```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt


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


# Hyperparameters
latent_size = 100
image_size = 64
batch_size = 128
num_epochs = 10
lr = 0.0002
beta1 = 0.5


# Image transform
transform = transforms.Compose([
    transforms.Resize(image_size),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])


# Load dataset
dataset = datasets.FashionMNIST(root='./data', train=True, transform=transform, download=True)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
100%|██████████| 26.4M/26.4M [00:02<00:00, 13.1MB/s]
100%|██████████| 29.5k/29.5k [00:00<00:00, 211kB/s]
100%|██████████| 4.42M/4.42M [00:01<00:00, 3.87MB/s]
100%|██████████| 5.15k/5.15k [00:00<00:00, 13.6MB/s]
```

```
# Generator
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(latent_size, 512, 4, 1, 0, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(True),

            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),

            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.ConvTranspose2d(64, 1, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, x):
        return self.main(x)


# Discriminator
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(1, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
```

```
            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.main(x).view(-1)


# Create models
G = Generator().to(device)
D = Discriminator().to(device)


# Loss and optimizers
criterion = nn.BCELoss()
optimizerG = optim.Adam(G.parameters(), lr=lr, betas=(beta1, 0.999))
optimizerD = optim.Adam(D.parameters(), lr=lr, betas=(beta1, 0.999))


# Fixed noise for final image generation
fixed_noise = torch.randn(64, latent_size, 1, 1, device=device)


# Training Loop
for epoch in range(1, num_epochs + 1):
    for i, (real_images, _) in enumerate(dataloader):
        real_images = real_images.to(device)
        batch_size = real_images.size(0)

        # Labels
        real_labels = torch.ones(batch_size, device=device)
        fake_labels = torch.zeros(batch_size, device=device)

        # Train Discriminator
        noise = torch.randn(batch_size, latent_size, 1, 1, device=device)
        fake_images = G(noise)
        outputs_real = D(real_images)
        outputs_fake = D(fake_images.detach())
        loss_real = criterion(outputs_real, real_labels)
        loss_fake = criterion(outputs_fake, fake_labels)
        loss_D = loss_real + loss_fake

        optimizerD.zero_grad()
        loss_D.backward()
        optimizerD.step()

        # Train Generator
        outputs = D(fake_images)
        loss_G = criterion(outputs, real_labels)
        optimizerG.zero_grad()
        loss_G.backward()
        optimizerG.step()

    print(f"Epoch [{epoch}/{num_epochs}]  Loss D: {loss_D.item():.4f}, Loss G: {loss_G.item():.4f}")
```

```
Epoch [1/10]  Loss D: 0.8698, Loss G: 2.8287
Epoch [2/10]  Loss D: 0.4581, Loss G: 3.8374
Epoch [3/10]  Loss D: 0.2553, Loss G: 4.0326
Epoch [4/10]  Loss D: 0.3072, Loss G: 4.2416
Epoch [5/10]  Loss D: 0.2609, Loss G: 5.2423
Epoch [6/10]  Loss D: 0.0473, Loss G: 4.7672
Epoch [7/10]  Loss D: 0.0679, Loss G: 3.9586
Epoch [8/10]  Loss D: 0.5269, Loss G: 3.6963
Epoch [9/10]  Loss D: 0.0429, Loss G: 4.8497
Epoch [10/10]  Loss D: 1.3891, Loss G: 3.4605
```

```
# Final plotting helper
def plot_fake_images(epoch, G, fixed_noise):
    G.eval()
    with torch.no_grad():
        fake_images = G(fixed_noise).detach().cpu()
    fake_images = fake_images * 0.5 + 0.5  # denormalize
    fig, ax = plt.subplots(8, 8, figsize=(8, 8))
    for i in range(64):
        ax[i//8, i%8].imshow(fake_images[i][0], cmap='gray')
        ax[i//8, i%8].axis('off')
    plt.suptitle(f"Generated Images after Epoch {epoch}")
    plt.tight_layout()
```

```
    plt.show()

# Plot only once after training
plot_fake_images(num_epochs, G, fixed_noise)
```

Start coding or generate with AI.