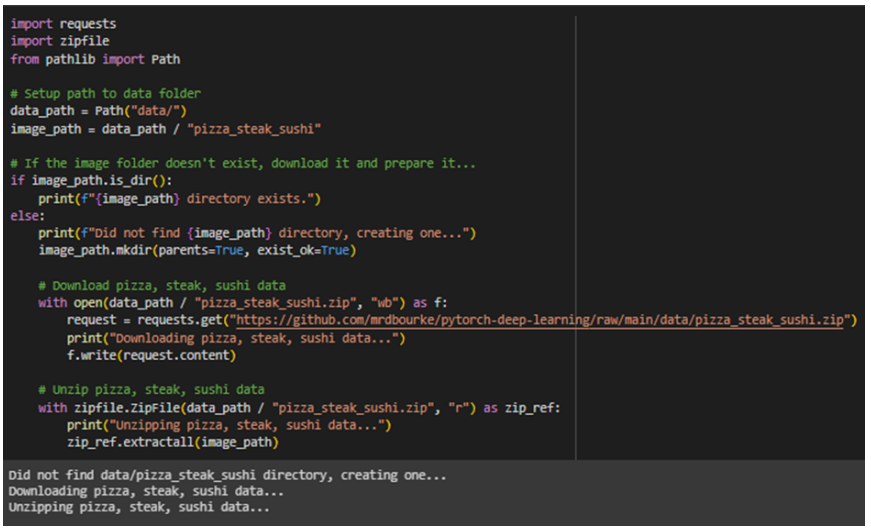
Nama : Andhika Aria Pratama N

NIM : 1103202121

04 PyTorch Custom Datasets

- Mengimpor PyTorch dan menyiapkan kode agnostik perangkat
  Mengimport library yang akan dipakai dalam tahap sekarang.

- Menginisialisasikan 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 one...")
    image_path.mkdir(parents=True, exist_ok=True)

    # Download pizza, steak, sushi data
    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("Unzipping pizza, steak, sushi data...")
        zip_ref.extractall(image_path)
```

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

Mengunduh data dan mengestrak data yang telah diunduh.

- Persiapan data

```python
import os
def walk_through_dir(dir_path):
  """
  Walks through dir_path returning its contents.
  Args:
    dir_path (str or pathlib.Path): target directory

  Returns:
    A print out of:
      number of subdiretories in dir_path
      number of images (files) in each subdirectory
      name of each subdirectory
  """
  for dirpath, dirnames, filenames in os.walk(dir_path):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```python
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'.
```

```python
# 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'))
```

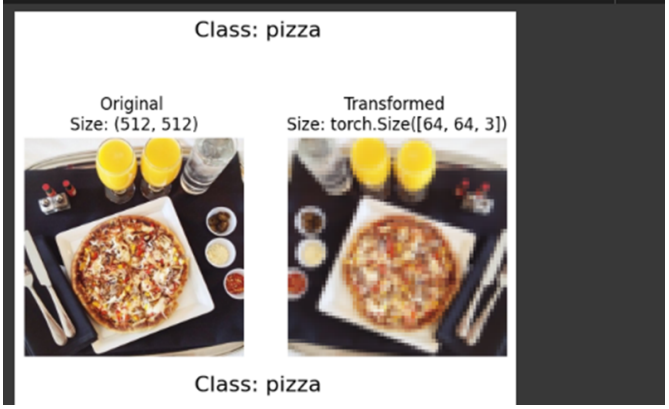
Membagi data train dan test.

- Visualisasi data

Pada tahap ini membuka data secara random.

- Transformasi data

```python
def plot_transformed_images(image_paths, transform, n=3, seed=42):
    """Plots a series of random images from image_paths.

    Will open n image paths from image_paths, transform them
    with transform and plot them side by side.

    Args:
        image_paths (list): List of target image paths.
        transform (PyTorch Transforms): Transforms to apply to images.
        n (int, optional): Number of images to plot. Defaults to 3.
        seed (int, optional): Random seed for the random generator. Defaults to 42.
    """
    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(1, 2)
            ax[0].imshow(f)
            ax[0].set_title(f"Original \nSize: {f.size}")
            ax[0].axis("off")

            # Transform and plot image
            # Note: permute() will change shape of image to suit matplotlib
            # (PyTorch default is [C, H, W] but Matplotlib is [H, W, C])
            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)
```



Mentransformasi data gambar. Transformasi dilakukan dalam pixel pixel dalam gambar yang dirubah ukurannya (shape data) agar menjadi data angka yang akan diolah nantinya.

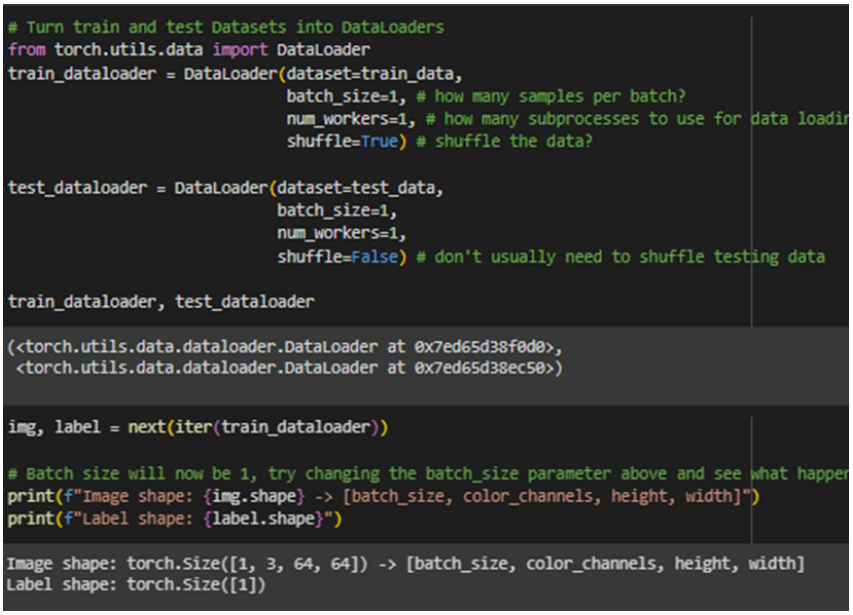
- Opsi 1: Memuat Data Gambar Menggunakan ImageFolder

Pada tahap ini, dilakukan penggunaan `ImageFolder` dari torchvision untuk membuat dataset dari direktori gambar. Dimuatlah dataset pelatihan (`train_data`) dan dataset pengujian (`test_data`) dengan menerapkan transformasi data tertentu. Data train dan test memuat sebanyak 225 data dan 75 data masing-masing. Kemudian, diambil nama kelas (class names) dari dataset pelatihan sebagai daftar (`class_names`) dan sebagai kamus (`class_dict`). Panjang dataset pelatihan dan dataset pengujian juga diperiksa.

Selanjutnya, informasi tentang gambar pertama dari dataset pelatihan dicetak, termasuk tensor gambar, bentuk gambar, tipe data gambar, dan label gambar. Dilakukan pengubahan urutan dimensi tensor gambar menggunakan metode `permute` untuk memungkinkan plot yang benar. Akhirnya, dilakukan visualisasi gambar pertama dari dataset pelatihan dengan menggunakan matplotlib.

Dalam tahap ini, juga terdapat beberapa penyesuaian dimensi tensor gambar dan tampilan visualisasi dari gambar pertama dataset pelatihan.

- Ubah gambar yang dimuat menjadi milik DataLoader



```
# Turn train and test Datasets into DataLoaders
from torch.utils.data import DataLoader
train_dataloader = DataLoader(dataset=train_data,
                              batch_size=1, # how many samples per batch?
                              num_workers=1, # how many subprocesses to use for data loadi
                              shuffle=True) # shuffle the data?

test_dataloader = DataLoader(dataset=test_data,
                             batch_size=1,
                             num_workers=1,
                             shuffle=False) # don't usually need to shuffle testing data

train_dataloader, test_dataloader
```

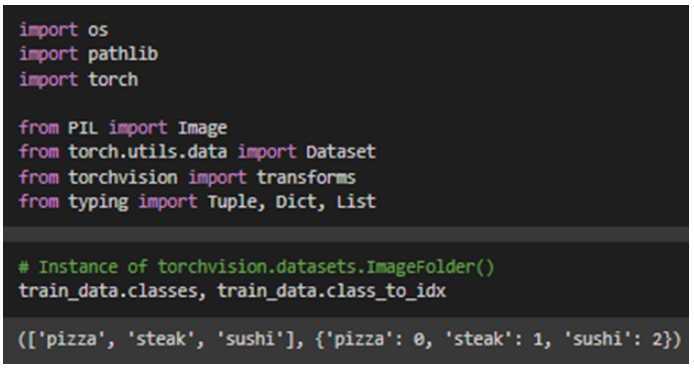
```
(<torch.utils.data.dataloader.DataLoader at 0x7ed65d38f0d0>,
 <torch.utils.data.dataloader.DataLoader at 0x7ed65d38ec50>)
```

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

# Batch size will now be 1, try changing the batch_size parameter above and see what happe
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 tahap ini, dataset pelatihan dan pengujian diubah menjadi DataLoader menggunakan modul `DataLoader` dari PyTorch. DataLoader memungkinkan pemrosesan data dalam batch dengan konfigurasi tertentu seperti ukuran batch, jumlah pekerja, dan pengacakan data. Setelah itu, dilakukan ekstraksi satu batch pertama dari DataLoader pelatihan, dan ditampilkan bentuk tensor gambar dan labelnya. Dengan menggunakan `DataLoader`, data dapat diproses secara efisien dalam batch, dan ukuran batch dapat diatur sesuai kebutuhan, seperti yang dapat dilihat pada bentuk tensor gambar dan label yang dicetak.

- Opsi 2: Memuat Data Gambar dengan Kumpulan Data Khusus



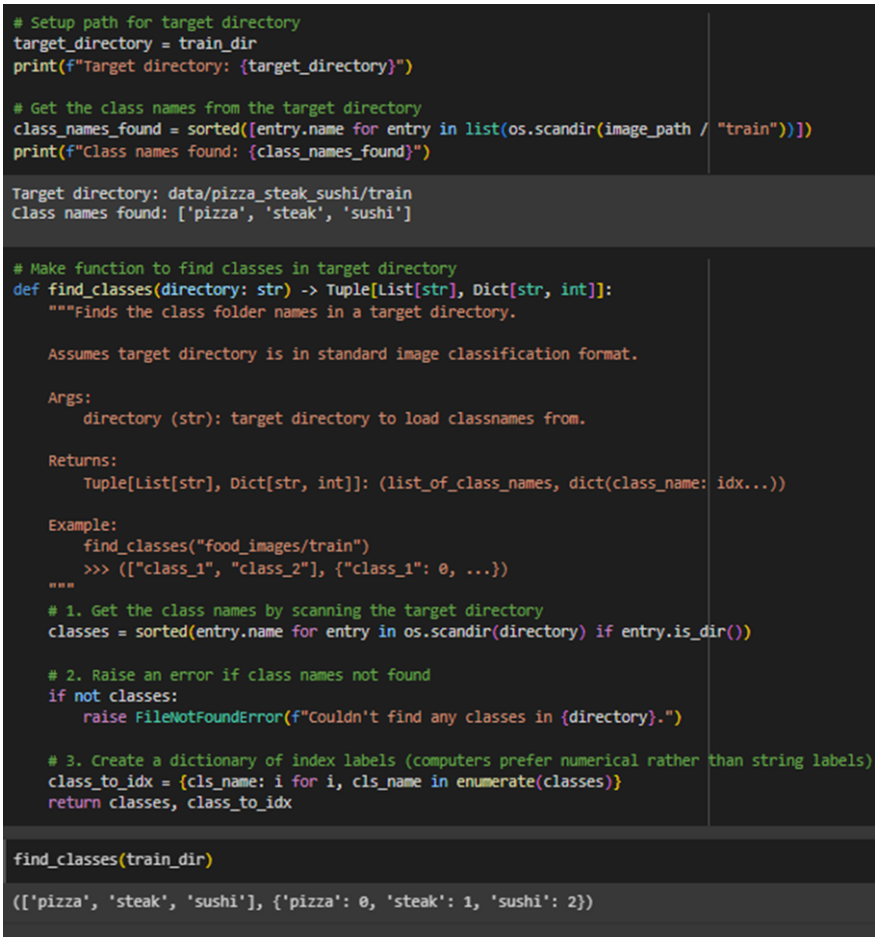
```
import os
import pathlib
import torch

from PIL import Image
from torch.utils.data import Dataset
from torchvision import transforms
from typing import Tuple, Dict, List
```

```
# Instance of torchvision.datasets.ImageFolder()
train_data.classes, train_data.class_to_idx
```

```
(['pizza', 'steak', 'sushi'], {'pizza': 0, 'steak': 1, 'sushi': 2})
```

Memuat library yang akan dipakai serta melihat data train lalu melihat urutan tiap jenisnya.

- Membuat fungsi pembantu untuk mendapatkan nama kelas



```
# Setup path for target directory
target_directory = train_dir
print(f"Target directory: {target_directory}")

# Get the class names from the target directory
class_names_found = sorted([entry.name for entry in list(os.scandir(image_path / "train"))])
print(f"Class names found: {class_names_found}")
```

```
Target directory: data/pizza_steak_sushi/train
Class names found: ['pizza', 'steak', 'sushi']
```

```
# Make function to find classes in target directory
def find_classes(directory: str) -> Tuple[List[str], Dict[str, int]]:
    """Finds the class folder names in a target directory.

    Assumes target directory is in standard image classification format.

    Args:
        directory (str): target directory to load classnames from.

    Returns:
        Tuple[List[str], Dict[str, int]]: (list_of_class_names, dict(class_name: idx...))

    Example:
        find_classes("food_images/train")
        >>> (["class_1", "class_2"], {"class_1": 0, ...})
    """
    # 1. Get the class names by scanning the target directory
    classes = sorted(entry.name for entry in os.scandir(directory) if entry.is_dir())

    # 2. Raise an error if class names not found
    if not classes:
        raise FileNotFoundError(f"Couldn't find any classes in {directory}.")

    # 3. Create a dictionary of index labels (computers prefer numerical rather than string labels)
    class_to_idx = {cls_name: i for i, cls_name in enumerate(classes)}
    return classes, class_to_idx
```

```
find_classes(train_dir)
```

```
(['pizza', 'steak', 'sushi'], {'pizza': 0, 'steak': 1, 'sushi': 2})
```

Pada tahap ini, path untuk direktori target diatur ke `train_dir`, dan kemudian dilakukan pencarian kelas-kelas yang ada di dalamnya. Fungsi `find_classes` dibuat untuk menemukan nama-nama kelas dan membentuk dictionary yang memetakan nama kelas ke indeks numerik. Pencarian tersebut dilakukan dengan menggabungkan data dari target direktori, dan hasilnya menunjukkan tiga kelas, yaitu 'pizza', 'steak', dan 'sushi', bersama dengan dictionary yang memetakan setiap kelas ke indeks numeriknya. Proses ini memastikan bahwa kelas-kelas yang ada di dalam dataset dapat diidentifikasi dan dipetakan ke indeks numerik untuk keperluan pelatihan model.

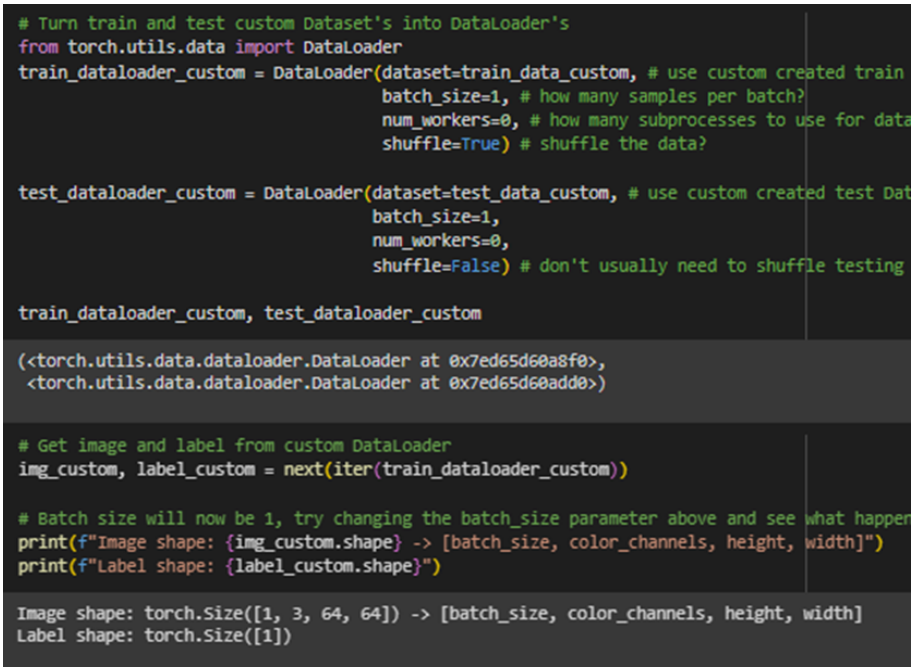
- Buat Kumpulan Data khusus untuk mereplikasi ImageFolder

Dalam tahap ini, sebuah kelas dataset khusus (`ImageFolderCustom`) dibuat, yang mewarisi dari `torch.utils.data.Dataset`. Kelas ini memanfaatkan path dan transformasi (opsional) sebagai parameter inisialisasi, dan memiliki metode untuk memuat gambar, menghitung panjang dataset, dan mendapatkan elemen dataset. Transformasi dapat diterapkan pada gambar sesuai dengan parameter yang diberikan pada inisialisasi. Dataset yang baru (`train_data_custom` dan `test_data_custom`) dibuat menggunakan kelas dataset khusus ini dengan transformasi yang berbeda untuk data pelatihan dan pengujian. Pengecekan dilakukan untuk memastikan bahwa dataset khusus ini setara dengan dataset yang dibuat menggunakan `ImageFolder` standar dalam hal jumlah sampel, kelas, dan pemetaan indeks kelas. Hasilnya menunjukkan bahwa kedua jenis dataset ini setara.

- Buat fungsi untuk menampilkan gambar acak

Pada tahap ini menampilkan beberapa gambar dari hasil train gambar sebelumnya.

- Ubah gambar yang dimuat khusus menjadi milik DataLoader



```
# Turn train and test custom Dataset's into DataLoader's
from torch.utils.data import DataLoader
train_dataloader_custom = DataLoader(dataset=train_data_custom, # use custom created train
                                     batch_size=1, # how many samples per batch?
                                     num_workers=0, # how many subprocesses to use for data
                                     shuffle=True) # shuffle the data?

test_dataloader_custom = DataLoader(dataset=test_data_custom, # use custom created test Dat
                                    batch_size=1,
                                    num_workers=0,
                                    shuffle=False) # don't usually need to shuffle testing

train_dataloader_custom, test_dataloader_custom
```

```
(<torch.utils.data.dataloader.DataLoader at 0x7ed65d60a8f0>,
 <torch.utils.data.dataloader.DataLoader at 0x7ed65d60add0>)
```

```
# Get image and label from custom DataLoader
img_custom, label_custom = next(iter(train_dataloader_custom))

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

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

Pada tahap ini, dataset khusus yang telah dibuat sebelumnya (`train_data_custom` dan `test_data_custom`) diubah menjadi DataLoader khusus (`train_dataloader_custom` dan `test_dataloader_custom`). DataLoader digunakan untuk mengelola data dengan menentukan ukuran batch, jumlah pekerja (num_workers) yang digunakan untuk memuat data, dan apakah data diacak atau tidak. Selanjutnya, dilakukan pengujian dengan mengambil satu batch dari DataLoader khusus untuk melihat bentuk (shape) gambar dan labelnya.

- Membuat transformasi dan memuat data untuk Model 0

Pada tahap ini, sebuah model TinyVGG untuk klasifikasi gambar dibuat dan dilatih menggunakan data DataLoader yang telah disiapkan sebelumnya. Model tersebut mengalami beberapa epoch melalui tahap training dan testing. Meskipun model dilatih, hasilnya menunjukkan bahwa performanya tidak optimal dengan akurasi yang rendah. Grafik loss dan akurasi dari hasil pelatihan dan pengujian juga menunjukkan tren yang tidak memuaskan, dengan kecenderungan overfitting, yang dapat dilihat dari perbedaan performa antara data pelatihan dan pengujian. Meskipun ada beberapa masalah, grafik ini memberikan informasi yang berharga untuk mengevaluasi dan memperbaiki model agar lebih efektif dan dapat generalisasi dengan baik.

- Model 1: TinyVGG dengan Augmentasi Data

Pada tahap ini, sebuah model TinyVGG dibuat dan dilatih menggunakan data augmentasi yang lebih kompleks dengan menggunakan TrivialAugmentWide. Transformasi ini

memperkenalkan variasi lebih besar dalam data pelatihan dengan mengubah secara acak citra seperti rotasi, skalasi, dan kecerahan. Model ini kemudian dilatih melalui beberapa epoch menggunakan data pelatihan yang telah diubah dan diuji pada data pengujian tanpa augmentasi. Namun, hasilnya menunjukkan bahwa performa model tidak mengalami peningkatan yang signifikan dibandingkan dengan model sebelumnya, bahkan menunjukkan tren yang serupa dengan akurasi yang rendah dan tidak ada perbaikan yang mencolok. Hasil ini menunjukkan bahwa mungkin diperlukan pendekatan lain dalam desain model atau teknik augmentasi untuk meningkatkan kinerja model dalam kasus ini.

- Membandingkan hasil model

```
import pandas as pd
model_0_df = pd.DataFrame(model_0_results)
model_1_df = pd.DataFrame(model_1_results)
model_0_df
```

	train_loss	train_acc	test_loss	test_acc
0	1.107836	0.257812	1.136208	0.260417
1	1.084645	0.425781	1.162168	0.197917
2	1.115317	0.292969	1.169556	0.197917
3	1.098814	0.414062	1.134430	0.197917
4	1.099029	0.292969	1.143381	0.197917

Kode tersebut menghasilkan dua DataFrames (`model_0_df` dan `model_1_df`) yang mewakili hasil pelatihan model `model_0` dan `model_1`. Setiap DataFrame berisi nilai kerugian (loss) dan akurasi untuk setiap epoch pada tahap pelatihan dan pengujian. Hasil dalam kolom tersebut mencerminkan performa model pada setiap langkah pelatihan dan pengujian, memberikan informasi tentang sejauh mana model dapat menyesuaikan diri dengan data pelatihan dan seberapa baik generalisasi pada data pengujian.

-