Muhammad Irfan Al Rasyid
1103200080

00. PyTorch Fundamentals

1. Pengantar PyTorch
PyTorch adalah kerangka kerja sumber terbuka untuk machine learning dan deep learning. Artikel ini menjelaskan peran PyTorch dalam memfasilitasi pengembangan algoritma dan manipulasi data dengan menggunakan kode Python.

2. Penggunaan PyTorch oleh Perusahaan Terkemuka
Perusahaan teknologi besar seperti Meta (Facebook), Tesla, dan Microsoft, serta entitas penelitian kecerdasan buatan seperti OpenAI, mengandalkan PyTorch dalam menggerakkan penelitian dan mengintegrasikan machine learning ke dalam produk-produk mereka.

3. Aplikasi PyTorch di Berbagai Industri
Selain digunakan di industri teknologi, PyTorch juga menemukan aplikasi di sektor pertanian, khususnya dalam mendukung computer vision pada traktor.

4. Keunggulan PyTorch Menurut Peneliti
PyTorch mendapat keunggulan di mata peneliti machine learning. Sebagai contoh, PyTorch menjadi kerangka kerja deep learning paling banyak digunakan pada Papers With Code per Februari 2022.

5. Modul Kursus dan Struktur Notebooks
Kursus ini terdiri dari beberapa modul (notebooks), dengan setiap notebook membahas konsep-konsep penting dalam PyTorch. Struktur modul dimulai dari pengenalan tensor, yang merupakan blok bangunan dasar dalam machine learning dan deep learning.

6. Konsep Dasar: Tensor
Notebook ini membahas konsep dasar machine learning dan deep learning, yaitu tensor. Beberapa topik yang dicakup termasuk pengenalan tensor, pembuatan tensor, dan mendapatkan informasi dari tensor.

7. Operasi pada Tensor: Manipulasi dan Bentuk
Modul-modul berikutnya membahas operasi lebih lanjut pada tensor, termasuk manipulasi tensor seperti penambahan, perkalian, dan kombinasi. Terdapat juga pembahasan mengenai penanganan bentuk tensor.

8. Penggunaan Tensor di Industri
Notebook tertentu membahas penggunaan tensor di industri, termasuk bagaimana tensor digunakan dalam computer vision pada traktor di sektor pertanian.

9. Bantuan dan Komunitas PyTorch

Pengguna PyTorch dapat memperoleh bantuan melalui GitHub kursus dan forum pengembang PyTorch. Artikel juga memberikan informasi mengenai impor PyTorch dan kompatibilitas versi yang disarankan.

11. Pengantar Tensors
Setelah mengimpor PyTorch, saatnya untuk memahami tentang tensors. Tensors merupakan blok bangunan dasar dalam machine learning, berfungsi untuk merepresentasikan data dalam bentuk numerik.

12. Membuat Tensors
PyTorch sangat menyukai tensors. Terdapat halaman dokumentasi khusus untuk kelas torch.Tensor. Kita dapat membuat tensor pertama kita dalam bentuk scalar (angka tunggal) dengan dimensi nol.

13. Scalar, Vector, dan Matrix
Scalar adalah angka tunggal, vector adalah tensor satu dimensi yang dapat berisi banyak angka, dan matrix adalah array dua dimensi. Kita dapat membuat tensor dalam bentuk vector dan matrix menggunakan PyTorch.

14. Tensor Tiga Dimensi
Kita dapat membuat tensor tiga dimensi, yang merupakan bentuk tensor yang lebih kompleks. Tensors dapat merepresentasikan berbagai jenis data dan memiliki dimensi yang dapat disesuaikan.

15. Tensors Acak, Nol, dan Satu
Dalam machine learning, seringkali kita memulai dengan tensors acak dan memperbarui angka-angka ini saat model melibatkan data. PyTorch menyediakan fungsi untuk membuat tensors acak, nol, dan satu.

16. Torch.arange() dan Torch.zeros_like()
Fungsi Torch.arange() digunakan untuk membuat tensor dengan rentang nilai tertentu. Selain itu, kita dapat membuat tensor yang berisi nol atau satu dengan menggunakan Torch.zeros_like() dan Torch.ones_like().

17. Jenis Data Tensors
PyTorch memiliki berbagai jenis data tensor, seperti float32, float16, dan float64, serta integer dengan berbagai presisi. Pemahaman jenis data tensor penting karena memengaruhi performa dan presisi perhitungan dalam deep learning.

18. Permasalahan Umum: Jenis Data dan Perangkat
Ketika bekerja dengan tensors, seringkali kita menghadapi masalah terkait jenis data dan perangkat. Penting untuk memastikan tensors memiliki jenis data dan perangkat yang sesuai untuk menghindari kesalahan dan masalah kompatibilitas.

19. Kesimpulan dan Sumber Daya
Artikel ini memberikan gambaran tentang konsep dasar tensors, pembuatan tensors dengan PyTorch, dan beberapa isu umum yang mungkin dihadapi. Pemahaman ini menjadi dasar untuk eksplorasi lebih lanjut dalam machine learning menggunakan PyTorch. Sumber daya

seperti dokumentasi PyTorch dan Wikipedia dapat membantu mendalami topik-topik tertentu.

01. PyTorch Workflow Fundamentals

Bab ini memfokuskan pada Fondasi Alur Kerja PyTorch (PyTorch Workflow Fundamentals) dengan penekanan khusus pada pemahaman dasar tentang regresi linear. Langkah pertama yang ditekankan adalah persiapan data, di mana beragam tipe data machine learning, mulai dari tabel angka hingga gambar, dijelaskan dalam konteks pengubahannya menjadi representasi angka dan pemilihan atau pembangunan model untuk memahami representasi tersebut. Pembahasan selanjutnya mencakup pembuatan data sintetis, dengan contoh penggunaan PyTorch untuk membuat tensor mewakili fitur input (X) dan label (y) dalam sebuah model regresi linear.

Pentingnya pemisahan data menjadi set latih dan uji ditekankan untuk mendukung proses pelatihan dan evaluasi model. Proses pembuatan model regresi linear menggunakan nn.Module PyTorch juga dibahas, dengan pemaparan mengenai langkah maju untuk menghitung prediksi. Loop pelatihan diimplementasikan untuk melatih model selama beberapa epoch, menggunakan fungsi loss (Mean Absolute Error - MAE) dan optimizer (Stochastic Gradient Descent - SGD) untuk memperbarui parameter model dan mengurangi kerugian.

Selanjutnya, dilakukan loop pengujian untuk mengevaluasi model pada set uji terpisah, dengan pemantauan kerugian MAE selama pelatihan dan pengujian. Visualisasi data menjadi elemen kunci, yang melibatkan pembuatan fungsi untuk memvisualisasikan data latih dan uji serta prediksi model. Bab ini juga membahas proses inferensi, menunjukkan cara membuat prediksi dengan model yang telah dilatih, dan memerinci evaluasi model dengan membandingkan parameter yang dipelajari dengan parameter asli pada data sintetis.

Terakhir, bab ini mengeksplorasi cara penyimpanan dan pemuatan model PyTorch, memberikan wawasan tentang bagaimana menyimpan model yang telah dilatih untuk digunakan di masa depan. Secara keseluruhan, bab ini menyajikan pandangan komprehensif terkait alur kerja PyTorch yang penting untuk pemahaman regresi linear, mencakup seluruh proses dari persiapan data hingga evaluasi model.

02. PyTorch Neural Network Classification

Penelitian ini membahas peran fungsi aktivasi non-linear dalam jaringan saraf tiruan (neural networks), khususnya pada model klasifikasi multi-kelas menggunakan PyTorch. Fungsi aktivasi non-linear memiliki peran penting dalam memodelkan pola data yang kompleks dan non-linear. Eksperimen dilakukan dengan mereplikasi fungsi aktivasi non-linear, seperti ReLU dan sigmoid, menggunakan PyTorch.

Dalam implementasi ReLU, nilai negatif pada data diubah menjadi nol, sementara implementasi sigmoid menghasilkan nilai antara 0 dan 1. Kedua fungsi aktivasi ini memberikan pemahaman visual tentang bagaimana data dapat diubah oleh fungsi non-linear.

Selanjutnya, model klasifikasi multi-kelas dibangun dengan memanfaatkan dataset yang dihasilkan menggunakan metode make_blobs() dari Scikit-Learn. Data diubah menjadi tensor dan dibagi menjadi set pelatihan dan pengujian.

Model jaringan saraf tiruan dibangun dengan menggunakan kelas BlobModel yang dirancang khusus untuk menangani masalah klasifikasi multi-kelas. Fungsi aktivasi non-linear seperti ReLU diterapkan pada lapisan tersembunyi model. Fungsi softmax digunakan untuk mengonversi keluaran model berupa logits menjadi probabilitas prediksi.

Pembelajaran model melibatkan perhitungan loss, akurasi, dan optimasi parameter-model menggunakan CrossEntropyLoss sebagai fungsi loss dan SGD sebagai optimizer. Evaluasi dilakukan pada set pengujian setiap 10 epoch selama 100 epoch.

Hasil eksperimen menunjukkan bahwa model yang dihasilkan mencapai akurasi tinggi, yakni 99.5%, pada set pengujian. Prediksi model berhasil dikonversi menjadi label kelas, menunjukkan keefektifan implementasi fungsi aktivasi non-linear dalam menangani masalah klasifikasi multi-kelas.

Penelitian ini memberikan pemahaman yang lebih dalam tentang peran fungsi aktivasi non-linear dalam konteks model klasifikasi multi-kelas, dengan implementasi praktis menggunakan PyTorch.

03. PyTorch Computer Vision

Dalam eksplorasi ini, kami membangun tiga model berbeda untuk tugas klasifikasi dataset yang diberikan. Model pertama adalah jaringan saraf dasar dengan dua lapisan linear. Model kedua identik dengan Model pertama, namun dengan penambahan fungsi aktivasi ReLU di antara lapisan linear. Model ketiga, mengadopsi arsitektur Convolutional Neural Network (CNN) yang terinspirasi oleh TinyVGG dari CNN Explainer.

Setelah melalui fase pelatihan dan pengujian, kami menganalisis performa ketiga model. Meskipun Model 2, yang menggunakan arsitektur CNN, menunjukkan peningkatan marginally dalam akurasi, perbedaannya tidak signifikan secara statistik dibandingkan dengan model linear. Hasil ini menyoroti kompleksitas permasalahan dataset dan menimbulkan pertanyaan tentang kebutuhan kompleksitas model dalam konteks spesifik ini.

Eksperimen ini menyoroti pentingnya pemilihan arsitektur model yang tepat sesuai dengan karakteristik dataset. Meskipun arsitektur CNN memberikan keunggulan dalam beberapa kasus, dalam kasus ini, keuntungannya relatif minimal. Dengan demikian, penyesuaian strategis dalam pemilihan model dapat mengoptimalkan kinerja pada tugas klasifikasi yang diberikan.

04. PyTorch Custom Datasets

Bab ini membahas langkah-langkah untuk membuat dataset kustom menggunakan PyTorch, dengan penekanan khusus pada implementasi kelas `ImageFolderCustom`. Dalam konteks ini, pembahasannya dimulai dengan menjelaskan opsi kedua dalam memuat data gambar saat pre-built dataset creator seperti `torchvision.datasets.ImageFolder()` tidak tersedia atau tidak sesuai dengan permasalahan khusus yang dihadapi. Untuk mengatasi ini, pembaca diajak memahami pro dan kontra dalam membuat dataset kustom.

Kelebihan pembuatan dataset kustom termasuk kemampuan untuk membuat dataset dari berbagai sumber data dan tidak terbatas pada fungsi-fungsi pre-built PyTorch. Namun, kekurangan utamanya adalah potensi penulisan kode yang lebih banyak, rentan terhadap kesalahan, dan masalah kinerja.

Langkah-langkah dalam membuat dataset kustom dimulai dengan pengenalan modul-modul yang dibutuhkan, seperti os, pathlib, torch, PIL, dan modul typing dari Python. Pembaca diperkenalkan pada pembuatan fungsi bantu untuk mendapatkan nama kelas dan mengatasi direktori target. Fungsi ini, `find_classes()`, akan memindai direktori target untuk mendapatkan nama kelas dan mengonversinya menjadi indeks numerik.

Selanjutnya, pembahasan berfokus pada pembuatan dataset kustom itu sendiri dengan mendefinisikan kelas `ImageFolderCustom`. Langkah-langkahnya termasuk inisialisasi dengan parameter direktori target dan transformasi opsional, pembuatan atribut-atribut seperti path gambar, transformasi, kelas, dan class_to_idx. Kelas ini juga mencakup metode untuk memuat gambar dari file, dan meng-overwrite metode `__len__` dan `__getitem__` untuk memberikan jumlah total sampel dalam dataset serta mendapatkan satu sampel data dari dataset.

Selanjutnya, pembaca diberikan contoh penerapan transformasi data lainnya, khususnya augmentasi data menggunakan `transforms.TrivialAugmentWide()`. Pembahasan dilanjutkan dengan cara mengonversi dataset kustom yang telah dibuat ke dalam DataLoader menggunakan `torch.utils.data.DataLoader()`. Pembaca juga diberikan contoh penggunaan fungsi bantu untuk menampilkan gambar secara acak dari dataset.

Bab ini secara umum mencakup langkah-langkah kunci dalam pembuatan dataset kustom, implementasi kelas, transformasi data, dan pemanfaatan DataLoader untuk melatih dan menguji model menggunakan PyTorch.

05. PyTorch Going Modular

Bab ini membahas perbedaan antara cell mode dan script mode dalam penggunaan notebook PyTorch. Cell mode adalah notebook yang berjalan secara normal, di mana setiap sel di notebook dapat berupa kode atau markdown. Sementara itu, script mode adalah notebook di mana banyak sel kode dapat diubah menjadi skrip Python.

Pertama, bab ini membahas cara mendapatkan data, yang melibatkan pengunduhan file zip dari GitHub menggunakan modul requests dan mengekstraknya ke dalam folder yang sesuai.

Selanjutnya, bab ini memperkenalkan pembuatan Dataset dan DataLoader dalam file data_setup.py. Fungsi create_dataloaders() dibuat untuk mengonversi data ke dalam format yang dapat digunakan oleh PyTorch, dan disimpan dalam file data_setup.py.

Bab ini juga membahas pembuatan model menggunakan file model_builder.py, di mana arsitektur TinyVGG dibuat sebagai kelas TinyVGG. Model ini dapat diakses dengan mengimpornya dari model_builder.py.

Selanjutnya, bab ini mencakup pembuatan fungsi train_step(), test_step(), dan train() dalam file engine.py. Fungsi-fungsi ini digunakan untuk melatih dan menguji model, serta menyimpan hasil pelatihan seperti loss dan akurasi.

Terakhir, bab ini membahas pembuatan fungsi save_model() dalam file utils.py, yang berfungsi untuk menyimpan model ke direktori target.

Semua langkah-langkah ini kemudian digabungkan dalam file train.py, yang berfungsi sebagai skrip utama untuk melatih model dengan menggunakan semua komponen yang telah dibuat sebelumnya. Skrip ini dapat dijalankan dari baris perintah untuk melatih model dengan hyperparameter yang telah ditentukan.

Dengan demikian, bab ini membawa pembaca melalui proses membangun model PyTorch secara modular dengan menempatkan setiap komponen ke dalam file terpisah untuk meningkatkan keterbacaan dan keterpisahan fungsionalitas.

06. PyTorch Transfer Learning

Untuk melakukan transfer learning menggunakan PyTorch dengan library torchvision, langkah-langkah awal melibatkan impor modul dan paket yang dibutuhkan, serta persiapan perangkat dan pengecekan versi PyTorch. Selanjutnya, data gambar pizza, steak, dan sushi diunduh dari GitHub dan disiapkan dalam DataLoader menggunakan skrip data_setup.py. Transformasi manual dan otomatis diterapkan untuk memastikan konsistensi format data input dengan model yang telah dilatih sebelumnya, seperti EfficientNet_B0.

Model yang sudah dilatih diunduh dari torchvision.models dan lapisan dasar (fitur) dibekukan agar tidak berubah selama pelatihan ulang. Lapisan output disesuaikan dengan jumlah kelas yang dibutuhkan, dan fungsi kehilangan serta optimizer ditentukan untuk pelatihan model. Proses pelatihan dilakukan dengan menggunakan fungsi train untuk beberapa epoch, dan kinerja model dievaluasi dengan memeriksa akurasi pada set pelatihan dan pengujian.

Selanjutnya, kurva kehilangan pelatihan dan pengujian divisualisasikan untuk memantau peningkatan model seiring waktu. Terakhir, fungsi untuk membuat prediksi pada gambar dari set pengujian diimplementasikan, dan model digunakan untuk mengklasifikasikan gambar

pizza, steak, dan sushi. Dengan langkah-langkah ini, transfer learning dapat diaplikasikan secara efektif untuk meningkatkan kinerja model dalam tugas klasifikasi yang spesifik.

07. PyTorch Experiment Tracking

Pada bagian ini yang berjudul "07. Pemantauan Eksperimen PyTorch," fokus penelitian tertuju pada pengaturan pemantauan eksperimen untuk model PyTorch. Tahap awal dimulai dengan mengunduh modul-modul yang diperlukan, termasuk skrip dari direktori "going_modular" yang telah dibuat pada bagian sebelumnya. Selain itu, paket torchinfo diperoleh untuk ringkasan visual model. Pentingnya penggunaan versi terbaru paket torchvision ditekankan.

Beralih ke langkah berikutnya, kami bertujuan untuk meningkatkan hasil pada dataset FoodVision Mini, yang sebelumnya digunakan untuk klasifikasi gambar pizza, steak, dan sushi. Proses ini melibatkan eksekusi eksperimen untuk meningkatkan kinerja model. Dataset, pizza_steak_sushi.zip, diunduh menggunakan kode yang difungsikan dari bagian sebelumnya, memungkinkan untuk dapat digunakan kembali.

Selanjutnya, data diubah menjadi PyTorch DataLoaders melalui fungsi create_dataloaders(). Untuk mempersiapkan gambar dengan benar untuk transfer learning, transformasi menggunakan transformasi yang dibuat secara manual dan transformasi yang dibuat secara otomatis dari torchvision.models diperlihatkan. Normalisasi gambar dalam format ImageNet menjadi hal yang penting, sejalan dengan harapan model yang telah dipretraining.

Beranjak ke depan, fokus beralih untuk mendapatkan model yang telah dipretraining, membekukan lapisan dasarnya, dan menyesuaikan kepala pengklasifikasian agar sesuai dengan jumlah kelas (pizza, steak, sushi). Model EfficientNet_B0 yang telah dipretraining diunduh dan dikonfigurasi untuk ekstraksi fitur.

Dengan model yang telah dipersiapkan, langkah selanjutnya melibatkan mendefinisikan fungsi kerugian (CrossEntropyLoss) dan pengoptimal (Adam) untuk pelatihan. Fungsi train() disesuaikan untuk menggabungkan SummaryWriter() untuk pemantauan eksperimen, memungkinkan pencatatan kerugian pelatihan dan uji, serta nilai akurasi.

Kelas SummaryWriter() diperkenalkan sebagai alat untuk menyimpan berbagai aspek kemajuan pelatihan model dalam format TensorBoard. Contoh instansi SummaryWriter() default dibuat, dan fungsi train() dimodifikasi untuk menggunakannya, melacak nilai kerugian dan akurasi.

Untuk melihat hasil model dalam TensorBoard, petunjuk khusus disediakan untuk berbagai lingkungan kode, seperti VS Code dan Notebook Jupyter/Colab.

Selain itu, fungsi pembantu bernama create_writer() diusulkan untuk menghasilkan instansi SummaryWriter() kustom untuk setiap eksperimen, memungkinkan pelacakan detail eksperimen, nama model, dan informasi tambahan dalam direktori log terpisah.

Secara ringkas, bagian ini membimbing pengguna melalui proses pengaturan pemantauan eksperimen di PyTorch, mulai dari mengunduh modul hingga membuat direktori log eksperimen kustom menggunakan SummaryWriter().

08. PyTorch Paper Replicating

Dalam tahap ini, fokusnya adalah pada penyusunan data untuk mereplikasi model Vision Transformer (ViT) dengan berbasis pada FoodVision Mini. Langkah pertama adalah mengunduh dataset gambar pizza, steak, dan sushi menggunakan fungsi download_data() dari helper_functions.py, dengan proses selanjutnya dilakukan melalui fungsi create_dataloaders() dalam data_setup.py. Dalam pengolahan data, dilakukan transformasi untuk menyesuaikan ukuran gambar dengan resolusi pelatihan 224x224 piksel, sesuai dengan referensi awal dari paper ViT.

Setelah mendapatkan data yang diperlukan, langkah selanjutnya adalah mengonversinya menjadi DataLoader. Proses ini melibatkan pembuatan transformasi untuk mempersiapkan gambar, dengan memperhatikan resolusi pelatihan dan ukuran batch yang dicontohkan dalam Table 3 pada paper ViT. Di sini, kita tidak menerapkan transformasi normalisasi karena model akan dilatih dari awal tanpa transfer learning.

Sebelum kita masuk ke implementasi lebih lanjut, penting untuk memahami tujuan utama, yaitu mereplikasi paper ViT untuk menyelesaikan permasalahan FoodVision Mini. Model yang diinginkan harus dapat menerima input berupa gambar pizza, steak, dan sushi, dan menghasilkan output berupa label yang diprediksi untuk setiap kategori. Prinsip dasar jaringan saraf, seperti lapisan (layer) dan blok, menjadi dasar dalam mereplikasi paper ViT ini, dan prosesnya akan dibagi menjadi langkah-langkah yang lebih terperinci.

Gambar 1 pada paper ViT memberikan gambaran umum tentang arsitektur yang terdiri dari dua jalur utama: pembentukan token dan blok transformer. Pembentukan token mengubah gambar input menjadi urutan patch dan menambahkan nomor posisi untuk menentukan urutan patch tersebut. Pada bagian ini, penting untuk memahami konsep lapisan dan blok dalam arsitektur jaringan saraf. Lapisan menerima input, melakukan fungsi tertentu, dan menghasilkan output, sedangkan blok adalah kumpulan lapisan yang melakukan serangkaian fungsi pada input dan menghasilkan output. Prinsip ini menjadi panduan dalam mereplikasi paper ViT secara bertahap, mulai dari lapisan dan blok hingga arsitektur keseluruhan.

Proses berikutnya adalah memahami komponen-komponen spesifik dalam arsitektur ViT. Detail-detail ini tersebar dalam paper dan menjadi penanda dalam mereplikasi. Gambar 1, empat persamaan pada bagian 3.1, dan Tabel 1 menjadi sumber utama untuk merancang arsitektur. Sebagai contoh, Figure 1 memberikan gambaran tentang komponen-komponen utama, seperti patch embedding, linear projection, layer normalization, multi-head attention, MLP (multilayer perceptron), transformer encoder, dan MLP head. Informasi ini menjadi dasar dalam mengimplementasikan setiap komponen menggunakan fungsi-fungsi PyTorch yang tersedia.

Langkah-langkah ini membimbing kita untuk mereplikasi model ViT secara bertahap, dari konsep dasar hingga implementasi komponen-komponennya menggunakan PyTorch. Proses ini didasarkan pada pemahaman terperinci terhadap paper ViT dan pengetahuan tentang prinsip dasar jaringan saraf. Dengan mendekomposisi paper menjadi potongan-potongan yang lebih kecil, proses replikasi menjadi lebih dapat dielami dan dipahami, menjadikan paper yang kompleks menjadi lebih terjangkau.

09. Pytorch model deployment

Proyek FoodVision Mini diperkenalkan sebagai upaya dalam pengenalan gambar makanan menggunakan model EfficientNetB2. Langkah-langkah persiapan, termasuk impor library, pengolahan data, dan konfigurasi model, mendirikan fondasi yang kuat bagi pemahaman proyek ini.

Fokus berikutnya adalah pada pelatihan model, di mana inisialisasi EfficientNetB2 hingga evaluasi kinerja dengan metrik akurasi dan loss diuraikan secara komprehensif. Penjelasan yang cermat memberikan wawasan mendalam tentang seluruh proses latihan yang dilakukan.

Pada bagian ketiga, proyek ini mengintegrasikan Gradio sebagai solusi deploy, menjembatani kesenjangan antara pengembangan model dan pengalaman pengguna. Pendekatan ini memungkinkan pembaca untuk melihat cara membuat antarmuka web interaktif untuk model, membuka pintu bagi eksplorasi lebih lanjut dan penggunaan praktis model.

Ringkasan menyeluruh menggambarkan kombinasi keahlian dalam pemrosesan data, pembelajaran mesin, dan antarmuka pengguna yang menyelaraskan langkah-langkah persiapan, pelatihan model, dan implementasi Gradio. FoodVision Mini menjadi solusi holistik dalam pengenalan gambar makanan.