Muhammad Nur Maajid_1103228145

Modul ke-9 ini, yang berjudul "Penyebaran Model PyTorch," menkitai Proyek Milestone 3 dari FoodVision Mini kita. Sejauh ini, proyek kita hanya memungkinkan akses model PyTorch kepada kita sendiri. Namun, sekarang kita akan membawa FoodVision Mini kita ke ranah publik dan membuatnya dapat diakses oleh orang lain.

Apa itu penyebaran model pembelajaran mesin?

Penyebaran model pembelajaran mesin adalah proses membuat model pembelajaran mesin Kita dapat diakses oleh orang atau sesuatu yang lain. Ini bisa berupa seseorang yang berinteraksi dengan model Kita atau sesuatu yang lain seperti program atau model lain yang berinteraksi dengan model pembelajaran mesin Kita.

Mengapa kita perlu menyebarluaskan model pembelajaran mesin?

Penyebaran model merupakan langkah penting setelah pelatihan karena meskipun kita dapat mendapatkan gambaran bagus tentang kinerja model melalui pengujian, kita tidak pernah tahu sejauh mana model akan berkinerja saat dilepaskan ke lingkungan yang sesungguhnya. Interaksi orang-orang yang belum pernah menggunakan model Kita seringkali akan mengungkapkan kasus-kasus uji yang tidak terpikirkan selama pelatihan.

Langkah-langkah dalam penyebaran model:

1. **Tempat Penyebaran (Deployment Location):** Pertanyaan utama di sini adalah apakah model akan ditempatkan di perangkat itu sendiri (on-device) atau di cloud. Masing-masing memiliki kelebihan dan kekurangan. On-device bisa sangat cepat dan menjaga privasi, tetapi terbatas pada daya komputasi dan penyimpanan perangkat. Di cloud, memiliki daya komputasi tanpa batas, tetapi biaya dan latensi jaringan bisa menjadi masalah.

2. **Bagaimana Model Akan Berfungsi (How it's Going to Function):** Pertanyaan berikutnya adalah apakah prediksi model akan dikembalikan secara langsung (real-time) atau secara periodik (batch). Ini menentukan apakah model akan memberikan prediksi segera atau dalam beberapa waktu tertentu.

Berbagai Cara Penyebaran Model Pembelajaran Mesin:
Beberapa opsi untuk menyebarluaskan model pembelajaran mesin termasuk on-device (misalnya menggunakan Google's ML Kit atau Apple's Core ML), di cloud (menggunakan layanan seperti AWS Sagemaker atau Google Cloud's Vertex AI), atau melalui API dengan alat seperti FastAPI atau TorchServe.

Rencana Modul 09: Penyebaran Model FoodVision Mini
Modul ini akan membahas langkah-langkah konkrit untuk menyebarluaskan model FoodVision Mini. Langkah-langkahnya mencakup eksperimen untuk membandingkan dua model terbaik kita (EffNetB2 dan ViT), membuat ekstraktor fitur untuk keduanya, membuat prediksi dan membandingkan hasilnya, serta mengimplementasikan aplikasi demo menggunakan Gradio dan menyebarkannya melalui Hugging Face Spaces. Tujuannya adalah untuk mendeploy model dengan kinerja 95%+ akurasi dan kecepatan inferensi real-time di atas 30 FPS.

0. Persiapan

- Mengimpor skrip Python seperti data_setup.py dan engine.py dari modul PyTorch yang telah dibuat sebelumnya.
- Mengunduh direktori going_modular dari repositori pytorch-deep-learning jika belum ada.
- Memastikan keberadaan paket torchinfo dan mengunduhnya jika belum tersedia.
- Memastikan keberadaan torchvision v0.13+.
- Memastikan keberadaan versi torch 1.12+ dan torchvision 0.13+. Jika tidak memenuhi syarat, menginstal versi malam.

1. Mendapatkan Data

- Mengunduh dataset 20% dari Food101 yang terdiri dari gambar pizza, steak, dan sushi.
- Mempersiapkan jalur direktori untuk data latih dan uji.

2. Rencana Eksperimen FoodVision Mini Deployment

- Membuat model FoodVision Mini yang memiliki kinerja tinggi dan cepat.
- Target kinerja: akurasi 95%+, kecepatan ~30FPS.
- Menekankan kecepatan, dengan lebih memilih model yang berkinerja 90%+ pada ~30FPS daripada model berkinerja 95%+ pada 10FPS.
- Menggunakan model terbaik dari bagian sebelumnya: EffNetB2 feature extractor dan ViT-B/16 feature extractor.

3. Membuat EffNetB2 Feature Extractor

- Mengatur berat pra-pelatihan EffNetB2 sebagai torchvision.models.EfficientNet_B2_Weights.DEFAULT.
- Mendapatkan transformasi gambar pra-pelatihan EffNetB2.
- Membuat model EffNetB2 dengan mengggunakan berat pra-pelatihan dan membekukan lapisan dasar.
- Mengganti kepala klasifikasi sesuai dengan jumlah kelas yang diinginkan.
- Membuat fungsi **create_effnetb2_model** untuk membuat model EffNetB2 dengan parameter yang dapat disesuaikan.
- Membuat DataLoader untuk EffNetB2 dengan menggunakan fungsi **create_dataloaders**.
- Melatih model EffNetB2 dengan optimisasi, fungsi kerugian, dan DataLoader.
- Melihat kurva kerugian EffNetB2.

3.1. Fungsi Pembuatan EffNetB2 Feature Extractor

Dibuat fungsi **create_effnetb2_model()** untuk membuat EffNetB2 feature extractor secara lebih terstruktur dan dapat digunakan kembali.

3.2. Pembuatan DataLoaders untuk EffNetB2

DataLoader untuk EffNetB2 dibuat menggunakan fungsi **data_setup.create_dataloaders()** dengan batch size 32 dan transformasi menggunakan **effnetb2_transforms**.

3.3. Pelatihan EffNetB2 Feature Extractor

Dilakukan pelatihan model EffNetB2 menggunakan optimizer Adam, fungsi kerugian CrossEntropyLoss, dan DataLoader yang telah dibuat. Pelatihan dilakukan selama 10 epoch.

3.4. Inspeksi Kurva Kerugian EffNetB2

Dilakukan inspeksi kurva kerugian model EffNetB2 setelah pelatihan.

3.5. Penyimpanan EffNetB2 Feature Extractor

Model EffNetB2 yang telah dilatih disimpan dalam file untuk penggunaan selanjutnya menggunakan fungsi **utils.save_model()**.

3.6. Pengecekan Ukuran EffNetB2 Feature Extractor

Ukuran model EffNetB2 yang disimpan diukur untuk mengevaluasi kecocokan dengan kriteria kecepatan FoodVision Mini.

3.7. Pengumpulan Statistik EffNetB2 Feature Extractor

Statistik hasil pelatihan model EffNetB2 seperti loss, akurasi, ukuran model, dan jumlah parameter dikumpulkan dalam sebuah dictionary untuk perbandingan dengan model

selanjutnya. Jumlah total parameter dihitung menggunakan **torch.numel()** pada parameter model EffNetB2.

4. Creating a ViT feature extractor

- Membuat ViT feature extractor menggunakan **torchvision.models.vit_b_16()**.
- Fungsi **create_vit_model** dibuat untuk membuat model ViT-B/16 dan transformasinya.
- Heads layer ViT disebut "heads" daripada "classifier".
- Model ViT dibekukan (frozen) kecuali lapisan classifier yang dapat di-train.
- Model ViT memiliki lebih banyak parameter daripada model EffNetB2.

4.1. Pembuatan DataLoaders untuk ViT

- DataLoader untuk ViT dibuat menggunakan fungsi **data_setup.create_dataloaders**.

4.2. Pelatihan Pemisah Fitur ViT

- Model ViT dilatih selama 10 epochs dengan optimizer Adam dan loss function CrossEntropyLoss.

4.3. Pemeriksaan Kurva Loss ViT

- Loss curves model ViT divisualisasikan menggunakan fungsi **plot_loss_curves**.

4.4. Penyimpanan Pemisah Fitur ViT

- Model ViT disimpan ke file menggunakan fungsi **utils.save_model**.

4.5. Pengecekan Ukuran Pemisah Fitur ViT

- Ukuran model ViT diperiksa dengan menggunakan **pathlib.Path.stat**.

4.6. Pengumpulan Statistik Pemisah Fitur ViT

- Jumlah parameter pada model ViT dihitung.
- Statistik model ViT dikumpulkan dalam dictionary.

5. Membuat Prediksi dengan Model yang Telah Dilatih dan Mengukur Waktu

- Fungsi **pred_and_store** dibuat untuk membuat dan menyimpan prediksi.
- Prediksi dilakukan pada dataset uji dengan model EffNetB2 dan ViT.
- Hasil prediksi diubah menjadi DataFrame untuk analisis.

5.1. Pembuatan Fungsi untuk Membuat Prediksi pada Dataset Pengujian

- Fungsi **pred_and_store** dibuat untuk membuat dan menyimpan prediksi pada dataset uji.

- Prediksi dilakukan satu per satu untuk mencerminkan penggunaan model pada perangkat dengan satu gambar.

5.2. Membuat dan Mengukur Prediksi dengan EffNetB2
- Prediksi dilakukan pada dataset uji dengan model EffNetB2.

- Prediksi dievaluasi dengan menghitung akurasi dan waktu rata-rata per prediksi.

5.3. Membuat dan Mengukur Prediksi dengan ViT
- Prediksi dilakukan pada dataset uji dengan model ViT.

- Prediksi dievaluasi dengan menghitung akurasi dan waktu rata-rata per prediksi.

6. Membandingkan Hasil Model, Waktu Prediksi, dan Ukuran
- Statistik model EffNetB2 dan ViT diubah menjadi DataFrame untuk perbandingan.

- Perbandingan dilakukan dengan membagi nilai ViT dengan EffNetB2 untuk menemukan rasio perbedaan antar model.

6.1. Memvisualisasikan tradeoff kecepatan vs. kinerja
- Membuat scatter plot untuk membandingkan waktu prediksi dan akurasi antara EffNetB2 dan ViT.

- Ukuran titik pada plot menunjukkan ukuran model.

Memberikan judul, label, dan memberikan anotasi model pada plot. Gradio adalah cara cepat dan menyenangkan untuk mendemo model machine learning dengan antarmuka web yang ramah.

Gradio menciptakan antarmuka dari input ke output dengan membuat instance dari **gr.Interface(fn, inputs, outputs)** di mana **fn** adalah fungsi Python yang memetakan input ke output.

Gradio memiliki banyak opsi input dan output yang dikenal sebagai "Components" untuk berbagai tipe data seperti gambar, teks, video, data tabular, audio, dan lainnya.

7. Membuat Demo FoodVision Mini dengan Gradio
7.1. Gradio Overview:
7.2. Membuat Fungsi untuk Memetakan Input dan Output:

Membuat fungsi **predict** yang melakukan transformasi dan prediksi menggunakan model EfficientNetB2 pada suatu gambar.

Fungsi **predict** akan mengembalikan prediksi (label dan probabilitas) serta waktu yang diperlukan untuk prediksi.

Membuat daftar gambar contoh untuk digunakan dalam demo Gradio.

Contoh berisi daftar file path gambar dari direktori pengujian.

Membuat antarmuka Gradio untuk menampilkan workflow: input gambar -> transformasi -> prediksi dengan EfficientNetB2 -> output: prediksi, probabilitas, waktu yang diperlukan.

Menggunakan **gr.Interface()** untuk membuat antarmuka dengan parameter yang sesuai, seperti input gambar dan output label prediksi serta waktu prediksi.

Meluncurkan demo Gradio untuk memvisualisasikan FoodVision Mini.

Hugging Face Spaces adalah sumber daya yang memungkinkan hosting dan berbagi aplikasi machine learning.

Spaces memungkinkan berbagi dan mendemo model-machine learning dengan mudah.

Untuk mendeploy demo Gradio, semua file terkait disimpan dalam satu direktori.

Struktur direktori mencakup model, script aplikasi, contoh gambar, dan file requirements.

Membuat direktori **demos/foodvision_mini/** untuk menyimpan file aplikasi FoodVision Mini.

Membuat direktori **examples/** untuk menyimpan contoh gambar yang akan digunakan dalam demo.

Memindahkan model EffNetB2 yang telah dilatih ke direktori aplikasi FoodVision Mini.

7.3. Membuat Daftar Gambar Contoh:
7.4. Membangun Antarmuka Gradio:
8. Mengubah Demo FoodVision Mini Gradio Menjadi Aplikasi yang Dapat Dideploy
8.1. Apa Itu Hugging Face Spaces:
8.2. Struktur Aplikasi Gradio yang Dideploy:
8.3. Membuat Direktori untuk Menyimpan File Aplikasi:
8.4. Membuat Direktori Contoh Gambar untuk Demo:
8.5. Memindahkan Model EffNetB2:

Membuat script **model.py** yang berisi fungsi **create_effnetb2_model** untuk membuat model EfficientNetB2 dan transformasinya.

Membuat script **app.py** yang menyatukan bagian-bagian utama untuk membuat demo Gradio.

Menyiapkan model, fungsi prediksi, dan mengatur antarmuka Gradio.

Membuat file **requirements.txt** untuk mencantumkan semua dependensi yang diperlukan untuk demo FoodVision Mini.

8.6. Membuat Script Python untuk Model (model.py):
8.7. Membuat Script Python untuk Aplikasi Gradio (app.py):
8.8. Membuat File Requirements (requirements.txt):
9. Menerapkan Aplikasi FoodVision Big ke Hugging Face Spaces

- Membahas cara mengunggah aplikasi FoodVision Mini ke Hugging Face Spaces.
- Menjelaskan dua opsi utama untuk mengunggah ke Hugging Face Space: melalui antarmuka web Hugging Face atau melalui baris perintah/terminal.
- Menyertakan bonus opsi penggunaan pustaka huggingface_hub untuk berinteraksi dengan Hugging Face.
- Memerlukan pendaftaran akun Hugging Face untuk menyimpan model.

9.1. Mengunduh File Aplikasi FoodVision Mini

- Memeriksa file demo dalam folder demos/foodvision_mini menggunakan perintah !ls.
- Menyusun file untuk diunggah ke Hugging Face dengan mengompresnya menjadi folder zip.
- Menyediakan perintah untuk mengunduh file zip jika dijalankan di Google Colab.

9.2. Menjalankan Demo FoodVision Mini Secara Lokal

- Memberikan instruksi untuk menguji aplikasi secara lokal setelah mengunduh file zip.
- Langkah-langkah mencakup unzip file, membuat lingkungan, mengaktifkan lingkungan, menginstal dependensi, dan menjalankan aplikasi.
- Menunjukkan contoh URL lokal tempat aplikasi dapat diakses.

9.3. Mengunggah ke Hugging Face

- Menjelaskan langkah-langkah mengunggah aplikasi FoodVision Mini ke Hugging Face menggunakan Git workflow.

- Meminta pengguna untuk mendaftar akun Hugging Face dan membuat Space baru.

- Menyertakan link menuju contoh Space yang telah dibuat.

10. Membuat FoodVision Big

- Merangkum pekerjaan yang telah dilakukan sejauh ini dan mengusulkan langkah selanjutnya, yaitu menciptakan FoodVision Big.

- FoodVision Big memiliki 101 kelas makanan berbeda, berbeda dengan FoodVision Mini yang memiliki 3 kelas.

10.1. Membuat Model dan Transforms untuk FoodVision Big

- Menggunakan model EfficientNetB2 untuk mengenali 101 kelas makanan.

- Menunjukkan cara membuat ringkasan model menggunakan torchinfo.

10.2. Mendapatkan Data untuk FoodVision Big

- Menggunakan torchvision.datasets.Food101() untuk mendapatkan dataset Food101.

- Menyiapkan transformasi untuk data pelatihan dan pengujian.

10.3. Membuat Subset dari Dataset Food101 untuk Eksperimen Cepat

- Opsional: Membuat subset dari Food101 dataset dengan membagi dataset menjadi 20% data pelatihan dan 20% data pengujian.

10.4. Mengubah Dataset Food101 Menjadi DataLoader

- Mengubah dataset FoodVision Big menjadi DataLoader untuk pelatihan dan pengujian.

10.5. Melatih Model FoodVision Big

- Melatih model FoodVision Big dengan mengatur optimizer, loss function, dan melatih model selama lima epoch.

- Menunjukkan hasil pelatihan dan akurasi pengujian.

10.6. Memeriksa Kurva Kerugian Model FoodVision Big

- Menampilkan kurva kerugian untuk memeriksa overfitting atau underfitting pada model.

10.7. Menyimpan dan Memuat Model FoodVision Big

- Menyimpan model FoodVision Big untuk digunakan nanti.

- Memuat kembali model untuk memastikan dapat dilakukan.

10.8. Memeriksa Ukuran Model FoodVision Big

- Memeriksa ukuran model FoodVision Big dan membandingkannya dengan FoodVision Mini.

11. Mengubah Model FoodVision Big Menjadi Aplikasi Terimplementasi

11.1. Membuat Struktur File dan Menyiapkan Contoh Gambar

- Membuat struktur folder untuk demo FoodVision Big.
- Mengunduh gambar contoh ("pizza-dad") dan memindahkannya ke direktori contoh aplikasi.

11.2. Menyimpan Nama Kelas Food101 ke dalam File (class_names.txt)

- Menyimpan nama kelas Food101 ke dalam file teks "class_names.txt".

11.3. Mengubah Model FoodVision Big Menjadi Skrip Python (model.py)

- Membuat skrip Python "model.py" yang dapat menginisialisasi model EffNetB2 beserta transformasinya.

11.4. Mengubah Aplikasi Gradio FoodVision Big Menjadi Skrip Python (app.py)

- Membuat skrip Python "app.py" untuk aplikasi Gradio FoodVision Big.
- Menyesuaikan impor dan pengaturan nama kelas.
- Menyiapkan model dan transformasi, serta fungsi prediksi.
- Menyiapkan antarmuka Gradio dengan judul, deskripsi, dan artikel yang sesuai.

11.5. Membuat File Persyaratan untuk FoodVision Big (requirements.txt)

- Membuat file "requirements.txt" yang berisi daftar dependensi untuk menjalankan aplikasi FoodVision Big.

11.6. Mengunduh File Aplikasi FoodVision Big

- Menggabungkan semua file yang diperlukan untuk aplikasi FoodVision Big ke dalam file zip.
- Mengunduh file zip aplikasi FoodVision Big.

11.7. Menerapkan Aplikasi FoodVision Big ke HuggingFace Spaces

- Membuat Hugging Face Space baru untuk aplikasi FoodVision Big dengan SDK Gradio.
- Mengunggah file aplikasi ke Hugging Face Space melalui Git dan Git LFS.
- Menunggu beberapa menit untuk proses pembangunan dan membuat aplikasi dapat diakses secara online.