```python
# File: 05_StyleGAN2_Custom_Dataset_FineTuning.py
# Topic: Fine-tune StyleGAN2 on custom dataset using PyTorch

import os
import torch
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import transforms
from PIL import Image
import numpy as np
import dnnlib
import legacy
from tqdm import tqdm

# Configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
batch_size = 4
num_epochs = 50
learning_rate = 0.002
save_interval = 5
dataset_path = "./custom_dataset"  # Directory with your images
output_dir = "./stylegan2_finetune_results"
os.makedirs(output_dir, exist_ok=True)

# 1. Load Pre-trained StyleGAN2
print("Loading pre-trained StyleGAN2...")
network_pkl = "https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-
pytorch/pretrained/ffhq.pkl"
with dnnlib.util.open_url(network_pkl) as f:
    G = legacy.load_network_pkl(f)['G_ema'].to(device)

# 2. Prepare Custom Dataset
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, root, transform=None):
        self.root = root
        self.transform = transform
        self.image_files = [f for f in os.listdir(root) if
f.endswith(('.jpg', '.png', '.jpeg'))]

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        img_path = os.path.join(self.root, self.image_files[idx])
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image
```

```
transform = transforms.Compose([
    transforms.Resize((256, 256)),  # StyleGAN2 default size
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

dataset = CustomDataset(dataset_path, transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# 3. Setup for Fine-Tuning
# Freeze early layers (optional for better stability)
for name, param in G.named_parameters():
    if 'b4' not in name and 'b8' not in name and 'b16' not in name and
'b32' not in name and 'b64' not in name:
        param.requires_grad = False

optimizer = optim.Adam(
    filter(lambda p: p.requires_grad, G.parameters()),
    lr=learning_rate,
    betas=(0.0, 0.99)
)

# 4. Training Loop
print("Starting fine-tuning...")
for epoch in range(num_epochs):
    progress_bar = tqdm(dataloader, desc=f"Epoch
{epoch+1}/{num_epochs}")

    for i, real_images in enumerate(progress_bar):
        real_images = real_images.to(device)

        # Generate random latent vectors
        z = torch.randn(batch_size, G.z_dim).to(device)

        # Generate fake images
        fake_images = G(z, None, truncation_psi=0.7, noise_mode='const')

        # Compute loss (simple L1 loss for reconstruction)
        loss = torch.nn.functional.l1_loss(fake_images, real_images)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        progress_bar.set_postfix({"Loss": loss.item()})

    # Save checkpoint periodically
    if (epoch + 1) % save_interval == 0:
        torch.save({
            'epoch': epoch,
            'model_state_dict': G.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
```

```
            'loss': loss,
        }, os.path.join(output_dir,
f'stylegan2_ft_epoch_{epoch+1}.pth'))

        # Generate sample images
        with torch.no_grad():
            test_z = torch.randn(4, G.z_dim).to(device)
            sample_images = G(test_z, None, truncation_psi=0.7)
            sample_images = (sample_images.permute(0, 2, 3, 1) * 127.5 +
128).clamp(0, 255).to(torch.uint8)
            sample_images = sample_images.cpu().numpy()

            # Save sample images
            for j, img in enumerate(sample_images):
                Image.fromarray(img).save(os.path.join(output_dir,
f'epoch_{epoch+1}_sample_{j}.png'))

print("Fine-tuning complete!")

# 5. Generate New Samples
def generate_samples(num_samples=8, truncation_psi=0.7):
    """Generate samples from fine-tuned model"""
    G.eval()
    with torch.no_grad():
        z = torch.randn(num_samples, G.z_dim).to(device)
        samples = G(z, None, truncation_psi=truncation_psi)
        samples = (samples.permute(0, 2, 3, 1) * 127.5 + 128).clamp(0,
255).to(torch.uint8)
        return samples.cpu().numpy()

# Generate and save final samples
final_samples = generate_samples(16)
for i, sample in enumerate(final_samples):
    Image.fromarray(sample).save(os.path.join(output_dir,
f'final_sample_{i}.png'))
```

PROF

Key Components Explained:

1. **Pre-trained Model Loading**:
    - Uses NVIDIA's official StyleGAN2-ADA implementation
    - Loads FFHQ (1024x1024) as base model
2. **Custom Dataset Preparation**:
    - Processes images to 256x256 resolution
    - Normalizes to [-1, 1] range (StyleGAN2 standard)
    - Handles common image formats (JPG, PNG, JPEG)
3. **Fine-Tuning Strategy**:

- Freezes early layers (more stable training)
- Only trains higher-resolution layers (b4-b64)
- Uses L1 loss for reconstruction (can be changed to perceptual loss)

4. **Training Process**:
   - Saves checkpoints periodically
   - Generates sample images during training
   - Uses Adam optimizer with β2=0.99

Setup Instructions:

1. **Prerequisites**:

```
pip install torch torchvision numpy pillow tqdm requests
git clone https://github.com/NVlabs/stylegan2-ada-pytorch.git
cd stylegan2-ada-pytorch
```

2. **Dataset Preparation**:
   - Place your images in `./custom_dataset/`
   - Recommended: 1000+ images for good results
   - Images should be cropped and aligned (like FFHQ)

3. **Recommended Folder Structure**:

```
project/
├── stylegan2-ada-pytorch/  # Official repo
├── custom_dataset/
│   ├── image1.jpg
│   ├── image2.png
│   └── ...
└── 05_StyleGAN2_Custom_Dataset_FineTuning.py
```

PROF

Advanced Options:

1. **For Better Quality**:

```
# Replace L1 loss with perceptual loss
from lpips import LPIPS
percept_loss = LPIPS(net='vgg').to(device)

# In training loop:
loss = percept_loss(fake_images, real_images).mean()
```

2. **For Larger Datasets**:

```
# Use larger batch sizes if VRAM permits
batch_size = 8  # or higher

# Enable mixed precision training
scaler = torch.cuda.amp.GradScaler()
with torch.cuda.amp.autocast():
    fake_images = G(z, None)
    loss = percept_loss(fake_images, real_images)
scaler.scale(loss).backward()
scaler.step(optimizer)
scaler.update()
```

3. **For Different Resolutions**:

```
# Load config-e for 256x256
network_pkl = "https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-
pytorch/pretrained/cifar10.pkl"

# Or config-f for 1024x1024
network_pkl = "https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-
pytorch/pretrained/ffhq.pkl"
```

Expected Output:

```
stylegan2_finetune_results/
├── stylegan2_ft_epoch_5.pth
├── stylegan2_ft_epoch_10.pth
├── ...
├── epoch_5_sample_0.png
├── epoch_10_sample_0.png
├── ...
└── final_sample_0.png
```

PROF

Troubleshooting:

1. **Out of Memory Errors**:
   - Reduce batch size
   - Use gradient accumulation:

```
accumulation_steps = 4
loss = loss / accumulation_steps
loss.backward()
```

```
if (i + 1) % accumulation_steps == 0:
    optimizer.step()
    optimizer.zero_grad()
```

2. **Mode Collapse**:
    - Increase diversity penalty
    - Add noise to latent vectors:

```
z = z + 0.01 * torch.randn_like(z)
```

3. **Poor Quality**:
    - Try different learning rates (0.001-0.01)
    - Unfreeze more layers
    - Increase dataset size/variety