```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, models
from PIL import Image
import matplotlib.pyplot as plt

# ---------- Step 1: Load images ----------
def load_image(img_path, max_size=400, shape=None):
    image = Image.open(img_path).convert('RGB')
    size = max_size if max(image.size) > max_size else max(image.size)
    if shape:
        size = shape
    in_transform = transforms.Compose([
        transforms.Resize(size),
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406),
                            (0.229, 0.224, 0.225))])
    image = in_transform(image)[:3, :, :].unsqueeze(0)
    return image

# ---------- Step 2: Display helper ----------
def im_convert(tensor):
    image = tensor.clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1, 2, 0)
    image = image * (0.229, 0.224, 0.225) + (0.485, 0.456, 0.406)
    image = image.clip(0, 1)
    return image

# ---------- Step 3: Load model ----------
vgg = models.vgg19(weights=models.VGG19_Weights.IMAGENET1K_V1).features

# Freeze parameters
for param in vgg.parameters():
    param.requires_grad_(False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
vgg.to(device)

# ---------- Step 4: Load content and style images ----------
content = load_image("content.jpg").to(device)
style = load_image("style.jpg", shape=content.shape[-2:]).to(device)

# ---------- Step 5: Define layers ----------
def get_features(image, model, layers=None):
    if layers is None:
        layers = {'0': 'conv1_1',
                  '5': 'conv2_1',
                  '10': 'conv3_1',
                  '19': 'conv4_1',
                  '21': 'conv4_2',  # content layer
                  '28': 'conv5_1'}
    features = {}
    x = image
    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x
    return features

# ---------- Step 6: Gram matrix ----------
def gram_matrix(tensor):
    _, d, h, w = tensor.size()
    tensor = tensor.view(d, h * w)
    gram = torch.mm(tensor, tensor.t())
    return gram

# ---------- Step 7: Extract features ----------
content_features = get_features(content, vgg)
style_features = get_features(style, vgg)

# ---------- Step 8: Compute style grams ----------
style_grams = {layer: gram_matrix(style_features[layer]) for layer in style_features}

# ---------- Step 9: Initialize target image ----------
target = content.clone().requires_grad_(True).to(device)

# ---------- Step 10: Define style weights ----------
style_weights = {'conv1_1': 1.0,
                 'conv2_1': 0.75,
                 'conv3_1': 0.2,
                 'conv4_1': 0.2,
                 'conv5_1': 0.2}
```

```
content_weight = 1  # α
style_weight = 1e6  # β

# ---------- Step 11: Optimization ----------
optimizer = optim.Adam([target], lr=0.003)
steps = 2000

for i in range(1, steps + 1):
    target_features = get_features(target, vgg)
    content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2']) ** 2)

    style_loss = 0
    for layer in style_weights:
        target_feature = target_features[layer]
        target_gram = gram_matrix(target_feature)
        style_gram = style_grams[layer]
        layer_style_loss = style_weights[layer] * torch.mean((target_gram - style_gram) ** 2)
        b, d, h, w = target_feature.shape
        style_loss += layer_style_loss / (d * h * w)

    total_loss = content_weight * content_loss + style_weight * style_loss

    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()

    if i % 200 == 0:
```

```
    print(f"Step {i}/{steps}, Total loss: {total_loss.item():.4f}")

# ---------- Step 12: Display output ----------
final_img = im_convert(target)
plt.imshow(final_img)
plt.axis("off")
plt.title("🎨 Styled Image")
plt.show()
```