ASSIGNMENT No: 05

Title: Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Problem Statement: Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Prerequisite:

Basics of Python

Software Requirements: Jupyter

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Outcomes:

After completion of this assignment students are able to Design and implement Deep Convolutional GAN to generate images of faces/digits from a set of given images.

Theory:

Deep Convolutional Generative Adversarial Network (DCGAN):

Deep Convolutional Generative Adversarial Network (DCGAN) represents a breakthrough in generative models, specifically designed for image generation tasks. DCGANs are an extension of the traditional GAN architecture, tailored for generating high-quality, coherent images.

Architecture:

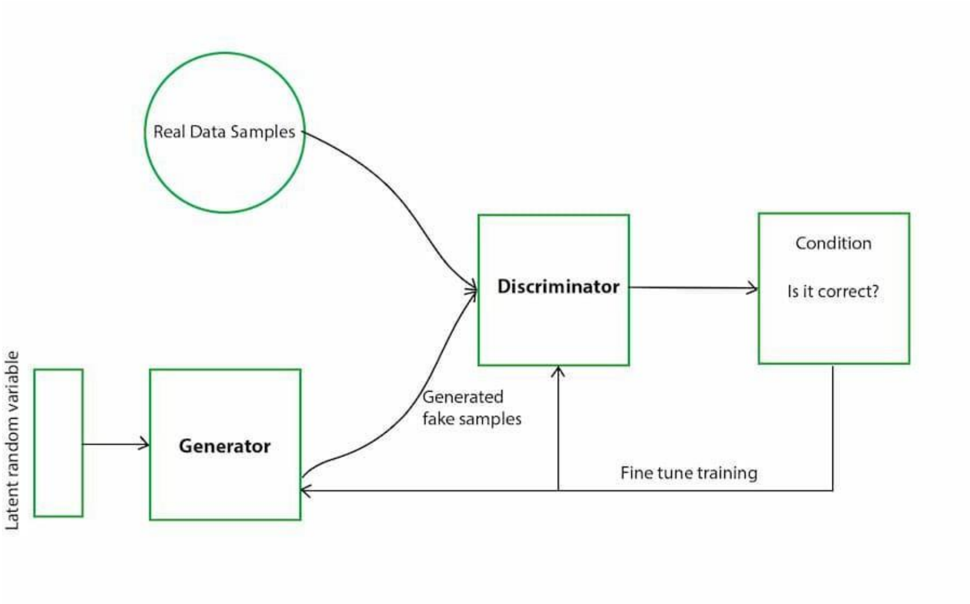
Generator:

The generator is responsible for synthesizing realistic images from random noise. It employs a series of transposed convolutional layers to transform the input noise into a complex image.
The generator's architecture typically consists of fractional-strided convolutions, batch normalization, and rectified linear unit (ReLU) activations. These architectural choices aid in preventing issues like mode collapse and vanishing gradients.

Discriminator:
The discriminator is a binary classifier tasked with distinguishing between real and generated images. It utilizes convolutional layers to extract hierarchical features from the input images. Similar to the generator, batch normalization and Leaky ReLU activations are commonly used in the discriminator to ensure stable training.

Key Design Principles:

Strided convolutions: Enable the network to learn spatial hierarchies effectively.
Batch normalization: Promotes stable and accelerated training by normalizing the input of each layer.
Leaky ReLU activations: Prevents the issue of "dying ReLU" by allowing a small, non-zero gradient for negative input values.
Transposed convolutions: Essential for upsampling the input noise and generating high- resolution images.



Training Process:

Input Noise:

The generator takes random noise as input, typically sampled from a Gaussian distribution. This noise is transformed into a synthetic image.

Adversarial Training:

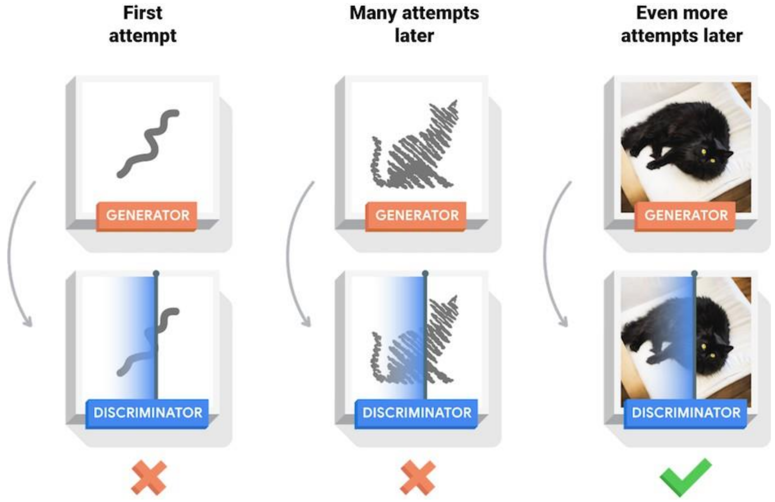
The discriminator evaluates both real and generated images, providing feedback to the generator. The generator aims to create images that are indistinguishable from real ones.

Discriminator Feedback:

The discriminator is trained to correctly classify real and generated images. It learns to distinguish subtle patterns and features in the images.

Generator Improvement:

The generator adjusts its parameters based on the feedback from the discriminator. This adversarial process continues iteratively, leading to the refinement of both networks.



Algorithm -

1. Initialize the generator and discriminator models with the specified architectures
2. Prepare the dataset for training, ensuring proper normalization and preprocessing.
3. Train the discriminator using real and generated images, updating its parameters.
4. Train the generator to produce realistic images that can deceive the discriminator
5. Iterate between discriminator and generator training to refine their capabilities over epochs.

Application -

1. Artistic image generation.
2. Image-to-image translation tasks.
3. Data augmentation processes.

Inference -

The GAN experiment involves training a generator and discriminator, optimizing their performance. The generator creates synthetic images, while the discriminator distinguishes between real and fake. Training alternates, aiming for the generator to produce realistic images. The final model can generate diverse images from noise, showcasing the GAN's ability in image synthesis.

Code -

```
from keras.models import Model from
keras.layers import Input,Dense import
numpy as np
import pandas as pd import
keras.backend as K
import matplotlib.pyplot as plt from
keras import preprocessing
from keras.models import Sequential #from
keras.layers import
Conv2D,Dropout,Dense,Flatten,Conv2DTranspose,BatchNormalization,LeakyReLU,Reshape  import
tensorflow as tf
from keras.layers import *
from keras.datasets import fashion_mnist
(train_x, train_y), (val_x, val_y) = fashion_mnist.load_data() train_x
= train_x/255.
val_x = val_x/255.
train_x=train_x.reshape(-1,28,28,1)
print(train_x.shape)
#train_x = train_x.reshape(-1, 784)
#val_x = val_x.reshape(-1, 784)
fig,axe=plt.subplots(2,2)
idx = 0
for i in range(2): for
    j in range(2):
        axe[i,j].imshow(train_x[idx].reshape(28,28),cmap='gray') idx+=1
train_x = train_x*2 - 1
print(train_x.max(),train_x.min()) generator
= Sequential()
generator.add(Dense(512,input_shape=[100]))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(256))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(128))
generator.add(LeakyReLU(alpha=0.2))
generator.add(BatchNormalization(momentum=0.8))
generator.add(Dense(784)) generator.add(Reshape([28,28,1]))
generator.summary()
discriminator = Sequential()
discriminator.add(Dense(1,input_shape=[28,28,1]))
```

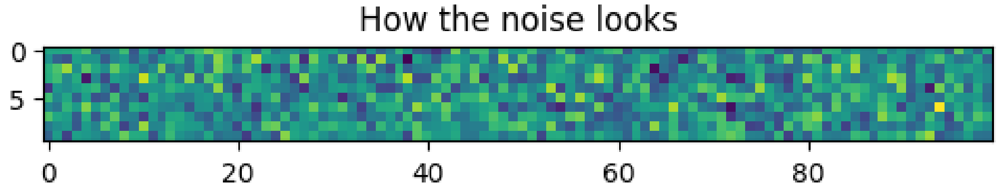
```
discriminator.add(Flatten()) discriminator.add(Dense(256))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(128))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(64))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.5))
discriminator.add(Dense(1,activation='sigmoid'))
discriminator.summary()
GAN =Sequential([generator,discriminator])
discriminator.compile(optimizer='adam',loss='binary_crossentropy')
discriminator.trainable = False
GAN.compile(optimizer='adam',loss='binary_crossentropy') GAN.summary()
epochs = 30
batch_size = 100
noise_shape=100
with tf.device('/gpu:0'):
 for epoch in range(epochs):
    print(f"Currently on Epoch {epoch+1}")

    for i in range(train_x.shape[0]//batch_size): if
        (i+1)%100 == 0:
            print(f"\tCurrently on batch number {i+1} of {train_x.shape[0]//batch_size}")

        noise=np.random.normal(size=[batch_size,noise_shape])

        gen_image = generator.predict_on_batch(noise)

        train_dataset = train_x[i*batch_size:(i+1)*batch_size]

        #training discriminator on real images
        train_label=np.ones(shape=(batch_size,1))
        #train_label=np.ones((batch_size, 1))
        discriminator.trainable = True
        #train_dataset=train_x[idx]
        d_loss_real=discriminator.train_on_batch(train_dataset,train_label) #training
        discriminator on fake images train_label=np.zeros(shape=(batch_size,1))
        d_loss_fake=discriminator.train_on_batch(gen_image,train_label)

        #training  generator

        noise=np.random.normal(size=[batch_size,noise_shape])
        train_label=np.ones(shape=(batch_size,1))
        discriminator.trainable = False

        d_g_loss_batch =GAN.train_on_batch(noise, train_label)

    #plotting generated images at the start and then after every 10 epoch if epoch
    % 10 == 0:
```

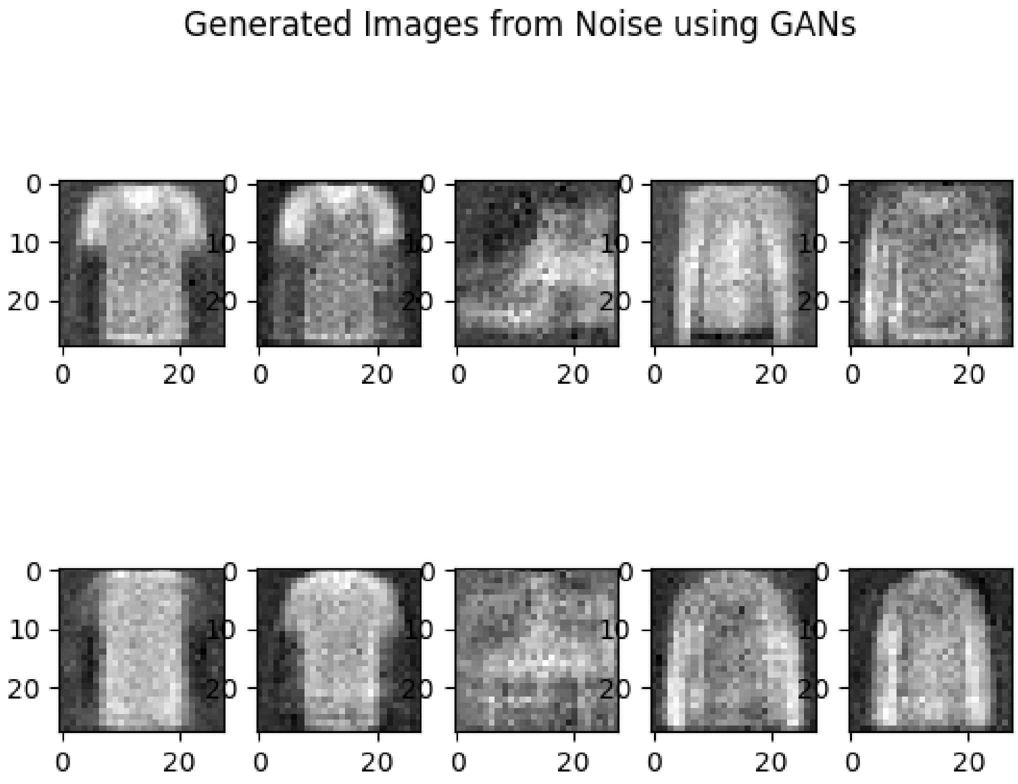
```
    samples = 10
    x_fake = generator.predict(np.random.normal(loc=0, scale=1, size=(samples, 100)))

    for k in range(samples):
      plt.subplot(2, 5, k+1)
      plt.imshow(x_fake[k].reshape(28, 28), cmap='gray')
      plt.xticks([])
      plt.yticks([])

    plt.tight_layout()
    plt.show()

print('Training is complete')
noise=np.random.normal(size=[10,noise_shape]) gen_image
= generator.predict(noise) plt.imshow(noise)
plt.title('How the noise looks')
fig,axe=plt.subplots(2,5)
fig.suptitle('Generated Images from Noise using GANs') idx=0
for i in range(2): for
  j in range(5):
     axe[i,j].imshow(gen_image[idx].reshape(28,28),cmap='gray') idx+=1
```

Output –

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)              Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 28, 28, 1)         2

 flatten (Flatten)           (None, 784)               0

 dense_5 (Dense)             (None, 256)               200960

 leaky_re_lu_3 (LeakyReLU)  (None, 256)                0

 dropout (Dropout)           (None, 256)               0

 dense_6 (Dense)             (None, 128)               32896

 leaky_re_lu_4 (LeakyReLU)  (None, 128)                0

 dropout_1 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 64)                8256
```

```
 leaky_re_lu_5 (LeakyReLU)  (None, 64)              0

 dropout_2 (Dropout)        (None, 64)              0

 dense_8 (Dense)            (None, 1)               65

=================================================================
Total params: 242179 (946.01 KB)
Trainable params: 242179 (946.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________


Model: "sequential_2"
_________________________________________________________________
 Layer (type)               Output Shape            Param #
=================================================================
 sequential (Sequential)    (None, 28, 28, 1)       320656

 sequential_1 (Sequential)  (None, 1)               242179

=================================================================
Total params: 562835 (2.15 MB)
Trainable params: 318864 (1.22 MB)
Non-trainable params: 243971 (953.01 KB)
_________________________________________________________________


        Currently on batch number 600 of 600
Currently on Epoch 30
        Currently on batch number 100 of 600
        Currently on batch number 200 of 600
        Currently on batch number 300 of 600
        Currently on batch number 400 of 600
        Currently on batch number 500 of 600
        Currently on batch number 600 of 600
Training is complete
```





References-

https://www.tensorflow.org/tutorials/generative/dcgan

Conclusion: Thus Designed and implemented Deep Convolutional GAN to generate images of faces/digits from a set of given images.