**Generation of image using GAN**

**Source code:**

import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

# Define the generator model

def build\_generator(latent\_dim):

model = models.Sequential()

model.add(layers.Dense(256, input\_dim=latent\_dim))

model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.BatchNormalization())

model.add(layers.Dense(512))

model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.BatchNormalization())

model.add(layers.Dense(1024))

model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.BatchNormalization())

model.add(layers.Dense(784, activation='tanh'))

model.add(layers.Reshape((28, 28, 1)))

return model

# Define the discriminator model

def build\_discriminator(input\_shape):

model = models.Sequential()

model.add(layers.Flatten(input\_shape=input\_shape))

model.add(layers.Dense(512))

model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.Dense(256))

model.add(layers.LeakyReLU(alpha=0.2))

model.add(layers.Dense(1, activation='sigmoid'))

return model

# Define the GAN model

def build\_gan(generator, discriminator):

discriminator.trainable = False

model = models.Sequential()

model.add(generator)

model.add(discriminator)

return model

# Load and preprocess the dataset (e.g., MNIST)

(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

# Set hyperparameters

latent\_dim = 100

epochs = 10000

batch\_size = 128

# Build and compile the discriminator

discriminator = build\_discriminator((28, 28, 1))

discriminator.compile(loss='binary\_crossentropy', optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0002, beta\_1=0.5))

# Build the generator

generator = build\_generator(latent\_dim)

# Build and compile the GAN

gan = build\_gan(generator, discriminator)

gan.compile(loss='binary\_crossentropy', optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0002, beta\_1=0.5))

# Training loop

for epoch in range(epochs):

# Sample random points in the latent space

random\_latent\_vectors = np.random.normal(size=(batch\_size, latent\_dim))

# Decode them to fake images

generated\_images = generator.predict(random\_latent\_vectors)

# Combine them with real images

real\_images = train\_images[np.random.randint(0, train\_images.shape[0], batch\_size)]

combined\_images = np.concatenate([generated\_images, real\_images])

# Assemble labels discriminating real from fake images

labels = np.concatenate([np.ones((batch\_size, 1)), np.zeros((batch\_size, 1))])

labels += 0.05 \* np.random.random(labels.shape)

# Train the discriminator

d\_loss = discriminator.train\_on\_batch(combined\_images, labels)

# Sample random points in the latent space

random\_latent\_vectors = np.random.normal(size=(batch\_size, latent\_dim))

# Assemble labels that say "all real images"

misleading\_targets = np.zeros((batch\_size, 1))

# Train the generator (via the GAN model)

a\_loss = gan.train\_on\_batch(random\_latent\_vectors, misleading\_targets)

# Print the progress

if epoch % 100 == 0:

print(f'Epoch: {epoch}, Discriminator Loss: {d\_loss}, Generator Loss: {a\_loss}')