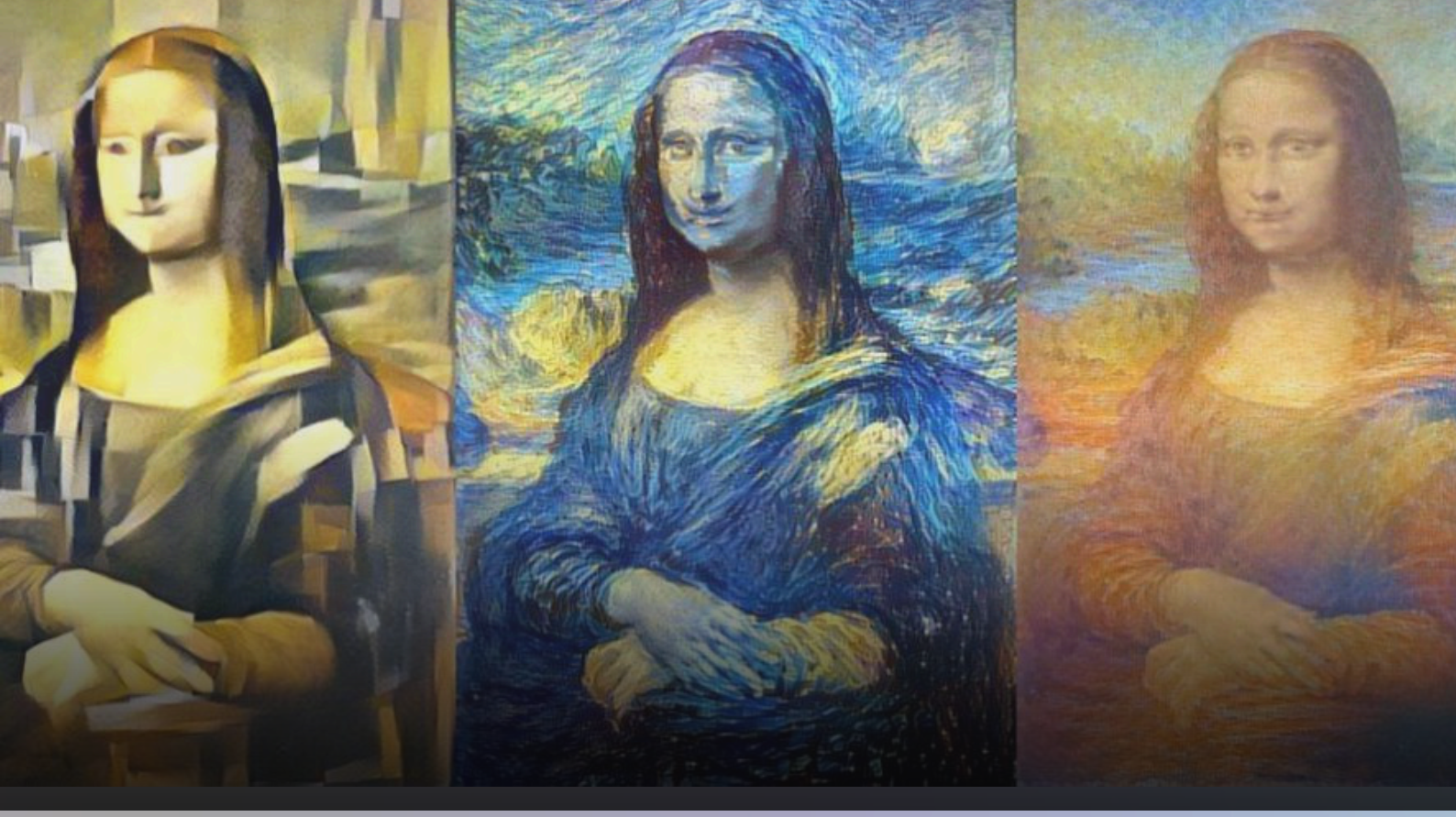


NEURAL STYLE TRANFSER

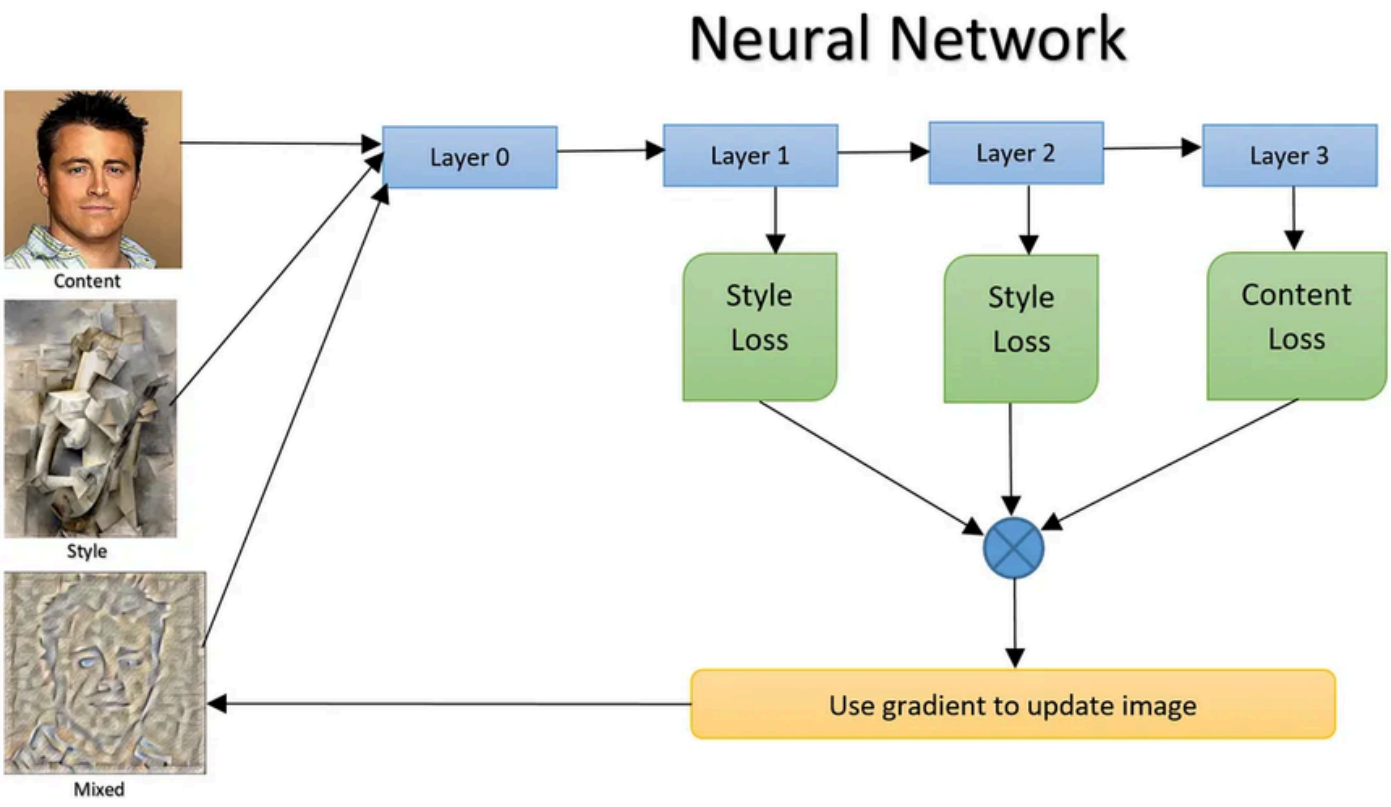
under supervision
Dr.shimaa yosry

BY: Hana Ahmed Nabhan

1.0 INTRODUCTION TO NEURAL STYLE TRANSFER

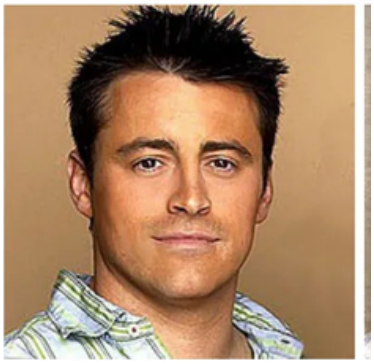
Neural Style Transfer (NST) is a very strong optimization method that has come to be the mainstay of generative arts and the most interesting application in the larger domain of computer vision. It is a systematic framework that allows for the artistic reimagining of an image by splitting and recombining the different parts of content and style from different sources. The strategic importance of NST is that it can show how deep learning models, originally trained for classification tasks, can be used for complex image synthesis and manipulation. This paper presents a detailed technical presentation of the NST workflow, starting from its architectural base and mathematical background to a comparison of optimization algorithms.

The NST workflow is essentially motivated by the interaction of three main image components:

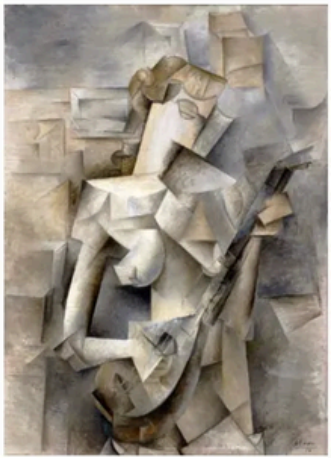


1.0 INTRODUCTION TO NEURAL STYLE TRANSFER

• Content Image: The content image is the primary image that delivers the fundamental structure, arrangement, and visible objects for the end product.



• Style Image: The style image acts as the artistic prototype, bringing in its own unique texture, color scheme, and brushstroke patterns.

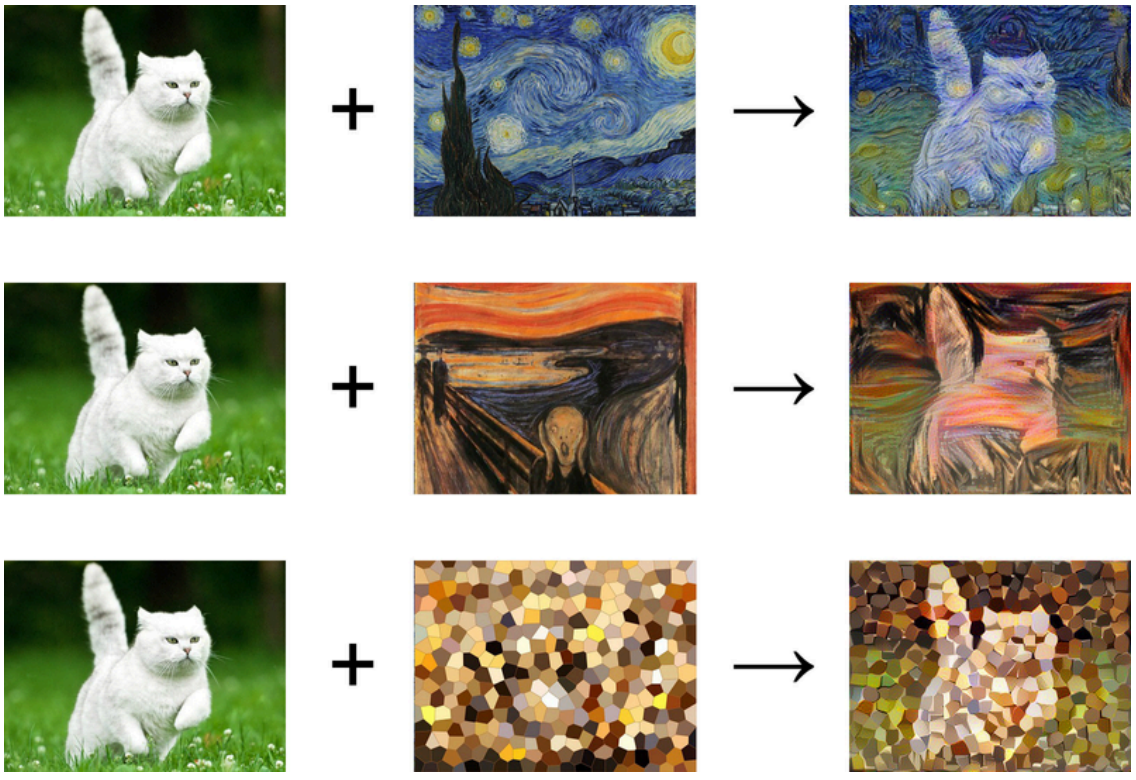


• Generated Image: The generated image represents the end product that is gradually optimized through iterations. Initially, it is either a duplicate of the content image or an imitation of random noise and is further changed to blend the content of the

1.0 INTRODUCTION TO NEURAL STYLE TRANSFER

first picture drawn in the manner of the second. The operational process of NST can be depicted in three basic steps at a high level:

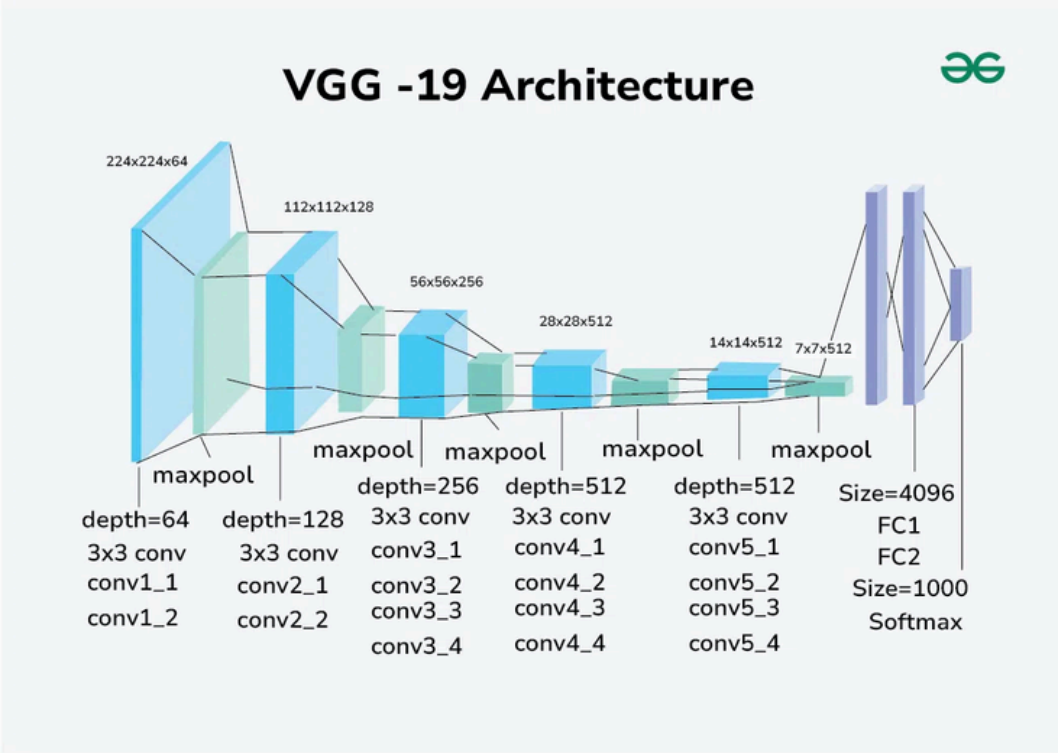
1. Content and style images are loaded and preprocessed to a standard format compatible with model input.
2. A pre-trained Convolutional Neural Network (CNN) like VGG19 is employed as a feature extractor and hierarchical representations of content and style from the corresponding images are captured.
3. The problem is posed as an optimization problem in which the produced picture is gradually modified to decrease a group of specified loss functions that quantify the dissimilarity in content and style.



1.0 INTRODUCTION TO NEURAL STYLE TRANSFER

The effectiveness of the whole procedure relies on the feature extraction proficiency of the CNN beneath. The specific function of the VGG19 network as the principal feature extractor in this execution will now be discussed by this paper.

2.0 ARCHITECTURAL FOUNDATION: VGG19 AS A FEATURE EXTRACTOR



Neural Style Transfer relies on a pre-trained Convolutional Neural Network (CNN) as its pivotal element. The strength of a network such as VGG19, which was initially designed for large-scale image classification, is in the deep and hierarchical feature approximations it captures. The first layers of the network are responsible for detecting low-level features like colors and basic textures, while the higher layers are able to identify more complex and abstract high-level features that correspond to intricate objects and shapes. This natural division of feature complexity is very important for the main purpose of NST which is to separate the "content" of one image from the "style" of another.

2.0 ARCHITECTURAL FOUNDATION: VGG19 AS A FEATURE EXTRACTOR

the VGG19 model in this implementation is loaded with weights that have been pre-trained on the ImageNet dataset. The model, however, is created devoid of its last classification layers (include_top=False), which means it has transformed from a classifier to an efficient feature extractor. This opens the way to the intermediate activations from its convolutional blocks being used as the definition of content and style.

```
vgg= VGG19(include_top=False, weights='imagenet')
```

It is a careful decision to pick certain layers to show content and style through the vertical arrangement of the network's learned features.

• **Content Representation:** The feature representation for the content image is taken from one of the deepest layers in the network: block5_conv2. The original VGG literature gives this layer a conceptual name of conv4_2. Content layers that are deeper are selected as they contain high-level structural information and the arrangement of objects in space, hence fine-grained textural details and colors are practically eliminated. This way, the optimization can keep the content image in a recognizable form.

2.0 ARCHITECTURAL FOUNDATION: VGG19 AS A FEATURE EXTRACTOR

- **Style Representation:** The style representation is obtained through the combination of layers from the top of the network to the bottom: block1_conv1, block2_conv1, block3_conv1, block4_conv1, and block5_conv1. These are equivalent to the original model's conv1_1 through conv5_1 layers. This technique of using multiple layers adds up to a very fine definition of style. The beginning layers take in the low-level aspects like colors and rudimentary textures, whereas the end layers take in the more intricate patterns and styles of brush strokes. All these layers together give a full and multi-scale view of the artistic features of the style image.

```
content_layer=['block5_conv2'] # 2nd layer inside 5th block
style_layer=[
    'block1_conv1',  # early layers capture simple textures
    'block2_conv1',
    'block3_conv1',
    'block4_conv1',
    'block5_conv1'   # very complex patterns
]
```

2.0 ARCHITECTURAL FOUNDATION: VGG19 AS A FEATURE EXTRACTOR

After successfully extracting the feature representations of content and style, the following step is to specify the mathematical formulas the loss functions that will assist the optimization algorithm in merging them together.

```
VGG19 feature extractor created!
Content layers: ['block5_conv2']
Style layers: ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1', 'block5_conv1']
```

3.0 THE MATHEMATICAL CORE: DECONSTRUCTING THE LOSS FUNCTIONS

Neural Style Transfer is a process that relies on a total loss function which is a weighted combination of three separate mathematical components. The main objective of the optimization algorithm is to change the pixels of the created image step by step until the total loss is minimized. All the components of the loss function operate together, each fulfilling a certain role imbuing the generated image with more or less content, style or visual consistency and thus, applying less or more penalties accordingly.

3.0 THE MATHEMATICAL CORE: DECONSTRUCTING THE LOSS FUNCTIONS

3.2.1 Content Loss

The role of the content loss function is to make certain that the produced image preserves the structure and the semantic meaning of the original content image. It receives the optimizer's penalty when an image deviates from the high-level characteristics of the original content. This is computed as the L2 distance or squared-error loss between the feature representations of the generated image and the content image. The comparison takes place at the network's one deep layer (block5_conv2) which is exclusively dedicated to high-level object representation.

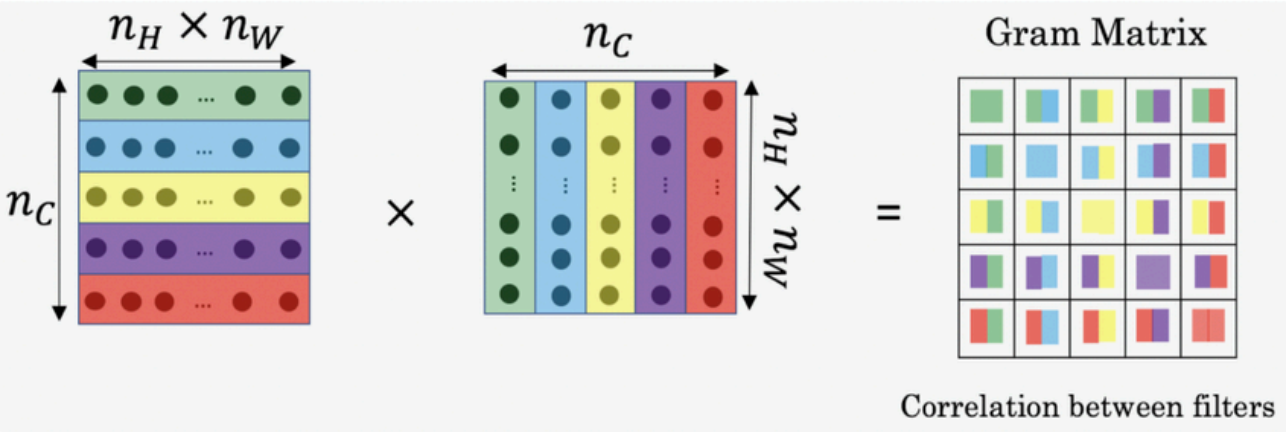
$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2$$

3.2.2 Style Loss and the Gram Matrix

In order to assess the difference in style, a less conventional and simpler method is to be used. The style loss function, which takes advantage of a key mathematical construct the Gram matrix, is the one that makes it possible.

To begin with, the Gram matrix is a very efficient way to get hold of the texture information. It indicates the degree of interaction between the outputs of different filters in one layer of the CNN. The process of the Gram matrix is based on finding the dot product of the activation vectors of different filters after they have been flattened, and it thus measures the tendency of features to occur together.

3.0 THE MATHEMATICAL CORE: DECONSTRUCTING THE LOSS FUNCTIONS



A straightforward analogy can be drawn from the source material that helps the understanding: "In the painting, for every 'Blue' I see, I also see 'Curved Lines'. Is that the case with your image too?" The Gram matrix embodies these sorts of feature co-occurrence connections which, in total, characterize the style of the image.

The Style Loss is computed by comparing the generated image's and the style image's textural patterns and determining the difference between them.

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

3.0 THE MATHEMATICAL CORE: DECONSTRUCTING THE LOSS FUNCTIONS

The computation involves adding the L2 distances of the Gram matrices of the two pictures, which are determined over the range of designated style layers. To make sure that each layer's contribution is just and is not based on the size of the layer, this loss is scaled down by the number of pixels present in the feature maps.

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

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

3.2.3 Total Variation Loss

Total Variation (TV) Loss is considered a regularization term. The smoothness and the appearance of spatial continuity between the neighboring pixels in the produced image are the main things that it is promoting. By imposing a penalty on the gradients between pixels, it practically acts like a de-noising agent. The main function of TV loss is to keep the pixelated results under control and at the same time maintain the high degree of visual coherence in the final image.

3.0 THE MATHEMATICAL CORE: DECONSTRUCTING THE LOSS FUNCTIONS

```python
def gram_matrix(input_tensor):
    result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor, input_tensor)
    #normalize (to be fair regradless how big or small the image layer)
    input_shape = tf.shape(input_tensor)
    num_locations = tf.cast(input_shape[1] * input_shape[2], tf.float32)
    return result / num_locations

def compute_content_loss(base_content, target):
    return tf.reduce_mean(tf.square(base_content - target))

def compute_style_loss(base_style, gram_target):
    gram_style = gram_matrix(base_style)
    return tf.reduce_mean(tf.square(gram_style - gram_target))

def total_variation_loss(image):
    # gradients
    x_deltas = image[:, :, 1:, :] - image[:, :, :-1, :]
    y_deltas = image[:, 1:, :, :] - image[:, :-1, :, :]
    return tf.reduce_mean(tf.abs(x_deltas)) + tf.reduce_mean(tf.abs(y_deltas))
```

3.3 The Total Loss Function

The optimizer's primary goal is to reduce the total loss, which is a weighted combination of three components stated before. The weights given to each component specifically decide the extent of content preservation, style adoption, and conversing

3.0 THE MATHEMATICAL CORE: DECONSTRUCTING THE LOSS FUNCTIONS

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

image smoothness. In the Adam optimizer's implementation, the weights given below are a clear indication of this balance:

- content_weight: 1000.0
- style_weight: 0.1
- total_variation_weight: 30

Theoretical loss functions were defined, and the last step involved the application of a practical optimization algorithm to iteratively minimize the total loss and create the final image.

```
content_weight = 1000.0
style_weight = 0.1
total_variation_weight = 30
```

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

Optimization is the iterative engine that fuels Neural Style Transfer. It systematically adjusts the image's pixels until the total loss function defined in the previous section is gradually decreased to an acceptable level. This section compares the two different optimization algorithms: the first-order Adam optimizer and the quasi-Newton L-BFGS-B optimizer, both used for this purpose.

4.2.1 The Adam Optimizer

The Adam optimizer implementation bases itself on the TensorFlow's tf.GradientTape for the automatic computation of the gradients of the total loss concerning the pixels of the generated image. Gradients are produced during each training step and then applied by the optimizer for the image update. This cycle continues for a given number of epochs until the loss reaches a minimum value.

1. Momentum

Momentum is used to accelerate the gradient descent process by incorporating an exponentially weighted moving average of past gradients. This helps smooth out the trajectory of the optimizatior allowing the algorithm to converge faster by reducing oscillations.

The update rule with momentum is:

$$w_{t+1} = w_t - \alpha m_t$$

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

2. RMSprop (Root Mean Square Propagation)

RMSprop is an adaptive learning rate method that improves upon AdaGrad. While AdaGrad accumulates squared gradients and RMSprop uses an exponentially weighted moving average of squared gradients, which helps overcome the problem of diminishing learning rates.

The update rule for RMSprop is:

$$w_{t+1} = w_t - \frac{\alpha_t}{\sqrt{v_t+\epsilon}} \frac{\partial L}{\partial w_t}$$

During a sample training run that spanned over 1500 epochs, the Adam optimizer showed continuous loss minimization as it steadily reduced the loss. The initial total loss amount to 3,683,107,584.00 and the final total loss was 95,819,664.00. A total of about 139.72 seconds was required for the entire optimization process, thereby establishing a performance baseline for this method.

```
Epoch 1400
Total loss: 105402040.00
Content loss: 1601512.75
Style loss: 103800024.00
TV loss: 503.43
Time elapsed: 130.52 seconds
---
Epoch 1500
Total loss: 95819664.00
Content loss: 1608851.38
Style loss: 94210304.00
TV loss: 509.19
Time elapsed: 139.72 seconds
---
```

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

Result of adam

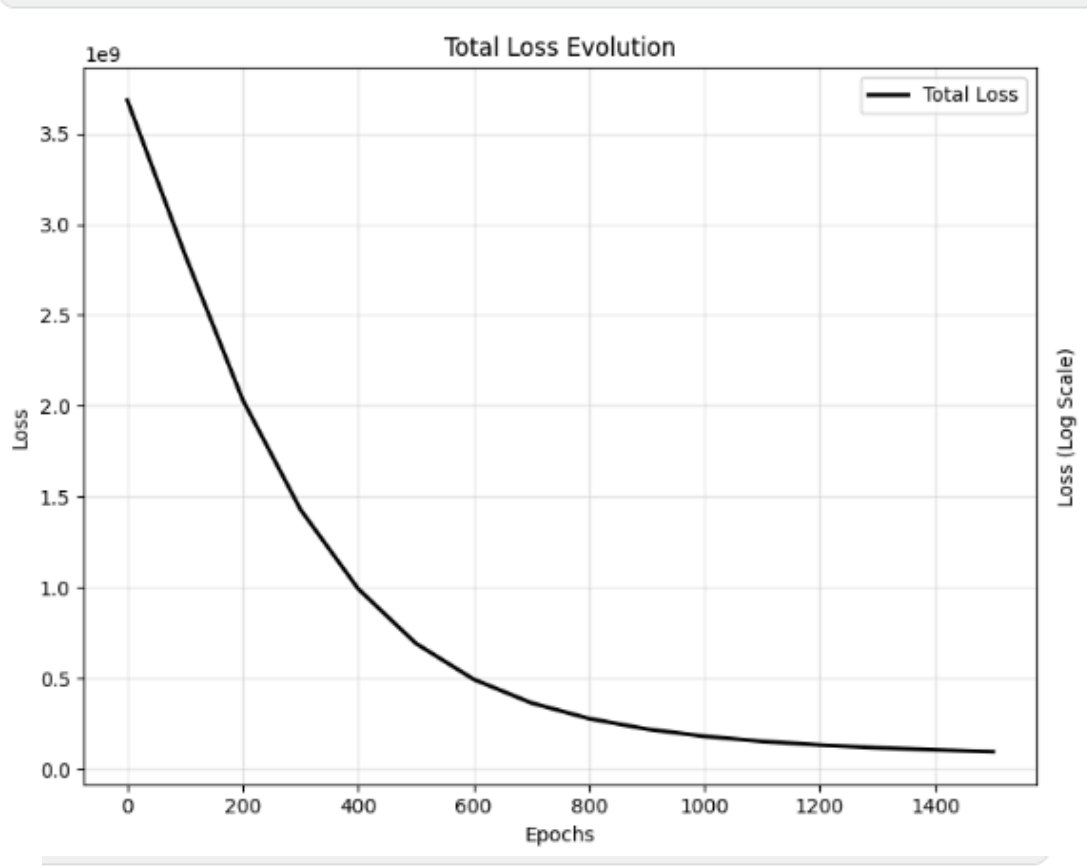
Content Image

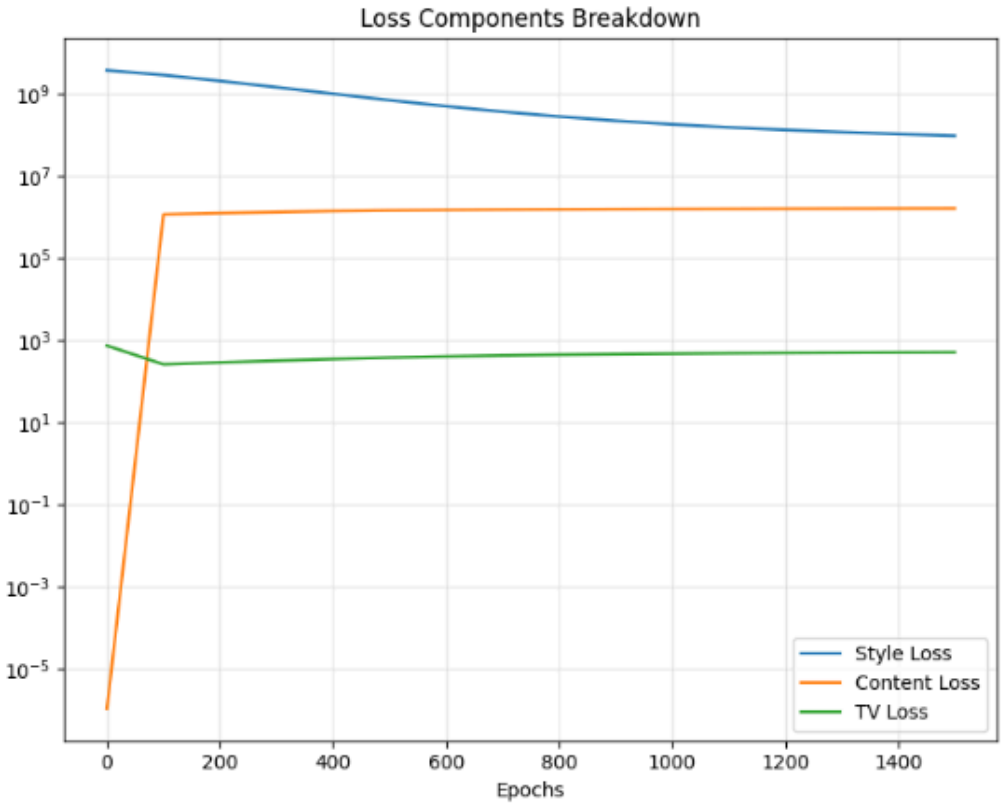
Style Image

Generated Image

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

Result of adam





4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

4.2.2 The L-BFGS-B Optimizer

The L-BFGS-B algorithm is a quasihybrid Newton method that is memory limited. It is said that "in general, it is faster, and the images produced are sharper because it applies more sophisticated mathematics (it estimates curvature, and not just slope)." This means that L-BFGS-B, unlike first-order methods like Adam, which only take into account the gradient (slope), can make use of second-order information (curvature), and hence, is able to find a more direct path to the loss minimum.

On the other hand, the use of L-BFGS-B in deep learning is a challenging task. A straightforward implementation may turn out to be very slow in terms of computing resources. This is because an optimizer like fmin_l_bfgs_b may try to get the loss value and the gradients in different function calls, thus causing a situation where there are two redundant passing of the model computations for each step. To get rid of this inefficiency, the Evaluator class is brought in. This class does a single computation pass to calculate loss and gradients at the same time, and it caches the results so that they can be served to the optimizer when needed without doing the computation again. This kind of efficient implementation turns out to make L-BFGS-B a very viable option for use.

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

The power of this method was seen clearly through the experiment runs. To illustrate, an optimization of 50 iterations was done in only 21.50 seconds. This is a big difference compared to the 139.72 seconds that were needed for 1500 epochs with Adam, thus confirming the assertion that L-BFGS-B is the best choice in terms of speed of convergence in the first place. It is supposed that the loss weights were one of the hyperparameters that were tuned along with the optimizer. While the Adam run employed the content_weight of 1000.0, the L-BFGS-B trials experimented with various proportions like content_weight = 0.5 and style_weight = 1.0 to realize the wanted artistic effect.

```
iteration = [0]
def callback(x):
    loss, _ = compute_loss_and_grads(x) # only for testing , because it doubles the runtime
    print(f'Iteration {iteration[0]}, Loss: {loss:.2e}')
    iteration[0] += 1

print('Starting optimization with L-BFGS-B')
start_time = time.time()

x_opt, loss_value, info = fmin_l_bfgs_b(
    compute_loss_and_grads,
    x,
    maxiter=50,#try to improve the image x times.
    maxfun=70,#don't call the loss function more than x+n times.
    disp=True,
    callback=callback
)
```

```
style_weight = 1.0
content_weight = 0.5
total_variation_weight = 30
```

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

Result of LBFGS

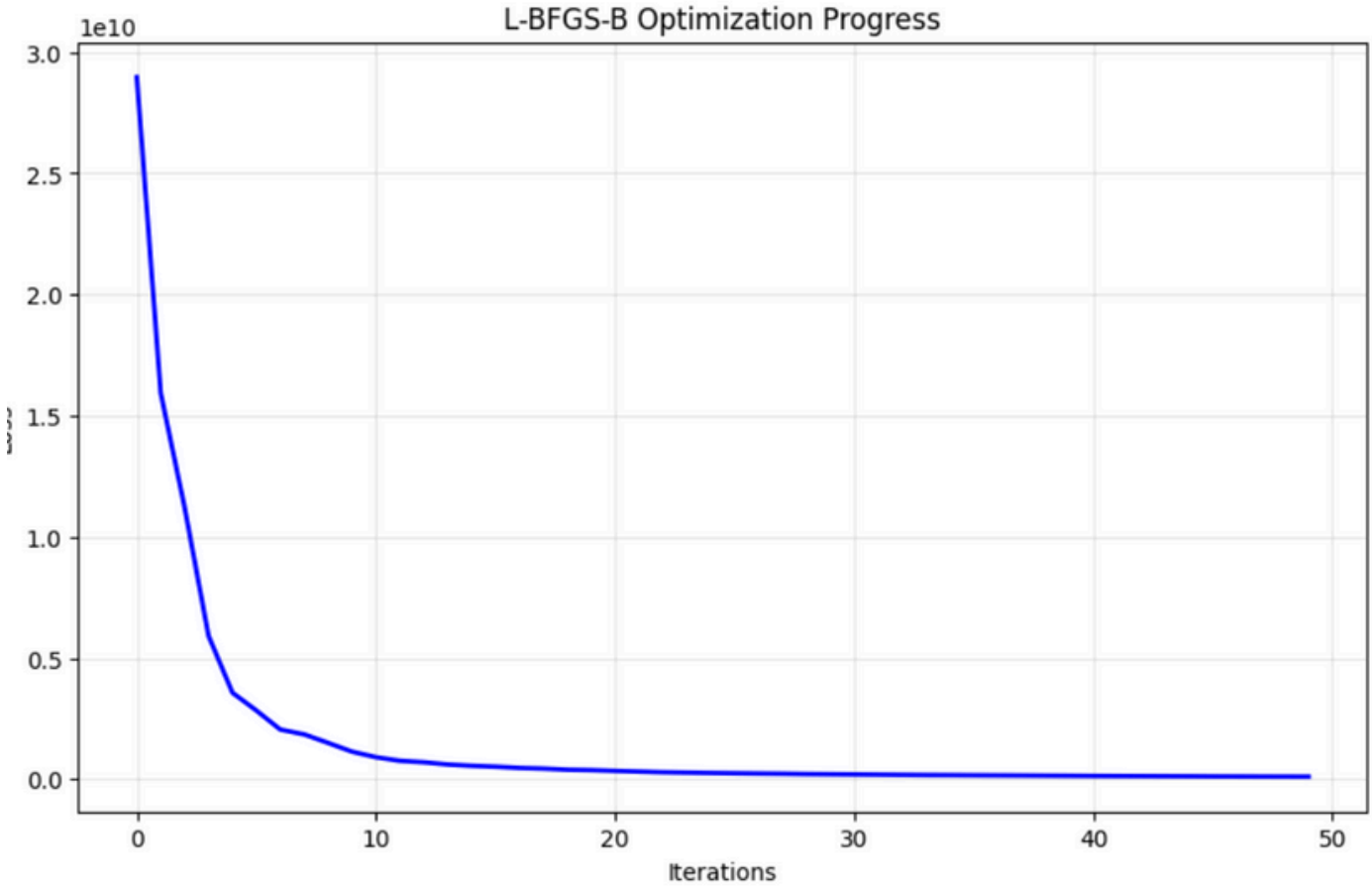
Content Image

Style Image

Generated Image

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

Result of LBFGS



4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

Result of Evaluator

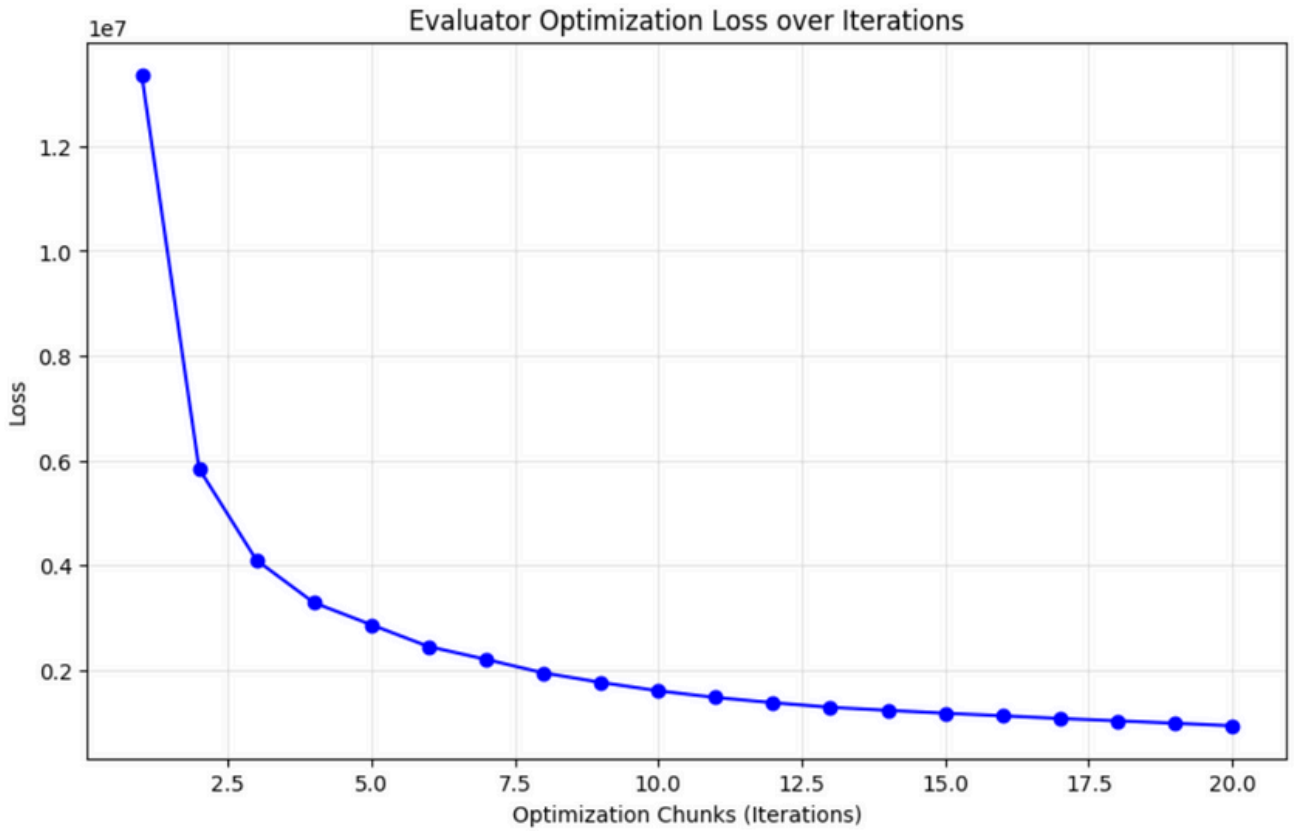
Content Image

Style Image

Generated Image

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

Result of Evaluator



4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

4.3 Comparative Analysis

The distinct characteristics of the Adam and L-BFGS-B optimizers, based on the efficient implementation of the latter, are summarized in the table below.

Feature	Adam Optimizer	L-BFGS-B Optimizer
Optimization Method	First-order gradient-based (slope)	Second-order quasi-Newton method (estimates curvature)
Reported Quality	Standard	Produces "sharper images"
Computational Cost	Lighter cost per iteration.	"Heavier" computational cost per step, but finds a more direct path to the solution.
Convergence Speed	Requires a high number of epochs (e.g., 1500).	Converges in far fewer iterations (e.g., 20-50), resulting in a much faster total optimization time.

4.0 THE OPTIMIZATION PROCESS: A COMPARATIVE ANALYSIS

4.3 Comparative Analysis

```
 LBFGS optimization completed in 21.50 seconds!
Displaying results...
```


```
 EVALUATOR optimization completed in 144.19 seconds!
Displaying results...
```

5.0 CONCLUSION

The present study has elucidated the technical methodology of the Neural Style Transfer technique, which is very effective in combining the artistic style and the structural content of the image. The process is based on the use of the pre-trained VGG19 network as a hierarchical feature extractor coupled with framing the image generation task as an optimization problem. By gradually refining a generated image through the minimization of a composite loss function that consists of content, style, and total variation components, the algorithm finally comes up with the desired artistic fusion.

The key findings resulting from the optimization algorithm comparative analysis emphasize an important trade-off in the choice of the implementation. The Adam optimizer, which combines simplicity with lower computational cost per step, requires many iterations for accurate convergence

5.0 CONCLUSION

On the other hand, the L-BFGS-B optimizer, a more complicated quasi-Newton method, manages to reach convergence in much fewer iterations while yielding qualitatively sharper images. Thus, if implemented correctly, this leads to a considerably quicker overall process, even though its computational cost per step is higher.

In the end, Neural Style Transfer emerges as a potent proof of the varying applications of the deep learning models. It illustrates the idea of using discriminative architectures for the creative and generative processes and vice versa, thus making the abstract ideas of "content" and "style" more evident through a solid mathematical and algorithmic framework.

6.0 REFERENCES

https://wandb.ai/wandb_fc/articles/reports/Exploring-Neural-Style-Transfer-with-Weights-Biases--Vmlldzo1NDM1MjA2

https://www.geeksforgeeks.org/deep-learning/adam-optimizer/

https://medium.com/data-science/neural-style-transfer-using-vgg-model-ff0f9757aafc

https://www.geeksforgeeks.org/computer-vision/vgg-net-architecture-explained/

https://d2l.ai/chapter_computer-vision/neural-style.html