```
print("E23CSEU2320")
print("LAB2")
```

```
E23CSEU2320
LAB2
```

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torchvision.utils import save_image
from collections import Counter

# ================= CONFIGURATION =================
dataset_choice = "fashion"        # CHANGED
epochs = 80                       # MORE EPOCHS
batch_size = 128
noise_dim = 100
save_interval = 5

device = torch.device("cuda" if torch.cuda.is_available() el

os.makedirs("generated_samples", exist_ok=True)
os.makedirs("final_generated_images", exist_ok=True)

# ================= DATASET =================
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])   # [-1, 1]
])

dataset = datasets.FashionMNIST(
    "./data", train=True, transform=transform, download=True
)

loader = torch.utils.data.DataLoader(
    dataset, batch_size=batch_size, shuffle=True
)

# ================= GENERATOR =================
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
```

```
            nn.Linear(noise_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 28*28),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.net(z)
        return img.view(z.size(0), 1, 28, 28)

# ================ DISCRIMINATOR ================
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(28*28, 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.net(img)

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

criterion = nn.BCELoss()

# 🔑 DIFFERENT LEARNING RATES (CRITICAL)
optimizer_G = optim.Adam(G.parameters(), lr=0.0002)
optimizer_D = optim.Adam(D.parameters(), lr=0.0001)

# ================ TRAIN GAN ================
for epoch in range(1, epochs + 1):
    correct = 0
    total = 0

    for imgs, _ in loader:
        imgs = imgs.to(device)
```

```
            imgs = imgs.to(device)
            batch = 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_out = D(imgs)
            real_loss = criterion(real_out, real_labels)

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

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

            preds = torch.cat([real_out, fake_out])
            labels = torch.cat([real_labels, fake_labels])
            correct += (preds.round() == labels).sum().item()
            total += labels.size(0)

            # ---- Train Generator ----
            optimizer_G.zero_grad()
            g_loss = criterion(D(fake_imgs), real_labels)
            g_loss.backward()
            optimizer_G.step()

        d_acc = 100 * correct / total

        print(f"Epoch {epoch}/{epochs} | "
              f"D_loss: {d_loss.item():.2f} | "
              f"D_acc: {d_acc:.2f}% | "
              f"G_loss: {g_loss.item():.2f}")

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

```
# ================ FINAL 100 IMAGES ================
z = torch.randn(100, noise_dim).to(device)
final_imgs = G(z)

for i in range(100):
    save_image(
        final_imgs[i],
        f"final_generated_images/img_{i+1}.png",
        normalize=True
    )

# ================ CLASSIFIER ================
class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28*28, 256),
            nn.ReLU(),
            nn.Linear(256, 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()

# Train classifier on REAL data
for _ in range(5):
    for imgs, labels in loader:
        imgs, labels = imgs.to(device), labels.to(device)
        optimizer_C.zero_grad()
        outputs = classifier(imgs)
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer_C.step()

# Evaluate generated images
classifier.eval()
with torch.no_grad():
    outputs = classifier(final_imgs)
    preds = torch.argmax(outputs, dim=1)

counts = Counter(preds.cpu().numpy())
```

```
print("\nLabel Distribution of Generated Images:")
for label, count in counts.items():
    print(f"Class {label}: {count}")
```

```
Epoch 35/80 | D_loss: 1.27 | D_acc: 44.24% | G_loss: 0.95
Epoch 36/80 | D_loss: 1.41 | D_acc: 45.07% | G_loss: 1.17
Epoch 37/80 | D_loss: 1.46 | D_acc: 43.19% | G_loss: 0.97
Epoch 38/80 | D_loss: 1.28 | D_acc: 44.58% | G_loss: 1.23
Epoch 39/80 | D_loss: 1.28 | D_acc: 43.70% | G_loss: 1.03
Epoch 40/80 | D_loss: 1.17 | D_acc: 43.84% | G_loss: 1.40
Epoch 41/80 | D_loss: 1.23 | D_acc: 43.47% | G_loss: 1.03
Epoch 42/80 | D_loss: 1.25 | D_acc: 44.24% | G_loss: 1.09
Epoch 43/80 | D_loss: 1.17 | D_acc: 42.34% | G_loss: 1.00
Epoch 44/80 | D_loss: 1.34 | D_acc: 43.38% | G_loss: 1.37
Epoch 45/80 | D_loss: 1.22 | D_acc: 42.47% | G_loss: 1.00
```

```
Class 7: 18
Class 6: 15
Class 2: 3
Class 4: 19
Class 5: 10
Class 9: 6
Class 3: 13
Class 8: 4
Class 0: 2
```

Start coding or generate with AI.