In [1]:

```python
# ======================================================
# 1. IMPORT LIBRARIES
# ======================================================
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, utils
from torch.utils.data import DataLoader
import os
import numpy as np
from collections import Counter

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# ======================================================
# 2. USER INPUT PARAMETERS
# ======================================================
dataset_choice = 'mnist'        # 'mnist' or 'fashion'
epochs = 30
batch_size = 128
noise_dim = 100
lr_G = 0.0002
lr_D = 0.0001
save_interval = 5

# ======================================================
# 3. DATASET LOADING
# ======================================================
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

if dataset_choice == 'mnist':
    dataset = datasets.MNIST('./data', train=True, download=True, transform=tr
elif dataset_choice == 'fashion':
    dataset = datasets.FashionMNIST('./data', train=True, download=True, trans
else:
    raise ValueError("Invalid dataset choice")

dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
img_shape = (1, 28, 28)

# ======================================================
# 4. OUTPUT FOLDERS
# ======================================================
os.makedirs("generated_samples", exist_ok=True)
os.makedirs("final_generated_images", exist_ok=True)

# ======================================================
# 5. GENERATOR
# ======================================================
```

```python
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(noise_dim, 256),
            nn.ReLU(True),
            nn.Linear(256, 512),
            nn.ReLU(True),
            nn.Linear(512, 1024),
            nn.ReLU(True),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        return img.view(img.size(0), *img_shape)

# ========================================================
# 6. DISCRIMINATOR
# ========================================================
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(int(np.prod(img_shape)), 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, img):
        img = img.view(img.size(0), -1)
        return self.model(img)

G = Generator().to(device)
D = Discriminator().to(device)

# ========================================================
# 7. LOSS & OPTIMIZERS
# ========================================================
criterion = nn.BCELoss()
optimizer_G = optim.Adam(G.parameters(), lr=lr_G, betas=(0.5, 0.999))
optimizer_D = optim.Adam(D.parameters(), lr=lr_D, betas=(0.5, 0.999))

# ========================================================
# 8. TRAINING LOOP
# ========================================================
for epoch in range(1, epochs + 1):
    D_loss_total, G_loss_total = 0.0, 0.0
    correct, total = 0, 0
```

```python
    for real_imgs, _ in dataloader:
        real_imgs = real_imgs.to(device)
        batch = real_imgs.size(0)

        # Label smoothing
        real_labels = torch.full((batch, 1), 0.9).to(device)
        fake_labels = torch.zeros(batch, 1).to(device)

        # --------------------
        # Train Discriminator
        # --------------------
        optimizer_D.zero_grad()

        real_loss = criterion(D(real_imgs), real_labels)

        z = torch.randn(batch, noise_dim).to(device)
        fake_imgs = G(z)
        fake_loss = criterion(D(fake_imgs.detach()), fake_labels)

        D_loss = real_loss + fake_loss
        D_loss.backward()
        optimizer_D.step()

        # Accuracy
        preds_real = (D(real_imgs) > 0.5).float()
        preds_fake = (D(fake_imgs.detach()) < 0.5).float()
        correct += preds_real.sum().item() + preds_fake.sum().item()
        total += batch * 2

        # --------------------
        # Train Generator (TWICE, FIXED)
        # --------------------
        for _ in range(2):
            optimizer_G.zero_grad()

            z = torch.randn(batch, noise_dim).to(device)   # NEW noise
            fake_imgs = G(z)                               # NEW graph

            G_loss = criterion(D(fake_imgs), real_labels)
            G_loss.backward()
            optimizer_G.step()

        D_loss_total += D_loss.item()
        G_loss_total += G_loss.item()

    D_acc = (correct / total) * 100

    print(f"Epoch {epoch}/{epochs} | "
          f"D_loss: {D_loss_total/len(dataloader):.3f} | "
          f"D_acc: {D_acc:.2f}% | "
          f"G_loss: {G_loss_total/len(dataloader):.3f}")
```

```python
    # Save generated samples
    if epoch % save_interval == 0:
        utils.save_image(fake_imgs[:25],
                         f"generated_samples/epoch_{epoch:02d}.png",
                         nrow=5,
                         normalize=True)

# ======================================================
# 9. GENERATE FINAL 100 IMAGES
# ======================================================
z = torch.randn(100, noise_dim).to(device)
final_images = G(z)

for i in range(100):
    utils.save_image(final_images[i],
                     f"final_generated_images/img_{i}.png",
                     normalize=True)

# ======================================================
# 10. SIMPLE CLASSIFIER
# ======================================================
class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

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

classifier = Classifier().to(device)
optimizer_C = optim.Adam(classifier.parameters(), lr=0.001)
loss_fn = nn.CrossEntropyLoss()

for epoch in range(3):
    for imgs, labels in dataloader:
        imgs, labels = imgs.to(device), labels.to(device)
        optimizer_C.zero_grad()
        loss = loss_fn(classifier(imgs), labels)
        loss.backward()
        optimizer_C.step()

# ======================================================
# 11. LABEL PREDICTION
# ======================================================
with torch.no_grad():
    preds = classifier(final_images).argmax(dim=1).cpu().numpy()

label_counts = Counter(preds)
```

```python
print("\nLabel Distribution of Generated Images:")
for label, count in sorted(label_counts.items()):
    print(f"Label {label}: {count}")
```

```
Using device: cuda
Epoch 1/30 | D_loss: 1.413 | D_acc: 52.23% | G_loss: 0.688
Epoch 2/30 | D_loss: 1.389 | D_acc: 51.30% | G_loss: 0.736
Epoch 3/30 | D_loss: 1.385 | D_acc: 56.16% | G_loss: 0.748
Epoch 4/30 | D_loss: 1.383 | D_acc: 60.36% | G_loss: 0.765
Epoch 5/30 | D_loss: 1.379 | D_acc: 62.42% | G_loss: 0.775
Epoch 6/30 | D_loss: 1.366 | D_acc: 65.51% | G_loss: 0.793
Epoch 7/30 | D_loss: 1.362 | D_acc: 66.95% | G_loss: 0.803
Epoch 8/30 | D_loss: 1.368 | D_acc: 67.15% | G_loss: 0.814
Epoch 9/30 | D_loss: 1.342 | D_acc: 69.60% | G_loss: 0.835
Epoch 10/30 | D_loss: 1.347 | D_acc: 68.41% | G_loss: 0.826
Epoch 11/30 | D_loss: 1.330 | D_acc: 69.90% | G_loss: 0.875
Epoch 12/30 | D_loss: 1.320 | D_acc: 72.01% | G_loss: 0.859
Epoch 13/30 | D_loss: 1.286 | D_acc: 73.76% | G_loss: 0.926
Epoch 14/30 | D_loss: 1.283 | D_acc: 73.89% | G_loss: 0.933
Epoch 15/30 | D_loss: 1.241 | D_acc: 76.87% | G_loss: 1.024
Epoch 16/30 | D_loss: 1.235 | D_acc: 77.65% | G_loss: 1.012
Epoch 17/30 | D_loss: 1.179 | D_acc: 79.90% | G_loss: 1.127
Epoch 18/30 | D_loss: 1.194 | D_acc: 79.81% | G_loss: 1.143
Epoch 19/30 | D_loss: 1.107 | D_acc: 81.61% | G_loss: 1.210
Epoch 20/30 | D_loss: 0.879 | D_acc: 87.78% | G_loss: 1.661
Epoch 21/30 | D_loss: 0.627 | D_acc: 94.55% | G_loss: 2.583
Epoch 22/30 | D_loss: 0.539 | D_acc: 96.02% | G_loss: 3.008
Epoch 23/30 | D_loss: 0.486 | D_acc: 97.27% | G_loss: 3.118
Epoch 24/30 | D_loss: 0.394 | D_acc: 98.91% | G_loss: 4.793
Epoch 25/30 | D_loss: 0.396 | D_acc: 99.08% | G_loss: 3.933
Epoch 26/30 | D_loss: 0.367 | D_acc: 99.56% | G_loss: 4.902
Epoch 27/30 | D_loss: 0.345 | D_acc: 99.80% | G_loss: 6.876
Epoch 28/30 | D_loss: 0.330 | D_acc: 100.00% | G_loss: 8.318
Epoch 29/30 | D_loss: 0.361 | D_acc: 99.40% | G_loss: 7.785
Epoch 30/30 | D_loss: 0.356 | D_acc: 99.53% | G_loss: 5.916

Label Distribution of Generated Images:
Label 0: 100
```