Adaptive Instance Normalization Style Transfer

Isaac Fung
ifung@umich.edu

Hersh Vakharia
hershv@umich.edu

Ali Baker
aliibrah@umich.edu

Anthony Ke
keanthon@umich.edu

1 Introduction

Do you want Van Gogh to paint your selfie? Using a technique called style transfer, you may be able to. Style transfer is a group of software algorithms that change the aesthetic and style of an image to match that of a target image. We decided to tackle the problem of style transfer since we are very interested in topics which synthesize the arts with computer technology. We were also inspired by the neural-style repository on GitHub created by Dr. Justin Johnson [1] and thought that changing the styles of images to an aesthetic which resembled some of the most famous paintings to be a very intriguing concept.

This problem is fascinating since it opens up an opportunity to create digital art through the use of machine learning. This problem can be extended to programs that make their own art and visualize their own worlds.

We are building off of many already-existing methods for style transfer. For instance, the paper "A Neural Algorithm of Artistic Style" [2] is one of the most well known style transfers that uses an optimization based style transfer. Other examples include the paper "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks" [3] which uses a CycleGAN to transfer the style between unpaired sets of images.

For this project, we decided to implement the adaptive instance normalization (AdaIN) style transfer method proposed by Huang *et al.* [4]. We also implemented the well-known optimization technique proposed by Gatys *et al.* [2] to compare and contrast the results.

The AdaIN style transfer method is an arbitrary style transfer method, meaning it trains on a set of images to "remember" styles so that it can apply an arbitrary style to an image after it is trained. The method involves using a pre-trained encoder to extract style and content features, and then the AdaIN formula to shift the statistics of the content image to match the statistics of the style image. It is then passed through a decoder, which is trained to produce the content image with the style image's style transferred to it.

2 Approach

We will be describing our approach to AdaIN style transfer in detail. Since the optimization based style transfer by Gatys *et al.* [2] is so well known, we will not go into great depth about its architecture or implementation.

2.1 Architecture

Our AdaIN based encoder-decoder architecture is heavily inspired by the architecture described by Huang *et al.* [4]. Figure 1 shows a diagram of the architecture. Style and content images are both passed through an encoder to extract feature maps from both. Our encoder is a pre-trained VGG-19 network, but only uses the layers up to and including `relu4_1`.

Both content and style feature maps were then passed through adaptive instance normalization (AdaIN):

$$\text{AdaIN}(c, s) = \sigma(s)\frac{c - \mu(c)}{\mu(c)} + \mu(s) \tag{1}$$

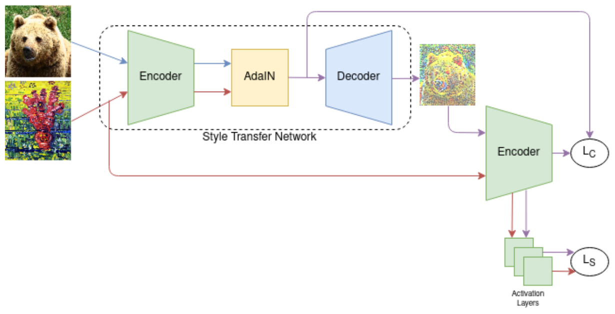


Figure 1: This is an overview of our AdaIN architecture. Images are passed through a pre-trained VGG-19 encoder to extract feature maps, and AdaIN matches the statistics between the two. The second encoder is used to compute content and style loss.

In equation 1, c is the content feature map, s is the style feature map, and σ and μ are the standard deviation and mean respectively. The result is a combined feature map that shifts the statistics of the content image to the statistics of the style image.

The output of AdaIN is then passed through a decoder. The decoder is a mirror of our encoder architecture, but replaces all max-pooling layers with nearest-neighbor up-sampling by a scale factor of 2. The decoder is the only piece of this network that is being trained. After training, the output of the decoder will produce the style-transferred image.

2.2 Loss

We calculated our loss using two components: a style loss L_s and a content loss L_c. These losses were combined into a single loss, L, using the formula:

$$L = L_c + \lambda L_s \tag{2}$$

The λ constant is a hyperparameter that denotes our style weight. To calculate style and content loss during training, we pass our original style image and our decoded image through the encoder. The feature maps produced by the encoder will be used to calculate content loss. For style loss, four activation layers from the encoder are stored. These layers were `relu1_1, relu2_1, relu3_1`, and `relu4_1`.

The content loss L_c is a metric to measure how well the generated output image is able to keep the content image's features. It is calculated using the mean squared error between the feature map of the decoded image and the feature map output of AdaIN.

$$L_c = \text{MSE}(d, \text{AdaIN}(c, s)) \tag{3}$$

In the equation above, d is the decoded image, c is the content image feature map, and s is the style image feature map.

The style loss is a metric which measured the difference in aesthetic, color, and style between the decoded image and the style image. The formula for the style loss is as follows:

$$L_s = \sum_{i=1}^{n} \text{MSE}(\mu(\phi_i(d) - \mu(\phi_i(s))) \quad + \\ \sum_{i=1}^{n} \text{MSE}(\sigma(\phi_i(d) - \sigma(\phi_i(s))) \tag{3}$$

Where ϕ_i denotes the ith activation layer that we saved from our encoder. This is essentially comparing the statistics of style image and the decoded image to determine how well the style is incorporated in the decoded image.

2.3 Training

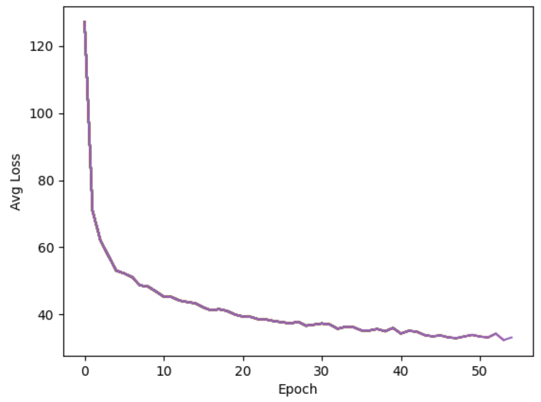


Figure 2: AdaIN transfer was trained over 54 epochs and the losses were averaged over the epoch. This is a plot of the average loss at each epoch.

We trained the AdaIN network over 54 epochs on a set of 1200 style and 1200 content images, with a batch size of 8 and a style weight of $\lambda = 20$. Training images were resized to 512 on their

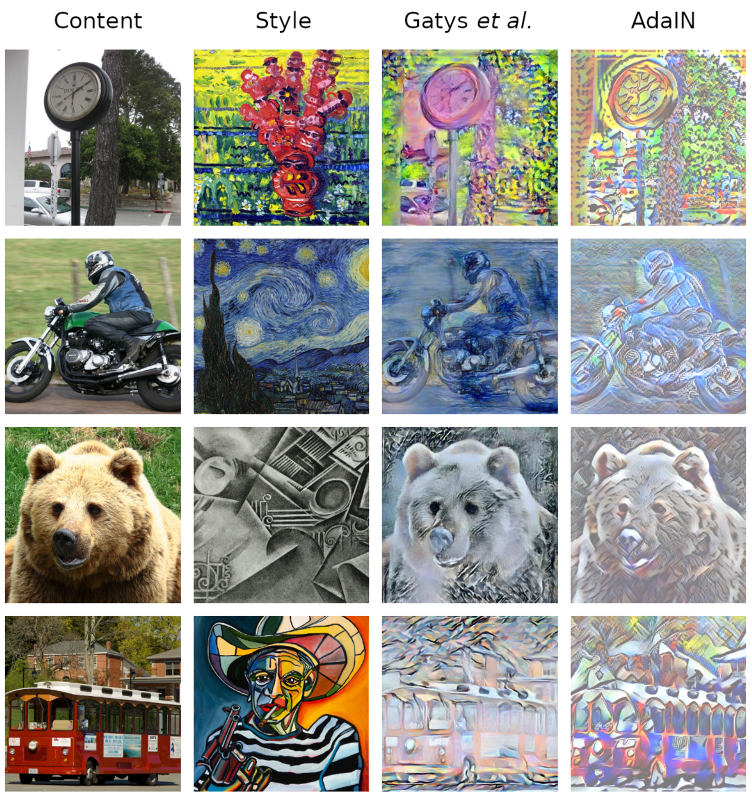


Figure 3: A comparison of our style transfer techniques.

smallest axis, and a random crop of 256x256 was done on that image. All images were normalized to have mean=[0.485, 0.456, 0.406] and stdev=[0.229, 0.224, 0.225] to match the pre-processing that our pre-trained VGG-19 network used. Test images were also normalized, but were de-normalized before the result was saved. We use the Adam optimization algorithm for gradient descent along with a learning rate of $1\text{e}{-4}$. The combined loss, L, was averaged every epoch, and the results have been plotted in Figure 2.

2.4 Gatys *et al.* Optimization Based Technique

The Gatys' technique involves defining a loss function that consists of a content loss (defined as the MSE loss between the feature map of one convolutional layer of the content image and that of a white noise) and a style loss (defined as the MSE loss of the gram matrices between the style image and the white noise).

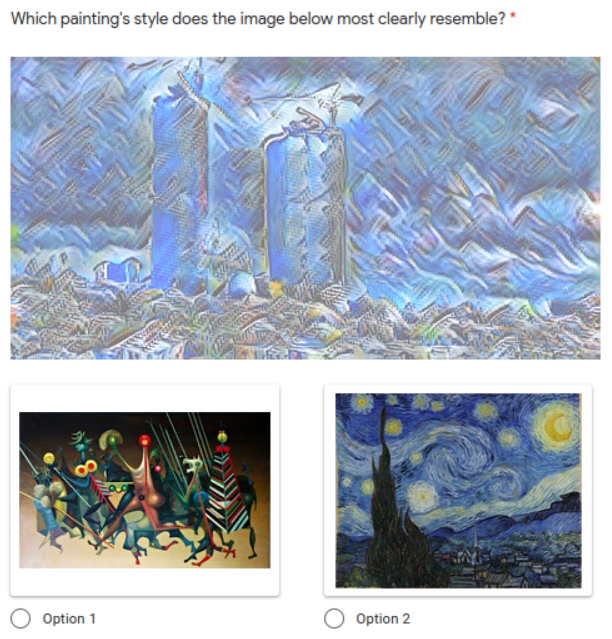
Gradient descent is performed to minimize the loss by updating the noise image after each iteration until the desired number of iterations is reached. The optimized noise image should now contain the style features of the style image while keeping the content of the content image.

The model was run with `conv4_2` as the content layer and (`conv1_1, conv2_1, conv3_1, conv4_1, conv5_1`) as the style layers. It was optimized over 50 epochs using the LBFGS optimizer with an learning rate of 1. The LBFGS optimizer was used after exploring Adam, SGD and Adadelta, since it produced the most visually pleasing images.

3 Experiments

3.1 Datasets

Publicly available datasets were used for training. For content images, we used the COCO dataset [5]. Style images were pulled from Wikiart, but provided by Kaggle [6].


Which painting's style does the image below most clearly resemble? *

Option 1

Option 2

Figure 4: An example question from our A/B testing evaluation method. In this case, the correct answer would be Option 2, as that was the painting that was used for the style of the original image.

3.2 Metric for Success

Art is subjective, and style transfer is an artistic effect. Therefore, it is hard to get a quantitative analysis of the success of our model. Therefore, we measure our metric for success qualitatively. We used an A/B testing approach to determine if a style transfer was successful. A Google Form was creating containing 5 tests. Figure 3 shows an example question from the form. The false style image was randomly selected from our training set.

Out of 65 responses, there were 250 correct answers and 75 incorrect answers. This gives our model a success rate of approximately 76.9%. This is fairly high, considering that we trained our model with limited computation power. From these results, we can say that our model is fairly successful in transferring style.

3.3 Comparison

A visual comparison of the Gatys *et al.* method [2] and AdaIN in Figure 3 shows that Gatys' optimization based technique produces a more pronounced style transfer. It was able to capture the texture of the style image better, and the colors were more well defined. This is because the optimization based technique explicitly relearns the style transfer for each set of content and style images while the AdaIN approach remembers arbitrary styles in the trained decoder over a large dataset of content and style images.

Additionally, we compared the run time of the two architectures. As we can see below, AdaIN has a much faster run time.

Technique	Time/sec (5 images)
AdaIN	26.46
Optimization	3921.12

4 Implementation

Our network architectures are based on existing papers [2, 4]. We utilize the PyTorch library to implement both the AdaIN style transfer and the optimization based style transfer. The AdaIN network was trained and run locally with CUDA optimizations on a Nvidia RTX 2080. The optimization based network was run on Google Colaboratory with CUDA optimizations on their GPUs. Both networks utilized a pre-trained VGG-19 provided by the torchvision library.

To evaluate our results, we used Google Forms to create a survey to send to students in EECS 442. Students in the class were asked on both the groupchat and Piazza to fill out the form.

5 Conclusion

According to our results, it shows that our style transfer program worked fairly well in transferring styles of paintings to other pictures. We are content with our results, however there is much more room for improvement. If we had more time and processing power, we would train the model on a much larger dataset and many more epochs to get more accurate results. We would also continue experimenting with other hyperparameters such as tuning the weight of the style loss and the learning rate.

References

[1] Justin Johnson. neural-style, 2015. URL `https://github.com/jcjohnson/neural-style`.

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[3] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-toimage translation

using cycle-consistent adversarial networks, 2020. Berkeley AI Research (BAIR) laboratory, UC Berkeley.

[4] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization, 2017. Department of Computer Science Cornell Tech, Cornell University.

[5] Coco dataset, 2017. URL `http://images.cocodataset.org/zips/val2017.zip`. 2017 Val images.

[6] Painter by numbers, 2016. URL `http://images.cocodataset.org/zips/val2017.zip`. Original images provided by wikiart.org.