Neural Combinations of Artistic Styles for Use in Multi-Style Transfer

Vivek Subramaniam
Columbia University
vs2575@columbia.edu

Nick Greene
Columbia University
ng2572@columbia.edu

Abstract

This paper aims to explore the use of different methods for combining two images of different artistic style using a neural network. The intention is that the combination style can be applied to a content image using standard neural artistic style-transfer algorithms to transfer multiple styles at once in a visually pleasing manner.

1. Introduction

Recent work has shown how neural networks are able to transfer the artistic style of one image, such a famous painting with a particular style, to a content image such as a photograph. We explore ways of modifying the neural style-transfer method used by Gatys et. al [6] to instead combine the styles of two different "style images" into one "multi-style image." This combined image can then be applied to a content image using the standard Gatys method as a way of transferring multiple styles at once. Our work explores how different methods of combining two style images into a multi-style image produce various apparent combinations of the two styles when the multi-style image is transferred to a content image.

1.1. Related Work

The application of convolutional neural networks to artistic style transfer found its roots in Gatys et al. 2015 [6]. Other approaches to artistic style transfer include the use of texture synthesis [3] or patch-based style-transfer [4] algorithms rather than neural style-transfer. However, there has been relatively little work investigating transfer of multiple styles simultaneously. To our knowledge, there is only one other project which directly investigates this problem aside from ours.

In this paper, the authors Cui et al. 2017 [1] train and utilize a neural style transfer network, which is a feed-forward CNN. They connect this to a pre-trained VGG-16 network [1]. However, results from this method of multi-style transfer appear to be only preliminary, and from a qualitative standpoint, they lack visually pleasing results for the transfer of multiple styles. Our work is unique not only in that we achieve more visually appealing results for transferring multiple styles simultaneously, but also in that we focus on the synthesis of two styles into one multi-style as an independent and modular step. This multi-style could be used for other applications rather than simply transferring multiple styles to an image.

Note: Gatys et al. have conducted further research on color transfer for neural style transfer. We do not investigate in this paper what role this color transfer work plays in combining multiple styles into one multi-style image.

2. Methodology

At its core, our multi-style method employs the core functionality of neural artistic style transfer seen in Gatys. We use a generative algorithm that performs iterative stochastic gradient descent to minimize a specified loss function. The major contribution of our work is how we modify the loss function proposed in Gatys in multiple ways to synthesize two style images into one combined style image. We then investigate how each original style manifested in the final output image after transfering the combined style to a content image. We have described these methods under the sections titled, "Varying α/β Weights," the "Zipper" method, and the "50/50" method.

2.1. Adopting Baseline Neural Style Transfer for Style Fusion (Gatys et al. 2015)

We utilize the VGG-19 deep learning model and the associated pre-trained ImageNet weights. Based on the methods used within the Gatys paper for artistic single-style transfer [6], we used feature extraction from the first five convolutional layers, and vary the influence of these feature extractions to the total loss function before using iterative gradient descent to generate an image which contains all of the extracted features at their respective influence levels. We chose VGG-19 over VGG-16 due to its rich feature representations for a large variety of images. We base our implementation of this style transfer algorithm off of the Keras example published on GitHub [2].

For fusing two styles using our methods, we will denote

the style images as S_1 and S_2, and the output fused style as f. For applying a style to a content image using the Gatys method, we'll call the content image C, and the style image s, and the output o. In this project, all images used are 400×400 pixels.

To perform the algorithm detailed in Gatys and also our modified versions, this first step is always to run s_1 and s_2 seperately through VGG-19 (ImageNet). We then use feature extraction from the first five convolutional layers of both VGG-19 instances to determine the total loss based on which loss function we are using. Gradient descent then iterates over this loss.

The full pipeline for combining two styles and then applying the result as a multi-style-transfer on a content image is as follows:

$$s_1 + s_2 \rightarrow f$$
$$f + c \rightarrow o$$

2.2. Stochastic Gradient Descent and Loss Function

After intercepting the first five layers of both VGG-19 instances, we perform feature extraction and loss calculation per layer. These loss values are all weighted in their contributions to the total loss function:

Because our loss functions are nearly identical to what is detailed in the Gatys paper aside from two modifications, we will only detail our modifications here. The full explanation of the loss functions can be seen in the Gatys paper.

Here we show our first modification: we have changed the total loss as follows:
Gatys Total loss is (L_{total}) as follows:

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

Our Total loss is (L_{total}) as follows:

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{style1}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style2}(\vec{a}, \vec{x})$$

Now we'll show the only other modification which is is in how L_{style} is computed.

Gatys style loss is computed exactly as:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l.$$

where w_l is strictly equal to $\beta/4$.

We calculate L_{style} in the same way except we manually vary all 5 w_l weights. To clarify, $w_1, w_2, ..., w_5$ are the loss weights corresponding to the feature extraction of the first five convolutional layers of VGG-19.

A very important point which can not be overlooked is that for all examples, the generative gradient descent iteration was not run beginning with random white noise. Instead of white noise, the gradient descent process began using S_1 ("style1") as a starting point when combining two

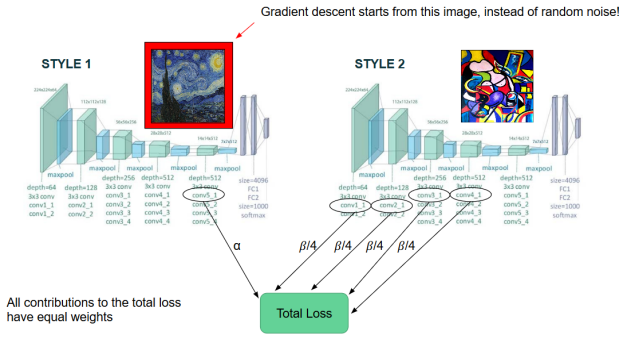

Figure 1. (α/β) weight setup

style images S_1 and S_2. Generation from white noise required upwards of 10,000 iterations to produce an image with only fair quality. By beginning generative iteration from whichever image was S_1 when combining two style images, we were able to reach relatively high-quality outputs after many orders of magnitude less iterations. We used 60 iterations for all cases of gradient descent in this paper, which were sufficient to produce an image representative of both styles and proper content.

Next we will see all of the ways in which we varied the weights for loss contribution.

2.3. Varying α/β Weights

As consistent with Gatys, we denote α and β as weighting factors for content and style reconstruction respectively. In this section, for s_1, $w_1 = w_2 = w_3 = w_4 = 0$, and $w_5 = \alpha$. For s_2, $w_1 = w_2 = w_3 = w_4 = \beta/4$, and w_5 = 0. We ran this setup multiple times on the same sets of images while varying the ratio α/β. This setup is very similar to the loss function in the Gatys standard style-transfer algorithm. The architecture can be seen in Figure 1

2.4. The "Zipper" Method

While we found some success in varying the α/β weights of the previous setup in an attempt to "evenly" combine multiple artistic styles, we also used a more novel set of weights which is a much more significant modification to the Gatys style transfer algorithm. we call it the "Zipper" method. The weights are as follows: $\alpha = \beta = 1$, for s_1, $w_1 = 1, w_2 = 0, w_3 = 1, w_4 = 0, w_5 = 1$, and for s_2, $w_1 = 0, w_2 = 1, w_3 = 0, w_4 = 1, w_5 = 0$. In Figure 2, the weights of value 1 are represented by arrows, and weights of value 0 are represented by a lack of arrows.

2.5. Gaussian Blurring on Zipper Output

After "zipping" the two styles together $s_1 + s_2 \rightarrow f$ seemed to yield promising results, it also appeared that there were some very high frequency edges and changes in the

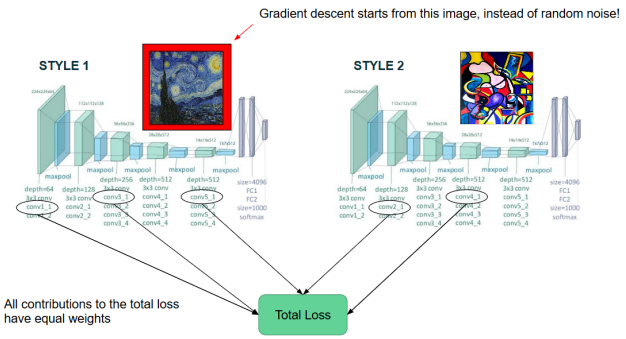

Figure 2. "Zipper" method weight setup.

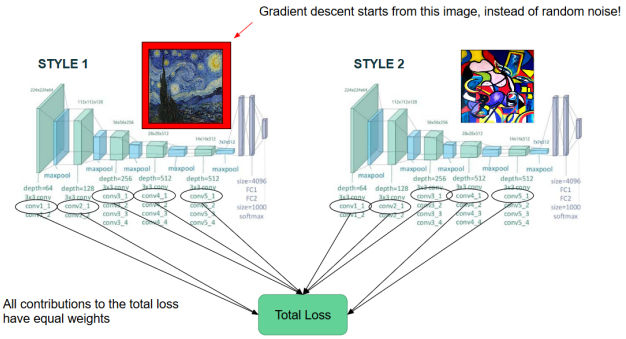

Figure 3. "50/50" method weight setup

combined style image and also in the final output after transferring the zipped style to a content image. We attempted to combat this with post processing. We applied Gaussian blurring with different kernel sizes to f before applying f to a content image to create a multi-style transferred output, $c + f \rightarrow o$.

2.6. The "50/50" Method

A last approach for combining styles was to have all $w_l = 1$ for both S_1 and s_2. This was in an effort to create a truly even combined style which equally contains elements of both extracted component to the loss function. The results of this method can be seen in Figure 3.

3. Results and Discussion

For our α/β weight setup, we first use a ratio $\alpha/\beta = 4 \times 10^1$, and we noticed that the features from s_2 showed up much more strongly than the features for s_1. This is because s_2 contributes lower layer-level information to the loss function for the generation of the combined style f. Then, transferring f to c does the same for the resulting o. This did not appear to have an even blend of apparent features of s_1 and s_2 in o. To attempt to create more even representation of s_1, and s_2 in 0, We increased our ratio to $\alpha/\beta = 4 \times 10^3$, and again to $\alpha/\beta = 4 \times 10^5$ to great success. See Figure 8.

While this gave improved results, we still had mostly lower level feature content from s_2 and higher level feature content from s_1 apparent in o. This is when we created the "zipper" weights. The zipper weights attempt to capture both lower and higher level feature information from both s_1 and s_2. This was very successful. See Figure 5.

However, we noticed that there were lots of sharp edges for s_1 "zipped" with s_2 to get f_{zip}. We applied Gaussian blur of kernel sizes 3 and 5 to f_{zip} before applying the result to a content image. We found that the smaller kernel size worked well for cleaning up the sharp edges in the final output o, to make it look more natural. A larger kernel size made o very blurry. We would've liked to go for an even smaller kernel size, however we were unable to since $3x3$ is the smallest possible Gaussian kernel size, and our images were only $400x400$ pixels. If we had larger images, a small Gaussian kernel would do well. Another possible way to reduce sharp edges in f_{zip} would be to run the gradient descent on white noise for approximately 100,000 generative iterations. Our Gaussian blur approach has the advantage of being much faster since we started gradient descent from s_1 instead of noise, and used a total of only 60 generative iterations. See Figure 4.

The final results of our investigation come from the "50/50" weight setup. We found that these results looked more "smudged" than the zipper setup, however, similar to the zipper setup, the results both in the combined style f, and on applying f to a content image to get o seemed to have a balanced visual representation of both s_1 and s_2. We feel that, especially for this "50/50" setup, we would obtain much different results if we were to begin the gradient descent on white noise rather than s_1. Equal weighting would mean that s_1, and s_2 would be "fighting" over features at all feature layer-levels in their contributions to the loss function, whereas in all our other weighting setups, "zipper," and "α/β", there are no overlapping layer-level contributions to the loss function. This would explain why our results for "50/50" look significantly more blurry than for "zipper." See Figure 7

3.1. Future Work

For the future, we would like to investigate how we can incorporate color transfer into this project. Gatys et al. 2016 published a second paper on color transfer using color histogram matching and luminance-only transfer [5]. Perhaps these techniques can be adopted in a similar manner to allow for color transfer in the neural multi-style transfer process.

In addition, we would like to explore the possibility of building off of this method to recreate art in the style fusion of three or more artists/works. While we have shown that multiple single-style transfers works for two images un-

der proper α/β values and content/style representation extraction, the configurations required for more than two style sources may differ in order to combine and produce features in a visually pleasing way.

Also, when experimenting with α/β weighting in the total loss function, we only altered the value of the content and style weights in the first pass through the VGG; we can also alter the α/β values in the second pass of the VGG (from the constant ratio $\alpha/\beta = 1.0/0.025 = 4 \times 10^1$) to see how changing this will result in a different final stylized image.

Finally, the images used within this project used famous paintings as styles and photographs as content bases. We can experiment with non-painting related images (i.e. other photographs with special effects) to see how images from different domains and eras work with these method. We can also experiment with images of larger sizes; the images used in this project were of size 400×400 pixels, so we can check to see if this method can be extrapolated and also if Gaussian blurring would have a more substantial/useful effect without worsening results.

4. Division of Project Responsibilities

All parts of this final project were completed by Subramaniam and Greene. Research, conceptualization of the idea, setup of the necessary infrastructure, implementation of the code, creation of the presentation, writing of the paper, etc. were all completed together by both authors.

4.1. Code/Presentation Slides

Code and notebooks for this project can be found at `https://github.com/vsub21/neural-multi-style-transfer`.
Slides can be found at `https://tinyurl.com/vs2575-coms4731-proj-slides`.

References

[1] C. Q. B. Cui and A. Wang. Multi-style transfer: Generalizing fast style transfer to several genres. *Stanford CS231N Projects*, 2017.

[2] F. Chollet. Keras. `https://github.com/fchollet/keras`, commit: fcf2ed7, 2018.

[3] M. Elad and P. Milanfar. Style-transfer via texture-synthesis. *CoRR*, abs/1609.03057, 2016.

[4] O. Frigo, N. Sabater, J. Delon, and P. Hellier. Split and match: Example-based adaptive patch sampling for unsupervised style transfer. 06 2016.

[5] L. A. Gatys, M. Bethge, A. Hertzmann, and E. Shechtman. Preserving color in neural artistic style transfer. *CoRR*, abs/1606.05897, 2016.

[6] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

Figure 4. Zipper Method – "All or Nothing" results with 3×3 (left) and 5×5 (right) Gaussian Blur applied. S_1 is starry.jpg, S_2 is picasso.jpg

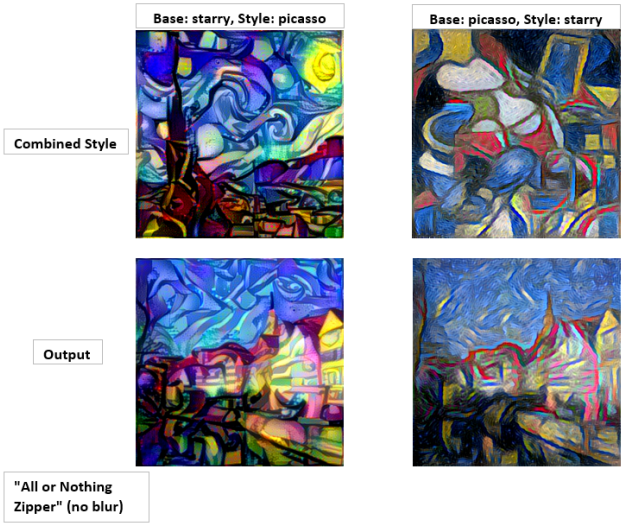

Figure 5. Zipper Method – "All or Nothing" results (no blur applied). S_1 is starry.jpg, S_2 is picasso.jpg.

Figure 6. Reproduced results using method from Gatys paper [6] for single-style transfer.

Figure 7. Zipper Method – "50/50" results. S_1 is starry.jpg, S_2 is picasso.jpg.

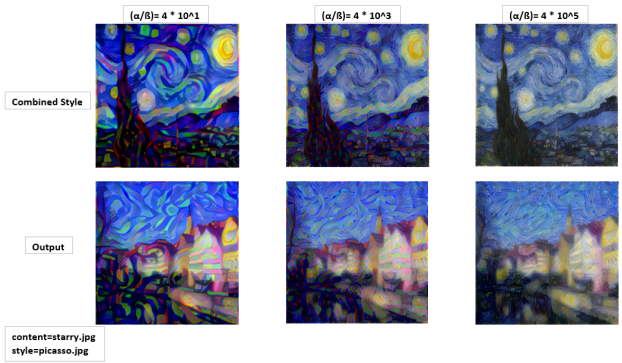

Figure 8. Results of varying α/β ratios. Ratios are 4×10^1 (left), 4×10^3 (center), and 4×10^5 (right). S_1 is starry.jpg, S_2 is picasso.jpg.