Arbitrary Neural Style Transfer
{youw, ningyuan, manz68}@stanford.edu

Arbitrary Neural Style Transfer

Willow Wu, Ningyuan Zhang, Man Zhu
{youw, ningyuan, manz68} @stanford.edu

Abstract

Style transfer models combine the content of one image with the style of another image. The popular fast neural style transfer method trains a feedforward neural network for one style. We want to build an arbitrary style transfer network that can transfer an image to unseen styles. To resolve this problem, we add a layer after the first few layers of VGG to train the style. We tried three types of layers: (1) An AdaIN layer to transfer the channel-wise mean and variance, (2) a CORAL layer to align the covariance of the generated and style image, and (3) a histogram layer to match the cumulative distribution functions.

1. Introduction

Style transfer is the technique of combining the content of one image with the style of another image. This problem is attractive to our team, because we are big fans of artwork and photoshop. Specifically, we would like to provide an artistic painting tool that allows people to create artistic pictures of any style they like. Existing apps only allow us to choose from existing style templates. The reason is that many of these apps are based on the fast neural style transfer algorithm, which trains a feedforward neural network for each of the style image. To overcome this limitation, we want to build an arbitrary neural style transfer network that can combine a content image with an arbitrary style image that the users upload.

2. Related Work

2.1 Neural Style Transfer

In the seminal paper of neural style transfer, Gatys et al. (2016) use a pre-trained CNN (VGG-19) to compute features for content images and gram matrices for style images. Then the algorithm computes per-pixel loss between the output and the ground-truth, and update the pixel values through multiple rounds of iterations. Specifically, the implementation takes one content image and one style image as inputs, and minimizes the sum of style loss and content loss. At every iteration, the pixel values of the generated image get updated. Finally, the goal is to generate an image that combines the content with the style.

2.2 Fast Neural Style Transfer

Since per-pixel loss may not accurately represent perceptual similarity, and it is slow to do forward and backward propagations for each input, fast neural network is introduced to resolve the problems. To resolve these problems, fast neural style transfer trains a feed-forward network for each layer, which reduces the need to go through backward propagation, and it also uses perceptual loss function to replace per-pixel loss. Specifically, Johnson at el. (2016) proposed using perceptual loss functions to train their networks instead of per-pixel loss. Perceptual loss is based on the differences between high-level feature representations extracted from pretrained convolutional neural networks and can capture perceptual differences between output and ground-truth images. Furthermore, to reduce the computational burden caused by multiple iterations of optimization, a feed-forward network is trained to quickly approximate solutions to the optimization problem proposed by Gatys et al. (2016). By combining a perceptual loss with the feed-forward network, the fast neural style transfer algorithm achieves high-quality style transfer at a much faster speed than the neural style transfer algorithm.

2.3 Arbitrary Style Transfer

While fast neural style transfer speeds up the model by using a feed-forward neural network, a drawback of this method is that the trained network is restricted to a single style, or a set of fixed styles. However, users may want to transform a content picture to an arbitrary style of their choice, which the fast neural style algorithm fails to handle. In comparison, the optimization-based neural style transfer can combine content images with arbitrary styles, but the process is too slow. To resolve this flexibility-speed tradeoff, Huang and Belongie (2017) propose to use an adaptive instance normalization (AdaIN) layer to perform style transfer. Specifically, they use the first few layers of a pre-trained VGG-19 network to encode

Arbitrary Neural Style Transfer
{youw, ningyuan, manz68}@stanford.edu

the content and style images. Then they add an AdaIN layer to conduct style transfer. This AdaIN layer adjusts the mean and variance of the content image to match the mean and variance of the style image. After the AdaIN layer, they add a decoder to convert the output to the original image space.

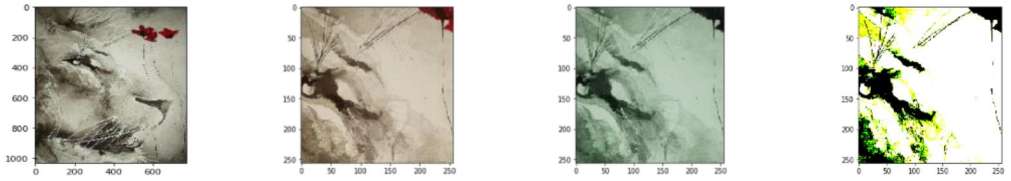
3. Dataset and Features

3.1 Data Sources

According to Huang and Belongie (2017), we use MS-COCO 2014 data[1] for our content images and we use WikiArt dataset[2] for our style images. The MS-COCO dataset has more than 330,000 images, and it is one of the most widely used dataset for computer vision tasks. The Wikiart dataset contains around 80,000 images of artworks.

3.2 Data preprocessing and data features

We clean the Wikiart dataset by filtering out images that are too large or has zero pixel. The MSCOCO dataset is already clean. For data preprocessing, we loop through both image datasets, filter out images not in png/jpg/jpeg format. Then we resize the smallest dimension to 512 while keeping the aspect ratio, and then randomly crop a region of [256, 256, 3] from the images. Before inputting into the neural network, we switch the image mode to BGR and then zero-center the image by subtracting BGR mean pixel, which is [103.939, 116.779, 123.68]. The raw features are RGB values of each pixel. Derived features (by kernel/filters) are at higher levels, such as edges, colors, and textures.



(1032, 774, 3) (256, 256, 3) RGB to BGR Zero centered

Figure 1. Examples of processed data piece

4. Models

4.1 Architecture

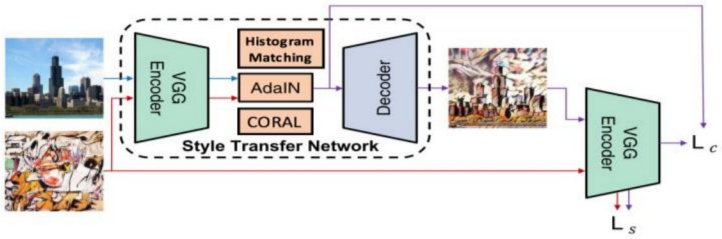


Figure 2. Arbitrary Style Transfer Architecture

The architecture of our model is shown in Figure 2. Specifically, the input is a pair of content image and style image, and the output is a combined image representing both the content and the style. The network is composed of an encoder, a matching layer, and a decoder. The encoder uses the first four layers of the VGG-19 network. The decoder uses the reverse structure of the encoder, where the pooling layers are replaced by nearest up-sampling to mirror the encoder input. The matching layer used by Huang and and Belongie (2017) is an AdaIN layer that matches the mean and variance between the encoded content and style features channel-wisely.

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

We use the same loss functions proposed by Huang and Belongie [3]. Specifically, we use the pre-trained VGG-19 to compute the loss function, which is content loss L_c plus the weighted style loss L_s with

[1] Common Objects in Context, MSCOCO. http://cocodataset.org/, 2014
[2] Painter by numbers, wikiart. https://www. kaggle.com/c/painter-by-numbers, 2016

Arbitrary Neural Style Transfer
{youw, ningyuan, manz68}@stanford.edu

weight λ. The content loss is the Euclidean distance between the target features and the generated image features. The style loss is the Euclidean distance of the mean and variance across all the layers. The formula is as following:

$$L = L_c + \lambda \cdot L_s \qquad L_c = \left\| f\left(g(t)\right) - t \right\|_2$$

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

where L is the total loss; L_c is the content loss; L_s is the style loss; λ is the weight of the style loss; t is the AdaIN layer output; $g(t)$ is the decoder function; $f\left(g(t)\right)$ is the encoder function; $\phi_i\left(g(t)\right)$ is a layer in relu_1_1, relu_2_1, relu_3, and relu_4_1 of the VGG-19; μ is the mean; σ is the variance.

However, the AdaIN layer may not capture the granularity of the style, because it only matches on the mean and the variance. We may further improve the style transfer quality by matching on higher-order statistics. We tried two methods in our implementation that replace the AdaIN layer. The first is to use correlation alignment (CORAL) that aligns the covariance of two matrices. The second is to use histogram matching that aligns the cumulative distribution functions. The implementations are included in the code, and we will discuss these two methods in detail below.

4.2 Correlation Alignment

It has been shown in many research papers that the good "similarity" between the distributions of training and test data can help decrease the test error of supervised methods (Saenko et al. 2010). Recently, Sun et. al (2016) find that domain shift can be a critical and effective method to improve the performance of machine learning methods in real world application. Specifically, they propose a method for unsupervised domain adaptation, called Correlation Alignment (CORAL). CORAL aligns the input feature distribution of the source and target domains by first whitening the source feature using its second-order statistics, named covariance, and then recoloring it with the covariance of the target distribution.

We implement the CORAL algorithm in tensorflow by treating preprocessed content features as source data and style features as target data. As CORAL is a channel-wise approach, we first flatten the input content or style features, which is a 4D tensor of $[m, n_h, n_w, n_c]$, into a corresponding 3D tensor of $[m, n_c, n_h, n_w]$. Then we compute the covariance statistics of the flatten 3D tensor by adding up a covariance matrix and an eye matrix of dimension $[m, n_c, n_c]$. After applying the whitening and recoloring linear transformation to content feature, we reshape the CORAL-adapted content tensor back to a 4D tensor of $[m, n_h, n_w, n_c]$ as the original. Through the CORAL layer, the domain of content features is shifted close to the style features domain.

4.3 Histogram matching

An alternative way to align the content image with the style image is to use histogram matching. Histogram matching maps the cumulative distribution function of an image to a specified distribution. Accordingly, Risser et al. (2016) propose to use histogram losses for style matching, which match the statistical distribution of activations within CNN layers. Specifically, they use an ordinary histogram matching to remap the output activations to match the activations of the input. The loss is computed by the Frobenius norm distance between the original activations and the remapped ones.

In our project, however, the inputs to be remapped are not images, but the outputs of the first four layers of VGG-19. Therefore, the values are continuous and unbounded, rather than integers ranging from 0 to 255. We tried to create clusters and rescale the values to 0-255, but cannot find a reliable transformation, particularly when the two distributions have little overlap. For this reason, we fail to implement histogram matching, but this is an interesting question for future work.

5. Experiments/Results/Discussion

5.1 Training

We use an Adam optimizer to train the decoder. The starting point for model tuning is the hyperparameters used by Huang and Belongie (2017). Two of the most Important hyperparameters in this model are the learning rate and the style weight λ in the loss function. Based on the Huang and Belongie's

Arbitrary Neural Style Transfer
{youw, ningyuan, manz68}@stanford.edu

(2017) implementation (learning rate = 2, $\lambda = 2$), we first fixed the learning rate to 10^{-4}, and tried different λ: 1.5, 2 and 2.5. We find that through 10,000 steps, $\lambda = 1.5$ provides the smallest loss. Then we fixed λ to 2, and tried different learning rates: 10^{-3}, 10^{-4}, 10^{-5}. Among the 5 combinations, we found that $\lambda = 2$, learning rate = 10^{-4} work the best. Due to limited computational resources and a longer-time to run the CORAL models, we use the same set of hyperparameters to train the CORAL model. The losses for hyper-parameter tuning and final choice of hyperparameters are provided in Table 1 and Table 2.

Table 1. Hyperparameter Tuning

	Content loss	Style loss	Total loss
$\lambda = 1.5$, learning rate = 10^{-4}	5563	2010	8579
$\lambda = 2.0$, learning rate = 10^{-4}	9632	2752	15137
$\lambda = 2.5$,learning rate = 10^{-4}	12334	26129	77657
$\lambda = 2.0$, learning rate = 10^{-3}	14433	42157	98747
$\lambda = 2.0$, learning rate = 10^{-5}	15599	20058	55715

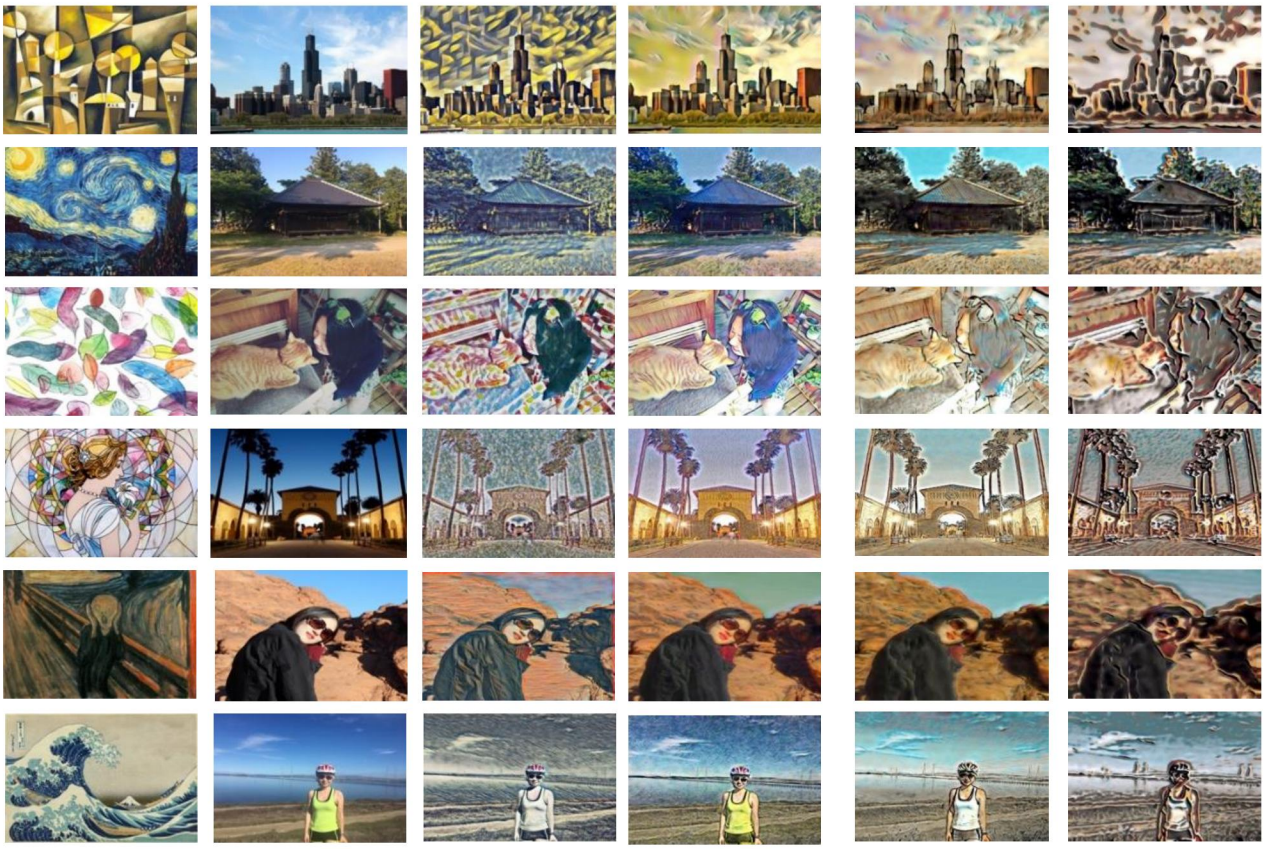
Table 2. Final Choice of Hyperparameters

Learning Rate	10^{-4}
Style Weight λ	1.5
Batch Size	8
Epoch	16
Epsilon	10^{-5}

Note: We use the default hyperparameters for the Adam optimizer (b_1=0.9, b_2=0.999, e=10^{-8})

5.2 Results

5.2.1 Qualitative Analysis



Style Content Fast AdaIN AdaIN (9000) CORAL (9000)

Figure 3. Style Transfer Result Examples

Comparing the third and the fourth columns, we can see that fast neural network works better at capturing higher level style features, such as less distinct color patterns and outlines, while arbitrary style transfer can capture lower level style features well, such as general tints of an image. Moreover, we find

that the CORAL layer may be better at capturing styles than the AdaIN layer. Since we have limited computational resources, and it takes four times longer train a CORAL model than an AdaIN model, we only train 9,000 steps of the CORAL model, and we compare the results of training 9,000 steps of the AdaIN model. Comparing the fifth and the sixth columns, we find that the CORAL model represents the style better than the AdaIN model, but the content loss of the CORAL model is large.

5.2.2 Quantitative Analysis

In accordance with the images above, Figure 5 shows that the content loss of the CORAL method has a sharp decrease at around 4,000 steps, but does not converge quickly afterwards. The style loss of the CORAL method sharply decreases after around 4,000 steps and then gradually decreases. In comparison, both the content and style losses of the AdaIN method has a more satisfying pattern of convergence.

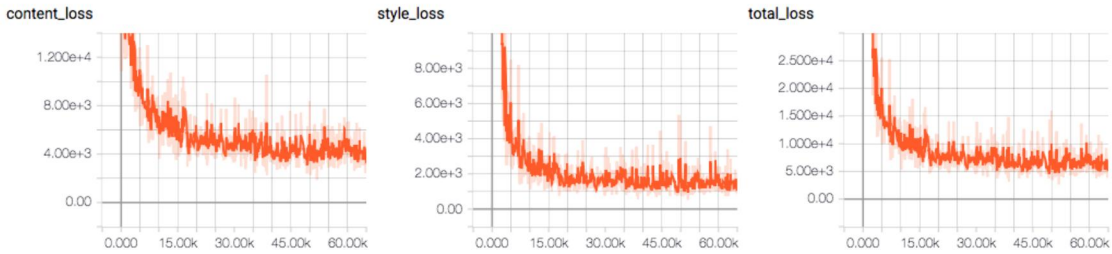


Figure 4. Losses of the Arbitrary Style Transfer with AdaIN (Up to 71,000 steps)

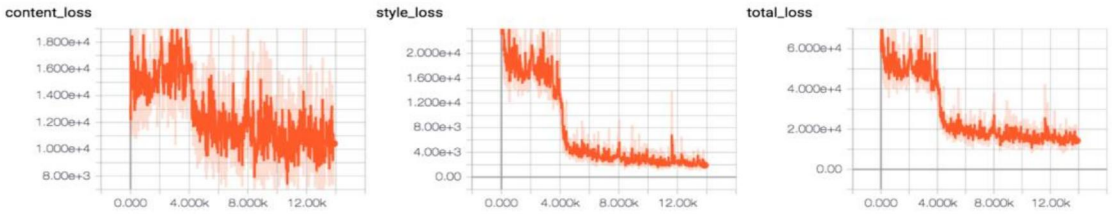


Figure 5. Losses of the Arbitrary Style Transfer with CORAL (Up to 9,000 steps)

5.3 Discussions and Conclusions

Comparing the arbitrary and the fast neural style transfer methods, we find that arbitrary style transfer models do not capture the style as well as fast neural style transfer. An intuitive explanation is that the fast method pre-trains one model for each style image, while the arbitrary method can be applied to any style. A more accurate explanation is two-folded. First, the arbitrary method only uses the first four layers of VGG, and this shallower architecture may not capture comprehensive features of the style. Second, AdaIN only matches the mean and variance, whereas gram matrix captures higher-order statistics.

To address the second issue above, we experiment with higher-order statistics to match the content and the style features. In theory, matching on covariance and cumulative distribution functions could improve the alignment between the encoded content and style features. We do find that the CORAL method represents the style well, but we find that there is a sudden decrease in the losses, and the content loss does not converge well. This pattern of result may be caused by the backpropagation of a CORAL layer, which computes the square root of a matrix.

6. Future work

Our project provides several avenues for future research. First, future work could implement histogram matching for the encoded features, and further explore why the content loss of CORAL layer does not converge well. Second, to promote style capturing, future work can use a deeper neural network with residual architecture by adding additional skips from the encoder. Finally, while the style transfer methods have been relatively well developed to generate satisfying results, the synthesized images have been criticized as lacking the “soul”. To address this problem, it may be interesting to train a discriminator that act as art critics, and the generator can possibly learn the “soul” of art.

Arbitrary Neural Style Transfer

{youw, ningyuan, manz68}@stanford.edu

7. Contributions

All team members researched, discussed and designed the final training architecture together, collaborated to finish the report and poster. For the specific tasks, Willow and Ningyuan coded for different matching layers, conducted the experiments and tuned the hyperparameters. Man organized the whole process, sorted out outputs, and analyzed the data.

References:

[1] Sun, Baochen, Jiash Feng, and Kate Saenko, "Return of Frustratingly Easy DomainAdaptation," *AAAI*, vol. 6, no. 7, 2016.

[2] Risser, Eric, Pierre Wilmot, and Connelly Barnes, "Stable and controllable neural texture synthesis and style transfer using histogram losses," *ArXiv e-prints 4*, 2017.

[3] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge, "Image style transfer using convolutional neural networks," In IEEE Conference on Computer Vision and Pattern Recognition (CVPR) '06, 2016, pp. 2414-2423.

[4] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," In European Conference on Computer Vision'10, 2016, pp. 694-711.

[5] Kate Saenko, "Synthetic to real adaptation with deep generative correlation alignment networks," *arXiv preprint arXiv:1701.05524*, 2010.

[6] Antonio Torralba and Alexei A. Efros., "Unbiased look at dataset bias," *CVPR*,2017.

[7] Huang, Xun, and Serge Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," *ICCV*, 2017.

[8] Fast Style Transfer, Logan Engstrom, https://github. com/lengstrom/fast-style-transfer/, 2016

[9] **Arbitrary style transfer,** elleryqueenhomels, https://github.com/elleryqueenhomels/arbitrary_style_transfer/, 2017