End-to-End Learning of One Objective Function to Represent Multiple Styles for Neural Style Transfer

Shubhang Desai
Stanford University
shubhang@stanford.edu

Abstract

We propose a novel architecture trained end-to-end for neural style transfer for multiple images. The architecture composes of two neural networks: an Image Transformation Network which transforms the content image into the stylized pastiche; and a proposed Normalization Network which produces unique instance normalization parameters for styles to be fed into the Image Transformation Network. The addition of a Normalization Network allows the style image to be varied over training and hence makes it an arbitrary variable. The network is successfully able to represent multiple unique artistic styles and, as suggested from an experiment using a large dataset of style images, arbitrary style transfer may be possible.

1 Introduction

Neural style transfer has progressed tremendously from the initially proposed optimization process [5] to the state-of-the-art feedforward approach using Image Transformation Networks [10]. One of the biggest puzzles remained in the fact that each of these networks could only produce pastiches of one given style, the style which was passed through the loss network during train time to compute the style losses. A key insight from Dumoulin et al. [3] showed that the key to representing a style was simply unique instance normalization parameters, demonstrating the insight's effectiveness by directly learning conditional instance normalization parameters for multiple styles in a single network. This paper directly builds off of that finding.

We propose a novel Normalization Network which gets as input a style image and outputs the instance normalization parameters that represent it. The output is then passed into the various instance normalization layers of the Instance Transformation Network, which the content image is passed through to produce a pastiche image.

This paper aims to answer whether or not many styles can be represented with such a system, and whether or not such a system could generalize to all styles and become a truly arbitrary style transfer system. Our results conclusively show that one objective function trained end-to-end is successful at representing multiple, artistically diverse styles. Our experiments on generalization show that there is potential for this architecture to be able to represent arbitrary styles.

2 Background/Related Work

There was quite a bit of setup which was required before neural style transfer could succeed. First, the problem of texture synthesis, producing an image given a texture is an input, was explored. Many older approaches [11, 1] attempted to produce textures by simply scanning the sample across and down to produce a larger image which still looks natural. This method works very well, but generally only for homogeneous textures. It wasn't until recently that this problem was tackled using a neural algorithm. Gatys et al. [6] proposed a method which extracted features from a pretrained network and used the Gram matrices of the feature maps to turn noise into a texture. The Gram matrix of the feature map contains information about texture, and so reducing the distance between the noise and a texture's Gram matrices at various layers will eventually turn the noise into a texture.

There was also significant research done into feature reconstruction to set the stage for neural style transfer. Yosinksi et al. [19] and Nguyen et al. [15] use gradient ascent to visualize what the network understands to be a certain class value at various layers. We essentially start with noise and compute gradients on the image to maximize a certain class value probability, giving us a look into what the networks believes to be an ideal class member. This technique was used by Google Deep Dream [14] to maximize the occurrence of certain features in an input picture of video. Along a similar track, Mahendran et al. [13] use an image's feature vector to produce another image which has a similar feature vector to the image's. It was found that doing this produced a result which looked similar to the image itself in shallow layers.

Using these insights, neural endeavors into artistic style transfer began with Gatys et al. [5], in which a VGG-19 network [17] pretrained on image classification is fed a content image, a style image, and a pastiche image which is initially noise. Similar to feature inversion and texture synthesis, the images' features are extracted from the network at certain layers in the network to compute style and content losses. These losses are minimized by computing gradients on the initial image. This makes this approach a slow optimization process. Johnson et al. [10] sought to remedy this by creating an Image Transformation Network which is trained with a dataset of content images to produce the pastiche image that is passed through the VGG-19 to compute losses as described above. Although this feedforward network achieved real-time style transfer, the approach was limited to one style per network. Soon after that paper, Ulyanov et al. [18] found that using instance normalization layers in place of batch normalization layers in the Image Transformation Network produced much better results. Dumoulin et al. [3] found that the gammas and betas of instance normalization layers in the Image Transformation Networks are the salient parameters in representing styles, and hence created conditional instance normalization layers (in which the gammas and betas were rows of matrices, one row for each style the network was trained on, and all of the normalization parameters were learned directly). Again, this method produces real-time style transfer and even allows a single network to produce more than one stylization; however, it was not yet clear whether or not the same intuition could be used to generalize to all styles.

Shortly before the start of this research, Huang et al [8] proposed a fixed-function to compute normalization parameters for arbitrary styles. However, the fixed-function approach makes the method inflexible as compared to a neural approach. Another approach [4] uses small patches of the pastiche as the image to stylize; this approach is able to learn arbitrary styles but is more adept at transferring color than style. Still another approach [20] uses a concept very similar to our Normalization Network, but instead of transforming a style into a unique representation which can be used as instance normalization parameters, the normalization parameters are taken from various layers in their network (called the Descriptive Network). This method does not conveniently describe every style with just one unique matrix and also produces results with more color transferred than style.

Concurrent to this research, a paper from Ghiasi et al. [7] was released which employs a similar network to the Normalization Network (called the Style Prediction Network). The primary different between our findings is that the Style Prediction Network in the Ghiasi paper is Inception network pretrained on image classification, while our Normalization Network is be trained end-to-end along with the Image Transformation Network.

3 Approach

3.1 Methods

At each of the layers described above, the features for the pastiche image and either content or style image are extracted. If the layer is used for content loss computation, then the content feature is extracted, and similarly for style. In the Gatys approach, to compute the content loss, we simply find the Euclidean distance between the two feature maps. However, we find that a simple change produces better results: instead of multiplying by the content weight after all of the content losses are summed, we multiply the matrices by the content weight before we take the distance. This then makes the equation for content loss computation:

$$\mathcal{L}_{content} = \sum_{l} \sum_{i,j} (\alpha C^{l}_{i,j} - \alpha P^{l}_{i,j})^2, \qquad (1)$$

where C^l and P^l are the features of the content and pastiche image extracted at layer l in the content layers, respectively, and α is the content weight.
A similar approach is taken to compute the style loss. However, the we must first compute the Gram matrix of the feature maps:

$$G^l_{i,k} = \sum_k F^l_{i,k} F^l_{j,k}, \tag{2}$$

for a given feature map F. We then use the Gram matrices, instead of the feature maps, to compute the style losses, again multiplying the weight before the computation:

$$\mathcal{L}_{style} = \sum_l \sum_{i,j} (\beta G^{s,l}_{i,j} - \beta G^{p,l}_{i,j})^2, \tag{3}$$

where $G^{s,l}$ and $G^{p,l}$ are the Gram matrices of the features of the style and pastiche image extracted at layer l in the style layers, respectively, and β is the style weight.
The total loss is simply the sum of the content and style losses:

$$\mathcal{L}_{total} = \mathcal{L}_{content} + \mathcal{L}_{style} \tag{4}$$

3.2 Architecture

As mentioned previously, the entire system consists of three networks: a pretrained Loss Network, an Image Transformation Network, and a Normalization Network. As proposed by Gatys et al. [5], the Loss Network is the VGG-19 Network [17], and the layers for feature extraction were CONV_1_1, CONV_1_2, CONV_1_3, CONV_1_4, CONV_1_5 for the style loss computation, and CONV_1_1 for content loss computation. The Image Transformation Network is almost exactly as defined as in [10]; the only difference is that we use a hard tanh instead of scaled tanh at the last layer of the network.
The Normalization Network is a simple network with three convolutional filters and two affine layers. The input size for the network is a 256×256 image. All convolutional filters are 9×9 with stride 2 and no padding. The first convolutional layer has an output channel size of 32; the next, of 64; and the final, of 128. The first affine layer has weight dimension of 625×256; the second has dimensions of 256×32.
The output is a 128×32 matrix. Each column in the matrix provides a gamma or beta for each of the 16 instance normalization layers of the Image Transformation Network. The largest feature dimension for any of the layers is 128, hence the first dimension of the output matrix. The extra parameters in the matrix are simply not used in the layers; since they will accumulate 0 gradients over the course of training, they do not pose any computational load on the process. See Fig. 1 for a diagram of the network architecture.

Because these parameters are used as scaling and shifting factors of the normalization layers, it is crucial that the final affine layer of the Normalization Network produce Gaussian distributions of low variance and mean 1 and 0 for the gammas and betas, respectively. We were able to accomplish this by multiplying all the randomly initialized parameters of the final affine layer by a very small weight, which we tune as a hyperparameter, and that the biases of that layer are initialized to 0 for columns corresponding to gamma values and 1 for those corresponding to beta values, thereby shifting the mean of the distribution [Fig. 2]. Ultimately, this design decision was what allowed the network to converge when multiple images were used as the style set.

4 Experiments

We use the model as described above for two tasks: overfitting on relatively small sets of a few styles and generalizing to all styles using a large dataset of training styles.

4.1 Data

The MS-COCO Captions Dataset [12] was used as the training set for the content images. A simple set of hand-picked paintings was used for the overfitting task; the Painter by Numbers Dataset [16] was used as the training set for the style images for the generalization task. The MS-COCO Dataset and Painter by Numbers Dataset each contains roughly 80k images.
The images from all sets (including the hand-picked set) were shaped into a 256×256 images. Aside from the scaling and cropping necessary to achieve this, there was no additional preprocessing on the images. During the overfitting task, the content set was sampled from ran-

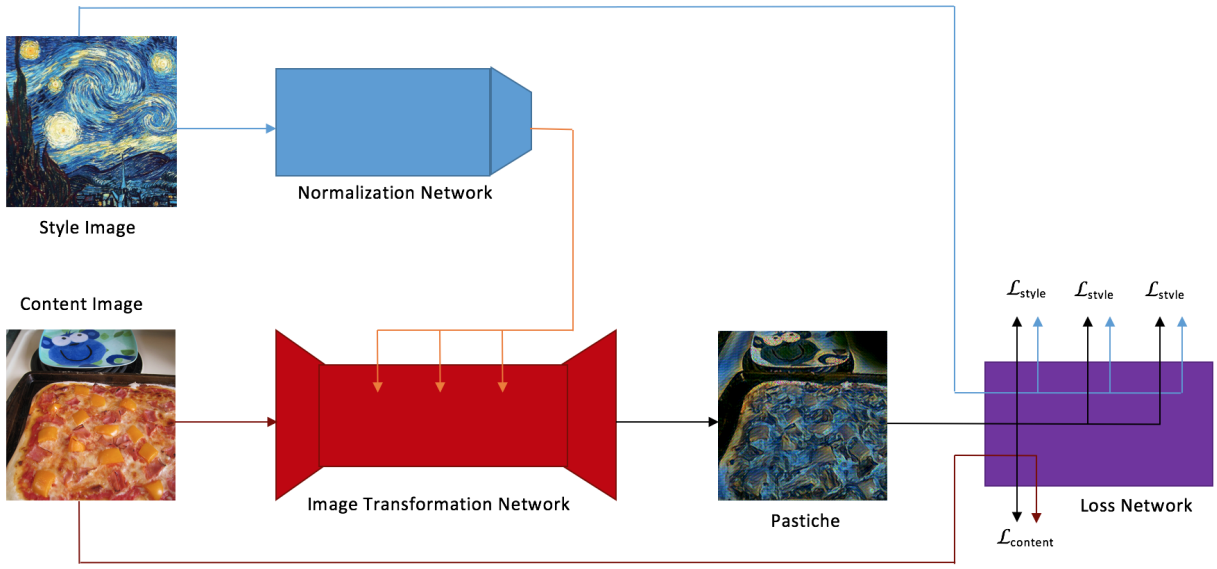


Figure 1: Network architecture: style image is passed into Normalization Network to produce instance normalization parameters, which are fed into Image Transformation Network. This in turn transforms content image into pastiche image, which is used along with the original style and content images to compute the style and content losses.

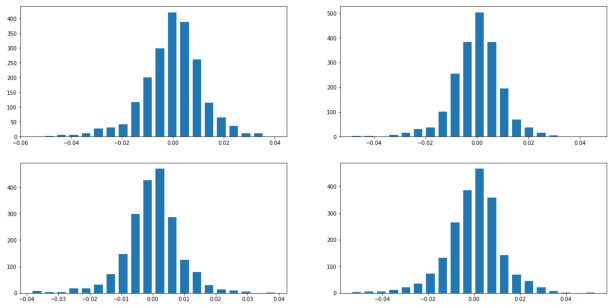
domly with batch size of four, and a small set of four paintings was looped though in a fix order with a batch size of one. During the generalization task, a random batch from both the content and style dataset was picked, with batch size of four and one, respectively.

4.2 Multiple Styles

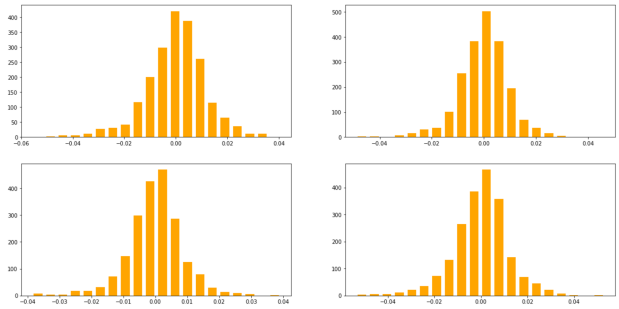
We ran two experiments with multiple styles, each using a unique set of paintings for the other (Fig. 3). The first set had 4 paintings, the second set had 16 paintings. It was found that a 5:1000 ratio of content to style weight was best to represent these sizes of the datasets.

The results of the four-painting experiment clearly show a single objective function that was able to learn multiple styles (Fig. 3a). The Normalization Network was able to be trained to represent multiple styles alongside the training of the Image Transformation Network. At first glance, it seems that the main feature that the network seems to represent is color. However, closer inspection shows that texture, such as smoothness of brush stroke and sharpness of edges, were also transferred onto the pastiche images. Pastiches of different styles seemed to exhibit similar feature even across styles. For example, in the first experiment, both the results of *Candy* and *Udnie* seem to have similar jagged edges and structures, albeit drastically different color schemes. This seems to suggest that the two paintings can be thought of as ”close” in style.

The results of the sixteen-painting experiment also seem to show a diverse set of styles being represented by just one function (Fig. 3b). There is much more noise in these images, and there seems to be more focus on texture transfer than color transfer; we believe that some additional hyperparameter tuning will be able to solve these issues. What is interesting to note is the fact that an unexpected hyperparameter had to be tuned in order to get this experiment to succeed: we had to change the standard deviation of the initial values of the Normalization Network to 1e-1 instead of 1e-2. In other words, instead of multiplying the weights of the Normalization Network’s final layer by 1e-



(a) Normalization weights (output columns 1-16)



(b) Normalization biases (output columns 17-32)

Figure 2: Normalization Network outputs: distributions of normalization weights (a) and biases (b) of four paintings (from left to right, top to bottom) *Candy*, *Udnie*, *The Starry Night*, and *Great Wave Off Kanagawa* as done for the four-style experiment.

2, we multiplied it by 1e-1. An explanation for this lies in the fact that the histograms of many styles will begin to look very similar if they have a small standard deviation. It therefore becomes difficult for the network to learn to differentiate them. In fact, before we adjusted this hyperparameter, all pastiches, no matter the style input, would be stylized in the same way, as if all the style images were being blended during training. This was what motivated us to increase the Normalization Network output variance, giving us more desirable results.

4.3 Style Generalization

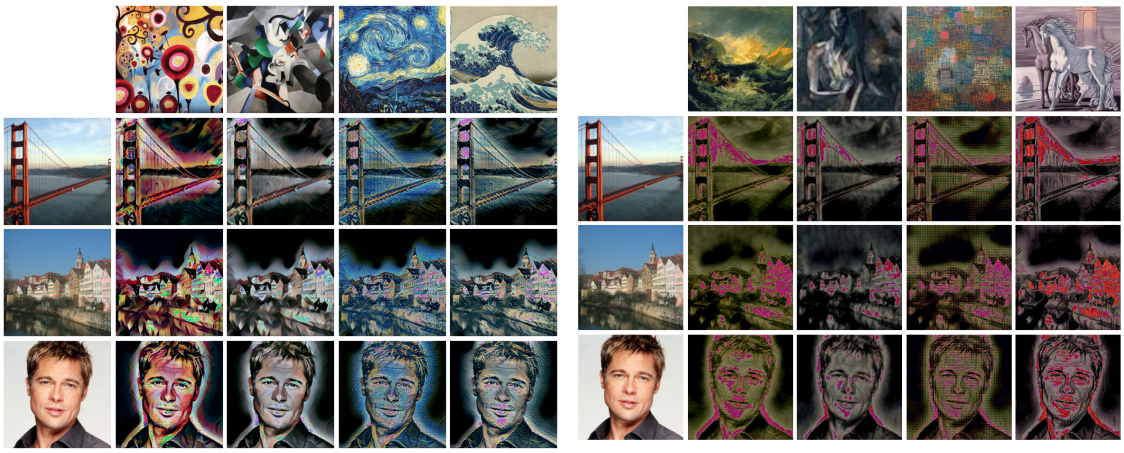
We ran four attempts of the generalization task, varying the hyperparameters over the course of the experiments. The results show the pastiches of two unique contents and styles; both of the styles had been seen by the network in training (Fig. 4) The first three attempts took place before we conducted the sixteen-painting experiment as described above. Similar to the problem we would later face in that experiment, we saw a ”blending” of all the styles in the pastiche images. No matter the style image that was input, the pastiche would look the same, always appearing muted in color.

We attempted to vary only the content-style weight ratio and learning rate to attempt to solve this issue. However, after doing the sixteen-painting experiment, we were motivated to change the value that was being multiplied to the Normalization Network's final layer from 1e-2 to 1e-1. Although it seems the only difference between the pastiches is color schemes which don't have much relation to the style, the fact that they *are* different colors is promising. It suggests that the network is learning unique styles, albeit not nearly as well as we'd like. There is much more work to be done in terms of tuning hyperparameters to get a more appealing result, but the different colors produced in the final experiment is at least promising.

5 Conclusion

By building out a network which allows the style image to be arbitrary by means of a secondary Normalization Network, we are able to train a network on multiple diverse paintings and even produce passable results on the Kaggle Painter by Numbers Dataset of 80k images. We have demonstrated that it is indeed possible to learn one objective function which can produce pastiche images of many styles. We can train all networks involved in this process end-to-end; there is no need to rely on pretrained networks at any stage. It may also be possible to train a network end-to-end that can generalize to all styles; however, the evidence for this hypothesis is current inconclusive.

The clear next step in this project is to continue to tune the hyperparameters of the generalization experiment to produce more appealing pastiches. The tuning of the Normalization Network output standard deviation will be crucial in being able to train the network to generalize to many styles. In addition, the experiments with multiple styles had considerable amount of noise; this can most likely be remedied by tuning hyperparameters as well. Essentially,



(a) Four-painting experiment

(b) Sixteen-style experiment (four styles shown)

Figure 3: Multi-style experiments

more experiments and hyperparameter search could give us much better results for both generalization and multiple styles.

Our current network can also be used to study artistic creativity. The fact that both the Normalization Network and Image Transformation Networks were trained at the same time is very intriguing. While pretrained networks produce successful results because they are trained to recognize features in an input, we have been able to train a network whose only task is to find the unique representation of a painting's style. This Normalization Network may therefore be able to give us more insight into human artistic creativity, giving it possible applications in psychology and art history. The relatively simple architecture of the Normalization Network also tells us that the features necessary to learn to uniquely represent artistic style are not as many as one may expect. These insights brought upon by the Normalization Network are all potential paths to further investigate in next steps.

Acknowledgements

We would like to thank Alexis Jacq [9] for providing a great starting place for the code. We would also like to thank PyTorch [2] for creating an incredible library that allows more and more aspiring academics to get involved in machine learning.

References

[1] Tian Qi Chen and Mark Schmidt. Texture synthesis by non-parametric sampling, 2017.

[2] Soumith Chintala and Adam Lerer. pytorch. `https://github.com/pytorch/pytorch`, 2015.

[3] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. 2016.

[4] Alexei A. Efros and Thomas K. Leung. Fast patch-based style transfer of arbitrary style, 1999.

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

[6] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. 2015.

[7] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. 2017.

[8] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. 2017.

[9] Alexis David Jacq. Pytorch-tutorials. `https://github.com/alexis-jacq`, 2017.

(a) Content:style weight: 5: 1000; Normalization Network output weight: 1e-2; learning rate: 1e-3

(b) Content:style weight: 5:1000; Normalization Network output weight: 1e-2; learning rate: 1e-4

(c) Content:style weight: 5:2000; Normalization Network output weight: 1e-2; learning rate: 1e-4

(d) Content:style weight: 5:1000; Normalization Network output weight: 1e-1; learning rate: 1e-3

Figure 4: Generalization experiments

[10] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. 2016.

[11] Li-Yi Wei Marc Levoy. Fast texture synthesis using tree-structured vector quantization, 2000.

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollr. Microsoft coco: Common objects in context. 2014.

[13] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them, 2014.

[14] Alexander Mordvintsev, Michael Tyka, and Christopher Olah. deepdream. `https://github.com/google/deepdream`, 2015.

[15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks, 2016.

[16] Kiri Nichol. Kaggle painter by numbers dataset. `https://www.kaggle.com/c/painter-by-numbers/data`.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. 2016.

[19] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. 2015.

[20] Hang Zhang and Kristin Dana. Multi-style generative network for real-time transfer, 2017.