Supplementary Material for 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

1. Details of Unfolding Optimization Loop

In this section, we go into detail about unfolding optimization loop. As mentioned in the main paper, one optimization step of [2] consists of two steps: (1) computing the gradient of texture loss with respect to the input, and (2) refining the gradient using the optimizer.

Computing the gradient The step (1) is an evaluation of the forward and backward propagation of the descriptive network, VGG19 [4]. The backward computation can be implemented by components, such as transposed convolutional layers, gated layers and tiled layers, in most deep learning frameworks.

Refining the gradient Unfolding the step (2) depends on the choice of optimizers. For *Stochastic Gradient Descent* (SGD), the optimizer is simply an addition of the gradient (scaled by the learning rate) and the input. For more advanced methods like Adam [3] and L-BFGS [5], the computation is more complex but still can be implemented by basic arithmetics.

- Adam keeps the moving average elementwise first-order and second-order statistics of the gradients and computes the adaptive learning rate for each parameter. The compuation of refined gradient Δx is defined as follows:

$$\begin{aligned} g_t &= \nabla_x \mathcal{L}_{tex}(x_t, \tilde{x}), \\ m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t, \\ \hat{m}_t &= m_t/(1-\beta_1^t), \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2, \\ \hat{v}_t &= v_t/(1-\beta_2^t), \\ \Delta x_t &= -\alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon), \end{aligned} \tag{1}$$

  where $\beta_1, \beta_2, \alpha, \epsilon$ are constant hyperparameters. The computation of moving average statistics can be implemented by a composite layer like BatchNorm.

- L-BFGS is a Quasi-Newton optimization method. It starts with an estimate of the optimal solution, x_0, and refines the estimate iteratively based on the history of gradients and updates. The computation of descent direction is defined as follows:

$$\begin{aligned} g_t &= \nabla_x \mathcal{L}_{tex}(x_t, \tilde{x}), \\ s_t &= x_{t+1} - x_t, \\ y_t &= g_{t+1} - g_t, \\ \rho_t &= 1/(y_t^T s_t), \\ H_{t+1} &= (I - \rho_t s_t y_t^T) H_t (I - \rho_t y_t s_t^T) + \rho_t s_t s_t^T, \\ z_t &= H_t g_t. \end{aligned} \tag{2}$$

  The inverse Hessian is not calculated explicitly. Instead, the search direction, z_t, is computed iteratively from the history of g_t, s_t and y_t. The scaling of the descent direction is determined by a line search method, which can be implemented by a conditional loop in a deep learning framework. The step length of 1 is often accepted in most iterations.

Therefore, both steps of (1) and (2) can be implemented by a feed-forward network and some additional arithmetic operations depending on the optimizer. The whole optimization process simply repeats the step for hundreds of times and finally derives a long computational graph.

2. More Experimental Results

We show more synthesized texture images using our method in Figure 1, 2, 3, 4, 5 and 6.

3. Extended Objective Function

We show synthesized texture images using our method with objective functions of [1] in Figure 7.

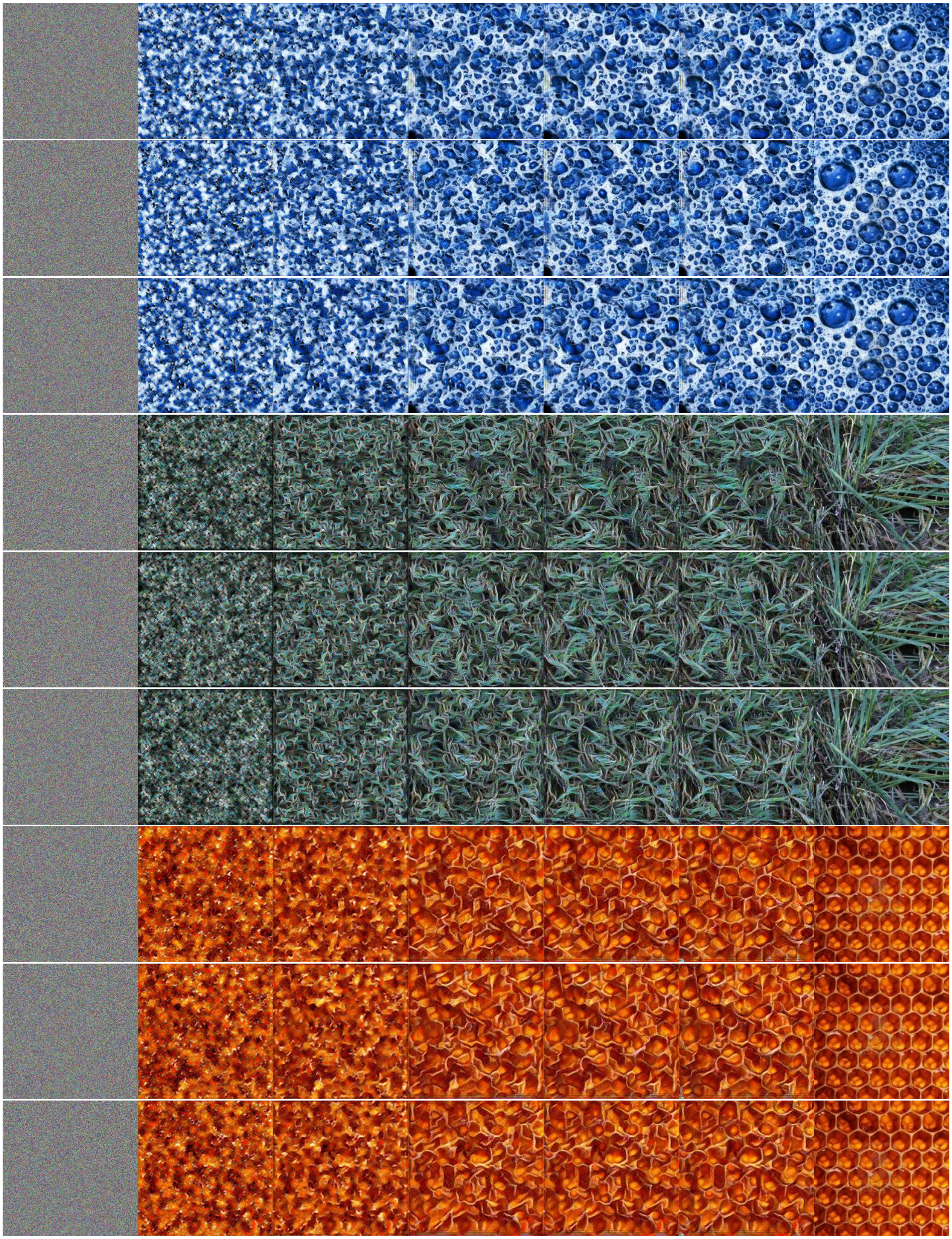

Figure 1. Results of ProPO. The leftmost column contains the input noise images $x^{[0]} \sim Z$. The rightmost column contains the target texture images $\tilde{x}$. The intermediate five columns contain the results of $x^{[1]}, x^{[2]}, x^{[3]}, x^{[4]}, x^{[5]}$ respectively. Each synthesis is repeated three times using different noise inputs.

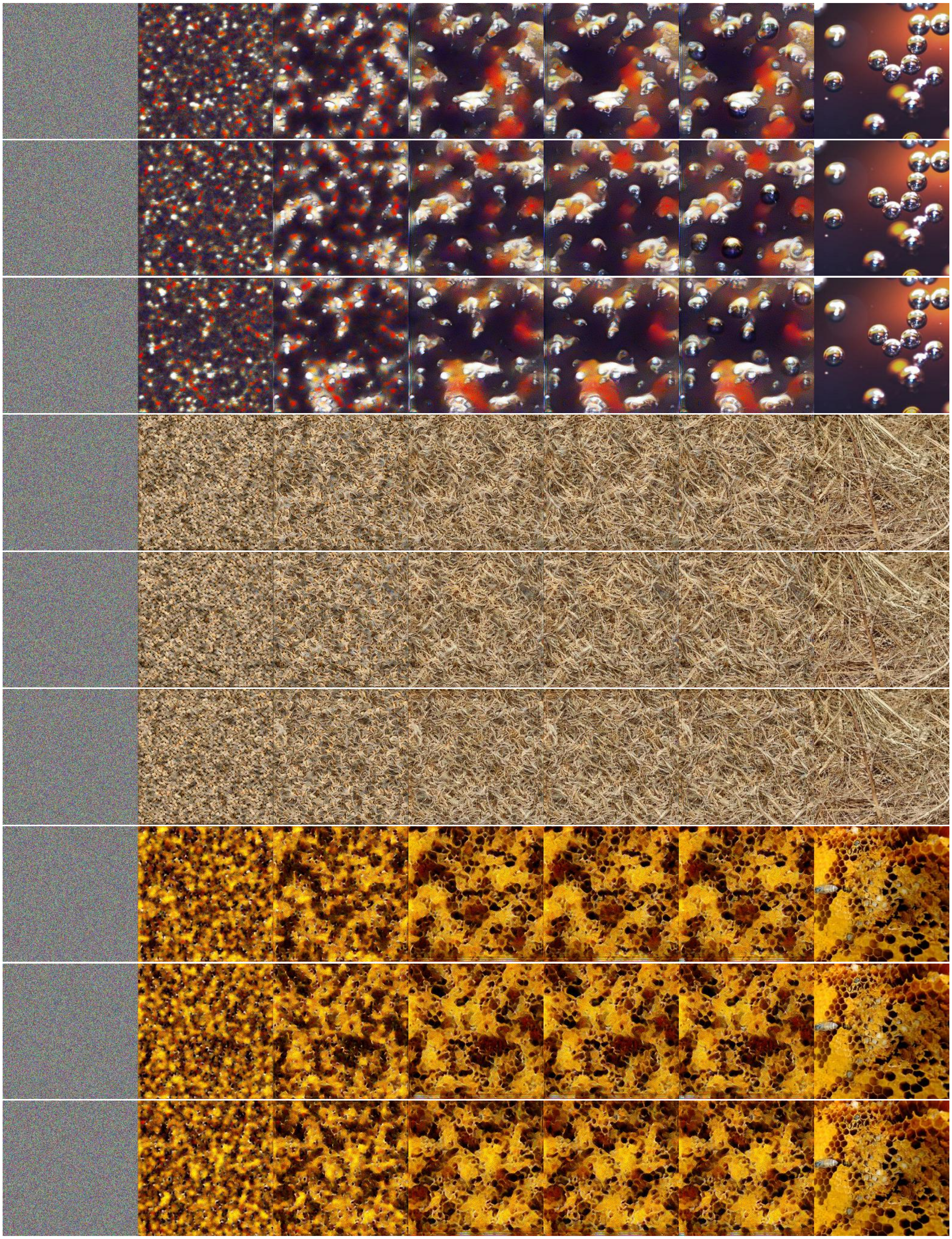

Figure 2. Results of ProPO. The leftmost column contains the input noise images $x^{[0]} \sim Z$. The rightmost column contains the target texture images $\tilde{x}$. The intermediate five columns contain the results of $x^{[1]}, x^{[2]}, x^{[3]}, x^{[4]}, x^{[5]}$ respectively. Each synthesis is repeated three times using different noise inputs.

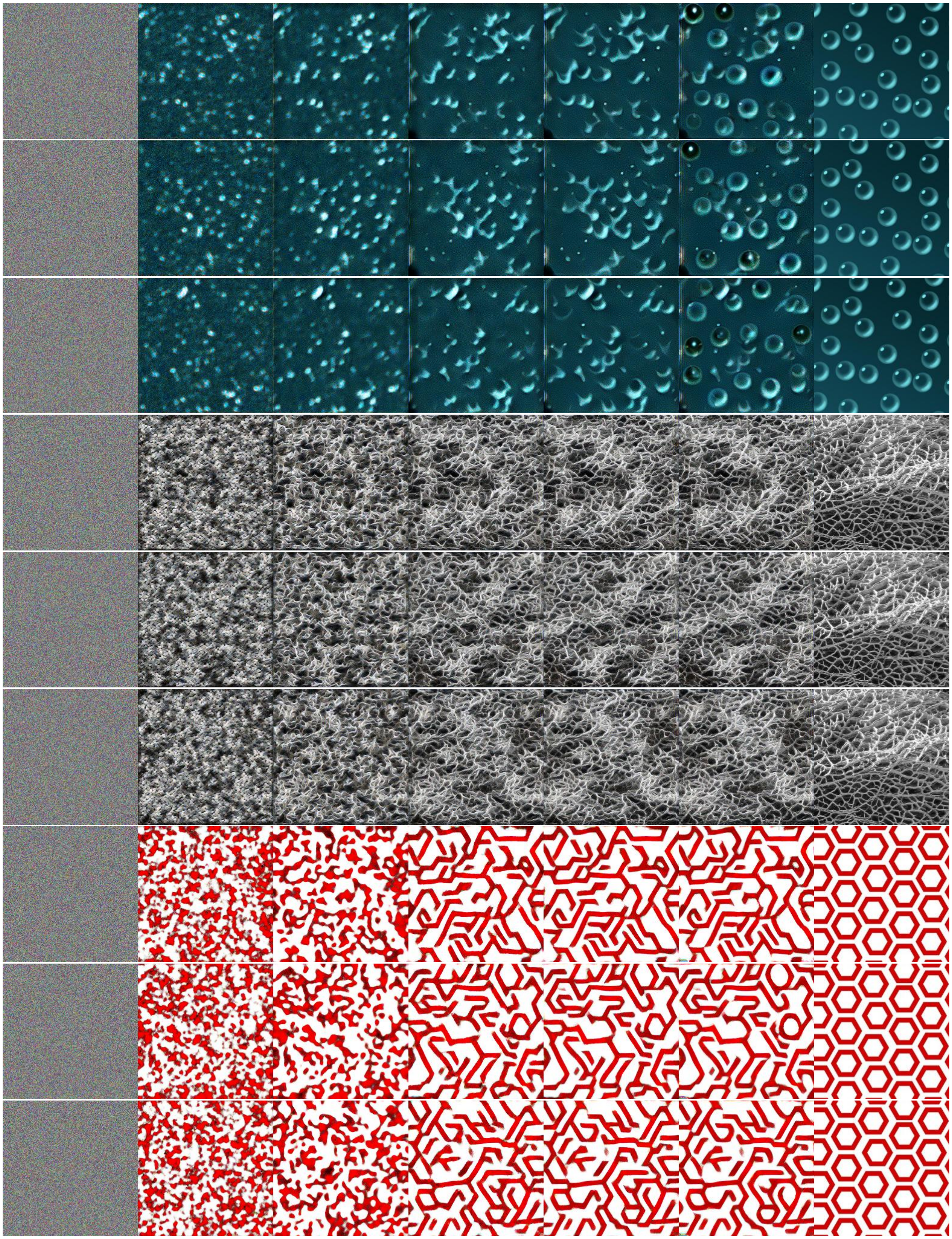

Figure 3. Results of ProPO. The leftmost column contains the input noise images $x^{[0]} \sim Z$. The rightmost column contains the target texture images $\tilde{x}$. The intermediate five columns contain the results of $x^{[1]}, x^{[2]}, x^{[3]}, x^{[4]}, x^{[5]}$ respectively. Each synthesis is repeated three times using different noise inputs.

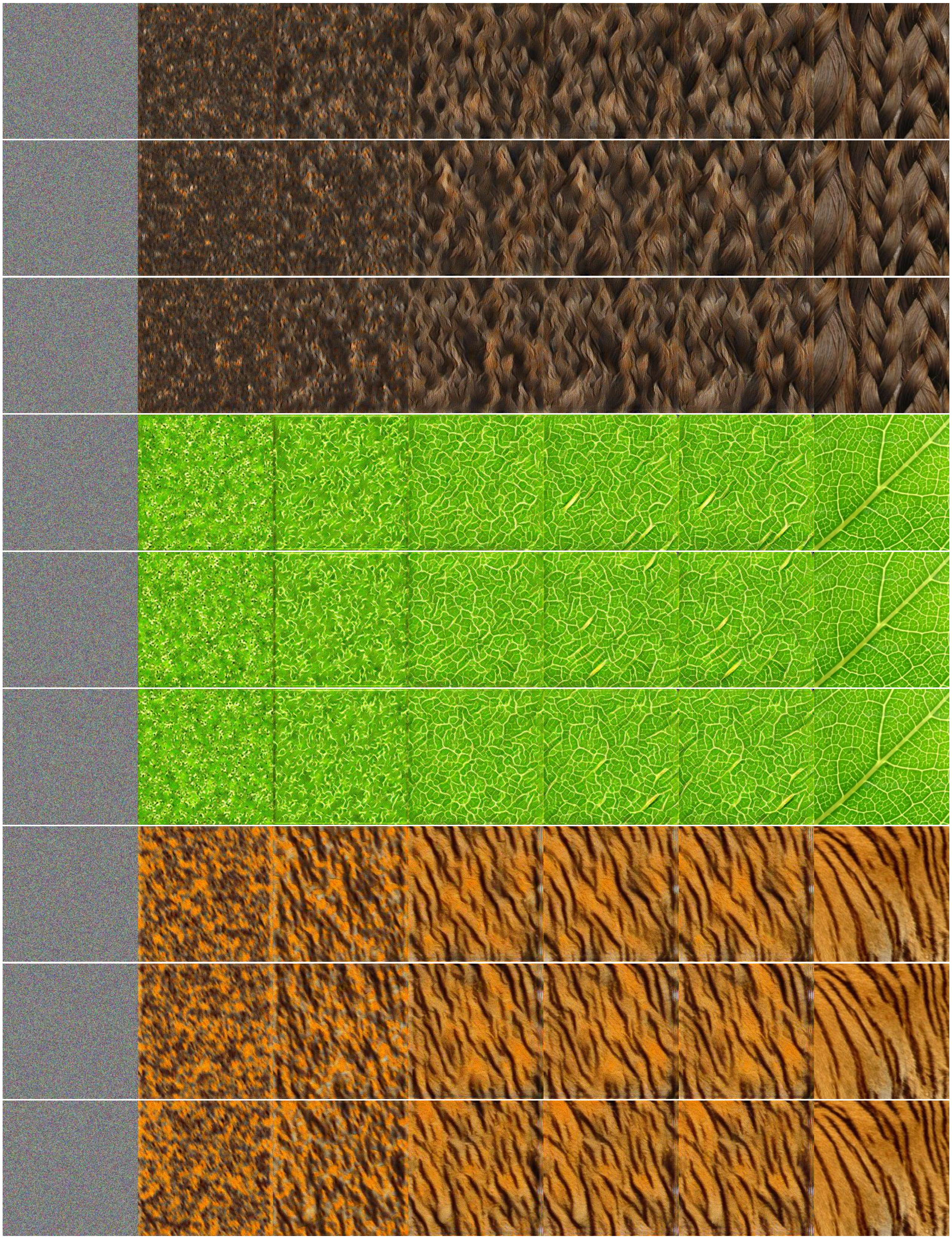

Figure 4. Results of ProPO. The leftmost column contains the input noise images $x^{[0]} \sim Z$. The rightmost column contains the target texture images $\tilde{x}$. The intermediate five columns contain the results of $x^{[1]}, x^{[2]}, x^{[3]}, x^{[4]}, x^{[5]}$ respectively. Each synthesis is repeated three times using different noise inputs.

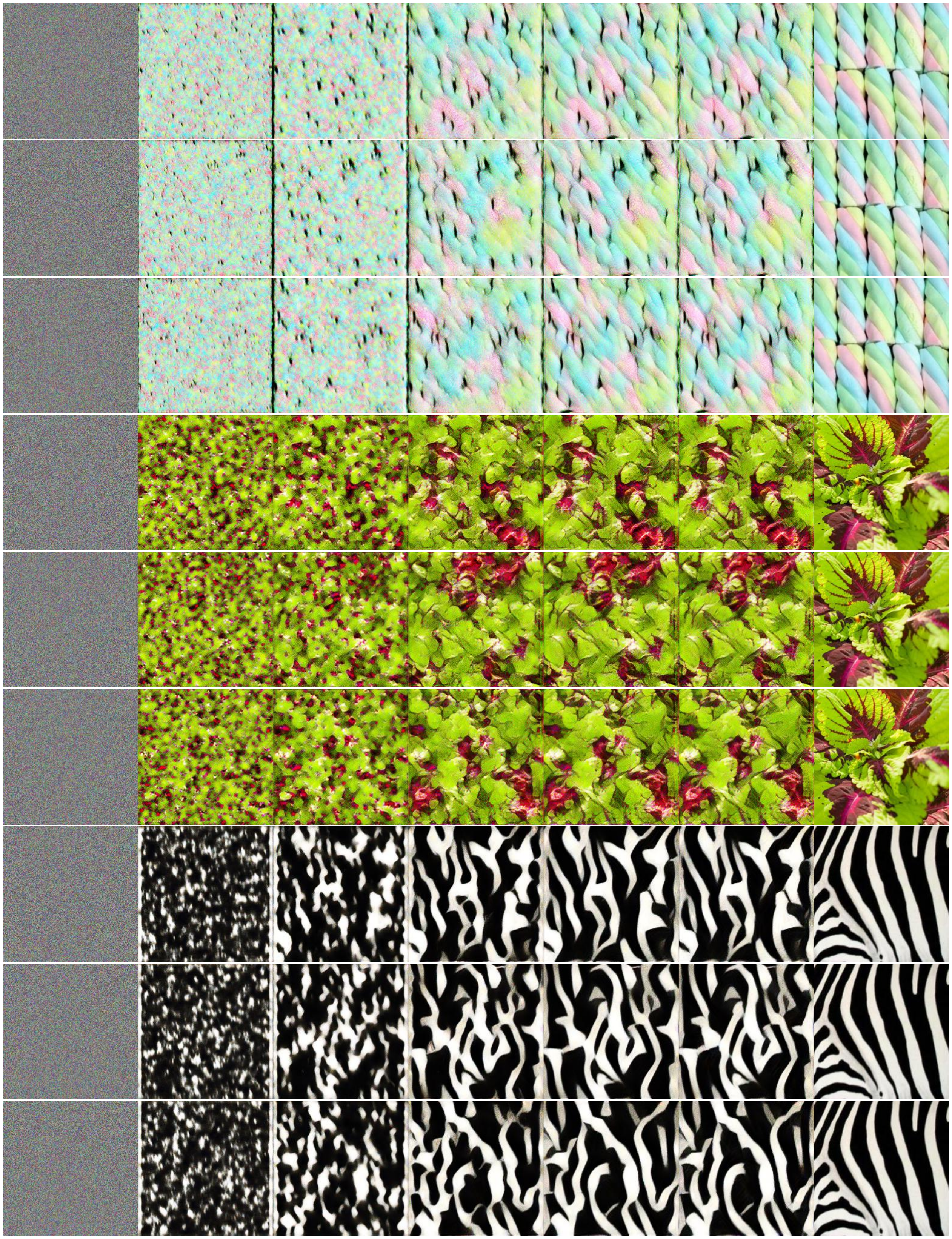

Figure 5. Results of ProPO. The leftmost column contains the input noise images $x^{[0]} \sim Z$. The rightmost column contains the target texture images $\tilde{x}$. The intermediate five columns contain the results of $x^{[1]}, x^{[2]}, x^{[3]}, x^{[4]}, x^{[5]}$ respectively. Each synthesis is repeated three times using different noise inputs.

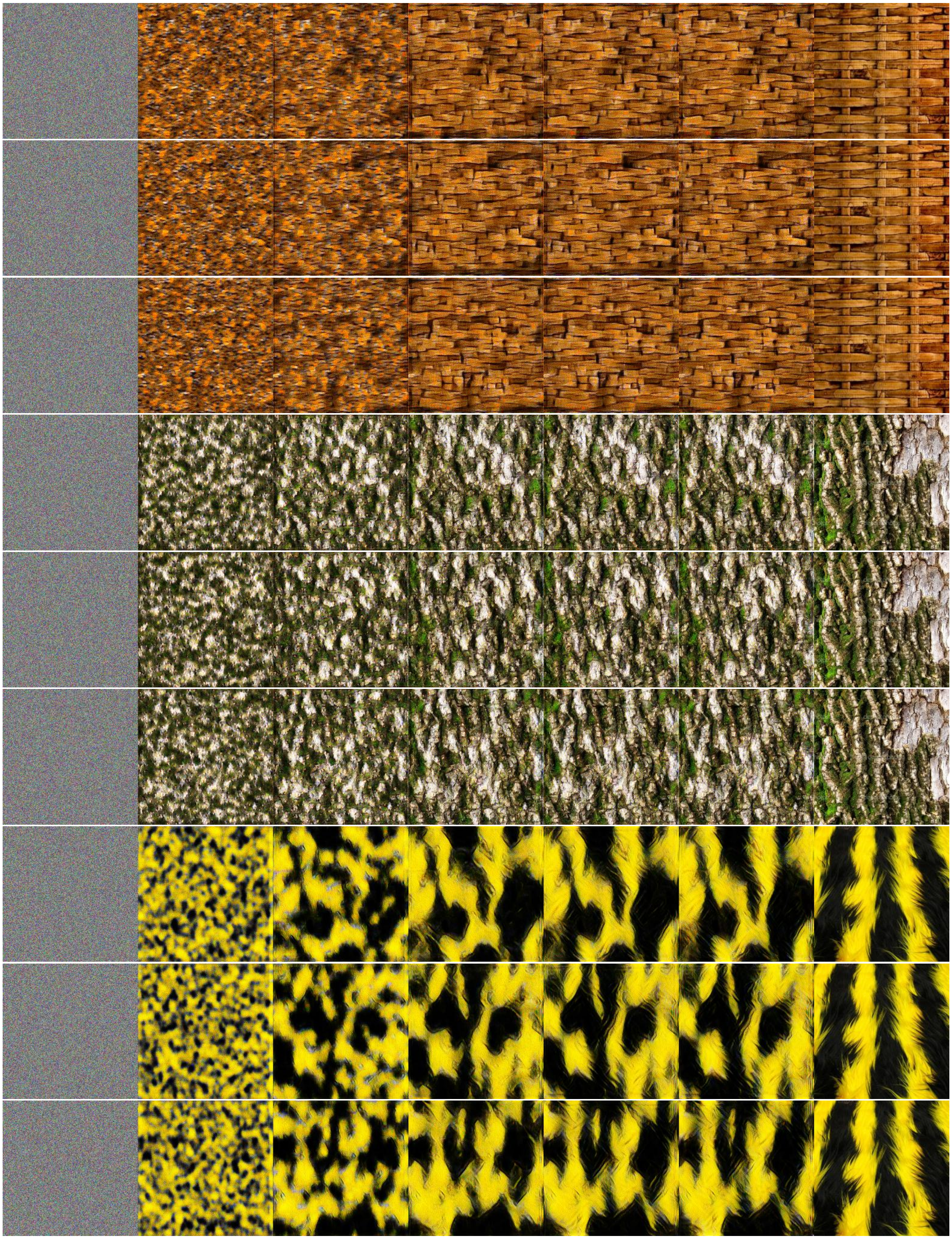

Figure 6. Results of ProPO. The leftmost column contains the input noise images $x^{[0]} \sim Z$. The rightmost column contains the target texture images $\tilde{x}$. The intermediate five columns contain the results of $x^{[1]}, x^{[2]}, x^{[3]}, x^{[4]}, x^{[5]}$ respectively. Each synthesis is repeated three times using different noise inputs.

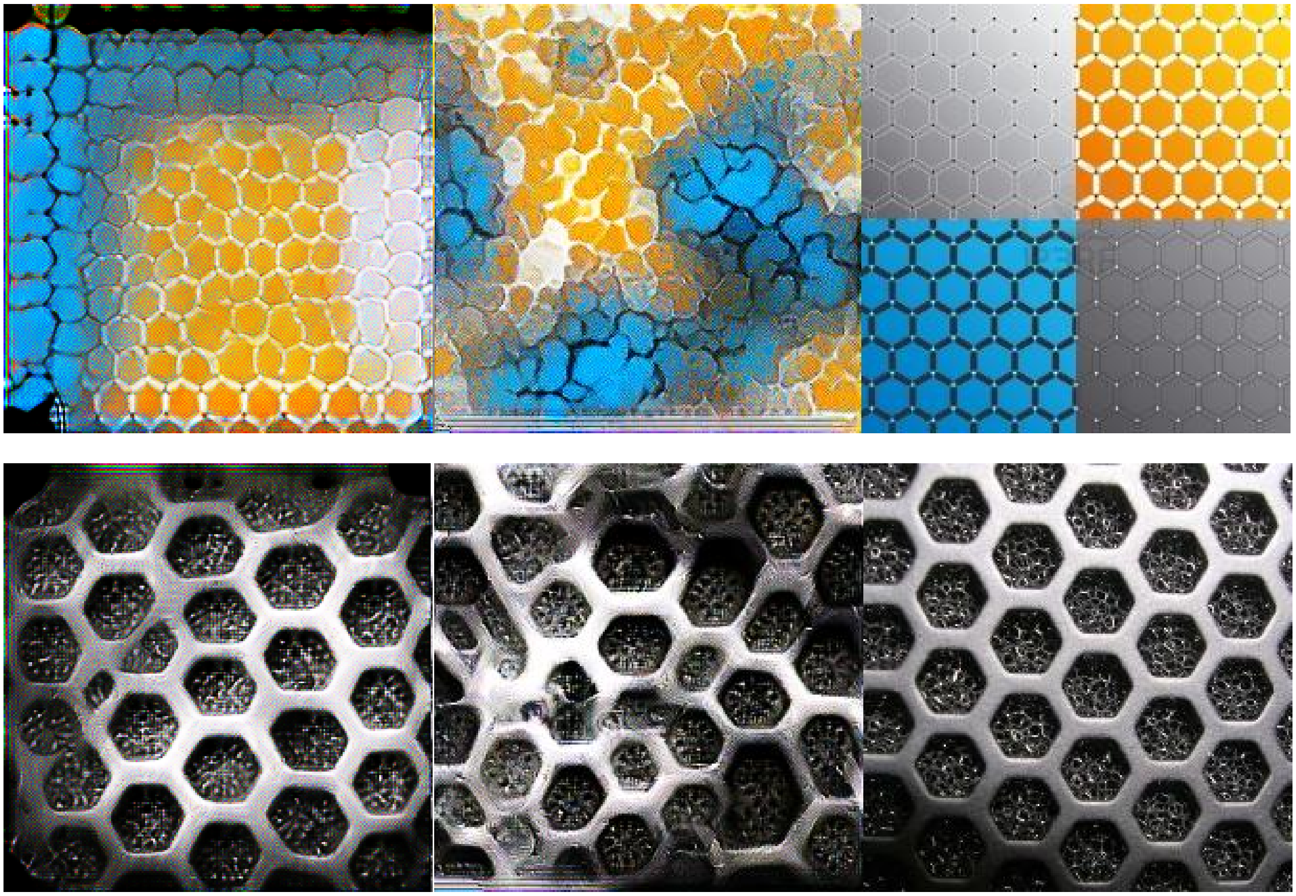

Figure 7. Results of our method (the second column) with long-range texture loss [1] (the first column).

References

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

[2] 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

[3] 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. 1

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv: Computer Vision and Pattern Recognition*, 2014. 1

[5] 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. 1