Neural Style Transfer

Introduction

Since the original Neural Style transfer method [1] was published, various methods have been developed for transforming images from one domain to another. 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 [2, 3, 4]. 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 project a simple yet effective approach [5] is used, that enables arbitrary style transfer in real-time. At the heart of the method is an adaptive instance normalization (AdaIN) layer that aligns the mean and variance of the content features with those of the style features, which allows it to achieve a high speed, without the restriction to a predefined set of styles.

Adaptive Instance Normalization

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. It 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.

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

Unlike Instance Normalization, proposed in [6]:

$$IN(x, y) = \alpha\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

AdaIN has no learnable parameters α and β, which allows to use any styles for transferring, even those that were not in the training set.

Architecture

The 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. In this project a simple encoder-decoder architecture is adopted, in which the encoder f is fixed to the first few layers of a pretrained VGG-19. After encoding the content and style images in feature space, both feature maps goes 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:

$$t = AdaIN(f(c), f(s))$$

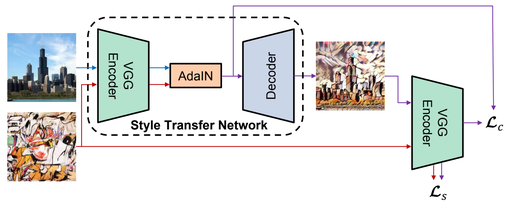

Figure 1: Architecture scheme.

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

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

Another important architectural choice is whether the decoder should use instance, batch, or no normalization layers. As discussed in [6], 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, there is no normalization layers in the decoder.

Training

The network is trained using 3236 random images of MS-COCO dataset [7] as content images and 3381 random images from dataset of paintings collected from WikiArt as style images. The adam optimizer and a batch size of 8 content-style image pairs is used. During training, the smallest dimension of both images is first resized to 512 while preserving the aspect ratio, then regions of size 256×256 are randomly croped . Since the network is fully convolutional, it can be applied to images of any size during testing.

To compute the loss function to train the decoder the pretrained VGG-19 is used:

$$L = L_c + \lambda L_s$$

where L_c is a content loss, L_s is a style loss and $\lambda = 10$ is a style loss weight.

$$L_c = ||f(g(t)) - t||_2$$

Since our AdaIN layer only transfers the mean and stan- dard deviation of the style features, our style loss only matches these statistics. So the style loss function is taken from [5]:

$$L_s = \sum_{i=1}^{P} ||\mu(\phi_i(g(t))) - \mu(\phi_i(s))||_2 +$$

$$+ \sum_{i=1}^{P} ||\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))||_2$$

where ϕ_i denotes a layer in VGG-19.

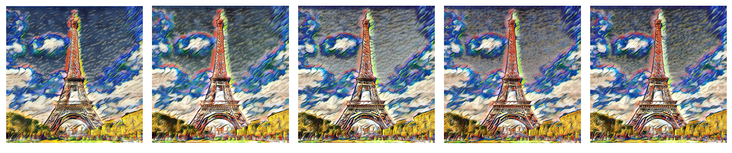

Figure 2: Visualization of model inference during training stages (from left to right: 20, 40, 60, 80, 100 thousands iterations).

Figure 2 and Figure 3 represent the training dynamics of the decoder. As we can see, content loss and style loss monotonically go down during training process, excpet for sharp explosion at 20 and 28 thousand iteration. This unusual behaviour can explained by relaunching training procedure in *Google Colab*.

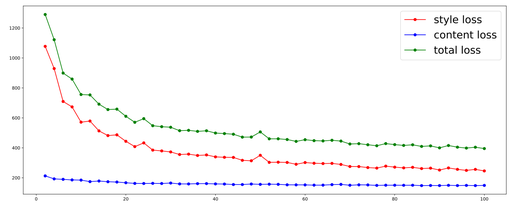

Figure 3: Loss vs number of training iterations.

Results

A social survey was created as a metric of the quality of the style transfer algorithm. Survey participants were asked to evaluate the quality of the transmission of style for 10 different pairs of content and style images on a 10-point scale. The survey involved 52 people. The average rating for each individual image ranges from 6.3 to 8.7, the average rating for all images is 7.3, which shows a fairly high quality of the algorithm performance.

The final of this project was the implementation of a telegram bot, that performs style transfer - @SpicyItalianBot. Code is available here.

Figure 4: An example of the project outcome.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[2] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[4] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. *CoRR*, abs/1701.01036, 2017.

[5] Xun Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *CoRR*, abs/1703.06868, 2017.

[6] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *CoRR*, abs/1603.03417, 2016.

[7] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.