```
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

dataset_choice = "mnist"
epochs = 50
batch_size = 128
noise_dim = 100
learning_rate = 0.0002
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)

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

if dataset_choice == "mnist":
    dataset = datasets.MNIST("./data", train=True, transform
else:
    dataset = datasets.FashionMNIST("./data", train=True, tr

loader = torch.utils.data.DataLoader(dataset, batch_size=bat

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)

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()
optimizer_G = optim.Adam(G.parameters(), lr=learning_rate)
optimizer_D = optim.Adam(D.parameters(), lr=learning_rate)

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

    for imgs, _ in loader:
        imgs = imgs.to(device)
        batch = imgs.size(0)

        real_labels = torch.ones(batch, 1).to(device)
        fake_labels = torch.zeros(batch, 1).to(device)

        # ---- Train Discriminator ----
        #      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()

        # Accuracy of discriminator
        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} | D_loss: {d_loss.item():
          f"D_acc: {d_acc:.2f}% | 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):
```

```
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}")
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 18.6MB/s]
100%|██████████| 28.9k/28.9k [00:00<00:00, 493kB/s]
100%|██████████| 1.65M/1.65M [00:00<00:00, 4.62MB/s]
100%|██████████| 4.54k/4.54k [00:00<00:00, 10.4MB/s]
Epoch 1/50 | D_loss: 0.12 | D_acc: 89.28% | G_loss: 5.78
Epoch 2/50 | D_loss: 0.06 | D_acc: 98.69% | G_loss: 9.07
Epoch 3/50 | D_loss: 0.08 | D_acc: 92.45% | G_loss: 5.39
Epoch 4/50 | D_loss: 0.77 | D_acc: 82.74% | G_loss: 2.76
Epoch 5/50 | D_loss: 0.53 | D_acc: 79.32% | G_loss: 2.05
Epoch 6/50 | D_loss: 0.52 | D_acc: 76.52% | G_loss: 3.83
Epoch 7/50 | D_loss: 0.54 | D_acc: 83.20% | G_loss: 2.41
Epoch 8/50 | D_loss: 0.82 | D_acc: 85.40% | G_loss: 1.62
Epoch 9/50 | D_loss: 0.68 | D_acc: 82.27% | G_loss: 2.23
Epoch 10/50 | D_loss: 2.26 | D_acc: 85.12% | G_loss: 1.56
Epoch 11/50 | D_loss: 0.59 | D_acc: 82.64% | G_loss: 2.06
Epoch 12/50 | D_loss: 0.44 | D_acc: 78.25% | G_loss: 2.44
Epoch 13/50 | D_loss: 0.69 | D_acc: 82.31% | G_loss: 2.52
Epoch 14/50 | D_loss: 0.58 | D_acc: 86.61% | G_loss: 2.34
Epoch 15/50 | D_loss: 0.28 | D_acc: 90.00% | G_loss: 3.23
Epoch 16/50 | D_loss: 0.46 | D_acc: 92.47% | G_loss: 4.60
Epoch 17/50 | D_loss: 0.56 | D_acc: 89.68% | G_loss: 3.17
Epoch 18/50 | D_loss: 0.28 | D_acc: 90.60% | G_loss: 3.90
Epoch 19/50 | D_loss: 1.20 | D_acc: 90.53% | G_loss: 2.66
Epoch 20/50 | D_loss: 0.49 | D_acc: 94.78% | G_loss: 9.31
Epoch 21/50 | D_loss: 0.49 | D_acc: 92.20% | G_loss: 7.49
Epoch 22/50 | D_loss: 0.17 | D_acc: 96.47% | G_loss: 6.62
Epoch 23/50 | D_loss: 0.20 | D_acc: 96.35% | G_loss: 3.84
Epoch 24/50 | D_loss: 0.77 | D_acc: 94.72% | G_loss: 4.71
Epoch 25/50 | D_loss: 0.11 | D_acc: 91.56% | G_loss: 6.49
Epoch 26/50 | D_loss: 0.10 | D_acc: 96.92% | G_loss: 5.48
Epoch 27/50 | D_loss: 0.09 | D_acc: 98.76% | G_loss: 10.35
Epoch 28/50 | D_loss: 0.27 | D_acc: 97.73% | G_loss: 7.51
Epoch 29/50 | D_loss: 0.11 | D_acc: 97.69% | G_loss: 5.89
Epoch 30/50 | D_loss: 0.22 | D_acc: 95.08% | G_loss: 5.61
Epoch 31/50 | D_loss: 0.43 | D_acc: 95.40% | G_loss: 4.38
Epoch 32/50 | D_loss: 0.32 | D_acc: 94.02% | G_loss: 5.16
Epoch 33/50 | D_loss: 0.16 | D_acc: 94.56% | G_loss: 6.68
Epoch 34/50 | D_loss: 0.30 | D_acc: 95.18% | G_loss: 3.92
Epoch 35/50 | D_loss: 0.35 | D_acc: 92.51% | G_loss: 5.60
Epoch 36/50 | D_loss: 0.63 | D_acc: 92.73% | G_loss: 3.67
Epoch 37/50 | D_loss: 0.45 | D_acc: 92.58% | G_loss: 4.00
Epoch 38/50 | D_loss: 0.34 | D_acc: 92.41% | G_loss: 4.28
Epoch 39/50 | D_loss: 0.37 | D_acc: 94.09% | G_loss: 4.08
Epoch 40/50 | D_loss: 0.42 | D_acc: 94.62% | G_loss: 3.81
Epoch 41/50 | D_loss: 0.35 | D_acc: 93.21% | G_loss: 4.40
Epoch 42/50 | D_loss: 0.44 | D_acc: 92.08% | G_loss: 4.15
Epoch 43/50 | D_loss: 0.57 | D_acc: 91.89% | G_loss: 3.42
Epoch 44/50 | D_loss: 0.52 | D_acc: 89.90% | G_loss: 5.26
Epoch 45/50 | D_loss: 0.55 | D_acc: 91.84% | G_loss: 4.38
Epoch 46/50 | D_loss: 0.67 | D_acc: 92.43% | G_loss: 2.94
Epoch 47/50 | D_loss: 0.40 | D_acc: 91.43% | G_loss: 3.93
Epoch 48/50 | D_loss: 0.59 | D_acc: 92.22% | G_loss: 3.97
Epoch 49/50 | D_loss: 0.66 | D_acc: 90.69% | G_loss: 3.17
Epoch 50/50 | D_loss: 0.57 | D_acc: 91.13% | G_loss: 3.06

Label Distribution of Generated Images:
Class 6: 10
```

Start coding or generate with AI.