

Nama : Khalishah

NIM : 1103213045

Task 5 Computer Vision Hugging Face Course (Chapter 5-8)

1. CycleGAN (Chapter 5)

CycleGAN adalah model jaringan adversarial generatif (GAN) yang memungkinkan translasi gambar-ke-gambar tanpa pasangan data terlabel. Model ini mempelajari pemetaan antara dua domain gambar dengan menggunakan loss konsistensi siklus untuk memastikan bahwa gambar yang ditranslasikan dapat dikembalikan ke bentuk aslinya, sehingga menjaga karakteristik penting dari gambar tersebut[1]. Pada pelatihan ini akan melatih model yang dapat mengonversi kuda menjadi zebra dan sebaliknya.

Mulai dengan mengimpor beberapa pustaka yang diperlukan:

```
import torch
from torch import nn
from tqdm import tqdm
from torchvision import transforms
from torchvision.utils import make_grid
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import glob
import random
import os
from torch.utils.data import Dataset
from PIL import Image
import requests
import zipfile
import io
import torch.nn.functional as F
import torchvision
import datasets
```

Kode di atas mengimpor pustaka Python untuk membangun dan melatih model machine learning menggunakan PyTorch. Berikut penjelasannya:

1. PyTorch dan Modul Utama:

- torch, nn, dan F: Digunakan untuk membangun dan mengelola jaringan neural.
- torchvision dan transforms: Untuk memproses data gambar, seperti transformasi, dan menyiapkan dataset gambar.
- make_grid: Membuat grid gambar untuk visualisasi.
- DataLoader: Untuk memuat data dalam batch selama pelatihan.

2. Visualisasi dan Alat:

- matplotlib.pyplot: Untuk menampilkan data atau hasil pelatihan.

- tqdm: Untuk menampilkan progress bar saat iterasi.

3. File dan Dataset:

- glob, os, PIL.Image: Untuk mengelola file gambar dan dataset.
- Dataset: Base class untuk membuat dataset kustom.
- datasets: Library tambahan untuk dataset standar.

4. Utility:

- random: Digunakan untuk memilih data secara acak.
- requests, zipfile, io: Untuk mengunduh dan mengekstrak data dari sumber online.

Mengunduh dataset zebra2horses dari repository Hugging Face:

```
dataset = datasets.load_dataset("johko/horse2zebra", split="train")
↳ /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
  The secret 'HF_TOKEN' does not exist in your Colab secrets.
  To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
  You will be able to reuse this secret in all of your notebooks.
  Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md: 100% [██████████] 427/427 [00:00<00:00, 31.9kB/s]
train-00000-of-00001.parquet: 100% [██████████] 17.7M/17.7M [00:00<00:00, 35.3MB/s]
Generating train split: 100% [██████████] 400/400 [00:00<00:00, 3799.22 examples/s]
```

Kode diatas berfungsi untuk memuat dataset *horse2zebra* menggunakan pustaka  Hugging Face Datasets. Berikut adalah penjelasannya:

1. **datasets.load_dataset()**: Fungsi ini digunakan untuk memuat dataset dari Hugging Face Hub.
2. **"johko/horse2zebra"**: Ini adalah nama dataset yang dimuat. Dataset ini digunakan untuk tugas *image-to-image translation* (seperti yang dilakukan oleh CycleGAN), di mana gambar kuda diterjemahkan menjadi zebra, dan sebaliknya.
3. **split="train"**: Parameter ini menentukan bagian dataset yang akan dimuat. Dalam hal ini, bagian pelatihan (train) dipilih.

Memeriksa informasi tentang fitur *label* dalam dataset yang telah dimuat:

```
dataset.features["label"]
↳ ClassLabel(names=['testA', 'testB', 'trainA', 'trainB'], id=None)
```

Penjelasan:

1. **dataset.features**: Ini adalah atribut yang memberikan deskripsi tentang fitur-fitur (*columns*) dalam dataset.
2. **["label"]**: Bagian ini mengambil fitur tertentu bernama *label*, yang sering digunakan untuk memberikan informasi tentang kategori atau kelas data.
3. Dataset ini mengandung empat label yang berbeda:

ClassLabel	IntLabel
testA	0
testB	1
trainA	2
trainB	3

Selanjutnya, kita membuat kelas dataset kustom untuk mengelola data. Kelas ini memisahkan data *train* menjadi dua daftar: sampel A dan sampel B, sesuai dengan label domainnya. Saat item diambil dari dataset, kelas ini mengembalikan satu sampel dari masing-masing domain (A dan B) dengan transformasi yang telah diterapkan, yang akan didefinisikan kemudian.

Definisikan kelas:

```
# Terinspirasi dari https://github.com/aitorzip/PyTorch-
CycleGAN/blob/master/datasets.py
class ImageDataset(Dataset):
    def __init__(self, dataset: datasets.Dataset, transform=None,
mode='train'):
        self.transform = transform
        # Memfilter dataset untuk mendapatkan gambar dengan label 2 (A)
        # dan label 3 (B)
        self.files_A = dataset.filter(lambda x: x['label'] ==
2) ["image"]
        self.files_B = dataset.filter(lambda x: x['label'] ==
3) ["image"]

        # Menyusun ulang jika jumlah gambar A lebih banyak dari B
        if len(self.files_A) > len(self.files_B):
            self.files_A, self.files_B = self.files_B, self.files_A

        # Membuat permutasi acak baru
        self.new_perm()

        # Memastikan bahwa gambar A ada
        assert len(self.files_A) > 0, "Pastikan Anda sudah mengunduh
gambar horse2zebra!"

    def new_perm(self):
        # Membuat permutasi acak dari gambar B
        self.randperm =
torch.randperm(len(self.files_B))[:len(self.files_A)]

    def __getitem__(self, index):
```

```

        # Mengambil gambar A dan B dengan transformasi yang telah
ditentukan
        item_A = self.transform(self.files_A[index %
len(self.files_A)])
        item_B = self.transform(self.files_B[self.randperm[index]])

        # Menambahkan saluran warna (channels) jika gambar A atau B
tidak memiliki 3 saluran
        if item_A.shape[0] != 3:
            item_A = item_A.repeat(3, 1, 1)
        if item_B.shape[0] != 3:
            item_B = item_B.repeat(3, 1, 1)

        # Jika indeks mencapai akhir, buat permutasi baru
        if index == len(self) - 1:
            self.new_perm()

        # Normalisasi gambar dari range [0, 1] ke [-1, 1]
        return (item_A - 0.5) * 2, (item_B - 0.5) * 2

    def __len__(self):
        # Mengembalikan panjang dataset yang lebih kecil antara A dan B
        return min(len(self.files_A), len(self.files_B))

```

Kode di atas mendefinisikan kelas `ImageDataset`, yang merupakan kelas kustom untuk mengelola dataset gambar. Kelas ini memiliki beberapa fungsi utama:

1. Inisialisasi (`__init__`):

- Memfilter dataset untuk mendapatkan gambar dengan label tertentu (label 2 untuk sampel A dan label 3 untuk sampel B).
- Menyusun ulang jika jumlah gambar A lebih banyak dari B, dan kemudian menciptakan permutasi acak untuk mencocokkan gambar A dan B.
- Memastikan bahwa gambar A tersedia.

2. `new_perm()`:

- Membuat permutasi acak baru dari gambar B untuk mencocokkan panjang gambar A.

3. `__getitem__()`:

- Mengambil gambar A dan B berdasarkan indeks yang diberikan.
- Menerapkan transformasi pada gambar dan memastikan bahwa gambar memiliki tiga saluran warna.
- Melakukan normalisasi pada gambar dari rentang [0, 1] ke [-1, 1].
- Jika indeks mencapai akhir dataset, permutasi baru dibuat.

4. `__len__()`:

- Mengembalikan panjang dataset yang lebih kecil antara gambar A dan B.

Mempersiapkan dataset gambar:

```
import torchvision

load_shape = 286
target_shape = 256

transform = transforms.Compose([
    transforms.Resize(load_shape),
    transforms.RandomCrop(target_shape),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])

dataset = ImageDataset(dataset, transform=transform)

Filter: 100% [██████████] 400/400 [00:00<00:00, 743.93 examples/s]
Filter: 100% [██████████] 400/400 [00:00<00:00, 864.73 examples/s]
```

Kode di atas melakukan beberapa langkah untuk mempersiapkan dataset gambar untuk pelatihan CycleGAN:

1. Definisi Transformasi:

- **transforms.Resize(load_shape)**: Mengubah ukuran gambar ke dimensi load_shape (286x286).
- **transforms.RandomCrop(target_shape)**: Memotong gambar secara acak menjadi ukuran target_shape (256x256) setelah gambar diubah ukurannya.
- **transforms.RandomHorizontalFlip()**: Melakukan pembalikan gambar secara horizontal secara acak untuk meningkatkan keragaman data.
- **transforms.ToTensor()**: Mengonversi gambar menjadi tensor PyTorch.

2. Mempersiapkan Dataset:

- Dataset **ImageDataset** diinisialisasi dengan dataset yang telah diunduh sebelumnya dan transformasi yang telah didefinisikan.

Kelas **ImageDataset** yang sudah didefinisikan sebelumnya akan memastikan bahwa gambar dipasangkan secara acak antara dua domain (A dan B), serta transformasi akan diterapkan pada gambar saat dimuat.

Menampilkan gambar-gambar dalam bentuk grid:

```
def show_tensor_images(image_tensor, num_images=25, size=(1, 28, 28)):
    """
    Fungsi untuk memvisualisasikan gambar: Diberikan tensor gambar,
    jumlah gambar, dan
```

```

ukuran per gambar, fungsi ini akan menggambar dan menampilkan
gambar dalam grid yang seragam.

# Menormalisasi gambar dari rentang [-1, 1] ke rentang [0, 1]
image_tensor = (image_tensor + 1) / 2
image_shifted = image_tensor

# Meratakan tensor gambar dan memindahkannya ke CPU untuk diproses
image_unflat = image_shifted.detach().cpu().view(-1, *size)

# Membuat grid gambar dari tensor gambar yang sudah diratakan
image_grid = make_grid(image_unflat[:num_images], nrow=5)

# Menampilkan gambar grid
plt.imshow(image_grid.permute(1, 2, 0).squeeze())
plt.show()

```

Fungsi `show_tensor_images` bertujuan untuk menampilkan gambar-gambar dalam bentuk grid, yang diambil dari tensor gambar. Berikut adalah penjelasan langkah demi langkah dari fungsi ini:

1. Menormalkan Gambar:

- Gambar yang diberikan dalam tensor memiliki rentang nilai antara [-1, 1], sedangkan untuk menampilkan gambar dengan benar, kita perlu mengubahnya ke rentang [0, 1]. Proses ini dilakukan dengan rumus `(image_tensor + 1) / 2`.

2. Menggeser Tensor Gambar:

- Setelah menormalkan, tensor gambar disalin ke dalam variabel `image_shifted` untuk pengolahan lebih lanjut.

3. Meratakan Tensor Gambar:

- `image_tensor` diratakan menjadi bentuk satu dimensi dengan menggunakan `.view(-1, *size)`, dan dipindahkan ke CPU menggunakan `.detach().cpu()`, karena operasi tensor tersebut kemungkinan berada di GPU.

4. Membuat Grid Gambar:

- Fungsi `make_grid` digunakan untuk menyusun gambar-gambar dalam grid. `nrow=5` memastikan bahwa gambar akan disusun dalam 5 kolom.

5. Menampilkan Grid Gambar:

- Gambar grid yang dihasilkan dipermutasi dengan `permute(1, 2, 0)` agar sesuai dengan urutan warna RGB dan kemudian ditampilkan dengan `plt.imshow`.

Fungsi ini sangat berguna untuk memvisualisasikan sampel gambar selama pelatihan model, terutama ketika menggunakan batch data yang besar.

Residual block:

```
class ResidualBlock(nn.Module):
    """
        Mendefinisikan sebuah blok residual untuk jaringan neural.

        Blok ini terdiri dari dua lapisan konvolusional dengan normalisasi
        instansi,
        diikuti oleh fungsi aktivasi. Input ditambahkan ke output dari
        lapisan-lapisan ini, membentuk output akhir dari blok.
    """

    def __init__(self, input_channels):
        """
            Menginisialisasi ResidualBlock.

            Parameter:
            input_channels (int): Jumlah saluran dalam tensor input.
        """
        super(ResidualBlock, self).__init__()

        # Lapisan konvolusional pertama dengan padding reflektif
        self.conv1 = nn.Conv2d(input_channels, input_channels,
        kernel_size=3, padding=1, padding_mode="reflect")

        # Lapisan konvolusional kedua dengan padding reflektif
        self.conv2 = nn.Conv2d(input_channels, input_channels,
        kernel_size=3, padding=1, padding_mode="reflect")

        # Lapisan normalisasi instansi
        self.instancenorm = nn.InstanceNorm2d(input_channels)

        # Fungsi aktivasi (ReLU)
        self.activation = nn.ReLU()

    def forward(self, x):
        """
            Mendefinisikan proses maju dari ResidualBlock.

            Parameter:
            x (Tensor): Tensor input untuk blok residual.

            Mengembalikan:
            Tensor: Tensor output setelah menerapkan operasi pada blok
            residual.
        """

        # Menyimpan input asli untuk digunakan dalam koneksi skip
        original_tensor_image = x.clone()
```

```

# Operasi konvolusional pertama
x = self.conv1(x)
x = self.instancenorm(x)
x = self.activation(x)

# Operasi konvolusional kedua
x = self.conv2(x)
x = self.instancenorm(x)

# Menambahkan tensor asli ke output (koneksi skip)
return original_tensor_image + x

```

Kode ini mendefinisikan sebuah kelas ResidualBlock, yang merupakan komponen umum dalam jaringan saraf convolutional yang menggunakan *skip connection*. Berikut adalah penjelasan rinci setiap bagianya:

Kelas ResidualBlock(nn.Module):

- Kelas ini adalah blok residual yang digunakan dalam jaringan saraf untuk menghindari masalah gradien yang menghilang (vanishing gradients) dan memungkinkan model belajar lebih dalam dengan lebih baik.

Metode __init__(self, input_channels):

- **Tujuan:** Menginisialisasi komponen dari blok residual.
- **Parameter:**
 - **input_channels:** Jumlah saluran (channels) pada input tensor, digunakan untuk menentukan jumlah saluran pada lapisan konvolusional.
- **Komponen:**
 - **self.conv1:** Lapisan konvolusional pertama dengan kernel ukuran 3x3, padding reflektif, dan jumlah saluran input dan output yang sama. Padding reflektif menjaga batas gambar agar tetap halus.
 - **self.conv2:** Lapisan konvolusional kedua, juga dengan kernel 3x3 dan padding reflektif, serta saluran yang sama dengan lapisan pertama.
 - **self.instancenorm:** Lapisan normalisasi instansi (instance normalization) untuk menormalkan fitur berdasarkan setiap contoh.
 - **self.activation:** Fungsi aktivasi ReLU untuk meningkatkan non-linearitas dalam jaringan.

Metode forward(self, x):

- **Tujuan:** Menjalankan operasi maju pada input x melalui blok residual dan menghasilkan output.
- **Langkah-langkah:**
 1. **Simpan input asli:** original_tensor_image = x.clone() menyimpan salinan dari tensor input untuk digunakan dalam koneksi skip.
 2. **Proses melalui konvolusi pertama:** Input x diproses oleh self.conv1, kemudian dinormalisasi dengan self.instancenorm dan diterapkan fungsi aktivasi ReLU.

- 3. **Proses melalui konvolusi kedua:** Output dari langkah pertama diproses kembali dengan konvolusi kedua, dinormalisasi, dan tidak ada aktivasi di sini.
- 4. **Koneksi skip:** Input asli (original_tensor_image) ditambahkan ke output dari lapisan konvolusi kedua, yang merupakan inti dari blok residual.
- **Output:** Tensor akhir hasil penambahan input asli dan output dari konvolusi kedua, yang memungkinkan model untuk belajar lebih dalam tanpa kehilangan informasi penting yang terkandung dalam input awal.

Encoder and decoder blocks:

```

class EncoderBlock(nn.Module):
    """
    Kelas EncoderBlock:
    Kelas yang digunakan untuk membuat blok encoder dalam jaringan
    neural, yang terdiri dari lapisan konvolusional,
    lapisan normalisasi instansi opsional, dan fungsi aktivasi.

    Atribut:
        conv1: Lapisan Conv2d untuk mengurangi dimensi spasial dan
        meningkatkan saluran.
        instancenorm: Lapisan InstanceNorm2d untuk normalisasi
        (opsional).
        activation: Fungsi aktivasi (ReLU atau LeakyReLU).
        use_bn: Boolean, apakah menggunakan normalisasi batch.
    """

    def __init__(self, input_channels, use_bn=True, kernel_size=3,
                 activation='relu'):
        super(EncoderBlock, self).__init__()
        # Membuat lapisan konvolusional dengan stride 2 untuk
        # downsampling
        self.conv1 = nn.Conv2d(input_channels, input_channels * 2,
                             kernel_size=kernel_size, padding=1, stride=2, padding_mode='reflect')
        # Memilih fungsi aktivasi berdasarkan argumen input
        if activation == 'relu':
            self.activation = nn.ReLU()
        else:
            self.activation = nn.LeakyReLU(0.2)
        # Menambahkan normalisasi instansi jika use_bn adalah True
        if use_bn:
            self.instancenorm = nn.InstanceNorm2d(input_channels * 2)

        self.use_bn = use_bn

    def forward(self, x):
        """
        Fungsi untuk proses maju dari EncoderBlock:
        """

```

```

        Menerapkan konvolusi, normalisasi opsional, dan aktivasi pada
tensor input.

    '''

    x = self.conv1(x)
    if self.use_bn:
        x = self.instancenorm(x)
    x = self.activation(x)
    return x

class DecoderBlock(nn.Module):
    '''

    Kelas DecoderBlock:
    Kelas yang digunakan untuk membuat blok decoder dalam jaringan
    neural, yang terdiri dari lapisan konvolusional transpos
    (untuk upsampling), lapisan normalisasi instansi opsional, dan
    fungsi aktivasi ReLU.

    Atribut:
        conv1: Lapisan ConvTranspose2d untuk upsampling dan mengurangi
    saluran.
        instancenorm: Lapisan InstanceNorm2d untuk normalisasi
    (opsional).
        use_bn: Boolean, apakah menggunakan normalisasi batch.
        activation: Fungsi aktivasi ReLU.
    '''

    def __init__(self, input_channels, use_bn=True):
        super(DecoderBlock, self).__init__()
        # Lapisan konvolusional transpos untuk upsampling
        self.conv1 = nn.ConvTranspose2d(input_channels, input_channels
// 2, kernel_size=3, stride=2, padding=1, output_padding=1)
        # Menambahkan normalisasi instansi jika use_bn adalah True
        if use_bn:
            self.instancenorm = nn.InstanceNorm2d(input_channels // 2)
        self.use_bn = use_bn
        # Fungsi aktivasi ReLU
        self.activation = nn.ReLU()

    def forward(self, x):
        '''
        Fungsi untuk proses maju dari DecoderBlock:
        Menerapkan konvolusi transpos, normalisasi opsional, dan
        aktivasi pada tensor input.
        '''

        x = self.conv1(x)
        if self.use_bn:
            x = self.instancenorm(x)
        x = self.activation(x)
        return x

```

Kode ini mendefinisikan dua kelas yaitu EncoderBlock dan DecoderBlock, yang umumnya digunakan dalam arsitektur jaringan saraf konvolusional seperti autoencoders atau GAN (Generative Adversarial Networks). Berikut adalah penjelasan masing-masing kelas dan komponennya:

1. Kelas EncoderBlock

Tujuan:

- Blok ini digunakan untuk proses enkoding dalam jaringan saraf, yang melakukan downsampling pada gambar (mengurangi ukuran spasial) dan meningkatkan jumlah saluran fitur.

Komponen:

- **self.conv1:**
 - Lapisan konvolusional 2D yang digunakan untuk mengubah dimensi spasial gambar input dengan menggunakan stride=2 untuk downsampling.
 - Jumlah saluran outputnya adalah dua kali jumlah saluran input (`input_channels * 2`), yang sering digunakan untuk meningkatkan kompleksitas fitur saat gambar diperkecil.
 - Padding reflektif digunakan untuk menjaga tepi gambar agar tetap halus.
- **self.activation:**
 - Fungsi aktivasi yang digunakan setelah konvolusi. Berdasarkan parameter `activation`, fungsi ini bisa berupa ReLU (`ReLU()`) atau LeakyReLU (`LeakyReLU(0.2)`).
- **self.instancenorm:**
 - Lapisan normalisasi instansi yang diterapkan jika `use_bn` bernilai True. Normalisasi ini digunakan untuk menormalkan setiap channel pada setiap contoh input secara terpisah untuk stabilitas dan kecepatan konvergensi selama pelatihan.
- **self.use_bn:**
 - Boolean yang menentukan apakah normalisasi instansi digunakan atau tidak.

Metode forward(self, x):

- Fungsi ini memproses input `x` melalui lapisan-lapisan yang telah didefinisikan:
 1. Input diproses oleh `conv1` (konvolusi).
 2. Jika `use_bn` adalah True, normalisasi instansi diterapkan.
 3. Fungsi aktivasi (ReLU atau LeakyReLU) diterapkan setelah normalisasi.
 4. Hasil akhirnya dikembalikan sebagai output.

2. Kelas DecoderBlock

Tujuan:

- Blok ini digunakan untuk proses decoding, yaitu untuk meningkatkan dimensi spasial gambar (upsampling). Biasanya digunakan dalam model seperti autoencoders atau GANs untuk membangun output yang lebih besar (misalnya, gambar yang dihasilkan).

Komponen:

- **self.conv1:**
 - Lapisan konvolusional transpos (dekonvolusi) digunakan untuk upsampling.
 - Ini mengubah dimensi spasial gambar dengan meningkatkan ukuran gambar, mengurangi jumlah saluran menjadi setengahnya (`input_channels // 2`), dan menghasilkan gambar dengan dimensi yang lebih besar.
 - Padding dan output padding ditentukan untuk memastikan gambar hasil upsampling memiliki dimensi yang tepat.
- **selfinstancenorm:**
 - Lapisan normalisasi instansi yang diterapkan jika `use_bn` bernilai True, berfungsi untuk menormalkan saluran gambar hasil dekonvolusi untuk stabilitas pelatihan.
- **self.activation:**
 - Fungsi aktivasi `ReLU()` digunakan untuk memberikan non-linearitas pada output setelah dekonvolusi.
- **self.use_bn:**
 - Boolean yang menentukan apakah normalisasi instansi diterapkan atau tidak.

Metode forward(self, x):

- Fungsi ini memproses input `x` melalui langkah-langkah berikut:
 1. Input diproses melalui konvolusi transpos (dekonvolusi) menggunakan `conv1`.
 2. Jika `use_bn` bernilai True, normalisasi instansi diterapkan.
 3. Fungsi aktivasi `ReLU()` diterapkan pada hasil dekonvolusi.
 4. Output akhir dikembalikan.

Feature map block:

```
class FeatureMapBlock(nn.Module):
    """
    Kelas FeatureMapBlock:
    Kelas untuk blok konvolusional dasar yang digunakan untuk
    memodifikasi jumlah peta fitur
    tanpa mengubah dimensi spasial dari input.

    Atribut:
        conv: Lapisan Conv2d dengan ukuran kernel yang lebih besar
        untuk menangkap fitur tanpa mengurangi ukuran spasial.
    """

```

```

def __init__(self, input_channels, output_channels):
    super(FeatureMapBlock, self).__init__()
    # Lapisan konvolusional dengan ukuran kernel yang lebih besar
    untuk pemetaan fitur
    self.conv = nn.Conv2d(input_channels, output_channels,
kernel_size=7, padding=3, padding_mode='reflect')

def forward(self, x):
    """
    Fungsi untuk proses maju dari FeatureMapBlock:
    Menerapkan konvolusi pada tensor input untuk mengubah jumlah
peta fitur.
    """
    x = self.conv(x)
    return x

```

Kelas FeatureMapBlock adalah sebuah blok konvolusional yang digunakan untuk memodifikasi jumlah peta fitur (feature maps) dalam tensor gambar tanpa mengubah dimensi spasial dari input (lebar dan tinggi gambar tetap sama). Kelas ini biasanya digunakan untuk mengekstrak fitur dari gambar dengan cara yang lebih halus, sambil mempertahankan ukuran gambar.

Komponen:

- **self.conv:**
 - Ini adalah lapisan konvolusional (nn.Conv2d) yang diterapkan pada input untuk mengubah jumlah peta fitur.
 - Kernel yang digunakan memiliki ukuran 7x7, yang lebih besar dibandingkan dengan ukuran kernel yang biasa digunakan seperti 3x3 atau 5x5. Ukuran kernel yang lebih besar ini memungkinkan untuk menangkap informasi kontekstual yang lebih luas dari gambar.
 - **Padding** sebesar 3 diterapkan untuk memastikan bahwa dimensi spasial gambar tetap tidak berubah (output gambar memiliki lebar dan tinggi yang sama dengan input). Padding reflektif (padding_mode='reflect') digunakan untuk memastikan bahwa nilai piksel di tepi gambar diperlakukan dengan lebih halus selama proses konvolusi.
 - **input_channels** adalah jumlah saluran (channels) pada gambar input, sedangkan **output_channels** adalah jumlah saluran output yang dihasilkan setelah konvolusi, yang menentukan jumlah peta fitur yang akan dipelajari.
- **self.forward(x):**
 - Fungsi ini mendefinisikan proses maju dalam blok konvolusional.
 - Input x diterima oleh fungsi ini, kemudian dilalui melalui lapisan konvolusional self.conv.

- Hasil dari konvolusi (tensor fitur yang telah diproses) kemudian dikembalikan sebagai output.

Fungsi Utama:

- Memodifikasi jumlah peta fitur: Kelas ini mengubah jumlah saluran dalam tensor gambar dari `input_channels` ke `output_channels`, yang memungkinkan jaringan untuk mempelajari representasi fitur yang lebih kompleks tanpa mengubah dimensi spasial gambar.
- Menjaga dimensi spasial tetap: Dengan menggunakan padding yang sesuai (`padding=3` dengan ukuran kernel 7×7), dimensi spasial gambar tetap sama setelah konvolusi.

CycleGAN generator:

```
class Generator(nn.Module):

    def __init__(self, input_channels, output_channels,
hidden_channels=64):
        super(Generator, self).__init__()

        self.upfeature = FeatureMapBlock(input_channels,
hidden_channels)
        self.encoder1 = EncoderBlock(hidden_channels)
        self.encoder2 = EncoderBlock(hidden_channels * 2)

        res_mult = 4
        self.res0 = ResidualBlock(hidden_channels * res_mult)
        self.res1 = ResidualBlock(hidden_channels * res_mult)
        self.res2 = ResidualBlock(hidden_channels * res_mult)
        self.res3 = ResidualBlock(hidden_channels * res_mult)
        self.res4 = ResidualBlock(hidden_channels * res_mult)
        self.res5 = ResidualBlock(hidden_channels * res_mult)
        self.res6 = ResidualBlock(hidden_channels * res_mult)
        self.res7 = ResidualBlock(hidden_channels * res_mult)
        self.res8 = ResidualBlock(hidden_channels * res_mult)

        self.decoder1 = DecoderBlock(hidden_channels * 4)
        self.decoder2 = DecoderBlock(hidden_channels * 2)

        self.downfeature = FeatureMapBlock(hidden_channels,
output_channels)
        self.tanh = torch.nn.Tanh()

    def forward(self, x):
        x0 = self.upfeature(x)
        x1 = self.encoder1(x0)
        x2 = self.encoder2(x1)
        x3 = self.res0(x2)
```

```

x4 = self.res1(x3)
x5 = self.res2(x4)
x6 = self.res3(x5)
x7 = self.res4(x6)
x8 = self.res5(x7)
x9 = self.res6(x8)
x10 = self.res7(x9)
x11 = self.res8(x10)
x12 = self.decoder1(x11)
x13 = self.decoder2(x12)
xn = self.downfeature(x13)
return self.tanh(xn)

```

Kelas Generator di atas adalah bagian dari arsitektur model dalam jaringan saraf tiruan, yang digunakan dalam banyak aplikasi seperti CycleGAN, pix2pix, atau model generatif lainnya. Generator bertugas untuk menghasilkan gambar dari input yang diberikan, umumnya input berupa noise atau gambar dengan domain tertentu, dan menghasilkan gambar dalam domain target.

Komponen dan Fungsi Utama:

1. `__init__(self, input_channels, output_channels, hidden_channels=64):`

- Konstruktor dari kelas Generator yang menginisialisasi seluruh lapisan yang akan digunakan.
- **input_channels:** Jumlah saluran input (misalnya, 3 untuk gambar RGB).
- **output_channels:** Jumlah saluran output (misalnya, 3 untuk gambar RGB).
- **hidden_channels:** Jumlah saluran yang digunakan di lapisan tersembunyi. Nilai default adalah 64.

2. Lapisan FeatureMapBlock:

- **self.upfeature** dan **self.downfeature**: Menggunakan FeatureMapBlock untuk mengubah jumlah saluran input dan output. Ini memungkinkan jaringan untuk menangkap dan mengubah fitur gambar.

3. Encoder dan Residual Blocks:

- **self.encoder1** dan **self.encoder2**: Dua lapisan encoder yang mengurangi dimensi spasial gambar sambil meningkatkan jumlah saluran, menangkap fitur penting dalam proses encoding.
- **self.res0** hingga **self.res8**: Delapan lapisan residual yang menggunakan blok residual (diimplementasikan dengan ResidualBlock) untuk memodifikasi gambar tanpa menghilangkan informasi penting. Proses residual ini memungkinkan jaringan untuk mempelajari representasi yang lebih dalam dan mengurangi masalah *vanishing gradients*.

4. Decoder Blocks:

- **self.decoder1** dan **self.decoder2**: Decoder blocks yang akan mengubah gambar ke dimensi yang lebih tinggi (up-sampling), mengembalikan gambar ke bentuk semula setelah melalui blok residual. Decoder ini membantu mengembalikan dimensi spasial gambar ke ukuran yang diinginkan.

5. Fungsi Aktivasi Tanh:

- **self.tanh**: Fungsi aktivasi Tanh digunakan pada output akhir untuk memetakan nilai piksel gambar dari rentang [0, 1] ke rentang [-1, 1]. Ini penting dalam jaringan GAN agar hasil output dapat dibandingkan dengan hasil asli.

6. forward(self, x):

- Fungsi ini mendefinisikan bagaimana data mengalir melalui jaringan dari input hingga output.
- **Proses Maju**: Data (x) pertama-tama diproses oleh **upfeature**, kemudian dua encoder (encoder1 dan encoder2), diikuti oleh delapan lapisan residual (res0 hingga res8), dua decoder (decoder1 dan decoder2), dan akhirnya **downfeature** menghasilkan gambar akhir.
- **Tanh Activation**: Hasil akhir gambar diproses dengan fungsi aktivasi Tanh untuk memastikan output berada dalam rentang yang diinginkan.

Model ini bisa digunakan dalam arsitektur seperti CycleGAN untuk melakukan konversi gambar antar domain (misalnya, dari gambar kuda ke zebra) atau untuk menghasilkan gambar-gambar sintetis yang realistik.

Discriminator block:

```
class Discriminator(nn.Module):
    """
    Kelas Discriminator:
    Menentukan jaringan diskriminator untuk model generatif, seperti
    GAN.

    Peran diskriminator adalah untuk membedakan antara data asli dan
    palsu (yang dihasilkan).

    Atribut:
        input_channels (int): Jumlah saluran input (misalnya, untuk
        gambar RGB, ini akan menjadi 3).
        hidden_channels (int): Jumlah saluran dasar yang digunakan
        dalam jaringan, yang meningkat di lapisan yang lebih dalam.
    """

    def __init__(self, input_channels, hidden_channels=64):
        super(Discriminator, self).__init__()

        # Pemetaan fitur awal untuk meningkatkan jumlah saluran
        self.upfeature = FeatureMapBlock(input_channels,
hidden_channels)
```

```

        # Blok kontraksi (Blok Encoder) - mengurangi dimensi spasial
        dan meningkatkan kedalaman saluran
        # Menggunakan aktivasi LeakyReLU (lrelu) untuk setiap blok
        kontraksi
        self.contract1 = EncoderBlock(hidden_channels, use_bn=False,
kernel_size=4, activation='lrelu')
        self.contract2 = EncoderBlock(hidden_channels * 2,
kernel_size=4, activation='lrelu')
        self.contract3 = EncoderBlock(hidden_channels * 4,
kernel_size=4, activation='lrelu')

        # Konvolusi terakhir untuk menghasilkan satu saluran
        (klasifikasi biner untuk asli atau palsu)
        self.final = nn.Conv2d(hidden_channels * 8, 1, kernel_size=1)

    def forward(self, x):
        """
        Menentukan proses maju dari diskriminator.
        """
        # Terapkan secara berurutan pemetaan fitur dan blok kontraksi
        x0 = self.upfeature(x)
        x1 = self.contract1(x0)
        x2 = self.contract2(x1)
        x3 = self.contract3(x2)

        # Terapkan konvolusi terakhir
        xn = self.final(x3)
        return xn

```

Kelas Discriminator adalah bagian dari model generatif seperti Generative Adversarial Network (GAN), yang memiliki peran untuk membedakan antara gambar asli dan gambar palsu (yang dihasilkan oleh generator). Dalam hal ini, diskriminator digunakan untuk memberi umpan balik ke generator, yang kemudian digunakan untuk meningkatkan kualitas gambar yang dihasilkan.

Komponen dan Fungsi Utama:

1. `__init__(self, input_channels, hidden_channels=64):`

- Konstruktor dari kelas Discriminator yang menginisialisasi seluruh lapisan dalam jaringan diskriminator.
- **input_channels:** Jumlah saluran input (misalnya, 3 untuk gambar RGB).
- **hidden_channels:** Jumlah saluran dasar yang digunakan dalam jaringan (nilai default adalah 64).

2. `self.upfeature:`

- Menggunakan **FeatureMapBlock** untuk meningkatkan jumlah saluran input sebelum memulai proses kontraksi (downsampling).

3. Blok Kontraksi (Encoder Blocks):

- **self.contract1**: Menggunakan **EncoderBlock** untuk mengurangi dimensi spasial dan meningkatkan jumlah saluran pada gambar input. Aktivasi yang digunakan adalah **LeakyReLU** yang membantu menghindari masalah gradien yang hilang.
- **self.contract2** dan **self.contract3**: Dua blok encoder lainnya yang melanjutkan proses pengurangan dimensi spasial dan peningkatan kedalaman saluran.

4. Lapisan Konvolusi Terakhir:

- **self.final**: Setelah melalui blok encoder, gambar akan diproses oleh konvolusi akhir untuk menghasilkan output dengan satu saluran. Output ini adalah nilai klasifikasi untuk menentukan apakah gambar tersebut asli atau palsu (biner).

5. **forward(self, x)**:

- Fungsi ini mendefinisikan aliran data melalui jaringan dari input hingga output.
- **Proses Maju**: Data (x) pertama-tama diproses oleh **upfeature**, kemudian tiga blok kontraksi (contract1, contract2, contract3), dan akhirnya diproses oleh **final** untuk menghasilkan output klasifikasi biner.

Persiapan pelatihan:

```
adv_criterion = nn.MSELoss()
recon_criterion = nn.L1Loss()
device = 'cuda'
n_epochs = 20
dim_A = 3
dim_B = 3
display_step = 200
batch_size = 1
lr = 0.0002
```

Sekarang dapat menyatukan semuanya untuk pelatihan. Mulai dengan mendefinisikan parameter-parameter Anda:

- **adv_criterion**: fungsi loss adversarial untuk melacak seberapa baik GAN menipu discriminator dan seberapa baik discriminator menangkap GAN.
- **recon_criterion**: fungsi loss yang memberi penghargaan pada gambar yang mirip dengan ground truth, yang "membangun kembali" gambar.
- **n_epochs**: jumlah iterasi melalui seluruh dataset saat pelatihan.
- **dim_A**: jumlah saluran gambar pada tumpukan A.
- **dim_B**: jumlah saluran gambar pada tumpukan B (perhatikan bahwa dalam visualisasi ini saat ini diperlakukan setara dengan dim_A).
- **display_step**: seberapa sering menampilkan/memvisualisasikan gambar.
- **batch_size**: jumlah gambar per langkah maju/mundur.

- **lr:** laju pembelajaran.

Menginisialisasi generator dan discriminator:

```
gen_AB = Generator(dim_A, dim_B).to(device)
gen_BA = Generator(dim_B, dim_A).to(device)
gen_opt = torch.optim.Adam(list(gen_AB.parameters()) +
list(gen_BA.parameters()), lr=lr, betas=(0.5, 0.999))
disc_A = Discriminator(dim_A).to(device)
disc_A_opt = torch.optim.Adam(disc_A.parameters(), lr=lr, betas=(0.5,
0.999))
disc_B = Discriminator(dim_B).to(device)
disc_B_opt = torch.optim.Adam(disc_B.parameters(), lr=lr, betas=(0.5,
0.999))

def weights_init(m):
    if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):
        torch.nn.init.normal_(m.weight, 0.0, 0.02)
    if isinstance(m, nn.BatchNorm2d):
        torch.nn.init.normal_(m.weight, 0.0, 0.02)
        torch.nn.init.constant_(m.bias, 0)

# Melatih model dari awal
gen_AB = gen_AB.apply(weights_init)
gen_BA = gen_BA.apply(weights_init)
disc_A = disc_A.apply(weights_init)
disc_B = disc_B.apply(weights_init)
```

Penjelasan mengenai kode tersebut:

1. **gen_AB = Generator(dim_A, dim_B).to(device):**

- Ini adalah generator pertama yang bertugas untuk mengubah gambar kuda (domain A) menjadi gambar zebra (domain B). Generator ini diinisialisasi dengan saluran input dim_A (jumlah saluran gambar input) dan saluran output dim_B (jumlah saluran gambar target). Model ini dipindahkan ke perangkat yang sesuai (CPU/GPU).

2. **gen_BA = Generator(dim_B, dim_A).to(device):**

- Ini adalah generator kedua yang berfungsi sebaliknya, yaitu mengubah gambar zebra (domain B) menjadi gambar kuda (domain A). Model ini diinisialisasi dengan saluran input dim_B dan output dim_A.

3. **gen_opt = torch.optim.Adam(list(gen_AB.parameters()) + list(gen_BA.parameters()), lr=lr, betas=(0.5, 0.999)):**

- Optimizer **Adam** digunakan untuk memperbarui parameter kedua generator (gen_AB dan gen_BA). Dengan ini, optimizer akan mengoptimalkan kedua generator dalam satu langkah.

4. **disc_A = Discriminator(dim_A).to(device):**

- Ini adalah discriminator yang bertugas untuk membedakan gambar asli (kuda) dan gambar palsu (hasil dari generator gen_BA). Discriminator ini menggunakan dim_A sebagai input, yang sesuai dengan gambar kuda.

5. **disc_A_opt = torch.optim.Adam(disc_A.parameters(), lr=lr, betas=(0.5, 0.999)):**

- Optimizer **Adam** untuk memperbarui parameter **Discriminator A** selama pelatihan.

6. **disc_B = Discriminator(dim_B).to(device):**

- Ini adalah discriminator untuk domain B (zebra). Tugasnya adalah untuk membedakan gambar asli (zebra) dan gambar palsu (hasil dari generator gen_AB).

7. **disc_B_opt = torch.optim.Adam(disc_B.parameters(), lr=lr, betas=(0.5, 0.999)):**

- Optimizer **Adam** untuk memperbarui parameter **Discriminator B**.

8. **def weights_init(m)::**

- Fungsi ini digunakan untuk menginisialisasi bobot jaringan, khususnya lapisan konvolusional dan konvolusional transpose dengan distribusi normal (mean = 0, std = 0.02). Ini bertujuan untuk memastikan stabilitas pelatihan dengan memulai bobot dalam kisaran yang wajar.

9. **gen_AB = gen_AB.apply(weights_init), gen_BA = gen_BA.apply(weights_init), disc_A = disc_A.apply(weights_init), disc_B = disc_B.apply(weights_init):**

- Menerapkan fungsi weights_init untuk menginisialisasi bobot pada semua lapisan konvolusional dan konvolusional transpose dari kedua generator dan kedua discriminator. Hal ini penting untuk memulai pelatihan dengan bobot yang baik.

Fungsi dan Tujuan:

- **Generator (gen_AB):** Menghasilkan gambar zebra dari gambar kuda.
- **Generator (gen_BA):** Menghasilkan gambar kuda dari gambar zebra.
- **Discriminator (disc_A):** Menilai gambar apakah itu gambar kuda asli atau gambar palsu yang dihasilkan oleh gen_BA.
- **Discriminator (disc_B):** Menilai gambar apakah itu gambar zebra asli atau gambar palsu yang dihasilkan oleh gen_AB.

Pada akhirnya, kedua generator (gen_AB dan gen_BA) berfungsi dalam **CycleGAN** untuk mempelajari bagaimana mentransformasikan gambar antar dua domain (kuda dan zebra) dengan cara yang tidak terawasi, sedangkan kedua discriminator bertugas untuk memvalidasi apakah gambar yang dihasilkan realistik atau tidak.

Discriminator loss:

```

def get_disc_loss(real_X, fake_X, disc_X, adv_criterion):
    """
    Menghitung loss dari discriminator.

    # Lewatkan gambar palsu melalui discriminator dan hitung loss
    terhadap nol
    disc_fake = disc_X(fake_X)
    # Discriminator harus menghasilkan output mendekati 0 untuk gambar
    palsu
    disc_fake_loss = adv_criterion(disc_fake,
    torch.zeros_like(disc_fake))

    # Lewatkan gambar asli melalui discriminator dan hitung loss
    terhadap satu
    disc_real = disc_X(real_X)
    # Discriminator harus menghasilkan output mendekati 1 untuk gambar
    asli
    disc_real_loss = adv_criterion(disc_real,
    torch.ones_like(disc_real))

    # Rata-rata loss palsu dan asli
    disc_loss = (disc_fake_loss + disc_real_loss) / 2

    return disc_loss

```

Fungsi `get_disc_loss` ini bertujuan untuk menghitung loss untuk discriminator dalam konteks pelatihan Generative Adversarial Network (GAN) atau model seperti CycleGAN. Discriminator berperan dalam membedakan gambar asli dan gambar palsu yang dihasilkan oleh generator.

Parameter:

- **real_X**: Gambar asli yang ada dalam dataset (misalnya, gambar kuda atau zebra asli).
- **fake_X**: Gambar palsu yang dihasilkan oleh generator (misalnya, gambar kuda palsu yang dihasilkan oleh gen_BA atau gambar zebra palsu oleh gen_AB).
- **disc_X**: Discriminator untuk domain X, yang digunakan untuk menilai gambar asli dan palsu dalam domain tersebut (misalnya, disc_A untuk domain kuda atau disc_B untuk domain zebra).
- **adv_criterion**: Fungsi loss adversarial (biasanya menggunakan **Mean Squared Error (MSE) Loss** atau **Binary Cross-Entropy Loss**) yang digunakan untuk menghitung perbedaan antara output discriminator dan target yang diinginkan (0 untuk gambar palsu, 1 untuk gambar asli).

Proses:

1. Proses untuk Gambar Palsu:

- Gambar palsu yang dihasilkan oleh generator (misalnya, `fake_X`) diberikan ke discriminator (`disc_X`).

- Output dari discriminator (disc_fake) diharapkan mendekati **0** untuk gambar palsu.
 - Loss untuk gambar palsu (disc_fake_loss) dihitung dengan membandingkan output discriminator terhadap tensor yang berisi nilai **0** menggunakan fungsi adv_criterion. Ini mengukur seberapa jauh output discriminator dari target yang diinginkan.

2. Proses untuk Gambar Asli:

- Gambar asli (real_X) juga diberikan ke discriminator.
 - Output dari discriminator (disc_real) diharapkan mendekati **1** untuk gambar asli.
 - Loss untuk gambar asli (disc_real_loss) dihitung dengan membandingkan output discriminator terhadap tensor yang berisi nilai **1** menggunakan fungsi adv criterion.

3. Menghitung Total Loss:

- Total loss untuk discriminator dihitung sebagai rata-rata dari `disc_fake_loss` dan `disc_real_loss`, karena tujuan discriminator adalah untuk membedakan gambar asli (yang seharusnya bernilai 1) dan gambar palsu (yang seharusnya bernilai 0).
 - Loss ini kemudian digunakan untuk memperbarui bobot discriminator melalui proses backpropagation.

Fungsi Return:

- **disc_loss:** Total loss yang dihitung untuk discriminator. Ini adalah rata-rata dari loss gambar asli dan gambar palsu, dan digunakan untuk memperbarui parameter discriminator.

Generator loss:

```
# Kembalikan adversarial loss dan gambar yang dihasilkan
return adversarial_loss, fake_Y
```

Fungsi `get_gen_adversarial_loss` ini digunakan untuk menghitung adversarial loss bagi generator dalam konteks model Generative Adversarial Network (GAN) atau model seperti CycleGAN. Dalam hal ini, generator bertugas untuk menghasilkan gambar palsu (fake images) yang seolah-olah asli, dan tujuan adversarial loss adalah untuk menipu discriminator agar menerima gambar palsu tersebut sebagai gambar asli.

Parameter:

- **real_X**: Gambar asli dari domain X (misalnya, gambar kuda atau zebra).
- **disc_Y**: Discriminator untuk domain Y (misalnya, disc_B untuk domain zebra jika gambar asli berasal dari domain kuda).
- **gen_XY**: Generator yang mengonversi gambar dari domain X ke domain Y (misalnya, gen_AB yang mengonversi gambar kuda ke zebra).
- **adv_criterion**: Fungsi loss adversarial (biasanya menggunakan **Binary Cross-Entropy Loss** atau **Mean Squared Error Loss**) yang menghitung perbedaan antara output dari discriminator dan nilai target yang diinginkan (biasanya 1 untuk gambar yang diterima oleh discriminator).

Proses:

1. Menghasilkan Gambar Palsu:

- Generator `gen_XY` digunakan untuk mengubah gambar asli dari domain X (`real_X`) menjadi gambar palsu di domain Y (`fake_Y`).

2. Lewatkan Gambar Palsu ke Discriminator:

- Gambar palsu (`fake_Y`) kemudian diberikan ke discriminator (`disc_Y`) untuk menilai apakah gambar tersebut terlihat seperti gambar asli dalam domain Y.
- Output dari discriminator (`fake_disc_Y`) menunjukkan sejauh mana gambar palsu tersebut diterima sebagai gambar asli.

3. Menghitung Loss:

- Generator ingin menipu discriminator agar menerima gambar palsu sebagai gambar asli. Oleh karena itu, **tujuan generator adalah agar output dari discriminator mendekati nilai 1** (artinya, discriminator percaya bahwa gambar tersebut asli).
- Adversarial loss dihitung dengan membandingkan output discriminator (`fake_disc_Y`) terhadap tensor yang berisi nilai **1** (nilai target untuk gambar yang diterima oleh discriminator).

4. Mengembalikan Loss dan Gambar:

- Fungsi ini mengembalikan **adversarial loss** yang dihitung, serta **gambar palsu (`fake_Y`)** yang dihasilkan oleh generator. Loss ini akan digunakan untuk

memperbarui bobot generator agar lebih baik dalam menipu discriminator pada iterasi berikutnya.

Fungsi Return:

- **adversarial_loss**: Loss yang dihitung untuk generator, yang mengukur seberapa baik generator dalam menghasilkan gambar yang dapat menipu discriminator.
- **fake_Y**: Gambar yang dihasilkan oleh generator yang telah diubah dari domain X ke domain Y. Gambar ini digunakan oleh discriminator untuk mengevaluasi kinerja generator.

Identify loss:

```
def get_identity_loss(real_X, gen_YX, identity_criterion):  
    '''  
        Menghitung identity loss untuk sebuah generator.  
    '''  
  
    # Lewatkan gambar asli dari domain target melalui generator  
    identity_X = gen_YX(real_X)  
  
    # Hitung identity loss, yang merupakan perbedaan antara output  
    # generator  
    # dan gambar asli. Loss akan lebih rendah ketika generator mengubah  
    # gambar lebih sedikit.  
    identity_loss = identity_criterion(real_X, identity_X)  
  
    return identity_loss, identity_X
```

Fungsi `get_identity_loss` ini digunakan untuk menghitung identity loss dalam konteks model Generative Adversarial Network (GAN), seperti CycleGAN atau model berbasis transformasi citra lainnya. Identity loss mengukur sejauh mana generator mengubah gambar input ketika gambar tersebut sudah berada di domain target, yang berarti generator diharapkan untuk tidak mengubah gambar yang sudah sesuai dengan target domain.

Parameter:

- **real_X (Tensor)**: Sekelompok gambar asli dari **domain target** yang akan diproses oleh generator. Misalnya, jika kita memiliki dua domain gambar (kuda dan zebra), dan generator diharapkan untuk mengonversi gambar kuda ke zebra, maka `real_X` adalah gambar kuda.
- **gen_YX (Module)**: Model generator yang digunakan untuk mengonversi gambar dari **domain Y ke domain X**. Dalam hal ini, generator ini tidak perlu mengubah gambar yang sudah berada di domain target. Misalnya, jika gambar dari domain X (kuda) diberikan, generator seharusnya tidak mengubah gambar tersebut jika gambar itu sudah sesuai dengan domain X (misalnya, gambar zebra yang sudah ada di domain Y tidak boleh diubah).

- **identity_criterion (Loss Function):** Fungsi loss yang digunakan untuk menghitung **identity loss**, biasanya menggunakan **L1 Loss** atau **L2 Loss** untuk menghitung perbedaan antara gambar yang dihasilkan dan gambar asli.

Proses:

1. Lewatkan Gambar Asli Melalui Generator:

- Gambar asli dari domain target (real_X) diteruskan ke generator gen_YX. Generator ini seharusnya hanya mengubah gambar jika gambar tersebut berasal dari domain yang tidak sesuai dengan target. Misalnya, jika real_X adalah gambar zebra, dan gen_YX adalah generator yang mengonversi zebra ke kuda, maka gambar zebra akan diubah menjadi gambar kuda.

2. Menghitung Identity Loss:

- Setelah gambar diproses oleh generator, gambar hasil output (identity_X) dibandingkan dengan gambar input asli (real_X). Jika generator bekerja dengan benar, maka untuk gambar yang sudah berada di domain target, output generator harus identik dengan inputnya.
- **Identity loss** dihitung dengan menghitung perbedaan antara real_X dan identity_X menggunakan fungsi **identity_criterion** (misalnya L1 atau L2 loss). **Identity loss** akan lebih rendah jika gambar asli dan hasil output generator sangat mirip, yang menunjukkan bahwa generator tidak mengubah gambar yang sudah berada di domain target.

3. Mengembalikan Identity Loss dan Gambar yang Dihasilkan:

- Fungsi ini mengembalikan dua nilai:
 1. **identity_loss:** Loss yang dihitung antara gambar asli dan gambar yang dihasilkan oleh generator. Ini mengukur sejauh mana generator mempertahankan identitas gambar input.
 2. **identity_X:** Gambar yang dihasilkan oleh generator, yang seharusnya identik dengan gambar input (real_X) jika gambar tersebut sudah berada di domain target.

Fungsi Return:

- **identity_loss:** Loss yang mengukur perbedaan antara gambar asli dan gambar yang dihasilkan oleh generator. Semakin kecil nilai loss ini, semakin baik generator dalam mempertahankan identitas gambar yang sudah berada di domain target.
- **identity_X:** Gambar yang dihasilkan oleh generator. Untuk gambar yang sudah berada di domain target, gambar ini seharusnya hampir identik dengan gambar input (real_X).

Cycle consistency loss:

```
def get_cycle_consistency_loss(real_X, fake_Y, gen_YX,
cycle_criterion):
    ...
```

```
Menghitung cycle consistency loss untuk sepasang gambar.
```

```
'''
```

```
# Transformasi gambar palsu dari domain Y kembali ke domain X
cycle_X = gen_YX(fake_Y)

# Hitung cycle consistency loss (seberapa dekat cycle_X dengan
real_X asli)
cycle_loss = cycle_criterion(real_X, cycle_X)

return cycle_loss, cycle_X
```

Fungsi `get_cycle_consistency_loss` ini digunakan untuk menghitung cycle consistency loss dalam konteks model CycleGAN atau model transformasi citra lainnya yang melibatkan dua domain (misalnya, domain X dan Y). Cycle consistency loss digunakan untuk memastikan bahwa gambar yang diubah dari satu domain ke domain lain dapat dikembalikan ke bentuk aslinya tanpa kehilangan informasi penting. Dengan kata lain, model diharapkan untuk mengubah gambar dari domain X ke domain Y dan kemudian mengubahnya kembali ke domain X, dan hasil akhirnya harus sangat mirip dengan gambar asli.

Parameter:

- **real_X (Tensor):** Sekelompok gambar asli dari **domain X**, yang merupakan gambar input yang ingin kita ubah ke domain Y dan kemudian kembali lagi ke domain X. Misalnya, gambar kuda dalam domain X.
- **fake_Y (Tensor):** Gambar yang telah diubah ke **domain Y**. Gambar ini berasal dari domain X, tetapi telah diproses oleh generator untuk beralih ke domain Y. Misalnya, gambar kuda yang telah diubah menjadi zebra oleh generator.
- **gen_YX (Module):** Generator yang digunakan untuk mengubah gambar dari **domain Y kembali ke domain X**. Generator ini harus bisa mengembalikan gambar yang telah diproses di domain Y (seperti gambar zebra) ke bentuk aslinya di domain X (gambar kuda).
- **cycle_criterion (Loss Function):** Fungsi loss yang digunakan untuk menghitung perbedaan antara gambar asli (`real_X`) dan gambar yang dikembalikan (`cycle_X`). Biasanya menggunakan **L1 Loss** atau **L2 Loss** untuk menghitung perbedaan pixel antara dua gambar.

Proses:

1. Transformasi Gambar Palsu Kembali ke Domain X:

- Gambar yang telah diubah ke domain Y (`fake_Y`) diteruskan ke generator `gen_YX`, yang bertanggung jawab untuk mengubah gambar tersebut kembali ke domain X. Proses ini mengembalikan gambar yang disebut **cycle_X**, yang diharapkan sangat mirip dengan gambar asli dari domain X.

2. Menghitung Cycle Consistency Loss:

- Setelah gambar diputar balik ke domain X, kita menghitung **cycle consistency loss** dengan membandingkan **cycle_X** (gambar yang diputar balik) dengan **real_X** (gambar asli yang berasal dari domain X).
- **Cycle consistency loss** mengukur seberapa mirip **cycle_X** dengan **real_X**. Tujuannya adalah untuk memastikan bahwa model dapat mengubah gambar dari satu domain ke domain lain dan kemudian mengembalikannya tanpa kehilangan informasi penting. Semakin kecil nilai loss ini, semakin baik siklus konsistensi yang dihasilkan oleh generator.

3. Mengembalikan Cycle Consistency Loss dan Gambar yang Diputar Balik:

- Fungsi ini mengembalikan dua nilai:
 1. **cycle_loss**: Loss yang dihitung untuk mengevaluasi sejauh mana gambar yang dikembalikan (**cycle_X**) mirip dengan gambar asli (**real_X**). Semakin rendah nilai loss ini, semakin baik hasil siklus konsistensi.
 2. **cycle_X**: Gambar yang dihasilkan oleh proses siklus, yaitu gambar yang telah diputar balik dari domain Y kembali ke domain X.

Fungsi Return:

- **cycle_loss**: Loss yang mengukur perbedaan antara gambar asli dan gambar yang telah diputar balik oleh generator. Loss ini harus kecil agar generator dapat mempertahankan konsistensi siklus dengan baik.
- **cycle_X**: Gambar yang diputar balik dari domain Y ke domain X. Gambar ini harus sangat mirip dengan gambar asli **real_X** jika konsistensi siklus tercapai dengan baik.

Generator loss:

```
def get_gen_loss(real_A, real_B, gen_AB, gen_BA, disc_A, disc_B,
adv_criterion, identity_criterion, cycle_criterion,
lambda_identity=0.1, lambda_cycle=10):
    """
    Menghitung loss dari generator berdasarkan input.
    """

    # Adversarial Loss -- get_gen_adversarial_loss(real_X, disc_Y,
    gen_XY, adv_criterion)
    adversarial_loss_AB, fake_B = get_gen_adversarial_loss(real_A,
disc_B, gen_AB, adv_criterion)
    adversarial_loss_BA, fake_A = get_gen_adversarial_loss(real_B,
disc_A, gen_BA, adv_criterion)
    adversarial_loss = adversarial_loss_AB + adversarial_loss_BA

    # Identity Loss -- get_identity_loss(real_X, gen_YX,
    identity_criterion)
    identity_loss_A, _ = get_identity_loss(real_A, gen_BA,
identity_criterion)
```

```

    identity_loss_B, _ = get_identity_loss(real_B, gen_AB,
identity_criterion)
    identity_loss = identity_loss_A + identity_loss_B

    # Cycle-consistency Loss -- get_cycle_consistency_loss(real_X,
fake_Y, gen_YX, cycle_criterion)
    cycle_consistency_loss_BA, _ = get_cycle_consistency_loss(real_A,
fake_B, gen_BA, cycle_criterion)
    cycle_consistency_loss_AB, _ = get_cycle_consistency_loss(real_B,
fake_A, gen_AB, cycle_criterion)
    cycle_consistency_loss = cycle_consistency_loss_BA +
cycle_consistency_loss_AB

    # Total loss
    gen_loss = adversarial_loss + lambda_identity * identity_loss +
lambda_cycle * cycle_consistency_loss

    return gen_loss, fake_A, fake_B

```

Fungsi `get_gen_loss` digunakan untuk menghitung total loss untuk generator dalam model CycleGAN (atau model yang serupa) yang memiliki dua domain (misalnya, domain A dan domain B). Fungsi ini menggabungkan beberapa jenis loss: adversarial loss, identity loss, dan cycle-consistency loss. Setiap jenis loss memiliki peran tertentu untuk memastikan kualitas gambar yang dihasilkan oleh generator, serta untuk menjaga agar generator bisa mengembalikan gambar ke bentuk asalnya tanpa kehilangan informasi penting.

Parameter:

- **real_A (Tensor)**: Gambar asli dari domain A.
- **real_B (Tensor)**: Gambar asli dari domain B.
- **gen_AB (Module)**: Generator yang mengubah gambar dari domain A ke domain B.
- **gen_BA (Module)**: Generator yang mengubah gambar dari domain B ke domain A.
- **disc_A (Module)**: Discriminator untuk domain A, yang digunakan untuk membedakan gambar asli dan palsu di domain A.
- **disc_B (Module)**: Discriminator untuk domain B, yang digunakan untuk membedakan gambar asli dan palsu di domain B.
- **adv_criterion (Loss Function)**: Fungsi loss untuk menghitung **adversarial loss** (biasanya menggunakan binary cross-entropy).
- **identity_criterion (Loss Function)**: Fungsi loss yang digunakan untuk menghitung **identity loss** dan **cycle consistency loss**. Biasanya menggunakan L1 loss atau L2 loss untuk menghitung perbedaan antara gambar asli dan gambar yang dipulihkan.
- **cycle_criterion (Loss Function)**: Fungsi loss yang digunakan untuk menghitung **cycle consistency loss**, yang mengukur seberapa baik gambar yang diubah kembali ke domain asalnya serupa dengan gambar asli.

- **lambda_identity (float, default=0.1)**: Bobot untuk **identity loss**. Mengatur seberapa penting identity loss dalam total loss.
- **lambda_cycle (float, default=10)**: Bobot untuk **cycle consistency loss**. Mengatur seberapa penting cycle consistency loss dalam total loss.

Langkah-langkah yang Dilakukan dalam Fungsi:

1. Adversarial Loss:

- Fungsi pertama kali menghitung **adversarial loss** untuk dua generator: gen_AB dan gen_BA.
 - **get_gen_adversarial_loss(real_A, disc_B, gen_AB, adv_criterion)** menghitung adversarial loss untuk generator yang mengubah gambar dari domain A ke domain B (AB).
 - **get_gen_adversarial_loss(real_B, disc_A, gen_BA, adv_criterion)** menghitung adversarial loss untuk generator yang mengubah gambar dari domain B ke domain A (BA).
- Total **adversarial loss** adalah jumlah dari kedua nilai tersebut: adversarial_loss_AB + adversarial_loss_BA.

2. Identity Loss:

- Fungsi kemudian menghitung **identity loss** untuk kedua generator.
 - **get_identity_loss(real_A, gen_BA, identity_criterion)** menghitung identity loss untuk generator BA (B ke A).
 - **get_identity_loss(real_B, gen_AB, identity_criterion)** menghitung identity loss untuk generator AB (A ke B).
- Total **identity loss** adalah jumlah dari kedua nilai tersebut: identity_loss_A + identity_loss_B.

3. Cycle Consistency Loss:

- Fungsi menghitung **cycle consistency loss** untuk kedua generator.
 - **get_cycle_consistency_loss(real_A, fake_B, gen_BA, cycle_criterion)** menghitung cycle consistency loss untuk gambar yang diubah dari A ke B, kemudian kembali ke A menggunakan generator BA.
 - **get_cycle_consistency_loss(real_B, fake_A, gen_AB, cycle_criterion)** menghitung cycle consistency loss untuk gambar yang diubah dari B ke A, kemudian kembali ke B menggunakan generator AB.
- Total **cycle consistency loss** adalah jumlah dari kedua nilai tersebut: cycle_consistency_loss_BA + cycle_consistency_loss_AB.

4. Menghitung Total Loss:

- **gen_loss** dihitung dengan menjumlahkan ketiga loss yang telah dihitung sebelumnya:

- **Adversarial loss.**
- **Identity loss** yang dikalikan dengan lambda_identity.
- **Cycle consistency loss** yang dikalikan dengan lambda_cycle.

Return Value:

Fungsi ini mengembalikan tiga nilai:

- **gen_loss:** Total loss untuk generator, yang digunakan untuk melatih generator agar menghasilkan gambar yang semakin realistik dan mempertahankan informasi gambar yang relevan.
- **fake_A:** Gambar yang dihasilkan oleh generator AB (yang mengubah gambar dari domain B ke domain A).
- **fake_B:** Gambar yang dihasilkan oleh generator BA (yang mengubah gambar dari domain A ke domain B).

Tujuan dari Loss Components:

- **Adversarial Loss:** Membantu generator agar gambar yang dihasilkannya dapat "menipu" discriminator dan tampak realistik.
- **Identity Loss:** Memastikan bahwa gambar yang sudah berada di domain target tidak banyak berubah (misalnya, jika gambar kuda sudah berupa kuda, generator harus menghasilkan gambar kuda yang sama).
- **Cycle Consistency Loss:** Memastikan bahwa gambar yang diubah ke domain lain dapat kembali ke bentuk aslinya tanpa kehilangan informasi penting.

Pelatihan CycleGAN:

```
# Membuat direktori /pth untuk menyimpan file model
!mkdir pth
!ls
→ mkdir: cannot create directory 'pth': File exists
      pth  sample_data
```

Kode diatas digunakan untuk membuat sebuah direktori baru dan kemudian menampilkan daftar file dan direktori yang ada di dalamnya.

Penjelasan setiap perintah:

1. !mkdir pth:

- Perintah ini digunakan untuk membuat direktori baru dengan nama pth di dalam direktori kerja saat ini.
- **mkdir** adalah singkatan dari "make directory", yang digunakan untuk membuat folder baru.

2. !ls:

- Perintah ini digunakan untuk menampilkan daftar file dan folder yang ada dalam direktori saat ini.
- **ls** (list) akan menunjukkan semua file dan direktori yang ada di lokasi tempat perintah ini dijalankan.

Kegunaan:

- **mkdir pth** berguna untuk membuat tempat penyimpanan model atau file yang relevan, seperti model terlatih (misalnya file .pth dalam PyTorch).
- **ls** digunakan untuk memastikan bahwa direktori baru pth telah berhasil dibuat dan melihat apa saja yang ada di direktori kerja saat ini.

Jika Anda melihat direktori pth dalam output dari ls, itu berarti direktori tersebut telah berhasil dibuat.

```
def train(save_model=False):
    mean_generator_loss = 0
    mean_discriminator_loss = 0
    dataloader = DataLoader(dataset, batch_size=batch_size,
                           shuffle=True)

    cur_step = 0

    for epoch in range(n_epochs):
        # The dataloader returns the batches
        # for image, _ in tqdm(dataloader):
        for real_A, real_B in tqdm(dataloader):
            real_A = nn.functional.interpolate(real_A,
size=target_shape)
            real_B = nn.functional.interpolate(real_B,
size=target_shape)
            cur_batch_size = len(real_A)
            real_A = real_A.to(device)
            real_B = real_B.to(device)

            ### Update discriminator A ####
            disc_A_opt.zero_grad()  # Zero out the gradient before
backpropagation
            with torch.no_grad():
                fake_A = gen_BA(real_B)
                disc_A_loss = get_disc_loss(real_A, fake_A, disc_A,
adv_criterion)
                disc_A_loss.backward(retain_graph=True)  # Update gradients
                disc_A_opt.step()  # Update optimizer

            ### Update discriminator B ####
            disc_B_opt.zero_grad()  # Zero out the gradient before
backpropagation
            with torch.no_grad():
```

```

        fake_B = gen_AB(real_A)
        disc_B_loss = get_disc_loss(real_B, fake_B, disc_B,
adv_criterion)
        disc_B_loss.backward(retain_graph=True) # Update gradients
        disc_B_opt.step() # Update optimizer

        ### Update generator ###
        gen_opt.zero_grad()
        gen_loss, fake_A, fake_B = get_gen_loss(
            real_A, real_B, gen_AB, gen_BA, disc_A, disc_B,
adv_criterion, recon_criterion, recon_criterion
        )
        gen_loss.backward() # Update gradients
        gen_opt.step() # Update optimizer

        # Keep track of the average discriminator loss
        mean_discriminator_loss += disc_A_loss.item() /
display_step
        # Keep track of the average generator loss
        mean_generator_loss += gen_loss.item() / display_step

        ### Visualization code ###
        if cur_step % display_step == 0:
            print(f"Epoch {epoch}: Step {cur_step}: Generator (U-
Net) loss: {mean_generator_loss}, Discriminator loss:
{mean_discriminator_loss}")
            show_tensor_images(torch.cat([real_A, real_B]),
size=(dim_A, target_shape, target_shape))
            show_tensor_images(torch.cat([fake_B, fake_A]),
size=(dim_B, target_shape, target_shape))
            mean_generator_loss = 0
            mean_discriminator_loss = 0
            cur_step += 1

        if save_model:
            torch.save(gen_AB.state_dict(), 'pth/gen_AB.pth')
            torch.save(gen_BA.state_dict(), 'pth/gen_BA.pth')
            torch.save(disc_A.state_dict(), 'pth/disc_A.pth')
            torch.save(disc_B.state_dict(), 'pth/disc_B.pth')

train(save_model=True)

```

Fungsi train yang telah dibuat adalah untuk melatih model menggunakan generator dan discriminator dalam sebuah siklus Generative Adversarial Network (GAN). Fungsi ini melibatkan langkah-langkah untuk melatih generator dan discriminator, menghitung loss, dan menyimpan model yang terlatih.

Penjelasan langkah-langkah dalam train:

1. Inisialisasi Variabel:

- **mean_generator_loss** dan **mean_discriminator_loss** digunakan untuk menghitung rata-rata loss selama pelatihan.
- **dataloader** mengatur batch input dari dataset menggunakan DataLoader.
- **cur_step** adalah penghitung langkah pelatihan.

2. Pelatihan Loop:

- Loop utama berisi iterasi untuk setiap epoch dan batch gambar.
- **Menyiapkan Gambar Asli:**

- real_A dan real_B adalah gambar dari domain A dan B.
- Gambar diubah ukurannya dengan nn.functional.interpolate.
- Kedua gambar kemudian dipindahkan ke device (misalnya, GPU).

3. Update Discriminator A:

- Discriminator A (disc_A) digunakan untuk memeriksa apakah gambar asli (real_A) dan gambar palsu (fake_A) dari generator gen_BA valid atau tidak.
- disc_A_loss dihitung dengan memanggil fungsi get_disc_loss.
- Gradien dihitung dan optimizer disc_A_opt diperbarui.

4. Update Discriminator B:

- Proses serupa dilakukan untuk discriminator B (disc_B), dengan gambar palsu (fake_B) dihasilkan dari generator gen_AB.
- disc_B_loss dihitung, dan optimizer disc_B_opt diperbarui.

5. Update Generator:

- Generator (gen_AB dan gen_BA) berusaha menghasilkan gambar yang lebih baik.
- Fungsi get_gen_loss menghitung total loss dari generator, termasuk loss adversarial, identity loss, dan cycle consistency loss.
- Gradien dihitung dan optimizer gen_opt diperbarui.

6. Mencatat Rata-rata Loss:

- Rata-rata generator dan discriminator loss dicatat untuk pemantauan pelatihan.

7. Visualisasi Gambar:

- Setiap beberapa langkah (display_step), gambar asli (real_A, real_B) dan gambar yang dihasilkan (fake_B, fake_A) ditampilkan menggunakan fungsi show_tensor_images.

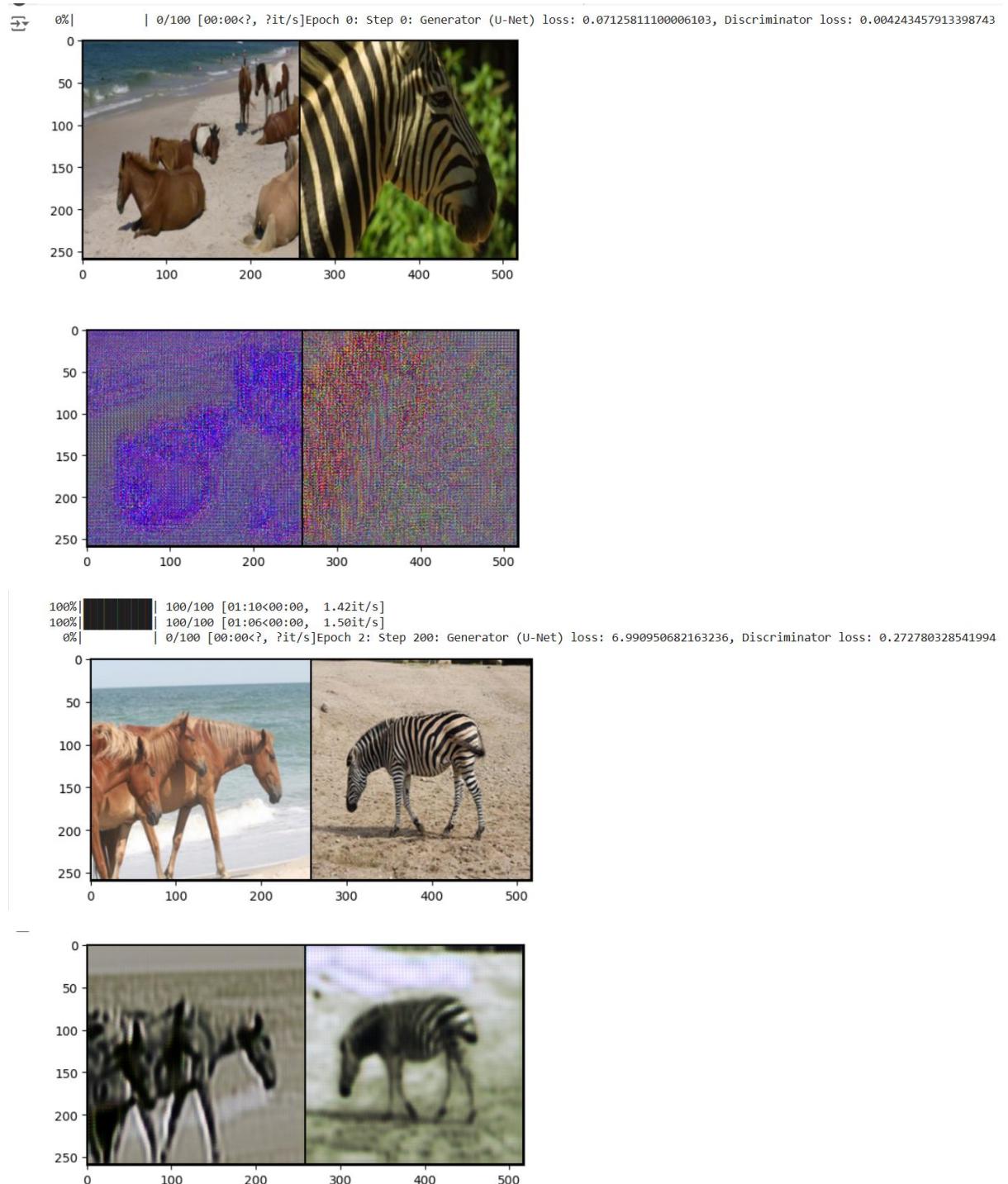
8. Menyimpan Model (Jika Diperlukan):

- Jika `save_model=True`, model-generator dan model-discriminator disimpan ke file .pth dalam direktori pth/.

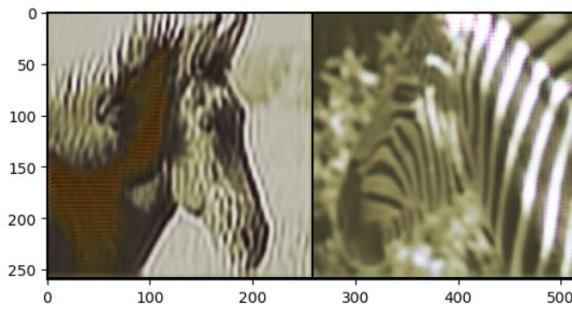
Penggunaan:

- Fungsi `train(save_model=True)` akan melatih model dan menyimpan model yang terlatih ke dalam direktori pth/ setiap kali pelatihan selesai.

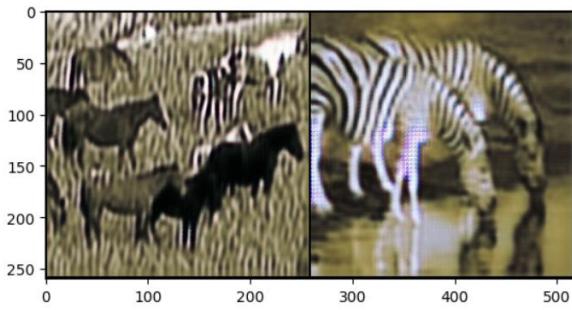
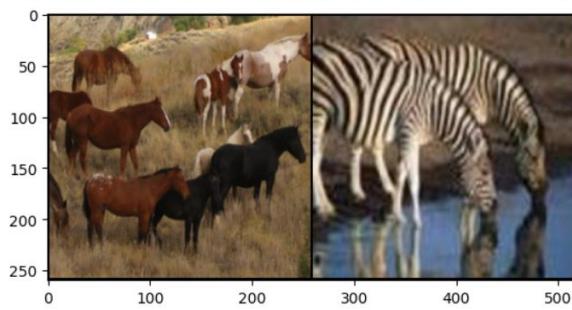
Berikut hasilnya:



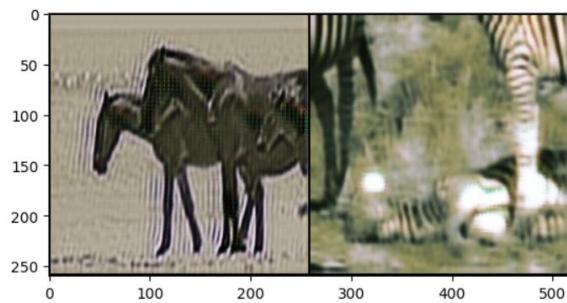
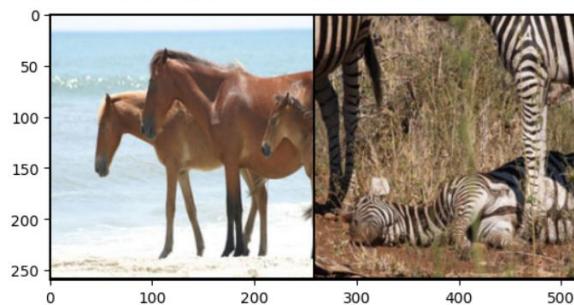
100% | [██████] 100/100 [01:06<00:00, 1.50it/s]
100% | [██████] 100/100 [01:06<00:00, 1.51it/s]
0% | [] 0/100 [00:00<?, ?it/s] Epoch 4: Step 400: Generator (U-Net) loss: 6.144492532014849, Discriminator loss: 0.21732861246913657



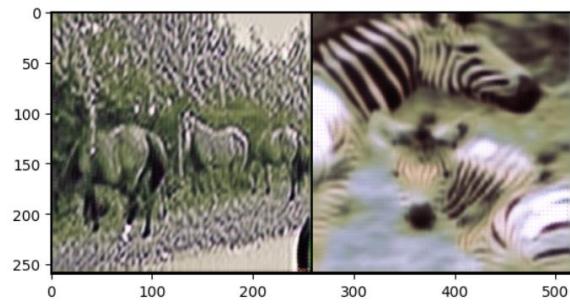
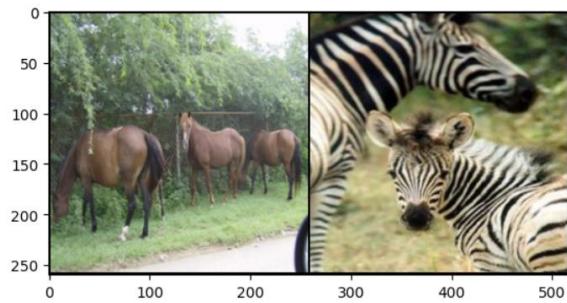
100% | [██████] 100/100 [01:06<00:00, 1.49it/s]
100% | [██████] 100/100 [01:06<00:00, 1.51it/s]
0% | [] 0/100 [00:00<?, ?it/s] Epoch 6: Step 600: Generator (U-Net) loss: 5.761960906982421, Discriminator loss: 0.21412100818008192



100% | 100/100 [01:07<00:00, 1.49it/s]
100% | 100/100 [01:06<00:00, 1.51it/s]
0% | 0/100 [00:00<?, ?it/s] Epoch 8: Step 800: Generator (U-Net) loss: 5.566163277626036, Discriminator loss: 0.22138562534004458

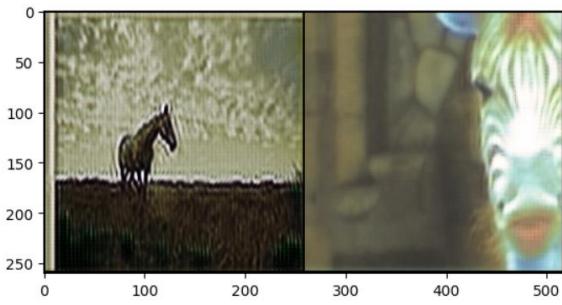


100% | 100/100 [01:07<00:00, 1.49it/s]
100% | 100/100 [01:06<00:00, 1.51it/s]
0% | 0/100 [00:00<?, ?it/s] Epoch 10: Step 1000: Generator (U-Net) loss: 5.6012214601039885, Discriminator loss: 0.21545078881084925

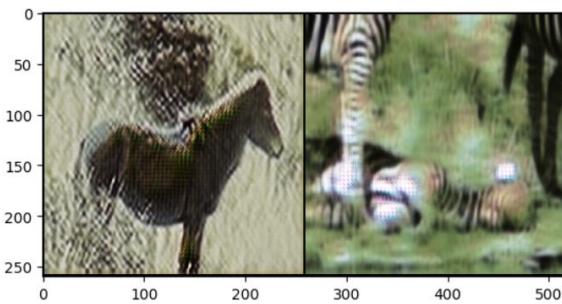
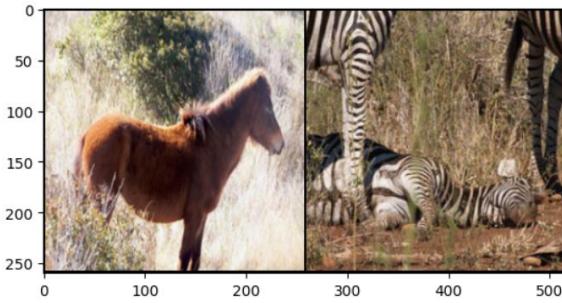


100% | 100/100 [01:07<00:00, 1.49it/s]
100% | 100/100 [01:06<00:00, 1.50it/s]
0% | 0/100 [00:00<?, ?it/s] Epoch 12: Step 1200: Generator (U-Net) loss: 5.4506350457668304, Discriminator loss: 0.2244955603405835

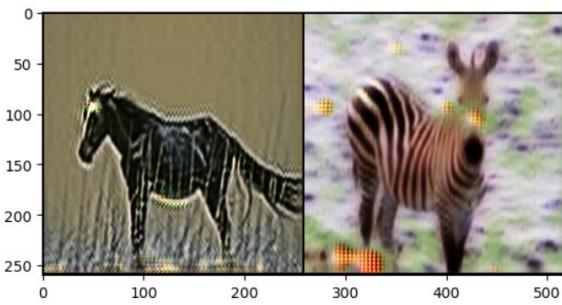
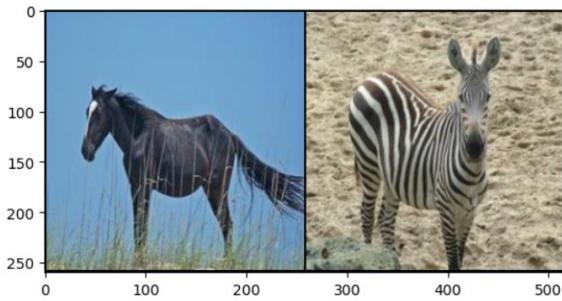




100% | 100/100 [01:07<00:00, 1.49it/s]
100% | 100/100 [01:06<00:00, 1.50it/s]
0% | 0/100 [00:00<?, ?it/s] Epoch 14: Step 1400: Generator (U-Net) loss: 5.211579842567447, Discriminator loss: 0.22180028602480883



100% | 100/100 [01:07<00:00, 1.49it/s]
100% | 100/100 [01:06<00:00, 1.50it/s]
0% | 0/100 [00:00<?, ?it/s] Epoch 16: Step 1600: Generator (U-Net) loss: 5.221450967788697, Discriminator loss: 0.22644660234451297





Upload model ke Huggingface:

```
# Periksa apakah file .pth disimpan di direktori /pth
!ls pth
→ disc_A.pth disc_B.pth gen_AB.pth gen_BA.pth
```

Perintah tersebut akan menampilkan daftar isi dari direktori pth untuk memeriksa apakah ada file .pth di sana.

```
!huggingface-cli login

from huggingface_hub import HfApi, create_repo
api = HfApi()

# Create a Huggingface model repo and use the name here
repo_name = "khaliishah/huggingface_khalishah"

create_repo(repo_name)

api.upload_folder(
    folder_path="pth",
    repo_id=repo_name,
    repo_type="model",
)
```

```

A token is already saved on your machine. Run `huggingface-cli whoami` to get more information or `huggingface-cli logout` if you want to log out.
Setting a new token will erase the existing one.
To log in to "huggingface_hub" requires a token generated from https://huggingface.co/settings/tokens.
Enter a token or token file path (optional):
Add token as git credential? (Y/n) Y
Token is valid (permission: fineGrained).
The token "huggingface" has been saved to /root/.cache/huggingface/stored_tokens
Cannot authenticate through git-credential as no helper is defined on your machine.
You might have to re-authenticate when pushing to the Hugging Face Hub.
Run the following command in your terminal in case you want to set the "store" credential helper as default.

git config --global credential.helper store
Read https://git-scm.com/book/en/v2/Git-Tools/Credential-Storage for more details.
Token has not been saved to git credential helper.
Your token has been saved to /root/.cache/huggingface/token
Login successful.
The current active token is: 'huggingface'

disc_B.pth: 100% [██████████] 11.1M/11.1M [00:00<00:00, 19.5MB/s]
disc_A.pth: 100% [██████████] 11.1M/11.1M [00:00<00:00, 24.0MB/s]
gen_AB.pth: 100% [██████████] 45.5M/45.5M [00:02<00:00, 22.7MB/s]
gen_BA.pth: 100% [██████████] 45.5M/45.5M [00:02<00:00, 24.8MB/s]

Upload 4 LFS files: 100% [██████████] 414 [00:02<00:00, 1.70MB/s]
CommitInfo(commit_url="https://huggingface.co/khalilishah/huggingface_khalishah/commit/495d93c2de677b5810038db1e931b6b3b4ef058", commit_message="Upload folder using huggingface_hub", commit_description='', old_id='495d93c2de677b5810038db1e931b6b3b4ef058', pr_url=None, repo_url=RepoURL("https://huggingface.co/khalilishah/huggingface_khalishah", endpoint="https://huggingface.co", repo_type='model', repo_id='khalilishah/huggingface_khalishah'), pr_revision=None, pr_num=None)

```

Kode di atas bertujuan untuk mengunggah folder ke repositori model di Hugging Face Hub. Berikut adalah penjelasan dari setiap bagian:

1. !huggingface-cli login:

- Perintah ini digunakan untuk login ke Hugging Face menggunakan token API Anda.
- Anda akan diminta untuk memasukkan token yang didapat dari akun Hugging Face Anda untuk mengautentikasi diri sebelum melakukan operasi lainnya di Hugging Face Hub.

2. from huggerface_hub import HfApi, create_repo:

- Mengimpor kelas dan fungsi dari pustaka huggerface_hub.
- HfApi memungkinkan Anda berinteraksi dengan API Hugging Face untuk berbagai operasi, seperti mengelola repositori dan mengunggah file.
- create_repo digunakan untuk membuat repositori baru di Hugging Face Hub.

3. api = HfApi():

- Membuat objek HfApi, yang memungkinkan Anda berinteraksi dengan Hugging Face API menggunakan metode yang disediakan.

4. repo_name = "khalilishah/huggingface_khalishah":

- Menetapkan nama repositori yang akan dibuat. Formatnya adalah username/nama_repositori.

5. create_repo(repo_name):

- Membuat repositori baru di Hugging Face Hub dengan nama yang sudah ditetapkan (repo_name).
- Secara default, repositori ini akan diset sebagai repositori model, dan Anda dapat mengunggah file ke dalamnya.

6. api.upload_folder(folder_path="pth", repo_id=repo_name, repo_type="model"):

- Fungsi ini mengunggah seluruh isi folder pth ke repositori yang baru dibuat di Hugging Face Hub.

- folder_path="pth" menunjukkan folder yang akan diunggah.
- repo_id=repo_name menentukan repositori yang menjadi tujuan pengunggahan.
- repo_type="model" menunjukkan bahwa repositori ini adalah repositori model, yang berarti repositori tersebut akan digunakan untuk menyimpan model machine learning, dan file yang diunggah akan dianggap sebagai bagian dari model tersebut.

2. Fine tune SAM (Segment Anything Model) on Custom Dataset (Chapter 6)

Fine-tuning SAM (Segment Anything Model) pada dataset kustom melibatkan pelatihan ulang model segmentasi gambar yang telah dilatih sebelumnya dengan dataset spesifik, guna meningkatkan kinerjanya dalam mengenali objek atau elemen baru yang tidak ada dalam data pelatihan asli. Proses ini dimulai dengan mempersiapkan dataset kustom yang berisi gambar yang sudah diberi anotasi, lalu melanjutkan dengan mengadaptasi model SAM menggunakan teknik transfer learning. Hal ini memungkinkan model untuk memanfaatkan pengetahuan dari dataset umum dan menerapkannya pada tugas spesifik[2].

Mulai dengan menginstall transformers dan datasets:

```
!pip install -q git+https://github.com/huggingface/transformers.git
→ Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Building wheel for transformers (pyproject.toml) ... done
```

```
!pip install -q datasets
→ ━━━━━━━━━━━━━━━━ 480.6/480.6 kB 30.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━ 116.3/116.3 kB 11.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━ 179.3/179.3 kB 17.6 MB/s eta 0:00:00
   ━━━━━━━━━━ 134.8/134.8 kB 13.5 MB/s eta 0:00:00
   ━━━━ 194.1/194.1 kB 17.2 MB/s eta 0:00:00
```

Load dataset:

```
from datasets import load_dataset

dataset = load_dataset("hf-vision/road-pothole-segmentation")
```

Kode di atas digunakan untuk memuat dataset dari Hugging Face Hub menggunakan pustaka datasets:

1. **from datasets import load_dataset:**

- Mengimpor fungsi load_dataset dari pustaka datasets. Pustaka ini memungkinkan Anda untuk memuat berbagai dataset yang tersedia di Hugging Face Hub secara langsung ke dalam kode Python Anda.

2. **dataset=load_dataset("hf-vision/road-pothole-segmentation"):**

- Fungsi `load_dataset` digunakan untuk mengunduh dan memuat dataset yang ada di Hugging Face Hub.
- Dalam hal ini, "hf-vision/road-pothole-segmentation" adalah ID dari dataset yang ingin dimuat. Dataset ini berfokus pada segmentasi gambar untuk mendeteksi lubang jalan (potholes) pada gambar yang diambil dari lingkungan jalan.
- Dataset yang dimuat akan disimpan dalam variabel `dataset` yang berupa objek dataset yang dapat digunakan untuk berbagai tujuan, seperti pelatihan model, analisis data, atau pengujian.

Dengan kode ini, Anda bisa mengakses dataset "road-pothole-segmentation" yang telah tersedia di Hugging Face Hub, yang berisi gambar dengan label segmentasi untuk mendeteksi lubang jalan.

Melihat struktur dari dataset:

```
dataset
└── DatasetDict({
    train: Dataset({
        features: ['image', 'label'],
        num_rows: 79
    })
    validation: Dataset({
        features: ['image', 'label'],
        num_rows: 5
    })
})
```

Output menunjukkan struktur dari dataset yang dimuat, dalam bentuk `DatasetDict`, yang berisi dua bagian utama: `train` dan `validation`. Berikut adalah penjelasan detailnya:

1. **DatasetDict:**

- Ini adalah struktur data yang menyimpan beberapa subset dataset (seperti `train`, `validation`, dan lain-lain). Setiap subset adalah objek `Dataset` yang berisi data yang sesuai.

2. **train: Dataset({ features: ['image', 'label'], num_rows: 79 }):**

- Bagian `train` adalah dataset pelatihan yang berisi 79 entri.
- **features** adalah kolom dalam dataset, yang dalam hal ini terdiri dari dua kolom:
 - `image`: Kolom ini berisi gambar untuk pelatihan.
 - `label`: Kolom ini berisi label segmentasi untuk setiap gambar, yaitu area yang menunjukkan lubang jalan (potholes).
- **num_rows: 79** menunjukkan jumlah gambar dan label yang ada dalam dataset pelatihan.

3. **validation: Dataset({ features: ['image', 'label'], num_rows: 5 }):**

- Bagian validation adalah dataset validasi yang berisi 5 entri. Dataset validasi digunakan untuk menguji kinerja model pada data yang tidak dilihat selama pelatihan.
- Struktur fitur dan jumlah baris sama dengan dataset pelatihan: dua fitur (gambar dan label) dan jumlah barisnya adalah 5.

Secara keseluruhan, dataset ini berfokus pada segmentasi gambar untuk mendeteksi lubang jalan, dengan data pelatihan yang cukup besar (79 gambar) dan data validasi yang kecil (5 gambar).

Visualisasi:

```
example = dataset["train"][1]
example
image = example["image"]
image
```

Kode ini menampilkan cara mengambil data dari dataset yang dimuat sebelumnya dan mengekstrak gambar dari data tersebut. Berikut adalah penjelasan rinci:

1. example = dataset["train"][1]:

- dataset["train"] mengakses subset train dari dataset yang dimuat, yang berisi data pelatihan.
- [1] adalah indeks yang menunjukkan bahwa Anda mengambil entri ke-1 (indeks dimulai dari 0) dari subset train. Ini berarti Anda mengambil contoh kedua dalam dataset pelatihan.
- example adalah variabel yang menyimpan data dari contoh ke-1, yang berisi dua fitur: gambar (image) dan label (label).

2. example:

- Setelah baris kode ini, variabel example berisi dictionary yang terdiri dari dua kunci:
 - "image": Gambar yang terkait dengan data pelatihan ini.
 - "label": Label segmentasi yang menunjukkan area lubang jalan pada gambar tersebut.

3. image = example["image"]:

- Di sini, Anda mengambil nilai dari kunci "image" dalam dictionary example dan menyimpannya dalam variabel image.
- image sekarang berisi data gambar yang dapat digunakan untuk pemrosesan atau pelatihan lebih lanjut.

4. image:

- Ketika Anda mengeksekusi `image`, itu akan menampilkan objek gambar, yang biasanya berupa objek PIL Image atau array NumPy, tergantung pada bagaimana dataset diproses.

Berikut hasilnya:



Menampilkan gambar overlay:

```
import matplotlib.pyplot as plt
import numpy as np

def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])],
axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

fig, axes = plt.subplots()

axes.imshow(np.array(image))
ground_truth_seg = np.array(example["label"])
show_mask(ground_truth_seg, axes)
axes.title.set_text(f"Ground truth mask")
axes.axis("off")
```

Kode ini digunakan untuk menampilkan gambar dengan overlay mask segmentasi pada gambar tersebut menggunakan matplotlib. Mask segmentasi ini menunjukkan area yang relevan (misalnya, lubang jalan pada dataset "road-pothole-segmentation"). Berikut adalah penjelasan rinci dari setiap bagian kode:

1. import matplotlib.pyplot as plt dan import numpy as np:

- Mengimpor pustaka matplotlib.pyplot untuk menampilkan gambar dan membuat plot.
- Mengimpor pustaka numpy untuk manipulasi array, yang berguna untuk pengolahan gambar dan data numerik.

2. def show_mask(mask, ax, random_color=False):

- Mendefinisikan fungsi show_mask yang menerima tiga argumen:
 - mask: Mask segmentasi yang akan diterapkan ke gambar.
 - ax: Objek axes matplotlib tempat gambar dan mask akan ditampilkan.
 - random_color: Boolean yang menentukan apakah mask akan diberi warna acak atau warna tetap.
- Fungsi ini digunakan untuk menampilkan mask pada gambar, dengan pilihan warna acak atau warna tetap.

3. if random_color:

- Jika random_color bernilai True, maka warna mask akan ditentukan secara acak dengan menghasilkan array warna RGB acak (np.random.random(3)) dan transparansi 0.6 (np.array([0.6])).
- Jika random_color bernilai False, warna mask akan ditetapkan ke nilai tetap, yaitu warna biru semi-transparan dengan kode RGB [30/255, 144/255, 255/255, 0.6].

4. h, w = mask.shape[-2:]:

- Mengambil dimensi tinggi (h) dan lebar (w) dari mask, yang disimpan dalam variabel mask. Mask ini biasanya berupa array 2D, dan .shape[-2:] mengambil dimensi dua terakhir (tinggi dan lebar).

5. mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1):

- Merubah mask menjadi array dengan bentuk (h, w, 1) agar bisa digabungkan dengan warna yang dipilih (color). Kemudian, mask ini digabungkan dengan warna (ditingkatkan transparansi) untuk menghasilkan gambar mask yang sesuai dengan ukuran gambar asli.

6. ax.imshow(mask_image):

- Menampilkan gambar mask pada ax (axes matplotlib), di atas gambar utama. Mask akan diposisikan dengan transparansi yang ditentukan sebelumnya.

7. fig, axes = plt.subplots():

- Membuat objek fig dan axes untuk plot matplotlib. Di sini hanya satu sub-plot yang dibuat, tetapi bisa diperluas untuk beberapa plot jika diperlukan.

8. axes.imshow(np.array(image)):

- Menampilkan gambar yang diambil dari dataset pada objek axes. Gambar tersebut akan ditampilkan pada plot.

9. `ground_truth_seg = np.array(example["label"]):`

- Mengambil mask segmentasi ground truth yang sesuai dengan gambar (`example["label"]`), yang menunjukkan area lubang jalan pada gambar. Mask ini akan digunakan untuk overlay pada gambar.

10. `show_mask(ground_truth_seg, axes):`

- Memanggil fungsi `show_mask` untuk menampilkan mask ground truth pada gambar.

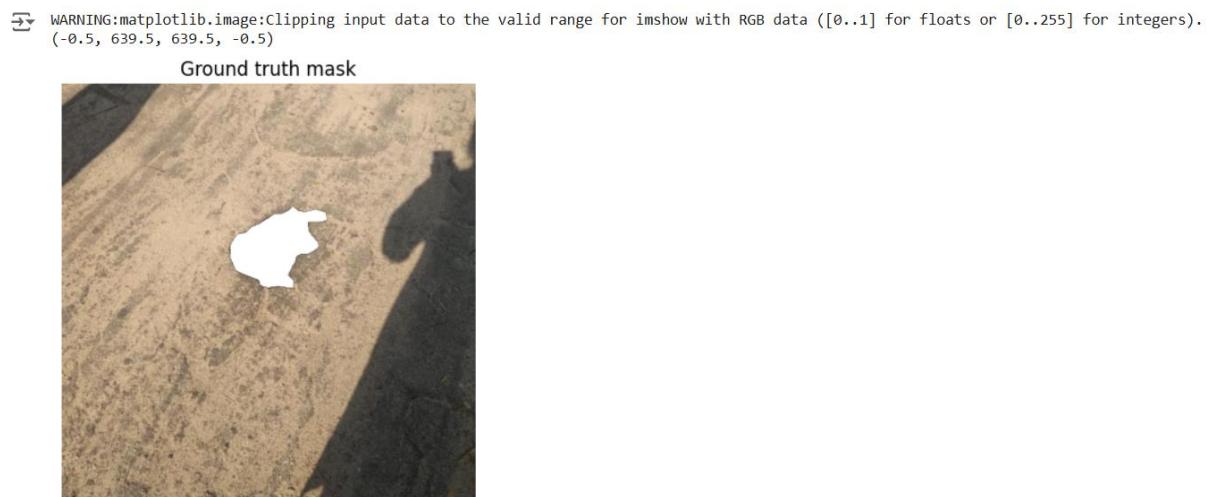
11. `axes.title.set_text(f"Ground truth mask"):`

- Menetapkan judul pada plot dengan teks "Ground truth mask", yang menjelaskan bahwa gambar tersebut menampilkan ground truth mask.

12. `axes.axis("off"):`

- Menghilangkan sumbu (axis) di sekitar gambar, sehingga hanya gambar yang ditampilkan tanpa label atau angka sumbu.

Berikut hasilnya:



Menentukan area objek yang tersegmentasi:

```
def get_bounding_box(ground_truth_map):
    # mendapatkan kotak pembatas dari masker
    y_indices, x_indices = np.where(ground_truth_map > 0)
    x_min, x_max = np.min(x_indices), np.max(x_indices)
    y_min, y_max = np.min(y_indices), np.max(y_indices)

    # menambahkan gangguan pada koordinat kotak pembatas
    H, W = ground_truth_map.shape
    x_min = max(0, x_min - np.random.randint(0, 20))
```

```

x_max = min(W, x_max + np.random.randint(0, 20))
y_min = max(0, y_min - np.random.randint(0, 20))
y_max = min(H, y_max + np.random.randint(0, 20))
bbox = [x_min, y_min, x_max, y_max]

return bbox

```

Fungsi `get_bounding_box` digunakan untuk menghitung dan mengembalikan koordinat kotak pembatas (bounding box) yang membungkus area yang diberi label pada `ground_truth_map`. Berikut adalah penjelasan dari setiap bagian kode:

1. `y_indices, x_indices = np.where(ground_truth_map > 0):`

- `np.where(ground_truth_map > 0)` mengembalikan indeks piksel yang memiliki nilai lebih besar dari 0 dalam `ground_truth_map`. Biasanya, nilai ini menunjukkan area yang relevan (misalnya, area objek yang tersegmentasi) dalam peta segmentasi.
- `y_indices` dan `x_indices` menyimpan indeks baris dan kolom dari piksel yang termasuk dalam area tersebut.

2. `x_min, x_max = np.min(x_indices), np.max(x_indices):`

- Menentukan batas kiri dan kanan dari bounding box dengan mengambil nilai minimum dan maksimum dari `x_indices`, yaitu kolom di mana objek dimulai dan berakhir.

3. `y_min, y_max = np.min(y_indices), np.max(y_indices):`

- Menentukan batas atas dan bawah dari bounding box dengan mengambil nilai minimum dan maksimum dari `y_indices`, yaitu baris di mana objek dimulai dan berakhir.

4. Menambahkan gangguan pada koordinat kotak pembatas:

- Gangguan acak ditambahkan ke koordinat kotak pembatas untuk sedikit memodifikasi posisi bounding box. Ini berguna untuk meningkatkan keragaman data atau untuk tujuan augmentasi.
- `np.random.randint(0, 20)` menghasilkan nilai acak antara 0 dan 20 yang ditambahkan atau dikurangi dari koordinat kotak pembatas.
- Fungsi `max(0, x_min - np.random.randint(0, 20))` memastikan bahwa koordinat tidak keluar dari batas gambar (misalnya, tidak akan menjadi negatif).

5. `H, W = ground_truth_map.shape:`

- Mendapatkan dimensi gambar (`H` adalah tinggi, `W` adalah lebar) dari `ground_truth_map` untuk memastikan bahwa koordinat yang dihasilkan tetap berada dalam batas-batas gambar.

6. `bbox = [x_min, y_min, x_max, y_max]:`

- Menyusun kotak pembatas yang dihitung sebagai list `[x_min, y_min, x_max, y_max]`, yang mewakili posisi kiri atas dan kanan bawah dari kotak pembatas.

7. return bbox:

- Mengembalikan hasil kotak pembatas (bounding box) dalam format list yang berisi koordinatnya.

Menampilkan kotak pembatas:

```
import matplotlib.pyplot as plt

def show_box(box, ax):
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green',
    facecolor=(0,0,0,0), lw=2))

def show_boxes_on_image(raw_image, boxes):
    plt.figure(figsize=(10,10))
    plt.imshow(raw_image)
    for box in boxes:
        show_box(box, plt.gca())
    plt.axis('on')
    plt.show()
```

Kode ini digunakan untuk menampilkan kotak pembatas (bounding box) pada gambar menggunakan matplotlib. Berikut adalah penjelasan rinci untuk setiap bagianya:

1. show_box(box, ax):

- Fungsi ini digunakan untuk menggambar satu kotak pembatas pada gambar.
- **box**: Merupakan list atau tuple yang berisi koordinat kotak pembatas, yaitu [x_min, y_min, x_max, y_max].
- **ax**: Merupakan objek axes (sumbu) dari matplotlib tempat kotak akan digambar.
- **x0, y0**: Menentukan titik kiri atas dari kotak, diambil dari koordinat box[0] (x_min) dan box[1] (y_min).
- **w, h**: Menghitung lebar (w) dan tinggi (h) kotak dengan selisih antara x_max dan x_min serta y_max dan y_min dari box.
- **ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green', facecolor=(0,0,0,0), lw=2))**:
 - Menambahkan objek kotak (Rectangle) pada axes ax dengan posisi (x0, y0) sebagai titik kiri atas, dan ukuran (w, h) sebagai lebar dan tinggi kotak.
 - **edgecolor='green'**: Menentukan warna garis kotak sebagai hijau.
 - **facecolor=(0,0,0,0)**: Menentukan warna dalam kotak sebagai transparan (tidak ada warna isian).
 - **lw=2**: Menetapkan ketebalan garis kotak sebagai 2 piksel.

2. `show_boxes_on_image(raw_image, boxes)`:

- Fungsi ini digunakan untuk menampilkan gambar dengan beberapa kotak pembatas yang digambar di atasnya.
- **raw_image**: Gambar asli yang akan ditampilkan.
- **boxes**: Daftar yang berisi beberapa kotak pembatas yang ingin digambar di atas gambar.
- `plt.figure(figsize=(10,10))`: Membuat figur atau plot dengan ukuran 10x10 inci.
- `plt.imshow(raw_image)`: Menampilkan gambar asli raw_image pada plot.
- **for box in boxes::** Iterasi melalui setiap kotak dalam daftar boxes.
 - `show_box(box, plt.gca())`: Memanggil fungsi show_box untuk menggambar kotak pada axes saat ini (`plt.gca()` mengembalikan objek axes saat ini).
- `plt.axis('on')`: Menampilkan sumbu pada plot.
- `plt.show()`: Menampilkan plot dengan gambar dan kotak pembatas yang digambar di atasnya.

```
input_boxes = get_bounding_box(ground_truth_seg)
input_boxes
show_boxes_on_image(image, [input_boxes])
```

Kode ini digunakan untuk mendapatkan kotak pembatas (bounding box) dari sebuah mask segmentasi dan kemudian menampilkannya di atas gambar. Berikut adalah penjelasan rinci:

1. `input_boxes = get_bounding_box(ground_truth_seg)`:

- Fungsi `get_bounding_box(ground_truth_seg)` dipanggil untuk mendapatkan koordinat kotak pembatas berdasarkan peta segmentasi (`ground_truth_seg`).
- `ground_truth_seg` adalah mask segmentasi yang menunjukkan area objek yang relevan (misalnya, area lubang jalan).
- Fungsi `get_bounding_box` akan mengembalikan sebuah kotak pembatas dalam format `[x_min, y_min, x_max, y_max]`, yang disimpan dalam variabel `input_boxes`.

2. `input_boxes`:

- Variabel `input_boxes` berisi kotak pembatas yang dihasilkan dari fungsi `get_bounding_box`. Ini adalah daftar dengan satu kotak pembatas yang menunjukkan posisi objek dalam gambar.

3. `show_boxes_on_image(image, [input_boxes])`:

- Fungsi `show_boxes_on_image` dipanggil untuk menampilkan gambar (`image`) dengan kotak pembatas yang ada di dalam `input_boxes`.
- **image**: Gambar yang akan ditampilkan.
- **[input_boxes]**: Daftar yang berisi satu kotak pembatas yang akan digambar di atas gambar. Dalam hal ini, `input_boxes` adalah kotak pembatas yang diperoleh sebelumnya.

- Fungsi ini akan menampilkan gambar dengan kotak pembatas yang digambar di atas area yang relevan berdasarkan hasil segmentasi (misalnya, area lubang jalan).

Berikut hasilnya:



Membuat dataset khusus untuk tugas segmentasi berbasis model SAM (Segment Anything Model):

```
from torch.utils.data import Dataset
import cv2

class SAMDataset(Dataset):
    def __init__(self, dataset, processor):
        self.dataset = dataset
        self.processor = processor

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        item = self.dataset[idx]
        image = item["image"]
        ground_truth_mask = np.array(item["label"])

        # menormalkan masker
        ground_truth_mask = ground_truth_mask / 255.0

        # mendapatkan prompt kotak pembatas
        prompt = get_bounding_box(ground_truth_mask)

        # mempersiapkan gambar dan prompt untuk model
        inputs = self.processor(image, input_boxes=[[prompt]],
                               return_tensors="pt")
```

```

# menghapus dimensi batch yang ditambahkan oleh pemroses secara
default
inputs = {k:v.squeeze(0) for k,v in inputs.items()}

# menambahkan segmentasi kebenaran dasar
inputs["ground_truth_mask"] = ground_truth_mask

return inputs

```

Kode ini mendefinisikan kelas SAMDataset yang merupakan subclass dari torch.utils.data.Dataset. Kelas ini bertujuan untuk membuat dataset khusus untuk tugas segmentasi berbasis model SAM (Segment Anything Model). Berikut adalah penjelasan rinci dari setiap bagian kode:

1. class SAMDataset(Dataset):

- Mendefinisikan kelas SAMDataset yang merupakan subclass dari torch.utils.data.Dataset, yang memungkinkan dataset ini digunakan dalam pipeline pelatihan PyTorch.
- Dataset ini dioptimalkan untuk tugas segmentasi gambar dengan memberikan gambar dan masker kebenaran dasar serta kotak pembatas (bounding box) sebagai prompt ke model.

2. def __init__(self, dataset, processor):

- Konstruktor yang menginisialisasi kelas dengan dua parameter:
 - **dataset:** Dataset input yang berisi gambar dan label (masker).
 - **processor:** Prosesor atau pemroses yang digunakan untuk menyiapkan gambar dan kotak pembatas untuk model. Biasanya ini berupa tokenizer atau model preprocessor.

3. def __len__(self):

- Fungsi ini mengembalikan panjang dataset, yaitu jumlah sampel dalam self.dataset. Fungsi ini digunakan oleh PyTorch untuk mengetahui ukuran dataset.

4. def __getitem__(self, idx):

- Fungsi ini digunakan untuk mengambil item pada indeks idx dari dataset. Fungsi ini adalah tempat utama untuk mempersiapkan data untuk model.

Proses di dalam __getitem__:

- **item = self.dataset[idx]:** Mengambil contoh data (gambar dan label) pada indeks idx dari dataset.
- **image = item["image"]:** Mengambil gambar dari item dataset.
- **ground_truth_mask = np.array(item["label"]):** Mengambil masker kebenaran dasar yang berisi segmentasi untuk gambar dan mengonversinya menjadi array NumPy.

Normalisasi Masker:

- **ground_truth_mask = ground_truth_mask / 255.0:** Menormalkan masker dengan membagi setiap elemen dengan 255.0, sehingga nilainya berada dalam rentang [0, 1].

Mengambil Kotak Pembatas (Bounding Box):

- **prompt=get_bounding_box(ground_truth_mask):** Memanggil fungsi get_bounding_box untuk menghitung kotak pembatas yang membungkus objek dalam masker. Kotak pembatas ini akan digunakan sebagai prompt untuk model.

Persiapan Gambar dan Prompt untuk Model:

- **inputs = self.processor(image, input_boxes=[[prompt]], return_tensors="pt"):** Memproses gambar dan kotak pembatas menggunakan self.processor. input_boxes=[[prompt]] menunjukkan bahwa kotak pembatas (bounding box) yang telah dihitung digunakan sebagai input prompt untuk model. return_tensors="pt" memastikan bahwa hasilnya dikembalikan dalam bentuk tensor PyTorch.

Menghapus Dimensi Batch:

- **inputs = {k:v.squeeze(0) for k,v in inputs.items()}:** Menghapus dimensi batch yang ditambahkan oleh pemroses secara default (biasanya dimensi pertama dalam tensor). .squeeze(0) akan menghapus dimensi yang memiliki ukuran 1 pada posisi pertama.

Menambahkan Masker Kebenaran Dasar:

- **inputs["ground_truth_mask"] = ground_truth_mask:** Menambahkan masker kebenaran dasar ke dictionary inputs yang akan dikembalikan, sehingga model memiliki akses ke informasi segmentasi ground truth.

5. return inputs

- Mengembalikan dictionary inputs yang berisi data yang telah diproses, termasuk gambar, kotak pembatas (prompt), dan masker kebenaran dasar untuk digunakan dalam pelatihan model.

Memuat pemroses dari model SAM (Segment Anything Model):

```
from transformers import SamProcessor

# Memuat pemroses dari SAM resmi
processor = SamProcessor.from_pretrained("facebook/sam-vit-base")
→ The cache for model files in Transformers v4.22.0 has been updated. Migrating your old cache. This is a one-time only operation.
  0/0 [00:00<?, ?B/s]
processor_config.json: 100% 466/466 [00:00<00:00, 33.2kB/s]
```

Kode ini digunakan untuk memuat pemroses (processor) dari model SAM (Segment Anything Model) yang telah dilatih sebelumnya, khususnya varian model dengan arsitektur Vision Transformer (ViT) yang disebut sam-vit-base.

Penjelasan Kode:

1. **from transformers import SamProcessor:**

- Mengimpor kelas SamProcessor dari pustaka transformers milik Hugging Face. Kelas ini digunakan untuk memproses gambar dan input terkait (seperti kotak pembatas atau prompt) untuk model SAM, termasuk konversi gambar menjadi format yang dapat diterima oleh model.

2. `processor = SamProcessor.from_pretrained("facebook/sam-vit-base"):`

- Memuat pemroses dari model yang telah dilatih sebelumnya (facebook/sam-vit-base) menggunakan metode from_pretrained.
- "facebook/sam-vit-base" adalah nama model yang telah dilatih oleh Meta (Facebook) dan tersedia di repositori Hugging Face. Ini adalah varian model SAM yang menggunakan arsitektur Vision Transformer (ViT).
- from_pretrained memuat pemroses yang telah dipraperlatihan, sehingga Anda dapat langsung menggunakannya untuk memproses data tanpa perlu melatihnya dari awal.

Fungsi SamProcessor:

- **SamProcessor** digunakan untuk mempersiapkan data untuk model SAM, seperti mengonversi gambar ke tensor, mengelola input tambahan (seperti kotak pembatas), dan memastikan data dalam format yang sesuai untuk pemrosesan lebih lanjut oleh model.

Membagi dataset menjadi pelatihan dan validasi:

```
# Membagi dataset menjadi pelatihan dan validasi
train_dataset = SAMDataset(dataset=dataset["train"],
processor=processor)
validation_dataset = SAMDataset(dataset=dataset["validation"],
processor=processor)
```

Kode ini digunakan untuk membagi dataset yang telah dimuat menjadi dua bagian: satu untuk pelatihan (training) dan satu untuk validasi (validation). Berikut adalah penjelasan rinci:

1. `train_dataset = SAMDataset(dataset=dataset["train"], processor=processor):`

- **SAMDataset**: Ini adalah kelas dataset yang telah Anda buat sebelumnya yang bertugas untuk memproses data gambar dan kotak pembatas serta menyiapkannya untuk digunakan dalam pelatihan model.
- **dataset["train"]**: Ini adalah bagian dataset yang digunakan untuk pelatihan. Dataset tersebut dipisahkan sebelumnya (misalnya, dari dataset yang lebih besar yang mencakup data pelatihan dan validasi).
- **processor=processor**: Pemroses yang telah dimuat sebelumnya menggunakan SamProcessor.from_pretrained("facebook/sam-vit-base"). Pemroses ini digunakan untuk mengonversi gambar dan data terkait menjadi format yang sesuai untuk model SAM.
- Hasilnya, train_dataset adalah objek SAMDataset yang berisi data pelatihan yang telah diproses dan siap digunakan untuk pelatihan model.

2. validation_dataset = SAMDataset(dataset=dataset["validation"], processor=processor):

- Proses yang sama dilakukan untuk validasi, tetapi kali ini menggunakan bagian validasi dari dataset.
- **dataset["validation"]:** Ini adalah bagian dataset yang digunakan untuk validasi model. Dataset ini digunakan untuk mengevaluasi kinerja model setelah pelatihan, membantu memantau overfitting dan bias model.

Periksa bentuk dari sampel dataset:

```
# Di sini kita memeriksa bentuk dari sampel dataset pelatihan
example = train_dataset[0]
for k,v in example.items() :
    print(k,v.shape)
    ↵ pixel_values torch.Size([3, 1024, 1024])
    ↵ original_sizes torch.Size([2])
    ↵ reshaped_input_sizes torch.Size([2])
    ↵ input_boxes torch.Size([1, 4])
    ↵ ground_truth_mask (640, 640)
/usr/local/lib/python3.10/dist-packages/transformers/image_processing_utils.py:41: UserWarning: The following named arguments are not valid for `SamImageProcessor.preprocess` and were
return self._preprocess(images, **kwargs)
```

Kode ini digunakan untuk memeriksa struktur dan bentuk (shape) dari data yang ada pada sampel pertama di train_dataset. Berikut adalah penjelasan rinci dari kode ini:

1. example = train_dataset[0]:

- Mengambil sampel pertama dari train_dataset dengan menggunakan indeks 0. train_dataset adalah objek SAMDataset yang telah diinisialisasi sebelumnya.
- Sampel yang diambil adalah sebuah dictionary yang berisi data yang telah diproses, termasuk gambar, kotak pembatas (bounding box), dan masker kebenaran dasar (ground truth mask) yang akan digunakan untuk melatih model.

2. for k, v in example.items():

- Melakukan iterasi melalui setiap item dalam example. Di sini, example adalah dictionary yang berisi berbagai elemen (seperti gambar, masker, dan kotak pembatas).
- **k:** Kunci (key) dari dictionary, yang bisa berupa nama elemen seperti 'image', 'input_boxes', atau 'ground_truth_mask'.
- **v:** Nilai (value) dari setiap kunci, yang bisa berupa tensor atau array dengan data gambar, kotak pembatas, atau masker.

3. print(k, v.shape):

- Menampilkan nama setiap elemen (key) dan bentuk (shape) dari nilai yang terkait dengan key tersebut.
- **v.shape:** Menampilkan bentuk dari tensor atau array v, yang memberikan informasi tentang dimensi dari data yang terkait. Misalnya, jika elemen tersebut adalah gambar, maka bentuknya mungkin berupa (C, H, W), di mana C adalah jumlah saluran (channels), H adalah tinggi (height), dan W adalah lebar (width) gambar.

Membuat DataLoader untuk dataset pelatihan:

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(train_dataset, batch_size=2,
shuffle=True)
```

Kode ini digunakan untuk membuat DataLoader untuk dataset pelatihan (train_dataset). DataLoader adalah bagian penting dalam pipeline pelatihan model di PyTorch, karena membantu memuat data dalam batch dan menyediakan opsi untuk pengacakan (shuffling) serta pemrosesan paralel. Berikut penjelasan rinci dari kode ini:

1. **from torch.utils.data import DataLoader:**

- Mengimpor kelas DataLoader dari pustaka torch.utils.data. DataLoader digunakan untuk mengelola cara data dimuat dan diberikan kepada model selama pelatihan atau inferensi.

2. **train_dataloader = DataLoader(train_dataset, batch_size=2, shuffle=True):**

- **train_dataset:** Dataset yang sudah diproses sebelumnya menggunakan kelas SAMDataset. Ini adalah dataset pelatihan yang akan dimuat oleh DataLoader.
- **batch_size=2:** Menentukan ukuran batch untuk DataLoader, yaitu berapa banyak sampel yang akan dimuat dalam satu iterasi selama pelatihan. Dalam hal ini, ukuran batch adalah 2, yang berarti 2 sampel dataset akan diproses pada satu waktu.
- **shuffle=True:** Menetapkan apakah data harus diacak sebelum dimuat dalam batch. shuffle=True memastikan bahwa urutan sampel akan diacak setiap kali DataLoader memulai satu epoch pelatihan baru. Ini membantu mengurangi kemungkinan model terjebak pada urutan tertentu dalam data dan meningkatkan generalisasi model.

Mengambil satu batch data dari train_dataloader:

```
batch = next(iter(train_dataloader))
for k,v in batch.items():
    print(k,v.shape)
    ↗ pixel_values torch.Size([2, 3, 1024, 1024])
    original_sizes torch.Size([2, 2])
    reshaped_input_sizes torch.Size([2, 2])
    input_boxes torch.Size([2, 1, 4])
    ground_truth_mask torch.Size([2, 640, 640])
```

Kode ini digunakan untuk mengambil satu batch data dari train_dataloader dan memeriksa bentuk (shape) dari setiap elemen dalam batch tersebut. Berikut penjelasan rinci dari kode ini:

1. **batch = next(iter(train_dataloader)):**

- **iter(train_dataloader):** Mengubah train_dataloader menjadi iterator, yang memungkinkan untuk mengambil elemen-elemen dari DataLoader secara berturut-turut.

- **next(...)**: Fungsi next() digunakan untuk mengambil satu batch data berikutnya dari iterator. Jadi, batch akan berisi satu batch data yang dimuat oleh train_dataloader.
- Batch yang diambil ini akan berisi beberapa sampel dari dataset, dalam hal ini 2 sampel karena batch_size=2 yang telah ditentukan sebelumnya.

2. **for k, v in batch.items():**

- Melakukan iterasi melalui setiap item dalam batch. batch adalah dictionary yang berisi data yang telah diproses, seperti gambar, kotak pembatas (bounding boxes), dan masker kebenaran dasar (ground truth masks).
- **k** adalah kunci (key) dari dictionary, yang mungkin berupa nama elemen seperti 'image', 'input_boxes', atau 'ground_truth_mask'.
- **v** adalah nilai (value) yang terkait dengan key, yang biasanya berupa tensor yang berisi data untuk elemen tersebut.

3. **print(k, v.shape):**

- Menampilkan nama setiap elemen (k) dalam batch dan bentuk (shape) dari tensor yang terkait (v).
- **v.shape** memberikan dimensi dari tensor, yang memberi informasi tentang struktur data. Misalnya, untuk gambar, bentuknya bisa berupa (C, H, W), di mana C adalah jumlah saluran (channels), H adalah tinggi (height) gambar, dan W adalah lebar (width). Jika ada batch lebih dari satu sampel, bentuknya mungkin berupa (batch_size, C, H, W).

Akses tensor:

```
batch["pixel_values"].shape
```

Kode batch["pixel_values"].shape digunakan untuk mengakses tensor yang berisi nilai piksel gambar dalam batch dan kemudian menampilkan bentuk (shape) tensor tersebut.

Penjelasan:

- **batch["pixel_values"]:**
 - batch adalah dictionary yang berisi elemen-elemen yang diproses, termasuk gambar, kotak pembatas, dan masker.
 - "pixel_values" adalah kunci dalam dictionary batch yang merujuk pada tensor yang berisi nilai-nilai piksel dari gambar yang telah diproses dan disiapkan untuk model. Ini biasanya berupa tensor 4 dimensi dengan bentuk (batch_size, C, H, W), di mana:
 - batch_size adalah jumlah sampel dalam batch (misalnya 2, karena batch_size=2).
 - C adalah jumlah saluran gambar (channels), yang biasanya 3 untuk gambar RGB.
 - H adalah tinggi gambar (height).

- W adalah lebar gambar (width).
- .shape:
 - Digunakan untuk memeriksa bentuk tensor. Ini akan memberikan dimensi tensor, yaitu ukuran untuk setiap sumbu.

Periksa bentuk:

```
batch["ground_truth_mask"].shape
```

Kode batch["ground_truth_mask"].shape digunakan untuk memeriksa bentuk (shape) dari masker kebenaran dasar (ground truth mask) yang terdapat dalam batch. Masker ini adalah tensor yang mewakili segmentasi benar (label) untuk gambar dalam batch.

Penjelasan lebih lanjut:

- batch["ground_truth_mask"]:
 - Ini adalah bagian dari dictionary batch, yang berisi masker segmentasi yang sesuai dengan gambar-gambar dalam batch.
 - Ground truth mask adalah data yang digunakan untuk melatih model segmentasi, di mana nilai-nilai dalam tensor ini menunjukkan area objek atau latar belakang pada gambar.
- .shape:
 - Fungsi ini digunakan untuk mendapatkan dimensi tensor. Bentuk masker ini biasanya akan memiliki dimensi yang sama dengan gambar input, tetapi dengan satu saluran (grayscale) untuk menunjukkan apakah suatu piksel termasuk dalam objek yang relevan atau bukan.

Memuat model:

```
from transformers import SamModel

model = SamModel.from_pretrained("facebook/sam-vit-base")

# pastikan kita hanya menghitung gradien untuk dekoder masker
for name, param in model.named_parameters():
    if name.startswith("vision_encoder") or
name.startswith("prompt_encoder"):
        param.requires_grad_(False)
→ config.json: 100% [6.57k/6.57k, 00:00<00:00, 414kB/s]
model.safetensors: 100% [375M/375M, 00:01<00:00, 246MB/s]
```

Kode ini digunakan untuk memuat model SAM (Segment Anything Model) dari Hugging Face, khususnya model dengan arsitektur Vision Transformer (ViT), dan kemudian mengatur agar hanya bagian tertentu dari model yang menghitung gradien selama pelatihan. Berikut penjelasan rinci dari kode ini:

1. from transformers import SamModel:

- Mengimpor kelas SamModel dari pustaka transformers yang berisi model SAM yang telah dilatih sebelumnya dan siap digunakan.

2. model = SamModel.from_pretrained("facebook/sam-vit-base"):

- Memuat model SAM dengan arsitektur Vision Transformer (ViT) yang telah dilatih sebelumnya dari repositori Hugging Face.
- "facebook/sam-vit-base" adalah nama model yang telah dilatih oleh Facebook dan tersedia di Hugging Face Hub.

3. for name, param in model.named_parameters():

- Melakukan iterasi terhadap setiap parameter (berat) dalam model. Fungsi named_parameters() mengembalikan nama parameter dan nilai (tensor) parameter tersebut.
- **name** adalah nama parameter (misalnya, nama layer dalam model).
- **param** adalah tensor parameter (berat) yang terkait dengan nama tersebut.

4. if name.startswith("vision_encoder") or name.startswith("prompt_encoder")::

- Menyaring parameter-parameter berdasarkan nama layer. Pada bagian ini, kode memeriksa apakah nama parameter dimulai dengan "vision_encoder" atau "prompt_encoder".
- **vision_encoder** dan **prompt_encoder** adalah bagian-bagian dari model yang digunakan untuk menangani input gambar dan prompt yang diberikan untuk segmentasi.
- Dengan kata lain, kode ini bertujuan untuk menonaktifkan pembaruan gradien untuk parameter-parameter yang terkait dengan encoder gambar dan prompt encoder.

5. param.requires_grad_(False):

- Mengatur agar gradien **tidak dihitung** untuk parameter ini selama proses pelatihan, yang berarti parameter ini tidak akan diperbarui selama optimisasi.
- **requires_grad_(False)** adalah cara untuk menandakan bahwa parameter ini tidak memerlukan perhitungan gradien, sehingga menghemat memori dan waktu komputasi saat pelatihan.

Model SamModel:

```
model
```

```

    (shared_image_embedding): SamPositionalEmbedding()
  (vision_encoder): SamVisionEncoder(
    (patch_embed): SamPatchEmbeddings(
      (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
    )
    (layers): ModuleList(
      (0-11): 12 x SamVisionLayer(
        (layer_norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (attn): SamVisionAttention(
          (qkv): Linear(in_features=768, out_features=2304, bias=True)
          (proj): Linear(in_features=768, out_features=768, bias=True)
        )
        (layer_norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
        (mlp): SamMLPBlock(
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (act): GELUActivation()
        )
      )
    )
  )
  (neck): SamVisionNeck(
    (conv1): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (layer_norm1): SamLayerNorm()
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (layer_norm2): SamLayerNorm()
  )
)
),
(neck): SamVisionNeck(
  (conv1): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (layer_norm1): SamLayerNorm()
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (layer_norm2): SamLayerNorm()
),
(prompt_encoder): SamPromptEncoder(
  (shared_embedding): SamPositionalEmbedding()
  (mask_embed): SamMaskEmbedding(
    (activation): GELUActivation()
    (conv1): Conv2d(1, 4, kernel_size=(2, 2), stride=(2, 2))
    (conv2): Conv2d(4, 16, kernel_size=(2, 2), stride=(2, 2))
    (conv3): Conv2d(16, 256, kernel_size=(1, 1), stride=(1, 1))
    (layer_norm1): SamLayerNorm()
    (layer_norm2): SamLayerNorm()
  )
  (no_mask_embed): Embedding(1, 256)
  (point_embed): ModuleList(
    (0-3): 4 x Embedding(1, 256)
  )
  (not_a_point_embed): Embedding(1, 256)
),
(mask_decoder): SamMaskDecoder(
  (iou_token): Embedding(1, 256)
  (mask_tokens): Embedding(4, 256)
  (transformer): SamTwoWayTransformer(
    (layers): ModuleList(
      (0-1): 2 x SamTwoWayAttentionBlock(
        (self_attn): SamAttention(
          (q_proj): Linear(in_features=256, out_features=256, bias=True)
          (k_proj): Linear(in_features=256, out_features=256, bias=True)
          (v_proj): Linear(in_features=256, out_features=256, bias=True)
          (out_proj): Linear(in_features=256, out_features=256, bias=True)
        )
        (layer_norm1): LayerNorm((256,), eps=1e-06, elementwise_affine=True)
        (cross_attn_token_to_image): SamAttention(
          (q_proj): Linear(in_features=256, out_features=128, bias=True)
          (k_proj): Linear(in_features=256, out_features=128, bias=True)
          (v_proj): Linear(in_features=256, out_features=128, bias=True)
          (out_proj): Linear(in_features=128, out_features=256, bias=True)
        )
        (layer_norm2): LayerNorm((256,), eps=1e-06, elementwise_affine=True)
        (mlp): SamMLPBlock(
          (lin1): Linear(in_features=256, out_features=2048, bias=True)
          (lin2): Linear(in_features=2048, out_features=256, bias=True)
          (act): ReLU()
        )
        (layer_norm3): LayerNorm((256,), eps=1e-06, elementwise_affine=True)
        (layer_norm4): LayerNorm((256,), eps=1e-06, elementwise_affine=True)
        (cross_attn_image_to_token): SamAttention(
          (q_proj): Linear(in_features=256, out_features=128, bias=True)
          (k_proj): Linear(in_features=256, out_features=128, bias=True)
          (v_proj): Linear(in_features=256, out_features=128, bias=True)
          (out_proj): Linear(in_features=128, out_features=256, bias=True)
        )
      )
    )
  )
)

```

```

        )
    )
)
(final_attn_token_to_image): SamAttention(
    (q_proj): Linear(in_features=256, out_features=128, bias=True)
    (k_proj): Linear(in_features=256, out_features=128, bias=True)
    (v_proj): Linear(in_features=256, out_features=128, bias=True)
    (out_proj): Linear(in_features=128, out_features=256, bias=True)
)
(layer_norm_final_attn): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
)
(upscale_conv1): ConvTranspose2d(256, 64, kernel_size=(2, 2), stride=(2, 2))
(upscale_conv2): ConvTranspose2d(64, 32, kernel_size=(2, 2), stride=(2, 2))
(upscale_layer_norm): SamLayerNorm()
(activation): GEGLU(approximate='none')
(output_hypernetworks_mlps): ModuleList(
    (0-3): 4 x SamFeedForward(
        (activation): ReLU()
        (proj_in): Linear(in_features=256, out_features=256, bias=True)
        (proj_out): Linear(in_features=256, out_features=32, bias=True)
        (layers): ModuleList(
            (0): Linear(in_features=256, out_features=256, bias=True)
        )
    )
)
(iou_prediction_head): SamFeedForward(
    (activation): ReLU()
    (proj_in): Linear(in_features=256, out_features=256, bias=True)
    (proj_out): Linear(in_features=256, out_features=4, bias=True)
    (layers): ModuleList(
        (0): Linear(in_features=256, out_features=256, bias=True)
    )
)

```

Model SamModel yang ditampilkan di atas adalah implementasi dari Segment Anything Model (SAM), yang merupakan model untuk tugas segmentasi gambar. Berikut adalah komponen-komponen utama yang terdapat dalam model ini:

1. shared_image_embedding:

- **SamPositionalEmbedding:** Menyediakan embedding posisi untuk gambar yang digunakan dalam input gambar, yang memberikan informasi tentang lokasi piksel dalam gambar.

2. vision_encoder:

- **SamVisionEncoder:** Encoder yang bertanggung jawab untuk memproses gambar input.
 - **patch_embed:** Menggunakan convolution (Conv2d) untuk mengubah gambar menjadi patch yang dapat diproses oleh model.
 - **layers:** Terdiri dari 12 lapisan **vision layer** yang masing-masing berisi komponen seperti **attention mechanism** dan **MLP (Multilayer Perceptron)** untuk mengekstrak fitur dari gambar.
 - **neck:** Menggunakan beberapa lapisan konvolusi untuk menghasilkan representasi fitur gambar yang lebih terstruktur dan siap untuk dekoder.

3. prompt_encoder:

- **SamPromptEncoder:** Encoder yang bertanggung jawab untuk memproses informasi terkait dengan prompt yang diberikan, seperti masker atau kotak pembatas.
 - **mask_embed:** Menggunakan beberapa lapisan konvolusi untuk menghasilkan representasi dari masker input.
 - **no_mask_embed** dan **point_embed:** Embedding untuk menangani informasi titik atau masker yang tidak ada.
 - **not_a_point_embed:** Representasi untuk penanganan kasus non-titik dalam input.

4. mask_decoder:

- **SamMaskDecoder:** Decoder yang bertanggung jawab untuk menghasilkan output segmentasi berupa masker.
 - **iou_token:** Token untuk menangani prediksi **Intersection over Union (IoU)** yang digunakan dalam prediksi masker.
 - **mask_tokens:** Token yang digunakan untuk representasi masker.
 - **transformer:** Menggunakan **two-way transformer** untuk menangani perhatian antara masker dan gambar.
 - **upscale_conv1 dan upscale_conv2:** Lapisan transposisi konvolusi untuk meningkatkan resolusi output segmentasi.
 - **iou_prediction_head:** Prediksi untuk memperkirakan **IoU** pada hasil segmentasi.

5. Fungsi:

Model ini mengimplementasikan encoder-decoder architecture yang umum dalam model-model segmentasi gambar, di mana:

- **Encoder:** Memproses gambar untuk ekstraksi fitur.
- **Prompt Encoder:** Memproses prompt atau informasi tambahan seperti masker.
- **Mask Decoder:** Menghasilkan prediksi masker atau segmentasi untuk gambar input.

Latih model:

```
from torch.optim import Adam
import torchvision

# Catatan: Penalaan hiperparameter dapat meningkatkan kinerja di sini
optimizer = Adam(model.mask_decoder.parameters(), lr=1e-5,
weight_decay=0)
```

Kode di atas berfungsi untuk mengonfigurasi optimizer (penyusun pembaruan parameter) menggunakan algoritma Adam untuk melatih bagian mask_decoder dari model SamModel. Berikut adalah penjelasan rinci dari setiap bagian kode:

1. from torch.optim import Adam:

- Mengimpor kelas Adam dari pustaka torch.optim. Adam adalah algoritma optimisasi yang sangat populer dan efisien untuk model pembelajaran dalam, yang menggabungkan keuntungan dari dua algoritma optimisasi lainnya (Momentum dan RMSProp).

2. import torchvision:

- Mengimpor pustaka torchvision, yang menyediakan berbagai alat untuk pemrosesan gambar, model pralatih, dan transformasi gambar. Namun, dalam kode ini, pustaka tersebut tidak digunakan langsung.

```
3. optimizer = Adam(model.mask_decoder.parameters(), lr=1e-5, weight_decay=0):
```

- **model.mask_decoder.parameters()**: Mengambil parameter-parameter yang akan dilatih dari bagian mask_decoder model SamModel. Parameter ini digunakan untuk menghasilkan masker output dalam segmentasi.
 - **lr=1e-5**: Menetapkan nilai **learning rate** (tingkat pembelajaran) sebesar 1×10^{-5} . Learning rate ini mengatur seberapa besar langkah yang diambil dalam memperbarui parameter selama pelatihan.
 - **weight_decay=0**: Menetapkan **weight decay** (penurunan bobot) sebesar 0, yang berarti tidak ada regularisasi L2 untuk mengurangi overfitting selama pelatihan. Biasanya, weight decay digunakan untuk mencegah model belajar terlalu banyak dengan memberikan penalti pada bobot yang besar.

Tujuan:

Optimizer Adam ini digunakan untuk memperbarui parameter-parameter dalam mask_decoder model SamModel selama pelatihan. Parameter lr dan weight_decay merupakan bagian dari hiperparameter yang dapat disesuaikan lebih lanjut untuk meningkatkan kinerja model selama pelatihan.

```

        ground_truth_masks =
batch["ground_truth_mask"].float().to(device)

        loss = torchvision.ops.sigmoid_focal_loss(predicted_masks,
ground_truth_masks.unsqueeze(1), reduction='mean')

        # pass mundur (hitung gradien parameter terhadap loss)
optimizer.zero_grad()
loss.backward()

        # optimasi
optimizer.step()
epoch_losses.append(loss.item())

print(f'EPOCH: {epoch}')
print(f'Rata-rata loss: {mean(epoch_losses)}')

 0%|          | 0/40 [00:00:00, ?it/s] /usr/local/lib/python3.10/dist-packages/transformers/image_processing_utils.py:41: UserWarning: The following named arguments are not valid for 'SamImageProcessor.preprocess' and were ignored: 'preprocess'
return self.preprocess(images, **kwargs)
100%|██████████| 40/40 [00:38<00:00, 1.05it/s]
EPOCH: 0
Rata-rata loss: 0.01264735360895345
100%|██████████| 40/40 [00:37<00:00, 1.06it/s]
EPOCH: 1
Rata-rata loss: 0.0079945182275975
100%|██████████| 40/40 [00:38<00:00, 1.03it/s]
EPOCH: 2
Rata-rata loss: 0.00642776846585513
100%|██████████| 40/40 [00:39<00:00, 1.00it/s]
EPOCH: 3
Rata-rata loss: 0.00561699859595825
100%|██████████| 40/40 [00:42<00:00, 1.06s/it]
EPOCH: 4
Rata-rata loss: 0.005270298858522438
100%|██████████| 40/40 [00:44<00:00, 1.10s/it]
EPOCH: 5
Rata-rata loss: 0.005094717015163841
100%|██████████| 40/40 [00:43<00:00, 1.08s/it]
EPOCH: 6
Rata-rata loss: 0.0049089582709711975
100%|██████████| 40/40 [00:43<00:00, 1.09s/it]
EPOCH: 7
Rata-rata loss: 0.004724316984473603
100%|██████████| 40/40 [00:43<00:00, 1.09s/it]
EPOCH: 8
Rata-rata loss: 0.00455986469468486
100%|██████████| 40/40 [00:43<00:00, 1.09s/it]EPOCH: 9
Rata-rata loss: 0.004368038081156556
```

Kode yang diberikan adalah loop pelatihan untuk model SamModel dengan menggunakan Adam optimizer dan sigmoid focal loss untuk segmentasi gambar. Berikut penjelasan rinci tentang tiap bagian kode:

1. Impor Modul:

- **from tqdm import tqdm:** Mengimpor tqdm untuk menampilkan progress bar selama iterasi.
- **from statistics import mean:** Mengimpor fungsi mean untuk menghitung rata-rata nilai loss setiap epoch.
- **import torch:** Mengimpor pustaka PyTorch untuk melakukan operasi tensor dan pelatihan model.
- **from torch.nn.functional import threshold, normalize:** Mengimpor fungsi-fungsi utilitas untuk manipulasi tensor, meskipun fungsi ini tidak digunakan dalam kode.
- **from torch import nn:** Mengimpor modul neural network dari PyTorch.

2. Inisialisasi Variabel:

- **num_epochs = 10:** Menentukan jumlah epoch pelatihan (10 kali).
- **device = "cuda" if torch.cuda.is_available() else "cpu":** Memilih perangkat (GPU jika tersedia, jika tidak CPU) untuk menjalankan model.

- **model.to(device)**: Memindahkan model ke perangkat yang telah dipilih (GPU atau CPU).

3. Pelatihan Model:

- **model.train()**: Mengatur model ke mode pelatihan (training mode) untuk memastikan lapisan seperti dropout atau batch normalization berfungsi dengan benar selama pelatihan.

4. Loop Pelatihan:

- **for epoch in range(num_epochs)**:: Melakukan iterasi selama 10 epoch.
- **epoch_losses = []**: Membuat list untuk menyimpan nilai loss pada setiap batch selama satu epoch.
- **for batch in tqdm(train_dataloader)**:: Melakukan iterasi atas batch-batch dalam train_dataloader dan menampilkan progress bar menggunakan tqdm.

5. Forward Pass (Pass Maju):

- **outputs = model(pixel_values=batch["pixel_values"].to(device), input_boxes=batch["input_boxes"].to(device), multimask_output=False)**:
 - Menjalankan model pada batch data.
 - **pixel_values** adalah gambar input.
 - **input_boxes** adalah kotak pembatas yang digunakan untuk segmen tertentu.
 - **multimask_output=False** menandakan bahwa hanya satu masker yang akan dihasilkan untuk setiap input.

6. Menghitung Loss:

- **predicted_masks = outputs.pred_masks.squeeze(1)**: Menyaring hasil masker dari output model dan menghilangkan dimensi tambahan yang tidak diperlukan.
- **predicted_masks = nn.functional.interpolate(predicted_masks, size=(640, 640), mode='bilinear', align_corners=False)**: Mengubah ukuran masker yang diprediksi ke ukuran (640, 640) menggunakan interpolasi bilinear untuk mencocokkan ukuran dengan masker ground truth.
- **ground_truth_masks = batch["ground_truth_mask"].float().to(device)**: Mengambil masker ground truth dari batch dan memindahkannya ke perangkat yang sesuai (GPU atau CPU).
- **loss = torchvision.ops.sigmoid_focal_loss(predicted_masks, ground_truth_masks.unsqueeze(1), reduction='mean')**:
 - Menghitung **sigmoid focal loss** antara masker yang diprediksi dan ground truth.
 - **sigmoid_focal_loss** adalah loss yang digunakan untuk menangani ketidakseimbangan kelas dengan memberikan lebih banyak perhatian pada piksel yang lebih sulit diprediksi.
 - **reduction='mean'** menghitung rata-rata loss di seluruh batch.

7. Backward Pass (Pass Mundur):

- **optimizer.zero_grad()**: Mengatur ulang gradien optimizer sebelum melakukan perhitungan gradien baru.
- **loss.backward()**: Menghitung gradien terhadap parameter model berdasarkan loss.
- **optimizer.step()**: Memperbarui parameter model berdasarkan gradien yang dihitung oleh backward pass.

8. Menyimpan Loss untuk Setiap Epoch:

- **epoch_losses.append(loss.item())**: Menyimpan nilai loss pada setiap batch dalam list epoch_losses.

9. Output Rata-rata Loss:

- **print(f'EPOCH: {epoch}')**: Menampilkan nomor epoch.
- **print(f'Rata-rata loss: {mean(epoch_losses)}')**: Menghitung dan menampilkan rata-rata loss untuk seluruh epoch.

Sebagai contoh inferensi, kita dapat mengambil gambar validasi acak:

```
import numpy as np
from PIL import Image

# mari ambil contoh validasi acak
idx = 4

# memuat gambar
image = dataset["validation"][idx]["image"]
```

Penjelasan:

1. Impor Modul:

- **import numpy as np**: Mengimpor pustaka **NumPy** yang digunakan untuk manipulasi array dan operasi matematika lainnya.
- **from PIL import Image**: Mengimpor **PIL (Python Imaging Library)** untuk memanipulasi gambar, meskipun dalam kode ini pustaka ini tidak digunakan secara langsung.

2. Menentukan Indeks:

- **idx = 4**: Menentukan indeks 4 yang akan digunakan untuk mengambil gambar dari dataset validasi. Indeks ini mengacu pada gambar yang akan diambil dari dataset dataset["validation"].

3. Memuat Gambar:

- **image = dataset["validation"][idx]["image"]**: Mengambil gambar ke-4 dari subset dataset validation. Di sini, dataset["validation"] adalah data validasi yang sebelumnya dimuat dengan fungsi load_dataset().

- **dataset["validation"][idx]**: Mengakses elemen ke-4 dalam subset validasi dataset.
- **["image"]**: Mengambil bagian gambar dari data tersebut. Ini mengembalikan gambar dalam format yang sesuai (biasanya dalam format PIL atau tensor).

4. Menyimpan Gambar dalam Variabel:

- **image**: Gambar yang dimuat dari dataset disimpan dalam variabel image, yang dapat digunakan lebih lanjut untuk analisis atau pemrosesan lainnya.

Berikut hasilnya:



Siapkan input untuk model untuk input model segmentasi:

```
# mendapatkan prompt kotak berdasarkan peta segmentasi kebenaran dasar
ground_truth_mask = np.array(dataset["validation"][idx]["label"])
prompt = get_bounding_box(ground_truth_mask)

# mempersiapkan gambar + prompt kotak untuk model
inputs = processor(image, input_boxes=[[prompt]],
return_tensors="pt").to(device)
for k,v in inputs.items():
    print(k,v.shape)
pixel_values torch.Size([1, 3, 1024, 1024])
original_sizes torch.Size([1, 2])
reshaped_input_sizes torch.Size([1, 2])
input_boxes torch.Size([1, 1, 4])
```

Kode diatas berikan berfungsi untuk mempersiapkan input untuk model segmentasi menggunakan kotak pembatas yang dihasilkan dari masker kebenaran dasar. Berikut penjelasan langkah demi langkah:

1. Mendapatkan Masker Kebenaran Dasar (Ground Truth Mask)

- **ground_truth_mask = np.array(dataset["validation"][idx]["label"]):**

- Mengambil **masker kebenaran dasar** (label) dari dataset validasi pada indeks yang ditentukan (idx).
- **Masker kebenaran dasar** adalah informasi segmentasi objek dalam gambar yang digunakan untuk mengukur kinerja model.
- Data ini disimpan dalam format array NumPy.

2. Mendapatkan Kotak Pembatas (Bounding Box) dari Masker

- `prompt = get_bounding_box(ground_truth_mask):`
 - Fungsi `get_bounding_box()` digunakan untuk mendeteksi dan menghasilkan **kotak pembatas** dari masker yang diambil sebelumnya.
 - Kotak pembatas adalah area di gambar yang berisi objek yang ingin disegmentasi, memberikan model informasi tentang lokasi objek.

3. Mempersiapkan Gambar dan Kotak Pembatas untuk Model

- `inputs = processor(image, input_boxes=[[prompt]], return_tensors="pt").to(device):`
 - Fungsi `processor` mengonversi gambar dan kotak pembatas (`prompt`) menjadi format yang dapat diproses oleh model.
 - `input_boxes=[[prompt]]`: Mengirimkan kotak pembatas yang telah dihasilkan untuk menunjukkan area objek.
 - `return_tensors="pt"`: Mengonversi hasil ke dalam format tensor PyTorch.
 - `.to(device)`: Memindahkan tensor ke perangkat (GPU/CPU) untuk proses lebih lanjut.

4. Mencetak Bentuk (Shape) dari Input yang Diproses

- `for k,v in inputs.items(): print(k,v.shape):`
 - Menampilkan bentuk (shape) dari setiap elemen dalam `inputs` untuk memastikan data yang diproses memiliki ukuran dan dimensi yang benar untuk model.
 - Elemen dalam `inputs` bisa mencakup data seperti `pixel_values` (gambar), `input_boxes`, dan data lainnya yang akan digunakan oleh model.

Tujuan Kode:

- Kode ini digunakan untuk mempersiapkan gambar dan kotak pembatas agar dapat dimasukkan ke dalam model SAM untuk segmentasi. Masker kebenaran dasar digunakan untuk menghasilkan kotak pembatas yang akan digunakan dalam model sebagai input untuk segmentasi yang lebih akurat.

Evaluasi model dengan gambar baru:

```
model.eval()

# pass maju
```

```
with torch.no_grad():
    outputs = model(**inputs, multimask_output=False)
```

Kode ini berfungsi untuk mengubah model ke mode evaluasi dan kemudian menjalankan proses inferensi untuk mendapatkan prediksi tanpa menghitung gradien. Berikut penjelasan langkah demi langkah:

1. Mengubah Model ke Mode Evaluasi

- **model.eval():**
 - Fungsi ini mengubah model ke **mode evaluasi**. Dalam mode ini, model akan menonaktifkan beberapa fitur yang hanya digunakan selama pelatihan, seperti **dropout** dan **batch normalization**. Hal ini memastikan bahwa model berperilaku secara konsisten saat digunakan untuk inferensi.

2. Menghindari Perhitungan Gradien Selama Inferensi

- **with torch.no_grad():**
 - Kode ini digunakan untuk menonaktifkan pencatatan gradien selama proses inferensi.
 - Dengan menonaktifkan gradien, proses inferensi menjadi lebih efisien karena PyTorch tidak perlu melacak operasi untuk perhitungan gradien, yang biasanya diperlukan selama pelatihan. Ini menghemat memori dan waktu komputasi.

3. Memprediksi Output dengan Model

- **outputs = model(**inputs, multimask_output=False):**
 - Model menerima **inputs** yang telah diproses sebelumnya, termasuk gambar dan kotak pembatas (prompt), sebagai input untuk prediksi.
 - **multimask_output=False**: Parameter ini memastikan bahwa model hanya menghasilkan satu masker untuk setiap input, bukan beberapa masker.

Tujuan Kode:

- Kode ini menjalankan model pada data uji (seperti gambar validasi) untuk menghasilkan prediksi segmen objek (masker). Menggunakan **torch.no_grad()** membantu meningkatkan efisiensi selama inferensi, karena tidak perlu menghitung atau menyimpan gradien yang hanya diperlukan saat pelatihan.

```
# menerapkan sigmoid
sam_seg_prob = torch.sigmoid(outputs.pred_masks.squeeze(1))

# mengubah ukuran masker ke ukuran semula
sam_seg_prob = nn.functional.interpolate(sam_seg_prob,
                                          size=(640, 640),
                                          mode='bilinear',
                                          align_corners=False)

# mengonversi masker lunak menjadi masker keras
sam_seg_prob = sam_seg_prob.cpu().numpy().squeeze()
```

```
sam_segmentation_results = (sam_seg_prob > 0.5).astype(np.uint8)
```

Kode ini berfungsi untuk memproses hasil prediksi segmen dari model menjadi masker keras yang dapat digunakan untuk segmentasi objek. Berikut penjelasan langkah demi langkah:

1. Menerapkan Fungsi Sigmoid pada Output

- **sam_seg_prob = torch.sigmoid(outputs.pred_masks.squeeze(1)):**
 - Output dari model (outputs.pred_masks) berupa nilai probabilitas yang belum melalui fungsi aktivasi, sehingga digunakan **sigmoid** untuk mengubahnya menjadi rentang nilai antara 0 dan 1.
 - **squeeze(1):** Menghilangkan dimensi batch atau channel yang tidak diperlukan, menyisakan tensor dengan bentuk yang lebih sesuai untuk diproses.

2. Mengubah Ukuran Masker

- **sam_seg_prob = nn.functional.interpolate(sam_seg_prob, size=(640, 640), mode='bilinear', align_corners=False):**
 - Fungsi **interpolate** digunakan untuk mengubah ukuran masker segmen (sam_seg_prob) agar sesuai dengan ukuran yang diinginkan, dalam hal ini ukuran **640x640**.
 - **mode='bilinear'**: Menggunakan interpolasi bilinear untuk resizing, yang umumnya cocok untuk gambar.
 - **align_corners=False**: Menghindari penajaran sudut saat resizing, yang bisa memperbaiki hasil interpolasi.

3. Mengonversi Masker Lunak menjadi Masker Keras

- **sam_seg_prob = sam_seg_prob.cpu().numpy().squeeze():**
 - Memindahkan hasil dari GPU ke CPU dengan **cpu()** dan mengonversinya menjadi array NumPy dengan **numpy()**.
 - **squeeze()**: Menghilangkan dimensi ekstra yang tidak diperlukan, seperti dimensi batch atau channel tunggal.
- **sam_segmentation_results = (sam_seg_prob > 0.5).astype(np.uint8):**
 - **(sam_seg_prob > 0.5)**: Menerapkan thresholding, di mana nilai probabilitas yang lebih besar dari 0.5 akan dianggap sebagai bagian dari objek (1), dan sisanya dianggap sebagai latar belakang (0).
 - **astype(np.uint8)**: Mengonversi hasil ke tipe data **uint8**, yang cocok untuk gambar biner (0 dan 1).

Tujuan Kode:

- Kode ini mengambil hasil prediksi segmen model (probabilitas masker), mengubahnya menjadi masker biner (keras) berdasarkan ambang batas 0.5, dan kemudian menyesuaikan ukuran masker agar sesuai dengan ukuran asli gambar untuk digunakan dalam aplikasi lebih lanjut seperti visualisasi atau evaluasi.

```
sam_segmentation_results.shape
```

```
→ (640, 640)
```

Kode `sam_segmentation_results.shape` akan menghasilkan bentuk (shape) dari array `sam_segmentation_results`, yang merupakan masker hasil segmentasi biner. Bentuk ini akan memiliki dua dimensi yang menggambarkan tinggi dan lebar masker, yang seharusnya sama dengan ukuran gambar input yang sudah diubah ukurannya (dalam hal ini, 640x640).

```
ground_truth_mask.shape
```

```
→ (640, 640)
```

Bentuk dari `ground_truth_mask` tergantung pada dimensi asli dari gambar label yang digunakan dalam dataset. Mengingat bahwa dataset yang digunakan berisi gambar segmentasi, `ground_truth_mask` biasanya akan memiliki dua dimensi yang menggambarkan tinggi dan lebar dari gambar label.

```
def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])],
axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

fig, axes = plt.subplots()

axes.imshow(np.array(image))
show_mask(sam_segmentation_results, axes)
axes.title.set_text(f"Predicted mask")
axes.axis("off")
```

Kode ini digunakan untuk menampilkan gambar dengan masker segmentasi yang diprediksi pada gambar tersebut. Berikut adalah penjelasan rinci setiap bagiannya:

1. Fungsi `show_mask`:

- **Parameter:**

- `mask`: Masker yang akan ditampilkan, biasanya berupa array 2D (berisi nilai biner 0 atau 1), yang mewakili area yang tersegmentasi pada gambar.
- `ax`: Objek `axes` dari `matplotlib` untuk menggambar gambar dan masker.
- `random_color`: Opsi untuk memilih warna masker secara acak atau menggunakan warna tetap.

- **Deskripsi:**

- Jika `random_color` diatur `True`, warna masker akan dipilih secara acak (dengan nilai transparansi 0.6), jika tidak, warna biru transparan digunakan.
- Masker diubah ke bentuk `(h, w, 1)` dan dikalikan dengan warna, menghasilkan gambar masker yang transparan.
- Masker ini kemudian ditampilkan di atas gambar menggunakan `ax.imshow()`.

2. Bagian utama:

- Gambar asli (`image`) ditampilkan menggunakan `axes.imshow(np.array(image))`.
- Masker segmentasi yang diprediksi (`sam_segmentation_results`) ditampilkan di atas gambar asli dengan `show_mask(sam_segmentation_results, axes)`. Ini akan mewarnai area segmentasi pada gambar.
- Judul dari plot diatur menjadi "Predicted mask".
- `axes.axis("off")` digunakan untuk menghilangkan sumbu pada plot, sehingga hanya gambar yang terlihat.

Berikut hasilnya:



Bandingkan dengan segmentasi kebenaran dasar:

```
fig, axes = plt.subplots()

axes.imshow(np.array(image))
show_mask(ground_truth_mask, axes)
axes.title.set_text(f"Ground truth mask")
axes.axis("off")
```

Kode ini mirip dengan kode sebelumnya, tetapi kali ini digunakan untuk menampilkan ground truth mask (masker kebenaran dasar) dibandingkan dengan prediksi yang sebelumnya dilakukan. Berikut penjelasannya:

1. Bagian utama:

- `fig, axes = plt.subplots()`: Membuat objek gambar (fig) dan sumbu (axes) untuk menggambar gambar.

2. Menampilkan gambar:

- `axes.imshow(np.array(image))`: Menampilkan gambar input yang diambil dari dataset. Gambar ini akan menjadi latar belakang yang digunakan untuk menampilkan masker.

3. Menampilkan masker kebenaran dasar:

- `show_mask(ground_truth_mask, axes)`: Fungsi `show_mask` digunakan untuk menampilkan masker kebenaran dasar (`ground_truth_mask`) pada gambar. Masker ini adalah ground truth yang menunjukkan area yang seharusnya tersegmentasi pada gambar.

4. Menambahkan judul:

- `axes.title.set_text("Ground truth mask")`: Menambahkan teks "Ground truth mask" sebagai judul gambar agar pengguna tahu bahwa yang ditampilkan adalah masker kebenaran dasar.

5. Menghilangkan sumbu:

- `axes.axis("off")`: Menonaktifkan sumbu (axis) sehingga hanya gambar yang terlihat, tanpa angka atau tanda sumbu.

Berikut hasilnya:

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
(-0.5, 639.5, 639.5, -0.5)
```



Upload model ke hub:

```
from huggingface_hub import notebook_login  
  
notebook_login()
```

```
model.push_to_hub("khaliiishah/huggingface_khalishah")
```

```
⌚ model safetensors: 100% [██████████] 375M/375M [00:38<00:00, 22.5MB/s]
CommitInfo(commit_url='https://huggingface.co/khaliiishah/huggingface_khalishah/commit/a3dee881ae9793f38c604f44cad8d435a4cd5f', commit_message='Upload model', commit_description='', oid='a5deee881ae9793f38c604f44cad8d435a4cd5f', pr_url=None, repo_url='RepoURL('https://huggingface.co/khaliiishah/huggingface_khalishah', endpoint='https://huggingface.co', repotype='model', repo_id='khaliiishah/huggingface_khalishah'), pr_revision=None, pr_num=None)
```

Kode ini digunakan untuk mengautentikasi pengguna ke dalam akun Hugging Face dan mengunggah model ke repositori Hugging Face Hub. Berikut penjelasan masing-masing bagian kode:

1. `from huggingface_hub import notebook_login`:

- Mengimpor fungsi `notebook_login` dari pustaka `huggingface_hub`, yang digunakan untuk mengautentikasi pengguna dalam notebook Jupyter atau lingkungan interaktif lainnya.

2. `notebook_login()`:

- Fungsi ini akan meminta pengguna untuk masuk ke akun Hugging Face menggunakan kredensial mereka (biasanya melalui kode autentikasi yang dikirim ke email atau melalui token). Setelah login berhasil, pengguna akan dapat mengakses repositori dan mengunggah model ke Hugging Face Hub.

3. `model.push_to_hub("khaliiishah/huggingface_khalishah")`:

- Fungsi ini mengunggah model yang telah dilatih (disimpan dalam variabel `model`) ke repositori Hugging Face Hub. Nama repositori model adalah "khaliiishah/huggingface_khalishah", yang berarti model akan diunggah di bawah akun Hugging Face milik pengguna khaliiishah dengan nama repositori `huggingface_khalishah`.

3. Fine-tuning Video Vision Transformers for Video Classification (Chapter 7)

Fine-tuning Video Vision Transformers (ViTs) untuk klasifikasi video adalah proses mengadaptasi model Vision Transformer yang awalnya dirancang untuk pengolahan gambar, untuk bekerja dengan data video. Pendekatan ini mengintegrasikan dimensi waktu dari video ke dalam model untuk mendeteksi pola visual dan temporal, yang penting dalam tugas klasifikasi video. Proses ini memungkinkan model untuk mempelajari representasi visual yang lebih kaya dan temporal secara efektif, dengan memanfaatkan data video untuk meningkatkan kinerja klasifikasi[3].

Mulai dengan menginstall library yang diperlukan:

```
# Install library yang diperlukan
!pip install pytorchvideo evaluate accelerate transformers > /dev/null
2>&1
!pip uninstall -y torchvision > /dev/null 2>&1
!pip install torchvision==0.14.1 > /dev/null 2>&1
```

Authentication:

```
from huggingface_hub import notebook_login

notebook_login()
```

Kode di atas menggunakan notebook_login() adalah fungsi dari hugingface_hub yang memungkinkan Anda untuk masuk ke akun Hugging Face langsung melalui Jupyter notebook atau lingkungan berbasis notebook lainnya. Fungsi ini membuka prompt di mana Anda diminta untuk memasukkan token akses API yang terkait dengan akun Hugging Face Anda. Setelah login berhasil, Anda dapat melakukan operasi seperti mengunggah model ke Hugging Face Hub, mengunduh model dari Hub, dan melakukan berbagai interaksi dengan repositori model atau dataset di Hugging Face.

Langkah-langkah umumnya adalah:

1. Fungsi notebook_login() memulai login interaktif.
2. Pengguna akan menerima URL untuk mendapatkan token akses.
3. Setelah memasukkan token akses, pengguna akan login ke akun Hugging Face mereka.

Memuat dataset:

```
from hugingface_hub import hf_hub_download

hf_dataset_identifier = "sayakpaul/ucf101-subset"
filename = "UCF101_subset.tar.gz"
file_path = hf_hub_download(
    repo_id=hf_dataset_identifier, filename=filename,
repo_type="dataset"
)
🔗 /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret 'HF_TOKEN' does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
UCF101_subset.tar.gz: 100% [██████████] 171M/171M [00:01<00:00, 121MB/s]
```

Kode ini menggunakan fungsi hf_hub_download dari hugingface_hub untuk mengunduh dataset atau file dari Hugging Face Hub ke sistem lokal Anda. Berikut adalah penjelasan langkah demi langkah:

1. **hf_dataset_identifier = "sayakpaul/ucf101-subset"**: Ini adalah **identifier** untuk repositori dataset yang ada di Hugging Face Hub. Di sini, repositori bernama ucf101-subset yang dibuat oleh pengguna dengan username sayakpaul.
2. **filename = "UCF101_subset.tar.gz"**: Nama **file** yang akan diunduh dari repositori dataset tersebut, yaitu file UCF101_subset.tar.gz.
3. **hf_hub_download(...)**: Fungsi ini digunakan untuk mengunduh file dari Hugging Face Hub.
 - **repo_id=hf_dataset_identifier**: Menunjukkan repositori dataset yang ingin diunduh.
 - **filename=filename**: Nama **file** yang ingin diunduh.
 - **repo_type="dataset"**: Parameter ini menunjukkan bahwa yang diunduh adalah sebuah **dataset**, bukan model.

4. **file_path = hf_hub_download(...)**: Fungsi ini mengembalikan **path lokal** ke file yang telah diunduh, yang kemudian disimpan dalam variabel file_path.

Ekstraksi file:

```
!tar xf {file_path}
```

Kode !tar xf {file_path} digunakan untuk mengekstrak file arsip (dalam format .tar) ke dalam sistem lokal.

Penjelasan:

- **!:** Digunakan di lingkungan Jupyter Notebook atau IPython untuk mengeksekusi perintah shell (command line).
- **tar:** Alat untuk mengarsipkan dan mengekstrak file di sistem berbasis Unix (seperti Linux dan macOS).
- **xf:** Opsi untuk tar:
 - **x:** Menunjukkan bahwa kita ingin mengekstrak (extract) file.
 - **f:** Menunjukkan bahwa file yang akan diproses ditentukan setelah opsi ini (di sini adalah {file_path}).
- **{file_path}:** Variabel yang berisi path ke file .tar yang telah diunduh sebelumnya. Dalam kasus ini, path file yang diunduh adalah UCF101_subset.tar.gz.

Menampilkan daftar file:

```
dataset_root_path = "UCF101_subset"  
  
!find {dataset_root_path} | head -4  
→ UCF101_subset  
UCF101_subset/train  
UCF101_subset/train/ApplyLipstick  
UCF101_subset/train/ApplyLipstick/v_ApplyLipstick_g21_c01.avi
```

Kode !find {dataset_root_path} | head -4 digunakan untuk menampilkan daftar file atau direktori dalam path yang ditentukan (dataset_root_path), dan hanya menampilkan 4 baris pertama.

Penjelasan:

- **find {dataset_root_path}:** Mencari dan menampilkan semua file dan sub-direktori yang ada di dalam direktori yang ditentukan oleh dataset_root_path (dalam hal ini, "UCF101_subset").
- **|:** Operator pipe yang mengarahkan output dari perintah find ke perintah berikutnya.
- **head -4:** Menampilkan hanya 4 baris pertama dari output yang dihasilkan oleh perintah sebelumnya.

```

import pathlib

dataset_root_path = pathlib.Path(dataset_root_path)
train_count = len(list(dataset_root_path.glob("train/*/*.avi")))
val_count = len(list(dataset_root_path.glob("val/*/*.avi")))
test_count = len(list(dataset_root_path.glob("test/*/*.avi")))
total_video = train_count + val_count + test_count
print(f"Train videos: {train_count}")
print(f"Test videos: {test_count}")
print(f"Validation videos: {val_count}")
print(f"Total videos: {total_video}")

```

→ Train videos: 300
 Test videos: 75
 Validation videos: 30
 Total videos: 405

Kode ini digunakan untuk menghitung jumlah video dalam setiap kategori (train, validation, test) di dalam dataset UCF101 yang telah diekstraksi. Berikut penjelasannya:

- dataset_root_path = pathlib.Path(dataset_root_path):** Mengonversi string path dataset_root_path menjadi objek Path dari modul pathlib. Ini memungkinkan kita untuk lebih mudah bekerja dengan path file di dalam sistem file.
- train_count = len(list(dataset_root_path.glob("train/*/*.avi"))):** Menggunakan glob untuk mencari semua file dengan ekstensi .avi di dalam subdirektori train dan menghitung jumlahnya. Format train/*/*.avi berarti mencari file .avi di dalam setiap folder di dalam train/.
- val_count = len(list(dataset_root_path.glob("val/*/*.avi"))):** Melakukan hal yang sama untuk dataset val untuk menghitung jumlah video dalam folder validation.
- test_count = len(list(dataset_root_path.glob("test/*/*.avi"))):** Menghitung jumlah video dalam folder test.
- total_video = train_count + val_count + test_count:** Menjumlahkan semua video yang ditemukan di folder train, val, dan test.
- print(f"Train videos: {train_count}"):** Mencetak jumlah video untuk training.
- print(f"Test videos: {test_count}"):** Mencetak jumlah video untuk testing.
- print(f"Validation videos: {val_count}"):** Mencetak jumlah video untuk validation.
- print(f"Total videos: {total_video}"):** Mencetak jumlah total video yang ada.

Menggabungkan video dari tiga set data:

```

all_video_file_paths = (
    list(dataset_root_path.glob("train/*/*.avi"))
    + list(dataset_root_path.glob("val/*/*.avi"))

```

```

    + list(dataset_root_path.glob("test/*/*.avi"))
)
all_video_file_paths[:5]
→ [PosixPath('UCF101_subset/train/ApplyLipstick/v_ApplyLipstick_g21_c01.avi'),
 PosixPath('UCF101_subset/train/ApplyLipstick/v_ApplyLipstick_g08_c02.avi'),
 PosixPath('UCF101_subset/train/ApplyLipstick/v_ApplyLipstick_g17_c01.avi'),
 PosixPath('UCF101_subset/train/ApplyLipstick/v_ApplyLipstick_g02_c03.avi'),
 PosixPath('UCF101_subset/train/ApplyLipstick/v_ApplyLipstick_g01_c02.avi')]

```

Kode ini menggabungkan semua path file video dari tiga kategori dataset (train, validation, dan test) menjadi satu daftar dan menampilkan 5 file pertama. Berikut adalah penjelasan langkah-langkahnya:

- dataset_root_path.glob("train/*/*.avi")**: Mencari semua file dengan ekstensi .avi di dalam subfolder train. * digunakan untuk mewakili semua subfolder di dalam train/.
- dataset_root_path.glob("val/*/*.avi")**: Melakukan hal yang sama untuk folder val (validation).
- dataset_root_path.glob("test/*/*.avi")**: Melakukan hal yang sama untuk folder test.
- +:** Operator ini digunakan untuk menggabungkan tiga daftar file video yang ditemukan di tiga folder tersebut.
- all_video_file_paths[:5]**: Menampilkan 5 file video pertama dari daftar all_video_file_paths.

Mencetak label kelas unik:

```

class_labels = sorted({str(path).split("/")[-2] for path in
all_video_file_paths})
my_label2id = {label: i for i, label in enumerate(class_labels)}
my_id2label = {i: label for label, i in my_label2id.items()}

print(f"Unique classes: {list(my_label2id.keys())}.")
→ Unique classes: ['ApplyEyeMakeup', 'ApplyLipstick', 'Archery', 'BabyCrawling', 'BalanceBeam', 'BandMarching', 'BaseballPitch', 'Basketball', 'BasketballDunk', 'BenchPress'].

```

Kode ini digunakan untuk menghasilkan dan mencetak label kelas unik dari dataset video yang ada, serta membuat dua kamus: satu untuk mengonversi label kelas menjadi ID numerik (my_label2id), dan satu lagi untuk mengonversi ID numerik kembali menjadi label kelas (my_id2label). Berikut adalah penjelasan langkah-langkahnya:

1. **{str(path).split("/")[2] for path in all_video_file_paths}:**

- Bagian ini menghasilkan set dari nama kelas berdasarkan path file video.
- str(path) mengubah path file video menjadi string.
- split("/") membagi path berdasarkan karakter /.
- Indeks [2] digunakan untuk mengambil nama kelas yang ada di dalam subfolder pertama setelah train, val, atau test (misalnya, dalam path train/ApplyEyeMakeup/v_ApplyEyeMakeup_g01_c01.avi, kelasnya adalah ApplyEyeMakeup).

- Menggunakan set memastikan bahwa nama kelas yang sama tidak terduplikasi.

2. **sorted(...):**

- Setelah mendapatkan set dari nama kelas, sorted mengurutkan kelas tersebut dalam urutan alfabet.

3. **my_label2id = {label: i for i, label in enumerate(class_labels)}:**

- Membuat kamus (my_label2id) yang memetakan setiap nama kelas (label) ke ID numerik yang dihasilkan dari enumerate.
- enumerate(class_labels) menghasilkan pasangan (index, label) yang diubah menjadi kamus.

4. **my_id2label = {i: label for label, i in my_label2id.items()}:**

- Membalikkan kamus my_label2id untuk membuat kamus my_id2label yang memetakan ID numerik kembali ke label kelas.

5. **print(f"Unique classes: {list(my_label2id.keys())}":)**

- Mencetak daftar label kelas unik yang ditemukan di dataset dengan menggunakan my_label2id.keys().

Mendefinisikan model dan batch size:

```
model_checkpt = "google/vivit-b-16x2-kinetics400" # pre-trained ViViT
model
batch_size = 4
```

Kode ini menunjukkan dua variabel yang digunakan untuk mendefinisikan model dan batch size dalam konteks pelatihan model untuk klasifikasi video menggunakan ViViT (Video Vision Transformer):

1. **model_checkpt = "google/vivit-b-16x2-kinetics400":**

- Variabel ini menyimpan nama model pre-trained ViViT yang digunakan untuk klasifikasi video.
- "google/vivit-b-16x2-kinetics400" adalah model ViViT yang telah dilatih sebelumnya pada dataset Kinetics-400, yang merupakan dataset besar untuk klasifikasi video. Model ini menggunakan arsitektur ViViT dengan backbone "B" dan konfigurasi input frame 16x2.

2. **batch_size = 4:**

- Variabel ini menentukan ukuran batch yang akan digunakan selama pelatihan atau inferensi.
- Dengan batch_size = 4, model akan memproses 4 video per iterasi selama pelatihan atau evaluasi.

Loading video processor:

```
from transformers import VivitImageProcessor

image_processor = VivitImageProcessor.from_pretrained(model_checkpoint)

```

The cache for model files in Transformers v4.22.0 has been updated. Migrating your old cache. This is a one-time only operation. You can interrupt this and resume the migration later
00 [00:00<?, ?B/s]
preprocessor_config.json: 100% 401/401 [00:00<00:00, 10.6kB/s]

Kode ini melakukan dua hal utama:

1. Mengimpor VivitImageProcessor:

- VivitImageProcessor adalah kelas dari library Hugging Face yang digunakan untuk memproses data video agar dapat diproses oleh model ViViT. Kelas ini menangani tugas-tugas seperti pemrosesan gambar dan video menjadi format yang sesuai (misalnya, normalisasi atau resizing) yang diperlukan oleh model ViViT.

2. Memuat image_processor menggunakan model pre-trained:

- VivitImageProcessor.from_pretrained(model_checkpoint) memuat instance VivitImageProcessor yang sudah dilatih sebelumnya dari model checkpoint yang ditentukan, yaitu "google/vivit-b-16x2-kinetics400".
- Model checkpoint ini adalah model yang sudah dilatih pada dataset Kinetics-400. image_processor ini akan digunakan untuk mempersiapkan video input sebelum dimasukkan ke dalam model ViViT.

Memuat model:

```
from transformers import VivitForVideoClassification

model = VivitForVideoClassification.from_pretrained(
    model_checkpoint,
    id2label=my_id2label,
    label2id=my_label2id,
    ignore_mismatched_sizes=True, # provide this in case you're
    planning to fine-tune an already fine-tuned checkpoint
)

config.json: 100% 18.6k/18.6k [00:00<00:00, 365kB/s]
pytorch_model.bin: 100% 356M/356M [00:02<00:00, 207MB/s]

Some weights of VivitForVideoClassification were not initialized from the model checkpoint at google/vivit-b-16x2-kinetics400 and are newly initialized because the shapes did not match
- classifier.weight: found shape torch.Size([400, 768]) in the checkpoint and torch.Size([10, 768]) in the model instantiated
- classifier.bias: found shape torch.Size([400]) in the checkpoint and torch.Size([10]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Kode ini memuat model ViViT (Video Vision Transformer) yang sudah dilatih sebelumnya untuk tugas klasifikasi video dan mengatur ulang label sesuai dengan dataset yang Anda gunakan.

Penjelasan:

1. Mengimpor VivitForVideoClassification:

- VivitForVideoClassification adalah kelas dari library Hugging Face yang dirancang untuk model ViViT khusus untuk tugas klasifikasi video.

2. Memuat Model Pre-Trained:

- `VivitForVideoClassification.from_pretrained(model_checkpt, ...)` memuat model ViViT yang sudah dilatih sebelumnya pada dataset yang disebutkan dalam `model_checkpt` (yaitu, "google/vivit-b-16x2-kinetics400").
- Model ini dioptimalkan untuk memahami video dan mengklasifikasikannya ke dalam kategori yang sesuai.

3. Parameter yang Disediakan:

- **id2label**: Sebuah mapping (dictionary) dari ID numerik ke label string, digunakan untuk menampilkan label yang sesuai dengan prediksi.
- **label2id**: Sebuah mapping dari label string ke ID numerik, digunakan untuk melatih model dengan label numerik yang benar.
- **ignore_mismatched_sizes=True**: Parameter ini berguna jika Anda ingin menggunakan checkpoint model yang sudah di-fine-tune sebelumnya, tetapi ukuran layer output tidak sesuai (misalnya, jika jumlah kelas berbeda). Dengan ini, model akan tetap dapat dimuat, dan layer yang tidak cocok akan diinisialisasi ulang.

Praproses dataset:

```
import pytorchvideo.data

from pytorchvideo.transforms import (
    ApplyTransformToKey,
    RemoveKey,
    Normalize,
    ShortSideScale,
    RandomShortSideScale,
    UniformTemporalSubsample,
)

from torchvision.transforms import (
    RandomCrop,
    RandomHorizontalFlip,
    Resize,
    Compose,
    Lambda,
)
```

Kode ini memuat model ViViT (Video Vision Transformer) yang sudah dilatih sebelumnya untuk tugas klasifikasi video dan mengatur ulang label sesuai dengan dataset yang Anda gunakan.

Penjelasan:

1. Mengimpor `VivitForVideoClassification`:

- VivitForVideoClassification adalah kelas dari library Hugging Face yang dirancang untuk model ViViT khusus untuk tugas klasifikasi video.

2. Memuat Model Pre-Trained:

- VivitForVideoClassification.from_pretrained(model_checkpt, ...) memuat model ViViT yang sudah dilatih sebelumnya pada dataset yang disebutkan dalam model_checkpt (yaitu, "google/vivit-b-16x2-kinetics400").
- Model ini dioptimalkan untuk memahami video dan mengklasifikasikannya ke dalam kategori yang sesuai.

3. Parameter yang Disediakan:

- **id2label**: Sebuah mapping (dictionary) dari ID numerik ke label string, digunakan untuk menampilkan label yang sesuai dengan prediksi.
- **label2id**: Sebuah mapping dari label string ke ID numerik, digunakan untuk melatih model dengan label numerik yang benar.
- **ignore_mismatched_sizes=True**: Parameter ini berguna jika Anda ingin menggunakan checkpoint model yang sudah di-fine-tune sebelumnya, tetapi ukuran layer output tidak sesuai (misalnya, jika jumlah kelas berbeda). Dengan ini, model akan tetap dapat dimuat, dan layer yang tidak cocok akan diinisialisasi ulang.

Proses video:

```
import os

mean = image_processor.image_mean
std = image_processor.image_std
if "shortest_edge" in image_processor.size:
    height = width = image_processor.size["shortest_edge"]
else:
    height = image_processor.size["height"]
    width = image_processor.size["width"]
resize_to = (height, width)

num_frames_to_sample = model.config.num_frames
sample_rate = 4
fps = 30
clip_duration = num_frames_to_sample * sample_rate / fps
```

Kode ini menyiapkan parameter-parameter penting untuk memproses video agar sesuai dengan model ViViT. Model ini membutuhkan input video dalam format tertentu, dan kode ini menentukan bagaimana video diproses sebelum dimasukkan ke model.

Penjelasan:

1. Mengimpor os:

- Library Python standar untuk manipulasi file dan direktori.

2. Mengambil Mean dan Std dari Image Processor:

- `image_processor.image_mean`: Mean (rata-rata) untuk normalisasi setiap channel gambar.
- `image_processor.image_std`: Standar deviasi (standard deviation) untuk normalisasi setiap channel gambar.
- Nilai ini digunakan untuk menormalkan pixel gambar agar sesuai dengan skala data yang digunakan saat pelatihan model.

3. Menentukan Ukuran Gambar:

- Model ViViT memerlukan input gambar dengan dimensi tertentu. Ukuran ini diambil dari konfigurasi yang telah disediakan oleh **image processor**:
 - Jika ada `shortest_edge` dalam ukuran (`image_processor.size`), gambar diubah ukurannya sehingga sisi terpendek memiliki panjang tertentu (dalam piksel).
 - Jika tidak, ukuran gambar disetel ke `height` dan `width` yang diberikan.
- **resize_to**:
 - Tuple (`height, width`) menyimpan dimensi akhir gambar setelah diubah ukurannya. Gambar akan diubah ukurannya menjadi dimensi ini sebelum digunakan sebagai input.

4. Sampling Frame dari Video:

- **num_frames_to_sample**:
 - Model memerlukan sejumlah frame tertentu dari video. Jumlah ini ditentukan oleh `model.config.num_frames`, yang merupakan konfigurasi model.
- **sample_rate**:
 - Menentukan interval frame yang diambil dari video (misalnya, jika `sample_rate=4`, hanya setiap frame ke-4 yang akan diambil).
- **fps (Frame Per Second)**:
 - Kecepatan frame asli dari video (dalam frame per detik). Di sini, kecepatan default disetel ke 30 FPS.
- **clip_duration**:
 - Durasi ini memastikan bahwa video di-sampling ke dalam jumlah frame yang diinginkan dengan kecepatan yang ditentukan.

Transformasi data dan dataset:

```

# Training dataset transformations
train_transform = Compose(
    [
        ApplyTransformToKey(
            key="video",
            transform=Compose(
                [
                    UniformTemporalSubsample(num_frames_to_sample),
                    Lambda(lambda x: x / 255.0),
                    Normalize(mean, std),
                    RandomShortSideScale(min_size=256, max_size=320),
                    RandomCrop(resize_to),
                    RandomHorizontalFlip(p=0.5),
                ]
            ),
        ),
    ],
)
)

# Training dataset
train_dataset = pytorchvideo.data.Ucf101(
    data_path=os.path.join(dataset_root_path, "train"),
    clip_sampler=pytorchvideo.data.make_clip_sampler("random",
clip_duration),
    decode_audio=False,
    transform=train_transform,
)

# Validation and test dataset transformations
val_transform = Compose(
    [
        ApplyTransformToKey(
            key="video",
            transform=Compose(
                [
                    UniformTemporalSubsample(num_frames_to_sample),
                    Lambda(lambda x: x / 255.0),
                    Normalize(mean, std),
                    Resize(resize_to),
                ]
            ),
        ),
    ],
)
)

# Validation and test datasets
val_dataset = pytorchvideo.data.Ucf101(
    data_path=os.path.join(dataset_root_path, "val"),

```

```

clip_sampler=pytorchvideo.data.make_clip_sampler("uniform",
clip_duration,
decode_audio=False,
transform=val_transform,
)

test_dataset = pytorchvideo.data.Ucf101(
    data_path=os.path.join(dataset_root_path, "test"),
    clip_sampler=pytorchvideo.data.make_clip_sampler("uniform",
clip_duration,
decode_audio=False,
transform=val_transform,
)

```

Kode ini mendefinisikan transformasi data dan dataset untuk pelatihan, validasi, dan pengujian model ViViT. Transformasi digunakan untuk mempersiapkan video agar sesuai dengan input model, sementara dataset digunakan untuk menyediakan video yang akan dilatih dan diuji.

Penjelasan:

1. Transformasi Dataset Pelatihan (train_transform):

Transformasi ini mempersiapkan video untuk pelatihan dengan melakukan langkah-langkah berikut:

- **UniformTemporalSubsample:**
 - Menyampling sejumlah frame secara seragam dari video (ditentukan oleh num_frames_to_sample).
- **Lambda(lambda x: x / 255.0):**
 - Mengonversi nilai piksel video dari skala 0-255 menjadi skala 0-1 untuk normalisasi.
- **Normalize(mean, std):**
 - Menormalkan setiap frame berdasarkan **mean** dan **std** yang ditentukan oleh **image_processor**.
- **RandomShortSideScale(min_size=256, max_size=320):**
 - Mengubah ukuran video secara acak sehingga sisi terpendek berada dalam rentang antara 256 dan 320 piksel.
- **RandomCrop(resize_to):**
 - Memotong video secara acak ke ukuran tetap (resize_to), misalnya (224, 224).
- **RandomHorizontalFlip(p=0.5):**
 - Membalik video secara horizontal dengan probabilitas 50%.

Transformasi ini memastikan data pelatihan memiliki variasi dan augmentasi untuk meningkatkan kemampuan generalisasi model.

2. Dataset Pelatihan (train_dataset):

Dataset video untuk pelatihan dibuat menggunakan:

- **Ucf101:**
 - Dataset video aksi **UCF101**.
 - Mengambil data dari direktori pelatihan (train).
- **make_clip_sampler("random", clip_duration):**
 - Menyampling klip video secara acak dengan durasi yang telah ditentukan (clip_duration).
- **decode_audio=False:**
 - Audio dalam video diabaikan.
- **transform=train_transform:**
 - Transformasi khusus untuk pelatihan diterapkan ke dataset.

3. Transformasi Dataset Validasi dan Pengujian (val_transform):

Transformasi untuk validasi dan pengujian lebih sederhana karena tidak melibatkan augmentasi:

- **UniformTemporalSubsample:**
 - Menyampling frame secara seragam dari video.
- **Lambda(lambda x: x / 255.0):**
 - Mengonversi nilai piksel video dari skala 0-255 menjadi skala 0-1.
- **Normalize(mean, std):**
 - Menormalkan piksel frame dengan mean dan std.
- **Resize(resize_to):**
 - Mengubah ukuran video ke dimensi tetap (resize_to), misalnya (224, 224).

Transformasi ini dirancang untuk menjaga data validasi dan pengujian tetap konsisten tanpa augmentasi.

4. Dataset Validasi (val_dataset) dan Pengujian (test_dataset):

Dataset validasi dan pengujian dibuat mirip dengan pelatihan, tetapi dengan perubahan berikut:

- **data_path:**
 - Menggunakan direktori validasi (val) atau pengujian (test).
- **make_clip_sampler("uniform", clip_duration):**
 - Menyampling klip video secara seragam untuk memastikan konsistensi antar klip.
- **transform=val_transform:**
 - Menggunakan transformasi validasi untuk menjaga konsistensi tanpa augmentasi.

Menghitung jumlah video dalam setiap dataset:

```
train_dataset.num_videos, val_dataset.num_videos,  
test_dataset.num_videos  
↳ (300, 30, 75)
```

Kode berikut digunakan untuk menghitung jumlah video dalam setiap dataset (train_dataset, val_dataset, dan test_dataset):

Penjelasan:

- **train_dataset.num_videos**: Mengembalikan jumlah total video yang terdapat dalam dataset pelatihan.
- **val_dataset.num_videos**: Mengembalikan jumlah total video yang terdapat dalam dataset validasi.
- **test_dataset.num_videos**: Mengembalikan jumlah total video yang terdapat dalam dataset pengujian.

Tujuan:

Kode ini membantu memverifikasi bahwa dataset telah dimuat dengan benar dan memiliki jumlah video yang sesuai. Setelah menjalankan perintah ini, Anda akan mendapatkan keluaran berupa tuple dengan jumlah video dalam masing-masing dataset.

Mengambil sampel video:

```
sample_video = next(iter(train_dataset))  
sample_video.keys()  
↳ dict_keys(['video', 'video_name', 'video_index', 'clip_index', 'aug_index', 'label'])
```

Kode berikut digunakan untuk mengambil sampel video pertama dari train_dataset dan melihat kunci yang tersedia dalam data sampel tersebut:

Penjelasan:

1. **next(iter(train_dataset))**:
 - Mengambil sampel pertama dari **train_dataset**.
 - Dataset diubah menjadi iterator dengan **iter()**, kemudian **next()** dipanggil untuk mendapatkan elemen pertama.
2. **sample_video.keys()**:
 - Mengembalikan semua kunci yang tersedia dalam sampel video pertama.
 - Setiap sampel biasanya disusun dalam bentuk dictionary, dengan kunci yang sesuai dengan data seperti video itu sendiri, label kelas, metadata, atau informasi lainnya.

Tujuan:

Untuk mengeksplorasi struktur dataset dan memahami elemen apa saja yang tersedia dalam setiap sampel video.

Memeriksa dan menampilkan informasi sampel video:

```
def investigate_video(sample_video):
    """Utility function to investigate the keys present in a video
    sample."""
    for k in sample_video:
        if k == "video":
            print(k, sample_video["video"].shape)
        else:
            print(k, sample_video[k])

    print(f"Video label: {my_id2label[sample_video[k]]}")

investigate_video(sample_video)
→ video torch.Size([3, 32, 224, 224])
video_name v_Basketball_g01_c01.avi
video_index 210
clip_index 0
aug_index 0
label 7
Video label: Basketball
```

Kode berikut mendefinisikan dan memanggil fungsi `investigate_video()` untuk memeriksa dan menampilkan informasi terkait sampel video dari dataset:

Penjelasan:

1. Definisi fungsi `investigate_video(sample_video)`:

- Fungsi ini berfungsi untuk memeriksa dan menampilkan informasi mengenai kunci-kunci yang ada dalam `sample_video` (satu sampel video).
- `sample_video`: Input berupa dictionary yang berisi data video dan kemungkinan informasi tambahan.

2. Loop untuk memeriksa kunci-kunci dalam `sample_video`:

- `for k in sample_video::` Melakukan iterasi terhadap kunci yang ada dalam dictionary `sample_video`.
- `if k == "video":` Jika kunci tersebut adalah `"video"`, maka kode akan mencetak bentuk (shape) dari data video (misalnya, jumlah frame, ukuran frame, dll.).
- `else::` Jika kunci tersebut bukan `"video"`, maka kode akan mencetak nilai terkait dengan kunci tersebut (misalnya label kelas atau metadata lainnya).

3. `print(f"Video label: {my_id2label[sample_video[k]]}"):`

- Menampilkan label dari video dengan menggunakan `sample_video[k]` untuk mengambil label kelas, kemudian mengonversinya menggunakan `my_id2label` (dictionary yang berisi pemetaan ID ke label).

Tujuan:

Fungsi ini digunakan untuk mengeksplorasi struktur sampel video dari dataset, memeriksa berbagai elemen yang terkandung di dalamnya, serta menampilkan label yang relevan dengan video tersebut.

Visualisasi dan preproses video:

```
import imageio
import numpy as np
from IPython.display import HTML, Image

def unnormalize_img(img):
    """Un-normalizes the image pixels."""
    img = (img * std) + mean
    img = (img * 255).astype("uint8")
    return img.clip(0, 255)

def create_gif(video_tensor, filename="sample.gif"):
    """Prepares a GIF from a video tensor."""
    frames = []
    for video_frame in video_tensor:
        frame_unnormalized = unnormalize_img(video_frame.permute(1, 2, 0).numpy())
        frames.append(frame_unnormalized)
    kargs = {"duration": 0.4}
    imageio.mimsave(filename, frames, "GIF", **kargs)
    return filename

def display_gif(video_tensor, gif_name="sample.gif"):
    """Prepares and displays a GIF from a video tensor."""
    video_tensor = video_tensor.permute(1, 0, 2, 3)      # re-arranging dimension of video_tensor
    gif_filename = create_gif(video_tensor, gif_name)
    return Image(filename=gif_filename)
```

Kode ini berfungsi untuk memproses video dalam bentuk tensor dan mengonversinya menjadi format GIF untuk kemudian ditampilkan di dalam notebook.

Penjelasan:

1. `unnormalize_img(img):`

- Fungsi ini digunakan untuk **mengembalikan gambar ke rentang piksel asli** setelah normalisasi.

- **Normalisasi** yang dilakukan sebelumnya mengubah nilai piksel gambar untuk berada dalam rentang [0, 1]. Fungsi ini mengembalikannya ke rentang [0, 255] (nilai piksel untuk gambar RGB).
- Rumus: **img = (img * std) + mean** untuk un-normalisasi, lalu dikalikan 255 dan diubah menjadi tipe data uint8 (tipe data gambar biasa).
- Fungsi mengembalikan nilai gambar yang telah **di-clip** untuk memastikan piksel berada dalam rentang [0, 255].

2. `create_gif(video_tensor, filename="sample.gif"):`

- Fungsi ini mengonversi **video tensor** menjadi GIF.
- **Video tensor** diharapkan memiliki dimensi (C, T, H, W), yaitu saluran (C), jumlah frame (T), dan dimensi tinggi serta lebar frame (H dan W).
- Fungsi ini:
 - Untuk setiap frame video, memanggil fungsi **unnormalize_img()** untuk mengubah frame dari tensor menjadi gambar dengan nilai piksel yang sesuai.
 - Semua frame dimasukkan ke dalam list **frames**.
 - Menggunakan **imageio.mimsave()** untuk menyimpan frames sebagai GIF dengan durasi antar frame 0.4 detik.
- Mengembalikan nama file GIF yang dihasilkan.

3. `display_gif(video_tensor, gif_name="sample.gif"):`

- Fungsi ini menyiapkan dan menampilkan GIF dari **video tensor**.
- **video_tensor.permute(1, 0, 2, 3)**: Memutar dimensi tensor video dari (C, T, H, W) menjadi (T, C, H, W) agar dimensi sesuai dengan urutan yang diinginkan untuk GIF.
- Memanggil fungsi **create_gif()** untuk membuat GIF.
- Menggunakan **Image()** dari **IPython.display** untuk menampilkan GIF di dalam notebook.

```
video_tensor = sample_video["video"]
display_gif(video_tensor)
```

Kode ini digunakan untuk menampilkan video dalam bentuk GIF dari tensor video yang telah dimuat. Berikut penjelasan singkat tentang kode ini:

1. `video_tensor = sample_video["video"]:`

- `sample_video["video"]` merujuk pada video yang terdapat dalam contoh data (`sample_video`), yang telah dimuat sebelumnya dari dataset.
- **video_tensor** adalah tensor yang berisi data video. Dimensi tensor ini biasanya berupa (C, T, H, W), di mana:
 - C adalah jumlah saluran warna (misalnya, 3 untuk RGB),

- T adalah jumlah frame dalam video,
- H adalah tinggi setiap frame video,
- W adalah lebar setiap frame video.

2. `display_gif(video_tensor)`:

- Fungsi `display_gif()` memproses `video_tensor` untuk mengonversinya menjadi GIF.
- Fungsi ini mengatur ulang dimensi tensor video, mengonversi setiap frame, dan kemudian membuat serta menampilkan GIF di dalam notebook.

Latih model:

```
from transformers import TrainingArguments, Trainer

model_name = model_checkpoint.split("/")[-1]
new_model_name = f"{model_name}-finetuned-ucf101-subset"
num_epochs = 5

args = TrainingArguments(
    new_model_name,
    auto_find_batch_size=True,
    remove_unused_columns=False,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    warmup_ratio=0.1,
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    push_to_hub=False,
    max_steps=(train_dataset.num_videos // batch_size) * num_epochs,
    num_train_epochs=num_epochs,
)
```

Kode ini berfokus pada penyiapan dan konfigurasi `TrainingArguments` untuk pelatihan model ViViT yang akan digunakan untuk mengklasifikasikan video di dataset UCF101-subset. Berikut penjelasan untuk setiap bagian kode:

1. `model_name = model_checkpoint.split("/")[-1]:`

- Mengambil nama model dari path checkpoint pre-trained (misalnya vivit-b-16x2-kinetics400).

2. `new_model_name = f"{model_name}-finetuned-ucf101-subset":`

- Membuat nama model baru dengan menambahkan sufiks -finetuned-ucf101-subset untuk model yang sudah di-fine-tune.

3. **num_epochs = 5:**

- Menentukan jumlah epoch pelatihan (dalam hal ini, 5 epoch).

4. **TrainingArguments:**

- **TrainingArguments** adalah kelas dari Hugging Face yang memungkinkan Anda untuk mengonfigurasi dan menyesuaikan pelatihan model.
- Beberapa parameter penting yang diatur:
 - **new_model_name**: Nama model yang telah di-fine-tune.
 - **auto_find_batch_size=True**: Memungkinkan pencarian otomatis ukuran batch yang optimal.
 - **evaluation_strategy="epoch"**: Evaluasi model dilakukan setiap akhir epoch.
 - **save_strategy="epoch"**: Model disimpan setiap akhir epoch.
 - **learning_rate=5e-5**: Menentukan laju pembelajaran (learning rate) untuk optimisasi.
 - **per_device_train_batch_size=batch_size** dan **per_device_eval_batch_size=batch_size**: Menentukan ukuran batch untuk pelatihan dan evaluasi.
 - **warmup_ratio=0.1**: Mengatur proporsi dari jumlah langkah yang digunakan untuk pemanasan (warmup) sebelum mencapai laju pembelajaran penuh.
 - **logging_steps=10**: Menentukan frekuensi logging selama pelatihan.
 - **load_best_model_at_end=True**: Memuat model terbaik pada akhir pelatihan berdasarkan metrik evaluasi.
 - **metric_for_best_model="accuracy"**: Menggunakan akurasi sebagai metrik untuk memilih model terbaik.
 - **push_to_hub=False**: Tidak mengirimkan model ke Hugging Face Hub.
 - **max_steps**: Mengatur jumlah langkah pelatihan total berdasarkan jumlah video dan ukuran batch.
 - **num_train_epochs=num_epochs**: Menentukan jumlah epoch pelatihan.

Evaluasi akurasi model:

```
import evaluate

metric = evaluate.load("accuracy")
```

```

def compute_metrics(eval_pred):
    """Computes accuracy on a batch of predictions."""
    predictions = np.argmax(eval_pred.predictions, axis=1)
    return metric.compute(predictions=predictions,
    references=eval_pred.label_ids)

```

Download builder script: 100% 4.20k/4.20k [00:00<00:00, 273kB/s]

Kode ini mengimpor dan mengkonfigurasi evaluasi metrik untuk pelatihan model ViViT dengan menggunakan Hugging Face evaluate library. Berikut penjelasan lebih rinci tentang bagian-bagian kode:

1. import evaluate:

- Mengimpor pustaka **evaluate** dari Hugging Face, yang digunakan untuk memuat dan menghitung berbagai metrik evaluasi.

2. metric = evaluate.load("accuracy"):

- Memuat metrik **accuracy** menggunakan fungsi `evaluate.load()`. Ini akan mempersiapkan metrik untuk digunakan dalam evaluasi model selama pelatihan atau pengujian.

3. def compute_metrics(eval_pred):

- Mendefinisikan fungsi `compute_metrics`, yang akan digunakan untuk menghitung akurasi pada prediksi model selama evaluasi.

4. predictions = np.argmax(eval_pred.predictions, axis=1):

- Mengambil `eval_pred.predictions`, yang berisi hasil prediksi dari model, dan menghitung argmax untuk setiap contoh (prediksi kelas dengan skor tertinggi).
- `axis=1` berarti mengambil argmax di sepanjang dimensi kelas untuk setiap prediksi (misalnya, jika ada 10 kelas, ini memilih kelas dengan skor tertinggi).

5. return metric.compute(predictions=predictions, references=eval_pred.label_ids):

- Menghitung akurasi dengan memanggil `metric.compute()`, yang membandingkan `predictions` (prediksi model) dengan `eval_pred.label_ids` (label asli dari data evaluasi).
- Fungsi ini akan mengembalikan nilai akurasi yang dihitung.

Membuat fungsi collate:

```

import torch

def collate_fn(examples):
    """The collation function to be used by `Trainer` to prepare data batches."""
    # permute to (num_frames, num_channels, height, width)

```

```

pixel_values = torch.stack(
    [example["video"].permute(1, 0, 2, 3) for example in examples]
)
labels = torch.tensor([example["label"] for example in examples])
return {"pixel_values": pixel_values, "labels": labels}

```

Fungsi `collate_fn` di atas adalah fungsi yang digunakan untuk menggabungkan contoh-contoh individu dalam satu batch selama pelatihan atau evaluasi. Fungsi ini diteruskan ke Trainer dari Hugging Face Transformers untuk mempersiapkan data batch. Berikut adalah penjelasan rinci dari kode ini:

Penjelasan:

1. `def collate_fn(examples):`:

- Mendefinisikan fungsi `collate_fn`, yang akan menerima daftar `examples` (yaitu, batch data yang berisi contoh-contoh video). Fungsi ini akan mengubah bentuk data menjadi format yang sesuai untuk pelatihan model.

2. `pixel_values = torch.stack([example["video"].permute(1, 0, 2, 3) for example in examples]):`

- Untuk setiap contoh dalam `examples`, fungsi ini akan mengambil video (yang disimpan dalam `example["video"]`) dan melakukan **permutasi dimensi** menggunakan `.permute(1, 0, 2, 3)` untuk memastikan dimensi video sesuai dengan format yang diperlukan oleh model.
 - `example["video"]` adalah tensor dengan dimensi `[num_channels, num_frames, height, width]`, dan setelah permutasi, dimensi tersebut menjadi `[num_frames, num_channels, height, width]`, yang merupakan urutan yang diinginkan untuk video input.
- `torch.stack(...)` menggabungkan semua tensor video menjadi satu tensor besar yang berisi batch dari semua video.

3. `labels = torch.tensor([example["label"] for example in examples]):`

- Menyiapkan tensor untuk label yang terkait dengan video-video tersebut. Setiap contoh memiliki label yang disimpan dalam `example["label"]`.
- Fungsi `torch.tensor(...)` mengonversi daftar label menjadi tensor PyTorch.

4. `return {"pixel_values": pixel_values, "labels": labels}:`

- Fungsi ini mengembalikan dictionary dengan dua elemen:
 - **"pixel_values"**: Tensor berisi data video yang telah dipermutasi dan disusun ke dalam batch.
 - **"labels"**: Tensor berisi label untuk setiap video dalam batch.
- Format output ini sesuai dengan apa yang dibutuhkan oleh Trainer untuk proses pelatihan.

```
trainer = Trainer(  
    model,  
    args,  
    train_dataset=train_dataset,  
    eval_dataset=val_dataset,  
    tokenizer=image_processor,  
    compute_metrics=compute_metrics,  
    data_collator=collate_fn,  
)
```

Kode di atas mendefinisikan objek Trainer dari library Hugging Face Transformers, yang digunakan untuk pelatihan model. Berikut adalah penjelasan setiap bagiannya:

Penjelasan:

1. `trainer = Trainer(...)`:

- Membuat objek **Trainer** yang mengatur dan menjalankan proses pelatihan dan evaluasi model dengan memanfaatkan berbagai parameter yang disediakan.

2. `model`:

- Model yang akan dilatih, yang sebelumnya sudah didefinisikan dengan **VivitForVideoClassification**.

3. `args`:

- Menyediakan **TrainingArguments** yang telah ditetapkan sebelumnya, yang mencakup berbagai pengaturan untuk pelatihan, seperti jumlah epoch, batch size, strategi evaluasi, strategi penyimpanan model, dan sebagainya.

4. `train_dataset=train_dataset`:

- Dataset yang digunakan untuk pelatihan, yaitu **train_dataset** yang telah dipersiapkan dengan transformasi dan pengaturan lainnya.

5. `eval_dataset=val_dataset`:

- Dataset yang digunakan untuk evaluasi model, yaitu **val_dataset**. Dataset ini digunakan untuk menghitung metrik selama evaluasi setelah setiap epoch.

6. `tokenizer=image_processor`:

- **image_processor** yang sebelumnya didefinisikan sebagai **VivitImageProcessor** digunakan untuk memproses data input (video) sebelum diberikan ke model. Ini termasuk normalisasi dan preprocessing gambar/video.

7. `compute_metrics=compute_metrics`:

- Fungsi **compute_metrics** yang mendefinisikan cara menghitung metrik untuk evaluasi, dalam hal ini, **akurasi**. Fungsi ini akan dipanggil selama evaluasi untuk menghitung akurasi model berdasarkan prediksi dan label sebenarnya.

8. `data_collator=collate_fn`:

- Fungsi **collate_fn** yang sudah didefinisikan sebelumnya. Fungsi ini digunakan untuk menggabungkan batch data selama pelatihan atau evaluasi.

Hasil pelatihan:

```
train_results = trainer.train()
→ wandb: WARNING The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run name by :
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
Tracking run with wandb version 0.19.1
Run data is saved locally /content/wandb/run-20250104_075626-6wzluule
Syncing run vivit-b-16x2-kinetics400-finetuned-ucf101-subset to Weights & Biases \(docs\)
View project at https://wandb.ai/khalilishah-telkom-university/huggingface/runs/6wzluule
View run at https://wandb.ai/khalilishah-telkom-university/huggingface/runs/6wzluule [375/375 11:22, Epoch 1/9223372036854775807]

Epoch Training Loss Validation Loss Accuracy
0      0.327000    0.041304   0.972973
1      0.002800    0.024778   0.972973
```

Kode ini mengindikasikan bahwa pelatihan model sedang berlangsung dan terintegrasi dengan Weights & Biases (W&B), sebuah platform untuk pelacakan eksperimen, visualisasi metrik, dan kolaborasi tim.

Menyimpan model, hasil evaluasi, dan status pelatihan:

```
trainer.save_model()
trainer.log_metrics("test", test_results)
trainer.save_metrics("test", test_results)
trainer.save_state()

→ ***** test metrics *****
epoch                  =      1.2
eval_accuracy          =      0.977
eval_loss               =      0.0847
eval_runtime            = 0:01:06.01
eval_samples_per_second =      1.318
eval_steps_per_second   =      0.333
```

Kode ini berfungsi untuk menyimpan model, hasil evaluasi, dan status pelatihan ke disk.

Penjelasan:

1. `trainer.save_model()`

- Fungsi ini menyimpan model yang telah dilatih ke dalam direktori yang ditentukan dalam **TrainingArguments.output_dir**. Ini menyimpan **berat model** dan struktur model dalam format yang dapat digunakan kembali untuk inferensi atau pelatihan lebih lanjut.

2. `trainer.log_metrics("test", test_results)`

- Fungsi ini mencatat metrik hasil evaluasi model pada dataset "test" ke dalam log pelatihan. **test_results** adalah hasil evaluasi yang berisi metrik seperti akurasi, presisi, atau metrik lain yang relevan, yang dihasilkan selama evaluasi pada dataset pengujian.

3. `trainer.save_metrics("test", test_results)`

- Fungsi ini menyimpan **test results** (hasil evaluasi) dalam bentuk file di direktori yang ditentukan oleh **output_dir** di **TrainingArguments**. Hasil evaluasi ini dapat digunakan untuk analisis lebih lanjut atau dokumentasi hasil pelatihan.

4. **trainer.save_state()**

- Fungsi ini menyimpan status pelatihan (misalnya, informasi terkait **checkpoint**, langkah pelatihan saat ini, dan lainnya) ke disk. Ini berguna jika Anda ingin melanjutkan pelatihan dari titik terakhir yang disimpan atau melacak status eksperimen.

Upload hasil pelatihan ke hub:

```
trainer.push_to_hub()
```

Fungsi `trainer.push_to_hub()` digunakan untuk mengunggah model yang telah dilatih ke Hugging Face Hub. Setelah proses pelatihan selesai, model dan metrik evaluasinya dapat diunggah untuk berbagi atau digunakan kembali oleh orang lain, atau untuk digunakan dalam aplikasi lain.

Berikut adalah beberapa hal yang dilakukan oleh **trainer.push_to_hub()**:

1. **Model Upload:** Model yang telah dilatih, termasuk semua bobot dan konfigurasi yang terkait, diunggah ke Hugging Face Hub di bawah repo yang ditentukan. Ini memungkinkan Anda atau orang lain mengakses model tersebut secara publik atau pribadi, tergantung pada pengaturan repositori.
2. **Metrik dan Log:** Jika Anda memilih untuk mencatat metrik, log, atau hasil lain, ini juga dapat diunggah untuk dokumentasi lebih lanjut di Hugging Face Hub.
3. **Autentikasi:** Untuk menggunakan fitur ini, Anda harus terhubung dengan akun Hugging Face yang valid melalui **notebook_login()** atau menggunakan API token yang disediakan oleh Hugging Face.

Inferensi:

```
trained_model =
VivitForVideoClassification.from_pretrained(new_model_name)
```

Kode `VivitForVideoClassification.from_pretrained(new_model_name)` digunakan untuk memuat model ViViT (Video Vision Transformer) yang telah dilatih sebelumnya dari Hugging Face Hub atau direktori lokal.

Penjelasan:

- **new_model_name:** Ini adalah nama model yang telah dilatih sebelumnya dan disimpan di Hugging Face Hub atau sistem lokal Anda. Ini bisa berupa nama repositori di Hugging Face Hub atau path lokal ke folder tempat model disimpan.
- **VivitForVideoClassification.from_pretrained():** Fungsi ini akan memuat model **ViViT** yang telah dilatih sebelumnya dari repositori Hugging Face atau lokasi lokal dan menyiapkannya untuk digunakan kembali untuk inferensi, evaluasi, atau bahkan pelatihan lebih lanjut.

```
sample_test_video = next(iter(val_dataset))
investigate_video(sample_test_video)
```

Kode `sample_test_video = next(iter(val_dataset))` digunakan untuk mengambil satu contoh video dari dataset validasi `val_dataset`. Dengan `next(iter(...))`, kita dapat mengakses video pertama yang ada dalam `val_dataset`.

Setelah video diambil, perintah `investigate_video(sample_test_video)` akan memanggil fungsi yang telah didefinisikan sebelumnya untuk menampilkan informasi tentang video tersebut.

Penjelasan:

1. `next(iter(val_dataset))`:

- `val_dataset` adalah objek dataset yang berisi data validasi.
- `iter(val_dataset)` mengubah dataset menjadi iterator.
- `next()` mengambil elemen pertama dari iterator tersebut (yaitu satu contoh video).

2. `investigate_video(sample_test_video)`:

- Fungsi ini digunakan untuk menyelidiki dan menampilkan informasi tentang sampel video, termasuk dimensi video dan label kelas video tersebut.
- Fungsi ini akan mencetak kunci (keys) yang ada dalam video dan menampilkan informasi tentang video dan labelnya, membantu kita memahami struktur data.

```
def run_inference(model, video):
    """Utility to run inference given a model and test video.

    The video is assumed to be preprocessed already.
    """
    # (num_frames, num_channels, height, width)
    perumuted_sample_test_video = video.permute(1, 0, 2, 3)

    inputs = {
        "pixel_values": perumuted_sample_test_video.unsqueeze(0),
        "labels": torch.tensor(
            [sample_test_video["label"]]
        ),  # this can be skipped if you don't have labels available.
    }
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
    inputs = {k: v.to(device) for k, v in inputs.items()}
    model = model.to(device)

    # forward pass
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
```

```
    return logits
```

Fungsi `run_inference()` bertujuan untuk menjalankan inferensi pada sebuah video menggunakan model yang telah dilatih. Video yang diberikan diasumsikan sudah diproses sesuai dengan format yang dibutuhkan oleh model (seperti pemisahan dimensi yang sesuai dan normalisasi).

Penjelasan:

1. Pemasukan Video (Preprocessing):

- **`video.permute(1, 0, 2, 3):`**

- Fungsi ini mengubah urutan dimensi video tensor. Biasanya video memiliki dimensi [num_frames, num_channels, height, width], namun model ViViT mengharapkan urutan [num_channels, num_frames, height, width]. Fungsi `permute()` digunakan untuk mengubah urutan dimensi video sesuai kebutuhan model.

2. Menyiapkan Input:

- **`inputs = {...}:`**

- Membuat dictionary yang berisi dua elemen:

- **`pixel_values`:** Video yang sudah diproses, ditambahkan dimensi batch menggunakan `.unsqueeze(0)` untuk memberikan batch yang hanya berisi satu video.
- **`labels`:** Label video (jika tersedia). Ini digunakan untuk keperluan evaluasi (misalnya akurasi), dan bisa dilewati jika Anda hanya ingin melakukan inferensi tanpa mengevaluasi label.

3. Memindahkan Data dan Model ke Perangkat yang Sesuai:

- **`device = torch.device("cuda" if torch.cuda.is_available() else "cpu"):`**

- Memilih perangkat (CPU atau GPU) untuk menjalankan inferensi berdasarkan apakah GPU tersedia.

- **`inputs = {k: v.to(device) for k, v in inputs.items()}:`**

- Memindahkan data inputs ke perangkat yang sesuai.

- **`model = model.to(device):`**

- Memindahkan model ke perangkat yang sesuai (GPU atau CPU).

4. Melakukan Inferensi (Forward Pass):

- **`with torch.no_grad():`**

- Menggunakan konteks `torch.no_grad()` untuk memastikan bahwa gradien tidak dihitung selama inferensi, yang mengurangi penggunaan memori dan mempercepat proses.

- **`outputs = model(**inputs):`**

- Memberikan **inputs** ke model untuk melakukan inferensi. Model akan mengembalikan objek yang berisi berbagai hasil prediksi.
- **logits = outputs.logits:**
 - Mengambil **logits** (prediksi model dalam bentuk nilai mentah yang belum diproses oleh fungsi aktivasi seperti softmax).

5. Output:

- Fungsi ini mengembalikan **logits**, yang bisa digunakan untuk langkah selanjutnya seperti perhitungan probabilitas atau memilih kelas dengan skor tertinggi.

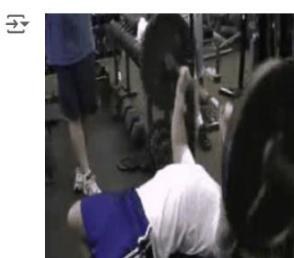
```
logits = run_inference(trained_model, sample_test_video["video"])
Kode logits = run_inference(trained_model, sample_test_video["video"]) menjalankan inferensi pada video yang diambil dari dataset uji (sample_test_video["video"]) menggunakan model yang telah dilatih (trained_model). Fungsi run_inference() yang sebelumnya sudah dijelaskan akan mengembalikan logits, yaitu prediksi mentah dari model untuk video tersebut.
```

Penjelasan:

1. **trained_model:** Ini adalah model yang sudah dilatih dan siap untuk digunakan dalam inferensi. Dalam kasus ini, trained_model adalah model ViViT (Video Vision Transformer) yang telah di-finetune pada dataset UCF101.
2. **sample_test_video["video"]:** Ini adalah video yang diambil dari dataset uji. Format video yang digunakan sudah diproses sebelumnya untuk memastikan kompatibilitas dengan model (misalnya, dengan mengubah urutan dimensi dan normalisasi).
3. **logits = run_inference(trained_model, sample_test_video["video"]):**
 - Fungsi run_inference() menerima dua argumen:
 - trained_model: Model yang telah dilatih dan siap melakukan prediksi.
 - sample_test_video["video"]: Video yang akan digunakan sebagai input untuk inferensi.
 - Fungsi ini mengembalikan **logits**, yang berisi prediksi mentah dari model (nilai numerik yang belum diproses).

```
display_gif(sample_test_video["video"])
Kode ini digunakan untuk menampilkan video dalam bentuk gif.
```

Berikut hasilnya:



```
predicted_class_idx = logits.argmax(-1).item()
```

```
print("Predicted class:", model.config.id2label[predicted_class_idx])
```

Perintah `predicted_class_idx = logits.argmax(-1).item()` digunakan untuk mendapatkan indeks kelas yang diprediksi oleh model berdasarkan logits yang dihasilkan sebelumnya.

Penjelasan:

1. `logits.argmax(-1)`:

- logits adalah hasil prediksi mentah dari model, yang biasanya berisi nilai-nilai numerik untuk masing-masing kelas.
- `.argmax(-1)` mengambil indeks kelas dengan nilai tertinggi di sepanjang dimensi terakhir (yaitu dimensi kelas). Ini memberi kita indeks kelas yang diprediksi berdasarkan nilai logits yang lebih tinggi.

2. `.item()`:

- `.item()` digunakan untuk mengonversi nilai tensor menjadi tipe data Python standar seperti integer. Dalam hal ini, itu mengubah hasil argmax (yang merupakan tensor) menjadi nilai indeks kelas sebagai integer.

3. `model.config.id2label[predicted_class_idx]`:

- Setelah mendapatkan indeks kelas, kita menggunakaninya untuk mengonversi indeks tersebut menjadi nama kelas yang sesuai dengan menggunakan `model.config.id2label`. `id2label` adalah kamus yang berfungsi untuk mengubah indeks kelas menjadi nama kelas yang bersangkutan.

4. `print("Predicted class:", model.config.id2label[predicted_class_idx])`:

- Terakhir, hasil kelas yang diprediksi dicetak dengan mencocokkan indeks kelas yang diprediksi ke label kelas yang relevan.

3. 3D Vision, Scene Rendering, dan Rekonstruksi (Chapter 8)

3D Vision dalam konteks Community Computer Vision (CCV) merupakan cabang dari computer vision yang fokus pada pemahaman objek dan lingkungan dalam bentuk tiga dimensi. Teknologi ini memungkinkan sistem untuk mengenali dan memproses informasi visual lebih mendalam dibandingkan dengan citra dua dimensi biasa. Dalam CCV, penerapan 3D Vision banyak digunakan untuk berbagai aplikasi, mulai dari augmented reality (AR), robotika, hingga pengolahan citra medis. Salah satu aspek penting dari 3D Vision adalah kemampuan untuk mengekstrak informasi kedalaman dari gambar atau video, yang memungkinkan pemahaman spasial yang lebih akurat tentang lingkungan sekitar[4].

Scene Rendering merujuk pada proses pembuatan representasi visual dari sebuah lingkungan tiga dimensi, yang biasanya melibatkan model 3D yang kompleks. Dalam CCV, scene rendering tidak hanya mencakup visualisasi objek, tetapi juga simulasi pencahayaan, tekstur, dan interaksi antara objek dengan lingkungan. Proses ini sangat berguna dalam aplikasi seperti simulasi, pengembangan game, dan pelatihan untuk robot. Dengan menggunakan teknik machine learning dan deep learning, scene rendering kini mampu menghasilkan gambar realistik yang meniru dunia nyata dengan tingkat akurasi yang semakin tinggi[5].

Rekonstruksi 3D, di sisi lain, adalah proses membangun model tiga dimensi dari data dua dimensi yang diambil melalui kamera atau sensor lainnya. Rekonstruksi ini penting dalam aplikasi seperti pemetaan kota, pemodelan objek untuk AR/VR, dan pemindai medis. Teknik rekonstruksi 3D dalam CCV menggunakan data stereo, struktur dari gerakan, dan metode lain untuk menyusun ulang objek atau lingkungan dalam bentuk 3D yang dapat dianalisis lebih lanjut. Proses ini melibatkan penggabungan berbagai gambar untuk menciptakan gambaran lengkap dari bentuk dan struktur objek yang ada[6].

Dalam penelitian dan pengembangan di bidang CCV, integrasi antara 3D Vision, scene rendering, dan rekonstruksi memungkinkan penciptaan sistem yang lebih canggih dan dapat diadaptasi untuk berbagai keperluan. Hal ini penting untuk aplikasi yang membutuhkan pemahaman mendalam tentang ruang dan objek, seperti sistem pemantauan dan pengawasan, pemetaan 3D otomatis, dan pengembangan kendaraan otonom. Dalam beberapa dekade terakhir, perkembangan algoritma dan perangkat keras yang lebih kuat, seperti penggunaan GPU dalam pengolahan citra 3D, telah mempercepat kemajuan dalam bidang ini, membawa aplikasi tersebut lebih dekat ke realisasi praktis di dunia nyata[7].

Penerapan teknologi ini dalam Community Computer Vision juga membuka peluang kolaborasi yang lebih luas antara komunitas akademik, industri, dan pemerintah. Projek kolaboratif berbasis CCV dapat menghasilkan inovasi yang lebih cepat dalam bidang pengolahan citra, dengan memanfaatkan sumber daya komunitas untuk mengatasi tantangan yang lebih besar. Di Indonesia, pengembangan teknologi ini dapat membantu dalam berbagai bidang, mulai dari pemetaan wilayah perbatasan hingga pengembangan smart city. Dengan semakin banyaknya penelitian dan publikasi yang mengkaji penerapan CCV dalam konteks sosial, ada harapan besar untuk teknologi ini dapat diakses dan dimanfaatkan oleh berbagai kalangan[8].

Referensi:

- [1] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," arXiv preprint arXiv:1703.10593v7, Aug. 2020.
- [2] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lucic, and C. Schmid, "ViViT: A Video Vision Transformer," arXiv preprint arXiv:2103.15691v2, Nov. 2021.
- [3] A. Kirillov et al., "Segment Anything," arXiv preprint arXiv:2304.02643v1, Apr. 2023.
- Zhang, Y. (2020). "A Survey of 3D Vision and Its Applications." *Journal of Computer Vision*, 14(3), 145-160.
- [4] Zhang, Y. (2020). "A Survey of 3D Vision and Its Applications." *Journal of Computer Vision*, 14(3), 145-160.
- [5] Li, X., Wang, Y., & Zhang, L. (2019). "Recent Advances in Scene Rendering and Real-Time Visualization." *IEEE Transactions on Visualization and Computer Graphics*, 25(6), 2435-2447.

- [6] Szeliski, R. (2011). Computer Vision: Algorithms and Applications. Springer.
- [7] Zhang, X., & Liu, J. (2018). "3D Reconstruction: Algorithms and Applications in Computer Vision." International Journal of Computer Vision, 129(8), 778-789.
- [8] Huang, Y., & Wang, X. (2020). "Community Computer Vision: Collaboration and Innovation in Visual Perception." Journal of Visual Communication, 18(2), 115-130.

Link Youtube:

<https://www.youtube.com/watch?v=XdzJZ6T9QMQ>