Raken Putra Athallah
1103204186

Pytorch Fundamentals

1.Dasar-dasar Pytorch

1.1 Pytorch

PyTorch adalah framework machine learning open-source yang dikembangkan oleh Facebook. Berbeda dengan beberapa framework lain, PyTorch didesain dengan fokus pada dinamika komputasi, memungkinkan peneliti dan praktisi untuk membuat dan melatih model machine learning dengan lebih fleksibel.

Fitur Utama PyTorch:

- Tensor Operations: PyTorch menggunakan konsep tensor sebagai struktur data utama. Tensor adalah array multidimensi yang menyediakan fondasi untuk merepresentasikan data dalam bentuk numerik, yang sangat penting dalam machine learning.
- Dynamic Computational Graph: Model dalam PyTorch menggunakan grafik komputasi dinamis. Ini berarti grafik komputasi dapat berubah selama runtime, memungkinkan fleksibilitas dalam membuat dan mengubah model secara interaktif.
- Eager Execution: PyTorch mengadopsi eager execution, yang memungkinkan pengguna untuk mengevaluasi ekspresi secara langsung, membuat debugging dan eksplorasi data lebih mudah.

1.2 Tensor pada Pytorch

Tensors adalah komponen dasar PyTorch, dan mereka memainkan peran kunci dalam merepresentasikan data. Tensors dapat memiliki berbagai dimensi, mirip dengan array multidimensi, dan mampu menyimpan berbagai jenis data.

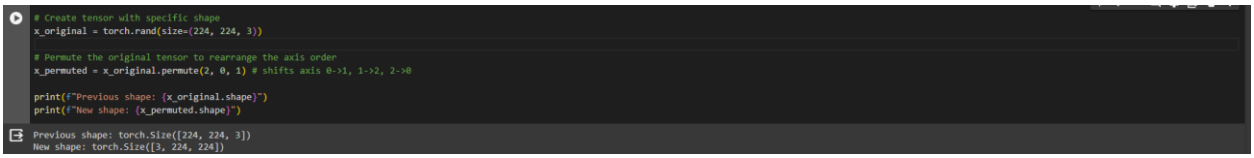
Operasi Umum pada Tensors:

- Pembuatan Tensors: Menggunakan torch.tensor() untuk membuat tensor dari data yang ada.
- Akses dan Manipulasi: Tensors dapat diakses dan dimanipulasi menggunakan indeks dan operasi seperti reshape, squeeze, dan transpose.
- Operasi Matematika: PyTorch menyediakan berbagai operasi matematika untuk tensors, termasuk perkalian, penambahan, dan fungsi aktivasi.

2. Manipulasi Tensors dengan PyTorch

Dalam penggunaan PyTorch, manipulasi tensors merupakan langkah kritis untuk memahami dan memproses data. Berikut adalah beberapa poin utama dalam pembahasan ini:

- **Pembuatan Tensors**: PyTorch menyediakan beberapa cara untuk membuat tensors. Selain torch.tensor(), kita dapat menggunakan torch.zeros(), torch.ones(), dan torch.rand() untuk menciptakan tensors dengan nilai awal yang sesuai.

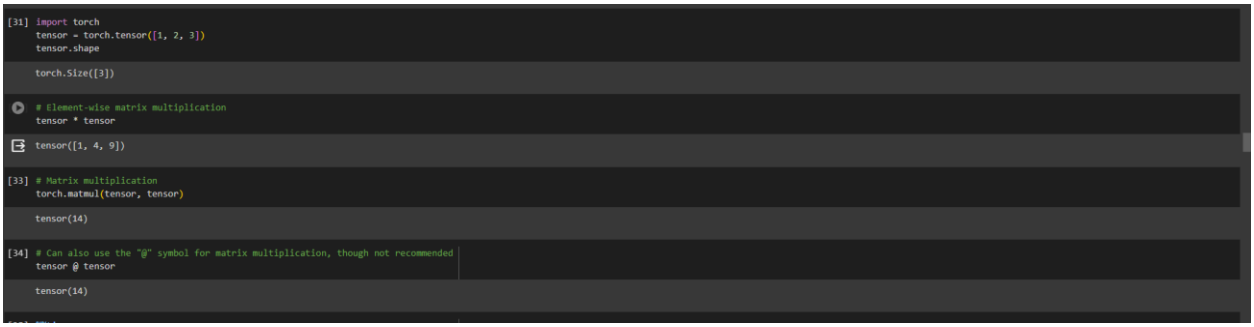


- **Akses dan Manipulasi**: Tensors dapat diakses menggunakan indeks seperti list atau array pada Python. Operasi seperti penambahan, pengurangan, dan perkalian dapat diaplikasikan dengan mudah.

3. Tensor Operation

- **Perkalian Matriks**

  Dalam konteks deep learning, perkalian matriks diperlukan saat menghubungkan neuron antar layer dalam suatu model. Operasi ini menciptakan hubungan antara setiap neuron, memungkinkan model untuk mengekstraksi fitur yang kompleks dari data.

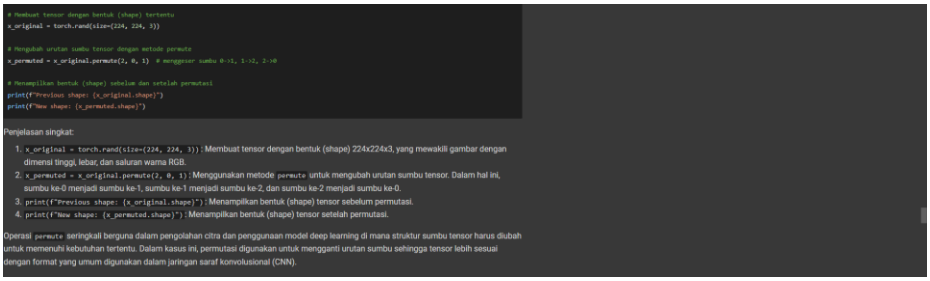


- **Transpose dan Permute**

  Operasi transpose dan permute pada tensors memberikan fleksibilitas dalam mengubah urutan dimensi, yang berguna untuk memenuhi kebutuhan model dan manipulasi data

  Menggunakan torch.transpose() untuk menukar dimensi pada suatu tensor.

  Penerapan torch.permute() untuk melakukan permutasi dimensi tensors.



```
# Membuat tensor dengan bentuk (shape) tertentu
x_original = torch.rand(size=(224, 224, 3))

# Mengubah urutan sumbu tensor dengan metode permute
x_permuted = x_original.permute(2, 0, 1)  # mengganti sumbu 0->1, 1->2, 2->0

# Menampilkan bentuk (shape) sebelum dan setelah permutasi
print(f"Previous shape: {x_original.shape}")
print(f"New shape: {x_permuted.shape}")
```

Penjelasan singkat:

1. `x_original = torch.rand(size=(224, 224, 3))` : Membuat tensor dengan bentuk (shape) 224x224x3, yang mewakili gambar dengan dimensi tinggi, lebar, dan saluran warna RGB.
2. `x_permuted = x_original.permute(2, 0, 1)` : Menggunakan metode `permute` untuk mengubah urutan sumbu tensor. Dalam hal ini, sumbu ke-0 menjadi sumbu ke-1, sumbu ke-1 menjadi sumbu ke-2, dan sumbu ke-2 menjadi sumbu ke-0.
3. `print(f"Previous shape: {x_original.shape}")` : Menampilkan bentuk (shape) tensor sebelum permutasi.
4. `print(f"New shape: {x_permuted.shape}")` : Menampilkan bentuk (shape) tensor setelah permutasi.

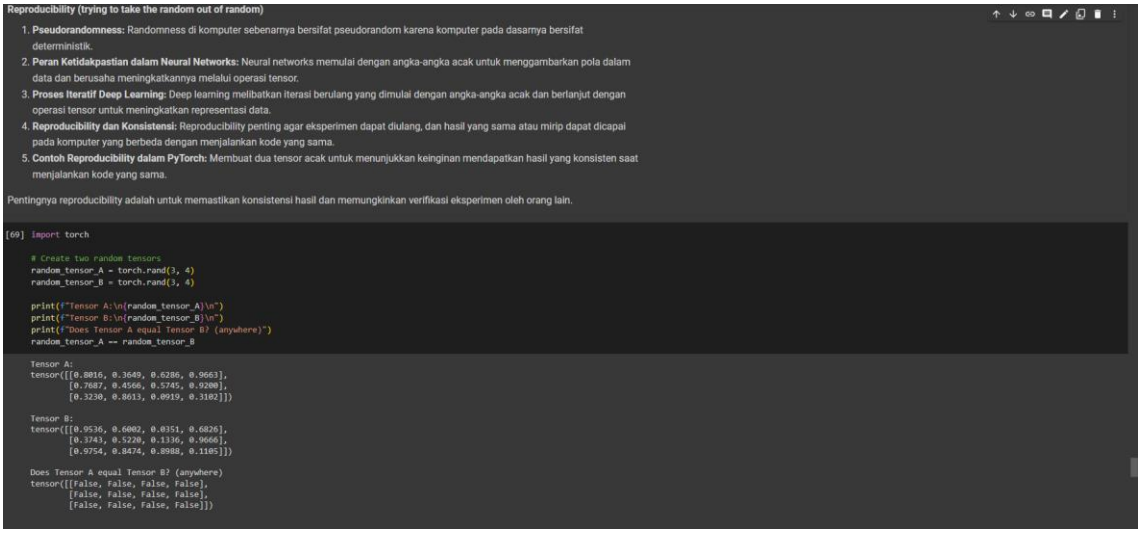
Operasi `permute` seringkali berguna dalam pengolahan citra dan penggunaan model deep learning di mana struktur sumbu tensor harus diubah untuk memenuhi kebutuhan tertentu. Dalam kasus ini, permutasi digunakan untuk mengganti urutan sumbu sehingga tensor lebih sesuai dengan format yang umum digunakan dalam jaringan saraf konvolusional (CNN).

4. Reproducibility dalam PyTorch

Reproducibility (kemampuan untuk mendapatkan hasil yang sama) adalah aspek penting dalam eksperimen dan pengembangan model deep learning. Dalam PyTorch, aspek-aspek berikut perlu diperhatikan untuk mencapai reproduktibilitas:

- Pseudorandomness dan Reproducibility: Dalam konteks komputasi, randomness yang dihasilkan oleh komputer sebenarnya bersifat pseudorandom. Maka dari itu, untuk eksperimen yang dapat diulang, perlu diperhatikan agar randomness yang dihasilkan bersifat dapat direproduksi.
- Contoh Reproducibility: Demonstrasi sederhana tentang bagaimana menjaga konsistensi hasil eksperimen. Penggunaan fungsi torch.manual_seed() memastikan inisialisasi yang sama dari generator angka acak, sehingga dapat dihasilkan hasil yang konsisten pada setiap eksekusi.

Reproducibility sangat penting untuk memverifikasi eksperimen dan memastikan bahwa hasilnya dapat diandalkan dan dapat diulang oleh pihak lain. Dengan penggunaan seed dan kontrol terhadap randomness, hasil yang sama dapat dicapai pada setiap eksekusi.



Reproducibility (trying to take the random out of random)

1. **Pseudorandomness:** Randomness di komputer sebenarnya bersifat pseudorandom karena komputer pada dasarnya bersifat deterministik.
2. **Peran Ketidakpastian dalam Neural Networks:** Neural networks memulai dengan angka-angka acak untuk menggambarkan pola dalam data dan berusaha meningkatkannya melalui operasi tensor.
3. **Proses Iteratif Deep Learning:** Deep learning melibatkan iterasi berulang yang dimulai dengan angka-angka acak dan berlanjut dengan operasi tensor untuk meningkatkan representasi data.
4. **Reproducibility dan Konsistensi:** Reproducibility penting agar eksperimen dapat diulang, dan hasil yang sama atau mirip dapat dicapai pada komputer yang berbeda dengan menjalankan kode yang sama.
5. **Contoh Reproducibility dalam PyTorch:** Membuat dua tensor acak untuk menunjukkan keinginan mendapatkan hasil yang konsisten saat menjalankan kode yang sama.

Pentingnya reproducibility adalah untuk memastikan konsistensi hasil dan memungkinkan verifikasi eksperimen oleh orang lain.

```
[69] import torch

    # Create two random tensors
    random_tensor_A = torch.rand(3, 4)
    random_tensor_B = torch.rand(3, 4)

    print(f"Tensor A:\n{random_tensor_A}\n")
    print(f"Tensor B:\n{random_tensor_B}\n")
    print(f"Does Tensor A equal Tensor B? (anywhere)")
    random_tensor_A == random_tensor_B

    Tensor A:
    tensor([[0.8016, 0.3649, 0.6286, 0.9663],
            [0.7687, 0.4566, 0.5745, 0.9200],
            [0.3230, 0.8613, 0.0919, 0.3102]])

    Tensor B:
    tensor([[0.9536, 0.6002, 0.0351, 0.6826],
            [0.3743, 0.5220, 0.1336, 0.9666],
            [0.9754, 0.8474, 0.8988, 0.1105]])

    Does Tensor A equal Tensor B? (anywhere)
    tensor([[False, False, False, False],
            [False, False, False, False],
            [False, False, False, False]])
```