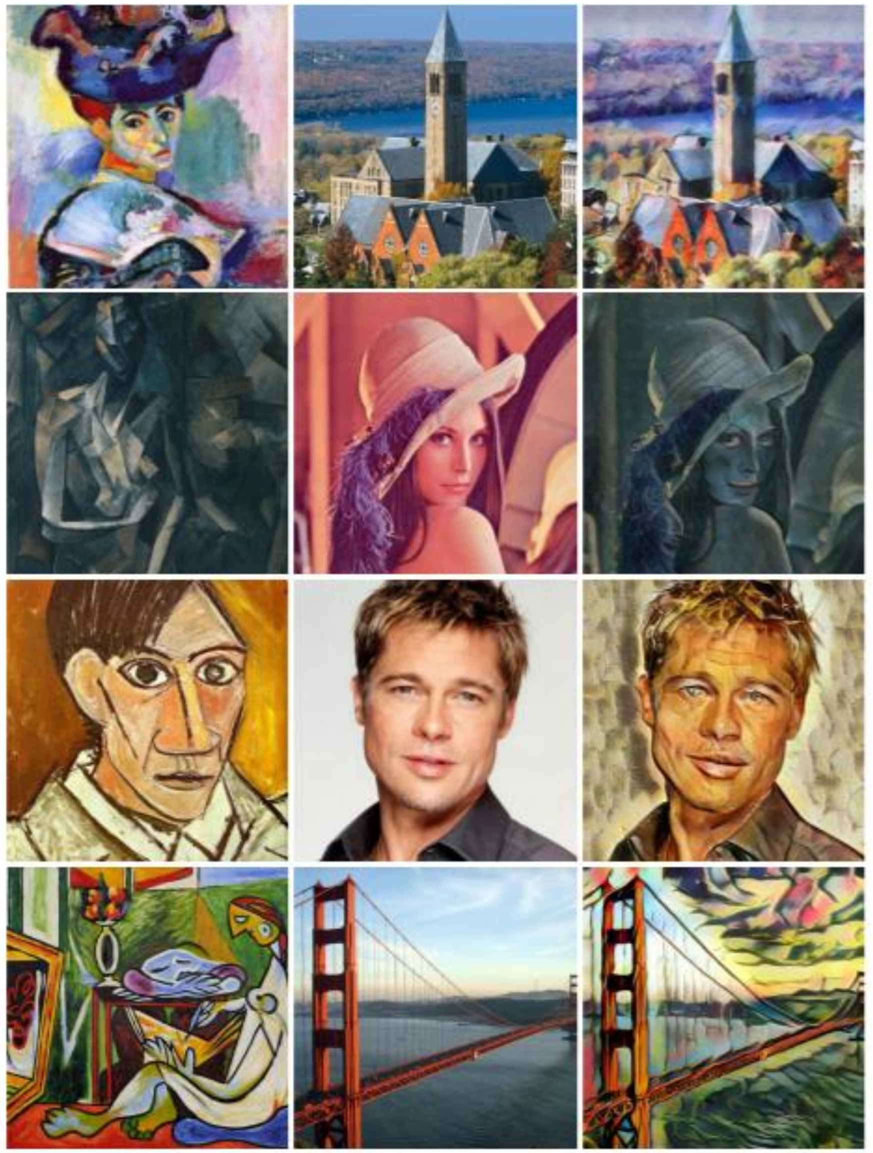
Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

Xun Huang Serge Belongie
Department of Computer Science & Cornell Tech, Cornell University

0. Abstract

Style Transfer



0. Abstract

- For the first time enables "**Arbitrary Style Transfer**" in **real-time**, using **Ada**ptive **I**nstance **N**ormalization(**AdaIN**) layer
- Aligns the mean and variance of the content features with those of the style features
- Using a **single feed-forward neural network**
- **Speedy** and **without restriction** to a pre-defined set of styles
- Flexible user control (Content-Style trade-off, Style interpolation, Color & Spatial controls and so on)

1. Introduction

1. Deep Neural Networks(DNNs) encode both content and style information of an image
2. The image style and content are separable.

➔ Style Transfer!

One dilemma in the Style Transfer Model:

- Flexibility(optimization process) VS. Speed(feed-forward network)

➔**Transfer arbitrary new styles in real-time**

➔Inspired by **Instance Normalization(IN) layer**

1. Introduction

- Flexibility(optimization process) VS. Speed(feed-forward network)

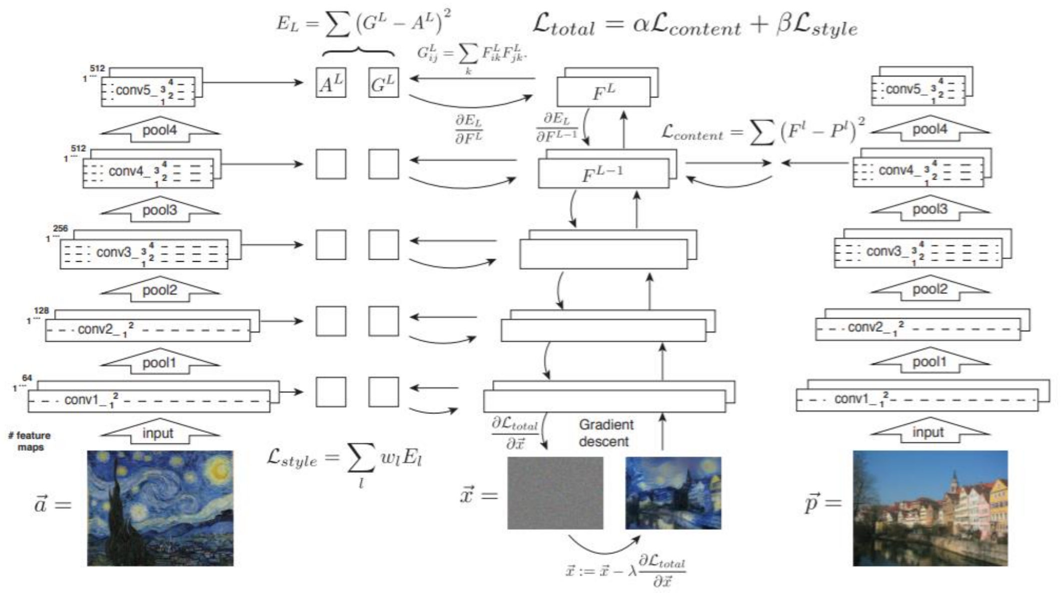

$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
$\mathcal{L}_{style} = \sum_l w_l E_l$
conv5_1
pool4
conv4_1
pool3
conv3_1
pool2
conv2_1
pool1
conv1_1
input
# feature maps
Gradient descent
$\vec{a} =$
$\vec{x} =$
$\vec{p} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$

Figure 2. Style transfer algorithm. First content and style features are extracted and stored. The style image $\vec{a}$ is passed through the network and its style representation A^l on all layers included are computed and stored (left). The content image $\vec{p}$ is passed through the network and the content representation P^l in one layer is stored (right). Then a random white noise image $\vec{x}$ is passed through the network and its style features G^l and content features F^l are computed. On each layer included in the style representation, the element-wise mean squared difference between G^l and A^l is computed to give the style loss $\mathcal{L}_{style}$ (left). Also the mean squared difference between F^l and P^l is computed to give the content loss $\mathcal{L}_{content}$ (right). The total loss $\mathcal{L}_{total}$ is then a linear combination between the content and the style loss. Its derivative with respect to the pixel values can be computed using error back-propagation (middle). This gradient is used to iteratively update the image $\vec{x}$ until it simultaneously matches the style features of the style image $\vec{a}$ and the content features of the content image $\vec{p}$ (middle, bottom).

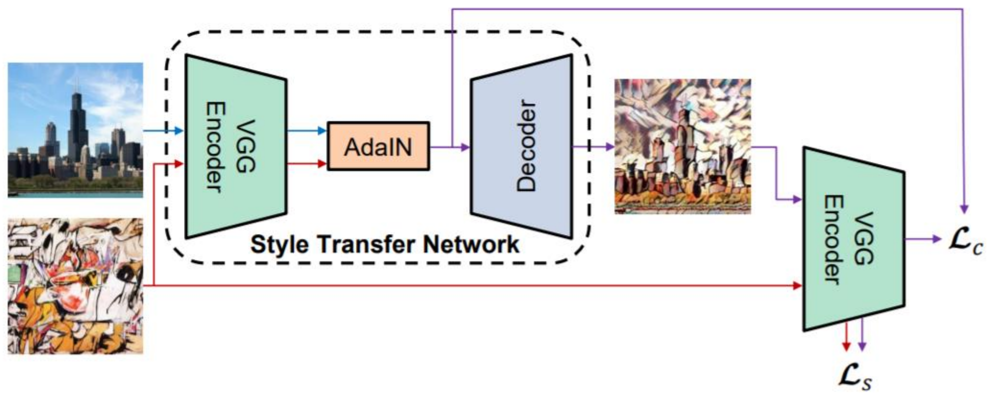


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same VGG encoder to compute a content loss $\mathcal{L}_c$ (Equ. 12) and a style loss $\mathcal{L}_s$ (Equ. 13).

2. Batch Normalization(Skip)

- Ease the training of feed-forward networks by normalizing feature statistics.
- Originally designed to accelerate training of discriminative networks, effective in generative image modeling
- Normalize the mean and standard deviation **for each individual feature channel**: $\gamma, \beta \in R^C$ are affine parameters learned from data

$$\mathrm{BN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c(x))^2 + \epsilon} \qquad \mu_c(x) = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

3. Instance Normalization

- Instance normalization is surprisingly effective in feed-forward style transfer. Why?
- Instance normalization performs a form of ***style normalization*** by **normalizing feature statistics**, namely the mean and variance, which have been found to carry the style information of an image.
- Although DNN serves as an image *descriptor,* we believe that the **feature statistics** of a *generator* network also **control the style** of the generated image.

3. Instance Normalization

- Different from BN layers, here $\mu(x)$ and $\sigma(x)$ are computed across spatial dimensions independently **for each channel and each sample.**
- $\gamma, \beta \in R^c$ are affine parameters learned from data

$$\mathrm{IN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\mu_{nc}(x) = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw}$$

$$\sigma_{nc}(x) = \sqrt{\frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_{nc}(x))^2 + \epsilon}$$

4. Conditional Instance Normalization

- Instead of learning a single set of affine parameters γ and β, a *conditional instance normalization* (CIN) layer that learns a different set of parameters γ^s and β^s **for each style *s***:

$$\mathrm{CIN}(x; s) = \gamma^s \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta^s$$

- During training,

1. a style image together with its index s are randomly chosen from a fixed set of style $s \in \{1, 2, \ldots, S\}$(e.g., $S = 32$) , then

2. The content image is processed by a style transfer network in which the corresponding γ^s and β^s are used in the CIN layers.

+ 2FS additional params ☹

5. Adaptive Instance Normalization(AdaIN)

- A simple extension to IN
- Adaptive Instance normalization **adjusts the mean and variance of the content input to match those of the style input**.
- It effectively combines the content of the former and the style latter by

1. **Transferring feature statistics**. Then,
2. A decoder network is learned to generate the final stylized image by inverting the AdaIN output back to the image space.

5. Adaptive Instance Normalization(AdaIN)

- AdaIN receives a content input x and a style input y, and simply aligns **the channel-wise** mean and variance of x to match those of y.
- **No learnable affine parameters** unlike BN, IN, CIN
- Instead, it **adaptively computes the affine parameters from the style input**
- Adaptive Instance normalization **adjusts the mean and variance of the content input to match those of the style input:**

$$\mathrm{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)$$

6. Style Transfer Network Architecture

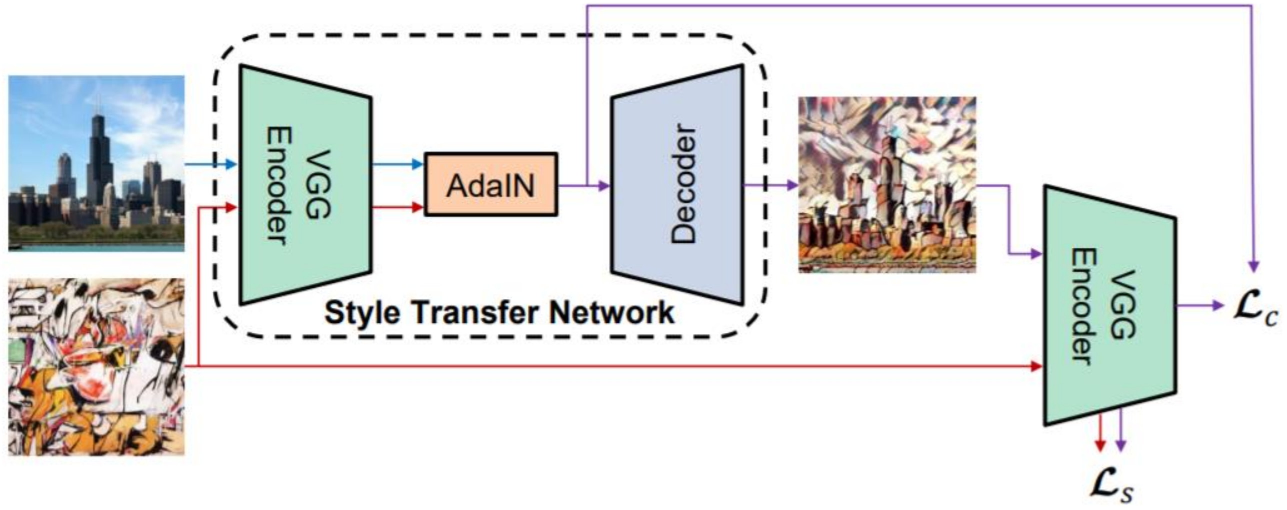


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same VGG encoder to compute a content loss $\mathcal{L}_c$ (Equ. 12) and a style loss $\mathcal{L}_s$ (Equ. 13).

- c : a content image
- s : an arbitrary style image
- f : the encoder
- t : the target feature map
- g : the decoder
- T : the stylized image

$$t = \text{AdaIN}(f(c), f(s))$$

$$T(c, s) = g(t)$$

7. Training

- Content images: MS-COCO, # = 80k
- Style images: a dataset of paintings mostly collected from WikiArt, # = 80k
- Adam optimizer
- A batch size of 8 content-style image pairs.
- Loss function:

$$\mathcal{L} = \mathcal{L}_c + \lambda \mathcal{L}_s$$

$$\mathcal{L}_s = \sum_{i=1}^{L} \|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 \quad + \sum_{i=1}^{L} \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2$$

$$\mathcal{L}_c = \|f(g(t)) - t\|_2$$

8. Results – Qualitive

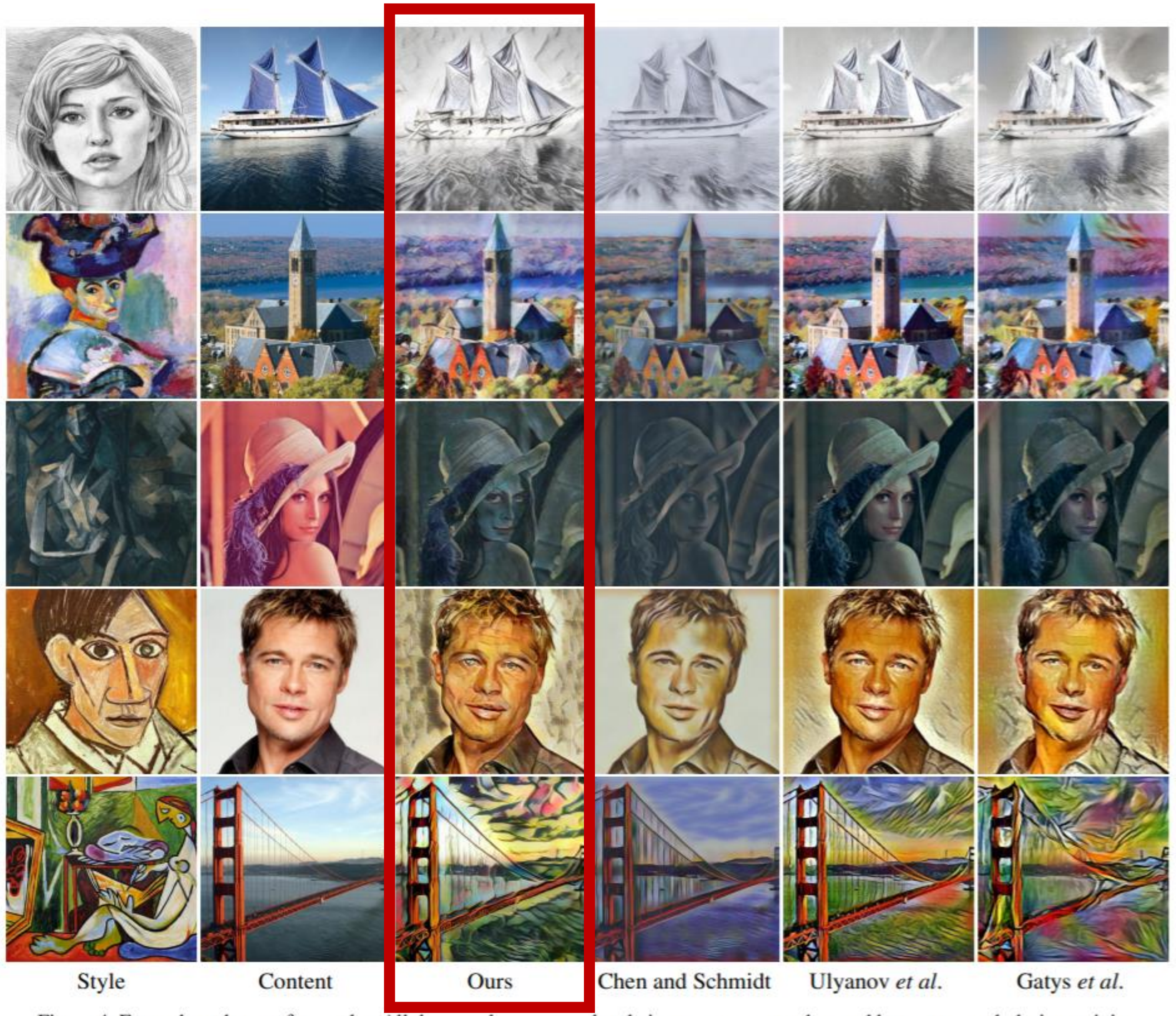
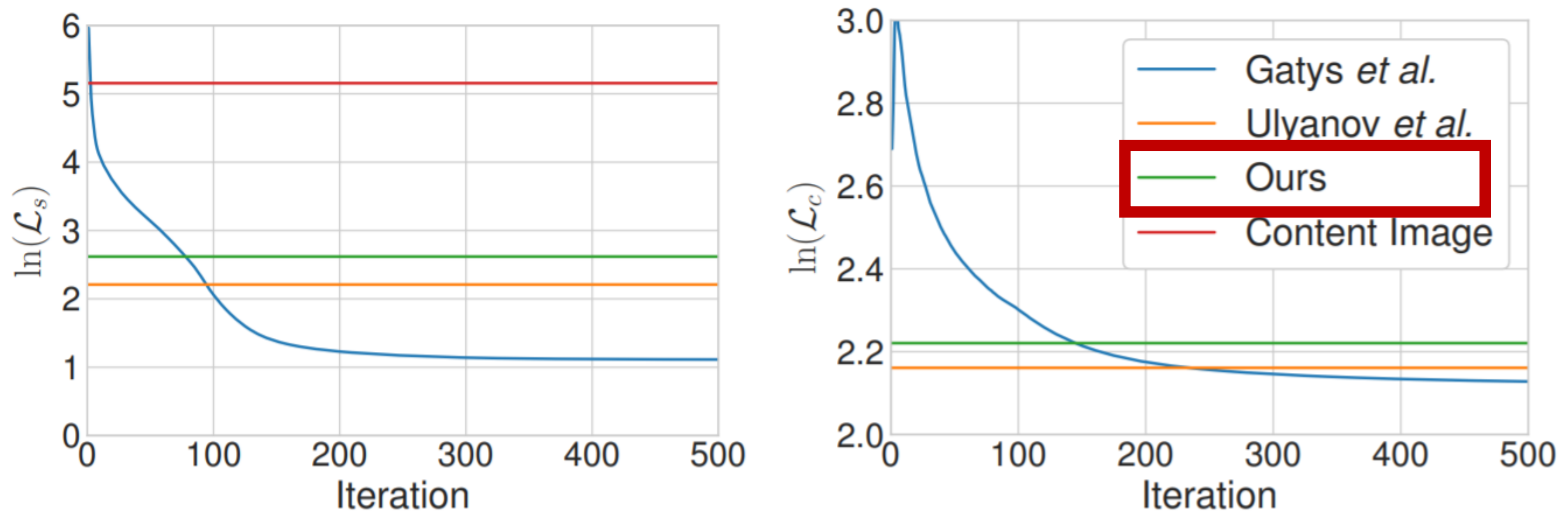


Figure 4. Example style transfer results. All the tested content and style images are never observed by our network during training.

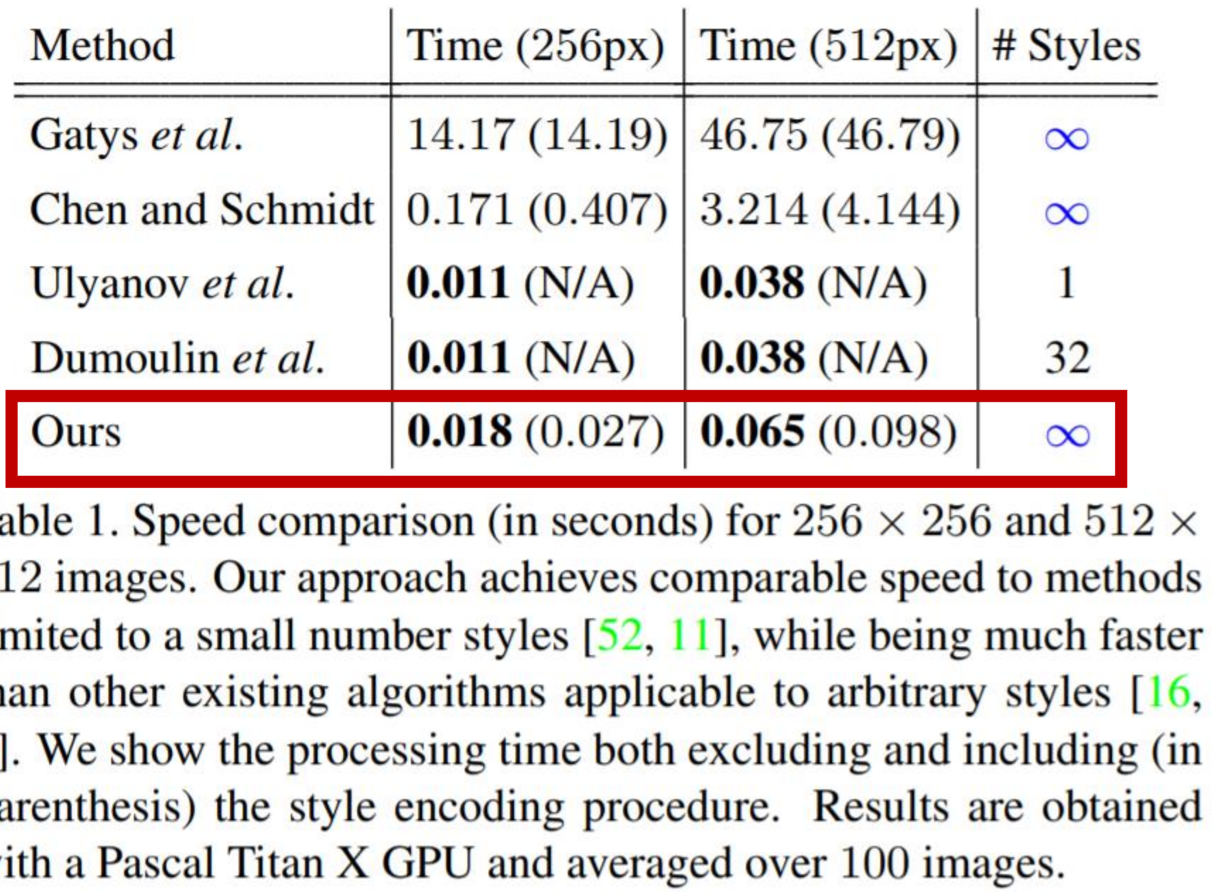
8. Results – Quantitive



(a) Style Loss (b) Content Loss

Figure 3. Quantitative comparison of different methods in terms of style and content loss. Numbers are averaged over 10 style images and 50 content images randomly chosen from our test set.

8. Results – Speed



Method	Time (256px)	Time (512px)	# Styles
Gatys *et al.*	14.17 (14.19)	46.75 (46.79)	∞
Chen and Schmidt	0.171 (0.407)	3.214 (4.144)	∞
Ulyanov *et al.*	**0.011** (N/A)	**0.038** (N/A)	1
Dumoulin *et al.*	**0.011** (N/A)	**0.038** (N/A)	32
Ours	**0.018** (0.027)	**0.065** (0.098)	∞

Table 1. Speed comparison (in seconds) for 256×256 and 512×512 images. Our approach achieves comparable speed to methods limited to a small number styles [52, 11], while being much faster than other existing algorithms applicable to arbitrary styles [16, 6]. We show the processing time both excluding and including (in parenthesis) the style encoding procedure. Results are obtained with a Pascal Titan X GPU and averaged over 100 images.

9. Additional experiments

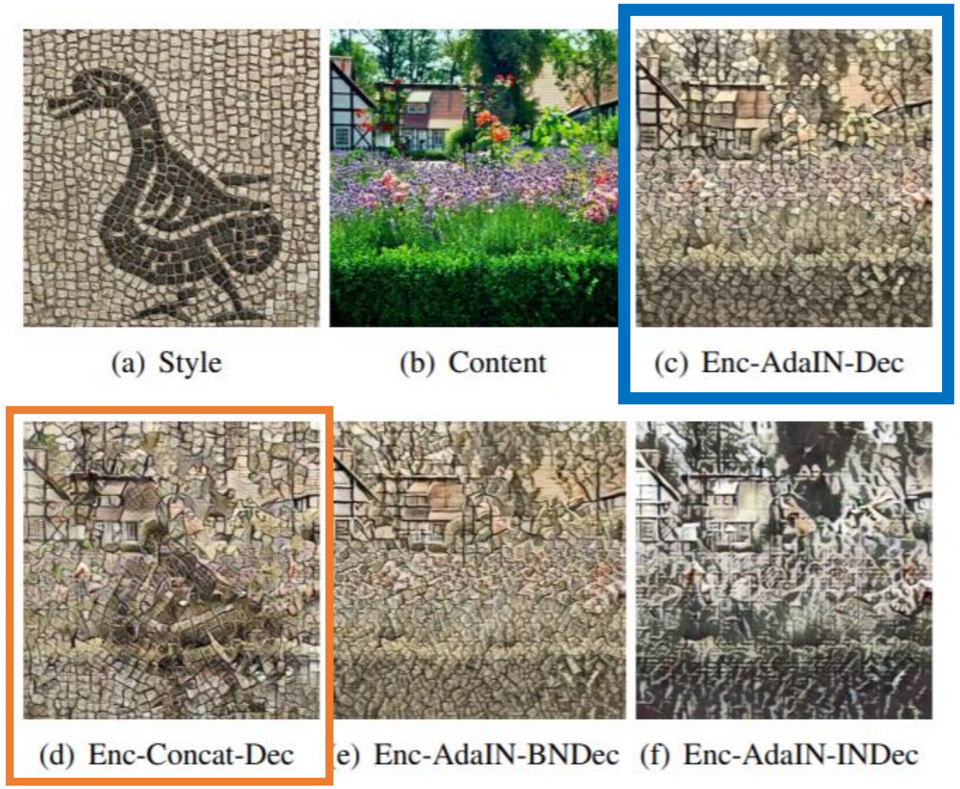


Figure 5. Comparison with baselines. AdaIN is much more effective than concatenation in fusing the content and style information. Also, it is important *not* to use BN or IN layers in the decoder.

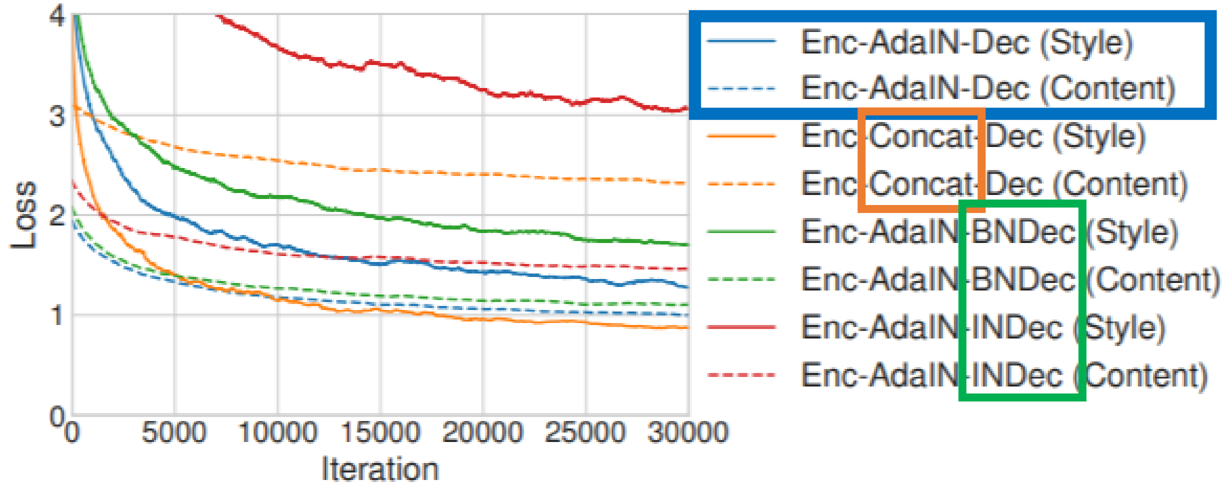


Figure 6. Training curves of style and content loss.

10. Runtime controls

Control the degree of stylization

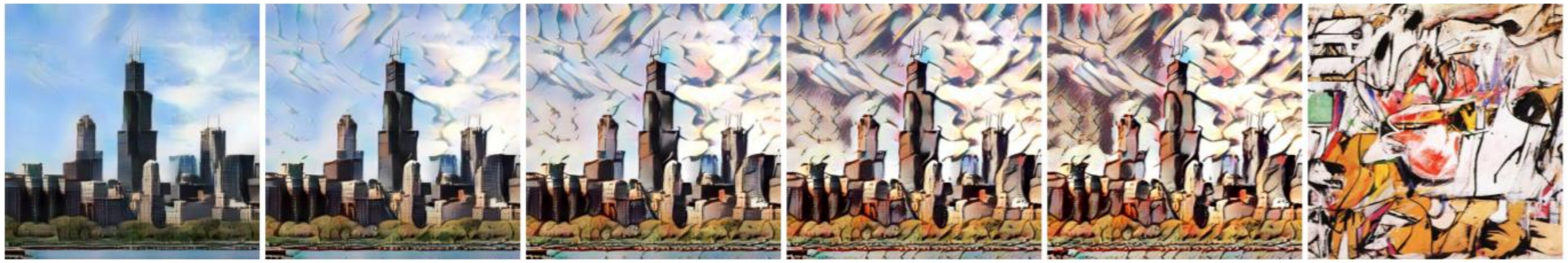


Figure 7. Content-style trade-off. At runtime, we can control the balance between content and style by changing the weight α in Equ. 14.

$$T(c, s, \alpha) = g(\boxed{(1 - \alpha})f(c) + \boxed{\alpha}\mathrm{AdaIN}(f(c), f(s)))$$

10. Runtime controls

Interpolate between different styles (e.g., a set of K style images)

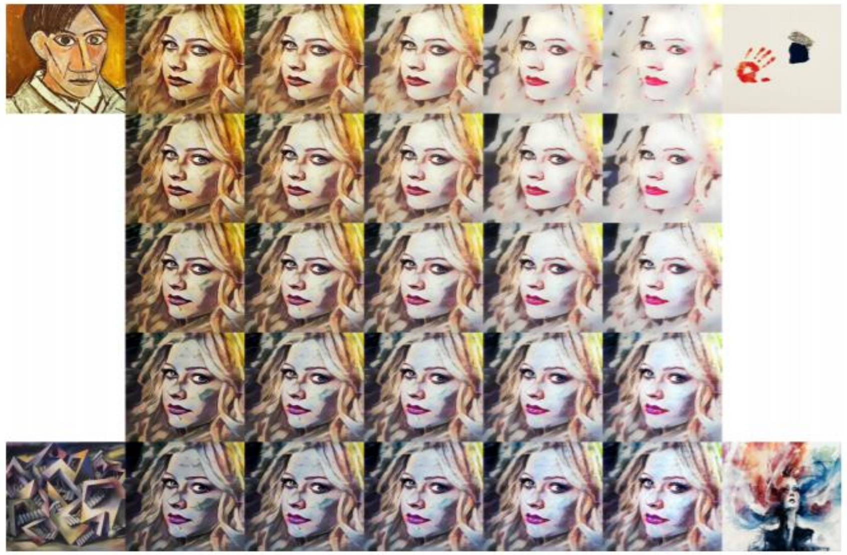


Figure 8. Style interpolation. By feeding the decoder with a convex combination of feature maps transferred to different styles via AdaIN (Equ. 15), we can interpolate between arbitrary new styles.

$$\sum_{k=1}^{K} w_k = 1$$

$$T(c, s_{1,2,\ldots K}, w_{1,2,\ldots K}) = g(\sum_{k=1}^{K} \boxed{w_k} \text{AdaIN}(f(c), f(s_k)))$$

10. Runtime controls

Transfer styles while preserving colors

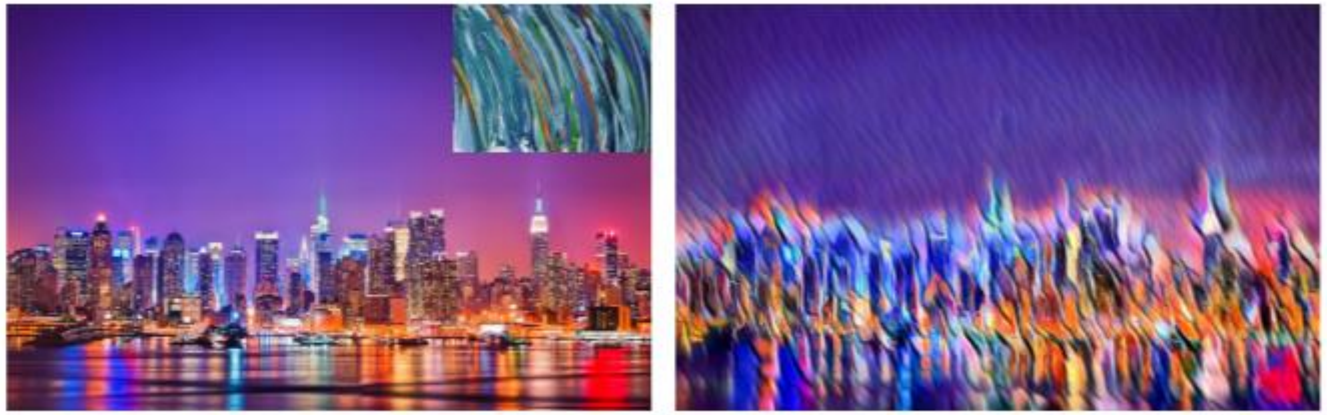


Figure 9. Color control. Left: content and style images. Right: color-preserved style transfer result.

10. Runtime controls

Use different styles in different spatial regions

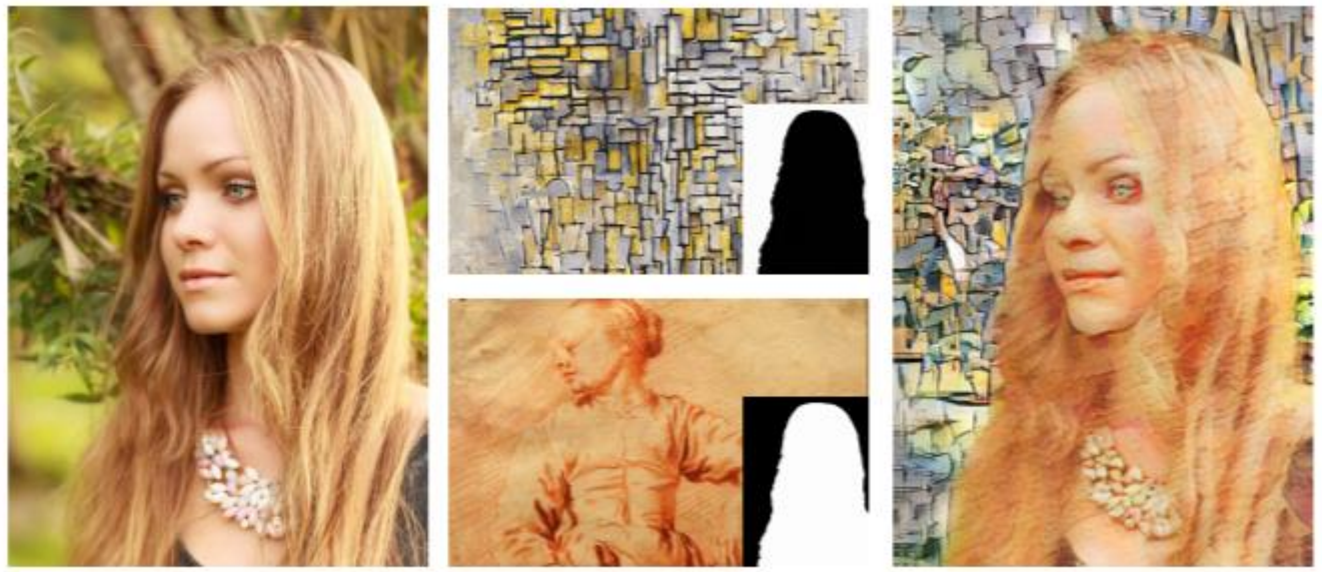


Figure 10. Spatial control. Left: content image. Middle: two style images with corresponding masks. Right: style transfer result.

Thank You ☺