
Neural Style Transfer

Yashila Bordag

Division of Computing, Data Science, and Society
Department of Classics
University of California, Berkeley
Berkeley, CA 94720
yashila.bordag@berkeley.edu

William Brandon

Department of Computer Science
Department of Mathematics
University of California, Berkeley
Berkeley, CA 94720
williambrandon@berkeley.edu

Wangda Lei

Department of Computer Science
University of California, Berkeley
Berkeley, CA 94720
radar1wd@berkeley.edu

This Note covers the use of Neural Style Transfer, how it works, and a few interesting examples using it. We assume the student has already seen Gradient Descent, Back Propagation, Convolutional Neural Networks, and how to use Pre-trained Models. We will also use TensorFlow, and briefly cover Hyperparameter Tuning, which would be useful for the student to know, but these topics are not necessary to master.

1 Algorithmic Understanding of Images: Motivation

One of the key areas of research in art history, archaeology, and other humanistic fields is qualifying art content and style. These fields pursue both intuitive and qualitative analysis of artwork of many forms including painting, photography, and cinematography. Traditional approaches in this vein require serious consideration of each piece, describing the interlocking relationship between the content, style, and context of an image - leveraging historical trends, comparanda, and presuppositions about the artist. However, these approaches are also time and resource intensive, heavily dependent on previous work in the field, and require relatively narrowly trained analysts, and cannot easily be applied en masse to large or diverse datasets of images due to these constraints.

This presents a unique issue of scale: though case studies provide interesting insights into narrow selections of work, broader analyses of large sets of paintings could take years or longer to complete. A potential solution to this problem is to develop an algorithmic approach in parsing the content and style of an image. Done effectively, this could allow for an algorithmic understanding of the elements of style and content in an image, as well as allow us to construct a distance of the style and content features to other images. If these features can be appropriately interpreted, one could analyze large sets of images for the style and content similarities between them and to make generalizations about subsets of the set or the set as a whole.

2 Feature Maps

In particular, we can draw a parallel between the features of an image and vectors in a vector space. Previously in EE16A, you should have been introduced to the concept of **vector spaces**, which are sets of vectors which are closed under addition and scalar multiplication. We can extend this definition to also include something called an **inner product**, which for our purposes allows us to define a distance between two vectors. In the context of feature spaces, we can then see the difference between two features in a **feature space**, which is a space encapsulating all the possible features an

image can have, and try to compare these features together mathematically. In our case, we will treat the **content space** and **style space** of an image as two different feature spaces.

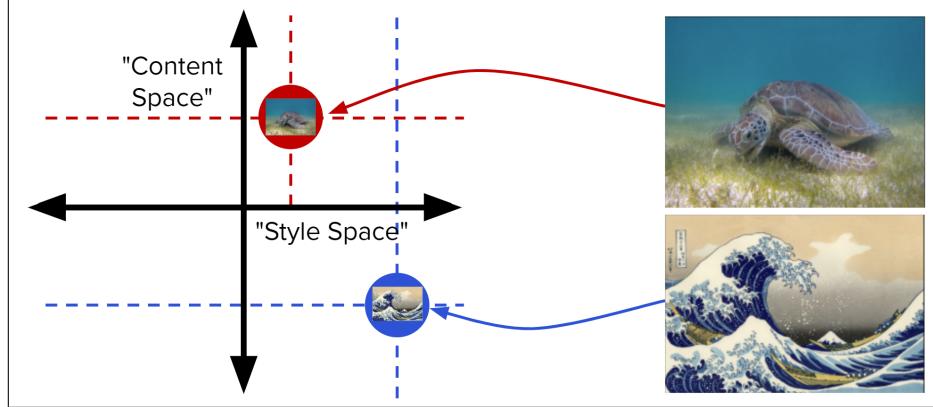


Figure 1: Here we can visualize our style and content spaces as the x- and y-axis respectively. Use this representation, we can see what the differences between the two images are.

In **Neural Style Transfer**, we will actually use a **pre-trained model** to take an image and extract the features in the feature space we want. This process of using a **pre-trained model** to find features is called using a **pre-trained feature map** and when good feature maps are available to break up the data we have into more useable chunks, it makes it much easier to focus on actually analyzing the data as opposed to spending time making the maps ourselves. Here we will try to minimize the distance between the image we generate and the content image in the content space, while also trying to minimize the distance between the generated image and the style image in the style space.

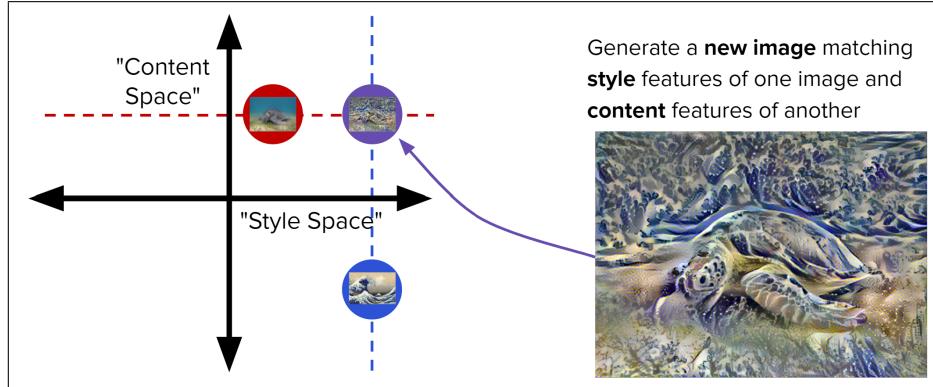


Figure 2: This is a broad-level visualization for how we are conceptually trying to generate the new image.

3 Neural Style Transfer

One recent tool developed to analyze images more efficiently is called the **VGG-Network**, a **Convolutional Neural Network** architecture developed by Karen Simonyan and Andrew Zisserman from Google DeepMind and the University of Oxford¹ which rivals human performance on visual object recognition tasks and was created for the 2014 ImageNet Challenge. We will explore a use of this network architecture in a technique called the **Neural Style Transfer Algorithm** in which we transfer the style of one image onto the content of another by extracting the style and content of the two initial images and minimizing the differences between them appropriately to create a blended image.

¹See Simonyan and Zisserman “Very Deep Convolutional Networks for Large-Scale Image Recognition.” (2015)

This is done by using the **pre-trained** feature maps of the VGG-network, with which we can generate the style and content features needed to generate a blended image. Detailed in Gatys et al. 2015², a **feature space** could be constructed to describe the style or content of an image. Using these features, we can calculate the difference between the derived style and content features of two images, and then minimize this loss to create a hybrid image, which transfers the style from one image onto the content of another. This is done by using an appropriate **loss function**, which can be interpreted as the distance, which we use to calculate the difference between the style of two images or the content of two images in the corresponding feature spaces. Then the algorithm tries to minimize this loss as much as possible so that the content features of the content image and the content features of the generated image is as small as possible while the style features of the style image and the style features of the generated image are as small as possible.

3.1 CNNs

We are able to find our features using a type of **pre-trained Deep Neural Network** called a **Convolutional Neural Network (CNN)**. A **Neural Network** is a class of algorithms which simulate systems of neurons with sets of weights and activation functions which correspond to the way biological neural networks propagate and process stimuli. The power of neural networks is found in the flexibility of the types of inputs and outputs a neural network can be trained to process and also the vast variety of functions it can mimic.

At a basic level, we are leveraging the activations we get between the layers of our pre-trained CNN to define our features. These activations are essentially the outputs of the network at certain intervals which informs us about how the network is parsing information out of the data. Very generally, CNNs abstract higher and higher level data from the input data as the CNN processes this data at later and later layers. In the VGG-19 model, as well as many other computer vision geared CNNs, lower level layers are associated with shapes and content which higher level layers focus on style and texture. The level at which we define low and high for content and style is variable depending on network and definition, but the overall structure tends to be the same across different architectures. In this note, we will walk you through how activations can influence the features produced by our model, but in the homework, the gradients with respect to the image will be generated for you.

Neural networks are trained using a process called **gradient descent** where we use an iterative approach to attempt to minimize a loss function (make the distance between the output and the desired output as small as possible). The algorithm essentially tries to minimize the loss by taking its gradient with respect to the weight which we want to train (in this case a connection between two ‘neurons’) and descending in the opposite direction. The update step for time t is usually denoted as follows:

$$w^{t+1} = w^t + \eta_t \frac{\partial Loss}{\partial w^t} \quad (1)$$

This can be extended to train entire matrices, which we represent the connections between neural layers as, where the update takes the following form:

$$w^{t+1} = w^t + \eta_t \nabla Loss, \quad (2)$$

Where $\nabla Loss$ is the **gradient** of the loss function with respect to the weights which we want to train. This is the *gradient* referred to in *gradient* descent.

This is enabled using a technique called **backpropagation** which allows us to efficiently calculate the loss gradients at each time step, without which training neural networks would not be feasible.

Deep Neural Networks describes a class of neural networks which have many layers; in our case, we will be using a relatively shallow DNN that is built into Keras called VGG19, which has 19 layers. The VGG-Network is more specifically a **Convolutional Neural Network (CNN)** which are regularized versions of fully connected networks, achieve this regularization by making complex connections between layers, and focus on parsing increasingly complex meaning over layers. Using this, we can essentially build up an understanding of an image, starting at the pixel level, then at

²See Gatys et al. “A neural algorithm of artistic style.” (2015)

the style level where we consider color palettes, contrasts, and light play, and finally content, where we understand the subject of an image. We leverage a model which has been **pre-trained** meaning that the model has pre-instantiated weights which we can use right off the bat, because it makes the amount of computation we do minimal.

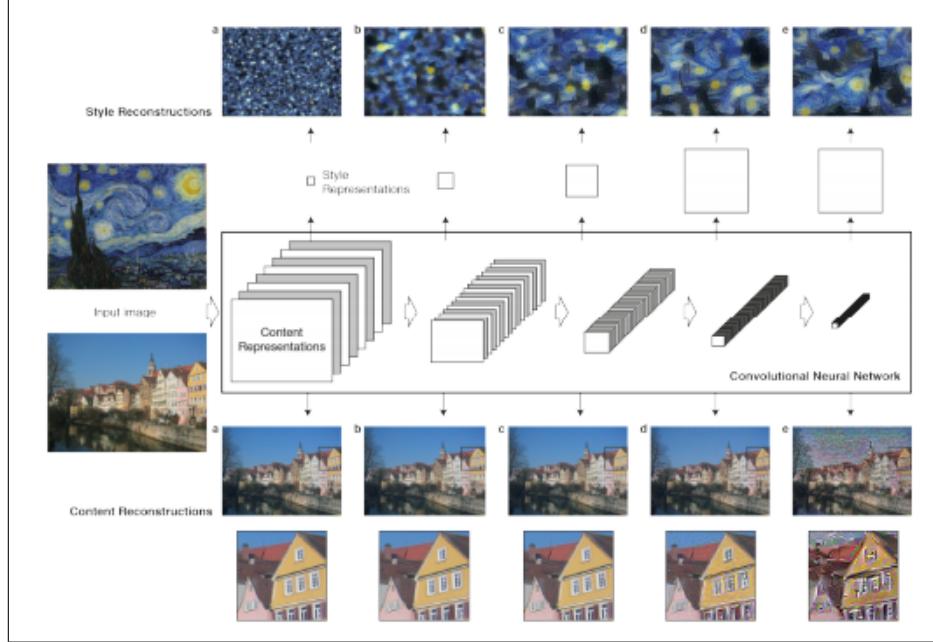


Figure 3: (from Gatys et al.) A representation of how the CNN treats the images over all the layers

3.2 Loss Function

To calculate the distance between two features, we need to define a **loss function** which calculates the perceived cost of a particular input given a certain objective. In our case, we want to reduce the difference between the style of our style image and the style of our generated image as much as possible while trying to reduce the difference between the content of our content image and the content of our generated image as much as possible. We can intuitively perceive the loss as a form of distance or cost, where the closer we get to the optimal outcome (in this case exact style or exact content) the smaller our loss should be.

For the content features, say we have the content image, \vec{p} , and the generated image, \vec{x} , which yield the content features P^l and F^l respectively. Then, we can use a simple squared-error loss to express the loss of a particular layer of \vec{x} :

$$L_{content}^l(\vec{p}, \vec{x}) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2 \quad (3)$$

And the total loss to be:

$$L_{content}(\vec{p}, \vec{x}) = \sum_{l \in L} L_{content}^l(\vec{p}, \vec{x}) \quad (4)$$

Then derivative of this content loss with respect to a particular activation is:

$$\frac{\partial L_{content}^l}{\partial F_{ij}^l} = F_{ij}^l - P_{ij}^l, \quad (5)$$

Next, we define the style loss in our algorithm, which is more involved than the content loss. Say we have the style source image, \vec{a} , and the candidate generated image, \vec{x} . We want to define the style

similarity between \vec{a} and \vec{x} as measured by a single layer l of the pre-trained model (in practice we measure the style distance using multiple layers; to do this, we simply compute the style distance using each layer individually and take the sum of the resulting distance scores).

Unlike the content loss, we don't want the style loss to depend too much on where in the image different features appear — instead, we just want the generated image to have the same *distribution* of style features as the style source image. To achieve this, we compute the style distance at layer l between two images using a two-step strategy: first, we generate a kind of statistical "summary" of the layer- l style features in each image; then, we measure the distance between the summaries.

Concretely, our "summaries" take the form of *Gram matrices*. Recall that the layer- l activations of the pre-trained model on a given input image consists of a collection of multiple feature maps, each of which is a rectangular array of real numbers which can be thought of as a grayscale "image" with pixels. The Gram matrix is then a matrix whose entries are the pairwise inner products between these rectangular arrays, viewed as one-dimensional vectors.

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l, \quad (6)$$

Then given the Gram matrix of \vec{x} at layer l , G^l , and the Gram matrix of \vec{a} at layer l , A^l , we can calculate the expected contribution of that layer to the style loss as:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \quad (7)$$

Where N_l is the number of distinct filters of the layer and M_l is size of the feature maps.

Then, given some w_l , which in our case will be $w_l = \frac{1}{|L|}$, we can calculate the style loss as follows:

$$L_{style}(\vec{a}, \vec{x}) = \sum_{l \in L} w_l E_l \quad (8)$$

Then the derivative of E_l , from which we can find the derivative of the style loss pretty easily, with respect to an activation is:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \frac{1}{N_l^2 M_l^2} (F^l)^T (G^l - A^l), \quad (9)$$

So our total loss can be calculated as:

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x}) \quad (10)$$

As we can see, our preference for content or style is determined by how we choose α and β . The calculation of the gradient step of the total loss function with respect to the activations is left up to the reader.

3.3 Neural Style Transfer Algorithm

Now that we have our loss functions and understand how gradient descent works, we can now explore how the Neural Style Transfer Algorithm minimizes the difference between the generated image and the two starting images. We will minimize our loss using the gradient descent algorithm and the computed gradients from the last two sections.

Essentially, we will define a minimization loop which will run for a predetermined number of repetitions, and will have a . Before the start of the loop, we will use the VGG-Network to find the features of the style and content images, and instantiate the variables necessary to run the optimization loop. At each timestep, we will recalculate the gradient and the total loss of the current generated image and if the loss is less than the best recorded loss, we will store it as the best candidate image so far. Then we can update the generated image with the update step defined below:

$$x^{t+1} = x^t - \eta_t \nabla L_{total}(\vec{p}, \vec{a}, \vec{x}^t), \quad (11)$$

Where $\nabla L_{total}(\vec{p}, \vec{a}, \vec{x})$ is calculated using the activation gradient. Thus the full Neural Style Transfer Algorithm is as follows:

Algorithm 1: Neural Style Transfer Algorithm

Data: Content Image \vec{p} and Style Image \vec{a}
Result: Returns Generated Image with Best Loss
 Find features of \vec{p}, \vec{a} ;
 $\vec{x}^0 = \vec{p}$;
for $t = 1$ to *num_iterations* **do**
 Find $\nabla L_{total}(\vec{p}, \vec{a}, \vec{x}^t)$;
 Find $L_{total}(\vec{p}, \vec{a}, \vec{x}^t)$;
 $\vec{x}^t = \vec{x}^{t-1} - \eta L_{total}(\vec{p}, \vec{a}, \vec{x}^{t-1})$;
if $L_{total}(\vec{p}, \vec{a}, \vec{x}^t) < L_{total}(\vec{p}, \vec{a}, \vec{x}^*)$ **then**
 | $\vec{x}^* = \vec{x}^t$;
 | $L_{total}(\vec{p}, \vec{a}, \vec{x}^*) = L_{total}(\vec{p}, \vec{a}, \vec{x}^t)$;
end
end
 Return \vec{x}^*

We can use other types of gradient steps as well, but that falls outside the scope of this course - other courses like EECS 127 and CS 189 cover this topic much more thoroughly.

Using this we can look at lots of different types of images and combine all sorts of different styles and content. Cool applications of the Neural Style Transfer algorithm include applying styles to videos instead of images and creating more complex feature maps for style, content, texture, lighting, and so forth to combine many different images and artists' work. Given the varying nature of the performance of different models on learning different aspects of the image, mixing and matching pre-trained feature maps might also increase the quality of trained images. We can also extend the loss functions to decrease the local differences in pixels, effectively smoothing the final image as well. More broadly, applying the technique to audio data and other types of data has interesting applications as well.

4 Applications of the Neural Style Transfer Algorithm

Now let's examine some examples!

4.1 Ball Shadow Examples

A common starting point for beginner artists is drawing a sphere and its shadow to learn about lighting and shading. Here we look at the content image in Figure 2, and observe how it is transformed by a series of style images.

4.1.1 Starry Night, Vincent Van Gogh

Here we look at the style transfer of Starry Night onto the ball shadow (Fig. 3, 4).

4.1.2 Pillars of Creation, Eagle Nebula

Next we look at the style transfer of the picture of the Pillars of Creation onto the ball shadow (Fig. 5, 6).

4.2 Oski

Now let's consider Oski (Figure 7)

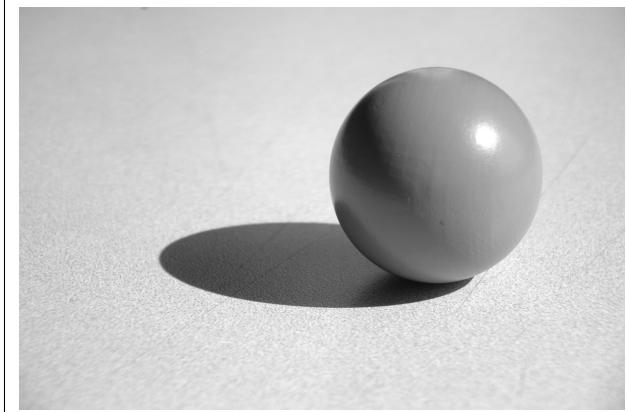


Figure 4: (from BYU—Idaho Art 110) A reference ball with shadow



Figure 5: (from Wikimedia Commons) Starry Night by Vincent Van Gogh

4.2.1 Old Man in Red, Rembrandt

Now we will use apply the style of a Rembrandt painting, the Portrait of an Old Man in Red (c. 1652-1654) onto Oski (Fig. 8, 9).

As we can see, the lighting and style that would expect do not appear. This is an interesting case of the limitations of our program - where when the algorithm cannot propose better images, it simply blurs and smooths the content as opposed to adding the dramatic lighting in the previous image.

4.2.2 Pillars of Creation

Finally, we will consider the style transfer of an image of the Pillars of Creation onto Oski (Fig. 10, 11).

As we can see, the transfer is still poor. This maybe because Rembrandt's style in the style image is too subtle to pick up on, or it may because the content features do not adequately separate our subject, Oski, from the background. In a more advanced style transfer, we could attempt to mix the style feature maps of one pre-trained model with the content feature maps of another pre-trained model, and optimize our choice of models based on the specific types of feature they are attempting to map.

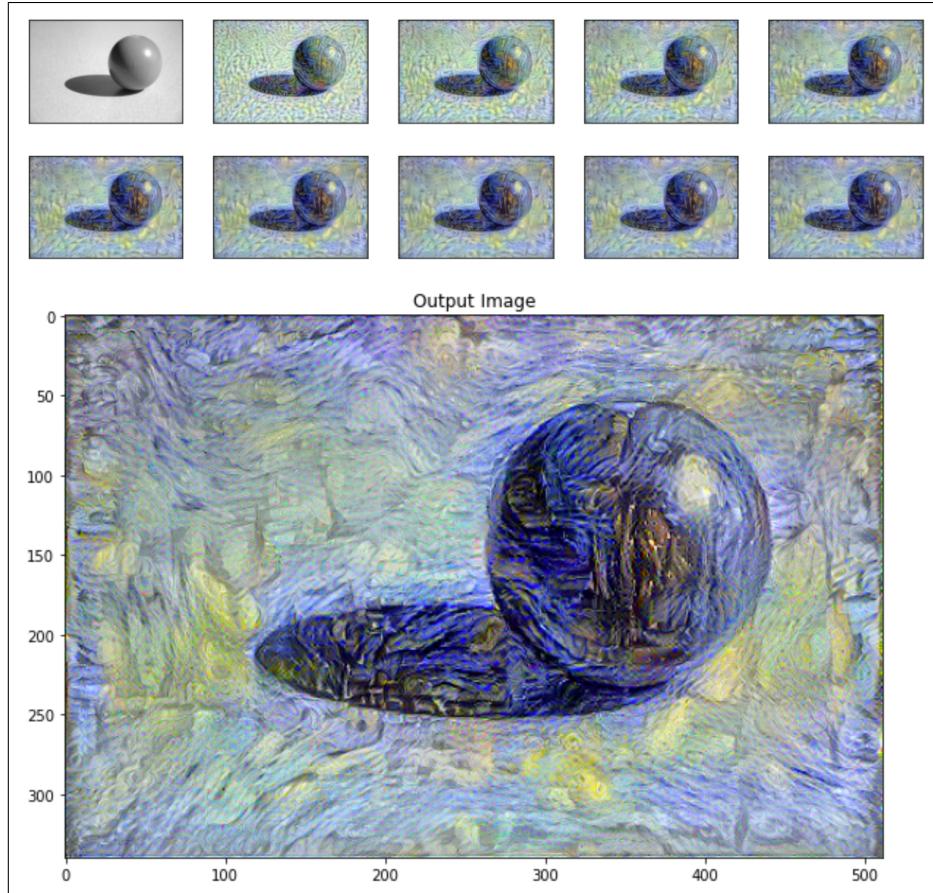


Figure 6: Generated image with Starry Night style transfer

References

- [1] Gatys, L. A., Ecker, A. S., Bethge, M. et al. "A neural algorithm of artistic style." arXiv preprint, 2015, <https://arxiv.org/abs/1508.06576>.
- [2] Simonyan, K., Zisserman, A. "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv preprint, 2015, <https://arxiv.org/pdf/1409.1556.pdf>.
- [3] Yuan, R. "Neural Style Transfer: Creating Art with Deep Learning using tf.keras and eager execution." Medium, 3, Aug. 2018, <https://medium.com/tensorflow/neural-style-transfer-creating-art-with-deep-learning-using-tf-keras-and-eager-execution-7d541ac31398>.
- [4] "Neural style transfer." TensorFlow.org, 2020, https://www.tensorflow.org/tutorials/generative/style_transfer.
- [5] Image of Ball and Shadow. Brigham Young University—Idaho, Department of Art, Art 110, 2010. https://courses.byui.edu/art110_new/art110/week06/images/sphere.jpg
- [6] All other images from Wikimedia Commons.

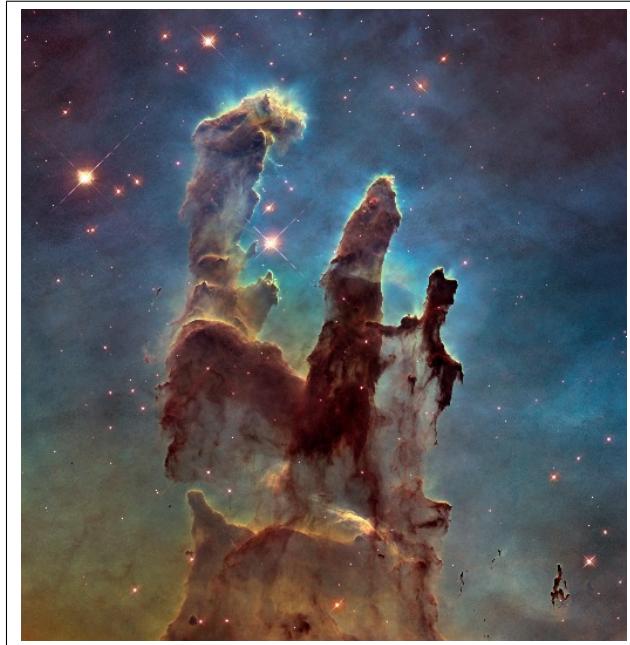


Figure 7: (from Wikimedia Commons) A picture of the Pillars of Creation in the Eagle Nebula (also known as Messier 16)

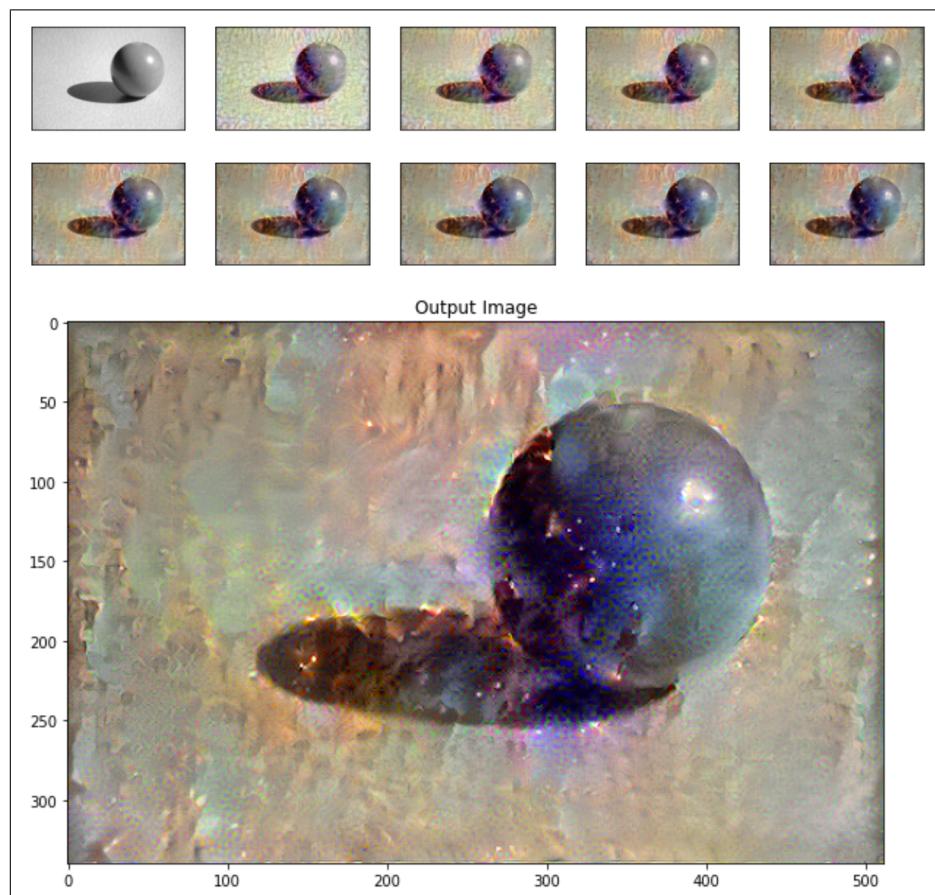


Figure 8: Generated image with Pillars of Creation style transfer



Figure 9: (from Wikimedia Commons) Oski

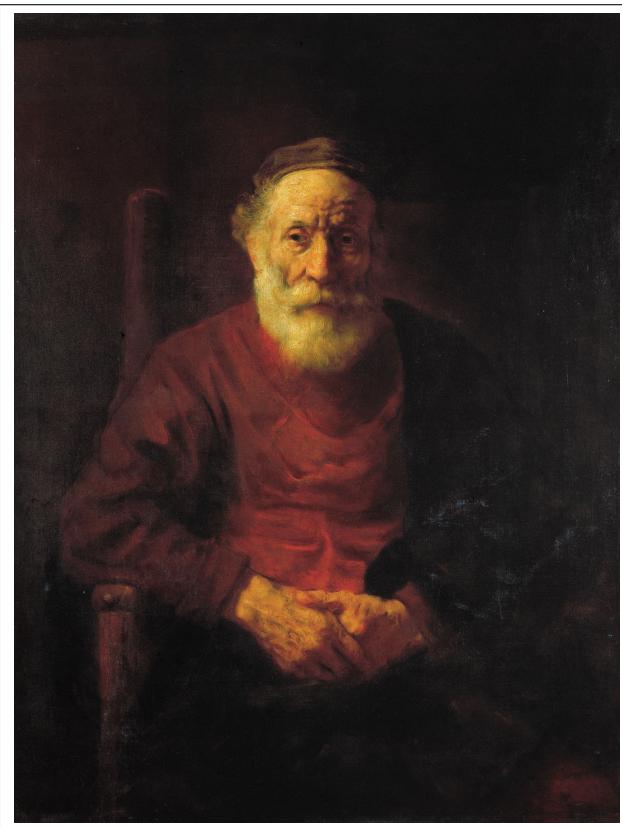


Figure 10: (from Wikimedia Commons) the Portrait of an Old Man in Red (c. 1652-1654), Rembrandt

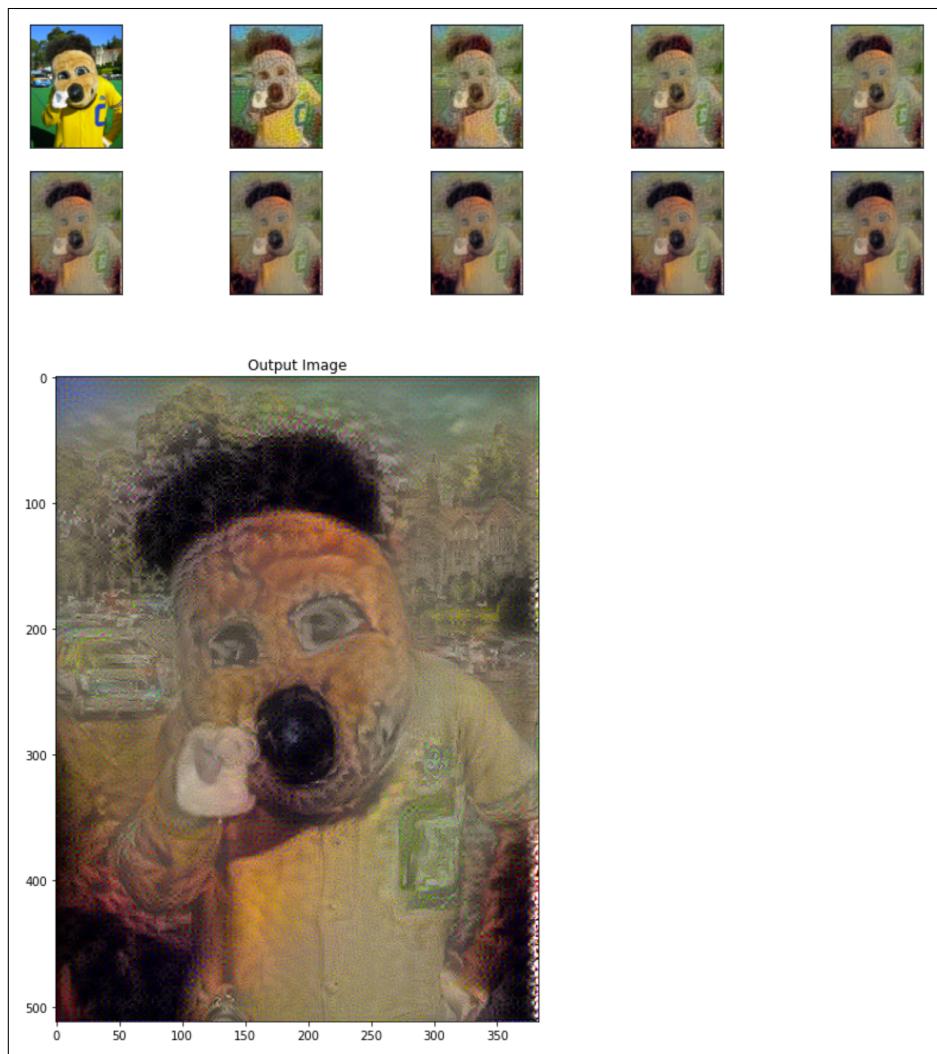


Figure 11: Generated image with the Rembrandt style transfer

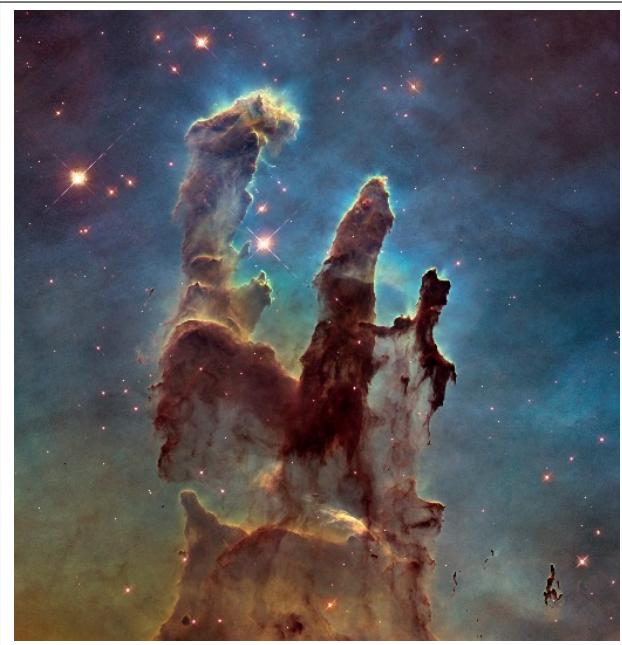


Figure 12: (from Wikimedia Commons) A picture of the Pillars of Creation in the Eagle Nebula (also known as Messier 16)

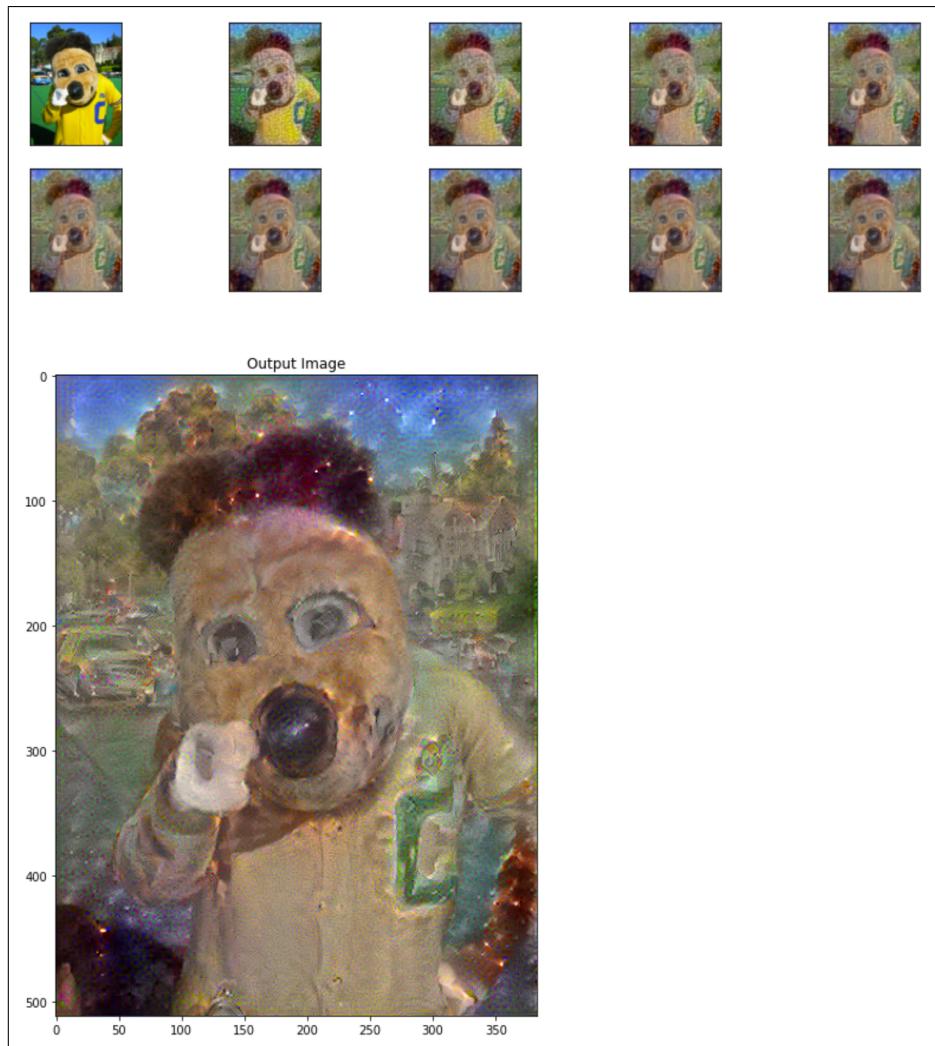


Figure 13: Generated image with the Pillars of Creation style transfer