 Requirements
Install the required libraries:
pip install torch torchvision matplotlib pillow

 Python Script: neural_style_transfer.py
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from PIL import Image
import matplotlib.pyplot as plt
import copy

# Load device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Image loader
def image_loader(image_path, max_size=400):
image = Image.open(image_path).convert('RGB')

size = max(image.size) if max(image.size) < max_size else max_size
in_transform = transforms.Compose([
transforms.Resize(size),
transforms.ToTensor(),
transforms.Normalize(
mean=[0.485, 0.456, 0.406], # ImageNet standards
std=[0.229, 0.224, 0.225])
])

image = in_transform(image).unsqueeze(0)
return image.to(device)

# Show image
def imshow(tensor, title=None):
image = tensor.clone().detach().cpu().squeeze(0)
image = transforms.ToPILImage()(image)
plt.imshow(image)
if title:
plt.title(title)
plt.axis('off')
plt.show()

# Load images
content_img = image_loader("content.jpg")
style_img = image_loader("style.jpg")

assert content_img.size() == style_img.size(), "Images must be the same size"

# Content loss
class ContentLoss(nn.Module):
```

```
def __init__(self, target):
super().__init__()
self.target = target.detach()
def forward(self, x):
self.loss = nn.functional.mse_loss(x, self.target)
return x

# Style loss
def gram_matrix(input):
b, c, h, w = input.size()
features = input.view(b * c, h * w)
G = torch.mm(features, features.t())
return G.div(b * c * h * w)

class StyleLoss(nn.Module):
def __init__(self, target_feature):
super().__init__()
self.target = gram_matrix(target_feature).detach()
def forward(self, x):
G = gram_matrix(x)
self.loss = nn.functional.mse_loss(G, self.target)
return x

# Load pre-trained VGG19
cnn = models.vgg19(pretrained=True).features.to(device).eval()

# Layers to use
content_layers = ['conv_4']
style_layers = ['conv_1', 'conv_2', 'conv_3', 'conv_4', 'conv_5']

# Build the model
def get_style_model_and_losses(cnn, style_img, content_img):
cnn = copy.deepcopy(cnn)

content_losses = []
style_losses = []

model = nn.Sequential()
i = 0 # increment every time we see a conv
for layer in cnn.children():
if isinstance(layer, nn.Conv2d):
i += 1
name = f'conv_{i}'
elif isinstance(layer, nn.ReLU):
name = f'relu_{i}'
layer = nn.ReLU(inplace=False)
elif isinstance(layer, nn.MaxPool2d):
name = f'pool_{i}'
elif isinstance(layer, nn.BatchNorm2d):
name = f'bn_{i}'
else:
```

```
continue

model.add_module(name, layer)

if name in content_layers:
target = model(content_img).detach()
content_loss = ContentLoss(target)
model.add_module(f"content_loss_{i}", content_loss)
content_losses.append(content_loss)

if name in style_layers:
target = model(style_img).detach()
style_loss = StyleLoss(target)
model.add_module(f"style_loss_{i}", style_loss)
style_losses.append(style_loss)

# Trim the model
for i in range(len(model) - 1, -1, -1):
if isinstance(model[i], (ContentLoss, StyleLoss)):
break
model = model[:i+1]

return model, style_losses, content_losses

# Input image (start from content image)
input_img = content_img.clone()

# Run style transfer
def run_style_transfer(cnn, content_img, style_img, input_img, num_steps=300,
style_weight=1e6, content_weight=1):
model, style_losses, content_losses = get_style_model_and_losses(
cnn, style_img, content_img
)
optimizer = optim.LBFGS([input_img.requires_grad_()])

print("Optimizing...")
run = [0]
while run[0] <= num_steps:
def closure():
input_img.data.clamp_(0, 1)
optimizer.zero_grad()
model(input_img)
style_score = sum(sl.loss for sl in style_losses)
content_score = sum(cl.loss for cl in content_losses)
loss = style_score * style_weight + content_score * content_weight
loss.backward()
run[0] += 1
if run[0] % 50 == 0:
print(f"Step {run[0]}, Style Loss: {style_score.item():.4f}, Content Loss: {content_score.item():.4f}")
return loss
```

```
optimizer.step(closure)

input_img.data.clamp_(0, 1)
return input_img

# Run the model
output = run_style_transfer(cnn, content_img, style_img, input_img)

# Show result
imshow(output, title="Styled Image")
```

 Folder Structure

```
neural_style_transfer/
    neural_style_transfer.py
    content.jpg # Your content photo
    style.jpg # Your style/art image
    output.png # Will be saved/generated
```