Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

Mostafa Pashazadeh
Introduction to Deep Learning for Computer Vision
Spring 2019

Abstract

This paper proposes a real-time style transfer algorithm that can transfer arbitrary new styles, in contrast to a single style or a set of styles. This algorithm is fast enough such that runs at 15 FPS with 512×512 *images while maintaining the flexibility of transferring arbitrary new styles in real-time. The underlying idea is a novel adaptive instance normalization (AdaIN) layer, which is similar to instance normalization (IN) but with affine parameters adaptively computed from the feature representations of an arbitrary style image. In addition, this method provides various user control at test time such as content-style trade-off and style interpolation, and needs no new training.*

1. Introduction

Style transfer is the process of rendering a content image in the artistic style of a style image. Former style transfer methods suffer from either running a separate slow iterative optimization process for each style transfer or they are restricted to a fixed style. This paper has shown that we can train one network to perform fast arbitrary style transfer in real time at test time.

Seminal former work on style transfer [1] as indicated in Figure 1, encodes the texture descriptor of the images, then uses a similar type of gradient descent procedure to synthesize a new image that matches the texture of the original image. It starts with a randomly initialized image, passes the generated image along with the style image and content image through a pre-trained network (often VGG [4]), computes the Gram matrices of the generated image and the style image at different layers as the texture descriptor of these images and then computes the style loss as the l2-norm between the Gram matrices of the generated image and the style image. The content loss is computed as the l2-norm between the features of the generated image and the content image. Then it minimizes the weighted combination of these losses through gradient descent method by updating the pixels of the generated image and eventually,

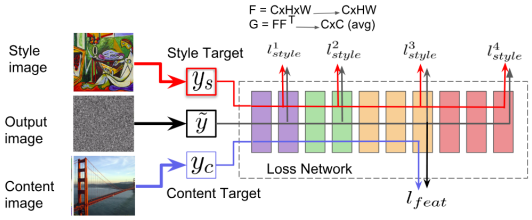

Figure 1. Optimization-based style transform algorithm.

generates an image that looks like the content image with the texture of the style image.
The drawback of this method is that it requires many forward and backward passes to generate the image and consequently, it is quite slow.

The solution presented in [3] is to train a network as illustrated in Figure 2. It receives the content image as input and directly synthesizes the stylized image rather than running a separate optimization procedure for each image.
Here we have a training set of content images and a style image. During the training phase, the network receive input images from the training set of content images. The way that we train this network is that we compute the same content and style loss and use the gradient descent method to update the weights of the feed forward network rather than the pixels of the generated image. Once it is trained, to produce the stylized image, we just need to pass the content image through the trained network.
The drawback of this method is that it is restricted to a fixed style.

To address the problem of flexibility and speed (which is the focus of this report), [2] presented a novel style transfer algorithm that resolves these fundamental problems. The method proposed here can perform the style transfer task with any arbitrary new style in real-time and a speed similar to the fastest feed-forward approach. The underlying idea is to adopt an AdaIN layer, which is a simple extension to IN layer. It transfers the mean and variance of the style image to the content image. These statistics in the feature space

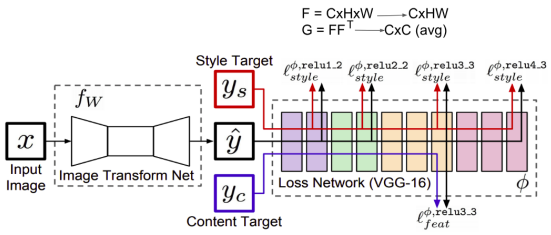

Figure 2. Feed forward network style transform algorithm.

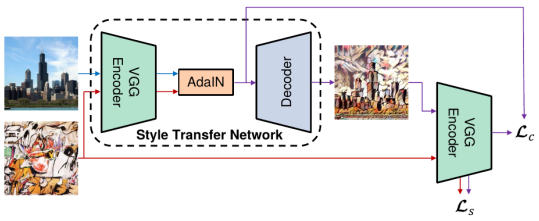

Figure 3. Overview of style transform algorithm.

represent the style information of an image with the same results as what we gain with Gram matrices. After this a decoder network is trained to generate synthesized image by inverting the AdaIN output back to the image space. This method is fast enough to process 56 FPS with 256×256 pixels and 15 FPS with 512×512 pixels while maintaining the flexibility of transferring arbitrary new styles in real-time. Furthermore, it provides quite a few user control at test time without any modification to training phase.

2. Contributions of the Original Paper

The arbitrary style transform algorithm is implied by the IN layer that normalizes the input image to a single style specified by its affine parameters. Therefore, it is likely to be true that we can render the input image to arbitrarily given styles by using adaptive affine transformations. The idea is a simple extension to IN, which is called adaptive instance normalization (AdaIN). AdaIN receives a content input x and a style input y, and adjust the channel-wise mean and variance of x to match those of y in the feature space. Unlike BN, IN or CIN, AdaIN has no affine parameters to learn. Instead, it adaptively computes the affine parameters from the style input as follows

$$AdaIN(x, y) = \sigma(y)(\frac{x - \mu(x)}{\sigma(x)}) + \mu(y) \tag{1}$$

It scales the normalized content input $\sigma(y)$, then shifts it with $\mu(y)$. Same as instance normalization, the statistics are calculated with respect to spatial dimensions for each channel and each sample.

$$\mu_{nc}(z) = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} z_{nchw} \tag{2}$$

$$\sigma_{nc}(z) = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(z_{nchw} - \mu_{nc}(z))^2 + \epsilon} \tag{3}$$

In brief, AdaIN encodes the texture of the content image as the texture of the style image by transferring channel-wise mean and variance in the feature space.

The arbitrary style transfer network T as shown in Figure 3, uses a content image c and an arbitrary style image s for its inputs. Then, it combines the content of the content image and the texture of the style image and generates a stylized output. It uses an encoder-decoder architecture. In this architecture, the encoder denoted by f is fixed to the first few layers of a pre-trained VGG-19. First, we encode the content and style images in feature space, then both feature maps are passed through an AdaIN layer where for the content feature maps, the mean and variance are matched to the mean and variance of the style feature maps. The result is the target feature maps t,

$$t = AdaIN(f(c), f(s)) \tag{4}$$

In order to generate the stylized image $T(c, s)$, then, a randomly initialized decoder g take the target feature map t and invert it back to the image space,

$$T(c, s) = g(t) \tag{5}$$

The decoder is implemented the same as the encoder and all the pooling layers are replaced by nearest up-sampling. Reflection padding is employed in both f and g. Also, it is important to decide what normalization layer the decoder should use. It can use either IN or BN, or no normalization layers at all. In IN, each sample is normalized to a single style. However, in BN, a batch of samples are forced to be centered around a certain style. Both are undesirable since the samples can have various styles. As a result, this algorithm does not uses normalization layers in the decoder. The pre-trained VGG is then used to compute the loss function to train the decoder. The loss function is a weighted combination of the content loss L_c and style loss L_s with the style loss weight λ,

$$L = L_c + \lambda L_s \tag{6}$$

The content loss is the Euclidean distance between the target features and the features of the output image. Instead of commonly used feature characteristics of the content image, it uses the AdaIN output t as the target features,

$$L_c = ||f(g(t)) - t||_2 \tag{7}$$

The style loss only matches the mean and standard deviation, instead of adjusting the Gram matrix, since the AdaIN layer only transfer the mean and the standard deviation of the style feature representations.

$$L_s = \sum_{i=1}^{L} \Big|\Big|\mu(\phi_i(g(t)) - \mu(\phi_i(s))\Big|\Big|_2 + \sum_{i=1}^{L} \Big|\Big|\sigma(\phi_i(g(t)) - \sigma(\phi_i(s))\Big|\Big|_2 \quad (8)$$

where each ϕ_i illustrate a layer in VGG-19 used to compute the style loss.

This method provides comparable quality to previous methods despite the fact that they focused either on flexibility or high speed transformation. The authors claim that it transfers the style in a more genuine manner and generalizes well considering that the network never gets to see the test styles during the training phase. Also, It runs at 56 FPS with 256×256 pixels and 15 FPS with 512×512 pixels, giving it the capability of working in real-time applications. An interesting feature of this algorithm is the user control that 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. Above all, these controls are applied in run-time and needs no new training.

3. Contributions of this Project

MS-COCO was used for training the network as the content images and a dataset of paintings from WikiArt as style images. Each dataset contains nearly 80,000 training examples. Adam optimizer with learning rate of 0.0001 and a batch size of 8 content-style image pairs was used. In order to train simultaneously on two datasets a new dataset class named ConcatDataset was created and concatenated the images as a way to extract samples from both content iterator and style iterator. During training, first the smallest dimension of each content image and style image is resized to 512 while preserving the aspect ratio, then regions of size 256×256 are randomly cropped. Since the network is fully convolutional, it can be applied to images of any size during testing. In addition, in order to avoid obstruction caused by system error message during training phase, we made use of try/except command to filter the datasets and prepare them before feeding them into the network.

In Figure 4, Figure 5 and Figure 6, training curves of style loss, content loss and the total loss are shown, respectively, and in Figure 7 and Figure 8, examples of style transfer results are shown. The style loss weight parameter λ was set to 4 during training phase. Note that all the test style images are never observed during the training of the model. Although the trends of loss values are expectedly decreasing, the final output of rendering the content image into the texture of the style image is inferior to the original results of the paper. The default values of hyper-parameters, style loss weight and learning rate decay, on the paper's git was misleading (styleWeight parameter is set to 0.01 and learningRateDecay is insignificant) and ended up with totally disappointing results even though everything was implemented in consistent with the paper's concepts and definitions and the lua script on the paper's git. Therefore, we performed a grid search for these hyper-parameters and these are the most decent results that we have got by the date we submitted this report. We know that there is still room for improvement, e.g., we could have made style loss decrease further by additional tuning or we could have allowed the training phase to go through more iterations to reduce the content loss further. However, we cautiously did the search for hyper-parameters with somehow small steps and evaluating each set was taking about two days. In addition, we were desperate to cut down the number of epochs to make time to evaluate more hyper-parameters before the due date. It should be mentioned that the more we moved forward in tuning by increasing the styleWeight, better results were obtained. It is most likely to get results closer to those of paper by further increasing this parameter. Moreover, the tiny ripples observed in the style loss curves is partly linked to the small size of test data set. This was again inevitable since loading big size of test data was taking quite a while.
Finally, Figure 9 illustrates interpolation between two style images s_1, s_2 with corresponding weights w_1, w_2 such that $\sum_{k=1}^{K} = 1$, by feeding the decoder with a convex combination of feature maps transferred to different styles via AdaIN (Equ. 1). This can be extended to several new styles.

$$T(c, s_{1,2}, w_{1,2}) = g(\sum_{k=1}^{K} w_k AdaIN(f(c), f(s_k))) \quad (9)$$

A short description for how to run the test function with the trained model to get the results was uploaded to the git (README.md).

4. Discussion and Future Directions

There is still room for improvement. For example, more advanced network architectures such as the residual architecture can be investigated rather than VGG network. Moreover, the AdaIN layer only matches the basic feature statistics (mean and variance). It is possible that further improve quality by transferring higher-order statistics replacing the AdaIN with correlation alignment.

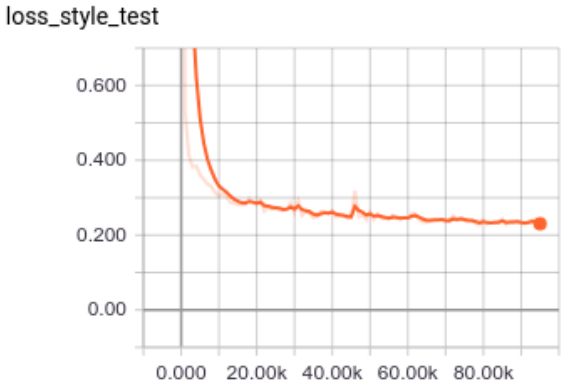

Figure 4. Style loss on test dataset.

Figure 5. Content loss on test dataset.

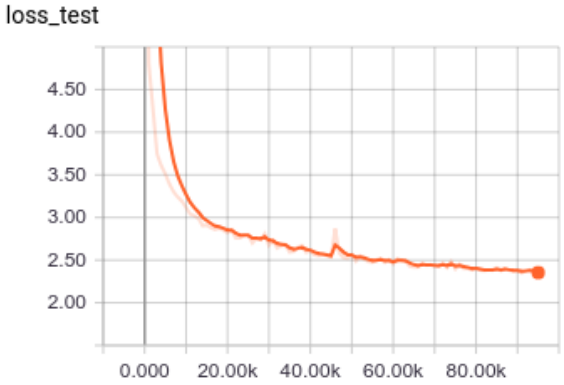

Figure 6. Total loss on test dataset.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[2] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

[3] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Figure 7. Example of style transfer results. The network has never seen the test style and content images.

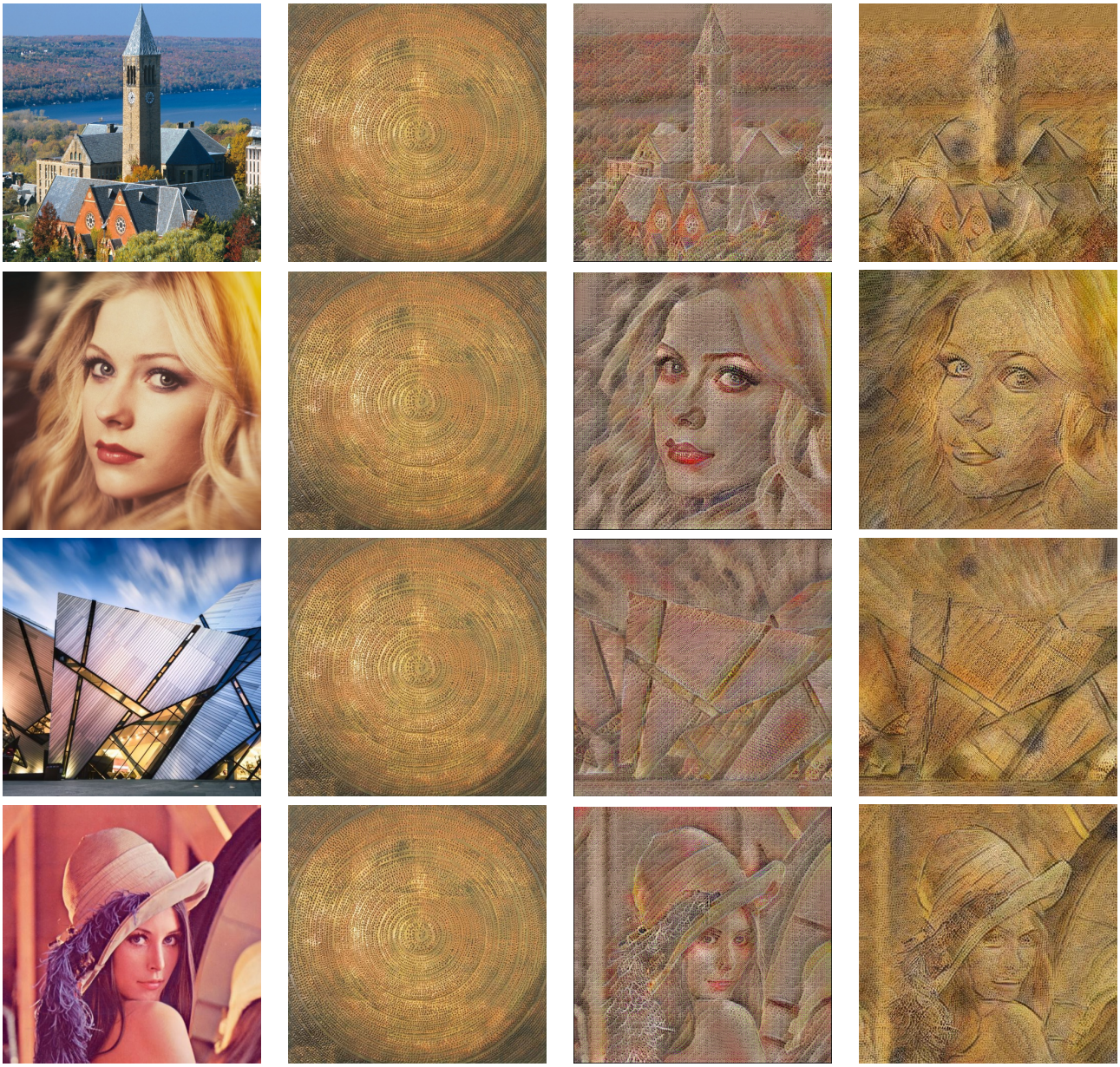

Figure 8. Example of style transfer results. The network has never seen the test style and content images.

Figure 9. Interpolation between two style images (Equ. 9). We can interpolate between arbitrary new styles.