Nama : Rahma Putri Prabowo

NPM : 11122170

1. Import Library

Pada tahapan ini Anda mendefinisikan terlebih dahulu library yang akan Anda gunakan.

```
from keras.datasets import mnist
```

```
from tensorflow.keras import Sequential
from keras.layers import BatchNormalization, Dense, Reshape, Flatten
from keras.layers import LeakyReLU
from tensorflow.keras.optimizers import Adam
import numpy as np
```

2. Mendefinisikan Variabel untuk Neural Network dan Data

Pada tahapan ini definisikan beberapa variabel yang diperlukan diantaranya:

- Ukuran gambar yang akan di generate oleh algoritma GAN
- Channel warna yang akan digunakan
- Variable noise / latent yang akan membentuk gambar
- Optimizer - Variable yang menentukan algoritma optimisasi pembelajaran yang akan dilakukan. Disini kita menggunakan algoritma Stochastic Gradient Decent dengan learning rate 0.0001. Learning rate mendefinisikan seberapa cepat algoritma mempelajari data yang diberikan, dan nilai 0.0001 adalah nilai yang direkomendasikan digunakan untuk pembentukan model GAN.

t View Run Kernel Settings Help Trusted

Code JupyterLab Python [conda env:base]

```
# Mendefinisikan variable gambar
# Ukuran disesai
img_width = 28
img_height = 28
channels = 1
img_shape = (img_width, img_height, channels)
latent_dim = 100
adam = Adam(learning_rate=0.0001)
```

3. Membentuk Generator

Generator adalah bagian dari GAN yang bertugas untuk belajar membuat data palsu dengan memasukkan umpan balik dari diskriminator. Generator belajar membuat diskriminator mengklasifikasikan outputnya sebagai yang sebenarnya atau nyata.

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

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

    model.add(Dense(512))
    ## add activation function
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))

    model.add(Dense(1024))
    ## add activation function
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))

    ##membuat model menjadi ukuran 28x28x1
    model.add(Dense(np.prod(img_shape), activation='tanh'))
    model.add(Reshape(img_shape))

    model.summary()
    return model
```

```
generator = build_generator()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 256)               25856

 leaky_re_lu_10 (LeakyReLU)  (None, 256)               0

 batch_normalization_6 (Bat  (None, 256)               1024
 chNormalization)

 dense_13 (Dense)            (None, 512)               131584

 leaky_re_lu_11 (LeakyReLU)  (None, 512)               0

 batch_normalization_7 (Bat  (None, 512)               2048
 chNormalization)

 dense_14 (Dense)            (None, 1024)              525312

 leaky_re_lu_12 (LeakyReLU)  (None, 1024)              0

 batch_normalization_8 (Bat  (None, 1024)              4096
 chNormalization)

 dense_15 (Dense)            (None, 784)               803600

 reshape_2 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 1493520 (5.70 MB)
Trainable params: 1489936 (5.68 MB)
Non-trainable params: 3584 (14.00 KB)
```

View Run Kernel Settings Help Trusted

Code JupyterLab Python [conda env:base]

4. Mendefinisikan Discriminator

Diskriminator dalam GAN hanyalah sebuah fungsi klasifikasi. Ia mencoba untuk membedakan data nyata dari data yang dibuat oleh generator. Diskriminator bisa menggunakan arsitektur jaringan apa pun yang sesuai dengan jenis data yang diklasifikasikan.

```
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(LeakyReLU(alpha=0.2))

    model.summary()
    return model

discriminator = build_discriminator()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_16 (Dense)            (None, 512)               401920

 leaky_re_lu_13 (LeakyReLU)  (None, 512)               0

 dense_17 (Dense)            (None, 256)               131328

 leaky_re_lu_14 (LeakyReLU)  (None, 256)               0

=================================================================
Total params: 533248 (2.03 MB)
Trainable params: 533248 (2.03 MB)
Non-trainable params: 0 (0.00 Byte)
```

5. Menghubungkan Discriminator dan Generator untuk membentuk GAN

Pada tahapan ini kita akan menghubungkan discriminator dan generator yang telah dibuat untuk membentuk sebuah GAN.

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

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

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

6. Training

Pada tahapan ini kita akan menghubungkan discriminator dan generator yang telah dibuat untuk membentuk sebuah GAN.

```
def save_imgs(epoch):
    r, c = 5, 5
    noise = np.random.normal(0, 1, (r * c, latent_dim))
    gen_imgs = generator.predict(noise)

    gen_imgs = 0.5 * gen_imgs + 0.5  # rescale to [0,1]

    fig, axs = plt.subplots(r, c, figsize=(5,5))
    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].axis('off')
            cnt += 1
    fig.savefig(f"images/mnist_{epoch}.png")
    plt.close()
```

7. Testing

Pada tahapan ini kita akan menghubungkan discriminator dan generator yang telah dibuat untuk membentuk sebuah GAN.

```
train(epochs=10000, batch_size=64, save_interval=1000)
```

TEORI PENJELASAAN :

1. Import Library

Bagian ini memuat semua pustaka yang diperlukan:

- **Keras/TensorFlow** → untuk membuat arsitektur jaringan saraf.
- **MNIST** → dataset gambar angka 0–9.
- **Layer (Dense, Flatten, Reshape, BatchNormalization, LeakyReLU)** → komponen utama membentuk generator dan discriminator.

- **Adam** → optimizer untuk memperbarui bobot jaringan.
- **NumPy** → operasi numerik.
- **Matplotlib** → visualisasi hasil gambar.

2. Definisi Variabel

Menentukan:

- Ukuran gambar (**28×28 piksel, grayscale, 1 channel**).
- **Latent dimension (100)** → ruang acak sebagai input generator.
- Optimizer **Adam** dengan *learning rate* kecil agar training stabil.

3. Generator

Model **Generator** bertugas membuat gambar palsu menyerupai dataset asli.

- Input: vektor acak berdimensi 100.
- Diproses lewat beberapa lapisan Dense (256 → 512 → 1024 neuron).
- Aktivasi **LeakyReLU** untuk menambah non-linearitas.
- **BatchNormalization** untuk menjaga distribusi data stabil.
- Output: layer dengan ukuran sama dengan gambar target (28×28×1), aktivasi **tanh** (output -1 sampai 1).

Hasilnya adalah gambar palsu yang menyerupai angka.

4. Discriminator

Model **Discriminator** bertugas membedakan gambar nyata (dari dataset) dan palsu (dari generator).

- Input: gambar 28×28×1 → dipipihkan (Flatten).
- Diproses melalui Dense layer (512, 256 neuron).
- Menggunakan **LeakyReLU** sebagai fungsi aktivasi.
- (Idealnya ada layer output Dense(1, sigmoid) untuk menghasilkan probabilitas real/fake).

5. Kompilasi Model

- Discriminator dikompilasi dengan **binary crossentropy** karena hanya membedakan 2 kelas (real/fake).
- Model **GAN** = Generator + Discriminator:
    - Discriminator dibuat **tidak trainable** agar saat GAN dilatih, hanya Generator yang belajar memperbaiki hasil.
    - Tujuannya: Generator belajar membuat gambar yang bisa "menipu" Discriminator.

6. Fungsi Simpan Gambar

Fungsi ini mengambil noise acak, menghasilkan gambar melalui Generator, lalu menyimpannya sebagai file gambar.

- Output dibuat dalam bentuk grid 5×5.
- Normalisasi ke skala [0,1] agar bisa divisualisasikan dengan baik.
- Setiap beberapa epoch hasil disimpan untuk memantau progres training.

7. Training

Proses training dilakukan dengan:

1. **Melatih Discriminator**
    - Menggunakan gambar asli dari MNIST (label 1).
    - Menggunakan gambar palsu dari Generator (label 0).
    - Discriminator belajar membedakan keduanya.
2. **Melatih Generator (via GAN)**
    - Memberi input noise acak.
    - Discriminator dipaksa "percaya" bahwa gambar palsu adalah nyata.
    - Generator memperbarui bobot agar hasil gambar makin mirip data asli.
3. **Iterasi (epochs)**
    - Dilakukan ribuan kali (misalnya 10.000 epoch).
    - Setiap beberapa interval, hasil gambar disimpan.