Problem Statement: Create and Art with Neural style transfer on given image using deep learning.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import vgg19
from tensorflow.keras import backend as K
import PIL.Image as pil_image

# Load content and style images
def load_image(img_path, max_dim=512):
    img = pil_image.open(img_path)
    long = max(img.size)
    scale = max_dim / long
    new_size = tuple([int(dim * scale) for dim in img.size])
    img = img.resize(new_size, pil_image.LANCZOS)

    img = np.array(img)
    img = np.expand_dims(img, axis=0)
    img = vgg19.preprocess_input(img)

    return img

# Load the content and style images
content_image_path = r"C:\Users\user\Downloads\mmm.jpg"  # Replace with the actual path
style_image_path = r"C:\Users\user\Downloads\nn.jpg"  # Replace with the actual path

content_img = load_image(content_image_path)
style_img = load_image(style_image_path)

# Display images
def imshow(img, title=None):
    if isinstance(img, np.ndarray):
```

```
        img = np.squeeze(img, axis=0)
    img = img.copy()
    img /= 255.0
    plt.imshow(img)
    if title:
        plt.title(title)
    plt.axis('off')

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
imshow(content_img, 'Content Image')
plt.subplot(1, 2, 2)
imshow(style_img, 'Style Image')
plt.show()

# Build the VGG19 model
def build_vgg19_model():
    vgg = vgg19.VGG19(weights='imagenet', include_top=False)

    # Define layers to use for content and style
    content_layers = ['block5_conv2']  # Content image feature extraction layer
    style_layers = ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv1', 'block5_conv1']  # Style
image feature extraction layers

    # Model to extract output from specific layers
    outputs = [vgg.get_layer(layer).output for layer in (content_layers + style_layers)]
    model = tf.keras.models.Model([vgg.input], outputs)

    return model, content_layers, style_layers

# Extract features from image using the model
def get_features(model, img):
    img = tf.convert_to_tensor(img, dtype=tf.float32)
```

```
    outputs = model(img)
    content_features = outputs[:len(content_layers)]
    style_features = outputs[len(content_layers):]

    return content_features, style_features

# Compute content loss
def content_loss(content, generated):
    return tf.reduce_mean(tf.square(content - generated))

# Compute style loss
def gram_matrix(x):
    x = tf.squeeze(x, axis=0)
    result = tf.linalg.einsum('bijc,bijd->bcd', x, x)
    return result / tf.cast(x.shape[1] * x.shape[2], tf.float32)

def style_loss(style, generated):
    S = gram_matrix(style)
    G = gram_matrix(generated)
    return tf.reduce_mean(tf.square(S - G))

# Total variation loss (helps with image smoothness)
def total_variation_loss(x):
    x = tf.squeeze(x, axis=0)
    a = tf.square(x[:, :-1, :-1, :] - x[:, 1:, :-1, :])
    b = tf.square(x[:, :-1, :-1, :] - x[:, :-1, 1:, :])
    return tf.reduce_sum(a + b)

# Define the total loss function
def compute_loss(model, content_img, style_img, generated_img, content_weight=1e3, style_weight=1e-2,
tv_weight=1e-6):
    content_features, style_features = get_features(model, content_img)
```

```
    generated_features = model(generated_img)

    content_loss_value = content_loss(content_features[0], generated_features[0])

    style_loss_value = 0
    for style_feat, gen_feat in zip(style_features, generated_features[1:]):
        style_loss_value += style_loss(style_feat, gen_feat)

    style_loss_value *= style_weight / len(style_features)

    tv_loss_value = total_variation_loss(generated_img)

    total_loss = content_weight * content_loss_value + style_loss_value + tv_weight * tv_loss_value
    return total_loss, content_loss_value, style_loss_value, tv_loss_value

# Optimizer for the image
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['generated_img']), all_loss

# Main function to run the NST
def run_nst(content_img, style_img, model, num_iterations=1000):
    generated_img = tf.Variable(content_img, dtype=tf.float32)

    optimizer = tf.optimizers.Adam(learning_rate=5.0)

    cfg = {
        'model': model,
        'content_img': content_img,
        'style_img': style_img,
```

```
        'generated_img': generated_img,
    }

    for i in range(num_iterations):
        grads, all_loss = compute_grads(cfg)

        optimizer.apply_gradients([(grads, generated_img)])

        total_loss, content_loss_value, style_loss_value, tv_loss_value = all_loss

        if i % 100 == 0:
            print(f"Iteration {i}")
            print(f"Total Loss: {total_loss.numpy():.4e}, Content Loss: {content_loss_value.numpy():.4e}, Style Loss: {style_loss_value.numpy():.4e}, TV Loss: {tv_loss_value.numpy():.4e}")

    return generated_img

# Build the model and run the NST process
model, content_layers, style_layers = build_vgg19_model()

generated_img = run_nst(content_img, style_img, model, num_iterations=1000)

# Display the final result
imshow(generated_img.numpy(), 'Generated Image')
plt.show()
```