04. PyTorch Custom Datasets Exercise

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

```
Thu Jan  4 10:49:37 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   54C    P8              10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

memeriksa apakah GPU tersedia

```
# 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
```

```
2.1.0+cu121
'cuda'
```

hasil output adalah 'cuda' yang artinya device GPU bersedia atau siap digunakan

1. Our models are underperforming (not fitting the data well). What are 3 methods for preventing underfitting? Write them down and explain each with a sentence.

The main idea of preventing underfitting is to: increase our model's predictive power.

1. **Add more layers/units to your model (increase model complexity).** This will potentially give the model more of an opportunity to learn generalizable patterns in the training data. For example, instead of using 2 hidden layers, use 4 hidden layers instead.
2. **Use transfer learning.** Transfer leanring helps to prevent underfitting by leveraging already existing/working patterns from one model/dataset and using them with your own problem.
3. **Train the model for longer.** Perhaps your initial training schedule didn't give your model enough opportunity to learn patterns in the data. Training your model for longer (more epochs) may give improved results.

2. Recreate the data loading functions we built in sections 1, 2, 3 and 4 of notebook 04. You should have train and test `DataLoader`'s ready to use.

```
# 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)
```

```
Did not find data/pizza_steak_sushi directory, creating...
Downloading pizza, steak, sushi data...
Unzipping pizza, steak, suhsi data to data/pizza_steak_sushi
```

langkah selanjutnya yaitu melakukan download dan mengestrak dataset beruapa gambar makanan dari kategori pizza, steak, dan sushi ke dalam folder yang sudah ditentukan 'image_path'.

```
# 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}'.")
```

Fungsi walk_through_dir digunakan untuk menjelajahi struktur direktori dan mengembalikan jumlah direktori dan jumlah file (gambar) dalam setiap direktori.

```
walk_through_dir(image_path)
```

```
There are 2 directories and 0 images in 'data/pizza_steak_sushi'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi/train'.
There are 0 directories and 78 images in 'data/pizza_steak_sushi/train/pizza'.
There are 0 directories and 75 images in 'data/pizza_steak_sushi/train/steak'.
There are 0 directories and 72 images in 'data/pizza_steak_sushi/train/sushi'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi/test'.
There are 0 directories and 25 images in 'data/pizza_steak_sushi/test/pizza'.
There are 0 directories and 19 images in 'data/pizza_steak_sushi/test/steak'.
There are 0 directories and 31 images in 'data/pizza_steak_sushi/test/sushi'.
```

hasil output yang diberikan adalah mengenai informasi jumlah direktori dan jumlah file di setiap level direktori di dalamnya.

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

train_dir, test_dir
```

```
(PosixPath('data/pizza_steak_sushi/train'),
 PosixPath('data/pizza_steak_sushi/test'))
```

Output yang diberikan menunjukkan dua path, yaitu path ke direktori train dan path ke direktori test. (PosixPath('data/pizza_steak_sushi/train'), PosixPath('data/pizza_steak_sushi/test')): adalah hasil output yang menunjukkan path ke direktori train dan test. Path tersebut menggunakan tipe PosixPath, yaitu tipe path yang digunakan oleh sistem operasi berbasis Unix, seperti Linux.

```
# 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")
```

```
[PosixPath('data/pizza_steak_sushi/train/pizza/1660415.jpg'), PosixPath('data/pizza_s
data/pizza_steak_sushi/train/sushi/1280119.jpg
sushi
Random image path: data/pizza_steak_sushi/train/sushi/1280119.jpg
Image class: sushi
Image height: 512
Image width: 512
```

langkah selanjutnya yaitu kode akan memvisualisasikan sebuah gambar dari dataset makanan yang terdiri dari pizza, steak, dan sushi. output yang dihasilkan adalah kelas, tinggi, dan lebar gambar

```
# 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);
```

Image class: sushi | Image shape: (512, 512, 3) -> [height, width, color_channels]

langkah selanjutnya yaitu dengan memvisualisasikan gambar yang telah dibuka sebelumnya dengan modul PIL.

We've got some images in our folders.

Now we need to make them compatible with PyTorch by:

1. Transform the data into tensors.
2. Turn the tensor data into a `torch.utils.data.Dataset` and later a `torch.utils.data.DataLoader`.

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

kode tersebut membuka jalan untuk menyusun pipeline transformasi sesuai dengan kebutuhan. fungsi dataloader disini digunakan untuk mengelola dataset dan membuat iterator untuk mendapatkan batch data

```
# 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
])
```

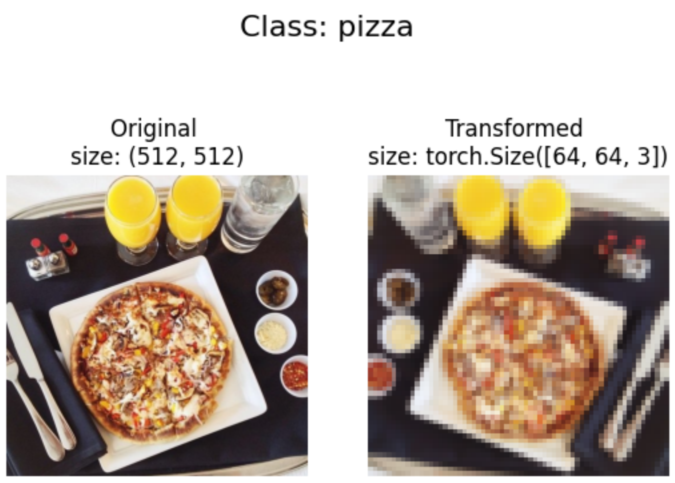
kode tersebut digunakan untuk mendefinisikan transformasi data menggunakan transforms.Compose dari torchvision.transforms.Transformasi kemudian dirancang untuk diaplikasikan pada setiap sampel gambar dalam dataset.

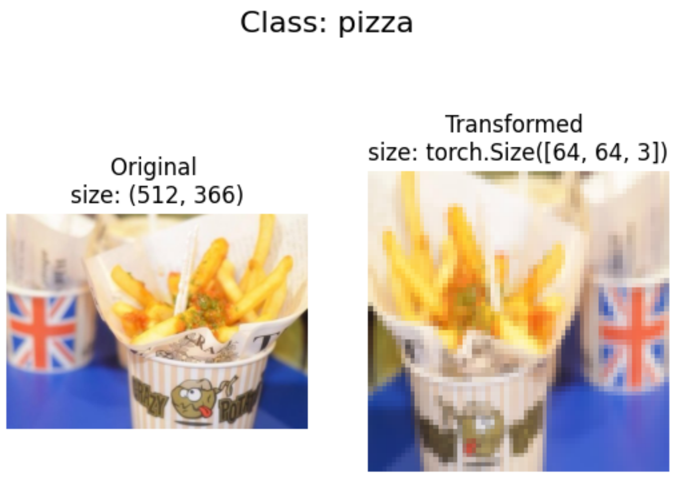
```
random.sample(image_path_list, k=3)
```

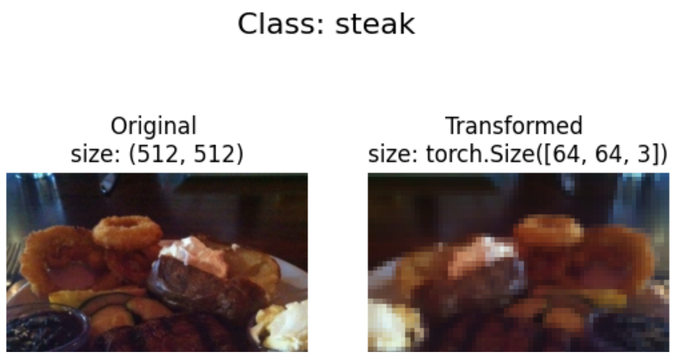
```
[PosixPath('data/pizza_steak_sushi/train/steak/225990.jpg'),
 PosixPath('data/pizza_steak_sushi/train/steak/482022.jpg'),
 PosixPath('data/pizza_steak_sushi/train/pizza/218711.jpg')]
```

kode tersebut digunakan untuk secara acak memilih 3 path gambar dari list. list yang ada di output berisikan 3 path gambar yang dipilih secara acak dari data set.Setiap path gambar merepresentasikan sebuah gambar di dalam dataset makanan. Path gambar ini mencakup informasi mengenai direktori (train/pizza/ atau train/steak/) dan nama file gambar itu sendiri (misalnya, 225990.jpg, 482022.jpg, atau 218711.jpg).

```
# 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)
```







Fungsi plot_transformed_images dibuat untuk memplot beberapa gambar dari dataset setelah melalui transformasi tertentu. Pemanggilan fungsi tersebut menghasilkan output untuk memplot 3 gambar yang dipilih secara acak dari dataset menggunakan transformasi data_transform.

Load image data using `ImageFolder`

```
# 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
```

```
     (Dataset ImageFolder
          Number of datapoints: 225
```

```
        Root location: data/pizza_steak_sushi/train
        StandardTransform
    Transform: Compose(
                   Resize(size=(64, 64), interpolation=bilinear, max_size=None, antialias=warn)
                   RandomHorizontalFlip(p=0.5)
                   ToTensor()
               ),
    Dataset ImageFolder
        Number of datapoints: 75
        Root location: data/pizza_steak_sushi/test
        StandardTransform
    Transform: Compose(
                   Resize(size=(64, 64), interpolation=bilinear, max_size=None, antialias=warn)
                   RandomHorizontalFlip(p=0.5)
                   ToTensor()
               ))
```

Kode ini menggunakan kelas ImageFolder dari torchvision.datasets untuk membuat dataset dari gambar-gambar dalam folder. output yang dihasilkan yaitu train_data: Merupakan objek dataset pelatihan yang telah dibuat dengan menggunakan ImageFolder dan juga test_data: Merupakan objek dataset pengujian yang telah dibuat dengan menggunakan ImageFolder.

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

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

kode mengakses atribut classes dari objek dataset pelatihan (train_data) yang dibuat menggunakan ImageFolder. hasil output tersebut merupakan list yang berisi nama nama kelas atau label yang dikenali oleh train_data

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

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

pada langkah ini kode akan memberi informasi yang memetakan nama kelas ke indeks numerik. pizza dengan indeks 0, steak dengan indeks 1, dan sushi dengan ideks 2

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

```
(225, 75)
```

jumlah sampel data dalam dataset pelatihan (train_data) dan dataset pengujian (test_data). hasil output menunjukkan train_data memiliki jumlah sampel 225 dan test_data memiliki jumlah sampel 75

```
# 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
```

```
(<torch.utils.data.dataloader.DataLoader at 0x78adcec5b490>,
 <torch.utils.data.dataloader.DataLoader at 0x78adf019c640>)
```

Kode ini menggunakan kelas DataLoader dari PyTorch untuk mengonversi dataset pelatihan (train_data) dan dataset pengujian (test_data) menjadi iterator yang dapat menghasilkan batch data.

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

```
(225, 75)
```

hasil output menunjukkan adanya jumlah batch dari setiap train_dataloader yaitu senilai 225 dan test_dataloader senilai 75

```
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}")
```

```
    Image shape: torch.Size([1, 3, 64, 64]) -> [batch_size, color_channels, height, width]
    Label shape: torch.Size([1])
```

pada kode ini menggunakan iter dan next untuk mengambil satu batch data dari train_dataloaderOutput menunjukkan shape dari batch gambar dan label. Misalnya, Image shape: torch.Size([1, 3, 64, 64]) -> [batch_size, color_channels, height, width] menunjukkan bahwa batch terdiri dari satu gambar, setiap gambar memiliki tiga saluran warna (RGB), dan ukuran gambar adalah 64x64 piksel.

3. 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 ini mendefinisikan arsitektur model neural network menggunakan modul PyTorch (torch.nn).

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

```
TinyVGG(
  (conv_block_1): Sequential(
    (0): Conv2d(3, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=2560, out_features=3, bias=True)
  )
)
```

Kode ini membuat sebuah objek model menggunakan kelas TinyVGG yang telah didefinisikan sebelumnya. model_0: Merupakan objek model yang telah dibuat, sesuai dengan arsitektur yang didefinisikan oleh kelas TinyVGG.

```
len(class_names)
```

```
3
```

kode tersebut menggunakan fungsi 'len(class_names)' untuk mendapatkan jumlah kelas atau label dalam dataset. output menunjukkan jumlah kelas atau label dalam dataset yaitu 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([[-0.0671, -0.0261, -0.0614]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
```

Kode ini memasukkan data dummy (dummy data) ke dalam model menggunakan objek model_0. output yang dihasilkan merupakan tensor hasil prediksi dari model terhadap data dummy

4. 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 ini dirancang untuk melakukan satu iterasi atau langkah pelatihan (training step) pada model menggunakan batch data dari dataloader pelatihan.

```
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 melakukan evaluasi (pengujian) model pada satu epoch menggunakan DataLoader.

```
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)

    # 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 ini dirancang untuk melatih dan menguji model pada dataset menggunakan pendekatan supervised learning

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

- Use `torch.optim.Adam()` with a learning rate of 0.001 as the optimizer.

```
# 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)
```

```
100%                                   5/5 [00:11<00:00, 2.61s/it]

Epoch: 1 | train_loss: 1.1182 | train_acc: 0.2889 | test_loss: 1.0962 | test_acc: 0.3
Epoch: 2 | train_loss: 1.1025 | train_acc: 0.3511 | test_loss: 1.1057 | test_acc: 0.2
Epoch: 3 | train_loss: 1.1015 | train_acc: 0.3422 | test_loss: 1.0993 | test_acc: 0.3
Epoch: 4 | train_loss: 1.0990 | train_acc: 0.3378 | test_loss: 1.0997 | test_acc: 0.3
```

Kode ini melibatkan pelatihan model model_0 menggunakan fungsi pelatihan yang disebut train, dengan menggunakan dataloader untuk dataset pelatihan (train_dataloader) dan dataset pengujian (test_dataloader). hasil pelatihan kemudian akan disimpan dalam variabel model_0_results. Proses pelatihan mencakup iterasi melalui dataset pelatihan selama 5 epoch, di mana model diperbarui menggunakan optimizer, dan evaluasi dilakukan pada dataset pengujian.

```
# 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)
```

```
100%                                        20/20 [00:45<00:00, 1.77s/it]
Epoch: 1 | train_loss: 1.1182 | train_acc: 0.2889 | test_loss: 1.0962 | test_acc: 0.3
Epoch: 2 | train_loss: 1.1025 | train_acc: 0.3511 | test_loss: 1.1057 | test_acc: 0.2
Epoch: 3 | train_loss: 1.1015 | train_acc: 0.3422 | test_loss: 1.0993 | test_acc: 0.3
Epoch: 4 | train_loss: 1.0990 | train_acc: 0.3378 | test_loss: 1.0997 | test_acc: 0.3
Epoch: 5 | train_loss: 1.0990 | train_acc: 0.3422 | test_loss: 1.1007 | test_acc: 0.3
Epoch: 6 | train_loss: 1.0995 | train_acc: 0.3422 | test_loss: 1.0978 | test_acc: 0.3
Epoch: 7 | train_loss: 1.0984 | train_acc: 0.3556 | test_loss: 1.1043 | test_acc: 0.4
Epoch: 8 | train_loss: 1.0665 | train_acc: 0.4222 | test_loss: 1.0014 | test_acc: 0.4
Epoch: 9 | train_loss: 1.0137 | train_acc: 0.4667 | test_loss: 1.0107 | test_acc: 0.3
Epoch: 10 | train_loss: 0.9487 | train_acc: 0.5511 | test_loss: 1.0771 | test_acc: 0.
Epoch: 11 | train_loss: 0.9006 | train_acc: 0.5822 | test_loss: 1.0689 | test_acc: 0.
Epoch: 12 | train_loss: 0.7871 | train_acc: 0.6356 | test_loss: 1.1206 | test_acc: 0.
Epoch: 13 | train_loss: 0.7464 | train_acc: 0.6444 | test_loss: 1.1542 | test_acc: 0.
Epoch: 14 | train_loss: 0.6875 | train_acc: 0.7067 | test_loss: 1.4049 | test_acc: 0.
Epoch: 15 | train_loss: 0.6798 | train_acc: 0.7067 | test_loss: 1.1523 | test_acc: 0.
Epoch: 16 | train_loss: 0.5781 | train_acc: 0.7689 | test_loss: 1.3585 | test_acc: 0.
Epoch: 17 | train_loss: 0.5069 | train_acc: 0.7867 | test_loss: 1.6351 | test_acc: 0.
Epoch: 18 | train_loss: 0.4929 | train_acc: 0.7822 | test_loss: 1.5498 | test_acc: 0.
Epoch: 19 | train_loss: 0.4764 | train_acc: 0.8044 | test_loss: 1.7841 | test_acc: 0.
```

Kode ini melatih model model_1 (yang merupakan objek dari kelas TinyVGG) untuk 20 epochs menggunakan fungsi pelatihan (train)

```
# 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)
```

```
100%                                              50/50 [01:32<00:00, 1.75s/it]
Epoch: 1 | train_loss: 1.1179 | train_acc: 0.2800 | test_loss: 1.0960 | test_acc: 0.3
Epoch: 2 | train_loss: 1.1010 | train_acc: 0.3244 | test_loss: 1.0996 | test_acc: 0.2
Epoch: 3 | train_loss: 1.0993 | train_acc: 0.2978 | test_loss: 1.0998 | test_acc: 0.3
Epoch: 4 | train_loss: 1.0990 | train_acc: 0.3333 | test_loss: 1.1001 | test_acc: 0.3
Epoch: 5 | train_loss: 1.0989 | train_acc: 0.3289 | test_loss: 1.1005 | test_acc: 0.3
Epoch: 6 | train_loss: 1.0995 | train_acc: 0.3422 | test_loss: 1.1009 | test_acc: 0.3
Epoch: 7 | train_loss: 1.0986 | train_acc: 0.3467 | test_loss: 1.1014 | test_acc: 0.3
Epoch: 8 | train_loss: 1.0985 | train_acc: 0.3467 | test_loss: 1.1014 | test_acc: 0.3
Epoch: 9 | train_loss: 1.0981 | train_acc: 0.3467 | test_loss: 1.0978 | test_acc: 0.3
Epoch: 10 | train_loss: 1.1167 | train_acc: 0.3422 | test_loss: 1.1017 | test_acc: 0.
Epoch: 11 | train_loss: 1.0988 | train_acc: 0.3467 | test_loss: 1.1022 | test_acc: 0.
Epoch: 12 | train_loss: 1.0977 | train_acc: 0.3467 | test_loss: 1.1064 | test_acc: 0.
Epoch: 13 | train_loss: 1.1073 | train_acc: 0.3822 | test_loss: 1.1059 | test_acc: 0.
Epoch: 14 | train_loss: 1.0947 | train_acc: 0.3556 | test_loss: 1.0988 | test_acc: 0.
Epoch: 15 | train_loss: 1.0671 | train_acc: 0.4800 | test_loss: 1.0668 | test_acc: 0.
Epoch: 16 | train_loss: 0.9707 | train_acc: 0.5467 | test_loss: 1.0992 | test_acc: 0.
Epoch: 17 | train_loss: 0.9127 | train_acc: 0.6089 | test_loss: 1.0318 | test_acc: 0.
Epoch: 18 | train_loss: 0.8900 | train_acc: 0.6089 | test_loss: 1.0212 | test_acc: 0.
Epoch: 19 | train_loss: 0.8733 | train_acc: 0.6044 | test_loss: 1.0186 | test_acc: 0.
Epoch: 20 | train_loss: 0.7874 | train_acc: 0.6444 | test_loss: 1.0485 | test_acc: 0.
Epoch: 21 | train_loss: 0.7854 | train_acc: 0.6622 | test_loss: 1.0081 | test_acc: 0.
Epoch: 22 | train_loss: 0.7314 | train_acc: 0.6800 | test_loss: 1.1870 | test_acc: 0.
Epoch: 23 | train_loss: 0.7109 | train_acc: 0.6889 | test_loss: 1.0907 | test_acc: 0.
Epoch: 24 | train_loss: 0.6342 | train_acc: 0.7067 | test_loss: 1.1220 | test_acc: 0.
Epoch: 25 | train_loss: 0.6170 | train_acc: 0.7511 | test_loss: 1.2681 | test_acc: 0.
Epoch: 26 | train_loss: 0.5452 | train_acc: 0.7733 | test_loss: 1.6367 | test_acc: 0.
Epoch: 27 | train_loss: 0.4993 | train_acc: 0.7511 | test_loss: 1.7456 | test_acc: 0.
Epoch: 28 | train_loss: 0.4610 | train_acc: 0.8133 | test_loss: 1.4969 | test_acc: 0.
Epoch: 29 | train_loss: 0.3965 | train_acc: 0.8400 | test_loss: 1.4497 | test_acc: 0.
Epoch: 30 | train_loss: 0.3307 | train_acc: 0.8711 | test_loss: 1.7504 | test_acc: 0.
Epoch: 31 | train_loss: 0.2968 | train_acc: 0.8800 | test_loss: 1.7944 | test_acc: 0.
Epoch: 32 | train_loss: 0.3163 | train_acc: 0.8756 | test_loss: 2.4163 | test_acc: 0.
Epoch: 33 | train_loss: 0.2602 | train_acc: 0.9022 | test_loss: 2.1035 | test_acc: 0.
Epoch: 34 | train_loss: 0.1881 | train_acc: 0.9378 | test_loss: 2.5821 | test_acc: 0.
Epoch: 35 | train_loss: 0.2177 | train_acc: 0.9200 | test_loss: 2.3344 | test_acc: 0.
Epoch: 36 | train_loss: 0.1770 | train_acc: 0.9422 | test_loss: 2.8147 | test_acc: 0.
Epoch: 37 | train_loss: 0.1308 | train_acc: 0.9556 | test_loss: 2.9852 | test_acc: 0.
Epoch: 38 | train_loss: 0.1105 | train_acc: 0.9644 | test_loss: 3.2620 | test_acc: 0.
Epoch: 39 | train_loss: 0.1181 | train_acc: 0.9733 | test_loss: 3.9537 | test_acc: 0.
Epoch: 40 | train_loss: 0.2107 | train_acc: 0.9333 | test_loss: 3.3042 | test_acc: 0.
Epoch: 41 | train_loss: 0.0880 | train_acc: 0.9644 | test_loss: 3.5190 | test_acc: 0.
Epoch: 42 | train_loss: 0.0705 | train_acc: 0.9778 | test_loss: 3.8276 | test_acc: 0.
Epoch: 43 | train_loss: 0.0446 | train_acc: 0.9911 | test_loss: 4.2371 | test_acc: 0.
Epoch: 44 | train_loss: 0.0625 | train_acc: 0.9822 | test_loss: 4.4770 | test_acc: 0.
Epoch: 45 | train_loss: 0.1027 | train_acc: 0.9644 | test_loss: 4.6629 | test_acc: 0.
Epoch: 46 | train_loss: 0.0427 | train_acc: 0.9822 | test_loss: 4.7732 | test_acc: 0.
Epoch: 47 | train_loss: 0.1549 | train_acc: 0.9556 | test_loss: 3.8312 | test_acc: 0.
Epoch: 48 | train_loss: 0.0990 | train_acc: 0.9689 | test_loss: 4.4685 | test_acc: 0.
Epoch: 49 | train_loss: 0.0257 | train_acc: 1.0000 | test_loss: 4.6516 | test_acc: 0.
```

Kode ini melakukan pelatihan model model_2 dengan menggunakan fungsi train. output yang dihasilkan merupakan objek yang berisi hasil pelatihan iterasi melalui seluruh dataset termasuk loss dan akurasi pada setiap epoch

It looks like our model is starting to overfit towards the end (performing far better on the training data than on the testing data).

In order to fix this, we'd have to introduce ways of preventing overfitting.

6. 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
```

```
100%                                                20/20 [00:37<00:00, 1.77s/it]
Epoch: 1 | train_loss: 1.0967 | train_acc: 0.3511 | test_loss: 1.0814 | test_acc: 0.3
Epoch: 2 | train_loss: 1.0392 | train_acc: 0.4578 | test_loss: 1.0210 | test_acc: 0.4
Epoch: 3 | train_loss: 0.9687 | train_acc: 0.5644 | test_loss: 0.9779 | test_acc: 0.4
Epoch: 4 | train_loss: 0.8835 | train_acc: 0.5956 | test_loss: 1.0319 | test_acc: 0.5
Epoch: 5 | train_loss: 0.8775 | train_acc: 0.5911 | test_loss: 0.9987 | test_acc: 0.4
Epoch: 6 | train_loss: 0.8381 | train_acc: 0.5956 | test_loss: 0.9721 | test_acc: 0.4
Epoch: 7 | train_loss: 0.8013 | train_acc: 0.6756 | test_loss: 1.0960 | test_acc: 0.4
Epoch: 8 | train_loss: 0.7482 | train_acc: 0.6978 | test_loss: 1.0315 | test_acc: 0.4
Epoch: 9 | train_loss: 0.7089 | train_acc: 0.6889 | test_loss: 1.0912 | test_acc: 0.4
Epoch: 10 | train_loss: 0.6997 | train_acc: 0.7289 | test_loss: 1.0511 | test_acc: 0.
Epoch: 11 | train_loss: 0.6250 | train_acc: 0.7378 | test_loss: 1.0843 | test_acc: 0.
Epoch: 12 | train_loss: 0.5687 | train_acc: 0.7733 | test_loss: 1.1463 | test_acc: 0.
Epoch: 13 | train_loss: 0.4729 | train_acc: 0.8000 | test_loss: 1.2856 | test_acc: 0.
Epoch: 14 | train_loss: 0.5137 | train_acc: 0.8000 | test_loss: 1.3481 | test_acc: 0.
Epoch: 15 | train_loss: 0.3688 | train_acc: 0.8578 | test_loss: 1.3398 | test_acc: 0.
Epoch: 16 | train_loss: 0.3276 | train_acc: 0.8844 | test_loss: 1.4977 | test_acc: 0.
Epoch: 17 | train_loss: 0.3034 | train_acc: 0.8756 | test_loss: 1.6402 | test_acc: 0.
Epoch: 18 | train_loss: 0.1891 | train_acc: 0.9378 | test_loss: 2.1131 | test_acc: 0.
Epoch: 19 | train_loss: 0.2673 | train_acc: 0.8933 | test_loss: 1.9429 | test_acc: 0.
```

model_3_results = train(...): Memanggil fungsi train untuk melatih model menggunakan dataloader pelatihan (train_dataloader) dan dataloader pengujian (test_dataloader) selama 20 epochs.

It looks like the model is still overfitting, even when changing the number of hidden units.

To fix this, we'd have to look at ways to prevent overfitting with our model.

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

- **Note:** You can use the custom data creation notebook to scale up your Food101 dataset.
- You can also find the already formatted double data (20% instead of 10% subset) dataset on GitHub, you will need to write download code like in exercise 2 to get it into this notebook.

```
# 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)
```

```
Did not find data/pizza_steak_sushi_20_percent directory, creating one...
Downloading pizza, steak, sushi 20% data...
Unzipping pizza, steak, sushi 20% data...
```

Kode ini mengunduh dan menyiapkan subset (20%) dari data Pizza/Steak/Sushi dari GitHub. with zipfile.ZipFile(data_path / "pizza_steak_sushi_20_percent.zip", "r") as zip_ref:: Membuka file ZIP yang telah diunduh. zip_ref.extractall(image_path): Mengekstrak semua file dalam ZIP ke folder image_path.

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

```
There are 2 directories and 0 images in 'data/pizza_steak_sushi_20_percent'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi_20_percent/train'.
There are 0 directories and 154 images in 'data/pizza_steak_sushi_20_percent/train/pizza'.
There are 0 directories and 146 images in 'data/pizza_steak_sushi_20_percent/train/steak'.
There are 0 directories and 150 images in 'data/pizza_steak_sushi_20_percent/train/sushi'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi_20_percent/test'.
There are 0 directories and 46 images in 'data/pizza_steak_sushi_20_percent/test/pizza'.
There are 0 directories and 58 images in 'data/pizza_steak_sushi_20_percent/test/steak'.
There are 0 directories and 46 images in 'data/pizza_steak_sushi_20_percent/test/sushi'.
```

Excellent, we now have double the training and testing images...

Kode ini menggunakan fungsi walk_through_dir untuk menampilkan statistik tentang jumlah direktori, subdirektori, dan gambar dalam struktur folder dataset. Output dari pemanggilan fungsi mencantumkan informasi tentang jumlah direktori, subdirektori, dan gambar dalam struktur folder dataset.

```
# 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
```

```
(PosixPath('data/pizza_steak_sushi_20_percent/train'),
 PosixPath('data/pizza_steak_sushi_20_percent/test'))
```

Kode ini mendefinisikan path untuk dataset pelatihan dan pengujian dengan mengganti nama direktori dari path 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 ini menggunakan modul PyTorch (torchvision) untuk membuat dataset dan dataloader dari subset (20%) dari dataset asli. Objek train_dataloader_20_percent dan test_dataloader_20_percent digunakan sebagai iterator untuk mendapatkan batch data selama pelatihan dan pengujian.

```
# 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
```

```
100%                                    20/20 [00:38<00:00, 1.81s/it]
Epoch: 1 | train_loss: 1.1041 | train_acc: 0.3312 | test_loss: 1.1005 | test_acc: 0.2
Epoch: 2 | train_loss: 1.0832 | train_acc: 0.3563 | test_loss: 1.0667 | test_acc: 0.4
Epoch: 3 | train_loss: 1.0303 | train_acc: 0.4437 | test_loss: 0.9698 | test_acc: 0.4
Epoch: 4 | train_loss: 0.9194 | train_acc: 0.5437 | test_loss: 0.9163 | test_acc: 0.5
Epoch: 5 | train_loss: 0.8717 | train_acc: 0.5875 | test_loss: 0.9509 | test_acc: 0.4
Epoch: 6 | train_loss: 0.8212 | train_acc: 0.6083 | test_loss: 1.0188 | test_acc: 0.4
Epoch: 7 | train_loss: 0.8850 | train_acc: 0.5917 | test_loss: 0.8875 | test_acc: 0.5
Epoch: 8 | train_loss: 0.7746 | train_acc: 0.6521 | test_loss: 0.8739 | test_acc: 0.6
Epoch: 9 | train_loss: 0.7600 | train_acc: 0.6729 | test_loss: 0.9146 | test_acc: 0.5
Epoch: 10 | train_loss: 0.7674 | train_acc: 0.6562 | test_loss: 0.8624 | test_acc: 0.
Epoch: 11 | train_loss: 0.7217 | train_acc: 0.6854 | test_loss: 0.8813 | test_acc: 0.
Epoch: 12 | train_loss: 0.6688 | train_acc: 0.7063 | test_loss: 0.8956 | test_acc: 0.
Epoch: 13 | train_loss: 0.6030 | train_acc: 0.7500 | test_loss: 0.9879 | test_acc: 0.
Epoch: 14 | train_loss: 0.6041 | train_acc: 0.7000 | test_loss: 0.9478 | test_acc: 0.
Epoch: 15 | train_loss: 0.5634 | train_acc: 0.7562 | test_loss: 0.9915 | test_acc: 0.
Epoch: 16 | train_loss: 0.5656 | train_acc: 0.7667 | test_loss: 0.9643 | test_acc: 0.
Epoch: 17 | train_loss: 0.5353 | train_acc: 0.7688 | test_loss: 1.0623 | test_acc: 0.
```

model=model_4: Model yang akan dilatih. train_dataloader=train_dataloader_20_percent: DataLoader untuk data pelatihan (dengan jumlah data ditingkatkan). test_dataloader=test_dataloader_20_percent: DataLoader untuk data pengujian (dengan jumlah data ditingkatkan). optimizer=optimizer: Optimizer yang digunakan selama pelatihan. epochs=20: Jumlah epoch (iterasi melalui seluruh dataset) yang digunakan untuk pelatihan.

Our model is still overfitting (because the train loss is far lower than the test loss), even with double the data...

It looks like we'll have to further investigate other methods of reducing overfitting to build a better model.

8. 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.

- Does the model you trained in exercise 7 get it right?
- If not, what do you think you could do to improve it?

```
# 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 mengunduh dan menyimpan gambar yang disebut "pizza_dad.jpeg" dari URL tertentu

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

```
tensor([[[154, 173, 181,  ...,  21,  18,  14],
         [146, 165, 181,  ...,  21,  18,  15],
         [124, 146, 172,  ...,  18,  17,  15],
         ...,
         [ 72,  59,  45,  ..., 152, 150, 148],
         [ 64,  55,  41,  ..., 150, 147, 144],
         [ 64,  60,  46,  ..., 149, 146, 143]],

        [[171, 190, 193,  ...,  22,  19,  15],
         [163, 182, 193,  ...,  22,  19,  16],
         [141, 163, 184,  ...,  19,  18,  16],
         ...,
         [ 55,  42,  28,  ..., 107, 104, 103],
         [ 47,  38,  24,  ..., 108, 104, 102],
         [ 47,  43,  29,  ..., 107, 104, 101]],
```