```
!ls
```

```
sample_data
```

```
# ===========================
# STEP 1: SETUP & IMPORTS
# ===========================
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
import os
from datetime import datetime

print("TensorFlow version:", tf.__version__)

# ===========================
# MOUNT GOOGLE DRIVE
# ===========================
from google.colab import drive
drive.mount('/content/drive')
```

```
TensorFlow version: 2.19.0
Mounted at /content/drive
```

```
# ===========================
# PROJECT DIRECTORIES
# ===========================
PROJECT_NAME = "MNIST_DCGAN"
BASE_DIR = f"/content/drive/MyDrive/{PROJECT_NAME}"

CHECKPOINT_DIR = os.path.join(BASE_DIR, "checkpoints")
IMAGE_DIR = os.path.join(BASE_DIR, "generated_images")
MODEL_DIR = os.path.join(BASE_DIR, "models")
PLOT_DIR = os.path.join(BASE_DIR, "plots")

for folder in [BASE_DIR, CHECKPOINT_DIR, IMAGE_DIR, MODEL_DIR, PLOT_DIR]
    os.makedirs(folder, exist_ok=True)

print("All folders created successfully")
```

```
All folders created successfully
```

```
# ===========================
# HYPERPARAMETERS
# ===========================
EPOCHS = 150          # More epochs = better quality
BATCH_SIZE = 256
LATENT_DIM = 100
SAVE_INTERVAL = 10
```

```
GEN_LR = 0.0002
DISC_LR = 0.0002
BETA_1 = 0.5

tf.random.set_seed(42)
np.random.seed(42)
```

```
# ===========================
# LOAD MNIST DATASET
# ===========================
(train_images, _), (_, _) = keras.datasets.mnist.load_data()

# Reshape and normalize (-1 to 1)
train_images = train_images.reshape(-1, 28, 28, 1).astype("float32")
train_images = (train_images - 127.5) / 127.5

dataset = tf.data.Dataset.from_tensor_slices(train_images)
dataset = dataset.shuffle(60000).batch(BATCH_SIZE, drop_remainder=True

print("MNIST dataset loaded")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-kera
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
MNIST dataset loaded
```

```
# ===========================
# GENERATOR MODEL
# ===========================
def build_generator():
    model = keras.Sequential([
        layers.Dense(7 * 7 * 256, use_bias=False, input_shape=(LATENT_
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Reshape((7, 7, 256)),

        layers.Conv2DTranspose(128, 5, strides=1, padding="same", use_
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Conv2DTranspose(64, 5, strides=2, padding="same", use_b
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Conv2DTranspose(1, 5, strides=2, padding="same",
                               use_bias=False, activation="tanh")
    ])
    return model

generator = build_generator()
generator.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Para
dense (Dense)	(None, 12544)	1,254,
batch_normalization (BatchNormalization)	(None, 12544)	50,
leaky_re_lu (LeakyReLU)	(None, 12544)	
reshape (Reshape)	(None, 7, 7, 256)	
conv2d_transpose (Conv2DTranspose)	(None, 7, 7, 128)	819,
batch_normalization_1 (BatchNormalization)	(None, 7, 7, 128)	5
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	
conv2d_transpose_1 (Conv2DTranspose)	(None, 14, 14, 64)	204,
batch_normalization_2 (BatchNormalization)	(None, 14, 14, 64)	2
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	
conv2d_transpose_2 (Conv2DTranspose)	(None, 28, 28, 1)	1,

```
 Total params: 2,330,944 (8.89 MB)
 Trainable params: 2,305,472 (8.79 MB)
 Non-trainable params: 25,472 (99.50 KB)
```

```
# ===========================
# DISCRIMINATOR MODEL
# ===========================
def build_discriminator():
    model = keras.Sequential([
        layers.Conv2D(64, 5, strides=2, padding="same",
                      input_shape=[28, 28, 1]),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

        layers.Conv2D(128, 5, strides=2, padding="same"),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

        layers.Flatten(),
        layers.Dense(1)
    ])
    return model

discriminator = build_discriminator()
discriminator.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
┃ Layer (type)                    ┃ Output Shape           ┃       Para
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
│ conv2d (Conv2D)                 │ (None, 14, 14, 64)     │         1,
├─────────────────────────────────┼────────────────────────┼──────────
│ leaky_re_lu_3 (LeakyReLU)       │ (None, 14, 14, 64)     │
├─────────────────────────────────┼────────────────────────┼──────────
│ dropout (Dropout)               │ (None, 14, 14, 64)     │
├─────────────────────────────────┼────────────────────────┼──────────
│ conv2d_1 (Conv2D)               │ (None, 7, 7, 128)      │       204,
├─────────────────────────────────┼────────────────────────┼──────────
│ leaky_re_lu_4 (LeakyReLU)       │ (None, 7, 7, 128)      │
├─────────────────────────────────┼────────────────────────┼──────────
│ dropout_1 (Dropout)             │ (None, 7, 7, 128)      │
├─────────────────────────────────┼────────────────────────┼──────────
│ flatten (Flatten)               │ (None, 6272)           │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_1 (Dense)                 │ (None, 1)              │         6,
└─────────────────────────────────┴────────────────────────┴──────────

 Total params: 212,865 (831.50 KB)
 Trainable params: 212,865 (831.50 KB)
 Non-trainable params: 0 (0.00 B)
```

```python
# ===========================
# LOSS & OPTIMIZERS
# ===========================
cross_entropy = keras.losses.BinaryCrossentropy(from_logits=True)

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    return real_loss + fake_loss

generator_optimizer = keras.optimizers.Adam(GEN_LR, beta_1=BETA_1)
discriminator_optimizer = keras.optimizers.Adam(DISC_LR, beta_1=BETA_1
```

```python
# ===========================
# TRAINING STEP
# ===========================
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, LATENT_DIM])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape
        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

    gen_gradients = gen_tape.gradient(gen_loss, generator.trainable_va
    disc_gradients = disc_tape.gradient(disc_loss, discriminator.traina

    generator_optimizer.apply_gradients(zip(gen_gradients, generator.t
    discriminator_optimizer.apply_gradients(zip(disc_gradients, discri

    return gen_loss, disc_loss
```

```
# ============================
# IMAGE SAVING FUNCTION
# ============================
def save_generated_images(epoch, seed):
    predictions = generator(seed, training=False)

    fig = plt.figure(figsize=(6, 6))
    for i in range(16):
        plt.subplot(4, 4, i + 1)
        plt.imshow(predictions[i, :, :, 0] * 0.5 + 0.5, cmap="gray")
        plt.axis("off")

    path = os.path.join(IMAGE_DIR, f"epoch_{epoch:04d}.png")
    plt.savefig(path)
    plt.close()
```

```
# ============================
# TRAINING LOOP
# ============================
seed = tf.random.normal([16, LATENT_DIM])
history = {"gen": [], "disc": []}

print("Starting Training...")

for epoch in range(1, EPOCHS + 1):
    gen_losses = []
    disc_losses = []

    for image_batch in dataset:
        g_loss, d_loss = train_step(image_batch)
        gen_losses.append(g_loss.numpy())
        disc_losses.append(d_loss.numpy())

    history["gen"].append(np.mean(gen_losses))
    history["disc"].append(np.mean(disc_losses))

    print(f"Epoch {epoch}/{EPOCHS} | G: {history['gen'][-1]:.4f} | D:

    if epoch % SAVE_INTERVAL == 0:
        save_generated_images(epoch, seed)
```

```
            generator.save(os.path.join(CHECKPOINT_DIR, f"generator_epoch_
            discriminator.save(os.path.join(CHECKPOINT_DIR, f"discriminato
```

```
Epoch 140/150 | G: 0.7837 | D: 1.3271
Epoch 149/150 | G: 0.7837 | D: 1.3271
Epoch 150/150 | G: 0.7855 | D: 1.3251
```

```
# ===========================
# SAVE FINAL OUTPUTS
# ===========================
generator.save(os.path.join(MODEL_DIR, "generator_final.keras"))
discriminator.save(os.path.join(MODEL_DIR, "discriminator_final.keras"

plt.plot(history["gen"], label="Generator Loss")
plt.plot(history["disc"], label="Discriminator Loss")
plt.legend()
plt.savefig(os.path.join(PLOT_DIR, "loss_plot.png"))
plt.close()

print("Training Complete & Everything Saved to Drive")
```

```
Training Complete & Everything Saved to Drive
```

Start coding or generate with AI.

Start coding or generate with AI.