Ex.No.8

BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

AIM:

To build a generative adversarial neural network using Keras/TensorFlow

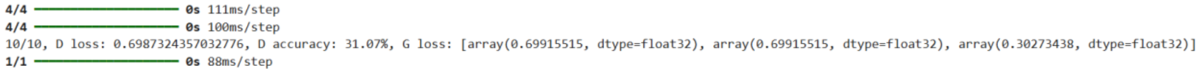
PROCEDURE:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a generative adversarial neural network using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

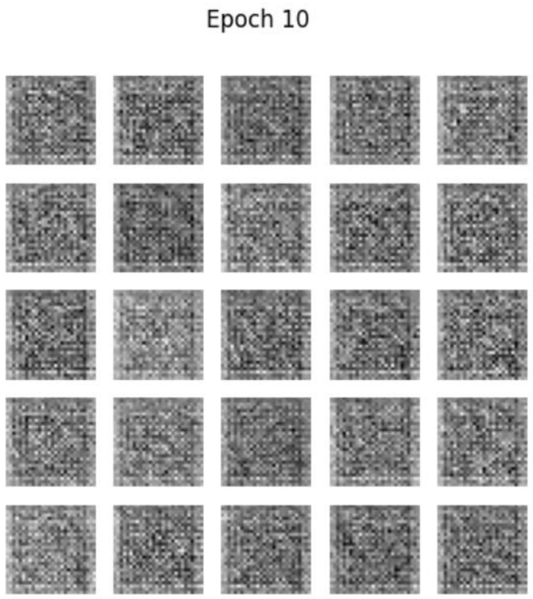
PROGRAM:

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Reshape, Flatten, BatchNormalization, LeakyReLU
from tensorflow.keras.layers import Conv2D, Conv2DTranspose
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

(x_train, _), (_, _) = mnist.load_data()
x_train = (x_train - 127.5) / 127.5
x_train = np.expand_dims(x_train, axis=-1)

latent_dim = 100
batch_size = 128
epochs = 10
sample_interval = 2

def build_generator():
    model = Sequential()
    model.add(Dense(7 * 7 * 128, input_dim=latent_dim))
    model.add(Reshape((7, 7, 128)))
    model.add(BatchNormalization())
    model.add(Conv2DTranspose(128, kernel_size=4, strides=2, padding="same",
activation='relu'))
    model.add(BatchNormalization())
    model.add(Conv2DTranspose(64, kernel_size=4, strides=2, padding="same",
activation='relu'))
    model.add(BatchNormalization())
    model.add(Conv2D(1, kernel_size=7, activation='tanh', padding="same"))
    return model

def build_discriminator():
    model = Sequential()
    model.add(Conv2D(64, kernel_size=3, strides=2, input_shape=(28, 28, 1),
padding="same"))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model

discriminator = build_discriminator()
discriminator.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])

generator = build_generator()

discriminator.trainable = False
gan = Sequential([generator, discriminator])
gan.compile(optimizer='adam', loss='binary_crossentropy')

def train_gan():
    real = np.ones((batch_size, 1))
    fake = np.zeros((batch_size, 1))

    for epoch in range(epochs):
        idx = np.random.randint(0, x_train.shape[0], batch_size)
        real_images = x_train[idx]
        noise = np.random.normal(0, 1, (batch_size, latent_dim))
        generated_images = generator.predict(noise)

        d_loss_real = discriminator.train_on_batch(real_images, real)
        d_loss_fake = discriminator.train_on_batch(generated_images, fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)


        noise = np.random.normal(0, 1, (batch_size, latent_dim))
        g_loss = gan.train_on_batch(noise, real)


        if (epoch + 1) % sample_interval == 0:
            print(f"{epoch + 1}/{epochs}, D loss: {d_loss[0]}, D accuracy: {100 *
d_loss[1]:.2f}%, G loss: {g_loss}")
            sample_images(epoch + 1)


def sample_images(epoch):
    noise = np.random.normal(0, 1, (25, latent_dim))
    generated_images = generator.predict(noise)


    generated_images = 0.5 * generated_images + 0.5


    plt.figure(figsize=(5, 5))
    for i in range(25):
        plt.subplot(5, 5, i + 1)
        plt.imshow(generated_images[i, :, :, 0], cmap='gray')
        plt.axis('off')
    plt.suptitle(f"Epoch {epoch}")
    plt.show()

train_gan()
```

OUTPUT:



```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
10/10, D loss: 0.6987324357032776, D accuracy: 31.07%, G loss: [array(0.69915515, dtype=float32), array(0.69915515, dtype=float32), array(0.30273438, dtype=float32)]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
```



RESULT:

Thus, a generative adversarial neural network using Keras/TensorFlow was successfully implemented.