Syifa Wanda Isnaini

1103201248

Machine Learning

04 PyTorch Custom Datasets

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.
   Penambahan Kekompleksan Model:

   Penjelasan: Underfitting dapat disebabkan oleh model yang terlalu sederhana untuk memahami kompleksitas data. Meningkatkan kompleksitas model dengan menambahkan lebih banyak lapisan atau unit dapat membantu model untuk lebih baik menangkap pola-pola yang rumit dalam data.
   Penambahan Data Latih:

   Penjelasan: Underfitting juga dapat terjadi ketika model tidak memiliki cukup data latih untuk memahami variasi yang ada dalam data. Dengan menambah jumlah data latih, model memiliki lebih banyak informasi untuk belajar dan dapat meningkatkan kemampuannya untuk menggeneralisasi ke data baru.
   Penyetelan Hyperparameter yang Lebih Baik:

   Penjelasan: Beberapa hyperparameter, seperti laju pembelajaran (learning rate) atau jumlah epoch, dapat mempengaruhi kemampuan model untuk belajar. Penyetelan hyperparameter yang lebih baik dapat membantu model untuk konvergen dengan lebih baik, menghindari underfitting karena ketidakmampuannya untuk mencapai solusi yang optimal.
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.

```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_sush
  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
```

Kode di atas bertujuan untuk mengunduh dan menyiapkan dataset yang berisi gambar-gambar pizza, steak, dan sushi. Mari kita jelaskan setiap baris kode:

python
Copy code
```
# Setup path to data folder
data_path = Path("data/")
image_path = data_path / "pizza_steak_sushi"
```
Membuat objek Path yang menunjuk ke folder data/ dan pizza_steak_sushi/. data_path digunakan untuk menyimpan semua data yang berkaitan dengan dataset.
python
Copy code
```
# 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)
```
Mengecek apakah folder pizza_steak_sushi/ sudah ada atau belum. Jika sudah ada, mencetak pesan bahwa folder tersebut sudah ada. Jika belum, membuat folder tersebut.
python
Copy code
```
# Download pizza, steak, sushi data (images from GitHub)
with open(data_path / "pizza_steak_sushi.zip", "wb") as f:
```

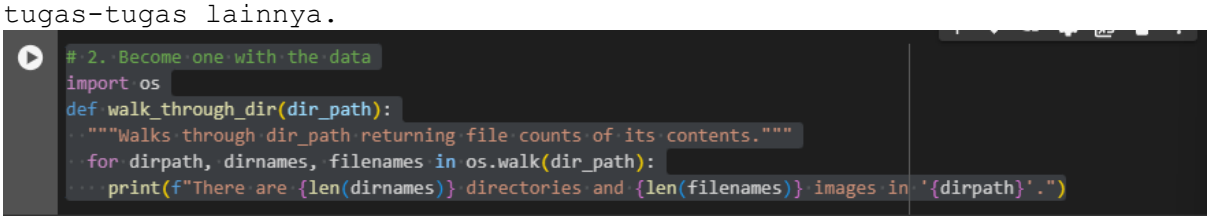
```
  request = requests.get("https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza_steak_sushi.zip")
  print("Downloading pizza, steak, sushi data...")
  f.write(request.content)
```

Mendownload file zip yang berisi gambar-gambar pizza, steak, dan sushi dari GitHub. File zip tersebut disimpan sebagai pizza_steak_sushi.zip dalam folder data/.

python
Copy code

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

Mengekstrak isi dari file zip ke dalam folder pizza_steak_sushi/. Gambar-gambar pizza, steak, dan sushi akan ditempatkan dalam folder tersebut untuk digunakan dalam pengembangan model atau tugas-tugas lainnya.



```
# 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 di atas mendefinisikan sebuah fungsi bernama walk_through_dir yang berguna untuk melihat isi dari suatu direktori dan menghitung jumlah file dan direktori di dalamnya. Berikut adalah penjelasan tiap baris kode:

python
Copy code

```
import os
```

Mengimpor modul os yang menyediakan fungsionalitas untuk berinteraksi dengan sistem operasi, termasuk operasi pada direktori dan file.

python
Copy code

```
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}'.")
```

Mendefinisikan fungsi walk_through_dir dengan parameter dir_path yang merupakan jalur direktori yang akan dijelajahi. Menggunakan os.walk untuk menjelajahi seluruh struktur direktori di dalam dir_path. Fungsi ini mengembalikan tiga nilai: dirpath (jalur ke direktori saat ini), dirnames (nama direktori di dalam dirpath), dan filenames (nama file di dalam dirpath).

Mencetak informasi mengenai jumlah direktori dan jumlah file di setiap direktori yang dijelajahi. Pesan ini mencakup informasi tentang direktori saat ini (dirpath), jumlah direktori di dalamnya (len(dirnames)), dan jumlah file di dalamnya (len(filenames)).

```
[ ]  # 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}'.")
```

```
[ ]  # 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'))
```

Kode yang diberikan adalah sebuah fungsi Python yang disebut walk_through_dir. Fungsi ini bertujuan untuk menjelajahi semua file dan direktori di dalam suatu direktori yang diberikan (dir_path) dan kemudian mencetak jumlah direktori dan jumlah file gambar (diasumsikan sebagai file berkas gambar) di setiap direktori tersebut.

Berikut adalah penjelasan rinci dari kode tersebut:

import os: Mengimpor modul operasi sistem, yang menyediakan fungsi-fungsi untuk berinteraksi dengan sistem file dan sistem operasi.

def walk_through_dir(dir_path): Mendefinisikan fungsi walk_through_dir dengan satu parameter, yaitu dir_path, yang merupakan path atau jalur direktori yang akan dijelajahi.

for dirpath, dirnames, filenames in os.walk(dir_path):: Menggunakan fungsi os.walk untuk melakukan iterasi melalui seluruh struktur direktori yang dimulai dari dir_path. Fungsi ini mengembalikan tiga nilai untuk setiap iterasi:

dirpath: Path direktori saat ini.
dirnames: Daftar nama direktori di dalam dirpath.
filenames: Daftar nama file di dalam dirpath.
print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'."): Mencetak jumlah direktori dan jumlah file gambar di dalam direktori saat ini. Format pesan menggunakan f-string untuk menggabungkan nilai-nilai dinamis ke dalam string.

Dengan menggunakan fungsi ini, Anda dapat memberikan jalur direktori sebagai argumen, dan fungsi akan menjelajahi direktori tersebut, memberikan informasi tentang jumlah direktori dan jumlah file gambar di setiap tingkatan direktori.
Kode ini menetapkan path atau jalur untuk direktori pelatihan (train) dan pengujian (test) dalam konteks suatu proyek yang melibatkan pengolahan gambar. Mari kita perinci:

train_dir = image_path / "train": Membentuk jalur untuk direktori pelatihan. image_path adalah variabel atau objek path yang kemungkinan besar telah diinisialisasi sebelumnya, dan / "train" menambahkan komponen direktori "train" ke path tersebut menggunakan operator pembagian ("/").

test_dir = image_path / "test": Membentuk jalur untuk direktori pengujian. Sama seperti langkah sebelumnya, / "test" menambahkan komponen direktori "test" ke path yang telah ada.

train_dir, test_dir: Kode ini mengeksekusi ekspresi tersebut untuk menetapkan nilai kedua variabel (train_dir dan test_dir) dan kemudian secara bersamaan mencetak keduanya. Ini memungkinkan pengguna untuk melihat jalur atau path yang telah dibentuk untuk direktori pelatihan dan pengujian.

Dengan menetapkan jalur ini, Anda dapat menggunakan variabel train_dir dan test_dir dalam proyek Anda, misalnya, untuk mengakses file gambar pelatihan dan pengujian, atau sebagai argumen untuk fungsi-fungsi lain yang memerlukan path direktor

```
[ ]  # 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 tersebut dirancang untuk memvisualisasikan suatu gambar dari dataset dengan beberapa langkah dan informasi terkait. Mari kita bahas setiap langkahnya:

import random dan from PIL import Image: Mengimpor modul random untuk menghasilkan angka acak dan modul Image dari library Python Imaging Library (PIL) untuk memanipulasi dan menampilkan gambar.

random.seed(42): Memberikan nilai seed untuk generator angka acak. Ini memastikan bahwa hasil angka acak yang dihasilkan oleh fungsi random tetap konsisten setiap kali program dijalankan, sehingga dapat direproduksi.

image_path_list = list(image_path.glob("*/*/*.jpg")): Mengumpulkan semua path file gambar dengan ekstensi ".jpg" dari suatu path utama (image_path). Path ini dibentuk dengan menggunakan metode glob, yang menggunakan pola wildcard ("*") untuk mencocokkan semua kombinasi direktori dan file dengan ekstensi ".jpg". Hasilnya disimpan dalam bentuk list.

random_image_path = random.choice(image_path_list): Memilih secara acak satu path gambar dari list yang telah dibuat. Ini dilakukan dengan menggunakan fungsi random.choice.

image_class = random_image_path.parent.stem: Mendapatkan nama kelas (label) dari path gambar yang dipilih secara acak. parent digunakan untuk mendapatkan direktori parent dari path, dan stem digunakan untuk mendapatkan nama tanpa ekstensi.

img = Image.open(random_image_path): Membuka gambar menggunakan modul Image dari PIL. Hasilnya disimpan dalam variabel img.

print(f"Random image path: {random_image_path}"): Mencetak path lengkap dari gambar yang dipilih secara acak.

print(f"Image class: {image_class}"): Mencetak nama kelas (label) dari gambar yang dipilih secara acak.

print(f"Image height: {img.height}") dan print(f"Image width: {img.width}"): Mencetak tinggi dan lebar gambar.

img: Menampilkan gambar menggunakan modul Image dari PIL.

Seluruh kode ini memberikan informasi dan visualisasi untuk satu gambar secara acak dari dataset, termasuk path gambar, kelas gambar, tinggi, dan lebar gambar.

Kode ini menggunakan Matplotlib untuk memvisualisasikan gambar yang telah dibuka sebelumnya menggunakan modul PIL. Berikut adalah penjelasan langkah-langkahnya:

import numpy as np dan import matplotlib.pyplot as plt: Mengimpor modul NumPy untuk manipulasi array dan modul Matplotlib untuk membuat plot dan visualisasi.

img_as_array = np.asarray(img): Mengonversi gambar yang telah dibuka sebelumnya (objek Image dari PIL) menjadi representasi array NumPy menggunakan fungsi np.asarray(). Ini memungkinkan kita untuk menggunakan gambar sebagai array dalam konteks Matplotlib.

plt.figure(figsize=(10, 7)): Membuat figur (plot) dengan ukuran 10x7 inch. Ini mengatur ukuran gambar yang akan digambar.

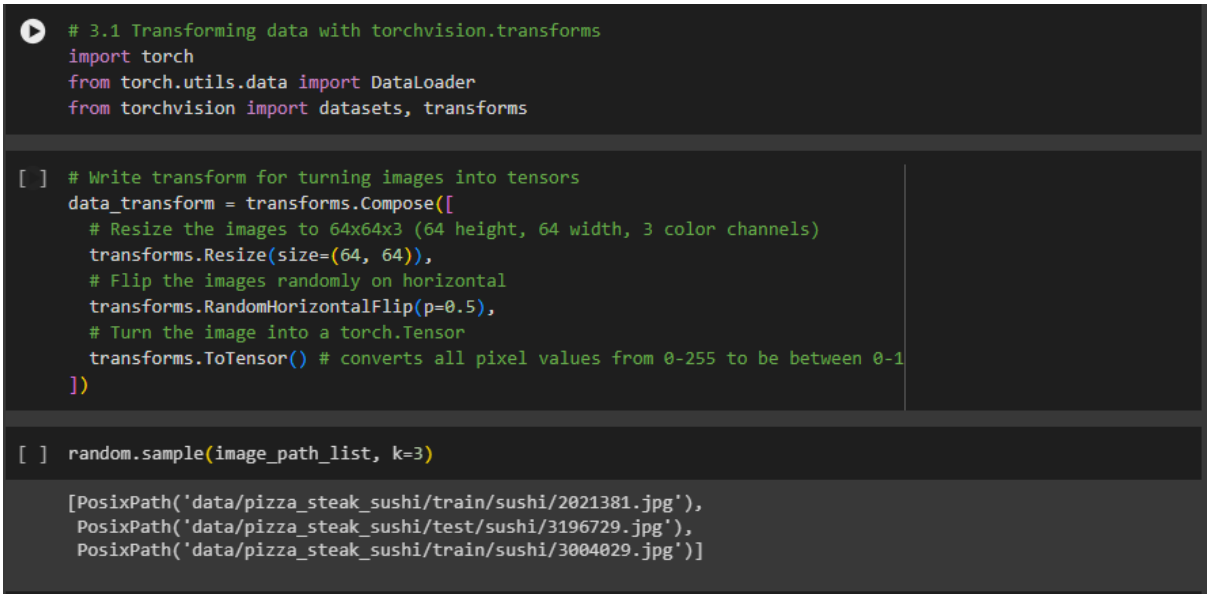
plt.imshow(img_as_array): Menampilkan gambar sebagai plot menggunakan fungsi imshow dari Matplotlib. Gambar diwakili oleh array img_as_array.

plt.title(f"Image class: {image_class} | Image shape: {img_as_array.shape} -> [height, width, color_channels]"): Menambahkan judul pada plot yang mencakup informasi tentang kelas gambar dan dimensi gambar (tinggi, lebar, dan jumlah saluran warna).

plt.axis(False): Menghilangkan sumbu pada plot, karena kita hanya tertarik pada visualisasi gambar tanpa koordinat sumbu.

plt.show(): Menampilkan plot gambar.

Dengan menggunakan Matplotlib, kita dapat dengan mudah memvisualisasikan gambar dalam format array NumPy dan menyertakan informasi tambahan seperti judul dan dimensi gambar.



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

```
# 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
])
```

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

```
[PosixPath('data/pizza_steak_sushi/train/sushi/2021381.jpg'),
 PosixPath('data/pizza_steak_sushi/test/sushi/3196729.jpg'),
 PosixPath('data/pizza_steak_sushi/train/sushi/3004029.jpg')]
```

ode ini menunjukkan penggunaan modul torch dan torchvision untuk melakukan transformasi pada data menggunakan torchvision.transforms. Transformasi ini dapat digunakan untuk mempersiapkan dan memodifikasi data sebelum dimasukkan ke dalam model neural network. Berikut penjelasan singkat dari setiap baris kode:

import torch: Mengimpor modul utama PyTorch.

from torch.utils.data import DataLoader: Mengimpor kelas DataLoader dari modul torch.utils.data. DataLoader digunakan untuk membuat iterator yang memudahkan iterasi melalui dataset dengan batch.

from torchvision import datasets, transforms: Mengimpor modul datasets dan transforms dari torchvision. datasets berisi kelas-kelas yang memungkinkan akses ke dataset umum, sementara transforms berisi fungsi-fungsi yang dapat digunakan untuk melakukan transformasi pada data.

Transformasi data seringkali diperlukan untuk mempersiapkan dataset sebelum diberikan ke dalam model neural network. Contoh transformasi yang umum dilakukan melibatkan normalisasi, resizing, flipping, dll.

Pada titik ini, transformasi yang sebenarnya belum ditunjukkan dalam kode tersebut. Transformasi akan didefinisikan lebih lanjut saat menggunakan torchvision.transforms untuk mempersiapkan data.

Dengan menggunakan torch.utils.data.DataLoader dan torchvision.transforms, kita dapat memuat dataset, menerapkan transformasi pada data, dan mengatur batch size untuk pelatihan model neural network dengan lebih efisien.

Kode tersebut mendefinisikan sebuah transformasi menggunakan torchvision.transforms.Compose yang terdiri dari beberapa langkah transformasi. Transformasi ini dirancang untuk mengubah citra menjadi tensor PyTorch dan melibatkan beberapa operasi seperti resizing, flipping, dan mengubah citra menjadi tensor. Berikut adalah penjelasan dari setiap langkah transformasi:

transforms.Resize(size=(64, 64)): Mengubah ukuran citra menjadi 64x64 piksel. Ini berguna karena beberapa model neural network memerlukan input citra dengan ukuran tertentu.

transforms.RandomHorizontalFlip(p=0.5): Melakukan flipping secara horizontal pada citra secara acak dengan peluang 50% (p=0.5). Ini dapat meningkatkan variasi data selama pelatihan model.

transforms.ToTensor(): Mengonversi citra menjadi tensor PyTorch. Operasi ini akan mengubah nilai piksel dari rentang 0-255 menjadi rentang 0-1. Tensor PyTorch adalah format data yang umum digunakan dalam ekosistem PyTorch untuk menyimpan dan memanipulasi data.

transforms.Compose([...]): Menggunakan Compose untuk menggabungkan semua transformasi menjadi satu. Compose memungkinkan kita untuk menerapkan serangkaian transformasi secara berurutan.

Transformasi ini sering digunakan dalam pra-pemrosesan data sebelum disertakan ke dalam model neural network. Dengan cara ini, data dapat dipersiapkan dengan cara yang konsisten dan sesuai dengan kebutuhan model. Transformasi ini kemudian dapat digunakan bersama dengan DataLoader untuk memuat dataset, menerapkan transformasi pada data, dan mengatur batch size untuk pelatihan model.

```
[ ]  # 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 tersebut mendefinisikan sebuah fungsi bernama plot_transformed_images yang digunakan untuk memplot beberapa

citra secara acak sebelum dan sesudah diterapkan transformasi. Fungsi ini membutuhkan beberapa argumen:

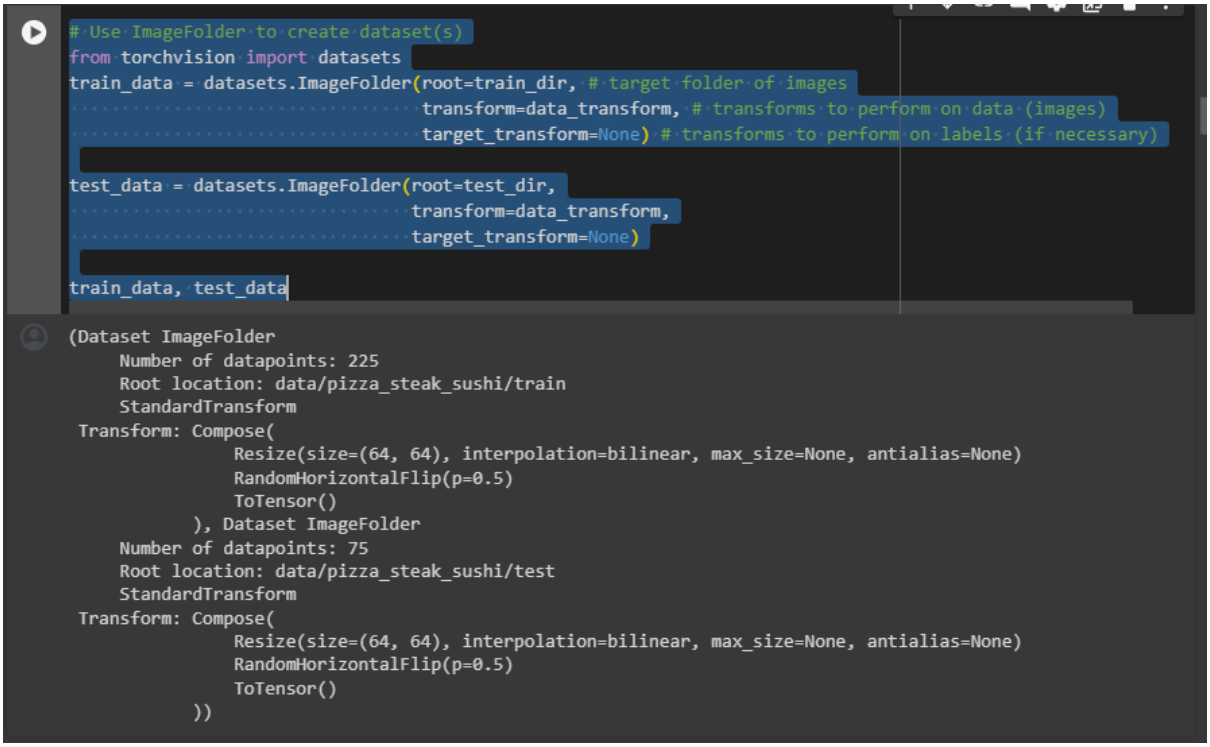
image_paths: List dari path citra yang akan diplot.
transform: Transformasi yang akan diterapkan pada citra.
n: Jumlah citra yang akan diplot secara acak.
seed: Nilai seed untuk menghasilkan hasil acak yang konsisten.
Berikut adalah penjelasan dari setiap bagian fungsi:

random.seed(seed): Mengeset seed untuk generator angka acak. Ini dilakukan untuk memastikan bahwa citra yang dipilih secara acak tetap konsisten setiap kali fungsi dijalankan.

random_image_paths = random.sample(image_paths, k=n): Memilih secara acak n path citra dari list image_paths menggunakan random.sample. Ini akan digunakan untuk memilih citra yang akan diplot.

Iterasi melalui setiap citra yang dipilih secara acak:

with Image.open(image_path) as f: Membuka citra menggunakan modul Image dari PIL dan menyimpannya dalam variabel f.
fig, ax = plt.subplots(nrows=1, ncols=2): Membuat subplot dengan dua kolom (original dan transformed) untuk setiap citra.
Menampilkan citra asli di kolom pertama (ax[0]) dan citra yang telah diterapkan transformasi di kolom kedua (ax[1]).
transformed_image = transform(f).permute(1, 2, 0):
Mengaplikasikan transformasi pada citra dan mengubah dimensi citra untuk kompatibilitas dengan matplotlib.
Menampilkan judul dan mengatur sumbu agar tidak terlihat.
fig.suptitle(f"Class: {image_path.parent.stem}", fontsize=16):
Menampilkan judul subplot dengan nama kelas citra, yang diambil dari nama direktori parent citra.
Memanggil fungsi plot_transformed_images dengan argumen yang sesuai, yaitu image_path_list, data_transform sebagai transformasi, dan n=3 untuk memplot tiga citra secara acak.



```
# 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=None)
               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=None)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
           ))
```

Kode tersebut menggunakan ImageFolder dari modul torchvision.datasets untuk membuat dataset dari direktori gambar pelatihan (train_data) dan direktori gambar pengujian (test_data). Berikut adalah penjelasan rinci:

from torchvision import datasets: Mengimpor modul datasets dari torchvision.

train_data = datasets.ImageFolder(root=train_dir, transform=data_transform, target_transform=None): Membuat objek dataset pelatihan menggunakan ImageFolder. Parameter yang digunakan adalah:

root=train_dir: Menunjukkan path atau jalur ke direktori yang berisi data pelatihan (dalam hal ini, path ke direktori "train").
transform=data_transform: Menggunakan transformasi data_transform yang telah didefinisikan sebelumnya. Transformasi ini akan diterapkan pada setiap citra di dataset.
target_transform=None: Jika ada transformasi yang perlu diterapkan pada label, Anda dapat menyertakan parameter target_transform. Namun, dalam contoh ini, tidak ada transformasi pada label yang diterapkan, sehingga diatur sebagai None.
test_data = datasets.ImageFolder(root=test_dir, transform=data_transform, target_transform=None): Membuat objek dataset pengujian dengan parameter serupa seperti pada train_data. Ini digunakan untuk dataset yang berisi gambar yang akan digunakan untuk menguji model setelah dilatih.

train_data, test_data: Mencetak objek dataset pelatihan dan pengujian. Dataset ini siap digunakan untuk melatih dan menguji model machine learning, dan dapat dimasukkan ke dalam DataLoader untuk pengolahan lebih lanjut.

Dengan menggunakan ImageFolder dan menyertakan transformasi pada data, kita dapat dengan mudah membuat dataset PyTorch dari struktur direktori yang sesuai dengan format dataset klasifikasi gambar standar.

Kode tersebut mengakses atribut classes dari objek dataset pelatihan (train_data) yang dibuat sebelumnya menggunakan ImageFolder. Atribut ini berisi daftar unik dari nama kelas (label) dalam dataset. Berikut adalah penjelasan singkat dari kode tersebut:

class_names = train_data.classes: Mengakses atribut classes dari objek dataset pelatihan (train_data). Atribut ini berisi daftar unik dari nama kelas (label) yang ditemukan dalam dataset.

class_names: Mencetak daftar nama kelas yang ditemukan dalam dataset. Daftar ini dapat digunakan untuk mengidentifikasi kelas apa saja yang ada dalam dataset dan akan membantu dalam interpretasi hasil model.

Misalnya, jika dataset adalah dataset klasifikasi gambar dengan tiga kelas: "cat", "dog", dan "bird", maka class_names akan berisi ['cat', 'dog', 'bird']. Informasi ini dapat digunakan, misalnya, saat memvisualisasikan hasil prediksi model atau mengevaluasi performa model.
Kode tersebut mengakses atribut class_to_idx dari objek dataset pelatihan (train_data) yang dibuat dengan menggunakan ImageFolder. Atribut ini berisi pemetaan antara nama kelas (label) dan indeks numerik yang sesuai. Berikut adalah penjelasan singkat dari kode tersebut:

class_dict = train_data.class_to_idx: Mengakses atribut class_to_idx dari objek dataset pelatihan (train_data). Atribut ini berisi sebuah kamus (dictionary) yang memetakan nama kelas (label) ke indeks numerik.

class_dict: Mencetak kamus yang berisi pemetaan antara nama kelas dan indeks numerik. Setiap pasangan kunci-nilai dalam kamus ini menunjukkan nama kelas dan indeks yang sesuai dalam dataset.

Misalnya, jika dataset adalah dataset klasifikasi gambar dengan tiga kelas: "cat", "dog", dan "bird", dan indeks yang sesuai adalah 0, 1, dan 2, maka class_dict akan menjadi {'cat': 0, 'dog': 1, 'bird': 2}. Informasi ini dapat digunakan untuk mengonversi antara nama kelas dan indeks numerik saat kita bekerja dengan model dan hasil prediksi.
Kode tersebut menggunakan fungsi len untuk mengukur panjang atau jumlah sampel dalam dataset pelatihan (train_data) dan dataset pengujian (test_data). Setiap sampel dalam dataset ini mencakup satu citra bersama dengan labelnya. Berikut adalah penjelasan singkat dari kode tersebut:

len(train_data): Mengukur jumlah sampel dalam dataset pelatihan. Ini mencakup seluruh citra dan label yang terdapat dalam direktori pelatihan.

len(test_data): Mengukur jumlah sampel dalam dataset pengujian. Ini mencakup seluruh citra dan label yang terdapat dalam direktori pengujian.

Kedua nilai ini akan memberikan informasi tentang seberapa besar dataset pelatihan dan pengujian. Panjang dataset pelatihan dan pengujian umumnya digunakan untuk mengatur berapa kali proses pelatihan dan evaluasi model akan dilakukan selama satu epoch.

Misalnya, jika len(train_data) menghasilkan 1000 dan len(test_data) menghasilkan 200, itu berarti dataset pelatihan memiliki 1000 sampel, dan dataset pengujian memiliki 200 sampel.

```
[ ] # 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 0x7fce57ec08d0>,
     <torch.utils.data.dataloader.DataLoader at 0x7fce57ec0dd0>)
```

Kode tersebut menggunakan modul torch.utils.data.DataLoader untuk membuat DataLoader yang akan digunakan selama pelatihan dan pengujian model. DataLoader ini memungkinkan kita untuk mengakses data secara efisien dalam bentuk batch. Berikut adalah penjelasan singkat dari kode tersebut:

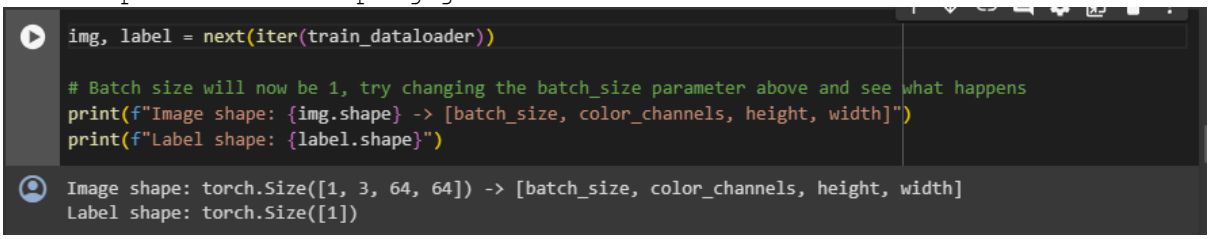
from torch.utils.data import DataLoader: Mengimpor kelas DataLoader dari modul torch.utils.data. DataLoader digunakan untuk membuat iterator yang memudahkan iterasi melalui dataset dengan batch.

BATCH_SIZE = 1: Menetapkan ukuran batch. Dalam hal ini, batch size diatur sebagai 1, yang berarti setiap iterasi DataLoader akan memberikan satu sampel pada model.

train_dataloader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE, num_workers=os.cpu_count(), shuffle=True): Membuat DataLoader untuk dataset pelatihan dengan parameter sebagai berikut:

dataset=train_data: Menggunakan objek dataset pelatihan yang telah dibuat sebelumnya.
batch_size=BATCH_SIZE: Menetapkan ukuran batch sesuai dengan nilai yang telah diatur sebelumnya.
num_workers=os.cpu_count(): Menentukan jumlah pekerja (workers) yang akan digunakan untuk memuat data secara paralel. os.cpu_count() memberikan jumlah core CPU yang tersedia di sistem.
shuffle=True: Mengacak urutan data di setiap epoch. Ini membantu model untuk melihat variasi data yang berbeda selama pelatihan.
test_dataloader = DataLoader(dataset=test_data, batch_size=BATCH_SIZE, num_workers=os.cpu_count(), shuffle=False): Membuat DataLoader untuk dataset pengujian dengan parameter yang serupa, kecuali shuffle diatur sebagai False karena tidak perlu mengacak urutan data pengujian.

train_dataloader, test_dataloader: Mencetak objek DataLoader untuk dataset pelatihan dan pengujian. DataLoader ini dapat digunakan untuk mengiterasi melalui data dalam bentuk batch selama pelatihan dan pengujian model.



```
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])
```

Kode tersebut menggunakan iter dan next untuk mengambil satu batch data dari DataLoader pelatihan (train_dataloader). Ini digunakan untuk menunjukkan dimensi dari batch yang dihasilkan. Berikut adalah penjelasan dari kode tersebut:

img, label = next(iter(train_dataloader)): Menggunakan fungsi iter dan next untuk mengambil satu batch dari DataLoader pelatihan. Variabel img akan berisi tensor yang mewakili citra-citra dalam batch tersebut, dan label akan berisi tensor yang mewakili label-label dari citra-citra tersebut.

print(f"Image shape: {img.shape} -> [batch_size, color_channels, height, width]"): Mencetak bentuk (shape) dari tensor img. Ini memberikan informasi tentang dimensi dari setiap citra dalam batch. Contoh outputnya mungkin adalah "Image shape: torch.Size([1, 3, 64, 64]) -> [batch_size, color_channels, height, width]". Dimensi 1 menunjukkan bahwa batch size adalah 1, 3 adalah jumlah saluran warna (misalnya, RGB), dan 64x64 adalah ukuran citra setelah transformasi.

print(f"Label shape: {label.shape}"): Mencetak bentuk (shape) dari tensor label. Ini memberikan informasi tentang dimensi dari label-label dalam batch. Contoh outputnya mungkin adalah "Label shape: torch.Size([1])", yang menunjukkan bahwa batch size label juga adalah 1.

Penting untuk diingat bahwa nilai batch_size dalam DataLoader sebelumnya diatur sebagai 1, sehingga setiap batch hanya berisi satu sampel citra. Jika batch_size diubah, dimensi dari img dan label akan disesuaikan sesuai dengan ukuran batch yang baru.

3. Recreate model 0 we built in section 7 of notebook

```
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 tersebut mendefinisikan suatu model neural network menggunakan modul PyTorch (torch) dengan arsitektur yang mirip dengan arsitektur VGG (Visual Geometry Group). Model ini disebut TinyVGG dan memiliki tiga bagian utama: dua blok konvolusi dan satu lapisan klasifikasi. Berikut adalah penjelasan rinci dari kode tersebut:

class TinyVGG(nn.Module):: Mendefinisikan kelas TinyVGG yang merupakan turunan dari kelas nn.Module, yang merupakan dasar untuk semua model PyTorch.

def __init__(self, input_shape, hidden_units, output_shape):: Metode konstruktor untuk kelas TinyVGG. Parameter:

input_shape: Jumlah saluran warna (misalnya, 3 untuk RGB), yang menentukan jumlah saluran warna dari citra input.
hidden_units: Jumlah saluran dalam blok konvolusi dan juga jumlah fitur dalam lapisan klasifikasi.
output_shape: Jumlah kelas dalam lapisan klasifikasi, yang menentukan output dari model.
super().__init__(): Memanggil konstruktor kelas induk (nn.Module) untuk melakukan inisialisasi.

self.conv_block_1: Blok konvolusi pertama. Terdiri dari:

Dua lapisan konvolusi dengan aktivasi ReLU di antara keduanya.
Operasi max pooling setelah lapisan kedua konvolusi.
self.conv_block_2: Blok konvolusi kedua. Sama seperti conv_block_1, terdiri dari dua lapisan konvolusi dengan aktivasi ReLU dan diakhiri dengan operasi max pooling.

self.classifier: Lapisan klasifikasi, terdiri dari:

Operasi flatten untuk meratakan output dari blok konvolusi menjadi vektor satu dimensi.
Lapisan linear (fully connected) yang menghubungkan input ke output.
def forward(self, x):: Metode forward yang mendefinisikan bagaimana input akan melewati model selama proses forward (selama perhitungan prediksi). Parameter:

x: Input citra yang akan diproses oleh model.
x = self.conv_block_1(x): Memproses input melalui blok konvolusi pertama.

x = self.conv_block_2(x): Memproses output blok konvolusi pertama melalui blok konvolusi kedua.

x = self.classifier(x): Memproses output blok konvolusi kedua melalui lapisan klasifikasi.

return x: Mengembalikan output dari model.

Model ini memiliki arsitektur yang sederhana dengan dua blok konvolusi dan satu lapisan klasifikasi. Ini dapat digunakan sebagai dasar untuk tugas klasifikasi gambar dengan dataset yang lebih kecil atau untuk keperluan pembelajaran dan eksperimen. Perlu diingat bahwa model ini masih bersifat sederhana dan untuk tugas nyata, model VGG yang lebih besar mungkin lebih sesuai.

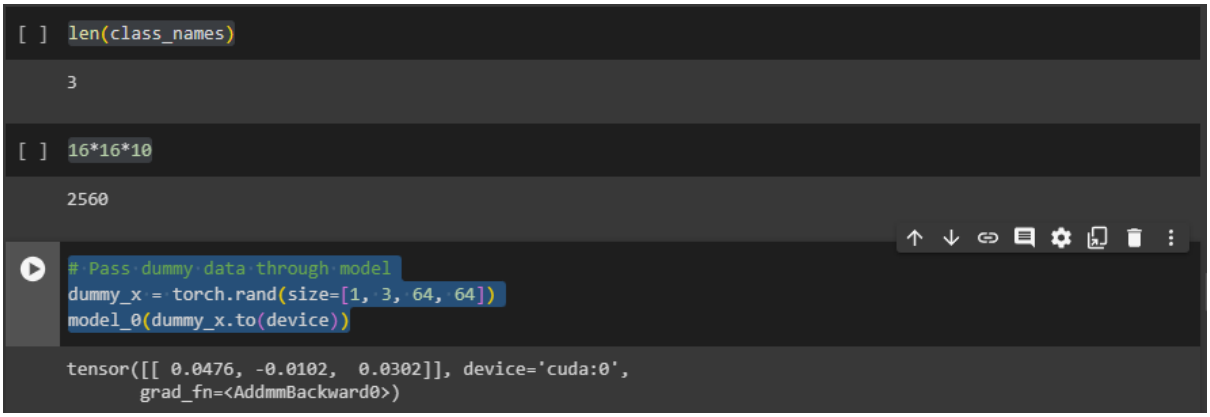
```
[ ]  model_0 = TinyVGG(input_shape = 3,
                       hidden_units=10,
                       output_shape=len(class_names)).to(device)
     model_0
```

```
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 tersebut membuat instance dari model TinyVGG dengan nama model_0 menggunakan parameter-parameter yang diberikan dan mengirimkan model tersebut ke perangkat yang ditentukan (dalam hal ini, device). Berikut adalah penjelasan dari kode tersebut:

model_0 = TinyVGG(input_shape=3, hidden_units=10, output_shape=len(class_names)).to(device): Membuat instance dari model TinyVGG dengan parameter-parameter berikut:

input_shape=3: Menunjukkan bahwa citra input memiliki 3 saluran warna, yang umumnya sesuai dengan citra RGB.
hidden_units=10: Menunjukkan jumlah saluran (fitur) dalam blok konvolusi dan jumlah fitur dalam lapisan klasifikasi.
output_shape=len(class_names): Menunjukkan jumlah kelas dalam lapisan klasifikasi, yang diambil dari jumlah nama kelas yang telah diperoleh sebelumnya dari dataset (class_names).
.to(device): Memindahkan model ke perangkat yang ditentukan (device). Ini dapat menjadi CPU atau GPU, tergantung pada konfigurasi perangkat PyTorch.
model_0: Mencetak model TinyVGG, memberikan informasi tentang arsitektur model, lapisan-lapisan yang terlibat, dan parameter-parameter lainnya.



Dengan langkah ini, model_0 siap untuk digunakan untuk pelatihan dan evaluasi pada dataset tertentu. Pada tahap selanjutnya, model ini dapat dioptimalkan dan disesuaikan dengan menggunakan dataset pelatihan.

Kode ini melakukan beberapa operasi untuk menguji dan memahami model model_0.

len(class_names): Mencetak jumlah kelas dalam dataset, yang telah diambil dari class_names. Ini memberikan informasi tentang berapa banyak kelas yang ada dalam dataset.

16*16*10: Menghitung nilai yang menunjukkan jumlah parameter dalam lapisan klasifikasi model model_0. Ini dihitung dengan mengambil jumlah fitur dalam lapisan klasifikasi (10), dikalikan dengan ukuran input yang dihasilkan dari blok konvolusi kedua (16x16). Ini memberikan jumlah parameter yang terlibat dalam lapisan klasifikasi.

dummy_x = torch.rand(size=[1, 3, 64, 64]): Membuat data citra dummy dengan menggunakan fungsi torch.rand. Citra ini memiliki dimensi [1, 3, 64, 64], yang sesuai dengan format batch size 1, 3 saluran warna, dan ukuran citra 64x64.

model_0(dummy_x.to(device)): Memasukkan citra dummy ke model model_0 yang telah diinisialisasi sebelumnya. dummy_x.to(device) digunakan untuk memastikan bahwa data masukan juga berada di perangkat yang sama dengan model (device). Operasi ini menghasilkan output dari model, yang dapat digunakan untuk melihat bentuk hasil dari setiap lapisan dan untuk keperluan diagnostik selama pembangunan model.

Poin-poin ini memberikan wawasan tentang model dan data yang digunakan dalam konteks pengembangan dan pengujian model sebelum melakukan pelatihan pada dataset yang lebih besar.

4. Create training and testis function 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)
    trin_acc += (y_pred_class == y).sum().item()/len(y_pred)

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

Kode tersebut mendefinisikan sebuah fungsi train_step yang merupakan langkah pelatihan (training step) satu iterasi. Fungsi ini digunakan untuk melatih model pada satu epoch (iterasi penuh melalui dataset pelatihan). Berikut adalah penjelasan dari setiap bagian kode tersebut:

def train_step(model: torch.nn.Module, dataloader: torch.utils.data.DataLoader, loss_fn: torch.nn.Module, optimizer: torch.optim.Optimizer):: Mendefinisikan fungsi train_step dengan parameter sebagai berikut:

model: Model neural network yang akan dilatih.
dataloader: DataLoader yang menyediakan akses ke dataset pelatihan.
loss_fn: Fungsi kerugian (loss function) yang akan digunakan untuk menghitung kerugian selama pelatihan.
optimizer: Optimizer yang digunakan untuk mengoptimalkan model.
model.train(): Mengatur model ke mode pelatihan. Ini memastikan bahwa model memahami bahwa sedang dalam fase pelatihan, yang penting untuk lapisan-lapisan seperti dropout atau batch normalization yang memiliki perilaku berbeda antara fase pelatihan dan evaluasi.

train_loss, train_acc = 0, 0: Inisialisasi nilai kerugian dan akurasi pelatihan.

for batch, (X, y) in enumerate(dataloader):: Loop melalui setiap batch dalam DataLoader pelatihan.

X, y = X.to(device), y.to(device): Mengirimkan data batch ke perangkat yang ditentukan (device), yang sesuai dengan perangkat tempat model berada.

y_pred = model(X): Melakukan langkah forward pass dengan memasukkan data batch ke model dan menghasilkan prediksi.

loss = loss_fn(y_pred, y): Menghitung nilai kerugian (loss) antara prediksi model dan label sebenarnya.

train_loss += loss.item(): Menyimpan dan mengakumulasi nilai kerugian untuk setiap batch.

optimizer.zero_grad(): Mengatur gradien model ke nol sebelum melakukan backpropagation pada setiap batch. Ini diperlukan karena PyTorch secara default akan mengakumulasi gradien dari iterasi sebelumnya.

loss.backward(): Menghitung gradien dari nilai kerugian terhadap parameter model.

optimizer.step(): Mengoptimalkan model dengan melakukan langkah optimisasi berdasarkan gradien yang dihitung.

y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1): Menghitung kelas prediksi dengan memilih indeks kelas dengan nilai probabilitas tertinggi dari keluaran model.

train_acc += (y_pred_class == y).sum().item()/len(y_pred): Menghitung dan mengakumulasi nilai akurasi untuk setiap batch.

train_loss = train_loss / len(dataloader): Menghitung nilai kerugian rata-rata per batch.

train_acc = train_acc / len(dataloader): Menghitung nilai akurasi rata-rata per batch.

return train_loss, train_acc: Mengembalikan nilai kerugian dan akurasi rata-rata per batch.

Fungsi ini mencakup langkah-langkah umum yang diperlukan selama pelatihan model: forward pass, perhitungan loss, backward pass (backpropagation), optimisasi, dan perhitungan metrik akurasi.

```
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
```

Kode tersebut mendefinisikan fungsi test_step yang digunakan untuk melakukan evaluasi (testing) pada model pada satu iterasi melalui dataset pengujian. Berikut adalah penjelasan dari setiap bagian kode tersebut:

def test_step(model: torch.nn.Module, dataloader: torch.utils.data.DataLoader, loss_fn: torch.nn.Module):: Mendefinisikan fungsi test_step dengan parameter sebagai berikut:

model: Model neural network yang akan dievaluasi.
dataloader: DataLoader yang menyediakan akses ke dataset pengujian.
loss_fn: Fungsi kerugian (loss function) yang akan digunakan untuk menghitung kerugian selama evaluasi.
model.eval(): Mengatur model ke mode evaluasi. Ini memastikan bahwa model memahami bahwa sedang dalam fase evaluasi, yang penting untuk lapisan-lapisan seperti dropout atau batch normalization yang memiliki perilaku berbeda antara fase pelatihan dan evaluasi.

test_loss, test_acc = 0, 0: Inisialisasi nilai kerugian dan akurasi pengujian.

with torch.inference_mode():: Mengaktifkan konteks inference mode menggunakan torch.inference_mode(). Ini mengatur model ke mode inference, yang dapat meningkatkan efisiensi evaluasi pada beberapa tipe lapisan.

for batch, (X, y) in enumerate(dataloader):: Loop melalui setiap batch dalam DataLoader pengujian.

X, y = X.to(device), y.to(device): Mengirimkan data batch ke perangkat yang ditentukan (device), yang sesuai dengan perangkat tempat model berada.

test_pred_logits = model(X): Melakukan langkah forward pass dengan memasukkan data batch ke model dan menghasilkan logits (nilai sebelum fungsi softmax) sebagai output.

loss = loss_fn(test_pred_logits, y): Menghitung nilai kerugian (loss) antara prediksi model dan label sebenarnya.

test_loss += loss.item(): Menyimpan dan mengakumulasi nilai kerugian untuk setiap batch.

test_pred_labels = test_pred_logits.argmax(dim=1): Menghitung label prediksi dengan memilih indeks kelas dengan nilai probabilitas tertinggi dari keluaran model.

test_acc += ((test_pred_labels == y).sum().item()/len(test_pred_labels)): Menghitung dan mengakumulasi nilai akurasi untuk setiap batch.

test_loss = test_loss / len(dataloader): Menghitung nilai kerugian rata-rata per batch.

test_acc = test_acc / len(dataloader): Menghitung nilai akurasi rata-rata per batch.

return test_loss, test_acc: Mengembalikan nilai kerugian dan akurasi rata-rata per batch selama evaluasi.

Fungsi ini mencakup langkah-langkah umum yang diperlukan selama evaluasi model: forward pass, perhitungan loss, dan perhitungan metrik akurasi. Konteks inference mode diaktifkan untuk meningkatkan efisiensi evaluasi.

```
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
```

Kode tersebut mendefinisikan fungsi train yang digunakan untuk melatih model pada sejumlah epoch tertentu. Fungsi ini mengintegrasikan langkah-langkah pelatihan dan evaluasi yang telah didefinisikan sebelumnya (train_step dan test_step). Berikut adalah penjelasan dari setiap bagian kode tersebut:

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):: Mendefinisikan fungsi train dengan parameter sebagai berikut:

model: Model neural network yang akan dilatih.
train_dataloader: DataLoader yang menyediakan akses ke dataset pelatihan.
test_dataloader: DataLoader yang menyediakan akses ke dataset pengujian.
optimizer: Optimizer yang digunakan untuk mengoptimalkan model.
loss_fn: Fungsi kerugian (loss function) yang akan digunakan selama pelatihan dan evaluasi.
epochs: Jumlah epoch (iterasi penuh melalui dataset pelatihan) yang akan dilakukan.
results = {"train_loss": [], "train_acc": [], "test_loss": [], "test_acc": []}: Membuat dictionary results yang akan menyimpan nilai kerugian dan akurasi pada setiap epoch untuk pelatihan dan evaluasi.

for epoch in tqdm(range(epochs)):: Loop melalui sejumlah epoch yang telah ditentukan, menggunakan tqdm untuk menampilkan bar progres.

train_loss, train_acc = train_step(model=model, dataloader=train_dataloader, loss_fn=loss_fn, optimizer=optimizer): Melakukan langkah pelatihan untuk satu epoch menggunakan fungsi train_step. Mengembalikan nilai kerugian dan akurasi selama pelatihan.

test_loss, test_acc = test_step(model=model, dataloader=test_dataloader, loss_fn=loss_fn): Melakukan langkah evaluasi untuk satu epoch menggunakan fungsi test_step. Mengembalikan nilai kerugian dan akurasi selama evaluasi.

print(f"Epoch: {epoch+1} | train_loss: {train_loss:.4f} | train_acc: {train_acc:.4f} | test_loss: {test_loss:.4f} | test_acc: {test_acc:.4f}"): Mencetak informasi tentang kerugian dan akurasi pada setiap epoch selama pelatihan dan evaluasi.

results["train_loss"].append(train_loss), results["train_acc"].append(train_acc), dll.: Menambahkan nilai kerugian dan akurasi pada setiap epoch ke dalam dictionary results.

return results: Mengembalikan dictionary results yang berisi nilai kerugian dan akurasi selama pelatihan dan evaluasi.

Fungsi ini menyederhanakan proses pelatihan dan evaluasi model dengan memanggil fungsi-fungsi yang telah didefinisikan sebelumnya pada setiap epoch dan menyimpan hasilnya dalam dictionary results. Dictionary ini dapat digunakan untuk memvisualisasikan dan menganalisis performa model sepanjang waktu.

5. Try training the model you made in exercise 3 for 5, 20, anda 50 epoch, what happens to the result?

```
[ ] # 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 [00:10<00:00, 2.02s/it]
Epoch: 1 | train_loss: 1.1180 | train_acc: 0.2756 | test_loss: 1.0962 | test_acc: 0.3333
Epoch: 2 | train_loss: 1.1011 | train_acc: 0.3422 | test_loss: 1.0998 | test_acc: 0.2533
Epoch: 3 | train_loss: 1.0993 | train_acc: 0.3022 | test_loss: 1.0999 | test_acc: 0.3333
Epoch: 4 | train_loss: 1.0990 | train_acc: 0.3289 | test_loss: 1.1003 | test_acc: 0.3600
Epoch: 5 | train_loss: 1.0991 | train_acc: 0.3333 | test_loss: 1.1003 | test_acc: 0.3333
```

Kode ini melatih model model_0 menggunakan fungsi train yang telah didefinisikan sebelumnya. Berikut adalah penjelasan dari setiap bagian kode tersebut:

torch.manual_seed(42), torch.cuda.manual_seed(42): Mengatur seed (benih) untuk mengontrol keacakan pada library PyTorch. Ini dilakukan untuk memastikan hasil yang dapat direproduksi.

model_0 = TinyVGG(input_shape=3, hidden_units=10, output_shape=len(class_names)).to(device): Membuat instance dari model TinyVGG dengan parameter-parameter tertentu dan mengirimkan model tersebut ke perangkat yang ditentukan (device).

loss_fn = nn.CrossEntropyLoss(): Membuat instance dari fungsi kerugian CrossEntropyLoss, yang umum digunakan untuk tugas klasifikasi.

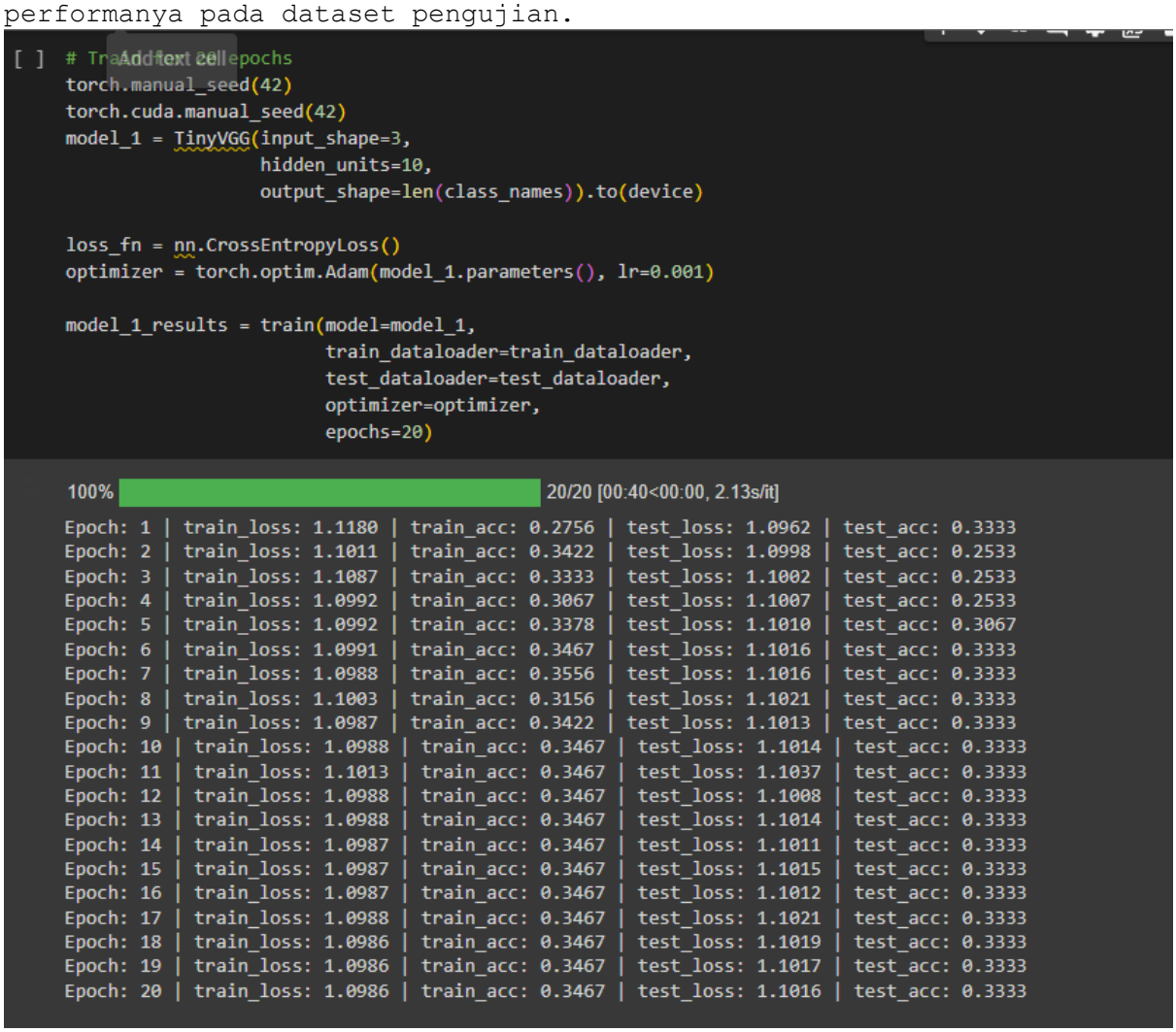
optimizer = torch.optim.Adam(model_0.parameters(), lr=0.001): Membuat instance dari optimizer Adam untuk mengoptimalkan parameter-parameter model model_0. Learning rate (tingkat pembelajaran) diatur sebesar 0.001.

model_0_results = train(...): Memanggil fungsi train untuk melatih model model_0. Menggunakan dataloader pelatihan (train_dataloader), dataloader pengujian (test_dataloader), optimizer yang telah dibuat, dan fungsi kerugian CrossEntropyLoss. Pelatihan dilakukan selama 5 epoch.

Hasil dari pelatihan (kerugian dan akurasi selama pelatihan dan evaluasi) disimpan dalam dictionary model_0_results, yang dapat digunakan untuk menganalisis performa model. Pada akhir

pelatihan, informasi tentang kerugian dan akurasi pada setiap epoch akan dicetak ke layar.

Selanjutnya, hasil pelatihan dapat digunakan untuk membuat plot atau melakukan analisis lebih lanjut untuk memahami sejauh mana model telah belajar dari dataset pelatihan dan sejauh mana performanya 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 [00:40<00:00, 2.13s/it]
Epoch: 1 | train_loss: 1.1180 | train_acc: 0.2756 | test_loss: 1.0962 | test_acc: 0.3333
Epoch: 2 | train_loss: 1.1011 | train_acc: 0.3422 | test_loss: 1.0998 | test_acc: 0.2533
Epoch: 3 | train_loss: 1.1087 | train_acc: 0.3333 | test_loss: 1.1002 | test_acc: 0.2533
Epoch: 4 | train_loss: 1.0992 | train_acc: 0.3067 | test_loss: 1.1007 | test_acc: 0.2533
Epoch: 5 | train_loss: 1.0992 | train_acc: 0.3378 | test_loss: 1.1010 | test_acc: 0.3067
Epoch: 6 | train_loss: 1.0991 | train_acc: 0.3467 | test_loss: 1.1016 | test_acc: 0.3333
Epoch: 7 | train_loss: 1.0988 | train_acc: 0.3556 | test_loss: 1.1016 | test_acc: 0.3333
Epoch: 8 | train_loss: 1.1003 | train_acc: 0.3156 | test_loss: 1.1021 | test_acc: 0.3333
Epoch: 9 | train_loss: 1.0987 | train_acc: 0.3422 | test_loss: 1.1013 | test_acc: 0.3333
Epoch: 10 | train_loss: 1.0988 | train_acc: 0.3467 | test_loss: 1.1014 | test_acc: 0.3333
Epoch: 11 | train_loss: 1.1013 | train_acc: 0.3467 | test_loss: 1.1037 | test_acc: 0.3333
Epoch: 12 | train_loss: 1.0988 | train_acc: 0.3467 | test_loss: 1.1008 | test_acc: 0.3333
Epoch: 13 | train_loss: 1.0988 | train_acc: 0.3467 | test_loss: 1.1014 | test_acc: 0.3333
Epoch: 14 | train_loss: 1.0987 | train_acc: 0.3467 | test_loss: 1.1011 | test_acc: 0.3333
Epoch: 15 | train_loss: 1.0987 | train_acc: 0.3467 | test_loss: 1.1015 | test_acc: 0.3333
Epoch: 16 | train_loss: 1.0987 | train_acc: 0.3467 | test_loss: 1.1012 | test_acc: 0.3333
Epoch: 17 | train_loss: 1.0988 | train_acc: 0.3467 | test_loss: 1.1021 | test_acc: 0.3333
Epoch: 18 | train_loss: 1.0986 | train_acc: 0.3467 | test_loss: 1.1019 | test_acc: 0.3333
Epoch: 19 | train_loss: 1.0986 | train_acc: 0.3467 | test_loss: 1.1017 | test_acc: 0.3333
Epoch: 20 | train_loss: 1.0986 | train_acc: 0.3467 | test_loss: 1.1016 | test_acc: 0.3333
```

Kode ini melakukan pelatihan model model_1 menggunakan fungsi train selama 20 epoch. Berikut adalah penjelasan dari setiap bagian kode tersebut:

torch.manual_seed(42), torch.cuda.manual_seed(42): Mengatur seed (benih) untuk mengontrol keacakan pada library PyTorch. Ini dilakukan untuk memastikan hasil yang dapat direproduksi.

model_1 = TinyVGG(input_shape=3, hidden_units=10, output_shape=len(class_names)).to(device): Membuat instance dari model TinyVGG dengan parameter-parameter tertentu dan mengirimkan model tersebut ke perangkat yang ditentukan (device).

loss_fn = nn.CrossEntropyLoss(): Membuat instance dari fungsi kerugian CrossEntropyLoss, yang umum digunakan untuk tugas klasifikasi.

optimizer = torch.optim.Adam(model_1.parameters(), lr=0.001): Membuat instance dari optimizer Adam untuk mengoptimalkan parameter-parameter model model_1. Learning rate (tingkat pembelajaran) diatur sebesar 0.001.

model_1_results = train(...): Memanggil fungsi train untuk melatih model model_1. Menggunakan dataloader pelatihan (train_dataloader), dataloader pengujian (test_dataloader), optimizer yang telah dibuat, dan fungsi kerugian CrossEntropyLoss. Pelatihan dilakukan selama 20 epoch.

Hasil dari pelatihan (kerugian dan akurasi selama pelatihan dan evaluasi) disimpan dalam dictionary model_1_results, yang dapat digunakan untuk menganalisis performa model. Pada akhir pelatihan, informasi tentang kerugian dan akurasi pada setiap epoch akan dicetak ke layar.

Pelatihan selama 20 epoch dapat memberikan model lebih banyak kesempatan untuk belajar dari dataset pelatihan, tetapi juga dapat meningkatkan risiko overfitting jika tidak diatasi. Oleh karena itu, penting untuk memonitor performa model pada dataset pengujian selama pelatihan untuk memastikan generalisasi yang baik.

```python
# 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 [01:40<00:00, 2.01s/it]
Epoch: 1 | train_loss: 1.1180 | train_acc: 0.2800 | test_loss: 1.0962 | test_acc: 0.3333
Epoch: 2 | train_loss: 1.1011 | train_acc: 0.3289 | test_loss: 1.0999 | test_acc: 0.2533
Epoch: 3 | train_loss: 1.0994 | train_acc: 0.2978 | test_loss: 1.1000 | test_acc: 0.3333
Epoch: 4 | train_loss: 1.0990 | train_acc: 0.3244 | test_loss: 1.1009 | test_acc: 0.3467
Epoch: 5 | train_loss: 1.0987 | train_acc: 0.3467 | test_loss: 1.1027 | test_acc: 0.3333
Epoch: 6 | train_loss: 1.1017 | train_acc: 0.3422 | test_loss: 1.1007 | test_acc: 0.3333
Epoch: 7 | train_loss: 1.0984 | train_acc: 0.3467 | test_loss: 1.1008 | test_acc: 0.3333
Epoch: 8 | train_loss: 1.0981 | train_acc: 0.3467 | test_loss: 1.1013 | test_acc: 0.3333
Epoch: 9 | train_loss: 1.0976 | train_acc: 0.3467 | test_loss: 1.1062 | test_acc: 0.3733
Epoch: 10 | train_loss: 1.1213 | train_acc: 0.3244 | test_loss: 1.1010 | test_acc: 0.3333
Epoch: 11 | train_loss: 1.0994 | train_acc: 0.3378 | test_loss: 1.1053 | test_acc: 0.3333
Epoch: 12 | train_loss: 1.0968 | train_acc: 0.3511 | test_loss: 1.0950 | test_acc: 0.3600
Epoch: 13 | train_loss: 1.0918 | train_acc: 0.4222 | test_loss: 1.0958 | test_acc: 0.3333
Epoch: 14 | train_loss: 1.0555 | train_acc: 0.4267 | test_loss: 1.0693 | test_acc: 0.4533
Epoch: 15 | train_loss: 0.9978 | train_acc: 0.5467 | test_loss: 1.0310 | test_acc: 0.4400
Epoch: 16 | train_loss: 0.9166 | train_acc: 0.5467 | test_loss: 1.1157 | test_acc: 0.4667
Epoch: 17 | train_loss: 0.8722 | train_acc: 0.6089 | test_loss: 1.0179 | test_acc: 0.5067
Epoch: 18 | train_loss: 0.8566 | train_acc: 0.6533 | test_loss: 1.0088 | test_acc: 0.4800
Epoch: 19 | train_loss: 0.8633 | train_acc: 0.6222 | test_loss: 1.0502 | test_acc: 0.4667
Epoch: 20 | train_loss: 0.7731 | train_acc: 0.6444 | test_loss: 1.3991 | test_acc: 0.5600
Epoch: 21 | train_loss: 0.7442 | train_acc: 0.7111 | test_loss: 1.0973 | test_acc: 0.4933
Epoch: 22 | train_loss: 0.7060 | train_acc: 0.7022 | test_loss: 1.4799 | test_acc: 0.5067
Epoch: 23 | train_loss: 0.7371 | train_acc: 0.6800 | test_loss: 1.3060 | test_acc: 0.5067
Epoch: 24 | train_loss: 0.7103 | train_acc: 0.7244 | test_loss: 1.3174 | test_acc: 0.5467
Epoch: 25 | train_loss: 0.8493 | train_acc: 0.6800 | test_loss: 1.1371 | test_acc: 0.4933
Epoch: 26 | train_loss: 0.6510 | train_acc: 0.7200 | test_loss: 1.5190 | test_acc: 0.5333
Epoch: 27 | train_loss: 0.6287 | train_acc: 0.7422 | test_loss: 1.7934 | test_acc: 0.4667
Epoch: 28 | train_loss: 0.5791 | train_acc: 0.7556 | test_loss: 1.5602 | test_acc: 0.5200
Epoch: 29 | train_loss: 0.5550 | train_acc: 0.7556 | test_loss: 2.1516 | test_acc: 0.5200
Epoch: 30 | train_loss: 0.5380 | train_acc: 0.7867 | test_loss: 1.4884 | test_acc: 0.4533
Epoch: 31 | train_loss: 0.4827 | train_acc: 0.7956 | test_loss: 1.8377 | test_acc: 0.5467
Epoch: 32 | train_loss: 0.5000 | train_acc: 0.7822 | test_loss: 2.0971 | test_acc: 0.5333
Epoch: 33 | train_loss: 0.3777 | train_acc: 0.8444 | test_loss: 2.6692 | test_acc: 0.4933
Epoch: 34 | train_loss: 0.3457 | train_acc: 0.8800 | test_loss: 2.4670 | test_acc: 0.5467
Epoch: 35 | train_loss: 0.3266 | train_acc: 0.8756 | test_loss: 2.0669 | test_acc: 0.5333
Epoch: 36 | train_loss: 0.2760 | train_acc: 0.9156 | test_loss: 3.0763 | test_acc: 0.4800
Epoch: 37 | train_loss: 0.2459 | train_acc: 0.9200 | test_loss: 2.9424 | test_acc: 0.5067
Epoch: 38 | train_loss: 0.1888 | train_acc: 0.9244 | test_loss: 3.1968 | test_acc: 0.4933
Epoch: 39 | train_loss: 0.1261 | train_acc: 0.9644 | test_loss: 3.4818 | test_acc: 0.4933
Epoch: 40 | train_loss: 0.1424 | train_acc: 0.9511 | test_loss: 3.3663 | test_acc: 0.5467
Epoch: 41 | train_loss: 0.1091 | train_acc: 0.9600 | test_loss: 3.8679 | test_acc: 0.4933
Epoch: 42 | train_loss: 0.1012 | train_acc: 0.9733 | test_loss: 4.1705 | test_acc: 0.4667
Epoch: 43 | train_loss: 0.1217 | train_acc: 0.9556 | test_loss: 3.6907 | test_acc: 0.5333
Epoch: 44 | train_loss: 0.0941 | train_acc: 0.9511 | test_loss: 3.6759 | test_acc: 0.5733
Epoch: 45 | train_loss: 0.1644 | train_acc: 0.9467 | test_loss: 4.7109 | test_acc: 0.5067
Epoch: 46 | train_loss: 0.1043 | train_acc: 0.9644 | test_loss: 2.6190 | test_acc: 0.4000
Epoch: 47 | train_loss: 0.2419 | train_acc: 0.9511 | test_loss: 4.9406 | test_acc: 0.4400
Epoch: 48 | train_loss: 0.2450 | train_acc: 0.9422 | test_loss: 4.3262 | test_acc: 0.5333
Epoch: 49 | train_loss: 0.0495 | train_acc: 0.9911 | test_loss: 4.5027 | test_acc: 0.5067
Epoch: 50 | train_loss: 0.0301 | train_acc: 0.9867 | test_loss: 4.4475 | test_acc: 0.5467
```

Sepertinya model kita mulai mengalami overfitting pada akhirnya (kinerjanya jauh lebih baik pada data pelatihan dibandingkan dengan data pengujian).

Untuk mengatasi ini, kita perlu memperkenalkan cara-cara untuk mencegah overfitting.
Kode ini melakukan pelatihan model model_2 menggunakan fungsi train selama 50 epoch. Berikut adalah penjelasan dari setiap bagian kode tersebut:

torch.manual_seed(42), torch.cuda.manual_seed(42): Mengatur seed (benih) untuk mengontrol keacakan pada library PyTorch. Ini dilakukan untuk memastikan hasil yang dapat direproduksi.

model_2 = TinyVGG(input_shape=3, hidden_units=10, output_shape=len(class_names)).to(device): Membuat instance dari model TinyVGG dengan parameter-parameter tertentu dan mengirimkan model tersebut ke perangkat yang ditentukan (device).

loss_fn = nn.CrossEntropyLoss(): Membuat instance dari fungsi kerugian CrossEntropyLoss, yang umum digunakan untuk tugas klasifikasi.

optimizer = torch.optim.Adam(model_2.parameters(), lr=0.001): Membuat instance dari optimizer Adam untuk mengoptimalkan parameter-parameter model model_2. Learning rate (tingkat pembelajaran) diatur sebesar 0.001.

model_2_results = train(...): Memanggil fungsi train untuk melatih model model_2. Menggunakan dataloader pelatihan (train_dataloader), dataloader pengujian (test_dataloader), optimizer yang telah dibuat, dan fungsi kerugian CrossEntropyLoss. Pelatihan dilakukan selama 50 epoch.

Hasil dari pelatihan (kerugian dan akurasi selama pelatihan dan evaluasi) disimpan dalam dictionary model_2_results, yang dapat digunakan untuk menganalisis performa model. Pada akhir pelatihan, informasi tentang kerugian dan akurasi pada setiap epoch akan dicetak ke layar.

Pelatihan selama 50 epoch memberikan model lebih banyak kesempatan untuk belajar dari dataset pelatihan, namun perlu diingat untuk memonitor performa model pada dataset pengujian untuk memastikan generalisasi yang baik dan mengidentifikasi potensi overfitting.

6. Double the number od hidden units in your model and train it for 2 epoch, what happens to the result?

```
# 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 [00:40<00:00, 2.02s/it]
Epoch: 1 | train_loss: 1.0885 | train_acc: 0.4311 | test_loss: 1.0891 | test_acc: 0.3733
Epoch: 2 | train_loss: 0.9876 | train_acc: 0.5600 | test_loss: 1.0393 | test_acc: 0.4267
Epoch: 3 | train_loss: 0.9533 | train_acc: 0.5733 | test_loss: 1.0021 | test_acc: 0.4800
Epoch: 4 | train_loss: 0.8844 | train_acc: 0.6089 | test_loss: 1.0621 | test_acc: 0.4667
Epoch: 5 | train_loss: 0.8777 | train_acc: 0.5733 | test_loss: 1.0660 | test_acc: 0.3600
Epoch: 6 | train_loss: 0.8130 | train_acc: 0.6400 | test_loss: 1.0001 | test_acc: 0.4267
Epoch: 7 | train_loss: 0.7961 | train_acc: 0.6800 | test_loss: 1.1541 | test_acc: 0.4667
Epoch: 8 | train_loss: 0.7252 | train_acc: 0.7022 | test_loss: 1.0889 | test_acc: 0.4667
Epoch: 9 | train_loss: 0.6729 | train_acc: 0.7022 | test_loss: 1.0845 | test_acc: 0.4667
Epoch: 10 | train_loss: 0.6619 | train_acc: 0.7333 | test_loss: 1.1311 | test_acc: 0.4800
Epoch: 11 | train_loss: 0.6275 | train_acc: 0.7156 | test_loss: 1.1086 | test_acc: 0.4667
Epoch: 12 | train_loss: 0.5418 | train_acc: 0.7511 | test_loss: 1.3979 | test_acc: 0.4267
Epoch: 13 | train_loss: 0.5407 | train_acc: 0.7689 | test_loss: 1.2876 | test_acc: 0.5467
Epoch: 14 | train_loss: 0.4709 | train_acc: 0.8133 | test_loss: 1.4039 | test_acc: 0.5467
Epoch: 15 | train_loss: 0.3491 | train_acc: 0.8578 | test_loss: 1.6932 | test_acc: 0.5067
Epoch: 16 | train_loss: 0.3077 | train_acc: 0.8933 | test_loss: 1.9600 | test_acc: 0.5067
Epoch: 17 | train_loss: 0.3277 | train_acc: 0.8356 | test_loss: 1.6580 | test_acc: 0.5867
Epoch: 18 | train_loss: 0.2825 | train_acc: 0.8756 | test_loss: 2.0573 | test_acc: 0.5867
Epoch: 19 | train_loss: 0.3079 | train_acc: 0.8756 | test_loss: 1.8595 | test_acc: 0.5867
Epoch: 20 | train_loss: 0.2437 | train_acc: 0.8978 | test_loss: 1.6954 | test_acc: 0.5733
```

Kode ini melibatkan pelatihan model model_3 dengan menggunakan 20 unit tersembunyi (hidden units), dua kali lipat dari jumlah unit tersembunyi yang digunakan pada model_1. Berikut adalah penjelasan dari setiap bagian kode tersebut:

torch.manual_seed(42), torch.cuda.manual_seed(42): Mengatur seed (benih) untuk mengontrol keacakan pada library PyTorch. Ini dilakukan untuk memastikan hasil yang dapat direproduksi.

model_3 = TinyVGG(input_shape=3, hidden_units=20, output_shape=len(class_names)).to(device): Membuat instance dari model TinyVGG dengan menggunakan 20 unit tersembunyi, dua kali lipat dari jumlah yang digunakan sebelumnya, dan mengirimkan model tersebut ke perangkat yang ditentukan (device).

loss_fn = nn.CrossEntropyLoss(): Membuat instance dari fungsi kerugian CrossEntropyLoss, yang umum digunakan untuk tugas klasifikasi.

optimizer = torch.optim.Adam(model_3.parameters(), lr=0.001): Membuat instance dari optimizer Adam untuk mengoptimalkan parameter-parameter model model_3. Learning rate (tingkat pembelajaran) diatur sebesar 0.001.

model_3_results = train(...): Memanggil fungsi train untuk melatih model model_3. Menggunakan dataloader pelatihan (train_dataloader), dataloader pengujian (test_dataloader), optimizer yang telah dibuat, dan fungsi kerugian CrossEntropyLoss. Pelatihan dilakukan selama 20 epoch.

Hasil dari pelatihan (kerugian dan akurasi selama pelatihan dan evaluasi) disimpan dalam dictionary model_3_results, yang dapat digunakan untuk menganalisis performa model. Pada akhir pelatihan, informasi tentang kerugian dan akurasi pada setiap epoch akan dicetak ke layar.

Peningkatan jumlah unit tersembunyi dapat meningkatkan kapasitas model, tetapi juga meningkatkan risiko overfitting. Oleh karena itu, penting untuk memonitor performa model pada dataset pengujian selama pelatihan untuk memastikan generalisasi yang baik.
Sepertinya model masih mengalami overfitting, bahkan ketika mengubah jumlah unit tersembunyi.

Untuk mengatasi ini, kita perlu mencari cara-cara untuk mencegah overfitting pada model kita.

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

```
# 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_su
    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 memiliki tujuan untuk mengunduh dan menyiapkan dataset yang berisi 20% dari dataset asli untuk kelas-kelas pizza, steak,

dan sushi. Berikut adalah penjelasan mengenai setiap bagian kode tersebut:

data_path = Path("data/"): Mengatur path (lokasi) untuk folder data.

image_path = data_path / "pizza_steak_sushi_20_percent": Menentukan path (lokasi) untuk folder yang akan berisi dataset yang sudah diunduh dan disiapkan.

if image_path.is_dir():: Mengecek apakah folder image_path sudah ada atau belum.

image_path.mkdir(parents=True, exist_ok=True): Jika folder image_path belum ada, maka kode akan membuatnya. Pengaturan parents=True memastikan bahwa folder induk yang tidak ada juga akan dibuat, dan exist_ok=True memastikan bahwa tidak ada exception akan dihasilkan jika folder tersebut sudah ada.

Download Data:

with open(data_path / "pizza_steak_sushi_20_percent.zip", "wb") as f:: Membuka file zip untuk menyimpan dataset yang akan diunduh.
request = requests.get("https://github.com/mrdbourke/pytorch-deep-learning/raw/main/data/pizza_steak_sushi_20_percent.zip"): Mengirim permintaan HTTP untuk mengunduh dataset dari GitHub.
f.write(request.content): Menyimpan isi dataset yang diunduh ke dalam file zip.
Unzip Data:

with zipfile.ZipFile(data_path / "pizza_steak_sushi_20_percent.zip", "r") as zip_ref:: Membuka file zip untuk mengekstrak isinya.
zip_ref.extractall(image_path): Mengekstrak semua file dari zip ke dalam folder image_path.
Dengan eksekusi kode tersebut, dataset yang berisi 20% dari dataset asli untuk kelas-kelas pizza, steak, dan sushi akan diunduh, diekstrak, dan disimpan di folder pizza_steak_sushi_20_percent.

```
[ ]  # 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 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 0 directories and 154 images in 'data/pizza_steak_sushi_20_percent/train/pizza'.
     There are 3 directories and 0 images in 'data/pizza_steak_sushi_20_percent/test'.
     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'.
     There are 0 directories and 46 images in 'data/pizza_steak_sushi_20_percent/test/pizza'.
```

Kode ini menggunakan fungsi walk_through_dir untuk menampilkan jumlah file (gambar) di dalam folder yang ditentukan oleh image_path. Fungsi walk_through_dir telah didefinisikan sebelumnya dalam sesi ini dan memberikan informasi tentang jumlah direktori dan jumlah file dalam suatu path.

walk_through_dir(image_path): Memanggil fungsi walk_through_dir dengan argumen image_path. Fungsi ini akan mengeksplorasi struktur folder di dalam image_path dan mencetak jumlah direktori dan jumlah file gambar di setiap direktori.
Output dari kode ini akan memberikan informasi tentang struktur dataset yang baru diunduh, termasuk jumlah gambar di setiap kelas (pizza, steak, sushi). Dengan informasi ini, Anda dapat memahami seberapa besar dataset yang akan digunakan untuk pelatihan dan evaluasi 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

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

     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 menentukan path (lokasi) untuk folder pelatihan (train_data_20_percent_path) dan folder pengujian (test_data_20_percent_path) dari dataset yang baru diunduh dan dipersiapkan (20% dari dataset asli). Berikut adalah penjelasan dari setiap bagian kode tersebut:

train_data_20_percent_path = image_path / "train": Menentukan path untuk folder pelatihan. Folder ini diharapkan berisi data pelatihan dari dataset yang baru diunduh.

test_data_20_percent_path = image_path / "test": Menentukan path untuk folder pengujian. Folder ini diharapkan berisi data pengujian dari dataset yang baru diunduh.

Kedua path ini akan digunakan untuk membuat dataset PyTorch menggunakan ImageFolder dan selanjutnya digunakan untuk membuat dataloader untuk pelatihan dan pengujian. Dengan membagi data menjadi dua set (pelatihan dan pengujian), kita dapat mengukur performa model pada dataset pengujian yang belum pernah dilihat sebelumnya untuk mendapatkan indikasi yang lebih baik tentang generalisasi model.
Kode ini menggunakan modul PyTorch torchvision untuk membuat dataset dan dataloader dari dataset yang baru diunduh dan dipersiapkan (20% dari dataset asli). Berikut adalah penjelasan dari setiap bagian kode tersebut:

simple_transform = transforms.Compose([...]): Membuat transformasi sederhana menggunakan transforms.Compose. Transformasi ini mencakup resizing gambar menjadi ukuran (64, 64) dan mengonversi gambar menjadi tensor menggunakan transforms.ToTensor().

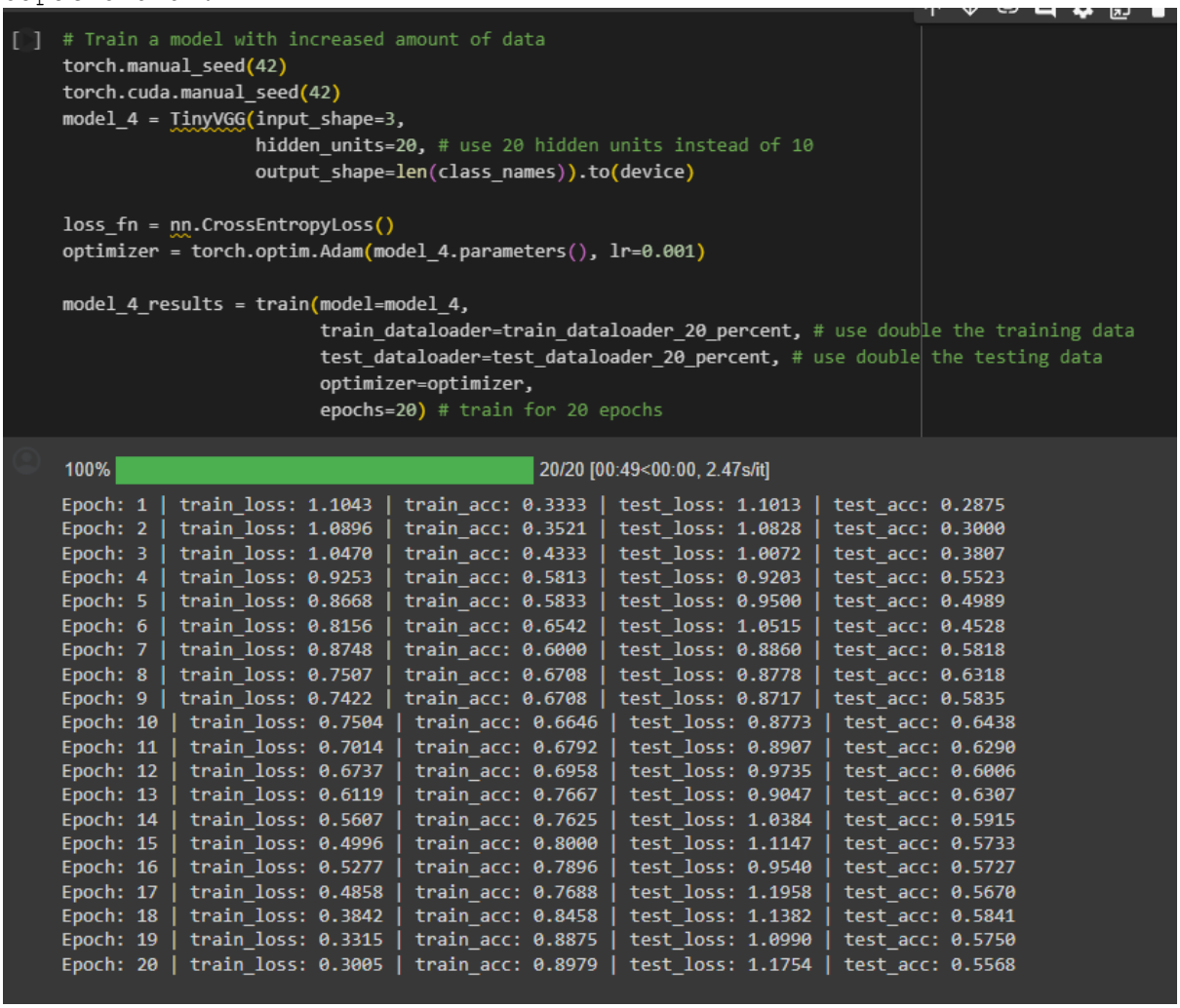
train_data_20_percent = ImageFolder(train_data_20_percent_path, transform=simple_transform): Membuat instance dari ImageFolder untuk dataset pelatihan. ImageFolder secara otomatis mengenali struktur folder yang sesuai dengan struktur dataset ImageNet, yaitu direktori kelas-kelas yang berisi gambar-gambar. Argument transform=simple_transform menunjukkan bahwa transformasi yang telah dibuat sebelumnya akan diterapkan pada setiap gambar dalam dataset pelatihan.

test_data_20_percent = ImageFolder(test_data_20_percent_path, transform=simple_transform): Membuat instance dari ImageFolder untuk dataset pengujian dengan konsep yang serupa.

Create Dataloaders:

train_dataloader_20_percent = DataLoader(train_data_20_percent, batch_size=32, num_workers=os.cpu_count(), shuffle=True): Membuat dataloader untuk dataset pelatihan. Dataloader ini memungkinkan untuk memuat data dalam batch dengan ukuran 32, menggunakan sejumlah pekerja yang setara dengan jumlah CPU yang tersedia (os.cpu_count()), dan mengacak urutan data (shuffle=True) setiap kali epoch dimulai.

test_dataloader_20_percent = DataLoader(test_data_20_percent, batch_size=32, num_workers=os.cpu_count(), shuffle=False): Membuat dataloader untuk dataset pengujian dengan konsep yang serupa. Namun, dalam hal ini, data pengujian tidak diacak (shuffle=False) untuk memastikan hasil pengujian konsisten dan dapat diukur.



```
[ ] # 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 [00:49<00:00, 2.47s/it]
Epoch: 1 | train_loss: 1.1043 | train_acc: 0.3333 | test_loss: 1.1013 | test_acc: 0.2875
Epoch: 2 | train_loss: 1.0896 | train_acc: 0.3521 | test_loss: 1.0828 | test_acc: 0.3000
Epoch: 3 | train_loss: 1.0470 | train_acc: 0.4333 | test_loss: 1.0072 | test_acc: 0.3807
Epoch: 4 | train_loss: 0.9253 | train_acc: 0.5813 | test_loss: 0.9203 | test_acc: 0.5523
Epoch: 5 | train_loss: 0.8668 | train_acc: 0.5833 | test_loss: 0.9500 | test_acc: 0.4989
Epoch: 6 | train_loss: 0.8156 | train_acc: 0.6542 | test_loss: 1.0515 | test_acc: 0.4528
Epoch: 7 | train_loss: 0.8748 | train_acc: 0.6000 | test_loss: 0.8860 | test_acc: 0.5818
Epoch: 8 | train_loss: 0.7507 | train_acc: 0.6708 | test_loss: 0.8778 | test_acc: 0.6318
Epoch: 9 | train_loss: 0.7422 | train_acc: 0.6708 | test_loss: 0.8717 | test_acc: 0.5835
Epoch: 10 | train_loss: 0.7504 | train_acc: 0.6646 | test_loss: 0.8773 | test_acc: 0.6438
Epoch: 11 | train_loss: 0.7014 | train_acc: 0.6792 | test_loss: 0.8907 | test_acc: 0.6290
Epoch: 12 | train_loss: 0.6737 | train_acc: 0.6958 | test_loss: 0.9735 | test_acc: 0.6006
Epoch: 13 | train_loss: 0.6119 | train_acc: 0.7667 | test_loss: 0.9047 | test_acc: 0.6307
Epoch: 14 | train_loss: 0.5607 | train_acc: 0.7625 | test_loss: 1.0384 | test_acc: 0.5915
Epoch: 15 | train_loss: 0.4996 | train_acc: 0.8000 | test_loss: 1.1147 | test_acc: 0.5733
Epoch: 16 | train_loss: 0.5277 | train_acc: 0.7896 | test_loss: 0.9540 | test_acc: 0.5727
Epoch: 17 | train_loss: 0.4858 | train_acc: 0.7688 | test_loss: 1.1958 | test_acc: 0.5670
Epoch: 18 | train_loss: 0.3842 | train_acc: 0.8458 | test_loss: 1.1382 | test_acc: 0.5841
Epoch: 19 | train_loss: 0.3315 | train_acc: 0.8875 | test_loss: 1.0990 | test_acc: 0.5750
Epoch: 20 | train_loss: 0.3005 | train_acc: 0.8979 | test_loss: 1.1754 | test_acc: 0.5568
```

Kode ini melakukan pelatihan model model_4 dengan menggunakan dataset yang telah diperbarui, yaitu dataset yang baru diunduh dan dipersiapkan (20% dari dataset asli). Berikut adalah penjelasan dari setiap bagian kode tersebut:

torch.manual_seed(42), torch.cuda.manual_seed(42): Mengatur seed (benih) untuk mengontrol keacakan pada library PyTorch. Ini dilakukan untuk memastikan hasil yang dapat direproduksi.

model_4 = TinyVGG(input_shape=3, hidden_units=20, output_shape=len(class_names)).to(device): Membuat instance dari model TinyVGG dengan menggunakan 20 unit tersembunyi, dua kali lipat dari jumlah yang digunakan pada model_1, dan mengirimkan model tersebut ke perangkat yang ditentukan (device).

loss_fn = nn.CrossEntropyLoss(): Membuat instance dari fungsi kerugian CrossEntropyLoss, yang umum digunakan untuk tugas klasifikasi.

optimizer = torch.optim.Adam(model_4.parameters(), lr=0.001): Membuat instance dari optimizer Adam untuk mengoptimalkan parameter-parameter model model_4. Learning rate (tingkat pembelajaran) diatur sebesar 0.001.

model_4_results = train(...): Memanggil fungsi train untuk melatih model model_4. Menggunakan dataloader pelatihan yang telah diperbarui (train_dataloader_20_percent), dataloader pengujian yang telah diperbarui (test_dataloader_20_percent), optimizer yang telah dibuat, dan fungsi kerugian CrossEntropyLoss. Pelatihan dilakukan selama 20 epoch.

Hasil dari pelatihan (kerugian dan akurasi selama pelatihan dan evaluasi) disimpan dalam dictionary model_4_results, yang dapat digunakan untuk menganalisis performa model. Pada akhir pelatihan, informasi tentang kerugian dan akurasi pada setiap epoch akan dicetak ke layar.

Dengan menggunakan dataset yang lebih besar (20% dari dataset asli), diharapkan model dapat memperoleh pemahaman yang lebih baik tentang data dan meningkatkan kemampuannya untuk generalisasi.

Model kita masih mengalami overfitting (karena kerugian pada data pelatihan jauh lebih rendah daripada kerugian pada data pengujian), bahkan dengan menggunakan jumlah data ganda...

Sepertinya kita perlu menyelidiki metode-metode lain untuk mengurangi overfitting dan membangun model yang lebih baik.

8. Make a prediction on your own custem image of pizza/steak/sushi (you could even download one from the internet) with your trained model from exercise 7 and ahre 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-
  f.write(request.content)
```

+ Code + Text

```
[ ] # 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]],

        [[119, 138, 147,  ...,  17,  14,  10],
         [111, 130, 145,  ...,  17,  14,  11],
         [ 87, 111, 136,  ...,  14,  13,  11],
         ...,
         [ 35,  22,   8,  ...,  52,  52,  48],
         [ 27,  18,   4,  ...,  50,  49,  44],
         [ 27,  23,   9,  ...,  49,  46,  43]]], dtype=torch.uint8)
```

Kode ini memiliki tujuan untuk mendapatkan gambar kustom dari internet, dalam hal ini disebut "pizza_dad.jpeg", dan memuatnya ke dalam notebook. Berikut adalah penjelasan dari setiap bagian kode tersebut:

custom_image = "pizza_dad.jpeg": Menyimpan nama file gambar kustom yang akan diunduh.

with open("pizza_dad.jpeg", "wb") as f:: Membuka file dengan nama "pizza_dad.jpeg" dalam mode penulisan biner ("wb"). File ini akan digunakan untuk menyimpan konten gambar yang diunduh.

request = requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-deep-learning/main/images/04-pizza-dad.jpeg"): Mengirimkan permintaan HTTP untuk mengunduh gambar dari URL yang diberikan. Gambar ini disediakan oleh penyedia gambar yang bersumber dari GitHub.

f.write(request.content): Menyimpan isi gambar yang diunduh ke dalam file "pizza_dad.jpeg". Dengan cara ini, gambar tersebut akan tersedia secara lokal di dalam notebook.

import torchvision: Mengimpor modul torchvision dari PyTorch, yang menyediakan fungsi-fungsi untuk bekerja dengan visi komputer.

img = torchvision.io.read_image(custom_image): Membaca gambar dari file dengan nama "pizza_dad.jpeg" menggunakan fungsi read_image dari modul torchvision.io. Hasilnya adalah representasi tensor dari gambar tersebut.

img: Menampilkan representasi tensor gambar yang telah dimuat. Tensor ini dapat digunakan untuk memasukkan gambar ke dalam model atau melakukan operasi lainnya yang berhubungan dengan visi komputer di dalam lingkungan PyTorch.

```
# 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 ini bertujuan untuk membuat prediksi menggunakan model model_4 pada gambar yang telah dimuat sebelumnya ("pizza_dad.jpeg"). Berikut adalah penjelasan dari setiap bagian kode tersebut:

model_4.eval(): Mengatur model ke mode evaluasi. Ini diperlukan karena beberapa lapisan, seperti dropout, dapat berperilaku berbeda antara mode pelatihan dan mode evaluasi.

with torch.inference_mode():: Menggunakan manajer konteks torch.inference_mode() untuk memastikan bahwa komputasi dilakukan tanpa perlu melacak grafik komputasi. Ini meningkatkan efisiensi saat memprediksi.

img = img / 255.: Mengonversi nilai piksel dalam gambar menjadi rentang antara 0 dan 1 dengan membagi setiap nilai piksel oleh 255.

resize = transforms.Resize((64, 64)): Membuat objek transformasi untuk meresize gambar menjadi ukuran (64, 64).

img = resize(img): Mengaplikasikan transformasi resize pada gambar.

batch = img.unsqueeze(0).to(device): Mengubah gambar menjadi batch tunggal dan mengirimkannya ke perangkat target (device).

y_pred_logit = model_4(batch): Memprediksi label logit menggunakan model model_4.

pred_label = torch.argmax(y_pred_logit, dim=1): Mengambil label prediksi dengan menggunakan argmax pada nilai logit. Jika model menggunakan fungsi aktivasi softmax pada outputnya, fungsi softmax tidak perlu diterapkan lagi untuk mendapatkan label prediksi.

plt.imshow(img.permute(1, 2, 0)): Menampilkan gambar yang diprediksi.

plt.title(f"Pred label: {class_names[pred_label]}"): Menampilkan judul plot dengan label prediksi.

plt.axis(False): Menyembunyikan sumbu pada plot.

Dengan kode ini, kita dapat melihat hasil prediksi model pada gambar "pizza_dad.jpeg" bersama dengan label prediksi yang ditampilkan pada plot.