Zero to Mastery Learn PyTorch for Deep Learning Course Summary

00. PyTorch Fundamentals

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Bagian: Introduction to PyTorch

1. Apa itu PyTorch?

- PyTorch adalah framework machine learning dan deep learning open source.

2. Penggunaan PyTorch:

- Memungkinkan manipulasi dan pemrosesan data.
- Memungkinkan penulisan algoritma machine learning dengan menggunakan kode Python.

3. Penggunaan di Industri dan Penelitian:

- Digunakan oleh perusahaan teknologi besar seperti Meta (Facebook), Tesla, dan Microsoft.
- Digunakan dalam riset kecerdasan buatan oleh perusahaan seperti OpenAI.
- Contoh penggunaan di Tesla untuk model computer vision pada mobil otonom.

4. Keunggulan PyTorch:

- Disukai oleh peneliti machine learning.
- Pada Februari 2022, PyTorch adalah framework deep learning paling banyak digunakan di Papers With Code.
- Mengatasi GPU acceleration untuk menjalankan kode lebih cepat.

Bagian: Introduction to Tensors

1. Apa itu Tensor?

- Tensor adalah blok dasar fundamental dari machine learning.
- Tugas tensor adalah merepresentasikan data secara numerik.

2. Pengantar Tensors:

- Tensor dapat merepresentasikan berbagai jenis data, misalnya, gambar dapat direpresentasikan sebagai tensor dengan bentuk [3, 224, 224] (saluran warna, tinggi, lebar).
- Tensors memiliki dimensi, contohnya, tensor gambar memiliki tiga dimensi: saluran warna, tinggi, dan lebar.

3. Membuat Tensors:

- Scalar: Tensors dengan satu angka (dimensi nol).
- Vector: Tensors satu dimensi yang dapat berisi banyak angka.
- Matrix: Tensors dua dimensi yang fleksibel dalam merepresentasikan data.
- Tensor: Tensors n-dimensi yang dapat merepresentasikan hampir segala sesuatu.

4. Manipulasi Tensors:

- Mendapatkan informasi dari tensors menggunakan ndim (jumlah dimensi) dan shape (bentuk tensor).
- Penanganan dimensi, dan contoh membuat tensor 3D.

5. Tensors Acak:

- Pembuatan tensors dengan angka acak menggunakan **torch.rand()**.

6. Tensors Berisi Nol dan Satu:

- Membuat tensors yang berisi nol atau satu dengan **torch.zeros()** dan **torch.ones()**.

7. Rangkaian dan Tensors Serupa:

- Membuat tensor berdasarkan range angka menggunakan **torch.arange()**.
- Membuat tensor yang sama dengan tensor lain menggunakan **torch.zeros_like()** dan **torch.ones_like()**.

8. Jenis Data Tensors:

- Terdapat berbagai jenis data tensor seperti **torch.float32**, **torch.float16**, **torch.int8**, dll.
- Precision datatypes berkaitan dengan tingkat detail yang digunakan untuk menggambarkan sebuah angka.

9. Issues yang Umum:

- Masalah umum termasuk masalah dimensi, jenis data, dan perangkat (device) ketika berinteraksi dengan tensors.

10. Device dan Dtype:

- Perhatikan masalah device dan dtype (tipe data) ketika bekerja dengan tensors, karena PyTorch menyukai konsistensi antara tensors.

Mendapatkan Informasi dari Tensors

1. Atribut Umum:

- Tiga atribut umum yang sering digunakan untuk mendapatkan informasi tentang tensors:
    - **shape:** Bentuk atau dimensi tensor.
    - **dtype:** Jenis data yang digunakan untuk menyimpan elemen-elemen dalam tensor.
    - **device:** Perangkat tempat tensor disimpan (biasanya GPU atau CPU).

Manipulasi Tensors (Operasi Tensor)

1. Operasi Dasar:

- Operasi dasar melibatkan penambahan, pengurangan, dan perkalian.
- Nilai dalam tensor tidak berubah kecuali jika direassign.

2. Reassign Tensor:

- Tensor dapat diubah nilainya dengan mereassign variabel tensor.

3. Fungsi Built-in:

- PyTorch menyediakan fungsi built-in seperti **torch.mul()** dan **torch.add()** untuk operasi dasar pada tensors.

4. Operasi Element-wise:

- Operasi element-wise melibatkan pengalihan operasi pada setiap elemen tensor.

5. Matrix Multiplication:

- Perkalian matriks adalah operasi umum dalam machine learning.
- PyTorch menyediakan fungsi **torch.matmul()** untuk melakukan perkalian matriks.

6. Catatan Penting:

- Pemahaman operasi matriks sangat penting dalam deep learning.
- Penggunaan fungsi built-in disarankan karena lebih cepat daripada implementasi manual.

7. Waktu Komputasi:

- Penggunaan **torch.matmul()** lebih cepat daripada mengimplementasikan operasi matriks secara manual dengan loop.

Umpan Balik dan Koreksi (Feedback and Corrections)

1. **Umpan Balik:**
   - Tutorial ini memberikan pemahaman yang baik tentang manipulasi tensors di PyTorch.
   - Penjelasan tentang operasi matriks, penggunaan **torch.matmul()**, dan visualisasi di http://matrixmultiplication.xyz/ memberikan tambahan konteks yang berguna.
   - Pengenalan tentang error bentuk (shape errors) dan cara menanggulanginya dengan mentransposisi matriks juga penting untuk dipahami.
2. **Koreksi/Penjelasan Tambahan:**
   - Pada bagian "Change tensor datatype," penjelasan tentang perbedaan precision dalam komputasi bisa lebih diuraikan secara lebih rinci agar lebih mudah dipahami oleh pembaca yang mungkin tidak memiliki latar belakang matematika/komputasi yang kuat.
   - Penambahan contoh atau penjelasan lebih lanjut pada bagian "Reshaping, stacking, squeezing and unsqueezing" bisa memberikan tambahan wawasan bagi pembaca.

Indexing pada Tensors (Pemilihan Data dari Tensors):

Pada PyTorch, Kita dapat menggunakan indeks untuk memilih data tertentu dari tensors. Indeks berjalan dari dimensi luar ke dimensi dalam. Misalnya, untuk tensor **x** dengan bentuk **(1, 3, 3)**, Kita dapat menggunakan indeks seperti **x[0][0][0]** untuk memilih nilai tertentu. Penggunaan **:** memungkinkan Kita untuk memilih semua nilai dalam suatu dimensi, dan koma (,) digunakan untuk menambah dimensi.

Contoh:

- **x[:, 0]** akan mengembalikan semua nilai dalam dimensi pertama dan indeks pertama dari dimensi kedua.
- **x[:, :, 1]** akan mengembalikan semua nilai dari dimensi pertama dan kedua, tetapi hanya indeks kedua dari dimensi ketiga.

Indeks pada tensors memungkinkan Kita memanipulasi dan memilih data sesuai kebutuhan.

Tensors PyTorch & NumPy:

PyTorch memudahkan interaksi dengan NumPy, suatu pustaka komputasi numerik populer dalam Python. Terdapat dua metode utama yang dapat digunakan untuk konversi antara NumPy dan PyTorch:

1. **torch.from_numpy(ndarray)**: Mengubah array NumPy menjadi tensor PyTorch.
2. **torch.Tensor.numpy()**: Mengubah tensor PyTorch menjadi array NumPy.

Contoh penggunaan:

- **torch.from_numpy(array)** mengubah array NumPy menjadi tensor PyTorch.
- **tensor.numpy()** mengubah tensor PyTorch menjadi array NumPy.

Perlu diperhatikan bahwa konversi dari NumPy ke PyTorch atau sebaliknya akan mempertahankan tipe data (dtype) asalnya, kecuali diubah secara eksplisit. Jika ingin mengubah tipe data, bisa menggunakan **.type()** untuk tensor atau mengatur dtype pada array NumPy.

Reproducibility in PyTorch:

Reproducibility adalah kemampuan untuk mendapatkan hasil yang sama atau sangat mirip ketika menjalankan kode pada dua komputer yang berbeda. Dalam konteks pembelajaran mesin dan jaringan saraf, ketidakpastian dapat disebabkan oleh pseudorandomness, yang merupakan hasil dari angka acak yang dihasilkan oleh komputer.

Dalam PyTorch, kita dapat menggunakan **torch.manual_seed(seed)** untuk mengontrol randomness dan menciptakan hasil yang dapat diulang. Sebagai contoh, dua tensor acak yang dihasilkan dengan seed yang sama akan memiliki nilai yang identik, memungkinkan eksperimen yang dapat diulang.

Menjalankan Tensor pada GPU (dan Meningkatkan Kecepatan Perhitungan)

Algoritma deep learning memerlukan banyak operasi numerik. Secara default, operasi-operasi ini sering dilakukan pada CPU (unit pemrosesan komputer). Namun, terdapat sebuah perangkat keras umum lainnya yang disebut GPU (unit pemrosesan grafis), yang seringkali jauh lebih cepat dalam melakukan jenis operasi tertentu yang diperlukan oleh jaringan saraf (seperti perkalian matriks) dibandingkan dengan CPU.

Komputer mungkin dilengkapi dengan GPU. Jika ya, sebaiknya menggunakan GPU tersebut setiap kali melatih jaringan saraf karena kemungkinan besar itu akan mempercepat waktu pelatihan secara dramatis.

Ada beberapa cara untuk pertama-tama mendapatkan akses ke GPU dan kedua, membuat PyTorch menggunakan GPU.

1. **Mendapatkan GPU**

- **Google Colab:** Mudah, Gratis digunakan, hampir tidak perlu setup, dapat berbagi pekerjaan dengan orang lain dengan mudah melalui tautan. Kekurangannya, tidak menyimpan output data Kita, komputasi terbatas, dan terkena batas waktu.
- **Gunakan milik Kita sendiri:** Sedang, Jalankan semuanya secara lokal di mesin Kita sendiri. Kelemahannya, GPU tidak gratis, membutuhkan biaya awal.
- **Cloud computing (AWS, GCP, Azure):** Sedang-Sulit, Biaya awal kecil, akses ke komputasi yang hampir tak terbatas. Kekurangannya, bisa mahal jika berjalan terus-menerus, memerlukan waktu untuk setup yang tepat.

2. **Mendapatkan PyTorch Berjalan di GPU**
    - Pastikan GPU dapat diakses menggunakan **torch.cuda.is_available()**. Jika bernilai True, PyTorch dapat melihat dan menggunakan GPU.
    - Tentukan perangkat yang akan digunakan (CPU atau GPU) menggunakan **torch.device("cuda" if torch.cuda.is_available() else "cpu")**.
3. **Meletakkan Tensor (dan Model) pada GPU**
    - Gunakan **to(device)** pada tensor (dan model) untuk menempatkannya pada perangkat tertentu (CPU atau GPU).
    - Saat menggunakan **to(device)**, itu mengembalikan salinan tensor tersebut. Untuk menimpanya, berikan kembali hasilnya ke tensor yang sama (reassign).
4. **Mengembalikan Tensor ke CPU**
    - Gunakan **Tensor.cpu()** untuk memindahkan tensor kembali ke CPU. Ini diperlukan jika ingin berinteraksi dengan tensor menggunakan NumPy, karena NumPy tidak menggunakan GPU.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

01. PyTorch Workflow Fundamentals

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Modul ini membuka pintu dalam pemahaman dasar tentang machine learning dan deep learning. Tujuan utamanya adalah menggunakan data masa lalu untuk membangun algoritma, seperti jaringan saraf (neural network), yang dapat menemukan pola dalam data tersebut. Setelah pola ditemukan, algoritma tersebut dapat digunakan untuk memprediksi peristiwa di masa depan.

Dalam modul ini, kita akan memulai dengan konsep yang sederhana, yakni garis lurus. Kemudian, kita akan melihat bagaimana membuat model PyTorch yang dapat mempelajari pola dari garis lurus tersebut dan mencocokkannya.

Berikut adalah langkah-langkah utama yang akan dibahas dalam modul ini:

1. Persiapan Data: Data dapat berasal dari berbagai sumber, namun pada awalnya, kita akan membuat data sederhana berupa garis lurus.
2. Pembuatan Model: Di sini, kita akan membuat model untuk mempelajari pola dalam data. Kita juga akan memilih fungsi kerugian (loss function), pengoptimal (optimizer), dan membangun loop pelatihan.
3. Penyesuaian Model dengan Data (pelatihan): Setelah memiliki data dan model, kita akan membiarkan model mencari pola dalam data pelatihan.
4. Pembuatan Prediksi dan Evaluasi Model (inferensi): Model kita telah menemukan pola dalam data, sekarang kita akan membandingkan temuannya dengan data pengujian aktual.
5. Penyimpanan dan Pemuatan Model: Mungkin Anda ingin menggunakan model Anda di tempat lain atau kembali menggunakannya nanti. Di sini, kita akan membahas cara menyimpan dan memuat model.
6. Penggabungan Semua Komponen: Mari menggabungkan semua langkah di atas dan melihatnya sebagai suatu kesatuan.

Persiapan dan Pemuatan Data

Dalam machine learning, data dapat berupa apa saja, mulai dari tabel angka, gambar, video, audio, hingga struktur protein dan teks. Proses machine learning melibatkan dua bagian utama, yakni mengubah data menjadi kumpulan angka representatif dan membangun atau memilih model untuk mempelajari representasi tersebut sebaik mungkin. Pada modul ini, kita mulai dengan menciptakan data garis lurus sebagai contoh. Langkah-langkah yang akan dijalankan meliputi:

1. Pembuatan parameter yang diketahui untuk data garis lurus.
2. Pembuatan data menggunakan linear regression dengan menggunakan PyTorch.
3. Pembagian data menjadi set pelatihan dan pengujian dengan proporsi 80:20.
4. Visualisasi data untuk memahami pola dan hubungan antara fitur (X) dan label (y).

Data yang sudah dipersiapkan dan divisualisasikan akan digunakan sebagai dasar untuk membangun model pada langkah selanjutnya.

Pembangunan Model

Dalam langkah ini, kita membangun model regresi linear menggunakan PyTorch. Model ini dibuat sebagai kelas Python yang mengimplementasikan fungsi regresi linear sederhana. Pada intinya, kita menggunakan modul **torch.nn** yang menyediakan blok bangunan untuk grafik komputasi, seperti **nn.Module**, **nn.Parameter**, dan **torch.optim**. Model regresi linear yang dibuat memiliki dua parameter, yaitu weight dan bias, yang akan disesuaikan selama proses pembelajaran.

Kita juga melakukan pengujian sederhana pada model yang telah dibuat dengan membuat beberapa prediksi pada data uji. Hasil prediksi awalnya buruk karena parameter model diinisialisasi secara acak. Pada langkah selanjutnya, kita akan melihat bagaimana model ini dapat disesuaikan untuk memprediksi data dengan lebih baik.

Rangkuman Bagian 3: Pelatihan Model

Pada langkah ini, kita melatih model regresi linear menggunakan PyTorch. Proses pelatihan ini melibatkan beberapa langkah kunci, yaitu:

1. **Inisialisasi Model**: Model diinisialisasi dengan parameter acak (weights dan bias).
2. **Pembuatan Fungsi Kerugian dan Optimizer**: Fungsi kerugian (loss function) digunakan untuk mengukur seberapa salah prediksi model dibandingkan dengan nilai sebenarnya. Optimizer, dalam hal ini Stochastic Gradient Descent (SGD), digunakan untuk mengupdate parameter model agar sesuai dengan pola data yang lebih baik.

3. **Loop Pelatihan (Training Loop)**: Model melakukan iterasi melalui data latih, melakukan prediksi, menghitung loss, melakukan backpropagation, dan memperbarui parameter untuk meminimalkan loss.
4. **Loop Pengujian (Testing Loop)**: Model diuji pada data uji untuk mengevaluasi seberapa baik model dapat memgeneralisasi ke data yang belum pernah dilihat selama pelatihan.
5. **Visualisasi Loss Curves**: Melihat kurva loss dari waktu ke waktu untuk menilai seberapa baik model telah belajar dari data.

Hasil dari pelatihan ini adalah model yang dapat memprediksi dengan lebih baik, dan kita melihat penurunan nilai loss pada setiap iterasi. Pada akhir pelatihan, kita melihat bahwa parameter model (weights dan bias) mendekati nilai yang sebenarnya, menunjukkan bahwa model telah mempelajari pola dalam data.

Penting untuk memahami konsep pelatihan dan evaluasi model, serta pengaruh hyperparameter seperti jumlah epoch dan learning rate dalam mencapai hasil yang baik. Langkah-langkah ini membantu kita memahami inti dari workflow pada PyTorch untuk mengembangkan model machine learning sederhana.

Melakukan Prediksi dengan Model PyTorch yang Telah Dilatih (Inference)

Setelah melatih model, langkah selanjutnya adalah membuat prediksi (inference) dengan model tersebut. Berikut adalah langkah-langkah utama untuk melakukan prediksi dengan model PyTorch:

1. **Mengatur Model ke Mode Evaluasi**: Menggunakan fungsi **model.eval()** untuk memastikan bahwa model berada dalam mode evaluasi, di mana beberapa fungsi yang tidak diperlukan selama pelatihan dinonaktifkan.
2. **Menggunakan Konteks Inference Mode**: Menggunakan **with torch.inference_mode():** sebagai konteks untuk membuat prediksi. Ini memastikan bahwa kalkulasi dilakukan dengan cepat dan efisien.
3. **Memastikan Kalkulasi pada Perangkat yang Sama**: Pastikan bahwa model dan data berada pada perangkat yang sama (CPU atau GPU) untuk mencegah kesalahan lintas perangkat.

Setelah langkah-langkah tersebut dilakukan, kita dapat membuat prediksi dengan menggunakan model yang telah dilatih pada data uji. Dalam contoh di atas, hasil prediksi (**y_preds**) telah dihasilkan dan visualisasi prediksi dilakukan menggunakan fungsi **plot_predictions()**.

Selanjutnya, kita akan menjelajahi cara menyimpan dan memuat model PyTorch.

Menyimpan dan Memuat Model PyTorch

Setelah melatih model PyTorch, langkah selanjutnya adalah menyimpannya untuk digunakan di lain waktu atau di tempat lain. PyTorch menyediakan beberapa metode utama untuk menyimpan dan memuat model, dan di bawah ini adalah tiga metode tersebut:

1. **torch.save**: Menyimpan objek ter-serialisasi ke disk menggunakan utilitas pickle Python. Model, tensor, dan berbagai objek Python lainnya seperti kamus dapat disimpan menggunakan torch.save.
2. **torch.load**: Menggunakan fitur deserialisasi pickle untuk memuat file objek Python ter-serialisasi (seperti model, tensor, atau kamus) ke dalam memori. Anda juga dapat menentukan perangkat mana objek harus dimuat ke (CPU, GPU, dll).
3. **torch.nn.Module.load_state_dict**: Memuat kamus parameter model (model.state_dict()) menggunakan objek state_dict() yang telah disimpan.

Disarankan untuk menyimpan dan memuat model hanya menggunakan state_dict(), karena metode ini lebih fleksibel dan tidak tergantung pada struktur kelas atau direktori yang spesifik.

1. **Menyimpan Model:**
   - Membuat direktori untuk menyimpan model (dalam contoh ini, "models").
   - Membuat jalur file untuk menyimpan model.
   - Menggunakan **torch.save(obj, f)** di mana obj adalah state_dict() model target dan f adalah nama file untuk menyimpan model.
2. **Memuat Model:**
   - Membuat instance baru dari model yang akan dimuat.
   - Memuat state_dict() model yang telah disimpan menggunakan **torch.load(f)** dan memasukkannya ke instance baru model menggunakan **loaded_model.load_state_dict()**.
3. **Melakukan Prediksi dengan Model yang Telah Dimuat:**
   - Mengatur model yang dimuat ke mode evaluasi.
   - Menggunakan konteks inference_mode untuk membuat prediksi.

Setelah memuat model, prediksi yang dihasilkan oleh model tersebut dapat dibandingkan dengan prediksi sebelumnya untuk memastikan bahwa model telah menyimpan dan memuat dengan benar.

Menggabungkan Semua Langkah

1. **Data**
    1. Membuat data dengan menentukan weight dan bias.
    2. Membuat range nilai antara 0 dan 1 untuk X.
    3. Menggunakan X, weight, dan bias untuk membuat label (y) menggunakan rumus regresi linear (y = weight * X + bias).
    4. Membagi data menjadi set pelatihan dan pengujian (80/20 split).
2. **Membangun Model Linear PyTorch**
    1. Membuat model linear menggunakan subclass nn.Module dan nn.Linear().
    2. Model memiliki satu lapisan linear dengan satu input dan satu output.
3. **Pelatihan Model**
    1. Menggunakan loss function nn.L1Loss() dan optimizer torch.optim.SGD().
    2. Pelatihan model selama serangkaian epoch dengan meletakkan data di perangkat yang tersedia (GPU jika tersedia, jika tidak, default ke CPU).
4. **Melakukan Prediksi**
    1. Mengubah model ke mode evaluasi.
    2. Melakukan prediksi pada data pengujian.
5. **Menyimpan dan Memuat Model**
    1. Menyimpan model ke file menggunakan torch.save() dengan menyertakan state_dict() model.
    2. Memuat model kembali menggunakan torch.load() dan load_state_dict().
    3. Evaluasi model yang dimuat untuk memastikan prediksinya sesuai dengan model sebelumnya.
6. **Keseluruhan**

Menggabungkan langkah-langkah di atas dalam satu rangkaian kode dengan membuatnya agnostik perangkat, sehingga dapat menggunakan GPU jika tersedia atau beralih ke CPU jika tidak.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

02. PyTorch Neural Network Classification

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Pada modul ini, kita memahami dasar-dasar masalah klasifikasi dalam machine learning. Klasifikasi melibatkan prediksi apakah suatu elemen masuk ke dalam suatu kategori tertentu atau tidak. Ada tiga jenis utama masalah klasifikasi: binary classification, multi-class classification, dan multi-label classification.

Arsitektur Jaringan Saraf untuk Klasifikasi

Sebelum kita masuk ke penulisan kode, mari kita lihat arsitektur umum dari jaringan saraf untuk masalah klasifikasi. Berikut adalah beberapa hyperparameter dan komponen kunci yang dapat disesuaikan sesuai dengan jenis klasifikasi:

Binary Classification:

- **Input layer shape (in_features):** Jumlah fitur pada data.
- **Hidden layer(s):** Jumlah dan konfigurasi hidden layer (minimal 1, maksimal tidak terbatas).
- **Neurons per hidden layer:** Jumlah neuron dalam setiap hidden layer (problem spesifik, umumnya 10 hingga 512).
- **Output layer shape (out_features):** 1 (satu kelas atau yang lain).
- **Hidden layer activation:** Biasanya ReLU (rectified linear unit) atau fungsi aktivasi lainnya.
- **Output activation:** Sigmoid (torch.sigmoid in PyTorch).
- **Loss function:** Binary crossentropy (torch.nn.BCELoss in PyTorch).
- **Optimizer:** SGD (stochastic gradient descent), Adam, dll.

Multiclass Classification:

- **Input layer shape (in_features):** Sama dengan jumlah fitur pada data.
- **Hidden layer(s):** Sama seperti pada binary classification.
- **Neurons per hidden layer:** Sama seperti pada binary classification.

- **Output layer shape (out_features):** 1 per kelas (misalnya, 3 untuk kelas makanan, orang, atau anjing).
- **Hidden layer activation:** Sama seperti pada binary classification.
- **Output activation:** Softmax (torch.softmax in PyTorch).
- **Loss function:** Cross entropy (torch.nn.CrossEntropyLoss in PyTorch).
- **Optimizer:** Sama seperti pada binary classification.

1. Persiapan Data Klasifikasi

1.1 Membuat dan Menyusun Data Klasifikasi

Data klasifikasi dihasilkan menggunakan metode **make_circles()** dari Scikit-Learn, menghasilkan dua lingkaran dengan titik-titik berwarna berbeda. Data terdiri dari dua fitur (X1 dan X2) dan label (y) yang berisi nilai 0 atau 1.

1.2 Visualisasi Data

Data disajikan dalam DataFrame Pandas dan divisualisasikan dengan scatter plot untuk memahami distribusi dan pola data.

1.3 Penyesuaian Input dan Output

Bentuk data (shape) dieksplorasi untuk memastikan konsistensi. Data diubah menjadi tensor untuk kompatibilitas dengan PyTorch, dan satu sampel data diperiksa untuk memahami strukturnya.

1.4 Konversi Data dan Pembagian

Data diubah menjadi tensor PyTorch, dan dilakukan pembagian data menjadi set pelatihan (80%) dan pengujian (20%) menggunakan train_test_split.

Sekarang, kita memiliki 800 sampel pelatihan dan 200 sampel pengujian yang siap digunakan untuk melatih dan menguji model klasifikasi.

2. Pembangunan Model

2.1 Persiapan Kode Agnostik Perangkat

Kode perangkat-agnostik memungkinkan model berjalan pada CPU atau GPU jika tersedia. Kode ini memastikan kompatibilitas perangkat.

2.2 Konstruksi Model

Model dibangun dengan membuat kelas yang mewarisi nn.Module. Dua lapisan nn.Linear dibuat dalam konstruktor untuk menangani bentuk input dan output data.

2.3 Fungsi Forward dan Instance Model

Fungsi forward() menentukan perhitungan langkah maju model. Model diinstansiasi dan dikirimkan ke perangkat target (CPU atau GPU).

2.4 Pembuatan Model dengan nn.Sequential (Opsional)

Diperkenalkan penggunaan nn.Sequential untuk konstruksi model yang lebih sederhana, namun custom nn.Module sering diperlukan untuk kasus yang lebih kompleks.

2.5 Prediksi dengan Model

Model diuji dengan melakukan prediksi pada data uji. Hasilnya dievaluasi untuk memastikan model memberikan keluaran yang sesuai.

2.6 Penyiapan Fungsi Kerugian dan Optimizer

Fungsi kerugian nn.BCEWithLogitsLoss() dipilih untuk masalah klasifikasi biner. Optimizer SGD dengan laju pembelajaran 0.1 digunakan untuk mengoptimalkan parameter model.

2.7 Fungsi Evaluasi (Opsional)

Fungsi evaluasi accuracy_fn() dibuat untuk mengukur akurasi model selama pelatihan dan evaluasi.

Dengan demikian, model telah berhasil dibangun dan siap untuk melalui proses pelatihan pada data klasifikasi.

3. Pelatihan Model

3.1. Pengubah Output Model ke Label Prediksi

Sebelum memulai langkah-langkah pelatihan, perlu dikonfirmasi bahwa model yang dibangun menghasilkan keluaran yang sesuai dengan label prediksi. Pada langkah ini, output raw model (logit) diubah menjadi probabilitas prediksi dan kemudian dibulatkan untuk mendapatkan label prediksi.

3.2 Pembangunan Loop Pelatihan dan Pengujian

Langkah-langkah pelatihan dan pengujian model dibangun dalam loop. Dalam setiap epoch, model diperbarui menggunakan optimizer dan hasil pelatihan serta pengujian dicetak setiap 10 epoch untuk memantau perkembangan.

3.3 Evaluasi Awal Model

Meskipun model telah melalui proses pelatihan, performanya masih belum memuaskan. Akurasi yang stagnan di sekitar 50% menunjukkan bahwa model saat ini tidak lebih baik daripada pengambilan keputusan acak. Evaluasi lebih lanjut dan penyesuaian mungkin diperlukan untuk meningkatkan kinerja model pada tugas klasifikasi ini.

4. Melakukan Prediksi dan Mengevaluasi Model

4.1 Penilaian Awal Melalui Metrik

Dari metrik yang telah diperoleh, terlihat bahwa model cenderung melakukan prediksi secara acak, sebagaimana ditunjukkan oleh akurasi yang stagnan di sekitar 50%. Namun, untuk mendapatkan pemahaman yang lebih mendalam, perlu dilakukan analisis visual.

4.2 Visualisasi dan Analisis Decision Boundary

Dalam upaya memahami lebih jauh, dilakukan visualisasi menggunakan decision boundary. Decision boundary adalah batas pemisah antara kelas yang dihasilkan oleh model. Plot ini memperlihatkan bahwa model saat ini mencoba memisahkan data dengan garis lurus, yang tidak sesuai dengan distribusi data yang berbentuk lingkaran. Hal ini menjelaskan performa rendah model, karena garis lurus hanya dapat memotong data menjadi dua bagian sejajar, tanpa mampu menangkap pola yang sesuai.

4.3 Identifikasi Underfitting

Dalam terminologi machine learning, kondisi di atas disebut sebagai *underfitting*. Underfitting terjadi ketika model tidak mampu menangkap pola yang ada dalam data, sehingga tidak dapat membuat prediksi yang akurat.

4.4 Perbaikan Model

Untuk meningkatkan performa model, perlu dilakukan penyesuaian agar model dapat menangkap pola yang lebih kompleks, khususnya pada distribusi data yang berbentuk lingkaran. Upaya perbaikan dapat melibatkan penambahan kompleksitas model atau penerapan teknik-teknik khusus seperti kernel tricks pada algoritma Support Vector Machine (SVM). Analisis visual ini memberikan wawasan tambahan yang dapat menjadi dasar untuk langkah-langkah perbaikan selanjutnya.

5. Meningkatkan Model (Dari Perspektif Model)

5.1 Pendekatan Peningkatan Model

Untuk mengatasi masalah *underfitting* pada model kita, kita dapat fokus pada perbaikan model itu sendiri (bukan pada data). Beberapa cara yang dapat diterapkan melibatkan penambahan lapisan atau unit tersembunyi, peningkatan jumlah *epochs*, perubahan fungsi aktivasi, penyesuaian *learning rate*, dan perubahan fungsi loss.

5.2 Penerapan Peningkatan Model

Mari coba memperbaiki model dengan menambahkan lapisan tambahan, meningkatkan jumlah unit tersembunyi, dan melatih untuk lebih lama (epochs=1000).

Kita telah membuat **CircleModelV1** yang memiliki arsitektur sedikit lebih kompleks. Namun, setelah pelatihan lebih lama, model ini masih tidak menunjukkan peningkatan yang signifikan dalam pembelajaran pola dari data.

5.3 Eksplorasi Masalah

Pertanyaannya sekarang, mengapa model kita masih gagal? Apakah model kita tidak dapat memodelkan data lingkaran dengan baik? Untuk menguji ini, kita membuat data linier dan melatih model (model_2) untuk melihat apakah dapat memodelkan data tersebut.

5.4 Penyesuaian Model untuk Data Linier

Model_2, yang dirancang untuk menangani data linier, menunjukkan hasil yang lebih baik daripada model_1 pada data tersebut. Pada kasus ini, model berhasil mengurangi loss selama pelatihan.

6. Puzzel yang Hilang: Non-linearitas

6.1 Membuat Ulang Data Non-linear (Lingkaran Merah dan Biru)

Kita telah melihat bahwa model kita dapat menggambar garis lurus (linear) berkat lapisan linearnya. Tetapi bagaimana jika kita memberinya kapasitas untuk menggambar garis yang tidak lurus (non-linear)?

Mari kita temukan jawabannya.

6.2 Membangun Model dengan Non-linearitas

Kita dapat meningkatkan kapasitas model kita dengan menambahkan fungsi aktivasi non-linear. Sebagai contoh, kita akan menggunakan ReLU (Rectified Linear Unit) di antara lapisan tersembunyi. Fungsi ini dapat membantu model mempelajari pola non-linear dari data.

6.3 Melatih Model dengan Non-linearitas

Setelah menambahkan ReLU, kita melatih model menggunakan data baru yang terdiri dari lingkaran merah dan biru. Kita memantau loss dan akurasi selama pelatihan untuk melihat apakah model mampu memahami pola non-linear.

6.4 Evaluasi Model yang Dilatih dengan Fungsi Aktivasi Non-linear

Setelah pelatihan, kita mengevaluasi model pada data uji dan memeriksa prediksinya. Terakhir, kita membandingkan pengambilan keputusan model yang menggunakan non-linearitas dengan model sebelumnya yang hanya menggunakan linearitas.

7. Menduplikasi Fungsi Aktivasi Non-linear

7.1 Replikasi Fungsi ReLU

Sebelumnya, kita telah melihat bagaimana menambahkan fungsi aktivasi non-linear ke dalam model dapat membantu dalam memodelkan data non-linear. Namun, apa sebenarnya yang dilakukan oleh fungsi aktivasi non-linear tersebut?

Mari kita coba mereplikasi beberapa fungsi aktivasi non-linear dan melihat apa yang mereka lakukan. Pertama, kita akan membuat data kecil menggunakan tensor PyTorch.

7.2 Fungsi ReLU

Fungsi ReLU (Rectified Linear Unit) mengubah nilai negatif menjadi 0 dan membiarkan nilai positif tidak berubah. Kita mencoba mereplikasi fungsi ini menggunakan PyTorch.

7.3 Fungsi Sigmoid

Selanjutnya, kita coba mengimplementasikan fungsi sigmoid yang telah kita gunakan sebelumnya. Fungsi sigmoid digunakan untuk menghasilkan nilai antara 0 dan 1, yang sering diinterpretasikan sebagai probabilitas.

7.4 Visualisasi Hasil

Setelah mereplikasi kedua fungsi tersebut, kita visualisasikan hasilnya. Dari grafik, kita dapat melihat perbedaan antara garis lurus awal dan garis yang telah diubah oleh fungsi aktivasi non-linear.

8. Menggabungkan Semua dalam Model PyTorch Multi-Class

8.1 Membuat Data Multi-Class

Sebelumnya, kita telah bekerja dengan masalah klasifikasi biner. Sekarang, mari terapkan semua konsep ini pada masalah klasifikasi multi-kelas. Perbedaan utama adalah bahwa kita sekarang memiliki lebih dari dua kelas sebagai opsi.

Untuk mulai masalah klasifikasi multi-kelas, kita akan membuat beberapa data multi-kelas. Kita akan menggunakan metode **make_blobs()** dari Scikit-Learn untuk membuat beberapa cluster atau kelompok data.

8.2 Membangun Model Klasifikasi Multi-Class di PyTorch

Kita akan membuat model serupa dengan **model_3**, namun kali ini model harus mampu menangani data multi-kelas. Kita akan membuat kelas **BlobModel** yang mengambil tiga hiperparameter: jumlah fitur masukan (**input_features**), jumlah fitur keluaran yang diinginkan (**output_features**), dan jumlah unit tersembunyi (**hidden_units**).

8.3 Membuat Fungsi Loss dan Optimizer untuk Model PyTorch Multi-Class

Karena kita menghadapi masalah klasifikasi multi-kelas, kita akan menggunakan metode **nn.CrossEntropyLoss()** sebagai fungsi loss. Dan kita akan tetap menggunakan SGD dengan tingkat pembelajaran 0.1 untuk mengoptimalkan parameter model.

8.4 Mendapatkan Probabilitas Prediksi untuk Model PyTorch Multi-Class

Sebelum melatih model, mari lakukan satu kali langkah maju (**forward pass**) dengan model untuk melihat apakah semuanya berfungsi. Hasil output yang kita dapatkan saat ini disebut logits. Untuk mengonversi logits menjadi probabilitas prediksi, kita akan menggunakan fungsi aktivasi softmax.

8.5 Membuat Loop Pelatihan dan Pengujian untuk Model PyTorch Multi-Class

Setelah semua persiapan selesai, kita dapat menulis loop pelatihan dan pengujian untuk meningkatkan dan mengevaluasi model. Kita akan menyesuaikan langkah-langkah untuk mengonversi output model (logits) menjadi probabilitas prediksi dan kemudian menjadi label prediksi.

8.6 Membuat dan Mengevaluasi Prediksi dengan Model PyTorch Multi-Class

Model terlatih sepertinya sudah berkinerja cukup baik. Namun, untuk memastikan, mari membuat beberapa prediksi dan memvisualisasikannya. Kita akan mengonversi logits prediksi model menjadi probabilitas prediksi menggunakan fungsi softmax dan kemudian menjadi label prediksi menggunakan argmax. Selanjutnya, kita akan membandingkan hasilnya dengan label uji dan mengevaluasi akurasi.

Terakhir, kita akan memvisualisasikan hasil prediksi menggunakan fungsi **plot_decision_boundary()**. Ingatlah bahwa data kita ada di GPU, sehingga kita perlu memindahkannya ke CPU untuk digunakan dengan matplotlib, dan fungsi **plot_decision_boundary()** melakukan ini secara otomatis.

9. Lebih Banyak Metrik Evaluasi Klasifikasi

Sejauh ini, kita baru membahas beberapa cara untuk mengevaluasi model klasifikasi (akurasi, loss, dan visualisasi prediksi). Ini adalah beberapa metode umum yang akan Anda temui dan merupakan titik awal yang baik.

Namun, Anda mungkin ingin mengevaluasi model klasifikasi Anda menggunakan lebih banyak metrik seperti yang tercantum di bawah ini:

1. **Akurasi (Accuracy):**
    - Mengukur dari 100 prediksi, berapa banyak yang benar oleh model Anda? Sebagai contoh, akurasi 95% berarti model mendapatkan 95/100 prediksi dengan benar.
2. **Presisi (Precision):**
    - Proporsi positif benar dibandingkan dengan jumlah total sampel. Presisi yang lebih tinggi mengarah ke lebih sedikit positif palsu (model memprediksi 1 ketika seharusnya 0).
3. **Recall:**
    - Proporsi positif benar dibandingkan dengan jumlah total positif benar dan negatif palsu (model memprediksi 0 ketika seharusnya 1). Recall yang lebih tinggi mengarah ke lebih sedikit negatif palsu.
4. **F1-score:**

- Menggabungkan presisi dan recall menjadi satu metrik. Nilai 1 adalah yang terbaik, 0 adalah yang terburuk.

5. **Confusion Matrix:**
    - Membandingkan nilai prediksi dengan nilai sebenarnya secara tabuler. Jika 100% benar, semua nilai dalam matriks akan berada pada garis diagonal dari kiri atas ke kanan bawah.
6. **Classification Report:**
    - Kumpulan beberapa metrik klasifikasi utama seperti presisi, recall, dan f1-score.

Scikit-Learn (perpustakaan pembelajaran mesin yang populer dan kelas dunia) memiliki banyak implementasi metrik-metrik di atas. Jika Anda mencari versi yang mirip dengan PyTorch, coba lihat TorchMetrics, terutama bagian klasifikasi.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

03. PyTorch Computer Vision

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Visi komputer adalah seni mengajari komputer untuk melihat. Ini dapat mencakup membangun model untuk mengklasifikasikan apakah suatu foto adalah foto kucing atau anjing (klasifikasi biner), atau mengklasifikasikan apakah suatu foto adalah kucing, anjing, atau ayam (klasifikasi multi-kelas). Selain itu, visi komputer juga mencakup mengidentifikasi di mana mobil muncul dalam bingkai video (deteksi objek) atau memahami bagaimana objek berbeda dalam gambar dapat dipisahkan (segmentasi panoptik).

1. Mendapatkan Dataset FashionMNIST:

- FashionMNIST adalah dataset computer vision yang berisi gambar berwarna abu-abu dari 10 jenis pakaian.
- Diunduh menggunakan PyTorch's **torchvision.datasets.FashionMNIST()** dengan parameter root, train, download, transform, dan target_transform.

1.1 Bentuk Input dan Output Model Computer Vision:

- Gambar direpresentasikan sebagai tensor dengan bentuk [color_channels, height, width].
- Color_channels=1 menunjukkan gambar grayscale. Untuk RGB, color_channels=3.
- Representasi CHW (Color Channels, Height, Width) atau HWC (Color Channels Last) digunakan.
- NHWC dianggap lebih baik secara kinerja, namun PyTorch secara default menggunakan NCHW.

1.2 Visualisasi Data:

- Visualisasi beberapa contoh dari dataset menggunakan matplotlib.
- Terdapat 60,000 sampel pelatihan dan 10,000 sampel pengujian.
- Terdapat 10 kelas pakaian, menjadikan masalahnya multi-class classification.
- Contoh visualisasi:
    - Menampilkan bentuk dan gambar dari satu sampel.

    - Menampilkan beberapa sampel dalam grid 4x4 dengan label kelas.
- Beberapa pertimbangan:
    - Dataset ini mungkin tidak terlihat estetis, tetapi prinsip membangun model akan tetap sama di berbagai masalah computer vision.
    - Meskipun terdapat 60,000 gambar, ini masih dianggap dataset kecil dalam deep learning.
    - Model PyTorch akan lebih cepat daripada menulis program untuk mengklasifikasikan setiap gambar secara manual.

2. Prepare DataLoader

Bagian kedua membahas persiapan data menggunakan **torch.utils.data.DataLoader**. DataLoader membantu memuat data ke dalam model, baik selama pelatihan maupun inferensi. Fungsinya adalah mengubah Dataset besar menjadi iterabel Python berupa potongan-potongan kecil yang disebut batch atau mini-batch, yang dapat diatur dengan parameter **batch_size**.

Penggunaan batch ini lebih efisien secara komputasional. Meskipun idealnya kita bisa melakukan forward pass dan backward pass pada seluruh data sekaligus, namun, saat menggunakan dataset yang sangat besar, kecuali jika memiliki daya komputasi tak terbatas, lebih mudah untuk memecahnya menjadi batch.

Pendekatan ini juga memberikan lebih banyak kesempatan bagi model untuk meningkatkan performanya. Dengan mini-batch (bagian kecil dari data), gradien descent dilakukan lebih sering per epoch (sekali per mini-batch daripada sekali per epoch).

Batch size yang baik untuk memulai adalah 32, tetapi karena ini adalah nilai yang dapat diatur (hyperparameter), Anda dapat mencoba berbagai nilai, meskipun umumnya kelipatan 2 sering digunakan (misalnya, 32, 64, 128, 256, 512).

Kode yang diberikan mengilustrasikan cara membuat DataLoader untuk set pelatihan dan pengujian dengan batch size yang ditetapkan. Output dari DataLoader juga ditampilkan, termasuk panjang DataLoader untuk set pelatihan dan pengujian.

Selanjutnya, ditunjukkan bagaimana data tetap tidak berubah dengan memeriksa satu sampel dari DataLoader pelatihan. Proses pengambilan batch dan penggunaan DataLoader untuk melatih model telah berhasil diimplementasikan dalam kode ini.

3. Model 0: Build a baseline model

3.1 Persiapan Loss, Optimizer, dan Metrik Evaluasi

Pada bagian ini, dibahas mengenai persiapan fungsi loss, optimizer, dan metrik evaluasi untuk tugas klasifikasi. Fungsi CrossEntropyLoss (**nn.CrossEntropyLoss()**) digunakan untuk tugas klasifikasi, sementara stochastic gradient descent (**torch.optim.SGD**) dipilih sebagai

optimizer. Selain itu, metrik akurasi diperkenalkan menggunakan fungsi **accuracy_fn** yang diimpor dari skrip **helper_functions**. Fungsi akurasi menghitung persentase prediksi yang benar.

3.2 Pembuatan Fungsi untuk Mengukur Waktu Eksperimen

Untuk mengukur waktu pelatihan model, dibuat fungsi pengukuran waktu (**print_train_time**). Fungsi ini menerima waktu awal dan waktu akhir pelatihan sebagai input dan mencetak waktu yang diperlukan untuk melatih model pada perangkat yang ditentukan.

3.3 Pembuatan Loop Pelatihan dan Pelatihan Model pada Batch Data

Bagian ini mengimplementasikan loop pelatihan untuk melatih model dasar (**model_0**). Loop pelatihan iterasi melalui epoch, dan untuk setiap epoch, memproses batch pelatihan menggunakan DataLoader (**train_dataloader**). Loop ini mencakup langkah forward pass, perhitungan loss, backward pass, dan langkah optimizer. Selain itu, loop pelatihan mengakumulasi loss pelatihan per epoch.

Setelah pelatihan, loop melanjutkan ke tahap pengujian, di mana model dievaluasi pada set uji (**test_dataloader**). Loop pengujian menghitung baik loss maupun akurasi. Progres keseluruhan, termasuk hasil pelatihan dan pengujian, dicetak selama setiap epoch.

4. Membuat Prediksi dan Mendapatkan Hasil Model 0

Dalam langkah ini, kita membuat prediksi menggunakan model 0 dan mendapatkan hasilnya. Kita telah membuat fungsi **eval_model** yang dapat menerima model yang telah dilatih, DataLoader, fungsi loss, dan fungsi akurasi. Fungsi ini menghitung loss dan akurasi pada set data yang diberikan.

pythonCopy code

Tentukan seed dan hitung hasil model 0 pada dataset uji torch.manual_seed(42) model_0_results = eval_model(model=model_0, data_loader=test_dataloader, loss_fn=loss_fn, accuracy_fn=accuracy_fn) model_0_results

Hasil yang diperoleh dari model 0 adalah sebagai berikut:

pythonCopy code

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

5. Menyiapkan Kode Agen-Langka (Device Agnostic)

Pada langkah ini, kita membuat kode agen-langka untuk memungkinkan penggunaan GPU jika tersedia. Kode ini mengidentifikasi apakah perangkat CUDA (GPU) tersedia dan mengatur perangkat sesuai. Jika GPU tersedia, kita akan menjalankan model pada GPU; jika tidak, kita akan menggunakan CPU..

6. Model 1: Membangun Model Lebih Baik dengan Non-Linearitas

Dalam langkah ini, kita membuat Model 1 yang lebih kompleks dengan menambahkan fungsi non-linear (nn.ReLU()) di antara setiap lapisan linear. Model ini dirancang untuk mengeksplorasi apakah menambahkan non-linearitas dapat meningkatkan performa model.

6.1 Fungsi-Fungsi Pelatihan dan Pengujian

Langkah ini melibatkan pembuatan fungsi-fungsi pelatihan dan pengujian yang lebih umum untuk digunakan dengan kedua model. Fungsi-fungsi ini dirancang untuk menerima model, DataLoader, fungsi loss, optimizer, dan fungsi akurasi. Ini membantu menjaga kode tetap bersih dan modular.

6.2 Evaluasi Model 1

Setelah pelatihan selesai, kita menggunakan fungsi **eval_model** yang diperbarui untuk mengevaluasi hasil Model 1 pada set data uji.

7: Model 2 - Membangun Convolutional Neural Network (CNN)

Pada langkah ini, kita akan membuat Convolutional Neural Network (CNN) atau ConvNet untuk meningkatkan kinerja model kita dalam menangani data visual.

7.1 Pemilihan Model

- CNN dikenal dapat menemukan pola dalam data visual, sehingga kita akan mencoba menggunakan CNN untuk meningkatkan hasil baseline.
- Model yang akan kita gunakan disebut TinyVGG dari situs CNN Explainer.
- TinyVGG mengikuti struktur tipikal dari CNN: Input layer -> [Convolutional layer -> activation layer -> pooling layer] -> Output layer.
- Pilihan model tergantung pada jenis data yang dihadapi. Untuk data terstruktur, kita bisa menggunakan model seperti Gradient Boosted Models atau Random Forests, sedangkan untuk data tak terstruktur seperti gambar, audio, atau teks, CNN atau Transformers lebih cocok.

7.2 Arsitektur TinyVGG

- TinyVGG memiliki dua blok utama, masing-masing terdiri dari lapisan-lapisan Convolutional, ReLU activation, dan MaxPooling.
- Struktur umum: Conv2d -> ReLU -> Conv2d -> ReLU -> MaxPool2d.
- Kita menggunakan kelas **FashionMNISTModelV2** untuk mengimplementasikan TinyVGG dengan menggunakan modul **nn.Conv2d** dan **nn.MaxPool2d** dari PyTorch.

7.3 Pembuatan Model CNN

class FashionMNISTModelV2(nn.Module):

```
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        # Definisikan struktur model dengan dua blok utama
        # ...
    def forward(self, x: torch.Tensor):
        # Implementasikan proses forward pass
        # ... return x
# Inisialisasi model
torch.manual_seed(42)
model_2 = FashionMNISTModelV2(input_shape=1, hidden_units=10, output_shape=len(class_names)).to(device)
```

7.4 Contoh Penggunaan Conv2d dan MaxPool2d

- **nn.Conv2d** digunakan untuk mengekstrak fitur dari gambar dengan menyaringnya menggunakan kernel tertentu.
- **nn.MaxPool2d** digunakan untuk mereduksi dimensi spasial gambar.
- Contoh penggunaan dan pengujian parameter dapat dilihat pada potongan kode.

8. Membandingkan Hasil Model dan Waktu Pelatihan

- Tiga model berbeda telah dilatih: model_0, model_1, dan model_2.
- model_0 adalah model dasar dengan dua layer nn.Linear().
- model_1 sama dengan model dasar, tetapi dengan layer nn.ReLU() di antara layer nn.Linear().
- model_2 adalah model CNN pertama yang meniru arsitektur TinyVGG pada situs CNN Explainer.
- Model-model ini dibangun dan dilatih untuk melihat kinerja terbaik.
- Hasil model disatukan dalam DataFrame untuk perbandingan.
- Ditambahkan waktu pelatihan ke dalam perbandingan hasil.

9. Membuat dan Menilai Prediksi Acak dengan Model Terbaik

- Model terbaik adalah model_2.
- Fungsi **make_predictions()** dibuat untuk membuat prediksi dengan model dan data yang diberikan.

- Prediksi dilakukan pada sampel uji dengan model_2.
- Probabilitas prediksi ditampilkan, dan prediksi dikonversi menjadi label.

10. Membuat Matriks Konfusi untuk Evaluasi Lebih Lanjut

- Matriks konfusi digunakan sebagai metode evaluasi visual.
- Prediksi dilakukan dengan model terlatih (model_2).
- Matriks konfusi dibuat menggunakan **torchmetrics.ConfusionMatrix**.
- Matriks konfusi diplot menggunakan **mlxtend.plotting.plot_confusion_matrix**.

11. Menyimpan dan Memuat Model Terbaik

- Model terbaik (model_2) disimpan dengan menyimpan **state_dict()** menggunakan **torch.save()**.
- Model terbaik dimuat kembali dengan membuat instansi baru dari kelas model dan menggunakan **load_state_dict()**.
- Model terlatih ulang untuk memastikan hasilnya sama dengan model sebelum penyimpanan.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

04. PyTorch Custom Datasets

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Modul ini fokus pada pembuatan dan penggunaan dataset kustom dalam PyTorch untuk pemodelan visi komputer. Pembahasan mencakup langkah-langkah untuk mengimpor dan mempersiapkan data, transformasi data, pembuatan dataset kustom, dan pengembangan model menggunakan dataset kustom tersebut.

1. Mendapatkan Data

- Langkah awal adalah memerlukan data.
- Dataset Food101 telah disiapkan, berisi 101.000 gambar dari 101 jenis makanan.
- Untuk latihan, digunakan subset dataset tersebut: 10% dari 3 kelas (pizza, steak, sushi).
- Data disiapkan dalam format klasifikasi gambar stkitar, terstruktur dalam direktori masing-masing kelas.
- Data diunduh dari GitHub dan disiapkan dengan Python.

2. Memahami Data (Persiapan Data)

- Melibatkan langkah penting sebelum membangun model, termasuk memahami struktur dan karakteristik data.
- Data terdiri dari gambar pizza, steak, dan sushi, dengan masing-masing kelas disimpan dalam direktori terpisah.
- Membuat fungsi untuk mengeksplorasi jumlah direktori dan gambar di setiap direktori.

3. Transformasi Data

- Langkah ini diperlukan untuk mempersiapkan data gambar sebelum digunakan oleh PyTorch.
- Menggunakan torchvision.transforms untuk mengubah gambar menjadi tensor dan memberikan beberapa transformasi seperti meresize dan flip horizontal.

- Dibuat fungsi untuk memvisualisasikan transformasi yang diterapkan pada beberapa gambar.

4. Memuat Data Gambar Menggunakan ImageFolder

- Menggunakan torchvision.datasets.ImageFolder untuk mengonversi data gambar menjadi Dataset PyTorch.
- Data transformasi diaplikasikan saat membuat Dataset.
- Dataset pelatihan dan pengujian diinisialisasi dengan menggunakan fungsi ImageFolder dan dicetak informasinya.
- Kelas-kelas dan panjang setiap Dataset diambil untuk referensi selanjutnya.
- Contoh gambar dan label dari Dataset ditampilkan.

5. Opsi 2 - Memuat Data Gambar dengan Dataset Kustom

Jika pembuat Dataset pra-dibangun seperti torchvision.datasets.ImageFolder() tidak ada, atau jika tidak ada yang sesuai dengan masalah khusus Kita, Kita dapat membuat Dataset kustom sendiri. Namun, ini memiliki kelebihan dan kekurangan.

Pro dari membuat Dataset kustom:

- Dapat membuat Dataset dari hampir segala sesuatu.
- Tidak terbatas pada fungsi Dataset pra-dibangun di PyTorch.

Kontra dari membuat Dataset kustom:

- Meskipun Kita dapat membuat Dataset dari hampir segala sesuatu, tidak berarti itu akan berhasil.
- Penggunaan Dataset kustom seringkali menghasilkan penulisan kode yang lebih banyak, yang dapat rentan terhadap kesalahan atau masalah kinerja.

Untuk melihatnya beraksi, kita akan bekerja menuju mereplikasi torchvision.datasets.ImageFolder() dengan mensubkelasinya menggunakan torch.utils.data.Dataset sebagai kelas dasar untuk semua Dataset di PyTorch.

Langkah-langkahnya melibatkan:

1. Mengimpor modul yang dibutuhkan.
2. Membuat fungsi pembantu untuk mendapatkan nama kelas.
3. Membuat kelas Dataset kustom kita sendiri untuk mereplikasi ImageFolder.
4. Membuat transformasi data untuk persiapan gambar.

5.1. Membuat Fungsi Pembantu untuk Mendapatkan Nama Kelas

Dalam langkah ini, kita membuat fungsi find_classes() yang dapat membuat daftar nama kelas dan kamus nama kelas dan indeks mereka berdasarkan path direktori target.

5.2. Membuat Dataset Kustom untuk Mereplikasi ImageFolder

Langkah ini melibatkan pembuatan kelas Dataset kustom, ImageFolderCustom, untuk mereplikasi fungsionalitas torchvision.datasets.ImageFolder(). Ini melibatkan mensubkelas torch.utils.data.Dataset, menginisialisasi dengan parameter target direktori dan transformasi opsional, dan mengimplementasikan metode **len** dan **getitem**.

5.3. Membuat Fungsi untuk Menampilkan Gambar Acak

Kita membuat fungsi display_random_images() untuk memvisualisasikan gambar-gambar dalam Dataset kita. Fungsi ini mengambil Dataset, jumlah gambar yang akan ditampilkan, kelas (opsional), dan seed acak untuk membuat gambar secara reproduktif.

5.4. Mengubah Gambar yang Dimuat Sendiri Menjadi DataLoader

Setelah kita membuat Dataset kustom kita, langkah selanjutnya adalah mengonversinya menjadi DataLoader menggunakan torch.utils.data.DataLoader(). DataLoader memungkinkan kita untuk memuat data dalam bentuk batch dan menyederhanakan proses pelatihan model.

6. Bentuk Lain dari Transformasi (Augmentasi Data)

Transformasi data dapat digunakan untuk mengubah gambar dengan berbagai cara, termasuk augmentasi data untuk meningkatkan keberagaman set pelatihan. Salah satu bentuk augmentasi data yang dijelaskan adalah transforms.TrivialAugmentWide() yang memanfaatkan kekuatan randomness untuk meningkatkan kinerja model.

Terakhir, kita menciptakan transformasi data untuk pelatihan dan pengujian yang mencakup augmentasi data pada set pelatihan dan konversi gambar menjadi tensor. Transformasi data ini digunakan untuk membuat DataLoader dari Dataset kustom kita.

Sekarang, kita telah memahami dan menerapkan langkah-langkah ini dalam konteks membuat Dataset kustom dan melakukan augmentasi data pada gambar.

7. Model 0: TinyVGG tanpa augmentasi data

7.1. Membuat Transformasi dan Memuat Data untuk Model 0

- Membuat transformasi sederhana, misalnya, meresize gambar menjadi (64, 64) dan mengubahnya menjadi tensor.
- Memuat data, mengonversi folder pelatihan dan pengujian ke dalam Dataset menggunakan torchvision.datasets.ImageFolder(), kemudian menjadi DataLoader menggunakan torch.utils.data.DataLoader().

7.2. Membuat Kelas Model TinyVGG

- Membuat model TinyVGG dengan lapisan konvolusi dan penggkitaan.
- Model ini digunakan untuk tugas klasifikasi gambar.

7.3. Melakukan Forward Pass pada Satu Gambar (Untuk Pengujian Model)

- Melakukan forward pass pada satu gambar untuk menguji model.
- Menggunakan torch.softmax() untuk mengonversi logits menjadi probabilitas prediksi.

7.4. Menggunakan torchinfo untuk Mendapatkan Gambaran Bentuk yang Dilalui oleh Model

- Menggunakan torchinfo untuk mendapatkan informasi rinci tentang model, termasuk jumlah parameter dan ukuran estimasi total.

7.5. Membuat Fungsi Loop Pelatihan dan Pengujian

- Membuat fungsi train_step() untuk pelatihan model pada DataLoader.
- Membuat fungsi test_step() untuk mengevaluasi model pada DataLoader.
- Membuat fungsi train() yang menggabungkan train_step() dan test_step() untuk pelatihan model selama beberapa epoch.

8. Evaluasi Model 0 dan Strategi untuk Memperbaikinya:

- Model 0 memberikan hasil yang kurang memuaskan.
- Beberapa strategi untuk meningkatkan kinerja model melibatkan penanganan overfitting dan underfitting.
- Strategi untuk mengatasi overfitting:
    - Mendapatkan lebih banyak data.
    - Menyederhanakan model.
    - Menggunakan augmentasi data.
    - Menggunakan transfer learning.
    - Menggunakan dropout layers, learning rate decay, dan early stopping.
- Strategi untuk mengatasi underfitting:
    - Menambahkan lebih banyak lapisan atau unit ke model.
    - Penyesuaian learning rate.
    - Menggunakan transfer learning.
    - Melatih model lebih lama.

- Mengurangi regularisasi.

8.1. Cara Mengatasi Overfitting:

- Menggunakan lebih banyak data.
- Menyederhanakan model.
- Menggunakan augmentasi data.
- Menggunakan transfer learning.
- Menggunakan lapisan dropout.
- Menggunakan penurunan tingkat pembelajaran.
- Menggunakan early stopping.

8.2. Cara Mengatasi Underfitting:

- Menambahkan lebih banyak lapisan/unit ke model.
- Menyesuaikan tingkat pembelajaran.
- Menggunakan transfer learning.
- Melatih model untuk lebih lama.
- Mengurangi penggunaan regularisasi.

8.3. Keseimbangan Antara Overfitting dan Underfitting:

- Tidak ada metode yang sempurna, dan keseimbangan antara overfitting dan underfitting adalah tantangan utama dalam pembelajaran mesin.
- Transfer learning sering digunakan sebagai solusi karena dapat mengambil model yang sudah bekerja di ruang masalah serupa dan menggunakannya pada dataset yang berbeda.

9. Model 1 - TinyVGG dengan Augmentasi Data

9.1. Transformasi Data dengan Augmentasi

- Membuat transformasi pelatihan dengan TrivialAugmentWide, meresize gambar, dan mengubahnya menjadi tensor.
- Membuat transformasi pengujian tanpa augmentasi data.

9.2. Membuat Dataset dan DataLoader

- Menggunakan **torchvision.datasets.ImageFolder()** untuk mengonversi folder gambar menjadi Dataset.
- Membuat DataLoader dengan batch_size=32 dan num_workers sesuai jumlah CPU pada mesin.

9.3. Membangun dan Melatih Model 1 (TinyVGG)

- Membuat model TinyVGG (model_1) dan mengirimkannya ke perangkat target.
- Melatih model dengan menggunakan fungsi pelatihan sebelumnya dengan parameter yang sesuai.
- Menggunakan CrossEntropyLoss sebagai fungsi kerugian dan Adam sebagai pengoptimal dengan lr=0.001.
- Melakukan pelatihan selama 5 epoch dan mencetak hasil pelatihan.

9.4. Plot Kurva Kerugian Model 1

- Menampilkan kurva kerugian model_1 menggunakan fungsi plot_loss_curves.

10. Membandingkan Hasil Model

1. **Mengubah Hasil Model ke DataFrame Pkitas**
    - Mengonversi hasil model_0 dan model_1 menjadi DataFrames.
2. **Membandingkan Hasil Model dengan Grafik**
    - Menggunakan matplotlib, membuat grafik untuk membandingkan model_0 dan model_1 dalam hal kerugian dan akurasi pada set pelatihan dan pengujian.

11. Membuat Prediksi pada Gambar Kustom

Langkah ini membahas cara membuat prediksi pada gambar kustom setelah melatih model. Dalam hal ini, model telah dilatih pada dataset gambar pizza, steak, dan sushi. Langkah-langkahnya adalah sebagai berikut:

11.1. Memuat gambar kustom dengan PyTorch

- Mendownload gambar kustom dari URL tertentu menggunakan Python's requests module.
- Membaca gambar dengan torchvision.io.read_image(), menghasilkan tensor uint8.
- Memastikan bahwa format gambar tersebut kompatibel dengan model yang telah dilatih.

11.2. Memprediksi gambar kustom dengan model PyTorch yang dilatih

- Melakukan transformasi pada gambar kustom agar memiliki format yang sama dengan data yang digunakan untuk melatih model.
- Menggunakan torchvision.transforms.Resize() untuk mengubah ukuran gambar.
- Membuat prediksi pada gambar kustom menggunakan model yang telah dilatih, dengan memperhatikan format, perangkat (device), dan ukuran gambar yang sesuai.

- Mengonversi output model dari logits ke probabilitas prediksi.
- Menampilkan gambar bersama dengan prediksi dan probabilitas prediksi.

11.3. Membuat fungsi untuk prediksi dan plot gambar

- Membuat fungsi yang menggabungkan langkah-langkah sebelumnya agar lebih mudah digunakan.
- Fungsi ini menerima model, path gambar target, nama kelas (opsional), transformasi (opsional), dan perangkat sebagai input.
- Fungsi menghasilkan prediksi pada gambar target dan menampilkannya bersama dengan prediksi dan probabilitas prediksi.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

05. PyTorch Going Modular

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Modul ini membahas cara mengubah kode notebook menjadi skrip Python. Proses ini melibatkan konversi sel kode yang paling berguna dari notebook 04, "PyTorch Custom Datasets," menjadi serangkaian skrip Python yang disimpan dalam direktori bernama "going_modular."

Apa itu "going modular"? "Going modular" melibatkan transformasi kode notebook (dari Jupyter Notebook atau Google Colab notebook) menjadi serangkaian skrip Python yang menawarkan fungsionalitas serupa. Misalnya, notebook kode dapat diubah menjadi beberapa file Python, seperti **data_setup.py**, **engine.py**, **model_builder.py**, **train.py**, dan **utils.py**, sesuai dengan kebutuhan proyek.

Mengapa Anda ingin "going modular"? Meskipun notebook berguna untuk eksplorasi dan eksperimen, untuk proyek berskala besar, skrip Python lebih direkomendasikan karena lebih mudah direproduksi dan dijalankan. Dibandingkan dengan notebook, skrip Python memiliki keuntungan dalam hal versioning, penggunaan bagian tertentu, dan kemasan kode yang lebih baik.

Pros dan Cons dari Notebooks vs. Python Scripts: *Notebooks:*

- **Pros:** Mudah untuk bereksperimen, memulai, dan berbagi. Visual dan grafis.
- **Cons:** Versioning sulit, sulit menggunakan bagian tertentu, dan visual yang berlebihan.

Python Scripts:

- **Pros:** Dapat mengemas kode, menggunakan Git untuk versioning, mendukung proyek berskala besar.
- **Cons:** Bereksperimen kurang visual, harus menjalankan seluruh skrip, kurang dukungan di cloud vendors untuk notebook.

Workflow Umum: Proyek machine learning sering dimulai dengan notebook untuk eksperimen cepat, kemudian bagian-bagian kode yang paling berguna dipindahkan ke skrip Python.

PyTorch in the Wild: Banyak repositori kode proyek machine learning berbasis PyTorch menyertakan instruksi menjalankan kode PyTorch dalam bentuk skrip Python.

0. Mode Seluler vs. Mode Skrip

- *Mode Seluler:* Notebook mode seperti "05. Going Modular Part 1 (cell mode)" dijalankan seperti notebook biasa, di mana setiap sel dalam notebook dapat berisi kode atau markup.
- *Mode Skrip:* Notebook mode skrip seperti "05. Going Modular Part 2 (script mode)" mirip dengan mode seluler, tetapi banyak sel kode dapat diubah menjadi skrip Python. Meskipun tidak perlu membuat skrip Python melalui notebook, mode skrip digunakan untuk menunjukkan salah satu cara mengubah notebook menjadi skrip Python.

1. Mendapatkan Data

- Data diunduh dari GitHub menggunakan modul requests Python untuk mengunduh file .zip dan mengekstraknya.
- Data terdiri dari gambar pizza, steak, dan sushi dalam format klasifikasi gambar standar.
- Proses ini menghasilkan struktur direktori yang berisi folder "train" dan "test" untuk masing-masing kategori.

2. Membuat Datasets dan DataLoaders (data_setup.py)

- Kode pembuatan PyTorch Dataset dan DataLoader diubah menjadi fungsi bernama **create_dataloaders()** dan ditulis ke dalam file **data_setup.py** menggunakan **%%writefile**.
- Fungsi ini menerima jalur direktori untuk data pelatihan dan pengujian, transformasi, ukuran batch, dan jumlah pekerja untuk membuat DataLoaders.
- Fungsi ini mengembalikan tupel (train_dataloader, test_dataloader, class_names) yang berisi DataLoaders untuk pelatihan dan pengujian serta daftar nama kelas.

3. Membuat Model (model_builder.py)

- Model TinyVGG yang telah dibuat sebelumnya di notebook sebelumnya ditempatkan dalam file **model_builder.py** menggunakan **%%writefile**.
- File ini berisi definisi kelas **TinyVGG** yang menciptakan arsitektur TinyVGG dalam PyTorch.
- Sekarang, model TinyVGG dapat diimpor menggunakan **from going_modular import model_builder**.

4. Membuat Fungsi train_step() dan test_step() serta Fungsi train() untuk Menggabungkannya

- **train_step()**: Melatih model PyTorch untuk satu epoch dengan melakukan langkah-langkah pelatihan (forward pass, perhitungan kerugian, optimasi) pada DataLoader.
- **test_step()**: Mengevaluasi model PyTorch untuk satu epoch dengan melakukan forward pass pada dataset pengujian.
- **train()**: Menggabungkan **train_step()** dan **test_step()** untuk sejumlah epoch tertentu dan mengembalikan hasil dalam bentuk kamus.

5. Membuat Fungsi untuk Menyimpan Model (utils.py)

- **save_model()**: Menyimpan model PyTorch ke direktori target dengan fungsi utilitas. Menyimpan model ini dilakukan dengan menyertakan fungsi dalam file **utils.py** untuk menyimpan kode yang berulang.

6. Melatih, Mengevaluasi, dan Menyimpan Model (train.py)

- **train.py**: Berkumpulnya semua fungsionalitas sebelumnya (data_setup.py, engine.py, model_builder.py, utils.py) dalam satu skrip Python. Skrip ini digunakan untuk melatih model PyTorch dengan satu baris kode di baris perintah. Hyperparameter dan konfigurasi lainnya dapat diatur melalui argumen baris perintah atau secara langsung dalam skrip.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

06. PyTorch Transfer Learning

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Modul ini memperkenalkan penggunaan teknik transfer learning dalam deep learning dengan PyTorch.

- **Pentingnya Transfer Learning:**
  - Beberapa model yang telah dibangun sebelumnya memiliki kinerja yang buruk.
  - Transfer learning memungkinkan penggunaan pola (bobot) yang telah dipelajari oleh model lain untuk masalah kita sendiri.
  - Dengan transfer learning, kita dapat mengambil pola dari model computer vision pada dataset ImageNet dan menggunakannya untuk FoodVision Mini kita.
- **Apa itu Transfer Learning:**
  - Transfer learning memungkinkan kita menggunakan pola yang telah dipelajari oleh model pada masalah lain untuk masalah kita sendiri.
- **Manfaat Transfer Learning:**
  - Dapat memanfaatkan model yang sudah ada yang terbukti berhasil pada masalah serupa.
  - Dapat memanfaatkan model yang sudah memiliki pola pada data serupa dengan data kita sendiri, seringkali menghasilkan hasil yang baik dengan data kustom yang lebih sedikit.
- **Aplikasi Transfer Learning pada FoodVision Mini:**
  - Transfer learning akan diuji pada FoodVision Mini dengan menggunakan model computer vision yang telah dilatih pada ImageNet.

0. Persiapan Awal

- Langkah ini mencakup langkah-langkah untuk mengimpor/mengunduh modul yang diperlukan.
- Kode menggunakan beberapa skrip Python yang telah dibuat sebelumnya, seperti data_setup.py dan engine.py dari Modul 05: PyTorch Going Modular.

- Mengunduh direktori going_modular dari repositori pytorch-deep-learning jika belum ada.
- Menginstal torchinfo jika belum tersedia untuk memberikan representasi visual dari model kita.
- Menggunakan versi nightly dari torch dan torchvision (versi v0.13+) untuk menjalankan notebook ini.
- Setelah itu, dilakukan import dan setup reguler serta menentukan perangkat (device) yang akan digunakan (cuda jika tersedia, jika tidak, menggunakan cpu).

1. Dapatkan Data

- Sebelum menggunakan transfer learning, kita membutuhkan dataset.
- Kode ini mendownload dataset pizza_steak_sushi.zip dari GitHub kursus dan mengekstraknya jika belum ada.
- Dataset ini berisi gambar pizza, steak, dan sushi.

2. Buat Dataset dan DataLoader

- Karena sudah mengunduh direktori going_modular, kita dapat menggunakan skrip data_setup.py untuk menyiapkan DataLoaders.
- Terlepas dari itu, karena akan menggunakan model terlatih dari torchvision.models, perlu menyiapkan transformasi khusus untuk gambar.

2.1. Manual Creation

Membuat pipeline transformasi secara manual dengan menggabungkan beberapa transformasi menggunakan **transforms.Compose**. Transformasi ini mencakup resize, konversi nilai piksel ke rentang [0, 1], dan normalisasi dengan mean dan std tertentu.

2.2. Auto Creation

Menggunakan fitur transformasi otomatis yang diperkenalkan di torchvision v0.13+. Fitur ini memungkinkan kita mendapatkan transformasi yang sesuai dengan model terlatih yang dipilih tanpa harus membuatnya secara manual. Sebagai contoh, transformasi untuk model EfficientNet_B0 diambil dari bobot terlatih tersebut.

Selanjutnya, DataLoaders dibuat dengan menggunakan fungsi **create_dataloaders** dari data_setup.py, baik dengan transformasi manual maupun otomatis.

3. Mendapatkan Model yang Sudah Dipretraining

Dalam langkah ini, kita akan menggunakan model yang sudah dipretraining untuk menyelesaikan tugas klasifikasi gambar makanan dengan FoodVision Mini. Meskipun kita telah membangun jaringan saraf PyTorch dari awal dalam beberapa notebook sebelumnya, model-model tersebut belum memberikan performa yang sesuai dengan harapan.

Oleh karena itu, kita akan menerapkan transfer learning. Ide dasar dari transfer learning adalah mengambil model yang sudah baik kinerjanya dalam ruang masalah yang mirip dengan milik kita, lalu menyesuaikannya dengan kasus penggunaan kita.

Dalam konteks ini, karena kita bekerja pada masalah computer vision (klasifikasi gambar dengan FoodVision Mini), kita dapat menemukan model klasifikasi yang sudah dipretraining di torchvision.models.

3.1 Pemilihan Model yang Sudah Dipretraining

Pemilihan model yang sesuai tergantung pada masalah dan perangkat yang Kita gunakan. Umumnya, angka yang lebih tinggi pada nama model (contoh: **efficientnet_b0()** hingga **efficientnet_b7()**) menunjukkan performa yang lebih baik tetapi dengan ukuran model yang lebih besar.

Sebagai contoh, jika Kita ingin menjalankan model pada perangkat mobile, Kita perlu memperhatikan sumber daya komputasi yang terbatas pada perangkat tersebut dan mungkin memilih model yang lebih kecil.

Namun, jika Kita memiliki daya komputasi tanpa batas, Kita dapat memilih model yang lebih besar. Pemahaman tentang pertukaran performa vs. kecepatan vs. ukuran ini akan berkembang dengan waktu dan praktik.

3.2 Persiapan Model yang Sudah Dipretraining

Dalam contoh ini, kita akan menggunakan model **efficientnet_b0()** yang sudah dipretraining. Model ini memiliki tiga bagian utama: **features** (kumpulan lapisan konvolusional dan lapisan aktivasi lainnya untuk mempelajari representasi dasar data visual), **avgpool** (mengambil rata-rata keluaran dari lapisan fitur dan mengubahnya menjadi vektor fitur), dan **classifier** (mengubah vektor fitur menjadi vektor dengan dimensi sesuai dengan jumlah kelas keluaran yang dibutuhkan).

Untuk memasukkan berat ImageNet yang sudah dipretraining, kita dapat menggunakan kode yang sama dengan yang kita gunakan untuk membuat transformasi.

Penting untuk dicatat bahwa arsitektur model dapat berubah seiring waktu dengan rilis penelitian baru. Oleh karena itu, disarankan untuk bereksperimen dengan beberapa arsitektur dan menyesuaikannya dengan masalah Kita.

3.3 Ringkasan Model dengan torchinfo.summary()

Untuk memahami lebih lanjut tentang model, kita dapat menggunakan metode **torchinfo.summary()**. Ini membutuhkan model, ukuran input, kolom informasi yang diinginkan, lebar kolom, dan pengaturan baris.

3.4 Membekukan Bagian Dasar Model dan Mengubah Layer Output

Transfer learning melibatkan membekukan beberapa lapisan dasar dari model yang sudah dipretraining (biasanya bagian **features**) dan kemudian menyesuaikan lapisan keluaran (juga disebut lapisan kepala/klasifikasi) agar sesuai dengan kebutuhan kita.

Dalam contoh ini, kita membekukan semua lapisan dasar dengan mengatur **requires_grad=False** untuk setiap parameter dalam **model.features**. Selanjutnya, kita mengubah lapisan keluaran atau bagian klasifikasi model untuk sesuai dengan jumlah kelas yang kita miliki.

4. Melatih Model:

- Model yang telah di-pretrain sekarang semi-frozen dan memiliki klasifikasi yang disesuaikan. Kita akan menerapkan transfer learning.
- Untuk melatih, kita perlu membuat fungsi loss dan optimizer.
- Menggunakan **nn.CrossEntropyLoss()** untuk fungsi loss karena ini adalah masalah klasifikasi multi-kelas.
- Menggunakan **torch.optim.Adam()** sebagai optimizer dengan lr=0.001.
- Melatih model menggunakan fungsi **train()** yang telah didefinisikan sebelumnya.
- Melatih hanya parameter classifier, sementara parameter lain sudah difreeze.
- Menetapkan random seeds dan menghitung waktu total pelatihan.

5. Evaluasi Model dengan Plotting Kurva Loss:

- Model tampaknya berperforma dengan baik. Untuk mengevaluasi lebih lanjut, kita akan plotting loss curves dari hasil pelatihan.
- Menggunakan fungsi **plot_loss_curves()** yang dibuat sebelumnya.
- Loss curves menunjukkan bahwa baik loss pada dataset pelatihan maupun uji mengarah ke arah yang benar, demikian juga dengan nilai akurasi.

6. Prediksi pada Gambar dari Dataset Uji:

- Model telah melatih dengan baik secara kuantitatif, sekarang mari kita lihat secara kualitatif dengan membuat prediksi pada gambar dari dataset uji.

- Membuat fungsi **pred_and_plot_image()** untuk membuat prediksi pada gambar uji.
- Fungsi ini memastikan bahwa gambar yang akan diprediksi memiliki format yang sesuai dengan gambar yang digunakan untuk melatih model.
- Menggunakan beberapa gambar acak dari dataset uji dan memvisualisasikan prediksinya
- Hasil prediksi menunjukkan kualitas yang jauh lebih baik dibandingkan dengan model TinyVGG sebelumnya.

6.1 **Prediksi pada Gambar Kustom:**

- Melakukan prediksi pada gambar kustom ("pizza-dad.jpeg").
- Menggunakan fungsi **pred_and_plot_image()** untuk membuat prediksi pada gambar tersebut.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

07. PyTorch Experiment Tracking

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Modul 7 membahas tentang Pelacakan Eksperimen PyTorch. Pada perjalanan pembuatan FoodVision Mini (model klasifikasi gambar untuk mengklasifikasikan gambar pizza, steak, atau sushi), sejumlah model telah dilatih. Sejauh ini, pelacakan dilakukan melalui kamus Python atau perbandingan metrik selama pelatihan. Namun, jika kita ingin menjalankan lusinan model yang berbeda sekaligus, diperlukan cara yang lebih baik, yaitu pelacakan eksperimen.

Pentingnya pelacakan eksperimen dalam machine learning karena eksperimen sangat penting. Untuk mencari tahu apa yang berhasil dan apa yang tidak, kita perlu melacak hasil eksperimen yang berbeda.

1. **Apa itu Pelacakan Eksperimen:**
    - Machine learning dan deep learning bersifat eksperimental.
    - Diperlukan untuk menciptakan berbagai model dan melacak hasil kombinasi berbagai data, arsitektur model, dan metode pelatihan.
    - Pelacakan eksperimen membantu menentukan apa yang berhasil dan apa yang tidak.
2. **Mengapa Melacak Eksperimen:**
    - Jika hanya menjalankan beberapa model, melacak hasil dalam print out dan beberapa kamus mungkin cukup.
    - Namun, ketika jumlah eksperimen meningkat, cara ini mungkin sulit dijaga.
    - Pelacakan eksperimen menjadi sangat penting karena jumlah eksperimen yang berbeda dapat menjadi sulit dijaga.
3. **Berbagai Cara Melacak Eksperimen Machine Learning:**
    - Ada berbagai cara melacak eksperimen machine learning, dan beberapa di antaranya mencakup penggunaan kamus Python, file CSV, print out, TensorBoard, Weights & Biases Experiment Tracking, dan MLFlow.
    - Setiap metode memiliki kelebihan dan kekurangan serta biaya yang berbeda.

0. Persiapan

1. Mengunduh modul-modul yang diperlukan dengan memeriksa dan menginstal versi PyTorch dan torchvision yang sesuai.
2. Menyiapkan kode untuk penggunaan perangkat agnostik (device-agnostic) dengan menentukan perangkat (CPU atau GPU).
3. Membuat fungsi set_seeds() untuk mengatur biji acak untuk operasi-operasi torch.

1. Mendapatkan Data

1. Mengunduh dataset "pizza_steak_sushi" yang digunakan untuk eksperimen FoodVision Mini.
2. Membuat fungsi download_data() untuk mengunduh dan mengekstrak dataset jika belum ada.
3. Mengonfigurasi path untuk dataset.

2. Membuat Datasets dan DataLoaders

1. Mengonfigurasi transformasi untuk mengubah gambar menjadi tensor dan normalisasi sesuai format ImageNet.
2. Membuat DataLoader menggunakan transformasi yang dibuat secara manual.
3. Membuat DataLoader menggunakan transformasi yang dibuat secara otomatis dari pretrained weights.

2.1. Membuat DataLoaders dengan Transformasi yang Dibuat Secara Manual

1. Mengonfigurasi path direktori untuk data pelatihan dan pengujian.
2. Mengatur normalisasi ImageNet sebagai bagian dari transformasi manual.
3. Membuat pipeline transformasi manual menggunakan torchvision.transforms.Compose().
4. Membuat DataLoader menggunakan fungsi create_dataloaders() dari modul data_setup.

2.2. Membuat DataLoaders dengan Transformasi yang Dibuat Secara Otomatis

1. Mengonfigurasi path direktori untuk data pelatihan dan pengujian.
2. Memilih pretrained weights (contoh: EfficientNet_B0) dari torchvision.models.

3. Menggunakan weights tersebut untuk mendapatkan transformasi otomatis dengan memanggil metode transforms().
4. Membuat DataLoader menggunakan fungsi create_dataloaders() dari modul data_setup.

3. Mendapatkan Model Pretrained

1. Mengunduh pretrained weights untuk model EfficientNet_B0 dari torchvision.
2. Membuat model dengan weights tersebut, kemudian membekukan lapisan dasar (base layers) dan mengganti classifier head untuk sesuaikan dengan jumlah kelas.

4. Melatih Model dan Melacak Hasil

1. Menentukan fungsi loss (CrossEntropyLoss) dan optimizer (Adam) untuk pelatihan model.
2. Mengonfigurasi SummaryWriter untuk melacak hasil eksperimen menggunakan TensorBoard.
3. Menggunakan fungsi train() untuk melatih model dan melacak hasil menggunakan SummaryWriter.

5. Memantau Hasil Model pada TensorBoard

1. Menggunakan kelas SummaryWriter() untuk menyimpan hasil model dalam format TensorBoard secara default di direktori "runs/".
2. TensorBoard adalah program visualisasi yang dibuat oleh tim TensorFlow untuk melihat dan memeriksa informasi tentang model dan data.
3. Memungkinkan visualisasi hasil secara interaktif selama dan setelah pelatihan.
4. Cara mengakses TensorBoard:
    - Di VS Code: Tekan SHIFT + CMD + P, cari perintah "Python: Launch TensorBoard".
    - Di Jupyter dan Colab Notebooks: Pastikan TensorBoard terinstal, muat dengan %load_ext tensorboard, dan lihat hasil dengan %tensorboard --logdir DIR_WITH_LOGS.
    - Juga dapat mengunggah eksperimen ke tensorboard.dev untuk berbagi secara publik.

6. Membuat Fungsi Pembantu untuk Membangun Instance SummaryWriter()

1. Kelas SummaryWriter() mencatat informasi ke direktori yang ditentukan oleh parameter log_dir.
2. Membuat fungsi bantu create_writer() untuk membuat instance SummaryWriter() dengan log_dir yang disesuaikan untuk setiap eksperimen.
3. Log_dir mengandung timestamp, nama eksperimen, nama model, dan informasi tambahan (opsional).
4. Dapat menyesuaikan direktori log untuk melacak berbagai detail seperti tanggal eksperimen, nama eksperimen, nama model, dan informasi tambahan.
5. Memperbarui fungsi pelatihan (train()) untuk menerima parameter writer sehingga setiap eksperimen dapat menggunakan instance SummaryWriter() yang berbeda.

6.1. Memperbarui Fungsi train() untuk Menyertakan Parameter Writer

1. Menambahkan parameter writer ke fungsi train() untuk memberikan kemampuan menggunakan instance SummaryWriter() yang berbeda untuk setiap eksperimen.
2. Saat memanggil train(), dapat memasukkan writer yang berbeda untuk setiap eksperimen, yang menghasilkan direktori log yang berbeda.
3. Fungsi train() kini dapat aktif memperbarui instance SummaryWriter() yang digunakan setiap kali dipanggil.

7. Menyiapkan Serangkaian Eksperimen Pemodelan

Dalam langkah ini, kita melakukan serangkaian eksperimen pemodelan untuk meningkatkan model FoodVision Mini tanpa membuatnya terlalu besar. Pendekatan ini melibatkan mencoba berbagai kombinasi data, model, dan jumlah epoch. Berikut adalah langkah-langkah utamanya:

7.1. Eksperimen Pemodelan:

- Pertanyaan besar dalam pembelajaran mesin adalah jenis eksperimen apa yang sebaiknya dijalankan.
- Tidak ada batasan eksperimen yang dapat dijalankan, sehingga pembelajaran mesin menjadi menarik dan menakutkan pada saat yang sama.
- Pada langkah ini, Kita perlu mengenakan mantel ilmuwan dan mengikuti moto praktisi pembelajaran mesin: eksperimen, eksperimen, eksperimen!
- Setiap hiperparameter menjadi titik awal untuk eksperimen yang berbeda.

Beberapa eksperimen yang dapat dijalankan:

- Mengubah jumlah epoch.
- Mengubah jumlah lapisan/unit tersembunyi.

- Mengubah jumlah data.
- Mengubah tingkat pembelajaran.
- Mencoba augmentasi data yang berbeda.
- Memilih arsitektur model yang berbeda.

Dengan latihan dan menjalankan banyak eksperimen yang berbeda, Kita akan mulai membangun intuisi tentang apa yang mungkin membantu model Kita.

7.2. Eksperimen yang Akan Dijalankan:

- Tujuan adalah meningkatkan model FoodVision Mini tanpa membuatnya terlalu besar.
- Model ideal mencapai tingkat akurasi set uji tinggi (90%+), tetapi tidak memakan waktu terlalu lama untuk melatih/membuat inferensi.
- Eksperimen ini mencakup kombinasi:
    - Jumlah data yang berbeda (10% dari Pizza, Steak, Sushi vs. 20%).
    - Model yang berbeda (EfficientNetB0 vs. EfficientNetB2).
    - Waktu pelatihan yang berbeda (5 epoch vs. 10 epoch).
- Eksperimen dilakukan secara bertahap meningkatkan jumlah data, ukuran model, dan lama pelatihan.

7.3. Unduh Dataset yang Berbeda:

- Sebelum menjalankan serangkaian eksperimen, pastikan dataset sudah siap.
- Diperlukan dua bentuk set pelatihan:
    - Set pelatihan dengan 10% data gambar Pizza, Steak, Sushi dari Food101 (sudah dibuat sebelumnya).
    - Set pelatihan dengan 20% data gambar Pizza, Steak, Sushi dari Food101.
- Kedua dataset ini diunduh dari GitHub.

7.4. Transformasi Dataset dan Pembuatan DataLoader:

- Transformasi dibuat untuk menyiapkan gambar untuk model.
- DataLoader dibuat dengan batch size 32 untuk semua eksperimen.

7.5. Membuat Model Pengambil Fitur:

- Dua model pengambil fitur dibuat: EfficientNetB0 dan EfficientNetB2.
- Lapisan dasar (fitur) dibekukan, dan lapisan pengklasifikasi diubah sesuai dengan masalah klasifikasi gambar.
- EfficientNetB2 memerlukan penyesuaian pada jumlah fitur input akhir.

7.6. Membuat Eksperimen dan Menyiapkan Kode Pelatihan:

- Dua daftar dan satu kamus dibuat untuk jumlah epoch yang akan diuji, model yang akan diuji, dan DataLoader yang berbeda.
- Eksperimen dijalankan dengan mengubah model, jumlah epoch, dan DataLoader.
- Model yang dilatih disimpan setiap selesai eksperimen untuk kemudian dapat digunakan untuk prediksi.

8. Meninjau Eksperimen pada TensorBoard

- Menggunakan TensorBoard untuk visualisasi hasil eksperimen.
- Dapat dilihat bahwa model EffNetB0 yang dilatih selama 10 epoch dengan 20% data mencapai kerugian uji terendah.
- Visualisasi hasil eksperimen dapat diunggah ke tensorboard.dev untuk dibagikan secara publik.

9. Memuat Model Terbaik dan Melakukan Prediksi

- Menganalisis log TensorBoard dan menemukan bahwa eksperimen kedelapan mencapai hasil terbaik secara keseluruhan (akurasi uji tertinggi, kerugian uji kedua terendah).
- Model terbaik adalah EffNetB2 dengan parameter gkita, 20% data pelatihan pizza, steak, sushi, dan dilatih selama 10 epoch.
- Meskipun hasilnya tidak jauh lebih baik dari model lain, model yang sama pada data yang sama mencapai hasil serupa dalam setengah waktu pelatihan.
- Memuat model terbaik yang telah disimpan dan memeriksa ukuran file model.
- Melakukan prediksi pada gambar uji yang tidak pernah dilihat sebelumnya.

9.1. Memprediksi pada Gambar Kustom dengan Model Terbaik

- Menampilkan prediksi pada gambar kustom (gambar "pizza dad") menggunakan model terbaik.
- Model terbaik berhasil memprediksi "pizza" dengan tingkat kepercayaan yang tinggi (0.978).

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

08. PyTorch Paper Replicating

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

Modul ini memperkenalkan Milestone Project 2, yang fokus pada replikasi sebuah makalah penelitian dalam bidang pembelajaran mesin menggunakan PyTorch. Proyek ini akan menciptakan Vision Transformer (ViT) dari awal dan mengaplikasikannya pada permasalahan FoodVision Mini.

- **Tujuan Paper Replicating:** Replikasi makalah penelitian merupakan upaya untuk mengimplementasikan teknik-teknik terkini dalam pembelajaran mesin ke dalam kode sehingga dapat digunakan untuk permasalahan khusus.
- **Struktur Makalah Penelitian:** Sebuah makalah penelitian pembelajaran mesin umumnya terdiri dari Abstract, Introduction, Method, Results, Conclusion, References, dan Appendix. Replikasi makalah melibatkan pengubahan gambar, diagram, rumus matematika, dan teks menjadi kode yang dapat digunakan, khususnya menggunakan PyTorch.
- **Alasan Replikasi Makalah:** Makalah penelitian pembelajaran mesin mencakup hasil penelitian tim yang memerlukan berbulan-bulan hingga bertahun-tahun kerja. Replikasi makalah tersebut dapat memberikan wawasan yang berharga dan menjadi latihan yang bagus untuk meningkatkan keterampilan pembelajaran mesin.
- **George Hotz Quote:** George Hotz, pendiri comma.ai, mendorong untuk meningkatkan keterampilan sebagai insinyur pembelajaran mesin dengan mendownload makalah, mengimplementasikannya, dan terus melakukannya untuk memperoleh keterampilan.
- **Tantangan Replikasi:** Meskipun menantang, banyak alat dan perpustakaan pembelajaran mesin seperti HuggingFace, PyTorch Image Models (timm library), dan fast.ai lahir untuk membuat penelitian pembelajaran mesin lebih dapat diakses.
- **Sumber Referensi Makalah:** Ada beberapa tempat untuk menemukan contoh kode untuk makalah penelitian pembelajaran mesin, termasuk arXiv, AK Twitter, Papers with Code, dan GitHub repository seperti vit-pytorch dari lucidrains.

Langkah 0: Persiapan

- Memastikan modul yang diperlukan telah diunduh.
- Mengimpor skrip Python seperti data_setup.py dan engine.py dari direktori going_modular yang diunduh dari repositori pytorch-deep-learning.
- Memastikan instalasi torchinfo dan torchvision versi 0.13+, serta versi torch 1.12+.
- Jika versi tidak sesuai, mengunduh dan menginstal versi yang diperlukan.
- Melanjutkan dengan impor modul, mengatur kode yang agnostik perangkat, dan mendapatkan skrip helper_functions.py dari GitHub.

1. Mendapatkan Data

- Mengunduh dataset gambar pizza, steak, dan sushi menggunakan fungsi download_data() dari helper_functions.py.
- Menyiapkan jalur direktori untuk gambar pelatihan dan pengujian.

2. Membuat Dataset dan DataLoader

- Membuat transformasi untuk mempersiapkan gambar, dengan mengatur ukuran gambar sesuai dengan yang disebutkan dalam ViT paper.
- Membuat DataLoader untuk pelatihan dan pengujian menggunakan fungsi create_dataloaders() dari data_setup.py.

2.1 Persiapkan Transformasi Gambar

- Buat ukuran gambar (IMG_SIZE) sesuai dengan yang tercantum pada Tabel 3 di paper Vision Transformer (ViT).
- Buat pipeline transformasi manual yang melibatkan Resize dan ToTensor untuk persiapan data.

2.2 Ubah Gambar menjadi DataLoader

- Buat DataLoader menggunakan fungsi create_dataloaders() dari data_setup.py.
- Tetapkan BATCH_SIZE sesuai kebutuhan, dan gunakan transformasi manual yang telah dibuat sebelumnya.
- Pin memory diaktifkan (pin_memory=True) untuk mempercepat komputasi.

2.3 Visualisasi Satu Gambar

- Dengan menggunakan DataLoader, dapatkan batch gambar dan label dari pelatihan.
- Pilih satu gambar dan label dari batch.
- Visualisasikan gambar dan label menggunakan matplotlib.

3. Visualisasi Satu Gambar

1. Mendapatkan satu batch gambar dan label dari DataLoader pelatihan.
2. Memilih satu gambar dan label dari batch tersebut.
3. Menampilkan bentuk batch dan label.
4. Menampilkan gambar dan label menggunakan matplotlib.

3.1 Input dan Output, Layer, dan Blok

- ViT adalah arsitektur jaringan saraf mendalam yang terdiri dari beberapa lapisan dan blok.
- Lapisan adalah bagian dari arsitektur yang mengambil input, melakukan operasi, dan menghasilkan output.
- Blok adalah kumpulan lapisan, yang juga mengambil input dan menghasilkan output.

3.2 Penguraian Arsitektur ViT

- Arsitektur ViT terdiri dari beberapa tahapan: a. Patch + Position Embedding (Input): Mengubah gambar input menjadi urutan patch gambar dan menambahkan nomor posisi. b. Linear projection of flattened patches (Embedded Patches): Merubah patch gambar menjadi embedding menggunakan proyeksi linear. c. Norm: Normalisasi lapisan (LayerNorm) digunakan untuk mengurangi overfitting. d. Multi-Head Attention (MSA): Lapisan self-attention yang menerima input dari LayerNorm. e. MLP (Multilayer Perceptron): Blok MLP yang terdiri dari dua lapisan Linear dengan fungsi aktivasi GELU di antaranya. f. Transformer Encoder: Kumpulan lapisan yang terdiri dari MSA dan MLP, dengan koneksi skip antar-lapisan. g. MLP Head: Lapisan keluaran arsitektur, mengonversi fitur yang dipelajari menjadi kelas keluaran.

3.2.1 Eksplorasi Gambar 1

- Gambar 1 dari paper ViT memberikan gambaran tentang model secara grafis.
- Fokus pada lapisan, input, output, blok, dan koneksi antar-lapisan.

3.2.2 Eksplorasi Empat Persamaan

- Empat persamaan di bagian 3.1 menyajikan matematika di balik empat komponen utama arsitektur ViT.
- Terkait dengan koneksi dan output dari lapisan dan blok pada Gambar 1.

3.2.3 Eksplorasi Persamaan 1

- Persamaan 1 membahas token kelas, embedding patch, dan embedding posisi input gambar.

3.2.4 Eksplorasi Persamaan 2

- Persamaan 2 menjelaskan bahwa setiap lapisan terdiri dari blok Multi-Head Attention (MSA) dan blok LayerNorm.

3.2.5 Eksplorasi Persamaan 3

- Persamaan 3 menyatakan bahwa setiap lapisan juga memiliki blok Multilayer Perceptron (MLP) dengan LayerNorm.

3.2.6 Eksplorasi Persamaan 4

- Persamaan 4 menggambarkan bahwa output akhir (y) dari arsitektur adalah hasil LayerNorm dari output lapisan terakhir (L).

3.2.7 Eksplorasi Tabel 1

- Tabel 1 menyajikan detail hyperparameter untuk model ViT berbagai ukuran (ViT-Base, ViT-Large, ViT-Huge).
- Hyperparameter mencakup jumlah lapisan, dimensi embedding, ukuran MLP, jumlah kepala, dan jumlah parameter.

4. Equation 1: Split data into patches and creating the class, position, and patch embedding

Mengimplementasikan Equation 1 untuk membagi data menjadi potongan dan membuat embedding kelas, posisi, dan potongan. Proses ini dimulai dengan membuat embedding untuk potongan gambar (patch embedding) pada arsitektur Vision Transformer (ViT)

4.1 **Menghitung Dimensi Input dan Output Patch Embedding Secara Manual:**

- Menciptakan nilai-nilai contoh untuk parameter seperti tinggi (height), lebar (width), jumlah saluran warna (color_channels), dan ukuran patch (patch_size).
- Menghitung jumlah patch (number_of_patches) berdasarkan parameter-parameter tersebut.

4.2 Mengonversi Image Menjadi Patch:

- Memvisualisasikan pengubahan satu gambar menjadi serangkaian patch.
- Melihat bagaimana top row dari piksel-piksel yang di-patch dapat divisualisasikan.

4.3 Membuat Patch Embedding dengan torch.nn.Conv2d():

- Menunjukkan cara menggunakan layer **torch.nn.Conv2d()** untuk membuat patch dari gambar.
- Menciptakan convolutional layer dengan kernel_size dan stride yang sesuai dengan ukuran patch.

- Menyaring gambar melalui convolutional layer untuk mendapatkan serangkaian feature map.

4.4 Meratakan Patch Embedding dengan torch.nn.Flatten():

- Menggunakan **torch.nn.Flatten()** untuk meratakan feature map menjadi urutan satu dimensi.
- Menyesuaikan dimensi tensor hasil untuk sesuai dengan bentuk keluaran yang diinginkan.

4.5 Mengubah Layer Penyematan Peta ViT menjadi Modul PyTorch

Bagian ini membahas pembuatan layer penyematan peta untuk ViT dalam bentuk modul PyTorch. Melibatkan subclassing dari **nn.Module** dan mencakup langkah-langkah berikut:

1. Membuat kelas **PatchEmbedding** yang merupakan subclass dari **nn.Module**.
2. Menginisialisasi kelas dengan parameter **in_channels=3**, **patch_size=16** (untuk ViT-Base), dan **embedding_dim=768**.
3. Membuat layer untuk mengubah gambar menjadi potongan menggunakan **nn.Conv2d**.
4. Membuat layer untuk meratakan peta fitur potongan menjadi satu dimensi menggunakan **nn.Flatten**.
5. Mendefinisikan metode **forward** untuk melewati input melalui layer yang dibuat pada langkah 3 dan 4.
6. Memastikan bentuk output sesuai dengan yang dibutuhkan oleh arsitektur ViT.

4.6 Membuat Penyemat Kelas Token

Pembahasan tentang pembuatan penyemat kelas token (class token) yang diperlukan dalam arsitektur ViT. Ini melibatkan pendekatan mirip dengan BERT, dengan menambahkan penyemat kelas pada awal urutan potongan gambar yang disematkan.

4.7 Membuat Penyemat Posisi

Membahas penciptaan penyemat posisi (position embedding) untuk menyimpan informasi posisi dalam urutan potongan gambar yang disematkan. Ini mencakup penggunaan penyemat posisi 1D yang dapat dipelajari.

4.8 Menggabungkan Semua Komponen: Dari Gambar ke Penyematan

Langkah-langkah akhir dalam menciptakan penyematan gambar dari input hingga output, sesuai dengan persamaan 1 dari bagian 3.1 dalam paper ViT. Proses ini melibatkan:

1. Menentukan ukuran potongan.
2. Mendapatkan gambar tunggal, mencetak bentuknya, dan menyimpan tinggi dan lebar.
3. Menambah dimensi batch pada gambar tunggal.

4. Membuat layer penyemat potongan menggunakan **PatchEmbedding**.
5. Melewati gambar melalui layer penyemat potongan.
6. Membuat penyemat kelas token.
7. Menyisipkan penyemat kelas token pada potongan gambar yang telah dibuat.
8. Membuat penyemat posisi.
9. Menambahkan penyemat posisi pada penyemat kelas token dan potongan gambar.

5. Membangun Transformer Encoder

Bagian ini fokus pada pembangunan Transformer Encoder, bagian kunci dalam arsitektur ViT. Transformer Encoder digunakan untuk mengolah penyematan peta gambar yang dihasilkan sebelumnya.

5.1 Membuat Layer Normalisasi dan Multi-Head Attention

Pembahasan dimulai dengan membuat layer normalisasi dan lapisan Multi-Head Attention.

1. Membuat kelas **MultiHeadSelfAttention** yang merupakan subclass dari **nn.Module**.
2. Menginisialisasi kelas dengan parameter **embedding_dim=768** dan **num_heads=12** (sesuai dengan ViT-Base).
3. Membuat lapisan Q, K, dan V untuk setiap kepala perhatian menggunakan **nn.Linear**.
4. Menggabungkan hasil Q, K, dan V untuk setiap kepala perhatian menggunakan fungsi **view** dan **transpose**.
5. Menerapkan perhatian self attention menggunakan matriks perhatian dan fungsi softmax.
6. Menggabungkan hasil self attention dari setiap kepala menggunakan lapisan linear.

5.2 Menerapkan Layer Normalisasi

Melibatkan implementasi layer normalisasi untuk mengatasi masalah munculnya nilai yang sangat besar atau sangat kecil selama pelatihan.

1. Membuat kelas **MLPHead** sebagai subclass dari **nn.Module**.
2. Menginisialisasi kelas dengan parameter **embedding_dim=768** dan **mlp_dim=3072** (sesuai dengan ViT-Base).
3. Membuat dua lapisan linear untuk ekspansi dan pemampatan.
4. Menambahkan fungsi aktivasi GELU dan dropout di antara kedua lapisan.

5.3 Menerapkan Multi-Head Attention

Menerapkan lapisan Multi-Head Attention untuk memproses penyematan peta gambar. Ini melibatkan pemisahan penyematan peta menjadi beberapa kepala untuk memahami hubungan antara potongan gambar.

1. Membuat kelas **TransformerEncoderBlock** sebagai subclass dari **nn.Module**.
2. Menginisialisasi kelas dengan parameter **embedding_dim=768**, **num_heads=12**, dan **mlp_dim=3072**.
3. Menerapkan lapisan normalisasi dan dropout pada input.
4. Meneruskan input melalui lapisan Multi-Head Self Attention.
5. Menambahkan koneksi residu dan normalisasi setelah lapisan pertama.
6. Meneruskan output melalui lapisan MLP Head.
7. Menambahkan koneksi residu dan normalisasi setelah lapisan kedua.
8. Menghasilkan output blok Transformer Encoder.

6. Membangun MLP Head

Bagian ini membahas pembuatan MLP Head, yang bertanggung jawab untuk mengubah keluaran dari Transformer Encoder menjadi representasi yang dapat digunakan untuk klasifikasi.

6.1 Menerapkan MLP Head

Implementasi MLP Head untuk mengubah output dari Transformer Encoder menjadi representasi yang sesuai untuk klasifikasi.

1. Membuat kelas **VisionTransformer** sebagai subclass dari **nn.Module**.
2. Menginisialisasi kelas dengan parameter arsitektur ViT, seperti **num_classes=1000** dan **num_layers=12** (untuk ViT-Base).
3. Membuat layer penyemat gambar, blok Transformer Encoder, dan lapisan klasifikasi akhir.
4. Meneruskan input gambar melalui layer penyemat, blok Transformer Encoder sebanyak **num_layers**, dan lapisan klasifikasi.

6.2 Menyusun Seluruh Arsitektur Model

Penggabungan semua komponen yang telah dibuat sebelumnya, termasuk penyematan gambar, Transformer Encoder, dan MLP Head, untuk membentuk arsitektur model ViT secara keseluruhan.

1. Menginisialisasi bobot model dengan inisialisasi He.

2. Menerapkan metode forward yang meneruskan input melalui layer penyemat, blok Transformer Encoder, dan lapisan klasifikasi.

7. Melatih dan Mengevaluasi Model

Bagian ini mencakup langkah-langkah untuk melatih dan mengevaluasi model Vision Transformer setelah keseluruhan arsitektur selesai dibangun.

7.1 Menyiapkan Data Pelatihan dan Pengujian

Langkah-langkah untuk menyiapkan data pelatihan dan pengujian yang akan digunakan untuk melatih dan mengevaluasi model.

1. Mendefinisikan transformasi data untuk augmentasi.
2. Menggunakan dataset ImageNet untuk pelatihan.

7.2 Melatih dan Mengevaluasi Model

Proses melatih model Vision Transformer menggunakan data pelatihan yang telah disiapkan sebelumnya, diikuti dengan evaluasi performa model pada data pengujian.

1. Menginisialisasi model, optimizer, dan fungsi kerugian.
2. Menerapkan loop pelatihan dengan perulangan epoch.
3. Melakukan pelatihan pada setiap batch gambar.
4. Menghitung loss dan melakukan backpropagation.
5. Evaluasi model pada set pengujian.

8. Menyatukan Semuanya untuk Membuat ViT

Setelah melalui banyak langkah, saatnya menyatukan semua komponen menjadi arsitektur Vision Transformer (ViT) yang lengkap. Proses ini melibatkan langkah-langkah berikut:

8.1 Persiapan ViT untuk Pelatihan

Menggabungkan semua blok yang telah dibuat ke dalam kelas **ViT** yang merupakan subclass dari **nn.Module**. Proses ini mencakup:

- Inisialisasi kelas dengan hyperparameter dari Tabel 1 dan Tabel 3.
- Membuat embedding token dan position embedding.
- Menyusun blok Transformer Encoder dalam urutan yang tepat.
- Membuat lapisan klasifikasi akhir (MLP Head).
- Menerapkan metode forward untuk meneruskan input melalui seluruh arsitektur.

8.2 Inisialisasi dan Visualisasi Model ViT

Melakukan inisialisasi bobot model dengan inisialisasi He dan menyajikan visualisasi model ViT menggunakan **torchinfo.summary()** untuk mendapatkan gambaran input dan output dari setiap lapisan.

9. Persiapan Kode Pelatihan untuk Model ViT

9.1 Pembuatan Optimizer

Membuat optimizer untuk mengoptimalkan parameter-model ViT menggunakan Adam optimizer. Menetapkan nilai hyperparameter seperti laju belajar, nilai beta, dan weight decay sesuai dengan yang dijelaskan dalam bagian 9.1.

9.2 Pembuatan Fungsi Kerugian

Menggunakan fungsi kerugian **torch.nn.CrossEntropyLoss()** untuk tugas klasifikasi multi-kelas.

9.3 Pelatihan Model ViT

Menyusun dan melatih model ViT dengan menggunakan fungsi pelatihan **train()** dari skrip **engine.py**. Memasukkan optimizer, fungsi kerugian, dan dataloader pelatihan dan pengujian ke dalam fungsi pelatihan.

9.4 Yang Hilang dalam Pengaturan Pelatihan

Menyadari bahwa hasil pelatihan mungkin tidak optimal karena perbedaan skala dengan paper ViT asli. Berbagai faktor seperti jumlah data, jumlah epoch, dan batch size yang lebih kecil dapat mempengaruhi hasil pelatihan.

9.5 Plot Kurva Kerugian Model ViT

Menggunakan fungsi **plot_loss_curves** dari **helper_functions.py** untuk memvisualisasikan kurva kerugian model ViT selama pelatihan.

10. Menggunakan model ViT yang sudah di-pretrain dari torchvision.models pada dataset yang sama:

- Pembahasan tentang manfaat menggunakan model yang sudah di-pretrain di bagian sebelumnya.
- Pada dasarnya, penggunaan transfer learning (menggunakan model yang sudah di-pretrain) sangat bermanfaat, terutama ketika pelatihan model sendiri dari awal tidak memberikan hasil optimal.

10.1 Mengapa menggunakan model yang sudah di-pretrain:

- Banyak penelitian machine learning modern menggunakan dataset besar dan sumber daya komputasi yang besar.
- Original ViT yang sudah terlatih mungkin tidak dianggap sebagai setup pelatihan "super besar" di era machine learning modern.
- Penggunaan model ViT-L/16 yang sudah di-pretrain pada dataset ImageNet-21k umumnya memberikan hasil baik dengan menggunakan sumber daya yang lebih sedikit.

10.2 Mendapatkan model ViT yang sudah di-pretrain dan membuat feature extractor:

- Mendapatkan model ViT yang sudah di-pretrain dari torchvision.models.
- Menyiapkan kode yang agnostik perangkat.
- Membuat model ViT-Base dengan patch size 16 sebagai feature extractor untuk FoodVision Mini.

10.3 Menyiapkan data untuk model ViT yang sudah di-pretrain:

- Mengunduh dan membuat DataLoaders untuk model ViT kita sendiri sebelumnya.
- Mengunduh gambar pizza, steak, dan sushi untuk Food Vision Mini.
- Transformasi gambar menjadi tensors dan DataLoaders.

10.4 Melatih model feature extractor ViT:

- Menggunakan optimizer Adam dengan learning rate 1e-3 dan loss function CrossEntropyLoss.
- Menggunakan engine.train() untuk melatih model.

10.5 Plot kurva loss model feature extractor ViT:

- Plot kurva loss model feature extractor ViT.

10.6 Menyimpan model feature extractor ViT dan memeriksa ukuran file:

- Menyimpan model menggunakan utils.save_model().
- Memeriksa ukuran file model ViT feature extractor.

11. Melakukan prediksi pada gambar kustom:

- Mengunduh gambar pizza dad dan menggunakan model ViT feature extractor untuk melakukan prediksi.

Zero to Mastery Learn PyTorch for Deep Learning Course Summary

09. PyTorch Model Deployment

Nama : Al Ghifary Akmal Nasheeri

NIM : 1103201242

Kelas : TK-44-06

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.

7. Membuat Demo FoodVision Mini dengan Gradio

7.1. Gradio Overview:

- 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.

7.2. 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.

7.3. Membuat Daftar Gambar Contoh:

- Membuat daftar gambar contoh untuk digunakan dalam demo Gradio.
- Contoh berisi daftar file path gambar dari direktori pengujian.

7.4. Membangun Antarmuka Gradio:

- 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.

8. Mengubah Demo FoodVision Mini Gradio Menjadi Aplikasi yang Dapat Dideploy

8.1. Apa Itu Hugging Face Spaces:

- Hugging Face Spaces adalah sumber daya yang memungkinkan hosting dan berbagi aplikasi machine learning.
- Spaces memungkinkan berbagi dan mendemo model-machine learning dengan mudah.

8.2. Struktur Aplikasi Gradio yang Dideploy:

- Untuk mendeploy demo Gradio, semua file terkait disimpan dalam satu direktori.
- Struktur direktori mencakup model, script aplikasi, contoh gambar, dan file requirements.

8.3. Membuat Direktori untuk Menyimpan File Aplikasi:

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

8.4. Membuat Direktori Contoh Gambar untuk Demo:

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

8.5. Memindahkan Model EffNetB2:

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

8.6. Membuat Script Python untuk Model (model.py):

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

8.7. Membuat Script Python untuk Aplikasi Gradio (app.py):

- Membuat script **app.py** yang menyatukan bagian-bagian utama untuk membuat demo Gradio.
- Menyiapkan model, fungsi prediksi, dan mengatur antarmuka Gradio.

8.8. Membuat File Requirements (requirements.txt):

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

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.