Neural Style Transfer: An Implementation Using VGG19

Sheheryar Sohail
03-134211-040
Department of Computer Science
Bahria University
Lahore, Pakistan
sohailsheheryar226@gmail.com

Murtaza Anwaar
03-134211-032
Department of Computer Science
Bahria University
Lahore, Pakistan
murtazaanwaar@gmail.com

***Abstract*— Neural style transfer is a technique that synthesizes an image by combining the content of one image with the style of another. This project implements neural style transfer using the VGG19 model. By leveraging pre-trained convolutional neural networks, we achieve compelling visual results where the content of a base image is reimagined in the artistic style of a second image. This report covers the methodology, implementation, and evaluation of the neural style transfer process.**

Keywords—style transfer, vgg19, neural network, imagenet

I. Introduction

The concept of style transfer was first introduced by **(Gatys, Ecker, & Bethge, 2015)**, who used the VGG network to extract hierarchical feature representations. Their pioneering work demonstrated how deep neural networks could separate and recombine image content and style. Subsequent research has focused on improving efficiency and quality, including methods such as perceptual loss functions **(Johnson, Alahi, & Fei-Fei, 2016)** and real-time style transfer **(Ulyanov, Vedaldi, & Lempitsky, 2017)**. The VGG19 model, known for its depth and performance, is a common choice for such tasks due to its ability to capture detailed feature representations at multiple levels.

II. Literatue Review

The concept of style transfer was first introduced by **(Gatys, Ecker, & Bethge, 2015)**, who used the VGG network to extract hierarchical feature representations. Their pioneering work demonstrated how deep neural networks could separate and recombine image content and style. Subsequent research has focused on improving efficiency and quality, including methods such as perceptual loss functions **(Johnson, Alahi, & Fei-Fei, 2016)** and real-time style transfer **(Ulyanov, Vedaldi, & Lempitsky, 2017)**. The VGG19 model, known for its depth and performance, is a common choice for such tasks due to its ability to capture detailed feature representations at multiple levels.

A. Evolution of Style Transfer Methods

1) Classical Style Transfer: ***(Gatys, Ecker, & Bethge, 2015)*** laid the foundation for neural style transfer by leveraging convolutional neural networks (CNNs) to achieve artistic effects. They utilized the VGG19 network to separate the content of an image from its style by computing content and style representations through different layers of the network. This method, though groundbreaking, was computationally intensive and time-consuming due to the optimization process required for each new image pair.

2) Perceptual Loss Functions: ***(Johnson, Alahi, & Fei-Fei, 2016)*** addressed the computational inefficiency by introducing perceptual loss functions for training feed-forward networks for style transfer. Their method enabled real-time style transfer by training a network to minimize a perceptual loss, which is defined based on differences in high-level feature representations rather than pixel-wise differences. This approach significantly reduced the computation time, making it feasible to apply style transfer in real-time applications.

3) Real-Time Style Transfer: Building on the work of ***(Johnson, Alahi, & Fei-Fei, 2016)***, ***(Ulyanov, Vedaldi, & Lempitsky, 2017)*** further improved real-time style transfer by proposing a simpler and faster method that achieved comparable quality. They introduced instance normalization, which helped in stabilizing the training and resulted in better visual quality of stylized images. Their approach made it possible to apply style transfer efficiently on consumer hardware, paving the way for broader applications in creative and multimedia industries.

B. Advanced Techniques and Improvements

1) AdaIN and Adaptive Style Transfer: The introduction of Adaptive Instance Normalization (AdaIN) by **(Huang & Belongie, 2017)** marked a significant advancement in style transfer techniques. AdaIN allowed the style of an image to be encoded in the mean and variance of feature activations, enabling style transfer without the need for additional optimization steps. This method facilitated arbitrary style transfer in real-time, where a single model could transfer multiple styles.

2) GAN-based Style Transfer: Generative Adversarial Networks (GANs) have also been utilized to improve style transfer. ***(Zhang & Dana, 2017)*** used CycleGANs to achieve style transfer by learning mappings between unpaired datasets. This method allowed for the transfer of styles from one domain to another without requiring paired training examples. GANs have also been employed to enhance the quality of style transfer results, making the stylized images more realistic and visually appealing.

3) Diverse and Controllable Style Transfer: Recent research has focused on providing more control over the style transfer process. Methods such as controllable style transfer allow users to adjust the degree of stylization and to blend multiple styles. ***(Huang & Belongie, 2017)*** introduced conditional instance normalization, enabling a single network to learn multiple styles by conditioning on style-specific parameters. This approach provided flexibility and expanded the creative possibilities for users.

III. Methodology

A. Dataset

For this project, two images are used: one for content and one for style. These images can be any arbitrary pictures selected to demonstrate the style transfer process. While the VGG19 model has been trained on the ImageNet dataset.

B. Preprocessing

Images are resized to 400x400 pixels and preprocessed to match the input requirements of the VGG19 network, which includes scaling pixel values and reordering color channels.

```
def load_and_process_img(path_to_img):
    img = load_img(path_to_img, target_size=(400, 400))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = tf.keras.applications.vgg19.preprocess_input(img)
  return img
```

C. Model Architecture

The VGG19 model, pre-trained on ImageNet, is used to extract content and style features. Specific layers are selected to capture content and style representations:

```
def get_model():
    vgg = VGG19(include_top=False, weights='imagenet')
    vgg.trainable = False
    content_layers = ['block5_conv2']
    style_layers = ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1', 'block5_conv1']
    content_outputs = [vgg.get_layer(name).output for name in content_layers]
    style_outputs = [vgg.get_layer(name).output for name in style_layers]
    model_outputs = content_outputs + style_outputs
  return Model(vgg.input, model_outputs)
```

D. Loss Functions

Content and style losses are computed using the mean squared error and Gram matrix, respectively. The total loss is a weighted sum of these losses:

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


def gram_matrix(input_tensor):
    channels = int(input_tensor.shape[-1])
    a = tf.reshape(input_tensor, [-1, channels])
    n = tf.shape(a)[0]
    gram = tf.matmul(a, a, transpose_a=True)
    return gram / tf.cast(n, tf.float32)


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


def compute_loss(model, loss_weights, init_image, gram_style_features, content_features):
    model_outputs = model(init_image)
    content_output_features = model_outputs[:len(content_features)]
    style_output_features = model_outputs[len(content_features):]


    content_score = 0
    style_score = 0


    weight_content, weight_style = loss_weights


    for target_content, comb_content in zip(content_features, content_output_features):
        content_score += get_content_loss(comb_content, target_content)


    for target_style, comb_style in zip(gram_style_features, style_output_features):
        style_score += get_style_loss(comb_style, target_style)


    content_score *= weight_content
    style_score *= weight_style


    loss = content_score + style_score
  return loss, content_score, style_score
```

E. *Optimization*

The Adam optimizer is used to minimize the total loss by updating the generated image iteratively:

```
def compute_grads(cfg):
    with tf.GradientTape() as tape:
        all_loss = compute_loss(**cfg)
    total_loss = all_loss[0]
    return tape.gradient(total_loss, cfg['init_image']), all_loss

def run_style_transfer(content_path, style_path, num_iterations=1000, content_weight=1e3, style_weight=1e-2):
    model = get_model()
    for layer in model.layers:
        layer.trainable = False

    content_image = load_and_process_img(content_path)
    style_image = load_and_process_img(style_path)

    content_features = model(content_image)[:1]
    style_features = model(style_image)[1:]
    gram_style_features = [gram_matrix(style_feature) for style_feature in style_features]

    init_image = tf.Variable(content_image, dtype=tf.float32)
    opt = tf.optimizers.Adam(learning_rate=5, beta_1=0.99, epsilon=1e-1)
    iter_count = 1

    best_loss, best_img = float('inf'), None

    loss_weights = (content_weight, style_weight)
    cfg = {
        'model': model,
        'loss_weights': loss_weights,
        'init_image': init_image,
        'gram_style_features': gram_style_features,
        'content_features': content_features
    }

    norm_means = np.array([103.939, 116.779, 123.68])
    min_vals = -norm_means
    max_vals = 255 - norm_means

    for i in range(num_iterations):
        grads, all_loss = compute_grads(cfg)
        loss, content_score, style_score = all_loss
        opt.apply_gradients([(grads, init_image)])
        clipped = tf.clip_by_value(init_image, min_vals, max_vals)
        init_image.assign(clipped)

        if loss < best_loss:
            best_loss = loss
            best_img = deprocess_img(init_image.numpy())

        if i % 100 == 0:
            print('Iteration: {}'.format(i))
            print('Total loss: {:.4e}, '
                  'style loss: {:.4e}, '
                  'content loss: {:.4e}'.format(loss, style_score, content_score))

    return best_img, best_loss

content_path = 'path_to_content_image.jpg'
style_path = 'path_to_style_image.jpg'
best_img, best_loss = run_style_transfer(content_path, style_path)
plt.imshow(best_img)
plt.show()
```

IV. Experiments & Results

A. *Evaluation Measures*

In neural style transfer, the evaluation of results is often subjective, focusing on the visual appeal of the generated images. However, quantitative measures are also used to assess the quality of content and style transfer:

1) Content Loss: Measures the difference between the feature representations of the content image and the generated image. Lower content loss indicates better content preservation.

Content Loss = 2.0556×10^6

2) Style Loss: Measures the difference between the style representations (Gram matrices) of the style image and the generated image. Lower style loss indicates better style transfer.

Style Loss = 2.1377×10^6

3) Total Loss: A weighted sum of content and style losses, balancing the importance of content preservation and style transfer.

Total Loss = 4.1934×10^6

B. *Hyperparameters*

The performance and output quality of neural style transfer are influenced by several hyperparameters. Key hyperparameters used in this project include:

1) Content Weight (α): Weight assigned to the content loss.

Value: 1×10^3

2) Style Weight (β): Weight assigned to the style loss.

Value: 1×10 ^ -2

3) Learning Rate: Step size for the optimizer.

Value: 5.0

4) Number of Iterations: Number of optimization steps.

Value: 1000

5) Optimizer: Algorithm used to minimize the loss.

Value: Adam optimizer with β1=0.99 and ϵ=1×10−1

C. *Result*

The results of the neural style transfer process are evaluated based on visual inspection and quantitative measures of content and style losses. The following results were obtained after 1000 iterations of optimization:

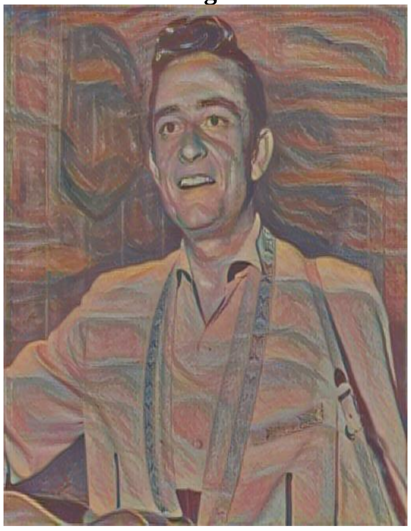
1) Generated Image: The final image produced by the neural style transfer process, displaying the content of the base image with the artistic style of the style image.

a) Visualization:

```
best_img, best_loss = run_style_transfer(content_path, style_path)
plt.imshow(best_img)
plt.title("Generated Image")
plt.show()
```

b) Sample Output:

Figure 1

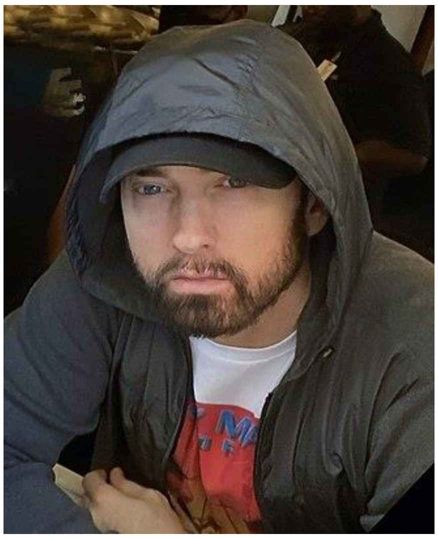


D. *Comparison with Original Images*

To better understand the effectiveness of the style transfer process, comparisons between the content image, style image, and generated image are provided.

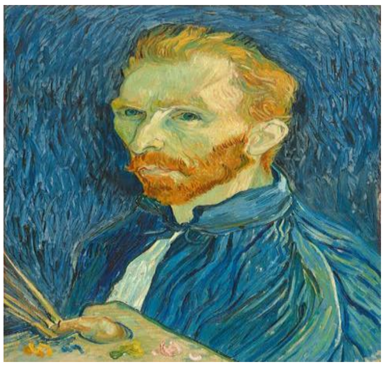
a) Content Image:

Figure 2



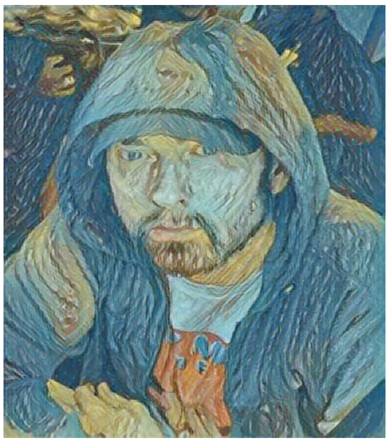
b) Style Image:

Figure 3



c) Generated Image:

Figure 4



V. DISCUSSION

A. *Comparison with State-of-the-Art Models*

Neural style transfer has evolved significantly since its inception. Modern techniques often employ alternative models like EfficientNet or use GANs for improved efficiency and realism. While VGG19 is effective, newer architectures may offer faster and more visually appealing results.

B. *Comparison Table*

Table 1. Comparison With Other Models

Model	**Method**	**Speed**	**Quality**	***Key Features***
VGG19 (Gatys et al.)	Optimization-based	Slow	High	Traditional style transfer, high detail
Johnson et al.	Perceptual Loss	Medium	Medium	Perceptual loss, real-time transfer
Ulyanov et al.	Instance Norm	Fast	Medium	Real-time transfer, instance normalization
GAN-based methods	Adversarial Loss	Fast	High	Realism, efficiency

VI. Conclusion

This project demonstrates the application of neural style transfer using the VGG19 model. By leveraging deep learning, we can create visually compelling images that blend content and style in a coherent manner. The VGG19 model's deep architecture allows for detailed feature extraction, enabling effective style transfer.

VII. Future Directions

Future work could explore the use of more efficient models and techniques, such as:

- Utilizing more recent CNN architectures like EfficientNet for improved performance.
- Implementing real-time style transfer methods using instance normalization.
- Exploring GAN-based approaches for enhanced realism and efficiency.
- Experimenting with different loss functions and optimization techniques to balance content and style better.

By advancing these areas, the field of neural style transfer can continue to produce more efficient, high-quality results for various applications.

References

Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A Neural Algorithm of Artistic Style.

Huang, X., & Belongie, S. (2017). Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization.

Hung, X., & Belongie, S. (2017). Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization.

Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual Losses for Real-Time Style Transfer and Super-Resolution.

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2017). Instance Normalization: The Missing Ingredient for Fast Stylization.

Zhang, H., & Dana, K. (2017). Multi-style Generative Network for Real-time Transfer.