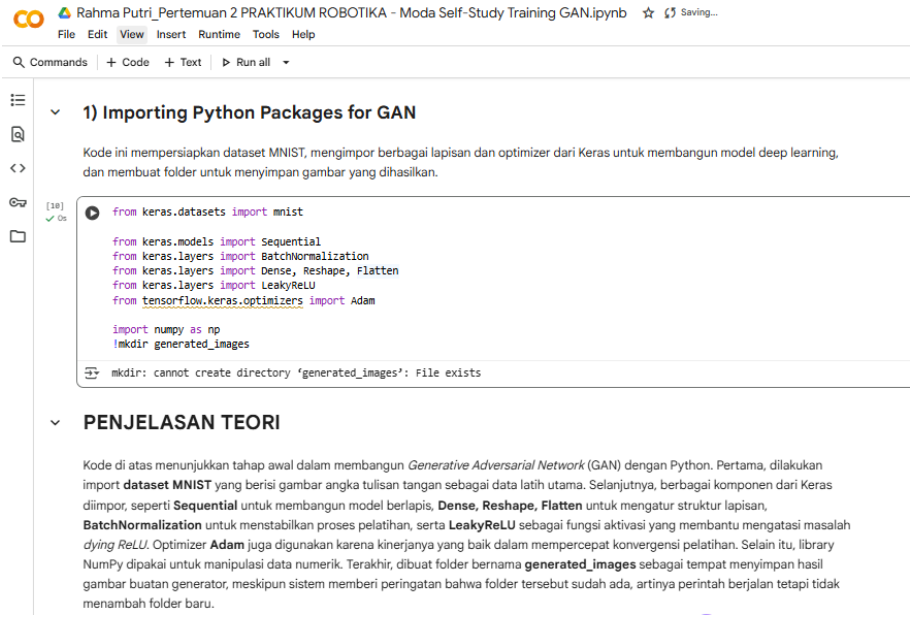
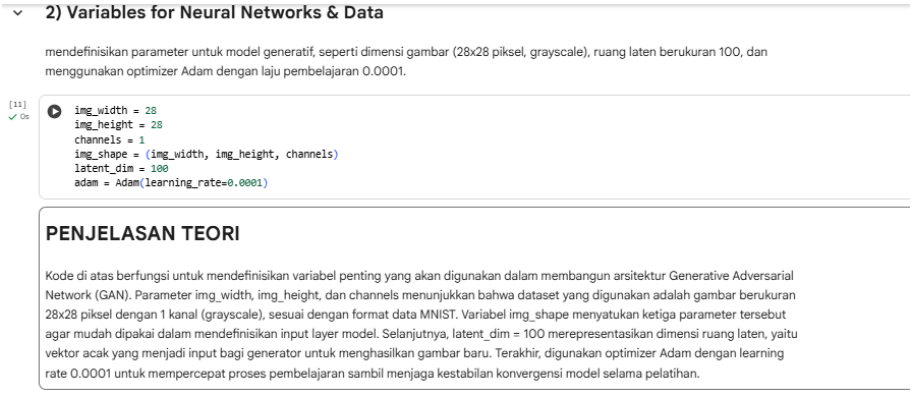
Nama : Rahma Putri Prabowo

NPM : 11122170





3) Building Generator

Fungsi ini membangun model generator untuk menghasilkan gambar. Model ini menggunakan beberapa lapisan Dense dengan aktivasi LeakyReLU dan BatchNormalization untuk meningkatkan stabilitas pelatihan. Di akhir, gambar dihasilkan dengan aktivasi tanh dan diubah bentuknya menjadi ukuran gambar yang diinginkan (28x28x1). Model generator ini diinisialisasi dengan dimensi ruang laten dan menghasilkan gambar dengan bentuk yang sesuai.

```
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()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: 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)
/usr/local/lib/python3.12/dist-packages/keras/src/layers/activations/leaky_relu.py:41: UserWarning: Argument `alpha` is deprecated. Use `negative_slope` instead.
  warnings.warn(
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_7 (Dense)	(None, 256)	25,856
leaky_re_lu_4 (LeakyReLU)	(None, 256)	0
batch_normalization_3 (BatchNormalization)	(None, 256)	1,024
dense_8 (Dense)	(None, 256)	65,792
leaky_re_lu_5 (LeakyReLU)	(None, 256)	0
batch_normalization_4 (BatchNormalization)	(None, 256)	1,024
dense_9 (Dense)	(None, 256)	65,792
leaky_re_lu_6 (LeakyReLU)	(None, 256)	0
batch_normalization_5 (BatchNormalization)	(None, 256)	1,024
dense_10 (Dense)	(None, 784)	201,488
reshape_1 (Reshape)	(None, 28, 28, 1)	0

```
Total params: 362,000 (1.38 MB)
Trainable params: 360,464 (1.38 MB)
Non-trainable params: 1,536 (6.00 KB)
```

PENJELASAN TEORI

Fungsi **build_generator()** pada kode di atas berfungsi untuk membangun arsitektur *generator* pada GAN, yaitu model yang bertugas menghasilkan gambar baru menyerupai data asli. Arsitektur dimulai dengan lapisan **Dense** berisi 256 neuron dengan input berupa vektor laten berdimensi 100. Setiap lapisan Dense diikuti oleh **LeakyReLU** untuk mencegah *dying ReLU* dan **BatchNormalization** untuk menstabilkan distribusi data selama pelatihan. Setelah tiga kali pengulangan blok Dense–LeakyReLU–BatchNormalization, ditambahkan lapisan Dense akhir dengan jumlah neuron sebesar **np.prod(img_shape)**, yaitu hasil perkalian dimensi gambar (28×28×1 = 784 piksel), serta menggunakan aktivasi **tanh** agar keluaran berada pada rentang -1 hingga 1 sesuai dengan normalisasi dataset. Lapisan ini kemudian di-*reshape* menjadi format gambar 28×28×1 (grayscale). Dari ringkasan model, terlihat total parameter sebesar **362.000** dengan sebagian besar dapat dilatih, menandakan bahwa generator memiliki kompleksitas yang cukup untuk mempelajari pola dari data MNIST.

Rahma Putri_Pertemuan 2 PRAKTIKUM ROBOTIKA - Moda Self-Study Training GAN.ipynb

File Edit View Insert Runtime Tools Help

ommands + Code + Text ▷ Run all

4) Building Discriminator

Fungsi ini membangun model diskriminator untuk membedakan gambar asli dan gambar yang dihasilkan oleh generator. Model ini memulai dengan lapisan Flatten untuk mengubah gambar menjadi vektor, diikuti oleh lapisan Dense dengan aktivasi LeakyReLU, dan akhirnya lapisan output dengan aktivasi sigmoid untuk menghasilkan prediksi antara 0 (gambar palsu) dan 1 (gambar asli). Diskriminator ini kemudian dikompilasi menggunakan loss binary_crossentropy dan optimizer Adam.

[13] ✓ 0s

```python
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_4"
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_11 (Dense)	(None, 512)	401,920
leaky_re_lu_7 (LeakyReLU)	(None, 512)	0
dense_12 (Dense)	(None, 256)	131,328
dense_13 (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)
```

PENJELASAN TEORI

Fungsi **build_discriminator()** di atas membangun arsitektur *discriminator*, yaitu model klasifikasi biner yang berperan untuk membedakan antara gambar asli dari dataset dan gambar palsu yang dihasilkan oleh generator. Model ini dimulai dengan lapisan **Flatten** untuk mengubah input gambar berukuran 28×28×1 menjadi vektor 784 elemen. Selanjutnya, digunakan lapisan **Dense(512)** yang cukup besar agar mampu menangkap pola kompleks pada data, kemudian diaktifkan dengan **LeakyReLU** untuk menghindari masalah *vanishing gradient*. Lapisan berikutnya adalah **Dense(256)** yang berfungsi sebagai lapisan tersembunyi tambahan untuk memperdalam representasi fitur, lalu diakhiri dengan lapisan **Dense(1, activation='sigmoid')** untuk menghasilkan output probabilitas antara 0 (palsu) dan 1 (asli). Model ini dikompilasi dengan **loss binary_crossentropy** karena merupakan kasus klasifikasi biner, serta menggunakan **optimizer Adam** untuk efisiensi pelatihan. Berdasarkan ringkasan model, jumlah parameter trainable mencapai **533.505**, menunjukkan bahwa discriminator memiliki kapasitas besar untuk membedakan pola data asli dan data sintetis.

5) Connecting Neural Networks to build GAN

Membangun model GAN (Generative Adversarial Network) dengan menggabungkan generator dan diskriminator. Diskriminator diatur agar tidak dapat dilatih (discriminator.trainable = False) saat pelatihan GAN, sehingga hanya generator yang diperbarui selama pelatihan. Model GAN ini kemudian dikompilasi dengan loss binary_crossentropy dan optimizer Adam. Setelah itu, ringkasan model ditampilkan.

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

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

```
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
sequential_3 (Sequential)	(None, 28, 28, 1)	362,000
sequential_4 (Sequential)	(None, 1)	533,505

```
Total params: 895,505 (3.42 MB)
Trainable params: 360,464 (1.38 MB)
Non-trainable params: 535,041 (2.04 MB)
```

PENJELASAN TEORI

Pada tahap ini, dibuat model **GAN (Generative Adversarial Network)** dengan cara menggabungkan generator dan discriminator menjadi satu arsitektur. Sebelum penggabungan, atribut **discriminator.trainable = False** diatur agar selama proses pelatihan GAN, hanya bobot pada generator yang diperbarui, sementara discriminator tetap statis. Hal ini penting karena tujuan utama dari pelatihan GAN adalah membuat generator semakin mampu menipu discriminator dengan menghasilkan gambar yang mirip dengan data asli. Setelah digabung, model GAN dikompilasi menggunakan **loss binary_crossentropy** karena tugas akhirnya berupa klasifikasi biner (asli atau palsu) dengan **optimizer Adam** untuk stabilitas pembelajaran. Dari ringkasan model terlihat bahwa GAN memiliki total parameter **895.505**, tetapi hanya sekitar **360.464** yang dapat dilatih (milik generator), sedangkan sisanya **535.041** tetap tidak dilatih karena bagian discriminator dibekukan. Struktur ini memastikan bahwa selama pelatihan, fokus pembaruan bobot hanya ada pada generator agar kualitas gambar hasil sintesis semakin meningkat.

6) Outputting Images

Menghasilkan gambar dari generator dan menyimpannya sebagai file PNG. Setiap kali dipanggil, fungsi membuat 25 gambar (5x5 grid) dengan menghasilkan noise acak, mengubahnya menjadi gambar oleh generator, dan menyesuaikan skala gambar ke rentang 0-1. Gambar yang dihasilkan kemudian disimpan dengan nama file yang meningkat secara bertahap.

```
#@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()
```

PENJELASAN TEORI

Fungsi **save_imgs(epoch)** pada kode di atas berfungsi untuk menampilkan sekaligus menyimpan hasil gambar yang dihasilkan oleh generator selama proses pelatihan GAN. Pertama, dibuat **noise acak** berdistribusi normal dengan dimensi sesuai ruang laten (100), yang kemudian dimasukkan ke generator untuk menghasilkan gambar sintetis. Fungsi ini menghasilkan **25 gambar** sekaligus, ditata dalam grid 5x5. Sebelum divisualisasikan, gambar hasil generator yang berada dalam rentang nilai [-1, 1] diskalakan ulang menjadi [0, 1] agar bisa ditampilkan dengan benar. Tiap gambar ditampilkan dalam format grayscale tanpa sumbu (*axis off*). Selanjutnya, grid gambar tersebut disimpan ke dalam folder **generated_images** dengan nama file yang selalu bertambah menggunakan variabel **save_name** agar tidak menimpa file sebelumnya. Dengan mekanisme ini, kita dapat memantau perkembangan kualitas gambar yang dihasilkan generator dari waktu ke waktu selama proses pelatihan.

7) Training GAN

Melatih model GAN untuk menghasilkan gambar. Dalam setiap epoch, fungsi ini:

1. Mengambil batch acak dari dataset MNIST dan mengubah skala gambar ke rentang -1 hingga 1.
2. Menghasilkan gambar palsu menggunakan generator.
3. Melatih diskriminator dengan gambar asli dan palsu untuk menghitung kerugian (d_loss).
4. Menghasilkan noise acak dan melatih generator melalui GAN untuk meminimalkan kerugian (g_loss).
5. Setiap interval tertentu (misalnya setiap 200 epoch), gambar yang dihasilkan disimpan.

Fungsi ini menjalankan pelatihan selama sejumlah epoch yang ditentukan dan mencetak kerugian serta akurasi diskriminator dan generator di setiap langkah.

[16] ✓ 49s

```
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)

    # Access the first element of g_loss, which represents the total loss
    g_loss_value = g_loss

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

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

  # print(valid)


train(500, batch_size=64, save_interval=200)
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
/usr/local/lib/python3.12/dist-packages/keras/src/backend/tensorflow/trainer.py:83: UserWarning: The model does not have any trainable weights.
  warnings.warn("The model does not have any trainable weights.")
******* 0 [D loss: 0.356249, acc: 76.17%] [G loss: 0.539660]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 286ms/step
0.00000001
saved
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
******* 1 [D loss: 0.456843, acc: 61.07%] [G loss: 0.533904]
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
******* 494 [D loss: 1.828520, acc: 50.15%] [G loss: 0.078166]
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
******* 495 [D loss: 1.830005, acc: 50.15%] [G loss: 0.078026]
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
******* 496 [D loss: 1.831523, acc: 50.15%] [G loss: 0.077883]
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
******* 497 [D loss: 1.833117, acc: 50.15%] [G loss: 0.077743]
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
******* 498 [D loss: 1.834696, acc: 50.15%] [G loss: 0.077606]
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
******* 499 [D loss: 1.836213, acc: 50.15%] [G loss: 0.077470]
```

PENJELASAN TEORI

Fungsi **train()** di atas merupakan inti dari proses pelatihan **Generative Adversarial Network (GAN)** menggunakan dataset MNIST. Pertama, data MNIST dimuat dan diskalakan ke rentang **[-1, 1]** agar sesuai dengan keluaran generator yang menggunakan aktivasi *tanh*. Selanjutnya, dibuat label target berupa **valid (1)** untuk gambar asli dan **fakes (0)** untuk gambar palsu. Pada setiap epoch, batch acak dari gambar asli dipilih, lalu generator menghasilkan gambar palsu dari noise acak. Diskriminator kemudian dilatih secara terpisah dengan data asli dan palsu untuk menghitung kerugian (**d_loss**). Setelah itu, generator dilatih melalui model GAN dengan tujuan menipu diskriminator, yakni membuat output mendekati label valid (1), sehingga diperoleh nilai kerugian generator (**g_loss**). Selama proses ini, dicetak informasi metrik berupa *loss* dan *accuracy* diskriminator serta *loss* generator pada setiap epoch. Selain itu, pada interval tertentu (misalnya setiap **200 epoch**), fungsi **save_imgs()** dipanggil untuk menyimpan hasil visualisasi gambar buatan generator. Dengan mekanisme ini, model GAN secara bertahap belajar menghasilkan gambar sintetis yang semakin mirip dengan data MNIST asli.

8) Making GIF

Membuat animasi GIF dari gambar yang dihasilkan selama pelatihan. Dengan menggunakan imageio, gambar yang disimpan dalam folder generated_images dibaca dan diurutkan. Gambar-gambar tersebut kemudian digabungkan menjadi sebuah file GIF (dcgan.gif) yang menampilkan proses pembuatan gambar dari generator selama pelatihan.

17] ✓ 0s

```
# 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)
```

```
/tmp/ipython-input-1295497862.py:11: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the current behavior (and
  image = imageio.imread(filename)
/tmp/ipython-input-1295497862.py:13: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the current behavior (and
  image = imageio.imread(filename)
```

PENJELASAN TEORI

Kode pada bagian ini digunakan untuk membuat **animasi GIF** yang menampilkan perkembangan kualitas gambar hasil generator selama pelatihan GAN. Pertama, ditentukan nama file output **dcgan.gif**. Selanjutnya, modul **glob** mencari semua file gambar PNG yang telah disimpan di folder *generated_images*, lalu file-file tersebut diurutkan agar animasi mengikuti urutan waktu pelatihan. Dengan menggunakan **imageio.get_writer()**, setiap gambar dibaca melalui **imageio.imread()** dan ditambahkan secara berurutan ke dalam GIF. Proses ini menghasilkan animasi yang memperlihatkan bagaimana generator secara bertahap belajar menghasilkan gambar angka tulisan tangan dari noise acak, sehingga kita bisa memvisualisasikan evolusi kualitas hasil sintesis dari awal hingga akhir pelatihan.