

NEURAL STYLE TRANSFER PROJECT REPORT

INTRODUCTION: Neural Style Transfer (NST) is a cutting-edge technique in deep learning that allows for the transformation of an ordinary photograph into a masterpiece by applying the artistic style of renowned painters such as Van Gogh, Picasso, or Monet. This project aims to develop a neural style transfer model that leverages state-of-the-art deep learning techniques to merge the content of one image with the style of another, creating stunning visuals that seamlessly blend the two.

OBJECTIVES:

- 1.Extracting stylistic features from a given artwork.
- 2.Applying these stylistic features to a different image while preserving the original content's structure and details.
- 3.Achieving a balance between the content and style to create visually appealing and artistically coherent images.

Techniques Used

This project explores techniques such as convolutional neural networks (CNNs), optimization algorithms, and perceptual loss functions. The model utilizes the VGG19 network, pre-trained on the ImageNet dataset, to extract features from the content and style images. The implementation avoids using external APIs and showcases the model through a web interface built with Streamlit.

Methodology:

1. Environment Setup

To run the neural style transfer model, ensure the necessary libraries are installed:

```
pip install tensorflow streamlit numpy pillow matplotlib
```

2. Loading and Preprocessing Images

The first step involves loading and preprocessing the content and style images. The images are resized and normalized to facilitate processing by the VGG19 model.

3. Feature Extraction with VGG19

The VGG19 model, pre-trained on the ImageNet dataset, is used to extract features from the content and style images. Specific layers of the VGG19 model are chosen to capture content and style representations

4. Defining the Style and Content Loss

The style loss is computed using the Gram matrix of the style image, which captures the correlations between different feature maps. The content loss is computed as the mean squared difference between the feature maps of the content image and the generated image.

5. Optimization Algorithm

An optimization algorithm is employed to iteratively update the generated image to minimize the combined style and content loss. The Adam optimizer is used for this purpose.

6. Combining Content and Style

The generated image is updated to balance the content and style features, producing a new image that retains the structure of the content image while adopting the style of the style image.

7. Streamlit Web Interface

A user-friendly web interface is created using Streamlit, allowing users to upload content and style images and view the generated stylized image

Significance of Content and Style Loss:

Content Loss

Definition: Content loss measures how well the generated image preserves the content of the original content image. It is typically defined as the Mean Squared Error (MSE)

between the feature representations of the content image and the generated image, extracted from a particular layer of a Convolutional Neural Network (CNN)

Significance:

- **Content Preservation:** The main role of content loss is to ensure that the generated image keeps the semantic information of the content image.
- **Feature Representation:** By comparing the feature representations from deep layers of VGG-19, content loss focus on higher-level content details rather than pixel-wise differences. •
- **Balancing:** In the NST process, content loss is balanced with style loss to achieve a mixup. The weight given to content loss determines how much the generated image will match the content image in terms of structure.

Style Loss:

Definition: Style loss measures how well the generated image captures the style of the style image. It is computed by comparing the Gram matrices of the feature maps from different layers of a CNN. The Gram matrix captures the correlations between different features, representing the texture and style.


Significance:

- **Texture and Patterns:** The primary role of style loss is to ensure that the generated image adopts the textures, colour, and patterns of the style image.
- **Feature Correlations:** By using the Gram matrix, style loss captures the combination of features across the image, which is essential for representing the style. •
- **Layer-wise Contributions:** Style loss is usually computed at multiple layers of the CNN. Each layer contributes different levels of detail, from fine textures in lower layers to more abstract patterns in higher layers. This multi-scale approach helps in capturing a comprehensive style representation • .
- **Balancing:** The weight assigned to style loss determines how much the generated image will adopt the style of the style image.

Code Implementation

Importing Libraries


python

 Copy code

```
import streamlit as st
import tensorflow as tf
import numpy as np
import PIL.Image
import matplotlib.pyplot as plt
import time
```

Setting up Streamlit

python


 Copy code

```
st.title("Neural Style Transfer")
```

Helper Functions

Convert Tensor to Image


python

 Copy code

```
def tensor_to_image(tensor):
    tensor = tensor * 255
    tensor = np.array(tensor, dtype=np.uint8)
    if np.ndim(tensor) > 3:
        tensor = tensor[0]
    return PIL.Image.fromarray(tensor)
```

Load and Preprocess Image

python

 Copy code

```
def load_img(uploaded_file):
    max_dim = 512
    img = tf.image.decode_image(uploaded_file, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)


    shape = tf.cast(tf.shape(img)[: -1], tf.float32)
    scale = max_dim / max(shape)
    new_shape = tf.cast(shape * scale, tf.int32)

    img = tf.image.resize(img, new_shape)
    img = img[tf.newaxis, :]
    return img
```



Display Image

python

 Copy code

```
def imshow(image, title=None):
    if len(image.shape) > 3:
        image = tf.squeeze(image, axis=0)
    plt.imshow(image)
    if title:
        plt.title(title)
    plt.show()
```

Uploading Images

python

Copy code

```
content_file = st.file_uploader("Upload Content Image", type=["jpg", "jpeg", "png"])
style_file = st.file_uploader("Upload Style Image", type=["jpg", "jpeg", "png"])

if content_file and style_file:
    content_image = load_img(content_file.read())
    style_image = load_img(style_file.read())

    st.image(tensor_to_image(content_image), caption='Content Image')
    st.image(tensor_to_image(style_image), caption='Style Image')
```

Define Layers for Style and Content

python

Copy code

```
content_layers = ['block5_conv2']
style_layers = ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1', 'block5_co
```

Load VGG19 Model

python

Copy code

```
def get_vgg_layers(layer_names):
    vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
    vgg.trainable = False
    outputs = [vgg.get_layer(name).output for name in layer_names]
    return tf.keras.Model([vgg.input], outputs)
```

Custom Model to Extract Style and Content

python

Copy code

```
class StyleContentModel(tf.keras.models.Model):
    def __init__(self, style_layers, content_layers):
        super(StyleContentModel, self).__init__()
        self.vgg = get_vgg_layers(style_layers + content_layers)
        self.style_layers = style_layers
        self.content_layers = content_layers
        self.num_style_layers = len(style_layers)
        self.vgg.trainable = False

    def call(self, inputs):
        inputs = inputs * 255.0
        preprocessed_input = tf.keras.applications.vgg19.preprocess_input(inputs)
        outputs = self.vgg(preprocessed_input)
        style_outputs, content_outputs = outputs[:self.num_style_layers], outputs[self.num_style_layers:]
        style_outputs = [self.gram_matrix(style_output) for style_output in style_outputs]
        content_dict = {content_name: value for content_name, value in zip(self.content_layers, content_outputs)}
        style_dict = {style_name: value for style_name, value in zip(self.style_layers, style_outputs)}
        return {'content': content_dict, 'style': style_dict}
```


Continue generating

@staticmethod

```
@staticmethod
def gram_matrix(input_tensor):
    result = tf.linalg.einsum('bi jc, bi jd-> bcd', input_tensor, input_tensor)
    num_locations = tf.shape(input_tensor)[1] * tf.shape(input_tensor)[2]
    return result / tf.cast(num_locations, tf.float32)
```

Extracting Style and Content


python

 Copy code

```
extractor = StyleContentModel(style_layers, content_layers)
style_targets = extractor(style_image)['style']
content_targets = extractor(content_image)['content']
```

Define Style and Content Loss Functions


python

 Copy code

```
def compute_loss(outputs, style_targets, content_targets, style_weight, content_weight, num_style_layers, num_content_layers):
    style_outputs = outputs['style']
    content_outputs = outputs['content']
    style_loss = tf.add_n([tf.reduce_mean((style_outputs[name] - style_targets[name]) ** 2) for name in style_outputs.keys()])
    style_loss *= style_weight / num_style_layers
    content_loss = tf.add_n([tf.reduce_mean((content_outputs[name] - content_targets[name]) ** 2) for name in content_outputs.keys()])
    content_loss *= content_weight / num_content_layers
    return style_loss + content_loss
```

Training Configuration

python

 Copy code

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.02, beta_1=0.99, epsilon=1e-1)
style_weight = 1e-2
content_weight = 1e4
total_variation_weight = 30
image = tf.Variable(content_image)
```

Training Step Function


```
python Copy code

@tf.function
def train_step(image, extractor, style_targets, content_targets, style_weight, content_weight):
    with tf.GradientTape() as tape:
        outputs = extractor(image)
        loss = compute_loss(outputs, style_targets, content_targets, style_weight, content_weight)
        loss += total_variation_weight * tf.image.total_variation(image)
    grad = tape.gradient(loss, image)
    optimizer.apply_gradients([(grad, image)])
    image.assign(tf.clip_by_value(image, 0.0, 1.0))
```

Training Loop

```
python Copy code

epochs = st.slider("Epochs", 1, 20, 10)
steps_per_epoch = st.slider("Steps per Epoch", 50, 200, 100)


start_time = time.time()

for epoch in range(epochs):
    for step in range(steps_per_epoch):
        train_step(image, extractor, style_targets, content_targets, style_weight, content_weight)
        st.text(f"Epoch {epoch + 1}, Step {step + 1}/{steps_per_epoch}")
        st.image(tensor_to_image(image), caption=f'Step {step + 1}', use_column_width=True)

total_time = time.time() - start_time
st.write(f"Total time: {total_time:.1f} seconds")
```

Saving and Displaying the Final Stylized Image

python

 Copy code

```
final_image = tensor_to_image(image)
final_image.save('stylized-image.png')
st.image(final_image, caption='Final Stylized Image')

st.download_button(label="Download Image", data
```

Results

Generated Images

The results of the neural style transfer are evaluated based on visual inspection. Below are some examples of the generated images:

CONTENT IMAGE :



STYLED IMAGE :



GENERATED STYLED IMAGE:



Discussion

Analysis of Results

The model combined the style of one image with the content of another. The generated images exhibit the artistic patterns of the style image while preserving the spatial structure of the content image.

Significance

- Artistic Applications: The ability to generate artwork by combining different styles and content.

- Educational Use: Demonstrates the application of convolutional neural networks (CNNs) in image processing.