Rethinking Style Transfer: From Pixels to Parameterized Brushstrokes

- Supplementary Material -

Dmytro Kotovenko* Matthias Wright* Arthur Heimbrecht Björn Ommer
IWR, Heidelberg Collaboratory for Image Processing, Heidelberg University

Contents

1. Videos 1

2. Controlling the Flow of Brushstrokes with User Input 1

3. Renderer 2
- 3.1. Brushstroke Parameterization 2
- 3.2. Tensor of Distances 2
- 3.3. Hardware, Runtime, Memory 3
  - 3.3.1 Number of Brushstrokes 3

4. Trained Renderer 3
- 4.1. Architecture 3
- 4.2. Training 4

5. Additional Results 4
- 5.1. Fitting Brushstrokes to Artwork 4
- 5.2. User Study 4

1. Videos

In order to give insights into the stylization procedure, we provide two videos, see:

- https://heibox.uni-heidelberg.de/f/77c92a1355904de6a6be/
- https://heibox.uni-heidelberg.de/f/c17c112570f646bab081/

In these videos we show how the stylization evolves over time, before and after pixel optimization. We also compare to Gatys et al. [3] and show how the user input (Sec. 2) influences the stylization. See Fig. 1.

2. Controlling the Flow of Brushstrokes with User Input

In Sec 5.4 of the paper, we showed how our method enables us to control the flow of brushstrokes. A user can draw arbitrary curves on the content image through a user interface and the brushstrokes in the stylized image will follow these curves. This can be achieved by adding a simple projection loss, which we will explain in this section.

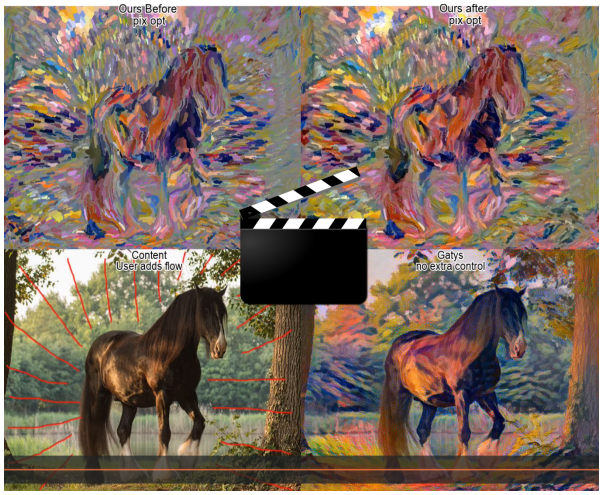


Figure 1. We provide videos that show how the stylization evolves over time. Moreover, we show how flow constraints change the stylization. See Sec. 1. Videos are attached to the supplementary material.

Each drawn curve is represented as a set of points $P_1, P_2, ..., P_M$ (do not confuse those with the control points for a Bézier curve). For each point P_i the approximate tangent vector $\mathbf{v}_i$ is computed as follows:

$$\mathbf{v}_i = \frac{\tilde{\mathbf{v}}_i}{||\tilde{\mathbf{v}}_i||}, \quad \tilde{\mathbf{v}}_i = \left(\frac{1}{Q} \sum_{j=1}^{Q} P_{i+j} \right) - P_i, \qquad (1)$$

where $Q = 3$ in all our experiments. Fig. 2 shows user drawn curves with corresponding tangent vectors.

As explained in Sec. 4 of the paper, each brushstroke is represented as a quadratic Bézier curve with additional parameters for location, width, and color. A quadratic Bézier curve is parameterized by three points: a start point, an end point, and a control point. Roughly speaking, the start and end points determine the curves orientation and the control

*Both authors contributed equally to this work.

Figure 2. Top: A user draws arbitrary curves through a user interface. Bottom: A curve is represented as a set of points. For each point we can compute an approximate tangent vector.

point determines the curvature. The projection loss is computed as follows:

1. Compute for each brushstroke the vector from the start point to the end point of the Bézier curve, see Fig. 3. We will refer to this vector as the *orientation vector* of a brushstroke.

2. For each tangent vector, compute the L nearest brushstrokes on the canvas. L is a hyperparameter and determines the range of the brushstrokes that will be affected by the drawn curves. Fig. 5 shows the influence of L on the stylization.

3. For each tangent vector, compute the projection of the orientation vectors from the nearest brushstrokes onto the tangent vector. Both the tangent vectors and the orientation vectors are normalized to unit length.

4. The projection loss encourages the absolute value of these projections to be 1. Since all vectors are normalized, the absolute value of the projections will be 1 if and only if the orientation vectors are parallel to the tangent vector. See Fig. 6 for an overview of the whole computation.

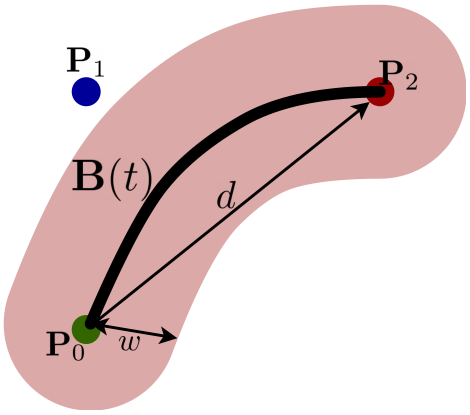


Figure 3. The brushstroke is parameterized by color $rgb \in \mathbb{R}^3$, width $w \in \mathbb{R}$ and Bézier curve $\mathbf{B}(t)$. The Bézier curve is defined by points $\mathbf{P}_0,\ \mathbf{P}_1,\ \mathbf{P}_2 \in \mathbb{R}^2$ and position on a curve $t \in [0;1]$. The direction (orientation) vector $d \in \mathbb{R}^2$ is used to simplify the strokes.

See Fig. 7 and 8 for more results.

3. Renderer

3.1. Brushstroke Parameterization

Each brushstroke is parameterized by color $rgb \in \mathbb{R}^3$, Bézier curve $\mathbf{B}(t)$ with $t \in [0;1]$ and width $w \in \mathbb{R}$. Bézier curve $\mathbf{B}(t)$ is introduced in the main paper in Eq. 4:

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2\,, 0 \leq t \leq 1. \quad (2)$$

From the formulation we see that $\mathbf{B}(t)$ depends on three points $\mathbf{P}_0,\ \mathbf{P}_1,\ \mathbf{P}_2 \in \mathbb{R}^2$ which we further call start point, control point and end point, respectively. We additionally define the direction (orientation) $d \in \mathbb{R}^2$ of a stroke as $d := \mathbf{P}_2 - \mathbf{P}_0$. This vector will be used in Sec. 2 to control the flow of the brushstrokes. This parameterization of a brushstrokes is illustrated in Fig. 3.

3.2. Tensor of Distances

As described in Alg. 1 of the main paper, the cornerstone of our rendering mechanism is a tensor of distances D between every sampled point on a brushstroke and each point on a canvas. We use a canvas $\mathcal{C}$ of size $H \times W$, where $H = W = 256$. We typically draw $N = 5000$ brushstrokes and on every brushstrokes we sample $S = 10$ points. This results in a tensor of shape $H \times W \times N \times S$. If we use `float32` data type taking 4 Bytes, then the distances tensor has size $256 * 256 * 5000 * 10 * 4 = 13107200000 \approx$ 13GB, which is infeasible in practice. However, we actually do not need to compute the distances between every pixel and every brushstroke since a pixel is only affected by a few

nearby brushstrokes, say by K nearest brushstrokes (in our implementation we set $K = 20$). With this in mind, we can reduce the tensor of distances D of shape $H \times W \times N \times S$ to the size $H \times W \times K \times S$ which requires $\frac{N}{K} = \frac{5000}{20} = 250$ times less memory, roughly 52MB.

In order to accomplish the tensor size reduction, we need to assign the K nearest brushstrokes to each pixel. However, for this we would need the tensor of distances of size $H \times W \times N$, which is not feasible for large values of N. This problem can be circumvented if we compute the distances from each of the N brushstrokes to a sparse subset of "anchor" points on the tensor of locations $\mathcal{C} \in \mathbb{R}^{H \times W}$. We create a tensor $\mathcal{C}_{\text{coarse}}$ of size $H' \times W'$ where $H' < H$ and $W' < W$ containing subset of tensor $\mathcal{C}$, in our case we set $H' = 0.1 \cdot H$ and $W' = 0.1 \cdot W$. Now we can effectively compute the tensor of distances between $\mathcal{C}_{\text{coarse}}$ and each brushstroke, it will have shape $H' \times W' \times N$. We left out dimension S because we only need to roughly estimate the distances of the brushstrokes that are close to the location, so we use the location coordinates of the whole brushstroke. Now we extract the K nearest strokes at every location and obtain a tensor of indices $idcs'$ having shape $H' \times W' \times K$. We then apply nearest neighbor upsampling to $idcs'$ across dimensions H and W and obtain $idcs$, a tensor of shape $H \times W \times K$. Thus, every pixel will have the same nearest neighbors as the nearest "anchor" point. This tensor of indices indicates the K nearest brushstrokes for each pixel of the canvas. Now using this tensor of indices $idcs$ and the `tf.gather` operation in TensorFlow we can effectively assign to every location only the K nearest strokes. We note that the same stroke is assigned to multiple locations but this does not hinder the optimization process because the stroke will just receive more gradient information.

3.3. Hardware, Runtime, Memory

Our stylization process consists of two stages. At the first stage we optimize brushstroke parameters, at the second stage we optimize individual pixels.

Brushstroke parameters optimization. We use a canvas of size $H \times W$, where $H = W = 256$ and optimize using our renderer for 1000 steps using the Adam Optimizer [7]. It takes around 3 minutes.

Pixel optimization. Now we upsample the canvas with fitted strokes to have the smallest image side of 1024px and keep the input content image aspect ratio. This image with fitted brushstrokes is used as both content image and initialization for the standard Gatys et al. stylization routine. We optimize for another 1000 steps using the Adam optimizer. It takes 4 more minutes to converge. All the experiments are conducted on NVIDIA TitanXP or NVIDIA 2080Ti graphic cards.

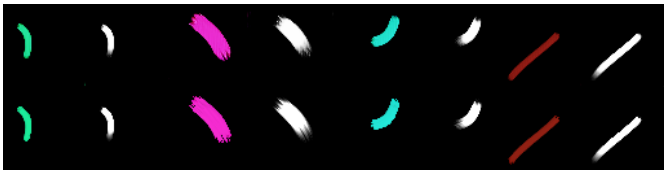


Figure 4. Generated brushstrokes using the trained renderer. Top row: generated brushstrokes. Bottom row: ground truth simulated in the FluidPaint environment [https://david.li/paint/]. The alpha mask is always white and located to the right of the brushstroke.

3.3.1 Number of Brushstrokes

The memory consumption does not depend on the number of strokes, see Sec. 3.2. However, the run-time reduces linearly as the number of strokes increases, see Tab.1.

4. Trained Renderer

In order to ablate our renderer, we trained a neural network that receives brushstroke parameters as input and generates the corresponding brushstrokes. The brushstrokes are parameterized as described in Sec. 4 of the main paper or in Sec. 3 of the supplementary. The network generates an RGB image of a brushstroke as well as an alpha mask. In order to render N brushstrokes onto a canvas, we first have to generate each brushstroke individually and then blend them together using the alpha masks. See Fig. 4 for some generated brushstrokes.

4.1. Architecture

The brushstroke generator follows the StyleGAN architecture [6] and consists of a mapping network f and a synthesis network g, here we adopt the notation from the StyleGAN paper. The mapping network f takes in the brushstroke parameters z and processes them using 4 fully-connected layers to create the latent vector w. The synthesis network g consists of 4 blocks, each consisting of an upsampling layer, a 3x3 convolutional layer, and an AdaIN layer [5]. The latent vector w is injected into the AdaIN layers using learned affine transformations. After every convolutional layers we also inject noise.
The discriminator follows the StyleGAN architecture [6] as well, however, it only consists of 8 layers.

Table 1. Run-time and memory analysis. Experiment conducted on a TITAN Xp GPU.

# Strokes	1K	5K	10K	15K	20K
Speed [iter/s]	1.17 ± 0.01	1.16 ± 0.01	1.08 ± 0.01	1.0 ± 0.01	0.93 ± 0.01
Memory [GB]	9550	9550	9550	9550	9550

4.2. Training

For training, we used the Wasserstein GAN loss [2] with gradient penalty [4] and a L2 loss (equally weighted). We used the Adam optimizer [7] with learning rate 0.0002.

5. Additional Results

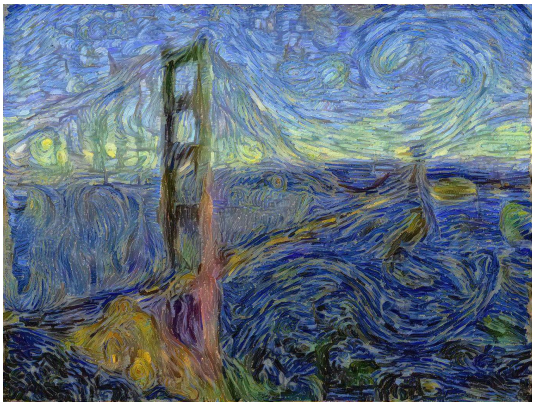
We provide additional stylization examples in Fig. 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26.

5.1. Fitting Brushstrokes to Artwork

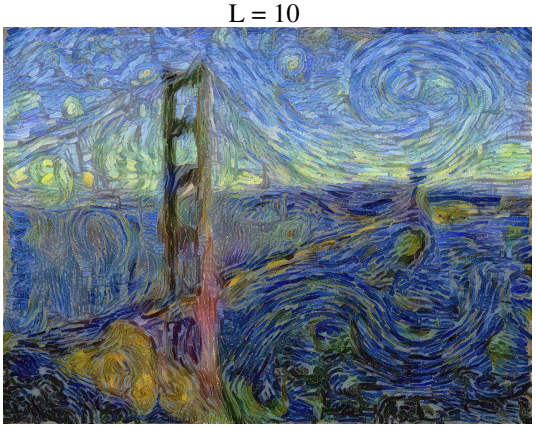
In Sec. 5.3 of the paper we showed how we can use our renderer to fit brushstrokes to paintings. Note that we use SLIC superpixels [1] to initialize the brushstrokes for this experiment. We only optimize the brushstroke parameters and did not apply pixel level optimization for this experiment. See Fig. 27, 28, 29 for additional results.

5.2. User Study

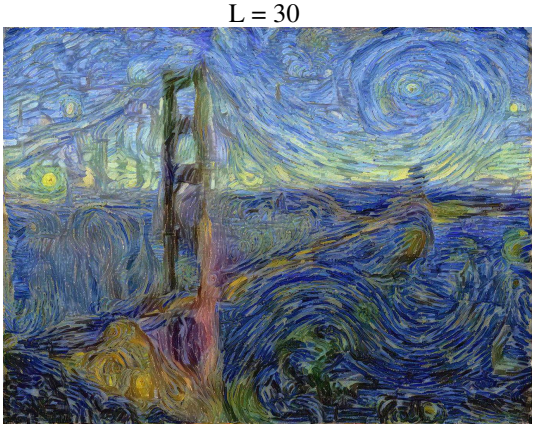
To evaluate the quality of the synthesized images we use two methods. First, we compute the deception score, as suggested by Sanakoyeu et al. [9]. This score indicates how similar is a given stylization to the actual style of the artist. Another way to evaluate the quality of images is to perform a user study. We show to a human subject cropouts from images obtained using different stylization approaches or real artworks and ask them to pick crops from a real artwork. Among the stylization approaches we have Gatys et al. [3], AST by Sanakoyeu et al. [9], WCT [8], and AdaIN [5]. At once we show 4 images: each drawn randomly and independently. In Fig. 30 we present randomly drawn example trials. Note that in Fig. 30 we do not provide images for WCT [8] and AdaIN [5] since those are easy to spot in most cases.



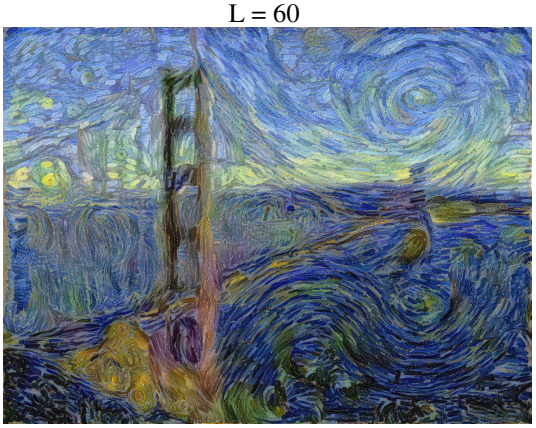
L = 10



L = 30



L = 60



L = 100

Figure 5. Controlling the flow of brushstrokes with different values for L. The larger the value L the more strokes around the user input are affected. User input is porvider in Fig.2.

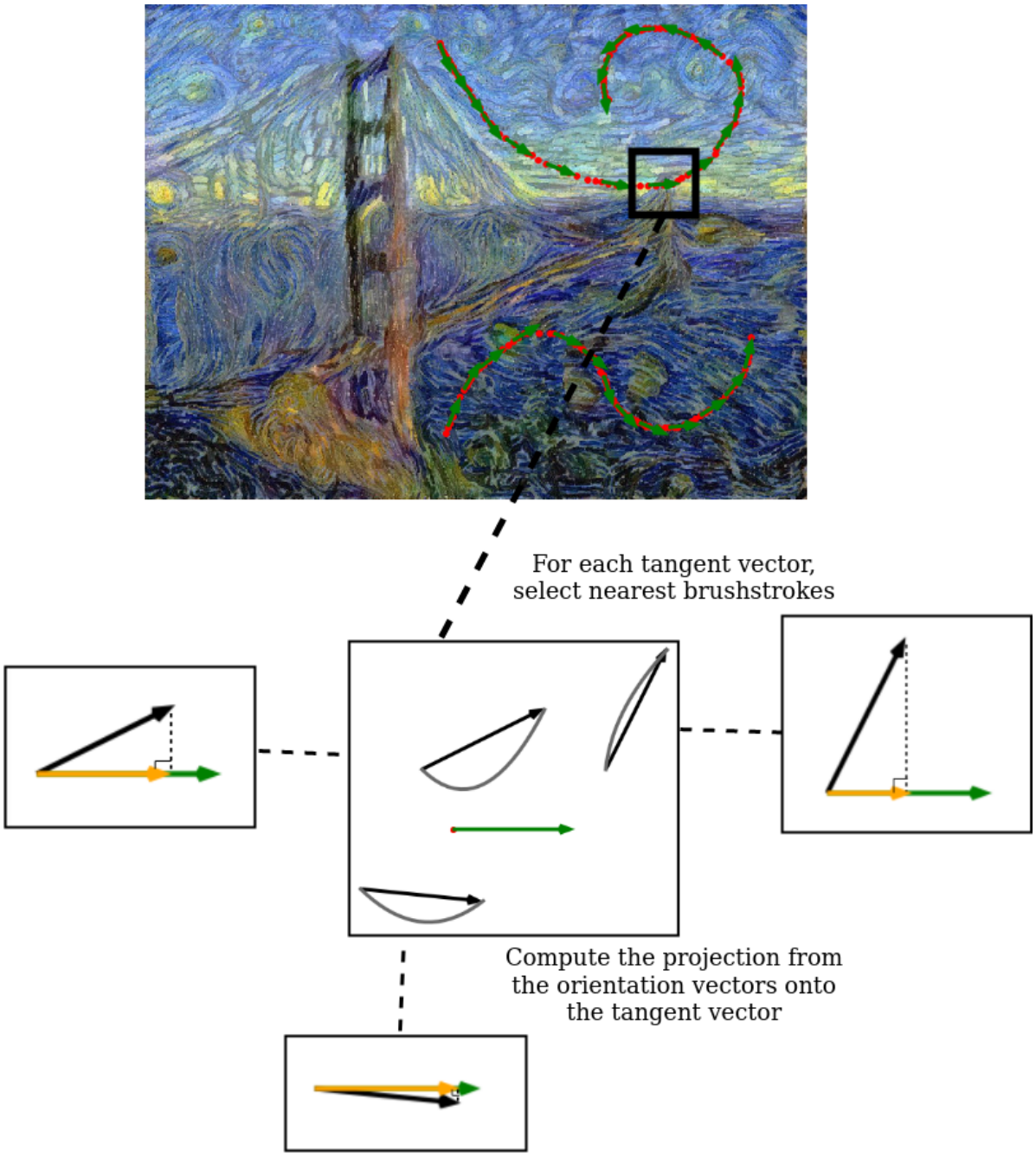
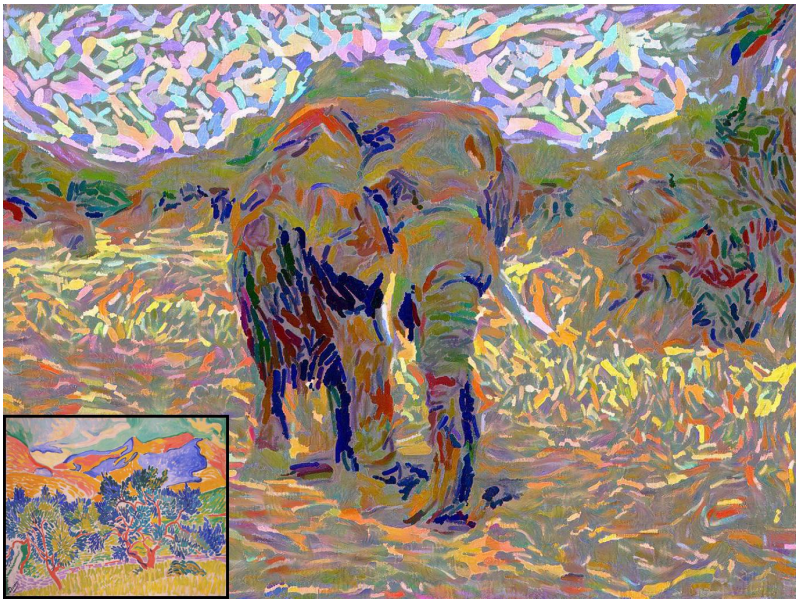
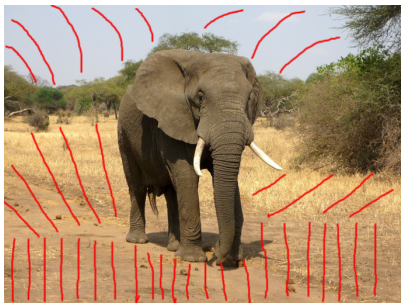


Figure 6. Overview of projection loss for a specific tangent vector. For each tangent vector (green), we select the closest brushstrokes (gray). We then take the orientation vector (black) for each brushstroke and compute the projection (yellow) onto the tangent vector.

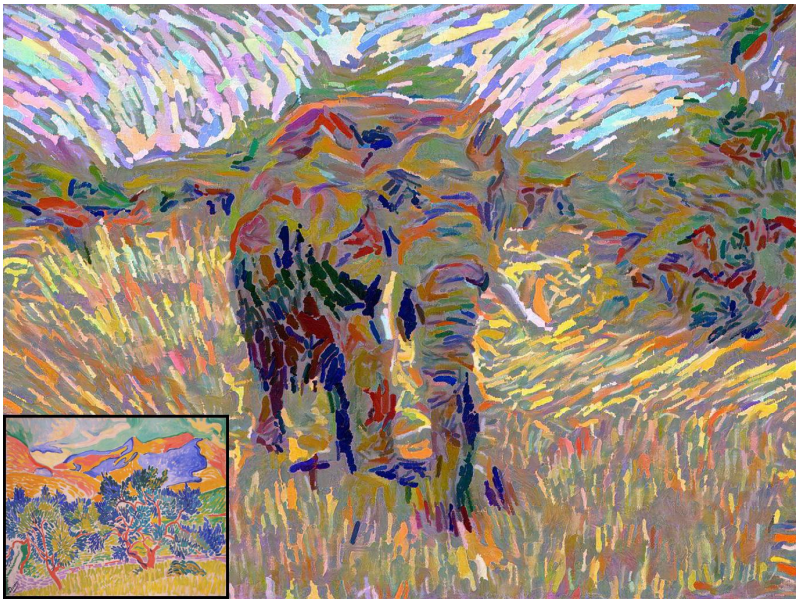
(a) Content



(b) Stylized without user input



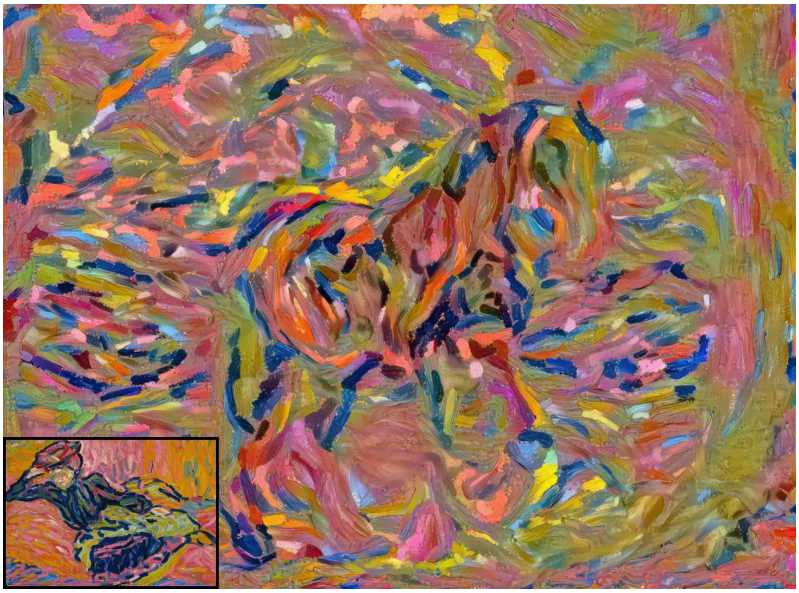
(d) Content with user input



(e) Stylized with user input

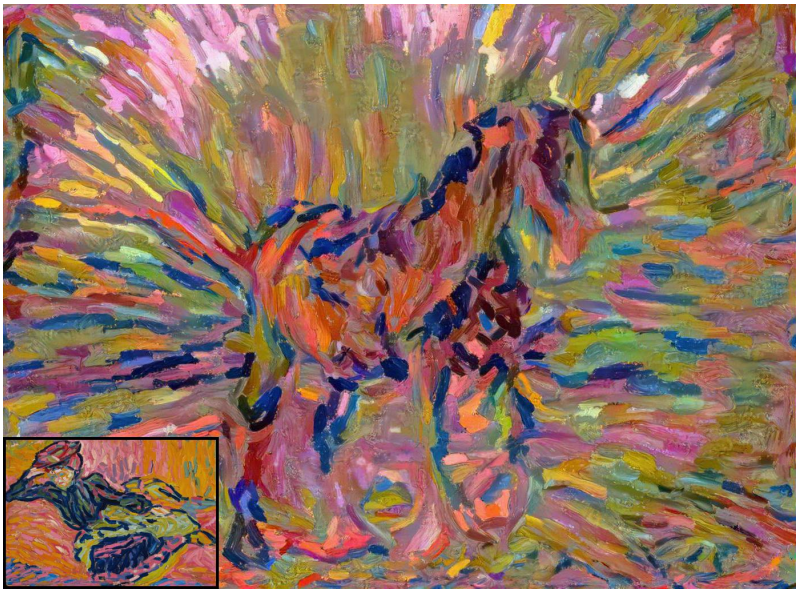
Figure 7. A user can draw curves on the content image and thus control the flow of the brushstrokes in the stylized image. Note that for the stylization with user input we also used (a) as content image. The control is imposed on the brushstroke parameters, not the pixels.

(a) Content



(b) Stylized without user input

(d) Content with user input



(e) Stylized with user input

Figure 8. A user can draw curves on the content image and thus control the flow of the brushstrokes in the stylized image. Note that for the stylization with user input we also used (a) as content image. The control is imposed on the brushstroke parameters, not the pixels.

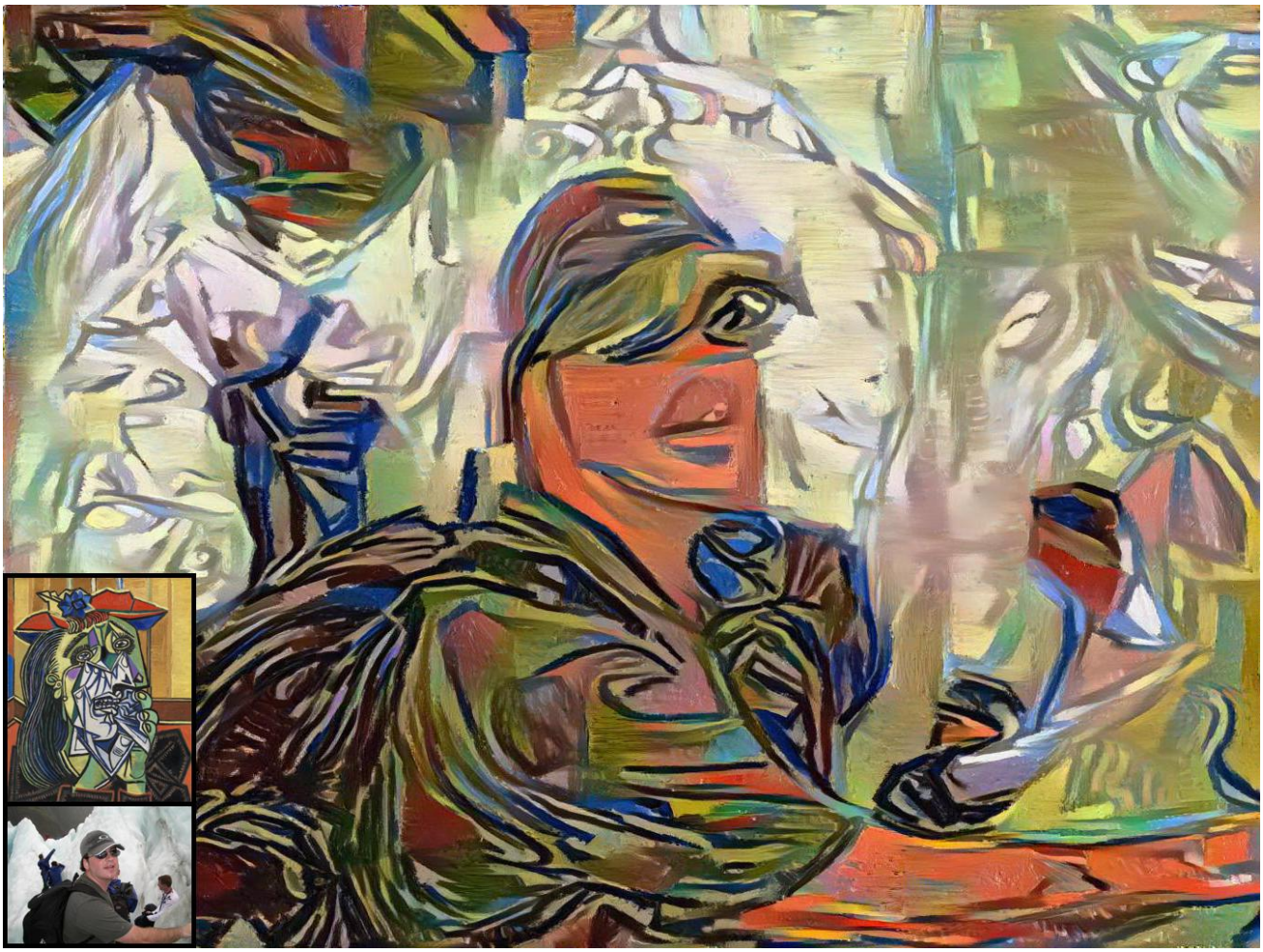


Figure 9. Style image: “The Weeping Woman” by Pablo Picasso.

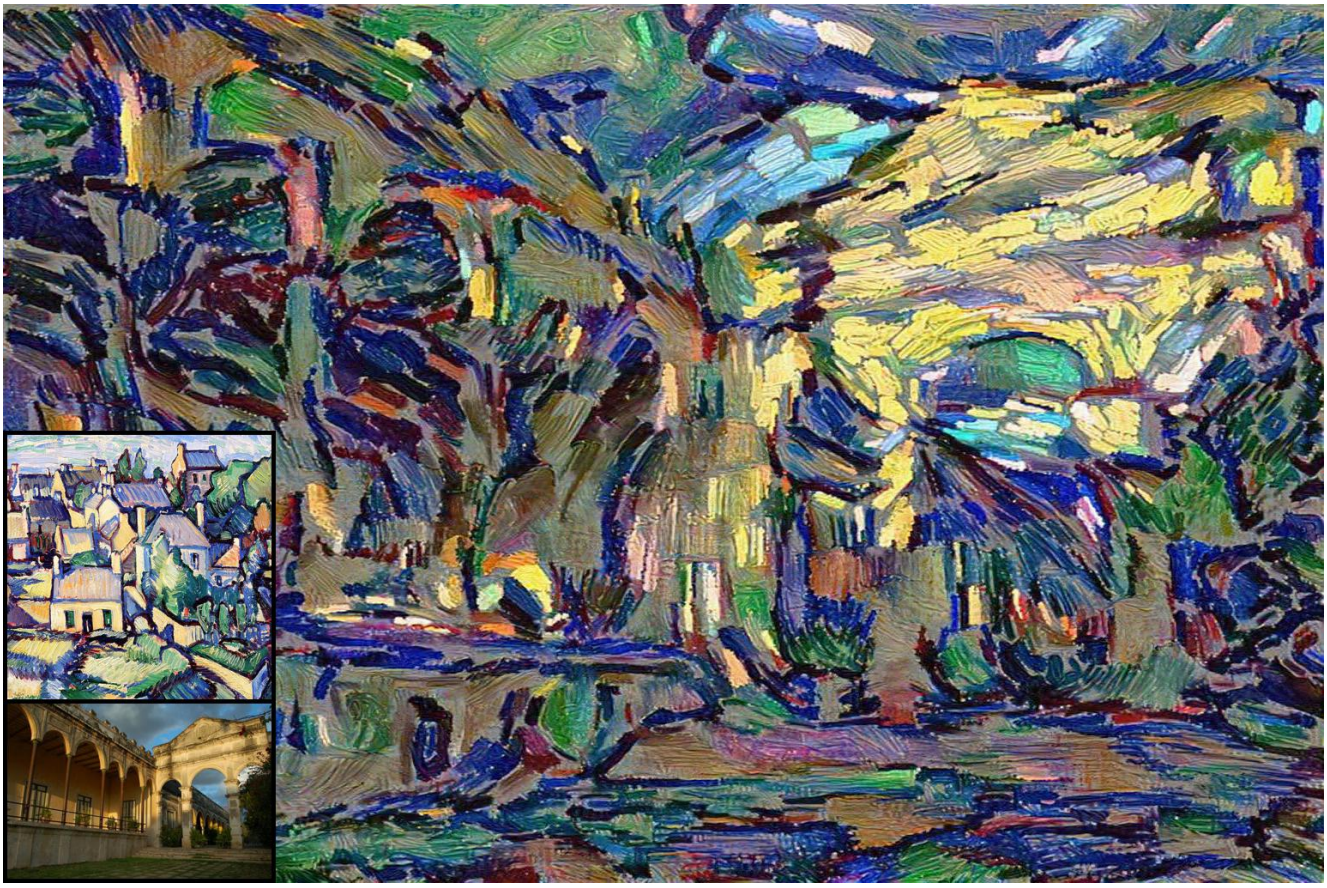


Figure 10. Style image: “Ile De Bréhat” by Samuel John Peploe.

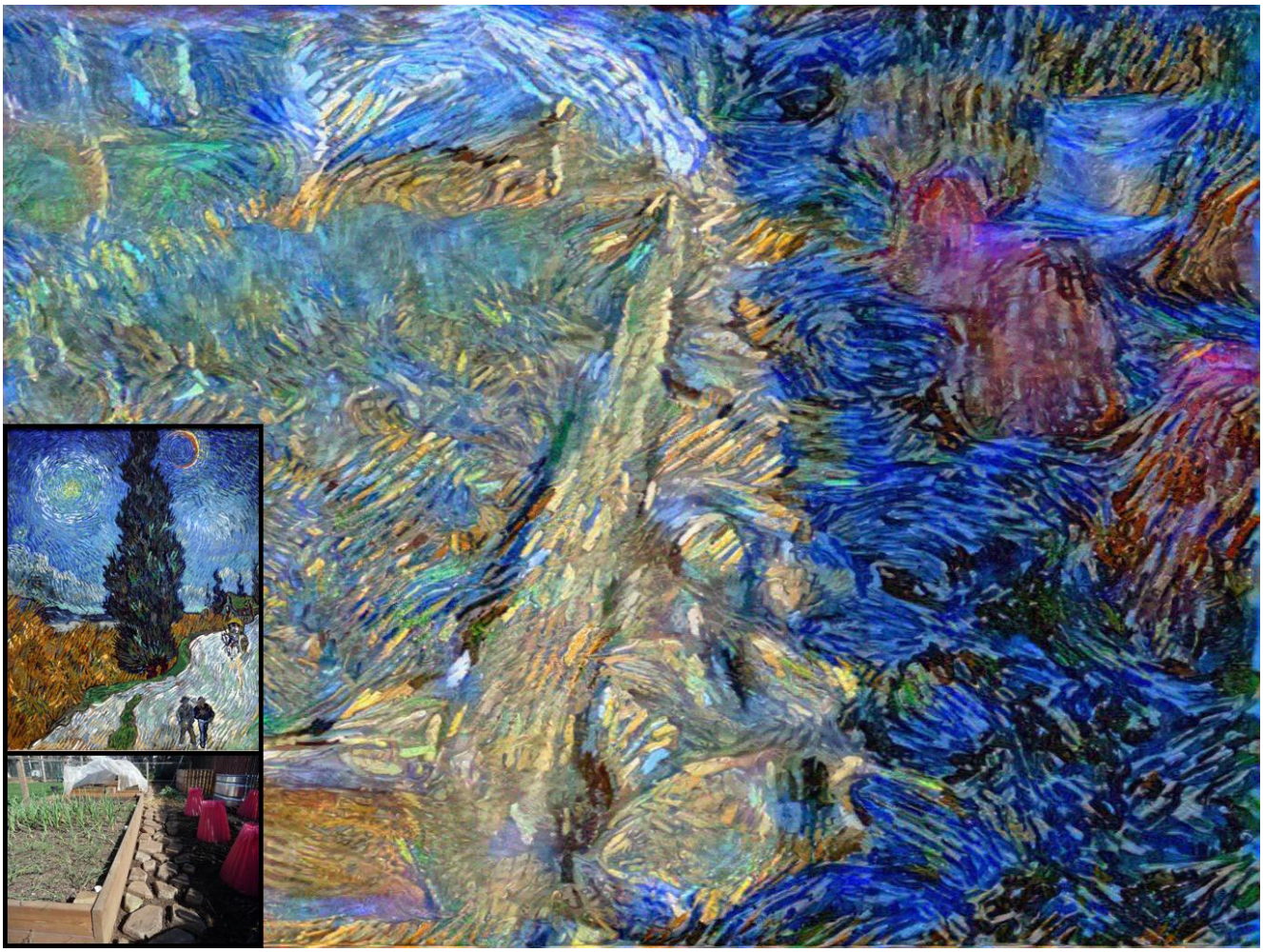


Figure 11. Style image: “Road with Cypress and Star” by Vincent van Gogh.

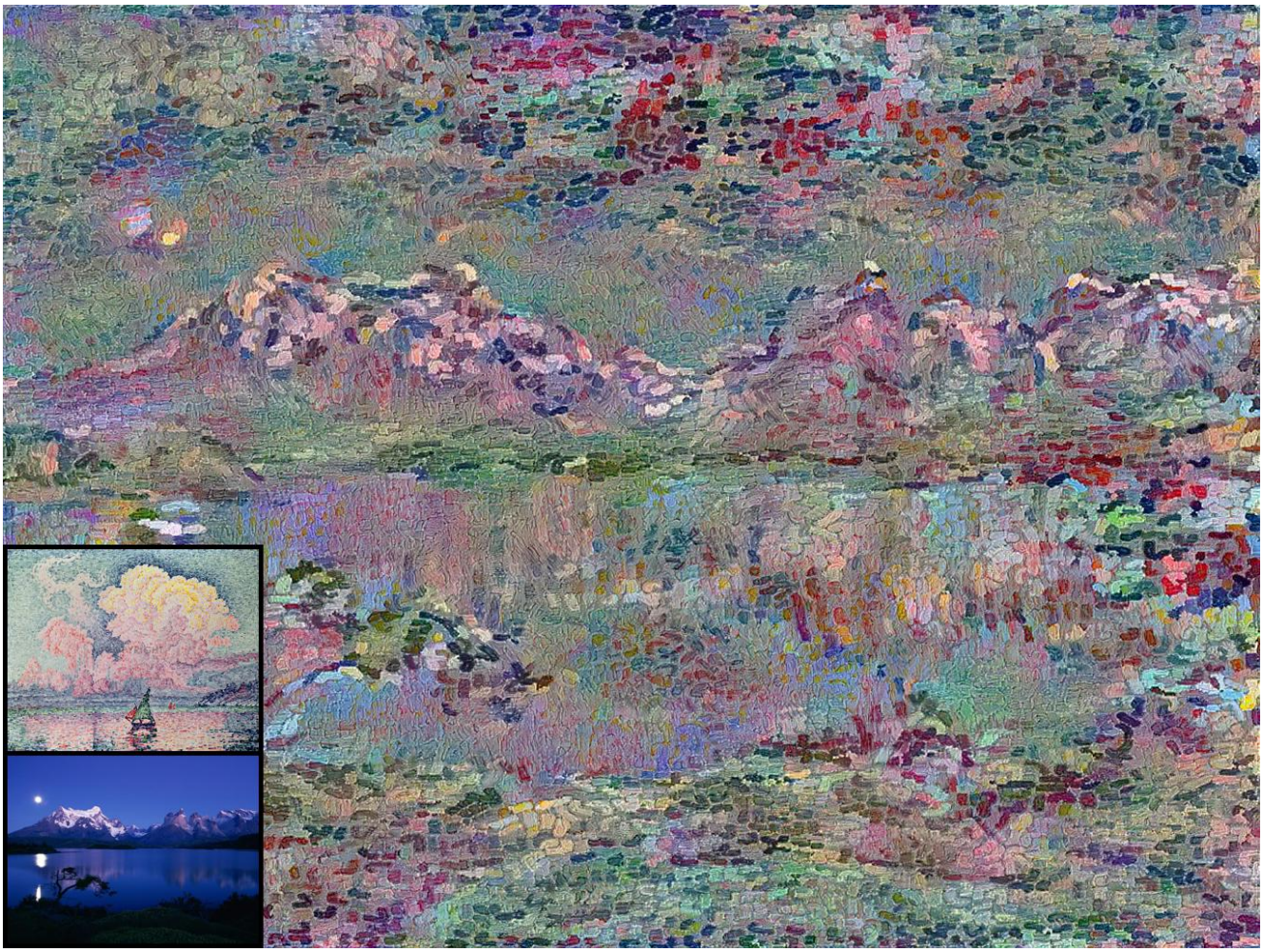


Figure 12. Style image: “Antibes, the Pink Cloud” by Paul Signac.

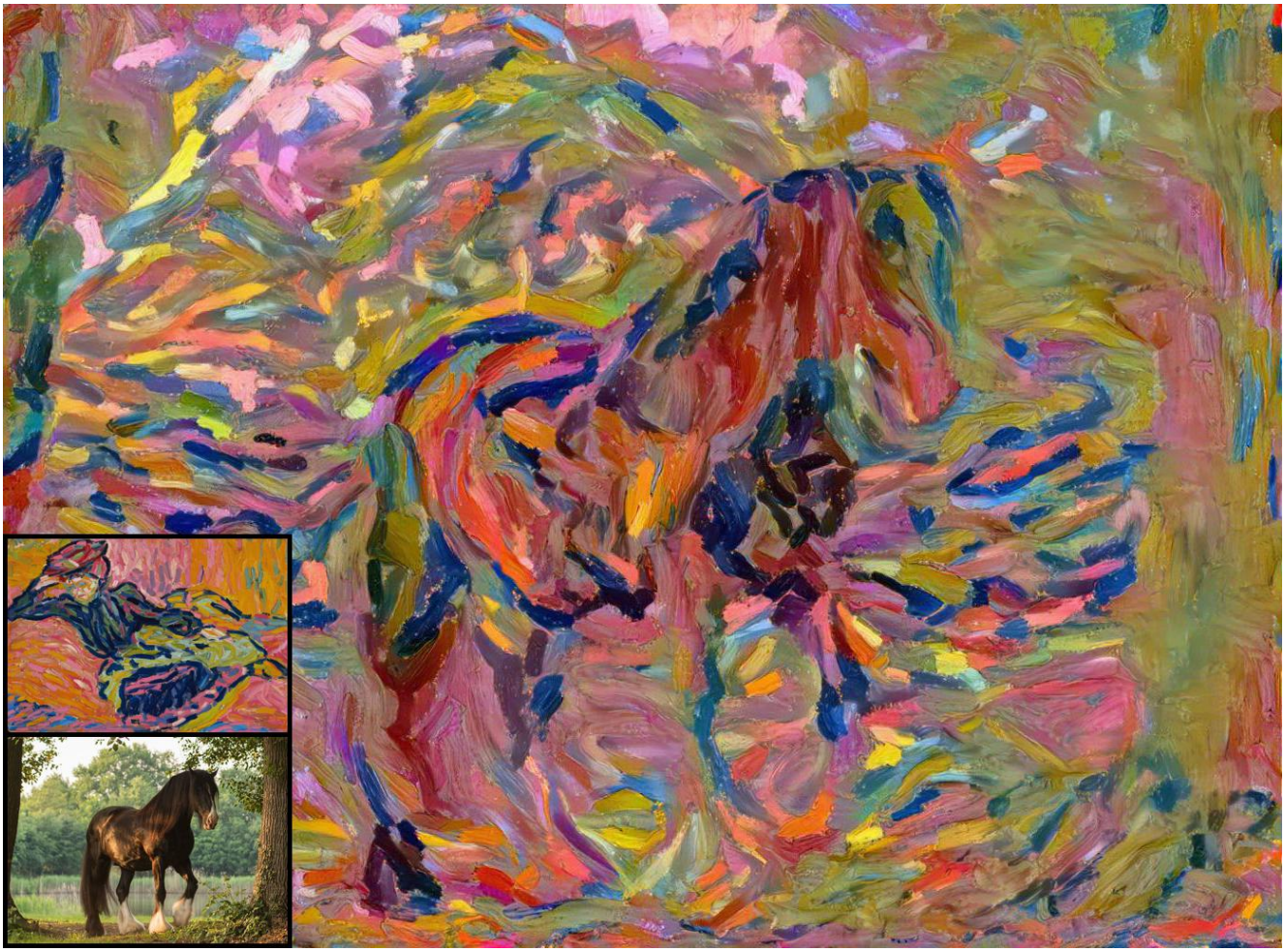


Figure 13. Style image: “Girl on a Divan” by Ernst Ludwig Kirchner.

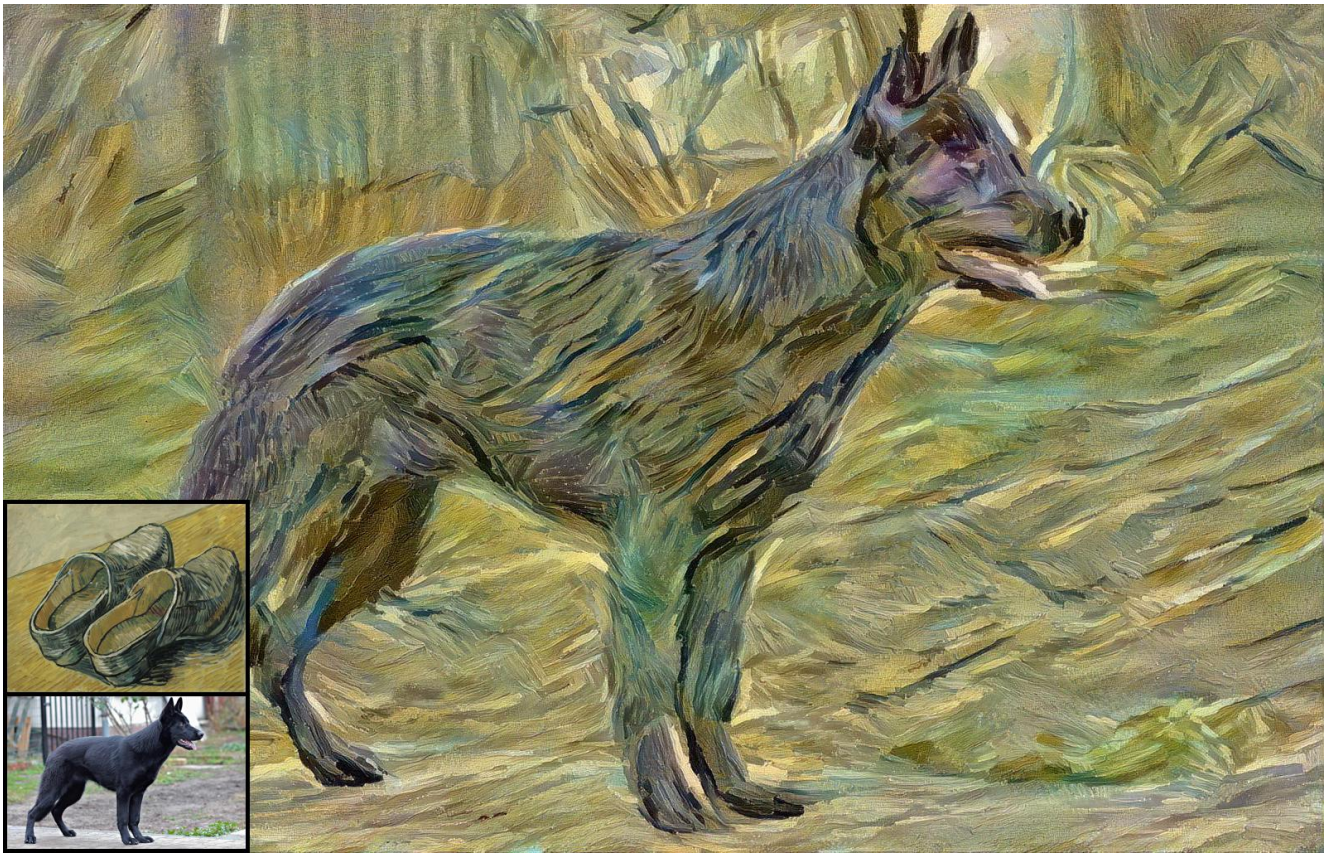


Figure 14. Style image: “A Pair of Leather Clogs” by Vincent van Gogh.

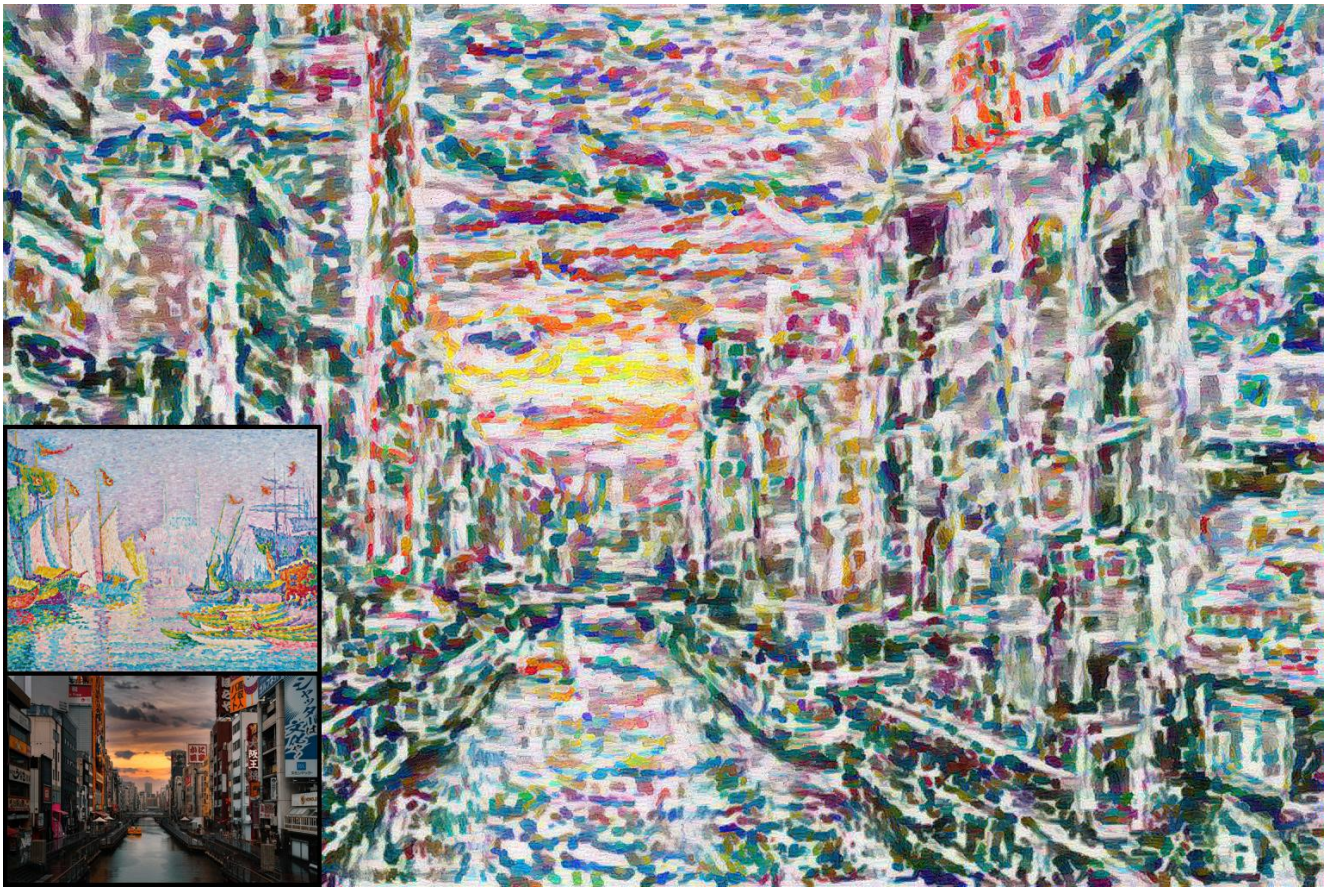


Figure 15. Style image: “La Corne d’Or” by Paul Signac.

Figure 16. Style image: “Self Portrait” by Pablo Picasso.

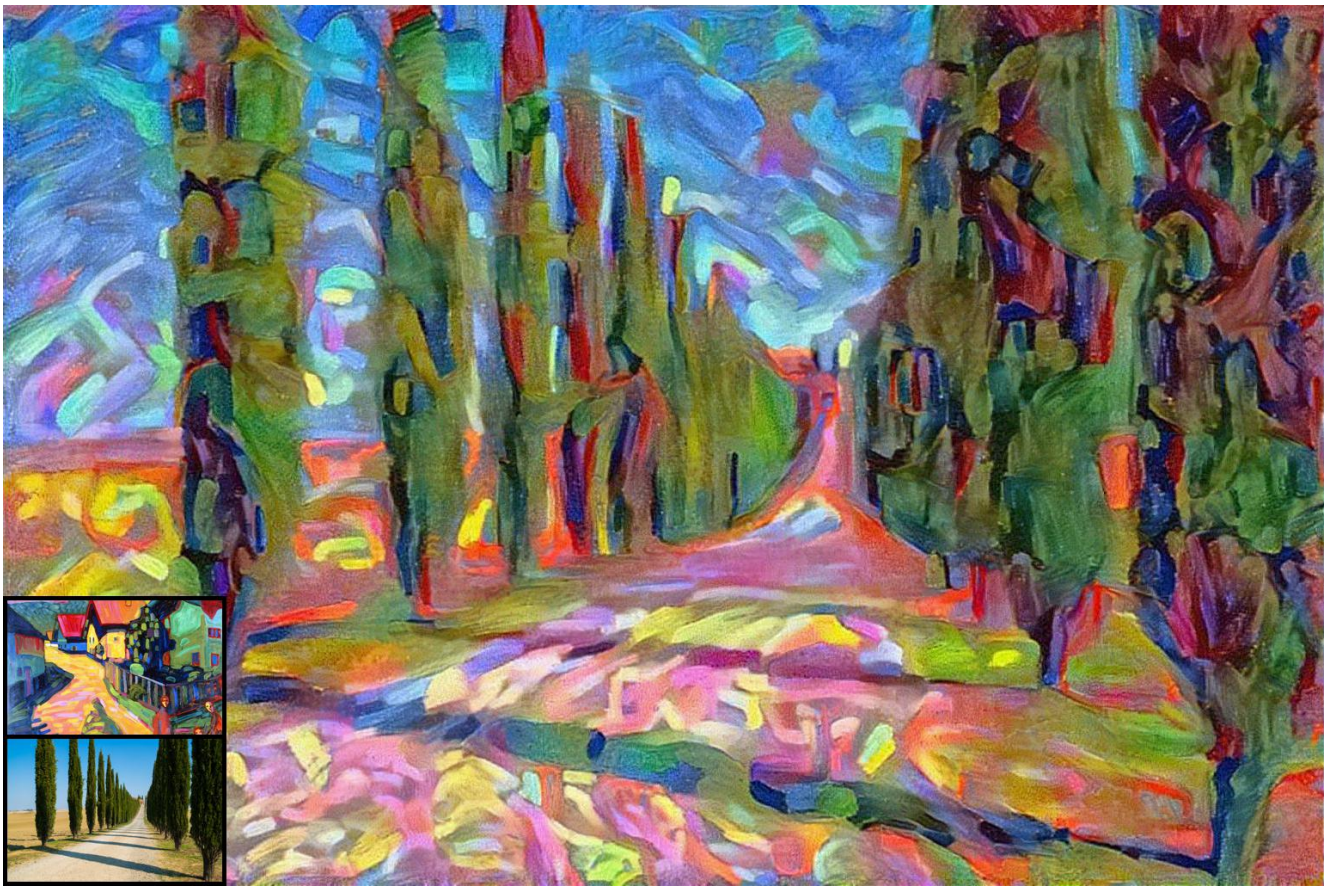


Figure 17. Style image: “Murnau Street With Women” by Wassily Kandinsky.

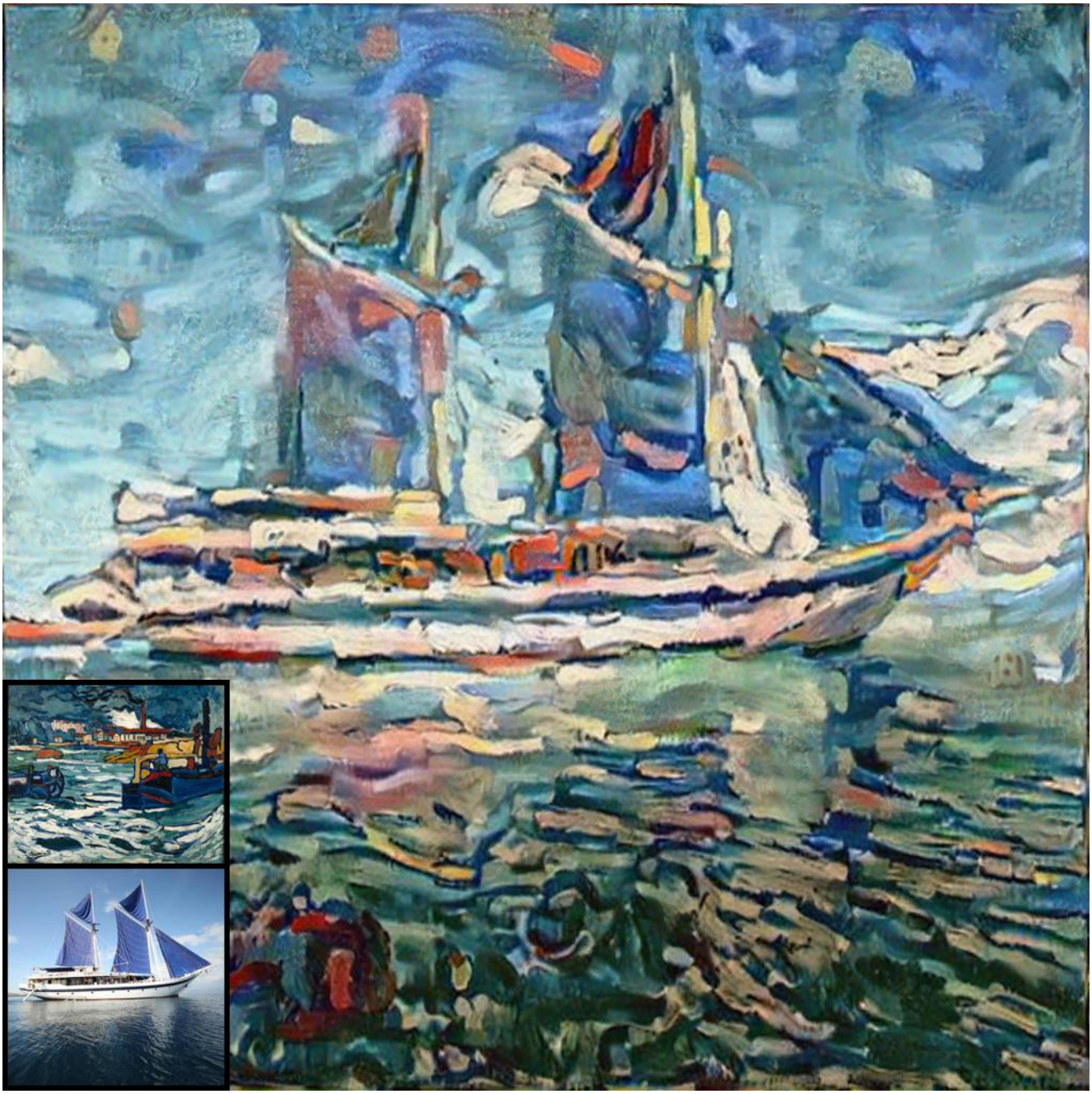


Figure 18. Style image: “Barges on the Seine” by Maurice de Vlaminck.

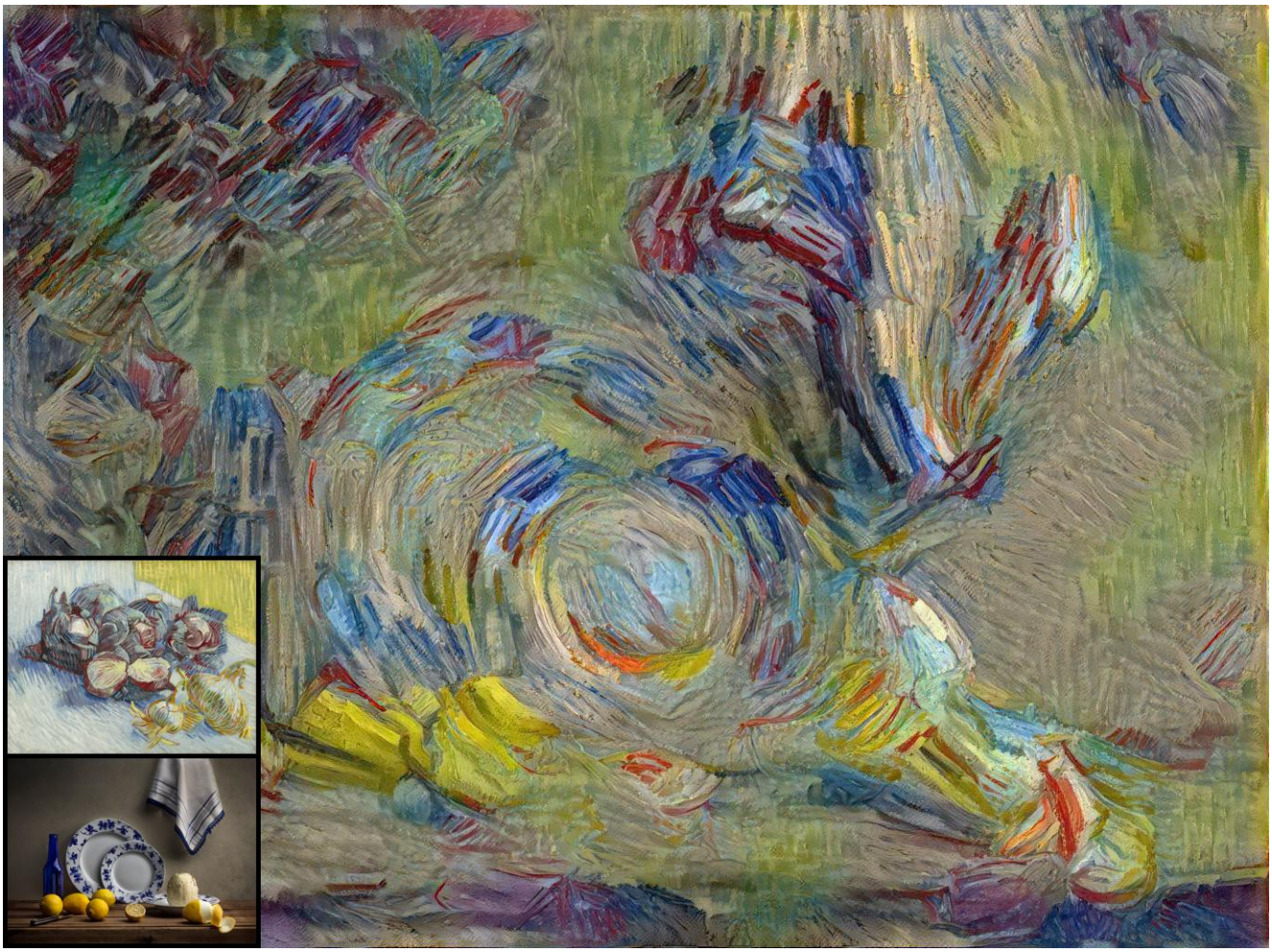


Figure 19. Style image: “Red Cabbages and Onions” by Vincent van Gogh.

Figure 20. Style image: “The Scream” by Edvard Munch.

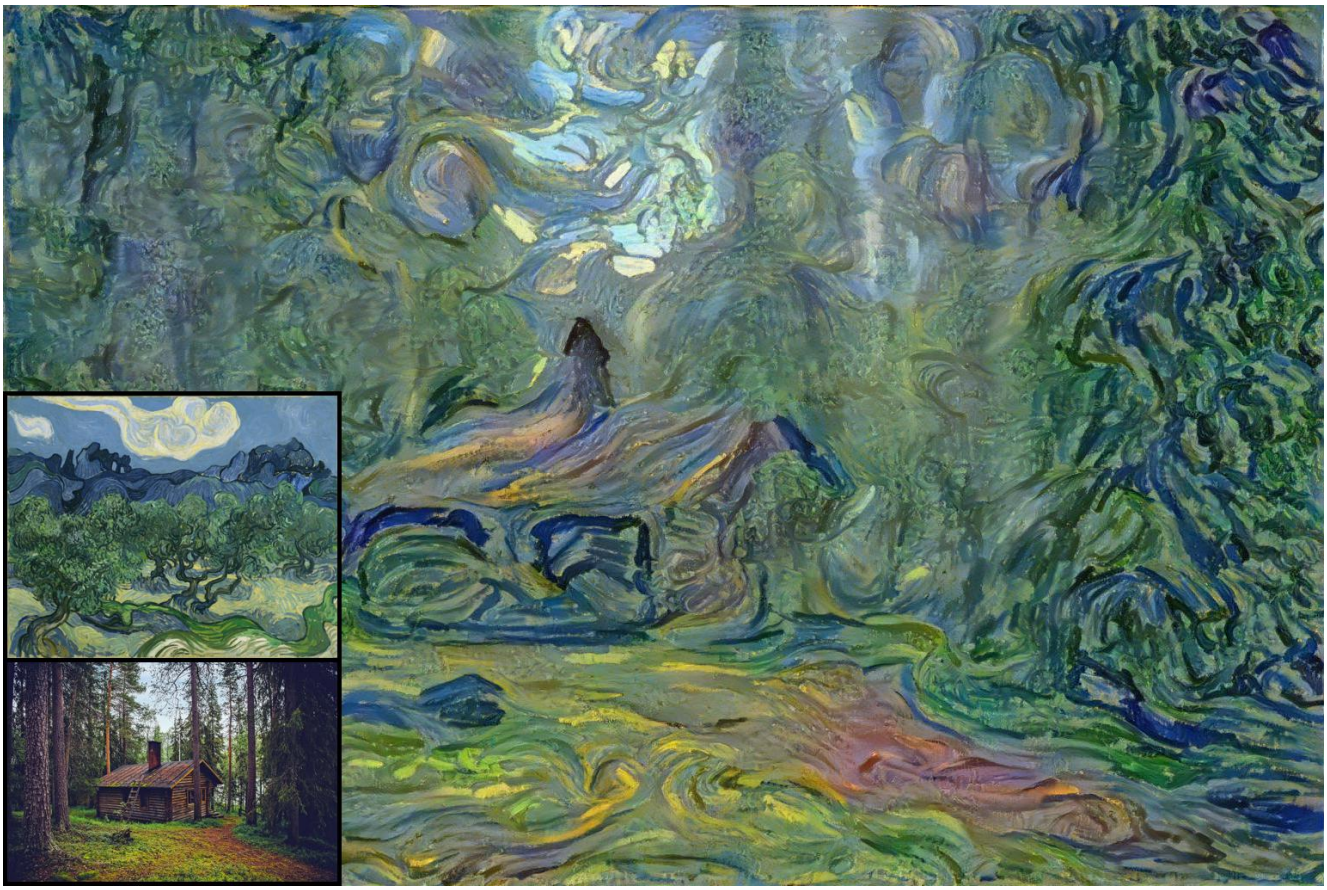


Figure 21. Style image: “The Olive Trees” by Vincent van Gogh.

Figure 22. Style image: “Spring in the Elm Forest” by Edvard Munch.

Figure 23. Style image: “Les Alyscamps” by Vincent van Gogh.

Figure 24. Style image: “The Olive Trees” by Vincent van Gogh.

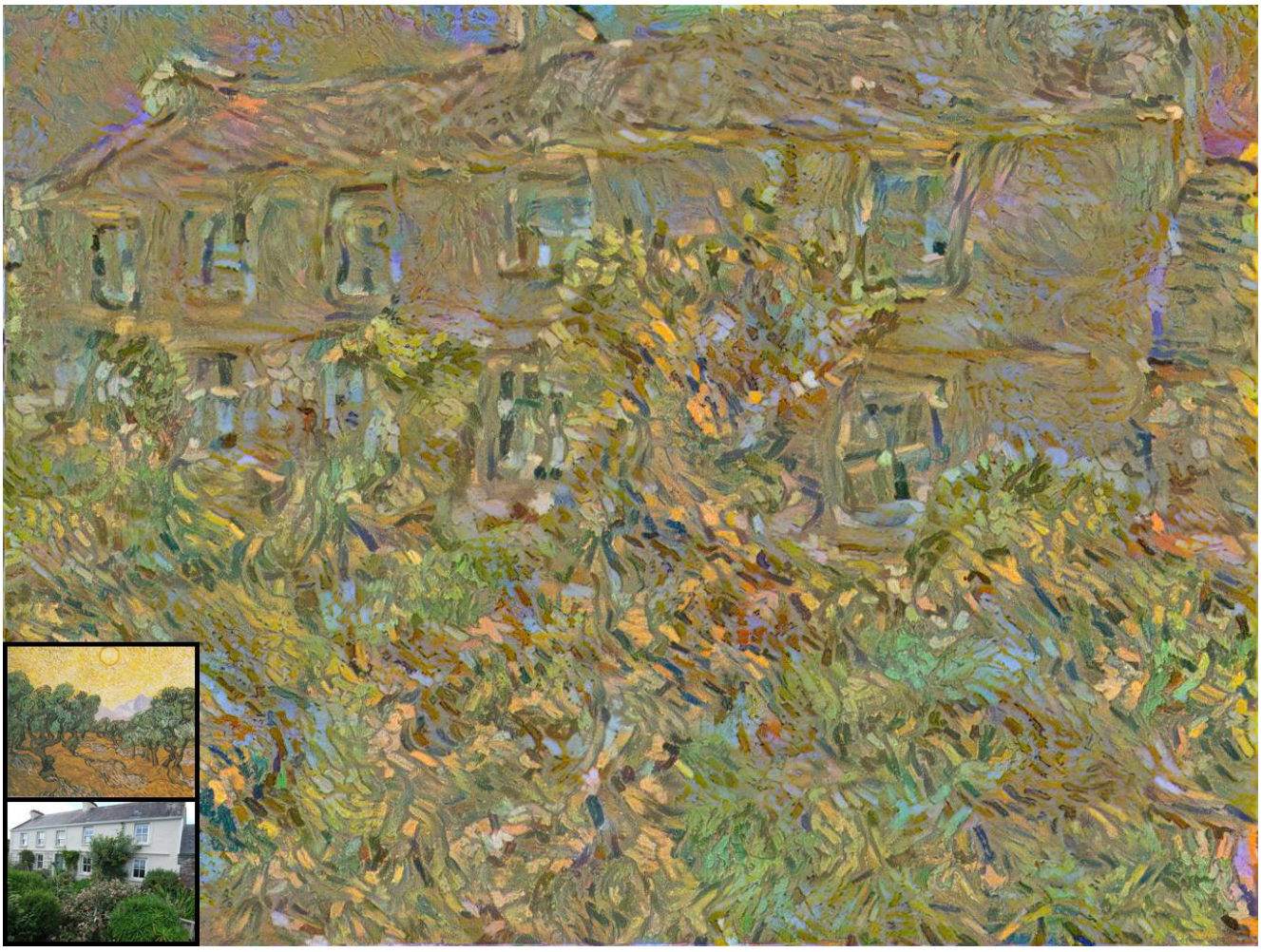


Figure 25. Style image: “Olive Trees with Yellow Sky and Sun” by Vincent van Gogh.

Figure 26. Style image: “Red Cabbages and Onions” by Vincent van Gogh.

Figure 27. Brushstroke approximation of “Starry Night” by Vincent van Gogh using 12.000 brushstrokes.

Figure 28. Brushstroke approximation of “Road with Cypress and Star” by Vincent van Gogh using 12.000 brushstrokes.

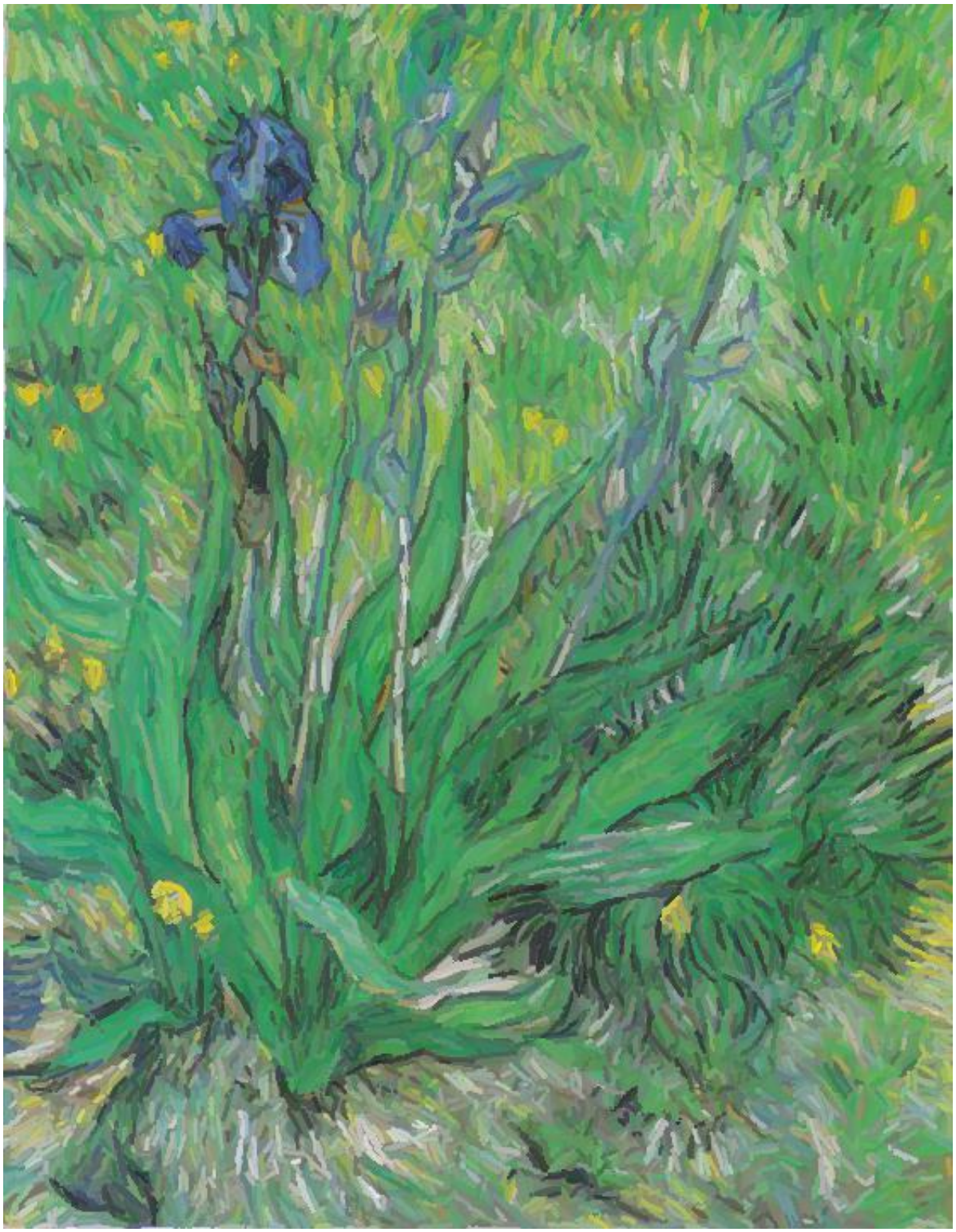


Figure 29. Brushstroke approximation of “Iris” by Vincent van Gogh using 12.000 brushstrokes.

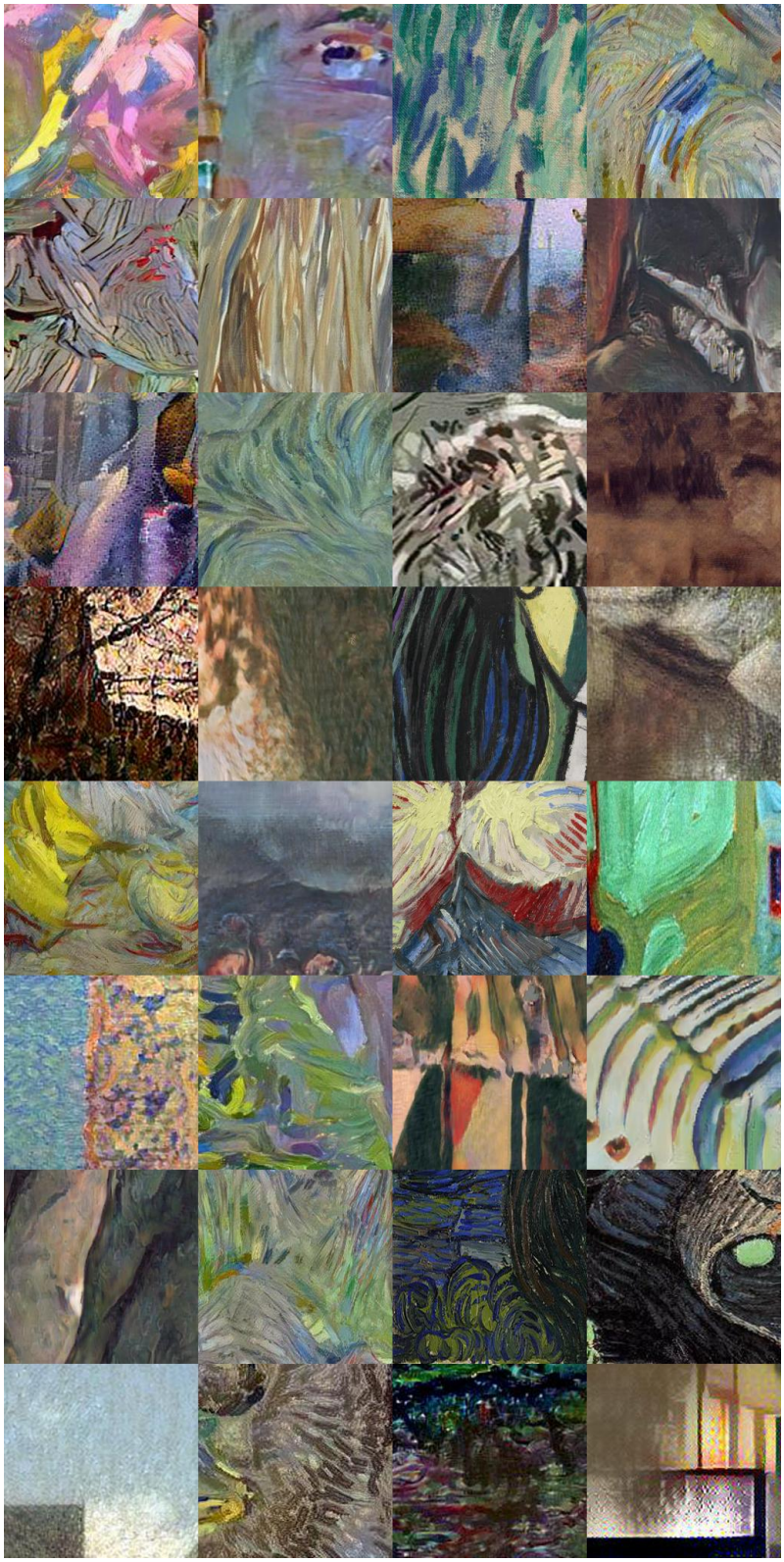


Figure 30. Randomly sampled patches from the user study. In each round we show one row and ask the users to mark all the patches cropped out of real artworks. Each crop in a row is drawn from either a real artwork ('real'), a stylization by Gatys et al. [3] ('Gatys'), a stylization by Sanakoyeu et al. [9] ('AST'), or from our method ('ours'). In this table we have restricted ourselves to only those 4 classes to make this quiz more difficult for the reader. Try to guess which are real. The answers are on the last page.

Solution to Fig. 30:
ours, ours, real, ours
ours, real, Gatys, AST
Gatys, ours, ours, AST
Gatys, AST, real, Gatys
ours, AST, real, real
Gatys, ours, AST, AST
AST, ours, real, real
Gatys, ours, Gatys, Gatys

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282, 2012. 4

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. 2017. 4

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, June 2016. 1, 4, 29

[4] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Adv. Neural Inform. Process. Syst.*, 2017. 4

[5] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Int. Conf. Comput. Vis.*, 2017. 3, 4

[6] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019. 3

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2014. 3, 4

[8] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Adv. Neural Inform. Process. Syst.*, 2017. 4

[9] Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Björn Ommer. A style-aware content loss for real-time hd style transfer. In *Eur. Conf. Comput. Vis.*, 2018. 4, 29