Neural Style Transfer

Ankit Lal
22113019

Abstract

This report explores the fascinating world of Neural Style Transfer (NST), a technique that merges the content of one image with the artistic style of another. Leveraging the power of convolutional neural networks (CNNs), NST allows users to create visually stunning and artistically coherent images, blurring the lines between photography and artistic expression. This report delves into the core concepts of NST, explores its implementation using a pre-trained VGG19 model, and discusses potential applications and future directions for this groundbreaking technology.

1. Introduction

The human desire to create art is deeply ingrained in our history. From cave paintings to modern masterpieces, art has served as a means of expression, communication, and exploration of the human experience. With the advent of deep learning, a new chapter unfolds in this artistic narrative. Neural Style Transfer (NST) emerges as a powerful tool, enabling users to seamlessly blend the content of a photograph with the artistic style of a renowned painter, like Van Gogh or Monet. This report delves into the technical aspects of NST, showcasing its ability to transform ordinary images into captivating works of art.

2. Neural Style Transfer

NST operates on the principle of content and style separation. Given a content image (C) and a style image (S), the goal is to generate a new image (X) that preserves the object details and spatial relationships from C while incorporating the artistic essence of S. This artistic essence is captured through the style of the brushstrokes, color palettes, and textures present in the style image.

At the heart of NST lies the power of convolutional neural networks (CNNs). These deep learning architectures are trained on vast image datasets, allowing them to learn intricate features and relationships within images. Pre-trained CNNs, like VGG19, serve as the backbone for NST. By analyzing both the content and style images through the pre-trained CNN, we can extract feature representations that encode the content and style information.

3. Content and Style Representation

The success of NST hinges on effectively separating content and style information.

3.1 Content Representation

Content refers to the objects and their spatial relationships within an image. To capture content, NST typically utilizes activations from earlier layers of the CNN. These layers learn to identify basic shapes, edges, and textures, forming a high-level understanding of the content.

3.2 Style Representation

Style, on the other hand, encompasses the artistic flourishes, brushstrokes, and color palettes that define a particular artist's work. NST achieves style representation through the Gram matrix. This matrix captures the correlations between feature activations across different channels in a specific layer of the CNN. By analyzing the Gram matrices of the style image and the generated image, NST can ensure that the generated image exhibits similar stylistic features to the style image.

4. Implementation using VGG19

The provided code snippet showcases a basic implementation of NST using the VGG19 model. Here's a breakdown of the key steps:

4.1 Pre-trained VGG19 Model

The code loads the pre-trained VGG19 model, leveraging its pre-trained convolutional layers for feature extraction.

4.2 Image Preprocessing

Both the content and style images undergo preprocessing to match the input format expected by the VGG19 model. This includes resizing, normalization, and mean subtraction.

4.3 Content and Style Loss Functions

The code defines two separate loss functions: content loss and style loss. The content loss measures the difference between the feature activations of the content image and the generated image in a specific layer of the VGG19 model.

The style loss compares the Gram matrices of the style image and the generated image across multiple layers of the VGG19 model.

4.4 Optimization

An optimization algorithm iteratively modifies the generated image to minimize the total loss, which is a weighted combination of content loss and style loss. This process ensures that the generated image retains the content of the original image while incorporating the style of the chosen artwork.

4.5 Image Postprocessing

Once the optimization process converges, the generated image is post-processed to remove normalization and restore it to its natural color range.

5. Applications of NST

NST presents a plethora of exciting possibilities beyond simply creating aesthetically pleasing images. Here are some potential applications:

5.1 Art Education

NST can be a valuable tool for art education, allowing students to explore and understand different artistic styles by applying them to their own photographs.

5.2 Image Restoration

NST can be employed for image restoration, where damaged or incomplete artworks can be partially reconstructed by incorporating the style of the original piece.

5.3 Product Design

NST can be leveraged in product design to generate creative product textures and patterns inspired by different artistic styles.

5.4 Fashion Design

Similarly, NST can be used in fashion design to create unique and innovative clothing patterns inspired by artistic movements.

6. Conclusion

Neural Style Transfer represents a significant advancement in the intersection of art and technology. By leveraging deep learning techniques, NST enables the seamless blending of content and artistic style, opening up new avenues for creative expression. The implementation using the VGG19 model demonstrates the feasibility and effectiveness of NST in generating visually stunning images. As technology continues to evolve, the potential applications of NST are boundless, promising exciting future directions for this groundbreaking technology.

References

1. Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). A Neural Algorithm of Artistic Style. arXiv:1508.06576.
2. Johnson, J., Alahi, A., & Fei-Fei, L. (2016). Perceptual Losses for Real-Time Style Transfer and Super-Resolution. arXiv:1603.08155.
3. Huang, X., & Belongie, S. (2017). Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. arXiv:1703.06868.
4. Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2017). Improved Texture Networks: Maximizing Quality and Diversity in Feed-Forward Stylization and Texture Synthesis. arXiv:1705.06830.
5. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., & Yang, M.-H. (2018). Universal Style Transfer via Feature Transforms. arXiv:1804.03547.
6. Jing, Y., Yang, Y., Feng, Z., Ye, J., Yu, Y., & Song, M. (2019). Neural Style Transfer: A Review. arXiv:1912.07921.

Appendix: Code Implementation

```python
import os
import sys
import scipy.io
import scipy.misc
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
```

```
from PIL import Image
from nst_utils import *
import numpy as np
import tensorflow as tf

class CONFIG:
    IMAGE_WIDTH = 400
    IMAGE_HEIGHT = 300
    COLOR_CHANNELS = 3
    NOISE_RATIO = 0.6
    MEANS = np.array([123.68, 116.779, 103.939]).reshape((1,1,1,3))
    VGG_MODEL = 'pretrained-model/imagenet-vgg-verydeep-19.mat'
    STYLE_IMAGE = 'images/stone_style.jpg'
    CONTENT_IMAGE = 'images/content300.jpg'
    OUTPUT_DIR = 'output/'

def load_vgg_model(path):
    vgg = scipy.io.loadmat(path)
    vgg_layers = vgg['layers']

    def _weights(layer, expected_layer_name):
        wb = vgg_layers[0][layer][0][0][2]
        W = wb[0][0]
        b = wb[0][1]
        layer_name = vgg_layers[0][layer][0][0][0][0]
        assert layer_name == expected_layer_name
        return W, b

    def _relu(conv2d_layer):
        return tf.nn.relu(conv2d_layer)

    def _conv2d(prev_layer, layer, layer_name):
        W, b = _weights(layer, layer_name)
        W = tf.constant(W)
        b = tf.constant(np.reshape(b, (b.size)))
        return tf.nn.conv2d(prev_layer, filter=W, strides=[1, 1, 1, 1],
padding='SAME') + b

    def _conv2d_relu(prev_layer, layer, layer_name):
        return _relu(_conv2d(prev_layer, layer, layer_name))
```

```
    def _avgpool(prev_layer):
        return tf.nn.avg_pool(prev_layer, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')

    graph = {}
    graph['input'] = tf.Variable(np.zeros((1, CONFIG.IMAGE_HEIGHT,
CONFIG.IMAGE_WIDTH, CONFIG.COLOR_CHANNELS)), dtype = 'float32')
    graph['conv1_1'] = _conv2d_relu(graph['input'], 0, 'conv1_1')
    graph['conv1_2'] = _conv2d_relu(graph['conv1_1'], 2, 'conv1_2')
    graph['avgpool1'] = _avgpool(graph['conv1_2'])
    graph['conv2_1'] = _conv2d_relu(graph['avgpool1'], 5, 'conv2_1')
    graph['conv2_2'] = _conv2d_relu(graph['conv2_1'], 7, 'conv2_2')
    graph['
```