gan

November 21, 2024

[1]:
```
import numpy as np
import tensorflow as tf
from keras.layers import Dense, Reshape,Flatten,␣
↪Dropout,BatchNormalization,LeakyReLU,Conv2D,Conv2DTranspose,Resizing,Input
from keras.models import Sequential
import matplotlib.pyplot as plt
from keras.datasets import mnist
from tqdm import tqdm
```

[2]:
```
(x_train,_),(_,_)=mnist.load_data()
```

[3]:
```
x_train = (x_train.astype(np.float32)-127.5)/127.5
x_train = np.expand_dims(x_train,axis=-1)
batch_size =8
img_shape = x_train.shape[1:]
z_dim =100
```

[4]:
```
def build_geneartor():
    model = Sequential()
    model.add(Dense(256,input_dim=z_dim))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    model.add(Dense(512))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    model.add(Dense(1024))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    model.add(Dense(np.prod(img_shape),activation='tanh'))
    model.add(Reshape(img_shape))
    model.summary()
    return model
```

[5]:
```
def build_discriminator():
    model = Sequential()
    model.add(Flatten(input_shape=img_shape))

    model.add(Dense(512))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.3))

    model.add(Dense(256))
    model.add(LeakyReLU(0.2))
    model.add(Dropout(0.3))

    model.add(Dense(1,activation='sigmoid'))
    model.summary()
    return model
```

[6]:
```
def build_gan(geneartor,discriminator):
    model = Sequential()
    model.add(geneartor)
    model.add(discriminator)
    model.summary()
    return model
```

[7]:
```
discriminator = build_discriminator()
discriminator.
 ↪compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

generator = build_geneartor()


gan = build_gan(generator,discriminator)
gan.compile(loss='binary_crossentropy',optimizer='adam')
```

```
C:\Users\Om Nagvekar\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0

```
 dense (Dense)                        (None, 512)                 ␣
↪401,920

 leaky_re_lu (LeakyReLU)              (None, 512)                 ␣
↪  0

 dropout (Dropout)                    (None, 512)                 ␣
↪  0

 dense_1 (Dense)                      (None, 256)                 ␣
↪131,328

 leaky_re_lu_1 (LeakyReLU)            (None, 256)                 ␣
↪  0

 dropout_1 (Dropout)                  (None, 256)                 ␣
↪  0

 dense_2 (Dense)                      (None, 1)                   ␣
↪257


 Total params: 533,505 (2.04 MB)

 Trainable params: 533,505 (2.04 MB)

 Non-trainable params: 0 (0.00 B)

C:\Users\Om Nagvekar\AppData\Local\Programs\Python\Python311\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"


 Layer (type)                         Output Shape                ␣
↪Param #

 dense_3 (Dense)                      (None, 256)                 ␣
↪25,856
```

```
 leaky_re_lu_2 (LeakyReLU)              (None, 256)                                 ␣
↪  0

 batch_normalization                    (None, 256)                                 ␣
↪1,024
 (BatchNormalization)                                                               ␣
↪

 dense_4 (Dense)                        (None, 512)                                 ␣
↪131,584

 leaky_re_lu_3 (LeakyReLU)              (None, 512)                                 ␣
↪  0

 batch_normalization_1                  (None, 512)                                 ␣
↪2,048
 (BatchNormalization)                                                               ␣
↪

 dense_5 (Dense)                        (None, 1024)                                ␣
↪525,312

 leaky_re_lu_4 (LeakyReLU)              (None, 1024)                                ␣
↪  0

 batch_normalization_2                  (None, 1024)                                ␣
↪4,096
 (BatchNormalization)                                                               ␣
↪

 dense_6 (Dense)                        (None, 784)                                 ␣
↪803,600

 reshape (Reshape)                      (None, 28, 28, 1)                           ␣
↪  0


 Total params: 1,493,520 (5.70 MB)

 Trainable params: 1,489,936 (5.68 MB)

 Non-trainable params: 3,584 (14.00 KB)

Model: "sequential_2"
```

```
Layer (type)                                 Output Shape                              ␣
↪Param #

sequential_1 (Sequential)                    (None, 28, 28, 1)                         ␣
↪1,493,520

sequential (Sequential)                      (None, 1)                                 ␣
↪533,505


Total params: 2,027,025 (7.73 MB)

Trainable params: 2,023,441 (7.72 MB)

Non-trainable params: 3,584 (14.00 KB)
```

[12]:
```python
def sample_images(epoch, grid_rows=5, grid_columns=5):
    noise = np.random.randn(grid_rows * grid_columns, z_dim)
    gen_imgs = generator.predict(noise)
    gen_imgs = 0.5 * gen_imgs + 0.5

    fig, axs = plt.subplots(grid_rows, grid_columns)
    cnt = 0
    for i in range(grid_rows):
        for j in range(grid_columns):
            axs[i, j].imshow(gen_imgs[cnt, :, :, 0], cmap='gray')
            axs[i, j].axis("off")
            cnt += 1
    plt.show()

train_gan(epochs=1000)
```

[8]:
```python
def train_gan(epochs,batch_size=64,sample_interval=10):
    (x_train,_),(_,_)= mnist.load_data()
    x_train = (x_train.astype(np.float16)-127.5)/127.5
    x_train = np.expand_dims(x_train,axis=-1)

    half_batch = int(batch_size/2)

    for epoch in tqdm(range(epochs)):
        #train discriminator
        idx = np.random.randint(0,x_train.shape[0],half_batch)
        real_img = x_train[idx]
        real_labels = np.ones((half_batch,1))
```

```
        discriminator.trainable=True
        fake_img = generator.predict(np.random.randn(half_batch,z_dim))
        fake_labels = np.zeros((half_batch,1))

        d_loss_real = discriminator.train_on_batch(real_img,real_labels)
        d_loss_fake = discriminator.train_on_batch(fake_img,fake_labels)
        d_loss = 0.5* np.add(d_loss_real,d_loss_fake)

        noise = np.random.randn(batch_size,z_dim)
        valid_labels = np.ones((batch_size,1))
        discriminator.trainable=False
        g_loss = gan.train_on_batch(noise,valid_labels)

        if epoch % sample_interval ==0:
            print(f"{epoch} [D loss:{d_loss[0]} | D Accuracy:{100*d_loss[1]}]␣
↪[G loss:{g_loss}]")
            sample_images(epoch)
```

[9]:
```
def build_generator2():
    model = Sequential()

    # Encoder
    model.add(Input(shape=(28,28,1)))
    model.add(Conv2D(256, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    model.add(Conv2D(512, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    model.add(Conv2D(1024, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    # Decoder
    model.add(Conv2DTranspose(1024, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    model.add(Conv2DTranspose(512, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())

    model.add(Conv2DTranspose(256, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))
    model.add(BatchNormalization())
```

```
    model.add(Conv2D(1, (5, 5), strides=(1, 1), activation='tanh'))
    model.summary()
    return model
```

[10]:
```
def build_discriminator2():
    model = Sequential()
    model.add(Input(shape=(28,28,1)))
    model.add(Conv2D(256, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    model.add(Conv2D(512, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    model.add(Conv2D(1024, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    model.add(Conv2D(512, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    model.add(Conv2D(256, (3, 3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    model.add(Flatten())
    model.add(Dense(64))
    model.add(LeakyReLU(0.2))
    model.add(Dense(1,activation='sigmoid'))
    model.summary()
    return model
```

[11]:
```
discriminator2 = build_discriminator2()
discriminator2.
 ↪compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

generator2 = build_generator2()


gan2 = build_gan(generator2,discriminator2)
gan2.compile(loss='binary_crossentropy',optimizer='adam')
```

```
Model: "sequential_3"


 Layer (type)                Output Shape              ␣
↪Param #
```

```
conv2d (Conv2D)                 (None, 14, 14, 256)          ␣
↪2,560

leaky_re_lu_5 (LeakyReLU)       (None, 14, 14, 256)          ␣
↪  0

conv2d_1 (Conv2D)               (None, 7, 7, 512)            ␣
↪1,180,160

leaky_re_lu_6 (LeakyReLU)       (None, 7, 7, 512)            ␣
↪  0

conv2d_2 (Conv2D)               (None, 4, 4, 1024)           ␣
↪4,719,616

leaky_re_lu_7 (LeakyReLU)       (None, 4, 4, 1024)           ␣
↪  0

conv2d_3 (Conv2D)               (None, 2, 2, 512)            ␣
↪4,719,104

leaky_re_lu_8 (LeakyReLU)       (None, 2, 2, 512)            ␣
↪  0

conv2d_4 (Conv2D)               (None, 1, 1, 256)            ␣
↪1,179,904

leaky_re_lu_9 (LeakyReLU)       (None, 1, 1, 256)            ␣
↪  0

flatten_1 (Flatten)             (None, 256)                  ␣
↪  0

dense_7 (Dense)                 (None, 64)                   ␣
↪16,448

leaky_re_lu_10 (LeakyReLU)      (None, 64)                   ␣
↪  0

dense_8 (Dense)                 (None, 1)                    ␣
↪ 65


Total params: 11,817,857 (45.08 MB)
```

```
 Trainable params: 11,817,857 (45.08 MB)

 Non-trainable params: 0 (0.00 B)

Model: "sequential_4"

 Layer (type)                         Output Shape                  ␣
↪Param #

 conv2d_5 (Conv2D)                    (None, 14, 14, 256)           ␣
↪2,560

 leaky_re_lu_11 (LeakyReLU)           (None, 14, 14, 256)           ␣
↪  0

 batch_normalization_3                (None, 14, 14, 256)           ␣
↪1,024
 (BatchNormalization)                                               ␣
↪

 conv2d_6 (Conv2D)                    (None, 7, 7, 512)             ␣
↪1,180,160

 leaky_re_lu_12 (LeakyReLU)           (None, 7, 7, 512)             ␣
↪  0

 batch_normalization_4                (None, 7, 7, 512)             ␣
↪2,048
 (BatchNormalization)                                               ␣
↪

 conv2d_7 (Conv2D)                    (None, 4, 4, 1024)            ␣
↪4,719,616

 leaky_re_lu_13 (LeakyReLU)           (None, 4, 4, 1024)            ␣
↪  0

 batch_normalization_5                (None, 4, 4, 1024)            ␣
↪4,096
 (BatchNormalization)                                               ␣
↪

 conv2d_transpose (Conv2DTranspose)   (None, 8, 8, 1024)            ␣
↪9,438,208
```

```
 leaky_re_lu_14 (LeakyReLU)              (None, 8, 8, 1024)                          ␣
↪  0

 batch_normalization_6                   (None, 8, 8, 1024)                          ␣
↪4,096
 (BatchNormalization)                                                                ␣
↪

 conv2d_transpose_1 (Conv2DTranspose)    (None, 16, 16, 512)                         ␣
↪4,719,104

 leaky_re_lu_15 (LeakyReLU)              (None, 16, 16, 512)                         ␣
↪  0

 batch_normalization_7                   (None, 16, 16, 512)                         ␣
↪2,048
 (BatchNormalization)                                                                ␣
↪

 conv2d_transpose_2 (Conv2DTranspose)    (None, 32, 32, 256)                         ␣
↪1,179,904

 leaky_re_lu_16 (LeakyReLU)              (None, 32, 32, 256)                         ␣
↪  0

 batch_normalization_8                   (None, 32, 32, 256)                         ␣
↪1,024
 (BatchNormalization)                                                                ␣
↪

 conv2d_8 (Conv2D)                       (None, 28, 28, 1)                           ␣
↪6,401


 Total params: 21,260,289 (81.10 MB)

 Trainable params: 21,253,121 (81.07 MB)

 Non-trainable params: 7,168 (28.00 KB)

Model: "sequential_5"
```

```
Layer (type)                                 Output Shape                              ␣
↪Param #

sequential_4 (Sequential)                    (None, 28, 28, 1)                         ␣
↪21,260,289

sequential_3 (Sequential)                    (None, 1)                                 ␣
↪11,817,857


Total params: 33,078,146 (126.18 MB)

Trainable params: 33,070,978 (126.16 MB)

Non-trainable params: 7,168 (28.00 KB)
```

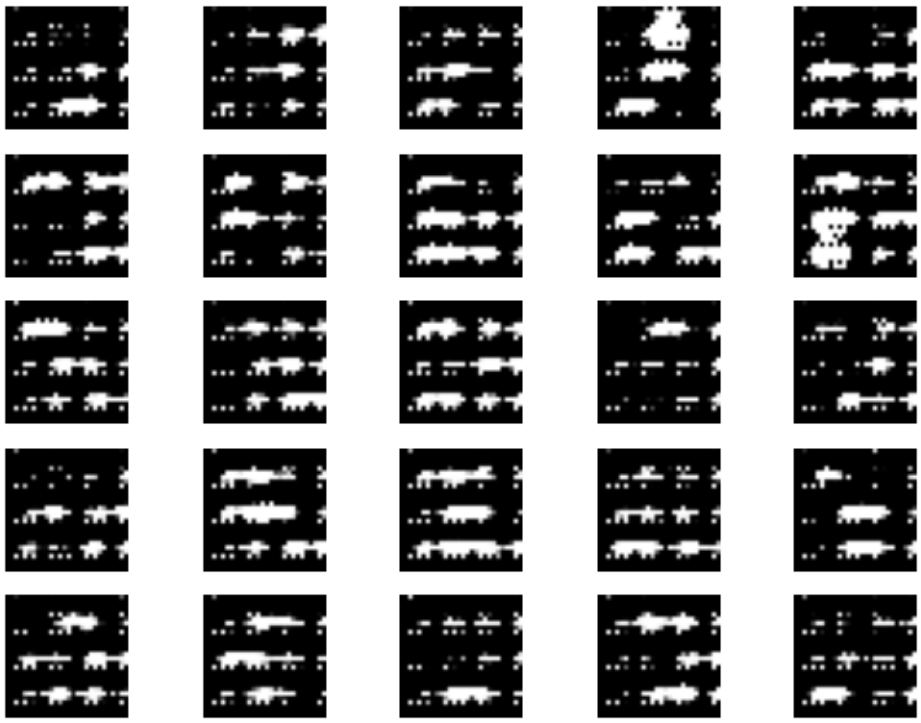
[1]:
```python
def sample_images2(epoch, grid_rows=5, grid_columns=5):
    noise = np.random.randn(grid_rows * grid_columns, 28,28,1)
    gen_imgs = generator2.predict(noise)
    gen_imgs = 0.5 * gen_imgs + 0.5

    fig, axs = plt.subplots(grid_rows, grid_columns)
    cnt = 0
    for i in range(grid_rows):
        for j in range(grid_columns):
            axs[i, j].imshow(gen_imgs[cnt, :, :, 0], cmap='gray')
            axs[i, j].axis("off")
            cnt += 1
    plt.show()
conter =0
def train_gan2(epochs, batch_size=64, sample_interval=10):
    (x_train, _), (_, _) = mnist.load_data()
    x_train = (x_train.astype(np.float32) - 127.5) / 127.5
    x_train = np.expand_dims(x_train, axis=-1)

    half_batch = int(batch_size / 2)

    for epoch in tqdm(range(epochs)):
        # Train discriminator 3 times
        for _ in range(3):
            if conter==5:
                break
            idx = np.random.randint(0, x_train.shape[0], half_batch)
            real_img = x_train[idx]
            real_labels = np.ones((half_batch, 1))
```

```
            discriminator2.trainable = True

            noise = np.random.randn(half_batch, 28, 28, 1)
            fake_img = generator2.predict(noise)
            fake_labels = np.zeros((half_batch, 1))

            d_loss_real = discriminator2.train_on_batch(real_img, real_labels)
            d_loss_fake = discriminator2.train_on_batch(fake_img, fake_labels)
            d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
        conter+=1
        for _ in range(3):
        # Train generator 3 time
            noise = np.random.randn(batch_size, 28, 28, 1)
            valid_labels = np.ones((batch_size, 1))

            discriminator2.trainable = False
            g_loss = gan2.train_on_batch(noise, valid_labels)

        if epoch % sample_interval == 0:
            print(f"{epoch} [D loss: {d_loss[0]} | D Accuracy: {100 * d_loss[1]:
↪.2f}] [G loss: {g_loss}]")
            sample_images2(epoch)
```

[16]:
```
from tensorflow.keras import backend as K

K.clear_session()
```

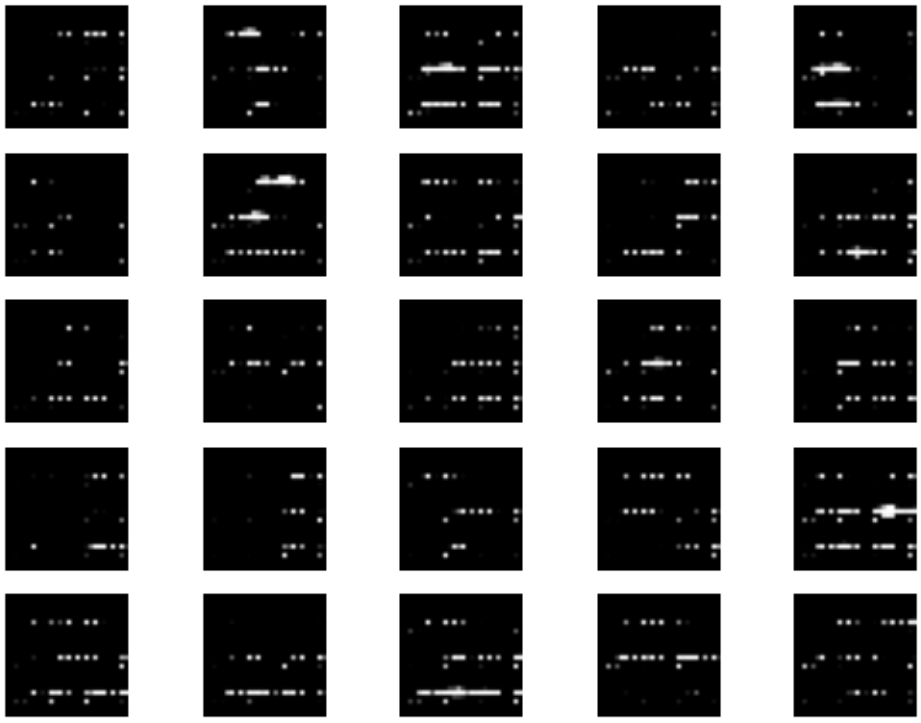
[ ]:
```
train_gan2(epochs=150)
```

```
  0%|
| 0/150 [00:00<?, ?it/s]

1/1             1s 796ms/step
0 [D loss: 1.0410714149475098 | D Accuracy: 94.02] [G loss: [array(1.0382953,
dtype=float32), array(1.0382953, dtype=float32), array(0.9403258,
dtype=float32)]]
1/1             0s 337ms/step
```



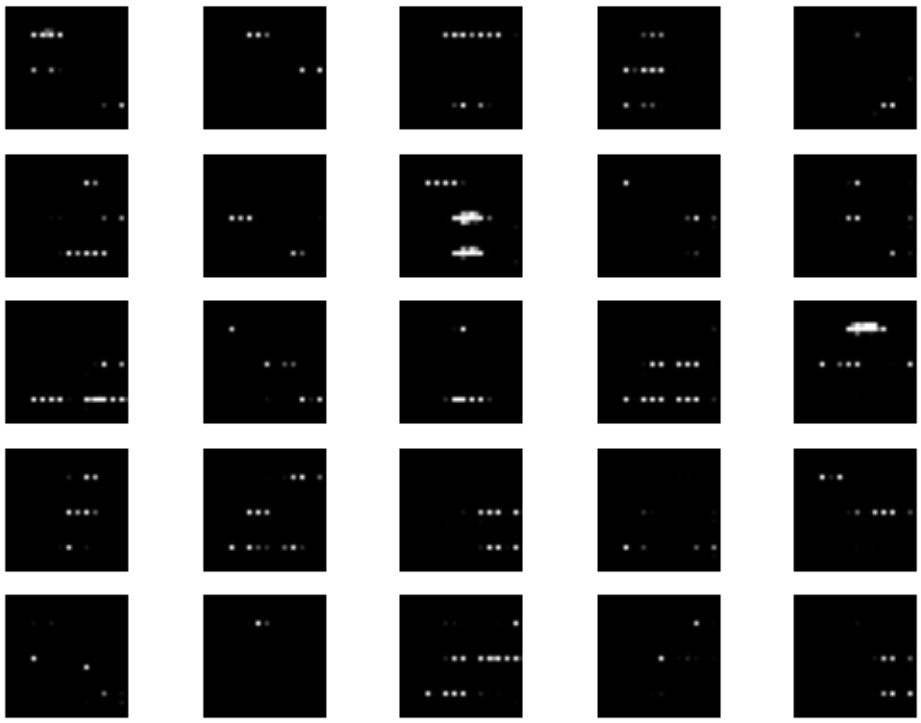
```
  1%|
| 1/150 [00:17<42:14, 17.01s/it]
1/1             0s 320ms/step
  1%|
| 2/150 [00:31<38:24, 15.57s/it]
1/1             0s 323ms/step
  2%|
| 3/150 [00:46<37:43, 15.40s/it]
1/1             0s 352ms/step
  3%|
| 4/150 [01:02<37:26, 15.39s/it]
1/1             0s 327ms/step
  3%|
| 5/150 [01:17<37:04, 15.34s/it]
1/1             0s 334ms/step
  4%|
| 6/150 [01:31<36:03, 15.02s/it]
1/1             0s 340ms/step
```

```
  5%|
| 7/150 [01:44<34:02, 14.28s/it]

1/1             0s 337ms/step

  5%|
| 8/150 [01:59<34:10, 14.44s/it]

1/1             0s 349ms/step

  6%|
| 9/150 [02:13<33:24, 14.21s/it]

1/1             0s 289ms/step

  7%|
| 10/150 [02:25<32:08, 13.78s/it]

1/1             0s 307ms/step
10 [D loss: 0.9423811435699463 | D Accuracy: 94.52] [G loss: [array(0.9401197,
dtype=float32), array(0.9401197, dtype=float32), array(0.9453125,
dtype=float32)]]
1/1             0s 324ms/step
```



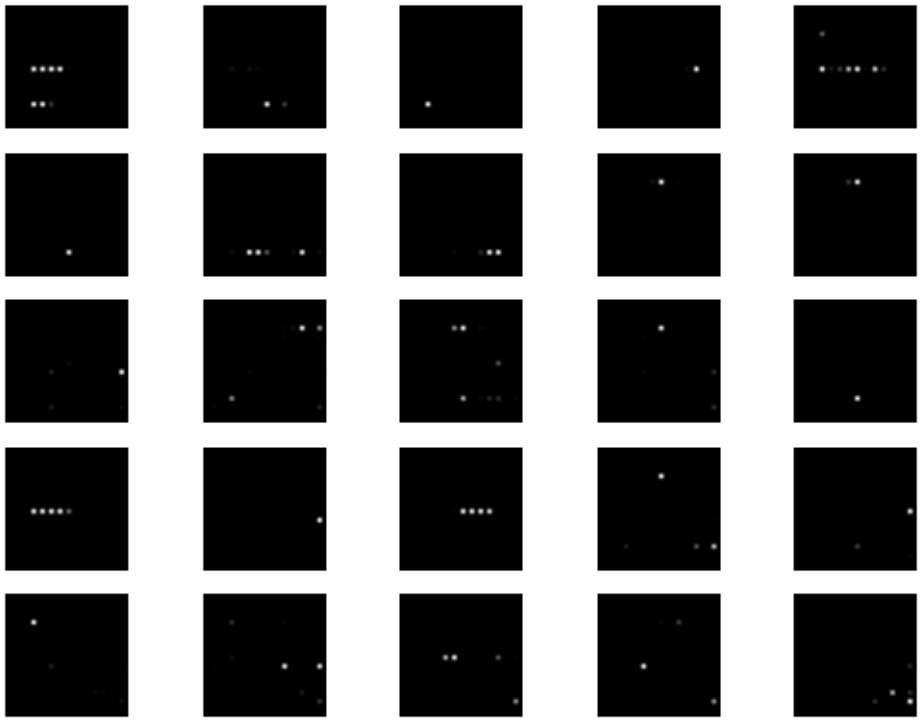
```
  7%|
| 11/150 [02:42<33:45, 14.57s/it]

1/1             0s 341ms/step
```

```
  8%|
| 12/150 [02:55<32:52, 14.30s/it]

1/1             0s 303ms/step

  9%|
| 13/150 [03:08<31:37, 13.85s/it]

1/1             0s 328ms/step

  9%|
| 14/150 [03:22<31:32, 13.92s/it]

1/1             0s 357ms/step

 10%|
| 15/150 [03:36<31:04, 13.81s/it]

1/1             0s 330ms/step

 11%|
| 16/150 [03:48<29:49, 13.35s/it]

1/1             0s 299ms/step

 11%|
| 17/150 [04:03<30:49, 13.91s/it]

1/1             0s 291ms/step

 12%|
| 18/150 [04:17<30:21, 13.80s/it]

1/1             0s 293ms/step

 13%|
| 19/150 [04:30<29:48, 13.65s/it]

1/1             0s 308ms/step

 13%|
| 20/150 [04:44<29:25, 13.58s/it]

1/1             0s 308ms/step
20 [D loss: 0.860283374786377 | D Accuracy: 94.96] [G loss: [array(0.85839266,
dtype=float32), array(0.85839266, dtype=float32), array(0.94969845,
dtype=float32)]]
1/1             0s 284ms/step
```



```
 14%|
| 21/150 [04:58<29:26, 13.69s/it]
1/1             0s 307ms/step
 15%|
| 22/150 [05:14<30:45, 14.42s/it]
1/1             0s 328ms/step
 15%|
| 23/150 [05:28<30:19, 14.33s/it]
1/1             0s 352ms/step
 16%|
| 24/150 [05:42<29:51, 14.21s/it]
1/1             0s 325ms/step
 17%|
| 25/150 [05:57<30:33, 14.67s/it]
1/1             0s 348ms/step
 17%|
| 26/150 [06:13<30:38, 14.82s/it]
1/1             0s 317ms/step
```

```
 18%|
| 27/150 [06:26<29:30, 14.39s/it]

1/1             0s 363ms/step

 19%|
| 28/150 [06:40<28:59, 14.26s/it]

1/1             0s 351ms/step

 19%|
| 29/150 [06:54<28:37, 14.20s/it]

1/1             0s 342ms/step

 20%|
| 30/150 [07:08<28:20, 14.17s/it]

1/1             0s 352ms/step
30 [D loss: 0.7908669114112854 | D Accuracy: 95.37] [G loss: [array(0.7892706,
dtype=float32), array(0.7892706, dtype=float32), array(0.953755,
dtype=float32)]]
1/1             0s 312ms/step
```

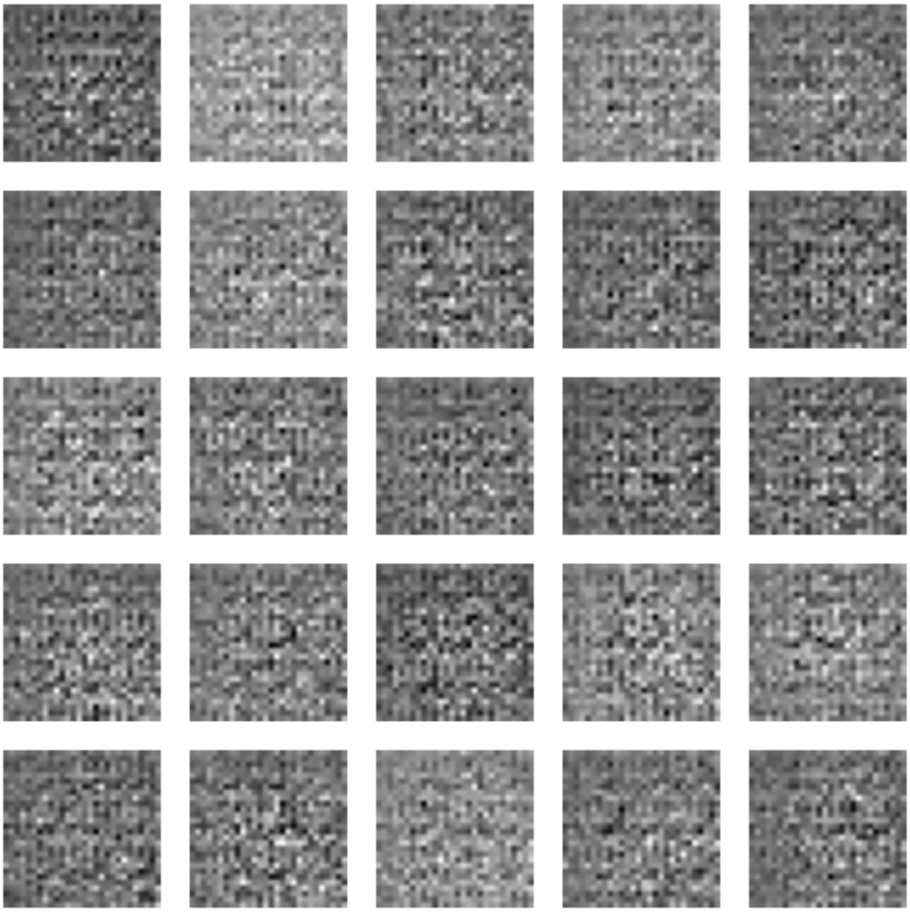


```
 21%|
| 31/150 [07:25<29:59, 15.12s/it]

1/1             0s 305ms/step
```

```
 21%|
| 32/150 [07:40<29:25, 14.96s/it]

1/1             0s 378ms/step

 22%|
| 33/150 [07:56<29:43, 15.24s/it]

1/1             0s 344ms/step

 23%|
| 34/150 [08:11<29:38, 15.33s/it]

1/1             0s 308ms/step

 23%|
| 35/150 [08:26<29:05, 15.17s/it]

1/1             0s 332ms/step
```

[15]:
```
sample_images2(20)
```

```
1/1             0s 352ms/step
```



[ ]: