NAMA : DERY HIDAYAT
NIM : 1103228181

Zero to Mastery Learn PyTorch for Deep Learning

00.pytorch_fundamentals

- Apa itu PyTorch?

PyTorch adalah sebuah perpustakaan atau framework open-source untuk komputasi numerik yang dirancang untuk memudahkan pengembangan dan penerapan model deep learning. Dirilis oleh Facebook, PyTorch menawarkan alat yang kuat dan fleksibel untuk membangun jaringan neural, melakukan komputasi tensor, serta melakukan proses pelatihan dan evaluasi model deep learning. Kelebihan PyTorch terletak pada pendekatan yang intuitif dalam pembuatan model, memungkinkan pengguna untuk dengan mudah mengembangkan prototipe model, melihat alur dari setiap operasi, dan mengelola model dengan lebih efisien. Dengan dukungan yang kuat dari komunitas serta integrasi yang baik dengan Python, PyTorch menjadi pilihan utama bagi para peneliti dan praktisi dalam dunia machine learning dan deep learning.

- PyTorch bisa digunakan untuk apa?

PyTorch digunakan untuk mengembangkan, melatih, dan mengevaluasi model neural network dalam bidang machine learning dan deep learning. Ini memfasilitasi pembuatan berbagai jenis model neural, pelatihan model dengan data yang ada, serta evaluasi performa model dalam tugas-tugas seperti klasifikasi, regresi, deteksi objek, pemrosesan bahasa alami, dan berbagai aplikasi yang memanfaatkan kecerdasan buatan. Dengan kemampuan komputasi tensor yang efisien, PyTorch memungkinkan implementasi dan eksperimen dengan model-machine learning yang kompleks, menjadikannya alat yang serbaguna dan kuat bagi para peneliti, praktisi, dan pengembang di berbagai industri dan bidang aplikasi.

- Siapa yang menggunakan PyTorch?

PyTorch digunakan oleh berbagai kalangan, termasuk para peneliti di bidang kecerdasan buatan, ilmuwan data, praktisi machine learning, dan pengembang perangkat lunak. Banyak perusahaan teknologi besar, institusi penelitian, dan startup mengadopsi PyTorch dalam penelitian, pengembangan produk, serta dalam membangun dan melatih model-machine learning yang kompleks. Selain itu, para mahasiswa, pengajar, dan komunitas akademis juga menggunakan PyTorch karena kesederhanaan, fleksibilitas, serta dukungan yang luas dari komunitas dan industri, menjadikannya salah satu pilihan utama dalam ekosistem machine learning saat ini.

- Mengapa menggunakan PyTorch?

PyTorch memiliki beberapa keunggulan yang membuatnya menjadi pilihan utama dalam pengembangan model-machine learning. Antara lain, PyTorch menawarkan pendekatan yang intuitif dalam pembuatan model, memudahkan pengguna untuk membuat prototipe, menguji gagasan baru, dan memahami alur dari setiap operasi.

NAMA : DERY HIDAYAT
NIM : 1103228181

- Mengimpor PyTorch

Untuk menggunakan PyTorch, kita dapat mengimpor library tersebut menggunakan pernyataan import torch. Ini mengizinkan akses ke berbagai fungsi dan kelas yang disediakan oleh PyTorch. Setelah diimpor, kita dapat membangun, melatih, dan mengevaluasi model-machine learning, melakukan komputasi tensor, serta menjalankan berbagai operasi terkait kecerdasan buatan dan pembelajaran mesin yang didukung oleh PyTorch. Importing PyTorch adalah langkah awal untuk memulai penggunaan dan eksplorasi kemampuan yang ditawarkan oleh framework ini dalam pengembangan model-machine learning.

- Pengantar tensor

Tensor adalah struktur data fundamental dalam PyTorch yang menyerupai array multidimensi dengan elemen-elemen numerik. Mirip dengan array NumPy, tensor dalam PyTorch dapat berisi bilangan bulat, bilangan riil, maupun nilai-nilai lainnya. Mereka dapat memiliki dimensi yang bervariasi, dari vektor (1 dimensi) hingga tensor multidimensi yang lebih kompleks. Tensor digunakan untuk merepresentasikan data dan mengelola aliran informasi dalam operasi-operasi matematika serta dalam pembuatan dan pelatihan model-machine learning.

- Membuat tensor
  Tensor dalam PyTorch adalah struktur data fundamental yang menyerupai array multidimensi. Dibandingkan dengan array NumPy, tensor ini mendukung perhitungan yang dioptimalkan untuk komputasi numerik, seperti pembuatan model-machine learning dan operasi matematika lainnya. Anda dapat membuat tensor dengan berbagai nilai seperti nol, satu, acak, atau dari data yang sudah ada seperti list Python atau array NumPy. Tensor ini merupakan komponen utama dalam proses pembelajaran mesin dan pengolahan data numerik dalam PyTorch.
- Tensor acak dalam PyTorch adalah tensor yang berisi nilai-nilai acak. PyTorch menyediakan fungsi untuk membuat tensor dengan nilai-nilai acak, baik dari distribusi seragam (antara 0 dan 1) maupun dari distribusi normal (dengan rata-rata 0 dan standar deviasi 1). Fungsi-fungsi ini memungkinkan pembuatan tensor dengan berbagai ukuran yang diisi dengan nilai-nilai acak, yang sering digunakan dalam inisialisasi bobot dalam model-machine learning, atau dalam proses-proses yang memerlukan data acak untuk pengujian atau eksperimen.
- Nol dan satu
  Tensor yang diisi dengan nilai nol (torch.zeros) dan satu (torch.ones) adalah struktur data khusus dalam PyTorch yang berguna dalam inisialisasi model-machine learning, manipulasi data, dan operasi-operasi matematika. Tensor dengan nilai nol memiliki semua elemen yang diisi dengan nol, sedangkan tensor dengan nilai satu memiliki semua elemen yang diisi dengan satu, memungkinkan penggunaan dalam berbagai konteks seperti pengaturan awal atau operasi-operasi yang memerlukan nilai-nilai ini sebagai referensi dalam perhitungan dan manipulasi data.
- Membuat rentang dan tensor sejenisnya
  Untuk membuat rentang nilai atau tensor serupa dengan rentang tersebut, PyTorch menyediakan beberapa fungsi seperti torch.arange() yang menghasilkan rentang nilai

berdasarkan parameter awal, akhir, dan langkah. Fungsi ini memungkinkan pembuatan tensor dengan nilai-nilai yang membentuk rentang yang disesuaikan. Selain itu, torch.linspace() menghasilkan tensor dengan nilai-nilai yang tersebar merata dalam rentang tertentu, sementara torch.logspace() menghasilkan tensor dengan nilai-nilai yang meningkat secara eksponensial berdasarkan logaritma. Fungsi-fungsi ini memungkinkan pembuatan tensor dengan rentang nilai tertentu dan memberikan fleksibilitas dalam pengaturan rentang nilai yang diinginkan untuk aplikasi-aplikasi yang berbeda dalam machine learning atau pemrosesan data numerik.

- Tipe data tensor
  Tensor dalam PyTorch memiliki beragam tipe data yang mempengaruhi representasi nilai-nilai di dalamnya. Tipe data default untuk tensor adalah torch.float32, namun PyTorch juga mendukung tipe data lain seperti torch.int, torch.float64, torch.bool, dan lainnya. Setiap tipe data memiliki kegunaan spesifiknya; misalnya, torch.float32 sering digunakan dalam operasi-operasi numerik standar, sementara torch.bool berguna dalam logika biner. Kemampuan untuk mengubah tipe data tensor memungkinkan pengguna untuk mengoptimalkan penggunaan memori, menyesuaikan dengan kebutuhan komputasi, dan memanipulasi data secara efisien dalam berbagai konteks pengolahan data dan machine learning.

- Mendapatkan informasi dari tensor

Untuk mendapatkan informasi penting dari tensor, PyTorch menyediakan beberapa atribut yang memberikan wawasan tentang karakteristik tensor. Atribut shape memberikan dimensi tensor, sedangkan dtype memberikan tipe data nilai-nilai di dalamnya. Selain itu, atribut size() memberikan ukuran total tensor, sementara ndim memberikan jumlah dimensi tensor. Melalui atribut-atribut ini, pengguna dapat memahami struktur tensor, tipe data yang digunakan, serta dimensi dan ukuran tensor yang digunakan, memberikan informasi krusial untuk pemrosesan dan manipulasi data, serta untuk pembentukan model-machine learning.

- Memanipulasi tensor (operasi tensor)

Operasi tensor dalam PyTorch mengizinkan pengguna untuk melakukan berbagai manipulasi data dan perhitungan matematika pada tensor. PyTorch menawarkan beragam operasi yang mencakup penjumlahan, pengurangan, perkalian, dan operasi matematika lainnya. Operasi broadcasting memungkinkan perhitungan antara tensor dengan bentuk yang berbeda secara otomatis, sementara fungsi seperti torch.transpose() dan torch.reshape() memungkinkan pengubahan bentuk dan pengurutan data dalam tensor. Fungsi-fungsi ini memfasilitasi transformasi data, manipulasi struktur tensor, dan penggunaan operasi matematika yang penting dalam pengembangan model-machine learning dan pemrosesan data numerik.

- Operasi dasar
  Operasi dasar pada tensor dalam PyTorch meliputi penjumlahan, pengurangan, perkalian, dan pembagian antara tensor. Penjumlahan dan pengurangan dapat dilakukan baik antara tensor yang memiliki dimensi yang sama maupun dalam

NAMA : DERY HIDAYAT
NIM : 1103228181

operasi broadcasting yang secara otomatis menyesuaikan bentuk tensor. Perkalian tensor dapat dilakukan baik dengan menggunakan perkalian matriks ataupun perkalian elemen-wise. Operasi-operasi dasar ini memungkinkan manipulasi dan transformasi data serta perhitungan matematika yang esensial dalam pengembangan model-machine learning dan pemrosesan data numerik dalam lingkungan PyTorch.

- Perkalian matriks (yang Anda perlukan)
  Perkalian matriks dalam PyTorch dapat dilakukan menggunakan fungsi torch.matmul() atau operator @. Operasi ini mengalikan dua tensor dengan aturan matematika perkalian matriks, yaitu antara tensor yang memiliki dimensi yang sesuai. Misalnya, untuk melakukan perkalian matriks antara matriks A (ukuran mxn) dan matriks B (ukuran nxp), hasilnya akan menjadi matriks baru dengan ukuran mxp. Perkalian matriks sangat penting dalam pembangunan model-machine learning, terutama dalam proses feedforward dan perhitungan bobot serta dalam operasi-operasi matematika yang melibatkan transformasi data dan perhitungan kompleks.

- Salah satu kesalahan paling umum dalam pembelajaran mendalam (kesalahan bentuk)

Salah satu kesalahan umum dalam pembelajaran mendalam terkait dengan kesalahan bentuk (shape) tensor. Kesalahan ini sering terjadi saat dimensi atau bentuk tensor tidak sesuai dengan ekspektasi dalam operasi-operasi tertentu, seperti pada saat melakukan perkalian matriks atau saat mentransformasi data. Hal ini bisa terjadi karena pengabaian terhadap operasi reshape atau transpose yang diperlukan, atau karena pemahaman yang kurang mengenai dimensi tensor yang dibutuhkan oleh suatu fungsi atau operasi tertentu. Kesalahan bentuk tensor dapat mengakibatkan error dalam perhitungan matematika atau dalam pelatihan model-machine learning, oleh karena itu memahami dan memeriksa secara cermat bentuk tensor adalah hal yang sangat penting dalam pengembangan aplikasi machine learning.

- Menemukan min, maks, mean, jumlah, dll (agregasi)
  Operasi agregasi seperti menemukan nilai minimum, maksimum, rata-rata, atau jumlah dari tensor dalam PyTorch dapat dilakukan dengan menggunakan fungsi-fungsi bawaan seperti torch.min(), torch.max(), torch.mean(), dan torch.sum(). Fungsi-fungsi ini memungkinkan pengguna untuk secara efisien melakukan operasi agregasi pada tensor, baik untuk satu dimensi atau lebih, dengan memberikan nilai-nilai agregat dari tensor, seperti nilai minimum atau maksimum di seluruh tensor, rata-rata nilai dalam tensor, atau total jumlah nilai dalam tensor. Operasi-operasi agregasi ini penting dalam analisis data, pemrosesan statistik, serta evaluasi kinerja model-machine learning.
- Posisi min/maks
  Untuk menemukan posisi (indeks) dari nilai minimum atau maksimum dalam tensor, PyTorch menyediakan fungsi seperti torch.argmin() dan torch.argmax(). Fungsi-fungsi ini mengembalikan indeks dari nilai terkecil atau terbesar dalam tensor sesuai dengan dimensi yang diinginkan. Misalnya, dengan menggunakan torch.argmin() pada tensor 1D, akan mengembalikan indeks dari nilai minimum, sedangkan dengan tensor multidimensi, parameter dimensi dapat diberikan untuk menentukan dimensi

mana yang ingin dicari nilai minimum atau maksimumnya. Kemampuan untuk menemukan posisi dari nilai ekstrem dalam tensor memungkinkan pengguna untuk melakukan analisis lebih lanjut terhadap data dan model-machine learning, seperti penentuan indeks yang sesuai dengan prediksi tertentu atau pemilihan elemen-elemen spesifik dalam tensor.

- Ubah tipe data tensor
  Untuk mengubah tipe data tensor dalam PyTorch, pengguna dapat menggunakan metode .to() pada tensor yang ingin diubah. Fungsi ini memungkinkan konversi tensor ke tipe data yang diinginkan, seperti mengubah tensor menjadi tipe data float, integer, boolean, dan lainnya. Contohnya, dengan menggunakan .to(torch.float64) pada suatu tensor, tipe data tensor tersebut dapat diubah menjadi torch.float64. Kemampuan untuk mengubah tipe data tensor memungkinkan fleksibilitas dalam pengolahan data numerik, pemilihan tipe data yang sesuai dengan kebutuhan perhitungan, serta penggunaan memori yang efisien dalam proses pembelajaran mesin dan analisis data.
- Membentuk kembali, menumpuk, meremas dan melepaskan
  Dalam PyTorch, operasi seperti reshape (view()), stacking (torch.stack()), squeezing (squeeze()), dan unsqueezing (unsqueeze()) memungkinkan manipulasi bentuk dan dimensi tensor. Reshape digunakan untuk mengubah dimensi tensor sesuai dengan format yang diinginkan, sementara stacking memungkinkan penggabungan tensor dalam dimensi tambahan. Squeezing menghilangkan dimensi dengan ukuran 1, sedangkan unsqueezing menambah dimensi dengan ukuran 1. Operasi-operasi ini berguna dalam mengubah struktur tensor untuk kompatibilitas operasi atau model, seperti menyesuaikan input model atau mempersiapkan data untuk operasi tertentu dalam machine learning.

- Pengindeksan (memilih data dari tensor)

Pengindeksan pada tensor di PyTorch memungkinkan pengguna untuk memilih data spesifik dari tensor berdasarkan indeks atau kriteria tertentu. Pengindeksan dapat dilakukan dengan menggunakan notasi slicing seperti tensor[start:stop] untuk memilih rentang data atau dengan memberikan indeks tertentu tensor[index] untuk mendapatkan nilai di posisi indeks tersebut. Selain itu, pengindeksan boolean memungkinkan pemilihan data berdasarkan kondisi logika, di mana hanya elemen-elemen yang memenuhi kriteria yang akan dipilih. Operasi pengindeksan ini berguna dalam memanipulasi data, memperoleh subset data yang dibutuhkan, atau dalam proses pengambilan data tertentu yang relevan dalam konteks pengolahan data atau machine learning.

- Tensor PyTorch dan NumPy

Tensor dalam PyTorch dan NumPy memiliki konsep yang mirip, keduanya mewakili struktur data multidimensi dengan kemampuan untuk melakukan operasi matematika dan manipulasi data. Salah satu perbedaan utama adalah dalam representasi memori: tensor PyTorch menggunakan representasi yang dioptimalkan untuk perhitungan yang memperhatikan

NAMA : DERY HIDAYAT
NIM : 1103228181

penggunaan GPU untuk percepatan perhitungan, sementara array NumPy lebih umum digunakan dalam pemrosesan data numerik pada CPU. PyTorch menyediakan kemampuan yang baik dalam pembangunan dan pelatihan model-machine learning, sementara NumPy merupakan pustaka yang kuat dalam pemrosesan data, analisis, dan komputasi ilmiah. Keduanya dapat saling berinteraksi dengan konversi mudah antara tensor PyTorch dan array NumPy untuk memfasilitasi penggunaan yang lebih luas dan interaksi yang lancar di antara kedua ekosistem tersebut.

- Reproduksibilitas (mencoba mengambil yang acak dari yang acak)

Reproduksibilitas dalam konteks data acak merujuk pada kemampuan untuk mendapatkan hasil yang sama dalam eksperimen atau perhitungan yang melibatkan penggunaan bilangan acak. Dalam PyTorch, untuk memastikan reproduksibilitas dari hasil yang melibatkan operasi-operasi acak, pengguna dapat mengatur dan mengontrol seed (benih) acak dengan menggunakan torch.manual_seed() dan torch.cuda.manual_seed_all(). Dengan mengatur seed yang sama sebelum melakukan operasi-acak, pengguna dapat memastikan bahwa hasil yang dihasilkan akan konsisten di berbagai platform atau lingkungan perhitungan. Ini penting dalam pengujian model-machine learning, pengevaluasian algoritma, atau dalam kasus-kasus di mana hasil yang konsisten diperlukan pada setiap iterasi perhitungan.

- Menjalankan tensor pada GPU (dan membuat komputasi lebih cepat)

Menjalankan tensor pada unit pemrosesan grafis (GPU) dalam PyTorch dapat mempercepat komputasi, terutama untuk perhitungan yang intensif secara komputasional. Dengan menggunakan perintah seperti .to('cuda'), tensor dapat dipindahkan dari memori pusat pemrosesan (CPU) ke memori GPU. Ini memungkinkan operasi-operasi tensor, seperti perhitungan matematika atau pembelajaran mesin, untuk dieksekusi secara paralel di GPU, yang umumnya memiliki kapabilitas perhitungan yang lebih cepat daripada CPU. Hal ini terutama menguntungkan dalam pembelajaran mesin di mana operasi matriks besar atau jaringan saraf dapat dijalankan secara lebih efisien di GPU, meningkatkan kinerja dan mempercepat waktu pelatihan model.

  - Mendapatkan GPU
    Untuk mendapatkan akses ke GPU dalam lingkungan PyTorch, pengguna memerlukan perangkat keras yang dilengkapi dengan unit pemrosesan grafis (GPU) yang kompatibel, seperti kartu grafis NVIDIA. Setelah memastikan perangkat keras sesuai, pengguna dapat memeriksa ketersediaan GPU dengan menggunakan torch.cuda.is_available(). Jika tersedia, pengguna dapat menentukan perangkat GPU untuk operasi-operasi tensor dengan mengatur perangkat target menggunakan torch.device() dan .to() dalam PyTorch, memungkinkan eksekusi perhitungan tensor yang lebih cepat dan efisien di GPU.
  - Membuat PyTorch berjalan di GPU
    Untuk menjalankan PyTorch di GPU, pastikan Anda memiliki perangkat keras dengan GPU yang didukung dan telah menginstal driver NVIDIA yang sesuai. Setelah itu, Anda dapat memeriksa ketersediaan GPU dalam PyTorch dengan

NAMA : DERY HIDAYAT
NIM : 1103228181

menggunakan torch.cuda.is_available(). Jika GPU tersedia, Anda dapat menggunakan perangkat GPU yang tersedia dengan mengatur perangkat target menggunakan torch.device() dan .to() dalam PyTorch, memindahkan tensor ke GPU untuk eksekusi operasi yang lebih cepat. Dengan memanfaatkan GPU, Anda dapat meningkatkan kinerja perhitungan, terutama dalam kasus pengolahan data besar atau dalam pelatihan model-machine learning yang kompleks.

- Menempatkan tensor (dan model) pada GPU
  Untuk menempatkan tensor atau model pada unit pemrosesan grafis (GPU) dalam PyTorch, pastikan GPU tersedia dengan torch.cuda.is_available(). Kemudian, tentukan perangkat GPU sebagai perangkat target menggunakan torch.device() dan .to(), dengan menyertakan perangkat GPU yang tersedia sebagai argumen. Misalnya, untuk menempatkan tensor pada GPU, gunakan tensor.to('cuda'). Demikian pula, untuk memindahkan model ke GPU, gunakan metode .to('cuda') pada setiap parameter model. Dengan cara ini, operasi-operasi tensor atau pelatihan model akan dieksekusi pada GPU, mempercepat komputasi dan meningkatkan kinerja secara keseluruhan.
- Memindahkan tensor kembali ke CPU
  Untuk memindahkan tensor dari unit pemrosesan grafis (GPU) kembali ke memori pusat pemrosesan (CPU) dalam PyTorch, pengguna dapat menggunakan metode .cpu(). Misalnya, menggunakan .cpu() pada tensor yang berada di GPU (tensor_on_gpu.cpu()) akan mengembalikan tensor ke CPU. Hal ini berguna ketika ingin melakukan manipulasi atau analisis lebih lanjut pada data yang telah dihitung di GPU, atau ketika ingin menampilkan hasil pada CPU. Proses pemindahan ini juga membebaskan sumber daya GPU untuk operasi lain, membantu dalam manajemen memori yang efisien.

01.PyTorch Workflow Fundamentals

- Apa yang akan kita bahas

Kita akan membahas tentang konsep dasar seputar Tensor dalam PyTorch, termasuk pembentukan, manipulasi, operasi dasar, pemindahan tensor ke GPU, dan topik terkait lainnya. Selain itu, kita akan menjelajahi penggunaan GPU untuk akselerasi komputasi, reproduksibilitas dalam penggunaan bilangan acak, serta berbagai fungsi dan operasi yang umum digunakan dalam PyTorch. Semoga pembahasan ini membantu dalam memahami dasar-dasar PyTorch dan penerapannya dalam pengembangan model-machine learning serta pengolahan data.

- Di mana Anda bisa mendapatkan bantuan?

Anda bisa mendapatkan bantuan untuk PyTorch dari berbagai sumber. Situs web resmi PyTorch menyediakan dokumentasi resmi, tutorial, dan forum komunitas yang dapat diakses secara online. Selain itu, platform seperti Stack Overflow, GitHub, dan berbagai forum dan grup diskusi online juga merupakan sumber bantuan yang baik, di mana para pengguna PyTorch berbagi pengetahuan, pengalaman, dan memecahkan masalah. Jika Anda memiliki akses, tutorial daring,

NAMA : DERY HIDAYAT
NIM : 1103228181

pelatihan, atau kelas online juga dapat menjadi sumber bantuan yang berguna untuk memahami lebih dalam tentang PyTorch.

1. Data (persiapan dan pemuatan)

Persiapan dan pemuatan data dalam PyTorch melibatkan pembuatan kelas dataset yang disesuaikan dengan struktur data Anda, menggunakan metode __len__ untuk menentukan ukuran dataset, dan metode __getitem__ untuk memuat sampel dari dataset berdasarkan indeks. Setelah dataset Anda didefinisikan, Anda dapat menggunakan DataLoader untuk mengelompokkan, mengacak, dan memuat data secara paralel. Transformasi, seperti normalisasi atau augmentasi, dapat diterapkan menggunakan modul transforms saat pemuatan data. Keseluruhan proses ini memungkinkan untuk menyiapkan dan memuat data secara efisien saat melakukan proses pelatihan atau evaluasi model di PyTorch.

- Pisahkan data menjadi set pelatihan dan pengujian

Untuk memisahkan dataset menjadi set pelatihan dan pengujian dalam PyTorch, Anda dapat menggunakan torch.utils.data.random_split, yang memungkinkan pemisahan dataset ke dalam dua bagian dengan proporsi yang ditentukan. Dengan menentukan proporsi masing-masing bagian, seperti 80% untuk pelatihan dan 20% untuk pengujian, Anda dapat membagi dataset. Setelahnya, Anda bisa membuat DataLoader untuk masing-masing bagian, memungkinkan iterasi dan pemrosesan terpisah untuk set pelatihan dan pengujian, penting untuk mengevaluasi kinerja model secara objektif menggunakan data yang tidak digunakan selama pelatihan. Hal ini membantu dalam mengukur dan memvalidasi keakuratan serta kualitas prediksi model pada data baru.

2. Membangun model

Membangun model dalam PyTorch melibatkan mendefinisikan arsitektur jaringan saraf tiruan dengan menggunakan modul torch.nn. Anda bisa membuat kelas yang mewarisi dari torch.nn.Module dan mendefinisikan lapisan-lapisan atau komponen-komponen jaringan saraf di dalamnya. Setiap lapisan atau komponen tersebut dapat berupa lapisan-lapisan linier seperti torch.nn.Linear, fungsi aktivasi seperti torch.nn.ReLU, atau lapisan-lapisan lainnya yang diperlukan dalam arsitektur jaringan Anda.

- Pentingnya pembuatan model PyTorch

Pembuatan model dalam PyTorch memiliki peran penting dalam pengembangan solusi machine learning. Melalui pembuatan model, Anda dapat mengatur arsitektur jaringan saraf yang sesuai dengan permasalahan yang dihadapi. Dengan menggunakan modul torch.nn, Anda dapat mendefinisikan lapisan-lapisan jaringan, fungsi aktivasi, dan operasi-operasi khusus lainnya yang dibutuhkan dalam pengolahan data. Model yang dibangun memungkinkan Anda untuk melakukan prediksi, klasifikasi, atau regresi terhadap data yang belum pernah dilihat sebelumnya, memungkinkan proses pembelajaran mesin.

- Memeriksa konten model PyTorch

NAMA : DERY HIDAYAT
NIM : 1103228181

Untuk memeriksa konten model PyTorch, Anda dapat mencetak model untuk melihat strukturnya, termasuk lapisan-lapisan yang telah Anda definisikan, dengan menampilkan setiap komponen dalam model. Anda juga dapat mengakses parameter-parameter model menggunakan metode model.parameters() untuk mendapatkan akses ke bobot dan bias di setiap lapisan. Melalui iterasi pada setiap lapisan dalam model, Anda bisa mendapatkan informasi lebih rinci tentang struktur dan konten tiap komponen, membantu Anda untuk melakukan pengecekan yang lebih mendalam terhadap model yang telah Anda bangun dalam PyTorch.

- Membuat prediksi menggunakan torch.inference_mode()

Dalam PyTorch, torch.inference_mode() bukanlah fungsi atau metode yang ada. Namun, untuk membuat prediksi menggunakan model yang telah dilatih, Anda dapat menggunakan model dengan mode evaluasi (model.eval()) untuk menentukan bahwa model sedang digunakan untuk inferensi, bukan pelatihan. Setelah model diubah ke mode evaluasi, Anda dapat memberikan data masukan kepada model menggunakan metode model.forward() atau hanya dengan memanggil model dengan data masukan sebagai argumen. Ini akan menghasilkan prediksi berdasarkan data yang diberikan dengan model yang telah dilatih sebelumnya dalam mode evaluasi. Perlu diingat bahwa Anda harus memastikan bahwa data yang diberikan kepada model telah dipersiapkan dengan benar, termasuk preprocessing yang mungkin dibutuhkan sebelum inferensi dilakukan.

3. Model kereta api

Model kereta api bisa merujuk pada beberapa hal tergantung pada konteksnya. Dalam dunia teknologi atau komputasi, istilah "model" juga dapat mengacu pada representasi matematis atau pemodelan matematika yang digunakan untuk meramalkan perilaku, seperti model prediktif dalam machine learning. Namun, jika berbicara secara harfiah tentang "model kereta api," itu mungkin merujuk pada representasi fisik, digital, atau matematis dari kereta api atau jaringan kereta api, yang bisa berupa gambaran topologi jaringan, pemodelan gerakan kereta api, atau analisis kinerja dan perencanaan dalam sistem transportasi kereta api. Istilah "model" di sini bisa merujuk pada berbagai konsep tergantung pada konteks dan bidangnya.

- Membuat fungsi kerugian dan pengoptimal di PyTorch

Dalam PyTorch, untuk melatih model, Anda dapat membuat fungsi kerugian (loss function) seperti torch.nn.CrossEntropyLoss() untuk masalah klasifikasi atau torch.nn.MSELoss() untuk masalah regresi. Fungsi kerugian ini menghitung seberapa jauh prediksi model dari nilai yang sebenarnya, yang nantinya akan dioptimalkan selama pelatihan. Selain itu, Anda dapat memilih algoritma optimasi seperti torch.optim.SGD() atau torch.optim.Adam() yang membantu menyesuaikan bobot model untuk mengurangi nilai kerugian. Dengan memasukkan fungsi kerugian dan pengoptimal ke dalam siklus pelatihan (loop), Anda dapat melatih model Anda sehingga dapat memberikan prediksi yang lebih baik sesuai dengan data latih yang diberikan.

- Membuat loop pengoptimalan di PyTorch

Untuk membuat loop pengoptimalan di PyTorch, Anda dapat menggunakan struktur dasar yang terdiri dari langkah-langkah utama: pertama, tentukan model Anda dan inisialisasi optimizer PyTorch dengan parameter yang ingin dioptimalkan serta learning rate. Kemudian, dalam loop

iterasi, lakukan forward pass untuk memprediksi output, hitung loss function antara prediksi dan target, lakukan backward pass untuk menghitung gradien, dan terakhir, perbarui parameter model menggunakan optimizer.step(). Pastikan untuk menjaga catatan loss dan iterasi dalam loop sehingga Anda dapat memantau proses pelatihan dan evaluasi model Anda.

- Lingkaran pelatihan PyTorch

Lingkaran pelatihan (training loop) dalam PyTorch mencakup beberapa langkah kunci untuk melatih model. Pertama, siapkan data pelatihan dan definisikan model yang ingin Anda latih. Kemudian, dalam iterasi, gunakan data pelatihan sebagai masukan untuk model dengan mode pelatihan (model.train()), hitung output model, hitung nilai kerugian (loss), dan kalkulasikan gradien menggunakan loss.backward() untuk menghitung arah perubahan parameter. Selanjutnya, gunakan optimasi (optimizer) untuk memperbarui parameter menggunakan gradien yang dihitung, dengan perintah optimizer.step(). Ulangi langkah-langkah ini untuk setiap iterasi (epoch) sampai kriteria berhenti yang ditentukan (seperti jumlah iterasi maksimum atau nilai kerugian yang cukup rendah) terpenuhi, dan model telah melatih dirinya dengan pola yang ada dalam data latih. Selama proses ini, Anda dapat memantau kemajuan pelatihan dan evaluasi model pada data validasi untuk memastikan kinerjanya membaik.

- Lingkaran pengujian PyTorch

Lingkaran pengujian (testing loop) dalam PyTorch biasanya melibatkan evaluasi kinerja model setelah proses pelatihan. Anda akan menggunakan data pengujian yang tidak digunakan selama pelatihan untuk menguji model yang telah dilatih sebelumnya. Dalam iterasi, gunakan model dengan mode evaluasi (model.eval()), lalu berikan data pengujian ke model untuk mendapatkan prediksi. Selanjutnya, hitung metrik evaluasi seperti akurasi, presisi, recall, atau metrik lainnya yang sesuai untuk masalah yang dihadapi. Evaluasi ini membantu menilai kinerja model dalam memprediksi data baru yang tidak terlihat selama pelatihan, memungkinkan untuk mengetahui seberapa baik model dapat melakukan generalisasi pada data baru yang tidak dilihat sebelumnya. Proses pengujian membantu memastikan bahwa model yang dilatih tidak hanya dapat mempelajari pola dalam data pelatihan, tetapi juga dapat memberikan prediksi yang akurat pada data baru.

4. Membuat prediksi dengan model PyTorch terlatih (inferensi)

Untuk membuat prediksi dengan model PyTorch yang telah dilatih (inferensi), Anda perlu mempersiapkan data uji dengan transformasi yang sama yang digunakan pada data pelatihan. Gunakan model dengan mode evaluasi (model.eval()) untuk memastikan bahwa model tidak memperbarui parameter saat melakukan inferensi. Kemudian, berikan data uji ke model, entah dengan menggunakan metode model.forward() atau langsung memanggil model dengan data uji sebagai argumen. Hasil dari inferensi tersebut akan menjadi prediksi model terhadap data baru yang diberikan, memungkinkan Anda untuk melihat bagaimana model mengklasifikasikan atau meramalkan nilai pada data yang tidak terlihat selama proses pelatihan.

5. Menyimpan dan memuat model PyTorch

NAMA : DERY HIDAYAT
NIM : 1103228181

Untuk menyimpan model PyTorch yang telah dilatih, Anda dapat menggunakan torch.save() untuk menyimpan model ke dalam file. Prosesnya melibatkan menyimpan status dari model, termasuk struktur arsitektur, parameter-parameter, dan informasi lain yang diperlukan untuk membangun ulang model. Setelah model disimpan, Anda dapat memuatnya kembali menggunakan torch.load() untuk mendapatkan kembali status model yang telah disimpan sebelumnya. Ini memungkinkan Anda untuk menggunakan kembali model yang telah dilatih tanpa perlu melatih ulang dari awal, memungkinkan penggunaan model pada situasi praktis di masa depan atau berbagi model dengan orang lain untuk penggunaan atau penelitian lanjutan.

- Menyimpan state_dict() model PyTorch

Fungsi state_dict() pada model PyTorch mengembalikan dictionary yang berisi status dari seluruh parameter yang terasosiasi dengan model. Anda bisa menyimpan dan memuat dictionary ini menggunakan torch.save() dan torch.load() untuk menghemat ruang penyimpanan serta memudahkan proses transfer dan pemuatan model. Dengan menyimpan state_dict() model, Anda bisa menghindari menyimpan seluruh struktur model secara terpisah, yang memungkinkan untuk lebih fleksibel dalam pengelolaan status model, termasuk kustomisasi dan transfer model antar perangkat atau platform. Proses ini memungkinkan Anda untuk memuat kembali parameter-parameter model ke model dengan struktur yang sama, memfasilitasi konsistensi dalam penggunaan dan pengembangan model PyTorch.

- Memuat state_dict() model PyTorch yang disimpan

Untuk memuat kembali state_dict() yang telah disimpan dari model PyTorch, Anda bisa menggunakan torch.load() untuk membaca file yang berisi dictionary state_dict() yang telah disimpan sebelumnya. Kemudian, Anda dapat menerapkan load_state_dict() pada model PyTorch untuk memuat kembali parameter-parameter yang terkandung dalam state_dict() tersebut ke model yang bersangkutan. Hal ini memungkinkan Anda untuk merekonstruksi status model dari parameter-parameter yang tersimpan sebelumnya, memungkinkan penggunaan ulang atau transfer parameter-model dengan mudah ke model yang sama atau berbeda.

6. Menyatukan semuanya

Dalam PyTorch, untuk menyimpan model yang telah dilatih, Anda dapat menggunakan torch.save() untuk menyimpan state_dict dari model ke dalam file. Setelah model disimpan, Anda dapat memuatnya kembali menggunakan torch.load() untuk mendapatkan kembali state_dict yang tersimpan sebelumnya. Dengan menyimpan state_dict, Anda dapat menghindari menyimpan seluruh struktur model secara terpisah, yang memungkinkan penghematan ruang penyimpanan dan transfer model yang lebih mudah antar perangkat atau platform. Proses ini memungkinkan Anda untuk merekonstruksi status model dari parameter-parameter yang tersimpan sebelumnya, memfasilitasi penggunaan ulang, pengembangan lanjutan, atau transfer model di masa depan.

- Data

Data dalam konteks PyTorch adalah informasi yang diperlukan untuk melatih, menguji, atau menerapkan model. Ini dapat berupa dataset untuk pelatihan dan pengujian, biasanya terdiri dari

NAMA : DERY HIDAYAT
NIM : 1103228181

input dan output yang sesuai, serta transformasi yang diperlukan untuk mempersiapkan data tersebut sebelum digunakan oleh model. Data juga dapat mencakup metadata, seperti label atau informasi kelas pada dataset klasifikasi, yang memungkinkan model untuk mempelajari pola dan menghasilkan prediksi atau output yang relevan. Pengelolaan data yang tepat, termasuk pemisahan dataset menjadi data pelatihan dan pengujian serta normalisasi atau pembersihan data, sangat penting dalam pengembangan model yang efektif dan generalisasi yang baik ke data baru.

- Membangun model linier PyTorch

Untuk membangun model linear dalam PyTorch, Anda bisa menggunakan modul torch.nn.Linear(). Langkah pertama adalah mendefinisikan struktur model dengan menentukan jumlah input (features) dan output yang diinginkan. Contohnya, untuk membuat model linear sederhana dengan 1 input dan 1 output, Anda bisa menggunakan torch.nn.Linear(1, 1) yang menunjukkan 1 input dan 1 output. Anda juga dapat menambahkan lapisan-lapisan tambahan untuk membuat model yang lebih kompleks dengan menambahkan lebih banyak neuron atau lapisan-lapisan berikutnya, yang memungkinkan model untuk mempelajari representasi yang lebih abstrak dari data. Setelah struktur model ditentukan, Anda dapat menggunakan model tersebut untuk melakukan inferensi atau melatihnya dengan data yang sesuai.

- Pelatihan

Pelatihan model dalam konteks PyTorch melibatkan serangkaian langkah yang melibatkan iterasi pada dataset pelatihan untuk mengoptimalkan parameter-parameter model. Langkah-langkah ini mencakup proses memasukkan data ke dalam model, menghitung output prediksi, mengukur kesalahan antara prediksi dan target yang sebenarnya, lalu melakukan optimisasi pada parameter-parameter model berdasarkan kesalahan tersebut. Proses ini dilakukan berulang kali (beberapa epoch) dengan menggunakan algoritma optimisasi seperti stochastic gradient descent (SGD) atau algoritma varian seperti Adam untuk menyesuaikan parameter-model agar memberikan hasil yang lebih baik. Tujuan dari pelatihan adalah agar model dapat belajar pola dari data dan memberikan prediksi yang tepat pada data baru yang belum pernah dilihat sebelumnya. Proses pelatihan model dilakukan dengan menggunakan data pelatihan dan validasi untuk mengoptimalkan kinerja model terhadap data baru yang tidak terlihat selama proses pelatihan.

- Membuat prediksi

Untuk membuat prediksi menggunakan model PyTorch yang telah dilatih, siapkan data yang akan diprediksi dalam format yang sesuai, pastikan model berada dalam mode evaluasi dengan model.eval(), kemudian gunakan data tersebut sebagai input ke model dengan model(input_data) untuk mendapatkan hasil prediksi, dimana input_data adalah tensor PyTorch yang berisi data yang akan diprediksi.

- Menyimpan dan memuat model

Untuk menyimpan model PyTorch setelah pelatihan, gunakan torch.save(model.state_dict(), 'nama_file.pth') untuk menyimpan parameter model ke dalam file. Kemudian, untuk memuat model yang telah disimpan, buat model dengan arsitektur yang sama, lalu load parameter yang telah disimpan menggunakan model.load_state_dict(torch.load('nama_file.pth')), yang akan

NAMA : DERY HIDAYAT
NIM : 1103228181

memuat parameter ke model yang sudah dibuat sebelumnya, sehingga model dapat digunakan untuk inferensi atau dilatih lebih lanjut. Pastikan bahwa arsitektur model yang digunakan untuk memuat cocok dengan arsitektur model saat.

02. PyTorch Neural Network Classification

- Apa yang dimaksud dengan masalah klasifikasi?

Masalah klasifikasi adalah jenis masalah dalam pembelajaran mesin yang bertujuan untuk memprediksi kategori atau kelas dari data input berdasarkan sejumlah fitur atau atribut yang diberikan. Tujuan utamanya adalah untuk mengelompokkan data ke dalam kategori atau kelas yang telah ditentukan sebelumnya. Contohnya adalah mengklasifikasikan email sebagai spam atau bukan spam, mengidentifikasi jenis bunga berdasarkan atributnya, atau mengenali digit tulisan tangan sebagai angka 0-9. Di sini, model belajar untuk memahami pola dari data latih yang diketahui labelnya dan kemudian menggeneralisasikan pola-pola tersebut untuk memprediksi kelas atau label dari data baru yang belum pernah dilihat sebelumnya.

- Arsitektur jaringan saraf klasifikasi

Arsitektur jaringan saraf untuk klasifikasi merupakan struktur model yang dirancang untuk memproses input dan mengklasifikasikan data ke dalam kategori atau kelas yang berbeda. Arsitektur ini umumnya terdiri dari lapisan-lapisan yang saling terhubung, seperti lapisan input, lapisan tersembunyi (hidden), dan lapisan output. Lapisan-lapisan tersebut memiliki neuron atau unit yang memiliki berbagai fungsi, seperti mempelajari representasi-fitur yang lebih abstrak dari data dan menghasilkan output yang sesuai dengan kelas yang ada. Contohnya, jaringan saraf konvolusional (CNN) umumnya digunakan untuk klasifikasi gambar dengan memanfaatkan lapisan konvolusi untuk mengekstraksi fitur-fitur visual, diikuti dengan lapisan pooling dan lapisan terhubung yang menghasilkan prediksi kelas dari gambar yang diberikan. Arsitektur klasifikasi yang efektif dan optimal bervariasi tergantung pada jenis data dan tugas klasifikasi yang dihadapi.

- Buatlah data klasifikasi dan siapkan

Data klasifikasi umumnya terdiri dari fitur atau atribut yang mendefinisikan setiap entitas dan label yang menunjukkan kategori atau kelas entitas tersebut. Sebagai contoh, untuk klasifikasi email sebagai spam atau bukan spam, fitur-fitur dapat berupa jumlah kata kunci tertentu, kehadiran tautan eksternal, atau pola-pola kata tertentu. Setiap email akan memiliki label yang menunjukkan apakah email tersebut termasuk spam atau tidak. Data klasifikasi biasanya terbagi menjadi dua bagian: data latih untuk melatih model, dan data uji untuk menguji performa model yang dilatih dalam mengklasifikasikan data baru yang belum pernah dilihat sebelumnya.

- Bentuk masukan dan keluaran

Dalam konteks klasifikasi, masukan (input) umumnya terdiri dari fitur atau atribut yang menggambarkan entitas yang ingin diklasifikasikan. Misalnya, jika kita ingin mengklasifikasikan gambar, masukan bisa berupa piksel-piksel gambar. Jika kita ingin mengklasifikasikan teks,

NAMA : DERY HIDAYAT
NIM : 1103228181

masukan bisa berupa kata-kata atau token-token yang mewakili teks tersebut. Sedangkan keluaran (output) dari model klasifikasi biasanya adalah prediksi label atau kelas dari entitas yang diberikan masukan. Dalam klasifikasi biner, keluaran bisa berupa label biner (misalnya, 0 atau 1, spam atau bukan spam), sedangkan dalam klasifikasi multi-kelas, keluaran bisa berupa label dari setiap kelas yang mungkin. Dengan kata lain, masukan adalah informasi yang dimasukkan ke dalam model untuk diproses, sedangkan keluaran adalah hasil prediksi atau klasifikasi yang dihasilkan oleh model dari masukan tersebut.

- Ubah data menjadi tensor dan buat pemisahan pelatihan dan pengujian

Untuk mengonversi data ke tensor dalam PyTorch, gunakan torch.tensor() dengan data masukan dan label yang ingin diubah ke dalam format tensor. Setelah itu, bagi data tersebut menjadi data latih dan data uji menggunakan metode seperti train_test_split() dari pustaka sklearn.model_selection, yang memungkinkan kita untuk memisahkan data dengan proporsi tertentu. Dengan mengatur test_size sebagai proporsi data uji, kita dapat memperoleh data latih dan data uji yang siap digunakan untuk melatih dan menguji model klasifikasi. Pastikan untuk menyesuaikan dengan format dan struktur data yang sesuai sebelum konversi ke tensor dan pemisahan data.

- Membangun model

Untuk membangun model klasifikasi menggunakan PyTorch, langkah pertama adalah mendefinisikan struktur model, misalnya, dengan membuat kelas turunan dari torch.nn.Module dan mendefinisikan lapisan-lapisan yang dibutuhkan seperti lapisan-lapisan linier (torch.nn.Linear) atau lapisan-lapisan konvolusi (torch.nn.Conv2d). Setelah itu, tentukan metode forward() untuk menentukan aliran data melalui model, menentukan bagaimana input akan melewati setiap lapisan. Selanjutnya, inisialisasi fungsi loss seperti torch.nn.CrossEntropyLoss() dan pilih optimizer seperti torch.optim.SGD atau torch.optim.Adam. Akhiri dengan melatih model menggunakan data latih dengan melakukan iterasi melalui epoch, melakukan prediksi pada data uji untuk evaluasi, dan melakukan pembaruan parameter menggunakan optimizer dengan optimizer.step(). Pastikan untuk menyesuaikan struktur model dan hiperparameter seperti learning rate, jumlah epoch, dan fungsi loss sesuai dengan kebutuhan tugas klasifikasi yang Anda hadapi.

- Atur fungsi kerugian dan pengoptimal

Untuk menyiapkan fungsi kerugian (loss function) dalam model klasifikasi PyTorch, Anda dapat menggunakan fungsi seperti torch.nn.CrossEntropyLoss() untuk masalah klasifikasi multi-kelas atau torch.nn.BCEWithLogitsLoss() untuk klasifikasi biner. Fungsi loss ini akan menghitung kerugian antara prediksi dan label yang sebenarnya. Selanjutnya, pilih pengoptimal (optimizer) seperti torch.optim.SGD() untuk stokastik gradient descent atau torch.optim.Adam() yang lebih adaptif dengan kecepatan pembelajaran yang berbeda-beda. Inisialisasikan optimizer dengan parameter-parameter seperti learning rate, momentum, dan weight decay sesuai kebutuhan, dan gunakan optimizer tersebut untuk memperbarui parameter-model saat proses pelatihan dengan memanggil optimizer.step(). Pastikan untuk menyesuaikan fungsi kerugian dan pengoptimal

NAMA : DERY HIDAYAT
NIM : 1103228181

dengan jenis masalah yang Anda hadapi serta eksperimen dengan berbagai pengaturan hiperparameter untuk meningkatkan kinerja model Anda.

- Train model

Dalam loop pelatihan PyTorch, iterasi dilakukan melalui data latih dalam batch-batch kecil. Setiap iterasi melibatkan langkah-langkah utama seperti menghapus gradien sebelum penyesuaian parameter, melakukan forward pass dengan input batch ke model untuk mendapatkan prediksi, menghitung loss antara prediksi dan label yang sebenarnya, melakukan backward pass untuk menghitung gradien loss terhadap parameter, dan akhirnya, menggunakan optimizer untuk mengoptimalkan parameter-model. Proses ini diulangi sepanjang epoch, dan selama proses ini, model secara bertahap belajar memperbarui bobotnya untuk mengurangi loss dan meningkatkan kinerja prediktifnya terhadap data latih.

- Beralih dari keluaran model mentah ke label prediksi (logit -> probabilitas prediksi -> label prediksi)

Beralih dari keluaran model mentah (logit) ke label prediksi melibatkan langkah-langkah tambahan untuk menghasilkan interpretasi yang lebih intuitif dari hasil prediksi. Logit yang merupakan nilai mentah dari model perlu diubah menjadi probabilitas menggunakan fungsi softmax, yang mengonversi nilai-nilai logit menjadi distribusi probabilitas untuk setiap kelas. Setelah mendapatkan probabilitas untuk setiap kelas, label prediksi dapat diambil dengan memilih kelas dengan probabilitas tertinggi sebagai prediksi akhir. Langkah ini membantu dalam menginterpretasikan hasil prediksi model dengan memberikan label atau kelas yang memiliki probabilitas paling tinggi, membuatnya lebih mudah dipahami secara intuitif.

- Membangun lingkaran pelatihan dan pengujian

Proses pelatihan dan pengujian model dalam PyTorch melibatkan pembuatan dua lingkaran terpisah: satu untuk pelatihan model dan satu lagi untuk pengujian. Untuk lingkaran pelatihan, iterasi dilakukan melalui data latih dalam batch-batch kecil. Dalam setiap iterasi, langkah-langkah utama meliputi: mengirimkan batch data ke model untuk mendapatkan prediksi, menghitung loss antara prediksi dan label yang sebenarnya, melakukan backward pass untuk menghitung gradien loss, dan mengoptimalkan parameter-model menggunakan optimizer. Selama proses ini, model terus diperbarui untuk meningkatkan performanya. Setelah pelatihan selesai, lingkaran pengujian dijalankan dengan meneruskan data uji ke model untuk memperoleh prediksi. Kemudian, hasil prediksi dibandingkan dengan label sebenarnya untuk mengukur performa model, seperti akurasi atau metrik evaluasi lainnya, untuk mengevaluasi seberapa baik model bekerja dengan data yang belum pernah dilihat sebelumnya.

- Buat prediksi dan evaluasi model

Untuk membuat prediksi menggunakan model yang telah dilatih di PyTorch, langkah pertama adalah mengirimkan data uji ke model menggunakan metode model.eval() untuk memastikan model berada dalam mode evaluasi. Kemudian, gunakan with torch.no_grad() untuk menonaktifkan perhitungan gradien saat melakukan prediksi. Selanjutnya, gunakan model untuk memperoleh prediksi dengan memasukkan data uji ke dalam model. Setelah mendapatkan

NAMA : DERY HIDAYAT
NIM : 1103228181

prediksi, Anda dapat menghitung metrik evaluasi seperti akurasi, presisi, recall, atau menggunakan metrik lainnya yang sesuai dengan tugas klasifikasi Anda untuk mengevaluasi kinerja model terhadap data uji tersebut.

- Meningkatkan model (dari perspektif model)

Meningkatkan model dari perspektif model dapat dilakukan dengan berbagai cara. Salah satunya adalah dengan menambahkan kompleksitas model, seperti menambahkan lebih banyak lapisan atau unit di dalam setiap lapisan. Ini dapat meningkatkan kapasitas model untuk mempelajari pola yang lebih kompleks dari data. Selain itu, Anda bisa menggunakan teknik-teknik seperti regularisasi, seperti dropout atau weight decay, untuk mencegah overfitting dan membuat model lebih umum atau general. Menggunakan arsitektur yang lebih canggih seperti pre-trained models atau menggabungkan teknik transfer learning juga bisa meningkatkan kinerja model. Percobaan dengan hyperparameter seperti learning rate, jumlah epoch, atau batch size juga bisa membantu menemukan konfigurasi yang lebih baik untuk model Anda. Terus melakukan eksperimen dan evaluasi terhadap berbagai strategi untuk meningkatkan performa model adalah kunci dalam peningkatan model dari segi arsitektur atau konfigurasi.

- Mempersiapkan data untuk melihat apakah model kita dapat memodelkan garis lurus

Untuk menguji apakah model dapat memodelkan garis lurus, pertama-tama, siapkan dataset yang berisi hubungan linier antara fitur dan target. Misalnya, buatlah dataset dengan satu fitur (X) yang memiliki hubungan linier dengan target (y). Buat data dengan persamaan $y = mx + c$ (dengan m sebagai gradien dan c sebagai konstanta) atau persamaan linier serupa. Kemudian, bagi dataset menjadi data latih dan data uji. Selanjutnya, latih model linier, seperti Regresi Linier di PyTorch, dengan menggunakan data latih dan evaluasi performa model dengan data uji. Jika model berhasil menyesuaikan data latih dan memberikan hasil yang baik pada data uji, menunjukkan bahwa model mampu memodelkan garis lurus. Jika hasilnya kurang memuaskan, bisa jadi perlu menyesuaikan kompleksitas model atau melakukan penyesuaian hiperparameter. Namun, perlu diingat bahwa keberhasilan model dalam memodelkan garis lurus bukanlah jaminan bahwa model tersebut akan sukses dalam memodelkan hubungan yang lebih kompleks dalam data yang lebih realistis.

- Menyesuaikan model_1 agar sesuai dengan garis lurus

Untuk menyesuaikan model agar sesuai dengan garis lurus, Anda dapat menggunakan Regresi Linier dalam PyTorch. Pertama, pastikan dataset yang Anda gunakan memiliki hubungan linier antara fitur dan target. Selanjutnya, buat model Regresi Linier yang memiliki satu lapisan linier dengan fungsi aktivasi identitas (linier). Kemudian, lakukan pelatihan model menggunakan data latih, dengan optimizer seperti SGD atau Adam, dan fungsi loss seperti Mean Squared Error (MSE). Sesuaikan jumlah epoch, learning rate, dan ukuran batch agar model belajar dengan baik tanpa overfitting atau underfitting. Setelah pelatihan selesai, evaluasi model menggunakan data uji untuk melihat seberapa baik model dapat memprediksi nilai target berdasarkan fitur yang diberikan.

- Bagian yang hilang: non-linearitas

NAMA : DERY HIDAYAT
NIM : 1103228181

Untuk membuat data non-linier dengan dua lingkaran terpisah, Anda dapat menggunakan pendekatan geometris dengan menghasilkan titik-titik secara acak di sekitar pusat lingkaran menggunakan koordinat polar (r, θ). Dalam pendekatan ini, titik-titik koordinat polar diubah menjadi koordinat kartesian (x, y) dengan menggunakan rumus trigonometri, yang kemudian ditetapkan sebagai data untuk mewakili dua kelompok yang membentuk dua lingkaran terpisah. Dengan cara ini, Anda bisa membuat dataset dengan dua kelompok yang memiliki pola non-linier, memungkinkan pengujian kemampuan model dalam memahami hubungan non-linier antara fitur dan label. Model yang mampu menangkap dan mempelajari pola non-linier ini akan lebih efektif dalam mengklasifikasikan data dan mengenali hubungan yang kompleks di antara kelompok-kelompok tersebut.

- Membuat ulang data non-linier (lingkaran merah dan biru)

Untuk membuat data non-linier dengan dua lingkaran terpisah, Anda dapat menggunakan pendekatan geometris dengan menghasilkan titik-titik secara acak di sekitar pusat lingkaran menggunakan koordinat polar (r, θ). Dalam pendekatan ini, titik-titik koordinat polar diubah menjadi koordinat kartesian (x, y) dengan menggunakan rumus trigonometri, yang kemudian ditetapkan sebagai data untuk mewakili dua kelompok yang membentuk dua lingkaran terpisah. Dengan cara ini, Anda bisa membuat dataset dengan dua kelompok yang memiliki pola non-linier yang sesuai untuk uji coba model klasifikasi.

- Membangun model dengan non-linearitas

Untuk membangun model yang mampu menangani non-linearitas, terutama dalam kasus data seperti dua lingkaran terpisah, Anda bisa menggunakan arsitektur jaringan saraf yang lebih kompleks, seperti jaringan saraf tiruan (neural networks) dengan lapisan-lapisan non-linear, seperti ReLU (Rectified Linear Unit) atau fungsi aktivasi lainnya di antara lapisan-lapisan. Contohnya, Anda bisa menggunakan jaringan saraf tiruan dengan beberapa lapisan tersembunyi yang memiliki fungsi aktivasi non-linear (ReLU, tanh, sigmoid, dll.) dan lapisan output dengan fungsi aktivasi yang sesuai untuk tugas klasifikasi (misalnya, softmax untuk klasifikasi multi-kelas). Ini memungkinkan model untuk mempelajari dan menangkap pola non-linear yang kompleks dalam data seperti dua lingkaran terpisah, membantu dalam pengklasifikasian dengan lebih baik dibandingkan dengan model linier yang hanya mampu menangani hubungan linear sederhana antara fitur dan label.

- Melatih model dengan non-linearitas

Untuk melatih model dengan kemampuan menangani non-linearitas seperti dua lingkaran terpisah, Anda dapat menggunakan jaringan saraf tiruan (neural networks) dengan fungsi aktivasi non-linear seperti ReLU atau sigmoid di antara lapisan-lapisan. Pertama, definisikan arsitektur model dengan lapisan-lapisan tersembunyi yang memiliki fungsi aktivasi non-linear. Kemudian, gunakan data yang telah Anda buat sebelumnya (dua lingkaran terpisah) sebagai data latih dan data uji. Selanjutnya, definisikan fungsi loss seperti CrossEntropyLoss() untuk klasifikasi multi-kelas atau BCEWithLogitsLoss() untuk klasifikasi biner, dan pilih optimizer seperti SGD atau Adam. Selama proses pelatihan, iterasi melalui epoch dan update parameter-model menggunakan gradien dari fungsi loss. Setelah pelatihan selesai, evaluasi model menggunakan data uji untuk

melihat seberapa baik model dapat membedakan antara kedua lingkaran. Proses ini memungkinkan model untuk mempelajari pola non-linear dan meningkatkan kemampuannya dalam memprediksi dengan benar kelas data yang kompleks seperti dua lingkaran terpisah.

- Mengevaluasi model yang dilatih dengan fungsi aktivasi non-linier

Setelah melatih model dengan fungsi aktivasi non-linear, langkah evaluasi terhadap model dilakukan dengan menerapkan model terlatih pada data uji dan menghitung metrik evaluasi yang relevan seperti akurasi, presisi, recall, atau F1-score untuk mengukur kinerja model dalam mengklasifikasikan data yang memiliki pola non-linear seperti dua lingkaran terpisah. Fungsi aktivasi non-linear membantu model dalam memahami dan menangkap pola yang kompleks, memungkinkan model untuk membedakan kelas-kelas yang terpisah dengan lebih baik dalam data uji yang belum pernah dilihat sebelumnya. Evaluasi ini membantu dalam memahami seberapa baik model mampu menggeneralisasi pola yang dipelajari selama pelatihan terhadap data baru yang tidak terlibat dalam proses pelatihan.

- Mereplikasi fungsi aktivasi non-linier

Fungsi aktivasi non-linear merupakan elemen penting dalam jaringan saraf yang memungkinkan model untuk mempelajari dan menangkap pola-pola yang kompleks dalam data. Beberapa fungsi aktivasi non-linear yang umum digunakan termasuk ReLU (Rectified Linear Unit), Sigmoid, dan Tanh. Mereplikasi fungsi aktivasi ini memungkinkan model jaringan saraf tiruan untuk menambahkan non-linearitas dalam pemrosesan data di setiap lapisannya. ReLU, yang mengonversi semua nilai negatif menjadi nol dan mempertahankan nilai positif, sering digunakan karena efisiensinya dalam pelatihan dan mengatasi masalah gradien yang menghilang. Sigmoid dan Tanh, dengan output terbatas antara 0-1 dan -1 hingga 1, secara bergantian digunakan terutama di lapisan-lapisan tersembunyi dalam model untuk mengontrol rentang nilai yang dipropagasi di antara lapisan-lapisan tersebut. Menyertakan fungsi aktivasi non-linear yang tepat dalam model memungkinkan jaringan untuk mempelajari hubungan non-linear antara fitur dan label dalam data.

- Menyatukan semuanya dengan membangun model PyTorch multikelas

Untuk membangun model klasifikasi multikelas dalam PyTorch, definisikan arsitektur jaringan saraf tiruan dengan beberapa lapisan tersembunyi yang menggunakan fungsi aktivasi non-linear seperti ReLU, diikuti oleh lapisan output dengan jumlah neuron yang sesuai dengan jumlah kelas yang ingin diprediksi, dan gunakan fungsi aktivasi softmax. Inisialisasi model dengan parameter yang sesuai, seperti ukuran input (jumlah fitur), ukuran lapisan tersembunyi, dan jumlah kelas output. Kemudian, tentukan loss function seperti CrossEntropyLoss() untuk klasifikasi multikelas dan optimizer seperti SGD atau Adam. Setelah itu, jalankan proses pelatihan dan evaluasi model menggunakan data latih dan data uji, iterasi melalui epoch, dan evaluasi kinerja model dengan metrik evaluasi yang relevan untuk tugas klasifikasi multikelas Anda.

- Membuat data klasifikasi kelas jamak

Untuk membuat data klasifikasi kelas jamak, Anda dapat menggunakan beberapa fitur untuk membedakan antara beberapa kelas. Misalnya, dalam kasus data citra, Anda dapat

NAMA : DERY HIDAYAT
NIM : 1103228181

mengumpulkan berbagai jenis citra yang mewakili berbagai kategori (misalnya, kucing, anjing, dan burung). Setiap citra akan diwakili oleh fitur-fitur seperti warna, tekstur, atau bentuk yang diekstraksi dengan teknik seperti ekstraksi fitur menggunakan metode Convolutional Neural Network (CNN) atau ekstraksi manual. Kemudian, gabungkan fitur-fitur ini dengan label kelas (misalnya, 0 untuk kucing, 1 untuk anjing, 2 untuk burung) untuk membentuk dataset klasifikasi. Pastikan untuk memiliki jumlah sampel yang seimbang untuk setiap kelas untuk hasil yang lebih baik saat melatih model. Proses ini memungkinkan Anda untuk membangun dataset yang memungkinkan model untuk belajar membedakan antara beberapa kelas yang berbeda dalam tugas klasifikasi kelas jamak.

- Membangun model klasifikasi kelas jamak di PyTorch

Untuk membangun model klasifikasi multikelas menggunakan PyTorch, langkah pertama adalah mendefinisikan arsitektur jaringan saraf tiruan dengan lapisan-lapisan tersembunyi yang memiliki fungsi aktivasi non-linear seperti ReLU di antara lapisan-lapisan tersebut. Setelahnya, persiapkan data dengan memuat, menormalisasi, dan membagi dataset menjadi data latih dan data uji. Selanjutnya, inisialisasi model dengan parameter yang sesuai, seperti ukuran input (jumlah fitur), ukuran lapisan tersembunyi, dan jumlah kelas output. Kemudian, tentukan loss function seperti CrossEntropyLoss() untuk klasifikasi multikelas dan optimizer seperti SGD atau Adam. Setelahnya, jalankan proses pelatihan dengan melakukan iterasi melalui epoch, menggunakan optimizer untuk mengoptimalkan parameter-model, dan evaluasi model menggunakan metrik evaluasi seperti akurasi atau F1-score untuk memahami performa model dalam mengklasifikasikan berbagai kelas pada data uji.

- Membuat fungsi kerugian dan pengoptimal untuk model PyTorch multikelas

untuk model multikelas dalam PyTorch, umumnya digunakan CrossEntropyLoss() sebagai fungsi kerugian karena cocok untuk tugas klasifikasi multikelas. Fungsi ini menghitung loss berdasarkan perbedaan antara distribusi probabilitas prediksi model dan label kelas sebenarnya. Sebagai contoh, jika output model menggunakan fungsi softmax, CrossEntropyLoss() secara efektif mengukur seberapa dekat distribusi probabilitas prediksi dengan distribusi one-hot encoded dari label kelas. Selanjutnya, dalam hal pengoptimal, menggunakan optimizer seperti SGD (Stochastic Gradient Descent) atau Adam seringkali digunakan, dengan kemampuan untuk mengoptimalkan parameter-model berdasarkan gradien dari fungsi kerugian tersebut selama proses pelatihan.

- Mendapatkan probabilitas prediksi untuk model PyTorch multikelas

Untuk mendapatkan probabilitas prediksi dari model PyTorch pada tugas klasifikasi multikelas, Anda dapat menggunakan fungsi softmax pada keluaran (output) dari model. Setelah mendapatkan hasil prediksi dari model pada data uji, fungsi softmax digunakan untuk mengonversi nilai-nilai output menjadi distribusi probabilitas yang menunjukkan probabilitas relatif dari setiap kelas. Ini memungkinkan Anda untuk memperoleh informasi probabilitas dari model terkait kemungkinan masing-masing kelas untuk setiap sampel data uji yang dievaluasi.

- Membuat loop pelatihan dan pengujian untuk model PyTorch multikelas

NAMA : DERY HIDAYAT
NIM : 1103228181

proses pelatihan dan pengujian model multikelas dalam PyTorch melibatkan iterasi melalui beberapa epoch untuk pelatihan. Dalam setiap epoch, model dievaluasi dengan data latih menggunakan fungsi kerugian, kemudian gradien loss dihitung untuk melakukan penyesuaian parameter-model menggunakan optimizer. Setelah selesai pelatihan, model dievaluasi dengan data uji untuk mengukur performa pada data yang tidak terlibat dalam proses pelatihan. Selama proses pengujian, output model dievaluasi menggunakan metrik evaluasi yang relevan seperti akurasi, presisi, recall, atau F1-score untuk memahami kinerja model dalam melakukan klasifikasi multikelas pada data uji. Iterasi ini memungkinkan peningkatan bertahap dalam kemampuan model dalam memahami pola-pola kompleks dalam data dan meningkatkan kemampuannya dalam mengklasifikasikan kelas-kelas yang berbeda.

- Membuat dan mengevaluasi prediksi dengan model kelas jamak PyTorch

Untuk membuat dan mengevaluasi prediksi dengan model multikelas dalam PyTorch, gunakan model yang telah dilatih untuk membuat prediksi pada data uji dengan mengatur mode evaluasi menggunakan model.eval() dan torch.no_grad() untuk memastikan tidak ada perhitungan gradien selama prediksi. Gunakan output dari model, misalnya dengan menggunakan fungsi torch.max() untuk mendapatkan kelas prediksi. Setelah itu, bandingkan hasil prediksi dengan label yang sebenarnya pada data uji, kemudian gunakan metrik evaluasi yang sesuai seperti akurasi, presisi, recall, atau F1-score untuk mengukur performa model dalam melakukan klasifikasi pada dataset kelas jamak dan memahami seberapa baik model dapat memprediksi label kelas yang benar.

- Metrik evaluasi klasifikasi lainnya

Metrik evaluasi klasifikasi lainnya yang penting meliputi presisi (precision), yang mengukur proporsi positif yang benar dari semua hasil yang diprediksi positif oleh model; recall, yang mengukur proporsi positif yang benar dari keseluruhan kelas positif dalam dataset; F1-score, merupakan harmonic mean dari presisi dan recall memberikan gambaran menyeluruh tentang kinerja model; matriks kebingungan (confusion matrix) yang memberikan informasi rinci tentang prediksi yang benar dan salah untuk setiap kelas; serta AUC-ROC (Area Under the Curve - Receiver Operating Characteristic) yang mengukur kualitas keseluruhan model dalam memisahkan kelas positif dan negatif pada berbagai thresholds. Kombinasi metrik-metrik ini memberikan pemahaman yang lebih lengkap tentang kehandalan dan kemampuan model dalam mengklasifikasikan data pada tugas klasifikasi multikelas.

- Metrik evaluasi klasifikasi lainnya

Selain akurasi, terdapat beberapa metrik evaluasi klasifikasi yang penting untuk memahami kinerja model. Precision mengukur seberapa banyak dari semua prediksi positif yang sebenarnya benar, recall mengukur seberapa banyak dari semua kelas positif yang terdeteksi dengan benar oleh model, F1-score merupakan harmonic mean antara precision dan recall yang memberikan gambaran komprehensif tentang performa model, terutama saat kelas yang tidak seimbang, AUC-ROC (Area Under the Curve - Receiver Operating Characteristic) mengukur kemampuan model dalam memisahkan kelas positif dan negatif pada berbagai thresholds, dan confusion matrix memberikan informasi tentang jumlah prediksi yang benar dan salah dalam setiap kelas, membantu dalam pemahaman kesalahan model pada kelas-kelas tertentu. Memahami dan

NAMA : DERY HIDAYAT
NIM : 1103228181
menggunakan kombinasi metrik-metrik ini memberikan pemahaman yang lebih komprehensif tentang kemampuan model dalam melakukan klasifikasi pada berbagai aspek dari data.

03. PyTorch Computer Vision

PyTorch adalah kerangka kerja pembelajaran mesin yang populer, dan dalam ranah visi komputer, ia menawarkan berbagai fitur dan alat. Berikut adalah penjelasan menyeluruh tentang PyTorch dalam konteks visi komputer:

1. Tensor

- **Dasar PyTorch**: PyTorch menggunakan Tensors, yang mirip dengan array NumPy, tetapi dengan dukungan GPU yang kuat untuk komputasi yang lebih cepat.
- **Manipulasi Data**: Tensors memungkinkan manipulasi data yang efisien dan merupakan blok bangunan utama untuk operasi visi komputer.

2. Autograd Module

- **Diferensiasi Otomatis**: Bagian penting dari pembelajaran mesin adalah menghitung gradien. Modul Autograd di PyTorch menyediakan diferensiasi otomatis untuk operasi tensor.
- **Pelatihan Model**: Ini sangat penting untuk proses pelatihan model, memungkinkan pembelajaran backpropagation yang mudah.

3. Model Neural Network

- **torch.nn**: Modul ini menyediakan semua blok bangunan yang diperlukan untuk membangun jaringan saraf.
- **Layer dan Arsitektur**: Ini termasuk layer seperti Convolutional layers, Pooling layers, dan Fully Connected layers, yang merupakan inti dari aplikasi visi komputer.

4. GPU Acceleration

- **CUDA Support**: PyTorch mendukung NVIDIA CUDA, memungkinkan komputasi yang sangat dipercepat dengan menggunakan GPU.
- **Peningkatan Efisiensi**: Ini sangat penting untuk memproses dataset besar dan melatih model yang kompleks dalam visi komputer.

5. Pretrained Models dan Transfer Learning

- **Model Zoo**: PyTorch menyediakan akses ke model yang telah dilatih sebelumnya seperti ResNet, VGG, dan AlexNet yang dapat digunakan untuk transfer learning.
- **Customization**: Pengguna dapat menyesuaikan model ini untuk kebutuhan spesifik mereka, yang sangat berguna dalam tugas seperti klasifikasi gambar dan deteksi objek.

6. Data Handling

NAMA : DERY HIDAYAT
NIM : 1103228181

- **torchvision**: Paket ini menyediakan dataset populer, transformasi gambar, dan utilitas pemuatan data yang memudahkan pengolahan dan pemuatan dataset gambar.
- **Dataloader**: Memfasilitasi penyediaan batch, shuffling, dan transformasi data, penting untuk mempersiapkan dataset untuk proses pelatihan.

7. Training and Evaluation

- **Loop Pelatihan**: PyTorch memungkinkan kontrol yang luas atas loop pelatihan, termasuk pengaturan batch size, jumlah epoch, dan strategi optimisasi.
- **Evaluasi Model**: Fungsi untuk menguji dan mengevaluasi kinerja model pada data yang tidak terlihat, penting untuk memvalidasi model visi komputer.

8. Serialization

- **Menyimpan dan Memuat Model**: PyTorch menyediakan cara untuk menyimpan model atau state dict untuk penggunaan di masa mendatang, memudahkan deployment model.

9. Komunitas dan Ekosistem

- **Dukungan Komunitas**: PyTorch memiliki komunitas pengembang yang besar dan aktif, menyediakan banyak sumber daya, tutorial, dan dukungan.
- **Integrasi dengan Ekosistem ML**: Mudah diintegrasikan dengan alat dan pustaka ML lainnya, seperti NumPy, Pandas, dan Matplotlib.

10. Advanced Features

- **Jaringan Saraf Spesifik**: Dukungan untuk RNN dan Transformer Networks, penting untuk tugas seperti segmentasi gambar dan pengenalan pola.
- **Custom Layer dan Functions**: Kemampuan untuk menambahkan layer khusus atau fungsi yang tidak standar untuk kebutuhan penelitian atau proyek khusus.

Dengan fitur-fitur ini, PyTorch menjadi salah satu pilihan utama untuk para peneliti dan praktisi di bidang visi komputer, memberikan keseimbangan yang baik antara fleksibilitas dan efisiensi dalam pembangunan dan pelatihan model-model visi komputer.

04. PyTorch Custom Datasets

Pembuatan dataset kustom di PyTorch memungkinkan peneliti dan pengembang untuk bekerja dengan data yang tidak termasuk dalam dataset standar seperti ImageNet, CIFAR-10, atau MNIST. Proses ini melibatkan beberapa langkah kunci:

1. Pemahaman Kelas Dataset

- **Kelas Dasar**: PyTorch menyediakan kelas abstrak **Dataset** yang perlu diperluas saat membuat dataset kustom.

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Metode Wajib**: Ada dua metode yang harus diimplementasikan:
    - **__len__**: Mengembalikan jumlah sampel dalam dataset.
    - **__getitem__**: Mengembalikan sampel (data dan label) dari dataset berdasarkan indeks.

2. Pembuatan Kelas Dataset Kustom

- **Definisi Kelas**: Kelas dataset kustom dibuat dengan meng-extend kelas **Dataset**.
- **Inisialisasi**: Biasanya menginisialisasi jalur file, transformasi data, dan operasi pemuatan data lainnya.
- **Pemuatan Data**: Metode **__getitem__** digunakan untuk membaca data dari disk, menerapkan transformasi, dan mengembalikan data.

3. Transformasi Data

- **torchvision.transforms**: PyTorch menyediakan modul ini untuk transformasi gambar umum seperti scaling, cropping, dan normalisasi.
- **Transformasi Kustom**: Pengguna juga dapat mendefinisikan transformasi kustom jika diperlukan.

4. Pemuatan dan Iterasi Data

- **DataLoader**: Setelah dataset kustom dibuat, **DataLoader** digunakan untuk mengiterasi dataset selama proses pelatihan.
- **Batching dan Shuffling**: **DataLoader** mendukung pembuatan batch, shuffling, dan pemuatan data secara paralel.

5. Integrasi dengan Pipeline Pelatihan

- **Loop Pelatihan**: Dataset yang dimuat dengan **DataLoader** kemudian digunakan dalam loop pelatihan.
- **Kompatibilitas dengan Model**: Dataset kustom harus mengembalikan data dalam format yang kompatibel dengan arsitektur model yang digunakan.

6. Contoh Penggunaan

Misalnya, jika Anda memiliki dataset gambar kucing dan anjing dengan label, kelas dataset kustom Anda akan:

- Menginisialisasi dengan jalur ke dataset.
- Menerapkan transformasi yang diperlukan pada gambar.
- Mengembalikan gambar dan label yang sesuai dalam **__getitem__**.

7. Manajemen Error dan Validasi

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Penanganan Kesalahan**: Tambahkan penanganan kesalahan untuk memastikan bahwa data yang tidak valid tidak memengaruhi proses pelatihan.
- **Splitting Data**: Memisahkan dataset menjadi training, validation, dan test sets jika diperlukan.

8. Efisiensi dan Skalabilitas

- **Pemuatan Data yang Efisien**: Gunakan teknik seperti pemuatan data secara lazy untuk menghemat memori.
- **Multiprocessing**: **DataLoader** mendukung multiprocessing untuk mempercepat pemuatan data.

9. Customisasi Lanjutan

- **Dukungan Format Data Khusus**: Jika bekerja dengan format data yang tidak umum, tambahkan logika pemuatan dan parsing khusus.
- **Integrasi dengan Transformasi Lanjutan**: Misalnya, penggunaan Augmentation Libraries seperti Albumentations.

Dengan membuat dataset kustom di PyTorch, Anda mendapatkan fleksibilitas untuk bekerja dengan hampir semua jenis data, yang sangat penting dalam proyek-proyek penelitian atau aplikasi industri khusus di bidang visi komputer, NLP, atau bidang lainnya.

05. PyTorch Going Modular

"Going Modular" di PyTorch merujuk pada praktik membagi kode dan arsitektur model pembelajaran mesin menjadi komponen yang terpisah dan dapat digunakan kembali. Ini memungkinkan pengembang untuk membuat kode yang lebih terorganisir, mudah dipelihara, dan fleksibel. Berikut adalah penjelasan menyeluruh tentang pendekatan modular di PyTorch:

1. Modularitas dalam Arsitektur Model

- **Pembagian Model**: Model dibagi menjadi bagian-bagian yang lebih kecil atau modul, seperti blok konvolusional, blok residual, atau blok attention.
- **torch.nn.Module**: Semua modul di PyTorch diwarisi dari **nn.Module**, yang menyediakan fungsionalitas dasar untuk membangun lapisan dan model.

2. Penggunaan nn.Module

- **Definisi Modul Kustom**: Modul kustom dibuat dengan meng-extend **nn.Module**, memungkinkan penggunaan dalam berbagai model.
- **Metode forward**: Setiap modul harus mendefinisikan metode **forward** untuk menentukan bagaimana data melewati modul.

3. Komponen Daur Ulang

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Penggunaan Ulang Lapisan**: Modul yang umum seperti lapisan konvolusi atau blok normalisasi dapat digunakan ulang dalam berbagai model.
- **Kustomisasi Fleksibel**: Modul individual dapat disesuaikan tanpa mengubah kode model secara keseluruhan.

4. Manajemen Parameter

- **Parameter Otomatis**: **nn.Module** secara otomatis mengelola parameter (berat dan bias), yang sangat berguna saat melatih dan menyimpan model.
- **Penanganan Parameter dalam Submodul**: **nn.Module** dapat berisi submodul lain, dan PyTorch secara otomatis melacak parameter di semua tingkatan.

5. Model Sequential dan Modular

- **nn.Sequential**: PyTorch menyediakan **nn.Sequential** untuk menumpuk modul secara linier. Ini cocok untuk arsitektur yang lebih sederhana.
- **Model Kustom yang Lebih Kompleks**: Untuk arsitektur yang lebih kompleks, definisi model kustom menggunakan modul individual memberikan lebih banyak fleksibilitas.

6. Ekstraksi Fitur dan Fine-Tuning

- **Model Pra-Latih**: Modul memudahkan ekstraksi fitur dari model yang telah dilatih sebelumnya atau melakukan fine-tuning pada bagian tertentu dari model tersebut.

7. Kode yang Lebih Bersih dan Terorganisir

- **Pemisahan Fungsi**: Dengan modularitas, fungsi spesifik dipisahkan ke dalam modul yang berbeda, membuat kode lebih mudah dibaca dan dipelihara.
- **Pengujian dan Debugging**: Modul yang lebih kecil lebih mudah diuji dan didebug secara terpisah.

8. Penerapan dalam Proyek Skala Besar

- **Skalabilitas**: Dalam proyek besar, modularitas memungkinkan tim untuk bekerja pada bagian yang berbeda dari model secara bersamaan tanpa mengganggu komponen lain.
- **Pemeliharaan dan Pembaruan**: Mudah untuk memperbarui atau memperbaiki bagian tertentu dari model tanpa harus mengubah keseluruhan sistem.

9. Integrasi dengan Alat Lain

- **Kompatibilitas dengan Ekosistem PyTorch**: Modul yang dibuat dapat dengan mudah diintegrasikan dengan alat lain dalam ekosistem PyTorch, seperti DataLoader, optimizer, dan lainnya.

10. Best Practices

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Desain Modular yang Efisien**: Saat mendesain modul, penting untuk mempertimbangkan keseimbangan antara fleksibilitas, efisiensi, dan kerumitan.
- **Dokumentasi dan Nama yang Jelas**: Memberikan dokumentasi yang baik dan nama variabel yang jelas untuk mempermudah pemahaman dan penggunaan modul oleh pengembang lain.

Menerapkan pendekatan modular dalam PyTorch tidak hanya meningkatkan kualitas dan kebersihan kode tetapi juga meningkatkan efisiensi pengembangan, terutama dalam proyek-proyek kolaboratif atau skala besar

06. PyTorch Transfer Learning

Transfer learning di PyTorch adalah teknik di mana model yang telah dilatih pada satu set data dan tugas (biasanya dataset besar dan tugas umum) digunakan sebagai titik awal untuk melatih model pada set data dan tugas baru yang berbeda. Ini sangat berguna dalam deep learning, karena dapat secara signifikan mengurangi waktu pelatihan dan memerlukan data yang lebih sedikit untuk mencapai kinerja yang tinggi.

1. Konsep Dasar Transfer Learning

- **Pemanfaatan Pengetahuan yang Ada**: Menggunakan model yang telah dilatih sebelumnya untuk memanfaatkan pengetahuan yang telah dipelajari, seperti fitur visual dari dataset besar seperti ImageNet.
- **Dua Pendekatan Utama**:
    - **Fine-Tuning**: Menyesuaikan seluruh model atau sebagian model pada dataset baru.
    - **Feature Extraction**: Menggunakan bagian dari model yang telah dilatih untuk mengekstrak fitur, dan hanya melatih layer tambahan yang ditambahkan.

2. Menggunakan Model Pra-Latih

- **Model Tersedia**: PyTorch menyediakan banyak model yang telah dilatih sebelumnya melalui **torchvision.models**, seperti ResNet, VGG, dan AlexNet.
- **Pemuatan Model**: Model dapat dimuat dengan atau tanpa bobot yang telah dilatih sebelumnya.

3. Adaptasi Model untuk Tugas Baru

- **Modifikasi Arsitektur**: Mengganti layer terakhir (biasanya fully connected layer) untuk menyesuaikan dengan jumlah kelas dalam dataset baru.
- **Parameter yang Dapat Dilatih**: Menentukan parameter mana yang akan dilatih ulang (fine-tuning) atau dibekukan.

4. Fine-Tuning Model

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Pelatihan Seluruh Model**: Dalam beberapa kasus, seluruh model dilatih ulang dengan learning rate yang sangat kecil untuk mengadaptasi bobot yang ada ke data baru.
- **Pelatihan Bagian Model**: Atau, hanya bagian tertentu dari model yang dilatih ulang, sementara sisanya dibekukan.

5. Feature Extraction

- **Layer Konvolusional sebagai Ekstraktor Fitur**: Dalam banyak arsitektur CNN, layer konvolusional awal mengekstrak fitur umum, yang dapat digunakan untuk tugas baru.
- **Penambahan Custom Classifier**: Layer klasifikasi baru yang sepenuhnya terhubung ditambahkan dan dilatih pada dataset target.

6. Pelatihan dan Evaluasi

- **Loop Pelatihan**: Setelah menyiapkan model, loop pelatihan standar PyTorch digunakan untuk melatih model pada dataset baru.
- **Evaluasi**: Model dievaluasi untuk memastikan bahwa ia generalisasi dengan baik pada data yang tidak terlihat.

7. Manajemen Data

- **DataLoader**: Penggunaan **DataLoader** PyTorch untuk memuat data dan label, dengan transformasi yang sesuai.
- **Augmentasi Data**: Terkadang augmentasi data digunakan untuk meningkatkan kinerja model pada dataset yang lebih kecil.

8. Regularisasi dan Optimasi

- **Penghindaran Overfitting**: Teknik seperti dropout, augmentasi data, atau regularisasi L2 dapat digunakan untuk menghindari overfitting, terutama ketika dataset target relatif kecil.
- **Optimizer**: Memilih optimizer yang tepat, seperti Adam atau SGD, dengan learning rate dan parameter lainnya yang disesuaikan untuk tugas.

9. Kasus Penggunaan

- **Visi Komputer**: Transfer learning sangat populer dalam tugas-tugas seperti klasifikasi gambar, deteksi objek, dan segmentasi.
- **NLP dan Domain Lain**: Meskipun umum dalam visi komputer, teknik ini juga dapat digunakan dalam NLP atau tugas pembelajaran mesin lainnya.

10. Best Practices dan Pertimbangan

- **Pemilihan Model Dasar**: Pilih model dasar yang relevan dengan tugas dan data.

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Fine-Tuning vs Feature Extraction**: Memutuskan antara fine-tuning seluruh model atau hanya menggunakan bagian dari model sebagai ekstraktor fitur.
- **Pengujian dengan Dataset Kecil**: Mulai dengan subset kecil dari data untuk cepat mengiterasi dan menemukan pengaturan yang efektif.

Transfer learning di PyTorch adalah metode yang efisien dan kuat yang memungkinkan pengembang dan peneliti untuk mengerjakan proyek pembelajaran mesin dengan lebih cepat dan dengan data yang lebih sedikit, sambil tetap mencapai hasil yang mengesankan.

07. PyTorch Experiment Tracking

Experiment tracking dalam konteks PyTorch adalah proses mencatat, membandingkan, dan menganalisis eksperimen yang dilakukan selama proses pembuatan dan pelatihan model pembelajaran mesin. Ini adalah aspek penting dari workflow pembelajaran mesin, terutama saat bekerja dengan banyak eksperimen yang berbeda atau dalam lingkungan kolaboratif. Berikut adalah penjelasan menyeluruh tentang experiment tracking di PyTorch:

1. Pentingnya Experiment Tracking

- **Reproduktibilitas**: Memungkinkan pengembang untuk mereproduksi hasil dan memahami perubahan yang memengaruhi kinerja model.
- **Analisis dan Perbandingan**: Memberikan kemampuan untuk menganalisis dan membandingkan berbagai eksperimen, termasuk arsitektur model, parameter, dan performa.

2. Parameter dan Metrik

- **Pelacakan Parameter**: Mencatat parameter seperti learning rate, batch size, dan arsitektur model.
- **Pencatatan Metrik**: Melacak metrik kinerja seperti loss, akurasi, precision, dan recall.

3. Logging dan Visualisasi

- **Tools untuk Logging**: Penggunaan alat seperti TensorBoard, MLflow, atau Weights & Biases untuk mencatat hasil eksperimen.
- **Visualisasi Data**: Memanfaatkan alat-alat ini untuk membuat visualisasi seperti grafik loss, akurasi, dan distribusi bobot.

4. Versi Model dan Data

- **Manajemen Versi Data**: Menggunakan alat seperti DVC (Data Version Control) untuk melacak versi dataset.
- **Manajemen Versi Model**: Menyimpan model di berbagai titik dalam proses pelatihan untuk membandingkan versi yang berbeda.

NAMA : DERY HIDAYAT
NIM : 1103228181

5. Hyperparameter Tuning

- **Pencatatan Eksperimen**: Melacak eksperimen dengan kombinasi hyperparameter yang berbeda.
- **Analisis Hasil**: Menggunakan experiment tracking untuk menganalisis mana kombinasi hyperparameter yang memberikan hasil terbaik.

6. Integrasi dengan PyTorch

- **Integrasi dengan DataLoader dan Model**: Mengintegrasikan alat tracking dengan loop pelatihan PyTorch.
- **Callbacks atau Hooks**: Implementasi callbacks atau hooks dalam PyTorch untuk otomatisasi logging.

7. Automasi dan Skalabilitas

- **Automasi Pelacakan**: Menyederhanakan proses pelacakan eksperimen dengan otomasi.
- **Skalabilitas**: Kemampuan untuk melacak eksperimen skala besar dengan banyak model atau set data.

8. Kolaborasi dan Berbagi

- **Berbagi Hasil**: Membagikan hasil eksperimen dengan anggota tim atau komunitas lebih luas.
- **Kolaborasi Tim**: Memungkinkan tim untuk kolaboratif bekerja pada eksperimen yang sama dan membandingkan hasil.

9. Penggunaan dalam Berbagai Skenario

- **Penelitian dan Pengembangan**: Dalam konteks R&D, experiment tracking membantu dalam menyusun publikasi atau berbagi temuan.
- **Produksi**: Memonitor performa dan perubahan pada model yang berjalan di lingkungan produksi.

10. Pertimbangan Keamanan dan Privasi

- **Keamanan Data**: Mengamankan data pelacakan, terutama jika termasuk data sensitif atau pribadi.
- **Kepatuhan terhadap Regulasi**: Memastikan bahwa proses pelacakan mematuhi regulasi seperti GDPR jika berlaku.

11. Best Practices

- **Konsistensi dan Standarisasi**: Menggunakan format dan metodologi yang konsisten untuk pelacakan.

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Dokumentasi**: Mendokumentasikan eksperimen dengan baik untuk referensi di masa depan.

Dengan experiment tracking, pengembang dan peneliti dapat secara sistematis mengelola proses pembelajaran mesin, membuatnya lebih terstruktur, efisien, dan transparan. Ini adalah bagian kunci dari siklus hidup pengembangan AI yang bertanggung jawab dan efektif.

08. PyTorch Paper Replicating

Replicating (atau mereplikasi) paper dalam konteks PyTorch mengacu pada proses mencoba menduplikasi hasil yang dilaporkan dalam publikasi ilmiah menggunakan PyTorch sebagai kerangka kerja pembelajaran mesin. Ini sering menjadi langkah penting dalam penelitian ilmiah, memungkinkan para peneliti untuk memvalidasi temuan, memahami secara mendalam teknik yang dijelaskan, dan menerapkan atau membangun atas penelitian tersebut. Berikut adalah penjelasan menyeluruh tentang proses replicating paper di PyTorch:

1. Pemahaman Paper

- **Membaca Secara Teliti**: Langkah pertama adalah membaca paper secara menyeluruh, memahami tujuan, metode, dan hasil yang dilaporkan.
- **Fokus pada Metodologi**: Khususnya memahami arsitektur model, dataset, proses pelatihan, dan metrik evaluasi yang digunakan.

2. Pengumpulan Data

- **Dataset**: Mendapatkan dataset yang digunakan dalam paper atau mencari alternatif yang serupa jika dataset asli tidak tersedia.
- **Preprocessing**: Mengimplementasikan proses preprocessing yang sesuai dengan yang dijelaskan dalam paper.

3. Membangun Model dengan PyTorch

- **Arsitektur Model**: Membangun arsitektur model yang sama dengan yang dijelaskan dalam paper menggunakan PyTorch.
- **Konsistensi**: Memastikan bahwa semua aspek model (seperti jumlah dan jenis layer, fungsi aktivasi) konsisten dengan deskripsi paper.

4. Pengaturan Eksperimen

- **Hyperparameter**: Menyetel hyperparameter sesuai dengan yang dilaporkan di paper, termasuk learning rate, ukuran batch, dan jumlah epoch.
- **Reproduksi Lingkungan Pelatihan**: Meniru lingkungan pelatihan seperti GPU dan versi PyTorch, jika memungkinkan.

5. Pelatihan dan Evaluasi

NAMA : DERY HIDAYAT
NIM : 1103228181

- **Pelatihan Model**: Melatih model menggunakan PyTorch, sering menggunakan teknik seperti early stopping dan validasi silang.
- **Evaluasi**: Menggunakan metrik yang sama dengan paper untuk mengevaluasi model, seperti akurasi, precision, recall, atau F1 score.

6. Debugging dan Penyesuaian

- **Pemeriksaan Perbedaan**: Jika hasil tidak cocok dengan paper, memeriksa dan men-debug setiap perbedaan dalam implementasi atau data.
- **A/B Testing**: Melakukan eksperimen dengan mengubah beberapa aspek untuk memahami dampaknya pada hasil.

7. Dokumentasi dan Pelaporan

- **Dokumentasi Proses**: Mencatat langkah-langkah yang diambil, masalah yang dihadapi, dan solusi yang ditemukan.
- **Pelaporan Hasil**: Menulis laporan atau jurnal yang mendokumentasikan proses dan temuan.

8. Berbagi Kode

- **Open Source**: Jika memungkinkan, membagikan kode sumber di platform seperti GitHub untuk memudahkan orang lain dalam mereplikasi atau membangun atas pekerjaan Anda.
- **Lisensi dan Kredit**: Memberikan kredit yang sesuai kepada penulis asli dan mengikuti pedoman lisensi yang relevan.

9. Kolaborasi dan Komunitas

- **Diskusi dengan Komunitas**: Berpartisipasi dalam forum atau komunitas ilmiah untuk mendiskusikan temuan dan memperoleh wawasan.
- **Kolaborasi**: Bekerja sama dengan peneliti lain bisa membantu dalam menangani masalah yang kompleks atau ambiguitas dalam paper.

10. Pengembangan Lanjutan

- **Ekstensi dan Modifikasi**: Setelah berhasil mereplikasi hasil, Anda mungkin ingin memodifikasi atau memperluas model atau eksperimen untuk eksplorasi lebih lanjut.
- **Kontribusi Kembali**: Memberikan kontribusi kembali ke komunitas dengan temuan atau peningkatan Anda sendiri.

Proses mereplikasi paper menggunakan PyTorch memerlukan keahlian teknis yang kuat, kemampuan analitis, dan perhatian terhadap detail. Ini bukan hanya soal mengkopi pekerjaan yang ada, tetapi juga memahami dan mungkin memperluas batas pengetahuan saat ini di bidang pembelajaran mesin dan kecerdasan buatan.

NAMA : DERY HIDAYAT
NIM : 1103228181

09. PyTorch Model Deployment

Deployment model di PyTorch melibatkan serangkaian langkah untuk mengintegrasikan model pembelajaran mesin yang telah dilatih ke dalam aplikasi produksi. Tujuannya adalah membuat model dapat digunakan oleh pengguna akhir atau sistem lain. Proses ini meliputi optimasi model, konversi ke format yang cocok, dan pengaturan infrastruktur untuk menjalankannya. Berikut adalah penjelasan menyeluruh tentang proses deployment model PyTorch:

1. Persiapan Model

- **Evaluasi dan Validasi**: Pastikan model telah dievaluasi secara menyeluruh dan validasi silang telah dilakukan untuk memastikan keandalannya.
- **Penyimpanan Model**: Model disimpan menggunakan **torch.save** untuk menyimpan state_dict atau model keseluruhan.

2. Optimasi Model

- **Quantization**: Mengurangi ukuran model dan meningkatkan kecepatan inferensi dengan mengubah tipe data, seperti dari float32 ke int8.
- **Pruning**: Mengurangi ukuran model dengan membuang bobot yang kurang penting.
- **Distillation**: Menggunakan model besar untuk melatih model yang lebih kecil dan lebih efisien.

3. Konversi Model

- **ONNX (Open Neural Network Exchange)**: Mengkonversi model PyTorch ke format ONNX untuk interoperabilitas lintas platform.
- **TorchScript**: Menggunakan TorchScript untuk membuat model yang dapat dijalankan secara independen dari kode Python asli, yang berguna untuk deployment di lingkungan non-Python.

4. Pemilihan Platform Deployment

- **Server Cloud**: Misalnya, menggunakan AWS, Google Cloud, atau Azure untuk hosting model.
- **Edge Devices**: Deployment di perangkat edge seperti ponsel atau IoT devices.
- **On-Premise Servers**: Untuk kontrol penuh atas data dan infrastruktur.

5. Containerization dan Microservices

- **Docker**: Menggunakan Docker untuk containerize model dan dependensinya, memudahkan deployment dan skalabilitas.
- **Microservices Architecture**: Membangun arsitektur berbasis layanan yang dapat dengan mudah diperbarui dan dikelola.

NAMA : DERY HIDAYAT
NIM : 1103228181

6. API dan Endpoint

- **REST API**: Membuat REST API menggunakan kerangka kerja seperti Flask atau FastAPI untuk memungkinkan akses ke model melalui HTTP.
- **gRPC**: Untuk komunikasi berkinerja tinggi antara server dan klien.

7. Load Balancing dan Skalabilitas

- **Manajemen Trafik**: Menggunakan load balancer untuk mendistribusikan permintaan secara efektif ke server-model.
- **Auto-Scaling**: Menyiapkan auto-scaling untuk mengelola beban kerja yang berfluktuasi.

8. Monitoring dan Logging

- **Tracking**: Memantau kinerja model di produksi dan mencatat permintaan dan tanggapan.
- **Alerts**: Mengatur alert untuk masalah performa atau downtime.

9. Pembaruan dan Manajemen Siklus Hidup

- **CI/CD Pipelines**: Mengintegrasikan model ke dalam pipeline Continuous Integration dan Continuous Deployment.
- **Iterasi Model**: Mengupdate dan mengiterasi model berdasarkan feedback dan data baru.

10. Keamanan dan Compliance

- **Enkripsi dan Keamanan Data**: Memastikan data diproses dan disimpan dengan aman.
- **Kepatuhan Regulasi**: Mematuhi standar industri dan regulasi, seperti GDPR.

11. Pertimbangan Khusus untuk Edge Deployment

- **Optimasi untuk Hardware Khusus**: Menyesuaikan model untuk keterbatasan sumber daya perangkat edge.
- **Inferensi On-Device**: Mengimplementasikan inferensi di perangkat untuk latensi rendah dan kemandirian.

12. Best Practices

- **Dokumentasi dan Pelatihan Pengguna**: Mendokumentasikan cara menggunakan endpoint model dan melatih pengguna atau pengembang yang akan berinteraksi dengan model.
- **Testing Thoroughly**: Melakukan pengujian ekstensif sebelum dan setelah deployment untuk meminimalkan masalah di produksi.

Deployment model PyTorch adalah proses yang kompleks dan memerlukan perhatian khusus pada aspek-aspek seperti performa, skalabilitas, keamanan, dan integrasi dengan sistem lain.

NAMA : DERY HIDAYAT
NIM : 1103228181

Pendekatan yang dipilih dapat bervariasi tergantung pada kebutuhan spesifik aplikasi dan infrastruktur yang tersedia.