Nama : Rahma Putri Prabowo

NPM : 11122170

Rahma Putri p_Pertemuan 3 - Moda Hands-On CiFAR GAN.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

1. Import Library

```
[7]
✓ 0s
from keras.datasets import cifar10, mnist
from keras.models import Model
from keras.models import Sequential
from keras.layers import Input
from keras.layers import Reshape
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import Dropout
from keras.layers import LeakyReLU
from tensorflow.keras.optimizers import Adam
import numpy as np
!mkdir generated_images
```

```
mkdir: cannot create directory ‘generated_images’: File exists
```

PENJELASAN TEORI

Pada tahap **import library** ini, dilakukan pemanggilan berbagai modul dan fungsi penting yang dibutuhkan untuk membangun arsitektur **Generative Adversarial Network (GAN)**. Library **Keras** digunakan untuk mengakses dataset seperti **CIFAR10** dan **MNIST**, serta menyediakan kelas dan lapisan neural network seperti **Sequential, Dense, Conv2D, Conv2DTranspose, Flatten, Reshape, Dropout**, dan **LeakyReLU** yang digunakan untuk membentuk generator maupun diskriminator. Selain itu, **Adam optimizer** diimpor dari TensorFlow untuk mengoptimalkan proses pelatihan, dan **NumPy** digunakan untuk operasi numerik seperti pembuatan noise acak. Baris terakhir membuat folder bernama **generated_images** yang berfungsi sebagai tempat penyimpanan hasil gambar sintetis yang dihasilkan oleh generator, namun muncul pesan bahwa folder sudah ada, menandakan direktori tersebut sebelumnya telah dibuat.

2. Mendefinisikan beberapa variabel

Mendefinisikan parameter untuk membangun sebuah model deep learning yang kemungkinan digunakan dalam arsitektur seperti Generative Adversarial Network (GAN) atau Convolutional Neural Network (CNN). Gambar yang akan diproses memiliki ukuran 32x32 piksel dengan 3 saluran warna (RGB), dan bentuk gambar tersebut ditetapkan dalam variabel img_shape. Selain itu, vektor laten yang digunakan untuk mempelajari representasi tersembunyi gambar memiliki dimensi 100. Optimizer yang dipilih untuk melatih model adalah Adam, dengan learning rate sebesar 0.0002.

```
[2]
✓
img_width = 32
img_height = 32
channels= 3
img_shape = (img_width, img_height, channels)
latent_dim = 100
adam = Adam(learning_rate=0.0002)
```

PENJELASAN TEORI

Kode di atas berfungsi untuk mendefinisikan beberapa **parameter utama** yang akan digunakan dalam membangun model deep learning berbasis **GAN atau CNN**. Variabel **img_width**, **img_height**, dan **channels** menunjukkan bahwa data input berupa gambar berukuran **32x32 piksel** dengan **3 saluran warna (RGB)**, sesuai dengan format dataset seperti CIFAR10. Ketiga parameter ini digabungkan dalam variabel **img_shape** yang mendeskripsikan bentuk input bagi model. Selanjutnya, **latent_dim = 100** digunakan untuk menentukan ukuran ruang laten, yaitu vektor acak yang menjadi input bagi generator dalam menghasilkan gambar sintetis baru. Terakhir, digunakan **optimizer Adam** dengan *learning rate* **0.0002**, yang dikenal efektif dalam mempercepat dan menstabilkan proses pelatihan jaringan saraf dalam menghasilkan gambar yang realistis.

3. Membentuk generator

Membangun generator untuk sebuah model GAN, yang berfungsi menghasilkan gambar dari vektor laten. Dimulai dengan dense layer yang mengubah input menjadi tensor, lalu menggunakan beberapa lapisan Conv2DTranspose untuk memperbesar gambar secara bertahap. Setiap lapisan dilengkapi dengan fungsi aktivasi LeakyReLU untuk menambah non-linearitas. Pada akhirnya, layer Conv2D terakhir membentuk gambar output dengan 3 channel (RGB) menggunakan aktivasi 'tanh'.

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

  # create first Layer, to receive the input
  model.add(Dense (256 * 4 * 4, input_dim = latent_dim))
  # 256 * 8 * 8; for upscaling the Layers
  #initial shape to construct into final shape

  # Create default activation function
  model.add(LeakyReLU(alpha = 0.2))

  # Create reshape Layer
  model.add(Reshape((4, 4,256)))
  # 6,8,256; reffers to first layer

  # Adding more Layers for neurons and better result
  model.add(Conv2DTranspose(128, (4,4), strides = (2,2), padding = 'same'))
  model.add(LeakyReLU(alpha= 0.2))
  model.add(Conv2DTranspose(128, (4,4), strides = (2,2), padding = 'same'))
  model.add(LeakyReLU (alpha= 0.2))
  model.add(Conv2DTranspose(128, (4,4), strides = (2,2), padding = 'same'))
  model.add(LeakyReLU(alpha= 0.2))
  # (4,4) >> filter size
  # strides (2,2) >> Convolutional Layers, that how NN understand images

  # Create Final output Layer and forming image shape
  # the shape (3, (3,3)) reffers to image shape :
  # >>> img_shape (1mg width, img_height, channels)
  model.add(Conv2D (3, (3,3), activation = 'tanh', padding = 'same'))

  model.summary()
  return model

generator = build_generator()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 4096)	413,696
leaky_re_lu (LeakyReLU)	(None, 4096)	0
reshape (Reshape)	(None, 4, 4, 256)	0
conv2d_transpose (Conv2DTranspose)	(None, 8, 8, 128)	524,416
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 128)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 16, 16, 128)	262,272
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 32, 32, 128)	262,272
leaky_re_lu_3 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d (Conv2D)	(None, 32, 32, 3)	3,459

```
Total params: 1,466,115 (5.59 MB)
Trainable params: 1,466,115 (5.59 MB)
Non-trainable params: 0 (0.00 B)
```

PENJELASAN TEORI

Fungsi **build_generator()** pada kode di atas digunakan untuk membangun arsitektur **generator** dalam model GAN, yang bertugas menghasilkan gambar sintetis dari vektor acak (*latent vector*). Proses dimulai dengan lapisan **Dense** berukuran besar untuk mengubah input berdimensi 100 menjadi tensor berukuran **4×4×256**, yang kemudian diaktifkan menggunakan **LeakyReLU** agar pembelajaran lebih stabil. Setelah itu, dilakukan **Reshape** agar tensor memiliki bentuk menyerupai gambar awal sebelum diperbesar secara bertahap melalui beberapa lapisan **Conv2DTranspose**, yang berfungsi melakukan *upsampling* (memperbesar dimensi gambar) hingga mencapai ukuran akhir **32×32 piksel**. Setiap lapisan dilengkapi dengan aktivasi **LeakyReLU** untuk menambah non-linearitas dan meningkatkan kualitas hasil gambar. Pada bagian akhir, lapisan **Conv2D** dengan **3 filter** dan aktivasi **tanh** digunakan untuk menghasilkan gambar akhir dengan **3 channel (RGB)** dalam rentang nilai [-1, 1]. Berdasarkan ringkasan model, generator memiliki **1.466.115 parameter trainable**, menunjukkan bahwa jaringan ini cukup kompleks untuk mempelajari distribusi data dan menghasilkan gambar yang menyerupai data asli.

4. Mendefinisikan discriminator

Membangun discriminator untuk GAN, yang berfungsi membedakan gambar asli dari yang palsu. Discriminator menggunakan beberapa lapisan Conv2D dengan fungsi aktivasi LeakyReLU untuk mengekstrak fitur gambar, kemudian lapisan Flatten dan Dropout untuk mencegah overfitting. Akhirnya, output dihasilkan oleh lapisan Dense dengan aktivasi sigmoid yang memberikan probabilitas apakah gambar itu nyata atau hasil buatan. Model ini dikompilasi menggunakan binary crossentropy sebagai loss dan Adam sebagai optimizer.

[4] ✓ 0s

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

    # Create Input layer and filter and stride layer. That makes NN understand (mage
    model.add(Conv2D(64, (3,3), padding = 'same', input_shape = img_shape))

    # Adding activation function
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Conv2D(128, (3,3), padding = 'same'))
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Conv2D(128, (3,3), padding = 'same'))
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Conv2D (256, (3,3), padding = 'same'))
    model.add(LeakyReLU(alpha = 0.2))
    model.add(Flatten())

    model.add(Dropout(0.4))
```

[4] ✓ 0s

```python
    # Create output Layer
    model.add(Dense(1, activation = 'sigmoid'))

    model.summary()
    return model

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

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Se
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 64)	1,792
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 64)	0
conv2d_2 (Conv2D)	(None, 32, 32, 128)	73,856
leaky_re_lu_5 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	147,584
leaky_re_lu_6 (LeakyReLU)	(None, 32, 32, 128)	0
conv2d_4 (Conv2D)	(None, 32, 32, 256)	295,168
leaky_re_lu_7 (LeakyReLU)	(None, 32, 32, 256)	0
flatten (Flatten)	(None, 262144)	0
dropout (Dropout)	(None, 262144)	0
dense_1 (Dense)	(None, 1)	262,145

```
Total params: 780,545 (2.98 MB)
Trainable params: 780,545 (2.98 MB)
Non-trainable params: 0 (0.00 B)
```

PENJELASAN TEORI

Fungsi **build_discriminator()** digunakan untuk membangun arsitektur *discriminator* dalam model GAN, yang berfungsi membedakan antara gambar asli dan gambar hasil buatan generator. Model ini dimulai dengan beberapa lapisan **Conv2D** berurutan dengan jumlah filter yang meningkat (64, 128, 256) dan fungsi aktivasi **LeakyReLU (α=0.2)** untuk mengekstraksi fitur penting dari gambar tanpa menyebabkan masalah *vanishing gradient*. Setelah itu, hasil ekstraksi fitur diubah menjadi vektor satu dimensi melalui lapisan **Flatten**, kemudian diterapkan **Dropout (0.4)** untuk mengurangi risiko *overfitting*. Lapisan terakhir adalah **Dense(1)** dengan aktivasi **sigmoid**, yang menghasilkan nilai probabilitas antara 0 dan 1 untuk menunjukkan apakah gambar tergolong asli atau palsu. Model ini dikompilasi menggunakan *loss function* **binary crossentropy** dan *optimizer* **Adam**, menghasilkan total **780.545 parameter trainable**, yang memungkinkan *discriminator* belajar secara efektif membedakan distribusi data nyata dan data hasil sintesis generator.

5. Menghubungkan Discriminator dan Generator untuk membentuk GAN

Membangun GAN dengan menggabungkan generator dan discriminator. Discriminator dibuat tidak dapat dilatih agar hanya generator yang dilatih. Vektor laten dimasukkan ke generator untuk menghasilkan gambar, lalu gambar ini dinilai oleh discriminator. Model GAN ini dikompilasi menggunakan binary crossentropy sebagai loss dan Adam sebagai optimizer, sehingga dapat melatih generator untuk menghasilkan gambar yang realistis.

[5]
✓ 0s

```
def build_gan(generator, discriminator):
    discriminator.trainable = False
    gan_input = Input(shape=(latent_dim,))
    generated_image = generator(gan_input)
    gan_output = discriminator(generated_image)
    gan = Model(gan_input, gan_output)

    gan.compile(loss='binary_crossentropy', optimizer=adam)
    return gan

gan = build_gan(generator, discriminator)
gan.summary()
```

```
Model: "functional_21"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 100)	0
sequential (Sequential)	(None, 32, 32, 3)	1,466,115
sequential_1 (Sequential)	(None, 1)	780,545

```
Total params: 2,246,660 (8.57 MB)
Trainable params: 1,466,115 (5.59 MB)
Non-trainable params: 780,545 (2.98 MB)
```

PENJELASAN TEORI

Fungsi **build_gan()** digunakan untuk membangun keseluruhan arsitektur **Generative Adversarial Network (GAN)** dengan cara menggabungkan model **generator** dan **discriminator**. Pada proses ini, parameter **discriminator** dibuat tidak dapat dilatih (**trainable = False**) agar selama pelatihan hanya **generator** yang diperbarui, sehingga generator belajar menghasilkan gambar yang mampu menipu discriminator. Arsitektur GAN dimulai dengan **input** berupa vektor laten berdimensi 100 yang dimasukkan ke generator untuk menghasilkan gambar berukuran 32×32 piksel dengan 3 saluran warna (RGB), kemudian gambar tersebut dievaluasi oleh discriminator untuk memberikan output probabilitas antara 0 (palsu) dan 1 (asli). Model GAN dikompilasi menggunakan **binary crossentropy** sebagai fungsi loss dan **Adam optimizer** dengan *learning rate* 0.0002, menghasilkan total **2.246.660 parameter**, di mana hanya **1.466.115 parameter** dari generator yang dilatih. Dengan struktur ini, GAN dapat melatih generator agar semakin mahir menciptakan gambar yang menyerupai data nyata berdasarkan umpan balik dari discriminator.