Assignment 4 :

In [1]:

```
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time
from tensorflow.keras import layers
from tensorflow.keras import activations
from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy import vstack
from numpy.random import randn
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import Dropout
from matplotlib import pyplot
from keras.layers import UpSampling2D
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import BatchNormalization
from IPython import display
```

In [2]:

```
(train_images, train_labels), (_, _) = 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  # Normalize the images to [-1, 1]
BUFFER_SIZE = 60000
BATCH_SIZE = 192
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).ba
tch(BATCH_SIZE)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.n
pz
11490434/11490434 [==============================] - 0s 0us/step
```

In [3]:

```
def make_generator_model():
    model = Sequential()
    n_nodes = 7 * 7 * 192
    model.add(Dense(n_nodes, input_dim=100))
    model.add(BatchNormalization())
    model.add(layers.Activation(activations.relu))
    model.add(Reshape((7, 7, 192)))
    model.add(Dropout(0.4))
    model.add(UpSampling2D(2))
    model.add(Conv2DTranspose(96, (5,5), strides=1, padding='same'))
    model.add(BatchNormalization())
    model.add(layers.Activation(activations.relu))
    model.add(UpSampling2D(2))
    model.add(Conv2DTranspose(48, (5,5), strides=1, padding='same'))
```

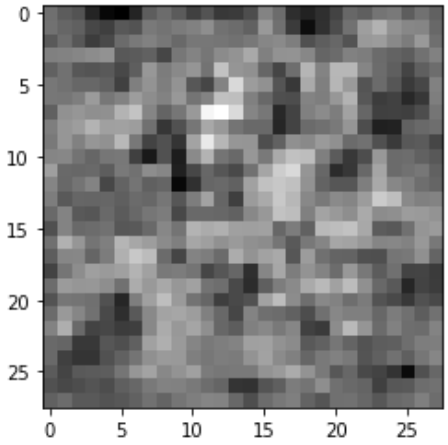
```
    model.add(BatchNormalization())
    model.add(layers.Activation(activations.relu))
    model.add(Conv2DTranspose(24, (5,5), strides=1, padding='same'))
    model.add(BatchNormalization())
    model.add(layers.Activation(activations.relu))
    model.add(Conv2DTranspose(1, (5,5), strides=1, padding='same'))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

In [4]:

```
generator = make_generator_model()

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

Out[4]:

```
<matplotlib.image.AxesImage at 0x7f3ca17395d0>
```



In [5]:

```
def make_discriminator_model():
    in_shape=(28,28,1)
    model = tf.keras.Sequential()
    model.add(Conv2D(32, (5,5), strides=(2, 2), padding='same', input_shape=in_shape))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Conv2D(64, (5,5), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Conv2D(128, (5,5), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Conv2D(256, (5,5), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

In [6]:

```
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

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)
```

```
    total_loss = real_loss + fake_loss
    return total_loss

def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

```
tf.Tensor([[0.49683115]], shape=(1, 1), dtype=float32)
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWar
ning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

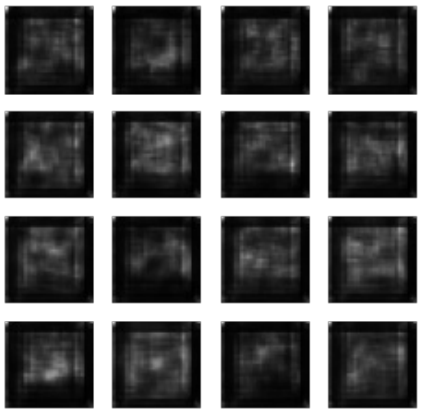
In [7]:

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
EPOCHS = 1
noise_dim = 100
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, noise_dim])

@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_va
riables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_v
ariables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator
.trainable_variables))

def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    display.clear_output(wait=True)
    generate_and_save_images(generator,
                             epoch + 1,
                             seed)

    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  display.clear_output(wait=True)
  generate_and_save_images(generator,
                           epochs,
                           seed)
```

```
def generate_and_save_images(model, epoch, test_input):
  predictions = model(test_input, training=False)

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

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

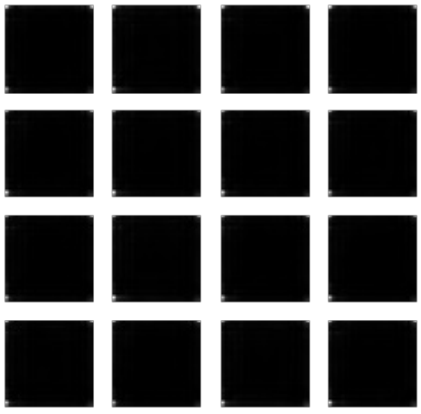
In [8]:

```
train(train_dataset, EPOCHS)
```



In [9]:

```
EPOCHS = 10
train(train_dataset, EPOCHS)
```



In [10]:

```
EPOCHS = 20
train(train_dataset, EPOCHS)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-10-2acff166190e> in <module>
      1 EPOCHS = 20
----> 2 train(train_dataset, EPOCHS)

<ipython-input-7-bab16639e2e4> in train(dataset, epochs)
     42
     43     for image_batch in dataset:
---> 44       train_step(image_batch)
     45
     46     # Produce images for the GIF as you go

/usr/local/lib/python3.7/dist-packages/tensorflow/python/util/traceback_utils.py in error
_handler(*args, **kwargs)
    148     filtered_tb = None
```

```
    149       try:
--> 150         return fn(*args, **kwargs)
    151       except Exception as e:
    152         filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in __call_
_(self, *args, **kwds)
    913
    914       with OptionalXlaContext(self._jit_compile):
--> 915         result = self._call(*args, **kwds)
    916
    917       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/def_function.py in _call(s
elf, *args, **kwds)
    945       # In this case we have created variables on the first call, so we run the
    946       # defunned version which is guaranteed to never create variables.
--> 947       return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    948     elif self._stateful_fn is not None:
    949       # Release the lock early so that multiple threads can perform the call

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in __call__(se
lf, *args, **kwargs)
   2452        filtered_flat_args) = self._maybe_define_function(args, kwargs)
   2453     return graph_function._call_flat(
-> 2454         filtered_flat_args, captured_inputs=graph_function.captured_inputs)  # py
lint: disable=protected-access
   2455
   2456   @property

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in _call_flat(
self, args, captured_inputs, cancellation_manager)
   1859       # No tape is watching; skip to running the function.
   1860       return self._build_call_outputs(self._inference_function.call(
-> 1861           ctx, args, cancellation_manager=cancellation_manager))
   1862     forward_backward = self._select_forward_and_backward_functions(
   1863         args,

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/function.py in call(self,
ctx, args, cancellation_manager)
    500               inputs=args,
    501               attrs=attrs,
--> 502               ctx=ctx)
    503         else:
    504           outputs = execute.execute_with_cancellation(

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in quick_execut
e(op_name, num_outputs, inputs, attrs, ctx, name)
     53     ctx.ensure_initialized()
     54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:
     57     if name is not None:

KeyboardInterrupt:
```

In [ ]:

```
EPOCHS = 50
train(train_dataset, EPOCHS)
```

Source : I used the main code in the tensorflow website and i modify it to concord with the template given by the asigmnent : https://www.tensorflow.org/tutorials/generative/dcgan

i also used this template : https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/

And i found multiple interesting application, used to create different image or other type of media : https://youtu.be/yz6dNf7X7SA

https://www.youtube.com/watch?v=uMN85KBV4Rw&ab_channel=mahmoudafifi

http://prosello.com/papers/text-gans-w17.pdf

I only do 1 and 10 epochs because it took me a entire day to do the 10 one. So i dont think i will be able to make the 20, 50.. epochs. And the printed image are fully black i dont know exactly why.