**Implement Generative Adversarial Networks to generate realistic Images. Use MNIST, Fashion MNIST or any human face datasets**

# Import required libraries

import tensorflow as tf

from tensorflow.keras import layers

import numpy as np

import matplotlib.pyplot as plt

# Load MNIST dataset

(x\_train, \_), (\_, \_) = tf.keras.datasets.mnist.load\_data()

x\_train = x\_train / 255.0 # Normalize the data to [0, 1]

x\_train = np.expand\_dims(x\_train, axis=-1) # Add a channel dimension

# Set parameters

BUFFER\_SIZE = 60000

BATCH\_SIZE = 128

NOISE\_DIM = 100

EPOCHS = 5

# Prepare the dataset

dataset = tf.data.Dataset.from\_tensor\_slices(x\_train).shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE)

# Generator model

def build\_generator():

model = tf.keras.Sequential([

layers.Dense(7\*7\*256, use\_bias=False, input\_shape=(NOISE\_DIM,)),

layers.BatchNormalization(),

layers.LeakyReLU(),

layers.Reshape((7, 7, 256)),

layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use\_bias=False),

layers.BatchNormalization(),

layers.LeakyReLU(),

layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use\_bias=False),

layers.BatchNormalization(),

layers.LeakyReLU(),

layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use\_bias=False, activation='tanh')

])

return model

# Discriminator model

def build\_discriminator():

model = tf.keras.Sequential([

layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input\_shape=[28, 28, 1]),

layers.LeakyReLU(),

layers.Dropout(0.3),

layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'),

layers.LeakyReLU(),

layers.Dropout(0.3),

layers.Flatten(),

layers.Dense(1)

])

return model

# Instantiate generator and discriminator

generator = build\_generator()

discriminator = build\_discriminator()

# Loss and optimizers

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\_optimizer = tf.keras.optimizers.Adam(1e-4)

discriminator\_optimizer = tf.keras.optimizers.Adam(1e-4)

# Training step

@tf.function

def train\_step(images):

noise = tf.random.normal([BATCH\_SIZE, NOISE\_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\_generator = gen\_tape.gradient(gen\_loss, generator.trainable\_variables)

gradients\_of\_discriminator = disc\_tape.gradient(disc\_loss, discriminator.trainable\_variables)

generator\_optimizer.apply\_gradients(zip(gradients\_of\_generator, generator.trainable\_variables))

discriminator\_optimizer.apply\_gradients(zip(gradients\_of\_discriminator, discriminator.trainable\_variables))

# Training loop

def train(dataset, epochs):

for epoch in range(epochs):

for image\_batch in dataset:

train\_step(image\_batch)

# Generate and display a sample image

generate\_and\_save\_images(generator, epoch + 1)

# Function to generate and save images

def generate\_and\_save\_images(model, epoch, num\_examples=16):

noise = tf.random.normal([num\_examples, NOISE\_DIM])

generated\_images = model(noise, training=False)

fig = plt.figure(figsize=(4, 4))

for i in range(num\_examples):

plt.subplot(4, 4, i + 1)

plt.imshow(generated\_images[i, :, :, 0] \* 127.5 + 127.5, cmap='gray')

plt.axis('off')

plt.show()

# Train the GAN

train(dataset, EPOCHS)