Nama : Muhammad Fauzan

NPM : 11122025

Kelas : 4KA07

PTA 2025/2026 | 4-FIKTI | Praktikum Robotika Cerdas

Pertemuan 3 - Moda Hands-On: CiFAR GAN

Laporan Minggu ke-3

Pengertian CiFAR GAN

GAN

GAN adalah jenis jaringan saraf tiruan yang digunakan untuk **menghasilkan data baru yang mirip dengan data asli**.
Diperkenalkan oleh *Ian Goodfellow (2014)*, GAN terdiri dari dua komponen utama:

Komponen	Fungsi
Generator (G)	Menciptakan gambar palsu dari *noise* (acak) dengan tujuan meniru gambar asli.
Discriminator (D)	Menilai apakah gambar yang diterima adalah asli (dari dataset CIFAR-10) atau palsu (dari generator).

Keduanya saling **berkompetisi**:

- Generator berusaha menipu Discriminator dengan gambar palsu yang realistis.
- Discriminator berusaha mengenali mana yang asli dan mana yang palsu.
  Hasil akhirnya: Generator menjadi sangat mahir menghasilkan gambar yang *nyaris tak bisa dibedakan* dari data asli.

CiFAR

CIFAR-10 adalah dataset gambar yang sangat populer di dunia *deep learning*, terutama untuk pelatihan model pengenalan dan pembangkitan gambar.

- Berisi **60.000 gambar berwarna** berukuran **32x32 piksel**.
- Terdiri dari **10 kelas** berbeda:

  pesawat, mobil, burung, kucing, rusa, anjing, katak, kuda, kapal, truk.

- Setiap kelas memiliki 6.000 gambar (50.000 untuk pelatihan, 10.000 untuk pengujian).

Dataset ini sering digunakan untuk melatih model **CNN (Convolutional Neural Network)** maupun **GAN (Generative Adversarial Network)** karena ukurannya kecil tapi cukup kompleks.

CiFAR GAN

CIFAR-GAN berarti *Generative Adversarial Network yang dilatih menggunakan dataset CIFAR-10*.

Tujuannya:

- **Membuat gambar baru** yang menyerupai 10 kelas objek di CIFAR-10 (misalnya mobil, anjing, pesawat, dll).
- Menguji kemampuan model GAN dalam **belajar distribusi gambar berwarna kecil**.

Biasanya arsitekturnya terdiri dari:

- **Generator:** Menggunakan lapisan *Dense*, *BatchNormalization*, dan *Conv2DTranspose* untuk menghasilkan gambar 32×32×3.
- **Discriminator:** Menggunakan lapisan *Conv2D* dan *LeakyReLU* untuk membedakan gambar asli dan palsu.
- **Optimisasi:** Dilakukan dengan *Adam optimizer*, dan fungsi kehilangan (*loss*) seperti *binary crossentropy*.

Implementasi CiFAR GAN

1. Import Library

```
from keras.datasets import cifar10, mnist
from keras.models import Sequential
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
```

A. Penjelasan Kode

Kode ini diawali dengan **mengimpor beberapa modul penting dari pustaka Keras dan TensorFlow**, serta modul lain dari Python standar.

- `from keras.datasets import cifar10, mnist` digunakan untuk mengambil **dataset gambar CIFAR-10 dan MNIST** yang umum digunakan untuk pelatihan model pengenalan atau pembangkitan gambar.
- `from keras.models import Sequential` digunakan untuk membuat **model berurutan (Sequential model)** di mana lapisan-lapisan (*layers*) ditambahkan satu per satu.
- `from keras.layers import ...` memuat berbagai jenis lapisan yang diperlukan untuk membentuk arsitektur jaringan:
    - `Reshape` digunakan untuk mengubah bentuk (dimensi) data, misalnya dari vektor menjadi citra 2D.
    - `Dense` adalah lapisan *fully connected*, di mana setiap neuron terhubung ke semua neuron di lapisan sebelumnya.
    - `Flatten` digunakan untuk mengubah data 2D (misalnya gambar) menjadi bentuk 1D agar bisa diproses oleh lapisan Dense.
    - `Conv2D` adalah lapisan **konvolusi dua dimensi**, digunakan untuk mengekstraksi fitur dari gambar.
    - `Conv2DTranspose` biasanya digunakan untuk **menghasilkan gambar baru (upsampling)**, sangat umum pada generator dalam GAN.
    - `Dropout` digunakan untuk **mengurangi overfitting** dengan menonaktifkan sebagian neuron secara acak saat pelatihan.
    - `LeakyReLU` adalah fungsi aktivasi non-linear yang memperbaiki kelemahan ReLU dengan tetap memberikan nilai kecil untuk input negatif, membantu aliran gradien agar tidak hilang (*vanishing gradient problem*).
- `from tensorflow.keras.optimizers import Adam` mengimpor **optimizer Adam**, algoritma populer untuk mempercepat dan menstabilkan proses pelatihan model.

- `import numpy as np` digunakan untuk **manipulasi data numerik**, seperti pengolahan matriks dan array.

Akhirnya, baris `!mkdir generated_images` adalah perintah sistem untuk **membuat folder baru bernama `generated_images`**. Folder ini biasanya digunakan untuk **menyimpan hasil gambar yang dihasilkan model**, misalnya gambar palsu dari generator dalam model GAN.

---

2. Parameters for Neural Networks & Data

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

A. Penjelasan Kode

Baris pertama hingga ketiga, yaitu:

```
img_width = 32
img_height = 32
channels = 3
```

menetapkan **dimensi gambar** yang digunakan. Dalam hal ini, setiap gambar memiliki ukuran **32 piksel × 32 piksel** dengan **3 saluran warna (RGB)**. Nilai-nilai tersebut diambil dari karakteristik dataset **CIFAR-10**, karena seluruh gambar dalam dataset itu berukuran 32×32 dan berwarna.

Selanjutnya:

```
img_shape = (img_width, img_height, channels)
```

digunakan untuk **menggabungkan ketiga nilai sebelumnya menjadi satu tuple** yang merepresentasikan bentuk (shape) gambar, yaitu `(32, 32, 3)`. Variabel ini nantinya akan dipakai saat mendefinisikan arsitektur jaringan — misalnya untuk lapisan *input* pada *Discriminator* atau *Generator*, agar model tahu ukuran data yang akan diproses.

Kemudian,

```
latent_dim = 100
```

menentukan ukuran **ruang laten (latent space)**, yaitu panjang vektor acak yang akan diberikan sebagai masukan ke **Generator**. Vektor ini berfungsi sebagai "benih" (*seed*) dari

mana Generator akan membentuk gambar palsu baru. Nilai 100 adalah ukuran yang umum digunakan, karena cukup besar untuk menampung variasi bentuk gambar tanpa membuat pelatihan terlalu berat.

Terakhir,

```
adam = Adam(learning_rate=0.0002)
```

mendefinisikan **optimizer** yang digunakan untuk memperbarui bobot jaringan selama pelatihan. Optimizer *Adam* (Adaptive Moment Estimation) merupakan algoritma populer karena dapat menyesuaikan *learning rate* secara adaptif dan mempercepat konvergensi model. Nilai `learning_rate=0.0002` adalah pengaturan yang sering digunakan dalam pelatihan **GAN**, karena terlalu besar dapat menyebabkan pelatihan tidak stabil, sedangkan terlalu kecil membuat proses belajar sangat lambat.

---

3. Building Generator

```
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)))
    # 8,8,256 ; 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 = (img_width, img_height, channels)
    model.add(Conv2D(3, (3,3), activation= 'tanh', padding =
'same'))

    #
    model.summary()
    return model

generator = build_generator()
```

A. Penjelasan Kode

Fungsi `build_generator()` digunakan untuk **membuat arsitektur Generator** menggunakan model berurutan (`Sequential`) dari Keras. Generator ini menerima **vektor acak berdimensi 100** (dari variabel `latent_dim)` sebagai masukan, lalu secara bertahap mengubahnya menjadi **gambar berwarna berukuran 32×32 piksel** seperti pada dataset CIFAR-10.

Pertama, baris:

```
model.add(Dense(256 * 4 * 4, input_dim = latent_dim))
```

menambahkan lapisan **Dense (fully connected layer)** dengan jumlah neuron sebesar `256 * 4 * 4 = 4096`. Lapisan ini menerima input acak berdimensi 100 dan mengubahnya menjadi representasi awal yang kaya fitur untuk nantinya dibentuk menjadi gambar. Lapisan ini bertindak sebagai “pondasi awal” dari Generator.

Selanjutnya:

```
model.add(LeakyReLU(alpha = 0.2))
```

menambahkan **fungsi aktivasi LeakyReLU**, yang digunakan agar jaringan dapat belajar pola non-linear dan menghindari masalah *dead neuron* (neuron yang tidak aktif sama sekali). Nilai `alpha=0.2` menentukan seberapa besar kebocoran (leak) untuk input negatif.

Kemudian, lapisan:

```
model.add(Reshape((4, 4, 256)))
```

berfungsi untuk **mengubah output vektor 1D menjadi tensor 3D** dengan bentuk 4×4 piksel dan 256 saluran fitur. Inilah titik awal di mana vektor acak mulai "dibentuk" menjadi citra dua dimensi.

Setelah itu, tiga lapisan berikut:

```
model.add(Conv2DTranspose(128, (4,4), strides = (2,2), padding = 'same'))
model.add(LeakyReLU(alpha= 0.2))
```

digunakan secara bertahap untuk melakukan **upsampling**, yaitu memperbesar dimensi gambar dari 4×4 → 8×8 → 16×16 → 32×32.
Lapisan **Conv2DTranspose** dikenal juga sebagai *deconvolution layer*, yang memungkinkan model menghasilkan gambar yang semakin besar dan detail pada setiap tahap. Lapisan *LeakyReLU* ditambahkan di antara setiap konvolusi agar model dapat belajar representasi yang lebih kompleks dan halus.

Terakhir, baris:

```
model.add(Conv2D(3, (3,3), activation='tanh', padding='same'))
```

merupakan **lapisan output** yang menghasilkan gambar RGB dengan 3 saluran warna. Fungsi aktivasi `tanh` digunakan karena nilai outputnya berada dalam rentang -1 hingga 1 — cocok untuk data gambar yang telah dinormalisasi dalam kisaran tersebut.

Perintah:

```
model.summary()
```

menampilkan **rangkuman arsitektur jaringan** di konsol, termasuk jumlah parameter dan ukuran setiap lapisan.
Kemudian `return model` mengembalikan objek model Generator yang telah dibuat, dan baris `generator = build_generator()` menyimpannya ke variabel `generator`.

B. HASIL

Model : "Sequential"

Layer (Type)	Output Shape	Parameter Count
Dense	(None, 4096)	413,696
LeakyReLU	(None, 4096)	0
Reshape	(None, 4, 4, 256)	0
Conv2DTranspose	(None, 8, 8, 128)	524,416
LeakyReLU	(None, 8, 8, 128)	0
Conv2DTranspose	(None, 16, 16, 128)	262,272

LeakyReLU	(None, 16, 16, 128)	0
Conv2DTranspose	(None, 32, 32, 128)	262,272
LeakyReLU	(None, 32, 32, 128)	0
Conv2D	(None, 32, 32, 3)	3,459

Total Parameter

- Total Parameters: 1,466,115 (≈ 5.59 MB)
- Trainable Parameters: 1,466,115
- Non-trainable Parameters: 0

C. Penjelasan Hasil

- **Lapisan pertama (Dense)** menghasilkan keluaran berukuran `(None, 4096)` dengan total **413.696 parameter**. Lapisan ini mengubah vektor acak berdimensi 100 menjadi representasi awal berdimensi besar (4096 unit).
- **Lapisan LeakyReLU** berikutnya tidak menambah parameter, hanya menambahkan fungsi aktivasi non-linear agar jaringan dapat belajar lebih efektif.
- **Lapisan Reshape** mengubah vektor 4096 menjadi bentuk tiga dimensi `(4, 4, 256)` sebagai tahap awal pembentukan gambar.
- **Tiga lapisan Conv2DTranspose** bertugas melakukan proses *upsampling* bertahap:
  - Lapisan pertama menghasilkan output `(8, 8, 128)` dengan **524.416 parameter**,
  - Lapisan kedua `(16, 16, 128)` dengan **262.272 parameter**,
  - Lapisan ketiga `(32, 32, 128)` juga dengan **262.272 parameter**.
    Masing-masing lapisan ini diikuti oleh **LeakyReLU** untuk menjaga kestabilan aliran gradien dan hasil pembelajaran yang halus.
- **Lapisan terakhir (Conv2D)** menghasilkan gambar akhir berukuran `(32, 32, 3)` (sesuai ukuran gambar CIFAR-10) dengan **3.459 parameter**. Fungsi aktivasi `tanh` pada lapisan ini memastikan nilai piksel berada pada rentang -1 hingga 1.

Di bagian bawah ditampilkan total keseluruhan:

Total parameter = 1.466.115, semuanya **trainable (dapat dilatih)**, yang berarti setiap parameter akan disesuaikan selama proses pelatihan Generator.

Ukuran model sekitar **5.59 MB**, yang tergolong kecil hingga sedang untuk model GAN pada dataset CIFAR-10.

---

4. Building Discriminator

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

```
    # Create input layer and filter and stride layer. That
makes NN understand image
    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))

    # 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'])
```

A. Penjelasan KODE

Pada bagian pertama, fungsi `build_discriminator()` membuat model bertipe **Sequential**, yang berarti lapisan-lapisan jaringan akan ditambahkan secara berurutan. Lapisan pertama adalah **Conv2D** dengan 64 filter berukuran 3x3 dan padding `'same'`, yang berarti ukuran keluaran sama dengan masukan. Lapisan ini digunakan untuk mengekstraksi fitur dasar dari gambar, seperti tepi atau tekstur. Parameter `input_shape = img_shape` mendefinisikan ukuran gambar masukan, yaitu 32x32 piksel dengan 3 saluran warna (RGB).

Selanjutnya, digunakan beberapa lapisan **LeakyReLU** dengan parameter `alpha = 0.2`, yaitu fungsi aktivasi yang membantu mencegah masalah *dead neurons* dengan membiarkan sedikit gradien mengalir saat nilai negatif. Setelah itu, beberapa lapisan **Conv2D** tambahan ditambahkan dengan jumlah filter yang meningkat (128 dan 256), bertujuan untuk menangkap pola yang lebih kompleks dari gambar masukan.

Kemudian, hasil dari lapisan konvolusi diubah menjadi bentuk vektor 1 dimensi menggunakan **Flatten()**, dan diberikan lapisan **Dropout(0.4)** untuk mencegah *overfitting* dengan cara menonaktifkan sebagian neuron secara acak saat pelatihan.

Terakhir, lapisan keluaran (**Dense**) memiliki 1 neuron dengan fungsi aktivasi **sigmoid**, yang menghasilkan output antara 0 dan 1. Nilai ini merepresentasikan probabilitas apakah gambar yang diberikan adalah asli (mendekati 1) atau palsu (mendekati 0).

Setelah model selesai dibuat, ia dikompilasi menggunakan **loss function** `binary_crossentropy` karena tugasnya bersifat klasifikasi biner (asli vs palsu). Optimizer yang digunakan adalah **Adam** dengan *learning rate* 0.0002, yang membantu mempercepat konvergensi model. Akhirnya, metrik evaluasi yang digunakan adalah **accuracy**, untuk mengukur sejauh mana *discriminator* mampu mengenali gambar dengan benar.

Secara keseluruhan, kode ini membangun *discriminator* sebagai "penilai" dalam GAN, yang belajar untuk semakin mahir membedakan gambar nyata dari gambar buatan *generator*, sehingga keduanya saling berkompetisi dan meningkatkan kualitas hasil akhir gambar yang dihasilkan.

B. HASIL

Model : "sequential_1"

Layer (Type)	Output Shape	Parameter Count
Conv2D	(None, 32, 32, 64)	1,792
LeakyReLU	(None, 32, 32, 64)	0
Conv2D	(None, 32, 32, 128)	73,856
LeakyReLU	(None, 32, 32, 128)	0
Conv2D	(None, 32, 32, 128)	147,584
LeakyReLU	(None, 32, 32, 128)	0
Conv2D	(None, 32, 32, 256)	295,168
LeakyReLU	(None, 32, 32, 256)	0
Flatten	(None, 262144)	0
Dropout	(None, 262144)	0
Dense	(None, 1)	262,145

Total Parameter

- Total Parameters: 780,545 (≈ 2.98 MB)
- Trainable Parameters: 780,545
- Non-trainable Parameters: 0

C. Penjelasan Hasil

Lapisan pertama, **Conv2D (conv2d_1)**, memiliki 64 filter berukuran 3x3 dan menghasilkan keluaran dengan dimensi `(32, 32, 64)`. Lapisan ini berfungsi untuk mengekstraksi fitur-fitur dasar dari gambar input (seperti tepi atau tekstur). Jumlah parameter pada lapisan ini adalah **1.792**, yang terdiri dari bobot dan bias yang dapat dilatih.

Selanjutnya, setiap lapisan konvolusi diikuti oleh **LeakyReLU** sebagai fungsi aktivasi (misalnya `leaky_re_lu_4, leaky_re_lu_5`, dan seterusnya). Fungsi aktivasi ini menjaga agar neuron tetap aktif meskipun menerima nilai negatif kecil, sehingga aliran gradien tidak terputus selama pelatihan.

Lapisan konvolusi berikutnya (`conv2d_2`, `conv2d_3`, dan `conv2d_4`) memiliki jumlah filter yang meningkat dari 128 hingga 256. Peningkatan jumlah filter ini membuat jaringan mampu menangkap pola visual yang lebih kompleks dan mendalam dari gambar. Total parameter dalam lapisan-lapisan ini juga meningkat secara signifikan — misalnya, **conv2d_4** memiliki **295.168 parameter** karena lebih banyak filter dan saluran masukan dari lapisan sebelumnya.

Setelah seluruh proses ekstraksi fitur, lapisan **Flatten** mengubah keluaran tiga dimensi menjadi vektor satu dimensi berukuran **262.144**, sehingga data dapat diproses oleh lapisan *Dense* di bagian akhir. Lapisan **Dropout** kemudian digunakan untuk menonaktifkan sebagian neuron secara acak selama pelatihan (dalam hal ini sebesar 40%), tujuannya untuk mengurangi risiko *overfitting*.

Lapisan terakhir adalah **Dense (dense_1)** dengan satu neuron keluaran dan fungsi aktivasi **sigmoid**, yang menghasilkan nilai antara 0 hingga 1. Nilai ini menandakan seberapa besar kemungkinan gambar yang diberikan merupakan gambar asli (mendekati 1) atau gambar palsu hasil *generator* (mendekati 0).

Secara keseluruhan, model ini memiliki **total 780.545 parameter yang dapat dilatih**, dengan ukuran sekitar **2,98 MB**. Semua parameter tersebut bersifat *trainable*, artinya akan diperbarui selama proses pelatihan untuk membuat *discriminator* semakin akurat dalam membedakan gambar asli dan palsu.

---

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()
```

A. Penjelasan Kode

Langkah pertama `GAN = Sequential()` membuat sebuah objek model berurutan baru bernama **GAN**. Ini akan menjadi model utama yang menyatukan dua komponen utama: *generator* dan *discriminator*. Sebelum model ini digabungkan, dilakukan perintah `discriminator.trainable = False`. Langkah ini sangat penting karena ketika melatih GAN, hanya *generator* yang perlu diperbarui bobotnya, sementara *discriminator* dibuat tidak dapat dilatih untuk sementara. Tujuannya adalah agar *generator* belajar menghasilkan gambar yang semakin realistis tanpa mengubah parameter dari *discriminator* pada saat yang sama.

Selanjutnya, `GAN.add(generator)` menambahkan model *generator* ke dalam arsitektur GAN. *Generator* bertugas membuat gambar palsu dari *noise* (data acak) yang nantinya akan diperiksa oleh *discriminator*. Kemudian, `GAN.add(discriminator)` menambahkan model *discriminator* setelah *generator*. Dengan urutan ini, setiap data acak yang dimasukkan ke GAN akan terlebih dahulu diubah menjadi gambar oleh *generator*, lalu hasilnya akan langsung diuji oleh *discriminator* untuk menilai apakah gambar tersebut tampak asli atau tidak.

Baris `GAN.compile(loss='binary_crossentropy', optimizer=adam)` digunakan untuk mengonfigurasi model agar siap dilatih. Fungsi *loss* yang digunakan adalah **binary crossentropy**, karena output yang diharapkan dari *discriminator* hanya dua kelas — yaitu *real (1)* atau *fake (0)*. Optimizer **Adam** dengan *learning rate* kecil (0.0002, seperti yang didefinisikan sebelumnya) digunakan agar pelatihan stabil dan konvergen secara bertahap.

Terakhir, `GAN.summary()` menampilkan ringkasan struktur model gabungan, termasuk jumlah parameter total dari *generator* dan *discriminator*, serta menunjukkan bagaimana kedua model tersebut tersusun secara berurutan. Secara keseluruhan, kode ini membentuk inti dari sistem GAN, di mana *generator* berperan sebagai "pembuat gambar palsu", sedangkan *discriminator* berperan sebagai "detektor keaslian gambar". Kedua model ini akan bersaing dalam proses pelatihan, sehingga *generator* semakin mahir menciptakan gambar yang menyerupai data asli.

B. HASIL

Model : "sequential_2"

Layer (Type)	Output Shape	Parameter Count
Sequential	(None, 32, 32, 3)	1,466,115
Sequential_1	(None, 1)	780,545

Total Parameter

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

C. Penjelasan HASIL

Lapisan pertama, **sequential**, merepresentasikan model *generator* yang sebelumnya telah dibuat. Output-nya memiliki bentuk `(None, 32, 32, 3)`, yang berarti model ini menghasilkan gambar berukuran **32x32 piksel** dengan **3 saluran warna (RGB)**. Jumlah parameter pada bagian ini adalah **1.466.115**, yang seluruhnya merupakan parameter **trainable** — artinya dapat diperbarui selama proses pelatihan. Hal ini sesuai dengan strategi

GAN, di mana pada tahap pelatihan gabungan, hanya *generator* yang dilatih agar semakin mahir membuat gambar yang realistis.

Lapisan berikutnya, **sequential_1**, merepresentasikan model *discriminator*. Model ini menerima output dari *generator* (gambar palsu) dan mengeluarkan nilai probabilitas `(None, 1)` — menunjukkan seberapa besar kemungkinan gambar tersebut dianggap "asli". Jumlah parameter pada *discriminator* adalah **780.545**, namun pada model GAN ini parameter tersebut bersifat **non-trainable**, karena sebelumnya telah diatur dengan perintah `discriminator.trainable = False.` Tujuannya adalah agar selama pelatihan GAN, hanya *generator* yang mengalami pembaruan bobot, sementara *discriminator* tetap digunakan sebagai penilai tetap (tidak berubah).

Secara keseluruhan, model GAN ini memiliki **total 2.246.660 parameter** atau sekitar **8,57 MB**. Dari jumlah tersebut, hanya **1.466.115 parameter (5,59 MB)** yang bersifat trainable, yaitu milik *generator*, sedangkan **780.545 parameter (2,98 MB)** milik *discriminator* tidak dilatih. Ringkasan ini memperlihatkan struktur hierarki GAN yang sesungguhnya: *generator* membuat gambar, dan *discriminator* mengevaluasinya, membentuk sistem pembelajaran dua arah yang saling berkompetisi untuk menghasilkan gambar yang semakin menyerupai data asli.

---

6. Outputting Images

```
import matplotlib.pyplot as plt
import os

def save_generated_images(epoch, generator, examples=100,
dim=(10, 10), figsize=(10, 10)):
    noise = np.random.normal(loc=0, scale=1, size=[examples,
latent_dim])
    generated_images = generator.predict(noise)

    # Rescale images to 0-1
    generated_images = 0.5 * generated_images + 0.5

    fig, axs = plt.subplots(dim[0], dim[1], figsize=figsize)
    cnt = 0
    for i in range(dim[0]):
        for j in range(dim[1]):
            axs[i, j].imshow(generated_images[cnt, :, :, :])
            axs[i, j].axis('off')
            cnt += 1
    fig.savefig(os.path.join('generated_images',
f'gan_generated_image_epoch_{epoch}.png'))
    plt.close()
```

A. Penjelasan Kode

Fungsi menerima beberapa parameter, yaitu:

- `epoch`: menandakan tahap pelatihan keberapa saat gambar disimpan.
- `generator`: model *generator* yang telah dilatih.
- `examples`: jumlah gambar yang ingin dihasilkan (default 100).
- `dim`: menentukan susunan grid gambar (misalnya 10x10).
- `figsize`: ukuran keseluruhan gambar keluaran yang akan disimpan.

Di dalam fungsi, pertama dibuat data acak bernama `noise` menggunakan `np.random.normal()`, yang menghasilkan vektor acak dari distribusi normal dengan rata-rata 0 dan standar deviasi 1. Vektor ini memiliki dimensi `[examples, latent_dim]`, yang akan menjadi input untuk *generator*. Model *generator* kemudian menghasilkan gambar sintetis dengan perintah `generator.predict(noise)`.

Karena output *generator* menggunakan aktivasi **tanh**, nilai piksel gambar berada dalam rentang -1 hingga 1. Oleh karena itu, baris `generated_images = 0.5 * generated_images + 0.5` digunakan untuk melakukan **rescaling** agar nilai piksel kembali ke rentang 0–1, sehingga dapat ditampilkan dengan benar menggunakan `matplotlib`.

Selanjutnya, `plt.subplots()` membuat grid berukuran sesuai parameter `dim`, misalnya 10 baris dan 10 kolom, dengan ukuran keseluruhan `figsize=(10, 10)`. Perulangan bersarang `for i in range(dim[0])` dan `for j in range(dim[1])` menampilkan setiap gambar hasil *generator* pada posisi grid yang sesuai. Sumbu setiap gambar dimatikan dengan `axs[i, j].axis('off')` agar hasil terlihat bersih tanpa garis koordinat.

Terakhir, gambar yang telah tersusun disimpan ke dalam folder **generated_images** menggunakan `fig.savefig()`, dengan nama file yang mencantumkan nomor *epoch* (misalnya `gan_generated_image_epoch_5.png`). Setelah itu, `plt.close()` digunakan untuk menutup *figure* agar tidak memakan memori.

Secara keseluruhan, fungsi ini berfungsi untuk **mendokumentasikan kemajuan visual model GAN**, sehingga peneliti atau pengembang dapat melihat bagaimana kualitas gambar yang dihasilkan meningkat seiring waktu selama proses pelatihan.

---

7. Training GAN

```
# Load the dataset
(X_train, _), (_, _) = cifar10.load_data()

# Rescale images from -1 to 1
X_train = (X_train.astype(np.float32) - 127.5) / 127.5

# Reshape data for the discriminator
```

```
X_train = X_train.reshape(X_train.shape[0], img_width,
img_height, channels)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-
python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 14s 0us/step
```

```
epochs = 4000
batch_size = 128
save_interval = 200

# Calculate the number of batches per epoch
batch_count = X_train.shape[0] // batch_size

# Adversarial ground truths
real = np.ones((batch_size, 1))
fake = np.zeros((batch_size, 1))
```

```
for epoch in range(epochs):
    # Train Discriminator
    # Select a random batch of real images
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    real_imgs = X_train[idx]

    # Generate a batch of fake images
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    fake_imgs = generator.predict(noise)

    # Train the discriminator
    d_loss_real = discriminator.train_on_batch(real_imgs,
real)
    d_loss_fake = discriminator.train_on_batch(fake_imgs,
fake)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    # Train Generator
    # Generate a batch of fake images
    noise = np.random.normal(0, 1, (batch_size, latent_dim))

    # Train the generator
    g_loss = GAN.train_on_batch(noise, real)

    # Print the progress
```

```
    if epoch % save_interval == 0:
        print(f"Epoch {epoch}/{epochs} [D loss:
{d_loss[0]:.4f}, acc.: {100*d_loss[1]:.2f}%] [G loss:
{g_loss:.4f}]")
        save_generated_images(epoch, generator)
```

A. Penjelasan Kode

Pertama, dataset CIFAR-10 dimuat menggunakan `cifar10.load_data()`, namun hanya bagian **X_train** (data gambar) yang digunakan karena labelnya tidak diperlukan dalam pelatihan GAN. Dataset CIFAR-10 berisi 60.000 gambar berwarna berukuran 32×32 piksel. Setelah itu, dilakukan proses **rescaling**: `X_train = (X_train.astype(np.float32) - 127.5) / 127.5`, yang mengubah nilai piksel dari rentang 0–255 menjadi -1 hingga 1, sesuai dengan keluaran dari *generator* yang menggunakan fungsi aktivasi **tanh**. Data kemudian di-*reshape* menjadi format `(jumlah_data, lebar, tinggi, channel)` agar cocok dengan input *discriminator*.

Selanjutnya, ditentukan parameter pelatihan:

- `epochs = 4000` → jumlah iterasi pelatihan,
- `batch_size = 128` → jumlah gambar yang diproses dalam satu kali pelatihan mini-batch,
- `save_interval = 200` → interval setiap berapa epoch hasil gambar disimpan.

Jumlah batch per epoch dihitung dengan `batch_count = X_train.shape[0] // batch_size`. Kemudian dibuat dua label target: `real` (semua bernilai 1) untuk gambar asli, dan `fake` (semua bernilai 0) untuk gambar palsu.

Di dalam loop `for epoch in range(epochs):`, proses pelatihan dilakukan dalam dua tahap:

1. **Melatih *Discriminator***
    - Dipilih secara acak sejumlah gambar asli dari dataset dengan `np.random.randint`.
    - Dibuat sekumpulan gambar palsu menggunakan *generator* dengan input berupa *noise* acak dari distribusi normal.
    - *Discriminator* dilatih dua kali menggunakan `train_on_batch:` pertama dengan gambar asli (`real_imgs, real`), dan kedua dengan gambar palsu (`fake_imgs, fake`).
    - Hasil dua pelatihan ini dirata-rata menjadi `d_loss`, yang menunjukkan seberapa baik *discriminator* mampu membedakan gambar asli dan palsu.
2. **Melatih *Generator***
    - Dihasilkan *noise* baru sebagai input.
    - *Generator* dilatih melalui model gabungan `GAN.train_on_batch(noise, real)`. Pada tahap ini, *discriminator* tidak diperbarui (karena telah diset `trainable=False`), sehingga *generator* berusaha menipu *discriminator* dengan menghasilkan gambar yang dianggap “real”.

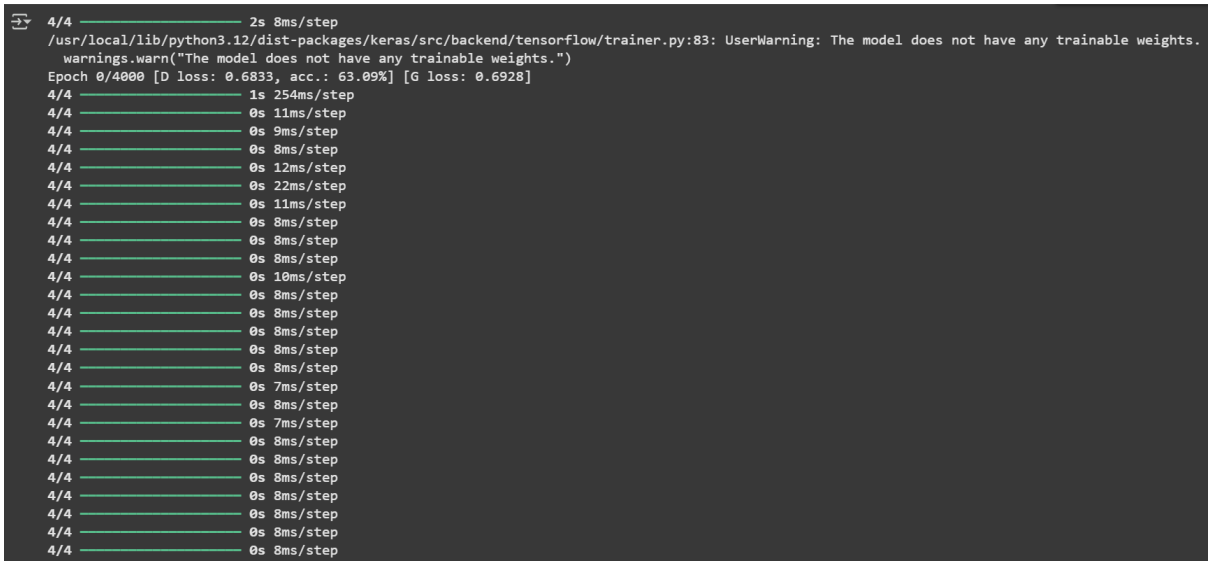
Setiap beberapa interval (misalnya setiap 200 epoch), hasil pelatihan ditampilkan di terminal dalam format:

```
Epoch 200/4000 [D loss: 0.5021, acc.: 81.25%] [G loss: 1.0432]
```

Informasi tersebut menunjukkan performa *discriminator* (loss dan akurasi) serta *generator* (loss). Selain itu, fungsi `save_generated_images(epoch, generator)` dipanggil untuk **menyimpan gambar hasil generasi** pada tahap tersebut ke folder `generated_images`, sehingga perkembangan kualitas gambar bisa dipantau dari waktu ke waktu.

Secara keseluruhan, kode ini mengimplementasikan **algoritma pelatihan adversarial**, di mana *generator* belajar untuk membuat gambar palsu yang semakin menyerupai data nyata, sementara *discriminator* terus beradaptasi untuk membedakannya. Kompetisi antara keduanya secara bertahap menghasilkan model *generator* yang mampu menciptakan gambar baru yang sangat realistis.

B. HASIL



C. Penjelasan HASIL

1. Baris utama hasil pelatihan

```
Epoch 0/4000 [D loss: 0.6833, acc.: 63.09%] [G loss: 0.6928]
```

- **Epoch 0/4000**
  Menunjukkan bahwa ini adalah *epoch* pertama dari total 4000 *epoch* yang direncanakan. Setiap *epoch* mewakili satu siklus lengkap pelatihan, di mana seluruh dataset telah digunakan untuk memperbarui bobot jaringan.

- **D loss: 0.6833**
  Menunjukkan *loss function* (kesalahan) dari model *Discriminator* setelah menerima data gambar asli dan gambar palsu. Nilai ini mendekati **0.69**, yang umum terjadi di awal pelatihan karena *discriminator* masih menebak secara acak (mendekati 50:50 antara gambar asli dan palsu).
- **acc.: 63.09%**
  Adalah **akurasi *discriminator***, menunjukkan bahwa pada tahap awal ini model sudah bisa mengenali sekitar **63%** gambar dengan benar sebagai "asli" atau "palsu". Nilai ini masih fluktuatif dan biasanya meningkat seiring pelatihan berjalan.
- **G loss: 0.6928**
  Adalah *loss function* dari model *Generator*. Nilai ini juga mendekati **0.69**, artinya *generator* pada awalnya masih kesulitan menipu *discriminator*. Seiring waktu, nilai *G loss* diharapkan turun, menandakan bahwa *generator* mulai mampu membuat gambar yang lebih realistis.

Secara keseluruhan, nilai-nilai ini menunjukkan bahwa pelatihan baru dimulai dan kedua model masih belajar "beradaptasi" satu sama lain.

2. Baris-baris berikutnya (misalnya: `4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step`)

Baris-baris ini berasal dari **proses batch pelatihan yang dilakukan oleh Keras**.

- `4/4` artinya ada **4 batch** yang diproses dalam satu *epoch* (karena dataset besar dibagi menjadi potongan kecil atau mini-batch dengan ukuran tertentu).
- Waktu seperti `1s 254ms/step` atau `0s 11ms/step` menunjukkan **lama waktu yang dibutuhkan untuk melatih tiap batch**.
- Lambang **progress bar (━━━━)** menunjukkan proses eksekusi berjalan (kadang muncul saat *predict()* atau *train_on_batch()* dijalankan oleh *generator* dan *discriminator*).

Jadi sederhananya, baris-baris tersebut adalah **indikator aktivitas training**, bukan hasil evaluasi performa.

---

8. Making a GIF

```
import imageio
import glob

anim_file = 'cifar10_gan.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
    filenames =
glob.glob('generated_images/gan_generated_image_epoch_*.png')
    filenames = sorted(filenames, key=lambda x:
int(os.path.basename(x).split('_')[4].split('.')[0]))
```

```
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)

print(f"GIF saved as {anim_file}")
```

A. Penjelasan Kode

Kode tersebut digunakan untuk membuat animasi GIF yang menampilkan perkembangan hasil gambar yang dihasilkan oleh model GAN selama proses pelatihan. Pertama, library `imageio` dan `glob` diimpor. Library `imageio` digunakan untuk membaca dan menulis file gambar atau animasi seperti GIF, sementara `glob` digunakan untuk mencari semua file gambar yang sesuai dengan pola nama tertentu di dalam folder. Nama file output ditentukan melalui variabel `anim_file = 'cifar10_gan.gif'`, yang menandakan bahwa hasil akhir dari proses ini akan disimpan dengan nama tersebut.

Selanjutnya, objek `writer` dari `imageio.get_writer()` dibuka dalam mode `'I'`, yang memungkinkan penulisan beberapa gambar secara berurutan ke dalam satu file GIF. Dalam blok ini, baris `filenames = glob.glob('generated_images/gan_generated_image_epoch_*.png')` digunakan untuk mengambil semua file gambar hasil pelatihan GAN yang disimpan di folder `generated_images` dengan pola nama seperti `gan_generated_image_epoch_0.png`, `gan_generated_image_epoch_200.png`, dan seterusnya. Agar urutan animasi sesuai dengan urutan pelatihan, daftar file tersebut diurutkan berdasarkan nomor epoch menggunakan fungsi `sorted()` dan ekspresi lambda yang mengekstrak angka epoch dari nama file.

Setelah itu, program membaca setiap gambar menggunakan `imageio.imread(filename)` dan menambahkannya ke animasi menggunakan `writer.append_data(image)`. Proses ini diulang untuk seluruh file gambar, sehingga menghasilkan rangkaian visual yang menunjukkan bagaimana kualitas gambar buatan generator meningkat seiring bertambahnya jumlah epoch. Setelah semua gambar ditambahkan, blok `with` secara otomatis menutup writer, dan pesan “GIF saved as cifar10_gan.gif” dicetak ke terminal sebagai tanda bahwa proses pembuatan animasi telah selesai.

Secara keseluruhan, fungsi utama kode ini adalah menggabungkan seluruh gambar hasil pelatihan model GAN ke dalam satu file animasi GIF. Dengan begitu, pengguna dapat melihat secara visual bagaimana generator belajar dari waktu ke waktu — mulai dari menghasilkan gambar yang masih acak hingga akhirnya mampu menghasilkan gambar yang lebih realistis dan menyerupai data asli.

B. Penjelasan HASIL

Hasil dari kode tersebut adalah sebuah file animasi berformat **GIF** dengan nama **`cifar10_gan.gif`**. GIF ini berisi rangkaian gambar yang dihasilkan oleh model **GAN (Generative Adversarial Network)** pada setiap interval epoch selama proses pelatihan.

Setiap gambar diambil dari folder **`generated_images`**, yang sebelumnya diisi oleh fungsi `save_generated_images()` pada tahap pelatihan.

Pada bagian awal animasi (epoch rendah seperti 0 atau 200), gambar-gambar yang muncul biasanya masih terlihat **acak, kabur, dan tidak memiliki bentuk yang jelas**. Hal ini wajar karena pada tahap awal, generator belum mampu memahami struktur data dari dataset CIFAR-10. Namun, seiring meningkatnya jumlah epoch, gambar-gambar dalam GIF mulai menunjukkan **pola, warna, dan bentuk yang semakin mirip dengan gambar asli CIFAR-10** (misalnya hewan atau objek tertentu).

Perubahan ini menggambarkan proses **pembelajaran progresif generator**. Di setiap epoch, generator memperbaiki kemampuannya untuk "menipu" discriminator dengan menghasilkan gambar yang terlihat lebih realistis. Sementara itu, discriminator terus berusaha meningkatkan kemampuannya dalam membedakan gambar asli dan palsu. Persaingan ini secara bertahap membuat generator menghasilkan gambar yang lebih tajam, memiliki komposisi warna yang lebih alami, dan struktur objek yang semakin menyerupai gambar nyata.

Dengan demikian, **hasil akhir dari kode ini bukan hanya file GIF semata**, tetapi juga merupakan **representasi visual dari perkembangan kemampuan model GAN selama pelatihan**. GIF tersebut berfungsi sebagai alat bantu evaluasi intuitif untuk melihat bagaimana kualitas gambar hasil generasi meningkat dari waktu ke waktu, tanpa harus menganalisis angka loss atau akurasi secara mendetail.

SELESAI.