Neural Style Transfer

Beatrice Caruntu

Abstract

This paper details the design, development, and evaluation of a Neural Style Transfer algorithm, where we apply the style of picture B onto picture A, while preserving the structure and components of picture A.

The steps for performing style transfer consist of visualizing the data, preprocessing/preparing our data, setting up loss functions, creating the model, and optimizing for loss function.

1. Introduction

Neural Style Transfer (NST) refers to a class of software algorithms that manipulate digital images, or videos, in order to adopt the appearance or visual style of another image.[3] The NST problem consists in taking a content image A and a style image B and combining them to produce a new image that has the contents of A and the style of B, as shown in Figure 1.

Figure 1: Neural Style Transfer Example: Left == Image A, Middle == Image B

As we can see in Figure 1, the output image is a blending between the two input images such that it looks like the content image, but "painted" in the style of the style image.

For another example, consider the below image of a turtle and Katsushika Hokusai's famous *The Great Wave off Kanagawa* painting (Figure 2):

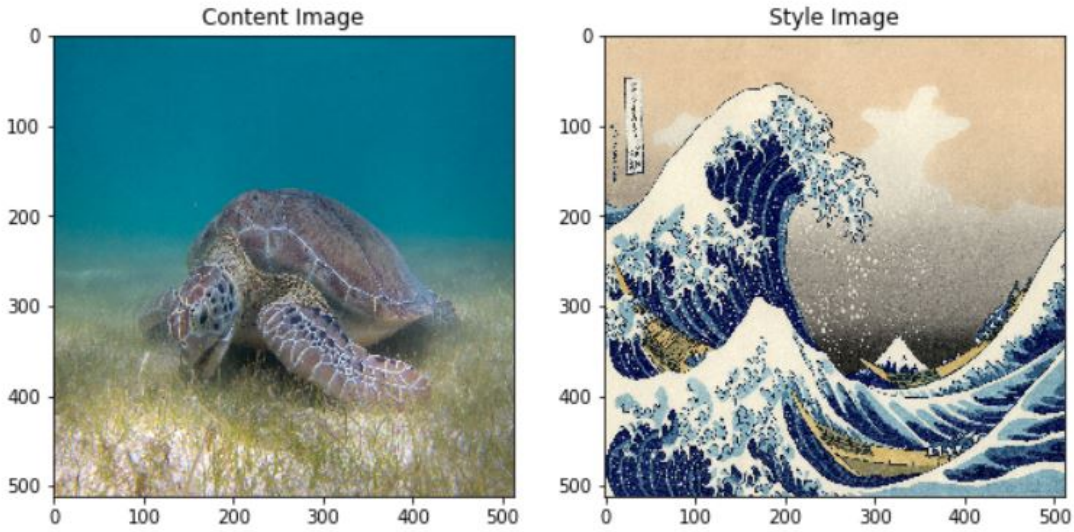


Figure 2: Left: image of a Turtle
Right: Katsushika Hokusai's *The Great Wave off Kanagawa*

Now, image that Hokusai decided to add the texture and style of his waves to the image of the turtle. We can easily image the output to look something like this:

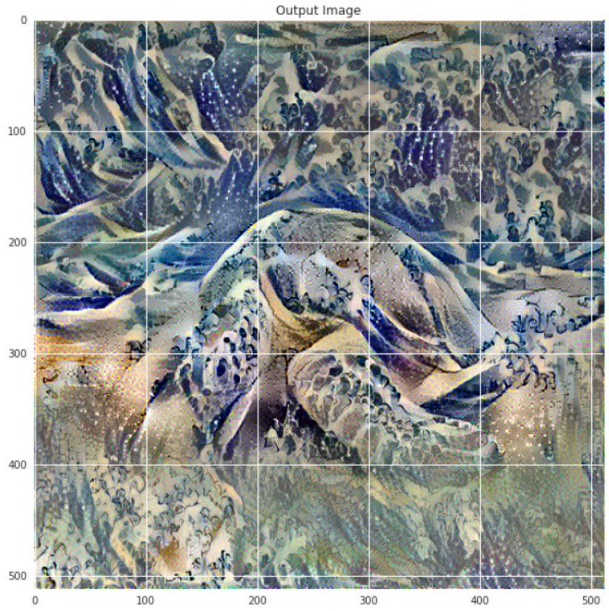


Figure 3: Output image

It is overwhelming that neural networks are capable of creating these artistic styles and NST showcases the aptitudes and internal representations of neural networks.

The principle behind Neural Style Transfer, as explained on Medium (), is to define two distance functions: one that describes how different the content of the images are – "Lcontent", and another one that describes the difference between the two images in terms of their style – "Lstyle". Then, given three images: a desired style image, a desired content image, and the input image (initialized with the content image), we try to transform the input image to minimize the content distance with the content image and its style distance with the style image. [2]

In summary, we will take the base input image, a content image that we want to match, and the style image that we want to match. We will transform the base input image by minimizing the content and style distances (losses) with backpropagation, creating an image that matches the content of the content image and the style of the style image.

2. Background and Related Work

The first paper on NST, called "A Neural Algorithm of Artistic Style", came out in August of 2015. [4] The 3 authors, Leon Gatys, Alexander Ecker, and Matthias Bethge have stated that *"The system uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images."*

A few months later, an app called Prisma was released and became famous overnight. Prisma is an app that allows the user to apply painting styles of famous painters to their own photos, and the resulting pictures are very appealing. A few newspapers described Prisma as *the app that turns photos into works of art*. [1]:

Figure 4: A photo of Serena Williams celebrating her 2016 Wimbledon victory gets the Prisma makeover. Photograph: Photo by Tom Jenkins

Since the first paper in 2015 and the wide use of this algorithm by Prisma, Neural Style Transfer has gotten a lot of attention and, to our luck, there have been a few other related papers that are trying to optimize this style technique.

The authors of "A Neural Algorithm of Artistic Style" [4] noted that humans have the ability to create art through a complex interplay between content and style of an image. However, there was no algorithmic way of doing so. Therefore, they decided to build up a neural network for that specific purpose. They also hoped that their work could help understand how people create and perceive artistic imagery in an algorithmic way.

In this paper, I want to replicate their steps because I want to learn more about deep learning and I think that implementing published deep learning papers from scratch would be a great way of doing so. I also think that Neural Style Transfer is a really interesting and fun topic to work with.

3. Implementation

3.1. Data Collection

For the problem of Neural Style Transfer, we do not need a training set since we are not training any network (our algorithm changes the initial image and does not touch the weights). I collected

20 style images and 20 content images from Google and will present a few results in the analysis section. Below are a few examples of content and style images that were collected:

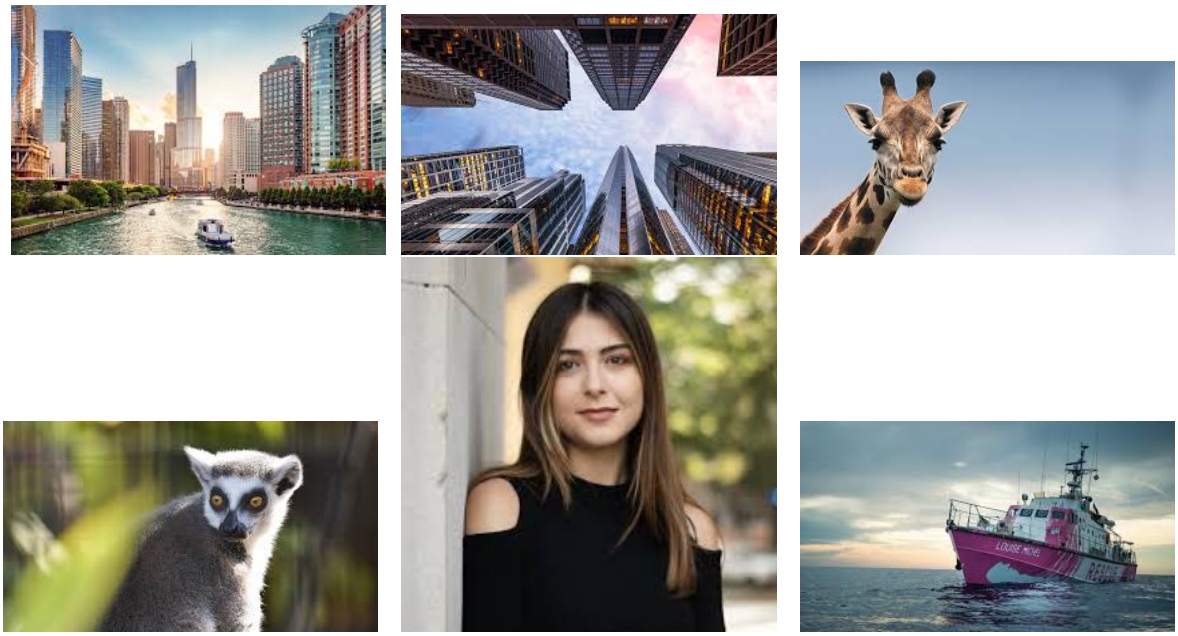


Figure 5: Content Images

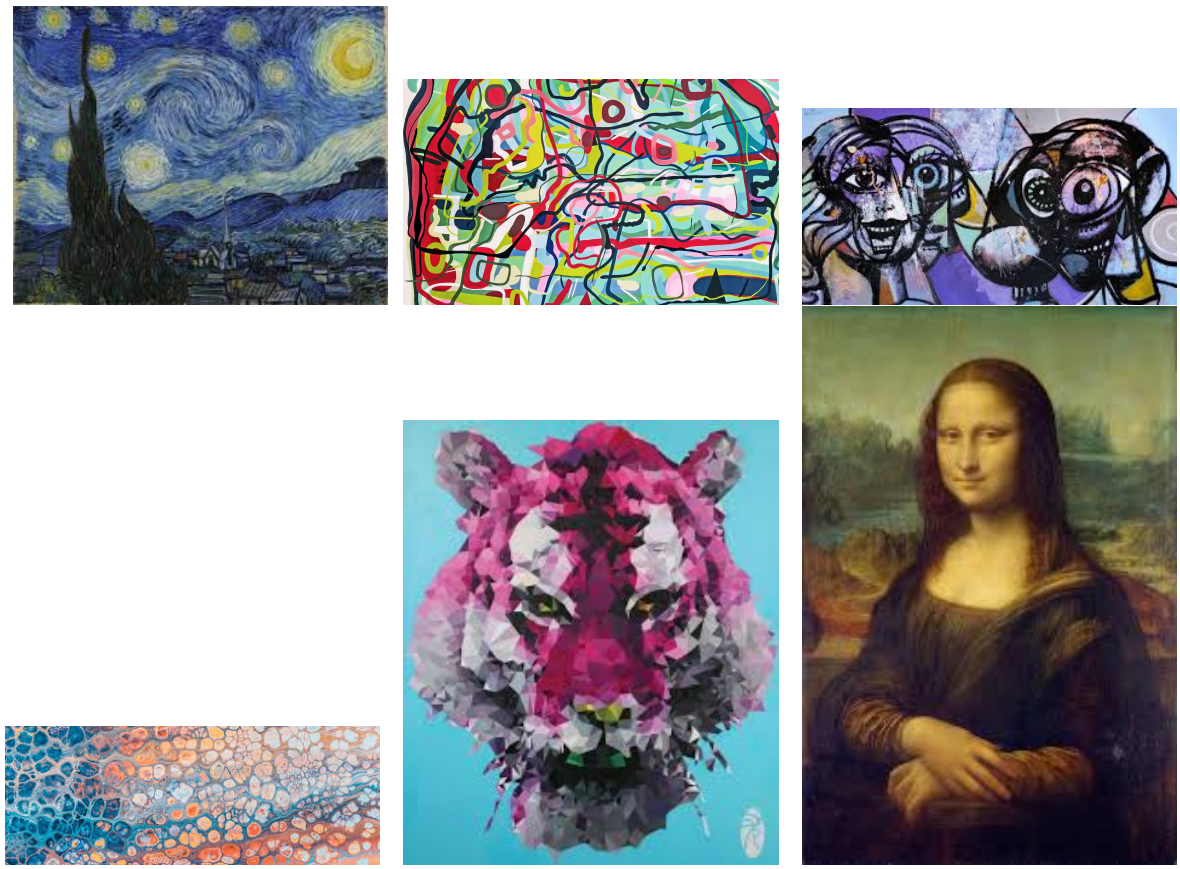


Figure 6: Style Images

3.2. The Model

In Neural Style Transfer we have two images- style and content. We need to copy the style from the style image and apply it to the content image. By, style we basically mean, the patterns, the brushstrokes, etc.

For this, we can use a pretrained VGG-16 net in TensorFlow. The original 2015 paper uses a VGG-19 net, but there is not much difference on impact between these two and VGG-16 is better implemented in TensorFlow. (VGG-16 and VGG-19 are neural networks that use convolutions).

The VGG-16 structure is shown in Figure 7, and the detailed layers are shown in Figure 8 .

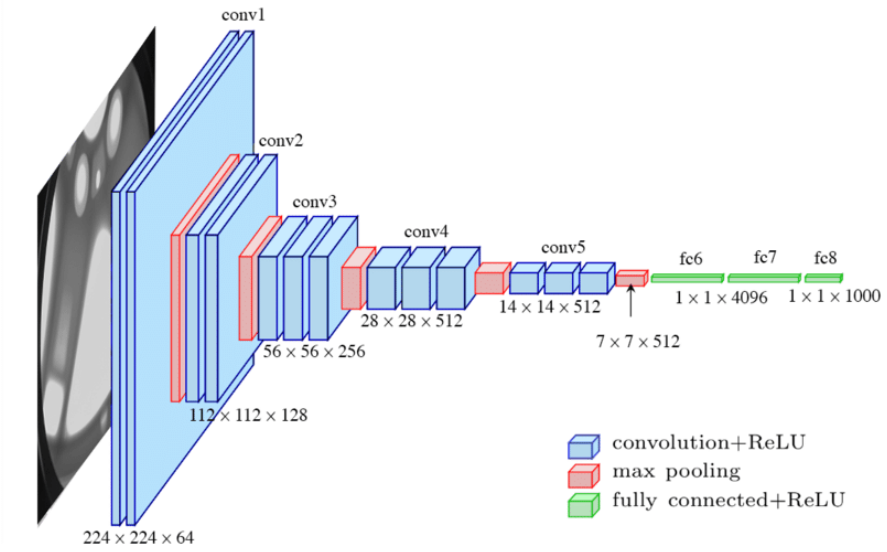


Figure 7: VGG-16

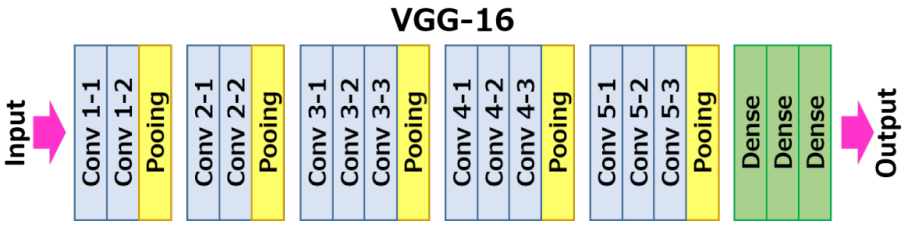


Figure 8: VGG-16 layers

3.3. Convolutional Networks

ConvNets, or Convolutional Networks, work on the basic principle of convolution. Say, for example we have an image and a filter. We slide the filter over the image and take as output, the weighted sum of the inputs covered by the filter, transformed by a nonlinearity such as ReLU (f(x) = max(0, x)) or tanh (f(x) = tanh(x)). Every filter has it's own set of weights, which do not change during the convolution operation.

In the Figure 9, the blue grid is the input. You can see the 3x3 region covered by the filter slide across the input (the dark blue region). The result of this convolution is called a feature map, which is depicted by the green colored grid. So, in a ConvNet, the input image is convolved with several

filters and filter maps are generated. These filter maps are then convolved with some more filters and some more feature maps are generated.

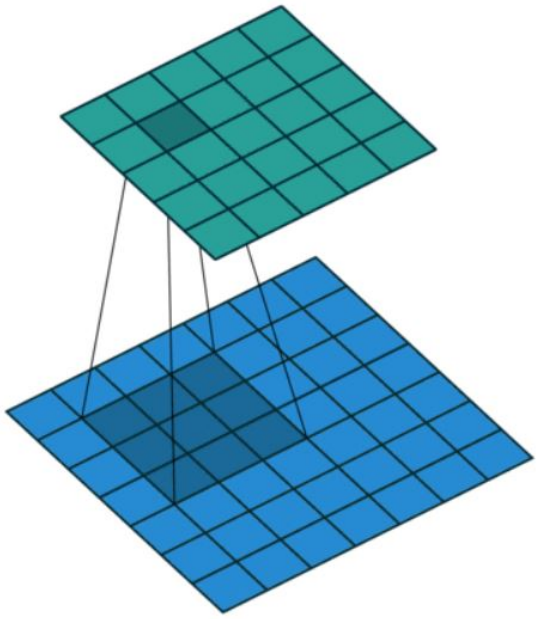


Figure 9: Convolution

Convolutional Networks also have an incredibly interesting feature. Notice in Figure 10that the maps in the lower layers look for low level features such as lines or blobs. As we go to the higher layers, out features become increasingly complex. Intuitively we can think of it this way- the lower layers capture low level features such as lines and blobs, the layer above that builds up on these low level features and calculates slightly more complex features, and so on... Thus, we can conclude that ConvNets develop a hierarchical representation of features. **This property is the basis of style transfer.**

Figure 10: Different levels of layers

3.4. Back to our model: Losses

While doing style transfer, compared to classification problems, we are not training a neural network. Rather, what we're doing is — we start from a blank image composed of random pixel values, and we optimize a cost function by changing the pixel values of the image. In simple terms, we start with a blank canvas and a cost function. Then we iteratively modify each pixel so as to minimize our cost function. To put it in another way, while training neural networks we update our weights and biases, but in style transfer, we keep the weights and biases constant, and instead, update our image.

For doing this, it is important that our cost function correctly represents our problem.The cost function has two terms- a style loss term and a content loss term.

Content Loss:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l} \right)^2$$

This is based on the intuition that images with similar content will have similar representation in the higher layers of the network.

$P^l$is the representation of the original image and F^l is the representation of the generated image in the feature maps of layer l.

Style Loss:

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

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Here,A^l is the representation of the original image and G^l is the representation of the generated image in layer l.$N_l$is the number of feature maps and $M_l$is the size of the flattened feature map in layer$l$. w_l is the weight given to the style loss of layer l. By style, we basically mean to capture brush strokes and patterns. So we mainly use the lower layers, which capture low level features (lines/blobs).

So, the total loss is:

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

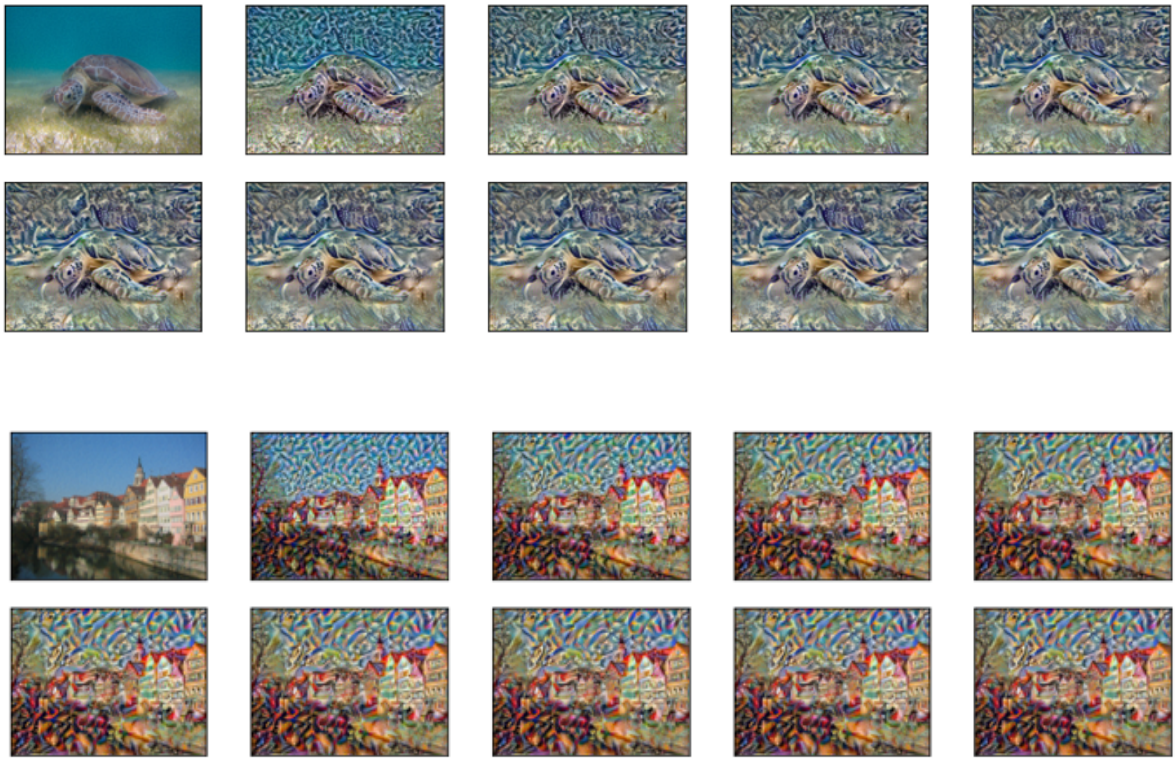
– where alpha and beta are weights for content and style, respectively. They can be tweaked to alter our final result.

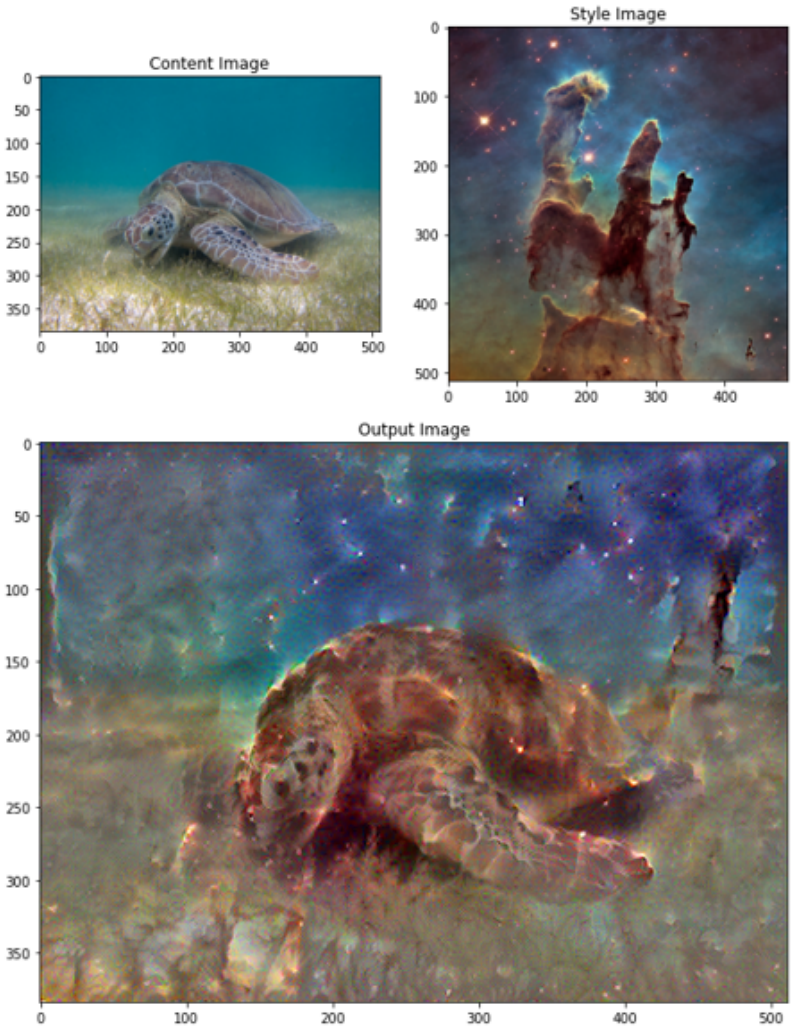
Our total loss function basically represents our problem- we need the content of the final image to be similar to the content of the content image and the style of the final image should be similar to the style of the style image.

Now all we have to do is to minimize this loss. We minimize this loss by changing the input to the network itself. We basically start with a blank grey canvas and start altering the pixel values so as to minimize the loss. Any optimizer can be used to minimize this loss. I have used gradient descent.

4. Results

My results ended up being quite artistic:





5. Conclusion and Future Work

This was an awesome project to work on. I have had a lot of fun playing around with images, but I also learned a lot about Convolutions and Neural Networks. I was mostly used to Classification Neural Networks that are changing the weights in their layers in order to learn. In this project I have learned that it is possible to not touch your layers at all and only update the initial input.

For future work, I am planning to use this technique in a Generative Adversarial Network.

Honor Code

This paper represents my own work in accordance with the university regulations.

/s/ Beatrice Caruntu

References

[1] "Prisma - the guardian." [Online]. Available: https://www.theguardian.com/technology/2016/jul/14/prisma-app-photography-artificial-intelligence-art

[2] "Why everyone is crazy for prisma, the app that turns photos into works of art." [Online]. Available: https://medium.com/tensorflow/neural-style-transfer-creating-art-with-deep-learning-using-tf-keras-and-eager-execution-7d541ac31398

[3] "Wikipedia neural style transfer." [Online]. Available: https://en.wikipedia.org/wiki/Neural_Style_Transfer

[4] M. B. Leon A. Gatys, Alexander S. Ecker, "A neural algorithm of artistic style," 2015. Available: https://arxiv.org/abs/1508.06576