Adaptive Instance Normalization for Style Transfer

April 30, 2019

This document is written as I read "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization" by Huang and Belongie [3].

1 Introduction

- The original style transfer paper [2] relies on a slow optimization process to do its job.
- There are attempts to make it faster by creating feed-forward networks for the task. However, each of these networks is specialized to a single style. A prior work that does arbitrary styles exist, but it is slower than single-style methods.
- The paper presents a method that can do arbitrary style transfer in real-time.
- The method is inspired by the the *instance normalization* (IN) layer [1, 9].
- The paper proposes an extension called *adaptive instance normalization* (AdaIN).
  - Given a content input and a style input, AdaIN adjusts the mean and variance of the content input to match those of the style input.
  - Output of AdaIN is fed to a decoder network to generate the final stylelized image.

2 Background

2.1 General Architecture for Style Transfer

- To perform style transfer with a feed-forward network, the batch of content image c is encoded into a high-level "feature map" $x = f(c)$ by a neural network ϕ. This is typically the first few layers of VGG-19 [7].
- The feature map x is then transformed in some way to get a "target" feature map t.
- The target feature map is then decoded by a network g to g4t the final image $g(t)$.

2.2 Batch Normalization

- The batch normalization (BN) layer normalizes the feature statistics of each batch to speedup feed-forward network training [4].
- Radford et al. uses batch normalization heavily in their DCGAN paper [6], proving that it is effective in generating images.

- Given an input patch $x \in \mathbb{R}^{N \times C \times H \times W}$, BN transforms the input as follows:

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

where

  - $\mu(x), \sigma(x) \in \mathbb{R}^C$ are the mean and standard deviation, computed across batch size and spatial dimensions independently for each feature channel:

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

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

    where ϵ is a small constant.
  - $\gamma, \beta \in \mathbb{R}^C$ are parameters learned from data.

- Note that $\mu(x)$ and $\sigma(x)$ are not defined at test time because the concept of mini-batch does not apply here. So, at test time, the two values are replaced by similar statistics computed from the dataset (or online while training).

2.3 Instance Normalization

- Ulyanov et al. proposes a feed-forward network for style transfer in 2016 [8], which contains a BN layer after each convolutional layer.

- Surprisingly, they later found that significant improvement can be achieved by simply replacing BN layers with IN layers:

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

The difference between IN and BN is that, now, $\mu(x)$ and $\sigma(x)$ are computed independenty for each channel and *sample*. In other words, $\mu(x)$ and $\sigma(x)$ are now elements of $\mathbb{R}^{N \times C}$, and:

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

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

Since the statistics are computed without relying on the existence of mini-batches, the IN layer remains the same at test time.

2.4 Conditional Instance Normalization

- The parameters γ and β above is learned one per a training dataset. Dumoulin et al. proposed 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, a style image together with its index s are randomly chosen from a fixed set of styles $s \in \{1, 2, \ldots, S\}$. The content image is processed with the same backbone network, using the corresponding γ^s and β^s in the CIN layers.
- CIN layers requires $2FS$ additional parameters, where F is the size of the feature map x. So, it can only deal with a finite number of styles.

3 Interpreting Instance Normalization

- Statistics of the feature map $x = f(t)$ can capture style information.
- The original paper [2] uses the second-order statistics. However, Li et al. [5] showed that matching statistics, such as channel-wise mean and variance, can also do style transfer.
- It is not well understood, though, why IN is effective.
  - Ulyanov et al. argues that this is because IN is invariant to constrast of the content image.
- The authors of the present paper argue Ulyanov et al. is wrong. The real reason is that IN performs some kind of *style normalization*. The evidences they provide are:
  - For the improved texture networks [9], IN performs better than BN.
  - If all training images are normalized to the same contrast, IN remains as more effective as BN in reducing style loss. This means Ulyanov et al.'s explanation does not hold. (If it were so, the difference should shrink.)
  - If all training images are normalized to the same style, then the difference between effectiveness of BN to IN shrinks. This means that IN performs some kind of style normalization.
- BN normalizes the statistics of the samples in a batch. This means that it tries to normalize all the images in the batch to the same style. However, each image has its own style, so it is not a good idea to normalize this with other images. While this effect can be fixed by the decoder, it makes training harder.
- IN, on the other hand, normalizes each image independenty; thereby allowing the training of decoder to focus on manipulating content directly.

4 Adaptive Instance Normalization

- The paper proposes a new layer called *adaptive instance normalization* (AdaIN). It takes two inputs:
  - x is the feature map of the content image c.
  - y is the feature map of the style image s.

  AdaIN produces the target feature map as follows:

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

  where $\mu(y)$ and $\sigma(y)$ are the mean and standard deviation of y computed independently between channels. In words, AdaIN simply matches the per-channel mean and variance of x to those of y.

5 Experimental Setup

5.1 Network Architecture

- The network takes two inputs: the content image c and the style image S.
- It produces an image whose content matches that of x and whose style matches that of s.
- The encoder network f is the first few layers of VGG-19. The paper uses up to `relu4_1`. This network is fixed; it is not changed by training.
- The network first computes $f(c)$ and $f(s)$.
- Then, it computes the target feature map $t = \text{AdaIN}(f(c), f(s))$.
- The part of the network that needs to be train is the decoder g that sends the feature map back to the normal image space. The generated stylized image is:

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

- The coder mirrors the encoder, with the exception that the pooling layers are replaced by nearest up-sampling.
- Minor details:
  - Reflection padding is used in both f and g to avoid border artifacts.
  - There are no normalization layers in the decoder because these would interfere with IN's operation.

5.2 Training

- Content images come from MS-COCO. Style images come from WikiArt data set in Kaggle.
- For each pair of image, resize the smallest dimension of both images to 512 while preserving the aspect ratio. Then, the resized image is randomly cropped to get regionds of size 256×256.
- The network was optimized by Adam with batch size of 8.
- The loss function is a weighted combination of the content loss and style loss:

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

- The content loss is the Euclidean distance betwen the target feature map and that of the output image:

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

- The style loss matches the statistics of various feature maps of the generated image with those of the style image:

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

  where ϕ_i denotes a layer in VGG-9. The paper uses 4 layers: `relu1_1`, `relu2_1`, `relu3_1`, and `relu4_1`.

References

[1] Dumoulin, V., Shlens, J., and Kudlur, M. A learned representation for artistic style. *CoRR abs/1610.07629* (2016).

[2] Gatys, L. A., Ecker, A. S., and Bethge, M. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016* (2016), pp. 2414–2423.

[3] Huang, X., and Belongie, S. J. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR abs/1703.06868* (2017).

[4] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR abs/1502.03167* (2015).

[5] Li, Y., Wang, N., Liu, J., and Hou, X. Demystifying neural style transfer. *CoRR abs/1701.01036* (2017).

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

[7] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR abs/1409.1556* (2014).

[8] Ulyanov, D., Lebedev, V., Vedaldi, A., and Lempitsky, V. S. Texture networks: Feed-forward synthesis of textures and stylized images. *CoRR abs/1603.03417* (2016).

[9] Ulyanov, D., Vedaldi, A., and Lempitsky, V. S. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. *CoRR abs/1701.02096* (2017).