Report on NST - The Basic and the AdaIN

By V Shashank Bharadwaj,22116099,ECE

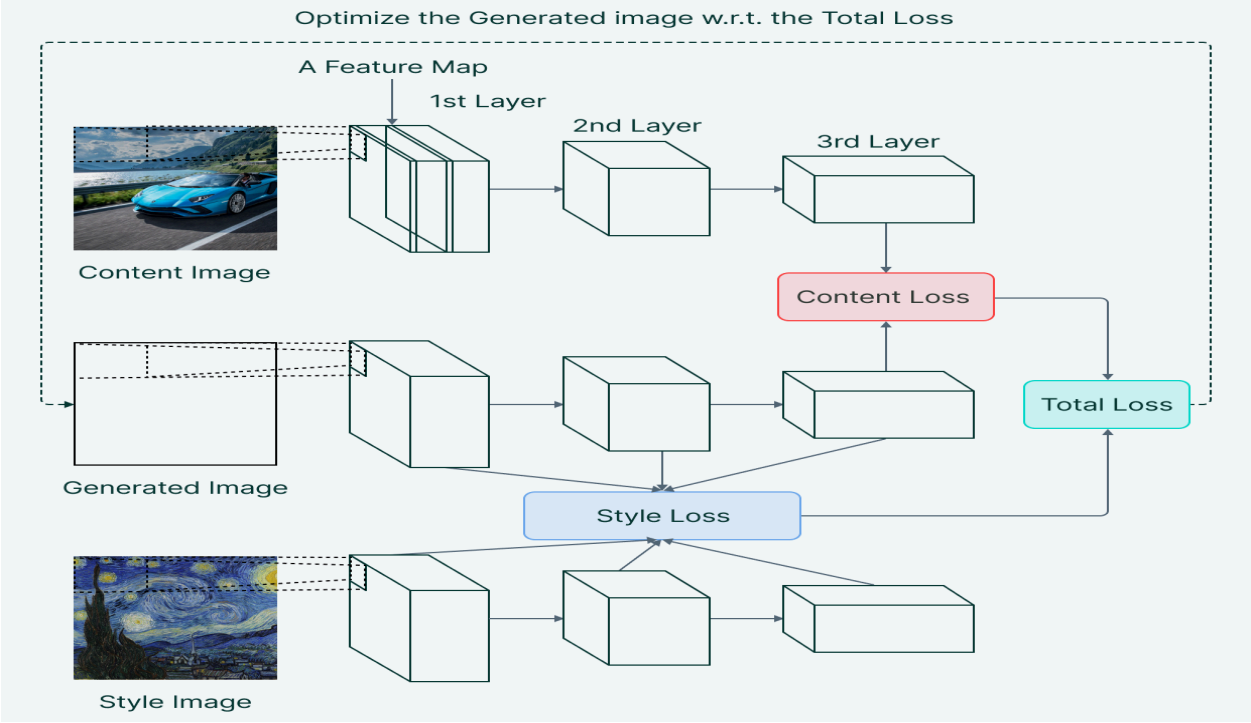
Introduction

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. Essentially, a technique and method to convert one image into another's style, without losing content. This project of mine explores how this can be achieved in two different ways, a basic optimization based approach, and a faster, more standard forward-only approach.

Approach and some Overview

First, the iterative, optimization based approach. The paper titled "A Neural Algorithm of Artistic Style" introduced us to the modern way of dealing with NST using deep neural networks. It also gave us the key concept of perceptual loss.

So How does the iterative approach work? We pass a starting image (white-noise or just pure black images perhaps are the best illustrators) through a decoder that has been pre-trained. The most widely used one is the VGG-19 model that I have used as well. The decoder acts as a feature extractor from which we get our content and style losses, which are the parts of the perceptual loss.



In this model, the ‘weights’ to be trained are the image itself. We treat the generated image as the only part of the network that is trainable, and go through backward passes and change the image.

Content loss

Content loss is simple in comparison to style loss : Pass the image through the decoder, choose one point/layer in the decoder, take the tensor obtained at the layer. We do this for both content image and generated image, and take an MSE loss between them. In essence, the difference between the tensors is obtained (which is also a tensor), each individual term is squared and then added.

The interesting thing about content loss, which arises due to the properties of image identification neural networks, is that an earlier layer considered for the content loss leads to more of an exact replica of the original image. The reason is because deeper layers end up specializing in detecting specific things in the network and actually lose spatial information on where said items were located in the image. So if a deeper layer is considered, the network may end up with an image which has all the same items but in a different location and style.

Style loss

This is, in my opinion, the more innovative part of Neural Style Transfer.
To match the style of the generated image and the style image, we minimize the mean-squared distance between the entries of the Gram matrix from the original image and the Gram matrix of the image to be generated. See image for illustration, the original paper explains it clearly.

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

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

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

Another important point to note is that the style loss uses multiple layers of the network as well as the multiple channels in each layer to make a complete style loss.

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

Where p is the content image, x is the generated image and a is the style image.

A point to note is that rather than alpha and beta themselves, their ratio is far more important for theoretical purposes. For practical purposes, we have to ensure they are large enough to prevent vanishing gradients in the final steps, and small enough to prevent exploding gradients in the initial steps.

Adaptive Instance Normalization

The reason why I specifically chose these two methods to implement is because both the optimization method and the AdaIN are **arbitrary style transfer algorithms.** There are other papers that implement possibly faster methods, but these two are unique in that they have no restriction on the style of the output image.

Now the reasoning of why some of AdaIN works is empirical. However, it works excellent, so we discuss and use it.

Before AdaIN, researchers figured out that batch normalization and instance normalization significantly sped up the training process of the feedforward neural network designed for NST.

What are these two things? Batch Normalization is re-centering and re-scaling a tensor to the mean and variance of the the batch before passing it on to the next layer. It ensures that all the vectors have the same mean and standard deviation.

In Batch Normalization, we compute the mean and standard deviation across the various channels for the entire mini batch.
In Instance Normalization, we compute the mean and standard deviation across each individual channel for a single example.

$$\mathrm{IN}(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

This instance normalization increased the rate of training even more than batch-norm. The next innovation was then a Conditional Instance Norm that said for each individual style, let there be different affine parameters. This development allowed multiple style usages in networks, but only pre-trained styles could be used.

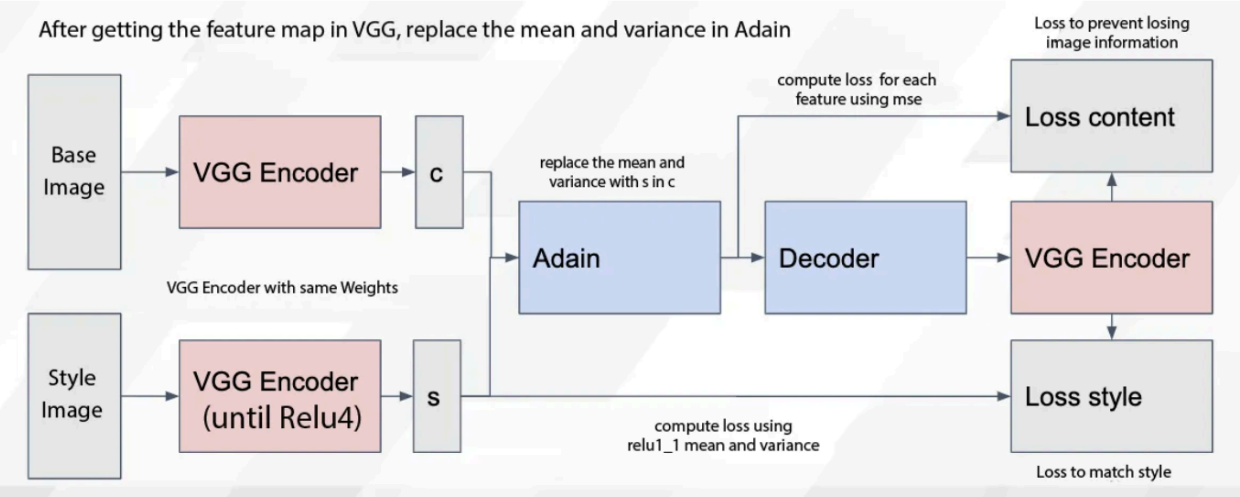
On further studying why the Instance Norm was faster, it was understood that the IN performs a kind of style Normalization. IN can normalize the style of each individual sample to the target style. Training is facilitated because the rest of the network can focus on content manipulation while discarding the original style information. The reason behind the success of CIN also becomes clear: different affine parameters can normalize the feature statistics to different values, thereby normalizing the output image to different styles.

Thus the idea for AdaIN :

$$\text{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

Where x is the content image and y is the style image. AdaIN receives a content input x and a style input y, and simply aligns the channelwise mean and variance of x to match those of y, and so having no learnable parameters.

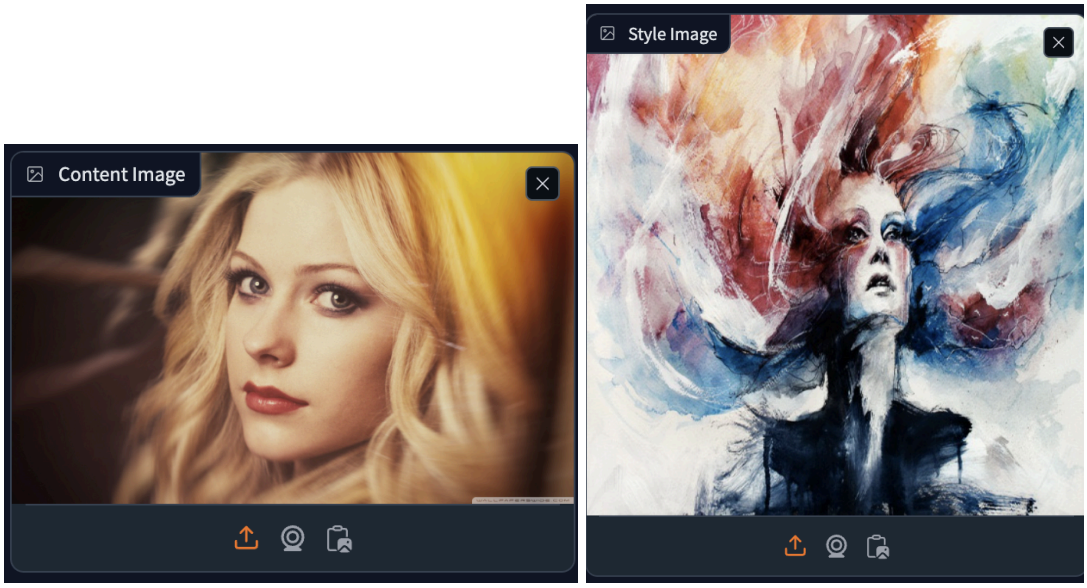
The limitation as in with any neural network is the available resources. Even more so in the case of feedforward networks like the AdaIN implementation I have used.



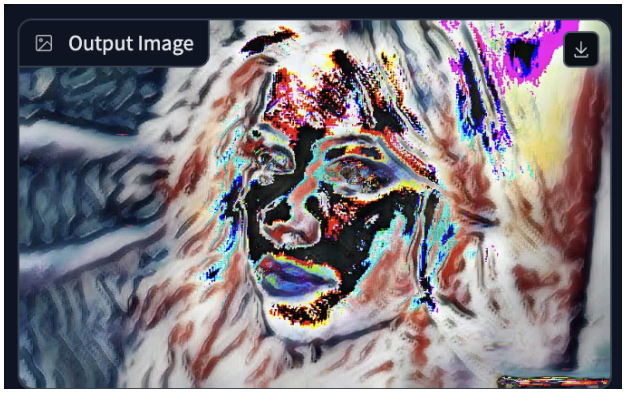
Some interesting Observations :

A way to visually test how well your model has understood the task is to just give the same image for both style and content, the output should be the same image. This is the case for the basic, optimization based approach(with enough epochs of training) and a well trained feed-forward network. Poorly trained networks do not perform well.
The one on the left is the one using the very well trained decoder given by the original authors themselves. The one on the right is a model trained by me for far fewer epochs. The difference is stark.

Results from AdaIN :



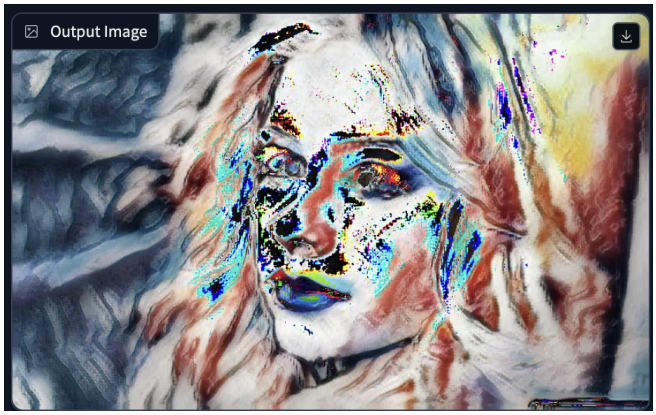
5000 Epochs of training:



10000:

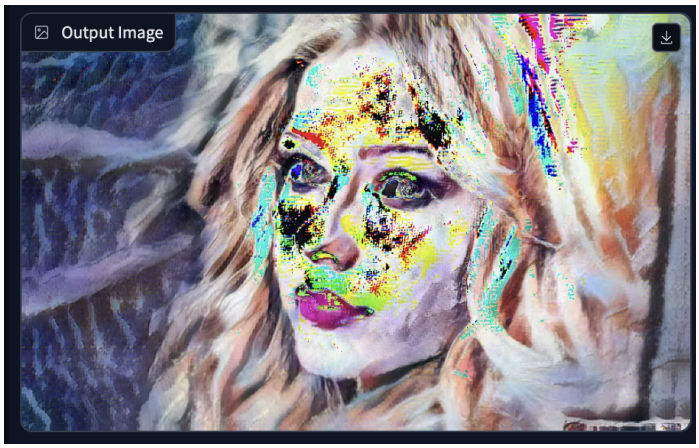
15000:

16000:

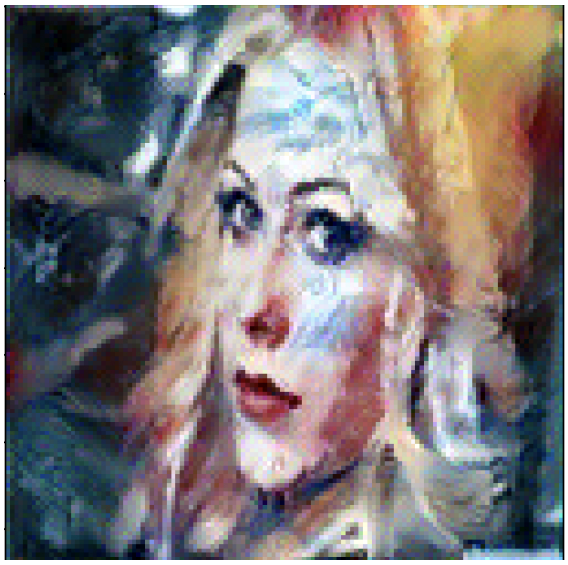


Decoder.pth:

160000 epochs (Not by me, but the unofficial pytorch implementation) :



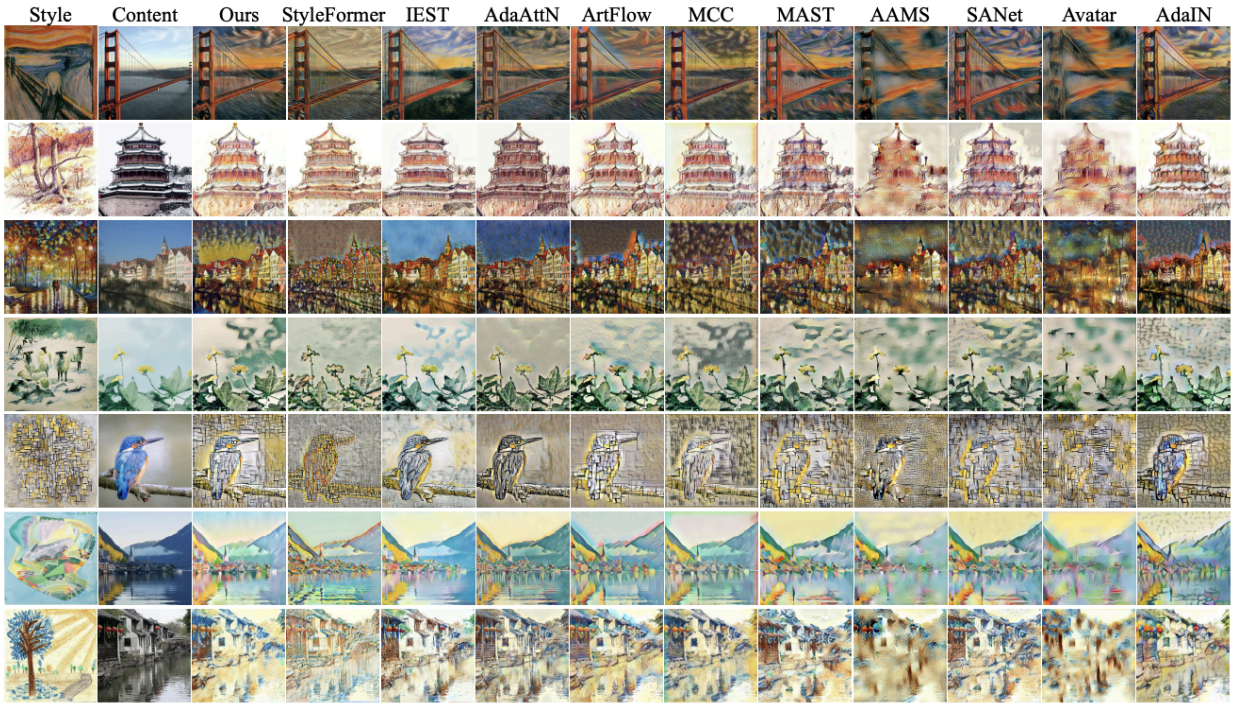
Iterative Approach :



The iterative approach has performed better, but it is far slower for individual tasks. A point of note is that while running the iterative approach, I did not use a gpu, so the resolution of the image is poorer in comparison to the other images.

Discussion and Improvements :

It is no difficult task to substitute the VGG-19 model with another model, so that can be done. NST can also be done with transformers. The following image is the results of the paper “StyTr2 : Image Style Transfer with Transformers” :



g. 4. Qualitative comparisons of style transfer results using different methods.

This clearly shows the potential of transformers in NST, but of course these are things that have already been accomplished. BUT I have much less experience working with transformers, so I did not attempt it.

A point of discussion regarding the AdaIN paper is the affine parameters mentioned in the CIN section. I have no knowledge of why the affine params are set to be the mean and variance, they just seem to work. It is also possible to maintain color control, which, while existing in the Github of the pytorch implementation, I have essentially ignored for my purposes. It's there in the files I have made, however it is also incomplete, for now.

AdaIN is the fastest among the feed-forward network NSTs and also has multi-style capability. The optimization based approach, while slow, also has the highest chance of giving us a good image given enough iterations.

The optimization approach implemented in my project does not start with white noise, but with the content image itself. This allows the network to only focus on style loss and thus give a satisfactory output in a reasonable amount of time, even on CPU.

References :

1 . A Neural Algorithm of Artistic Style - https://arxiv.org/abs/1508.06576

2 . Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization - https://arxiv.org/abs/1703.06868
3 . The unofficial pytorch implementation of the paper - Link is to the GitHub