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.