```
from google.colab import files
import zipfile
import os

# Upload the zip file
uploaded = files.upload()

# Unzip the uploaded file
for filename in uploaded.keys():
    with zipfile.ZipFile(filename, 'r') as zip_ref:
        zip_ref.extractall('/content')

print("Dataset extracted!")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving custom_dataset.zip.zip to custom_dataset.zip.zip
Dataset extracted!
```

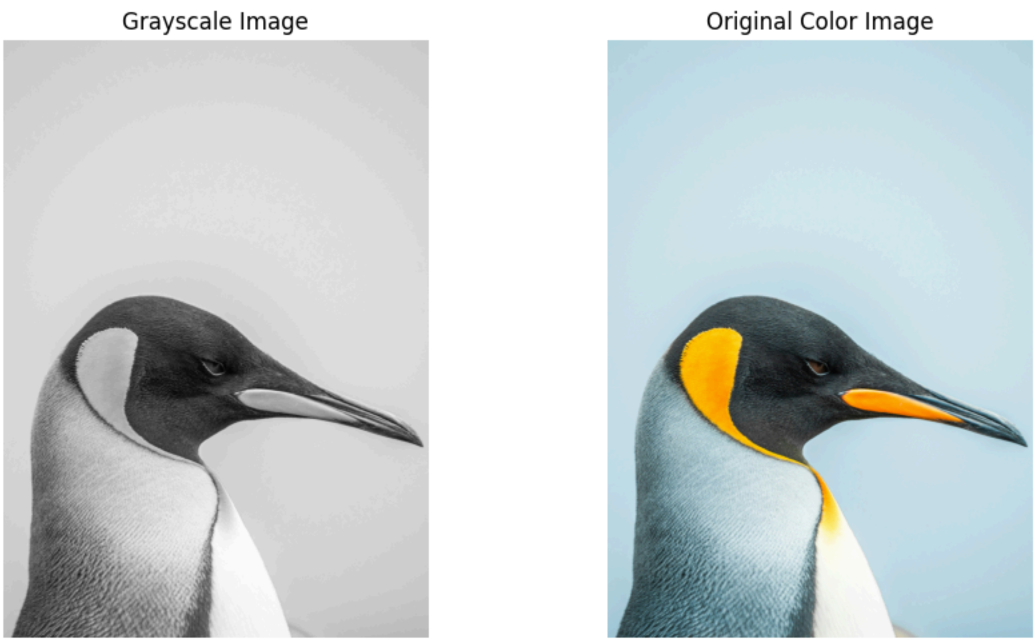
```
import os

# Specify the dataset directory
dataset_path = '/content/custom_dataset'

# List the folder contents
for subdir in ['color', 'grayscale']:
    folder_path = os.path.join(dataset_path, subdir)
    print(f"Contents of {folder_path}:")
    print(os.listdir(folder_path))
```

```
Contents of /content/custom_dataset/color:
['matthew-stephenson-EWJyQTLSo5o-unsplash.jpg', 'art-institute-of-chicago-x3NIjSe1ZA0-unsplash.jpg', 'minyeong-jeong-zUk-3kvcsaA-unsplas
Contents of /content/custom_dataset/grayscale:
['matthew-stephenson-EWJyQTLSo5o-unsplash.jpg', 'art-institute-of-chicago-x3NIjSe1ZA0-unsplash.jpg', 'minyeong-jeong-zUk-3kvcsaA-unsplas
```

```
import os
from PIL import Image

# Paths for color and grayscale folders
color_folder = "/content/custom_dataset/color"
grayscale_folder = "/content/custom_dataset/grayscale"

# Ensure the grayscale folder exists
os.makedirs(grayscale_folder, exist_ok=True)

# Convert each image in the color folder to grayscale
for img_name in os.listdir(color_folder):
    if img_name.endswith(('.png', '.jpg', '.jpeg')):
        img_path = os.path.join(color_folder, img_name)
        grayscale_path = os.path.join(grayscale_folder, img_name)

        # Open the color image and convert it to grayscale
        try:
            img = Image.open(img_path).convert("L")  # Convert to grayscale
            img.save(grayscale_path)
            print(f"Converted: {img_name} to grayscale.")
        except Exception as e:
            print(f"Error converting {img_name}: {e}")

print(f"Grayscale images saved in: {grayscale_folder}")
```

```
Converted: matthew-stephenson-EWJyQTLSo5o-unsplash.jpg to grayscale.
Converted: art-institute-of-chicago-x3NIjSe1ZA0-unsplash.jpg to grayscale.
Converted: minyeong-jeong-zUk-3kvcsaA-unsplash.jpg to grayscale.
Converted: jennifer-kalenberg-SLhuy1zeLsY-unsplash.jpg to grayscale.
Converted: lawrence-krowdeed-3WIfakzXoys-unsplash.jpg to grayscale.
Converted: gaku-suyama-Pd8rITTUA0w-unsplash.jpg to grayscale.
Converted: sehoon-ye-pp_7zvfCoG0-unsplash.jpg to grayscale.
Converted: lina-bob-up4IZxCQgT8-unsplash.jpg to grayscale.
Converted: royce-fonseca-_rc7z579KqA-unsplash.jpg to grayscale.
Converted: karsten-winegeart-EBE3dJlUhGE-unsplash.jpg to grayscale.
Grayscale images saved in: /content/custom_dataset/grayscale
```

```
import matplotlib.pyplot as plt

# List a grayscale and a color image for comparison
grayscale_images = os.listdir(grayscale_folder)
color_images = os.listdir(color_folder)

# Load a sample grayscale and color image
gray_img = Image.open(os.path.join(grayscale_folder, grayscale_images[0]))
color_img = Image.open(os.path.join(color_folder, color_images[0]))

# Display the images
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(gray_img, cmap='gray')
axs[0].set_title("Grayscale Image")
axs[0].axis('off')

axs[1].imshow(color_img)
axs[1].set_title("Original Color Image")
axs[1].axis('off')

plt.tight_layout()
plt.show()
```

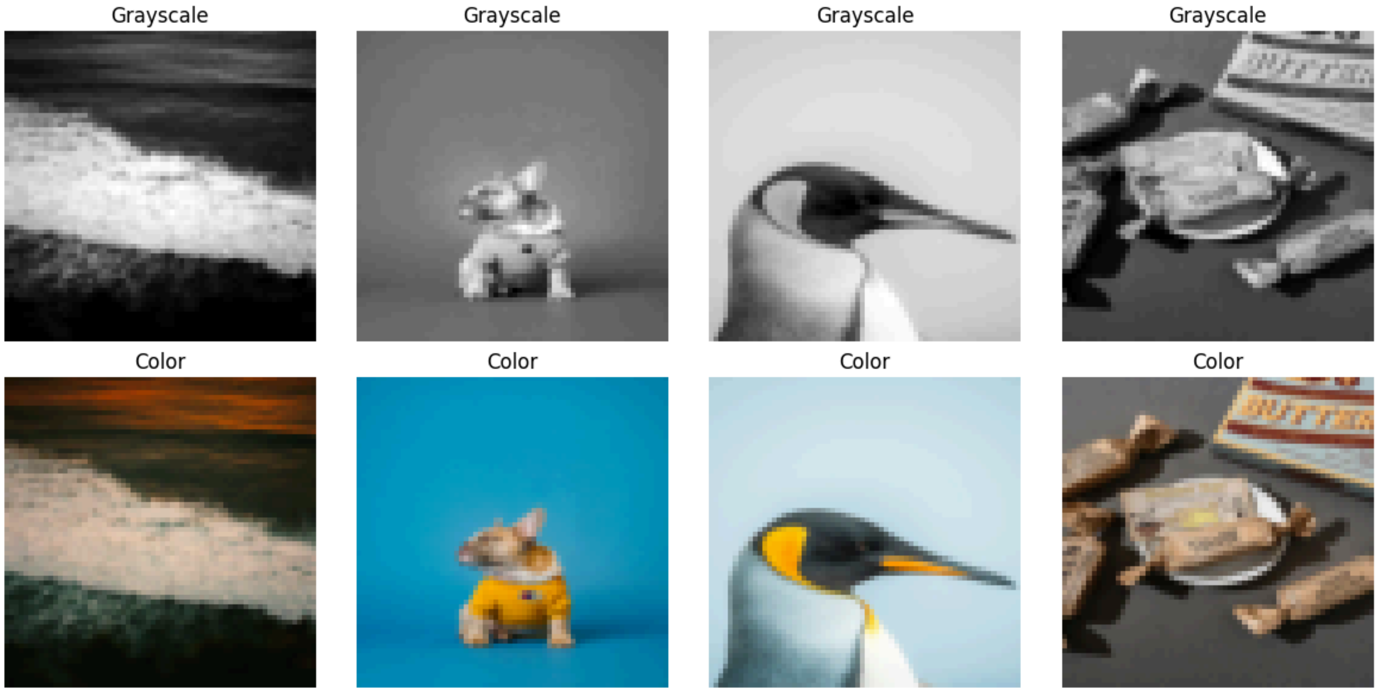



```
import torch
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import os

# Custom dataset class
class GrayscaleToColorDataset(Dataset):
    def __init__(self, grayscale_dir, color_dir, transform=None):
        self.grayscale_dir = grayscale_dir
        self.color_dir = color_dir
        self.transform = transform
        self.image_names = os.listdir(grayscale_dir)

    def __len__(self):
        return len(self.image_names)

    def __getitem__(self, idx):
        img_name = self.image_names[idx]

        # Load grayscale and color images
        gray_img = Image.open(os.path.join(self.grayscale_dir, img_name)).convert("L")
        color_img = Image.open(os.path.join(self.color_dir, img_name))

        # Apply transformations if specified
```

```
        if self.transform:
            gray_img = self.transform(gray_img)
            color_img = self.transform(color_img)

        return gray_img, color_img

# Define transformations
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize((64, 64)),  # Resize to 64x64
    transforms.ToTensor(),       # Convert to tensor
    transforms.Normalize((0.5,), (0.5,))  # Normalize to [-1, 1]
])

# Dataset paths
grayscale_dir = "/content/custom_dataset/grayscale"
color_dir = "/content/custom_dataset/color"

# Create dataset and DataLoader
dataset = GrayscaleToColorDataset(grayscale_dir, color_dir, transform=transform)
train_loader = DataLoader(dataset, batch_size=16, shuffle=True)

print("Dataset and DataLoader created successfully!")
```

```
Dataset and DataLoader created successfully!
```

```
import matplotlib.pyplot as plt

# Get a batch of data
gray_batch, color_batch = next(iter(train_loader))

# Display a few images
fig, axs = plt.subplots(2, 4, figsize=(12, 6))

for i in range(4):
    # Display grayscale image
    axs[0, i].imshow(gray_batch[i][0].numpy(), cmap='gray')
    axs[0, i].axis('off')
    axs[0, i].set_title("Grayscale")

    # Display color image
    axs[1, i].imshow(color_batch[i].permute(1, 2, 0).numpy() * 0.5 + 0.5)  # Denormalize
    axs[1, i].axis('off')
    axs[1, i].set_title("Color")

plt.tight_layout()
plt.show()
```




```python
import torch.nn as nn

class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=4, stride=2, padding=1),  # Grayscale input
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),
        )

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1),  # RGB output
            nn.Tanh(),  # Normalize output to [-1, 1]
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded


class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(4, 64, kernel_size=4, stride=2, padding=1),  # Concatenated grayscale+color input
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
            nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1),
```

```
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),
            nn.Conv2d(256, 1, kernel_size=4, stride=1, padding=0),
            nn.Sigmoid(),  # Output a probability
        )

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

```
import torch

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

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

# Loss functions
criterion_GAN = nn.BCELoss()  # Binary Cross-Entropy for real/fake classification
criterion_L1 = nn.L1Loss()    # L1 loss for pixel similarity

# Optimizers
lr = 0.0002
beta1 = 0.5
optimizer_G = torch.optim.Adam(G.parameters(), lr=lr, betas=(beta1, 0.999))
optimizer_D = torch.optim.Adam(D.parameters(), lr=lr, betas=(beta1, 0.999))

print("Generator and Discriminator initialized!")
```

```
Generator and Discriminator initialized!
```

```
# Test with a sample grayscale batch
gray_sample = next(iter(train_loader))[0].to(device)  # Only grayscale input
fake_color = G(gray_sample)  # Generate color image

# Test discriminator
real_image = next(iter(train_loader))[1].to(device)  # Real color image
d_input_real = torch.cat((gray_sample, real_image), 1)  # Concatenate grayscale and real
d_input_fake = torch.cat((gray_sample, fake_color), 1)  # Concatenate grayscale and fake

real_output = D(d_input_real)
fake_output = D(d_input_fake)

print("Generator output shape:", fake_color.shape)
print("Discriminator real output shape:", real_output.shape)
print("Discriminator fake output shape:", fake_output.shape)
```

```
Generator output shape: torch.Size([10, 3, 64, 64])
Discriminator real output shape: torch.Size([10, 1, 5, 5])
Discriminator fake output shape: torch.Size([10, 1, 5, 5])
```

```
num_epochs = 100  # Number of epochs
lambda_L1 = 100  # Weight for L1 loss

for epoch in range(num_epochs):
    for i, (gray_input, real_color) in enumerate(train_loader):
        # Move data to the correct device
        gray_input = gray_input.to(device)  # Grayscale input
        real_color = real_color.to(device)  # Real color image
        batch_size = gray_input.size(0)

        # Forward pass through the discriminator to get output size
        real_input = torch.cat((gray_input, real_color), 1)
        real_output = D(real_input)

        # Create labels for real and fake images with the correct spatial dimensions
        valid = torch.ones_like(real_output, device=device)  # Real labels
        fake = torch.zeros_like(real_output, device=device)  # Fake labels

        # ---------------------
        #  Train Discriminator
```

```
        # ---------------------
        optimizer_D.zero_grad()

        # Real images
        d_loss_real = criterion_GAN(real_output, valid)

        # Fake images
        fake_color = G(gray_input)
        fake_input = torch.cat((gray_input, fake_color), 1)
        fake_output = D(fake_input)
        d_loss_fake = criterion_GAN(fake_output, fake)

        # Total discriminator loss
        d_loss = (d_loss_real + d_loss_fake) / 2
        d_loss.backward()
        optimizer_D.step()

        # -----------------
        #  Train Generator
        # -----------------
        optimizer_G.zero_grad()

        # Generate fake images
        fake_color = G(gray_input)
        fake_input = torch.cat((gray_input, fake_color), 1)
        fake_output = D(fake_input)

        # Adversarial loss
        g_loss_GAN = criterion_GAN(fake_output, valid)

        # L1 loss
        g_loss_L1 = criterion_L1(fake_color, real_color)

        # Total generator loss
        g_loss = g_loss_GAN + lambda_L1 * g_loss_L1
        g_loss.backward()
        optimizer_G.step()

        # Print progress (Optional: You can comment this out if you don't need batch-wise output)
        if i % 10 == 0:
            print(f"[Epoch {epoch+1}/{num_epochs}] [Batch {i}/{len(train_loader)}] "
                  f"[D loss: {d_loss.item():.4f}] [G loss: {g_loss.item():.4f}]")

    # After completing all 100 epochs, generate and save/display results
    if (epoch + 1) == num_epochs:  # Only after the last epoch
        G.eval()  # Set generator to evaluation mode
        with torch.no_grad():
            # Get a batch of gray input images for testing (take first 4 images)
            sample_fake = G(gray_input[:4])
        G.train()  # Set generator back to training mode

        # Visualize results after the whole 100 epochs
        fig, axs = plt.subplots(2, 4, figsize=(12, 6))

        for idx in range(4):
            # Show grayscale input images
            axs[0, idx].imshow(gray_input[idx][0].cpu().numpy(), cmap='gray')
            axs[0, idx].axis('off')
            axs[0, idx].set_title("Grayscale Input")

            # Show generated fake color images
            axs[1, idx].imshow(sample_fake[idx].permute(1, 2, 0).cpu().numpy() * 0.5 + 0.5)
            axs[1, idx].axis('off')
            axs[1, idx].set_title("Generated Color")

        plt.tight_layout()
        plt.show()

        # Optionally save the model or checkpoint after the final epoch
        torch.save(G.state_dict(), 'generator_final.pth')
        torch.save(D.state_dict(), 'discriminator_final.pth')
```

```
[Epoch 1/100] [Batch 0/1] [D loss: 0.3431] [G loss: 7.9375]
[Epoch 2/100] [Batch 0/1] [D loss: 0.3107] [G loss: 8.1142]
[Epoch 3/100] [Batch 0/1] [D loss: 0.3305] [G loss: 8.0108]
[Epoch 4/100] [Batch 0/1] [D loss: 0.3266] [G loss: 8.0940]
[Epoch 5/100] [Batch 0/1] [D loss: 0.3164] [G loss: 8.4453]
[Epoch 6/100] [Batch 0/1] [D loss: 0.3128] [G loss: 8.2003]
[Epoch 7/100] [Batch 0/1] [D loss: 0.3094] [G loss: 7.8730]
[Epoch 8/100] [Batch 0/1] [D loss: 0.3824] [G loss: 7.7136]
[Epoch 9/100] [Batch 0/1] [D loss: 0.3803] [G loss: 7.9068]
[Epoch 10/100] [Batch 0/1] [D loss: 0.4228] [G loss: 7.5045]
[Epoch 11/100] [Batch 0/1] [D loss: 0.3222] [G loss: 7.4971]
[Epoch 12/100] [Batch 0/1] [D loss: 0.4015] [G loss: 7.5350]
[Epoch 13/100] [Batch 0/1] [D loss: 0.4123] [G loss: 7.1637]
[Epoch 14/100] [Batch 0/1] [D loss: 0.4661] [G loss: 7.2521]
[Epoch 15/100] [Batch 0/1] [D loss: 0.4391] [G loss: 7.1376]
[Epoch 16/100] [Batch 0/1] [D loss: 0.3944] [G loss: 7.7902]
[Epoch 17/100] [Batch 0/1] [D loss: 0.3927] [G loss: 7.4672]
[Epoch 18/100] [Batch 0/1] [D loss: 0.3859] [G loss: 7.2777]
[Epoch 19/100] [Batch 0/1] [D loss: 0.4452] [G loss: 7.6027]
[Epoch 20/100] [Batch 0/1] [D loss: 0.4012] [G loss: 7.5568]
[Epoch 21/100] [Batch 0/1] [D loss: 0.4211] [G loss: 7.6895]
[Epoch 22/100] [Batch 0/1] [D loss: 0.4989] [G loss: 7.0026]
[Epoch 23/100] [Batch 0/1] [D loss: 0.4552] [G loss: 7.3536]
[Epoch 24/100] [Batch 0/1] [D loss: 0.5454] [G loss: 6.7348]
[Epoch 25/100] [Batch 0/1] [D loss: 0.4960] [G loss: 7.6139]
[Epoch 26/100] [Batch 0/1] [D loss: 0.7914] [G loss: 7.0083]
[Epoch 27/100] [Batch 0/1] [D loss: 0.5273] [G loss: 7.8416]
[Epoch 28/100] [Batch 0/1] [D loss: 1.4312] [G loss: 6.8338]
[Epoch 29/100] [Batch 0/1] [D loss: 0.9658] [G loss: 7.9729]
[Epoch 30/100] [Batch 0/1] [D loss: 1.9015] [G loss: 6.5029]
[Epoch 31/100] [Batch 0/1] [D loss: 0.8267] [G loss: 6.2624]
[Epoch 32/100] [Batch 0/1] [D loss: 0.6328] [G loss: 6.3648]
[Epoch 33/100] [Batch 0/1] [D loss: 0.6261] [G loss: 6.4342]
[Epoch 34/100] [Batch 0/1] [D loss: 0.5909] [G loss: 6.3500]
[Epoch 35/100] [Batch 0/1] [D loss: 0.5197] [G loss: 6.4015]
[Epoch 36/100] [Batch 0/1] [D loss: 0.4903] [G loss: 6.3536]
[Epoch 37/100] [Batch 0/1] [D loss: 0.4535] [G loss: 6.4423]
[Epoch 38/100] [Batch 0/1] [D loss: 0.4369] [G loss: 6.4955]
[Epoch 39/100] [Batch 0/1] [D loss: 0.3966] [G loss: 6.5897]
[Epoch 40/100] [Batch 0/1] [D loss: 0.3978] [G loss: 6.5722]
[Epoch 41/100] [Batch 0/1] [D loss: 0.3641] [G loss: 6.8870]
[Epoch 42/100] [Batch 0/1] [D loss: 0.3811] [G loss: 6.6632]
[Epoch 43/100] [Batch 0/1] [D loss: 0.3337] [G loss: 7.3224]
[Epoch 44/100] [Batch 0/1] [D loss: 0.3731] [G loss: 6.7308]
[Epoch 45/100] [Batch 0/1] [D loss: 0.3294] [G loss: 7.1830]
[Epoch 46/100] [Batch 0/1] [D loss: 0.3293] [G loss: 6.9522]
[Epoch 47/100] [Batch 0/1] [D loss: 0.3454] [G loss: 6.9040]
[Epoch 48/100] [Batch 0/1] [D loss: 0.3405] [G loss: 6.6994]
[Epoch 49/100] [Batch 0/1] [D loss: 0.3207] [G loss: 7.1182]
[Epoch 50/100] [Batch 0/1] [D loss: 0.3059] [G loss: 7.8415]
[Epoch 51/100] [Batch 0/1] [D loss: 0.3070] [G loss: 7.9173]
[Epoch 52/100] [Batch 0/1] [D loss: 0.3743] [G loss: 6.8856]
[Epoch 53/100] [Batch 0/1] [D loss: 0.3187] [G loss: 6.7659]
[Epoch 54/100] [Batch 0/1] [D loss: 0.3412] [G loss: 6.7269]
[Epoch 55/100] [Batch 0/1] [D loss: 0.3306] [G loss: 6.9259]
[Epoch 56/100] [Batch 0/1] [D loss: 0.3387] [G loss: 7.0214]
[Epoch 57/100] [Batch 0/1] [D loss: 0.3308] [G loss: 7.3287]
[Epoch 58/100] [Batch 0/1] [D loss: 0.3196] [G loss: 6.8742]
[Epoch 59/100] [Batch 0/1] [D loss: 0.3068] [G loss: 7.0354]
[Epoch 60/100] [Batch 0/1] [D loss: 0.2853] [G loss: 7.2838]
[Epoch 61/100] [Batch 0/1] [D loss: 0.2898] [G loss: 7.5029]
[Epoch 62/100] [Batch 0/1] [D loss: 0.3543] [G loss: 6.9020]
[Epoch 63/100] [Batch 0/1] [D loss: 0.3111] [G loss: 7.1485]
[Epoch 64/100] [Batch 0/1] [D loss: 0.3670] [G loss: 6.8148]
[Epoch 65/100] [Batch 0/1] [D loss: 0.3156] [G loss: 7.3358]
[Epoch 66/100] [Batch 0/1] [D loss: 0.3633] [G loss: 6.6861]
[Epoch 67/100] [Batch 0/1] [D loss: 0.2997] [G loss: 6.9796]
[Epoch 68/100] [Batch 0/1] [D loss: 0.3335] [G loss: 7.1556]
[Epoch 69/100] [Batch 0/1] [D loss: 0.2717] [G loss: 7.3085]
[Epoch 70/100] [Batch 0/1] [D loss: 0.3787] [G loss: 6.8841]
[Epoch 71/100] [Batch 0/1] [D loss: 0.3516] [G loss: 6.9433]
[Epoch 72/100] [Batch 0/1] [D loss: 0.4398] [G loss: 6.3756]
[Epoch 73/100] [Batch 0/1] [D loss: 0.4152] [G loss: 6.3951]
[Epoch 74/100] [Batch 0/1] [D loss: 0.3364] [G loss: 7.4552]
[Epoch 75/100] [Batch 0/1] [D loss: 0.3992] [G loss: 7.0351]
[Epoch 76/100] [Batch 0/1] [D loss: 0.4423] [G loss: 6.6102]
[Epoch 77/100] [Batch 0/1] [D loss: 0.4677] [G loss: 6.2332]
[Epoch 78/100] [Batch 0/1] [D loss: 0.4428] [G loss: 6.2237]
[Epoch 79/100] [Batch 0/1] [D loss: 0.3368] [G loss: 7.1982]
[Epoch 80/100] [Batch 0/1] [D loss: 0.3410] [G loss: 6.9629]
[Epoch 81/100] [Batch 0/1] [D loss: 0.3127] [G loss: 7.1278]
[Epoch 82/100] [Batch 0/1] [D loss: 0.3052] [G loss: 7.1256]
[Epoch 83/100] [Batch 0/1] [D loss: 0.2822] [G loss: 7.6308]
[Epoch 84/100] [Batch 0/1] [D loss: 0.3998] [G loss: 6.3775]
```

```
[Epoch 85/100] [Batch 0/1] [D loss: 0.3759] [G loss: 6.3990]
[Epoch 86/100] [Batch 0/1] [D loss: 0.4380] [G loss: 6.7184]
[Epoch 87/100] [Batch 0/1] [D loss: 0.3941] [G loss: 6.7968]
[Epoch 88/100] [Batch 0/1] [D loss: 0.4004] [G loss: 6.4105]
[Epoch 89/100] [Batch 0/1] [D loss: 0.4378] [G loss: 6.9830]
[Epoch 90/100] [Batch 0/1] [D loss: 0.5274] [G loss: 6.2809]
[Epoch 91/100] [Batch 0/1] [D loss: 0.4685] [G loss: 7.1637]
[Epoch 92/100] [Batch 0/1] [D loss: 0.6428] [G loss: 5.8314]
[Epoch 93/100] [Batch 0/1] [D loss: 0.5158] [G loss: 7.0782]
[Epoch 94/100] [Batch 0/1] [D loss: 0.8373] [G loss: 5.6372]
[Epoch 95/100] [Batch 0/1] [D loss: 0.7739] [G loss: 7.1387]
[Epoch 96/100] [Batch 0/1] [D loss: 1.1597] [G loss: 5.4875]
[Epoch 97/100] [Batch 0/1] [D loss: 0.7841] [G loss: 5.9157]
[Epoch 98/100] [Batch 0/1] [D loss: 0.7963] [G loss: 5.6613]
[Epoch 99/100] [Batch 0/1] [D loss: 0.9441] [G loss: 5.4293]
[Epoch 100/100] [Batch 0/1] [D loss: 0.7450] [G loss: 5.6287]
```