A Neural Algorithm of Artistic Style

E4040.2018Fall.ARAG.report
Akanksha Rajput ar3879, Gurkanwar Singh gs3006, Alokik Mishra am4797
Columbia University

Abstract

We reproduce the results of a paper by Gatys et al. to separate and recombine style representation of an image with the content representation of another image to produce an artistic image, using a CNN based neural algorithm. Further, we discuss some variations to improve the results of the algorithm.

1. Introduction

Humans have until now been unable to come up with an algorithmic process of composing an image with an interplay between the content and style of different images. A paper by Gatys, Ecker and Bethge first proposed a neural representation to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images. This has paved the way for building several creative photo and video apps in addition to advancing the algorithmic understanding of how humans create and perceive artistic imagery.

The algorithm is based on a Convolutional Neural Network (CNN) algorithm where in the higher layers the input image is transformed into representations which care about the actual *content* of the image. The *style* representation is extracted from the lower layers. We combine the two representations with their respective weight factors to generate an image with a mix of style and content of the respective images.

2. Summary of the Original Paper

In this section we will briefly discuss the broad methodology and results from the paper "A Neural Algorithm of Artistic Style" (Gatys, Ecker, Bethge, 2015)[1].

2.1. Original Paper Methodology

The paper uses CNNs, in particular the 19 layer VGG[2] (see figure 1) to separate artistic style from an image content. A few changes are made to the model including using average pooling instead of max pooling, and omitting the fully connected layers. The authors propose that the content representation is learned through the feature responses in the higher layers of the network. Artistic style on the other hand can be captured through the correlations of the different filter responses. The authors then used fit a white noise image using a combined loss function assigning weights to a loss function for the content representation learned from a photograph and the style representation learned from a piece of art.

2.2. Original Paper Results

The authors were able to reproduce the image 'Neckrafront' in various different renditions, combining the content of the image successfully with several different pieces of art. A few of the outcomes can be seen in figure 2 below. The two images seen combine the content of the 'Neckarfront' with the styles of *The Shipwreck of the Minotaur* by J.M.W and *Der Schrei* by Edvard Munch,1893.

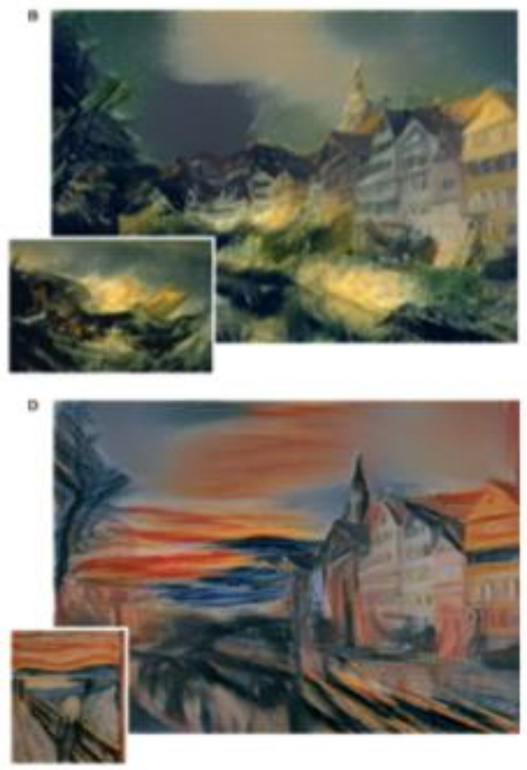


Figure 1: Some reproductions from the original paper.

3. Our Methodology

In this section we will first describe our methodology, beginning with our objective and then a description of our design, including areas we explore beyond the existing works.

3.1. Objective and Technical Challenges

We begin by first replicating the work of the author using the original images. We take use the existing 19-layer VGG architecture, following the authors' example of removing the FC layers and using average pooling rather than max pooling in our model. Our first objective is to ascertain that we can indeed represent similar aesthetics to those found in the original paper. We then go further by exploring new areas such as using multiple pieces of art for a "joint-style" transfer. We also explore the trade-off between the runtime and aesthetic improvement of the output image. One of the challenges was that since there was no 'test' set we had no standard metric to validate our model. Thus while we could optimize weights, optimizing runtime was significantly more difficult and require subjective judgement.

3.2. Problem formulation and design

We follow the authors of the original paper by minimizing the following combined loss function on a white noise image:

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

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the relative weights assigned to content and style respectively, $\vec{\boldsymbol{x}}$ is the white noise image, $\vec{\boldsymbol{p}}$ is the content image, and $\vec{\boldsymbol{a}}$ is the style image. Taking the derivative of the above function with respect to $\vec{\boldsymbol{x}}$ allows us to use both the content representation from the filters based on $\boldsymbol{\mathcal{L}_{content}(\vec{p},\vec{x})}$ and from the correlations between the filters based on $\boldsymbol{\mathcal{L}_{style}(\vec{a},\vec{x})}$.
A possible extension to the model is to use multiple pieces for style where the model would look like:

$$\boldsymbol{\mathcal{L}_{total}(\vec{p},\overrightarrow{a_1},\overrightarrow{a_2},\vec{x}\,,) = \alpha\mathcal{L}_{content}(\vec{p},\vec{x}) + \beta_1\mathcal{L}_{style}(\overrightarrow{a_1},\vec{x}) + \beta_2\mathcal{L}_{style}(\overrightarrow{a_2},\vec{x})}$$

With the weights $\boldsymbol{\alpha, \beta_1, \beta_2}$ chosen appropriately based on the degree of desired influence of the style on the output image.
To arrive at a decision of after how many iterations we can say that the output image is good enough, we propose a function for quality of image Q(t) with the following desirable characteristics:

$$\frac{\partial Q_t}{\partial t} \geq 0 \text{ and } \frac{\partial^2 Q_t}{\partial t^2} < 0$$

4. Implementation

In this section, we describe the different parts of our implementation of the algorithm in Tensorflow from the preprocessing of the input content and style images to the use of a pre-trained VGG-19 model on them to finally producing an image combining the style and content of the input images.

4.1. Deep Learning Network

4.1.1. Architectural Block Diagrams

We use the feature space provided by a pre-trained VGG-

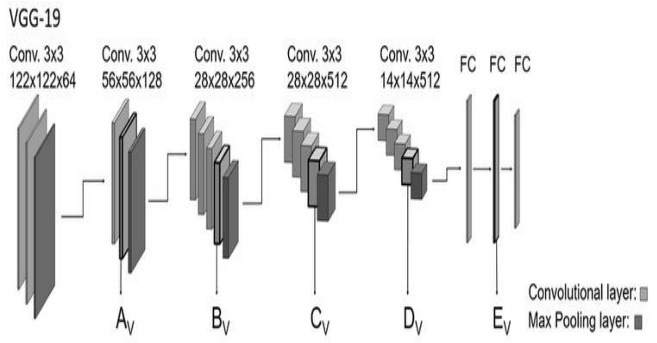


Figure 2 The 19 - layer VGG net. Note that for the paper, the authors chose to use average pooling layers rather than max pooling and omitted the 3 FC layers.(Image: https://www.researchgate.net/figure/VGG-19-architecture-with-marked-cut-off-layers-used-for-transfer-learning_fig1_325869080)

19 model (with 16 convolutional and 5 pooling layers) as shown in Figure 3.

4.1.2. Training Algorithm Details

After loading the VGG-19 model and setting the initial parameters like image height and width, we implement a function to build the result of the operation of convolution function and ReLU activation function on the input image at each layer of the VGG. We have replaced the max-pooling operation by average pooling as it gives more appealing results. We then load the content and style images, resize them to a fixed size and then subtract the mean values from them, because this is the mean to subtract from input to VGG when it was used to train. We then define the functions to calculate the content loss and style loss. Content loss is defined as follows:

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

Where F^l_{ij} is the activation of the ith filter at position j in layer l applied on the original content image and P^l_{ij} is the feature representation of the image to be generated. The idea is to perform gradient descent on a white noise image to find another image that matches the feature responses of the original image. 'conv4_2' is usually used to build the content representation.

To build a style representation, we compute the Gram matrix which calculates the correlations between the different filter responses by taking the inner product between the vectorized feature map I and j in layer l. The contribution of layer l to the Style loss is given as:

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

Where G^l_{ij} is the Gram matrix of the image to be generated and A^l_{ij} is the Gram matrix of the input style image, where layer l has N distinct filters and N of size M (product of height and width of the feature map).

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

The total style loss is then:

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

Where w_l is the weight of layer l. We use 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' layers to build this style representation. To generate the final image as a mix of content and style image, we calculate the total loss as:

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

Where α and β are the weighting factors for content and style reconstruction respectively.

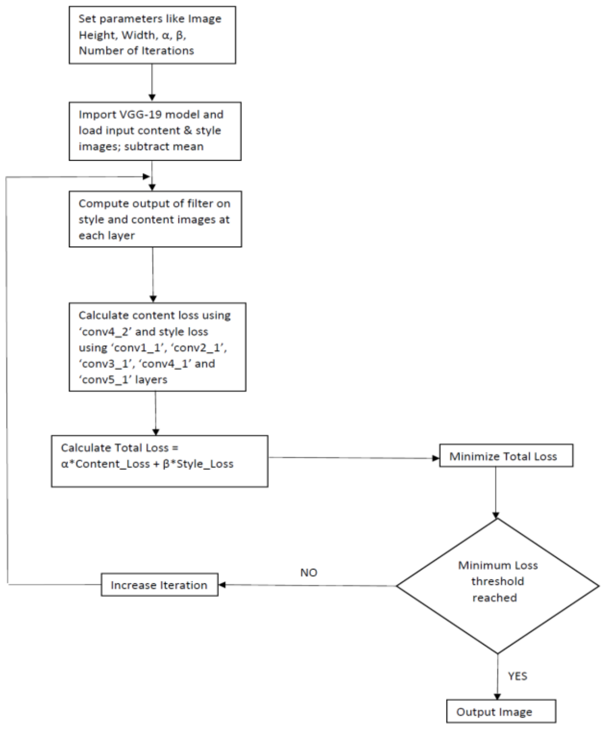
We then initialize the Tensorflow session and minimize the total loss using an optimizer like Adam, because it is an adaptive optimization technique and converges relatively quickly. We have used 5000 iterations of the minimization cycle to get our result image.

4.1.3. Data Used

We tried many different artistic images including *The Starry Night* by Vincent van Gogh, *The Shipwreck of the Minotaur* by J.M.W. Turner, *Guernica* by Pablo Picasso and content images including the Tubingen photo (as used in the paper) and some photos of New York. The pre-trained VGG-19 model is taken from the MatConvNet MATLAB toolbox.

4.1.4. Flowcharts

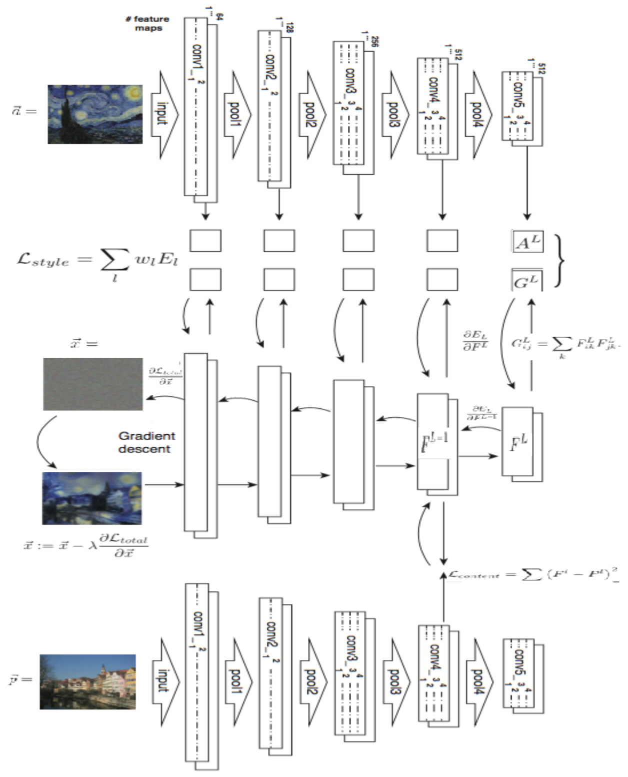
Below (Figure 3) is the flow of the our implementation of the algorithm in Tensorflow:



Set parameters like Image Height, Width, α, β, Number of Iterations
Import VGG-19 model and load input content & style images; subtract mean
Compute output of filter on style and content images at each layer
Calculate content loss using 'conv4_2' and style loss using 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' and 'conv5_1' layers
Calculate Total Loss = α*Content_Loss + β*Style_Loss
Minimize Total Loss
Minimum Loss threshold reached
NO
Increase Iteration
YES
Output Image

4.2. Software Design

Figure 4 below illustrates the overall design of the whole process of our implementation.



# feature maps
64
128
256
512
512
$\vec{a} =$
input
conv1_1 2
pool1
conv2_1 2
pool2
conv3_1 2 3 4
pool3
conv4_1 2 3 4
pool4
conv5_1 2 3 4
A^L
G^L
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial E_L}{\partial F^L}$
$G_{ij}^L = \sum_k F_{ik}^L F_{jk}^L$
$\vec{x} =$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
F^{L-1}
F^L
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
$\vec{p} =$

The pseudo code for various sections of our algorithm implementation is as follows:

Algorithm 1: Computation of Content Loss

```
Loss_content(model)
  Input: A Tensorflow session and VGG-19 model
  Initialization:
  L ⟵ Layer to compute loss at
  n ⟵ Number of filters at layer l
  m ⟵ Number of pixel positions in a filter
  $F^l_{ij}$ ⟵ Activation of ith filter at position j on input image
  $P^l_{ij}$ ⟵ activation of ith filter at position j on white noise image
  for i = 1 to n do
      for j = 1 to m do
          $L_{content} \longleftarrow L_{content} + 0.5*(F^l_{ij} - P^l_{ij})^2$
  end for
  Return $L_{content}$
  Output: content loss at layer l
```

Algorithm 2: Computation of Style Loss

```
Loss_style(model)
  Input: A Tensorflow session and VGG-19 model
  Initialization:
  L ⟵ list of n Layers to compute loss at
  w ⟵ list of weights $w_l$ of each layer
  N ⟵ Number of filters at layer l
  M ⟵ Height*Width of filter at layer l
  $A^l_{ij}$ ⟵ Gram matrix of input image
  $G^l_{ij}$ ⟵ Gram matrix of generated image
  for i = 1 to n do
      for j = 1 to m do
          $E_l \longleftarrow E_l + \frac{1}{4N^2M^2}(G^l_{ij} - A^l_{ij})^2$
  end for
  for l = 1 to n do
      $L_{style} \longleftarrow L_{style} + w_l E_l$
  Return $L_{style}$
  Output: style loss of combination of layers
```

Algorithm 3: Combining content and style algorithm

```
  Input: One content image $I_c$ and one style image $I_s$
  Initialization:
      feed $I_c$, $I_s$ to VGG-19
      $L_{content}$ ⟵ Loss_content(model)
      $L_{style}$ ⟵ Loss_style(model)
      $L_{total} \longleftarrow \alpha * L_{content} + \beta * L_{style}$
for i = 1 to n do
      $L_{min} \longleftarrow minimize(L_{total})$
end
Output: Style transfer result image $I_o$
```

Algorithm 4: Combining content and multiple styles

```
  Input: One content image $I_c$ and vector of style image $\boldsymbol{I_s}$
  Initialization:
      feed $I_c$, $\boldsymbol{I_s}$ to VGG-19
      $L_{content}$ ⟵ Loss_content(model)
      for j = 1 to $\| \boldsymbol{I_s} \|$ do
          $L_{style\_j}$ ⟵ Loss_style_j(model)

      $L_{total} \longleftarrow \alpha * L_{content} + \Sigma_j (\beta_j * L_{style\_j})$
for i = 1 to n do
      $L_{min} \longleftarrow minimize(L_{total})$
end
Output: Style transfer result image $I_o$
```

5. Results

5.1. Project Results

We successfully reproduced the mix of images from the content and style images as shown in the original paper. The results are shown in Figure. The number above each column indicates the ratio α/β between the emphasis on matching the content of the photograph and the style of the artwork. We find that as α/β decreases, the contribution of the style part in the final image increases.

5.2. Comparison of Results

In this section, we discuss our results starting with a discussion of the runtime and loss improvement tradeoff, optimization of weights and then multiple style transfers. Further, we study a comparison with other methodologies.

5.2.1. Improvement over time

The training loss exhibited a sharp decrease in the initial 200 iterations across a range of different weight values, followed by eventual flattening (see figure 5A below). The decision of an 'optimal' aesthetic was difficult, however we believe that for a 600x800 pixel image, 2000 iterations is generally enough to optimize the style transfer. The runtime for 2000 iterations of our algorithm on a standard local machine (CPU) took around 4 hours, compared with 12 minutes on a GPU.

5.2.2. Multiple Styles

The style transfer method stayed robust to using multiple style images. We used both the *Starry Night* and '*Der Schrie'* as style images on the *Neckarfront*. The results (see figure 5B) were interesting and elements of both styles can be observed. For the images presented here we used the

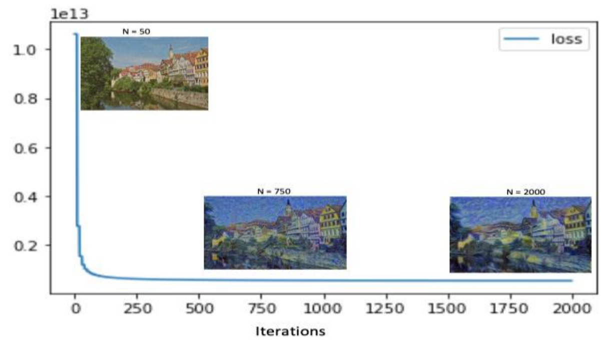


Figure 5A: Loss as a function of iterations. Style image used is Starry Night and content is Neckarfront.

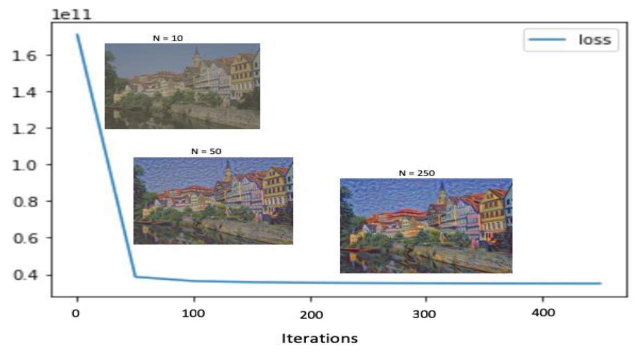


Figure 5B: Loss as a function of iterations. Style images used are Starry Night and Der Schrie, content used is Neckarfront

$\beta_1/\beta_2 = 2$ for the *Starry Night* and *Der Schrie* respectively, although that particular combination can be changed to reflect the desired combination of influence for the different styles. This ratio of the weights can be observed with the style from the *Starry Night* playing a significantly larger role in the resulting image.

5.2.3. Hyperparameter Tuning

The key hyperparameters to tune were the (i) alpha-beta ratio of content to style influence, (ii) the noise with which to initialize the result image, (iii) the normalizing constants, and (iv) the weights for the style layers.

(i) The original paper suggested using an alpha beta ratio of $\alpha/\beta = 10^{-3}$, and that is what we used as the default ratio in our models. We experimented by changing the weights to be more even. The image below (Figure 6) is for a α/β ratio of 5/100 after 750 iterations and the degradation in the quality of the aesthetic is apparent. Experimenting with some other ratios as well, we ultimately concur with the authors of the suggested ratio being between 10^{-3} and 10^{-4}.

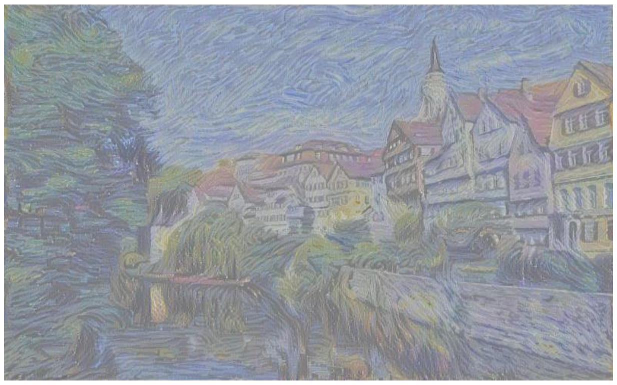


Figure 6: Image with α/β = .05 at iteration 750

(ii) We initially attempted to use a pure white noise when training, but after several thousand iterations continued to get poor results. We then substituted the white noise initialization with the content image augmented with noise and achieved much better results after the first 50-100 iterations.

(iii) The original paper uses 0.5 as the normalization constant in the formula for content loss. We find that it takes significantly more time for style transfer to start appearing if this formula is used (Figure 7 shows the result at iteration 1000). Hence we try using a normalization constant equivalent to the one used in style loss and find that it drastically improves the optimization process. Taking the dimensions of gram matrix in the style loss into account, we modify the formula for content loss as follows:

$L_{content} = \sum \frac{1}{4NM} * (F^l_{ij} - P^l_{ij})^2$

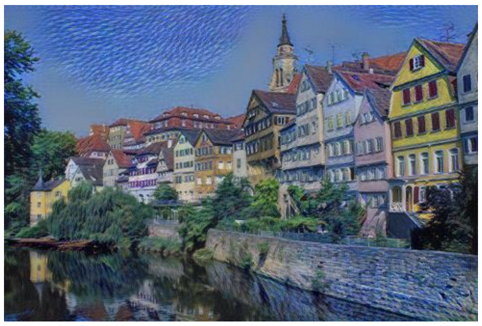


Figure 7: Image at Iteration 1000

(iv) We followed the authors suggestion of using identical weights for all the style layers, however instead of using 1/number of active layers, we used $w_l = 1.5$ for all layers. We found this to give the most aesthetically pleasing results in an acceptable number of iterations.

5.2.4. Comparison with other papers

The parametric method used in the Gatys et al. (2015) paper is often contrasted with non-parametric techniques such as those presented in Li and Wand[4] (2016) using Markov Random Fields on the higher level CCN layers. The non-parametric methods tend to use local patches in the minimization functions and are thus better able to preserve local features of style. The focus on preserving local features means there are no global constraints and the non-parametric methods suffer from wash-out artifacts[5].

An interesting new methodology presented in Gu et al. [3] (2018) which combines the parametric and non-parametric neural methods. They propose a method called the deep feature shuffle which minimizes an objective function with a similar content loss as the Gatys et al. (2015) but a replacement for style loss which join minimization utilizing both non-parametric nearest neighbor search and some global constraints. The plots below (Figure 5) from the Gu et al. (2018) paper provide a good comparison of the local-global tradeoff between parametric and non-parametric methods along with the overall more balanced performance of their model.

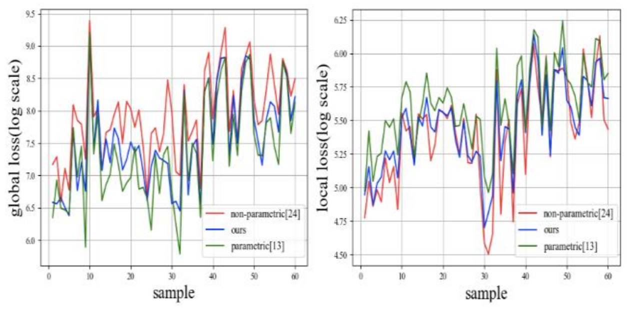


Figure 8. Comparison between parametric & non-parametric methods

6. Conclusion

Artistic style transfer is an interesting area of research within computer vision. The Gatys et al. (2015) paper was monumental in sparking the interest and has been added to by several other methodologies such as the deep feature shuffle proposed in Gu et al. (2018). In this paper we replicated the original results presented in the Gatys et al. (2015) paper using the VGG-19 CNN. We also explored several combinations of weights and noise initialization in an effort to get a more appealing aesthetic result. An extension to the model was using multiple style images rather than a single one and modifying the loss functions and weights appropriately.

One of the largest challenges that we faced was trying to optimize the tradeoff between additional runtime and improvements in the loss/aesthetics of the image. Given the lack of a 'test set' for validation and a monotonically decreasing (with iterations) loss function, we were forced to use an arbitrary threshold – around 2000 iterations - for when the change in loss was small enough for the training to have been 'optimized'.

7. References

[1] Bitbucket

[2] L. A. Gatys, A. S. Ecker, M. Bethge, A Neural Algorithm of Artistic Style, *arXiv:1508.06576v2*, 2016.

[3] S. Gu, C. Chen, J. Liao, L. Yuan, Arbitrary Style Transfer with Deep Feature Reshuffle, *arXiv:1805.04103v4*, 2018.

[4] C. Li and M. Wand, Combining markov random fields and convolutional neural networks for image synthesis.preprint *arXiv:1601.04589*, 2016.

[5] O. Jamriska, J. Fiser, P. Asente, J. Lu, E. Shechtman, and D. Sykora. Lazyfluids: Appearance transfer for fluid animations. *vACM Trans. Graph. (Proc. of SIGGRAPH), 34(4):92*, 2015.

8. Appendix

8.1. Individual student contributions in fractions - table

	ar3879	gs3006	am4797
Last Name	Rajput	Singh	Mishra
Fraction of (useful) total contribution	1/3	1/3	1/3
Contribution 1	Code – paper reproduction, comments, File Organization	Code - Debugging	Code – Debugging, Extensions
Contribution 2	Report - Results	Report - Abstract, Introduction, Implementation, Results, Flowcharts	Report – Summary of Original, Our methodology, Results, Conclusion