# Ex No: 9 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 simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

# PROGRAM:

!pip install tensorflow tensorflow-gpu matplotlib tensorflow-datasets ipywidgets

!pip list

# Bringing in tensorflow import tensorflow as tf

gpus = tf.config.experimental.list\_physical\_devices('GPU') for gpu in gpus:

tf.config.experimental.set\_memory\_growth(gpu, True) # Brining in tensorflow datasets for fashion mnist

import tensorflow\_datasets as tfds

# Bringing in matplotlib for viz stuff from matplotlib import pyplot as plt

# Use the tensorflow datasets api to bring in the data source ds = tfds.load('fashion\_mnist', split='train') ds.as\_numpy\_iterator().next()['label']

# Do some data transformation import numpy as np

# Setup connection aka iterator dataiterator = ds.as\_numpy\_iterator() # Getting data out of the pipeline

dataiterator.next()['image']

# Setup the subplot formatting

fig, ax = plt.subplots(ncols=4, figsize=(20,20)) # Loop four times and get images

for idx in range(4):

# Grab an image and label sample = dataiterator.next()

# Plot the image using a specific subplot ax[idx].imshow(np.squeeze(sample['image'])) # Appending the image label as the plot title ax[idx].title.set\_text(sample['label'])

# Scale and return images only def scale\_images(data):

image = data['image'] return image / 255

# Reload the dataset

ds = tfds.load('fashion\_mnist', split='train')

# Running the dataset through the scale\_images preprocessing step ds = ds.map(scale\_images)

# Cache the dataset for that batch ds = ds.cache()

# Shuffle it up

ds = ds.shuffle(60000)

# Batch into 128 images per sample ds = ds.batch(128)

# Reduces the likelihood of bottlenecking ds = ds.prefetch(64) ds.as\_numpy\_iterator().next().shape

# Bring in the sequential api for the generator and discriminator from tensorflow.keras.models import Sequential

# Bring in the layers for the neural network

from tensorflow.keras.layers import Conv2D, Dense, Flatten, Reshape, LeakyReLU, Dropout,

UpSampling2D

def build\_generator(): model = Sequential()

# Takes in random values and reshapes it to 7x7x128 # Beginnings of a generated image model.add(Dense(7\*7\*128, input\_dim=128)) model.add(LeakyReLU(0.2)) model.add(Reshape((7,7,128)))

# Upsampling block 1 model.add(UpSampling2D()) model.add(Conv2D(128, 5, padding='same')) model.add(LeakyReLU(0.2))

# Upsampling block 2 model.add(UpSampling2D()) model.add(Conv2D(128, 5, padding='same')) model.add(LeakyReLU(0.2))

# Convolutional block 1 model.add(Conv2D(128, 4, padding='same')) model.add(LeakyReLU(0.2))

# Convolutional block 2 model.add(Conv2D(128, 4, padding='same')) model.add(LeakyReLU(0.2))

# Conv layer to get to one channel

model.add(Conv2D(1, 4, padding='same', activation='sigmoid'))

return model

generator = build\_generator() generator.summary()

img = generator.predict(np.random.randn(4,128,1)) # Generate new fashion

img = generator.predict(np.random.randn(4,128,1)) # Setup the subplot formatting

fig, ax = plt.subplots(ncols=4, figsize=(20,20)) # Loop four times and get images

for idx, img in enumerate(img):

# Plot the image using a specific subplot ax[idx].imshow(np.squeeze(img))

# Appending the image label as the plot title ax[idx].title.set\_text(idx)

def build\_discriminator(): model = Sequential()

# First Conv Block

model.add(Conv2D(32, 5, input\_shape = (28,28,1))) model.add(LeakyReLU(0.2)) model.add(Dropout(0.4))

# Second Conv Block model.add(Conv2D(64, 5)) model.add(LeakyReLU(0.2)) model.add(Dropout(0.4))

# Third Conv Block model.add(Conv2D(128, 5))

model.add(LeakyReLU(0.2)) model.add(Dropout(0.4))

# Fourth Conv Block model.add(Conv2D(256, 5)) model.add(LeakyReLU(0.2)) model.add(Dropout(0.4))

# Flatten then pass to dense layer model.add(Flatten()) model.add(Dropout(0.4)) model.add(Dense(1, activation='sigmoid'))

return model

discriminator = build\_discriminator() discriminator.summary()

img = img[0] img.shape

discriminator.predict(img)

# Adam is going to be the optimizer for both from tensorflow.keras.optimizers import Adam

# Binary cross entropy is going to be the loss for both from tensorflow.keras.losses import BinaryCrossentropy g\_opt = Adam(learning\_rate=0.0001)

d\_opt = Adam(learning\_rate=0.00001) g\_loss = BinaryCrossentropy()

d\_loss = BinaryCrossentropy()

# Importing the base model class to subclass our training step

from tensorflow.keras.models import Model class FashionGAN(Model):

def init (self, generator, discriminator, \*args, \*\*kwargs): # Pass through args and kwargs to base class

super(). init (\*args, \*\*kwargs)

# Create attributes for gen and disc self.generator = generator self.discriminator = discriminator

def compile(self, g\_opt, d\_opt, g\_loss, d\_loss, \*args, \*\*kwargs): # Compile with base class

super().compile(\*args, \*\*kwargs)

# Create attributes for losses and optimizers self.g\_opt = g\_opt

self.d\_opt = d\_opt self.g\_loss = g\_loss self.d\_loss = d\_loss

def train\_step(self, batch): # Get the data real\_images = batch

fake\_images = self.generator(tf.random.normal((128, 128, 1)), training=False)

# Train the discriminator

with tf.GradientTape() as d\_tape:

# Pass the real and fake images to the discriminator model yhat\_real = self.discriminator(real\_images, training=True) yhat\_fake = self.discriminator(fake\_images, training=True) yhat\_realfake = tf.concat([yhat\_real, yhat\_fake], axis=0)

# Create labels for real and fakes images

y\_realfake = tf.concat([tf.zeros\_like(yhat\_real), tf.ones\_like(yhat\_fake)], axis=0)

# Add some noise to the TRUE outputs

noise\_real = 0.15\*tf.random.uniform(tf.shape(yhat\_real)) noise\_fake = -0.15\*tf.random.uniform(tf.shape(yhat\_fake)) y\_realfake += tf.concat([noise\_real, noise\_fake], axis=0)

# Calculate loss - BINARYCROSS

total\_d\_loss = self.d\_loss(y\_realfake, yhat\_realfake)

# Apply backpropagation - nn learn

dgrad = d\_tape.gradient(total\_d\_loss, self.discriminator.trainable\_variables) self.d\_opt.apply\_gradients(zip(dgrad, self.discriminator.trainable\_variables))

# Train the generator

with tf.GradientTape() as g\_tape: # Generate some new images

gen\_images = self.generator(tf.random.normal((128,128,1)), training=True)

# Create the predicted labels

predicted\_labels = self.discriminator(gen\_images, training=False)

# Calculate loss - trick to training to fake out the discriminator

total\_g\_loss = self.g\_loss(tf.zeros\_like(predicted\_labels), predicted\_labels)

# Apply backprop

ggrad = g\_tape.gradient(total\_g\_loss, self.generator.trainable\_variables) self.g\_opt.apply\_gradients(zip(ggrad, self.generator.trainable\_variables))

return {"d\_loss":total\_d\_loss, "g\_loss":total\_g\_loss} # Create instance of subclassed model

fashgan = FashionGAN(generator, discriminator) # Compile the model

fashgan.compile(g\_opt, d\_opt, g\_loss, d\_loss)

import os

from tensorflow.keras.preprocessing.image import array\_to\_img from tensorflow.keras.callbacks import Callback

class ModelMonitor(Callback):

def init (self, num\_img=3, latent\_dim=128): self.num\_img = num\_img

self.latent\_dim = latent\_dim

def on\_epoch\_end(self, epoch, logs=None):

random\_latent\_vectors = tf.random.uniform((self.num\_img, self.latent\_dim,1)) generated\_images = self.model.generator(random\_latent\_vectors) generated\_images \*= 255

generated\_images.numpy() for i in range(self.num\_img):

img = array\_to\_img(generated\_images[i]) img.save(os.path.join('images', f'generated\_img\_{epoch}\_{i}.png'))

# Recommend 2000 epochs

hist = fashgan.fit(ds, epochs=20, callbacks=[ModelMonitor()])

plt.suptitle('Loss') plt.plot(hist.history['d\_loss'], label='d\_loss') plt.plot(hist.history['g\_loss'], label='g\_loss') plt.legend()

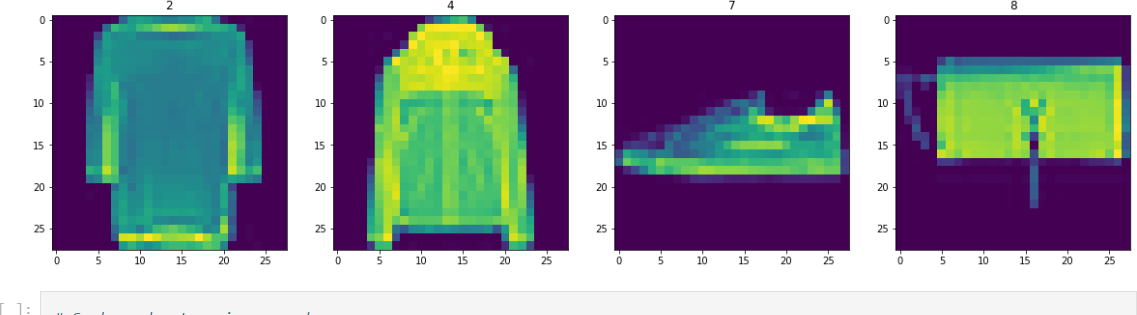
plt.show()

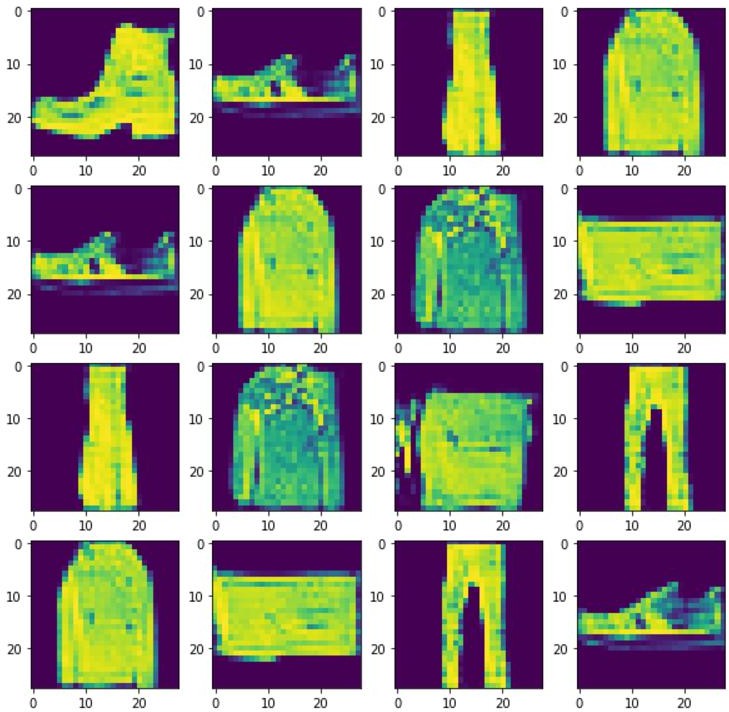
generator.load\_weights(os.path.join('archive', 'generatormodel.h5')) imgs = generator.predict(tf.random.normal((16, 128, 1)))

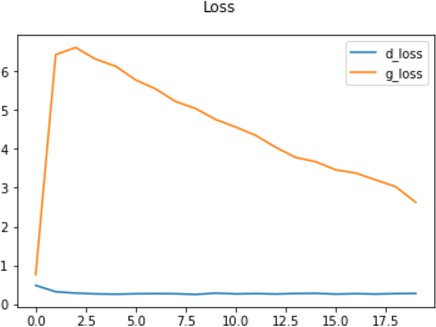
fig, ax = plt.subplots(ncols=4, nrows=4, figsize=(10,10)) for r in range(4):

for c in range(4): ax[r][c].imshow(imgs[(r+1)\*(c+1)-1])

generator.save('generator.h5') discriminator.save('discriminator.h5') **OUTPUT:**







# RESULT:

Thus a generative adversarial neural network using Keras/TensorFlow is built.