Project Report On

# Image Style Transfer Using Neural Networks

## Chapter 1: Introduction

Style transfer is an area of computer vision that refers to the conversion of visual style of an image or video without changing its content. This technique allows us to combine artistic style from one image with another image's content, resulting in images or videos with visually appealing compositions.

This concept has gained popularity because of its wide-ranging applications across fields like photography, cinematography, and digital art. It is frequently used in camera filters on various social media platforms like Snapchat and Instagram. The style can be instantly transferred in real-time within milliseconds.

Through the advancements in image processing and deep learning, automated style transfer algorithms have emerged. Traditional signal processing methods converted the image data through filters and mathematical operations. However it lacked the ability to replicate complex artistic styles.

In contrast, deep learning models like CNN can analyse vast image datasets to automatically learn diverse artistic styles. It is revolutionising how we transform styles by making the results more realistic. This utilisation of neural networks in style transfer tasks is commonly known as Neural Style Transfer or NST in short.

In this project, we have implemented a NST technique that utilises a pre-trained VGG neural network model for feature extraction. Then we define content, style, and total variation loss functions to optimise a generated image, resulting in visually appealing compositions by combining content and style images.

# Chapter 2: Problem Statement

The problem statement of the project is to implement NST using a pre-trained VGG neural network model to combine the content image with the artistic style of another image.

This involves defining appropriate loss functions, including style, content, and total variation losses, and iteratively optimising a generated image to minimise these losses (Specifically using different optimization algorithms like Adam, RMSprop and Adagrad) to achieve a balance between retaining the high-level content attributes of the content image and integrating the low-level stylistic traits of the style image.

The process involves a forward pass through the VGG model to extract features, followed by a backward pass to compute gradients and update the generated image.

Here are some of the questions that we aimed to find out through this project:

1. How do different optimization algorithms (Adam, Adagrad, Rmsprop) perform in the context of image style transfer, particularly in terms of convergence speed and quality of output?
2. What are the effects of varying learning rates and epochs on the performance of optimization algorithms in image style transfer?
3. How can feature maps and filter maps from pre-trained convolutional neural networks (VGG) be effectively visualised to aid in image style transfer tasks?
4. How can loss functions be optimised to better capture the difference between content and style images in style transfer tasks?

# Chapter 3: Proposed System / implementation

In our project, we've combined deep learning methods, particularly NST, with optimization techniques to develop a system that can produce artistic compositions. These compositions mix the style of one image with the content of another. We've used a pre-trained neural network to extract features from both images in the form of feature maps. Then, we've created appropriate loss functions to quantify the differences in content and style representations. To improve the resulting image and reduce the losses, we've applied optimization algorithms iteratively with varying number of epochs and learning rates to find the most suitable algorithm and hyperparameters.

## 3.1. Feature extraction using pretrained model (VGG19):

Our approach relies heavily on a pre-trained model, particularly VGG19. By making use of this pre-trained network, we take advantage of its capability to effectively capture minute patterns and semantic details from images. VGG19 is a convolutional neural network (CNN) architecture comprising 19 layers. It was developed by the Visual Geometry Group at the University of Oxford. It is trained on a vast dataset known as ImageNet which has around 1000 different target classes. Thus, this network has learned to identify a diverse range of visual patterns and objects. We have used pre-trained ImageNet weights from the VGG19 CNN model to generate feature maps of our input images.

The model architecture comprises a series of convolutional layers followed by max-pooling layers. The convolutional layers are crucial in extracting features from the input image, while the max-pooling layers are used to downsample the feature maps. These convolution layers play a very important role in image based applications by extracting features that capture various spatial patterns and structures. VGG uses 3x3 convolution filters and 2x2 pooling filters. VGG19, specifically, consists of 16 convolutional layers. Each convolution layer is followed by a rectified linear activation function or ReLU for introducing non-linearity. These convolutional layers are organised into several blocks, with each containing multiple convolutional layers.

Following the convolutional layers, VGG19 consists of three fully connected layers, succeeded by a softmax layer for classification, particularly in image recognition tasks. These fully connected layers merge the features extracted by the convolutional layers to generate predictions regarding the input image.

The architecture of VGG19 is depicted in figure 2 which highlights different dimensions of the layers and layer names.

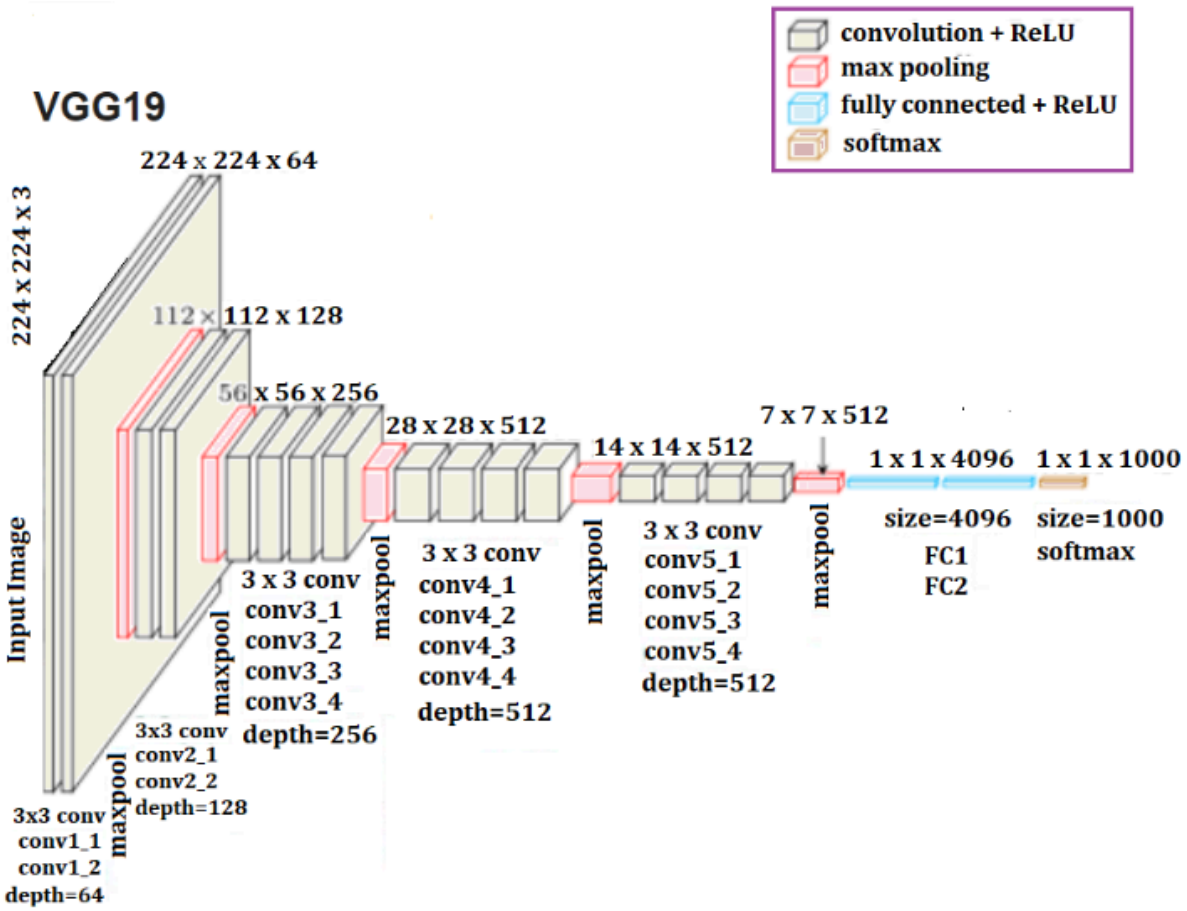



Figure 2: VGG-19 architecture Source: [7]

One important thing to note is that, In NST, the fully connected layers (last 2 layers along with softmax) are removed, and only the convolutional layers are utilised for feature extraction. By utilising VGG, NST can extract high-level content and style features from images. The content features show what the content image looks like, while the style features show the textures, colours, and patterns of the artistic style in the style image. Figure 3 represents a basic image transfer style network.

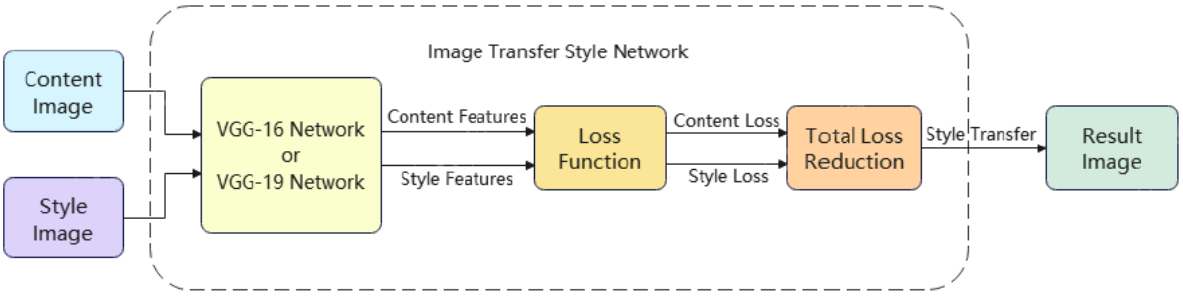



Figure 3: Image transfer style network Source: [4]

In the VGG19 network, earlier in the network is fine-tuned tight details like horizontal lines, vertical lines, diagonal lines, brightness, etc. and as we move through the network, we build up larger and larger sorts of features and also more abstract.

Because deeper features are more effective, the "block5_conv2" or "conv5_2" convolutional layer is selected to extract content information. On the other hand, for the style image, multiple layers from different depths of the neural network are often utilised to capture a comprehensive representation of style features. This approach allows the NST algorithm to extract style information ranging from low-level textures to high-level patterns present in the style image. We have used the combination of "block1_conv1," "block2_conv1," "block3_conv1," "block4_conv1," and "block5_conv1," to capture various aspects of style.

For example the following figure 5 & 4 show the 64 feature maps of the first layer along with the filters of the first layer.

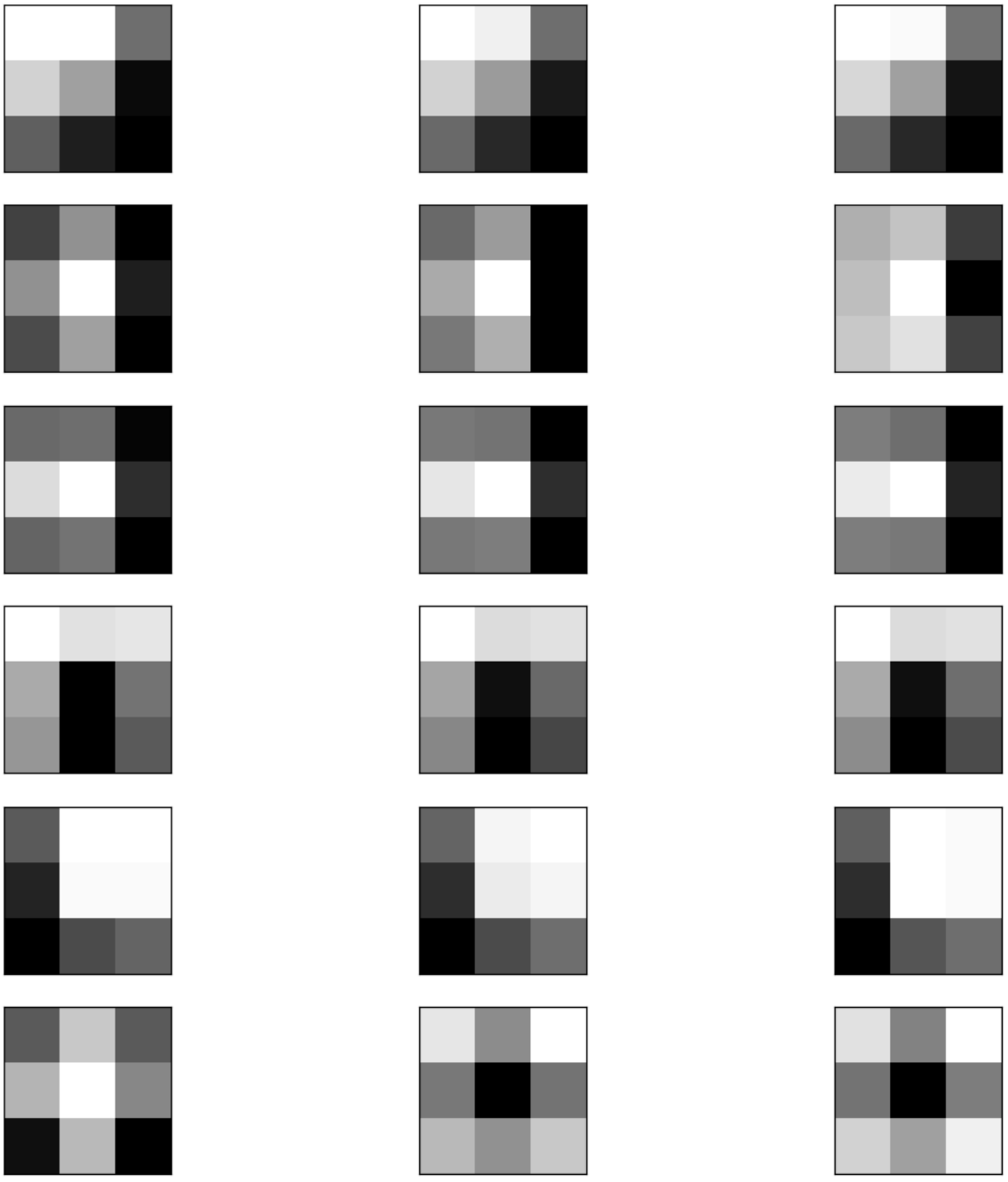


Figure 4: Filters of the first layers

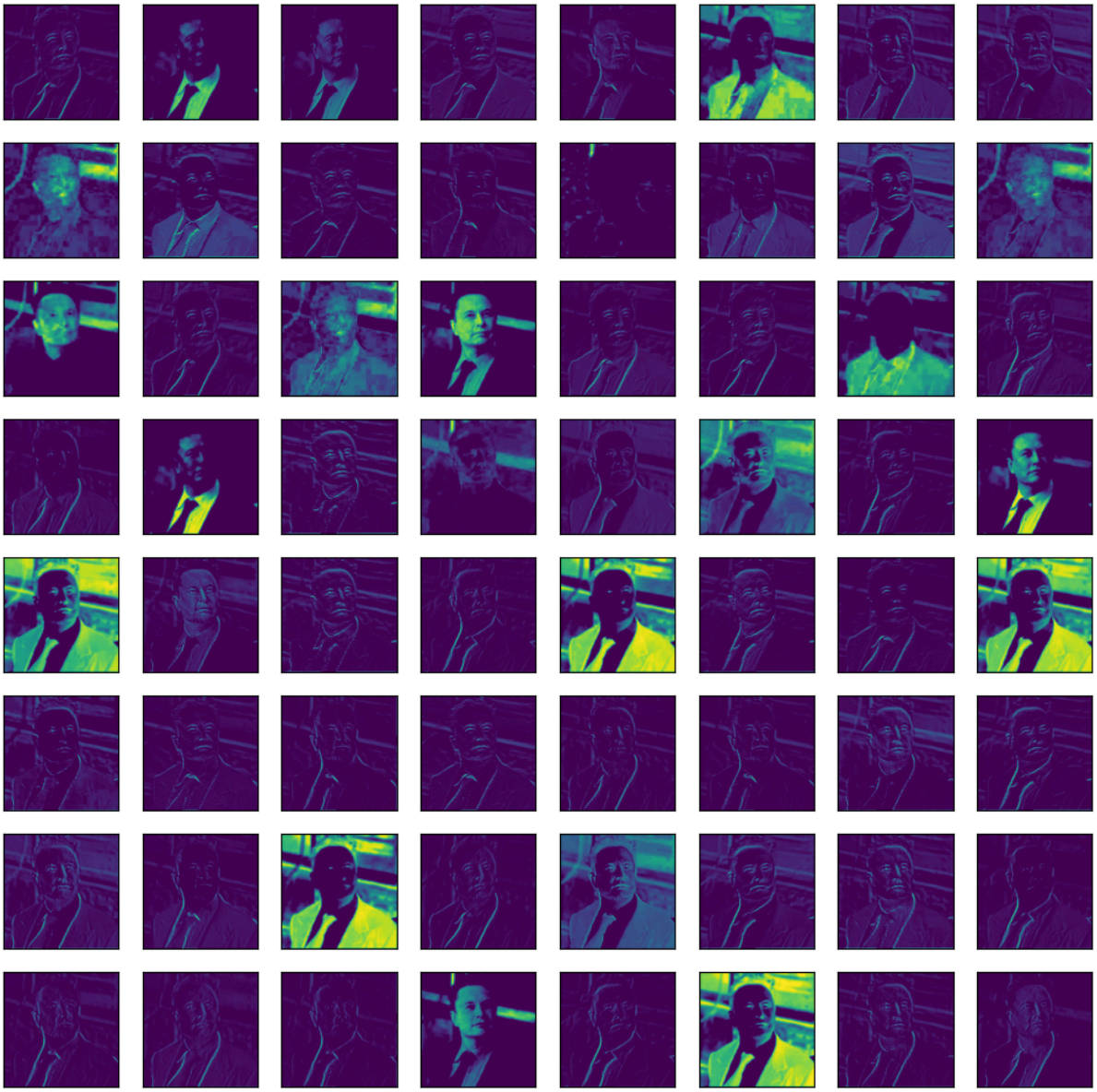


Figure 5: 64 feature maps of the first hidden layer for an image containing a portrait

## 3.2. Loss functions and Gram matrix:

We can measure the differences in content and style between images by establishing appropriate loss functions derived from the representations mentioned above. Two main loss functions are defined: content loss and style loss.

**Important note:** Before moving forward consider the following notations - Let $a$ represent the style image, $x$ represent the generated image, weights are represented by α and β each corresponding to content and style image losses. $l$ denotes the layer $l$ of the VGG19 network. $F_{ij}^{l}$ and $P_{ij}^{l}$ denotes the style image activation value $j$ and content image at the feature map $i$.

**Content loss:** The content loss is computed by comparing the feature maps of the combined image with those of the content image. This loss encourages the generated to retain the image

structure and content of the content image. It is typically computed as the mean squared error (MSE) between feature maps extracted from the convolutional layers of a pre-trained neural network. $N_l$ represents the $l$ later $N_l$ feature mapping where each size is $M_l$. $G_{ij}^l$ and $A_{ij}^l$ denotes the representation of style in style image and content image.

$$L_{content}(p,\ x,\ l)\ =\ \frac{1}{2}\sum_{ij}(F_{ij}^l - P_{ij}^l)^2$$

**Gram matrix:** To compute the style representation, we calculate the Gram matrix for each layer of interest. It is used to capture the style information of an image by examining the correlations between different features. It is calculated by computing the inner product of the feature maps generated by a particular layer and then averaging over all spatial locations. The inner product $G_{ij}^l$ is between the vector feature map denoted as $i$ and $j$ in $l$ layer. Then the style of the gram matrix is defined as

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

**Style loss:** To measure the difference in style between the style image and the generated image, we use the MSE between the Gram matrices of the feature maps for selected layers in the VGG19 network.

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

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

**Total loss:** The total loss used in NST is a weighted addition of the style and the content loss. The weights for each loss term are typically chosen empirically to balance the contributions of content and style to the generated image.

$$L_{total}(p,\ a,\ x)\ =\ \alpha\, L_{content}(p,\ x)\ +\ \beta\, L_{style}(a,\ x)$$

In this equation, weighted alpha and beta are coefficients used to balance the contributions of the style and the content loss to the overall loss function. These coefficients decide how much importance to give to maintaining the content of the content image and adding the style of the style image. These coefficients can be fine tuned to get the right mix between keeping the content and transferring the style. The values of alpha and beta depend on what we want the final image to look like. For our project, we've chosen the style weight to be 1e-2 and the content weight to be 1e4.

## 3.3. Optimizers:

To improve the generated image iteratively and reduce the losses discussed above, we've used optimization algorithms like Adam, Adagrad, and RMS-prop. These algorithms help in adjusting the parameters of the generated image in a way that decreases the overall loss. The goal is to move towards a final result that combines the style and content of the input images in a visually pleasing way.

**Updation algorithm in NST:**

1. Let $I_{comb}$ be the combination image
2. Initialize $I_{comb}$ randomly
3. For $e$ in range (epochs)
    a. Calculate $Loss(I_{comb})$
    b. Calculate $\frac{\partial Loss}{\partial I_{comb}}$
    c. $I_{comb} -= \lambda \frac{\partial Loss}{\partial I_{comb}}$ (General update rule e.g. SGD)

**Adam Optimizer:**

Adam, short for Adaptive Moment Estimation, is an adaptive optimization technique that combines the advantages of AdaGrad and RMSprop optimizers. It maintains two moving averages of gradients: the second moment (uncentered variance) and the first moment (mean). Using these estimates, the algorithm updates parameters while adjusting the learning rate for each parameter independently. The update formula for Adam optimizer can be expressed as follows:

Here $m_t$ and $v_t$ are the first and second moment estimates of the gradients. $g_t$ is gradient at time step $t$. $\beta_1$ and $\beta_2$ are decay rates for the estimates of moments. $\eta$ is the learning rate $\varepsilon$ is a constant. It is very small and is used to prevent division by zero.

- $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
- $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
- $\hat{m} = \frac{m}{1-\beta_1^t}$
- $\hat{v} = \frac{v}{1-\beta_2^t}$
- $\theta_{t+1} = \theta_t - \frac{\eta \cdot \hat{m}}{\sqrt{\hat{v}}+\epsilon}$

**RMSprop Optimizer:**

RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimization technique designed to beat the problem of diminishing learning rates seen in AdaGrad. It works by maintaining a moving average of squared gradients and adjusting learning rates for each parameter independently. The update formula for RMSprop optimizer is as follows:

Here $v_t$ is the exponentially decaying average of squared gradients. β is the decay rate. η is the learning rate. ε is a constant to avoid division by zero.

- $v_t = \beta \cdot v_{t-1} + (1 - \beta) \cdot g_t^2$
- $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t$

**AdaGrad Optimizer:**

AdaGrad (Adaptive Gradient Algorithm) is an adaptive learning rate optimization method. It is designed to give more weight to infrequent features by adapting the learning rate for each parameter using historical gradients. AdaGrad accumulates squared gradients over time. This is used to adjust the learning rate for each parameter independently. Over time, the denominator of the learning rate update can become too large which can cause the learning rate to become too small. The update rule for AdaGrad optimizer is as follows:

Here $g_t$ is the gradient at time step $t$. η is the learning rate. ε is a constant to avoid division by zero. $G_{t+1}$ is the accumulated squared gradient.

- $G_{t+1} = G_t + g_t^2$
- $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_{t+1} + \epsilon}} \cdot g_t$

## 3.4. Final algorithm:

Considering the steps mentioned earlier, we can outline the algorithm for Neural Style Transfer as follows:

Start

1. Load Content Image
2. Load Style Image
3. Preprocess Images
4. Load Pretrained VGG Model
5. Define Content and Style Layers
6. Extract Content and Style Features
7. Initialize Generated Image
8. Define Loss Functions

Loop until convergence or desired number of iterations:

9. Forward Pass:
   - 9.1 Compute Content Loss:
     - 9.1.1 Extract Content Features of Generated Image
     - 9.1.2 Compute Mean Squared Error (MSE) between Content Features of Content and Generated Image
   - 9.2 Compute Style Loss:
     - 9.2.1 Extract Style Features of Generated Image
     - 9.2.2 Compute Gram Matrix of Style Features for Content and Style Images

9.2.3 Compute Mean Squared Error (MSE) between Style Features Gram Matrices
9.3 Compute Total Variation Loss:
9.3.1 Compute Image Gradient along x and y directions
9.3.2 Compute Total Variation Loss using Image Gradients
9.4 Compute Total Loss:
9.4.1 Combine Content, Style, and Total Variation Losses with respective weights

10. Backward Pass:
10.1 Compute Gradient of Total Loss with respect to Generated Image Pixels

11. Update Generated Image using Optimizer

12. Record Loss Values

13. Post-process Generated Image
14. Save Generated Image
15. Save Loss Values
16. Visualize Loss Curve
17. Visualize Content, Style, and Generated Images

End

## Chapter 4: Evaluation of results

We experiment with three optimizers: Adam, RMSprop, and AdaGrad, each offering unique advantages in terms of convergence speed and stability. We evaluate the performance of our system using both quantitative metrics, such as loss values, and qualitative assessments based on visual inspection of the generated images. Additionally, we fine-tune hyperparameters, including learning rates and the number of optimization iterations, to achieve optimal results. Fine-tuning involves experimenting with different parameter settings and evaluating their impact on the quality of the generated images.

The NST algorithm is provided with two input images: a content image featuring a dog and a style image depicting the renowned painting 'Starry Night' by artist Van Gogh. Both images are of identical size as seen in figure 6.

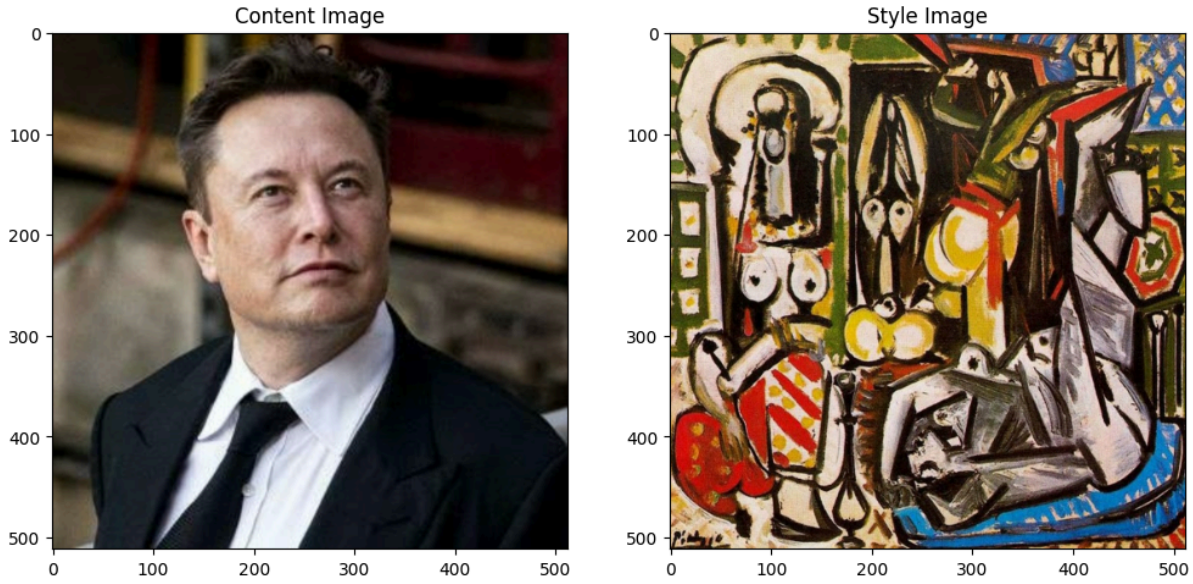
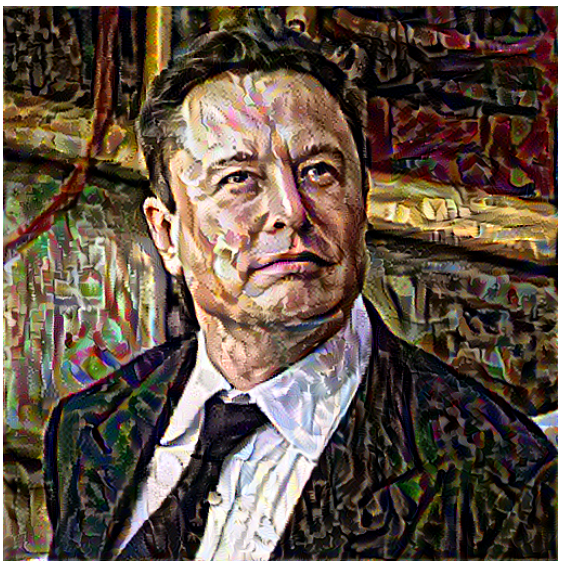
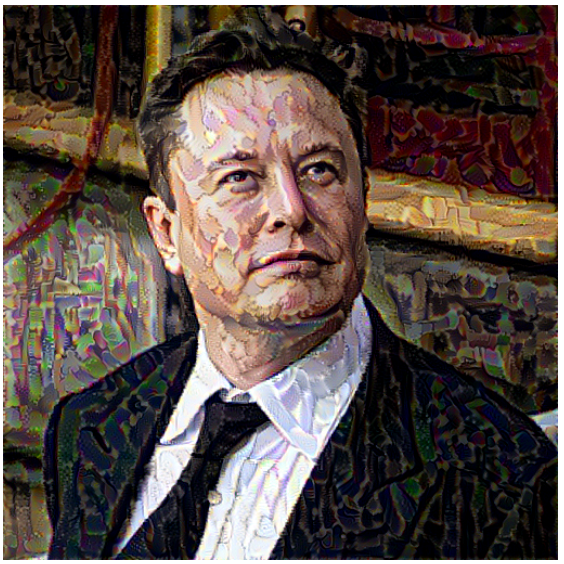
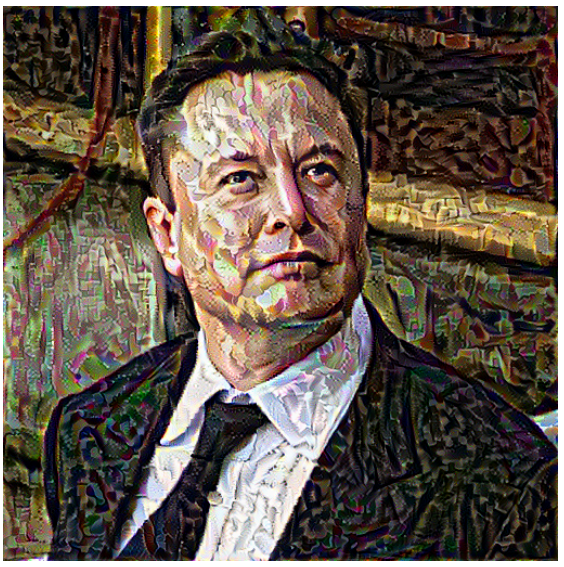
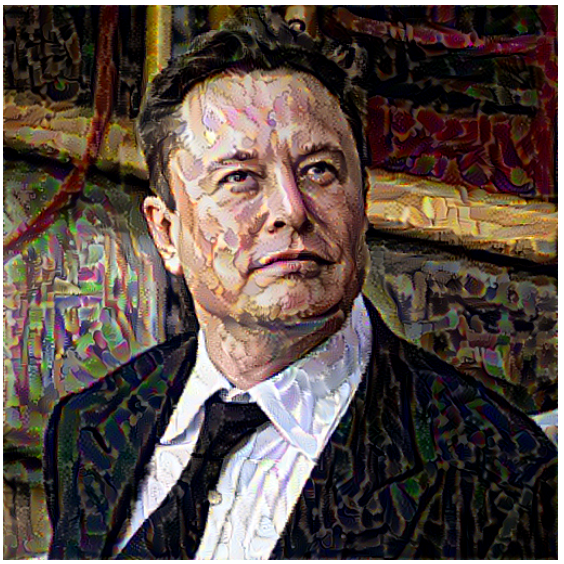
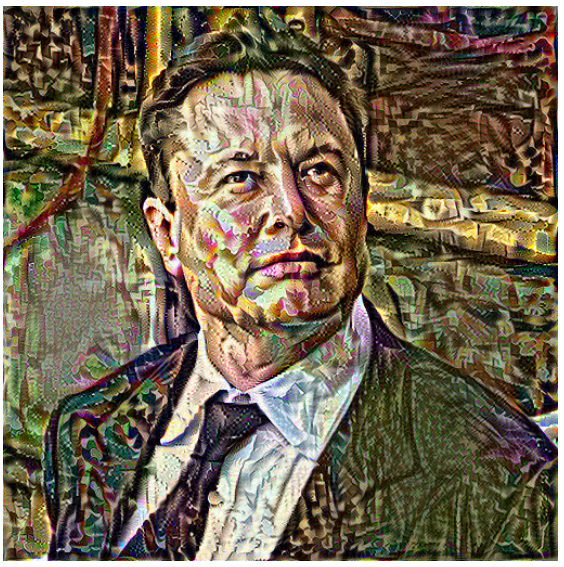
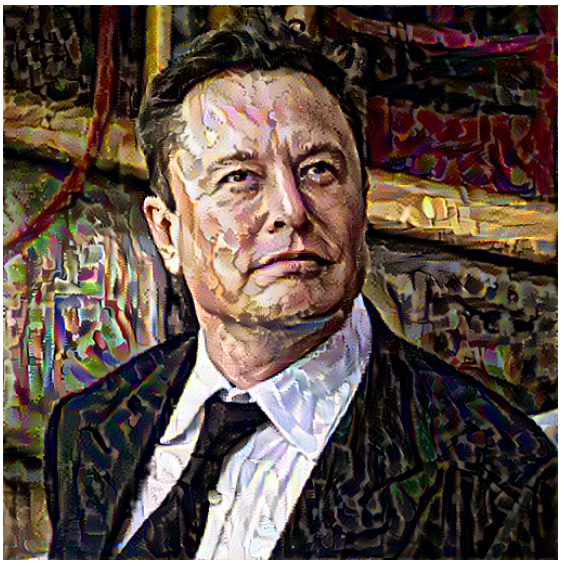



Figure 6: Content and Style Image We Provided To The NST Algorithm

Table 1: Output image and total time taken for NST using different optimizers with different epochs and train steps.

| Optimizer | Learning rate | Epochs | Train step | Total time | Output |
|---|---|---|---|---|---|
| Adam | 0.02 | 50 | 250 | 30.6 |  |
| Adagrad | 0.02 | 50 | 250 | 30.4 |  |

| | | | | | |
|---|---|---|---|---|---|
| Adagrad | 0.1 | 50 | 250 | 30.2 |  |
| Adam | 0.02 | 100 | 500 | 58.0 |  |
| Adagrad | 0.02 | 100 | 500 | 58.1 |  |

| | | | | | |
|---|---|---|---|---|---|
| RMSprop | 0.02 | 100 | 500 | 57.2 |  |
| Adam | 0.02 | 500 | 2500 | 151.6 |  |
| Adagrad | 0.02 | 500 | 2500 | 151.8 |  |

| | | | | | |
|---|---|---|---|---|---|
| RMSprop | 0.02 | 500 | 2500 | 149.6 |  |
| RMSprop | 0.001 | 500 | 2500 | 152.9 |  |

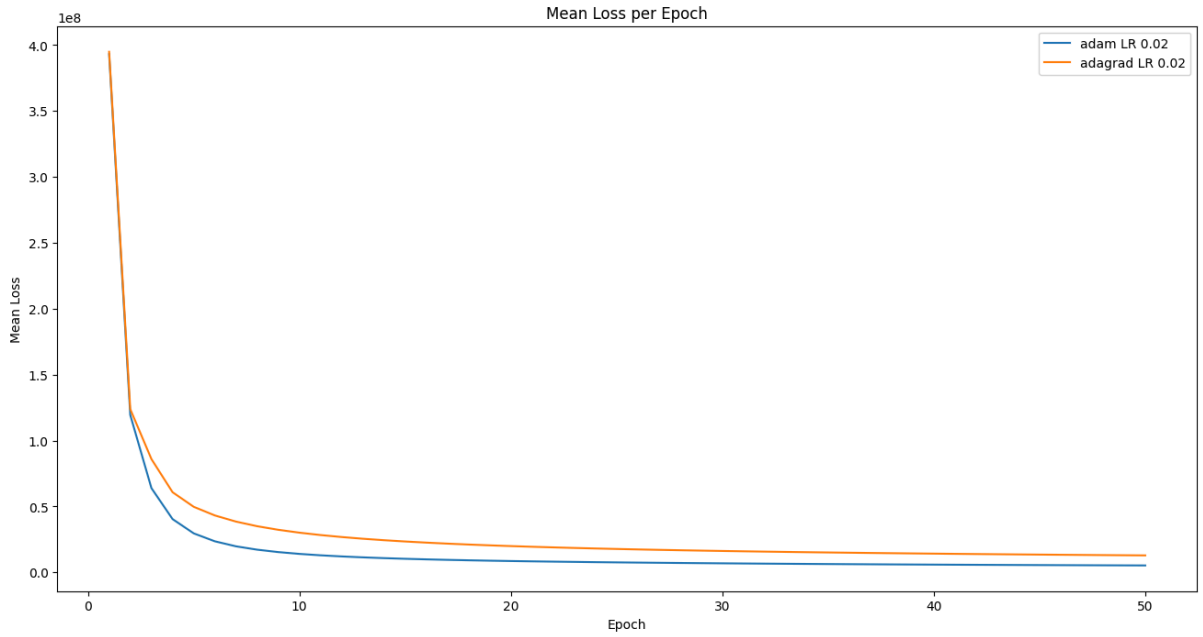



Figure 7: plot loss for Adagrad and Adam with LR=0.02 and epochs=50

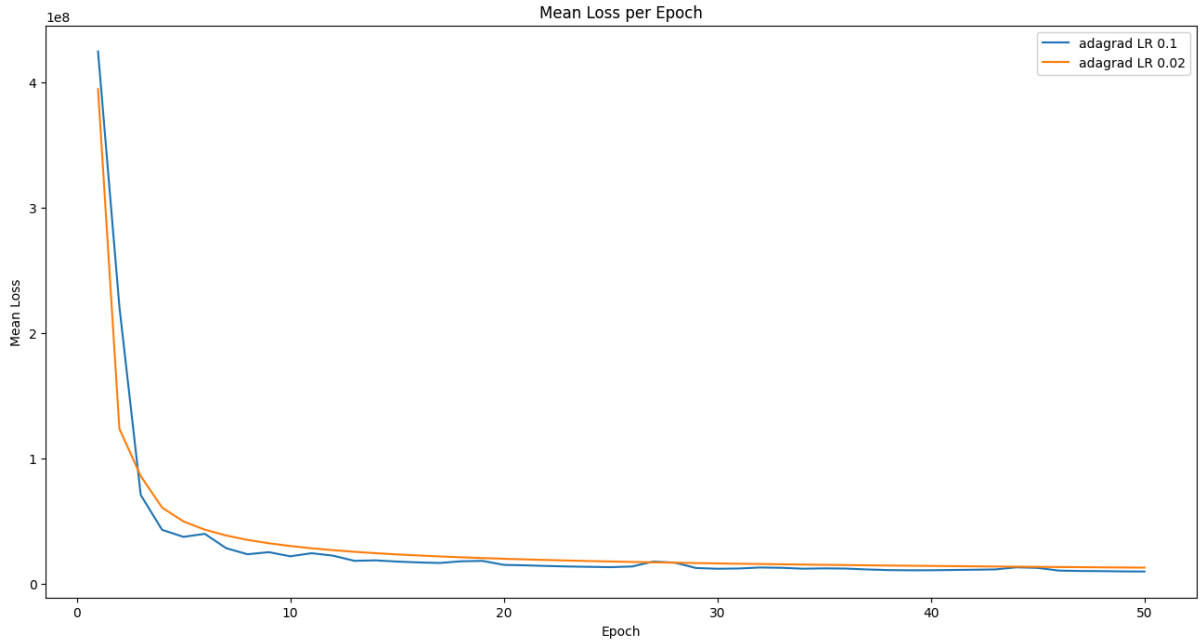



Figure 8: plot loss for Adagrad and Adam with LR=0.1 and epochs=50

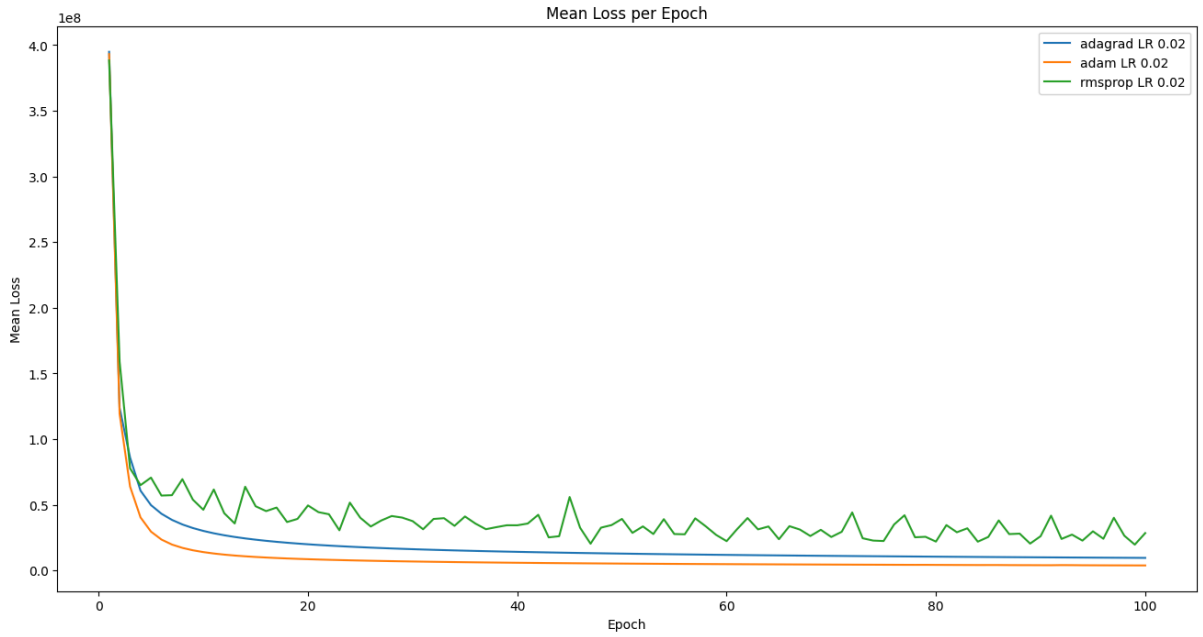



Figure 9: plot loss for Adagrad, RMSprop and Adam with LR=0.02 and epochs=100

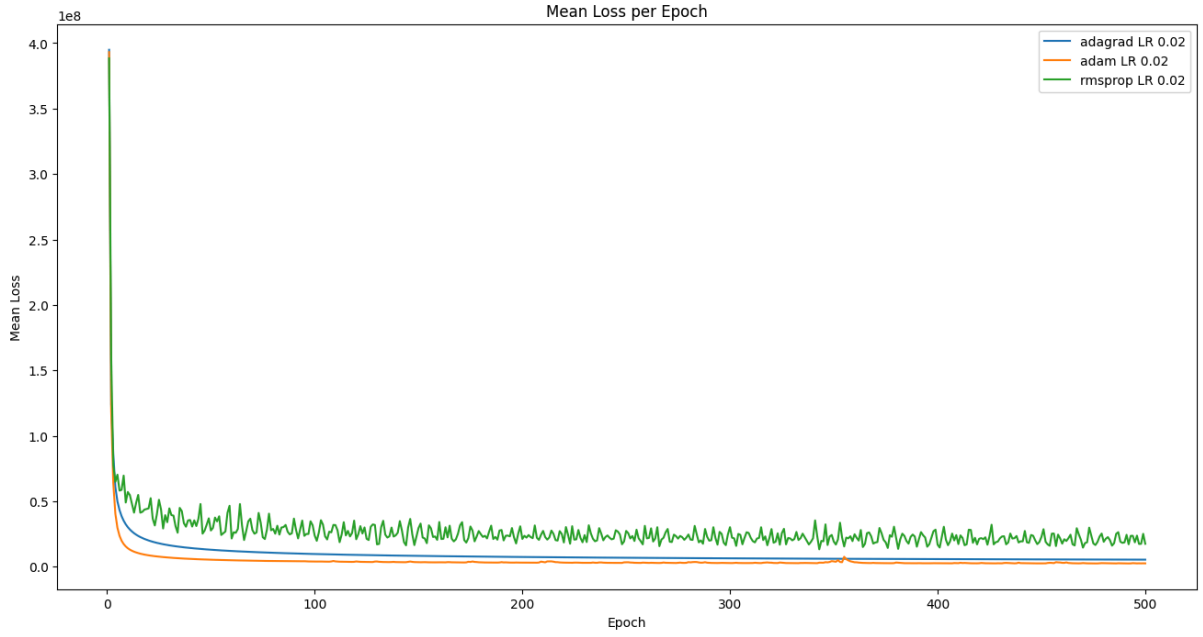



Figure 10: plot loss for Adagrad, RMSprop and Adam with LR=0.02 and epochs=500

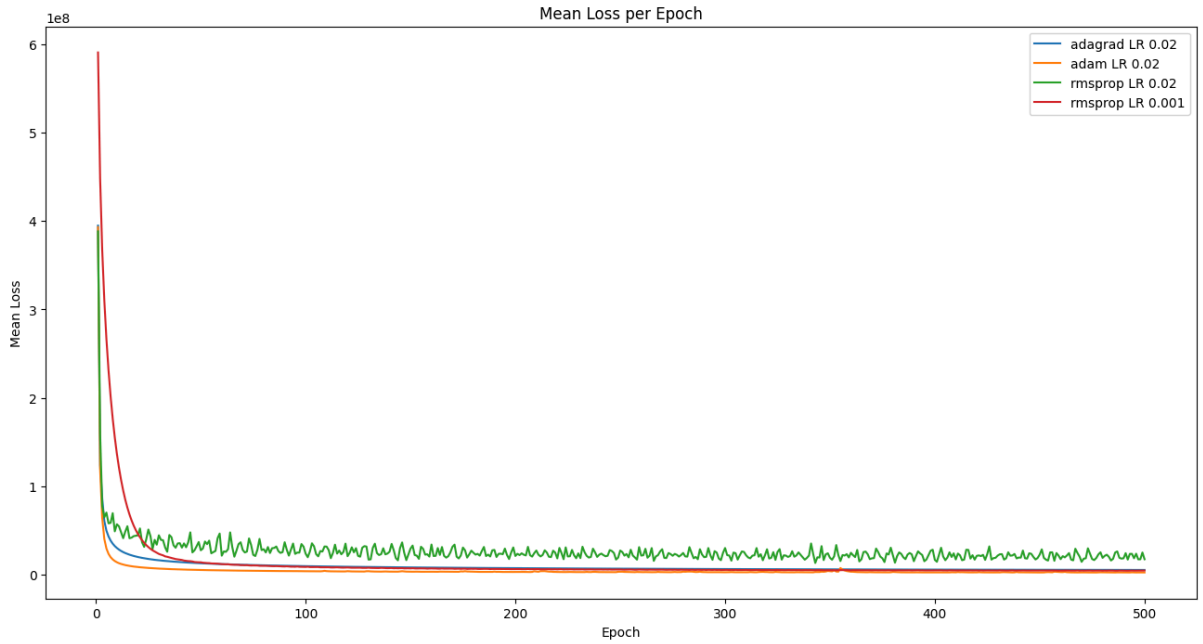



Figure 10: plot loss for Adagrad, RMSprop and Adam with different LR's and epochs=500

## Chapter 5: Important Code Snippets

VGG layers:

```
def vgg_layers(layer_names):
  print(layer_names)
  # Load model
  vgg = tf.keras.applications.VGG19(include_top=False,
weights='imagenet')
  # We only need feature extraction
  vgg.trainable = False
  # Get Outputs for layer names specified in the model
  outputs = [vgg.get_layer(name).output for name in layer_names]
  # New model with specified layers only
  model = tf.keras.Model([vgg.input], outputs)
  # Model summary
  print(model.summary())

  return model
```

Gram matrix:

```
def gram_matrix(input_tensor):
  result = tf.linalg.einsum('bijc,bijd->bcd', input_tensor,
input_tensor)
  input_shape = tf.shape(input_tensor)
  num_locations = tf.cast(input_shape[1]*input_shape[2], tf.float32)
  return result/(num_locations)
```

Train step function:

```
@tf.function()
def train_step(image,opt):
  with tf.GradientTape() as tape:
    outputs = extractor(image)
    loss = style_content_loss(outputs)

  grad = tape.gradient(loss, image)
  opt.apply_gradients([(grad, image)])
  image.assign(clip_0_1(image))

  return loss, image
```

Run optimizer function:

```
def run_optimizer(opt, image, epochs, steps_per_epoch):
  losses = []
  result_image= None
  start = time.time()
  step = 0
  for n in range(epochs):
    epoch_losses=[]

    for m in range(steps_per_epoch):
      step += 1
      loss, result_image = train_step(image,opt)
      epoch_losses.append(loss)
      print(".", end='', flush=True)
    display.clear_output(wait=True)
    display.display(tensor_to_image(image))
    print("Train step: {}".format(step))
    epoch_mean_loss = tf.reduce_mean(epoch_losses)
    losses.append(epoch_mean_loss)

  end = time.time()
  print("Total time: {:.1f}".format(end-start))
  return losses, result_image
```

Running Adam optimizer with 50 epochs and LR=0.02:

```
# Adam Optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.02)

content_image = image
image_variable = tf.Variable(content_image)

# Run the optimizer
adam_losses_50_lr1, adam_50_lr1=run_optimizer(optimizer,
image_variable, epochs=50, steps_per_epoch=5)
```

# Chapter 6: Conclusion and Future Work

In conclusion, Neural Style Transfer represents a powerful and versatile technique for creating artistic images by combining content and style features from different images. Optimizers played a crucial role in training neural networks for Neural Style Transfer. By adapting the learning rates and updating parameters efficiently, optimizers help in accelerating convergence and improving the quality of stylized images.
In conclusion, this report has provided a detailed exploration of Neural Style Transfer, covering its background, proposed system architecture, code implementation, optimization techniques, model architecture explanation, and performance comparison. This detailed discussion provides insights into the working of optimizers in the context of NST, which can help people interested in NST make informed decisions when designing and optimising neural style transfer systems. We found VGG to be faster specifically for image style transfer. Using VGG19 we were able to get more features.

Future research directions for NST include exploring novel loss functions, investigating alternative architectures, improving computational efficiency, and extending the application domain to video and three-dimensional (3D) content where there are more than three loss functions.

# Chapter 9: References

[1] J. Liao, "A Study on Neural Style Transfer Methods for Images," 2022 2nd International Conference on Big Data, Artificial Intelligence and Risk Management (ICBAR), Xi'an, China, 2022, pp. 60-64, doi: 10.1109/ICBAR58199.2022.00019. keywords: {Deep learning;Training;Big Data;Rendering (computer graphics);Generative adversarial networks;Image filtering;Risk management;neural style transfer;deep learning;convolutional neural network;image rendering;generative adversarial nets;texture synthesis}

[2] Deshmane, Aishwarya. (2023). Image Style Transfer using Neural Network.

[3] L. A. Gatys, A. S. Ecker and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 2414-2423, doi: 10.1109/CVPR.2016.265. keywords: {Image reconstruction;Neural networks;Image representation;Semantics;Neuroscience;Feature extraction;Visualization}

[4] Y. Tao, "Image Style Transfer Based on VGG Neural Network Model," 2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), Dalian, China, 2022, pp. 1475-1482, doi: 10.1109/AEECA55500.2022.9918891. keywords: {Training;Electrical engineering;Computer vision;Computational modeling;Image processing;Neural networks;Production;Image Style Transfer;Convolutional Neural Network;VGG-16;VGG-19}

[5] A. Ratra, A. Agarwal, V. Sharma, S. Vats, S. Singh and V. Kukreja, "A Comparative Analysis of Fast-style Transfer and VGG19-GramMatrix Approach to Neural Style Transfer," 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), Trichy, India, 2023, pp. 25-30, doi: 10.1109/ICAISS58487.2023.10250595. keywords: {Geometry;Analytical models;Neural Style Transfer;Visual Geometry Group;Gram Matrix;Fast-style transfer;TensorFlow-hub model;Image stylization;deep learning;Image processing;Image analysis;Image synthesis;Image transformation;Comparative study;Performance evaluation}

[6] H. Ye, W. Liu and Y. Liu, "Image Style Transfer Method Based on Improved Style Loss Function," 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 2020, pp. 410-413, doi: 10.1109/ITAIC49862.2020.9338927. keywords: {Image quality;Image texture;PSNR;Neural networks;Feature extraction;Distortion;Information technology;image style transfer;gram matrix;neural network;feature extraction}

[7] Dey, Sandipan. (2018). Hands-on Image Processing in Python, Publisher(s): Packt Publishing, ISBN: 9781789343731