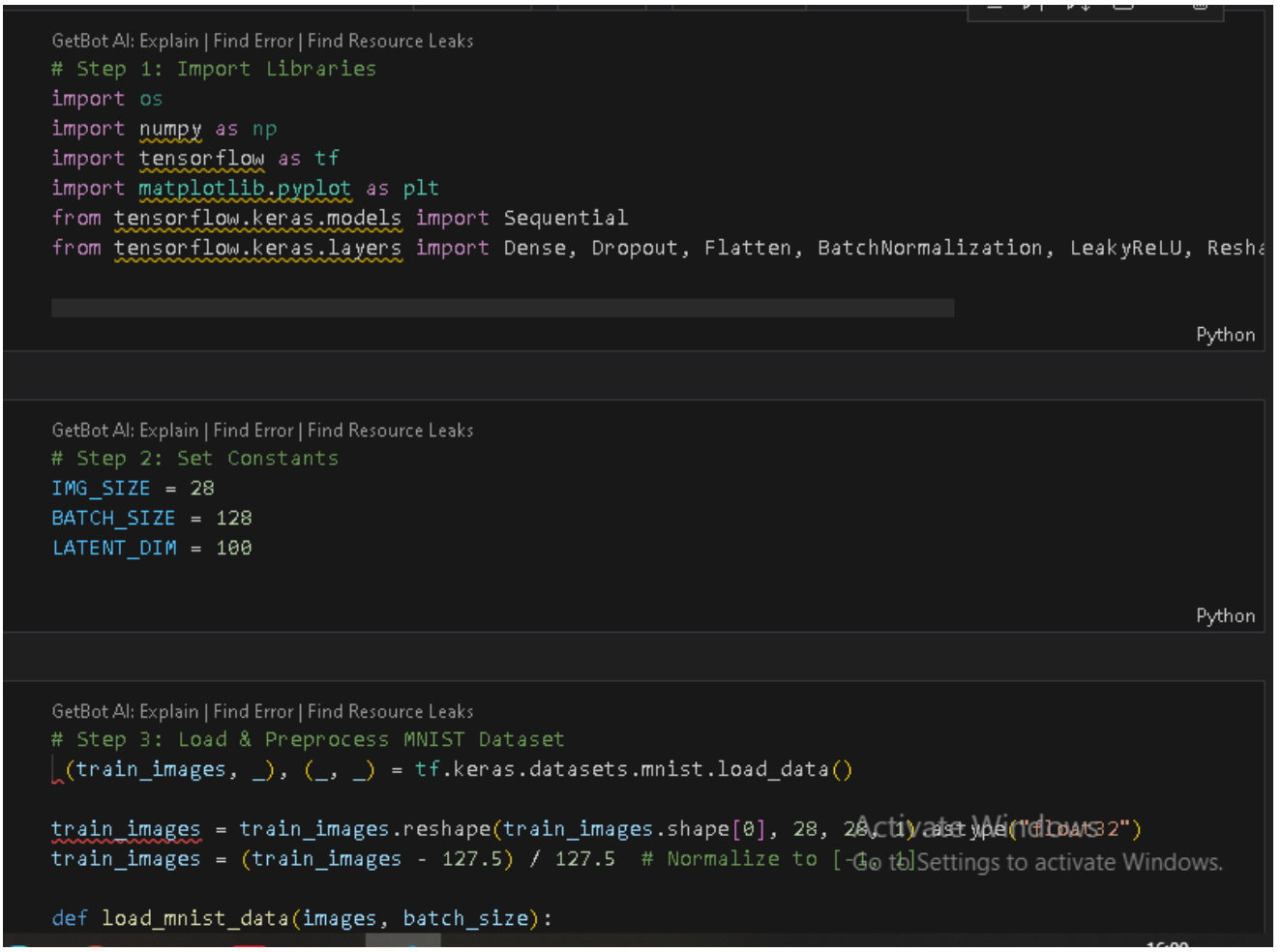
Marwadi University Marwadi Chandarana Group	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**	
Subject: Gen AI	**Write a code for DC_GAN. USE DATASET MNIST FROM KERAS**	
Experiment	**Date:**	**Enrolment No:92200133020**

CODE:



```
GetBot AI: Explain | Find Error | Find Resource Leaks
# Step 1: Import Libraries
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, BatchNormalization, LeakyReLU, Resha
```

Python

```
GetBot AI: Explain | Find Error | Find Resource Leaks
# Step 2: Set Constants
IMG_SIZE = 28
BATCH_SIZE = 128
LATENT_DIM = 100
```

Python

```
GetBot AI: Explain | Find Error | Find Resource Leaks
# Step 3: Load & Preprocess MNIST Dataset
(train_images, _), (_, _) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype("float32")
train_images = (train_images - 127.5) / 127.5  # Normalize to [-1, 1]

def load_mnist_data(images, batch_size):
```

```python
train_images = (train_images - 127.5) / 127.5  # Normalize to [-1, 1]

def load_mnist_data(images, batch_size):
    dataset = tf.data.Dataset.from_tensor_slices(images)
    dataset = dataset.shuffle(60000).batch(batch_size).prefetch(tf.data.AUTOTUNE)
    return dataset

load = load_mnist_data(train_images, BATCH_SIZE)
for image_batch in load.take(1):
    print(image_batch.shape)
```

Python

```
(128, 28, 28, 1)
```

GetBot AI: Explain | Find Error | Find Resource Leaks

```python
#  Step 4: Define Generator
def build_generator():
    model = Sequential([
        Dense(7*7*256, use_bias=False, input_shape=(LATENT_DIM,)),
        BatchNormalization(),
        LeakyReLU(),
        Reshape((7, 7, 256)),

        Conv2DTranspose(128, 5, strides=1, padding='same', use_bias=False),
        BatchNormalization(),
        LeakyReLU(),

        Conv2DTranspose(64, 5, strides=2, padding='same', use_bias=False),
        BatchNormalization(),
        LeakyReLU()
```

```python
        LeakyReLU(),

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

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

Python

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 12544)	1,254,400
batch_normalization (BatchNormalization)	(None, 12544)	50,176
leaky_re_lu_5 (LeakyReLU)	(None, 12544)	0
reshape_1 (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose_3 (Conv2DTranspose)	(None, 7, 7, 128)	819,200
batch_normalization_1 (BatchNormalization)	(None, 7, 7, 128)	512
leaky_re_lu_6 (LeakyReLU)	(None, 7, 7, 128)	0

conv2d_transpose_4 (Conv2DTranspose)	(None, 14, 14, 64)	204,800
batch_normalization_2 (BatchNormalization)	(None, 14, 14, 64)	256
leaky_re_lu_7 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_5 (Conv2DTranspose)	(None, 28, 28, 1)	1,600

Total params: 2,330,944 (8.89 MB)

Trainable params: 2,305,472 (8.79 MB)

Non-trainable params: 25,472 (99.50 KB)

GetBot AI: Explain | Find Error | Find Resource Leaks

```
# Step 5: Define Discriminator
def build_discriminator():
    model = Sequential([
        Conv2D(64, 5, strides=2, padding='same', input_shape=[28, 28, 1]),
        LeakyReLU(),
        Dropout(0.3),
```

```
        Conv2D(128, 5, strides=2, padding='same'),
        LeakyReLU(),
        Dropout(0.3),

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

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

Python

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 14, 14, 64)	1,664
leaky_re_lu_8 (LeakyReLU)	(None, 14, 14, 64)	0
dropout_2 (Dropout)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 7, 7, 128)	204,928
leaky_re_lu_9 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_3 (Dropout)	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dense_3 (Dense)	(None, 1)	6,273

Total params: 212,865 (831.50 KB)

Trainable params: 212,865 (831.50 KB)

Non-trainable params: 0 (0.00 B)


```
# Step 6: Define Loss & Optimizers
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

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

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

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

gen_opt = tf.keras.optimizers.Adam(1e-4)
disc_opt = tf.keras.optimizers.Adam(1e-4)
```

Python

GetBot AI: Explain | Find Error | Find Resource Leaks

```python
# Step 7: Training Step Function
@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:
        gen_images = generator(noise, training=True)

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

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

    gradients_gen = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_disc = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    gen_opt.apply_gradients(zip(gradients_gen, generator.trainable_variables))
    disc_opt.apply_gradients(zip(gradients_disc, discriminator.trainable_variables))
```


GetBot AI: Explain | Find Error | Find Resource Leaks

```python
# Step 8: Generate and Show Sample Images
def generate_images(model, test_input):
    predictions = model(test_input, training=False)
    fig = plt.figure(figsize=(4, 4))

    for i in range(predictions.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
        plt.axis('off')
    plt.show()
```

Python

GetBot AI: Explain | Find Error | Find Resource Leaks

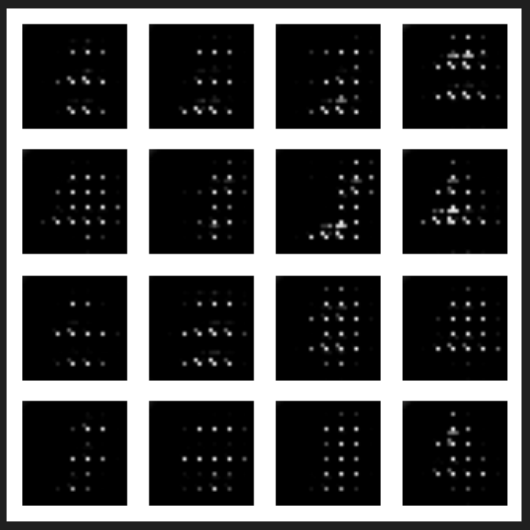
```python
# Step 9: Train the GAN
EPOCHS = 50
seed = tf.random.normal([16, LATENT_DIM])

def train(dataset, epochs):
    for epoch in range(epochs):
        for image_batch in dataset:
            train_step(image_batch)
        print(f"Epoch {epoch+1} completed")
        generate_images(generator, seed)

train(load, EPOCHS)
```

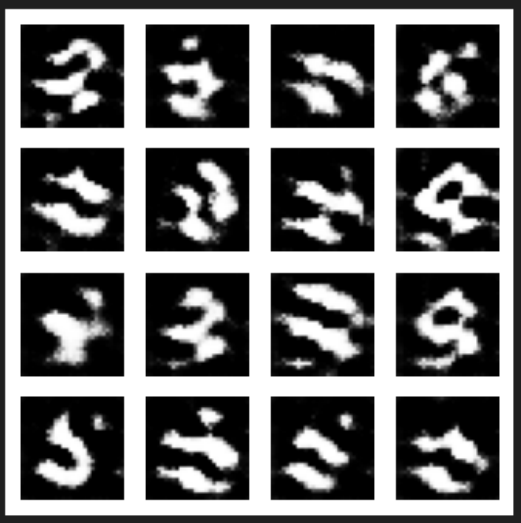

Python

```
Epoch 1 completed
```

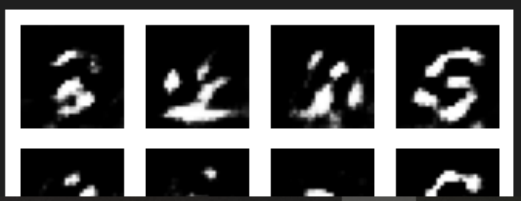


```
Epoch 2 completed
```

Epoch 3 completed



Epoch 4 completed



Epoch 50 completed