REPORT OF PyTorch – FINAL EXAM

Learn PyTorch for Deep Learning: Zero to Mastery

Oleh :

BAGUS MAHARDIKA SANTOSO
1103204028

PRODI S1 TEKNIK KOMPUTER
FAKULTAS TEKNIK ELEKTRO
UNIVERSITAS TELKOM
BANDUNG
2024

CHAPTER 00

PYTORCH FUNDAMENTALS

0.1 Importing PyTorch

pengimporan PyTorch dan pemeriksaan versi di lingkungan Python import torch mengimpor pustaka PyTorch ke dalam lingkungan Python Anda. Ini memberikan akses ke berbagai fungsi dan modul yang disediakan oleh PyTorch untuk keperluan pengolahan tensor dan pembelajaran mesin.

torch.__version__ memeriksa versi PyTorch yang terinstal

0.2 Tensor

- Scalar: Sebuah skalar adalah tensor dimensi nol, yang merupakan angka tunggal.
- Vector: tensor dimensi satu, yang dapat berisi banyak angka.
- Matrix: tensor dimensi dua, mirip dengan vektor tetapi dengan dimensi ekstra.
- Tensor: Tensor tiga dimensi dapat digunakan untuk merepresentasikan struktur data yang lebih kompleks.

0.3 Memanipulasi Tensor (Operasi Tensor)

- Penjumlahan dan Perkalian: menggunakan elemen “+” penjumlahan dan “*” perkalian
- Pengurangan dan Penugasan Kembali
- Penggunaan Fungsi Torch: torch.multiply(tensor, 10)
- Perkalian Matriks

  Perkalian Matriks: torch.matmul(tensor, tensor), melakukan perkalian matriks dengan menggunakan metode torch.matmul(). Ini menunjukkan bahwa hasil perkalian matriks dari tensor [1, 2, 3] dengan dirinya sendiri adalah tensor skalar 14.
- Penggunaan Operator "@" untuk Perkalian Matriks (Opsional): kita juga dapat menggunakan operator @ untuk melakukan perkalian matriks. Namun, perlu diingat bahwa ini tidak direkomendasikan, dan lebih baik menggunakan torch.matmul() karena lebih eksplisit.

Operasi tensor ini adalah bagian penting dari pembelajaran mesin dan jaringan saraf, dan pemahaman mendalam tentang manipulasi tensor sangat penting dalam mengembangkan model yang efektif.

0.4 Menemukan min, max, mean, sum, dll (agregasi)

1. Membuat Tensor

   x = torch.arange(0, 100, 10)

membuat tensor x dengan nilai mulai dari 0, berakhir pada 90, dan dengan langkah 10. Hasilnya adalah tensor: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90].

2. Menemukan Minimum dan Maksimum

   print(f"Minimum: {x.min()}")

   print(f"Maximum: {x.max()}")

   menggunakan metode .min() untuk menemukan nilai minimum dan .max() untuk menemukan nilai maksimum dari tensor. Outputnya adalah:

   Minimum: 0

   Maximum: 90

3. Menemukan Mean dan Sum

   print(f"Mean: {x.type(torch.float32).mean()}")

   print(f"Sum: {x.sum()}")

   Untuk menghitung mean, kita perlu mengonversi tipe data tensor ke float32 menggunakan .type(torch.float32) karena fungsi .mean() memerlukan tipe data float. Kemudian, kita menggunakan .mean() dan .sum() untuk menemukan nilai mean dan sum dari tensor. Outputnya adalah:

   Mean: 45.0

   Sum: 450

4. Posisi Minimum dan Maksimum

   print(f"Index where max value occurs: {x.argmax()}")

   print(f"Index where min value occurs: {x.argmin()}")

   menggunakan metode .argmax() dan .argmin() untuk menemukan indeks dari nilai maksimum dan minimum dalam tensor. Outputnya adalah:

   Index where max value occurs: 8

   Index where min value occurs: 0

0.5 Mengubah tipe data tensor

menggambarkan cara mengubah tipe data tensor menggunakan metode .type() pada tensor PyTorch.

1. Membuat Tensor dan Memeriksa Tipe Data Default

   # Create a tensor and check its datatype

   tensor = torch.arange(10., 100., 10.)

   tensor.dtype

   # Output: torch.float32

   Membuat tensor tensor dengan menggunakan torch.arange() dan kemudian memeriksa tipe datanya dengan atribut .dtype. Tipe data defaultnya adalah torch.float32.

2. Mengubah Tipe Data Tensor

```
# Create a float16 tensor
tensor_float16 = tensor.type(torch.float16)
tensor_float16
# Output: tensor([10., 20., 30., 40., 50., 60., 70., 80., 90.], dtype=torch.float16)
```

membuat tensor baru tensor_float16 dengan mengubah tipe datanya menjadi torch.float16 menggunakan metode .type(). Hasilnya adalah tensor dengan tipe data yang diinginkan.

3. Mengubah Tipe Data Menjadi Integer

```
# Create an int8 tensor
tensor_int8 = tensor.type(torch.int8)
tensor_int8
# Output: tensor([10, 20, 30, 40, 50, 60, 70, 80, 90], dtype=torch.int8)
```

Kita dapat mengubah tipe data menjadi integer, seperti dalam contoh di atas yang menggunakan torch.int8.

0.6 Membentuk kembali, menumpuk, meremas, dan melepaskan pemerasan

- Membentuk Ulang Tensor: Menggunakan torch.reshape()
- Menggunakan torch.view(): Kita juga dapat menggunakan metode .view() untuk mengubah tampilan tensor tanpa mengubah data asli. Hasilnya sama dengan penggunaan .reshape().
- Menumpuk Tensor: menumpuk tensor x empat kali di sepanjang dimensi baru (dimensi 0) menggunakan torch.stack().
- Meremas dan Melepaskan Pemerasan
- meremas tensor x_reshaped untuk menghapus dimensi ekstra yang berukuran 1 menggunakan torch.squeeze().
- Menggunakan torch.unsqueeze(): melepaskan pemerasan pada tensor x_squeezed dengan menambahkan dimensi ekstra pada indeks tertentu menggunakan torch.unsqueeze().

0.7 Tensor PyTorch & NumPy

Interaksi dengan NumPy:

- NumPy dan PyTorch berinteraksi dengan baik, dan Anda dapat mengonversi tensor PyTorch ke array NumPy dan sebaliknya menggunakan torch.from_numpy() dan .numpy().
- Perubahan pada satu objek dapat tercermin pada objek lainnya jika keduanya berbagi memori.

CHAPTER 01

PyTorch Workflow Fundamentals

1.1 PyTorch Workflow

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
   - Bangun Loop Pelatihan
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, vidio, 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.

Lalu Pisahkan data menjadi set pelatihan dan tes: Disini mumpunyai 40 sampel untuk pelatihan dan 10 sampel untuk pengujian. Selanjutnya buat visualisasi dari data yang sudah di pisahkan. Dan disini yang tadinya data berupa angka yang kita buat secara random menjadi data yang sudah di olah menjadi visualisasi garis lurus.

1.3 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 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)
- Propagasi Maju (Forward Propagation)
- Perhitungan Kerugian (Loss Computation)
- Propagasi Mundur (Backward Propagation)
- Optimasi (Optimization)
- Evaluasi (Optional)

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.

1.4 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:

   Fungsi .eval() digunakan untuk mengatur model dalam mode evaluasi. Hal ini memastikan bahwa selama prediksi, model tidak melakukan perhitungan tambahan seperti pelatihan, normalisasi dropout, dan lain-lain.Lakukan Propagasi Maju (Forward Propagation):

3. Atur Konteks Mode Inferensi

   menggunakan torch.inference_mode() sebagai manajer konteks untuk membuat prediksi. Ini mengoptimalkan beberapa perhitungan untuk meningkatkan kecepatan.

0.8 Menyimpan dan memuat model PyTorch

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

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

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 Buatkan Data Klasifikasi dan siapkan

kita pertama-tama membuat data klasifikasi menggunakan metode make_circles dari Scikit-Learn. Data ini memiliki dua fitur (X1 dan X2) dan label biner (0 atau 1).

1. Membuat Data Klasifikasi: Kita membuat 1000 sampel menggunakan metode make_circles, yang menghasilkan dua lingkaran dengan titik-titik berwarna berbeda. Noise diberikan untuk menambahkan sedikit variasi pada titik-titik.
2. Menampilkan 5 Data Pertama: Menampilkan 5 nilai pertama dari fitur (X) dan label (y) untuk memberikan gambaran tentang bagaimana data terlihat.
3. Membuat DataFrame:
4. Data diubah ke dalam DataFrame pandas untuk kemudahan visualisasi dan pemrosesan.
5. Menghitung Jumlah Label: Memeriksa jumlah nilai dalam setiap kelas (0 atau 1) untuk memastikan bahwa dataset seimbang.
6. Visualisasi Data: Melakukan visualisasi scatter plot dari data dengan warna sesuai dengan labelnya. Dari plot ini, kita dapat melihat bahwa dua lingkaran dengan warna yang berbeda telah dibuat, dan kita perlu membuat model untuk memisahkan keduanya.

2.2 Membangun Model

- Menyiapkan Kode Agnostik Perangkat:
  device = "cuda" if torch.cuda.is_available() else "cpu"
  Menentukan perangkat tempat model dan data akan berjalan. Jika GPU tersedia, kita akan menggunakan GPU ("cuda"), dan jika tidak, kita akan menggunakan CPU ("cpu").
- Membuat kelas model
  Membuat kelas model yang mengandung dua lapisan linier. Lapisan pertama (layer_1) mengubah fitur input dari 2 menjadi 5, dan lapisan kedua (layer_2) mengubahnya menjadi 1 (output yang diharapkan). Metode forward mendefinisikan langkah-langkah perhitungan pada saat inferensi.
- Menginstansi Model
  Menginstansiasi model dan mengirimkannya ke perangkat yang telah ditentukan (CPU atau GPU).
- Membuat Prediksi
  Membuat prediksi menggunakan model pada data uji. Namun, pada tahap ini, model belum dilatih, jadi prediksinya mungkin tidak bermakna.

Model yang telah dibangun memiliki dua lapisan linier yang diatur untuk menangani input dengan 2 fitur dan menghasilkan output dengan 1 fitur. Selain itu, model telah diinstansiasi pada perangkat yang telah ditentukan.

2.3 Train Model

Dalam langkah-langkah pelatihan model klasifikasi menggunakan PyTorch, berikut adalah langkah-langkah untuk mengubah output raw model menjadi label prediksi yang dapat dibandingkan dengan label sebenarnya:

- Output Logit dari Model:
  Output ini dikenal sebagai logit, hasil langsung dari perhitungan model pada data uji. Logit ini adalah nilai yang belum dimodifikasi atau diaktivasi.
- Menggunakan Fungsi Sigmoid:
  Fungsi aktivasi sigmoid digunakan untuk mengubah logit menjadi nilai probabilitas antara 0 dan 1. Ini memberikan kita perkiraan seberapa yakin model terhadap setiap kelas.
- Mengubah Probabilitas Menjadi Label:
  Fungsi ini membulatkan nilai probabilitas ke 0 atau 1, menciptakan label prediksi. Jika probabilitas lebih besar atau sama dengan 0,5, diprediksi sebagai kelas 1; jika kurang dari 0,5, diprediksi sebagai kelas 0.
- Membandingkan Dengan Label Sebenarnya:
  Memeriksa kesamaan antara label prediksi dan label sebenarnya. squeeze() digunakan untuk menghilangkan dimensi tambahan yang mungkin ada dalam tensor.
  Outputnya menunjukkan bahwa prediksi model (setelah dibulatkan) cocok dengan label sebenarnya.

2.4 Membangun Loop pelatihan dan pengujian

Perhatian utama di sini adalah bahwa performa model tampaknya tidak meningkat seiring berjalannya waktu dan epoch. Beberapa hal yang bisa diperhatikan dari hasil tersebut adalah:

- Akurasi Tetap di Sekitar 50%: Akurasi pada data pelatihan dan pengujian tetap sekitar 50%. Ini menunjukkan bahwa model tidak mempelajari pola yang signifikan dari data dan mungkin hanya melakukan prediksi yang tidak lebih baik dari acak.
- Kehilangan (Loss) Tidak Berkurang: Meskipun model dilatih selama beberapa epoch, kehilangan (loss) pada data pelatihan tidak menunjukkan penurunan yang signifikan. Ini bisa menjadi tanda bahwa model belum menemukan konsep atau pola yang berguna dalam data pelatihan.
- Pengujian dan Pelatihan Memberikan Hasil Serupa: Akurasi dan kehilangan pada data pelatihan dan pengujian tampaknya memberikan hasil yang serupa. Ini bisa menjadi pertanda bahwa model mungkin tidak memahami dengan baik hubungan dalam data dan memprediksi dengan cara yang serupa pada kedua set data.
- Mungkin Diperlukan Pengaturan Lebih Lanjut: Ada kemungkinan bahwa parameter atau arsitektur model yang digunakan mungkin tidak cocok untuk data tersebut. Mungkin diperlukan eksplorasi lebih lanjut untuk menemukan parameter atau arsitektur yang lebih sesuai.
- Memeriksa Data dan Visualisasi: Penting untuk memeriksa dan memvisualisasikan data untuk memahami apakah ada pola atau karakteristik tertentu yang bisa membantu meningkatkan kinerja model.

2.5 Buat Prediksi dan evaluasi model

Masalah yang terlihat pada visualisasi decision boundary memberikan beberapa wawasan tentang mengapa model tidak berkinerja dengan baik:

- Batas Keputusan yang Sederhana: Model yang dibuat memiliki batas keputusan yang linear (garis lurus) di antara kelas 0 dan 1. Namun, karena data yang dibuat menggunakan make_circles (lingkaran), batas keputusan yang linear tidak dapat memisahkan dengan baik antara kedua kelas. Oleh karena itu, model tidak dapat menangkap pola yang rumit dari data.
- Kurangnya Kemampuan Model untuk Menangkap Pola Non-linear: Model yang digunakan (dengan 2 layer linear) mungkin tidak memiliki kapasitas yang cukup untuk menangkap pola non-linear dari data lingkaran. Sebagai solusi, model yang lebih kompleks atau struktur jaringan saraf yang lebih dalam mungkin diperlukan untuk menangani data dengan pola yang lebih kompleks.
- Pentingnya Menyesuaikan Model dengan Data: Dalam kasus ini, model yang dipilih dan arsitektur jaringan saraf mungkin tidak cocok dengan karakteristik data. Oleh karena itu, memilih model yang sesuai dengan jenis data (seperti yang dicerminkan dalam visualisasi) dapat meningkatkan kinerja.
- Eksplorasi Model dan Hyperparameter: Diperlukan eksplorasi lebih lanjut pada arsitektur model dan hyperparameter untuk meningkatkan kinerja pada tugas klasifikasi ini. Mungkin perlu mencoba model dengan lebih banyak lapisan, unit tersembunyi, atau menggunakan model yang telah terbukti efektif untuk tugas klasifikasi non-linear.

2.6 Meningkatkan model (dari perspektif model)

Meskipun telah ditambahkan lapisan tambahan dan model dilatih lebih lama, tampaknya model masih tidak mampu menangkap pola data yang kompleks.

1. Kapasitas Model Tidak Cukup: Meskipun menambahkan lapisan dan unit tersembunyi bisa meningkatkan kapasitas model, terkadang hal ini belum tentu cukup. Model yang digunakan mungkin masih tidak memiliki kemampuan untuk menangkap pola non-linear yang rumit dari data lingkaran.
2. Fungsi Aktivasi yang Tidak Sesuai: Mengubah fungsi aktivasi atau menggunakan fungsi aktivasi non-linear tertentu, seperti ReLU (Rectified Linear Unit) pada lapisan tersembunyi, dapat membantu model mempelajari pola yang lebih kompleks. Beberapa fungsi aktivasi memiliki keunggulan dalam menangkap representasi non-linear dari data.
3. Eksplorasi Hyperparameter yang Lebih Mendalam: Selain menambahkan lapisan dan unit tersembunyi, eksplorasi hyperparameter lainnya seperti tingkat pembelajaran, fungsi kerugian, atau metode pengoptimalan mungkin diperlukan untuk meningkatkan performa model. Pengaturan hyperparameter yang kurang tepat dapat menyebabkan model sulit untuk konvergen.

4. Pertimbangan Fungsi Kerugian: Fungsi kerugian yang digunakan (dalam hal ini, BCEWithLogitsLoss) mungkin memerlukan penyesuaian. Beberapa fungsi kerugian lebih sesuai untuk tugas klasifikasi tertentu, terutama ketika berurusan dengan data yang memiliki karakteristik khusus.
5. Perlu Menggunakan Fungsi Aktivasi di Setiap Lapisan Tersembunyi: Menambahkan fungsi aktivasi non-linear seperti ReLU pada setiap lapisan tersembunyi dapat membantu model menangkap lebih banyak informasi dan pola dari data.

2.7 Melatih model dengan non-lineritas

- Disini kita menambahkan pelatihan model dengan non-lineritas. Dengan menambahkan fungsi aktivasi non-linear, seperti ReLU atau sigmoid, kita memberikan model kemampuan untuk memodelkan pola non-linear dalam data. Sebelumnya, model tanpa fungsi aktivasi non-linear hanya dapat membuat prediksi linear atau menggunakan garis lurus untuk memisahkan kelas-kelas pada data.
- Fungsi aktivasi non-linear memberikan fleksibilitas pada model untuk menangkap pola yang lebih kompleks, seperti data lingkaran pada kasus ini. Jika kita hanya menggunakan model dengan lapisan linear tanpa fungsi aktivasi non-linear, model tersebut tidak dapat menangkap struktur lingkaran dan cenderung menghasilkan prediksi yang buruk.
- Dengan menambahkan fungsi ReLU atau sigmoid, model dapat mempelajari representasi non-linear dari data. Fungsi-fungsi aktivasi ini memperkenalkan kompleksitas ke dalam model, memungkinkannya untuk mengatasi masalah non-linear dan meningkatkan performa pada data yang memiliki struktur non-linear.

2.8 Membuat data klasifikasi kelas multi

1. Membuat Data Multi-Kelas: Menggunakan metode make_blobs dari Scikit-Learn, data multi-kelas dibuat dengan jumlah kelas (NUM_CLASSES) sebanyak 4, jumlah fitur (NUM_FEATURES) sebanyak 2, dan beberapa parameter lainnya seperti cluster_std dan random_state yang mempengaruhi sebaran data.
2. Mengubah Data Menjadi Tensor: Data yang telah dibuat diubah dari format NumPy array menjadi tensor PyTorch menggunakan torch.from_numpy untuk fitur (X_blob) dan label (y_blob). Fitur dikonversi menjadi tensor float, sementara label dikonversi menjadi tensor LongTensor karena ini adalah masalah klasifikasi multi-kelas.
3. Pemisahan Data: Data dibagi menjadi set pelatihan dan pengujian menggunakan train_test_split dari Scikit-Learn. Pada contoh ini, 80% data digunakan sebagai set pelatihan dan 20% sebagai set pengujian.
4. Visualisasi Data: Data ditampilkan dalam bentuk scatter plot dengan warna yang mewakili kelasnya. Gambar ini memberikan gambaran visual tentang sebaran data dalam dua dimensi.

2.9 Membangun model klasifikasi multi-kelas di PyTorch

Disini kita berfokus pada pembuatan model untuk menangani masalah klasifikasi multi-kelas di PyTorch.

1. Pengecekan Perangkat (Device-Agnostic Code): Kode dimulai dengan pengecekan ketersediaan GPU (cuda) dan menentukan perangkat yang akan digunakan (GPU jika tersedia, jika tidak, CPU). Ini memungkinkan fleksibilitas penggunaan GPU atau CPU tergantung pada ketersediaan perangkat keras.
2. Definisi Kelas Model: Kelas model, disebut BlobModel, dibuat sebagai subclass dari nn.Module. Kelas ini memiliki tiga parameter utama: input_features (jumlah fitur input), output_features (jumlah kelas/output yang diinginkan), dan hidden_units (jumlah neuron pada setiap lapisan tersembunyi). Konstruktor (__init__) menginisialisasi lapisan-lapisan linear menggunakan nn.Sequential.
3. Instansiasi Model dan Pemindahan ke Perangkat Tertentu: Sebuah instansiasi dari BlobModel dibuat dengan menggunakan parameter yang telah ditentukan (jumlah fitur input, jumlah kelas, dan jumlah neuron tersembunyi). Model ini kemudian dipindahkan ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).
4. Tampilan Model: Model ditampilkan untuk memberikan pandangan struktur lapisan-lapisan linear yang telah dibuat.

CHAPTER 03
PYTORCH COMPUTER VISION

3.1 Computer Vision libraries in PyTorch

Disini kita melakukan import library yang akan digunakan di modul ini seperti library torch dan torchvision dan juga nemabahkan 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

- Disini kita akan mendownload dataset yang sudah disediakan sebelumnya dan dataset tersebut berupa data fashion.
- Kita 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

Disini kita melakukan visualisasi data Sepatu dengan menggunakan library matplotlib.pyplot as plt dan membuatkannya menjadi dua dengan warna yang berbeda yaitu green dan grey.

3.3 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.
- Length of train dataloader: 1875 batches of 32, Length of test dataloader: 313 batches of 32
  1. Dataloader pelatihan (train dataloader) memiliki panjang 1875 batch, dan setiap batch memiliki ukuran 32.
  2. Dataloader pengujian (test dataloader) memiliki panjang 313 batch, dan setiap batch juga memiliki ukuran 32.

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: (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].

- Setelah itu saya melakukan 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 pembanding dalam pembangunan model lebih kompleks.
- Setelah itu saya membuat loss function dan evaluasi metric agar kita dapat melihat akurasi yang dihasilkan pada saat melakukan training.
- 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 menunjukan 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).

3.7 Membuat Prediksi dan Mendapatkan Hasil Model 0

Disini saya akan membuat prediksi dan mencoba mendapatkan hasil dari model yang sudah saya training.

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.

3.8 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. Untuk melakukannya, kami akan memanfaatkan nn. Conv2d() dan nn. MaxPool2d() lapisan dari .torch.nn

3.9 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

3.10 Membuat dan Menilai Prediksi Acak dengan Model Terbaik

Disini kita melakukan penilaian prediksi acak dengan model terbaik membuat plot visualizes.

3.11 Membuat Matriks Konfusi untuk Evaluasi Lebih Lanjut

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

3.12 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).

4.2 data preparation

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

4.3 Transforming data

Mengubah data dengan torchvision.transforms, torchvision.transforms berisi banyak metode pra-bangun untuk memformat gambar, mengubahnya menjadi tensor dan bahkan memanipulasinya untuk augmentasi data.

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

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.

4.5 Bentuk Lain dari Transformasi (Augmentasi Data)

Disini kita melakukan tranformasi dengan bentuk lain dengan parameter utama yaitu, 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.

4.7 Exploring loss curves

4.6.1 Underfitting

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

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

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%.

- Potensi Overfitting atau Underfitting:

  Dengan melihat hasil tersebut, sulit untuk dengan pasti menyimpulkan apakah model mengalami overfitting atau underfitting.

4.9 Compare model results

Disini kita akan membandingkan hasil model yang sudah kita buat. visualisasi hasil pelatihan dan evaluasi dua model (Model 0 dan Model 1) berdasarkan dataframe model_0_df dan model_1_df. Grafik-garafik yang dihasilkan mencakup train loss, test loss, train accuracy, dan test accuracy untuk setiap model.

Train Loss (Subplot 1):

- Grafik menunjukkan tren penurunan train loss seiring berjalannya epoch. Model 1 tampaknya memiliki train loss yang lebih rendah daripada Model 0.

Test Loss (Subplot 2):

- Grafik menunjukkan tren test loss yang umumnya turun seiring dengan epoch. Model 1 juga tampak memiliki test loss yang lebih rendah dibandingkan dengan Model 0.

Train Accuracy (Subplot 3):

- Grafik menunjukkan tren peningkatan train accuracy seiring berjalannya epoch. Sepertinya Model 1 memiliki train accuracy yang lebih tinggi daripada Model 0.

Test Accuracy (Subplot 4):

- Grafik menunjukkan tren peningkatan test accuracy seiring dengan epoch. Model 1 juga tampak memiliki test accuracy yang lebih tinggi dibandingkan dengan Model 0.

Analisis Keseluruhan:

- Secara keseluruhan, Model 1 tampaknya memberikan kinerja yang lebih baik daripada Model 0 berdasarkan tren train loss, test loss, train accuracy, dan test accuracy.
- Model yang baik harus mampu mengurangi loss (train dan test) seiring berjalannya waktu dan meningkatkan akurasi.

CHAPTER 05
PYTORCH GOING MODULAR

5.1 Going Modular

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

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 menulisnya 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

kita dapat mengimpornya menggunakan:

```
import torch

# Import model_builder.py

from going_modular import model_builder

device = "cuda" if torch.cuda.is_available() else "cpu"


# Instantiate an instance of the model from the "model_builder.py" script

torch.manual_seed(42)

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

5.7 Membuat dan fungsi dan menggabungkannyatrain_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.

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.

1. Fungsi save_model:

   Menerima tiga parameter:

   - model: Model PyTorch yang akan disimpan.
   - target_dir: Direktori tempat model akan disimpan.
   - model_name: Nama file untuk menyimpan model. Harus berakhir dengan ".pt" atau ".pth".

   Membuat direktori target jika belum ada.

   Membuat jalur penyimpanan model.

   Menyimpan state_dict dari model ke jalur yang ditentukan.

2. Penggunaan Fungsi:

   Jika kita ingin menggunakan fungsi ini, cukup impor modul utils.py dan panggil save_model dengan argumen yang sesuai.

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

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,

CHAPTER 06
TRANSFER LEARNING PYTORCH

6.1 Dapatkan Data

- Langkah pertama yang lakukan adalah mendownload dan mempersiapkan dataset "pizza_steak_sushi".
- Selanjutnya, membuat jalur ke direktori pelatihan dan pengujian (train_dir dan test_dir).

6.2 Buat Dataset dan DataLoader

Dalam tahap ini, Kita telah menyiapkan transformasi data yang dibutuhkan untuk menggunakan model terlatih dari torchvision.models.

Transformasi 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 menggunakan model dengan pembobotan DEFAULT atau 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.

6.3 Mendapatkan Model yang Sudah Dipretraining

pemilihan model tergantung pada dua faktor utama:

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.

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.

6.4 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

Berikut langkah-langkah dalam pembuatan pipeline transformasi:

1. Resize (Ubah Ukuran)
2. ToTensor:
   Menggunakan transforms.ToTensor() untuk mengubah nilai piksel gambar menjadi rentang antara 0 dan 1.
3. Normalize (Normalisasi):
   Menggunakan transforms.Normalize untuk melakukan normalisasi.

Pipeline transformasi ini dirancang untuk mempersiapkan gambar agar sesuai dengan prasyarat model-model pra-terlatih dari torchvision.models.

6.5 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.

- Bobot Model EfficientNet
  - kita menggunakan model EfficientNet, dan bobot arsitektur model tersebut didefinisikan oleh
  - torchvision.models.EfficientNet_B0_Weights.DEFAULT.

6.6 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.

6.7 Melatih Model

Dalam bagian ini, kita menggunakan model EfficientNet_B0 pada tugas klasifikasi gambar.

1. Persiapan Data:
    - Dataset "pizza_steak_sushi" diunduh dan dipersiapkan untuk pelatihan.
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. Pelatihan Model:
    - Fungsi kerugian CrossEntropyLoss dan optimizer Adam digunakan.
    - Model dilatih selama 5 epoch.
    - Akurasi model pada dataset pengujian meningkat menjadi sekitar 85%.
4. Waktu Pelatihan:
    - Waktu pelatihan dicatat dan mencapai sekitar 9 detik dengan GPU.

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

6.8 Evaluasi Model dengan Plotting Kurva Loss

Grafik kurva kerugian yang dihasilkan memberikan gambaran yang positif tentang kinerja model.

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.
2. Kurva Pengujian:
    - Kurva kerugian pada dataset pengujian juga menunjukkan tren penurunan yang konsisten. Tidak ada overfitting, yang akan terlihat jika kurva pengujian mulai meningkat setelah suatu titik.

berdasarkan grafik kurva kerugian, menunjukkan konvergensi yang baik dan mampu melakukan klasifikasi dengan akurasi yang tinggi pada dataset pengujian.

6.9 Prediksi pada Gambar dari Dataset Uji

Berikut adalah penjelasan dan kesimpulan dari pengujian prediksi pada gambar-gambar dari set tes:

1. Gambar Pertama:

Prediksi: "Sushi" dengan probabilitas sekitar 99.6%.

2. Gambar Kedua:

   Prediksi: "Steak" dengan probabilitas sekitar 99.9%.

3. Gambar Ketiga:

   Prediksi: "Pizza" dengan probabilitas sekitar 99.8%.

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

CHAPTER 07
PYTORCH EXPERIMENT TRACKING

7.1 Dapatkan data

Pada bagian ini, kita melakukan beberapa persiapan:

1. Memeriksa Versi Torch dan Torchvision:
2. Impor Modul dan Pustaka:

   Melakukan impor modul dan pustaka yang diperlukan seperti matplotlib, torch, torchvision, dan torchinfo.
3. Mengunduh Skrip Modular dari Repository:
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.

- 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.
   - 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 transformasi manual yang mencakup normalisasi gambar ke format ImageNet.

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.

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 standar dari dataset ImageNet.
2. Transformasi Manual:

   Transformasi manual dibuat menggunakan transforms.Compose yang menggabungkan serangkaian transformasi.
3. DataLoader dan Informasi Kelas:
   - DataLoader untuk pelatihan dan pengujian telah dibuat menggunakan fungsi data_setup.create_dataloaders.
   - 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.

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().
3. DataLoader dan Informasi Kelas:
   - DataLoader untuk pelatihan dan pengujian telah dibuat menggunakan fungsi data_setup.create_dataloaders.
   - Batch size ditetapkan sebesar 32.

7.4 Train model amd track results

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

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().

7.5 Lihat hasil model di TensorBoard

kita dapat mendapatkan wawasan yang lebih baik tentang bagaimana model kita berkembang selama pelatihan.

7.6 Membuat fungsi pembantu untuk melacak eksperimen

- Penambahan Parameter writer:

Parameter ini memungkinkan kita untuk secara dinamis memasukkan writer yang akan digunakan untuk melacak hasil eksperimen.

- 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.

- Penutup SummaryWriter:

Hal ini memastikan bahwa setiap eksperimen mendapatkan direktori lognya sendiri.

- Log ke Direktori Log yang Berbeda:

Direktori log untuk setiap eksperimen ditentukan oleh fungsi create_writer.

- 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.

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 soerang lelaki 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:
   - 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.

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. Selanjutnya, kita dapat memvisualisasikan beberapa tambalan di baris atas gambar. Dan akhirnya, kita menghasilkan visualisasi untuk seluruh gambar dengan berbagai tambalan:

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:

- 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).
- 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.
- 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:

- Menggunakan nn.TransformerEncoderLayer, untuk membuat versi yang setara.
- Menginisialisasi lapisan PyTorch dengan hyperparameter yang sama seperti ViT-Base.
- Menggunakan torchinfo.summary() untuk mendapatkan ringkasan input dan output.

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 kertas ViT untuk model ViT-Base.
2. Penyesuaian Ukuran Gambar:
3. Kelas dan Penyematan Posisi:
4. Dropout dan Embedding Patch:
   - Menyiapkan layer dropout untuk embedding
   - Membuat layer embedding patch menggunakan kelas PatchEmbedding yang telah dibuat sebelumnya.
5. Transformer Encoder Blocks:
   - Membuat serangkaian blok Transformer Encoder dengan meneruskan daftar blok Transformer Encoder ke nn.Sequential()
6. Kepala MLP (Classifier):
   - Membuat kepala MLP dengan lapisan Normalisasi (LN) dan lapisan linier ke nn.Sequential().
7. Metode Forward:
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.

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.

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.

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. kita berhasil menerapkan model Vision Transformer yang telah dilatih pada gambar khusus dan berhasil membuat prediksi dengan benar..

CHAPTHER 09

PYTORCH MODEL DEPLOYMENT

9.1 Mendapatkan pengaturan

- Kita mengimpor modul dan skrip yang diperlukan untuk bagian ini, termasuk matplotlib untuk visualisasi, torch dan torchvision.
- Mencoba mengimpor torchinfo.
- Mencoba mengimpor skrip dan modul lain
- Kita menentukan perangkat (device) yang akan digunakan untuk melatih model.

9.2 Dapatkan data

1. Mendownload Dataset, kita mengunduh dataset yang berisi gambar-gambar pizza, steak, dan sushi yang mewakili 20% dari dataset Food101.
2. 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).

9.3 Membuat ekstraktor fitur EffNetB2

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

1. Persiapan Ekstraktor Fitur EffNetB2:
   - menyiapkan bobot terlatih EffNetB2 menggunakan torchvision.models.EfficientNet_B2_Weights.DEFAULT.
   - transformasi gambar model terlatih dengan metode transforms().
   - Model EffNetB2 dibuat sebagai instance dari torchvision.models.efficientnet_b2
   - 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(), 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 dan fungsi loss diatur.

- Model dilatih selama 10 epoch.

5. Mengamati Kurva Kerugian EffNetB2:
   - 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 diukur dalam byte dan dikonversi menjadi megabyte.
8. Mengumpulkan Statistik EffNetB2:

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()
   - Dalam fungsi ini, digunakan torchvision.models.vit_b_16() untuk mendapatkan model ViT-B/16 dan konfigurasi bobot terlatih ViT yang sesuai.
   - 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_dataloaders()
3. Pelatihan Ekstraktor Fitur ViT:
   - Pelatihan model ViT dilakukan selama 10 epoch
4. Mengamati Kurva Kerugian ViT:
5. Menyimpan Ekstraktor Fitur ViT:
   - Model ViT yang telah dilatih disimpan ke file menggunakan fungsi utils.save_model().
6. Memeriksa Ukuran Model ViT:
7. Mengumpulkan Statistik Ekstraktor Fitur ViT:

Hasil dari ekstraktor fitur ViT mencapai akurasi lebih dari 95%

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()

Pada bagian ini Melakukan Beberapa Tugas

1. Menerima Input dan Parameter
2. Membuat dan Menyimpan Prediksi:
3. Mengembalikan daftar kamus prediksi untuk seluruh dataset uji.

9.5.2 Prediksi dengan Model EffNetB2

- Menggunakan pred_and_store() untuk membuat prediksi dengan model EfficientNetB2 pada dataset pengujian.
- 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.

9.5.3 Prediksi dengan Model ViT

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.

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)

1. Pembuatan DataFrame
2. Rasio Perbandingan ViT ke EffNetB2

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

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.

2. Membuat Fungsi untuk Memprediksi
    - Put EffNetB2 on CPU: Memastikan model EffNetB2 berada di CPU.
    - Membuat Fungsi Predict
    - 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.
    - 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.

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:
2. Buat Repositori di Spaces:
    - Pergi ke Hugging Face Spaces.
    - Klik tombol "Create" untuk membuat repositori baru.
    - Beri nama repositori Kita
    - Tentukan deskripsi
    - 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. 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.

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:
2. Kloning dan Persiapan:

   Menggunakan perintah git clone, Kita mengklon repositori Spaces ke lokal.

   Mengkompres dan mengunduh file demo.
3. Git LFS untuk File Besar:

   Jika ada file yang lebih besar dari 10MB, Kita menggunakan Git LFS.

   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.