^{(l)}}\}_{l=1}^{L}$, where (l) is the index of layer, $N^{(l)}$ is the spatial dimension and $C^{(l)}$ is the number of channels in layer (l). The summary statistics are defined by the correlations, *i.e.* the Gram matrix[1] $G^{(l)} \in \mathbb{R}^{C^{(l)} \times C^{(l)}}$, between

[1]The notation is different from the one in the original paper. Here we use the normalized Gram matrix to counteract the change of image size.

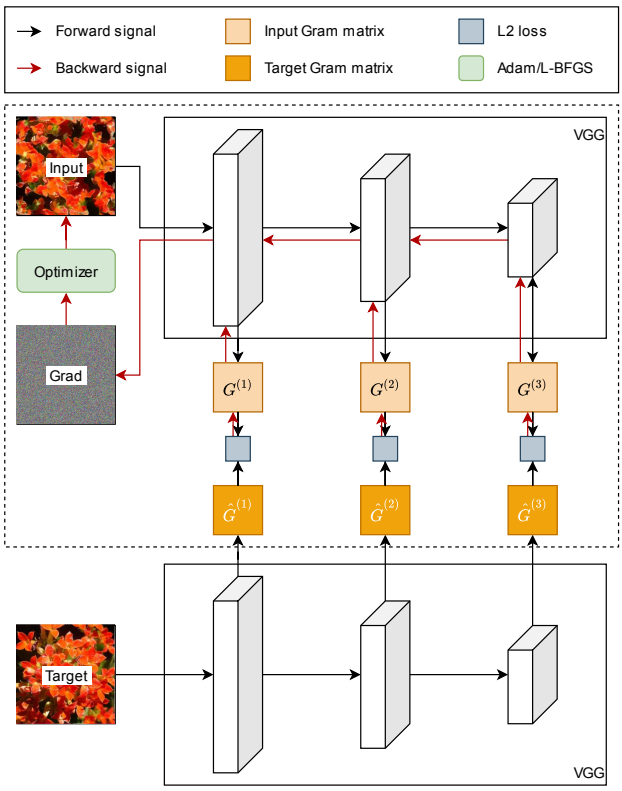

Figure 2. Texture synthesis using CNN [5]. The architecture and parameters are mainly designed for forward usage, and thus, may not be suitable for efficient texture synthesis. In Section 3.3, we replace the backward signals (red arrows) by a learnable network to synthesize in one step.

the responses of different features:

$$G_{ij}^{(l)} = \frac{1}{N^{(l)}} \sum_{k \in [N^{(l)}]} F_{k,i}^{(l)} F_{k,j}^{(l)}, \quad i, j \in [C^{(l)}]. \tag{1}$$

To synthesize a new texture given the example image $\tilde{x}$, they pose the generation process as an optimization problem by solving

$$\arg\min_{x \in \mathcal{X}} \mathcal{L}_{tex}(x, \tilde{x}; [L]) = \sum_{l=1}^{L} \left\| G^{(l)}(x) - G^{(l)}(\tilde{x}) \right\|_2^2, \tag{2}$$

where $\mathcal{X}$ is the image space and $[L]$ denotes the set of layers involved in the calculation. The objective function $\mathcal{L}_{tex}$ is usually named as texture loss in the related works. In practice, local optimization methods (*e.g.*, Adam [12] and L-BFGS [26]) are used to refine the image being generated. Specifically, the initial image is sampled from a noise distribution $x \sim Z$ and the optimizer iteratively refine the image x based on the current (and/or the history of) gradient $\partial \mathcal{L}_{tex} / \partial x$. The optimization step is often repeated hundreds of times to derive a high-quality texture image.

Typically it takes more than one minutes to synthesize a 256x256 image, which is not acceptable for fast texture synthesis applications.

3.2. Unfolding Optimization Loop

The idea of unfolding iterative optimization algorithms into feed-forward neural networks has been applied in the area of compressive sensing [7, 24]. In this subsection, we justify conceptually the feasibility of unfolding the optimization process of [5] using common components in neural networks. For synthesizing a single texture, the target Gram matrices are computed once and fixed during optimization. The effective optimization loop is surrounded by the dashed box in Figure 2. The forward signal is passed through the adjusted VGG network [5] to compute the texture loss given in Eq. 2. The network consists of several convolutional layers followed by ReLU layers, and occasionally average pooling layers for downsampling. The backward signal (red arrows) can be divided into two parts: (1) calculating the gradient of loss with respect to the input, and (2) refining the gradient using the optimizer. For the operations in (1), $\partial \mathcal{L}_{tex}/\partial F^{(l)}$ is a series of matrix-vector multiplications and can be implemented by a 1x1 convolutional layer. The gradient of a convolutional layer is a transposed convolutional layer with shared parameters. The gradient of a ReLU layer is a gated layer, and that of an average pooling layer is a tiled (or nearest neighbor upsampling) layer with a constant multiplier. Thus, the computation of (1) can be implemented by a feed-forward network. For (2), it depends on the choice of the optimizer:

- Adam [12] is an efficient optimization method using only the first-order gradients. This method keeps the moving average statistics of elementwise mean and variance of the gradients and computes the adaptive learning rate for each parameter. The computation can be done by using basic arithmetic operations following the formula.

- L-BFGS [26] is a Quasi-Newton optimization method. It maintains a history of gradients and updates and adjusts the direction of the new step by the history information. Most operations are inner-product of vectors and basic arithmetics. The only tricky part is the line search step which requires trial and error, and this can be implemented by a conditional loop. For detailed discussions, we refer the readers to the supplementary materials.

Therefore, the optimization step can be conceptually implemented by a feed-forward network with some additional arithmetic operations depending on the optimizer. The optimization process simply repeats the step for hundreds of times and finally derives a long computation graph.

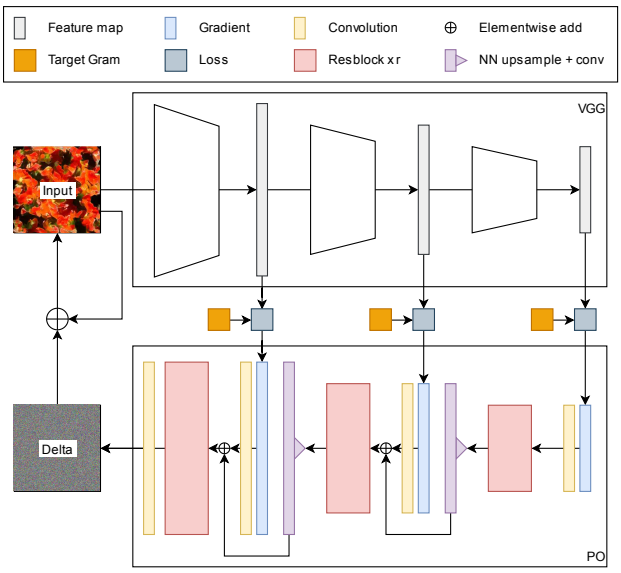

Figure 3. Pseudo optimizer (PO). Our method aims to simulate the optimization process using a feed-forward network. The PO module takes the per-layer gradients calculated from the descriptive network (VGG19) as input and predicts the change to refine the input image. The updater being used is simply an elementwise addition.

3.3. Pseudo Optimizer

Unfolding the optimization loop only provides a way to interpret the iterative algorithm in a feed-forward manner. However, the computation time remains unchanged. One way to reduce the computation time is to limit the number of unfolded iterative steps, but this is achieved at the cost of quality. We observe that the calculation of gradient is closely coupled with the descriptive network which is mainly trained for usage in the forward direction, and thus may not be suitable for efficient texture synthesis. We delicately design a new feed-forward network named Pseudo Optimizer (PO) to simulate the optimization process. The architecture is demonstrated in Figure 3.

The PO network reuses the forward part of the descriptive network (black arrows in Figure 2) to extract the gradient information and replaces the backward part (red arrows in Figure 2) with a learnable network. Specifically, it takes the per-layer gradients of the texture loss (2) as inputs and predicts the pixel-wise modification Δx to the input image:

$$\Delta x = PO\left(\left\{\frac{\partial \mathcal{L}_{tex}(x, \tilde{x}; [L])}{\partial F^{(l)}(x)}\right\}_{l=1}^{L}\right). \tag{3}$$

We write the function on the right as $PO^{[L]}(x, \tilde{x})$ for short. As neural networks are known to be good universal approximators [20], we can train the network to output the optimal solution of (2). However, this is not feasible in practice,

because each noise input may correspond to different local optimum and it takes a too long time to compute the optimum. Therefore, instead of finding the local optimum for each input, we encourage the modified image to be close to the optimal solution of (2) by again applying the texture loss to the output and train the network $PO^{[L]}$ to minimize the following objective:

$$\mathcal{S}^{[L]}(\tilde{x}) \triangleq \mathop{\mathbb{E}}_{x\sim Z} \mathcal{L}_{tex}(x + \Delta x, \tilde{x}; [L]). \tag{4}$$

3.4. Adaptive Pseudo Optimizer

The original optimization-based method [5] is a fully adaptive method. That is, the target texture image can be an arbitrary image and does not need to be seen by the model before. To make our PO network an adaptive method, the input of the network, *i.e*. the per-layer gradients, needs to be descriptive enough. In the following, we discuss the relation between our method and adaptive instance normalization, a key component in adaptive texture synthesis and style transfer.

Relation to Adaptive Instance Normalization Adaptive instance normalization (AdaIN) is widely used in fast adaptive texture synthesis and style transfer [10, 14]. Basically, the activations in some layer of a network is normalized and then transformed by the statistics of the target image $\tilde{x}$:

$$F^{(out)}_{k,i} = \sqrt{\frac{\sigma^2_i(\tilde{F})}{\sigma^2_i(F) + \epsilon}} \left(F_{k,i} - \mu_i(F)\right) + \mu_i(\tilde{F}), \tag{5}$$

where $F := F(x), \tilde{F} := F(\tilde{x})$, μ_i and σ^2_i are the instance-channel-wise mean and variance. WCT [14] further extends the linear transform to a combination of whitening and coloring transforms. Their common idea is to integrate the information to change from the input domain to the target domain with the transform. Our method uses the per-layer gradients as such information:

$$\frac{\partial \mathcal{L}_{tex}(x, \tilde{x})}{\partial F^{(l)}_{k,\cdot}(x)} = \frac{4}{N^{(l)}} \left(G^{(l)}(x) - G^{(l)}(\tilde{x})\right) F^{(l)}_{k,\cdot}(x). \tag{6}$$

The transform matrix is defined by the difference between the input and target Gram matrices with a constant multiplier.

Training an adaptive pseudo optimizer (AdaPO) is quite straightforward by using a set of target images $\{\tilde{x}_i\}_{i=1}^n$. The training objective is defined as follows:

$$\mathcal{S}^{[L]}_{ada}(\{\tilde{x}_i\}_{i=1}^n) \triangleq \frac{1}{n} \sum_{i=1}^{n} \mathcal{S}^{[L]}(\tilde{x}_i). \tag{7}$$

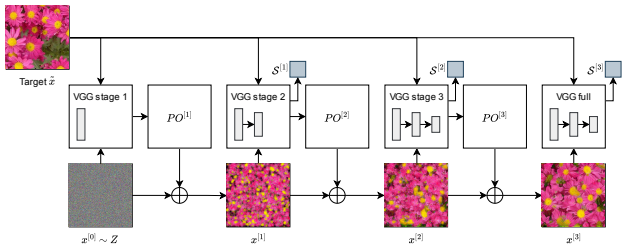

Figure 4. Progressive Pseudo Optimizer (ProPO). The actual number of stages is 5. To save space, we only draw 3 in this figure.

3.5. Progressive Pseudo Optimizer

We further propose a progressive architecture (Figure 4) for training stabililty and diversified results. The progressive model (named ProPO) consists of multi-stage refinements. At the first stage, the PO network takes the noise input $x^{[0]}$ and employs the first level of texture loss to refine the input image. In the following stages, higher levels of texture loss are gradually added to the objectives and the PO network refines the output from the previous stage. Formally, the intermediate images are defined by:

$$x^{[0]} \sim Z(\text{the noise distribution}), \tag{8}$$
$$x^{[m]} = x^{[m-1]} + PO^{[m]}(x^{[m-1]}, \tilde{x}), 1 \le m \le L, \tag{9}$$

and $x^{[L]}$ is regarded as the final output. We impose the texture loss corresponding to the level of input gradients on the intermediate results. The final objective function is the average of texture losses from all stages:

$$\mathcal{S}^{[L]}_{pro}(\{\tilde{x}_i\}_{i=1}^n) \triangleq \frac{1}{L} \sum_{m=1}^{L} \frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{x^{[0]}\sim Z} \mathcal{L}_{tex}(x^{[m]}, \tilde{x}_i; [m]). \tag{10}$$

4. Experiments

4.1. Experimental Setup

Dataset We collect a small dataset consisting of 47 floral images from the internet for illustration purpose, and conduct some preliminary experiments on this dataset. Additionally, we use the Describable Texture Dataset [2] (DTD), a larger dataset, for testing the robustness and scalability of our method. This dataset consists of 47 texture categories and each contains 120 images. We resize the shortest edge of images to 256 and randomly crop and flip patches from the images for data augmentation.

Baselines We summarize the properties of related baseline methods in Table 1. TextureNet [21] is designed for synthesis a single texture and is not scalable to multiple or adaptive syntheses. AdaIN [10] is designed for a closely

Method	Speed	Scalability
Gatys *et al*. [5]	slow	arbitrary
TextureNet [21]	fast	single
MultiTexture [13]	fast	multiple
AdaIN [10]	fast	arbitrary
WCT [14]	fast	arbitrary
PO (ours)	fast	arbitrary

Table 1. Comparison of texture synthesis methods in aspects of speed and scalability. "Single" means the network is trained to synthesize a single texture. "Multiple" means the network is trained to synthesize a fixed set of textures. "Arbitrary" means the network is fully adaptive and can synthesizes in an online manner.

related task, style transfer. Although their model can be adapted to transfer a noise image, we find their synthesized textures are not satisfactory and thus exclude their model from our baselines. In the Section 4.2, we mainly compare the performance of Gatys *et al*. [5], MultiTexture [13], WCT [14] and our method. We use the projects of baselines that can be publicly downloaded from the Internet for the experiments.

Training details Following the setting of [5], five levels of feature maps are extracted from *conv1_1*, *pool1*, *pool2*, *pool3* and *pool4* layers in the VGG19 network. We number them from 1 to 5 and set $L = 5$ for all models. The architecture of the PO network is shown in Figure 3. The per-layer gradient is first passed through two convolutional layers and $r = 2$ residual blocks [8] and then upsampled to match the size of the next feature map. We use the nearest neighbor upsampling followed by a convolutional layer. The outputs of convolutional layers are followed by InstanceNorm [22] and Leaky ReLU layers. We use the ProPO structure in all the experiments because it produces results with better quality and richer diversity than what AdaPO does. Later they are compared in the ablation study. We train the ProPO model to minimize the objective (10) using Adam [12] optimizer. The hyperparameters are set as follows: $batchsize = 1, lr = 2e-4, beta1 = 0.5, beta2 = 0.999$. The number of training iterations is 800K and the learning rate is linearly decreased to 0 after 400K iterations.

4.2. Experimental Results

In the following, we present the experimental results of baseline methods and ours and compare their performance in the aspects of efficiency, quality, diversity, and scalability. More results of our method (ProPO) can be found in Figure 6.

Comparison with Gatys *et al*. [5] is a fully adaptive method that can produce high-quality and diversified results. The main drawback is that it requires a heavy optimization process. We compare the trade-off between the quality and efficiency of this method and ours. We report the texture loss (2) in Table 2 and plot the loss curves over time (iterations) in Figure 5. Gatys *et al*.'s method converges approximately after 400 iterations and achieves a superior texture loss comparing to other fast feed-forward methods and ours. However, it takes more than one minute to synthesize that using a 2080 Ti GPU and 16 CPU cores. Our unoptimized code can synthesize over 13 images per second on a single GPU, which outperforms Gatys *et al*.'s method by a large margin. The qualitative results can be viewed in Figure 6. We can see that the synthesized images (6th columns) are very similar to the target images (the last column) in both aspects of coarse and fine structures.

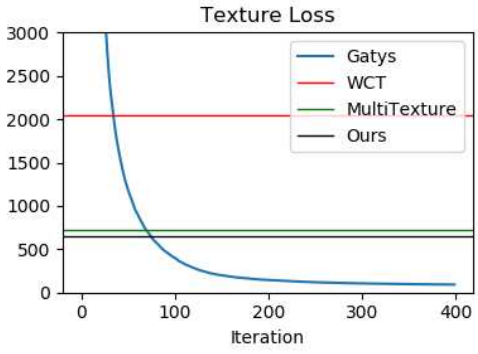

Figure 5. Curves of texture loss. Gatys *et al*.'s method finally outperforms the other three feed-forward methods. While our methods can synthesize over 13 images per second on a single GPU.

Comparison with fast feed-forward methods. For the fast feed-forward methods, we focus on comparing their diversity and scalability. Diversity can measured by the diversity loss defined in [13]. We modify the formula by taking the expectation of the loss in a set of synthesized images $\{x_i\}_{i=1}^b$:

$$\mathcal{L}_{div}(\{x_i\}_{i=1}^b) = \mathop{\mathbb{E}}_{i \neq j} \left\| F^{(div)}(x_i) - F^{(div)}(x_j) \right\|_1, \quad (11)$$

where $F^{(div)}$ is the feature map at the *conv4_2* layer of the VGG19 network. Following their setting, we synthesize $b = 5$ images for each texture and compute the diversity loss for each method respectively. The numeric results are listed in Table 2. We can see that our method generally outperforms the other two fast feed-forward methods. The qualitative results are shown in Figure 7. Our synthesized texture images present richer diversity in both coarse and fine details. For validating the robustness and scalability, we train our method with the same structure on DTD. The training data size increases from 47 to 5760. All of the three methods have a raise in the texture loss. Our method still

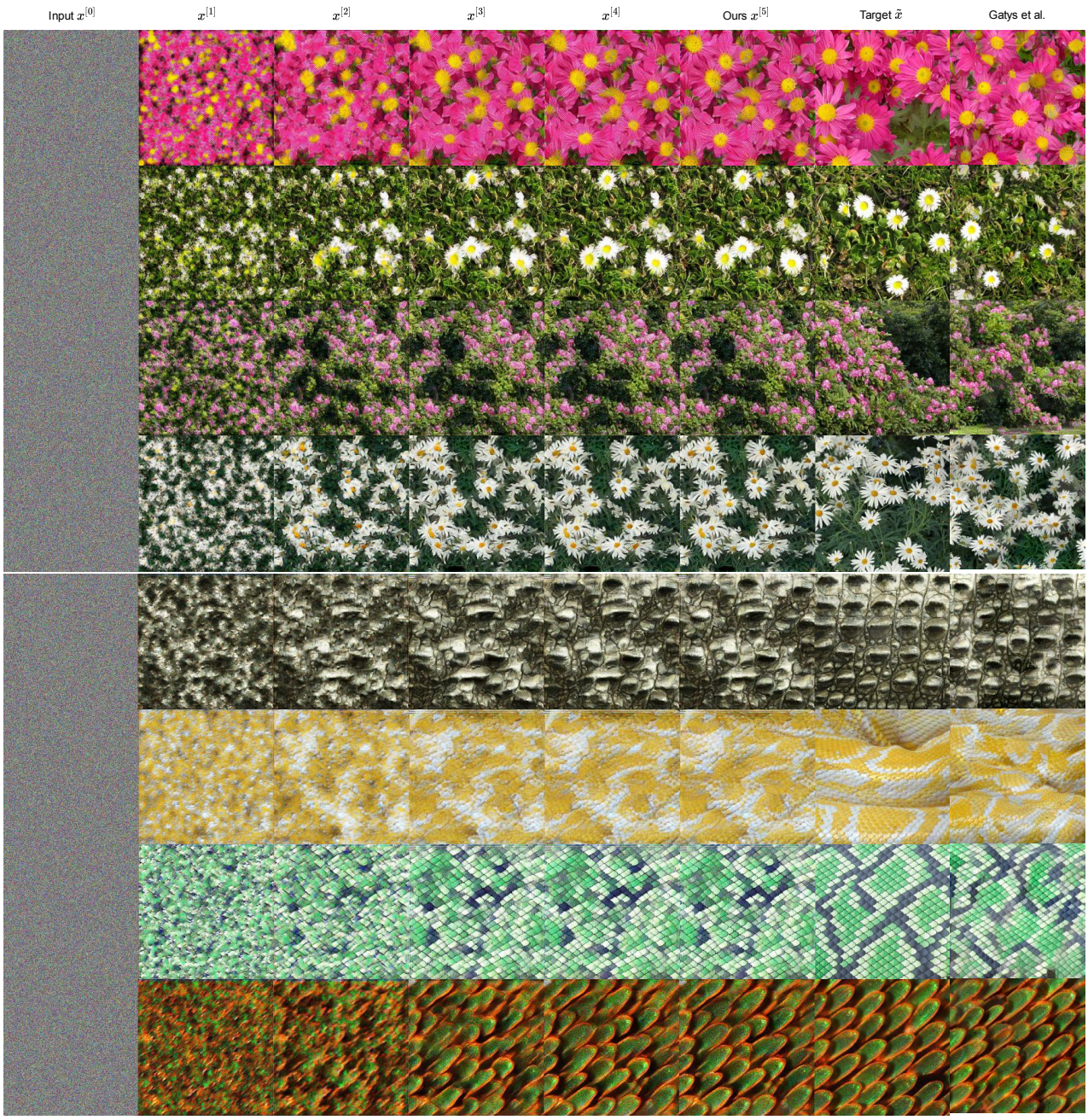

Figure 6. Results of ProPO. The leftmost column contains the input noise images $x^{[0]} \sim Z$. The second to the fifth columns contain the results of $x^{[1]}, x^{[2]}, x^{[3]}, x^{[4]}$ respectively. The sixth column contains the output of our method, $x^{[5]}$. The seventh column contains the target texture images $\tilde{x}$. The rightmost column contains the results of Gatys et al. [5].

achieves the best performance. We further test our method on unseen images collected from the Internet. The results are demonstrated in Figure 8. Most colors and textures can be preserved, which shows that our method can generalize to unseen images. While some large-scale patterns cannot be recognized and this is to be investigated in the future.

4.3. Ablation Study

Design of architecture When training using the single-stage model, AdaPO, we find that the texture loss is in general higher than the one using ProPO. Moreover, we often observe frame-like artifacts in the synthesized images of AdaPO (Figure 9). We believe that it is very difficult

Figure 7. Synthesized results (the three on the left) using the same target image (right) and different noise inputs.

Figure 8. Synthesized results (left) of unseen images (right).

Method	Quality ↓		Diversity ↑
	Floral	DTD	Floral
Gatys *et al*. [5]	90.1	N/A	433.2
MultiTexture [13]	719.8	6639.0	385.9
WCT [14]	2042.5	6673.2	309.3
PO (ours)	645.9	4672.5	397.1

Table 2. Comparison of adaptive (multiple) texture synthesis methods. Quality is measured by the texture loss (2) (the lower the better). Diversity is measured by the diversity loss (11) (the higher the better). It takes a too long time to synthesize images on DTD using Gatys *et al*.'s method. So we omit their results on DTD. We want to mention that their results usually achieve better quality and richer diversity.

Figure 9. Results of AdaPO. Frame-like artifacts are frequently observed in the synthesized results (left).

to directly synthesize from a noise image to an image that matches the full levels of texture features. Inspired by [14], we propose the multi-stage model (ProPO) to synthesize the output by gradually matching the features from lower levels to higher levels. Taking the first stage $PO^{[1]}$ (Figure 4) as example, the neuron in the first layer has a small receptive field and it is responsible for low-level features, like colors. It is relatively easier to synthesize a small color patch from a noise input. The multi-stage effects can be observed in Figure 6. We also observe that the multi-stage structure results in richer diversity. We conjecture that the progressive structure can encourage the locality of our optimizer as the output is synthesized incrementally and the following PO modules tend to make as little modification as possible to the previous results.

Extended objective function We observe that our framework is not tied to texture loss. The only requirement is that the objective is differentiable. This allows us to use extended texture losses, *e.g*. shifted Gram matrix [1] and multi-scale Gram matrix [19]. We show some results using the objective of [1] in the supplementary materials.

5. Conclusion

We propose a new framework, named Pseudo Optimizer (PO), for fast texture synthesis. Our method simulates the optimization process of Gatys *et al*.'s method [5] using a feed-forward network (AdaPO). The network takes the per-layer gradients from the texture loss to predict the modification to make the input closer to the optimal solutions. We further proposed a progressive architecture (ProPO) to improve the quality and diversity of synthesized images. Both networks are fully adaptive and can synthesize images controlled by the target image both inside and outside of the training dataset. Extensive experiments are conducted on two datasets. Our model can synthesize visually pleasing and high-quality texture images in near-realtime. Our method outperforms other fast synthesis methods in aspects of quality, diversity, and scalability.

Although our method runs faster than Gatys *et al*.'s method in the inference phase, there are still opportunities for further improvements in the quality. We also plan to investigate the performance on large-scale self-supervised learning tasks using different objective functions.

Acknowledgement This work is partially supported by Science and Technology Service Network Initiative of Chinese Academy of Sciences (KFJ-STS-QYZX-092), Guangdong Special Support Program (2016TX03X276), and National Natural Science Foundation of China (U1813218, U1713208), Shenzhen Basic Research Program (JCYJ20170818164704758, CXB201104220032A), the Joint Lab of CAS-HK, Shenzhen Institute of Artificial Intelligence and Robotics for Society.

References

[1] Guillaume Berger and Roland Memisevic. Incorporating long-range consistency in cnn-based texture generation. *arXiv: Computer Vision and Pattern Recognition*, 2016. 2, 8

[2] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. 5

[3] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 341–346, New York, NY, USA, 2001. Association for Computing Machinery. 2

[4] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1033–1038 vol.2, 1999. 2

[5] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems 28*, pages 262–270. Curran Associates, Inc., 2015. 1, 2, 3, 4, 5, 6, 7, 8

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. 2

[7] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 399–406, USA, 2010. Omnipress. 2, 3, 4

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 6

[9] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 229–238, New York, NY, USA, 1995. Association for Computing Machinery. 2

[10] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1510–1519, 2017. 1, 2, 5, 6

[11] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *The European Conference on Computer Vision (ECCV)*, September 2018. 2

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 3, 4, 6

[13] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Diversified texture synthesis with feed-forward networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017. 1, 2, 6, 8

[14] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems 30*, pages 386–396. Curran Associates, Inc., 2017. 1, 2, 5, 6, 8

[15] Mingyu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. Few-shot unsupervised image-to-image translation. *arXiv: Computer Vision and Pattern Recognition*, 2019. 2

[16] Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *arXiv: Image and Video Processing*, 2019. 3

[17] Eero P. Simoncelli and Javier Portilla. Texture characterization via joint statistics of wavelet coefficient magnitudes. In *Proc. 5th Int'l Conf. on Image Processing Chicago, IL*, pages 4–7. IEEE Computer Society, 1998. 2

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv: Computer Vision and Pattern Recognition*, 2014. 3

[19] Xavier Snelgrove. High-resolution multi-scale neural texture synthesis. In *SIGGRAPH Asia 2017 Technical Briefs*, SA '17, pages 13:1–13:4, New York, NY, USA, 2017. ACM. 2, 8

[20] Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233 – 268, 2017. 2, 3, 4

[21] Dmitry Ulyanov, Vadim Lebedev, Andrea, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1349–1357, New York, New York, USA, 20–22 Jun 2016. PMLR. 1, 2, 5, 6

[22] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. 6

[23] Pierre Wilmot, Eric Risser, and Connelly Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *CoRR*, abs/1701.08893, 2017. 2

[24] Jian Zhang and Bernard Ghanem. Ista-net: Interpretable optimization-inspired deep network for image compressive sensing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2, 3, 4

[25] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *ACM Trans. Graph.*, 37(4), July 2018. 2

[26] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, Dec. 1997. 3, 4

[27] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards

a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, Apr. 1998. 2