GANs for Image Generation

Objective

To train a Generative Adversarial Network (GAN) capable of generating new, realistic images that resemble images from a specific dataset (either CelebA faces or Fashion MNIST clothing items).

Dataset Used:

Fashion MNIST: A dataset of grayscale images representing 10 categories of clothing and footwear articles.

- **Source**: Zalando Research; available on Kaggle, TensorFlow Datasets, and other major machine learning libraries.
- **Description**: Contains 70,000 grayscale images of fashion products from 10 categories, each of size 28x28 pixels.
  - Training Set: 60,000 images.
  - Test Set: 10,000 images.
- **Features**:
  - 784 pixel values (28x28 flattened) representing pixel intensities (0–255) for each grayscale image.
- **Classes**: 10 (T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot)

Theory:

Generative Adversarial Networks (GANs) employ a game-theoretic approach to generative modeling. They consist of two core neural networks trained simultaneously in competition:

1. **The Generator (G):** This network's goal is to create synthetic data (in this case, images) that is indistinguishable from real data. It typically takes a random noise vector (latent variable) as input and transforms it through several layers (often using upsampling or transposed convolutions) to produce an image of the desired dimensions. The generator learns by receiving feedback from the discriminator. Its objective is to produce outputs that the discriminator incorrectly classifies as 'real'.

2. **The Discriminator (D):** This network acts as a binary classifier. Its goal is to distinguish between real images (coming from the actual dataset) and fake images (produced by the generator). It takes an image as input and outputs a probability indicating whether the image is real (probability close to 1) or fake (probability close to 0). It is trained on both real examples (labeled as 'real') and fake examples generated by G (labeled as 'fake').

Training Dynamics:

The training process involves an adversarial game:

- The discriminator is trained to improve its ability to correctly classify real and fake images. Its loss function penalizes misclassifications.

- The generator is trained to fool the discriminator. Its loss function penalizes the discriminator *correctly* identifying its output as fake. Effectively, the generator aims to maximize the discriminator's classification error on fake images.

This competition drives both networks to improve. As the discriminator gets better at spotting fakes, the generator must produce increasingly realistic images to fool it. The ideal outcome is reaching a Nash equilibrium where the generator produces perfect replicas of the data distribution, and the discriminator can only guess randomly (outputting 0.5 probability).

Key Monitoring Aspects:

- **Loss Functions:** Monitoring the loss of both the generator and discriminator is crucial. Typically, Binary Cross-Entropy loss is used. Ideally, the losses should stabilize, indicating a balance in training. If one loss drops to zero while the other explodes, it often signifies a problem like the generator overpowering the discriminator or vice-versa (or mode collapse, where the generator only produces a limited variety of outputs).

- **Generated Images:** Periodically visualizing the images produced by the generator from fixed noise vectors provides a qualitative assessment of learning progress. Early in training, images will look like noise, but over time, they should gain structure and realism resembling the training dataset.

Code:

```
import torch
import torchvision
from torch import nn
from torch import optim
from torchvision.datasets import FashionMNIST
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm
# Set random seed for reproducibility
torch.manual_seed(42)
# Define hyperparameters and variables
LEARNING_RATE = 0.0005
BATCH_SIZE = 1024
IMAGE_SIZE = 64
EPOCHS = 50
image_channels = 1
noise_channels = 256
gen_features = 64
disc_features = 64
# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define the transform
data_transforms = transforms.Compose([
    transforms.Resize(IMAGE_SIZE),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)),
])
```

```
# Load the dataset
dataset = FashionMNIST(root="dataset/", train=True, transform=data_transforms,
download=True)
dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=True,
num_workers=2)

# Generator
class Generator(nn.Module):
    def __init__(self, noise_channels, image_channels, features):
        super(Generator, self).__init__()
        """
        The generator model uses 4 ConvTranspose blocks. Each block contains
        a ConvTranspose2d, BatchNorm2d, and ReLU activation.
        """
        self.model = nn.Sequential(
            # Transpose block 1
            nn.ConvTranspose2d(noise_channels, features*16, kernel_size=4,
stride=1, padding=0),
            nn.ReLU(),
            # Transpose block 2
            nn.ConvTranspose2d(features*16, features*8, kernel_size=4,
stride=2, padding=1),
            nn.BatchNorm2d(features*8),
            nn.ReLU(),
            # Transpose block 3
            nn.ConvTranspose2d(features*8, features*4, kernel_size=4,
stride=2, padding=1),
            nn.BatchNorm2d(features*4),
            nn.ReLU(),
            # Transpose block 4
            nn.ConvTranspose2d(features*4, features*2, kernel_size=4,
stride=2, padding=1),
            nn.BatchNorm2d(features*2),
            nn.ReLU(),
            # Last transpose block
            nn.ConvTranspose2d(features*2, image_channels, kernel_size=4,
stride=2, padding=1),
            nn.Tanh(),
        )

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

# Discriminator
class Discriminator(nn.Module):
    def __init__(self, image_channels, features):
        super(Discriminator, self).__init__()
        """
```

```
        The discriminator model has 5 Conv blocks with Conv2d, BatchNorm, and
LeakyReLU activation.
        """
        self.model = nn.Sequential(
            # Conv block 1
            nn.Conv2d(image_channels, features, kernel_size=4, stride=2,
padding=1),  # 64x64 -> 32x32
            nn.LeakyReLU(0.2),
            # Conv block 2
            nn.Conv2d(features, features*2, kernel_size=4, stride=2,
padding=1),  # 32x32 -> 16x16
            nn.BatchNorm2d(features*2),
            nn.LeakyReLU(0.2),
            # Conv block 3
            nn.Conv2d(features*2, features*4, kernel_size=4, stride=2,
padding=1),  # 16x16 -> 8x8
            nn.BatchNorm2d(features*4),
            nn.LeakyReLU(0.2),
            # Conv block 4
            nn.Conv2d(features*4, features*8, kernel_size=4, stride=2,
padding=1),  # 8x8 -> 4x4
            nn.BatchNorm2d(features*8),
            nn.LeakyReLU(0.2),
            # Conv block 5
            nn.Conv2d(features*8, 1, kernel_size=4, stride=1, padding=0),  #
4x4 -> 1x1
            nn.Sigmoid(),
        )

    def forward(self, x):
        output = self.model(x)
        return output.view(-1)  # Flatten to [batch_size]

# Load models
gen_model = Generator(noise_channels, image_channels, gen_features).to(device)
disc_model = Discriminator(image_channels, disc_features).to(device)

# Setup optimizers
gen_optimizer = optim.Adam(gen_model.parameters(), lr=LEARNING_RATE,
betas=(0.5, 0.999))
disc_optimizer = optim.Adam(disc_model.parameters(), lr=LEARNING_RATE,
betas=(0.5, 0.999))

# Define loss function
criterion = nn.BCELoss()

# Set models to training mode
gen_model.train()
```

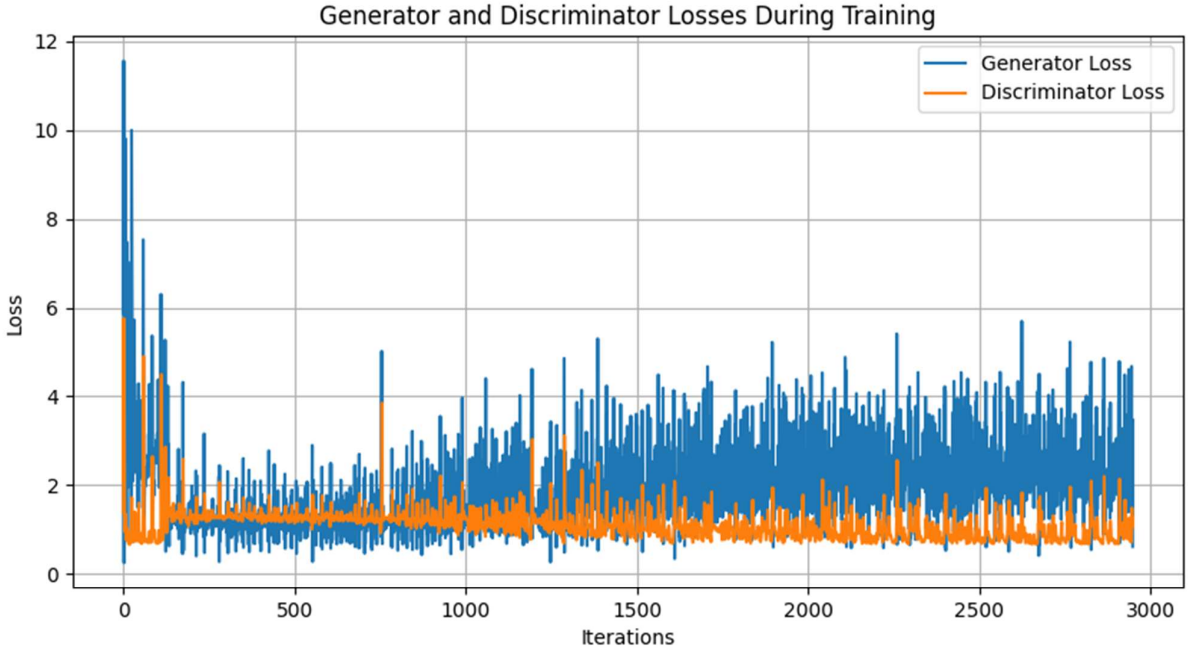
```
disc_model.train()

# Define labels with smoothing
fake_label = 0.1
real_label = 0.9

# Define fixed noise for consistent image generation
fixed_noise = torch.randn(64, noise_channels, 1, 1).to(device)

# TensorBoard writers
writer_real = SummaryWriter(f"runs/fashion/test_real")
writer_fake = SummaryWriter(f"runs/fashion/test_fake")

# Loss tracking
gen_losses = []
disc_losses = []

# Training loop
print("Start training...")
step = 0

for epoch in range(EPOCHS):
    for batch_idx, (data, target) in enumerate(tqdm(dataloader, desc=f"Epoch
{epoch}")):
        data = data.to(device)
        batch_size = data.shape[0]

        # Train discriminator on real data
        disc_model.zero_grad()
        label = (torch.ones(batch_size) * real_label).to(device)
        output = disc_model(data).reshape(-1)
        real_disc_loss = criterion(output, label)
        d_x = output.mean().item()

        # Train discriminator on fake data
        noise = torch.randn(batch_size, noise_channels, 1, 1, device=device)
        fake = gen_model(noise)
        label = (torch.ones(batch_size) * fake_label).to(device)
        output = disc_model(fake.detach()).reshape(-1)
        fake_disc_loss = criterion(output, label)

        # Calculate discriminator loss
        disc_loss = real_disc_loss + fake_disc_loss
        disc_loss.backward()
        disc_optimizer.step()

        # Train generator
        gen_model.zero_grad()
```

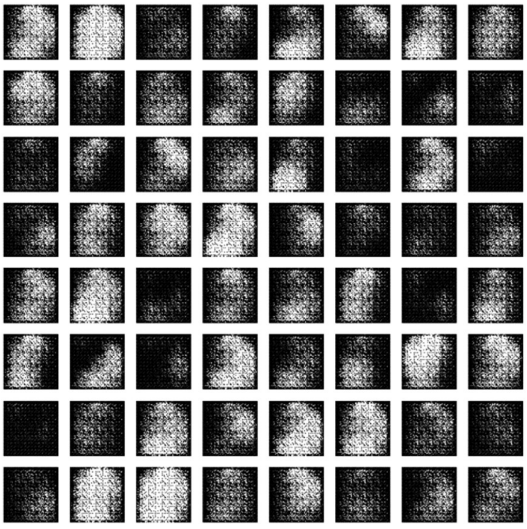
```
        label = torch.ones(batch_size).to(device)
        output = disc_model(fake).reshape(-1)
        gen_loss = criterion(output, label)
        gen_loss.backward()
        gen_optimizer.step()
        # Store losses
        gen_losses.append(gen_loss.item())
        disc_losses.append(disc_loss.item())
        # Log losses and images
        if batch_idx % 50 == 0:
            step += 1
            print(
                f"Epoch: {epoch} ===== Batch: {batch_idx}/{len(dataloader)}
===== "
                f"Disc loss: {disc_loss:.4f} ===== Gen loss: {gen_loss:.4f}"
            )
            with torch.no_grad():
                fake_images = gen_model(fixed_noise)
                img_grid_real = torchvision.utils.make_grid(data[:40],
normalize=True)
                img_grid_fake = torchvision.utils.make_grid(fake_images[:40],
normalize=True)
                writer_real.add_image("Real images", img_grid_real,
global_step=step)
                writer_fake.add_image("Generated images", img_grid_fake,
global_step=step)
    # Save generated images every 10 epochs or at epoch 1
    if (epoch + 1) % 10 == 0 or epoch == 0:
        with torch.no_grad():
            fake_images = gen_model(fixed_noise).detach().cpu()
        plt.figure(figsize=(10, 10))
        for j in range(64):
            plt.subplot(8, 8, j+1)
            plt.imshow(fake_images[j].squeeze(), cmap='gray')
            plt.axis('off')
        plt.savefig(f'generated_images_epoch_{epoch+1}.png')
        plt.close()
# Plot losses
plt.figure(figsize=(10, 5))
plt.plot(gen_losses, label='Generator Loss')
plt.plot(disc_losses, label='Discriminator Loss')
plt.title('Generator and Discriminator Losses During Training')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig('loss_plot.png')
plt.close()
```

Output:

```
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-images-idx3-ubyte.gz to
dataset/FashionMNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 26.4M/26.4M [00:03<00:00, 7.61MB/s]
Extracting dataset/FashionMNIST/raw/train-images-idx3-ubyte.gz to
dataset/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/train-labels-idx1-ubyte.gz to
dataset/FashionMNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 29.5k/29.5k [00:00<00:00, 139kB/s]
Extracting dataset/FashionMNIST/raw/train-labels-idx1-ubyte.gz to
dataset/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-images-idx3-ubyte.gz to dataset/FashionMNIST/raw/t10k-
images-idx3-ubyte.gz
100%|██████████| 4.42M/4.42M [00:01<00:00, 2.51MB/s]
Extracting dataset/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to
dataset/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/t10k-labels-idx1-ubyte.gz to dataset/FashionMNIST/raw/t10k-
labels-idx1-ubyte.gz
100%|██████████| 5.15k/5.15k [00:00<00:00, 12.4MB/s]
Extracting dataset/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to
dataset/FashionMNIST/raw

Start training...
Epoch 0:   0%|          | 0/59 [00:00<?, ?it/s]
Epoch: 0 ===== Batch: 0/59 ===== Disc loss: 1.3884 ===== Gen loss: 5.8818
Epoch 0:  86%|████████▌ | 51/59 [01:59<00:20,  2.58s/it]
Epoch: 0 ===== Batch: 50/59 ===== Disc loss: 0.7427 ===== Gen loss: 2.7016
Epoch 0: 100%|██████████| 59/59 [02:18<00:00,  2.35s/it]
Epoch 1:   2%|▏         | 1/59 [00:03<02:54,  3.01s/it]
Epoch: 1 ===== Batch: 0/59 ===== Disc loss: 4.9010 ===== Gen loss: 3.2427
Epoch 1:  86%|████████▌ | 51/59 [02:07<00:20,  2.54s/it]
Epoch: 1 ===== Batch: 50/59 ===== Disc loss: 0.7456 ===== Gen loss: 2.1578
Epoch 1: 100%|██████████| 59/59 [02:26<00:00,  2.49s/it]
Epoch 2:   2%|▏         | 1/59 [00:03<03:00,  3.12s/it]
Epoch: 2 ===== Batch: 0/59 ===== Disc loss: 1.0038 ===== Gen loss: 3.7746
Epoch 2:  86%|████████▌ | 51/59 [02:08<00:20,  2.55s/it]
Epoch: 2 ===== Batch: 50/59 ===== Disc loss: 1.1447 ===== Gen loss: 1.4508
Epoch 2: 100%|██████████| 59/59 [02:27<00:00,  2.51s/it]
Epoch 3:   2%|▏         | 1/59 [00:03<02:55,  3.02s/it]
Epoch: 3 ===== Batch: 0/59 ===== Disc loss: 1.4702 ===== Gen loss: 1.0909
Epoch 3:  86%|████████▌ | 51/59 [02:08<00:20,  2.56s/it]
Epoch: 3 ===== Batch: 50/59 ===== Disc loss: 1.2188 ===== Gen loss: 1.1054
Epoch 3: 100%|██████████| 59/59 [02:27<00:00,  2.50s/it]
…
Epoch 47:   2%|▏         | 1/59 [00:03<02:55,  3.02s/it]
```

```
Epoch: 47 ===== Batch: 0/59 ===== Disc loss: 1.0231 ===== Gen loss: 1.4087
Epoch 47:  86%|████████▌ | 51/59 [02:08<00:20,  2.54s/it]
Epoch: 47 ===== Batch: 50/59 ===== Disc loss: 0.7345 ===== Gen loss: 2.9863
Epoch 47: 100%|██████████| 59/59 [02:27<00:00,  2.49s/it]
Epoch 48:   2%|          | 1/59 [00:02<02:53,  2.99s/it]
Epoch: 48 ===== Batch: 0/59 ===== Disc loss: 1.0890 ===== Gen loss: 1.2302
Epoch 48:  86%|████████▌ | 51/59 [02:08<00:20,  2.55s/it]
Epoch: 48 ===== Batch: 50/59 ===== Disc loss: 0.9191 ===== Gen loss: 3.7380
Epoch 48: 100%|██████████| 59/59 [02:27<00:00,  2.50s/it]
Epoch 49:   2%|          | 1/59 [00:03<02:55,  3.02s/it]
Epoch: 49 ===== Batch: 0/59 ===== Disc loss: 0.7499 ===== Gen loss: 2.4667
Epoch 49:  86%|████████▌ | 51/59 [02:08<00:20,  2.53s/it]
Epoch: 49 ===== Batch: 50/59 ===== Disc loss: 0.7376 ===== Gen loss: 2.4567
Epoch 49: 100%|██████████| 59/59 [02:27<00:00,  2.50s/it]
```





Epoch 1

Epoch 10

Epoch 20

Epoch 30

Epoch 40

Epoch 50

Conclusion:

A successful experiment will yield a generator network that has learned the underlying patterns and distribution of the chosen dataset (Fashion MNIST). This demonstrates the GAN's ability to perform generative modelling, creating new, plausible images that share the characteristics of the real data, even though they are not exact copies of any training examples. The quality of the generated images and the stability of the loss curves indicate the success level.