1) Importing Python Packages for GAN

```
!pip install tensorflow==2.8.0
```

```
  Downloading tensorboard-2.8.0-py3-none-any.whl.metadata (1.9 kB)
Collecting tf-estimator-nightly==2.8.0.dev2021122109 (from tensorflow==2.8.0)
  Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting keras<2.9,>=2.8.0rc0 (from tensorflow==2.8.0)
  Downloading keras-2.8.0-py2.py3-none-any.whl.metadata (1.3 kB)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboar
Collecting google-auth-oauthlib<0.5,>=0.4.1 (from tensorboard<2.9,>=2.8->tensorflow==2.8.0)
  Downloading google_auth_oauthlib-0.4.6-py2.py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.9,
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<
Collecting tensorboard-data-server<0.7.0,>=0.6.0 (from tensorboard<2.9,>=2.8->tensorflow==2.8.0)
  Downloading tensorboard_data_server-0.6.1-py3-none-manylinux2010_x86_64.whl.metadata (1.1 kB)
Collecting tensorboard-plugin-wit>=1.6.0 (from tensorboard<2.9,>=2.8->tensorflow==2.8.0)
  Downloading tensorboard_plugin_wit-1.8.1-py3-none-any.whl.metadata (873 bytes)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-au
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-aut
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from request
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=0.11
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modu
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthli
Downloading tensorflow-2.8.0-cp310-cp310-manylinux2010_x86_64.whl (497.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 497.6/497.6 MB 3.8 MB/s eta 0:00:00
Downloading tf_estimator_nightly-2.8.0.dev2021122109-py2.py3-none-any.whl (462 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 462.5/462.5 kB 14.1 MB/s eta 0:00:00
Downloading keras-2.8.0-py2.py3-none-any.whl (1.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 57.4 MB/s eta 0:00:00
Downloading Keras_Preprocessing-1.1.2-py2.py3-none-any.whl (42 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 3.9 MB/s eta 0:00:00
Downloading tensorboard-2.8.0-py3-none-any.whl (5.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 69.8 MB/s eta 0:00:00
Downloading google_auth_oauthlib-0.4.6-py2.py3-none-any.whl (18 kB)
Downloading tensorboard_data_server-0.6.1-py3-none-manylinux2010_x86_64.whl (4.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.9/4.9 MB 84.8 MB/s eta 0:00:00
Downloading tensorboard_plugin_wit-1.8.1-py3-none-any.whl (781 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 781.3/781.3 kB 36.0 MB/s eta 0:00:00
Installing collected packages: tf-estimator-nightly, tensorboard-plugin-wit, keras, tensorboard-data-server, ker
  Attempting uninstall: keras
    Found existing installation: keras 3.4.1
    Uninstalling keras-3.4.1:
      Successfully uninstalled keras-3.4.1
  Attempting uninstall: tensorboard-data-server
    Found existing installation: tensorboard-data-server 0.7.2
    Uninstalling tensorboard-data-server-0.7.2:
      Successfully uninstalled tensorboard-data-server-0.7.2
  Attempting uninstall: google-auth-oauthlib
    Found existing installation: google-auth-oauthlib 1.2.1
    Uninstalling google-auth-oauthlib-1.2.1:
      Successfully uninstalled google-auth-oauthlib-1.2.1
  Attempting uninstall: tensorboard
```

```
from keras.datasets import mnist

from keras.models import Sequential
from keras.layers import BatchNormalization
from keras.layers import Dense, Reshape, Flatten
from keras.layers import LeakyReLU
from tensorflow.keras.optimizers import Adam

import numpy as np
!mkdir generated_images
```

Kode di atas mengimpor beberapa pustaka dan modul yang diperlukan untuk membangun dan melatih model pembelajaran mendalam (deep learning) menggunakan dataset MNIST. Degan menggunakan TensorFlow 2.8.0 dan Keras untuk mengimplementasikan jaringan saraf. Dataset MNIST yang berisi gambar digit tangan akan diambil, dan model Sequential akan dibangun menggunakan lapisan BatchNormalization, Dense, Reshape, Flatten, dan LeakyReLU. Adam digunakan sebagai optimizer untuk mempercepat proses pembelajaran. Di akhir, direktori generated_images dibuat untuk menyimpan gambar hasil yang dihasilkan dari proses pelatihan model.

2) Variables for Neural Networks & Data

```
img_width = 28
img_height = 28
channels = 1
img_shape = (img_width, img_height, channels)
latent_dim = 100
adam = Adam(learning_rate=0.0001)
```

Kode di atas mengatur beberapa parameter untuk pelatihan model deep learning pada dataset gambar. img_width dan img_height masing-masing ditetapkan ke 28, yang menunjukkan bahwa gambar input akan memiliki ukuran 28x28 piksel. channels ditetapkan ke 1, menunjukkan bahwa gambar akan dalam format grayscale (hitam-putih). Variabel img_shape menggabungkan ketiga parameter ini menjadi tuple (28, 28, 1) yang menunjukkan bentuk gambar input. latent_dim diatur ke 100, yang biasanya digunakan dalam model seperti GAN (Generative Adversarial Networks) untuk menentukan dimensi dari ruang laten (kode acak yang diubah menjadi gambar). adam mendefinisikan optimizer Adam dengan tingkat pembelajaran sebesar 0,0001, yang akan digunakan untuk memperbarui bobot model selama proses pelatihan.

3) Building Generator

```
def build_generator():
  model = Sequential()

  model.add(Dense(256, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(BatchNormalization(momentum=0.8))

  model.add(Dense(256))
  model.add(LeakyReLU(alpha=0.2))
  model.add(BatchNormalization(momentum=0.8))

  model.add(Dense(256))
  model.add(LeakyReLU(alpha=0.2))
  model.add(BatchNormalization(momentum=0.8))

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

  model.summary()
  return model

generator = build_generator()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               25856

 leaky_re_lu (LeakyReLU)     (None, 256)               0

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dense_1 (Dense)             (None, 256)               65792
 leaky_re_lu_1 (LeakyReLU)   (None, 256)               0

 batch_normalization_1 (Batc  (None, 256)              1024
 hNormalization)

 dense_2 (Dense)             (None, 256)               65792

 leaky_re_lu_2 (LeakyReLU)   (None, 256)               0

 batch_normalization_2 (Batc  (None, 256)              1024
 hNormalization)

 dense_3 (Dense)             (None, 784)               201488

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

=================================================================
Total params: 362,000
Trainable params: 360,464
Non-trainable params: 1,536
_________________________________________________________________
```

Kode di atas mendefinisikan fungsi build_generator() yang membangun arsitektur generator untuk model GAN. Generator ini terdiri dari beberapa lapisan Dense dengan 256 neuron, diikuti oleh fungsi aktivasi LeakyReLU untuk menangani masalah "dying ReLU," serta BatchNormalization untuk stabilitas pelatihan. Lapisan akhir menggunakan tanh untuk menghasilkan output berupa gambar dengan bentuk (28x28 piksel, grayscale). Setelah itu, output diubah bentuknya dengan Reshape agar sesuai dengan dimensi gambar. Fungsi ini mengembalikan model generator, yang nantinya akan menghasilkan gambar dari input acak.

4) Building Discriminator

```
def build_discriminator():
  model = Sequential()

  model.add(Flatten(input_shape=img_shape))
  model.add(Dense(512))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dense(256))
  model.add(Dense(1, activation='sigmoid'))

  model.summary()
  return model

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense_4 (Dense)             (None, 512)               401920

 leaky_re_lu_3 (LeakyReLU)   (None, 512)               0

 dense_5 (Dense)             (None, 256)               131328

 dense_6 (Dense)             (None, 1)                 257

=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
_________________________________________________________________
```

Kode di atas mendefinisikan fungsi build_discriminator() yang membangun arsitektur discriminator untuk model GAN. Discriminator ini memproses gambar input dengan melapisinya melalui Flatten untuk meratakan gambar, diikuti oleh dua lapisan Dense dengan 512 dan 256 neuron, serta fungsi aktivasi LeakyReLU untuk menangani masalah "dying ReLU." Pada lapisan terakhir, sigmoid digunakan untuk mengeluarkan probabilitas antara 0 dan 1, menentukan apakah gambar input asli atau hasil dari generator. Setelah itu, model dikompilasi menggunakan binary_crossentropy sebagai fungsi kerugian dan Adam sebagai optimizer, dengan akurasi sebagai metrik evaluasi.

5) Connecting Neural Networks to build GAN

```
GAN = Sequential()
discriminator.trainable = False
GAN.add(generator)
GAN.add(discriminator)

GAN.compile(loss='binary_crossentropy', optimizer=adam)
GAN.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 28, 28, 1)         362000

 sequential_1 (Sequential)   (None, 1)                 533505

=================================================================
Total params: 895,505
Trainable params: 360,464
Non-trainable params: 535,041
_________________________________________________________________
```

Kode di atas menggabungkan generator dan discriminator untuk membentuk model GAN (Generative Adversarial Network). Discriminator diatur menjadi tidak dapat dilatih (trainable=False) untuk memastikan bahwa hanya generator yang diperbarui selama pelatihan gabungan. Setelah itu, generator ditambahkan ke dalam GAN diikuti oleh discriminator, sehingga GAN secara keseluruhan dapat mengambil input dari generator dan menghasilkan gambar yang akan dievaluasi oleh discriminator. Model GAN kemudian dikompilasi menggunakan fungsi kerugian binary_crossentropy dan optimizer Adam. Fungsi summary() digunakan untuk menampilkan struktur model GAN yang telah dibuat.

6) Outputting Images

```
#@title
## **7) Outputting Images**
import matplotlib.pyplot as plt
import glob
import imageio
import PIL

save_name = 0.00000000

def save_imgs(epoch):
    r, c = 5, 5
    noise = np.random.normal(0, 1, (r * c, latent_dim))
    gen_imgs = generator.predict(noise)
    global save_name
    save_name += 0.00000001
    print("%.8f" % save_name)

    # Rescale images 0 - 1
    gen_imgs = 0.5 * gen_imgs + 0.5

    fig, axs = plt.subplots(r, c)
    cnt = 0
    for i in range(r):
        for j in range(c):
            axs[i,j].imshow(gen_imgs[cnt, :,:,0], cmap='gray')
            # axs[i,j].imshow(gen_imgs[cnt])
            axs[i,j].axis('off')
            cnt += 1
    fig.savefig("generated_images/%.8f.png" % save_name)
    print('saved')
    plt.close()
```

Kode di atas mendefinisikan fungsi save_imgs() yang bertujuan untuk menyimpan gambar yang dihasilkan oleh GAN setelah setiap epoch pelatihan. Fungsi ini menghasilkan input acak (noise), yang digunakan oleh generator untuk membuat gambar. Gambar yang dihasilkan diubah skalanya dari [-1, 1] menjadi [0, 1] untuk visualisasi. Selanjutnya, gambar ditampilkan dalam grid 5x5 menggunakan matplotlib dan disimpan dengan nama unik dalam folder generated_images. Setiap kali fungsi ini dipanggil, gambar baru disimpan, dan plot ditutup untuk menghemat memori.

7) Training GAN

```
def train(epochs, batch_size=64, save_interval=200):
  (X_train, _), (_, _) = mnist.load_data()

  # print(X_train.shape)
  #Rescale data between -1 and 1
  X_train = X_train / 127.5 -1.
  # X_train = np.expand_dims(X_train, axis=3)
  # print(X_train.shape)

  #Create our Y for our Neural Networks
  valid = np.ones((batch_size, 1))
  fakes = np.zeros((batch_size, 1))

  for epoch in range(epochs):
    #Get Random Batch
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    #Generate Fake Images
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    gen_imgs = generator.predict(noise)

    #Train discriminator
    d_loss_real = discriminator.train_on_batch(imgs, valid)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fakes)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))

    #inverse y label
    g_loss = GAN.train_on_batch(noise, valid)

    print("******* %d [D loss: %f, acc: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100* d_loss[1], g_loss))

    if(epoch % save_interval) == 0:
      save_imgs(epoch)

  # print(valid)
```

```
train(30000, batch_size=64, save_interval=200)
```

```
Streaming output truncated to the last 5000 lines.
******* 25048 [D loss: 0.497097, acc: 78.12%] [G loss: 1.870629]
******* 25049 [D loss: 0.411296, acc: 82.03%] [G loss: 1.991264]
******* 25050 [D loss: 0.458011, acc: 78.91%] [G loss: 1.896154]
******* 25051 [D loss: 0.422382, acc: 79.69%] [G loss: 1.845564]
******* 25052 [D loss: 0.425483, acc: 82.03%] [G loss: 1.878022]
******* 25053 [D loss: 0.386282, acc: 84.38%] [G loss: 1.817748]
******* 25054 [D loss: 0.474064, acc: 76.56%] [G loss: 1.922760]
******* 25055 [D loss: 0.490260, acc: 75.00%] [G loss: 1.681170]
******* 25056 [D loss: 0.528329, acc: 75.78%] [G loss: 1.751010]
******* 25057 [D loss: 0.543004, acc: 75.00%] [G loss: 1.512944]
******* 25058 [D loss: 0.452843, acc: 78.91%] [G loss: 1.617875]
```

```
******* 25059 [D loss: 0.421911, acc: 82.81%] [G loss: 1.717595]
******* 25060 [D loss: 0.498479, acc: 75.78%] [G loss: 1.697359]
******* 25061 [D loss: 0.449429, acc: 76.56%] [G loss: 1.837943]
******* 25062 [D loss: 0.359810, acc: 88.28%] [G loss: 1.907816]
******* 25063 [D loss: 0.566714, acc: 75.78%] [G loss: 1.266432]
******* 25064 [D loss: 0.485042, acc: 75.78%] [G loss: 1.214413]
******* 25065 [D loss: 0.458769, acc: 80.47%] [G loss: 1.738909]
******* 25066 [D loss: 0.364135, acc: 81.25%] [G loss: 2.274985]
******* 25067 [D loss: 0.383136, acc: 82.81%] [G loss: 2.500970]
******* 25068 [D loss: 0.461297, acc: 82.03%] [G loss: 1.840734]
******* 25069 [D loss: 0.525301, acc: 75.78%] [G loss: 1.217048]
******* 25070 [D loss: 0.521756, acc: 72.66%] [G loss: 1.441391]
******* 25071 [D loss: 0.437680, acc: 85.16%] [G loss: 2.045457]
******* 25072 [D loss: 0.382891, acc: 83.59%] [G loss: 2.156328]
******* 25073 [D loss: 0.508789, acc: 79.69%] [G loss: 2.157140]
******* 25074 [D loss: 0.412342, acc: 84.38%] [G loss: 1.811631]
******* 25075 [D loss: 0.513014, acc: 75.78%] [G loss: 1.513061]
******* 25076 [D loss: 0.456453, acc: 77.34%] [G loss: 1.535066]
******* 25077 [D loss: 0.474533, acc: 77.34%] [G loss: 1.754072]
******* 25078 [D loss: 0.437499, acc: 82.81%] [G loss: 1.971852]
******* 25079 [D loss: 0.386361, acc: 82.81%] [G loss: 1.817902]
******* 25080 [D loss: 0.484674, acc: 80.47%] [G loss: 1.644559]
******* 25081 [D loss: 0.551357, acc: 75.78%] [G loss: 1.780514]
******* 25082 [D loss: 0.445160, acc: 82.81%] [G loss: 1.841163]
******* 25083 [D loss: 0.414466, acc: 84.38%] [G loss: 1.681660]
******* 25084 [D loss: 0.472717, acc: 78.91%] [G loss: 2.246826]
******* 25085 [D loss: 0.297833, acc: 89.84%] [G loss: 1.985623]
******* 25086 [D loss: 0.574902, acc: 75.78%] [G loss: 1.816491]
******* 25087 [D loss: 0.436690, acc: 81.25%] [G loss: 1.832191]
******* 25088 [D loss: 0.430413, acc: 81.25%] [G loss: 1.965428]
******* 25089 [D loss: 0.478264, acc: 79.69%] [G loss: 1.962949]
******* 25090 [D loss: 0.488987, acc: 79.69%] [G loss: 1.816114]
******* 25091 [D loss: 0.320049, acc: 89.06%] [G loss: 1.688707]
******* 25092 [D loss: 0.400479, acc: 85.16%] [G loss: 1.566740]
******* 25093 [D loss: 0.449797, acc: 81.25%] [G loss: 1.851594]
******* 25094 [D loss: 0.490802, acc: 75.00%] [G loss: 1.484410]
******* 25095 [D loss: 0.397467, acc: 85.16%] [G loss: 1.693041]
******* 25096 [D loss: 0.438772, acc: 76.56%] [G loss: 1.593485]
******* 25097 [D loss: 0.373933, acc: 86.72%] [G loss: 1.776646]
******* 25098 [D loss: 0.367024, acc: 84.38%] [G loss: 2.089563]
******* 25099 [D loss: 0.343639, acc: 87.50%] [G loss: 2.135706]
******* 25100 [D loss: 0.541440, acc: 77.34%] [G loss: 1.683367]
******* 25101 [D loss: 0.501016, acc: 74.22%] [G loss: 1.667705]
******* 25102 [D loss: 0.407964, acc: 78.91%] [G loss: 1.959993]
******* 25103 [D loss: 0.496725, acc: 79.69%] [G loss: 1.977823]
******* 25104 [D loss: 0.427938, acc: 79.69%] [G loss: 2.043293]
```

Kode di atas mendefinisikan fungsi train() yang melatih model GAN menggunakan dataset MNIST selama sejumlah epoch sebanyak 30000. Data gambar dilatih dengan mereskalanya ke rentang [-1, 1]. Pada setiap epoch, batch acak dari gambar asli diambil, lalu gambar palsu dihasilkan menggunakan noise acak yang diumpankan ke generator. Discriminator dilatih untuk membedakan antara gambar asli dan palsu, dan hasil kerugian (d_loss) dihitung dari keduanya. Setelah itu, generator dilatih melalui GAN dengan membalik label gambar agar bisa menipu discriminator. Fungsi mencetak kerugian dan akurasi dari discriminator serta kerugian dari generator. Setiap beberapa epoch, gambar yang dihasilkan disimpan menggunakan fungsi save_imgs().

8) Making GIF

```
# Display a single image using the epoch number
# def display_image(epoch_no):
#   return PIL.Image.open('generated_images/%.8f.png'.format(epoch_no))

anim_file = 'dcgan.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
  filenames = glob.glob('generated_images/*.png')
  filenames = sorted(filenames)
  for filename in filenames:
    image = imageio.imread(filename)
    writer.append_data(image)
  image = imageio.imread(filename)
  writer.append_data(image)
```

```
<ipython-input-10-5d911b6b7554>:11: DeprecationWarning: Starting with ImageIO v3 the behavior of this function wil
  image = imageio.imread(filename)
<ipython-input-10-5d911b6b7554>:13: DeprecationWarning: Starting with ImageIO v3 the behavior of this function wil
  image = imageio.imread(filename)
```

Kode di atas membuat file animasi GIF dcgan.gif dari gambar yang dihasilkan oleh model GAN. Pertama, kode ini mengumpulkan semua file gambar PNG dari folder generated_images dan mengurutkannya. Kemudian, menggunakan imageio, kode ini membuka penulis GIF dan menambahkan setiap gambar ke dalam file animasi. Terakhir, gambar terakhir juga ditambahkan untuk memastikan bahwa gambar terakhir muncul di akhir animasi. Hasil akhirnya adalah sebuah GIF yang menunjukkan perkembangan gambar yang dihasilkan oleh model selama pelatihan.