Real-time Neural Style Transfer

Joseph Cheng, 15/12/2020

1 Introduction

Style transfer, or more specifically neural style transfer (NST), describes the technique where the style of one image is applied to the content of another image. The resultant image should keep its content (E.g., if the original image is a horse, the output should still look like a horse) while being radically different in terms of style (E.g., different texture/colours). This technique was first described in a paper by Gatys in 2015 [1]. Real-time NST aims to solve the issues that normal NST suffers from that prevent it being used in real-time applications.

2 Neural Style Transfer

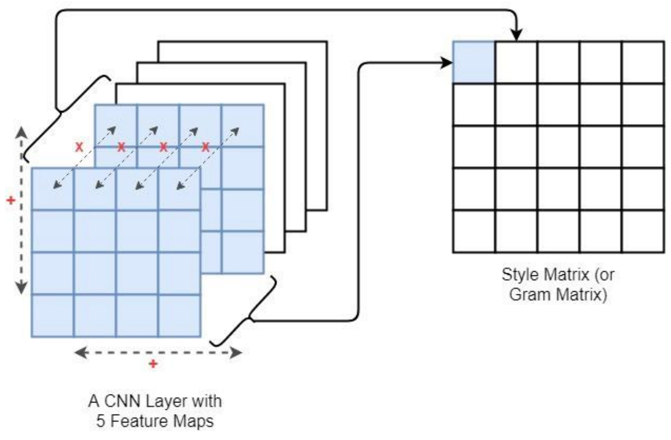
This section aims to explain the paper described by Gatys [1] as well as the results of the implementation of the paper. In a pretrained convolutional neural network (CNN), the outputs of the intermediate layers can be extracted and used to extract content and style information about an image. This forms the basis of our NST algorithm. The deeper we traverse into the CNN, the more specific the features the layers represent. This means that the bottom layer's output can be extracted to represent our content information. Similarly, the upper layers can be extracted to represent our style information.

Loss functions. In the paper, Gatys describes 2 loss functions for the NST algorithm: content loss, and style loss. Minimizing content loss ensures that the activations in the lower layers (closer to the output layer) are similar between the content image and the generated image. This is a loss function that focuses more on per-pixel values as we use the lower layers. The content loss defined in the paper is in essence the RMSE between the activations produced by the image and generated layer.

$$L_{content} = \frac{1}{2} \sum_{i,j} (A^l_{ij}(g) - A^l_{ij}(c))^2$$

Content Loss Formula [6]

Style loss on the other hand is much more complex and difficult to define. This is because style is represented by multiple layers, hence we need to find a way to combine the outputs of all the layers in the CNN. We do this by computing a style matrix, which is essentially a Gram matrix of all the feature maps.



How the Style Matrix is Computed for a CNN Layer with 5 Feature Maps [6]

The idea is that "style information is measured as the amount of correlation present between feature maps in a given layer" [6]. Hence, the intuition behind the Gram matrix is that we are trying to capture the "distribution of features" of a set of feature maps in each layer. By minimizing the RMSE between the style matrix of the style image and the generated image, we are essentially matching the distribution of the features between the two images.

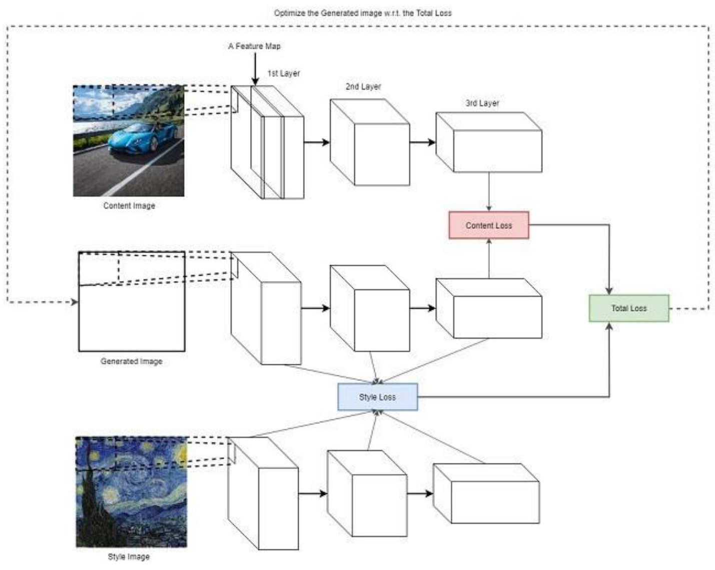
$$L_{style} = \sum_l w^l L^l_{style} \text{ where,}$$

$$L^l_{style} = \frac{1}{M^l} \sum_{ij} (G^l_{ij}(s) - G^l_{ij} g)^2 \text{ where,}$$

$$G^l_{ij}(I) = \sum_k A^l_{ik}(I) A^l_{jk}(I).$$

Style Loss Formula [6]

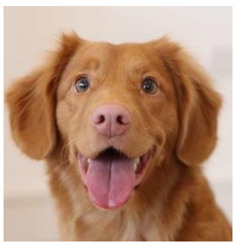
Running neural style transfer. In essence, style transfer is an optimization problem. The algorithm is simple once we define both our loss functions. First, we create two models from a pretrained CNN (VGG16) loaded with the weights obtained by training on the ImageNet dataset. The two models will be our style and content extractors.



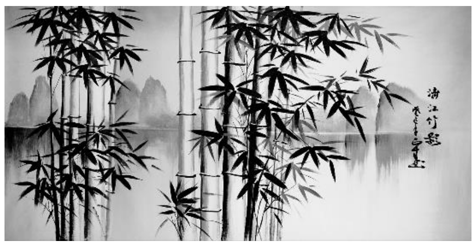
High level Architecture of NST [6]

We then initialize our generated image (g) with our content image (c). This means that initially, content loss is 0 as there is no difference between (c) and (g). We then perform regular gradient descent, computing the content and style losses, and finally applying the gradients to (g) itself after every batch.

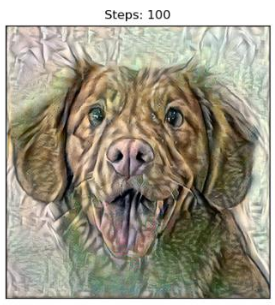
Results. Below are the results of the style transfer algorithm described above. The implementation follows the Keras/TensorFlow tutorial closely.



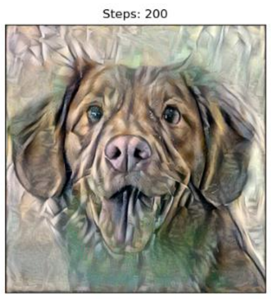
Content Image



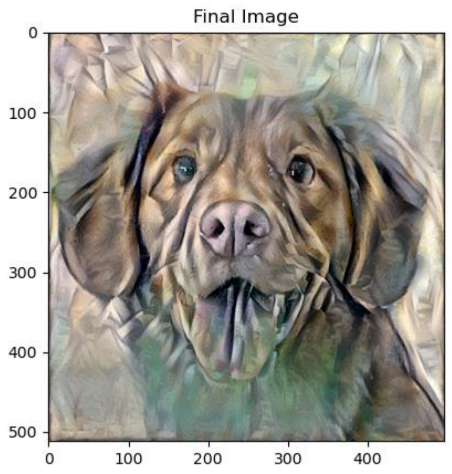
Style Image



After 100 Steps



After 200 Steps



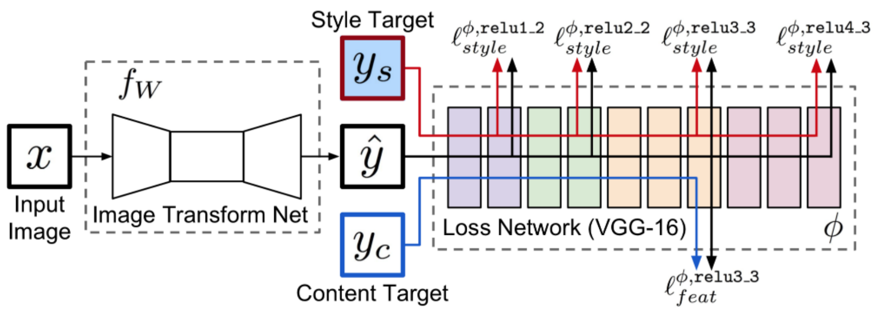
Final Image (300 Steps)

As we can see, the longer we train, the more “blended” the generated image is. The style starts blending in with the content image and we achieve a cohesive final generated image. The time taken for the final image to be generated is about 40-60 seconds. This was run on a GTX 1050 Ti GPU and TensorFlow GPU mode was on. Next, we will see why this algorithm is unsuitable for real-time style transfer and the solution to the issues.

3 Real-Time Style Transfer

Real-time NST refers to stylizing a video stream in real-time. The main challenge, as you might have already guessed, is speed. Previously, our implementation of NST can style 1 image every 40 seconds at best. This is unacceptable for real-time applications as it would basically be less than 1 FPS. The paper by Johnson [2] aims to solve this issue.

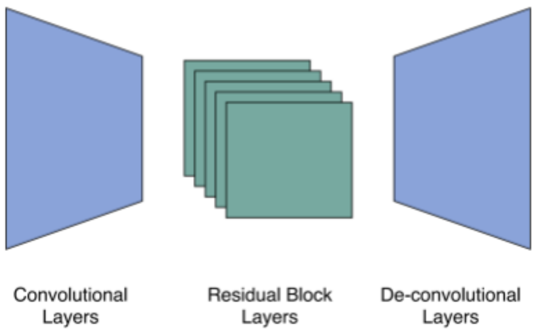
Image transformation net. The key reason why normal NST is slow is because we need to perform multiple forward feed/back propagation passes through the pretrained network in run-time. However, if we can shift this portion offline if we train an image transformation network. This image transformation net sits on top of our loss network, which is essentially what we have been doing in our normal NST regarding computing the loss functions. This image transformation net takes in the content image and generates a new image based on the style image it was trained on.



Style Target
y_s
f_W
x
Input Image
Image Transform Net
$\hat{y}$
y_c
Content Target
$\ell^{\phi,\text{relu1\_2}}_{style}$
$\ell^{\phi,\text{relu2\_2}}_{style}$
$\ell^{\phi,\text{relu3\_3}}_{style}$
$\ell^{\phi,\text{relu4\_3}}_{style}$
Loss Network (VGG-16)
ϕ
$\ell^{\phi,\text{relu3\_3}}_{feat}$

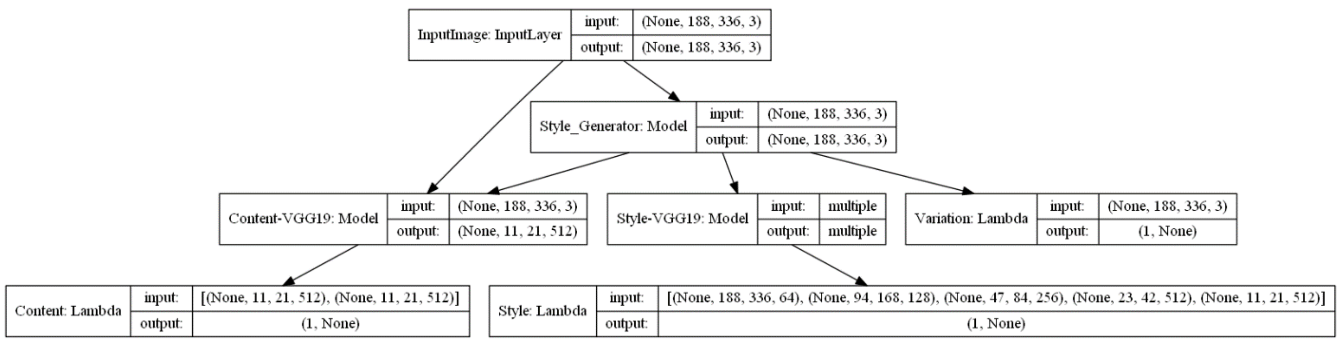
High-level architecture [3]

As for the architecture of the image transform net recommended in the paper by Johnson [2], it will consist of 3 convolutional layers, followed by 5 residual blocks, and ending with 3 de-convolutional layers or upsampling layers. Each convolutional block uses ReLU as its activation function and is immediately followed by batch normalization. The high-level architecture of the image transform net can be seen below.



High-level architecture of Image Transform net [3]

Finally, the complete training model implemented in Keras is shown below.



InputImage: InputLayer | input: (None, 188, 336, 3) | output: (None, 188, 336, 3)
Style_Generator: Model | input: (None, 188, 336, 3) | output: (None, 188, 336, 3)
Content-VGG19: Model | input: (None, 188, 336, 3) | output: (None, 11, 21, 512)
Style-VGG19: Model | input: multiple | output: multiple
Variation: Lambda | input: (None, 188, 336, 3) | output: (1, None)
Content: Lambda | input: [(None, 11, 21, 512), (None, 11, 21, 512)] | output: (1, None)
Style: Lambda | input: [(None, 188, 336, 64), (None, 94, 168, 128), (None, 47, 84, 256), (None, 23, 42, 512), (None, 11, 21, 512)] | output: (1, None)

Training. In the paper, it was recommended to train the model on the MS COCO 2014 [8] dataset. However, the dataset was too big, containing about 85,000 images. Hence, I opted for the similar but smaller Pascal VOC 2012 [9] dataset. I tried training the model on 20 epochs, with a batch size of 2. However, the results were not great. Hence, I opted training for more epochs (50) instead. Each epoch takes about 5 hours on the school computer, which would mean a total of 250 hours or 10+ days. This was way too long, and I had to do it on my laptop instead. On my laptop, the model training per epoch was 40mins, which gave it a reasonable training time of about 30+ hours.

Complications. Prior to training, my laptop has a history of overheating. 20 hours into training, my laptop made a huge screeching sound and I unfortunately had to turn off my laptop at that point. The result was an untrained model and a blown laptop fan.

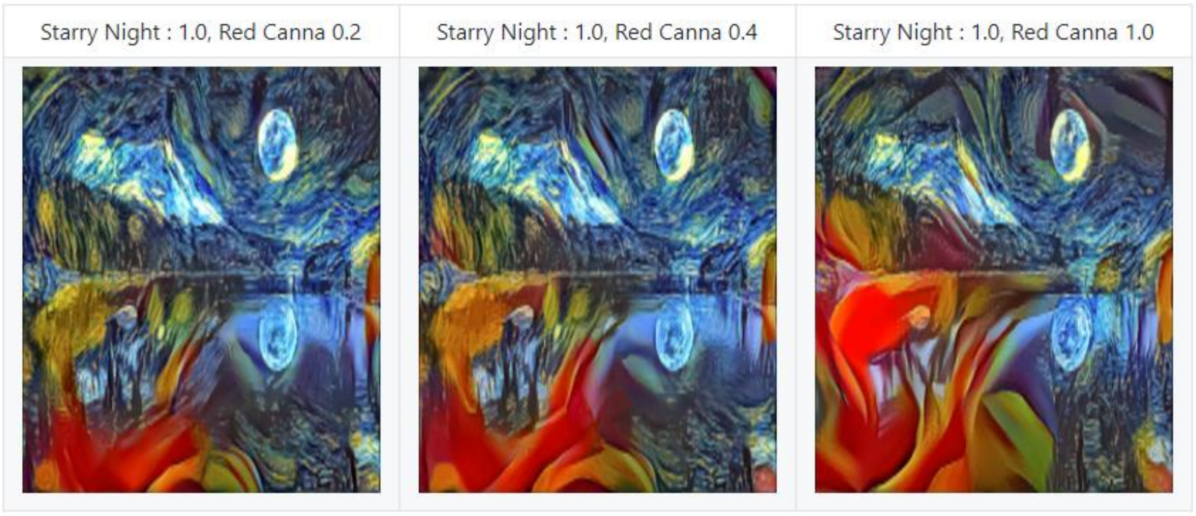
I had no time to do a new model, however, I found online weights that were trained in the same way that I had implemented my model. The results were great compared to our first NST algorithm. I read each frame of the video feed and passed it through our model and the displayed the generated image. The results are an estimated 24fps stylized video stream.

4 Related Work

While the applications of style transfer seem narrow, there are many other variations and additions to style transfer that make it interesting. One interesting addition is style transfer with colour preservation [7]. Usually when we perform style transfer, the colour of the original content image is changed as well. This is because colour is captured as style information. Colour preservation counters that by ensuring that the generated image has the same colours as the content image while retaining the other stylistic features of the style image.

Another addition to style transfer is removing the restriction of just 1 style image. [7]

The below are the results after 50 iterations using 3 different style weights :


Starry Night : 1.0, Red Canna 0.2
Starry Night : 1.0, Red Canna 0.4
Starry Night : 1.0, Red Canna 1.0

Other interesting additions are masked transfer, silhouette transfer, and texture transfer. These techniques can be used together or in stages or just individually, which makes them more interesting and worth looking at.

References

[1] A Neural Algorithm of Artistic Style, Gatys, 2015

[2] Perceptual Losses for Real-Time Style Transfer and Super-Resolution, Justin Johnson, 2016

[3] Real Time Video Neural Style Transfer, Kate Bäumli, 2018

[4] Real-time Style Transfer, Chimezie Iwuanyanwu, 2019

[5] Stabilizing neural style-transfer for video, Jeffrey Rainy, 2018

[6] Intuitive Guide to Neural Style Transfer,Thushan Ganegedara, 2019

[7] Neural Style Transfer, titu1994,

[8] MS COCO 2014 Dataset

[9] Pascal VOC 2012 Dataset