9: SIMPLE GAN WITH TENSORFLOW

AIM:

To build a basic GAN (Generative Adversarial Network) to generate synthetic images.

PROCEDURE:

1. Define generator and discriminator models.
2. Compile the models.
3. Train both models in adversarial fashion.
4. Generate and visualize fake samples.
5. Monitor loss during training.

CODE:

```
import tensorflow as tf
from tensorflow.keras import layers
import numpy as np

# Load and preprocess Fashion MNIST
(X_train, _), _ = tf.keras.datasets.fashion_mnist.load_data()
X_train = (X_train.astype("float32") - 127.5) / 127.5
X_train = np.expand_dims(X_train, axis=-1)

# Generator model
def build_generator():
    model = tf.keras.Sequential([
```

```
        layers.Dense(128, input_shape=(100,)),
        layers.LeakyReLU(0.2),
        layers.BatchNormalization(),
        layers.Dense(784, activation='tanh'),
        layers.Reshape((28, 28, 1))
    ], name="Generator")
    return model

# Discriminator model
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Flatten(input_shape=(28, 28, 1)),
        layers.Dense(128),
        layers.LeakyReLU(0.2),
        layers.Dense(1, activation='sigmoid')
    ], name="Discriminator")
    return model

# Instantiate and compile
generator = build_generator()
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy',
optimizer=tf.keras.optimizers.Adam(0.0002), metrics=['accuracy'])

# GAN model
z = layers.Input(shape=(100,))
img = generator(z)
discriminator.trainable = False
validity = discriminator(img)
```

```
gan = tf.keras.Model(z, validity, name="GAN")
gan.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(0.0002))

# Train once
half_batch = 32
real_imgs = X_train[np.random.randint(0, X_train.shape[0], half_batch)]
noise = np.random.normal(0, 1, (half_batch, 100))
fake_imgs = generator.predict(noise)

# Train discriminator
d_loss_real = discriminator.train_on_batch(real_imgs, np.ones((half_batch, 1)))
d_loss_fake = discriminator.train_on_batch(fake_imgs, np.zeros((half_batch, 1)))
d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

# Train generator
noise = np.random.normal(0, 1, (64, 100))
g_loss = gan.train_on_batch(noise, np.ones((64, 1)))

# Print losses
print("\n Discriminator Loss: {:.4f}, Accuracy: {:.2f}%".format(d_loss[0], d_loss[1]*100))
print("Generator Loss: {:.4f}".format(g_loss))

# Display model summaries in table format
print("\n Generator Summary:")
generator.summary()

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

OUTPUT:

```
 Discriminator Loss: 1.0208, Accuracy: 25.00%
Generator Loss: 0.8533

 Generator Summary:
Model: "Generator"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	12,928
leaky_re_lu (LeakyReLU)	(None, 128)	0
batch_normalization (BatchNormalization)	(None, 128)	512
dense_1 (Dense)	(None, 784)	101,136
reshape (Reshape)	(None, 28, 28, 1)	0

```
Total params: 114,576 (447.56 KB)
Trainable params: 114,320 (446.56 KB)
Non-trainable params: 256 (1.00 KB)

Discriminator Summary:
Model: "Discriminator"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense_2 (Dense)	(None, 128)	100,480
leaky_re_lu_1 (LeakyReLU)	(None, 128)	0
dense_3 (Dense)	(None, 1)	129

```
Total params: 100,609 (393.00 KB)
Trainable params: 0 (0.00 B)
Non-trainable params: 100,609 (393.00 KB)
```

RESULT:

Generated synthetic data points (MNIST-like) using a simple GAN structure.