Nama : Nabilah Salwa

NIM : 1103204060

UAS Machine Learning

04. PyTorch Custom Datasets Exercise

```
# Check for GPU
!nvidia-smi
```

Kode `!nvidia-smi` merupakan sebuah perintah yang biasanya digunakan dalam lingkungan Jupyter Notebook atau terminal untuk memeriksa informasi terkait GPU (Graphics Processing Unit) Nvidia pada suatu sistem. Perintah ini menggunakan `!` untuk mengeksekusi perintah di luar lingkungan Python, dan `nvidia-smi` adalah singkatan dari Nvidia System Management Interface. Saat dieksekusi, perintah ini mengembalikan informasi waktu nyata tentang penggunaan GPU, alokasi memori, suhu, dan detail lainnya. Hal ini berguna terutama dalam konteks pengembangan dan penggunaan deep learning, di mana GPU sering digunakan untuk mempercepat komputasi. Output dari perintah ini menampilkan tabel yang memberikan informasi terperinci tentang status dan penggunaan GPU Nvidia yang terdeteksi pada sistem.

```
# Import torch
import torch

# Exercises require PyTorch > 1.10.0
print(torch.__version__)

# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

Kode di atas pertama-tama mengimpor library PyTorch dan mencetak versi yang diinstal. Selanjutnya, kode menentukan variabel `device` untuk menentukan apakah sistem mendukung GPU atau tidak. Jika GPU tersedia, `device` diatur sebagai "cuda", jika tidak, diatur sebagai "cpu". Variabel `device` kemudian dicetak untuk memberikan informasi tentang penggunaan perangkat keras pada sistem, khususnya apakah GPU dapat digunakan atau tidak.

1. Recreate the data loading functions we built in sections 1, 2, 3 and 4 of notebook 04. You should have train and test

```python
# 1. Get data
import requests
import zipfile
from pathlib import Path

# Setup path to data folder
data_path = Path("data/")
image_path = data_path / "pizza_steak_sushi"

# If the image folder doesn't exist, download it and prepare it...
if image_path.is_dir():
  print(f"{image_path} directory exists.")
else:
  print(f"Did not find {image_path} directory, creating...")
  image_path.mkdir(parents=True, exist_ok=True)

# Download pizza, steak, sushi data (images from GitHub)
with open(data_path / "pizza_steak_sushi.zip", "wb") as f:
  request = requests.get("https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza_steak_sushi.zip")
  print("Downloading pizza, steak, sushi data...")
  f.write(request.content)

# Unzip pizza, steak, sushi data
with zipfile.ZipFile(data_path/"pizza_steak_sushi.zip", "r") as zip_ref:
  print(f"Unzipping pizza, steak, suhsi data to {image_path}")
  zip_ref.extractall(image_path)
```

Kode tersebut bertujuan untuk memperoleh dataset yang terdiri dari gambar-gambar pizza, steak, dan sushi. Pertama, kode menentukan path untuk menyimpan dataset dan membuat direktori jika belum ada. Selanjutnya, kode mengunduh file zip yang berisi gambar-gambar tersebut dari repositori GitHub menggunakan modul `requests`, dan kemudian mengekstraknya ke direktori yang telah ditentukan menggunakan modul `zipfile`. Jika direktori sudah ada, pesan mencetak bahwa direktori tersebut sudah ada; jika tidak, pesan mencetak bahwa direktori sedang dibuat. Dataset ini umumnya digunakan untuk latihan dalam pembelajaran mesin atau deep learning, terutama ketika membangun model untuk mengenali kategori makanan.

```python
# 2. Become one with the data
import os
def walk_through_dir(dir_path):
  """Walks through dir_path returning file counts of its contents."""
  for dirpath, dirnames, filenames in os.walk(dir_path):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

Kode tersebut mendefinisikan fungsi `walk_through_dir(dir_path)` yang mengiterasi melalui isi direktori pada `dir_path` menggunakan modul `os.walk()`. Selama iterasi, fungsi mencetak jumlah sub-direktori dan jumlah file gambar dalam setiap direktori yang ditemui. Fungsi ini bertujuan memberikan pemahaman tentang struktur dataset, termasuk berapa banyak sub-direktori dan gambar yang ada di dalamnya. Hal ini membantu pengguna memahami secara rinci distribusi data dalam dataset yang telah diunduh dan diekstrak sebelumnya.

```
walk_through_dir(image_path)
```

Panggilan fungsi `walk_through_dir(image_path)` bertujuan untuk memberikan informasi terperinci tentang struktur direktori dataset yang telah diunduh dan diekstrak sebelumnya. Fungsi tersebut menggunakan modul `os.walk()` untuk mengiterasi melalui semua sub-direktori dan file dalam direktori yang ditentukan (`image_path`). Setiap iterasi mencetak jumlah sub-direktori dan jumlah file gambar di setiap direktori, memberikan gambaran tentang distribusi kelas dan jumlah gambar yang ada dalam dataset, yang berguna untuk pemahaman awal sebelum memproses data lebih lanjut dalam pembangunan model machine learning atau deep learning.

```
# Setup train and testing paths
train_dir = image_path / "train"
test_dir = image_path / "test"

train_dir, test_dir
```

Kode tersebut menetapkan path untuk direktori pelatihan ("train_dir") dan direktori pengujian ("test_dir") dalam dataset. Dengan membagi dataset menjadi direktori pelatihan dan pengujian, umumnya digunakan dalam pembelajaran mesin untuk mengevaluasi kinerja model pada data yang tidak digunakan selama pelatihan. Pemisahan ini membantu mengukur sejauh mana model dapat generalisasi pada data baru. Dengan menetapkan path untuk kedua direktori ini, kode ini mempersiapkan struktur direktori untuk memisahkan data pelatihan dan pengujian dalam pembangunan model machine learning atau deep learning selanjutnya.

```python
# Visualize an image
import random
from PIL import Image

# Set seed
# random.seed(42)

# 1. Get all image paths (* means "any combination")
image_path_list = list(image_path.glob("*/*/*.jpg"))
print(image_path_list[:3])

# 2. Get random image path
random_image_path = random.choice(image_path_list)
print(random_image_path)

# 3. Get image class from path name
image_class = random_image_path.parent.stem
print(image_class)

# 4. Open image
img = Image.open(random_image_path)

# Print metadata
print(f"Random image path: {random_image_path}")
print(f"Image class: {image_class}")
print(f"Image height: {img.height}")
print(f"Image width: {img.width}")
img
# Image.open("/content/data/pizza_steak_sushi/test/pizza/194643.jpg")
```

Kode di atas memiliki tujuan untuk memvisualisasikan sebuah gambar secara acak dari dataset yang telah disiapkan sebelumnya. Pertama, kode mengumpulkan semua path gambar dalam bentuk list menggunakan `glob` dan mencetak tiga path pertama. Selanjutnya, kode memilih secara acak satu path gambar dari list tersebut, mengekstrak kelas gambar dari struktur path, dan membuka gambar menggunakan modul PIL (Python Imaging Library). Metadata dari gambar seperti path, kelas, tinggi, dan lebar kemudian dicetak, dan gambar ditampilkan untuk visualisasi. Pemahaman tentang metadata dan tampilan visual ini membantu pengguna memahami karakteristik dasar dari dataset yang akan digunakan dalam proses pembangunan model.

```python
# Do the image visualization with matplotlib
import numpy as np
import matplotlib.pyplot as plt

# Turn the image into an array
img_as_array = np.asarray(img)

# Plot the image
plt.figure(figsize=(10, 7))
plt.imshow(img_as_array)
plt.title(f"Image class: {image_class} | Image shape: {img_as_array.shape} -> [height, width, color_channels]")
plt.axis(False);
```

Kode di atas menggunakan Matplotlib untuk memvisualisasikan gambar yang telah dibuka sebelumnya. Pertama, gambar diubah menjadi representasi array menggunakan NumPy.

Selanjutnya, menggunakan fungsi `plt.imshow()`, gambar dirender pada suatu plot dengan informasi tambahan seperti kelas gambar dan dimensi array gambar. Ukuran plot diatur dengan `plt.figure(figsize=(10, 7))`, dan parameter `plt.axis(False)` digunakan untuk menghilangkan tanda sumbu. Visualisasi ini memberikan pandangan langsung terhadap konten gambar dan membantu pengguna untuk memahami secara visual representasi dari data gambar dalam dataset yang digunakan.

```
# 3.1 Transforming data with torchvision.transforms
import torch
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

Kode di atas menunjukkan bagian dari proses transformasi data menggunakan modul `torchvision.transforms` dari PyTorch. Transformasi ini umumnya digunakan dalam pembangunan model deep learning untuk mempersiapkan data sebelum dimasukkan ke dalam model. Dalam hal ini, transformasi data seperti normalisasi dan augmentasi dapat diterapkan menggunakan modul tersebut. Transformasi ini kemudian dapat digunakan bersama dengan modul `datasets` dan `DataLoader` dari PyTorch untuk membuat iterator data yang akan digunakan selama proses pelatihan model. Transformasi data adalah langkah penting dalam memastikan data siap untuk diproses oleh model, dan `torchvision.transforms` menyediakan berbagai fungsi untuk tujuan tersebut.

```
# Write transform for turning images into tensors
data_transform = transforms.Compose([
  # Resize the images to 64x64x3 (64 height, 64 width, 3 color channels)
  transforms.Resize(size=(64, 64)),
  # Flip the images randomly on horizontal
  transforms.RandomHorizontalFlip(p=0.5),
  # Turn the image into a torch.Tensor
  transforms.ToTensor() # converts all pixel values from 0-255 to be between 0-1
])
```

Kode di atas mendefinisikan transformasi data menggunakan modul `transforms` dari PyTorch. Transformasi ini dirancang untuk memproses gambar menjadi format yang cocok untuk pelatihan model. Transformasi tersebut mencakup resize gambar menjadi ukuran 64x64 piksel dengan tiga saluran warna, penerapan flip horizontal secara acak dengan peluang 50%, dan mengubah gambar menjadi tensor PyTorch. Hal ini memastikan bahwa gambar-gambar dalam dataset akan diubah menjadi tensor yang dapat digunakan oleh model dalam proses pembelajaran. Transformasi ini umumnya digunakan untuk meningkatkan keberagaman data pelatihan dan memastikan bahwa data sesuai dengan format yang diharapkan oleh model.

```
random.sample(image_path_list, k=3)
```

Kode di atas menggunakan fungsi `random.sample()` untuk mengambil tiga elemen secara acak dari list `image_path_list`. Fungsi ini memungkinkan pengguna untuk secara acak memilih beberapa elemen dari suatu populasi tanpa penggantian. Dalam konteks ini, `image_path_list` merupakan list yang berisi path gambar-gambar dalam dataset. Dengan

menggunakan `random.sample(image_path_list, k=3)`, tiga path gambar dipilih secara acak untuk tujuan yang mungkin mencakup pengujian atau visualisasi dataset.

```
# Write a function to plot transformed images
def plot_transformed_images(image_paths, transform, n=3, seed=42):
  """Plots a series of random images from image_paths."""
  random.seed(seed)
  random_image_paths = random.sample(image_paths, k=n)
  for image_path in random_image_paths:
    with Image.open(image_path) as f:
      fig, ax = plt.subplots(nrows=1, ncols=2)
      ax[0].imshow(f)
      ax[0].set_title(f"Original \nsize: {f.size}")
      ax[0].axis("off")

      # Transform and plot image
      # permute() the image to make sure it's compatible with matplotlib
      transformed_image = transform(f).permute(1, 2, 0)
      ax[1].imshow(transformed_image)
      ax[1].set_title(f"Transformed \nsize: {transformed_image.shape}")
      ax[1].axis("off")

      fig.suptitle(f"Class: {image_path.parent.stem}", fontsize=16)

plot_transformed_images(image_path_list,
                        transform=data_transform,
                        n=3)
```

Kode di atas mendefinisikan fungsi `plot_transformed_images`, yang bertujuan untuk memvisualisasikan transformasi dari beberapa gambar secara acak dalam dataset. Fungsi ini menerima path gambar, transformasi, jumlah gambar yang akan divisualisasikan (parameter `n`), dan seed untuk keperluan pengacakan. Setiap gambar diproses oleh transformasi yang telah ditentukan sebelumnya (`data_transform`) dan dua subplot diperlihatkan: subplot pertama menampilkan gambar asli beserta ukurannya, sementara subplot kedua menampilkan gambar yang telah melalui transformasi beserta dimensinya. Informasi kelas gambar juga ditambahkan sebagai judul utama pada setiap subplot. Fungsi ini membantu pengguna untuk memahami efek dari transformasi yang diterapkan pada dataset, berguna dalam pemrosesan dan pemahaman awal terhadap data sebelum diterapkan ke dalam model.

```
# Use ImageFolder to create dataset(s)
from torchvision import datasets
train_data = datasets.ImageFolder(root=train_dir, # target folder of images
                                  transform=data_transform, # transforms to perform on data (images)
                                  target_transform=None) # transforms to perform on labels (if necessary)

test_data = datasets.ImageFolder(root=test_dir,
                                 transform=data_transform,
                                 target_transform=None)

train_data, test_data
```

Kode di atas menggunakan kelas `ImageFolder` dari modul `torchvision.datasets` untuk membuat dataset pelatihan (`train_data`) dan dataset pengujian (`test_data`). Kedua dataset

ini dibentuk berdasarkan struktur direktori yang telah ditetapkan sebelumnya untuk data pelatihan dan pengujian. Transformasi data (`data_transform`) juga diterapkan pada dataset menggunakan parameter `transform`. `ImageFolder` secara otomatis mengenali struktur direktori dan mengasumsikan bahwa setiap sub-direktori dalam direktori utama merupakan kelas yang berbeda, dengan gambar-gambar yang terdapat di dalamnya sebagai contoh-contoh dari kelas tersebut. Dataset yang terbentuk ini nantinya dapat digunakan bersama dengan `DataLoader` untuk memproses data dalam batch selama pelatihan atau pengujian model.

```
# Get class names as a list
class_names = train_data.classes
class_names
```

```
['pizza', 'steak', 'sushi']
```

```
# Can also get class names as a dict
class_dict = train_data.class_to_idx
class_dict
```

```
{'pizza': 0, 'steak': 1, 'sushi': 2}
```

```
# Check the lengths
len(train_data), len(test_data)
```

```
(225, 75)
```

Kode di atas mengambil informasi tentang nama kelas dalam dataset, baik dalam bentuk list (`class_names`) maupun dictionary (`class_dict`). List `class_names` berisi nama-nama kelas yang diakses melalui atribut `classes` dari dataset pelatihan. Selain itu, dictionary `class_dict` dibentuk melalui atribut `class_to_idx`, yang memetakan nama kelas ke indeks numerik. Informasi ini berguna untuk pemahaman terkait struktur kelas dalam dataset dan dapat digunakan untuk interpretasi hasil keluaran model. Selain itu, panjang dataset pelatihan dan pengujian juga diperiksa melalui fungsi `len()` untuk memastikan jumlah sampel yang ada pada masing-masing dataset.

```
# Turn train and test Datasets into DataLoaders
from torch.utils.data import DataLoader
BATCH_SIZE = 1
train_dataloader = DataLoader(dataset=train_data,
                              batch_size=BATCH_SIZE,
                              num_workers=os.cpu_count(),
                              shuffle=True)

test_dataloader = DataLoader(dataset=test_data,
                             batch_size=BATCH_SIZE,
                             num_workers=os.cpu_count(),
                             shuffle=False)

train_dataloader, test_dataloader
```

Kode di atas menggunakan modul `DataLoader` dari PyTorch untuk mengonversi dataset pelatihan (`train_data`) dan dataset pengujian (`test_data`) menjadi iterator (`train_dataloader` dan `test_dataloader`), yang dapat digunakan untuk memproses data dalam batch selama pelatihan atau pengujian model. Pengaturan seperti ukuran batch (`BATCH_SIZE`), jumlah pekerja (`num_workers`) yang bekerja paralel untuk memuat data, dan pengacakan data (`shuffle`) dapat disesuaikan sesuai kebutuhan. Iterator ini berguna dalam mengelola data secara efisien dan memastikan model mendapatkan sejumlah data yang representatif selama pelatihan dan pengujian.

```
# How many batches of images are in our data loaders?
len(train_dataloader), len(test_dataloader)
```

Kode di atas menggunakan fungsi `len()` untuk menghitung jumlah batch dalam dataloader pelatihan (`train_dataloader`) dan dataloader pengujian (`test_dataloader`). Panjang dataloader diukur dalam jumlah batch, di mana setiap batch memuat sejumlah gambar sesuai dengan ukuran batch yang telah ditetapkan sebelumnya. Informasi ini berguna untuk memahami seberapa banyak batch yang akan digunakan dalam setiap iterasi selama pelatihan atau pengujian model, memberikan gambaran tentang seberapa besar dataset yang akan diakses oleh model pada setiap langkah iteratifnya.

```
img, label = next(iter(train_dataloader))

# Batch size will now be 1, try changing the batch_size parameter above and see what happens
print(f"Image shape: {img.shape} -> [batch_size, color_channels, height, width]")
print(f"Label shape: {label.shape}")
```

Kode di atas menggunakan fungsi `next(iter(train_dataloader))` untuk mengambil satu batch pertama dari dataloader pelatihan (`train_dataloader`). Batch ini terdiri dari gambar (`img`) dan label (`label`) yang diambil dari dataset pelatihan. Dengan mencetak bentuk (`shape`) dari batch tersebut, kita dapat melihat dimensi dan struktur data dalam batch, di mana `img.shape` memberikan informasi tentang dimensi gambar (batch size, jumlah saluran warna, tinggi, lebar), dan `label.shape` memberikan informasi tentang dimensi label. Pengguna dapat mencoba mengubah parameter `BATCH_SIZE` pada dataloader sebelumnya untuk mengamati perbedaan dalam bentuk batch yang dihasilkan.

2. Recreate model_0 we built in section 7 of notebook 04.

```
import torch
from torch import nn

class TinyVGG(nn.Module):
  def __init__(self, input_shape, hidden_units, output_shape):
    super().__init__()
    self.conv_block_1 = nn.Sequential(
        nn.Conv2d(in_channels=input_shape,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=1),
        nn.ReLU(),
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2)
    )
    self.conv_block_2 = nn.Sequential(
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=1),
```

```
        nn.ReLU(),
        nn.Conv2d(in_channels=hidden_units,
                  out_channels=hidden_units,
                  kernel_size=3,
                  stride=1,
                  padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2)
    )
    self.classifier = nn.Sequential(
        nn.Flatten(),
        nn.Linear(in_features=hidden_units*16*16,
                  out_features=output_shape))

def forward(self, x):
    x = self.conv_block_1(x)
    # print(f"Layer 1 shape: {x.shape}")
    x = self.conv_block_2(x)
    # print(f"Layer 2 shape: {x.shape}")
    x = self.classifier(x)
    # print(f"Layer 3 shape: {x.shape}")
    return x
```

Kode di atas mendefinisikan sebuah model neural network sederhana bernama `TinyVGG`. Model ini terinspirasi dari arsitektur VGG dan terdiri dari dua blok konvolusi (`conv_block_1` dan `conv_block_2`) yang masing-masing memiliki dua lapisan konvolusi dengan aktivasi ReLU diikuti oleh operasi max pooling. Blok konvolusi ini diikuti oleh lapisan linear

(`classifier`) yang bertugas menggabungkan fitur-fitur hasil ekstraksi dari blok konvolusi ke dalam lapisan linear dengan fungsi aktivasi linear. Model ini dirancang untuk tugas klasifikasi, di mana inputnya diharapkan berupa gambar dengan jumlah saluran warna sesuai dengan `input_shape`. Arsitektur dan ukuran lapisan model dapat disesuaikan dengan mengubah parameter-parameter seperti `input_shape`, `hidden_units`, dan `output_shape`. Metode `forward` digunakan untuk mendefinisikan aliran maju atau forward pass dalam model.

```
model_0 = TinyVGG(input_shape = 3,
                  hidden_units=10,
                  output_shape=len(class_names)).to(device)
model_0
```

Kode di atas membuat sebuah instance dari model `TinyVGG` dengan nama `model_0`. Model ini dikonfigurasi dengan parameter-parameter seperti `input_shape=3` (mewakili jumlah saluran warna pada gambar), `hidden_units=10` (menentukan jumlah filter pada lapisan konvolusi), dan `output_shape=len(class_names)` (mengacu pada jumlah kelas dalam dataset). Model ini kemudian dipindahkan ke perangkat keras yang telah ditentukan sebelumnya (`device`). Proses ini dapat melibatkan penggunaan GPU jika GPU tersedia dan diaktifkan. Model ini dapat diperiksa untuk memastikan struktur dan konfigurasinya dengan mencetak `model_0`. Model yang telah dibuat ini dapat digunakan untuk melatih dan menguji klasifikasi gambar pada dataset yang telah disiapkan sebelumnya.

```
len(class_names)
```

```
3
```

```
16*16*10
```

```
2560
```

```
# Pass dummy data through model
dummy_x = torch.rand(size=[1, 3, 64, 64])
model_0(dummy_x.to(device))
```

```
tensor([[ 0.0169,  0.0104, -0.0204]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
```

Kode pertama (`len(class_names)`) menghasilkan nilai yang menunjukkan jumlah kelas dalam dataset, di mana `class_names` merupakan list yang berisi nama-nama kelas. Kode kedua (`16*16*10`) merupakan perhitungan teoritis yang mencoba memberikan estimasi jumlah parameter yang akan dipelajari oleh model. Ini didasarkan pada ukuran dimensi fitur keluaran dari lapisan konvolusi terakhir (`hidden_units*16*16`) dan jumlah neuron pada lapisan linear (`output_shape`). Kode terakhir membuat data dummy (`dummy_x`) dengan ukuran batch 1 dan dimensi gambar 3 saluran warna, 64 piksel tinggi, dan 64 piksel lebar. Data dummy ini kemudian diteruskan melalui model (`model_0`) yang telah dibuat sebelumnya menggunakan perangkat keras yang telah ditentukan (`device`), memberikan keluaran prediksi model untuk data dummy tersebut. Proses ini membantu memastikan bahwa

model dapat melakukan forward pass dengan benar dan memberikan keluaran yang sesuai dengan struktur yang diharapkan.

3. Create training and testing functions for model_0.

```
def train_step(model: torch.nn.Module,
               dataloader: torch.utils.data.DataLoader,
               loss_fn: torch.nn.Module,
               optimizer: torch.optim.Optimizer):

  # Put the model in train mode
  model.train()

  # Setup train loss and train accuracy values
  train_loss, train_acc = 0, 0

  # Loop through data loader and data batches
  for batch, (X, y) in enumerate(dataloader):
    # Send data to target device
    X, y = X.to(device), y.to(device)

    # 1. Forward pass
    y_pred = model(X)
    # print(y_pred)

    # 2. Calculate and accumulate loss
    loss = loss_fn(y_pred, y)
    train_loss += loss.item()

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    # Calculate and accumualte accuracy metric across all batches
    y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1)
    train_acc += (y_pred_class == y).sum().item()/len(y_pred)

  # Adjust metrics to get average loss and average accuracy per batch
  train_loss = train_loss / len(dataloader)
  train_acc = train_acc / len(dataloader)
  return train_loss, train_acc
```

Fungsi `train_step` di atas digunakan untuk melatih model pada satu epoch melalui data yang diberikan oleh dataloader pelatihan. Dalam setiap iterasi, fungsi ini mengirimkan data ke perangkat keras yang ditentukan (`device`), melakukan forward pass untuk mendapatkan prediksi (`y_pred`), menghitung dan mengakumulasi nilai loss menggunakan fungsi loss yang telah diberikan, melakukan backpropagation (`loss.backward()`), dan mengoptimalkan parameter model menggunakan optimizer. Selain itu, akurasi pelatihan juga dihitung dan diakumulasi untuk setiap batch. Pada akhir iterasi, nilai loss dan akurasi dihitung ulang

sebagai rata-rata per batch. Fungsi ini memberikan informasi tentang performa model selama pelatihan dan dapat digunakan untuk memantau kemajuan pelatihan.

```python
def test_step(model: torch.nn.Module,
              dataloader: torch.utils.data.DataLoader,
              loss_fn: torch.nn.Module):

  # Put model in eval mode
  model.eval()

  # Setup the test loss and test accuracy values
  test_loss, test_acc = 0, 0

  # Turn on inference context manager
  with torch.inference_mode():
    # Loop through DataLoader batches
    for batch, (X, y) in enumerate(dataloader):
      # Send data to target device
      X, y = X.to(device), y.to(device)

      # 1. Forward pass
      test_pred_logits = model(X)
      # print(test_pred_logits)

      # 2. Calculuate and accumulate loss
      loss = loss_fn(test_pred_logits, y)
      test_loss += loss.item()

      # Calculate and accumulate accuracy
      test_pred_labels = test_pred_logits.argmax(dim=1)
      test_acc += ((test_pred_labels == y).sum().item()/len(test_pred_labels))

  # Adjust metrics to get average loss and accuracy per batch
  test_loss = test_loss / len(dataloader)
  test_acc = test_acc / len(dataloader)
  return test_loss, test_acc
```

Fungsi `test_step` digunakan untuk mengevaluasi performa model pada data pengujian menggunakan dataloader pengujian. Pertama, fungsi menempatkan model dalam mode evaluasi (`model.eval()`), kemudian melakukan iterasi pada batch-batch dalam dataloader pengujian. Pada setiap iterasi, dilakukan forward pass untuk mendapatkan prediksi (`test_pred_logits`), menghitung dan mengakumulasi nilai loss dengan menggunakan fungsi loss yang telah diberikan, dan menghitung serta mengakumulasi akurasi pengujian. Penggunaan `torch.inference_mode()` memastikan bahwa operasi-inferensi seperti dropout dinyalakan agar model dapat memberikan prediksi yang konsisten. Akhirnya, nilai loss dan akurasi dihitung ulang sebagai rata-rata per batch. Fungsi ini memberikan informasi tentang sejauh mana model dapat menggeneralisasi pada data pengujian dan membantu dalam penilaian kinerja model secara menyeluruh.

```python
from tqdm.auto import tqdm

def train(model: torch.nn.Module,
          train_dataloader: torch.utils.data.DataLoader,
          test_dataloader: torch.utils.data.DataLoader,
          optimizer: torch.optim.Optimizer,
          loss_fn: torch.nn.Module = nn.CrossEntropyLoss(),
          epochs: int = 5):

  # Create results dictionary
  results = {"train_loss": [],
             "train_acc": [],
             "test_loss": [],
             "test_acc": []}

  # Loop through the training and testing steps for a number of epochs
  for epoch in tqdm(range(epochs)):
    # Train step
    train_loss, train_acc = train_step(model=model,
                                       dataloader=train_dataloader,
                                       loss_fn=loss_fn,
                                       optimizer=optimizer)
    # Test step
    test_loss, test_acc = test_step(model=model,
                                    dataloader=test_dataloader,
                                    loss_fn=loss_fn)
```

```python
    # Print out what's happening
    print(f"Epoch: {epoch+1} | "
          f"train_loss: {train_loss:.4f} | "
          f"train_acc: {train_acc:.4f} | "
          f"test_loss: {test_loss:.4f} | "
          f"test_acc: {test_acc:.4f}"
    )

    # Update the results dictionary
    results["train_loss"].append(train_loss)
    results["train_acc"].append(train_acc)
    results["test_loss"].append(test_loss)
    results["test_acc"].append(test_acc)

  # Return the results dictionary
  return results
```

Fungsi `train` di atas digunakan untuk melatih dan mengevaluasi model selama beberapa epoch menggunakan dataloader pelatihan dan pengujian. Pada setiap epoch, fungsi melakukan pelatihan model dengan memanggil `train_step` dan mengukur performa model pada data pengujian menggunakan `test_step`. Hasil loss dan akurasi pada setiap langkah pelatihan dan pengujian dicetak, dan nilai tersebut disimpan dalam sebuah dictionary hasil (`results`). Fungsi ini memberikan pemantauan langsung terhadap kemajuan pelatihan dan pengujian, dan hasilnya dapat digunakan untuk analisis lebih lanjut atau visualisasi. Penggunaan `tqdm` memberikan bar progres untuk menampilkan estimasi waktu yang tersisa selama pelatihan.

4. Try training the model you made in exercise 3 for 5, 20 and 50 epochs, what happens to the results?

```
# Train for 5 epochs
torch.manual_seed(42)
torch.cuda.manual_seed(42)
model_0 = TinyVGG(input_shape=3,
                  hidden_units=10,
                  output_shape=len(class_names)).to(device)

loss_fn=nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_0.parameters(), lr=0.001)

model_0_results = train(model=model_0,
                        train_dataloader=train_dataloader,
                        test_dataloader=test_dataloader,
                        optimizer=optimizer,
                        epochs=5)
```

Kode di atas melatih model `TinyVGG` selama 5 epoch menggunakan dataloader pelatihan dan pengujian. Model tersebut diinisialisasi dengan seed tertentu untuk memastikan reproduktibilitas hasil pelatihan. Fungsi loss yang digunakan adalah `CrossEntropyLoss`, dan optimizer yang digunakan adalah Adam dengan laju pembelajaran (learning rate) 0.001. Hasil pelatihan, termasuk loss dan akurasi pada setiap langkah, disimpan dalam dictionary `model_0_results`. Proses ini memberikan gambaran tentang performa model selama pelatihan dan pengujian pada dataset gambar yang telah disiapkan.

```
# Train for 20 epochs
torch.manual_seed(42)
torch.cuda.manual_seed(42)
model_1 = TinyVGG(input_shape=3,
                  hidden_units=10,
                  output_shape=len(class_names)).to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_1.parameters(), lr=0.001)

model_1_results = train(model=model_1,
                        train_dataloader=train_dataloader,
                        test_dataloader=test_dataloader,
                        optimizer=optimizer,
                        epochs=20)
```

Kode di atas melatih model `TinyVGG` selama 20 epoch menggunakan dataloader pelatihan dan pengujian. Model ini diinisialisasi dengan seed tertentu untuk memastikan reproduktibilitas hasil pelatihan. Fungsi loss yang digunakan adalah `CrossEntropyLoss`, dan optimizer yang digunakan adalah Adam dengan laju pembelajaran (learning rate) 0.001. Hasil pelatihan, termasuk loss dan akurasi pada setiap langkah, disimpan dalam dictionary `model_1_results`. Dengan melatih model selama 20 epoch, kita dapat melihat bagaimana performa model berkembang lebih lanjut dan mungkin mencapai tingkat akurasi yang lebih stabil atau konvergen pada dataset yang diberikan.

```
# Train for 50 epochs
torch.manual_seed(42)
torch.cuda.manual_seed(42)
model_2 = TinyVGG(input_shape=3,
                  hidden_units=10,
                  output_shape=len(class_names)).to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_2.parameters(), lr=0.001)

model_2_results = train(model=model_2,
                        train_dataloader=train_dataloader,
                        test_dataloader=test_dataloader,
                        optimizer=optimizer,
                        epochs=50)
```

Kode di atas melatih model `TinyVGG` selama 50 epoch menggunakan dataloader pelatihan dan pengujian. Model ini diinisialisasi dengan seed tertentu untuk memastikan reproduktibilitas hasil pelatihan. Fungsi loss yang digunakan adalah `CrossEntropyLoss`, dan optimizer yang digunakan adalah Adam dengan laju pembelajaran (learning rate) 0.001. Hasil pelatihan, termasuk loss dan akurasi pada setiap langkah, disimpan dalam dictionary `model_2_results`. Dengan melatih model untuk lebih banyak epoch, kita dapat mengamati sejauh mana model dapat terus meningkatkan kinerjanya atau mencapai tingkat konvergensi yang optimal pada dataset yang diberikan.

5. Double the number of hidden units in your model and train it for 20 epochs, what happens to the results?

```
# Double the number of hidden units and train for 20 epochs
torch.manual_seed(42)
torch.cuda.manual_seed(42)
model_3 = TinyVGG(input_shape=3,
                  hidden_units=20, # use 20 hidden units instead of 10
                  output_shape=len(class_names)).to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_3.parameters(), lr=0.001)

model_3_results = train(model=model_3,
                        train_dataloader=train_dataloader,
                        test_dataloader=test_dataloader,
                        optimizer=optimizer,
                        epochs=20) # train for 20 epochs
```

Kode di atas melatih model `TinyVGG` dengan menggandakan jumlah unit tersembunyi menjadi 20 dan melatihnya selama 20 epoch menggunakan dataloader pelatihan dan pengujian. Model ini diinisialisasi dengan seed tertentu untuk memastikan reproduktibilitas hasil pelatihan. Fungsi loss yang digunakan adalah `CrossEntropyLoss`, dan optimizer yang digunakan adalah Adam dengan laju pembelajaran (learning rate) 0.001. Hasil pelatihan, termasuk loss dan akurasi pada setiap langkah, disimpan dalam dictionary `model_3_results`. Menggandakan jumlah unit tersembunyi dapat mempengaruhi kapasitas model dan dapat

memberikan pemahaman tentang bagaimana model merespons terhadap perubahan arsitektur pada dataset yang diberikan.

6. Double the data you're using with your model from step 6 and train it for 20 epochs, what happens to the results?

```
# Download 20% data for Pizza/Steak/Sushi from GitHub
import requests
import zipfile
from pathlib import Path

# Setup path to data folder
data_path = Path("data/")
image_path = data_path / "pizza_steak_sushi_20_percent"

# If the image folder doesn't exist, download it and prepare it...
if image_path.is_dir():
    print(f"{image_path} directory exists.")
else:
    print(f"Did not find {image_path} directory, creating one...")
    image_path.mkdir(parents=True, exist_ok=True)

# Download pizza, steak, sushi data
with open(data_path / "pizza_steak_sushi_20_percent.zip", "wb") as f:
    request = requests.get("https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza_steak_sushi_20_percent.zip")
    print("Downloading pizza, steak, sushi 20% data...")
    f.write(request.content)

# Unzip pizza, steak, sushi data
with zipfile.ZipFile(data_path / "pizza_steak_sushi_20_percent.zip", "r") as zip_ref:
    print("Unzipping pizza, steak, sushi 20% data...")
    zip_ref.extractall(image_path)
```

Kode di atas digunakan untuk mengunduh dan menyiapkan dataset gambar Pizza, Steak, dan Sushi sebesar 20% dari ukuran penuhnya. Dataset ini diunduh dari repositori GitHub dan disimpan dalam direktori baru (`pizza_steak_sushi_20_percent`). Jika direktori tersebut sudah ada, kode hanya mencetak pesan bahwa direktori tersebut sudah ada. Setelah mengunduh data, kode mengekstraknya dari file zip dan menyimpannya dalam direktori yang telah disiapkan. Dataset yang lebih kecil ini dapat digunakan untuk percobaan atau pemantauan cepat tanpa perlu mengunduh seluruh dataset aslinya.

```
# See how many images we have
walk_through_dir(image_path)
```

Kode di atas menggunakan fungsi `walk_through_dir` untuk menampilkan jumlah direktori, subdirektori, dan gambar pada suatu path yang diberikan, dalam hal ini, path menuju dataset gambar Pizza, Steak, dan Sushi yang sebesar 20% (`image_path`). Informasi ini diperoleh dari hasil iterasi menggunakan `os.walk`, dan outputnya mencakup jumlah direktori, subdirektori, serta jumlah gambar yang terdapat dalam dataset tersebut. Ini memberikan gambaran tentang struktur dan ukuran dataset yang digunakan untuk pelatihan dan pengujian model.

```
# Turn the data into datasets and dataloaders
train_data_20_percent_path = image_path / "train"
test_data_20_percent_path = image_path / "test"

train_data_20_percent_path, test_data_20_percent_path
```

Kode di atas menentukan path untuk direktori pelatihan (`train_data_20_percent_path`) dan pengujian (`test_data_20_percent_path`) dari dataset gambar Pizza, Steak, dan Sushi yang sebesar 20%. Ini merupakan langkah persiapan untuk membuat dataset dan dataloader pada percobaan atau analisis yang akan dilakukan terhadap dataset yang lebih kecil ini. Dengan menentukan path ini, kita dapat menggunakan informasi tersebut untuk membentuk dataset dan dataloader sesuai dengan struktur data yang telah dipersiapkan sebelumnya.

```
from torchvision.datasets import ImageFolder
from torchvision import transforms
from torch.utils.data import DataLoader

simple_transform = transforms.Compose([
  transforms.Resize((64, 64)),
  transforms.ToTensor()
])

train_data_20_percent = ImageFolder(train_data_20_percent_path,
                                    transform=simple_transform)

test_data_20_percent = ImageFolder(test_data_20_percent_path,
                                   transform=simple_transform)

# Create dataloaders
train_dataloader_20_percent = DataLoader(train_data_20_percent,
                                         batch_size=32,
                                         num_workers=os.cpu_count(),
                                         shuffle=True)

test_dataloader_20_percent = DataLoader(test_data_20_percent,
                                        batch_size=32,
                                        num_workers=os.cpu_count(),
                                        shuffle=False)
```

Kode di atas menggunakan modul `ImageFolder` dari torchvision untuk membuat dataset pelatihan (`train_data_20_percent`) dan dataset pengujian (`test_data_20_percent`) dari gambar Pizza, Steak, dan Sushi yang sebesar 20%. Transformasi sederhana, seperti meresize gambar menjadi ukuran 64x64 piksel dan mengonversinya ke dalam format tensor, diterapkan pada kedua dataset tersebut menggunakan objek `simple_transform`. Selanjutnya, dataloader untuk pelatihan (`train_dataloader_20_percent`) dan pengujian (`test_dataloader_20_percent`) dibuat dengan menentukan ukuran batch, jumlah pekerja (num_workers), dan apakah data harus diacak atau tidak. Dengan langkah ini, dataset dan dataloader yang sesuai dengan transformasi dan pengaturan yang diinginkan telah dibuat, siap untuk digunakan dalam proses pelatihan dan pengujian model.

```
# Train a model with increased amount of data
torch.manual_seed(42)
torch.cuda.manual_seed(42)
model_4 = TinyVGG(input_shape=3,
                  hidden_units=20, # use 20 hidden units instead of 10
                  output_shape=len(class_names)).to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_4.parameters(), lr=0.001)

model_4_results = train(model=model_4,
                        train_dataloader=train_dataloader_20_percent, # use double the training data
                        test_dataloader=test_dataloader_20_percent, # use double the testing data
                        optimizer=optimizer,
                        epochs=20) # train for 20 epochs
```

Kode di atas melatih model `TinyVGG` dengan konfigurasi yang sama seperti sebelumnya, tetapi menggunakan dataset yang lebih besar (20% dari ukuran penuhnya) dari gambar Pizza, Steak, dan Sushi. Model ini diinisialisasi dengan seed tertentu untuk memastikan reproduktibilitas hasil pelatihan. Fungsi loss yang digunakan adalah `CrossEntropyLoss`, dan optimizer yang digunakan adalah Adam dengan laju pembelajaran (learning rate) 0.001. Hasil pelatihan, termasuk loss dan akurasi pada setiap langkah, disimpan dalam dictionary `model_4_results`. Dengan menggunakan dataset yang lebih besar, kita dapat melihat apakah model dapat menghasilkan kinerja yang lebih baik atau lebih stabil pada ukuran data yang lebih signifikan.

7. Make a prediction on your own custom image of pizza/steak/sushi (you could even download one from the internet) with your trained model from exercise 7 and share your prediction.

```
# Get a custom image
custom_image = "pizza_dad.jpeg"
with open("pizza_dad.jpeg", "wb") as f:
  request = requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-deep-learning/main/images/04-pizza-dad.jpeg")
  f.write(request.content)
```

Kode di atas digunakan untuk mengunduh dan menyimpan gambar khusus ("pizza_dad.jpeg") dari repositori GitHub. Menggunakan fungsi `requests.get`, gambar tersebut diambil dari URL tertentu, kemudian disimpan di direktori lokal dengan nama file yang sesuai. Tindakan ini dilakukan untuk mendapatkan gambar kustom yang akan digunakan sebagai contoh untuk diberikan kepada model pada tahap prediksi.

```
# Load the image
import torchvision
img = torchvision.io.read_image(custom_image)
img
```

Kode di atas menggunakan modul `torchvision.io` untuk membaca gambar ("pizza_dad.jpeg") ke dalam format tensor menggunakan fungsi `read_image`. Gambar tersebut dihasilkan sebagai tensor PyTorch, yang dapat digunakan sebagai input untuk model yang telah dilatih untuk melakukan prediksi. Proses ini memungkinkan gambar kustom ini diolah secara

kompatibel dengan model dan dapat diuji untuk melihat bagaimana model merespons terhadap gambar tersebut.

```
# View the image
plt.figure(figsize=(10, 7))
plt.imshow(img.permute(1, 2, 0)) # matplotlib likes images in HWC (height, width, color_channels) format not CHW (color_channels, height, width)
plt.axis(False);
```

Kode di atas menggunakan modul `matplotlib.pyplot` untuk menampilkan gambar kustom ("pizza_dad.jpeg") yang telah diubah formatnya dari tensor PyTorch ke format yang dapat diterima oleh matplotlib. Dengan mempermute dimensi tensor menggunakan `permute(1, 2, 0)`, gambar disusun kembali ke format HWC (tinggi, lebar, saluran warna) yang disukai oleh matplotlib. Gambar tersebut kemudian ditampilkan dalam ukuran dan tata letak yang ditentukan oleh `plt.figure` dan `plt.imshow`, sementara `plt.axis(False)` digunakan untuk menyembunyikan sumbu pada plot.

```
# Make a prediction on the image
model_4.eval()
with torch.inference_mode():
  # Get image pixels into float + between 0 and 1
  img = img / 255.

  # Resize image to 64x64
  resize = transforms.Resize((64, 64))
  img = resize(img)

  # Turn image in single batch and pass to target device
  batch = img.unsqueeze(0).to(device)

  # Predict on image
  y_pred_logit = model_4(batch)

  # Convert pred logit to pred label
  # pred_label = torch.argmax(torch.softmax(y_pred_logit, dim=1), dim=1)
  pred_label = torch.argmax(y_pred_logit, dim=1) # get same results as above without torch.softmax

# Plot the image and prediction
plt.imshow(img.permute(1, 2, 0))
plt.title(f"Pred label: {class_names[pred_label]}")
plt.axis(False);
```

Kode di atas melakukan prediksi pada gambar kustom ("pizza_dad.jpeg") menggunakan model `TinyVGG` yang telah dilatih (`model_4`). Prosesnya melibatkan beberapa langkah pra-pemrosesan, seperti mengubah piksel gambar ke dalam format float dan rentang antara 0 dan 1, meresize gambar menjadi ukuran 64x64, dan mengirimnya sebagai batch tunggal ke perangkat target (device). Setelah itu, model memberikan prediksi dalam bentuk logit, yang kemudian dikonversi menjadi label prediksi menggunakan fungsi `torch.argmax`. Hasil prediksi dan gambar ditampilkan bersamaan dengan label prediksi pada plot menggunakan modul `matplotlib.pyplot`.