**Artistic Neural Style Transfer using Convolutional Neural Networks**

by Tarun Narayanan V (2018)

**ABSTRACT**

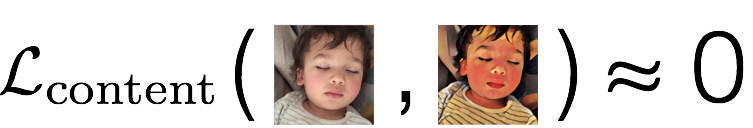
Gatys et al. (2015a) showed that pair-wise products of features in a convolutional network are a very effective representation of image textures. We propose a simple modification to that representation which makes it possible to incorporate long range structure into image generation, and to render images that satisfy various symmetry constraints. We show how this can greatly improve rendering of regular textures and of images that contain other kinds of symmetric structure. We also present applications to inpainting and season transfer.

**HYPOTHESIS**

There’s an amazing mobile app out right now called Prisma that transforms your photos into works of art using the styles of famous artwork and motifs. The app performs this style transfer with the help of a branch of machine learning called convolutional neural networks. Prisma is a mobile app that allows you to transfer the style of one image, onto the content of another, to arrive at gorgeous results like these:

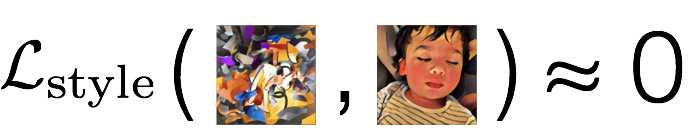


Like many people, I found much of the output of this app very pleasing, and I got curious as to how it achieves its visual effect. At the outset you can imagine that it’s somehow extracting low level features like the colour and texture from one image (that we’ll call the style image, ss) and applying it to more semantic, higher level features like a toddler’s face on another image (that we’ll call the content image, cc) to arrive at the style-transferred image.

Let’s suppose that we had a way of measuring how different in content two images are from one another. Which means we have a function that tends to 0 when its two input images (cc and xx) are very close to each other in terms of content, and grows as their content deviates. We call this function the content loss.

*A schematic of the content loss.*

Let’s also suppose that had another function that told us how close in style two images are to one another. Again, this function grows as its two input images (ss and xx) tend to deviate in style. We call this function the style loss.



*A schematic of the style loss.*

Suppose we had these two functions, then the style transfer problem is easy to state, right? All we need to do is to find an image xx that differs as little as possible in terms of content from the content image cc, while simultaneously differing as little as possible in terms of style from the style image ss. In other words, we’d like to simultaneously minimize both the style and content losses.

This is what is stated in slightly scary math notation below:

x∗=argminx(αLcontent(c,x)+βLstyle(s,x))

Here, α and β are simply numbers that allow us to control how much we want to emphasise the content relative to the style. We’ll see some effects of playing with these weighting factors later.

Now the crucial bit of insight in a paper by Gatys et al. is that the definitions of these content and style losses are based not on per-pixel differences between images, but instead in terms of higher level, more perceptual differences between them. Interesting, but then how does one go about writing a program that understands enough about the meaning of images to perceive such semantic differences?

Well, it turns out we don’t. At least not in the classic sense of a program with a fixed set of rules. We instead turn to machine learning, which is a great tool to solve general problems like these that seem intuitive to state, but where it’s hard to explicitly write down all the steps you need to follow to solve it.

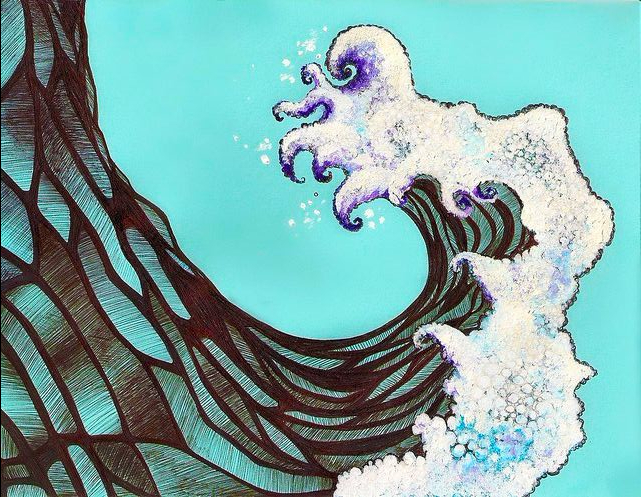
WORKING

Let’s cut to the chase so how did I finally implement the idea I comprehended into code? Specifically, <100 lines of code?  
<https://github.com/tarunn2799/Neural-Style-Transfer-in-100-lines>

The code looks pretty cluttered but it’s fairly simple.

**NOTE: Before copying this code, make sure that the directory you run this in has the vgg16\_weights.npz file. You can do this my simply opening CMD, navigating to the directory in which this code is, and running the following command:**

wget <http://www.cs.toronto.edu/~frossard/vgg16/vgg16_weights.npz>





*Content Style*



And the Output:

[Recall](https://harishnarayanan.org/writing/artistic-style-transfer/#so-what-is-prisma-and-how-might-it-work) that we had a content image cc and a style image ss, and the core of this effect was to somehow extract the style (the colour and textures) from ss and apply it to the content (the structure) from cc to arrive at the sort of example output shown above

We are looking for an image xx that differs as little as possible in terms of content from the content image cc, while simultaneously differing as little as possible in terms of style from the style image ss. The only hitch in this plan was that we didn’t have a convenient way to gauge the differences between images in terms of content and style.

But now we do.

It turns out that convnets pre-trained for image classification have already learnt to encode perceptual and semantic information that we need to measure these semantic difference terms! The primary explanation for this is that when learning object recognition, the network has to become invariant to all image variation that’s superfluous to object identity.

The algorithm has trained itself to transform the raw input pixels into category scores, it has learnt to encode quite a bit of perceptual and semantic information about images. The neural style algorithm introduced in [Gatys et al. (2015)](https://arxiv.org/abs/1508.06576) plays with these representations to first define the semantic loss terms (Lcontent(c,x)Lcontent(c,x) and Lstyle(s,x)Lstyle(s,x)) and then uses these terms to pose the optimisation problem for style transfer.

CODE EXPLANATION:

**Line 1-7:**

We import the content and style image, choose the content and style layers that will be used for loss calculations, set the content weight, style weight, total variational weight, learning rate, and the number of iterations.

**Line 9-16:**

We calculate the style loss here. It is a little complicated. We have a **gram matrix**which has the wonderful property of capturing global statistics across the image due to spatial averaging. This implies that the gram matrix is fully ‘blind’ to the global arrangement of objects inside the reference image. Read [this excellent paper](https://arxiv.org/pdf/1606.01286.pdf)( <https://arxiv.org/pdf/1606.01286.pdf> ) for a more comprehensive understanding of the concept.

**Line 22-62:**

We get the VGG model from the weights file specified and post and preprocessing of images is performed.

**Line 63-68:**

Calculate the content layer for the content image provided.

**Line 69-74:**

Calculate the style layers for the style image provided.

**Line 74-99:**

**Get the model**, find the content loss (**contl**), style loss (**stylel**), total variational loss (**tvl**), and find out the total loss by **summing** the three losses.  
**Line 89**sets the adam optimizer which minimizes the total loss (totL). **Line 92**starts the loop for optimizing the image. In **Line 97** We store the output with the lowest loss every 100 iterations. This can help us get the result of the minimum loss encountered in the selected number of iterations. We this process the image and save it as ‘output.

**NOTE:** **You can see clearly how g.device(‘/cpu:0’) has been set. This was done purposely as I ran the code on  Intel® Nervana™ DevCloud. If you have a CUDA capable GPU, be sure to change them to ‘/gpu:0’ to run on your device.**

**CONCLUSION**

Our road to the solution of the style transfer problem is a bit more straightforward in hindsight. We started at a seemingly innocuous place, the image classification problem. Before long, we realised that the semantic gap between the input representation (raw pixels) and the task at hand made it nearly impossible to write down a classifier as an explicit program. So we turned to supervised learning, which is a great tool for times like these when you have a problem that seems intuitive to state — and you have lots of examples as to what you intend — but where it’s hard to write down all the solution steps.

As we worked through more and more sophisticated supervised learning solutions to the image classification problem, we found that deep convolutional networks are really good at it. Not only that, we understood that they perform so well because they’re great at learning representations of the input data suitable to the task at hand. They do this automatically without the need for feature engineering, which would otherwise require a significant amount of domain expertise. It is the internal representations of this knowledge that we creatively repurposed to perform our style transfer.

These results were mostly generated with the following weights for the different loss terms: content weight c\_w = 0.025, style weight s\_w = 5, and total variation weight t\_v\_w = 0.1. This choice came from a lot of systematic experimentation across ranges for these parameters. One of these experiments is captured in the following animation that fixes the c\_w = 0.025, t\_v\_w = 0.1 and varies s\_w between 0.1 and 10 in a few steps.



.

REFERENCES:

1. [The Stanford course on Convolutional Neural Networks](http://cs231n.stanford.edu/) and [accompanying notes](http://cs231n.github.io/)
2. [Deep Learning Book](http://www.deeplearningbook.org/)
3. [Deep Visualization Toolbox](http://yosinski.com/deepvis)
4. [A Neural Algorithm of Artistic Style](https://arxiv.org/abs/1508.06576), the seminal article
5. [Very Deep Convolutional Networks for Large-Scale Image Recognition](https://arxiv.org/abs/1409.1556)
6. [Perceptual Real-Time Style Transfer](https://arxiv.org/abs/1603.08155) and [supplementary material](https://cs.stanford.edu/people/jcjohns/papers/fast-style/fast-style-supp.pdf)
7. TensorFlow Website: MNIST digit classification tutorial [for beginners](https://www.tensorflow.org/get_started/mnist/beginners) and [for experts](https://www.tensorflow.org/get_started/mnist/pros), [Deep CNNs](https://www.tensorflow.org/tutorials/deep_cnn), [playground](http://playground.tensorflow.org/)
8. [Keras as a simplified interface to TensorFlow](https://blog.keras.io/keras-as-a-simplified-interface-to-tensorflow-tutorial.html)
9. A Neural Algorithm of Artistic Style, [Leon A. Gatys](https://arxiv.org/find/cs/1/au:+Gatys_L/0/1/0/all/0/1), [Alexander S. Ecker](https://arxiv.org/find/cs/1/au:+Ecker_A/0/1/0/all/0/1), [Matthias Bethge](https://arxiv.org/find/cs/1/au:+Bethge_M/0/1/0/all/0/1).
10. INCORPORATING LONG-RANGE CONSISTENCY IN CNN-BASED TEXTURE GENERATION, Guillaume Berger & Roland Memisevic, Department of Computer Science, University of Montreal

Special thanks to Guillem Cucurull for his github [repository](https://github.com/gcucurull/neural-art-transfer) on Neural art transfer.