Picasso's Marilyn Monroe and Other Blends: Neural Style in TensorFlow

Kathryn Siegel
Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA 02139
ksiegel@mit.edu

Abstract

Artistic styles have evolved over centuries, with distinct styles developed in different regions and time periods. Certain artistic styles, such as Impressionism or Cubism, are immediately recognizable by the human eye, regardless of the content or subject of the images. This paper describes a system that uses a Convolutional Neural Network to separate and mix the style and content of images. The system implements the algorithm presented in "A Neural Algorithm of Artistic Style," and is the first such implementation using Google's Deep Learning library, TensorFlow [10]. This paper also presents novel implementations for combining multiple styles in one image and generating videos using the algorithm. Results are of equal quality to the aforementioned paper, and of equal or higher quality than other implementations found online.

1. Introduction

Paintings from certain eras in art history emulate distinct styles. For example, Impressionism is defined by small brushstrokes and flowing patterns, whereas Cubism is defined by abstract geometric shapes. Notably, content-agnostic styles define these artistic eras,. Our research studies a method for regenerating images in the style of a certain era, using one or more images from that era as a style source. These generated images, if not for anachronistic subject content, look as if they are from the era of the style input image(s).

Not all images generated using the style of one image and the content of another are visually appealing. Our work seeks to identify the qualities of style and content input images that generate pleasant results. We study multi-style blends and video generation, sampling many different artistic styles and subjects. This paper presents the most distinctive results from our experimentation.

Through this work, we also compare TensorFlow [10], Google's new machine learning engine, with Torch. Torch is one of the most popular machine learning systems used for deep learning; Torch implementations of the neural style algorithm exist [6]. We compare the time-per-iteration and required number of iterations for our TensorFlow implementation and the Torch implementation, identifying the more robust system.

2. Related Work

Our work is based on an algorithm presented by Leon Gatys, Alexander Ecker, and Matthias Bethge in their paper, "A Neural Algorithm of Artistic Style." In their paper, they discuss how Convolutional Neural Nets learn to recognize objects; the neural nets first learn to extract the distinctive aspects of the content of the image, and then use that information to identify the object. As a result, layers that an image encounters later in a neural net trained for image recognition capture aspects of the image relating strongly to its content. On the other hand, because of this layer-to-layer trend, we can capture the style representation of the image by comparing different layers and finding the correlation between corresponding features. Our methods draw heavily from the methods reported in the Gatys paper [13].

Several other implementations of this neural style algorithm exist, the most popular of which is Justin Johnson's open-source Torch implementation. Written in Lua, the system achieves results of equal quality to the paper, and also allows users to blend multiple styles into one image. While we used Johnson's results as a quality benchmark, we did not use his code to direct our implementation [14]. At the outset of our research, no other TensorFlow implementations existed for the neural style algorithm. However, one other implementation has since been published and references our implementation [16].

Additionally, several other individuals have attempted to create videos using the neural style algorithm. Each frame of these videos was created separately using the algorithm, so the video as a whole has visually disorienting discontinuities between contiguous frames [12]. Our research presents an improved method of video generation to eliminate these discontinuities for a more aesthetically pleasing result.

Figure 1. The third image above, generated by our implementation of neural style, combines the content of a photograph of the MIT Great Dome [1] with the style of Leonid Afremov's "Rain Princess" [11] using a decaying learning rate schedule.

3. Methods

We use a VGG-19 neural net to extract image content and style, as recommended by the Gatys paper [13]. We begin with an input image of random noise, and extract the content and style representation of this image. We also feed the content image and style image we want to combine through the VGG-19 neural net. The algorithm calculates loss functions for content and style, and then uses an Adam Optimizer for gradient descent. Let $\vec{c}$ and $\vec{x}$ be the original content and generated output images, and let C^l and X^l be the feature representations of these inputs at layer l. We calculate content loss using C^l and X^l at the relu4_2 layer of the VGG-19 neural net using the following equation, where w_c is content weight.

$$\mathcal{L}_{content}(\vec{x}, \vec{c}, l) = 2w_c * \frac{\sum_{i,j} (X^l_{ij} - C^l_{ij})^2}{size(C^l)} \quad (1)$$

Our approach differs slightly from the methods presented in the Gatys paper. We use mean squared error instead of squared error in the loss calculation, as the former produces better results. The Gatys paper uses layers other than relu4_2 in its calculation. For our style loss calculation, we compute the correlation between the feature responses of different layers of the neural net using a Gram matrix. Let G^l_x and G^l_s be the Gram matrices for the original style and generated output images, w_s be the style weight, and L be the set of layers at which we calculate style loss. L includes the layers relu1_1, relu2_1, relu3_1, relu4_1, and relu5_1. We calculate style loss using the following equation.

$$\mathcal{L}_{style} = w_s * \sum_{l \in L} 2 * \frac{\sum_{i,j} (G^l_{xi,j} - G^l_{si,j})^2}{size(G^l_s)} \quad (2)$$

Our implementation then uses back-propagation to adjust the input image such that its content and style representations more closely match the content representation of the content image and the style representation of the style image, respectively. Over many iterations, this input image of random noise is transformed into an image with the desired content and style. The examples in this paper were generated using 1000 to 10,000 iterations.

4. Results

Our implementation achieves results of equal quality to the results from the Gatys paper; figure 1 shows a sample result produced by our implementation [13]. In this section, I provide example results from our implementation and discuss our process for generating multi-style images and neural style videos. The code for our implementation can be found at git.io/style.

4.1. Core Implementation

Our implementation uses the algorithm introduced in the methods section to generate an output image; there are several parameters used in this training process. We tuned the algorithm parameters such that the output seemed balanced and aesthetically pleasing. These parameters included content weight, style weight, TV weight (which helps filter out noise), learning rate, style scale, and number of training iterations. In general, we found a content weight 2-3 orders of magnitude higher than the style weight led to the best results. These relative weights produced images that were immediately identifiable as the content image, did not significantly warp key edges, and imbued the style image strongly within the image. Figure 2 visually demonstrates the visual effect of modulating the content weight to style weight ratio from the values we determined to be optimal. A TV weight of about the same magnitude as the style weight eliminated out-of-place pixels that were the product of noise. While the learning rate had less of an impact on the outputted image, we obtained slightly more aesthetically pleasing results with a decaying learning rate. Furthermore, our implementation produced the best results when the style scale was left untouched, likely because the image style was more recog-

Figure 2. This sequence of images demonstrates the effect of modulating the content weight to style weight ratio. The top panel of the image shows the two images used to generate the blends: Picasso's "Le Rêve" [5] is the style input and a black-and-white photograph of Marilyn Monroe [4] is the content input. The bottom row of images ranges from a very low content-to-style weight ratio on the left to a very high content-to-style weight ratio on the right. The leftmost images demonstrate the extreme warping effect created by too high of a style weight. The rightmost images demonstrate how too high of a content weight results in the style being indistinguishable. Our default content-to-style weight ratio was 500, which was used to generate the fourth blended image.

nizable at its original scale. Finally, images produced by the system emulated the style image more thoroughly with more training iterations; we determined that training with over 2000 iterations produces the best results.

4.2. Multi-Style Support

Our implementation accepts multiple style inputs; users can specify the style blend weights corresponding to the inputted style images. We combined many different styles to determine the factors that would contribute towards an aesthetically pleasing result. We found that mixing blockier styles such as Cubism with more fine-grained styles such as Impressionism gave the best results. Specifically, when mixing a blocky style with a more detailed style, one can still clearly identify both styles in the resulting image. Blending two style of similar texture size produced images of ambiguous style and grainy image quality. Figure 3 demonstrates how the textures of the style inputs impact the aesthetic quality of the result. The style images in the top row both have fine-grained textures; the leaves in Kahlo's painting have small details, and Monet's painting is formed of the characteristic small brushstrokes of the impressionist era. The resulting image, shown as the fourth image in the top row, does not clearly emulate either style, and the blended style is noisy and aesthetically unpleasing. In contrast, the style images in the bottom row have blocky and fine-grained style textures, respectively. The resulting image, shown as the fourth image in the bottom row, clearly possesses aspects of both input styles; the two styles seem to complement, rather than conflict with, each other.

Furthermore, we allow users to specify style blend weights for the inputted style images. Varying the style blend weights of two style images revealed that fine-grain textures tend to dominate the appearance of the result. The Van Gogh-Picasso-Stata blend in figure 3 was generated by setting the Picasso style blend weight four times greater than the Van Gogh style blend weight. When we experimented with weighing the Van Gogh style more, we found that the defining aspects of Picasso's style in the resulting image became nearly indistinguishable.

4.3. Video Generation

We developed a novel approach to generating video using our implementation of the neural style algorithm. Our script runs the algorithm on each frame of the video separately, similarly to other implementations; however, we seed the training run for every frame with the previous generated neural style frame. This method produces sensible continuities between frames, significantly decreasing any disorienting jumps of the style's manifestation in contiguous frames. Sample videos can be found by visiting http://kt-siegel.com/neural-style-web/.

4.4. Comparison to Torch Implementation

Our implementation took approximately three times longer per iteration than the Torch implementation of neural style. Nonetheless, our implementation generated results of equal quality to the Torch implementation when run for

Figure 3. The figure above shows two different multi-style image blends. In each row, the content image input is shown first, followed by the two blended style images. The last image in each row is the output image produced by our implementation. The input images in the top row from left to right are a photograph of the Golden Gate Bridge [7], Frida Kahlo's "Self-Portrait with Thorn Necklace and Hummingbird" [9], and Claude Monet's "Lavacourt Under Snow" [8]. The input images in the bottom row from left to right are a photograph of the MIT Stata Center [15], Pablo Picasso's "Dora Maar" [2], and Vincent Van Gogh's "Starry Night" [3]

.

the same number of iterations. System inefficiencies clearly exist within TensorFlow; since it is a new library actively maintained by Google, we expect the performance of TensorFlow to improve over time as the system is refined.

5. Conclusion

Our implementation of "A Neural Algorithm of Artistic Style" generates aesthetically pleasing style and content blends given the appropriate input parameters. This work explores the optimal parameter settings for image blend generation and studies the effects of modulating these parameters away from their optimal values. Our approach diverges slightly from the Gatys paper; we calculate loss using a slightly different error metric and different layers of the neural net, and we use an Adam Optimizer for gradient descent rather than L-BFGS [13]. While we found the adjustments to the loss calculation improved the visual results, we could not integrate L-BFGS with our system, because it is not supported in TensorFlow. Future work could explore adding L-BFGS to TensorFlow, then substituting the Adam Optimizer with this alternative algorithm.

Regarding our style blending technique, the system generates visually compelling results given style inputs with very different texture sizes, but does not blend similar textures well. Future research could investigate methods to improve multi-style blends. Finally, our video processing technique produced visually coherent videos using the neural style algorithm, an improvement over existing open-source implementations.

References

[1] http://www.therpf.com/showthread.php?t=97257.

[2] https://s-media-cache-ak0.pinimg.com/236x/5f/c9/4f/5fc94f54df98d50bfae9b1807f3c5272.jpg.

[3] http://www.wikiart.org/en/vincent-van-gogh/the-starry-night-1889.

[4] Audrey hepburn or marilyn monroe? http://www.playbuzz.com/ufg201110/audrey-hepburn-or-marilyn-monroe.

[5] Le rêve. https://en.wikipedia.org/wiki/Le_R%C3%AAve_(painting).

[6] Torch. https://github.com/torch/torch7.

[7] https://en.wikipedia.org/wiki/File:GoldenGateBridge-001.jpg, 2007.

[8] https://commons.wikimedia.org/wiki/File:Monet,_Lavacourt-Sunshine-and-Snow.jpg, 2010.

[9] https://en.wikipedia.org/wiki/Frida_Kahlo, 2015.

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. War-

den, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[11] L. Afremov. Rain princess. http://afremov.com/RAIN-PRINCESS.html.

[12] M. Bartoli. neural-animation. https://github.com/mbartoli/neural-animation, 2015.

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

[14] J. Johnson. neural-style. https://github.com/jcjohnson/neural-style, 2015.

[15] L. Speck. Looking at mit stata center. http://larryspeck.com/2011/04/07/mit-stata-center/, 2011.

[16] woodrush. "neural art" in tensorflow. https://github.com/woodrush/neural-art-tf, 2015.