Whole-artist imitation with convolutional neural networks

Jesse Mu*‡, Andrew Francl†‡
Boston College
Email: *muj@bc.edu, †francl@bc.edu
‡*Both authors contributed equally*

***Abstract*—We implemented a machine learning algorithm described in a recent paper by Gatys et al [1]. The algorithm in the paper describes an optimization procedure using a convolutional neural network that enables a given photograph to be recreated in the style of other photographs, notably paintings. Specifically, the algorithm is able to take a painting and learn its stylistic composition rather than its content, then apply this stylistic information to create a new version of the image that stylistically matches the original painting. We create a Python command-line tool, `art.py`, that wraps around the algorithm and extends its functionality by enabling color-invariant replication according to a randomly selected work in an artist's oeuvre, increasing the versatility and authenticity of the algorithm's results. We also discuss possible extensions to the algorithm that will allow for even more realistic "whole-artist" replication in this vein.**

I. Introduction

Recently, a set of algorithms and methods in a field of computer science called deep learning has seen significant success in various machine learning tasks, far surpassing the performance of other statistical methods [2]. These systems, known as "neural networks" (NNs) (Figure 1) and loosely inspired by biological, connectivist models of human neuroanatomy, have been especially successful in computer vision, where a specific set of networks called "convolutional neural networks" (CNNs) mimic the complex perceptual systems in the human visual cortex [3]. In particular, hidden layers of deep CNNs are capable of learning hierarchical representations of objects by first analyzing individual pixels and then combining these pixels to make increasingly complex features (Figure 2).

A. Learning style

Similarly to how content representations are learned by CNNs, Gatys et al. discovered that CNNs can learn representations of style by examining certain "stylistic layers" in the network. The key discovery of the paper is that these representations are separable; that is, modifications to an image can change the responses in the stylistic and content layers relatively independently. This results in interesting potential for optimization, whereby the style and content of two distinct images are combined. This results in interesting potential for optimization, whereby the style and content of two distinct images are combined.

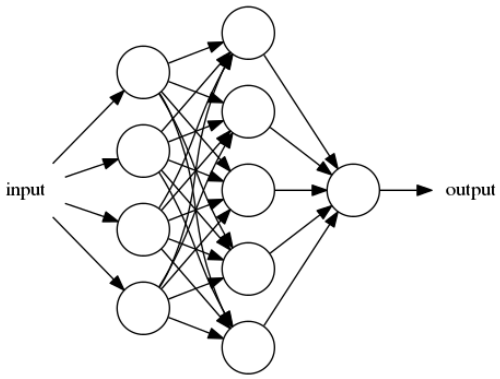


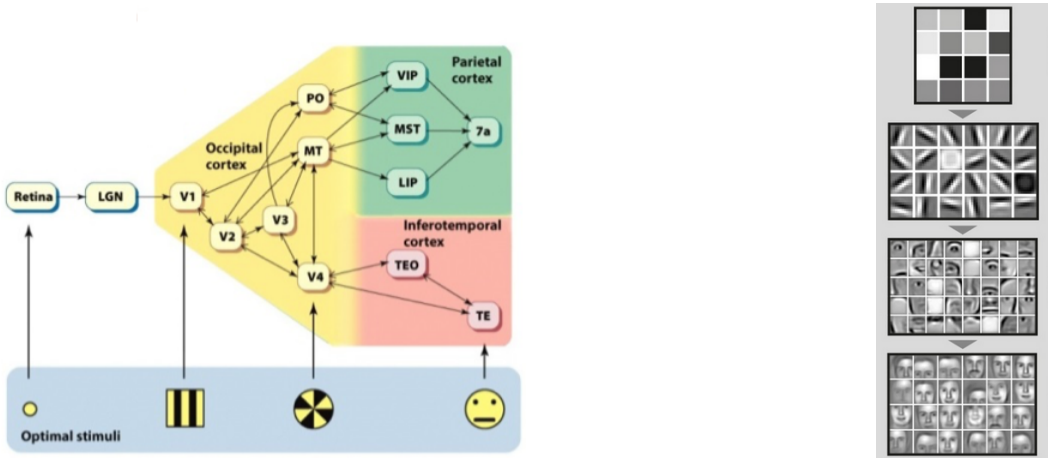
Fig. 1: Basic neural network structure, consisting of 4 input neurons (left), 5 hidden neurons (center), and one output neuron (right). Computation at each neuron is a relatively simple function e.g. sigmod activation function; numerous combinations of simple functions like these enable complex, nonlinear output.

II. Methods

We use the VGG-19 Deep Convolutional Neural Network [6] trained for stylistic representations by researchers at the Max Planck Institute for Biological Cybernetics [1]. Using this model with the Caffe deep learning framework [7], we extract high level representations of style and content by capturing the response of predetermined layers in the network.

A. Loss function

Using these stylistic features Gladys et al. create a loss function that jointly takes into account the stylistic



(a) Visual Cortex (b) CNN

Fig. 2: Comparison of object recognition hierarchies in (a) the human visual cortex and (b) a convolutional deep neural network. Processing complexity increases as layers become deeper in both models. Photo credit: (a) [4], (b) [5].

distance from the candidate image to the style image and the content distance from the candidate image to the content image. By jointly minimizing this function, the style of the candidate image is transformed while still attempting to maintain as much of the content representation as possible, leading to a stylistically transformed image. If $\vec{c}$ is the content image, $\vec{s}$ is the style image, and $\vec{n}$ is the candidate image, the loss function is

$$\mathcal{L}_{total}(\vec{n}, \vec{c}, \vec{s}) = \alpha \mathcal{L}_{content}(\vec{n}, \vec{c}) + \beta \mathcal{L}_{style}(\vec{n}, \vec{s}), \quad (1)$$

where α and β are configurable parameters specifying the relative emphasis placed on the content and the style differences of the candidate image, respectively. In our implementation, α and β are encoded as one configurable style/content ratio.

The derivative of this loss function provides us with an indication of the direction to modify the image so that the loss function is most minimized.

B. Gradient Descent

Using these two metrics, we use a first-order optimization technique known as gradient descent (specifically, L-BFGS) to minimize the differences between the filter responses of the content and style images and that of the image we are creating This is achieved by modifying the image by the negative direction of the filter gradient magnitude, which allows us to quickly approach a local minimum of the loss function.

Using the above procedure, we capture higher level representations of the style of the style image and the content of the content image, initialize a new white noise image, and perform gradient descent to minimize the previously discussed loss function. After an adequate number of iterations (empirically determined to be around 300), this procedure produces an image that the CNN hidden layers determine to have style similar to that of the style target image and content similar to that of the content target image.

C. Implementational Details

The original VGG-19 network was released to the public using the C++ deep learning framework Caffe [7], so we decided to use Caffe's native Python bindings to implement our algorithm, as both of us were familiar with Python. In addition to the Pycaffe bindings, additional libraries used to handle numerical computation were Scipy for optimization and matrix multiplication (`scipy.optimize.fmin_l_bfgs_b`, `scipy.linalg.blas.sgemm`) and Skimage for image processing. Jupyter notebook was used to display results from our rented server.

Because the deep learning methods used in this paper and similar tasks are very computationally intensive, we used a GPU-specialized Amazon AWS EC2 instance (g2.2xlarge) with Nvidia CUDA and CuDNN deep learning libraries to provide computational power for this project. Running a single forward pass of the network on an input image took around 45 seconds on an average Intel laptop; when computing on a server with Nvidia CUDA and CuDNN, the same forward pass takes less than one second. This enabled us to iterate rapidly,

debugging and testing new configurations of variables to create the optimal algorithm implementation.

D. Parameter Tuning

We made several parameters configurable, including settings for image width, number of iterations, stylistic content weighting, and resize parameters. These are detailed in the Appendix and discussed in the results.

E. Extending the algorithm: whole-artist replication

We attempted to extend the original algorithm by creating a program that replicated an image not just in the style of one artist's painting (e.g. "Starry Night"), but rather attempted to imitate how an artist would paint an arbitrary image. There were two obstacles we attempted to overcome.

a) Stochastic artist model: Attempts to train gradient descent on the arithmetic mean of multiple stylistic target images resulted in meaningless images (Figure 3). We hypothesize this is because interesting feature details in each image get averaged out. Additionally, the optimized image does not reflect a global average, but rather, parts of the image reach local optima as they become close to one specific stylistic image, as visible in the figure below.

Fig. 3: Result of optimizing according to the stylistic mean of 45 of Picasso's cubist works (Still Life with the Caned Chair, Guernica, etc.)

A workaround we devised was to create stylistic representations of an artist based on randomly selecting a single artwork from that corpus. To obtain the necessary quantity of artwork, we built an algorithm to scrape Wikiart for all of one artist's paintings. We then create an image that serves as an adequate representation of an artist's style for the particular content we are modifying by randomly selecting an image in the body of work by that artist, and running that new image through the network. This representation is then used in the gradient descent process previously described.

b) Color transfer: Color was tightly correlated with our stylistic features; regardless of the original color of the content image, training it on a heavily colored image such as Starry Night overwhelmingly skewed the final image towards the color of the style image (see Figure 6). In order to circumvent this and obtain a more natural stylistic transfer, regardless of color, we used a color transfer algorithm [8] to map the color from the content target image to the style target image prior to generating higher level style representations with the convolutional neural network. This resulted in stylistic representations which had colors similar to that of the content image.

III. Results

Results were good and reflected the high-quality images described in the original paper. We tried various combinations of images and paintings, and found good results applying various paintings to a publicly available image of the Eiffel Tower; those results are shown in Figure 5. What's more, by using the aforementioned computer architecture, we were able to achieve good runtimes; 200 iterations of gradient descent on a final 512 px width image generally took only around 10 minutes on our EC2 instance.

A. Parameter Tuning

c) Gradient descent iterations: We found that gradient descent on content images showed diminishing perceptual quality returns. The first 40 or so iterations were the biggest contributors to the overall quality of the image; after that, 100–150 iterations showed good strengthening of the texture of the image, removing more microscopic details and overall participating in the "smoothing" of the image. Examining differences in single iterations in the 150+ range show almost no perceptual difference, but it is possible to see further mild improvements when making jumps from 150 to 300+ iterations. The default convergence parameter in scipy's L-BFGS algorithm resulted in most images converging after 600 or 700 iterations; results at this level were not visibly different than results obtained at 300 iterations. See Figure 6 for a demonstration of the effect of the number of iterations on image quality.

d) Style/content ratio: The optimal style/content ratio depended on the specific image. We found that weights of 100 ($\alpha = 1, \beta = 100$) or 1000 ($\alpha = 1, \beta = 1000$) in favor of the stylistic loss function worked well for most images.

e) Style image resize: The style image resize parameter depends on the size of the image to be produced. One limitation of this algorithm is that it is not invariant to various resolutions of style and content images. Figure 4 shows the effect of scaling the stylistic image and producing an identically-sized content image; larger scaling captures more microscopic, texture-specific features, while smaller scaling captures more macroscopic features. In the case of Starry Night, more overall curvature in brushstrokes is seen with the 1.2x style scaling versus the 1.8x style scaling. As the image to be produced increases in resolution, the stylistic image should change also to keep the level of detail acquired similar.

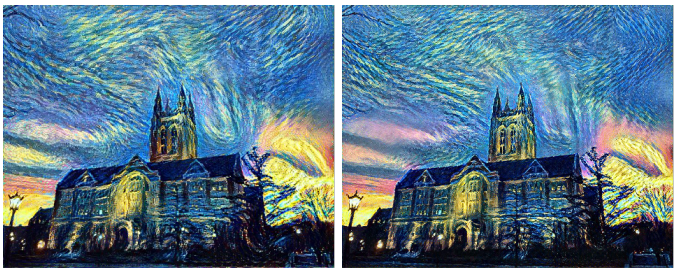


Fig. 4: Effect of scaling the stylistic image by 1.2x and 1.8x, respectively, on the stylistic content of the identically-sized output image.

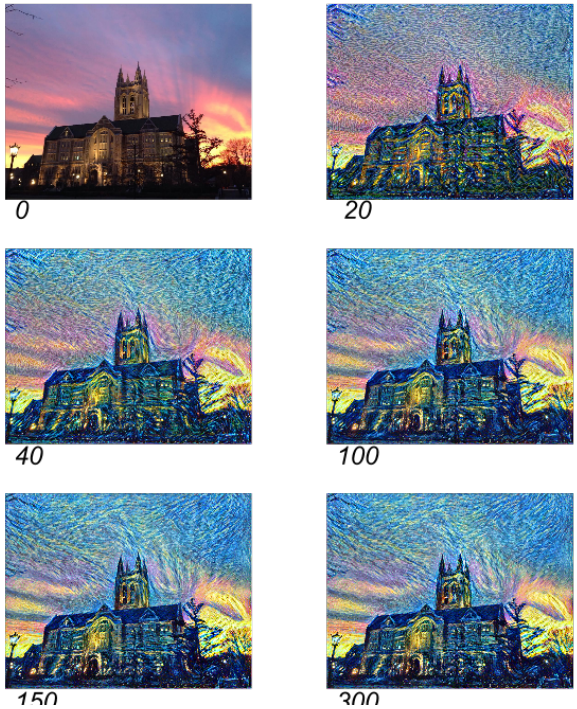


Fig. 6: Progress of numerous iterations of L-BFGS gradient descent on an image of Gasson Hall at Boston College in Chestnut Hill, MA. Candidate image was initialized as the original image.

B. Extensions

Our application of a color transfer algorithm worked well for some images but was rather weaker for others. The best results produced by the color transfer algorithm worked on stylistic images that had relatively mild coloring, so the color of the content image was able to dominate the color of the stylistic image. An example of using color transfer to keep the color of the content target image is illustrated in Figure 7.

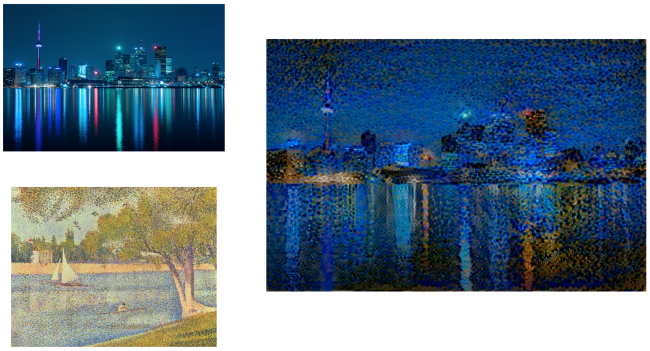


Fig. 7: Example of applying color transfer from a nighttime picture of the Toronto skyline to Seurat's *The Seine and la Grande Jette - Springtime*.

IV. Conclusion

The project provided us with a platform on which to investigate deep learning and its potential uses in computer vision. While we were able to successfully recreate the algorithm laid out in Gatys et al [1], our initial attempts at creating a stylistic representation left much to be desired. This led us to take look at the problem from a less technical perspective and realize that it may be adequate to simply randomly select the work form the artist and use that as a stylistic reference. Additionally, we encountered significant difficult separating texture and large scale features from color. We managed to find a partial solution to this problem by mapping the color space from the target content image to the target style image prior to building the higher-level representations with the network but this often produced substandard results and is certainly in need of improvement. A natural next step for this project is to address the color representation by attempting to build our target representations in HSV color space, which inherently separates texture from color (Figure 8).

However it is likely that this will challenge the con-

volutional neural network's ability to build reasonable higher-level representations of images because the network was trained to accept input images in the RGB color space. After having completed the initial phase of this project, we believe the most important step moving forward will be to gain a better understanding of the convolutional neural network we are using by running gradient descent on one layer at a time to see what it is capturing. Up until this point, we have been treating the network as a black box that is simply feeding it images and capturing the response of predetermined layers. In order to extend this work in any meaningful way it has become clear that understanding the network on a deeper level will be key.

Lastly, this project demonstrated the potential of deep learning in concrete terms that is widely accessible to those outside of computer science, especially in the liberal arts tradition of Boston College. This proved incredibly rewarding as it allowed us to encourage our friends, family, and general community to learn more about Artificial Intelligence and how it impacts their life.

REFERENCES

[1] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *CoRR*, vol. abs/1508.06576, 2015. [Online]. Available: http://arxiv.org/abs/1508.06576

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, insight. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[3] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, no. 11, pp. 1019–1025, 1999.

[4] M. Gazzaniga, R. Ivry, and G. Mangun, *Cognitive Neuroscience: The Biology of the Mind*. W.W. Norton, 2013.

[5] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen, "UFLDL Tutorial," 2010. [Online]. Available: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *CoRR*, vol. abs/1408.5093, 2014. [Online]. Available: http://arxiv.org/abs/1408.5093

[8] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Computer graphics and applications*, no. 5, pp. 34–41, 2001.

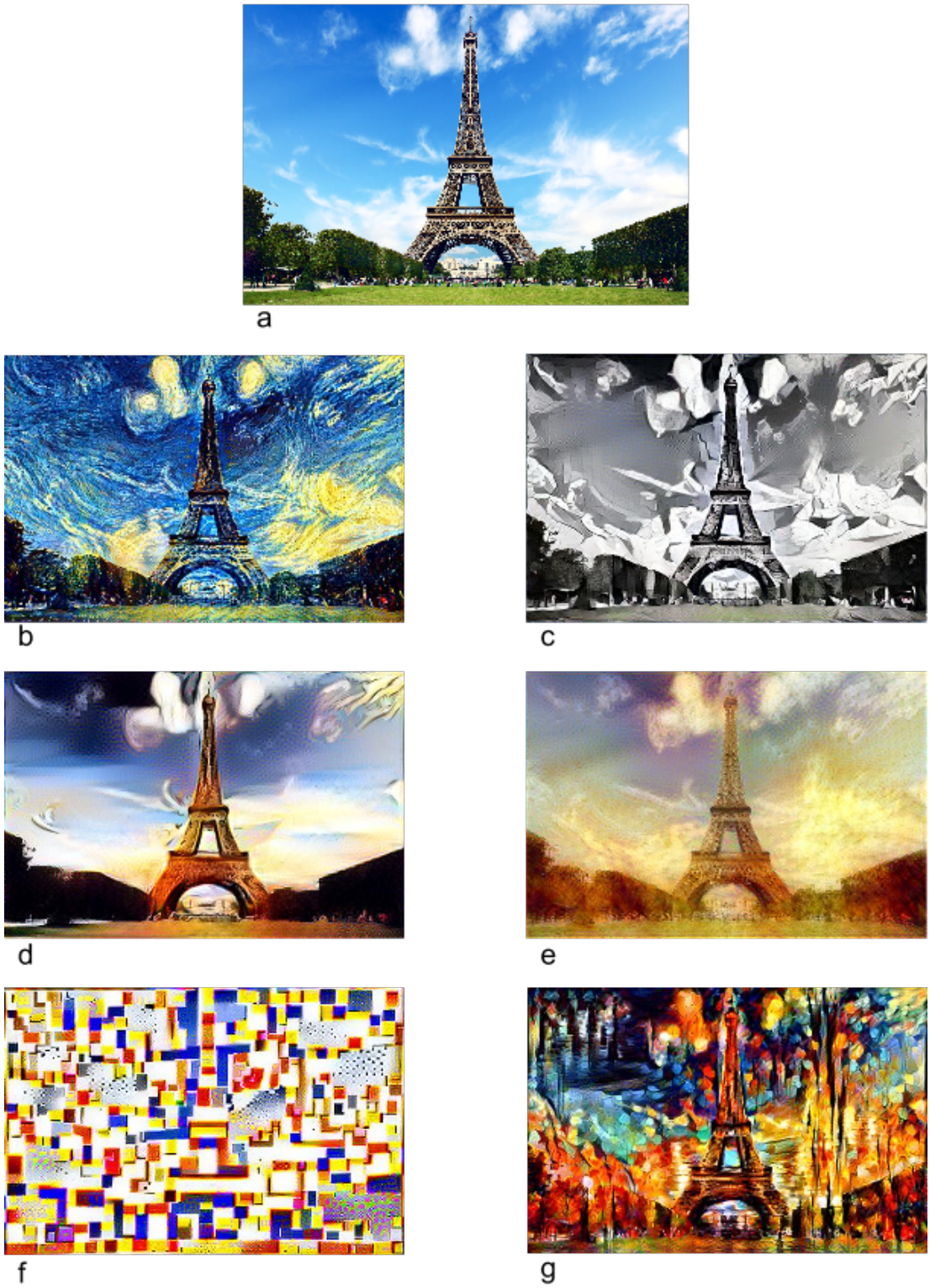


Fig. 5: Result of applying several paintings to an image of the eiffel tower. Original image shown at top (a). Then, applied paintings are (b) The Starry Night, Vincent Van Gogh; (c) Guernica, Pablo Picasso; (d) The Persistence of Memory, Salvador Dalì; (e) Rain, Steam, and Speed, JMW Turner; (f) Broadway Boogie-Woogie, Piet Mondrian; (g) Farewell to Anger, ieonid Afremov.

(a) Original (b) RGB Channels (c) HSV Channels

Fig. 8: Comparison of original image with information in the images' RGB and HSV spaces.

APPENDIX

OUTPUT OF ART.PY HELP

```
usage: art.py [-h] [-A ARTIST] [-n NUMITER] [-w WIDTH] [-i {rand,content}]
              [-p PRINT_RATE] [-s STYLE_SCALE] [-c--color-transfer]
              content_image style_images [style_images ...]

positional arguments:
  content_image         Content image
  style_images          Style image(s)

optional arguments:
  -h, --help            show this help message and exit
  -A ARTIST, --artist ARTIST
                        Artist to imitate, Script will randomly choose an
                        artwork from this artist. Artist's work must be saved
                        in wikiart/artist_name!
  -n NUMITER, --numiter NUMITER
                        Number of iterations
  -w WIDTH, --width WIDTH
                        Max image width
  -i {rand,content}, --init {rand,content}
                        Initialize image from noise (rand) or original image
  -p PRINT_RATE, --print_rate PRINT_RATE
                        How often to save intermediate images
  -s STYLE_SCALE, --style_scale STYLE_SCALE
                        Resize style image - changes resolution of features
  -c--color-transfer    Apply color transfer algorithm to attempt to change
                        style image to match color of the content image.
```