EXERCISE NO. 9
SIMPLE GAN USING TENSORFLOW

AIM:
To implement a simple GAN using TensorFlow.

ALGORITHM:

1. Import the necessary libraries.
2. Load & preprocess Fashion MNIST.
3. Build the generator and the discriminator.
4. Define the losses and the optimisers.
5. Train the model to fit the dataset.
6. Display the model summaries.

PROGRAM:

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt

(x_train, _), _ = tf.keras.datasets.fashion_mnist.load_data()
x_train = np.expand_dims(x_train, axis=-1)
x_train = tf.image.resize(x_train, [32, 32])
x_train = tf.image.grayscale_to_rgb(x_train)
x_train = x_train / 127.5 - 1.0

BATCH_SIZE = 128
LATENT_DIM = 100
EPOCHS = 100
NUM_EXAMPLES = 10

train_dataset = tf.data.Dataset.from_tensor_slices(x_train).shuffle(10000).batch(BATCH_SIZE)

def build_generator():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(LATENT_DIM,)),
        layers.Dense(8 * 8 * 128, use_bias=False),
```

```
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Reshape((8, 8, 128)),

        layers.Conv2DTranspose(64, 4, strides=2, padding='same', use_bias=False),
        layers.BatchNormalization(),
        layers.LeakyReLU(),

        layers.Conv2DTranspose(3, 4, strides=2, padding='same', use_bias=False, activation='tanh'),
    ])
    return model

def build_discriminator():
    model = tf.keras.Sequential([
        tf.keras.Input(shape=(32, 32, 3)),
        layers.Conv2D(64, 4, strides=2, padding='same'),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

        layers.Conv2D(128, 4, strides=2, padding='same'),
        layers.LeakyReLU(),
        layers.Dropout(0.3),

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

cross_entropy = tf.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 = build_generator()
discriminator = build_discriminator()

print("Generator Summary:")
generator.summary()

print("\nDiscriminator Summary:")
discriminator.summary()

gen_optimizer = tf.keras.optimizers.Adam(1e-4)
disc_optimizer = tf.keras.optimizers.Adam(1e-4)

@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)

    gradients_of_gen = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_disc = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    gen_optimizer.apply_gradients(zip(gradients_of_gen, generator.trainable_variables))
    disc_optimizer.apply_gradients(zip(gradients_of_disc, discriminator.trainable_variables))

def train(dataset, epochs):
    for epoch in range(epochs):
        for batch in dataset:
            train_step(batch)
        if (epoch + 1) % 20 == 0 or epoch == epochs - 1:
```

```
        print(f"Epoch {epoch + 1}/{epochs}")
        generate_images(generator)

train(train_dataset, EPOCHS)
```

OUTPUT:

Generator Summary:
Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 8192)	819,200
batch_normalization (BatchNormalization)	(None, 8192)	32,768
leaky_re_lu (LeakyReLU)	(None, 8192)	0
reshape (Reshape)	(None, 8, 8, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 16, 16, 64)	131,072
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256
leaky_re_lu_1 (LeakyReLU)	(None, 16, 16, 64)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 32, 32, 3)	3,072

Total params: 986,368 (3.76 MB)
Trainable params: 969,856 (3.70 MB)
Non-trainable params: 16,512 (64.50 KB)

Discriminator Summary:
Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 16, 64)	3,136
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
conv2d_1 (Conv2D)	(None, 8, 8, 128)	131,200
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 128)	0
dropout_1 (Dropout)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 1)	8,193

Total params: 142,529 (556.75 KB)
Trainable params: 142,529 (556.75 KB)
Non-trainable params: 0 (0.00 B)

RESULT:

Thus the program has been successfully implemented and verified.