arXiv:1703.06868v1 [cs.CV] 20 Mar 2017

Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

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

Abstract

Gatys et al. recently introduced a neural algorithm that renders a content image in the style of another image, achieving so-called style transfer. However, their framework requires a slow iterative optimization process, which limits its practical application. Fast approximations with feed-forward neural networks have been proposed to speed up neural style transfer. Unfortunately, the speed improvement comes at a cost: the network is usually tied to a fixed set of styles and cannot adapt to arbitrary new styles. In this paper, we present a simple yet effective approach that for the first time enables arbitrary style transfer in real-time. At the heart of our method is a novel adaptive instance normalization (AdaIN) layer that aligns the mean and variance of the content features with those of the style features. Our method achieves speed comparable to the fastest existing approach, without the restriction to a pre-defined set of styles. In addition, our approach allows flexible user controls such as content-style trade-off, style interpolation, color & spatial controls, all using a single feed-forward neural network.

1. Introduction

The seminal work of Gatys *et al*. [15] showed that deep neural networks (DNNs) encode not only the content but also the *style* information of an image. Moreover, the image style and content are somewhat separable: it is possible to change the style of an image while preserving its content. The style transfer method of [15] is flexible enough to combine content and style of arbitrary images. However, it relies on an optimization process that is prohibitively slow.

Significant effort has been devoted to accelerating neural style transfer. Several independent works have attempted to train feed-forward neural networks that perform fast stylization [23, 50, 30]. A major limitation of most feed-forward methods is that each network is restricted to a single style. There are some recent works addressing this problem, but they are either still limited to a finite set of styles [10, 31], or much slower than the single-style transfer methods [5].

In this work, we present the first neural style transfer algorithm that resolves this fundamental flexibility-speed dilemma. Our approach can transfer arbitrary new styles in real-time, combining the flexibility of the optimization-based framework [15] and the speed similar to the fastest feed-forward approaches [23, 51]. Our method is inspired by the *instance normalization* (IN) [51, 10] layer, which is surprisingly effective in feed-forward style transfer. To explain the success of instance normalization, we propose a new interpretation that instance normalization performs style normalization by normalizing feature statistics, which have been found to carry the style information of an image [15, 29, 32]. Motivated by our interpretation, we introduce a simple extension to IN, namely *adaptive instance normalization* (AdaIN). Given a content input and a style input, AdaIN simply adjusts the mean and variance of the content input to match those of the style input. Through experiments, we find AdaIN effectively combines the content of the former and the style latter by transferring feature statistics. A decoder network is then learned to generate the final stylized image by inverting the AdaIN output back to the image space. Our method is nearly three orders of magnitude faster than [15], without sacrificing the flexibility of transferring inputs to arbitrary new styles. Furthermore, our approach provides abundant user controls at runtime, without any modification to the training process.

2. Related Work

Style transfer. The problem of style transfer has its origin from non-photo-realistic rendering [27], and is closely related to texture synthesis and transfer [12, 11, 13]. Some early approaches include histogram matching on linear filter responses [18] and non-parametric sampling [11, 14]. These methods typically rely on low-level statistics and often fail to capture semantic structures. Gatys *et al*. [15] for the first time demonstrated impressive style transfer results by matching feature statistics in convolutional layers of a DNN. Recently, several improvements to [15] have been proposed. Li and Wand [29] introduced a framework based on markov random field (MRF) in the deep feature space to enforce local patterns. Gatys *et al*. [16] proposed ways to control the color preservation, the spatial location, and the scale of style transfer. Ruder *et al*. [44] improved the quality

of video style transfer by imposing temporal constraints.

The framework of Gatys *et al*. [15] is based on a slow optimization process that iteratively updates the image to minimize a content loss and a style loss computed by a loss network. It can take minutes to converge even with modern GPUs. On-device processing in mobile applications is therefore too slow to be practical. A common workaround is to replace the optimization process with a feed-forward neural network that is trained to minimize the same objective [23, 50, 30]. These feed-forward style transfer approaches are about three orders of magnitude faster than the optimization-based alternative, opening the door to real-time applications. Wang *et al*. [52] enhanced the granularity of feed-forward style transfer with a multi-resolution architecture. Ulyanov *et al*. [51] proposed ways to improve the quality and diversity of the generated samples. However, the above feed-forward methods are limited in the sense that each network is tied to a fixed style. To address this problem, Dumoulin *et al*. [10] introduced a single network that is able to encode 32 styles and their interpolations. Concurrent to our work, Li *et al*. [31] proposed a feed-forward architecture that can synthesize up to 300 textures and transfer 16 styles. Still, the two methods above cannot adapt to arbitrary styles that are not observed during training.

Very recently, Chen and Schmidt [5] introduced a feed-forward method that can transfer arbitrary styles thanks to a style swap layer. Given feature activations of the content and style images, the style swap layer replaces the content features with the closest-matching style features in a patch-by-patch manner. Nevertheless, their style swap layer creates a new computational bottleneck: more than 95% of the computation is spent on the style swap for 512×512 input images. Our approach also permits arbitrary style transfer, while being 1-2 orders of magnitude faster than [5].

Another central problem in style transfer is which style loss function to use. The original framework of Gatys *et al*. [15] matches styles by matching the second-order statistics between feature activations, captured by the Gram matrix. Other effective loss functions have been proposed, such as MRF loss [29], adversarial loss [30], histogram loss [53], CORAL loss [40], MMD loss [32], and distance between channel-wise mean and variance [32]. Note that all the above loss functions aim to match some feature statistics between the style image and the synthesized image.

Deep generative image modeling. There are several alternative frameworks for image generation, including variational auto-encoders [26], auto-regressive models [39], and generative adversarial networks (GANs) [17]. Remarkably, GANs have achieved the most impressive visual quality. Various improvements to the GAN framework have been proposed, such as conditional generation [42, 22], multi-stage processing [8, 19], and better training objectives [45, 1]. GANs have also been applied to style transfer [30] and cross-domain image generation [49, 3, 22, 37, 36, 24].

3. Background

3.1. Batch Normalization

The seminal work of Ioffe and Szegedy [21] introduced a batch normalization (BN) layer that significantly ease the training of feed-forward networks by normalizing feature statistics. BN layers are originally designed to accelerate training of discriminative networks, but have also been found effective in generative image modeling [41]. Given an input batch $x \in \mathbb{R}^{N \times C \times H \times W}$, BN normalizes the mean and standard deviation for each individual feature channel:

$$\text{BN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta \tag{1}$$

where $\gamma, \beta \in \mathbb{R}^C$ are affine parameters learned from data; $\mu(x), \sigma(x) \in \mathbb{R}^C$ are the mean and standard deviation, computed across batch size and spatial dimensions independently for each feature channel:

$$\mu_c(x) = \frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \tag{2}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW} \sum_{n=1}^{N} \sum_{h=1}^{H} \sum_{w=1}^{W} (x_{nchw} - \mu_c(x))^2 + \epsilon} \tag{3}$$

BN uses mini-batch statistics during training and replace them with popular statistics during inference, introducing discrepancy between training and inference. Batch renormalization [20] was recently proposed to address this issue by gradually using popular statistics during training. As another interesting application of BN, Li *et al*. [33] found that BN can alleviate domain shifts by recomputing popular statistics in the target domain. Recently, several alternative normalization schemes have been proposed to extend BN's effectiveness to recurrent architectures [34, 2, 46, 7, 28, 43].

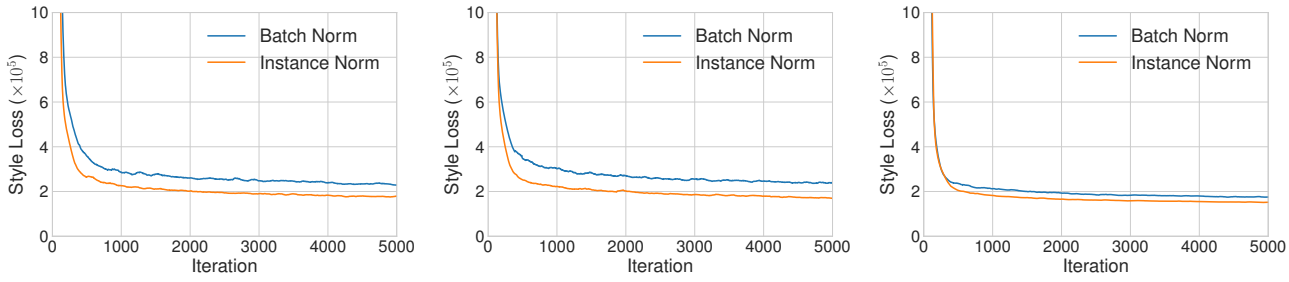
3.2. Instance Normalization

In the original feed-forward stylization method [50], the style transfer network contains a BN layer after each convolutional layer. Surprisingly, Ulyanov *et al*. [51] found that significant improvement could be achieved simply by replacing BN layers with IN layers:

$$\text{IN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta \tag{4}$$

Different from BN layers, here $\mu(x)$ and $\sigma(x)$ are computed across spatial dimensions independently for each channel *and each sample*:

$$\mu_{nc}(x) = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{nchw} \tag{5}$$



(a) Trained with original images. (b) Trained with contrast normalized images. (c) Trained with style normalized images.

Figure 1. To understand the reason for IN's effectiveness in style transfer, we train an IN model and a BN model with (a) original images in MS-COCO [35], (b) contrast normalized images, and (c) style normalized images using a pre-trained style transfer network [23]. The improvement brought by IN remains significant even when all training images are normalized to the same contrast, but are much smaller when all images are (approximately) normalized to the same style. Our results suggest that IN performs a kind of style normalization.

$$\sigma_{nc}(x) = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_{nc}(x))^2 + \epsilon} \quad (6)$$

Another difference is that IN layers are applied at test time unchanged, whereas BN layers usually replace mini-batch statistics with population statistics.

3.3. Conditional Instance Normalization

Instead of learning a single set of affine parameters γ and β, Dumoulin *et al*. [10] proposed a *conditional instance normalization* (CIN) layer that learns a different set of parameters γ^s and β^s for each style s:

$$\text{CIN}(x; s) = \gamma^s \left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta^s \quad (7)$$

During training, a style image together with its index s are randomly chosen from a fixed set of styles $s \in \{1, 2, ..., S\}$ ($S = 32$ in their experiments). The content image is then processed by a style transfer network in which the corresponding γ^s and β^s are used in the CIN layers. Surprisingly, the network can generate images in completely different styles by using the *same* convolutional parameters but *different* affine parameters in IN layers.

Compared with a network without normalization layers, a network with CIN layers requires $2FS$ additional parameters, where F is the total number of feature maps in the network [10]. Since the number of additional parameters scales linearly with the number of styles, it is challenging to extend their method to model a large number of styles (*e.g.*, tens of thousands). Also, their approach cannot adapt to arbitrary new styles without re-training the network.

4. Interpreting Instance Normalization

Despite the great success of (conditional) instance normalization, the reason why they work particularly well for style transfer remains elusive. Ulyanov *et al*. [51] attribute the success of IN to its invariance to the contrast of the content image. However, IN takes place in the feature space, therefore it should have more profound impacts than a simple contrast normalization in the pixel space. Perhaps even more surprising is the fact that the affine parameters in IN can completely change the style of the output image.

It has been known that the convolutional feature statistics of a DNN can capture the style of an image [15, 29, 32]. While Gatys *et al*. [15] use the second-order statistics as their optimization objective, Li *et al*. [32] recently showed that matching many other statistics, including channel-wise mean and variance, are also effective for style transfer. Motivated by these observations, we argue that instance normalization performs a form of *style normalization* by normalizing feature statistics, namely the mean and variance. Although DNN serves as a image *descriptor* in [15, 32], we believe that the feature statistics of a *generator* network can also control the style of the generated image.

We run the code of improved texture networks [51] to perform single-style transfer, with IN or BN layers. As expected, the model with IN converges faster than the BN model (Fig. 1 (a)). To test the explanation in [51], we then normalize all the training images to the same contrast by performing histogram equalization on the luminance channel. As shown in Fig. 1 (b), IN remains effective, suggesting the explanation in [51] to be incomplete. To verify our hypothesis, we normalize all the training images to the same style (different from the target style) using a pre-trained style transfer network provided by [23]. According to Fig. 1 (c), the improvement brought by IN become much smaller when images are already style normalized. The remaining gap can explained by the fact that the style normalization with [23] is not perfect. Also, models with BN trained on style normalized images can converge as fast as models with IN trained on the original images. Our results indicate that IN does perform a kind of style normalization.

Since BN normalizes the feature statistics of a batch of

samples instead of a single sample, it can be intuitively understood as normalizing a batch of samples to be centered around a single style. Each single sample, however, may still have different styles. This is undesirable when we want to transfer all images to the same style, as is the case in the original feed-forward style transfer algorithm [50]. Although the convolutional layers might learn to compensate the intra-batch style difference, it poses additional challenges for training. On the other hand, IN can normalize the style of each individual sample to the target style. Training is facilitated because the rest of the network can focus on content manipulation while discarding the original style information. The reason behind the success of CIN also becomes clear: different affine parameters can normalize the feature statistics to different values, thereby normalizing the output image to different styles.

5. Adaptive Instance Normalization

If IN normalizes the input to a single style specified by the affine parameters, is it possible to adapt it to arbitrarily given styles by using adaptive affine transformations? Here, we propose a simple extension to IN, which we call 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. Unlike BN, IN or CIN, AdaIN has no learnable affine parameters. Instead, it adaptively computes the affine parameters from the style input:

$$\text{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y) \tag{8}$$

in which we simply scale the normalized content input with $\sigma(y)$, and shift it with $\mu(y)$. Similar to IN, these statistics are computed across spatial locations.

Intuitively, let us consider a feature channel that detects brushstrokes of a certain style. A style image with this kind of strokes will produce a high average activation for this feature. The output produced by AdaIN will have the same high average activation for this feature, while preserving the spatial structure of the content image. The brushstroke feature can be inverted to the image space with a feed-forward decoder, similar to [9]. The variance of this feature channel can encoder more subtle style information, which is also transferred to the AdaIN output and the final output image.

In short, AdaIN performs style transfer in the feature space by transferring feature statistics, specifically the channel-wise mean and variance. Our AdaIN layer plays a similar role as the style swap layer proposed in [5]. While the style swap operation is very time-consuming and memory-consuming, our AdaIN layer is as simple as an IN layer, adding almost no computational cost.

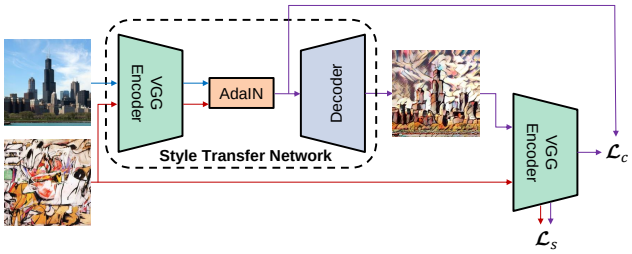


Figure 2. An overview of our style transfer algorithm. We use the first few layers of a fixed VGG-19 network to encode the content and style images. An AdaIN layer is used to perform style transfer in the feature space. A decoder is learned to invert the AdaIN output to the image spaces. We use the same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

6. Experimental Setup

Fig. 2 shows an overview of our style transfer network based on the proposed AdaIN layer. Code and pre-trained models (in Torch 7 [6]) are available at: https://github.com/xunhuang1995/AdaIN-style

6.1. Architecture

Our style transfer network T takes a content image c and an arbitrary style image s as inputs, and synthesizes an output image that recombines the content of the former and the style latter. We adopt a simple encoder-decoder architecture, in which the encoder f is fixed to the first few layers (up to `relu4_1`) of a pre-trained VGG-19 [47]. After encoding the content and style images in feature space, we feed both feature maps to an AdaIN layer that aligns the mean and variance of the content feature maps to those of the style feature maps, producing the target feature maps t:

$$t = \text{AdaIN}(f(c), f(s)) \tag{9}$$

A randomly initialized decoder g is trained to map t back to the image space, generating the stylized image $T(c, s)$:

$$T(c, s) = g(t) \tag{10}$$

The decoder mostly mirrors the encoder, with all pooling layers replaced by nearest up-sampling to reduce checkerboard effects. We use reflection padding in both f and g to avoid border artifacts. Another important architectural choice is whether the decoder should use instance, batch, or no normalization layers. As discussed in Sec. 4, IN normalizes each sample to a single style while BN normalizes a batch of samples to be centered around a single style. Both are undesirable when we want the decoder to generate images in vastly different styles. Thus, we do *not* use normalization layers in the decoder. In Sec. 7.1 we will show that IN/BN layers in the decoder indeed hurt performance.

6.2. Training

We train our network using MS-COCO [35] as content images and a dataset of paintings mostly collected from WikiArt [38] as style images, following the setting of [5]. Each dataset contains roughly $80,000$ training examples. We use the adam optimizer [25] and a batch size of 8 content-style image pairs. During training, we first resize the smallest dimension of both images to 512 while preserving the aspect ratio, then randomly crop regions of size 256×256. Since our network is fully convolutional, it can be applied to images of any size during testing.

Similar to [50, 10, 51], we use the pre-trained VGG-19 [47] to compute the loss function to train the decoder:

$$\mathcal{L} = \mathcal{L}_c + \lambda \mathcal{L}_s \tag{11}$$

which is a weighted combination of the content loss $\mathcal{L}_c$ and the style loss $\mathcal{L}_s$ with the style loss weight λ. The content loss is the Euclidean distance between the target features and the features of the output image. We use the AdaIN output t as the content target, instead of the commonly used feature responses of the content image. We find this leads to slightly faster convergence and also aligns with our goal of inverting the AdaIN output t.

$$\mathcal{L}_c = \|f(g(t)) - t\|_2 \tag{12}$$

Since our AdaIN layer only transfers the mean and standard deviation of the style features, our style loss only matches these statistics. Although we find the commonly used Gram matrix loss can produce similar results, we match the IN statistics because it is conceptually cleaner. This style loss has also been explored by Li *et al*. [32].

$$\mathcal{L}_s = \sum_{i=1}^{L} \|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 \quad + \\ \sum_{i=1}^{L} \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2 \tag{13}$$

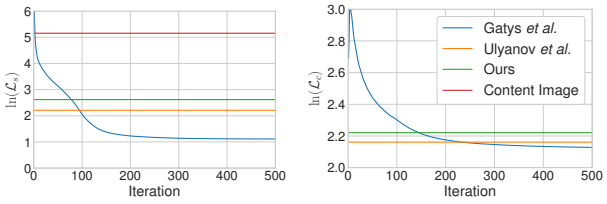
where each ϕ_i denotes a layer in VGG-19 used to compute the style loss. In our experiments we use `relu1_1`, `relu2_1`, `relu3_1`, `relu4_1` layers with equal weights.

7. Results

7.1. Comparison with other methods

In this subsection, we compare our approach with three types of style transfer methods: 1) the flexible but slow optimization-based method [15], 2) the fast feed-forward method restricted to a single style [51], and 3) the flexible patch-based method of medium speed [5]. If not mentioned otherwise, the results of compared methods are obtained by running their code with the default configurations. [1] For

[1] We run 500 iterations of [15] using Johnson's public implementation: https://github.com/jcjohnson/neural-style



(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.

[5], we use a pre-trained inverse network provided by the authors. All the test images are of size 512×512.

Qualitative Examples. In Fig. 4 we show example style transfer results generated by compared methods. Note that all the test style images are *never* observed during the training of our model, while the results of [51] are obtained by fitting one network to each test style. Even so, the quality of our stylized images is quite competitive with [51] and [15] for many images (*e.g.*, row $1, 2, 3$). In some other cases (*e.g.*, row 5) our method is slightly behind the quality of [51] and [15]. This is not unexpected, as we believe there is a three-way trade-off between speed, flexibility, and quality. Compared with [5], our method appears to transfer the style more faithfully for most compared images. The last example clearly illustrates a major limitation of [5], which attempts to match each content patch with the closest-matching style patch. However, if most content patches are matched a few style patches that are not representative of the target style, the style transfer would fail. We thus argue that matching global feature statistics is a more general solution, although in some cases (*e.g.*, row 3) the method of [5] can also produce appealing results.

Quantitative evaluations. Does our algorithm trade off some quality for higher speed and flexibility, and if so by how much? To answer this question quantitatively, we compare our approach with the optimization-based method [15] and the fast single-style transfer method [51] in terms of the content and style loss. Because our method uses a style loss based on IN statistics, we also modify the loss function in [15] and [51] accordingly for a fair comparison (their results in Fig. 4 are still obtained with the default Gram matrix loss). The content loss shown here is the same as in [51, 15]. The numbers reported are averaged over 10 style images and 50 content images randomly chosen from the test set of the WikiArt dataset [38] and MS-COCO [35].

As shown in Fig. 3, the average content and style loss of our synthesized images are slightly higher but comparable to the single-style transfer method of Ulyanov *et al*. [51]. In particular, both our method and [51] obtain a style loss similar to that of [15] between 50 and 100 iterations of optimiza-

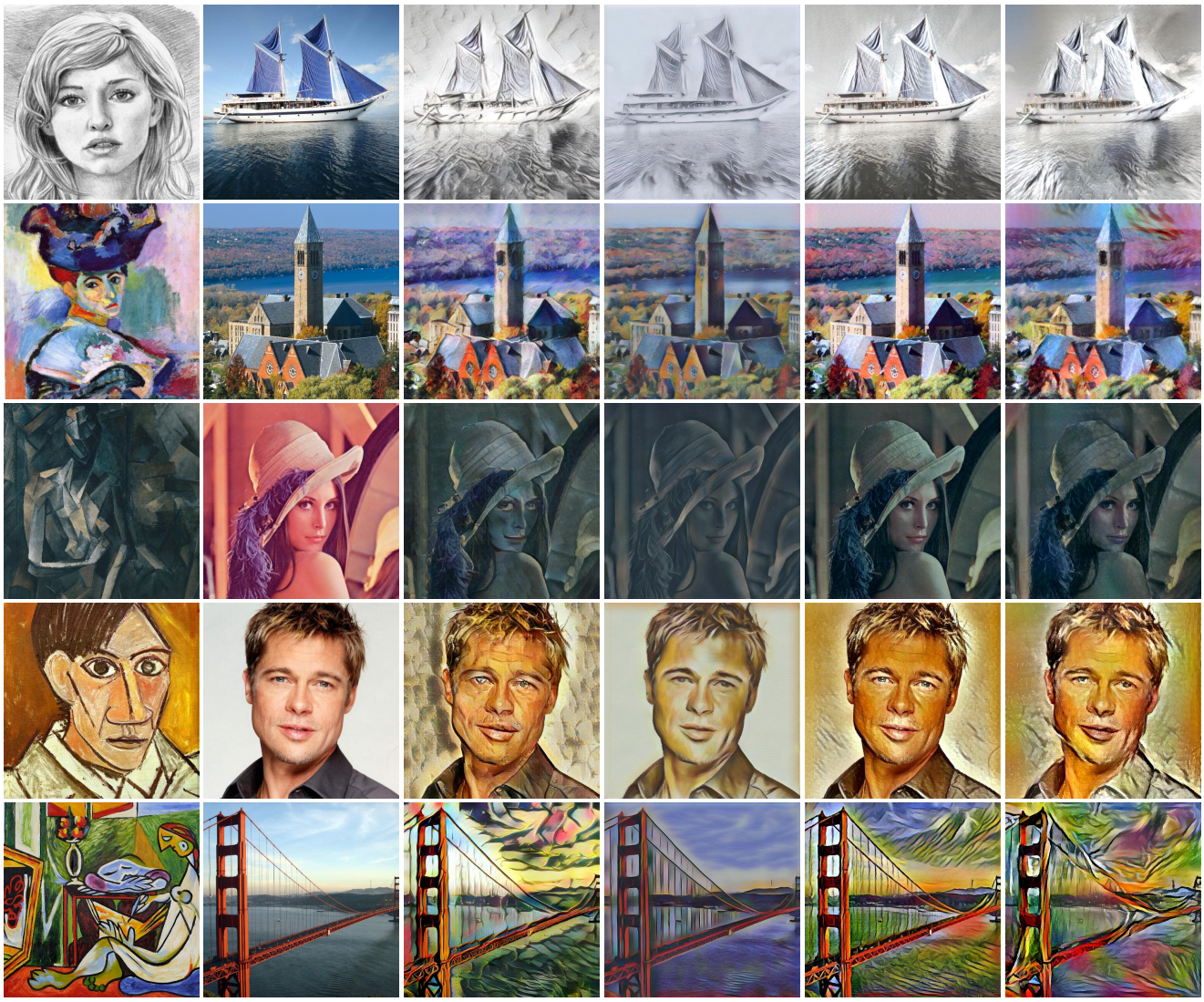


Figure 4. Example style transfer results. All the tested content and style images are never observed by our network during training.

tion. This demonstrates the strong generalization ability of our approach, considering that our network has never seen the test styles during training while each network of [51] is specifically trained on a test style. Also, note that our style loss is much smaller than that of the original content image.

Speed analysis. Most of our computation is spent on content encoding, style encoding, and decoding, each roughly taking one third of the time. In some application scenarios such as video processing, the style image needs to be encoded only once and AdaIN can use the stored style statistics to process all subsequent images. In some other cases (*e.g.*, transferring the same content to different styles), the computation spent on content encoding can be shared.

In Tab. 1 we compare the speed of our method with previous ones [15, 51, 10, 5]. Excluding the time for style encoding, our algorithm runs at 56 and 15 FPS for 256×256 and 512×512 images respectively, making it possible to process arbitrary user-uploaded styles in real-time. Among algorithms applicable to arbitrary styles, our method is nearly 3 orders of magnitude faster than [15] and 1-2 orders of magnitude faster than [5]. The speed improvement over [5] is particularly significant for images of higher resolution, since the style swap layer in [5] does not scale well to high resolution style images. Moreover, our approach achieves comparable speed to feed-forward methods limited to a few styles [51, 10]. The slightly longer processing time of our method is mainly due to our larger VGG-based network, instead of methodological limitations. With a more efficient architecture, our speed can be further improved.

Method	Time (256px)	Time (512px)	# Styles
Gatys *et al*. [15]	14.17 (14.19)	46.75 (46.79)	∞
Chen and Schmidt [5]	0.171 (0.407)	3.214 (4.144)	∞
Ulyanov *et al*. [51]	**0.011** (N/A)	**0.038** (N/A)	1
Dumoulin *et al*. [10]	**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 [51, 10], while being much faster than other existing algorithms applicable to arbitrary styles [15, 5]. 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.

7.2. Additional experiments.

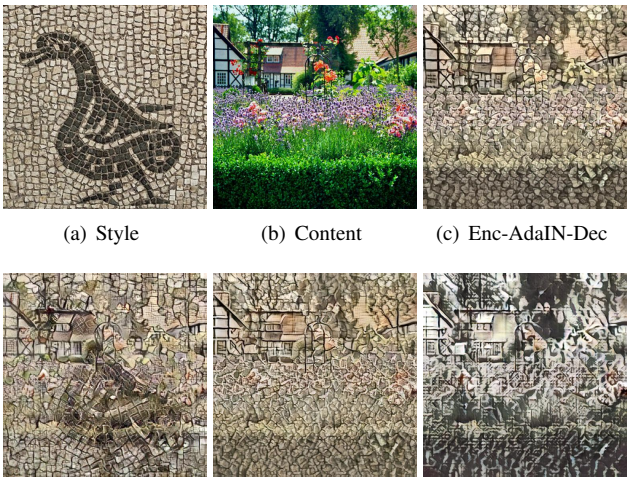
In this subsection, we conduct experiments to justify our important architectural choices. We denote our approach described in Sec. 6 as Enc-AdaIN-Dec. We experiment with a model named Enc-Concat-Dec that replaces AdaIN with concatenation, which is a natural baseline strategy to combine information from the content and style images. In addition, we run models with BN/IN layers in the decoder, denoted as Enc-AdaIN-BNDec and Enc-AdaIN-INDec respectively. Other training settings are kept the same.

In Fig. 5 and 6, we show examples and training curves of the compared methods. In the image generated by the Enc-Concat-Dec baseline (Fig. 5 (d)), the object contours of the style image can be clearly observed, suggesting that the network fails to disentangle the style information from the content of the style image. This is also consistent with Fig. 6, where Enc-Concat-Dec can reach low style loss but fail to decrease the content loss. The models with BN/IN layers in the decoder also obtain qualitatively worse results and consistently higher losses. The results with IN layers are especially poor. This once again verifies our claim that IN layers tend to normalize the output to a single style and thus should be avoided when we want to generate images in different styles. We believe this negative result, which can be directly inferred from our interpretation of IN, might also have implications for other image generation tasks.

7.3. Runtime controls

To further highlight the flexibility of our method, we show that our style transfer network allows users to control the degree of stylization, interpolate between different styles, transfer styles while preserving colors, and use different styles in different spatial regions. Note that all these controls are only applied at runtime using the same network, without any modification to the training procedure.

Content-style trade-off. The degree of style transfer can be controlled during training by adjusting the style weight



(a) Style (b) Content (c) Enc-AdaIN-Dec

(d) Enc-Concat-Dec (e) Enc-AdaIN-BNDec (f) Enc-AdaIN-INDec

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.

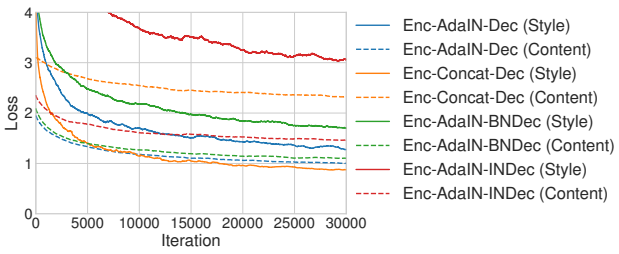


Figure 6. Training curves of style and content loss.

λ in Eqa. 11. In addition, our method allows content-style trade-off at test time by interpolating between feature maps that are fed to the decoder. Note that this is equivalent to interpolating between the affine parameters of AdaIN.

$$T(c, s, \alpha) = g((1-\alpha)f(c) + \alpha\text{AdaIN}(f(c), f(s))) \quad (14)$$

The network faithfully reconstructs the content image when $\alpha = 0$, and synthesizes the most stylized image when $\alpha = 1$. As shown in Fig. 7, a smooth transition between content-similarity and style-similarity can be observed by changing α from 0 to 1.

Style interpolation. To interpolate between a set of K style images $s_1, s_2, ..., s_K$ with corresponding weights $w_1, w_2, ..., w_K$ such that $\sum_{k=1}^{K} w_k = 1$, we similarly interpolate between feature maps (results shown in Fig. 8):

$$T(c, s_{1,2,...K}, w_{1,2,...K}) = g(\sum_{k=1}^{K} w_k \text{AdaIN}(f(c), f(s_k))) \quad (15)$$

Spatial and color control. Gatys *et al*. [16] recently introduced user controls over color information and spatial locations of style transfer, which can be easily incorporated into

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.

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.

our framework. To preserve the color of the content image, we first match the color distribution of the style image to that of the content image (similar to [16]), then perform a normal style transfer using the color-aligned style image as the style input. Examples results are shown in Fig. 9.

In Fig. 10 we demonstrate that our method can transfer different regions of the content image to different styles. This is achieved by performing AdaIN separately to different regions in the content feature maps using statistics from different style inputs, similar to [4, 16] but in a completely feed-forward manner. While our decoder is only trained on inputs with homogeneous styles, it generalizes naturally to inputs in which different regions have different styles.

8. Discussion and Conclusion

In this paper, we argue that instance normalization performs style normalization by normalizing feature statistics. Motivated by this interpretation, we then present a simple extension named adaptive instance normalization (AdaIN) that can adapt to arbitrary styles. Our style transfer network for the first time achieves arbitrary style transfer in real-time, thanks to the AdaIN layer. Beyond the fascinating applications, we believe this work also sheds light on our understanding of deep image representations in general.

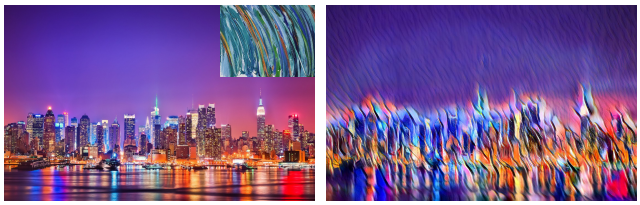


Figure 9. Color control. Left: content and style images. Right: color-preserved style transfer result.

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

It is interesting to discuss the conceptual differences between our approach and previous neural style transfer methods based on feature statistics of a DNN. Gatys *et al*. [15] employ an optimization process to manipulate pixel values to match feature statistics. The optimization process is replaced by feed-forward neural networks in [23, 50, 51]. Still, the network is trained to modify pixel values to *indirectly* match feature statistics. We adopt a very different approach that *directly* aligns statistics in the feature space *in one shot*, then inverts the features back to the pixel space.

Given the simplicity of our approach, we believe there is still substantial room for improvement. In future works we plan to explore more advanced network architectures such as the residual architecture [23] or an architecture with additional skip connections from the encoder [22]. We also plan to investigate more complicated training schemes like the incremental training [31]. Moreover, our AdaIN layer only aligns the most basic feature statistics (mean and variance). It is possible that replacing AdaIN with correlation alignment [48] or histogram matching [53] could further improve quality by transferring higher-order statistics. Another interesting direction is to apply AdaIN to texture synthesis.

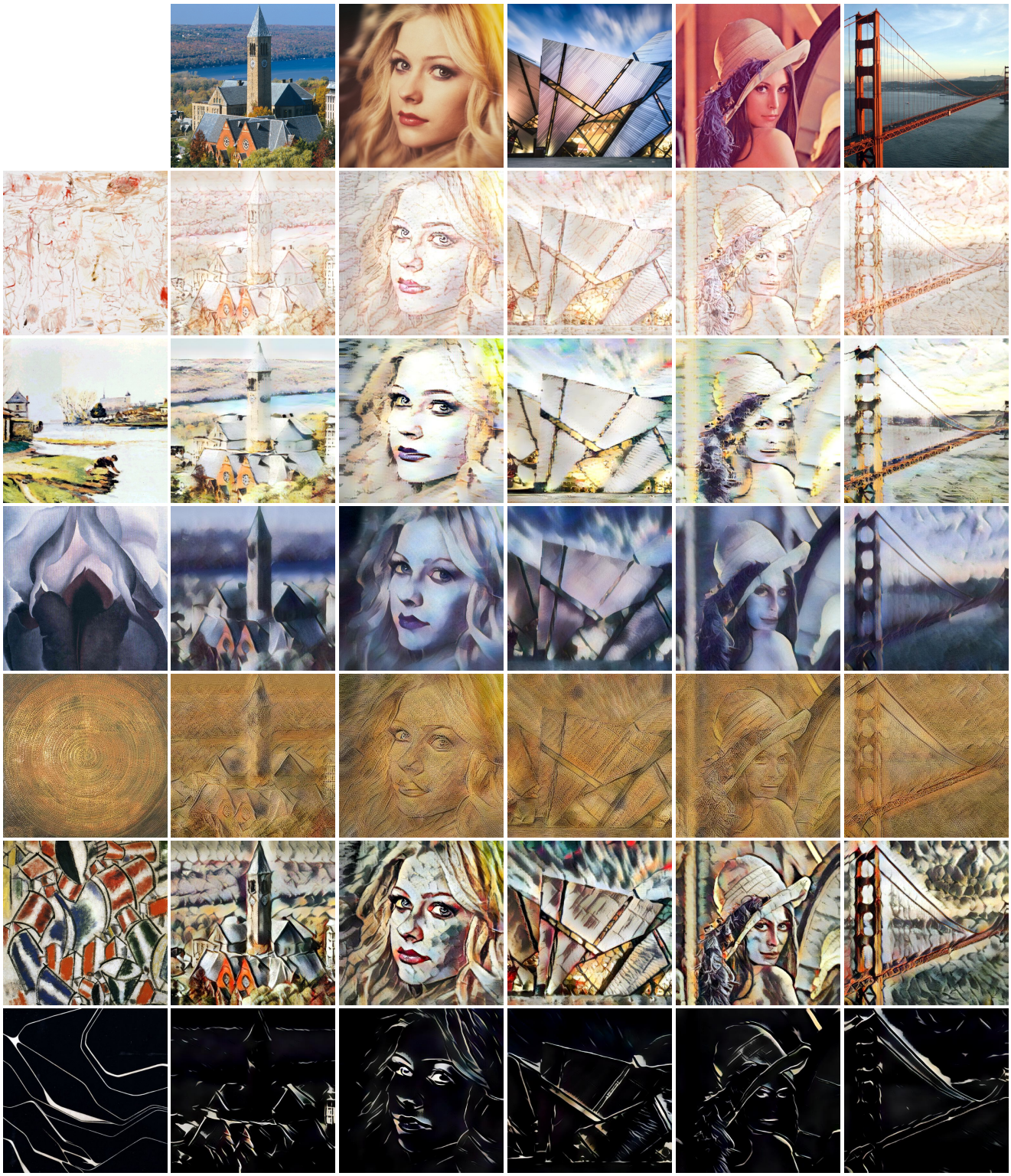


Figure 11. More examples of style transfer. Each row shares the same style while each column represents the same content. As before, the network has never seen the test style and content images.

Acknowledgments

We would like to thank Andreas Veit for helpful discussions. This work was supported in part by a Google Focused Research Award, AWS Cloud Credits for Research and a Facebook equipment donation.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. 2

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2

[3] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. *arXiv preprint arXiv:1612.05424*, 2016. 2

[4] A. J. Champandard. Semantic style transfer and turning two-bit doodles into fine artworks. *arXiv preprint arXiv:1603.01768*, 2016. 8

[5] T. Q. Chen and M. Schmidt. Fast patch-based style transfer of arbitrary style. *arXiv preprint arXiv:1612.04337*, 2016. 1, 2, 4, 5, 6, 7

[6] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *NIPS Workshop*, 2011. 4

[7] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent batch normalization. In *ICLR*, 2017. 2

[8] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015. 2

[9] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *CVPR*, 2016. 4

[10] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. In *ICLR*, 2017. 1, 2, 3, 5, 6, 7

[11] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH*, 2001. 1

[12] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, 1999. 1

[13] M. Elad and P. Milanfar. Style-transfer via texture-synthesis. *arXiv preprint arXiv:1609.03057*, 2016. 1

[14] O. Frigo, N. Sabater, J. Delon, and P. Hellier. Split and match: example-based adaptive patch sampling for unsupervised style transfer. In *CVPR*, 2016. 1

[15] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016. 1, 2, 3, 5, 6, 7, 8

[16] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman. Controlling perceptual factors in neural style transfer. In *CVPR*, 2017. 1, 7, 8

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 2

[18] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH*, 1995. 1

[19] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie. Stacked generative adversarial networks. In *CVPR*, 2017. 2

[20] S. Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. *arXiv preprint arXiv:1702.03275*, 2017. 2

[21] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *JMLR*, 2015. 2

[22] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 2, 8

[23] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 1, 2, 3, 8

[24] T. Kim, M. Cha, H. Kim, J. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*, 2017. 2

[25] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

[26] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014. 2

[27] J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg. State of the" art: A taxonomy of artistic stylization techniques for images and video. *TVCG*, 2013. 1

[28] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio. Batch normalized recurrent neural networks. In *ICASSP*, 2016. 2

[29] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *CVPR*, 2016. 1, 2, 3

[30] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *ECCV*, 2016. 1, 2

[31] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang. Diversified texture synthesis with feed-forward networks. In *CVPR*, 2017. 1, 2, 8

[32] Y. Li, N. Wang, J. Liu, and X. Hou. Demystifying neural style transfer. *arXiv preprint arXiv:1701.01036*, 2017. 1, 2, 3, 5

[33] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou. Revisiting batch normalization for practical domain adaptation. *arXiv preprint arXiv:1603.04779*, 2016. 2

[34] Q. Liao, K. Kawaguchi, and T. Poggio. Streaming normalization: Towards simpler and more biologically-plausible normalizations for online and recurrent learning. *arXiv preprint arXiv:1610.06160*, 2016. 2

[35] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014. 3, 5

[36] M.-Y. Liu, T. Breuel, and J. Kautz. Unsupervised image-to-image translation networks. *arXiv preprint arXiv:1703.00848*, 2017. 2

[37] M.-Y. Liu and O. Tuzel. Coupled generative adversarial networks. In *NIPS*, 2016. 2

[38] K. Nichol. Painter by numbers, wikiart. https://www.kaggle.com/c/painter-by-numbers, 2016. 5

[39] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016. 2

[40] X. Peng and K. Saenko. Synthetic to real adaptation with deep generative correlation alignment networks. *arXiv preprint arXiv:1701.05524*, 2017. 2

[41] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 2

[42] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *ICML*, 2016. 2

[43] M. Ren, R. Liao, R. Urtasun, F. H. Sinz, and R. S. Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. In *ICLR*, 2017. 2

[44] M. Ruder, A. Dosovitskiy, and T. Brox. Artistic style transfer for videos. In *GCPR*, 2016. 1

[45] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, 2016. 2

[46] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS*, 2016. 2

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 4, 5

[48] B. Sun, J. Feng, and K. Saenko. Return of frustratingly easy domain adaptation. In *AAAI*, 2016. 8

[49] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. In *ICLR*, 2017. 2

[50] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016. 1, 2, 4, 5, 8

[51] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, 2017. 1, 2, 3, 5, 6, 7, 8

[52] X. Wang, G. Oxholm, D. Zhang, and Y.-F. Wang. Multimodal transfer: A hierarchical deep convolutional neural network for fast artistic style transfer. *arXiv preprint arXiv:1612.01895*, 2016. 2

[53] P. Wilmot, E. Risser, and C. Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *arXiv preprint arXiv:1701.08893*, 2017. 2, 8