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
radarlwd@berkeley.edu

This Note covers the use of Neural Style Transfer, how it works, and a few interesting examples using it.

1 How do we understand images?

One of the key areas of research in art history, archaeology, and other 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 required 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 Neural Style Transfer

One recent tool developed to this end is called the **VGG-Network**, a **Convolutional Neural Network** architecture developed by Karen Simonyan and Andrew Zisserman from Google DeepMind and the University of Oxford[1] 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.

[1] See 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. 2015[2], a **feature space** - a finite-dimensional vector space, in which we can define a distance between feature vectors - 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.

2.1 CNNs

We are able to find our features using a type of **pre-trained Deep Neural Network** called a **Convolutional Neural Network**. 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.

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:

$$w^{t+1} = w^t + \eta_t \frac{\partial Loss}{\partial w^t} \tag{1}$$

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

$$w^{t+1} = w^t + \eta_t \nabla Loss, \tag{2}$$

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 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.

2.2 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

[2]See Gatys et al. "A neural algorithm of artistic style." (2015)

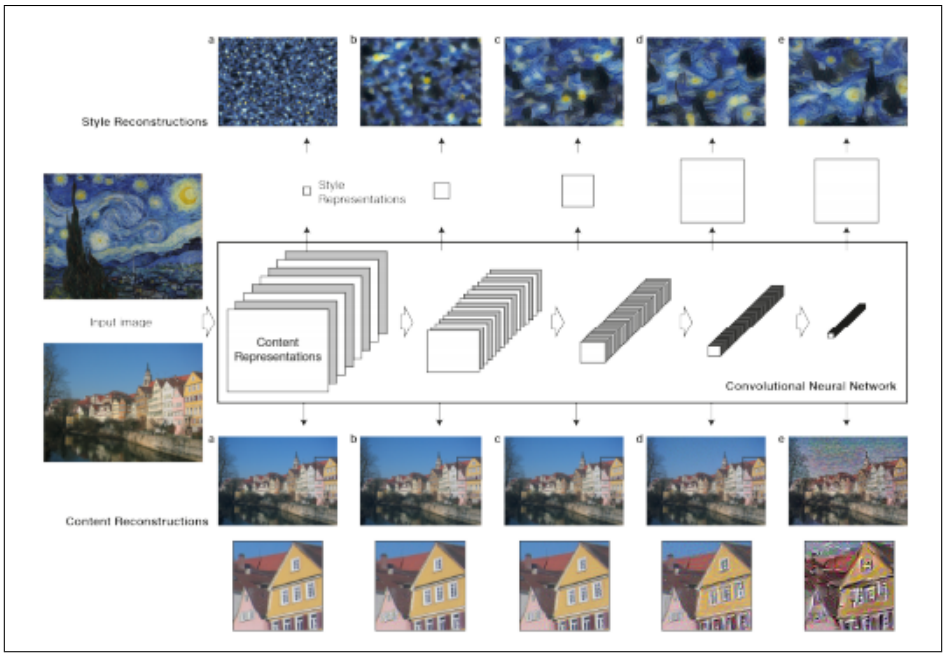

Figure 1: (from Gatys et al.) A representation of how the CNN treats the images over all the layers

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 $\vec{x}$:

$$L^l_{content}(\vec{p}, \vec{x}) = \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2 \tag{3}$$

And the total loss to be:

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

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

$$\frac{\partial L^l_{content}}{\partial F^l_{ij}} = F^l_{ij} - P^l_{ij}, \tag{5}$$

Next, we define the style loss in our algorithm, which is more involved than the content loss. Say we have the style image, $\vec{a}$, and the generated image, $\vec{x}$, then instead of directly comparing the CNN responses at every level, we try to find the feature correlations between different filter responses. The filter responses at each layer have no natural ordering, which means that the individual pixel locations do not matter - only the relative locations of some pixels to establish the way strokes are made and what the contrasts look like are trained by the model. We find the correlations by finding the Gram matrix G^l, where G^l_{ij} is the inner product between the vectorized feature map i and j in layer l, and is defined as follows:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}, \tag{6}$$

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:

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

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:

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

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

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

So our total loss can be calculated as:

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

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.

2.3 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:

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

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 $L_{total}(\vec{p}, \vec{a}, \vec{x^t})$;
  $\vec{x^t} = x^{\vec{t-1}} - \eta L_{total}(\vec{p}, \vec{a}, x^{\vec{t-1}})$;
  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 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.

3 Applications of the Neural Style Transfer Algorithm

Now let's examine some examples!

3.1 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.

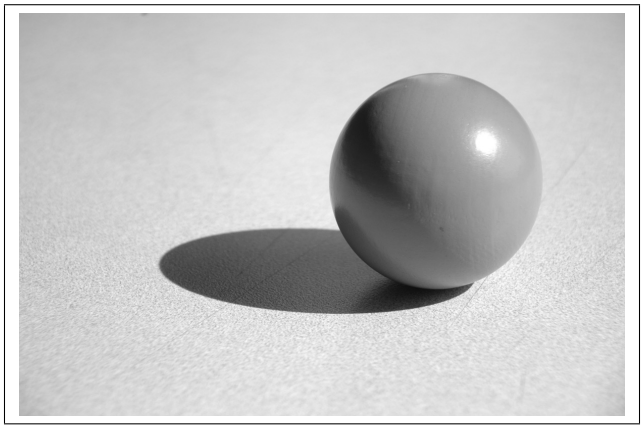

Figure 2: (from BYU—Idaho Art 110) A reference ball with shadow

3.1.1 Starry Night, Vincent Van Gogh

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

Figure 3: (from Wikimedia Commons) Starry Night by Vincent Van Gogh

3.1.2 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).

3.2 Oski

Now let's consider Oski (Figure 7)

3.2.1 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.

3.2.2 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.

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.

[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.

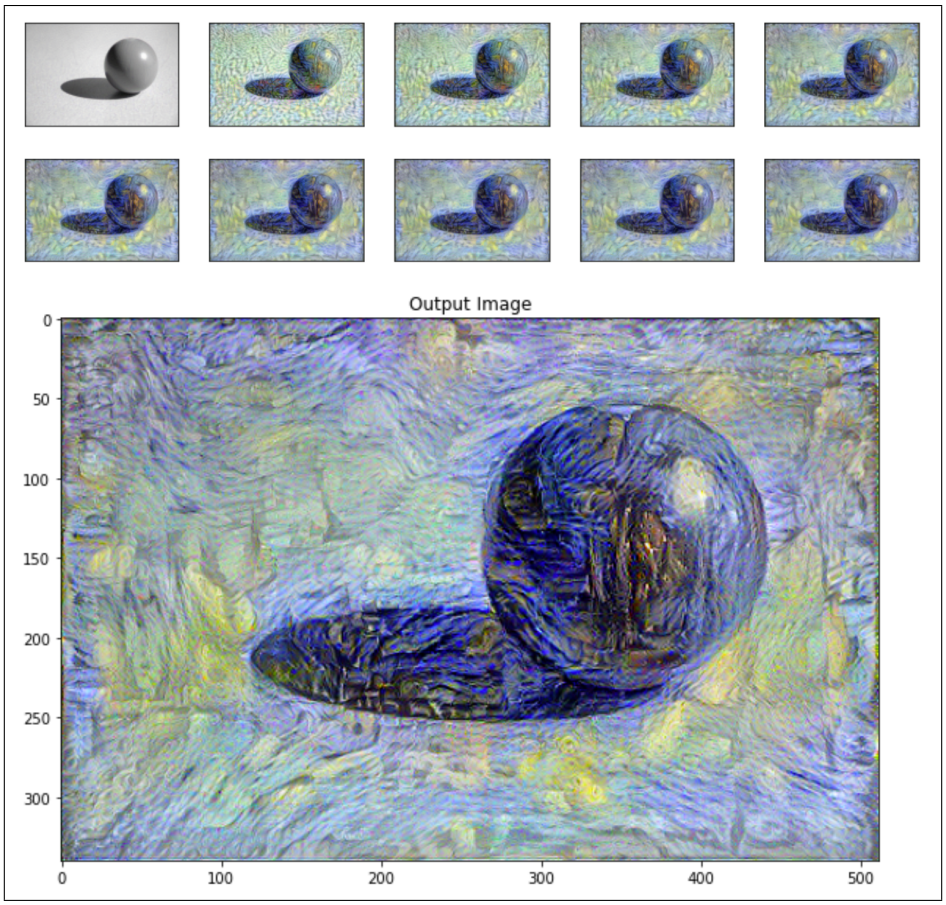

Figure 4: Generated image with Starry Night style transfer

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

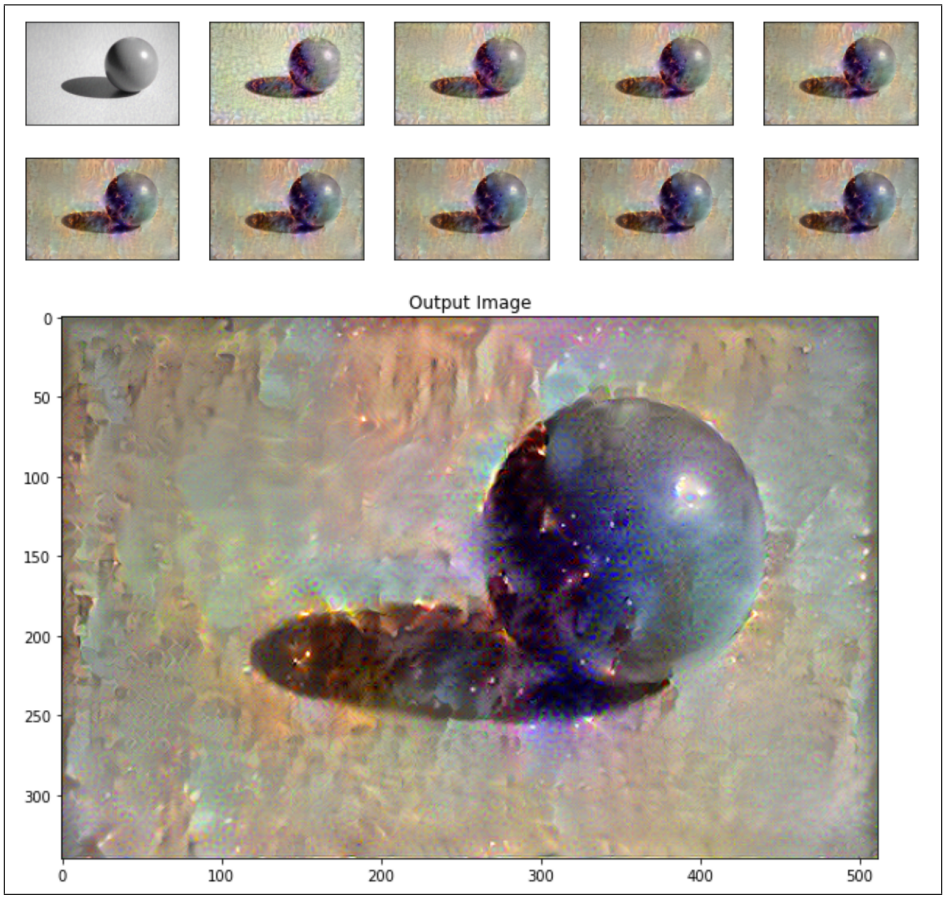

Figure 6: Generated image with Pillars of Creation style transfer

Figure 7: (from Wikimedia Commons) Oski

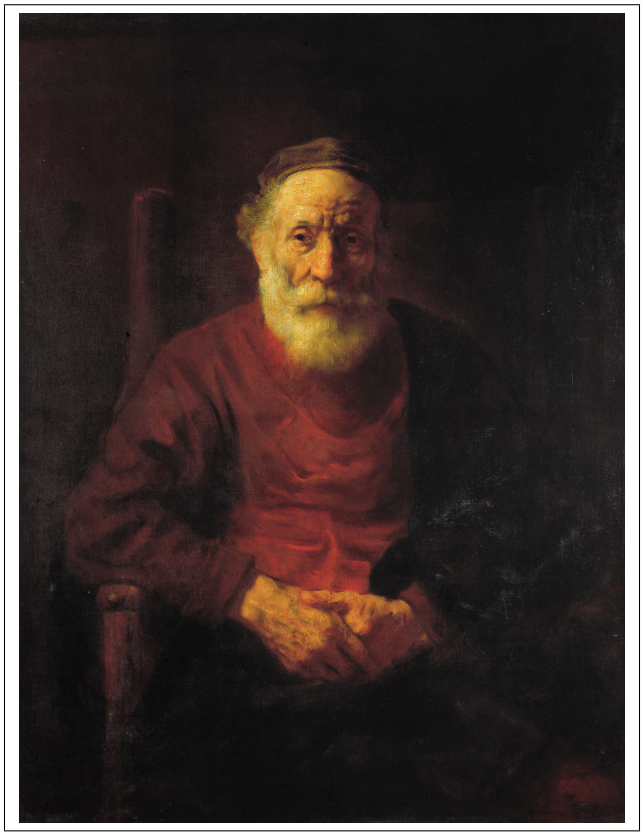

Figure 8: (from Wikimedia Commons) the Portrait of an Old Man in Red (c. 1652-1654), Rembrandt

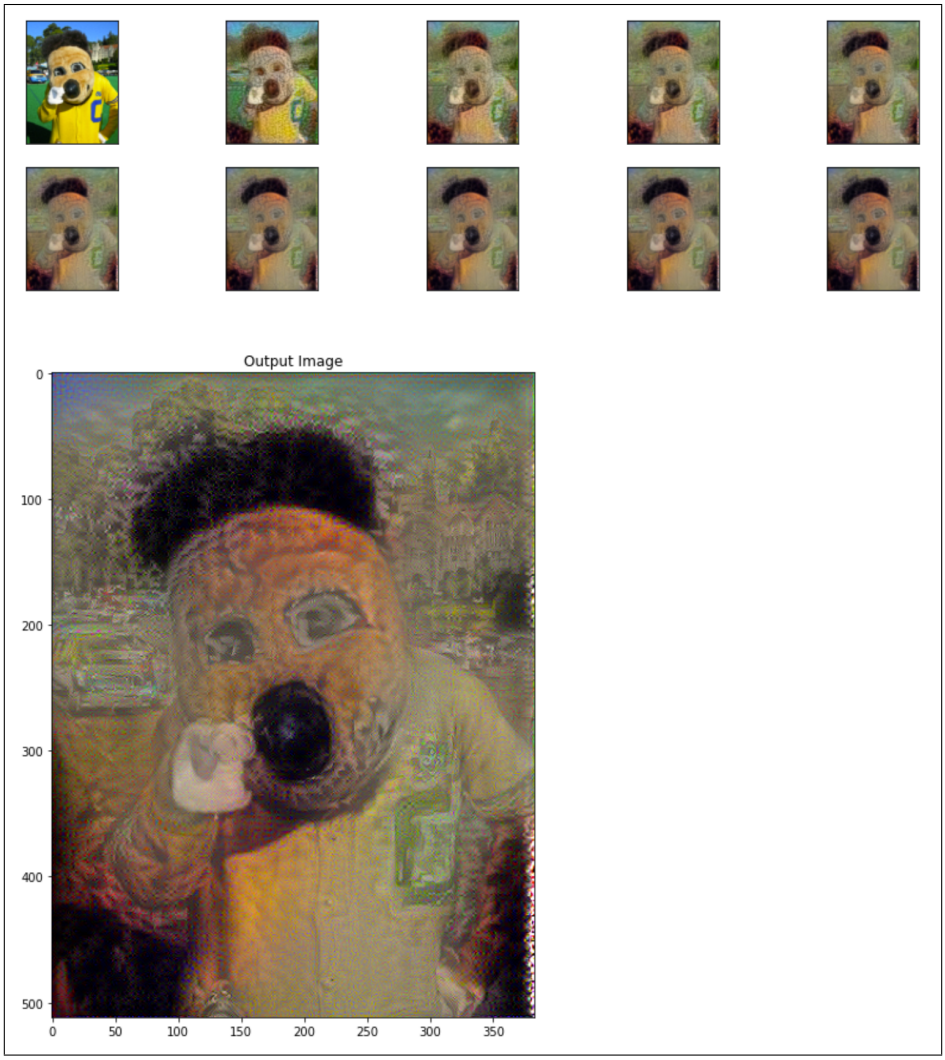

Figure 9: Generated image with the Rembrandt style transfer

Figure 10: (from Wikimedia Commons) A picture of the Pillars of Creation in the Eagle Nebula (also known as Messier 16)

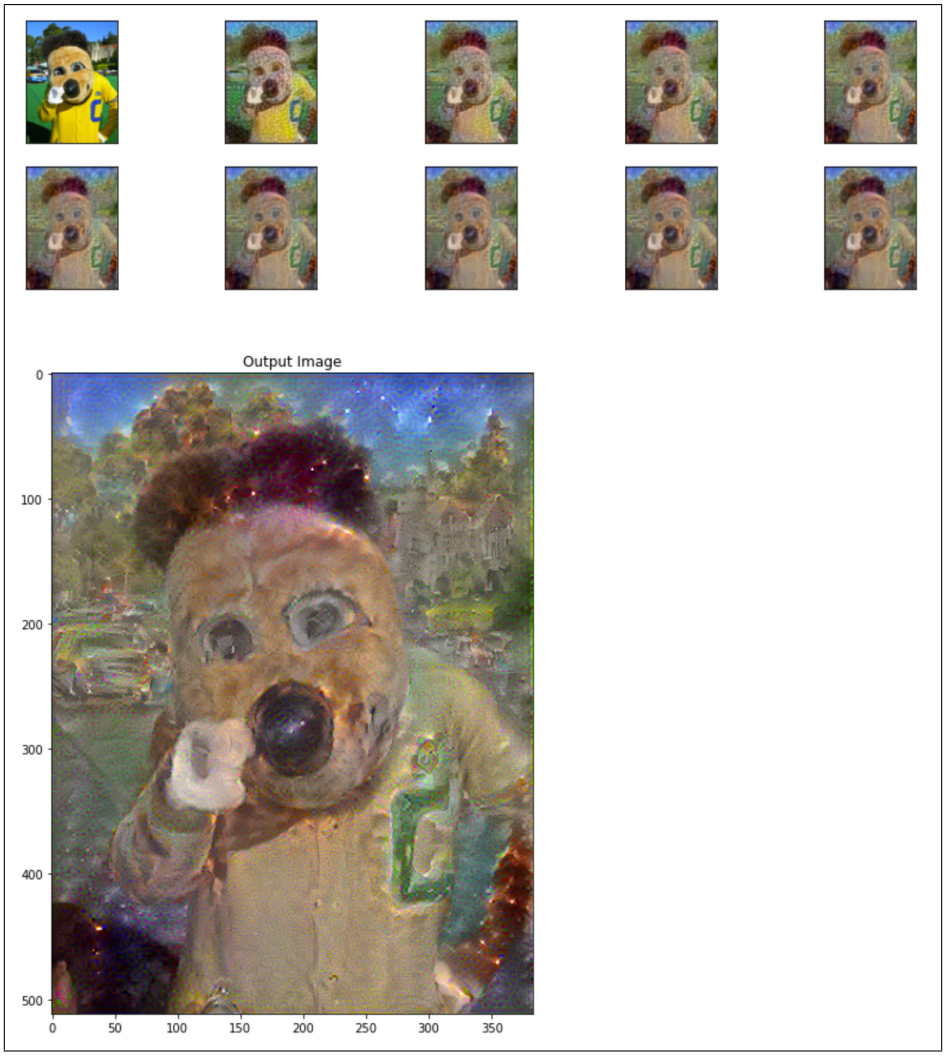

Figure 11: Generated image with the Pillars of Creation style transfer