Supplementary Materials: Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

Xun Huang[1] Serge Belongie[1,2]
[1]Department of Computer Science, Cornell University [2]Cornell Tech
{xh258,sjb344}@cornell.edu

1. Loss curves during training

In Fig. 1 we show the content and style loss in training and test set. The training loss is post-processed with median filtering of window size 2000 and the test loss is computed every 2000 iterations. We did not observe much overfitting: the training and test loss trend are almost the same. We conjecture that the task is so difficult that the network with millions of parameters still underfit. With a more expressive architecture, the results might be further improved.

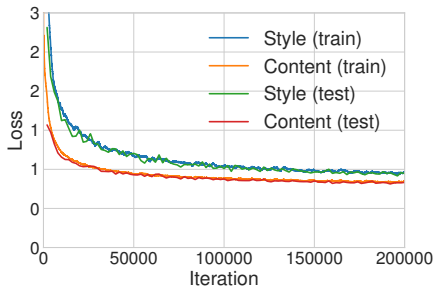

Figure 1. Loss curves during training.

2. Model and training details

Model. Tab. 1 shows the detailed architecture of the decoder. All the convolutional layers are of stride 1, filter size 3×3, and "valid" padding of type reflection. Note that we do not use any normalization layers. The VGG-19 network [4] used as the encoder and loss network is a normalized version as in Gatys *et al*. [1]. In addition to the normalization conducted by Gatys *et al*., we also replace all zero paddings with reflection paddings.

Training. The convolutional weights are initialized with *Xavier* [2] initialization. The style weight λ is set to 0.1. We use Adam [3] optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate is initialized to $\alpha_0 = 10^{-4}$ and decays as follows:

$$\alpha_t = \frac{\alpha_0}{1 + kt}$$

where $k = 5 \times 10^{-5}$ and $t \leq T$ is the current number of iterations. We train the network for $T = 300,000$ iterations which takes roughly one week on a single Pascal Titan X. However, visually appealing results can be obtained with only $30,000$ iterations.

Layer type	# Channels
Conv-ReLU	256
Upsampling	256
Conv-ReLU	256
Conv-ReLU	256
Conv-ReLU	256
Conv-ReLU	128
Upsampling	128
Conv-ReLU	128
Conv-ReLU	64
Upsampling	64
Conv-ReLU	64
Conv	3

Table 1. Architecture of the decoder.

3. More examples

See Fig. 3 in the next page.

4. Effect of using different layers

Fig. 2 shows the effect of using different layers to perform AdaIN. Using `relu4_1` obtains perceptually better results than earlier layers.

Figure 2. Effect of using different layers. Original content and style images can be found in Fig. 7 of the main paper.

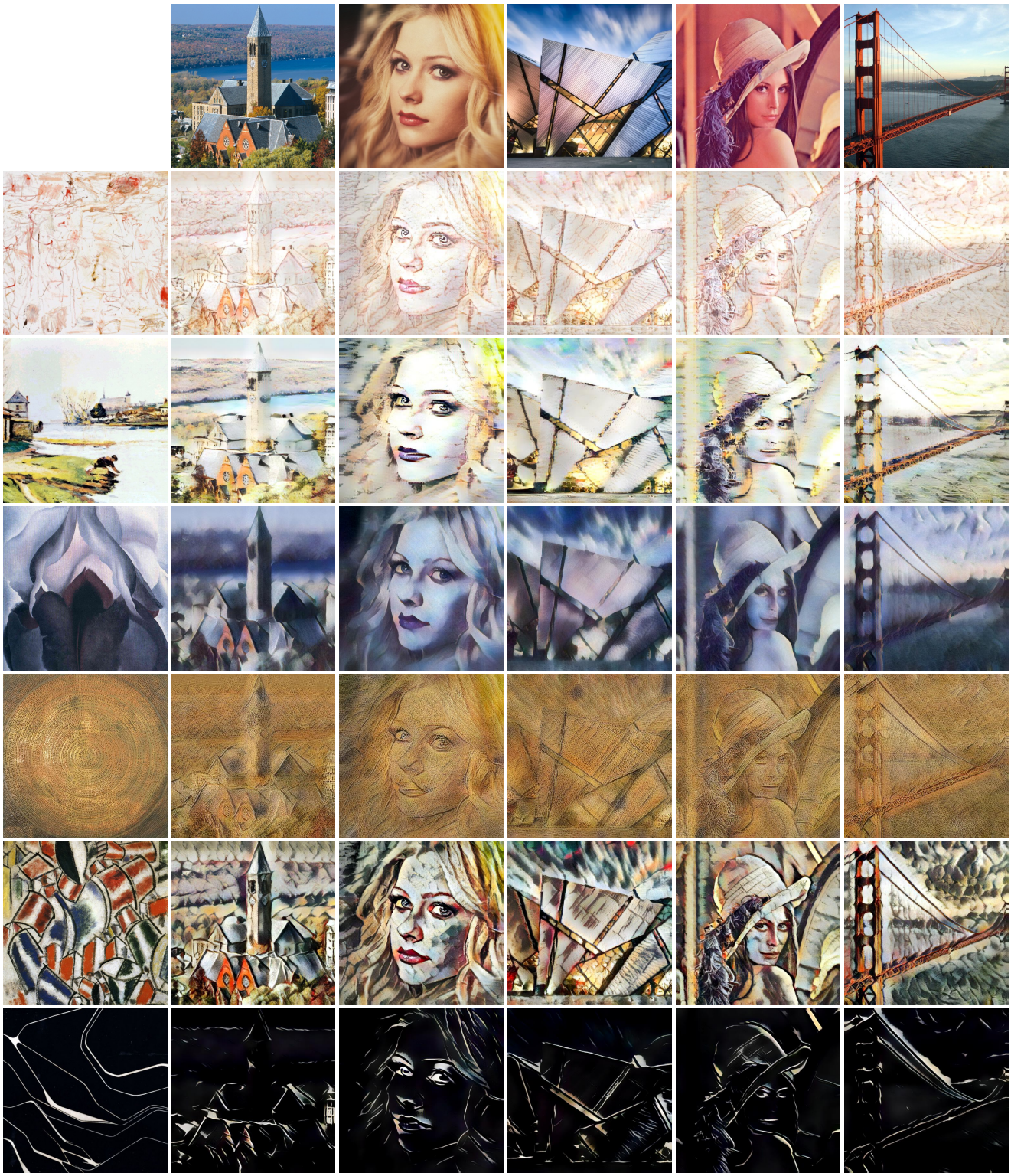

Figure 3. More examples of style transfer. Each row shares the same style while each column represents the same content. As before, the network has never seen the test style and content images.

References

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

[2] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. 1

[3] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 1

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1