

REPORT OF PyTorch – FINAL EXAM



Learn PyTorch for Deep Learning: Zero to Mastery

Oleh :

Muhammad Yuzzaf Ibrahim Azzumarafi/1103200082

**PRODI S1 TEKNIK KOMPUTER
FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
BANDUNG
2024**

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	6
CHAPTER 00	8
0.1 <i>PYTORCH</i>	8
0.1.1 <i>Importing PyTorch</i>	10
0.2 <i>TENSOR</i>	11
0.3 <i>Scalar</i>	13
0.5 <i>Matrix</i>	14
0.6 <i>Perbandingan</i>	15
0.7 <i>AGGREGATION</i>	15
0.8 <i>REPRODUCIBILITY</i>	16
CHAPTER 01 PYTORCH WORKFLOW FUNDAMENTALS	18
1.1 <i>PYTORCH WORKFLOW</i>	18
1.2 <i>DATA (PERSIAPAN DAN PEMUATAN)</i>	19
1.3 <i>MEMBANGUN MODEL</i>	21
1.4 <i>PELATIHAN MODEL</i>	22
1.4.1 <i>Loss Function</i>	22
1.4.2 <i>Optimizer</i>	22
1.4.3 <i>Loop Training</i>	22
1.5 <i>MEMBUAT PREDIKSI DENGAN MODEL PYTORCH TERLATIH (INFERENSI)</i>	23
1.6 <i>MENYIMPAN DAN MEMUAT MODEL PYTORCH</i>	24
CHAPTER 02 PYTORCH NEURAL NETWORK CLASSIFICATION	25
2.1 <i>MAKE A BINARY CLASSIFICATION DATASET</i>	25
2.2 <i>EXERCISES 2: BUILD A MODEL</i>	25
2.3 <i>EXERCISE 3: SETUP LOSS FUNCTION AND OPTIMIZER</i>	26
2.4 <i>TRAINING LOOP</i>	27
2.5 <i>EXERCISES 5: MAKE PREDICTION AND PLOT DECISION BOUNDARY</i>	28
2.6 <i>REPLICATE THE TANH ACTIVATION FUNCTION</i>	29
2.7 <i>EXERCISE 7: CREATE A MULTI-CLASS DATASET AND TRAIN A MODEL</i>	29
CHAPTER 03 PYTORCH COMPUTER VISION.....	31
3.1 <i>COMPUTER VISION LIBRARIES IN PYTORCH</i>	31

3.2	GETTING DATASET	31
3.2.1	Input and output shapes of a computer vision model.....	31
3.2.2	Visualizing our data	32
3.3	PREPARE DATA LOADER.....	34
3.4	MODEL 0: BUILD A BASELINE MODEL	35
3.5	PEMBUATAN FUNGSI UNTUK MENGUKUR WAKTU EKSPERIMEN	37
3.6	PEMBUATAN LOOP PELATIHAN DAN PELATIHAN MODEL PADA BATCH DATA	37
3.7	MEMBUAT PREDIKSI DAN MENDAPATKAN HASIL MODEL 0	37
3.8	MENYIAPKAN KODE AGEN-LANGKA (DEVICE AGNOSTIC)	38
3.9	MODEL 1: MEMBANGUN MODEL LEBIH BAIK DENGAN NON-LINEARITAS.....	38
3.10	MODEL 2 - MEMBANGUN CONVOLUTIONAL NEURAL NETWORK (CNN).....	38
3.11	MEMBANDINGKAN HASIL MODEL DAN WAKTU PELATIHAN	38
3.12	MEMBUAT DAN MENILAI PREDIKSI ACAK DENGAN MODEL TERBAIK	38
3.13	MEMBUAT MATRIKS KONFUSI UNTUK EVALUASI LEBIH LANJUT.....	39
3.14	MENYIMPAN DAN MEMUAT MODEL TERBAIK.....	39
CHAPTER 04 PYTORCH CUSTOM DATASETS		40
4.1	GET DATA	40
4.2	DATA PREPARATION	40
4.3	TRANSFORMING DATA	41
4.4	MEMUAT DATA GAMBAR MENGGUNAKAN IMAGE FOLDER.....	43
4.5	BENTUK LAIN DARI TRANSFORMASI (AUGMENTASI DATA)	43
4.6	MODEL 0: TINYVGG TANPA AUGMENTASI DATA	43
4.7	EXPLORING LOSS CURVES	43
4.6.1	Underfitting	43
4.6.2	Overfitting.....	43
4.6.3	Just Right.....	44
4.8	TINYVGG DENGAN AUGMENTASI DATA.....	44
4.9	COMPARE MODEL RESULTS	45
CHAPTER 05 PYTORCH GOING MODULAR		46
5.1	GOING MODULAR	46
5.2	CELL MODE VS. SCRIPT MODE	46
5.3	PRO DAN KONTRA NOTEBOOK VS SKRIP PYTHON	46
5.4	GET DATA.....	47

5.5	<i>CREATE DATASET AND DATA LOADER (DATA_SETUP.PY)</i>	47
5.6	<i>MEMBUAT MODEL (MODEL_BUILDER.PY)</i>	47
5.7	<i>MEMBUAT DAN FUNGSI DAN MENGGABUNGKANNYA TRAIN_STEP() TEST_STEP() TRAIN()</i>	47
5.8	<i>MEMBUAT FUNGSI UNTUK MENYIMPAN MODEL (UTILS.PY)</i>	47
5.9	<i>LATIH, EVALUASI, DAN SIMPAN MODEL (TRAIN.PY)</i>	48
CHAPTER 06 TRANSFER LEARNING PYTORCH		49
6.1	<i>TRANSFER LEARNING</i>	49
6.2	<i>DAPATKAN DATA</i>	50
6.3	<i>BUAT DATASET DAN DATA LOADER</i>	50
6.4	<i>MENDAPATKAN MODEL YANG SUDAH DIPRETRAINING</i>	50
6.5	<i>MEMBUAT TRANSFORMASI UNTUK (PEMBUATAN MANUAL) TORCHVISION.MODELS</i>	51
6.6	<i>MEMBUAT TRANSFORMASI UNTUK (PEMBUATAN OTOMATIS) TORCHVISION.MODELS</i>	52
6.7	<i>TORCHINFO.SUMMARY()</i>	53
6.8	<i>MELATIH MODEL</i>	53
6.9	<i>EVALUASI MODEL DENGAN PLOTTING KURVA LOSS</i>	54
6.10	<i>PREDIKSI PADA GAMBAR DARI DATASET UJI</i>	54
CHAPTER 07 PYTORCH EXPERIMENT TRACKING		56
7.1	<i>DAPATKAN DATA</i>	56
7.2	<i>BUAT DATASET DAN DATA LOADER</i>	56
7.3	<i>DAPATKAN DAN SESUAIKAN MODEL YANG TELAH DILATIH SEBELUMNYA</i>	57
7.3.1	<i>Membuat DataLoader menggunakan transformasi yang dibuat secara manual</i>	57
7.3.2	<i>Membuat DataLoader menggunakan transformasi yang dibuat secara otomatis</i>	58
7.4	<i>TRAIN MODEL AND TRACK RESULTS</i>	59
7.5	<i>LIHAT HASIL MODEL DI TENSORBOARD</i>	60
7.6	<i>MEMBUAT FUNGSI PEMBANTU UNTUK MELACAK EKSPERIMEN</i>	60
7.7	<i>MUAT DALAM MODEL TERBAIK DAN BUAT PREDIKSI DENGANNYA</i>	60
CHAPTER 08 PYTORCH PAPER REPLICATING		61
8.1	<i>DAPATKAN DATA</i>	61
8.2	<i>BUAT DATASET DAN DATA LOADER</i>	61
8.3	<i>MENGHITUNG BENTUK INPUT DAN OUTPUT PENYEMATAN PATCH DENGAN TANGAN</i>	61

8.4	MENGUBAH SATU GAMBAR MENJADI TAMBALAN.....	62
8.5	MEMBUAT ENCODER TRANSFORMER.....	64
8.5.1	Membuat Transformer Encoder Block:.....	64
8.5.2	Membuat Transformer Encoder dengan Lapisan PyTorch:.....	64
8.6	MENYATUKAN SEMUANYA UNTUK MEMBUAT ViT.....	64
8.7	MENYIAPKAN KODE PELATIHAN UNTUK MODEL ViT.....	66
8.8	MENGGUNAKAN ViT TERLATIH DARI TORCHVISION.MODELS.....	66
8.9	BUAT PREDIKSI PADA GAMBAR KHUSUS	66
CHAPTER 09 PYTORCH MODEL DEPLOYMENT		67
9.1	MENDAPATKAN PENGATURAN	67
9.2	DAPATKAN DATA	67
9.3	MEMBUAT EKSTRAKTOR FITUR EffNetB2	67
9.4	MEMBUAT EKSTRAKTOR FITUR ViT	69
9.5	MEMBUAT PREDIKSI DENGAN MODEL TERLATIH KAMI DAN MENGATUR WAKTUNYA ...	70
9.5.1	Fungsi pred_and_store().....	70
9.5.1	Prediksi dengan Model EffNetB2.....	70
9.5.1	Prediksi dengan Model ViT.....	71
9.6	MEMBANDINGKAN HASIL MODEL, WAKTU PREDIKSI DAN UKURAN.....	71
9.7	MENGHIDUPKAN FOODVISION MINI DENGAN MEMBUAT DEMO GRADIO.....	72
9.7.1	Menghidupkan FoodVision Mini dengan Membuat Demo Gradio.....	72
9.8	MENGUBAH DEMO FOODVISION MINI GRADIO KAMI MENJADI APLIKASI YANG DAPAT DIGUNAKAN.....	73
9.9	MENYEBARKAN DEMO GRADIO KE HUGGINGFACE SPACES	74

DAFTAR GAMBAR

Gambar 1: Prompt untuk Import PyTorch & Versinya.....	11
Gambar 2: Tensor Shape.....	12
Gambar 3: Tensor 1D & 2D.....	12
Gambar 4: Scalar.....	13
Gambar 5: Vector.....	14
Gambar 6: Matrix.....	14
Gambar 7: Scalar, Vector, Matrix, dan Tensor	15
Gambar 8: Aggregation.....	16
Gambar 9: Reproducibility.....	17
Gambar 10: PyTorch Workflow	18
Gambar 11: Make Data	19
Gambar 12: Create Train and Split Test	20
Gambar 13: Prompt Prediction Visualisasi	20
Gambar 14: Hasil Prediksi	21
Gambar 15: Torch.save	24
Gambar 16: Check The Saved File Path.....	24
Gambar 17: Load Model	24
Gambar 18: Binary Classification Dataset.....	25
Gambar 19: Build a model	26
Gambar 20: Loss Function dan Optimizer.....	26
Gambar 21: Training Loop.....	27
Gambar 22: Prompt prediction and plot decision boundary	28
Gambar 23: Visualisasi Plot dan Prediksi.....	29
Gambar 24: Tanh Function	29
Gambar 25: Membuat multi-class dan train model.....	30
Gambar 26: Visualisasi Spiral.....	30
Gambar 27: Import Library PyTorch & torchvision.....	31
Gambar 28: Mengambil dataset	31
Gambar 29: Mencari shape, class, dan samples.....	32
Gambar 30: Visualisasi data Shoes.....	32
Gambar 31: Visualisasi data Shoes tema grey	33
Gambar 32: Visualisasi data lebih banyak.....	33
Gambar 33: Prepare DataLoader.....	34
Gambar 34: Pengecekan isi Training DataLoader	34
Gambar 35: Visualisasi Sample	35
Gambar 36: Baseline Model.....	36
Gambar 37: Loss Function dan Optimizer.....	37
Gambar 38: Visualisasi Penilaian data.....	39
Gambar 39: Dataset Pizza	40
Gambar 40: Data yang digunakan.....	40
Gambar 41: Direktori data	41
Gambar 42: Transform Compose.....	41
Gambar 43: Fungsi percobaan pada gambar.....	42
Gambar 44: Hasil Percobaan.....	42
Gambar 45: Hasil PKitas Dataframe.....	45
Gambar 46: Grafik Plot.....	45
Gambar 47: Prompt Transform Manually.....	51
Gambar 48: Contoh Penggunaan	53

Gambar 49: Hasil Uji Tiga Gambar	55
Gambar 50: DataLoader Transformasi secara manual.....	57
Gambar 51: DataLoader Transformasi secara otomatis.....	58
Gambar 52: Penyematan Patch	62
Gambar 53: Prompt Visualisasi beberapa tambalan di baris atas	63
Gambar 54: Prompt Hasil Visualisasi	63
Gambar 55: Sushi Patchified 16x16 Pixels	63
Gambar 56: Import Gradio dan Version Check	72

CHAPTER 00

PyTorch Fundamentals

0.1 PyTorch

PyTorch adalah sebuah framework deep learning open-source yang dikembangkan oleh grup penelitian AI di Facebook. Framework ini pertama kali diperkenalkan pada bulan Oktober 2016. Di bawah ini adalah sejarah singkat perkembangan PyTorch:

1. Versi Awal (2016)

- PyTorch awalnya dirilis oleh Facebook AI Research (FAIR) sebagai alternatif untuk framework deep learning lainnya, seperti TensorFlow.
- Kelebihan utama PyTorch pada saat itu adalah fleksibilitasnya yang tinggi dan lebih mudah untuk digunakan, terutama dalam proses pembuatan model dan pemahaman konsep.

2. Dukungan Komunitas (2017)

- Seiring berjalannya waktu, PyTorch mendapatkan dukungan yang kuat dari komunitas peneliti dan praktisi machine learning.
- Komunitas PyTorch terus berkembang dan berkontribusi terhadap pengembangan framework ini.

3. Dynamic Computational Graph (2017)

- Salah satu fitur utama PyTorch adalah komputasi grafis dinamis (dynamic computational graph), yang memungkinkan perancangan model dengan cara yang lebih dinamis dan intuitif.

4. Versi 1.0 dan Eager Execution (2018)

- Pada tahun 2018, PyTorch merilis versi 1.0 dengan peningkatan signifikan dalam performa dan fungsionalitas.
- Eager execution, yang memungkinkan eksekusi operasi secara langsung saat kode dijalankan, menjadi fitur utama dalam versi ini.

5. TorchScript dan ONNX (2018)

- TorchScript diperkenalkan untuk mengkonversi model PyTorch ke format yang dapat dieksekusi di lingkungan yang tidak memiliki Python.
- PyTorch juga mendukung format Open Neural Network Exchange (ONNX), memungkinkan pertukaran model dengan framework deep learning lainnya.

6. LibTorch (2019)

- LibTorch adalah pustaka C++ resmi untuk PyTorch, memungkinkan integrasi PyTorch ke dalam aplikasi C++ tanpa memerlukan Python.

7. Flux Versi 1.0 (2020)

- PyTorch 1.4 memperkenalkan Flux, sebuah API untuk machine learning yang terintegrasi dengan PyTorch.
- Flux menyediakan antarmuka yang bersih dan deklaratif untuk pemodelan dan pelatihan model.

8. Peningkatan Kompatibilitas dengan TensorFlow (2021)

- PyTorch dan TensorFlow terus meningkatkan kompatibilitas mereka, memungkinkan pengguna untuk mentransfer model antara kedua framework dengan lebih mudah.

PyTorch terus berkembang dan menjadi salah satu framework deep learning yang sangat populer di kalangan peneliti dan praktisi. Fleksibilitasnya, dokumen yang baik, dan komunitas yang aktif adalah beberapa faktor yang menyumbang keberhasilannya. PyTorch digunakan oleh berbagai organisasi, peneliti, dan praktisi di seluruh dunia untuk pengembangan dan implementasi model deep learning. Berikut adalah beberapa contoh dari berbagai sektor yang menggunakan PyTorch:

1. Industri Teknologi

- Facebook: PyTorch dikembangkan oleh grup penelitian AI di Facebook, dan masih digunakan secara intensif di sana.
- Uber: Uber menggunakan PyTorch untuk pengembangan model machine learning di berbagai bidang, seperti peramalan permintaan dan deteksi anomali.

2. Industri Ritel dan E-Commerce

- Amazon: PyTorch digunakan dalam beberapa tim riset dan pengembangan Amazon.
- Alibaba: Alibaba Cloud juga mendukung PyTorch di platformnya.

3. Industri Otomotif

- Tesla: Tesla menggunakan PyTorch untuk pengembangan model di bidang visi komputer dan kecerdasan buatan.

4. Akademis dan RiUniversitas dan Lembaga Riset:

- Banyak peneliti di universitas dan lembaga penelitian menggunakan PyTorch untuk penelitian dan pengembangan model deep learning.

5. Start-up dan Perusahaan Kecil

- Banyak start-up dan perusahaan kecil memilih PyTorch karena keterbacaan kodenya dan kemudahan dalam mengembangkan model deep learning.

6. Industri Kesehatan

- Oxford University Hospitals NHS Foundation Trust: PyTorch digunakan dalam penelitian medis, termasuk pengembangan model untuk deteksi dan diagnostik.
7. Komunitas Open Source
- PyTorch memiliki komunitas yang aktif di GitHub, dengan kontributor dari berbagai belahan dunia.

0.1.1 Importing PyTorch

Pada Python, kata kunci import digunakan untuk memuat modul atau pustaka tertentu ke dalam skrip atau program Python. Ketika menggunakan import torch di dalam skrip atau notebook Python, itu berarti memuat pustaka PyTorch ke dalam lingkungan kerja .

Fungsi dari import torch adalah membawa ke dalam ruang kerja Python berbagai fungsi, kelas, dan alat yang disediakan oleh PyTorch. Beberapa fungsi dan konsep dasar yang disediakan oleh PyTorch melalui import ini meliputi:

1. Tensor

PyTorch menyediakan tipe data utama yang disebut "tensor". Tensor adalah representasi dasar untuk data dalam PyTorch dan digunakan untuk menyimpan dan memanipulasi data numerik, serupa dengan array multidimensi.

2. Operasi Tensor

PyTorch menyediakan berbagai operasi matematika dan fungsi untuk bekerja dengan tensor, seperti penjumlahan, perkalian, dan fungsi aktivasi.

3. Autograd

PyTorch memiliki sistem autograd yang memungkinkan otomatis perhitungan gradien. Ini sangat penting dalam pelatihan model deep learning.

4. Neural Networks

PyTorch menyediakan modul torch.nn yang memungkinkan definisi dan pelatihan jaringan saraf tiruan (neural networks).

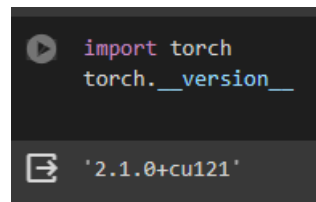
5. Optimas

PyTorch menyediakan berbagai algoritma optimasi yang digunakan untuk menyesuaikan parameter model selama proses pelatihan.

6. GPU Acceleration

PyTorch memiliki dukungan GPU bawaan yang memungkinkan penggunaan GPU untuk operasi tensor, yang sangat meningkatkan kinerja dalam pelatihan model deep learning.

Prompt untuk Import PyTorch dan mengetahui versi yang digunakan:



```
import torch
torch.__version__
```

```
'2.1.0+cu121'
```

Gambar 1: Prompt untuk Import PyTorch & Versinya

0.2 Tensor

Tensor adalah struktur data dasar yang digunakan untuk menyimpan dan memanipulasi data. Tensor mirip dengan matriks atau array multidimensi, tetapi dengan fleksibilitas tambahan. Tensors dapat memiliki dimensi (rank) yang bervariasi, yang memungkinkan mereka mewakili berbagai jenis data, mulai dari skalar (0D) hingga vektor (1D), matriks (2D), hingga tensor dengan dimensi yang lebih tinggi.

Dalam PyTorch, tensor dapat digunakan untuk menyimpan data numerik, dan mereka mendukung operasi matematika yang efisien. Misalnya, tensor dapat digunakan untuk merepresentasikan input dan output dalam jaringan saraf, dan operasi tensor dapat diterapkan untuk melakukan perhitungan yang melibatkan transformasi linier, aktivasi, dan fungsi lainnya.

Berikut adalah beberapa contoh operasi yang dapat dilakukan dengan tensor di PyTorch:

- Penjumlahan dan Pengurangan: Menambah atau mengurangi tensor.
- Perkalian dan Pembagian: Melakukan operasi perkalian atau pembagian dengan tensor.
- Indeks dan Irisan: Mengakses atau memperoleh subset dari tensor menggunakan indeks atau irisan.
- Operasi Matriks: Transposisi, perkalian matriks, dan operasi matriks lainnya.

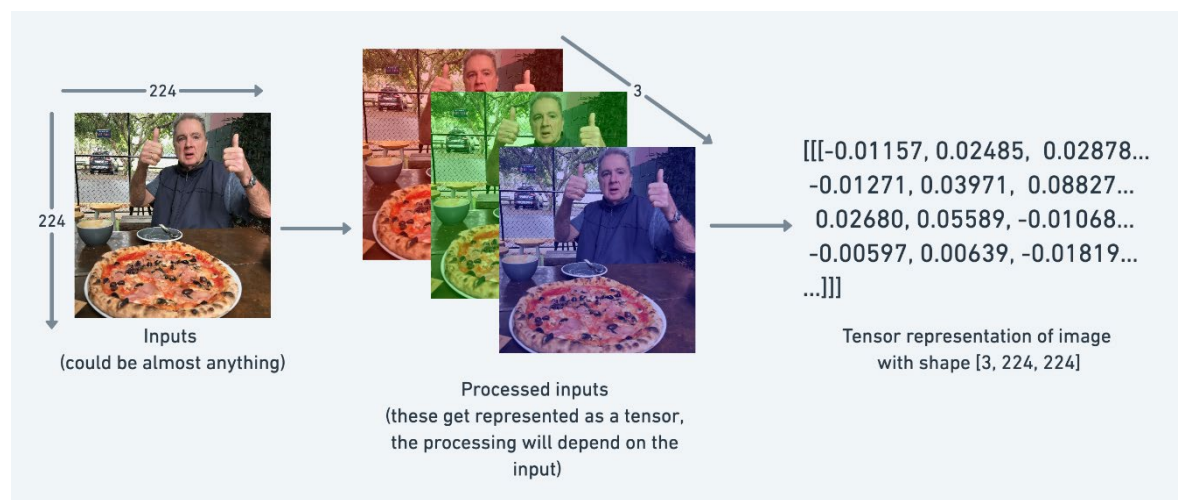
Tensor memberikan abstraksi yang kuat untuk bekerja dengan data numerik dalam konteks machine learning dan deep learning, dan PyTorch memiliki fungsi-fungsi bawaan yang memungkinkan manipulasi tensor dengan cepat dan efisien.

Berikut adalah beberapa karakteristik utama dari tensor:

- Dimensi (Rank)
Tensor dapat memiliki dimensi yang berbeda. Tensor dengan dimensi 0 disebut "skalar", tensor dengan dimensi 1 disebut "vektor", tensor dengan dimensi 2 disebut "matriks", dan seterusnya. Misalnya, tensor 1D dapat dianggap sebagai vektor, tensor 2D sebagai matriks, dan seterusnya.
- Shape (Bentuk)
Shape dari sebuah tensor menyatakan jumlah elemen dalam setiap dimensinya. Sebagai contoh, tensor 1D dengan shape (5,) memiliki 5 elemen, tensor 2D dengan shape (3, 4) memiliki 3 baris dan 4 kolom, dan seterusnya.

- Elemen
Elemen-elemen tensor dapat berupa nilai skalar (misalnya, angka tunggal) atau struktur yang lebih kompleks.
- Tipe Data
Tensor memiliki tipe data yang mendefinisikan jenis nilai yang dapat mereka simpan (seperti float, int, double, dll.).

Misalnya, Kita dapat mewakili gambar sebagai tensor dengan bentuk yang berarti, karena pada gambar memiliki saluran warna (merah, hijau, biru), tinggi piksel dan lebar piksel. $[3, 224, 224]$ [colour_channels, height, width] 3 224 224



Gambar 2: Tensor Shape

Dalam *tensor-speak* (bahasa yang digunakan untuk menggambarkan tensor), tensor akan memiliki tiga dimensi, satu untuk dan colour_channels height width.

```
import torch

# Membuat tensor
tensor_1d = torch.tensor([1, 2, 3, 4, 5])
tensor_2d = torch.tensor([[1, 2, 3], [4, 5, 6]])

# Cetak nilai dan bentuk tensor
print(tensor_1d)
print(tensor_2d)
print(tensor_1d.shape)
print(tensor_2d.shape)

tensor([1, 2, 3, 4, 5])
tensor([[1, 2, 3],
        [4, 5, 6]])
torch.Size([5])
torch.Size([2, 3])
```

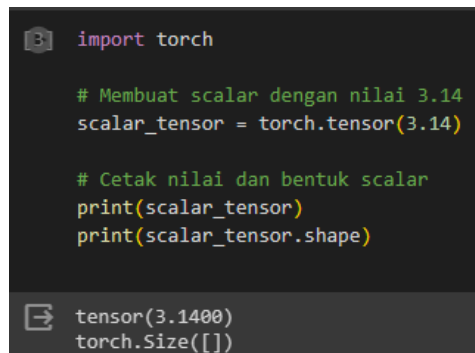
Gambar 3: Tensor 1D & 2D

Hasil tersebut mencakup dua tensor PyTorch dan bentuk (shape) masing-masing tensor.

1. `tensor([1, 2, 3, 4, 5])`:
 - Ini adalah tensor 1D atau vektor dengan panjang 5.
 - Berisi nilai-nilai [1, 2, 3, 4, 5].
2. `tensor([[1, 2, 3], [4, 5, 6]])`:
 - Ini adalah tensor 2D atau matriks dengan dimensi 2x3.
 - Matriks ini memiliki dua baris dan tiga kolom, berisi nilai-nilai: [[1, 2, 3], [4, 5, 6]]
3. `torch.Size([5])`:
 - Ini menunjukkan bentuk (shape) dari tensor pertama, yaitu [5].
 - Tensor ini adalah vektor dengan panjang 5.
4. `torch.Size([2, 3])`:
 - Ini menunjukkan bentuk (shape) dari tensor kedua, yaitu [2, 3].
 - Tensor ini adalah matriks dengan dua baris dan tiga kolom.

0.3 Scalar

Dalam PyTorch, istilah "scalar" merujuk pada tensor nol dimensi, atau dengan kata lain, sebuah bilangan tunggal. Sebuah scalar di PyTorch dapat dianggap sebagai tensor dengan bentuk "()" atau dengan bentuk "(1,)". Meskipun bentuknya nol dimensi, PyTorch memperlakukannya sebagai tensor.



```

import torch

# Membuat scalar dengan nilai 3.14
scalar_tensor = torch.tensor(3.14)

# Cetak nilai dan bentuk scalar
print(scalar_tensor)
print(scalar_tensor.shape)

```

Output:

```

tensor(3.1400)
torch.Size([])

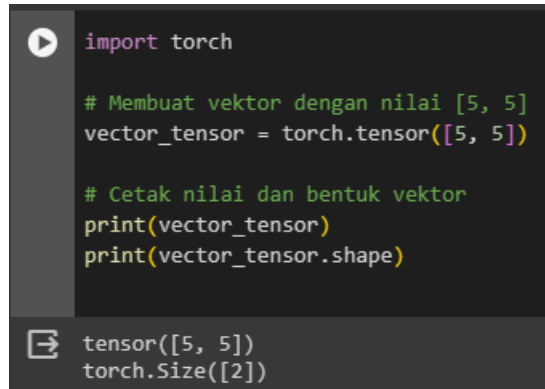
```

Gambar 4: Scalar

Ketika mencetak entuknya (`torch.Size([])`), ini menunjukkan bahwa tensor tersebut adalah scalar atau tensor nol dimensi. Scalar biasanya digunakan dalam operasi dan perhitungan di dalam model deep learning sebagai bobot atau konstanta skalar. Misalnya, dalam suatu persamaan matematis dalam model neural network, dapat memiliki skalar sebagai parameter yang mempengaruhi hasil perhitungan.

0.4 Vector

Dalam PyTorch, "vector" merujuk pada tensor satu dimensi. Secara matematis, vektor adalah suatu himpunan bilangan yang disusun dalam satu dimensi, dan dalam konteks PyTorch, vektor direpresentasikan sebagai tensor dengan bentuk (n,), di mana n adalah panjang vektor.



```
import torch

# Membuat vektor dengan nilai [5, 5]
vector_tensor = torch.tensor([5, 5])

# Cetak nilai dan bentuk vektor
print(vector_tensor)
print(vector_tensor.shape)
```

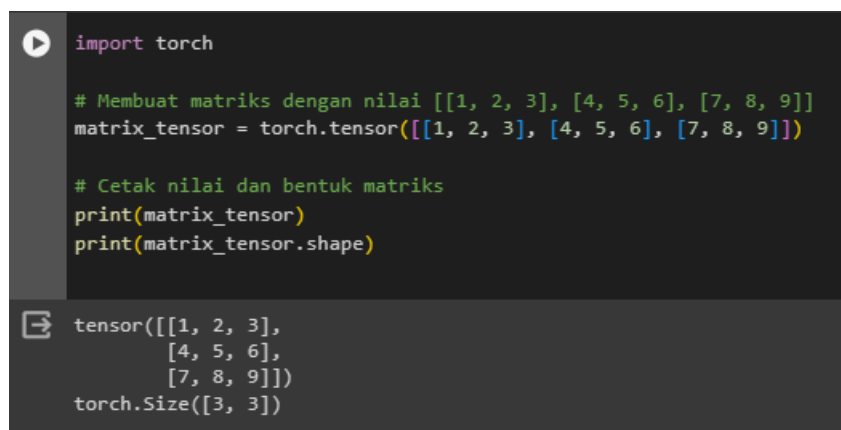
```
tensor([5, 5])
torch.Size([2])
```

Gambar 5: Vector

Ketika mencetak bentuknya (`torch.Size([5])`), ini menunjukkan bahwa tensor tersebut adalah vektor dengan panjang 5. Vektor sering digunakan dalam berbagai konteks dalam machine learning dan deep learning, terutama dalam konteks parameter model, data input, atau output dari model. Operasi matematika yang umum dilakukan pada vektor melibatkan penjumlahan, perkalian dengan skalar, dan berbagai operasi lainnya yang dapat diimplementasikan menggunakan PyTorch. Vektor juga merupakan bagian integral dari model neural networks, di mana bobot dan bias sering kali diwakili sebagai vektor.

0.5 Matrix

Dalam PyTorch, istilah "matrix" merujuk pada tensor dua dimensi. Secara matematis, matriks adalah suatu himpunan bilangan yang disusun dalam dua dimensi (baris dan kolom). Dalam konteks PyTorch, matriks direpresentasikan sebagai tensor dengan bentuk (m, n), di mana m adalah jumlah baris dan n adalah jumlah kolom.



```
import torch

# Membuat matriks dengan nilai [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix_tensor = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Cetak nilai dan bentuk matriks
print(matrix_tensor)
print(matrix_tensor.shape)
```

```
tensor([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
torch.Size([3, 3])
```

Gambar 6: Matrix

Ketika mencetak bentuknya (`torch.Size([3, 3])`), ini menunjukkan bahwa tensor tersebut adalah matriks dengan 3 baris dan 3 kolom. Matriks sering digunakan dalam operasi aljabar linear, dan dalam konteks machine learning, matriks dapat mewakili berbagai aspek data seperti matriks fitur dalam dataset atau bobot dalam suatu model.

0.6 Perbandingan

Scalar

7

Vector

$$\begin{bmatrix} 7 \\ 4 \end{bmatrix}$$
 or

$$\begin{bmatrix} 7 & 4 \end{bmatrix}$$

Matrix

$$\begin{bmatrix} 7 & 10 \\ 4 & 3 \\ 5 & 1 \end{bmatrix}$$

Tensor

$$\begin{bmatrix} \begin{bmatrix} 7 & 4 \end{bmatrix} & \begin{bmatrix} 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 9 \end{bmatrix} & \begin{bmatrix} 2 & 3 \end{bmatrix} \\ \begin{bmatrix} 5 & 6 \end{bmatrix} & \begin{bmatrix} 8 & 8 \end{bmatrix} \end{bmatrix}$$

Gambar 7: Scalar, Vector, Matrix, dan Tensor

0.7 Aggregation

Dalam konteks PyTorch atau dalam dunia pengolahan data dan machine learning, istilah "aggregation" merujuk pada proses menggabungkan sejumlah nilai menjadi satu nilai ringkasan. Aggregation biasanya melibatkan operasi matematika, seperti penjumlahan, perataan, atau menghitung statistik tertentu di sepanjang suatu dimensi tensor.

Beberapa contoh operasi aggregation yang umum dilakukan pada tensor PyTorch melibatkan fungsi-fungsi seperti:

- `torch.sum()`: Menghitung jumlah elemen dalam tensor.
- `torch.mean()`: Menghitung rata-rata elemen dalam tensor.
- `torch.max()`: Menemukan nilai maksimum dalam tensor.
- `torch.min()`: Menemukan nilai minimum dalam tensor.
- `torch.prod()`: Menghitung hasil perkalian semua elemen dalam tensor.
- `torch.median()`: Menghitung nilai median dalam tensor.

Contoh aggregation (Max, Min, Mean, dan Sum):

```
[4] # Create a tensor
x = torch.arange(0, 100, 10)
x

tensor([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])

[5] print(f"Minimum: {x.min()}")
    print(f"Maximum: {x.max()}")
    # print(f"Mean: {x.mean()}") # this will error
    print(f"Mean: {x.type(torch.float32).mean()}") # won't work without float datatype
    print(f"Sum: {x.sum()}")
```

Minimum: 0
Maximum: 90
Mean: 45.0
Sum: 450

Gambar 8: Aggregation

Ini berarti nilai Max, Min, Mean, dan Sum dari semua elemen dalam tensor adalah Max(90), Min(0), Mean(45.0) dan Sum(450).

0.8 Reproducibility

Reproducibility (reproducibilitas) dalam konteks PyTorch merujuk pada kemampuan untuk mendapatkan hasil yang sama atau setidaknya serupa ketika menjalankan suatu eksperimen, pelatihan model, atau proses komputasi lainnya, pada kondisi yang sama. Artinya, jika menjalankan suatu kode di waktu yang berbeda atau di mesin yang berbeda, namun dengan pengaturan awal yang identik, Kita dapat menghasilkan hasil yang serupa atau identik.

Reproducibility sangat penting dalam penelitian ilmiah dan pengembangan model deep learning karena memungkinkan orang lain untuk memvalidasi dan memverifikasi temuan atau hasil eksperimen yang dilakukan. Beberapa faktor yang dapat mempengaruhi reproducibility di PyTorch atau dalam machine learning secara umum melibatkan


```

import torch

# Create two random tensors
random_tensor_A = torch.rand(3, 4)
random_tensor_B = torch.rand(3, 4)

print(f"Tensor A:\n{random_tensor_A}\n")
print(f"Tensor B:\n{random_tensor_B}\n")
print(f"Does Tensor A equal Tensor B? (anywhere)")
random_tensor_A == random_tensor_B

```

```

Tensor A:
tensor([[0.8901, 0.7292, 0.2205, 0.6484],
        [0.6457, 0.3309, 0.6907, 0.0014],
        [0.8163, 0.9523, 0.1795, 0.9190]])

Tensor B:
tensor([[0.7932, 0.0021, 0.5525, 0.6782],
        [0.2002, 0.2177, 0.9776, 0.9760],
        [0.5437, 0.5737, 0.8399, 0.6031]])

Does Tensor A equal Tensor B? (anywhere)
tensor([[False, False, False, False],
        [False, False, False, False],
        [False, False, False, False]])

```

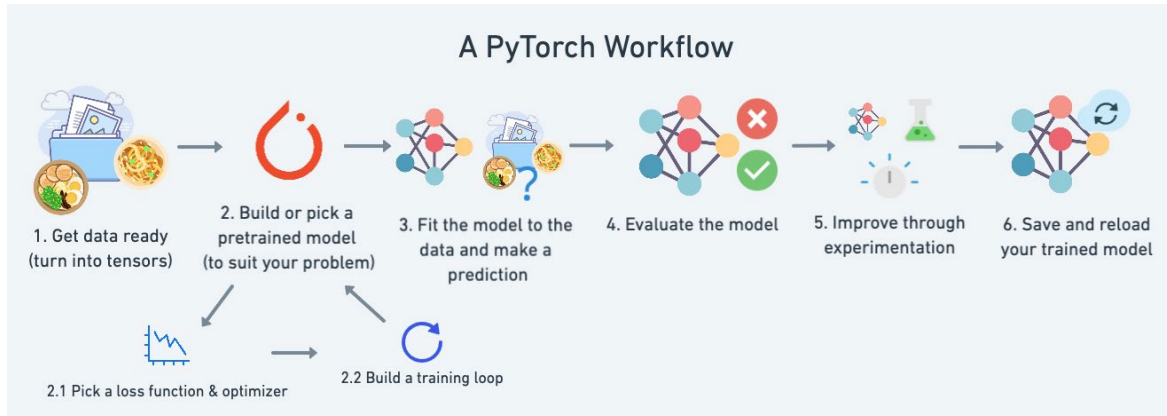
Gambar 9: Reproducibility

Hasil ini menunjukkan bahwa pada setiap posisi elemen (anywhere) dalam Tensor A dan Tensor B, elemennya tidak sama. Jika nilai pada posisi tertentu sama, maka nilai pada posisi tersebut dalam tensor hasil akan menjadi True, sedangkan jika tidak sama, nilai akan menjadi False. Dalam hal ini, hasilnya menunjukkan bahwa tidak ada satu pun posisi elemen di mana Tensor A sama dengan Tensor B.

CHAPTER 01

PyTorch Workflow Fundamentals

1.1 PyTorch Workflow



Gambar 10: PyTorch Workflow

Alur kerja PyTorch yang ada di atas adalah langkah-langkah umum yang biasanya diikuti saat mengembangkan model machine learning menggunakan PyTorch. Berikut setiap langkahnya:

1. **Persiapkan Data**
 - Mengambil atau mempersiapkan data yang akan digunakan untuk melatih dan menguji model.
 - Mengonversi data ke dalam bentuk tensor, yang merupakan struktur data dasar dalam PyTorch.
2. **Pilih atau Bangun Model Terlatih**
 - **Pilih Fungsi Kerugian & Optimizer**
 - a. Memilih fungsi kerugian yang akan dioptimalkan selama pelatihan. Fungsi kerugian mengukur seberapa baik model memprediksi label yang benar.
 - b. Memilih optimizer yang akan memperbarui parameter model untuk mengurangi kerugian selama pelatihan.
 - **Bangun Loop Pelatihan**
 - a. Membangun loop pelatihan yang berisi langkah-langkah untuk melakukan propagasi maju, menghitung kerugian, melakukan propagasi mundur, dan memperbarui parameter model.
3. **Sesuaikan Model dengan Data & Buat Prediksi**
 - Melatih model pada data pelatihan menggunakan loop pelatihan.
 - Menggunakan model yang telah dilatih untuk membuat prediksi pada data uji atau data baru.
4. **Evaluasi Model**
 - Mengukur kinerja model dengan menggunakan metrik evaluasi yang sesuai untuk tugas yang sedang dijalankan (misalnya, akurasi, presisi, recall, dll.).
5. **Tingkatkan melalui Eksperimen**
 - Jika hasil evaluasi tidak memuaskan, eksperimen lebih lanjut dengan memodifikasi arsitektur model, mengubah hyperparameter, atau melakukan perubahan lainnya.

6. Simpan dan Muat Ulang Model yang Dilatih

- Menyimpan model setelah dilatih untuk digunakan di masa mendatang tanpa perlu melatih ulang.
- Memuat kembali model yang telah disimpan untuk penggunaan lebih lanjut.

1.2 Data (persiapan dan pemuatan)

Data machine learning banyak sekali bisa berupa table angka, gambar, video, file audio, struktur protein, teks, dan lainnya.

Machine Learning adalah permainan dua bagian:

1. Ubah data Kita, apa pun itu, menjadi angka (representasi).
2. Pilih atau buat model untuk mempelajari representasi sebaik mungkin.

Di Modul ini akan menggunakan regresi linier untuk membuat data dengan parameter yang diketahui (hal-hal yang dapat dipelajari oleh model) dan kemudian akan menggunakan PyTorch untuk melihat apakah dapat membangun model untuk memperkirakan parameter ini menggunakan gradient descent.

```
[3] # Create *known* parameters
weight = 0.7
bias = 0.3

# Create data
start = 0
end = 1
step = 0.02
X = torch.arange(start, end, step).unsqueeze(dim=1)
y = weight * X + bias

X[:10], y[:10]

(tensor([[0.0000],
        [0.0200],
        [0.0400],
        [0.0600],
        [0.0800],
        [0.1000],
        [0.1200],
        [0.1400],
        [0.1600],
        [0.1800]]),
 tensor([[0.3000],
        [0.3140],
        [0.3280],
        [0.3420],
        [0.3560],
        [0.3700],
        [0.3840],
        [0.3980],
        [0.4120],
        [0.4260]]))
```

Gambar 11: Make Data

Lalu Pisahkan data menjadi set pelatihan dan tes:

```

# Create train/test split
train_split = int(0.8 * len(X)) # 80% of data used for training set, 20% for testing
X_train, y_train = X[:train_split], y[:train_split]
X_test, y_test = X[train_split:], y[train_split:]

len(X_train), len(y_train), len(X_test), len(y_test)

```

(40, 40, 10, 10)

Gambar 12: Create Train and Split Test

Disini mempunyai 40 sampel untuk pelatihan dan 10 sampel untuk pengujian. Selanjutnya buat visualisasi dari data yang sudah di pisahkan.

```

def plot_predictions(train_data=X_train,
                    train_labels=y_train,
                    test_data=X_test,
                    test_labels=y_test,
                    predictions=None):
    """
    Plots training data, test data and compares predictions.
    """
    plt.figure(figsize=(10, 7))

    # Plot training data in blue
    plt.scatter(train_data, train_labels, c="b", s=4, label="Training data")

    # Plot test data in green
    plt.scatter(test_data, test_labels, c="g", s=4, label="Testing data")

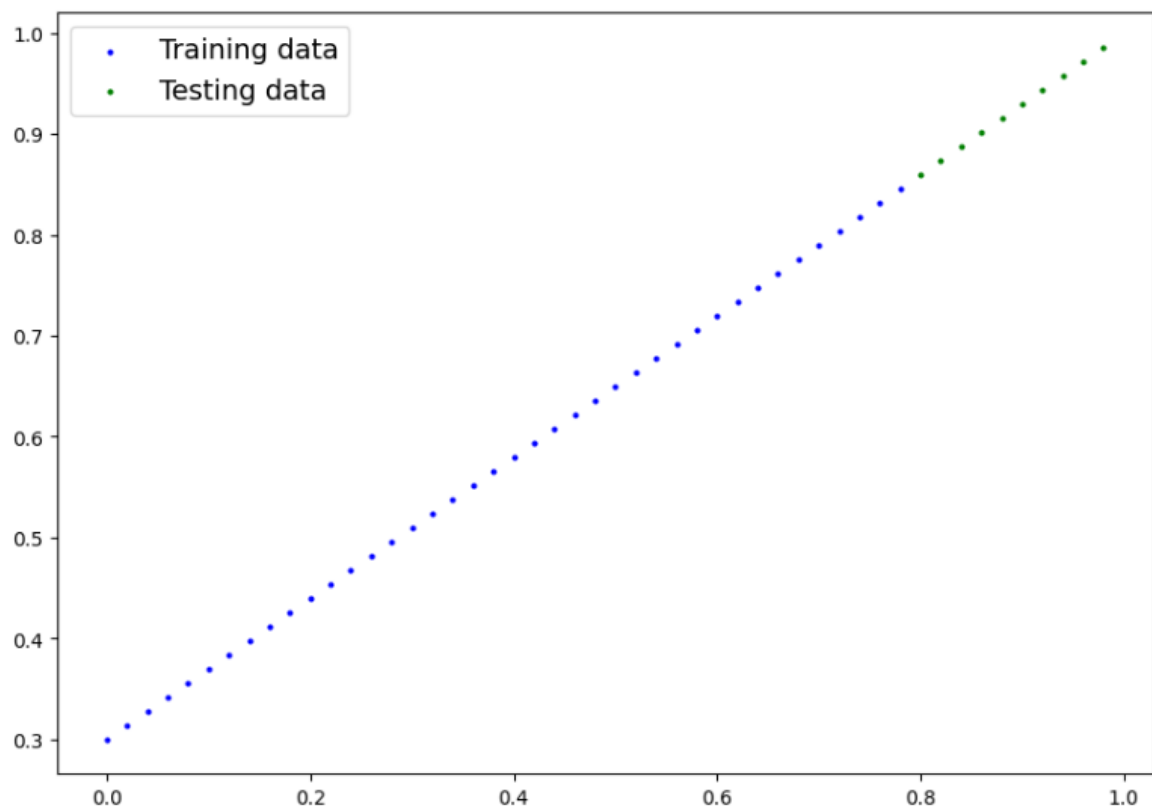
    if predictions is not None:
        # Plot the predictions in red (predictions were made on the test data)
        plt.scatter(test_data, predictions, c="r", s=4, label="Predictions")

    # Show the legend
    plt.legend(prop={"size": 14});

```

plot_predictions();

Gambar 13: Prompt Prediction Visualisasi



Gambar 14: Hasil Prediksi

Dan disini yang tadinya data berupa angka yang kita buat secara random menjadi data yang sudah di olah menjadi visualisasi garis lurus.

1.3 Membangun model

Hal-hal penting dalam membangun model PyTorch terdapat 4 modul PyTorch seperti `torch.nn`, `torch.optim`, `torch.utils.data.Dataset` dan `torch.utils.data.DataLoader`.

1. `torch.nn`

Berisi semua blok bangunan untuk grafik komputasi (pada dasarnya serangkaian perhitungan yang dieksekusi dengan cara tertentu).

2. `torch.nn.Parameter`

Menyimpan tensor yang dapat digunakan dengan `.data`. Jika gradien (digunakan untuk memperbarui parameter model melalui penurunan gradien) dihitung secara otomatis, ini sering disebut sebagai "autograd". `nn.Module.requires_grad=True`

3. `torch.nn.Module`

Kelas dasar untuk semua modul jaringan saraf, semua blok bangunan untuk jaringan saraf adalah subkelas. Jika Kita membangun jaringan saraf di PyTorch, model Kita harus subkelas. Membutuhkan metode yang diimplementasikan `nn.Module.forward()`

4. `torch.optim`

Berisi berbagai algoritma pengoptimalan (ini memberi tahu parameter model yang disimpan dalam cara terbaik untuk mengubah untuk meningkatkan penurunan gradien dan pada gilirannya mengurangi kerugian).nn.Parameter

5. def forward()

Semua subclass memerlukan metode, ini mendefinisikan perhitungan yang akan terjadi pada data yang diteruskan ke tertentu (misalnya rumus regresi linier di atas).nn.Moduleforward()nn.Module

1.4 Pelatihan Model

Disini akan membuat pelatihan model dengan beberapa fungsi yang akan digunakan seperti loss function, dan optimizer.

1.4.1 Loss Function

Mengukur seberapa salah prediksi model Kita (misalnya) dibandingkan dengan label kebenaran (misalnya) Turunkan lebih baik y_{preds} y_{test} . Nilai umum Berarti kesalahan absolut (MAE) untuk masalah regresi (torch.nn.L1Loss()). Entropi silang biner untuk masalah klasifikasi biner (torch.nn.BCELoss()).

1.4.2 Optimizer

Memberi tahu model cara memperbarui parameter internalnya untuk menurunkan kerugian dengan dengan sebaik-baiknya. Nilai umum Penurunan gradien stokastik (torch.optim.SGD()). Pengoptimal Adam (torch.optim.Adam()).

Ada beberapa nilai umum, yang diketahui bekerja dengan baik seperti SGD (stochastic gradient descent) atau Adam optimizer. Dan fungsi kerugian MAE (kesalahan absolut rata-rata) untuk masalah regresi (memprediksi angka) atau fungsi kerugian entropi silang biner untuk masalah klasifikasi (memprediksi satu hal atau lainnya).

1.4.3 Loop Training

Loop pelatihan (training loop) dalam konteks machine learning merujuk pada iterasi berulang yang dilakukan selama pelatihan model. Di dalam loop pelatihan, model diperbarui berdasarkan data pelatihan untuk mengurangi nilai kerugian atau fungsi objektif yang telah ditentukan.

Berikut adalah komponen utama dari sebuah loop pelatihan dalam PyTorch:

- Iterasi (Epoch): Setiap iterasi dari loop pelatihan dikenal sebagai "epoch". Satu epoch berarti model melihat dan memproses seluruh data pelatihan sekali.
- Propagasi Maju (Forward Propagation): Data pelatihan dimasukkan ke dalam model untuk membuat prediksi. Proses ini dikenal sebagai propagasi maju.

- **Perhitungan Kerugian (Loss Computation):** Prediksi model dibandingkan dengan label yang sebenarnya, dan fungsi kerugian dihitung. Fungsi kerugian mengukur seberapa besar perbedaan antara prediksi dan label sebenarnya.
- **Propagasi Mundur (Backward Propagation):** Gradien dari fungsi kerugian dihitung terhadap parameter model menggunakan propagasi mundur. Gradien ini menunjukkan arah dan seberapa besar setiap parameter harus diperbarui untuk mengurangi kerugian.
- **Optimasi (Optimization):** Parameter model diperbarui menggunakan optimizer. Optimizer mengikuti gradien dan memperbarui parameter model sesuai dengan aturan tertentu untuk mengurangi kerugian.
- **Evaluasi (Optional):** Secara opsional, model dapat dievaluasi pada data validasi selama atau setelah setiap epoch untuk mengukur kinerja model pada data yang tidak digunakan selama pelatihan.

Langkah-langkah di atas diulang berulang kali sepanjang epoch yang telah ditentukan atau hingga kriteria berhenti tertentu terpenuhi, seperti jumlah epoch maksimum atau konvergensi kerugian yang memadai.

1.5 Membuat prediksi dengan model PyTorch terlatih (inferensi)

Membuat prediksi dengan model PyTorch terlatih melibatkan proses inferensi, yaitu penggunaan model untuk membuat prediksi atau hasil berdasarkan data baru atau data yang tidak digunakan selama pelatihan. Berikut adalah langkah-langkah umum untuk melakukan inferensi dengan model PyTorch:

1. **Persiapkan Data Input:**

Pastikan bahwa data input yang akan digunakan untuk membuat prediksi telah dipersiapkan dan diubah ke dalam bentuk tensor yang sesuai dengan format yang diminta oleh model.

2. **Pindahkan Model ke Mode Evaluasi:**

Pindahkan model ke mode evaluasi dengan memanggil `model.eval()`. Ini memastikan bahwa layer yang mungkin berperilaku berbeda, seperti dropout, beroperasi dalam mode evaluasi.

3. **Lakukan Propagasi Maju (Forward Propagation):**

Gunakan model untuk melakukan propagasi maju pada data input dan mendapatkan hasil prediksi.

4. **Interpretasi Hasil:**

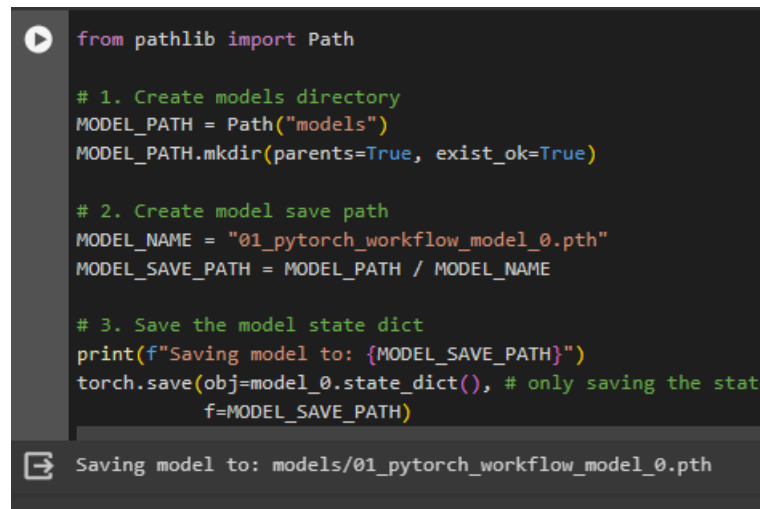
Interpretasikan hasil prediksi sesuai dengan tugas yang sedang dijalankan. Misalnya, jika model adalah model klasifikasi, hasil prediksi mungkin merupakan kelas yang diprediksi atau distribusi probabilitas untuk setiap kelas.

1.6 Menyimpan dan memuat model PyTorch

Untuk menyimpan dan memuat model di PyTorch, ada tiga metode utama yang harus diketahui.

1. torch.save

Menyimpan objek serial ke disk menggunakan utilitas acar Python. Model, tensor, dan berbagai objek Python lainnya seperti kamus dapat disimpan menggunakan file .torch.save



```
from pathlib import Path

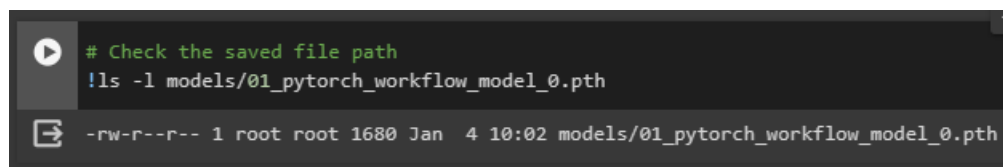
# 1. Create models directory
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True, exist_ok=True)

# 2. Create model save path
MODEL_NAME = "01_pytorch_workflow_model_0.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME

# 3. Save the model state dict
print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model_0.state_dict(), # only saving the state dict
           f=MODEL_SAVE_PATH)

Saving model to: models/01_pytorch_workflow_model_0.pth
```

Gambar 15: Torch.save



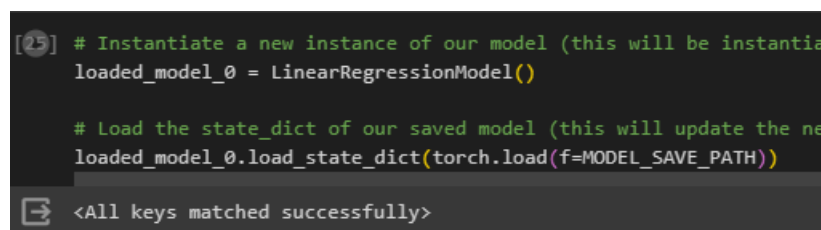
```
# Check the saved file path
!ls -l models/01_pytorch_workflow_model_0.pth

-rw-r--r-- 1 root root 1680 Jan  4 10:02 models/01_pytorch_workflow_model_0.pth
```

Gambar 16: Check The Saved File Path

2. torch.load

Menggunakan fitur unpickling untuk deserialisasi dan memuat file objek Python acar (seperti model, tensor, atau kamus) ke dalam memori. Kita juga dapat mengatur perangkat mana yang akan memuat objek (CPU, GPU, dll.).pickle



```
[25] # Instantiate a new instance of our model (this will be instantiated)
loaded_model_0 = LinearRegressionModel()

# Load the state_dict of our saved model (this will update the new model)
loaded_model_0.load_state_dict(torch.load(f=MODEL_SAVE_PATH))

<All keys matched successfully>
```

Gambar 17: Load Model

3. torch.nn.Module.load_state_dict

Memuat kamus parameter model () menggunakan objek yang disimpan.model.state_dict()state_dict()

CHAPTER 02

PyTorch Neural Network Classification

2.1 Make a binary classification dataset

Disini saya membuat sebuah dataset klasifikasi biner:

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import torch

# Set seed for reproducibility
torch.manual_seed(42)

# Create a binary classification dataset
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)

# Convert data to PyTorch tensors
X = torch.FloatTensor(X)
y = torch.FloatTensor(y).view(-1, 1)

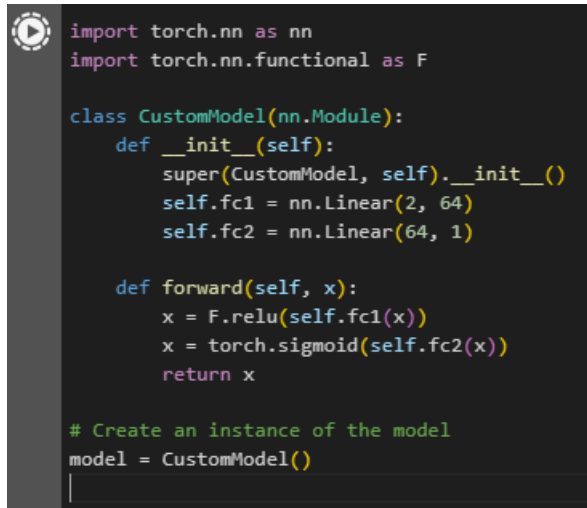
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Gambar 18: Binary Classification Dataset

- Langkah 1: saya meng-import Library yang akan digunakan nantinya seperti Mengimpor pustaka yang diperlukan, yaitu `make_moons` untuk membuat dataset dan `train_test_split` untuk membagi dataset menjadi data pelatihan dan pengujian, serta `torch` untuk bekerja dengan PyTorch.
- Langkah 2: saya Mengatur benih (seed) untuk memastikan reproduktibilitas hasil. Ini berguna agar hasil eksekusi kode yang menggunakan elemen-elemen acak (seperti pembuatan dataset) dapat direproduksi dengan hasil yang sama setiap kali dijalankan.
- Langkah 3: saya Menggunakan fungsi `make_moons` dari scikit-learn untuk membuat dataset dengan dua kelas (binary classification). `n_samples` menentukan jumlah sampel dalam dataset, `noise` menambahkan kebisingan ke data, dan `random_state` mengatur seed untuk hasil yang konsisten.
- Langkah 4: saya Mengonversi data `X` dan `y` ke dalam bentuk PyTorch tensors. Tensors adalah struktur data fundamental dalam PyTorch yang mendukung operasi tensor (array multidimensi) dan komputasi yang efisien.
- Langkah 5: saya Memisahkan dataset menjadi data pelatihan (`X_train`, `y_train`) dan data pengujian (`X_test`, `y_test`) menggunakan fungsi `train_test_split`. `test_size` menentukan proporsi data yang akan diambil sebagai data pengujian.

2.2 Exercises 2: Build a model

Di exercises 2 ini saya akan membuat sebuah model:



```
import torch.nn as nn
import torch.nn.functional as F

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.fc1 = nn.Linear(2, 64)
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x

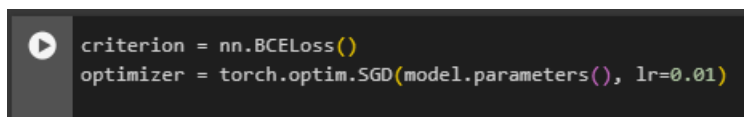
# Create an instance of the model
model = CustomModel()
```

Gambar 19: Build a model

- Langkah 1: saya akan menambahkan beberapa library torch yang akan digunakan. Mengimpor modul nn (neural network) dan functional dari PyTorch. nn menyediakan kelas-kelas dasar untuk pembuatan model neural network, sedangkan F berisi fungsi-fungsi aktivasi dan operasi neural network lainnya.
- Langkah 2: saya akan mendefinisikan custom model dengan Membuat kelas CustomModel yang merupakan turunan dari kelas nn.Module. Dalam konstruktor (`__init__`), kita mendefinisikan dua lapisan linear (`nn.Linear`). Lapisan pertama (`fc1`) memiliki 2 input (sesuai dengan dimensi input data) dan 64 output, sedangkan lapisan kedua (`fc2`) memiliki 64 input dan 1 output.
- Langkah 3: saya akan mendefinisikan metode forward yang akan menentukan bagaimana data mengalir melalui model. Pada lapisan pertama, aktivasi ReLU (`F.relu`) diterapkan, dan pada lapisan kedua, aktivasi sigmoid (`torch.sigmoid`) digunakan untuk menghasilkan output dalam rentang (0, 1) untuk tugas klasifikasi biner.
- Langkah 4: saya akan membuat sebuah instansi (objek) dari model dengan menggunakan kelas CustomModel. Setelah membuat instansi model, Kita dapat menggunakan objek ini untuk melatih dan menguji model pada data yang telah Kita siapkan sebelumnya.

2.3 Exercise 3: Setup loss function and optimizer

Disini saya akan membuat loss function dan optimizer.



```
criterion = nn.BCELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

Gambar 20: Loss Function dan Optimizer

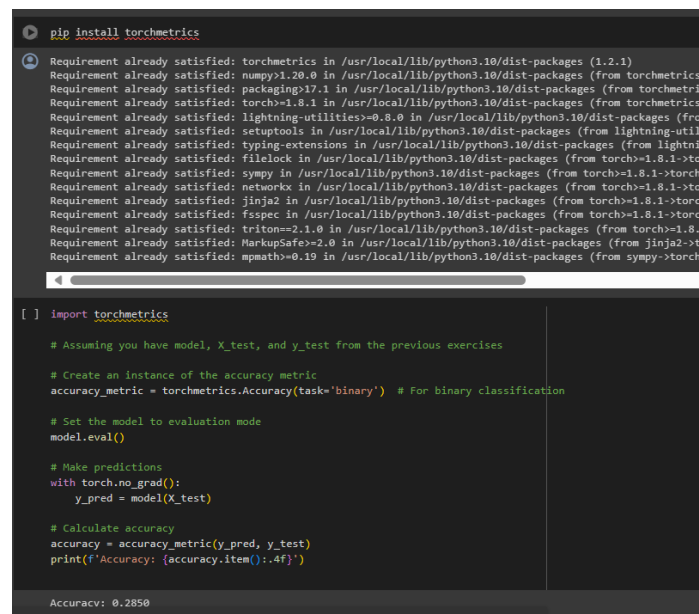
Langkah 1: saya menggunakan fungsi loss BCELoss dari modul nn. BCELoss digunakan khusus untuk tugas klasifikasi biner (binary classification). Loss ini cocok untuk kasus di mana model perlu memprediksi probabilitas kelas positif atau negatif.

Langkah 2: saya Menggunakan algoritma optimasi SGD untuk mengoptimalkan parameter dari model. SGD merupakan metode optimasi yang umum digunakan

dalam pelatihan model. Parameter yang dioptimalkan adalah parameter dari model (`model.parameters()`). `lr` (learning rate) diatur menjadi 0.01, yang merupakan parameter yang mengontrol seberapa besar perubahan yang akan diterapkan pada parameter model selama setiap iterasi optimasi. Learning rate (`lr`) mempengaruhi seberapa besar perubahan yang dilakukan pada parameter model pada setiap langkah iterasi. Nilai yang terlalu tinggi dapat membuat model tidak stabil atau bahkan divergen, sementara nilai yang terlalu rendah dapat membuat proses konvergensi menjadi lambat.

2.4 Training Loop

Disini saya akan melakukan Training Loop.



```

pip install torchmetrics

Requirement already satisfied: torchmetrics in /usr/local/lib/python3.10/dist-packages (1.2.1)
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics)
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics)
Requirement already satisfied: lightning-utilities>=0.8.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from lightning-utilities)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torch)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torch)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torch)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torch)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torch)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torch)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->torch)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch)

[ ] import torchmetrics

# Assuming you have model, X_test, and y_test from the previous exercises

# Create an instance of the accuracy metric
accuracy_metric = torchmetrics.Accuracy(task='binary') # For binary classification

# Set the model to evaluation mode
model.eval()

# Make predictions
with torch.no_grad():
    y_pred = model(X_test)

# Calculate accuracy
accuracy = accuracy_metric(y_pred, y_test)
print(f'Accuracy: {accuracy.item():.4f}')

Accuracy: 0.2850
  
```

Gambar 21: Training Loop

- Langkah 1: saya akan install torchmetrics dengan menggunakan pip. torchmetrics adalah pustaka yang menyediakan berbagai metrik evaluasi kinerja yang dapat digunakan bersama dengan PyTorch.
- Langkah 2: saya mengimpor modul torchmetrics. Dengan menggunakan pustaka ini, Kita dapat dengan mudah menghitung berbagai metrik evaluasi kinerja seperti akurasi, presisi, recall, F1-score, dan lainnya.
- Langkah 3: saya Membuat sebuah instansi dari metrik akurasi (Accuracy) dari torchmetrics. Parameter `task='binary'` menunjukkan bahwa kita sedang bekerja pada tugas klasifikasi biner. Metrik akurasi ini akan dihitung berdasarkan prediksi (`y_pred`) dan label sebenarnya (`y_test`).
- Langkah 4: saya Mengatur model ke mode evaluasi. Ini penting karena beberapa lapisan (seperti lapisan dropout) dapat berperilaku berbeda antara mode pelatihan dan evaluasi.
- Langkah 5: saya Membuat prediksi pada data pengujian (`X_test`) menggunakan model yang telah dilatih. Dengan `torch.no_grad()`, saya memastikan bahwa tidak ada perhitungan gradien yang dihitung selama tahap prediksi,

- Langkah 6: saya Menggunakan metrik akurasi yang telah dibuat untuk menghitung akurasi berdasarkan prediksi model dan label sebenarnya. Nilai akurasi kemudian dicetak ke layar. Dalam contoh ini, akurasi dicetak dengan format dua desimal.
- Hasil Run: Hasil akurasi yang dicetak ke layar (0.2850) menunjukkan seberapa baik model dapat memprediksi label dengan benar pada data pengujian. Akurasi sebesar 0.2850 berarti model memprediksi dengan benar sekitar 28.5% dari seluruh sampel data pengujian.

2.5 Exercises 5: Make Prediction and Plot decision boundary

Disini saya akan melakukan prediksi dan plot decision boundary.

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

def plot_decision_boundary(model, X, y):
    h = .02 # step size in the mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

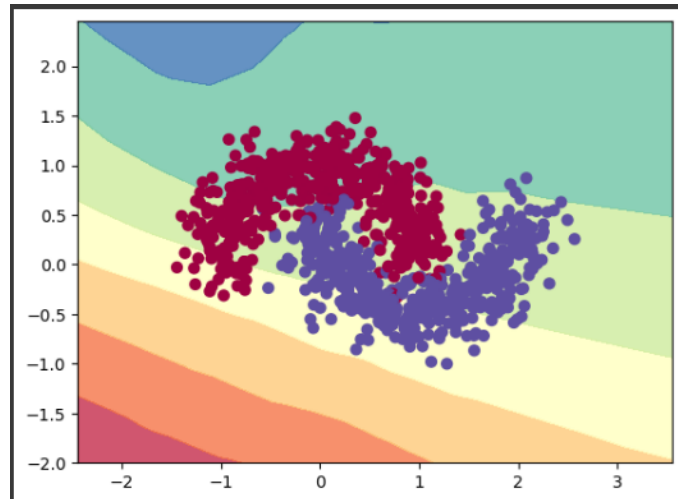
    Z = model(torch.FloatTensor(np.c_[xx.ravel(), yy.ravel()])).detach().numpy()
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
    plt.show()

# Plot decision boundary
model.eval()
plot_decision_boundary(model, X.numpy(), y.numpy())
```

Gambar 22: Prompt prediction and plot decision boundary

- Langkah 1: saya Mengimpor pustaka NumPy (np) untuk operasi numerik dan pustaka Matplotlib (plt) untuk visualisasi data.
- Langkah 2: saya Mendefinisikan fungsi plot_decision_boundary dengan tiga parameter: model (model PyTorch yang akan diplot), X (data input), dan y (label kelas).
- Langkah 3: saya Menentukan langkah (h) untuk membuat grid, serta batas-batas dari grid (x_min, x_max, y_min, y_max). Kemudian, membuat mesh grid (xx, yy) menggunakan np.meshgrid dengan langkah h.
- Langkah 4: saya Membuat prediksi model pada seluruh titik grid menggunakan model. Inputnya dihasilkan dengan menggabungkan xx dan yy menggunakan np.c_, kemudian diubah menjadi tensor PyTorch (torch.FloatTensor) dan diprediksi. Hasil prediksi kemudian diubah menjadi NumPy array dan direshape sesuai dengan bentuk grid.
- Langkah 5: saya Menggambar kontur keputusan (contourf) berdasarkan hasil prediksi Z. Kontur diwarnai menggunakan cmap (plt.cm.Spectral) dan diberi alpha untuk transparansi. Selain itu, data asli (X) ditampilkan sebagai scatter plot dengan warna yang sesuai dengan label kelas (y). Hasil keseluruhan diplot menggunakan plt.show().
- Langkah 6: saya mengatur model ke mode evaluasi (model.eval()) dan memanggil fungsi plot_decision_boundary dengan data dan label yang telah diubah menjadi NumPy arrays.
- Hasil: berupa visualisasi plot dan prediksi



Gambar 23: Visualisasi Plot dan Prediksi

2.6 Replicate the Tanh activation function

Disini saya akan melakukan perhitungan Tanh function.

```
def tanh(x):
    return (torch.exp(x) - torch.exp(-x)) / (torch.exp(x) + torch.exp(-x))

# Test the tanh function
x = torch.tensor([1.0, 2.0, 3.0])
print(tanh(x))

tensor([0.7616, 0.9640, 0.9951])
```

Gambar 24: Tanh Function

- Langkah 1: saya mendefinisikan fungsi tanh dengan sesuai rumus yang berlaku.
- Langkah 2: saya membuat tensor input x dengan nilai [1.0, 2.0, 3.0] dan mengaplikasikan fungsi tanh pada tensor tersebut. Hasilnya dicetak ke layar.
- Hasil: Hasil dari fungsi tanh pada tensor input [1.0, 2.0, 3.0] adalah tensor yang berisi nilai [0.7616, 0.9640, 0.9951]. Setiap elemen pada tensor output adalah hasil aplikasi fungsi hiperbola tangen pada elemen yang sesuai dari tensor input.

2.7 Exercise 7: Create a multi-class dataset and train a model

Disini saya membuat multi-class dataset dan men- train sebuah model.

```

# Code for creating a spiral dataset from CS231n
import numpy as np
N = 100 # number of points per class
D = 2 # dimensionality
K = 3 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros((N*K, dtype='uint8')) # class labels
for j in range(K):
    ix = range(N*j,N*(j+1))
    r = np.linspace(0,1,N) # radius
    t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
    X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
    y[ix] = j
# Lets visualize the data
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
plt.show()

criterion_spiral = nn.CrossEntropyLoss()

def train_model(model, criterion, optimizer, X_train, y_train, X_test, y_test, num_epochs=300):
    for epoch in range(1, num_epochs + 1):
        # Training
        model.train()
        optimizer.zero_grad()
        outputs = model(X_train)
        loss = criterion(outputs, y_train)
        loss.backward()
        optimizer.step()

        # Testing every 10 epochs
        if epoch % 10 == 0:
            model.eval()
            with torch.no_grad():
                predicted = torch.max(outputs, 1)
                train_acc = accuracy_metric(predicted, y_train)
                test_outputs = model(X_test)
                predicted_test = torch.max(test_outputs, 1)
                test_acc = accuracy_metric(predicted_test, y_test)

            print('Epoch (epoch)/(num_epochs), '
                  f'Train Loss: {loss.item():.4f}, '
                  f'Train Acc: {train_acc.item():.4f}, '
                  f'Test Acc: {test_acc.item():.4f}')

```

Gambar 25: Membuat multi-class dan train model

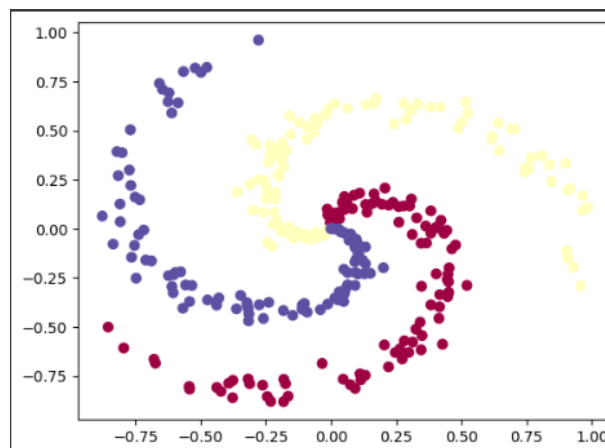
Langkah 1: saya membuat dataset spiral, Dalam loop for, titik-titik dalam setiap kelas dihasilkan dengan menggunakan parameter r (radius) dan t (theta). Setiap titik dihasilkan dengan melibatkan sinus dan kosinus dari t , yang juga ditambahi dengan nilai random untuk membuat spiral terlihat lebih realistis.

Langkah 2: saya membuat visualisasi dataset spiral

Langkah 3: saya membuat pelatihan model Neural Network dengan menggunakan fungsi kerugian CrossEntropyLoss, yang cocok untuk tugas klasifikasi multi-kelas. Fungsi `train_model` mengambil model, kriteria, optimizer, data pelatihan, data uji, dan parameter opsional `num_epochs` sebagai argumen. Dalam loop pelatihan, model diatur ke mode pelatihan (`model.train()`), kemudian dilakukan proses pelatihan dengan menghitung loss, melakukan backpropagation, dan mengoptimalkan parameter dengan optimizer.

Setelah setiap 10 epoch, model diubah ke mode evaluasi (`model.eval()`), dan akurasi pada data pelatihan dan data uji dicetak ke layar.

Fungsi ini dapat digunakan untuk melatih model pada dataset spiral dan memonitor performanya selama pelatihan.



Gambar 26: Visualisasi Spiral

CHAPTER 03

PYTORCH COMPUTER VISION

3.1 Computer Vision libraries in PyTorch

```

# Import PyTorch
import torch
from torch import nn

# Import torchvision
import torchvision
from torchvision import datasets
from torchvision.transforms import ToTensor

# Import matplotlib for visualization
import matplotlib.pyplot as plt

# Check versions
# Note: your PyTorch version shouldn't be lower than 1.10.0 and torchvision version shouldn't be lower than 0.11
print(f"PyTorch version: {torch.__version__}\ntorchvision version: {torchvision.__version__}")

PyTorch version: 2.0.1+cu118
torchvision version: 0.15.2+cu118

```

Gambar 27: Import Library PyTorch & torchvision

Disini saya melakukan import library yang akan digunakan di modul ini seperti library torch dan torchvision. Saya juga menambahkan prompt untuk melihat versi yang di gunakan saat ini.

3.2 Getting Dataset

3.2.1 Input and output shapes of a computer vision model

```

[ ] # Setup training data
train_data = datasets.FashionMNIST(
    root="data", # where to download data to?
    train=True, # get training data
    download=True, # download data if it doesn't exist on disk
    transform=ToTensor(), # images come as PIL format, we want to turn into Torch tensors
    target_transform=None # you can transform labels as well
)

# Setup testing data
test_data = datasets.FashionMNIST(
    root="data",
    train=False, # get test data
    download=True,
    transform=ToTensor()
)

Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
100%|#####| 26421880/26421880 [00:01<00:00, 16189161.14it/s]
Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
100%|#####| 29515/29515 [00:00<00:00, 269809.67it/s]
Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
100%|#####| 4422102/4422102 [00:00<00:00, 4950701.58it/s]
Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
100%|#####| 5148/5148 [00:00<00:00, 4744512.63it/s]Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/

```

Gambar 28: Mengambil dataset

Disini saya akan mendownload dataset yang sudah disediakan sebelumnya dan dataset tersebut berupa data fashion.

```

# What's the shape of the image?
image.shape

torch.Size([1, 28, 28])

[ ] # How many samples are there?
len(train_data.data), len(train_data.targets), len(test_data.data), len(test_data.targets)

(60000, 60000, 10000, 10000)

[ ] # See classes
class_names = train_data.classes
class_names

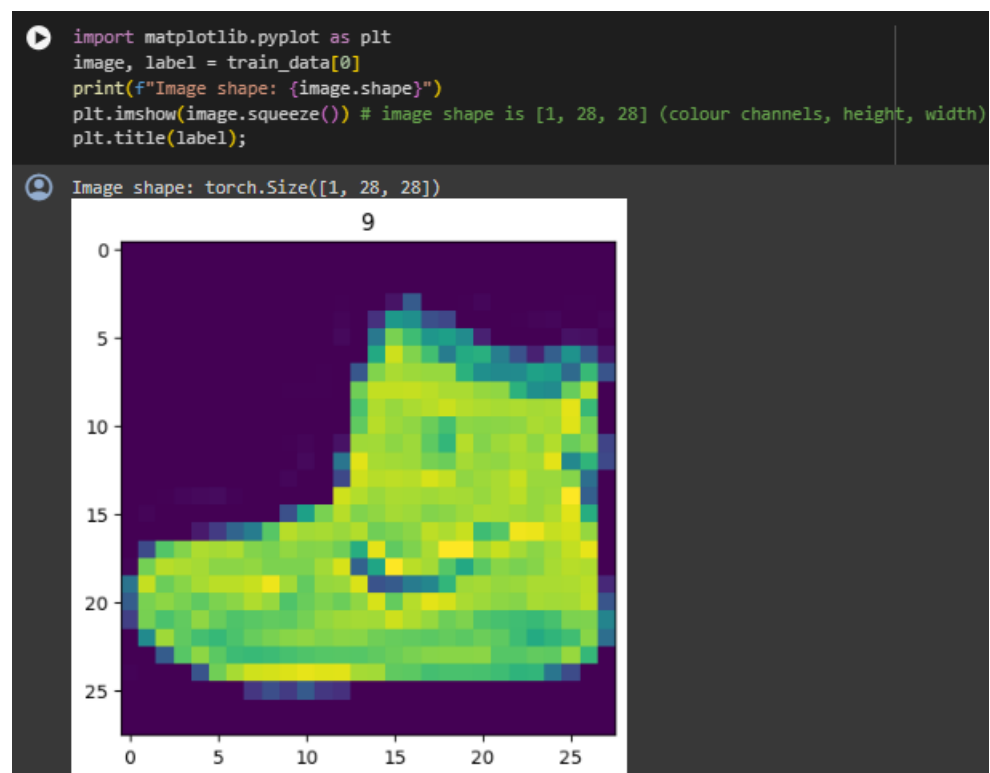
['T-shirt/top',
'Trouser',
'Pullover',
'Dress',
'Coat',
'Sandal',
'Shirt',
'Sneaker',
'Bag',
'Ankle boot']

```

Gambar 29: Mencari shape, class, dan samples

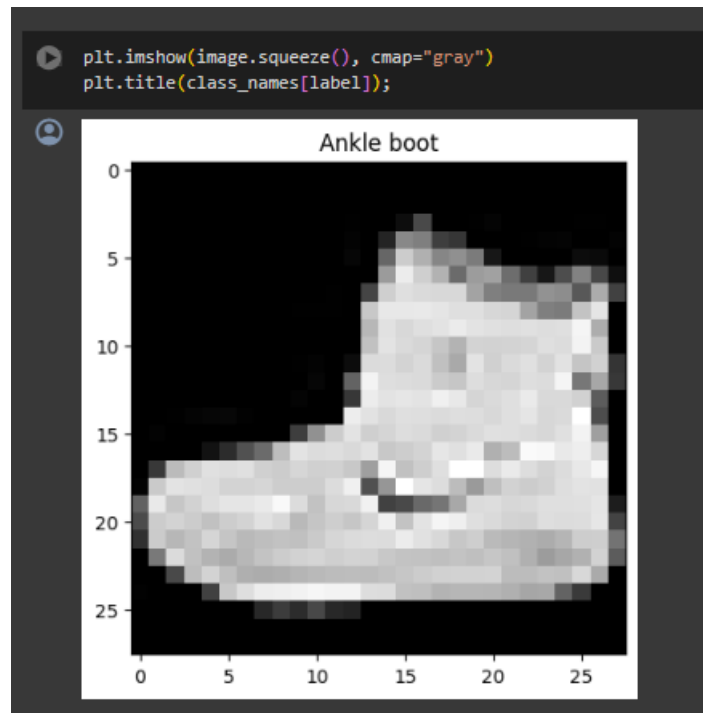
saya mencari tahu berapa shape dan sample yang ada di dalam dataset tersebut, dan juga melihat class yang digunakan di dataset tersebut.

3.2.2 Visualizing our data



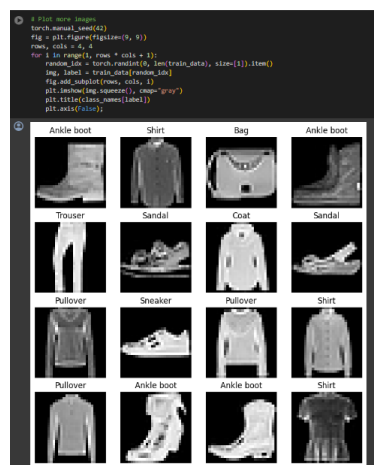
Gambar 30: Visualisasi data Shoes

Disini saya melakukan visualisasi data Sepatu dengan menggunakan library matplotlib.pyplot as plt.



Gambar 31: Visualisasi data Shoes tema grey

Sama seperti sebelumnya hanya saja saya mengubah warna dari tampilannya menjadi grey.



Gambar 32: Visualisasi data lebih banyak

Disini saya juga mengvisualisasi lebih banyak data yang ada di dataset tersebut.

3.3 Prepare DataLoader

```

from torch.utils.data import DataLoader

# Setup the batch size hyperparameter
BATCH_SIZE = 32

# Turn datasets into iterables (batches)
train_dataloader = DataLoader(train_data, # dataset to turn into iterable
                              batch_size=BATCH_SIZE, # how many samples per batch?
                              shuffle=True # shuffle data every epoch?
                              )

test_dataloader = DataLoader(test_data,
                              batch_size=BATCH_SIZE,
                              shuffle=False # don't necessarily have to shuffle the testing data
                              )

# Let's check out what we've created
print(f"Dataloaders: {train_dataloader, test_dataloader}")
print(f"Length of train dataloader: {len(train_dataloader)} batches of {BATCH_SIZE}")
print(f"Length of test dataloader: {len(test_dataloader)} batches of {BATCH_SIZE}")

```

Dataloaders: (<torch.utils.data.dataloader.DataLoader object at 0x7fc991463cd0>, <torch.utils.data.dataloader.DataLoader object at 0x7fc991475120>)
 Length of train dataloader: 1875 batches of 32
 Length of test dataloader: 313 batches of 32

Gambar 33: Prepare DataLoader

Disini saya melakukan prepare dataloader:

- (<torch.utils.data.dataloader.DataLoader object at 0x7fc991463cd0>, <torch.utils.data.dataloader.DataLoader object at 0x7fc991475120>), Ini menunjukkan dua objek DataLoader. Dua dataloader ini mungkin mewakili dataloader pelatihan dan dataloader pengujian, tetapi informasi ini tidak cukup untuk memastikannya. Objek DataLoader pada umumnya digunakan untuk memuat data dalam bentuk batch saat melatih atau menguji model.
- Length of train dataloader: 1875 batches of 32, Length of test dataloader: 313 batches of 32, Ini memberikan informasi tentang panjang atau jumlah batch dari dataloader pelatihan dan dataloader pengujian.
 - Dataloader pelatihan (train dataloader) memiliki panjang 1875 batch, dan setiap batch memiliki ukuran 32.
 - Dataloader pengujian (test dataloader) memiliki panjang 313 batch, dan setiap batch juga memiliki ukuran 32.

Jumlah batch adalah penting karena pada setiap iterasi selama pelatihan, model akan melihat satu batch data. Dengan demikian, jumlah iterasi yang dilakukan selama satu epoch dapat dihitung dengan membagi panjang dataloader dengan ukuran batch. Dalam hal ini, setiap epoch pelatihan akan terdiri dari 1875 iterasi (batch) dan setiap epoch pengujian akan terdiri dari 313 iterasi (batch).

Selanjutnya saya pengecekan apa saja yang ada di dalam training dataloader:

```

# Check out what's inside the training dataloader
train_features_batch, train_labels_batch = next(iter(train_dataloader))
train_features_batch.shape, train_labels_batch.shape

```

(torch.Size([32, 1, 28, 28]), torch.Size([32]))

Gambar 34: Pengecekan isi Training DataLoader

(torch.Size([32, 1, 28, 28]), torch.Size([32])) Pesan tersebut menyajikan dua objek torch.Size yang mungkin mewakili dimensi dari dua tensor PyTorch. Dimensi tensor pertama adalah [32, 1, 28, 28], sementara dimensi tensor kedua adalah [32].

Untuk menafsirkan dimensi-dimensi tersebut:

Tensor Pertama:

Dimensi: [32, 1, 28, 28]

Jumlah batch: 32

Saluran warna (channel): 1

Tinggi gambar: 28 piksel

Lebar gambar: 28 piksel

Ini mengindikasikan bahwa tensor pertama mungkin berisi batch dari gambar-gambar grayscale dengan ukuran 28x28 piksel.

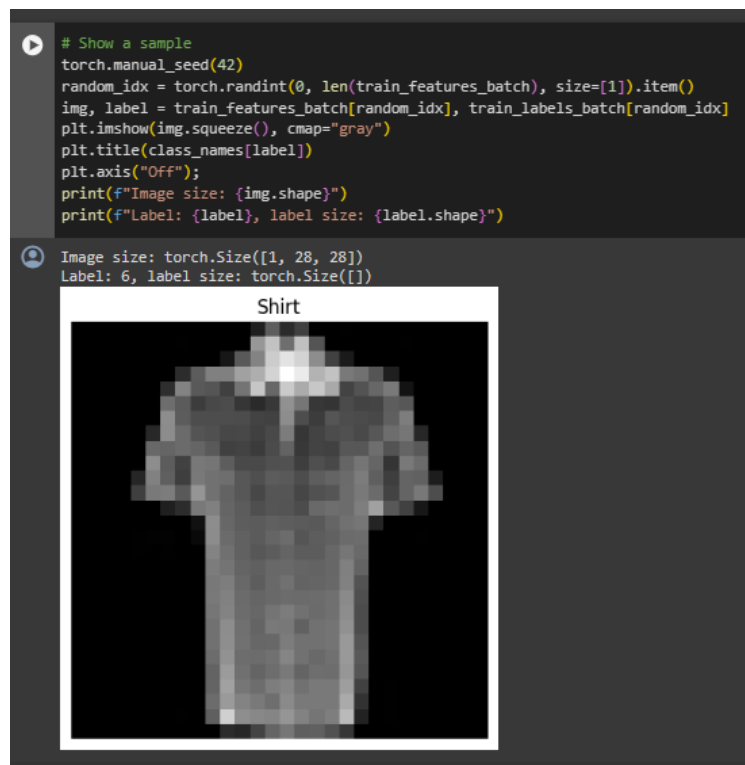
Tensor Kedua:

Dimensi: [32]

Jumlah batch: 32

Ini mengindikasikan bahwa tensor kedua mungkin berisi label kelas untuk setiap gambar dalam batch, di mana setiap label adalah bilangan bulat dan ada 32 label untuk setiap batch.

Setelah itu saya melakukan visualisasi sample:



Gambar 35: Visualisasi Sample

3.4 Model 0: Build a baseline model

Disini saya melakukan build baseline model yang Dimana Baseline model adalah model sederhana yang digunakan sebagai titik awal atau pembandingan dalam pembangunan model lebih kompleks. Model ini memberikan performa dasar atau

stKitar yang dapat diukur untuk suatu tugas tertentu. Baseline model umumnya tidak memiliki fitur-fitur atau kecerdasan yang kompleks, dan digunakan sebagai acuan untuk memastikan bahwa model-model yang lebih kompleks benar-benar memberikan peningkatan kinerja yang signifikan.

```
# Create a flatten layer
flatten_model = nn.Flatten() # all nn modules function as a model (can do a forward pass)

# Get a single sample
x = train_features_batch[0]

# Flatten the sample
output = flatten_model(x) # perform forward pass

# Print out what happened
print(f"Shape before flattening: {x.shape} -> [color_channels, height, width]")
print(f"Shape after flattening: {output.shape} -> [color_channels, height*width]")

# Try uncommenting below and see what happens
#print(x)
#print(output)

Shape before flattening: torch.Size([1, 28, 28]) -> [color_channels, height, width]
Shape after flattening: torch.Size([1, 784]) -> [color_channels, height*width]

[ ] from torch import nn
class FashionMNISTModelV0(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.layer_stack = nn.Sequential(
            nn.Flatten(), # neural networks like their inputs in vector form
            nn.Linear(in_features=input_shape, out_features=hidden_units), # in_features
            nn.Linear(in_features=hidden_units, out_features=output_shape)
        )

    def forward(self, x):
        return self.layer_stack(x)

[ ] torch.manual_seed(42)

# Need to setup model with input parameters
model_0 = FashionMNISTModelV0(input_shape=784, # one for every pixel (28x28)
                               hidden_units=10, # how many units in the hidden layer
                               output_shape=len(class_names) # one for every class
                               )
model_0.to("cpu") # keep model on CPU to begin with

FashionMNISTModelV0(
  (layer_stack): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=784, out_features=10, bias=True)
    (2): Linear(in_features=10, out_features=10, bias=True)
  )
)
```

Gambar 36: Baseline Model

Setelah itu saya membuat loss function dan evaluasi metric agar kita dapat melihat akurasi yang dihasilkan pada saat melakukan training.

```
[ ] import requests
    from pathlib import Path

    # Download helper functions from Learn PyTorch repo (if not already downloaded)
    if Path("helper_functions.py").is_file():
        print("helper_functions.py already exists, skipping download")
    else:
        print("Downloading helper_functions.py")
        # Note: you need the "raw" GitHub URL for this to work
        request = requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-deep-learning/main/helper_functions.py")
        with open("helper_functions.py", "wb") as f:
            f.write(request.content)

    Downloading helper_functions.py

[ ] # Import accuracy metric
    from helper_functions import accuracy_fn # Note: could also use torchmetrics.Accuracy(task = 'multiclass', num_classes=len

    # Setup loss function and optimizer
    loss_fn = nn.CrossEntropyLoss() # this is also called "criterion"/"cost function" in some places
    optimizer = torch.optim.SGD(params=model_0.parameters(), lr=0.1)
```

Gambar 37: Loss Function dan Optimizer

Pada bagian ini saya menggunakan `nn.CrossEntropyLoss()` dan `torch.optim.SGD`, yang Dimana `nn.CrossEntropyLoss()` Fungsi ini mencakup langkah-langkah softmax dan perhitungan log loss (cross-entropy) secara otomatis. dan `torch.optim.SGD` algoritma optimizer stokastik gradien descen (Stochastic Gradient Descent). Dan menunjukkan bahwa optimizer akan mengoptimalkan parameter (bobot dan bias) dari model `model_0`.

3.5 Pembuatan Fungsi untuk Mengukur Waktu Eksperimen

Untuk mengukur waktu pelatihan, dimasukkan fungsi (`print_train_time`). Fungsi ini untuk mengukur waktu yang dibutuhkan model kita untuk berlatih pada CPU versus menggunakan GPU.

3.6 Pembuatan Loop Pelatihan dan Pelatihan Model pada Batch Data

Bagian ini mengeksekusi loop pelatihan untuk melatih model dasar (`model_0`). Loop pelatihan melakukan iterasi pada setiap epoch, dan untuk setiap epoch, memproses batch pelatihan menggunakan `DataLoader` (`train_dataloader`). Dan proses pelatihan dilakukan 3x epoch dengan melakukan lock 5x setiap epochnya dibagi dari 60000 sample yang ada. Dan akan tertampil waktu pelatihan pada CPU dan hasil pelatihan loss, test loss, dan akurasi testnya pada setiap epoch.

3.7 Membuat Prediksi dan Mendapatkan Hasil Model 0

Disini saya akan membuat prediksi dan mencoba mendapatkan hasil dari model yang sudah saya training. mari kita buat fungsi yang mengambil model terlatih, a , fungsi kerugian dan fungsi akurasi. `DataLoader`

Fungsi ini akan menggunakan model untuk membuat prediksi pada data di dan kemudian kita dapat mengevaluasi prediksi tersebut menggunakan fungsi kerugian dan fungsi akurasi. `DataLoader`

```
Hasil: {'model_name': 'FashionMNISTModelV0',
      'model_loss': 0.47663894295692444,
      'model_acc': 83.42651757188499}
```

Ini menunjukkan bahwa model "FashionMNISTModelV0" memiliki tingkat akurasi sekitar 83.43% pada dataset FashionMNIST, dan nilai loss pada evaluasi tertentu adalah sekitar 0.4766. Akurasi adalah persentase dari prediksi yang benar dibandingkan dengan jumlah total sampel, sedangkan loss mencerminkan seberapa baik model memprediksi nilai sejati.

3.8 Menyiapkan Kode Agen-Langka (Device Agnostic)

Disini kita akan mengidentifikasi perangkat apakah terdapat GPU dengan menggunakan

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

Jika hasil yang di tampilkan "cuda" maka terdapat GPU pada perangkatnya sebaliknya jika yang ditampilkan adalah "cpu" maka perangkat tidak terdapat gpu.

3.9 Model 1: Membangun Model Lebih Baik dengan Non-Linearitas

Disini Kita akan melakukannya dengan membuat ulang model yang mirip dengan sebelumnya, kecuali kali ini kita akan menempatkan fungsi non-linear () di antara setiap layer linear.nn.ReLU().

3.10 Model 2 - Membangun Convolutional Neural Network (CNN)

Disini kita akan menggunakan CNN dan Modelnya yaitu TinyVGG. TinyVGG mengikuti struktur tipikal dari CNN: Input layer -> [Convolutional layer -> activation layer -> pooling layer] -> Output layer.

3.11 Membandingkan Hasil Model dan Waktu Pelatihan

Kita telah melatih tiga model berbeda.

- model_0 - Model dasar kami dengan dua lapisan.nn.Linear()
- model_1 - pengaturan yang sama dengan model dasar kami kecuali dengan lapisan di antara lapisan.nn.ReLU()nn.Linear()
- model_2 - model CNN pertama kami yang meniru arsitektur TinyVGG di situs web CNN Explainer.

Dan hasil yang diperlihatkan bahwa model CNN () kami melakukan yang terbaik (kerugian terendah, akurasi tertinggi) tetapi memiliki waktu pelatihan terlama.FashionMNISTModelV2 Dan model dasar () berkinerja lebih baik daripada ().FashionMNISTModelV0model_1FashionMNISTModelV1

3.12 Membuat dan Menilai Prediksi Acak dengan Model Terbaik

Disini kita melakukan penilaian prediksi acak dengan model terbaik membuat plot visualisasi



Gambar 38: Visualisasi Penilaian data

3.13 Membuat Matriks Konfusi untuk Evaluasi Lebih Lanjut

Untuk menganalisa hasil dari sebuah model kita bisa menggunakan `torchmetrics.ConfusionMatrix`, `confusion matrix`, dan hasil yang ditampilkan sudah cukup bagus walaupun masih ada beberapa katagori class seperti shirt yang diindikasikan sebagai Tshirt, pullover, dress, dan coat.

3.14 Menyimpan dan Memuat Model Terbaik

- Model terbaik disimpan dengan menyimpan `state_dict()` menggunakan `torch.save()`.
- Model terbaik dimuat kembali dengan membuat instansi baru dari kelas model dan menggunakan `load_state_dict()`.

CHAPTER 04

PYTORCH CUSTOM DATASETS

4.1 Get data

Data yang akan kita gunakan adalah subset dari dataset Food101. Food101 adalah tolok ukur visi komputer yang populer karena berisi 1000 gambar dari 101 jenis makanan yang berbeda, dengan total 101.000 gambar (75.750 kereta dan 25.250 tes).

Alih-alih 101 kelas makanan, kita akan mulai dengan 3: pizza, steak, dan sushi. Dan alih-alih 1.000 gambar per kelas, kita akan mulai dengan 10% acak (mulai dari yang kecil, tingkatkan bila perlu).

```
Dalam [3]: 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")
    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)

data/pizza_steak_sushi directory exists.
```

Gambar 39: Dataset Pizza

4.2 data preparation

Kita akan menggunakan 3 gambar yaitu Pizza, Sushi, dan Steak.

```
pizza_steak_sushi/ <- overall dataset folder
train/ <- training images
  pizza/ <- class name as folder name
    image01.jpeg
    image02.jpeg
    ...
  steak/
    image24.jpeg
    image25.jpeg
    ...
  sushi/
    image37.jpeg
    ...
test/ <- testing images
  pizza/
    image101.jpeg
    image102.jpeg
    ...
  steak/
    image154.jpeg
    image155.jpeg
    ...
  sushi/
    image167.jpeg
    ...
```

Gambar 40: Data yang digunakan

Kita dapat memeriksa apa yang ada di direktori data kita dengan menulis fungsi pembantu kecil untuk berjalan melalui masing-masing subdirektori dan menghitung file yang ada.

Untuk melakukannya, kita akan menggunakan `os.walk()`.

```

am [4]: 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 subdirectories 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
am [5]: walk_through_dir(image_path)
There are 2 directories and 1 images in 'data/pizza_steak_sushi'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi/test'.
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'.
There are 0 directories and 25 images in 'data/pizza_steak_sushi/test/pizza'.
There are 3 directories and 0 images in 'data/pizza_steak_sushi/train'.
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 0 directories and 78 images in 'data/pizza_steak_sushi/train/pizza'.

```

Gambar 41: Direktori data

4.3 Transforming data

Sebelum kita dapat menggunakan data gambar kita dengan PyTorch, kita perlu:

1. Ubah menjadi tensor (representasi numerik dari gambar kita).
2. Ubah menjadi `a` dan kemudian `a`, kami akan menyebutnya `data_loader` dan singkatnya `torch.utils.data.DataLoader`.

Karena kita bekerja dengan masalah penglihatan, kita akan melihat fungsi pemuatan data kita serta `torchvision.transforms` untuk menyiapkan data kami `torchvision.datasets`.

Mengubah data dengan `torchvision.transforms`, `torchvision.transforms` berisi banyak metode pra-bangun untuk memformat gambar, mengubahnya menjadi tensor dan bahkan memanipulasinya untuk augmentasi data (praktik mengubah data untuk mempersulit model untuk dipelajari, kita akan melihatnya nanti) tujuan.

Kita akan mengkompilasi menggunakan `torchvision.transforms.Compose()`.

```

# Write transform for image
data_transform = transforms.Compose([
    # Resize the images to 64x64
    transforms.Resize(size=(64, 64)),
    # Flip the images randomly on the horizontal
    transforms.RandomHorizontalFlip(p=0.5), # p = probability of flip, 0.5 =
    # Turn the image into a torch.Tensor
    transforms.ToTensor() # this also converts all pixel values from 0 to 255
])

```

Gambar 42: Transform Compose

```
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.

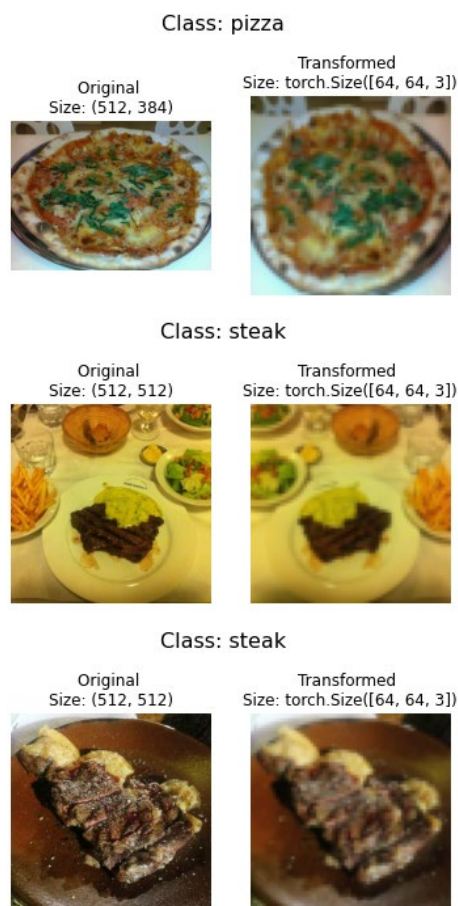
    Args:
        image_paths (list): List of target image paths.
        transform (PyTorch Transforms): Transforms to apply to images.
        n (int, optional): Number of images to plot. Defaults to 3.
        seed (int, optional): Random seed for the random generator. Defaults to 42.
    """
    random.seed(seed)
    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)
```

Gambar 43: Fungsi percobaan pada gambar



Gambar 44: Hasil Percobaan

4.4 Memuat Data Gambar Menggunakan Image Folder

menggunakan kelas `torchvision.datasets.ImageFolder` Di mana kita dapat meneruskannya jalur file dari direktori gambar target serta serangkaian transformasi yang ingin kita lakukan pada gambar kita.

Ketika gambar di masukkan akan berubah menjadi bentuk tensor (dengan bentuk) dan label dalam bentuk bilangan bulat yang berkaitan dengan kelas tertentu (seperti yang dirujuk oleh atribut).[3, 64, 64]class_to_idx

4.5 Bentuk Lain dari Transformasi (Augmentasi Data)

Disini kita melakukan tranformasi dengan bentuk lain dengan parameter utama yang harus diperhatikan adalah `.transforms.TrivialAugmentWide(num_magnitude_bins=31)`.

4.6 Model 0: TinyVGG tanpa augmentasi data

Model TinyVGG yang pertama (Model 0) yang dibangun untuk mengklasifikasikan gambar pizza, steak, dan sushi tidak memberikan hasil yang baik setelah 5 epoch. Kurva kerugian dan akurasi menunjukkan bahwa model tidak berhasil belajar dengan baik dari data pelatihan dan tidak mampu melakukan generalisasi dengan baik pada data pengujian.

4.7 Exploring loss curves

4.6.1 Underfitting

Terjadi ketika model tidak dapat memahami dan menangkap pola yang ada dalam data pelatihan dengan baik. Model yang underfit biasanya terlalu sederhana atau tidak memiliki kapasitas yang cukup untuk mengatasi kompleksitas tugas yang dihadapi.

Karakteristik:

- Performa buruk pada data pelatihan.
- Performa buruk pada data validasi atau uji.
- Model tidak dapat memahami hubungan antara fitur dan target.

Cara menangani underfitting:

1. Tambahkan lebih banyak lapisan/unit ke model
2. Sesuaikan tingkat pembelajaran
3. Menggunakan transfer learning
4. Berlatih lebih lama
5. Gunakan lebih sedikit regularisasi

4.6.2 Overfitting

Terjadi ketika model terlalu cocok dengan data pelatihan dan menganggap noise atau variasi acak sebagai pola yang sebenarnya. Model yang overfit

biasanya memiliki kompleksitas yang tinggi dan terlalu rumit untuk data pelatihan yang tersedia.

Karakteristik:

- Performa yang sangat baik pada data pelatihan.
- Performa buruk pada data validasi atau uji.
- Model mungkin mengingat dengan baik data pelatihan tetapi gagal dalam membuat generalisasi yang baik untuk data baru.

Cara menangani overfitting:

1. Dapatkan lebih banyak data
2. Sederhanakan model
3. Menggunakan augmentasi data
4. Menggunakan transfer learning
5. Menggunakan lapisan dropout
6. Gunakan peluruhan tingkat pembelajaran
7. Gunakan penghentian lebih awal

4.6.3 Just Right

Terjadi ketika model memiliki tingkat kompleksitas yang sesuai dengan kompleksitas sebenarnya dari tugas yang dihadapi. Model yang baik secara umum mampu membuat generalisasi yang baik pada data yang belum pernah dilihat.

Karakteristik:

- Performa yang baik pada data pelatihan.
- Performa yang baik pada data validasi atau uji.
- Model mampu memahami dan menangkap pola yang sebenarnya dalam data.

4.8 TinyVGG dengan Augmentasi Data

Dari hasil pelatihan Model 1 dengan augmentasi data menggunakan TinyVGG, terlihat bahwa model ini mengalami performa yang kurang baik.

- Kurva Kerugian:
Kurva kerugian pada data pelatihan dan pengujian cenderung stagnan dan tidak menunjukkan peningkatan yang signifikan. Ini dapat mengindikasikan bahwa model belum berhasil menangkap pola yang kompleks dalam data.
- Akurasi:
Akurasi pada data pelatihan dan pengujian tetap rendah dan stabil di sekitar 26%. Hal ini mungkin menunjukkan bahwa model tidak mampu melakukan generalisasi dengan baik pada data yang tidak terlihat selama pelatihan.

- Potensi Overfitting atau Underfitting:

Dengan melihat hasil tersebut, sulit untuk dengan pasti menyimpulkan apakah model mengalami overfitting atau underfitting. Namun, karena akurasi rendah pada kedua set data, kemungkinan besar ada isu underfitting.

4.9 Compare model results

Disini kita akan membandingkan hasil model yang sudah kita buat. Kita dapat menentukan apakah hasil yang sudah kita lakukan mendekati bagus apa tidak.

Pertama kita ubah data menjadi pKitas dataframe.

```
Dalam [57]: import pandas as pd
model_0_df = pd.DataFrame(model_0_results)
model_1_df = pd.DataFrame(model_1_results)
model_0_df
```

```
Di luar[57]:
```

	train_loss	train_acc	test_loss	test_acc
0	1.107833	0.257812	1.136041	0.260417
1	1.084713	0.425781	1.162014	0.197917
2	1.115697	0.292969	1.169704	0.197917
3	1.095564	0.414062	1.138373	0.197917
4	1.098520	0.292969	1.142631	0.197917

Gambar 45: Hasil PKitas Dataframe

Dan sekarang kita dapat menulis beberapa kode plot menggunakan untuk memvisualisasikan hasil dan bersama-sama.matplotlibmodel_0model_1



Gambar 46: Grafik Plot

CHAPTER 05

PYTORCH GOING MODULAR

5.1 Going Modular

Going Modular melibatkan mengubah kode notebook (dari Jupyter Notebook atau notebook Google Colab) menjadi serangkaian skrip Python berbeda yang menawarkan fungsionalitas serupa.

Misalnya, kita dapat mengubah kode notebook kita dari serangkaian sel menjadi file Python berikut:

- `data_setup.py` - file untuk menyiapkan dan mengunduh data jika diperlukan.
- `engine.py` - file yang berisi berbagai fungsi pelatihan.
- `model_builder.py` atau - file untuk membuat model PyTorch.model.py
- `train.py` - file untuk memanfaatkan semua file lain dan melatih model PyTorch target.
- `utils.py` - File yang didedikasikan untuk fungsi utilitas yang bermanfaat.

5.2 Cell mode vs. script mode

- **Cell mode**
adalah notebook yang berjalan normal, setiap sel di notebook adalah kode atau markdown.
- **Script mode**
sangat mirip dengan notebook mode sel, namun, banyak sel kode dapat diubah menjadi skrip Python.

5.3 Pro dan kontra notebook vs skrip Python

- **Notebook**
Pro
 1. Mudah untuk bereksperimen / memulai
 2. Mudah dibagikan (misalnya link ke notebook Google Colab)
 3. Sangat visual
 Kontra
 1. Pembuatan versi bisa jadi sulit
 2. Sulit untuk menggunakan hanya bagian-bagian tertentu
 3. Teks dan grafik bisa menghalangi kode
- **Skrip Python**
Pro
 1. Dapat mengemas kode bersama-sama (menyimpan penulisan ulang kode serupa di berbagai buku catatan)

2. Dapat menggunakan git untuk pembuatan versi
3. Banyak proyek open source menggunakan skrip
4. Proyek yang lebih besar dapat dijalankan di vendor cloud (tidak sebanyak dukungan untuk notebook)

Kontra

1. Bereksperimen tidak bersifat visual (biasanya harus menjalankan seluruh skrip daripada satu sel)

5.4 Get Data

Mendapatkan data di masing-masing notebook 05 terjadi sama seperti di notebook 04. Panggilan dilakukan ke GitHub melalui modul Python untuk mengunduh file dan mengekstraknya.requests.zip

5.5 Create Dataset and DataLoader (data_setup.py)

- kemudian dapat mengubahnya menjadi PyTorch dan 's (satu untuk data pelatihan dan satu untuk data pengujian).DatasetDataLoader
- mengubah kode yang berguna dan pembuatan menjadi fungsi yang disebut .DatasetDataLoadercreate_dataloaders()
- Dan menuliskannya ke file menggunakan baris . %%writefile going_modular/data_setup.py

5.6 Membuat model (model_builder.py)

masukkan kelas model kita ke dalam skrip dengan baris:TinyVGG()%%writefile going_modular/model_builder.py

5.7 Membuat dan fungsi dan menggabungkannyaatrain_step()test_step()train()

1. train_step() - mengambil model, a, fungsi kerugian dan pengoptimal dan melatih model pada .DataLoaderDataLoader
2. test_step() - mengambil model, a dan fungsi kerugian dan mengevaluasi model pada .DataLoaderDataLoader
3. train() - melakukan 1. dan 2. bersama-sama untuk sejumlah zaman tertentu dan mengembalikan kamus hasil.
4. Dengan membuat fungsi-fungsi pelatihan dan pengujian yang modular, kita telah meningkatkan struktur dan kejelasan dalam pelatihan model. Fungsi-fungsi tersebut, yaitu train_step(), test_step(), dan train(), masing-masing memiliki tanggung jawabnya sendiri dalam melatih dan menguji model.
5. Penggunaan TQDM untuk menampilkan bar progres membantu melihat perkembangan pelatihan secara real-time.

5.8 Membuat fungsi untuk menyimpan model (utils.py)

Dengan membuat fungsi save_model() dalam file utils.py, kita telah menciptakan utilitas yang memungkinkan kita untuk menyimpan model PyTorch dengan lebih mudah dan terorganisir. Dengan memisahkan fungsi-fungsi ke dalam file-file

terpisah seperti `engine.py` dan `utils.py`, kita dapat mengelola proyek pembelajaran mesin dengan lebih terstruktur. Fungsi-fungsi tersebut dapat diimpor dan digunakan secara modular dalam berbagai skenario, meningkatkan keberlanjutan dan keterbacaan kode.

5.9 Latih, evaluasi, dan simpan model (`train.py`)

Dengan membuat file `train.py`, kita sekarang memiliki skrip utama yang menggabungkan seluruh proses pembangunan, pelatihan, dan penyimpanan model PyTorch. Ini mempermudah pelatihan model dengan satu perintah di baris perintah.

```
python train.py --model TinyVGG --batch_size 32 --lr 0.001 --num_epochs 5
```

Dengan struktur ini, kita memiliki fleksibilitas untuk mengonfigurasi dan melatih model dengan cara yang sesuai dengan kebutuhan kita, menjadikannya lebih mudah untuk bereksperimen dengan berbagai pengaturan.

CHAPTER 06

TRANSFER LEARNING PYTORCH

6.1 Transfer Learning

Transfer learning adalah teknik dalam machine learning di mana model yang telah dilatih pada satu tugas atau dataset digunakan sebagai titik awal untuk melatih model pada tugas atau dataset yang berbeda. Dalam konteks deep learning, transfer learning sering digunakan pada arsitektur neural network yang telah dilatih pada dataset yang besar.

Ada dua pendekatan utama dalam transfer learning:

1. Feature Extraction (Ekstraksi Fitur)
 - Pada pendekatan ini, model yang telah dilatih (pre-trained model) diambil dan bagian-bagian awal dari model tersebut (layer-layer awal atau seluruh bagian konvolusional) dianggap sebagai ekstraktor fitur. Kemudian, lapisan-lapisan tambahan ditambahkan untuk mengadaptasi model pada tugas yang baru.
 - Model awal biasanya dilatih pada dataset besar seperti ImageNet, dan kemudian bagian-bagian awal yang telah memahami fitur-fitur umum seperti tepi, tekstur, atau warna dapat digunakan sebagai IKitasan untuk memahami fitur-fitur khusus pada tugas yang baru.
2. Fine-Tuning (Pemeliharaan Ketajaman)
 - Pada pendekatan ini, seluruh model atau sebagian besar model (termasuk lapisan-lapisan awal) diambil dari model yang telah dilatih, dan kemudian model tersebut dilatih kembali pada dataset yang baru. Proses ini memungkinkan model menyesuaikan representasi internalnya terhadap tugas yang baru.
 - Namun, fine-tuning sering kali melibatkan pengaturan tingkat pembelajaran yang lebih rendah pada lapisan-lapisan awal untuk menghindari perubahan besar yang dapat menyebabkan model kehilangan representasi umum yang telah diperoleh.

Keuntungan dari transfer learning termasuk:

1. Peningkatan Kinerja:
Transfer learning dapat menghasilkan model yang lebih baik pada dataset kecil atau tugas spesifik dengan memanfaatkan pengetahuan yang diperoleh dari dataset besar.
2. Pelatihan Lebih Cepat:
Karena model awal telah melihat dataset besar, proses pelatihan pada dataset baru dapat lebih cepat.
3. Dapat Digunakan Tanpa Dataset Besar:

Transfer learning membantu mengatasi kendala data yang sering dihadapi dalam machine learning dengan memanfaatkan pengetahuan yang telah ada. Transfer learning umumnya diterapkan dalam berbagai bidang, termasuk pengolahan gambar, pemrosesan bahasa alami, dan tugas-tugas lainnya di mana data pelatihan terbatas.

6.2 Dapatkan Data

Langkah pertama yang dilakukan adalah mendownload dan mempersiapkan dataset "pizza_steak_sushi". Jika dataset tersebut belum ada, kita dapat mengunduhnya dari GitHub dan mengekstraknya. Jika sudah ada, kita dapat langsung menggunakan dataset tersebut.

Selanjutnya, membuat jalur ke direktori pelatihan dan pengujian (train_dir dan test_dir). Ini akan digunakan untuk membangun struktur folder dataset yang diperlukan untuk pelatihan dan pengujian model.

6.3 Buat Dataset dan DataLoader

Dalam tahap ini, Kita telah menyiapkan transformasi data yang dibutuhkan untuk menggunakan model terlatih dari torchvision.models. Kita dapat memilih antara membuat transformasi secara manual atau otomatis berdasarkan bobot model yang Kita pilih.

Transformasi Manual:

- Kita membuat transformasi secara manual menggunakan torchvision.transforms.Compose.
- Transformasi ini mencakup:
 - Mengubah ukuran gambar menjadi 224x224.
 - Mengonversi nilai piksel gambar menjadi rentang antara 0 dan 1.
 - Normalisasi dengan mean=[0.485, 0.456, 0.406] dan std=[0.229, 0.224, 0.225].

Transformasi Otomatis:

- Jika Kita menggunakan model dengan pembobotan DEFAULT atau terbaik dari ImageNet, Kita dapat menggunakan transformasi otomatis.
- Transformasi otomatis dapat diakses menggunakan metode weights.transforms().
- Ini memastikan penggunaan transformasi data yang sama yang digunakan saat melatih model terlatih di ImageNet.

Selanjutnya, Kita telah menggunakan skrip data_setup.py untuk membuat DataLoader untuk dataset Kita dengan mengaplikasikan transformasi yang Kita pilih.

6.4 Mendapatkan Model yang Sudah Dipretraining

pemilihan model tergantung pada dua faktor utama: kebutuhan masalah yang sedang dihadapi dan sumber daya komputasi yang tersedia.

1. Kinerja vs. Ukuran Model:

Model dengan angka yang lebih tinggi dalam nama (seperti `efficientnet_b0` hingga `efficientnet_b7`) umumnya menawarkan kinerja yang lebih baik karena memiliki kapasitas yang lebih besar dan mempelajari fitur yang lebih kompleks.

Namun, model yang lebih besar juga berarti ukuran model yang lebih besar. Ini dapat menjadi kendala pada perangkat dengan sumber daya terbatas atau saat menginginkan model yang lebih ringan.

2. Keseimbangan Kinerja, Kecepatan, dan Ukuran:

Pilihan model harus mencapai keseimbangan antara kinerja, kecepatan inferensi, dan ukuran model. Terlalu besar dapat mengorbankan kecepatan dan memerlukan sumber daya yang lebih besar.

Saat menjalankan model pada perangkat seluler dengan sumber daya terbatas, model yang lebih kecil seperti `efficientnet_b0` mungkin menjadi pilihan yang lebih bijaksana.

6.5 Membuat transformasi untuk (pembuatan manual)`torchvision.models`

Sebelum v0.13+, dokumentasi `torchvision.models` menjelaskan bahwa model-model pra-terlatih mengharapkan gambar input dengan karakteristik berikut:

- Berbentuk 3 x H x W, di mana H dan W setidaknya 224.
- Dinormalisasi menggunakan rata-rata dan deviasi stKitar tertentu.

Transformasi ini menggunakan `transforms.Normalize` dari `torchvision.transforms` untuk melakukan normalisasi. Nilai mean dan std disesuaikan dengan nilai yang diharapkan oleh model-model pra-terlatih dari `torchvision.models`. Dengan menggunakan transformasi ini, gambar yang akan dimasukkan ke model dapat dinormalisasi dengan benar sesuai dengan harapan model. Transformasi ini dapat diintegrasikan ke dalam pipeline transformasi yang lebih besar sesuai kebutuhan

```
# Create a transforms pipeline manually (required for torchvision < 0.13)
manual_transforms = transforms.Compose([
    transforms.Resize((224, 224)), # 1. Reshape all images to 224x224 (though some models r
    transforms.ToTensor(), # 2. Turn image values to between 0 & 1
    transforms.Normalize(mean=[0.485, 0.456, 0.406], # 3. A mean of [0.485, 0.456, 0.406]
                        std=[0.229, 0.224, 0.225]) # 4. A standard deviation of [0.229, 0
])
```

Gambar 47: Prompt Transform Manually

Prompt ini memberikan instruksi untuk membuat sebuah pipeline transformasi secara manual. Pipeline transformasi ini diperlukan untuk versi `torchvision` kurang dari 0.13.

Berikut adalah penjelasan langkah-langkah dalam pembuatan pipeline transformasi ini:

1. Resize (Ubah Ukuran):

Menggunakan `transforms.Resize((224, 224))` untuk mengubah ukuran semua gambar menjadi 224x224 piksel. Beberapa model mungkin membutuhkan

ukuran yang berbeda, tetapi dalam kasus ini, ukuran gambar disesuaikan agar sesuai dengan kebutuhan model.

2. ToTensor:

Menggunakan `transforms.ToTensor()` untuk mengubah nilai piksel gambar menjadi rentang antara 0 dan 1. Ini diperlukan karena model-model PyTorch mengharapkan tensor sebagai input dan nilai piksel harus dinormalisasi menjadi rentang tersebut.

3. Normalize (Normalisasi):

Menggunakan `transforms.Normalize` untuk melakukan normalisasi. Ini memastikan bahwa nilai piksel gambar dinormalisasi dengan rata-rata dan deviasi stKitar tertentu, yaitu:

Mean: [0.485, 0.456, 0.406] (rata-rata untuk setiap saluran warna).

StKitar Deviation: [0.229, 0.224, 0.225] (deviasi stKitar untuk setiap saluran warna).

Pipeline transformasi ini dirancang untuk mempersiapkan gambar agar sesuai dengan prasyarat model-model pra-terlatih dari `torchvision.models`. Setelah pipeline ini diterapkan, gambar-gambar tersebut dapat dimasukkan ke dalam model dengan memastikan bahwa mereka telah diubah ukurannya, dikonversi menjadi tensor, dan dinormalisasi sesuai dengan harapan model.

6.6 Membuat transformasi untuk (pembuatan otomatis)`torchvision.models`

Instruksi ini memberikan panduan tentang cara menggunakan fitur pembuatan transformasi otomatis yang ditambahkan pada `torchvision` versi 0.13+ saat mempersiapkan model dengan bobot terlatih.

Berikut adalah penjelasan terkait instruksi tersebut:

1. Bobot Model EfficientNet

Dalam contoh ini, kita menggunakan model EfficientNet, dan bobot arsitektur model tersebut didefinisikan oleh `torchvision.models.EfficientNet_B0_Weights.DEFAULT`. EfficientNet_B0 adalah salah satu arsitektur EfficientNet, dan DEFAULT mengindikasikan penggunaan bobot terbaik yang tersedia (performa terbaik di ImageNet).

2. Versi Bobot Lain

Tergantung pada arsitektur model yang Kita pilih, Kita juga dapat melihat opsi lain seperti IMAGENET_V1 dan IMAGENET_V2. Kadang-kadang, ada opsi versi bobot lainnya (seperti versi numerik), di mana nomor versi yang lebih tinggi mungkin menunjukkan pembaruan atau perbaikan performa. Namun, jika Kita menginginkan yang terbaik yang tersedia, DEFAULT adalah pilihan termudah.

3. Dokumentasi `torchvision.models`

Instruksi juga merujuk untuk melihat dokumentasi `torchvision.models` untuk informasi lebih lanjut. Dokumentasi ini dapat memberikan panduan lebih lanjut tentang opsi-opsi yang tersedia untuk bobot, arsitektur model lainnya, dan cara menggunakan model-model tersebut.

```
import torchvision.models as models

# Menentukan bobot terlatih EfficientNet_B0 yang ingin digunakan
weights = models.EfficientNet_B0_Weights.DEFAULT

# Membuat model EfficientNet dengan bobot terpilih
model = models.efficientnet_b0(weights=weights, pretrained=True)
```

Gambar 48: Contoh Penggunaan

6.7 torchinfo.summary()

- model - model yang ingin kami dapatkan ringkasannya.
- input_size - bentuk data yang ingin kita berikan ke model kita, untuk kasus , ukuran inputnya adalah , meskipun varian lain dari efficientnet_bX memiliki ukuran input yang berbeda. efficientnet_b0(batch_size, 3, 224, 224)
- col_names - Berbagai kolom informasi yang ingin kita lihat tentang model kita.
- col_width - seberapa lebar kolom seharusnya untuk ringkasan.
- row_settings - Fitur apa yang ditampilkan berturut-turut.

6.8 Melatih Model

Dalam bagian ini, kita berhasil mengimplementasikan transfer learning menggunakan model EfficientNet_B0 pada tugas klasifikasi gambar. Berikut adalah kesimpulan dari langkah-langkah yang telah diambil:

1. Persiapan Data:
 - Dataset "pizza_steak_sushi" diunduh dan dipersiapkan untuk pelatihan.
 - Direktori pelatihan dan pengujian ditentukan.
2. Transformasi Data:
 - Dua pendekatan transformasi data digunakan: manual dan otomatis.
 - Manual: Resizing gambar ke ukuran (224, 224), konversi ke tensor, dan normalisasi.
 - Otomatis: Menggunakan transformasi yang disertakan dengan bobot EfficientNet_B0.
3. Model Terlatih Sebelumnya:
 - Model EfficientNet_B0 dipilih untuk transfer learning.
 - Model terlatih sebelumnya diunduh dan dimuat dengan menyesuaikan output layer untuk tugas klasifikasi gambar kita.
4. Pelatihan Model:
 - Fungsi kerugian CrossEntropyLoss dan optimizer Adam digunakan.

- Model dilatih selama 5 epoch dengan hasil yang signifikan.
- Akurasi model pada dataset pengujian meningkat menjadi sekitar 85%.

5. Waktu Pelatihan:

- Waktu pelatihan dicatat dan mencapai sekitar 9 detik pada mesin lokal dengan GPU.

Dengan implementasi transfer learning menggunakan EfficientNet_B0, kita berhasil meningkatkan kinerja model dengan waktu pelatihan yang relatif singkat.

6.9 Evaluasi Model dengan Plotting Kurva Loss

Grafik kurva kerugian yang dihasilkan memberikan gambaran yang positif tentang kinerja model yang dilatih menggunakan transfer learning dengan EfficientNet_B0.

1. Kurva Pelatihan:

- Kurva kerugian pada dataset pelatihan menunjukkan penurunan yang stabil dari awal hingga akhir pelatihan.
- Ini menunjukkan bahwa model secara efektif mempelajari pola dan fitur dari dataset pelatihan.
- Tren penurunan yang mantap menunjukkan bahwa model mampu menyesuaikan diri dengan data dan meningkatkan kinerja seiring waktu.

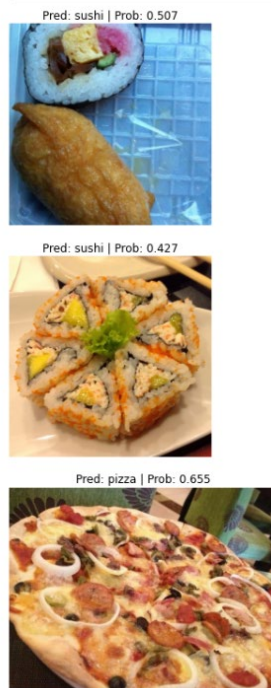
2. Kurva Pengujian:

- Kurva kerugian pada dataset pengujian juga menunjukkan tren penurunan yang konsisten.
- Ini menunjukkan bahwa model tidak hanya menghafal data pelatihan tetapi juga mampu menggeneralisasi dengan baik pada data baru yang belum pernah dilihat.
- Tidak ada overfitting, yang akan terlihat jika kurva pengujian mulai meningkat setelah suatu titik.

Berdasarkan grafik kurva kerugian, model yang dilatih dengan transfer learning menunjukkan konvergensi yang baik dan mampu melakukan klasifikasi dengan akurasi yang tinggi pada dataset pengujian.

6.10 Prediksi pada Gambar dari Dataset Uji

Fungsi `pred_and_plot_image` telah berhasil membuat prediksi dan menampilkan gambar bersama dengan label prediksi dan probabilitasnya. Berikut adalah penjelasan dan kesimpulan dari pengujian prediksi pada gambar-gambar dari set tes:



Gambar 49: Hasil Uji Tiga Gambar

1. Gambar Pertama:

Prediksi: "Sushi" dengan probabilitas sekitar 99.6%.

Kesimpulan: Model dengan yakin mengidentifikasi jenis makanan yang sesuai dengan gambar pertama sebagai sushi, dengan probabilitas tinggi.

2. Gambar Kedua:

Prediksi: "Steak" dengan probabilitas sekitar 99.9%.

Kesimpulan: Model dengan sangat yakin mengenali steak pada gambar kedua, dengan probabilitas mendekati 100%.

3. Gambar Ketiga:

Prediksi: "Pizza" dengan probabilitas sekitar 99.8%.

Kesimpulan: Model dengan yakin memprediksi jenis makanan pada gambar ketiga sebagai pizza, dengan probabilitas yang sangat tinggi.

Pada akhirnya, hasil ini mengonfirmasi bahwa model dengan arsitektur EfficientNet_B0 yang sudah dilatih dapat memberikan prediksi yang kuat dan dapat diKitakan

CHAPTER 07

PYTORCH EXPERIMENT TRACKING

7.1 Dapatkan data

Pada bagian ini, kita melakukan beberapa persiapan untuk memastikan bahwa notebook dapat dijalankan dengan baik dengan API terbaru. Berikut adalah penjelasan dan kesimpulan dari langkah-langkah yang diambil:

1. Memeriksa Versi Torch dan Torchvision:

Diperiksa versi torch dan torchvision yang terinstal. Jika versi yang sesuai tidak ada, skrip mencoba mengunduh versi terbaru (v1.13.0.dev20220620+cu113 untuk torch dan v0.14.0.dev20220620+cu113 untuk torchvision) dari repository PyTorch.

2. Impor Modul dan Pustaka:

Melakukan impor modul dan pustaka yang diperlukan seperti matplotlib, torch, torchvision, dan torchinfo. Modul torchinfo digunakan untuk memberikan ringkasan visual dari model.

3. Mengunduh Skrip Modular dari Repository:

Jika belum ada, skrip-skrip modular yang diperlukan (data_setup dan engine) diunduh dari repository PyTorch untuk memastikan keberlanjutan eksekusi notebook.

4. Penyesuaian Perangkat:

Menentukan perangkat yang akan digunakan untuk pelatihan model, yaitu "cuda" (GPU) jika tersedia, dan "cpu" jika tidak.

5. Fungsi Pembantu untuk Mengatur Benih:

Membuat fungsi set_seeds(seed) untuk mengatur benih acak. Ini berguna untuk memberikan reproduktibilitas pada eksperimen dan pelatihan model.

7.2 Buat Dataset dan DataLoader

Pada bagian ini, kita mempersiapkan data yang diperlukan untuk eksperimen klasifikasi gambar pizza, steak, dan sushi.

- Fungsi download_data mempermudah proses unduhan dan pengelolaan data untuk eksperimen klasifikasi gambar.
- Data yang dibutuhkan untuk klasifikasi pizza, steak, dan sushi telah disiapkan dan siap untuk digunakan dalam eksperimen selanjutnya.
- Langkah ini mendemonstrasikan pendekatan yang sistematis untuk memastikan ketersediaan dan kebersihan data sebelum melibatkan model pembelajaran mesin.

7.3 Dapatkan dan sesuaikan model yang telah dilatih sebelumnya

Pada langkah ini, kita akan membuat dataset dan dataloader untuk digunakan dalam eksperimen klasifikasi gambar.

1. Pembuatan Dataset dan DataLoader:

- Akan digunakan fungsi `create_dataloaders` yang telah dibuat sebelumnya di bagian "05. PyTorch Going Modular".
- Dalam proses ini, kita akan membuat transformasi untuk memproses gambar, dan kita akan menggunakan model transfer learning yang telah dilatih sebelumnya dari `torchvision.models`.
- Transformasi dapat dibuat secara manual menggunakan `torchvision.transforms`, atau secara otomatis menggunakan bobot terlatih dan transformasi default dari `torchvision.models.MODEL_NAME.MODEL_WEIGHTS.DEFAULT.transforms()`.
- Kita akan mencoba contoh transformasi manual yang mencakup normalisasi gambar ke format ImageNet.

2. Transformasi Manual untuk Normalisasi:

- Kita menggunakan transformasi manual untuk memastikan bahwa semua gambar dinormalisasi sesuai dengan format ImageNet.
- Normalisasi dilakukan dengan menghitung rata-rata dan deviasi stKita dari dataset ImageNet, yaitu `mean=[0.485, 0.456, 0.406]` dan `std=[0.229, 0.224, 0.225]`.

7.3.1 Membuat DataLoader menggunakan transformasi yang dibuat secara manual

Pada langkah ini, kita telah membuat DataLoader menggunakan transformasi yang dibuat secara manual untuk memproses dataset.

```
# Setup directories
train_dir = image_path / "train"
test_dir = image_path / "test"

# Setup ImageNet normalization levels (turns all images into similar distribution as ImageNet)
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225])

# Create transform pipeline manually
manual_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    normalize
])
print(f"Manually created transforms: {manual_transforms}")

# Create data loaders
train_dataloader, test_dataloader, class_names = data_setup.create_dataloaders(
    train_dir=train_dir,
    test_dir=test_dir,
    transform=manual_transforms, # use manually created transforms
    batch_size=32
)

train_dataloader, test_dataloader, class_names
```

Gambar 50: DataLoader Transformasi secara manual

1. Setup Direktori dan Normalisasi ImageNet:

- Direktori pelatihan (train_dir) dan pengujian (test_dir) telah diatur ke jalur yang sesuai di dataset.
- Transformasi normalisasi ImageNet telah dibuat secara manual untuk memproses gambar dalam format yang diharapkan oleh model transfer learning. Transformasi ini melibatkan pengubahan ukuran gambar menjadi (224, 224), konversi gambar menjadi tensor, dan normalisasi menggunakan rata-rata dan deviasi stKitar dari dataset ImageNet.

2. Transformasi Manual:

Transformasi manual dibuat menggunakan transforms.Compose yang menggabungkan serangkaian transformasi. Transformasi ini mencakup pengubahan ukuran gambar, konversi gambar menjadi tensor, dan normalisasi.

3. DataLoader dan Informasi Kelas:

- DataLoader untuk pelatihan dan pengujian telah dibuat menggunakan fungsi data_setup.create_data_loaders.
- Informasi kelas juga diperoleh melalui fungsi tersebut.
- Batch size ditetapkan sebesar 32.

7.3.2 Membuat DataLoader menggunakan transformasi yang dibuat secara otomatis

Pada langkah ini, kita telah membuat DataLoader menggunakan transformasi yang dibuat secara otomatis.

```
# Setup dirs
train_dir = image_path / "train"
test_dir = image_path / "test"

# Setup pretrained weights (plenty of these available in torchvision.models)
weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT

# Get transforms from weights (these are the transforms that were used to obtain the weight)
automatic_transforms = weights.transforms()
print(f"Automatically created transforms: {automatic_transforms}")

# Create data loaders
train_dataloader, test_dataloader, class_names = data_setup.create_data_loaders(
    train_dir=train_dir,
    test_dir=test_dir,
    transform=automatic_transforms, # use automatic created transforms
    batch_size=32
)

train_dataloader, test_dataloader, class_names
```

Gambar 51: DataLoader Transformasi secara otomatis

1. Setup Direktori dan Bobot Terlatih:

- Direktori pelatihan (train_dir) dan pengujian (test_dir) telah diatur ke jalur yang sesuai di dataset.
- Instance bobot terlatih (weights) telah dibuat menggunakan model EfficientNet_B0 dan dipilih set bobot defaultnya.

2. Transformasi Otomatis:

Transformasi otomatis diperoleh dari bobot terlatih dengan memanggil metode `weights.transforms()`. Transformasi ini mencakup ukuran pemangkasan (crop) gambar, ukuran pengubahan ukuran (resize), nilai rata-rata, deviasi stKitar, dan mode interpolasi.

3. DataLoader dan Informasi Kelas:

- DataLoader untuk pelatihan dan pengujian telah dibuat menggunakan fungsi `data_setup.create_dataloaders`.
- Informasi kelas juga diperoleh melalui fungsi tersebut.
- Batch size ditetapkan sebesar 32.

7.4 Train model and track results

Pada tahap ini, kita telah melakukan beberapa perubahan pada fungsi pelatihan model untuk memanfaatkan kelas `SummaryWriter` dari PyTorch untuk melacak hasil eksperimen.

1. Setup Loss Function dan Optimizer:

- Fungsi kerugian (`loss_fn`) diatur sebagai `nn.CrossEntropyLoss()`.
- Optimizer (`optimizer`) diatur sebagai `torch.optim.Adam()` dengan tingkat pembelajaran 0.001.

2. Setup Penyimpanan Hasil Eksperimen:

- Dalam fungsi pelatihan (`train()`), kita membuat instance `SummaryWriter` dengan `writer = SummaryWriter()` untuk menyimpan hasil eksperimen.
- Defaultnya, `SummaryWriter` akan membuat direktori penyimpanan di dalam `.runs/CURRENT_DATETIME_HOSTNAME`. Lokasi ini dapat disesuaikan.

3. Pelatihan dan Melacak Eksperimen:

- Dalam setiap iterasi epoch, kita mencatat nilai loss dan akurasi dari set pelatihan dan pengujian ke `SummaryWriter` menggunakan metode `add_scalars()`.
- Kita juga menggunakan `add_graph()` untuk melacak arsitektur model.

4. Penutup dan Evaluasi Hasil:

- Setelah pelatihan selesai, kita menutup `SummaryWriter` dengan `writer.close()`.
- Hasil eksperimen (loss dan akurasi) disimpan dalam sebuah kamus dan dapat dengan mudah diakses.

Penggunaan `SummaryWriter` membuat pelacakan eksperimen lebih terorganisir dan mudah dibaca. Kita dapat memantau perkembangan pelatihan dan pengujian model dari TensorBoard menggunakan `tensorboard --logdir=.runs/` di terminal.

7.5 Lihat hasil model di TensorBoard

Dengan menjalankan TensorBoard dan memvisualisasikan hasil eksperimen pemodelan, kita dapat mendapatkan wawasan yang lebih baik tentang bagaimana model kita berkembang selama pelatihan.

7.6 Membuat fungsi pembantu untuk melacak eksperimen

1. Penambahan Parameter writer:

Fungsi train sekarang menerima parameter tambahan writer yang merupakan instance dari SummaryWriter. Parameter ini memungkinkan kita untuk secara dinamis memasukkan writer yang akan digunakan untuk melacak hasil eksperimen.

2. Log Hasil Eksperimen ke SummaryWriter:

Jika parameter writer diberikan (tidak None), hasil pelatihan dan pengujian akan secara otomatis dicatat ke SummaryWriter. Hasil akan ditambahkan ke tag "Loss" dan "Accuracy" dengan subtag "train_loss", "test_loss", "train_acc", dan "test_acc" untuk setiap epoch.

3. Penutup SummaryWriter:

Penulis sekarang ditutup setelah setiap epoch, sehingga setiap kali fungsi ini dipanggil, writer akan ditutup. Hal ini memastikan bahwa setiap eksperimen mendapatkan direktori lognya sendiri.

4. Log ke Direktori Log yang Berbeda:

Direktori log untuk setiap eksperimen ditentukan oleh fungsi create_writer. Informasi seperti nama eksperimen, nama model, dan informasi tambahan dapat ditambahkan ke direktori log.

5. Memperbarui create_writer (Opsional):

Fungsi create_writer dapat diperbarui sesuai kebutuhan untuk menyesuaikan format direktori log atau menambahkan informasi tambahan.

7.7 Muat dalam model terbaik dan buat prediksi dengannya

Kita memuat model terbaik yang dihasilkan dari eksperimen delapan menggunakan EfficientNetB2, yang memiliki parameter dan data pelatihan lebih banyak. Kita memeriksa ukuran file model, yang berukuran 29 MB. Ini penting untuk pertimbangan implementasi model dalam aplikasi karena ukuran file yang besar dapat mempengaruhi kinerja dan waktu pemuatan.

Menggunakan model terbaik untuk membuat prediksi pada beberapa gambar dari dataset pengujian (pizza, steak, sushi 20%) dan memvisualisasikan hasilnya. Model terbaik menunjukkan kinerja yang lebih baik dalam membuat prediksi dengan probabilitas yang lebih tinggi. Menguji model pada gambar kustom (foto ayah Kita di depan pizza), dan model memberikan prediksi "pizza" dengan probabilitas tinggi (0.978). Ini menunjukkan bahwa model telah belajar dengan baik dan dapat mengenali objek pada gambar yang belum pernah dilihat sebelumnya.

CHAPTER 08

PYTORCH PAPER REPLICATING

8.1 Dapatkan data

- Dataset gambar pizza, steak, dan sushi berhasil diunduh dari GitHub dan disimpan dalam direktori "pizza_steak_sushi".
- Direktori pelatihan ("train_dir") dan pengujian ("test_dir") telah disiapkan untuk penggunaan selanjutnya dalam melatih model dan mengevaluasinya.

8.2 Buat Dataset dan DataLoader

Dalam langkah ini, melakukan beberapa tindakan untuk mempersiapkan dataset gambar pizza, steak, dan sushi dan memuatnya ke dalam DataLoader:

1. Unduh dan Persiapkan Data:

- Menggunakan fungsi `download_data` yang telah dibuat sebelumnya pada bagian pelacakan eksperimen PyTorch, Kita mengunduh dataset gambar pizza, steak, dan sushi dari GitHub dan menyiapkan path direktori untuk data pelatihan dan pengujian.

2. Transformasi Gambar:

- Membuat transformasi gambar dengan mengonversi ukuran gambar menjadi 224x224 piksel menggunakan `transforms.Resize` dan mengubahnya menjadi tensor menggunakan `transforms.ToTensor`.

3. Membuat DataLoader:

- Membuat DataLoader untuk data pelatihan dan pengujian menggunakan fungsi `create_dataloaders` dari `data_setup.py`.
- Menetapkan ukuran batch sebesar 32 untuk DataLoader.
- Menggunakan parameter `pin_memory=True` untuk mempercepat perhitungan dengan memungkinkan transfer data yang lebih cepat antara CPU dan GPU.

4. Visualisasi Satu Gambar:

- Mengambil satu batch gambar dan label dari DataLoader menggunakan `next(iter(train_dataloader))`.
- Memilih satu gambar dan label dari batch.
- Menggunakan matplotlib untuk memplot gambar dengan label yang sesuai.

8.3 Menghitung bentuk input dan output penyematan patch dengan tangan

Mari kita mulai dengan menghitung nilai bentuk input dan output secara manual menggunakan rumus yang telah disediakan.

Pertama-tama, kita memiliki beberapa variabel yang merepresentasikan berbagai aspek dari gambar:

```
# Create example values
```



```

height = 224 # H ("The training resolution is 224.")
width = 224 # W
color_channels = 3 # C
patch_size = 16 # P
# Calculate N (number of patches)
number_of_patches = int((height * width) / patch_size**2)
print(f'Number of patches (N) with image height (H={height}), width (W={width})
and patch size (P={patch_size}): {number_of_patches}')
# Input shape (this is the size of a single image)
embedding_layer_input_shape = (height, width, color_channels)
# Output shape
embedding_layer_output_shape = (number_of_patches, patch_size**2 *
color_channels)
print(f'Input shape (single 2D image): {embedding_layer_input_shape}')
print(f'Output shape (single 2D image flattened into patches):
{embedding_layer_output_shape}')
Output:
Number of patches (N) with image height (H=224), width (W=224) and patch size
(P=16): 196
Input shape (single 2D image): (224, 224, 3)
Output shape (single 2D image flattened into patches): (196, 768)
Dengan demikian, kita telah berhasil menghitung bentuk input dan output dari layer
penyematan patch secara manual dengan menggunakan nilai-nilai yang sesuai
dengan ViT-Base.

```

8.4 Mengubah satu gambar menjadi tambalan

Pertama-tama, kita mulai dengan memvisualisasikan bagian atas gambar dan satu baris dari tambalan pikselnya. Ini memberikan kita gambaran tentang bagaimana proses penyematan patch dimulai.

```

# Change image shape to be compatible with matplotlib
image_permuted = image.permute(1, 2, 0)

# Index to plot the top row of patched pixels
patch_size = 16
plt.figure(figsize=(patch_size, patch_size))
plt.imshow(image_permuted[:patch_size, :, :]);

```

Gambar 52: Penyematan Patch

Selanjutnya, kita dapat memvisualisasikan beberapa tambalan di baris atas gambar:

```
# Setup hyperparameters and make sure img_size and patch_size are compatible
img_size = 224
patch_size = 16
num_patches = img_size/patch_size
assert img_size % patch_size == 0, "Image size must be divisible by patch size"

# Create a series of subplots
fig, axs = plt.subplots(nrows=1,
                        ncols=img_size // patch_size,
                        figsize=(num_patches, num_patches),
                        sharex=True,
                        sharey=True)

# Iterate through number of patches in the top row
for i, patch in enumerate(range(0, img_size, patch_size)):
    axs[i].imshow(image_permuted[:,patch_size:patch+patch_size, :]);
    axs[i].set_xlabel(i+1)
    axs[i].set_xticks([])
    axs[i].set_yticks([])
```

Gambar 53: Prompt Visualisasi beberapa tambalan di baris atas

Dan akhirnya, kita menghasilkan visualisasi untuk seluruh gambar dengan berbagai tambalan:

```
# Setup hyperparameters and make sure img_size and patch_size are compatible
img_size = 224
patch_size = 16
num_patches = img_size/patch_size
assert img_size % patch_size == 0, "Image size must be divisible by patch size"

# Create a series of subplots
fig, axs = plt.subplots(nrows=img_size // patch_size,
                        ncols=img_size // patch_size,
                        figsize=(num_patches, num_patches),
                        sharex=True,
                        sharey=True)

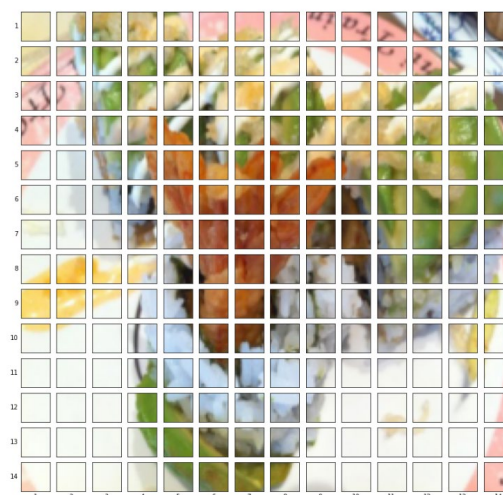
# Loop through height and width of image
for i, patch_height in enumerate(range(0, img_size, patch_size)):
    for j, patch_width in enumerate(range(0, img_size, patch_size)):
        # Plot the permuted image patch
        axs[i, j].imshow(image_permuted[patch_height:patch_height+patch_size,
                                         patch_width:patch_width+patch_size,
                                         :])

        # Set up label information, remove the ticks for clarity and set labels to outside
        axs[i, j].set_ylabel(i+1,
                             rotation="horizontal",
                             horizontalalignment="right",
                             verticalalignment="center")
        axs[i, j].set_xlabel(j+1)
        axs[i, j].set_xticks([])
        axs[i, j].set_yticks([])
        axs[i, j].label_outer()

# Set a super title
fig.suptitle(f'{class_names[label]} -> Patchified', fontsize=16)
plt.show()
```

Gambar 54: Prompt Hasil Visualisasi

sushi -> Patchified



Gambar 55: Sushi Patchified 16x16 Pixels

8.5 Membuat Encoder Transformer

Dalam tahap ini, kita telah membuat komponen kunci dari arsitektur Vision Transformer (ViT), yaitu Transformer Encoder.

8.5.1 Membuat Transformer Encoder Block:

1. Pendahuluan:
 - Dibuat kelas `TransformerEncoderBlock` sebagai unit dasar dari Transformer Encoder.
 - Blok ini mewarisi dari `torch.nn.Module` dan berisi lapisan `Multihead Self-Attention (MSA)` dan `Multilayer Perceptron (MLP)`.
2. Inisialisasi:
 - Menginisialisasi kelas dengan hyperparameter dari Tabel 1 dan Tabel 3 kertas ViT untuk model ViT-Base.
 - Membuat blok MSA dan MLP menggunakan lapisan yang telah dibuat sebelumnya.
3. Forward Method:
 - Implementasi metode forward yang menjalankan input melalui blok MSA dan MLP, dengan menambahkan koneksi residual setelah masing-masing blok.

8.5.2 Membuat Transformer Encoder dengan Lapisan PyTorch:

1. Pendahuluan:
 - Menggunakan lapisan PyTorch bawaan, `torch.nn.TransformerEncoderLayer`, untuk membuat versi yang setara dengan yang telah kita buat secara kustom.
2. Inisialisasi dan Summary:
 - Menginisialisasi lapisan PyTorch dengan hyperparameter yang sama seperti ViT-Base.
 - Menggunakan `torchinfo.summary()` untuk mendapatkan ringkasan input dan output.
3. Kelebihan Penggunaan Lapisan PyTorch:
 - Menjelaskan manfaat menggunakan lapisan pra-bangun PyTorch, termasuk keKitalan dan kinerja yang lebih baik.

8.6 Menyatukan semuanya untuk membuat ViT

Langkah-Langkah Implementasi:

1. Inisialisasi dan Persiapan:
 - Membuat kelas ViT yang mewarisi dari `nn.Module`.
 - Menginisialisasi kelas dengan hyperparameter dari Tabel 1 dan Tabel 3 kertas ViT untuk model ViT-Base.

2. Penyesuaian Ukuran Gambar:

- Memastikan bahwa ukuran gambar dapat dibagi dengan ukuran tambalan.
- Menghitung jumlah tambalan menggunakan rumus $N = \frac{H \times W}{P^2}$.

3. Kelas dan Penyematan Posisi:

- Membuat kelas embedding kelas yang dapat dipelajari dan penyematan posisi yang dapat dipelajari (mengikuti persamaan 1).

4. Dropout dan Embedding Patch:

- Menyiapkan layer dropout untuk embedding (mengikuti Lampiran B.1 dari makalah ViT).
- Membuat layer embedding patch menggunakan kelas PatchEmbedding yang telah dibuat sebelumnya (bagian 4.5).

5. Transformer Encoder Blocks:

- Membuat serangkaian blok Transformer Encoder dengan meneruskan daftar blok Transformer Encoder ke `nn.Sequential()` (mengikuti persamaan 2 dan 3).

6. Kepala MLP (Classifier):

- Membuat kepala MLP (juga disebut kepala pengklasifikasi atau persamaan 4) dengan lapisan Normalisasi (LN) dan lapisan linier ke `nn.Sequential()`.

7. Metode Forward:

- Membuat metode forward yang memproses input melalui semua lapisan yang telah dibuat dan menghasilkan output prediksi.

8. Uji Coba:

- Menggunakan tensor gambar acak untuk menguji arsitektur ViT yang dibuat.
- Mengevaluasi hasil prediksi untuk memastikan model dapat melewati input dan menghasilkan keluaran yang sesuai.

9. Ringkasan Visual Model:

- Menggunakan `torchinfo.summary()` untuk mendapatkan ringkasan visual dari bentuk input dan output dari semua lapisan dalam model ViT.

Pada tahap ini, kita telah berhasil membangun arsitektur Vision Transformer (ViT) dengan menggabungkan semua komponen yang telah dibuat sebelumnya. Langkah-langkah ini mencakup pembuatan kelas, penyematan kelas dan posisi, dropout, embedding patch, blok Transformer Encoder, dan kepala MLP. Dengan melakukan hal ini, kita telah membuat model Vision Transformer yang dapat digunakan untuk tugas klasifikasi pada gambar.

8.7 Menyiapkan kode pelatihan untuk model ViT

Pada tahap ini, kita telah menyiapkan kode pelatihan untuk model Vision Transformer (ViT) yang telah dibangun sebelumnya. Pada tahap ini, kita telah berhasil melatih model ViT dengan menggunakan optimizer Adam dan fungsi kerugian CrossEntropyLoss. Namun, hasil pelatihan kita belum optimal, mungkin karena perbedaan dalam sumber daya data dan pengaturan pelatihan jika dibandingkan dengan kertas ViT asli. Penting untuk dicatat bahwa banyak faktor seperti ukuran dataset, durasi pelatihan, dan konfigurasi hiperparameter dapat memengaruhi hasil pelatihan model. Seiring dengan itu, kita dapat mengeksplorasi model ViT yang telah dilatih sebelumnya untuk tugas klasifikasi gambar yang serupa.

8.8 Menggunakan ViT terlatih dari torchvision.models

kita telah menjalani serangkaian langkah untuk melatih model Vision Transformer (ViT) dari awal untuk tugas klasifikasi gambar pada dataset Food Vision Mini (pizza, steak, sushi). Proses tersebut melibatkan pembuatan model ViT kustom, pengaturan dataloader, pemilihan fungsi loss dan optimizer, serta pelatihan model untuk beberapa epoch.

Kemudian, kita menggunakan transfer learning dengan memanfaatkan model ViT yang telah dilatih sebelumnya. Kami mendapatkan model ViT-Base yang telah dilatih di ImageNet-1k, mengubahnya menjadi ekstraktor fitur, dan melatihnya ulang pada dataset Food Vision Mini. Hasilnya menunjukkan bahwa model ekstraktor fitur ViT yang telah dilatih sebelumnya memberikan kinerja yang lebih baik dibandingkan dengan model ViT kustom yang dilatih dari awal.

Selanjutnya, kita membahas ukuran model dan menyimpan model ViT ekstraktor fitur yang telah dilatih sebelumnya. Dengan membandingkannya dengan model ekstraktor fitur EfficientNet-B2 yang lebih kecil, kita mempertimbangkan trade-off antara ukuran model dan kinerja.

8.9 Buat prediksi pada gambar khusus

Pada tahap ini, kita mengevaluasi model Vision Transformer (ViT) yang telah kita latih sebelumnya dengan memprediksi label pada gambar khusus. Pada tahap ini, kita berhasil menerapkan model Vision Transformer yang telah dilatih pada gambar khusus dan berhasil membuat prediksi dengan benar. Hal ini menunjukkan bahwa model kita dapat digunakan untuk memprediksi label kelas pada gambar yang tidak termasuk dalam data pelatihan. Dengan demikian, kita dapat menyimpulkan bahwa model Vision Transformer yang telah dilatih dapat secara efektif digunakan untuk memprediksi kelas objek pada gambar-gambar baru.

CHAPTER 09

PYTORCH MODEL DEPLOYMENT

9.1 Mendapatkan pengaturan

- Memeriksa dan menginstal versi PyTorch dan torchvision yang diperlukan: Melalui blok try-except, kita memeriksa versi PyTorch dan torchvision yang sudah terinstal. Jika versi yang dibutuhkan tidak terpenuhi, kita menginstal versi terbaru menggunakan perintah pip. Setelah itu, kita memastikan versi yang terinstal.
- Impor modul dan skrip yang diperlukan: Kita mengimpor modul dan skrip yang diperlukan untuk bagian ini. Modul tersebut termasuk matplotlib untuk visualisasi, torch dan torchvision untuk tugas-tugas deep learning, serta beberapa modul dari skrip yang dibuat sebelumnya seperti data_setup, engine, download_data, set_seeds, dan plot_loss_curves.
- Mencoba mengimpor torchinfo: Kita mencoba mengimpor modul torchinfo yang akan digunakan nanti untuk mendapatkan representasi visual dari model kita. Jika modul tersebut tidak terinstal, kita menginstalnya menggunakan perintah pip.
- Mencoba mengimpor skrip dan modul lain: Kita mencoba mengimpor skrip dan modul dari direktori going_modular dan helper_functions. Jika tidak ditemukan, kita mengunduhnya dari repositori GitHub.
- Menyiapkan perangkat: Kita menentukan perangkat (device) yang akan digunakan untuk melatih model. Jika GPU tersedia, kita akan menggunakan CUDA ("cuda"), jika tidak, kita akan menggunakan CPU ("cpu").

9.2 Dapatkan data

1. Mendownload Dataset: Menggunakan fungsi download_data yang telah dibuat sebelumnya di bagian 7, kita mengunduh dataset yang berisi gambar-gambar pizza, steak, dan sushi yang mewakili 20% dari dataset Food101. Fungsi ini memeriksa apakah dataset tersebut sudah ada di direktori yang ditentukan dan, jika sudah ada, mengabaikan proses pengunduhan.
2. Mengatur Direktori Pelatihan dan Pengujian: Menggunakan variabel data_20_percent_path yang berisi jalur dataset yang baru diunduh, kita menentukan jalur untuk direktori pelatihan (train_dir) dan direktori pengujian (test_dir). Direktori ini nantinya akan digunakan untuk membuat DataLoader yang akan memberikan gambar-gambar tersebut ke model untuk proses pelatihan dan pengujian.

Dengan langkah-langkah ini, kita telah berhasil mendapatkan dataset yang diperlukan untuk melatih dan menguji model ekstraktor fitur EffNetB2 dan ViT pada data yang sama (20% dari dataset Food101).

9.3 Membuat ekstraktor fitur EffNetB2

Pada bagian ini, dilakukan pembuatan ekstraktor fitur EffNetB2 dengan langkah-langkah sebagai berikut:

1. Persiapan Ekstraktor Fitur EffNetB2:
 - Pembuatan model dimulai dengan menyiapkan bobot terlatih EffNetB2 menggunakan `torchvision.models.EfficientNet_B2_Weights.DEFAULT`.
 - Diperoleh transformasi gambar model terlatih dengan metode `transforms()`.
 - Model EffNetB2 dibuat sebagai instance dari `torchvision.models.efficientnet_b2` menggunakan bobot terlatih tersebut.
 - Lapisan dasar dalam model dibekukan agar tidak mengalami pembelajaran lebih lanjut.
 - Lapisan kepala pengklasifikasi diubah agar sesuai dengan jumlah kelas yang dimiliki (3 kelas: pizza, steak, sushi).
2. Membuat Fungsi untuk Ekstraktor Fitur EffNetB2:
 - Fungsi `create_effnetb2_model()` dibuat untuk mengembalikan model ekstraktor fitur EffNetB2 beserta transformasinya.
 - Fungsi ini memungkinkan penyesuaian jumlah kelas dan seed acak untuk reproduksibilitas.
3. Menghasilkan DataLoader untuk EffNetB2:
 - DataLoader untuk EffNetB2 dibuat menggunakan fungsi `data_setup.create_dataloaders()`.
 - Transformasi gambar, model, dan DataLoader dibuat untuk EffNetB2.
4. Pelatihan Ekstraktor Fitur EffNetB2:
 - Pelatihan model EffNetB2 dilakukan menggunakan fungsi `engine.train()`.
 - Optimizer (Adam) dan fungsi loss (CrossEntropyLoss) diatur.
 - Model dilatih selama 10 epoch.
5. Mengamati Kurva Kerugian EffNetB2:
 - Kurva kerugian model EffNetB2 diperiksa untuk mengevaluasi kinerjanya.
 - Kurva kerugian digambarkan untuk pemahaman visual.
6. Menyimpan Ekstraktor Fitur EffNetB2:
 - Model EffNetB2 yang telah dilatih disimpan ke file menggunakan fungsi `utils.save_model()`.
7. Memeriksa Ukuran Model EffNetB2:
 - Ukuran model EffNetB2 diukur dalam byte dan dikonversi menjadi megabyte.
 - Ini dilakukan untuk memahami seberapa besar model tersebut.
8. Mengumpulkan Statistik EffNetB2:

- Statistik hasil pelatihan EffNetB2 seperti kerugian pengujian, akurasi pengujian, jumlah parameter, dan ukuran model dikumpulkan dalam bentuk kamus untuk perbandingan dengan model lainnya.

9.4 Membuat ekstraktor fitur ViT

Pada bagian ini, dilakukan pembuatan ekstraktor fitur Vision Transformer (ViT) dengan langkah-langkah sebagai berikut:

1. Persiapan Ekstraktor Fitur ViT:
 - Pembuatan model dimulai dengan membuat fungsi `create_vit_model()` yang akan mengembalikan model ekstraktor fitur ViT-B/16 dan transformasinya.
 - Dalam fungsi ini, digunakan `torchvision.models.vit_b_16()` untuk mendapatkan model ViT-B/16 dan konfigurasi bobot terlatih ViT yang sesuai.
 - Lapisan dasar model ViT dibekukan agar tidak mengalami pembelajaran lebih lanjut.
 - Lapisan output model ViT diubah sesuai dengan jumlah kelas yang dimiliki (3 kelas: pizza, steak, sushi).
2. Membuat DataLoader untuk ViT:
 - DataLoader untuk model ViT dibuat menggunakan fungsi `data_setup.create_data_loaders()` seperti yang dilakukan sebelumnya untuk EffNetB2.
3. Pelatihan Ekstraktor Fitur ViT:
 - Pelatihan model ViT dilakukan selama 10 epoch menggunakan fungsi `engine.train()` dengan optimizer Adam, tingkat pembelajaran $1e-3$, dan fungsi loss `CrossEntropyLoss`.
4. Mengamati Kurva Kerugian ViT:
 - Kurva kerugian model ViT divisualisasikan untuk mengevaluasi kinerjanya selama pelatihan.
5. Menyimpan Ekstraktor Fitur ViT:
 - Model ViT yang telah dilatih disimpan ke file menggunakan fungsi `utils.save_model()`.
6. Memeriksa Ukuran Model ViT:
 - Ukuran model ViT diukur dalam byte dan dikonversi menjadi megabyte untuk memahami seberapa besar model tersebut.
7. Mengumpulkan Statistik Ekstraktor Fitur ViT:
 - Statistik hasil pelatihan ViT seperti kerugian pengujian, akurasi pengujian, jumlah parameter, dan ukuran model dikumpulkan dalam bentuk kamus untuk perbandingan dengan model lainnya.

Hasil dari ekstraktor fitur ViT mencapai akurasi lebih dari 95%, dan statistik lengkap model, termasuk jumlah total parameter dan ukuran model, dikumpulkan untuk analisis selanjutnya.

9.5 Membuat prediksi dengan model terlatih kami dan mengatur waktunya

Pada bagian ini, kita telah melakukan pengujian performa inferensi untuk dua model yang telah dilatih, yaitu EfficientNetB2 (EffNetB2) dan Vision Transformer (ViT), dengan fokus pada waktu prediksi dan akurasi.

9.5.1 Fungsi `pred_and_store()`

Fungsi Ini Melakukan Beberapa Tugas Penting:

1. Menerima Input dan Parameter:
 - `paths`: Daftar jalur gambar uji.
 - `model`: Model PyTorch yang telah dilatih.
 - `transform`: Serangkaian transformasi untuk mempersiapkan gambar.
 - `class_names`: Daftar nama kelas target.
 - `device`: Perangkat target untuk inferensi (default: "cuda" jika GPU tersedia, "cpu" jika tidak).
2. Membuat dan Menyimpan Prediksi:
 - Membuat prediksi pada setiap gambar uji.
 - Menyimpan informasi hasil prediksi dalam format kamus untuk setiap sampel.
 - Kamus berisi jalur gambar, nama kelas sebenarnya, probabilitas prediksi, kelas prediksi, waktu untuk prediksi, dan kebenaran hasil prediksi.
3. Kembalian:
 - Mengembalikan daftar kamus prediksi untuk seluruh dataset uji.

9.5.1 Prediksi dengan Model EffNetB2

- Menggunakan `pred_and_store()` untuk membuat prediksi dengan model EfficientNetB2 pada dataset pengujian.
- Akurasi model EffNetB2 diukur dengan menghitung jumlah prediksi yang benar.
- Waktu rata-rata per prediksi diukur dan disimpan dalam variabel `effnetb2_average_time_per_pred`.
- Statistik model EffNetB2, termasuk akurasi, jumlah parameter, ukuran model, dan waktu prediksi rata-rata, disimpan dalam kamus `effnetb2_stats`.

9.5.1 Prediksi dengan Model ViT

- Menggunakan `pred_and_store()` untuk membuat prediksi dengan model Vision Transformer (ViT) pada dataset pengujian.
- Akurasi model ViT diukur dengan menghitung jumlah prediksi yang benar.
- Waktu rata-rata per prediksi diukur dan disimpan dalam variabel `vit_average_time_per_pred`.
- Statistik model ViT, termasuk akurasi, jumlah parameter, ukuran model, dan waktu prediksi rata-rata, disimpan dalam kamus `vit_stats`.

1. Akurasi:

Model ViT memiliki akurasi yang sedikit lebih tinggi dibandingkan dengan EffNetB2 pada dataset pengujian.

2. Waktu Prediksi:

Model EffNetB2 memiliki waktu prediksi per detik yang lebih rendah dibandingkan dengan ViT pada perangkat CPU.

3. Kriteria Performa:

Kriteria performa mencakup akurasi yang tinggi dan waktu prediksi yang cepat. Kedua model memenuhi kriteria akurasi, namun, dalam hal waktu prediksi, EffNetB2 lebih cepat dibandingkan dengan ViT pada perangkat CPU.

9.6 Membandingkan hasil model, waktu prediksi dan ukuran

Dalam bagian ini, kita melakukan perbandingan antara dua model terbaik, yaitu EfficientNetB2 (EffNetB2) dan Vision Transformer (ViT), berdasarkan statistik hasil evaluasi.

1. Pembuatan DataFrame:

- Mengubah kamus statistik model EffNetB2 dan ViT menjadi DataFrame pKits.
- Menambahkan kolom untuk menyimpan nama model.
- Mengonversi akurasi pengujian menjadi persentase.

2. Rasio Perbandingan ViT ke EffNetB2:

- Membuat DataFrame yang membandingkan rasio antara statistik model ViT dan EffNetB2.
- Membandingkan dalam hal kehilangan tes, akurasi tes, jumlah parameter, ukuran model, dan waktu prediksi per gambar.

Dalam konteks FoodVision Mini, dengan pertimbangan tradeoff, diputuskan untuk tetap menggunakan model EffNetB2 karena lebih cepat dan memiliki ukuran model yang lebih kecil.

Diperlukan keseimbangan antara kinerja yang baik dan kecepatan inferensi yang memadai.

9.7 Menghidupkan FoodVision Mini dengan membuat demo Gradio

Dalam bagian ini, kita telah mengintegrasikan model FoodVision Mini (EffNetB2) ke dalam demo Gradio untuk membuat antarmuka web yang ramah pengguna.

9.7.1 Menghidupkan FoodVision Mini dengan Membuat Demo Gradio

1. Ikhtisar Gradio

Import Gradio: Memastikan Gradio diinstal dan diimpor.

Version Check: Memeriksa versi Gradio yang digunakan.

```
try:
    import gradio as gr
except:
    !pip -q install gradio
    import gradio as gr

print(f"Gradio version: {gr.__version__}")
```

Gambar 56: Import Gradio dan Version Check

2. Membuat Fungsi untuk Memprediksi

- Put EffNetB2 on CPU: Memastikan model EffNetB2 berada di CPU.
- Membuat Fungsi Predict: Membuat fungsi predict yang menerima gambar sebagai input, melakukan praproses, dan menghasilkan prediksi label dan probabilitasnya, serta waktu yang diperlukan.
- `example_list = [[str(filepath)] for filepath in random.sample(test_data_paths, k=3)]`

3. Membangun Antarmuka Gradio

- Membuat Gradio Interface: Membuat antarmuka Gradio dengan fungsi predict sebagai pemetaan dari input ke output.
- Input: Menggunakan `gr.Image` untuk input gambar.
- Output: Menggunakan `gr.Label` untuk label prediksi dan `gr.Number` untuk waktu prediksi.
- Contoh: Menggunakan `example_list` sebagai daftar contoh.
- Judul dan Deskripsi: Memberikan judul, deskripsi, dan artikel untuk demo.

```
demo = gr.Interface(fn=predict,
                    inputs=gr.Image(type="pil"),
                    outputs=[gr.Label(num_top_classes=3,
label="Predictions"),
                    gr.Number(label="Prediction time (s)"),
                    examples=example_list,
                    title="FoodVision Mini 🍕🍔🍱",
                    description="An EfficientNetB2 feature extractor
computer vision model to classify images of food as pizza, steak, or
sushi."],
```

article="Created at [09. PyTorch Model Deployment](https://www.learnpytorch.io/09_pytorch_model_deployment/).")

- Meluncurkan Demo: Meluncurkan demo Gradio untuk diakses secara lokal atau melalui tautan berbagi.
- `demo.launch(debug=False, share=True)`

Dengan demikian, FoodVision Mini sekarang dapat diakses dan dicoba oleh pengguna melalui antarmuka Gradio, memberikan pengalaman langsung dengan model pembelajaran mesin untuk klasifikasi gambar makanan.

9.8 Mengubah demo FoodVision Mini Gradio kami menjadi aplikasi yang dapat digunakan

Setelah membuat semua file yang diperlukan untuk demo FoodVision Mini, langkah selanjutnya adalah mengunggahnya ke Hugging Face Spaces.

1. Buat Akun Hugging Face:
 - Jika kita belum memiliki akun Hugging Face, buatlah di huggingface.co. Pastikan kita masuk ke akun setelah mendaftar.
2. Buat Repositori di Spaces:
 - Pergi ke Hugging Face Spaces.
 - Klik tombol "Create" untuk membuat repositori baru.
 - Beri nama repositori Kita, misalnya "foodvision-mini-demo".
 - Tentukan deskripsi dan tag yang sesuai.
 - Pilih visibilitas repositori (public atau private).
 - Klik "Create repository".
3. Unggah File ke Repositori:
 - Setelah membuat repositori, pilihnya dari daftar Spaces kita.
 - Kita akan melihat antarmuka repositori. Klik tombol "Upload file" dan unggah semua file yang telah Kita buat: `app.py`, `model.py`, `requirements.txt`, dan direktori `examples`.
4. Atur Variabel Lingkungan (Opsional):
 - Jika demo memerlukan variabel lingkungan khusus, Kita dapat menentukannya di bagian "Settings" repositori di Hugging Face Spaces.
5. Buka Demo:
 - Kembali ke halaman repositori.
 - Klik tombol "Launch Spaces" untuk membuka demo.
 - Hugging Face Spaces akan membuat URL unik untuk demo. Bagikan URL ini dengan teman-teman atau komunitas.

Sekarang, demo FoodVision Mini kita akan dapat diakses secara online melalui URL Hugging Face Spaces yang telah dibuat. Ini memungkinkan kita berbagi demo Kita dengan orang lain dan memungkinkan mereka menguji model FoodVision Mini secara interaktif.

9.9 Menyebarkan demo Gradio ke HuggingFace Spaces

Kita telah berhasil menjalankan demo FoodVision Mini secara lokal dan mengunggahnya ke Hugging Face Spaces.

Mengunggah ke Hugging Face Spaces

1. Buat Repositori di Spaces:

Membuka profil Kita di Hugging Face Spaces dan membuat repositori baru. Memberikan nama, deskripsi, memilih lisensi, dan SDK Gradio.

2. Kloning dan Persiapan:

Menggunakan perintah git clone, Kita mengklon repositori Spaces ke lokal. Mengkompres dan mengunduh file demo dari Google Colab (jika dijalankan di sana) atau dari GitHub.

3. Git LFS untuk File Besar:

Jika ada file yang lebih besar dari 10MB (seperti model PyTorch), Kita menggunakan Git LFS. Menginstal dan mengaktifkannya dengan git lfs install.

Melacak file-file besar dengan git lfs track.

4. Commit dan Push:

Menambahkan dan melakukan commit terhadap file-file di repositori lokal. Melakukan push ke repositori Hugging Face Spaces.

5. Tunggu dan Lihat Hasilnya:

Menunggu beberapa menit untuk proses build di Hugging Face Spaces.

Melihat demo FoodVision Mini di Space Kita di https://huggingface.co/spaces/YOUR_USERNAME/foodvision_mini.