Hierarchical Multi-Style Transfer

Junyi Qiu
UC San Diego
9500 Gilman Dr, La Jolla, CA 92093
juq005@eng.ucsd.edu

Jason Quach
UC San Diego
9500 Gilman Dr, La Jolla, CA 92093
j6quach@gmail.com

Figure 1. Style transfer on Geisel Library.

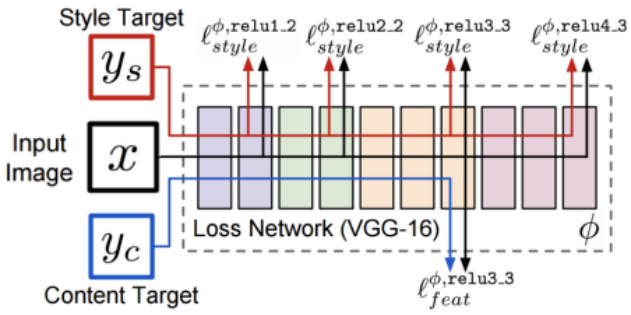

Figure 2. Iterative Based Approach.

Abstract

We explore the realm of style transfer based neural networks and perform a case study between alternative loss network architectures. In addition, we studied two of the most common style transfer approaches, the iterative based method and feed-forward transformation network method. Lastly, we develop a novel technique for combining multiple styles on a single mixed image. By assigning each style a subset of the available convolutional layers from some particular loss network, we can control what specific features in the layer hierarchy to mix together in the final result. This ultimately results in a coherent image that seamlessly blends together styles along its hierarchy.

1. Introduction

With the advent of the first style transfer paper [2] which highlighted the idea that style and content feature representations found within Convolutional Neural Networks trained on object recognition were separable for the most part, we can exploit these separable representations to mix together 2 different image sources one of which provides the content and the other the style. The resulting synthesized image maintains the global arrangement from the content image while the texture, color, and local features have been inherited from the style image. This is done by minimizing filter responses between the source image and a white noise image in order to produce visual information that is encoded at any particular layer. Minimizing multiple layers with either a style or content loss effectively allows us to combine information at different levels of the hierarchy to produce a visually appealing mixed image.

2. Case Studies

The largest constraint with style transfer is the computational expense it consumes, its sensitivity to noise and its consistency on the content and the style. Thus, here we perform a few case studies in order to explore the different possible approaches towards improving consistency and computational efficiency on images.

2.1. Iterative Based Approach

The first neural style transfer algorithm proposed by Gatys et al. [2] is based on the iterative approach. It takes a randomly generated image as the initial result, and iteratively change the value of its pixels to capture both features of the content and the style under the guidance of the pretrained neural network (like VGG-19). This is very similar to the CNN training process, except the variables to be trained are the values of the pixels on the input image instead of the weights in the neural network.

Figure 3. Comparison on stylic effect between AlexNet and VGG-19.

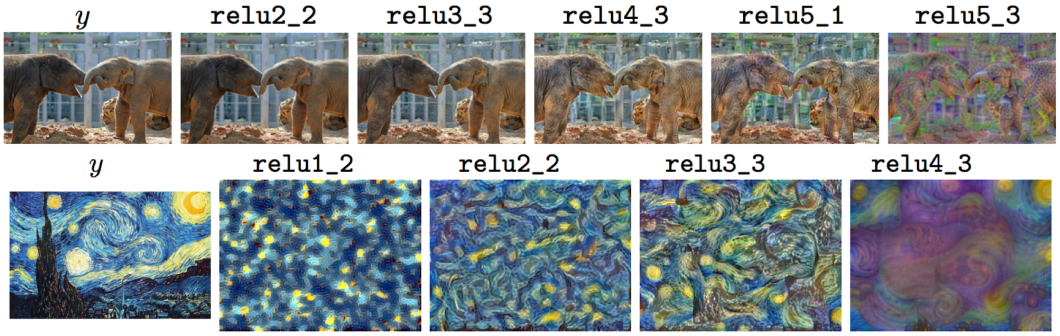

Figure 4. Comparison on reconstruction effects among different layers. Row 1: Content reconstruction. Row 2: Style reconstruction. The loss network is VGG-19.

Gatys et al. used two different losses to measure how far the generated image is from the target content and the target style. The content feature directly uses the outputs of layers, since the loss network is self is a powerful CNN for object recognition. Thus, the content loss of the generated image is defined as:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 \tag{1}$$

where $\vec{p}$ and $\vec{x}$ are the target content image and the generated image, and P^l and F^l are their respective content representations in layer l.

The style feature uses the correlations between the different filter responses in layer l, the Gram matrix. It is originally designed to capture texture information in a texture synthesis algorithm [1]. And the style loss is defined as:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}, l) = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \tag{2}$$

$$= \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (\sum_k F_{ik}^l F_{jk}^l - \sum_k P_{ik}^l P_{jk}^l)^2 \tag{3}$$

where $\vec{a}$ and $\vec{x}$ are the target style image and the generated image, A^l and G^l are their respective style representations in layer l, and P^l and F^l are their respective vectorized feature map in layer l.

The total loss for generating an image combines both two losses:

$$\mathcal{L}_{total}(\vec{p}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x}) \tag{4}$$

but with different weights.

The iterative approach has high computation capacity and takes very long time to generate one image. But it is flexible to combine any style with the content images.

Figure 5. Rows 1,2: Using max pooling layers with reconstruction loss layers 0-8,9-15. Rows 2,3: Using average pooling layers with reconstruction loss layers 0-8,9-15.

2.2. Alternative Loss Network

Most neural style transfer algorithms use VGG-19 as the loss network. This network is very deep and has many convolutional layers, so it has high computation capacity. We change the neural network in the loss backend to AlexNet, and compare the stylic effect between two network architectures.

The resulting images given by VGG-19 on average have higher consistency of the content and the style than those from AlexNet. The images generated by AlexNet have several problems. They do not have the same brushing texture as the target style image. And they also keep original boundaries and shapes of objects in the images. Some of these images even have glitch regions. However, the images generated by VGG-19 have learning more details from the original style, like the brushwork, the merging and reshaping of objects and so on. Both network architecture can transfer the colors of the style well.

The difference in the style transfer effect comes from the difference in their capabilities to abstract feature representations from images. AlexNet is a small CNN, trained for 10 classes objects, while VGG-19 is a large, very deep CNN, trained for 1000 classes objects. So the representations given by VGG-19 are stronger and richer in details than the ones of AlexNet.

2.3. Effects from Different Layers

In the transfer algorithm proposed by Gatys et al. [2], the outputs from one high layer are used for the content loss, and outputs from several layers, ranging from low to high are used for the style loss. We compare the style transfer results by different layers to know the difference in their effects.

As for the content feature representation, the lower layers can preserve the color, shapes and position of objects in the images, while the higher layers only preserve the basic contours and relative positions, but allow more freedom in colors and specific boundaries. The same thing appears in the style representation. The lower layers generate tiny texture patterns and preserve the original color. But the higher layers can generate style patterns in the large scale, preserve the brushwork, but allow more freedom in colors.

The difference between the lower layers and the higher ones is due to the numbers of pooling layers. When there are more pooling layers between the image input and the layer, the outputs of that layers will correspond to larger regions on the input image, and provide large-scale representation. The difference of the style representations from lower and higher layers also gives us inspirations on how to apply multiple styles onto one content image.

Figure 6. 3 Styles transferred to a single content image. It becomes noticeably difficult to see the content image but the presence of each style can be seen

3. Hierarchical Multi-Style Transfer

3.1. Method

We modify the network architecture to allow for multiple styles to be inserted into the network. This involves assigning each style a subset of the available convolutional layers and adding a reconstruction loss layer to each of the assigned subsets. Afterwards, we calculate the gradients per reconstruction loss layer for each style with respect to each pixel in our final image. This also requires modifying the original loss function from [2] in order to achieve this extension.
Original loss function [2]:

$$\mathcal{L}_{total}(\vec{p},\vec{x}) = \alpha\mathcal{L}_{content}(\vec{p},\vec{x}) + \beta\mathcal{L}_{style}(\vec{a},\vec{x}) \qquad (5)$$

Our loss function:

$$\mathcal{L}_{total}(\vec{p},\vec{a_i},...,\vec{a_n},\vec{x}) = \alpha\mathcal{L}_{content}(\vec{p},\vec{x}) + \sum_{i=1}^{n}\beta_i\mathcal{L}_{style_i}(\vec{a_i},\vec{x}) \qquad (6)$$

We now have n styles each with its own respective weight, β_i and reconstruction loss layers $\vec{a_i}$. They are all simultaneously minimized in the combined loss function to generate the final mixed image.

3.2. Experiment

A VGG-19 loss network pretrained for image recognition is used in order to extract our feature maps from each convolutional layer as in the original paper. However, the choice of reconstruction loss layers per style is kept to the beholder. In addition, we remove the FC Layers because they are unnecessary for this task. The notion of hierarchy simply refers to the feature maps that are conveyed along the convolutional layers of our loss network. Lower layer styles will capture smaller features with lower complexity but as you travel through the hierarchy your receptive field size and complexity increases with higher layer styles. With this intuition in mind interesting mixtures can arrive now that you can selectively have styles with different hierarchical features.

This certainly imposes the idea that in order to exhibit different noticeable style features in our final image, it would be ideal to initialize the network with reconstruction loss layers pertaining to each style as continuous segments sub-divided by average or max pooling layers. This effectively maximizes the difference in hierarchical features among the styles. Furthermore, with the added ability to tune the weights of each style we can suppress or enhance multiple styles along the network enabling users additional control over the output.

We use the iterative approach to generating these images in order to maximize style transferring and use gradient descent as our optimizer. These images were generated using a GTX 1080 Ti with all multi-style transfer images being 512x512 pixels in size with 500 iterations.

With two style we create an architecture with reconstruction loss layers half split between the first and second style, all image have the same weights. As we can see in Figure 5, the left styles convey smaller features such as color and texture while the right most style appears in the more global and spatial context within the mixed images. We also compare the effects of average and max pooling on the loss architecture. For the top two rows in Figure 2 we notice more defined features and complexity within the images under max pooling while the bottom rows exhibit a more smooth texture while still retaining the presence of both styles.

We extend this application further to even 3 styles as seen in Figure 6. The architecture used for this application had the 3 styles assign to convolutional layers 0-4,4-12,and 12-15 respectively with the content image having a weight 12 and the layers having weights of 10, 20, and 10. All three styles do seem apparent in the image, however the presence of the content image itself is reduced. The outlines from the first style can be seen along the contours or edges of the image, while the second style can be seen intermediately with spirals appearing within the image. The third style can be seen globally in the spatial sense around the background as well as in the form of each character's body shape as well. It becomes increasingly more difficult to notice the affects of later styles along the hierarchy and only under careful selection of styles or weight-tuning can one make all styles present in this manner. Max pooling is used instead as it would appear to have more defined features within the image.

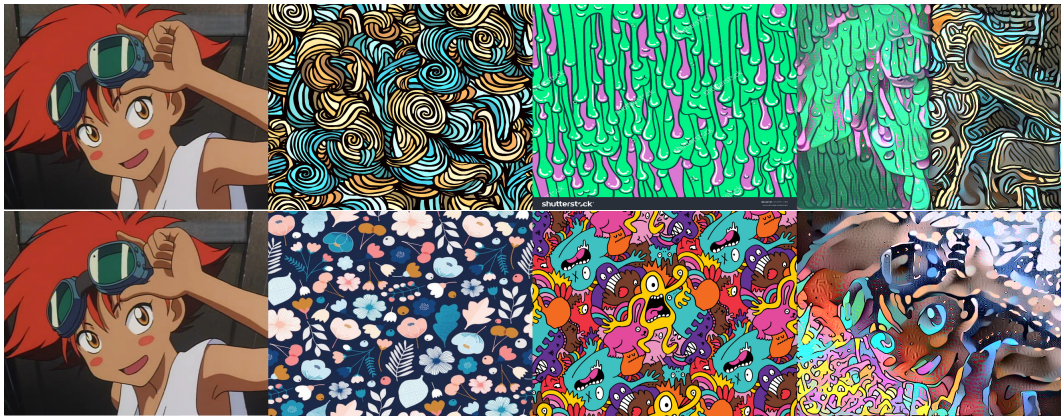

Figure 7. Using masking layers to suppress styles spatially

3.3. Spatial Multi-Style Transfer

Another experiment we performed with multiple styles is the use of masking layers to suppress style reconstruction gradients spatially. Each style will have its own mask. This is a simple adjustment to the loss function and we simply append a normalized weighted masking layer to each reconstruction loss layer. This enables a smooth transition between styles while retaining content information within the image. The results can be found in Figure 7 where it exhibits multiple styles spatially under 4 different masks respectively to each style.

4. Discussion

With a multitude of different loss architectures to choose from, it may be possible to increase the number of styles that one can mix by increasing the number of convolutional layers. Perhaps the use of a deep residual network could offer more insight and capabilities towards extending multi-style transfer. Furthermore, since the number of hyper-parameters increases as we add more styles and convolutional layers there is much more experimentation that could be done to find optimal or visually appealing combinations of weights as well as layer subset selection with styles. Lastly, the code could be improved upon. Using a different optimizer such as L-BFGS may offer more appealing results

5. Conclusion

We conclude our paper by remarking on the fact that style transfer has gained large traction over the past years and there is still much to discover within its domain. The use of transformation networks increases its run-time capability and more recently a push towards utilizing optical flow to enable stable videos has begun under research. Here we discover a new way to stylize images utilizing multiple styles with a hierarchical thought process in mind in order to generate visually appealing images. Code is available at [3].

References

[1] L. A. Gatys, A. S. Ecker, , and M. Bethge. Texture synthesis using convolutional neural networks. *Advances in Neural Information Processing Systems*, pages 262–270, 2015. 2

[2] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style, 2015. 1, 3, 4

[3] J. Quach. Hierarchical Multi-Style Transfer. https://github.com/nu1ptr/hierarchical-multi-style-transfer. 5