**BY USING GAN GENERATING FAKE FACES**

Import necessary Libraries

import tensorflow as tf

import keras

from keras import layers

import numpy as np

import matplotlib.pyplot as plt

import cv2

import os

from tqdm import tqdm

import re

from keras.preprocessing.image import img\_to\_array

## Load data

def sorted\_alphanumeric(data):

convert = lambda text: int(text) if text.isdigit() else text.lower()

alphanum\_key = lambda key: [convert(c) for c **in** re.split('([0-9]+)',key)]

return sorted(data,key = alphanum\_key)

*# defining the size of the image*

SIZE = 128

\_img = []

path = '../input/face-mask-lite-dataset/without\_mask'

files = os.listdir(path)

files = sorted\_alphanumeric(files)

for i **in** tqdm(files):

if i == 'seed9090.png':

break

else:

img = cv2.imread(path + '/'+i,1)

*# open cv reads images in BGR format so we have to convert it to RGB*

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

*#resizing image*

img = cv2.resize(img, (SIZE, SIZE))

img = (img - 127.5) / 127.5

imh = img.astype(float)

\_img.append(img\_to\_array(img))

91%|█████████ | 9090/10000 [07:55<00:47, 19.10it/s]

## Visailze our images

def plot\_images(sqr = 5):

plt.figure(figsize = (10,10))

plt.title("Real Images",fontsize = 35)

for i **in** range(sqr \* sqr):

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

plt.imshow(\_img[i]\*0.5 + 0.5 )

plt.xticks([])

plt.yticks([])

*# to plot images*

plot\_images(6)

## 

Here, i have defined batch size so that these batches of images can be fed directly to the discriminator network.

batch\_size = 32

dataset=tf.data.Dataset.from\_tensor\_slices(np.array(\_img)).batch(batch\_size)

# Generator

latent\_dim = 100

def Generator():

model = tf.keras.Sequential()

model.add(layers.Dense(128\*128\*3, use\_bias=False, input\_shape=(latent\_dim,)))

model.add(layers.Reshape((128,128,3)))

*# downsampling*

model.add(tf.keras.layers.Conv2D(128,4, strides=1, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.Conv2D(128,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2D(256,4, strides=1, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.Conv2D(256,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2DTranspose(512, 4, strides=1,padding='same',kernel\_initializer='he\_normal',use\_bias=False))

model.add(tf.keras.layers.Conv2D(512,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.LeakyReLU())

*#upsampling*

model.add(tf.keras.layers.Conv2DTranspose(512, 4, strides=1,padding='same',kernel\_initializer='he\_normal',use\_bias=False))

model.add(tf.keras.layers.Conv2DTranspose(512, 4, strides=2,padding='same',kernel\_initializer='he\_normal',use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2DTranspose(256, 4, strides=1,padding='same',kernel\_initializer='he\_normal',use\_bias=False))

model.add(tf.keras.layers.Conv2DTranspose(256, 4, strides=2,padding='same',kernel\_initializer='he\_normal',use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2DTranspose(128, 4, strides=2,padding='same',kernel\_initializer='he\_normal',use\_bias=False))

model.add(tf.keras.layers.Conv2DTranspose(128, 4, strides=1,padding='same',kernel\_initializer='he\_normal',use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2DTranspose(3,4,strides = 1, padding = 'same',activation = 'tanh'))

return model

generator = Generator()

generator.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense (Dense) (None, 49152) 4915200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

reshape (Reshape) (None, 128, 128, 3) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d (Conv2D) (None, 128, 128, 128) 6144

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 64, 64, 128) 262144

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization (BatchNo (None, 64, 64, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu (LeakyReLU) (None, 64, 64, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 64, 64, 256) 524288

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 32, 32, 256) 1048576

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_1 (Batch (None, 32, 32, 256) 1024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_1 (LeakyReLU) (None, 32, 32, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose (Conv2DTran (None, 32, 32, 512) 2097152

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 16, 16, 512) 4194304

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_2 (LeakyReLU) (None, 16, 16, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_1 (Conv2DTr (None, 16, 16, 512) 4194304

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_2 (Conv2DTr (None, 32, 32, 512) 4194304

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_2 (Batch (None, 32, 32, 512) 2048

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_3 (LeakyReLU) (None, 32, 32, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_3 (Conv2DTr (None, 32, 32, 256) 2097152

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_4 (Conv2DTr (None, 64, 64, 256) 1048576

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_3 (Batch (None, 64, 64, 256) 1024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_5 (Conv2DTr (None, 128, 128, 128) 524288

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_6 (Conv2DTr (None, 128, 128, 128) 262144

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_4 (Batch (None, 128, 128, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_transpose\_7 (Conv2DTr (None, 128, 128, 3) 6147

=================================================================

Total params: 25,379,843

Trainable params: 25,377,283

Non-trainable params: 2,560

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Discriminator

def Discriminator():

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Input((SIZE, SIZE, 3)))

model.add(tf.keras.layers.Conv2D(128,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2D(128,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2D(256,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2D(256,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Conv2D(512,4, strides=2, padding='same',kernel\_initializer='he\_normal', use\_bias=False))

model.add(tf.keras.layers.LeakyReLU())

model.add(tf.keras.layers.Flatten())

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

return model

discriminator = Discriminator()

discriminator.summary()

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_5 (Conv2D) (None, 64, 64, 128) 6144

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_5 (Batch (None, 64, 64, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_4 (LeakyReLU) (None, 64, 64, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_6 (Conv2D) (None, 32, 32, 128) 262144

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_6 (Batch (None, 32, 32, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_5 (LeakyReLU) (None, 32, 32, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_7 (Conv2D) (None, 16, 16, 256) 524288

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_7 (Batch (None, 16, 16, 256) 1024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_6 (LeakyReLU) (None, 16, 16, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_8 (Conv2D) (None, 8, 8, 256) 1048576

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_8 (Batch (None, 8, 8, 256) 1024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_7 (LeakyReLU) (None, 8, 8, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_9 (Conv2D) (None, 4, 4, 512) 2097152

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

leaky\_re\_lu\_8 (LeakyReLU) (None, 4, 4, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 8192) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 1) 8193

=================================================================

Total params: 3,949,569

Trainable params: 3,948,033

Non-trainable params: 1,536

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### plot image generated by generator before training

noise = np.random.normal(-1,1,(1,100))

img = generator(noise)

plt.imshow(img[0,:,:,0])

plt.show()

### Defining loss function and optimizer

optimizer = tf.keras.optimizers.RMSprop(

lr=.0001,

clipvalue=1.0,

decay=1e-8

)

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(fake\_output, real\_output):

fake\_loss = cross\_entropy(tf.zeros\_like(fake\_output),fake\_output)

real\_loss = cross\_entropy(tf.ones\_like(real\_output),real\_output)

return fake\_loss + real\_loss

### Defining training steps

def train\_steps(images):

noise = np.random.normal(0,1,(batch\_size,latent\_dim))

with tf.GradientTape() as gen\_tape , tf.GradientTape() as disc\_tape:

generated\_images = generator(noise)

fake\_output = discriminator(generated\_images)

real\_output = discriminator(images)

gen\_loss = generator\_loss(fake\_output)

dis\_loss = discriminator\_loss(fake\_output, real\_output)

gradient\_of\_generator = gen\_tape.gradient(gen\_loss, generator.trainable\_variables)

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

optimizer.apply\_gradients(zip(gradient\_of\_generator,generator.trainable\_variables))

optimizer.apply\_gradients(zip(gradient\_of\_discriminator, discriminator.trainable\_variables))

loss = {'gen loss':gen\_loss,

'disc loss': dis\_loss}

return loss

## function to plot generated images

def plot\_generated\_images(square = 5, epochs = 0):

plt.figure(figsize = (10,10))

for i **in** range(square \* square):

if epochs != 0:

if(i == square //2):

plt.title("Generated Image at Epoch:**{}\n**".format(epochs), fontsize = 32, color = 'black')

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

noise = np.random.normal(0,1,(1,latent\_dim))

img = generator(noise)

plt.imshow(np.clip((img[0,...]+1)/2, 0, 1))

plt.xticks([])

plt.yticks([])

plt.grid()

import time

def train(epochs,dataset):

for epoch **in** range(epochs):

start = time.time()

print("**\n**Epoch : **{}**".format(epoch + 1))

for images **in** dataset:

loss = train\_steps(images)

print(" Time:**{}**".format(np.round(time.time() - start),2))

print("Generator Loss: **{}** Discriminator Loss: **{}**".format(loss['gen loss'],loss['disc loss']))

## Training

train(5,dataset)

**OUTPUT**

plot\_generated\_images(1)



plot\_generated\_images(2)



plot\_generated\_images(5)



plot\_generated\_images(7)

  
generator.save('generator.h5')

discriminator.save("discriminator.h5")