Technical Report UAS Machine Learning

Deep Learning with PyTorch

Disusun Oleh :

Aldi Fauzan 1103204130

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

UNIVERSITAS TELKOM

2023

I. Pendahuluan

PyTorch adalah framework deep learning open-source berbasis Python yang kuat dan fleksibel. Dengan PyTorch, pengembang dapat dengan mudah membangun dan melatih jaringan saraf tiruan, mengoptimalkan model, dan melakukan visualisasi data. Framework ini memiliki keandalan dan kecepatan eksekusi yang tinggi, cocok untuk mengembangkan prototipe model deep learning.

Dalam tutorial ini, kita akan mempelajari dasar-dasar deep learning menggunakan PyTorch, termasuk konstruksi jaringan saraf tiruan, metode pelatihan, fungsi loss, dan optimisasi. PyTorch menyediakan antarmuka yang intuitif dan mudah dipahami untuk mendefinisikan arsitektur jaringan, menghitung gradien, dan memperbarui parameter model. Hal ini memungkinkan pengguna untuk fokus pada eksplorasi model dan pengembangan algoritma, dengan menyederhanakan proses implementasi.

Selain itu, kita juga akan melihat fitur-fitur PyTorch seperti pemrosesan data, transfer learning, dan fine-tuning. PyTorch menyediakan alat-alat yang kuat untuk mempersiapkan dan mengelola data pelatihan, termasuk transformasi data dan penggunaan data loader. Dengan menggunakan transfer learning, kita dapat memanfaatkan pengetahuan yang telah dipelajari oleh model sebelumnya untuk mempercepat pelatihan model baru atau mengatasi keterbatasan data. Ini merupakan strategi yang sangat berguna dalam pengembangan model deep learning yang efektif. Dengan memahami konsep-konsep ini, kita akan dapat mengoptimalkan dan memanfaatkan kekuatan PyTorch dalam mengembangkan model deep learning yang efektif untuk berbagai tugas.

II. Analyze the Code

a. Tensor Basic

Fungsi seperti torch.empty(), torch.rand(), dan torch.zeros() digunakan untuk membuat tensor dengan nilai kosong, acak, atau nol sesuai ukuran yang ditentukan. Fungsi seperti x.size() dan x.dtype digunakan untuk mendapatkan ukuran dan tipe data dari tensor. Fungsi torch.tensor() digunakan untuk membuat tensor dari list atau array, dan dapat diatur requires_grad=True untuk perhitungan gradien. Operasi matematika seperti penjumlahan, pengurangan, perkalian, dan pembagian dapat dilakukan antara tensor. Pengindeksan tensor dilakukan menggunakan notasi x[baris, kolom] atau x[:, indeks]. Pengubahan bentuk tensor dapat dilakukan dengan fungsi view(). Konversi antara tensor PyTorch dan array NumPy dapat dilakukan dengan numpy() dan torch.from_numpy(). Perangkat CUDA dan CPU dapat digunakan dengan fungsi to() dan torch.cuda.is_available(). Fungsi torch.ones_like() membuat tensor dengan elemen 1 sesuai ukuran tensor yang diberikan, dan to("cpu") memindahkan tensor ke CPU.

b. Autograd

Dalam kode yang diberikan, terdapat penggunaan fitur autograd dalam framework PyTorch. Autograd merupakan komponen penting dalam PyTorch yang memungkinkan perhitungan gradien otomatis pada tensor. Pada awalnya, tensor x didefinisikan dengan requires_grad=True, yang mengindikasikan bahwa kita ingin menghitung gradien terhadap tensor tersebut. Selanjutnya, dilakukan operasi pada tensor seperti penambahan dan perkalian untuk menghasilkan tensor baru y.

Informasi mengenai fungsi yang menghasilkan tensor tersebut dapat diperoleh melalui atribut grad_fn. Kemudian, dilakukan operasi kompleks pada tensor y, seperti perpangkatan dan perkalian dengan skalar. Selanjutnya, tensor z dihitung sebagai rata-rata tensor y. Dalam hal ini, autograd secara otomatis menghitung gradien tensor z terhadap tensor x menggunakan metode backward(). Gradien tersebut dapat diakses melalui atribut grad pada tensor x. Selain itu, fitur autograd juga memungkinkan penghitungan gradien pada komputasi yang lebih kompleks, seperti pada contoh penggunaan tensor y yang mengalami perkalian berulang. Selanjutnya, dilakukan demonstrasi penggunaan gradien dengan vektor gradien pada tensor x. Fungsi b.grad_fn menunjukkan fungsi yang menghasilkan tensor b, yang dihitung melalui operasi matematika pada tensor a. Selanjutnya, terdapat contoh untuk mengubah status requires_grad pada tensor a menggunakan metode requires_grad_(), serta penggunaan detach() untuk menghasilkan tensor b tanpa memerlukan perhitungan gradien. Terakhir, penggunaan torch.no_grad() digunakan untuk melaksanakan operasi tanpa perhitungan gradien pada tensor x. Contoh terakhir menggambarkan penggunaan autograd dalam konteks pelatihan model, di mana gradien dihitung dan digunakan untuk memperbarui bobot model.

c. Backpropagation

Dalam implementasi kode tersebut menggunakan PyTorch, dilakukan proses backpropagasi dalam suatu jaringan saraf tiruan. Tahapan dimulai dengan mendefinisikan tensor-tensor terkait, seperti tensor input (x), tensor output target (y), dan tensor parameter (w) yang akan dioptimalkan. Dengan mengatur requires_grad=True pada tensor parameter, memungkinkan perhitungan gradien terhadap parameter tersebut. Selanjutnya, dilakukan perhitungan prediksi (y_predicted) berdasarkan perkalian tensor parameter dengan tensor input. Kemudian, dilakukan perhitungan loss yang merupakan selisih kuadrat antara prediksi dan tensor output target. Dengan memanggil metode backward() pada tensor loss, gradien dari parameter yang memiliki requires_grad=True, dalam hal ini tensor parameter w, dapat dihitung. Gradien tersebut kemudian digunakan untuk memperbarui nilai parameter dengan menggunakan metode torch.no_grad(), yang memastikan perubahan nilai tidak mempengaruhi perhitungan gradien. Pada akhirnya, dilakukan pengaturan nilai gradien parameter menjadi nol untuk memulai perhitungan gradien pada batch atau episode selanjutnya. Keseluruhan proses ini mencerminkan langkah-langkah fundamental dalam backpropagasi, di mana gradien loss digunakan untuk mengoptimalkan nilai parameter agar model dapat menghasilkan prediksi yang lebih akurat terhadap target yang diinginkan.

Gradientdescent_manually

Diimplementasikan algoritma gradient descent secara manual untuk menyelesaikan masalah regresi sederhana. Digunakan library NumPy untuk manipulasi array. Pertama, array X didefinisikan sebagai input dan array Y sebagai target output dengan tipe data float32. Parameter w diinisialisasi dengan nilai awal 0.0. Fungsi forward(x) mengembalikan hasil perkalian parameter w dengan input x. Fungsi loss(y, y_pred) digunakan untuk menghitung mean squared error (MSE) antara target output y dan prediksi y_pred. Fungsi gradient(x, y, y_pred) menghitung gradien

MSE terhadap parameter w menggunakan rumus matematis yang sesuai. Dalam loop for, dilakukan iterasi sebanyak n_iters. Pada setiap iterasi, dilakukan perhitungan prediksi y_pred, loss l, dan gradien dw. Parameter w diperbarui dengan mengurangi hasil perkalian learning_rate dengan gradien dw. Selama pelatihan, dicetak nilai parameter w dan loss l pada setiap dua iterasi. Setelah selesai iterasi, dicetak prediksi akhir dengan memanggil forward(5).

d. Gradientdescent_auto

Digunakan PyTorch untuk mengimplementasikan algoritma gradient descent secara otomatis. Pertama, kita mendefinisikan tensor input (X) dan target output (Y) dengan tipe data float32. Parameter w diinisialisasi dengan nilai awal 0.0 dan requires_grad=True untuk mengaktifkan perhitungan gradien terhadap parameter tersebut. Fungsi forward(x) menghasilkan prediksi dengan mengalikan tensor w dengan input x. Fungsi loss(y, y_pred) digunakan untuk menghitung mean squared error (MSE) antara target output y dan prediksi y_pred. Selama pelatihan, dalam loop iterasi, kita menghitung prediksi y_pred, loss l, dan melakukan perhitungan gradien dengan memanggil metode backward() pada tensor loss. Parameter w diperbarui menggunakan nilai gradien yang dikalikan dengan learning_rate. Setelah selesai iterasi, dicetak prediksi akhir menggunakan fungsi forward(5). Dengan PyTorch, perhitungan gradien dan pembaruan parameter dilakukan secara otomatis, memudahkan implementasi algoritma gradient descent.

e. loss_and_optimizer

Kode menggunakan PyTorch untuk mengimplementasikan algoritma gradient descent dengan fungsi loss dan optimizer bawaan. Input dan target output didefinisikan sebagai tensor float32, dan parameter w diinisialisasi dengan requires_grad=True untuk menghitung gradien. Fungsi forward(x) menghasilkan prediksi berdasarkan perkalian tensor w dengan input x. Pada setiap iterasi, dilakukan perhitungan prediksi, loss, dan gradien menggunakan fungsi loss dan metode backward(). Optimizer SGD digunakan untuk mengupdate parameter dengan memperbarui nilai w menggunakan metode step(). Dalam loop iterasi, nilai parameter w dan loss dicetak setiap sepuluh iterasi. Setelah selesai iterasi, prediksi akhir dicetak menggunakan fungsi forward(5). Dengan menggunakan fungsi loss dan optimizer bawaan, implementasi algoritma gradient descent menjadi lebih sederhana dan efisien dalam mengoptimalkan parameter.

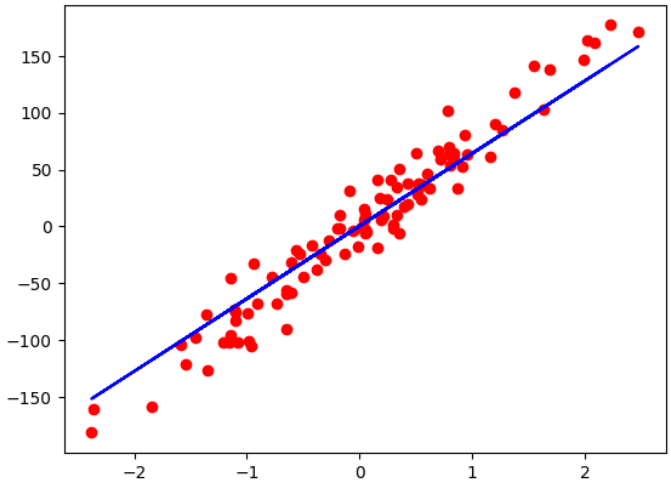
f. model_loss_and_optimizer

Kode pada model_loss_and_optimizer mengimplementasikan sebuah model linier menggunakan PyTorch. Model linier ini memiliki input size dan output size yang ditentukan oleh jumlah fitur dalam dataset. Pada awalnya, dilakukan pencetakan jumlah sampel dan fitur dalam dataset. Kemudian, sebuah tensor X_test dibuat untuk menguji model pada input 5. Model linier dibangun menggunakan kelas nn.Linear dengan ukuran input dan output yang sesuai. Prediksi sebelum pelatihan dicetak dengan memanggil model(X_test). Selanjutnya, ditentukan learning_rate dan jumlah iterasi. Fungsi loss yang digunakan adalah Mean Squared Error (MSE), dan optimizer yang digunakan adalah Stochastic Gradient Descent (SGD) dengan parameter model.parameters() dan learning rate yang ditentukan. Pada setiap iterasi,

dilakukan perhitungan prediksi y_predicted, perhitungan loss l, backward propagation, pembaruan parameter model menggunakan optimizer.step(), dan mengatur gradien menjadi nol menggunakan optimizer.zero_grad(). Selama pelatihan, dicetak nilai parameter w dan loss l pada setiap sepuluh iterasi. Setelah selesai iterasi, dicetak prediksi akhir setelah pelatihan dengan memanggil model(X_test).

g. linear_regression

Digunakan PyTorch untuk mengimplementasikan regresi linear. Dataset regresi linear dibuat menggunakan make_regression() dari sklearn.datasets, dan data tersebut dikonversi menjadi tensor PyTorch. Model regresi linear dengan ukuran input dan output yang sesuai didefinisikan menggunakan nn.Linear. Selama pelatihan, prediksi dilakukan dengan model, dan loss dihitung menggunakan MSE. Backward propagation digunakan untuk menghitung gradien, dan pembaruan parameter dilakukan menggunakan optimizer SGD. Selama pelatihan, nilai loss dicetak setiap 10 epoch. Setelah pelatihan, model digunakan untuk memprediksi data fitur, dan hasil prediksi divisualisasikan. Dengan PyTorch, implementasi regresi linear, perhitungan loss, optimisasi, dan visualisasi prediksi menjadi lebih mudah dan efisien.



h. logistic_regression

Dataset kanker payudara dimuat menggunakan `datasets.load_breast_cancer()` dan kemudian dibagi menjadi data pelatihan dan pengujian. Data tersebut kemudian diproses dengan normalisasi menggunakan `StandardScaler` dari `sklearn.preprocessing`. Model regresi logistik dengan satu layer linear didefinisikan menggunakan modul `nn.Linear`. Selama pelatihan, kita melakukan prediksi pada data pelatihan, menghitung loss menggunakan Binary Cross Entropy (BCE), melakukan optimisasi dengan Stochastic Gradient Descent (SGD), dan mencetak loss setiap 10 epoch. Setelah pelatihan, kita menggunakan model yang telah dilatih untuk melakukan prediksi pada data pengujian dan menghitung akurasi prediksi.

i. Dataloader

Digunakan `Dataset` dan `DataLoader` untuk mengatur dan memuat data dengan efisien dalam proses pelatihan model. Pertama, mendefinisikan kelas `WineDataset` sebagai turunan dari `Dataset`, di mana dataset wine dimuat dan dipisahkan menjadi

fitur dan label yang dijadikan tensor PyTorch. Kemudian, menggunakan `DataLoader`, kita memuat dataset dalam batch dengan menentukan ukuran batch, pengocokan data, dan jumlah pekerja. Dalam loop pelatihan, kita mengambil batch secara berurutan dari data pelatihan dan melooping melalui setiap batch untuk melakukan iterasi pada model. Kode juga memberikan contoh penggunaan dataset MNIST, di mana dataset tersebut diunduh dan dimuat menggunakan `DataLoader`. Dengan demikian, dengan bantuan `Dataset` dan `DataLoader`, kita dapat dengan mudah mengatur dan memuat data dalam batch yang diperlukan untuk pelatihan model.

j. Transformer

Digunakan torch.utils.data.Dataset untuk mendefinisikan kelas WineDataset yang memuat dataset wine. Dataset ini dapat dilengkapi dengan transformasi data yang dapat didefinisikan dengan menggunakan kelas transformasi seperti ToTensor dan MulTransform. Transformasi ToTensor mengubah data dari numpy array menjadi tensor PyTorch, sedangkan transformasi MulTransform mengalikan faktor pada fitur data. Dengan adanya transformasi, kita dapat dengan mudah mengubah atau memodifikasi format data sebelum digunakan dalam proses pelatihan model. Hal ini mempermudah pengolahan data dan memastikan data siap digunakan sesuai dengan kebutuhan model yang akan dilatih.

k. softmax_and_crossentropy

Diimplementasikan fungsi softmax dan cross-entropy, serta menggunakan kelas-kelas terkait dalam numpy dan PyTorch. Fungsi softmax digunakan untuk mengubah output menjadi distribusi probabilitas, yang berguna dalam masalah klasifikasi multi-kelas. Fungsi cross-entropy digunakan untuk mengukur perbedaan antara distribusi probabilitas prediksi dan distribusi probabilitas aktual, sehingga memberikan informasi tentang seberapa baik model mengeneralisasi data. Dalam PyTorch, kita menggunakan kelas nn.CrossEntropyLoss yang secara efisien menggabungkan softmax dan cross-entropy. Selain itu, kita juga mendefinisikan dua model neural network yang berbeda menggunakan nn.Module, dan menggunakan kriteria loss yang sesuai dengan jenis masalah yang dihadapi. Fungsi-fungsi tersebut digunakan untuk melatih dan mengukur performa model dalam tugas klasifikasi dengan menggunakan softmax dan cross-entropy sebagai alat evaluasi.

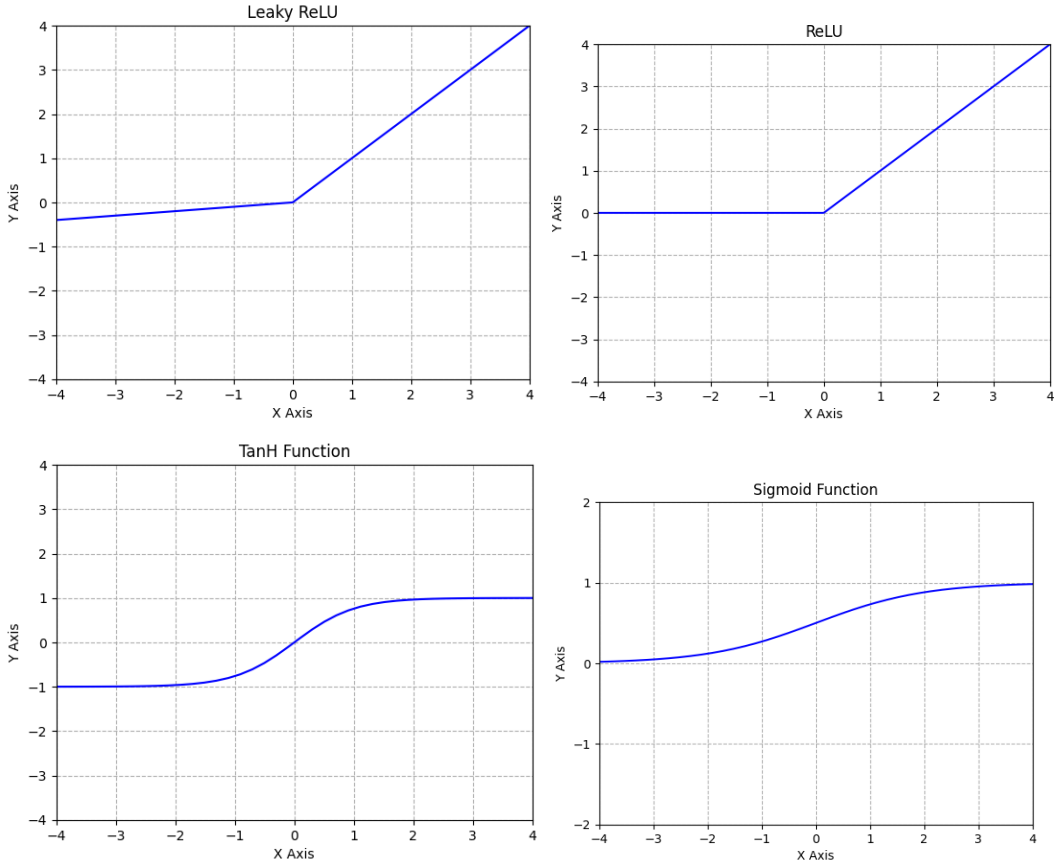
l. activation_functions

Fungsi-fungsi aktivasi seperti softmax, sigmoid, tangen hiperbolik (tanh), ReLU, dan Leaky ReLU digunakan untuk memperkenalkan non-linearitas pada model neural network. Fungsi-fungsi tersebut mengubah output dari suatu layer menjadi bentuk yang lebih berguna, seperti distribusi probabilitas, nilai antara 0 dan 1, atau rentang nilai yang lebih luas. Kemudian, dalam kelas NeuralNet, fungsi-fungsi aktivasi tersebut digunakan untuk menghitung output dari setiap lapisan model neural network, memperkenalkan kemampuan model untuk mempelajari hubungan non-linear antara fitur-input dan output yang diinginkan.

m. plot_activations

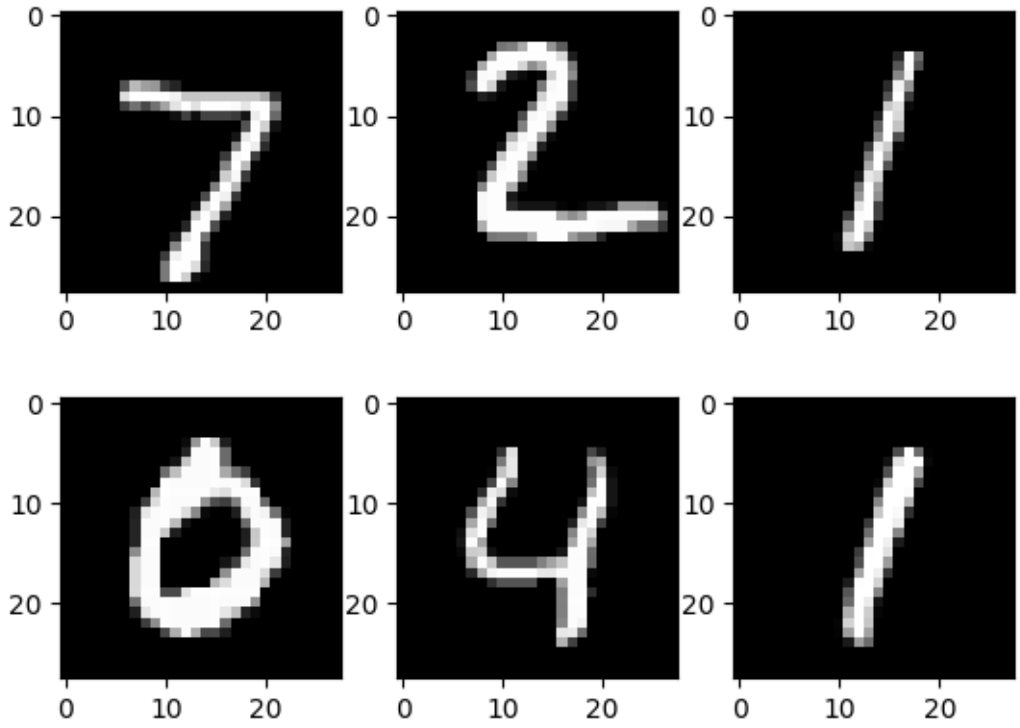
Digunakan matplotlib untuk menghasilkan visualisasi grafik dari beberapa fungsi aktivasi umum seperti sigmoid, tanh, ReLU, Leaky ReLU, dan fungsi langkah (step

function). Setiap grafik menggambarkan hubungan antara input dan output dari masing-masing fungsi aktivasi, memberikan pemahaman visual tentang bagaimana fungsi-fungsi tersebut mengubah nilai input menjadi output dalam model neural network. Melalui grafik ini, kita dapat melihat karakteristik masing-masing fungsi, seperti sigmoid yang menghasilkan output antara 0 dan 1, tanh yang menghasilkan output antara -1 dan 1, ReLU yang mengizinkan output positif tanpa batas atas, dan Leaky ReLU yang mengizinkan output negatif dengan kemiringan kecil. Visualisasi ini membantu kita memahami bagaimana perubahan nilai input mempengaruhi output dan mengapa fungsi-fungsi ini digunakan dalam jaringan saraf untuk memperkenalkan non-linearitas dan meningkatkan kemampuan model dalam mempelajari pola yang kompleks.



n. feedforward

Kode tersebut merupakan implementasi dari feedforward neural network menggunakan PyTorch untuk melakukan klasifikasi pada dataset MNIST. Pertama, dataset MNIST diunduh dan dibagi menjadi data pelatihan dan data pengujian. Selanjutnya, model neural network dengan lapisan input, lapisan tersembunyi, dan lapisan output didefinisikan. Fungsi forward pada model menghitung output berdasarkan input, dan fungsi backward menghitung gradien loss untuk mengoptimalkan parameter model menggunakan algoritma Adam. Selama pelatihan, model diterapkan pada data pelatihan dalam batch, loss dihitung, dan gradien digunakan untuk memperbarui parameter. Setelah pelatihan selesai, akurasi model diukur pada data pengujian untuk mengevaluasi performanya.

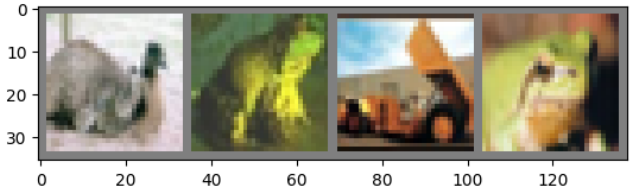


Epoch [1/2], Step [100/600], Loss: 0.2948
Epoch [1/2], Step [200/600], Loss: 0.1802
Epoch [1/2], Step [300/600], Loss: 0.1638
Epoch [1/2], Step [400/600], Loss: 0.1239
Epoch [1/2], Step [500/600], Loss: 0.1582
Epoch [1/2], Step [600/600], Loss: 0.2397
Epoch [2/2], Step [100/600], Loss: 0.1778
Epoch [2/2], Step [200/600], Loss: 0.0591
Epoch [2/2], Step [300/600], Loss: 0.1062
Epoch [2/2], Step [400/600], Loss: 0.1064
Epoch [2/2], Step [500/600], Loss: 0.0699
Epoch [2/2], Step [600/600], Loss: 0.0868
Accuracy of the network on the 10000 test images: 97.03 %

o. cnn

Diimplementasikan Convolutional Neural Network (CNN) menggunakan PyTorch untuk melakukan klasifikasi pada dataset CIFAR-10. Pertama, dataset CIFAR-10 diunduh dan dibagi menjadi data pelatihan dan data pengujian. Kemudian, model CNN dengan lapisan konvolusi, lapisan pengelompokan, dan lapisan linier (fully connected) didefinisikan. Fungsi forward pada model melakukan operasi konvolusi, pengelompokan, dan aktivasi ReLU. Setelah itu, data pelatihan diterapkan pada model dalam batch, loss dihitung, dan gradien digunakan untuk memperbarui parameter model menggunakan algoritma SGD. Setelah pelatihan selesai, akurasi model diukur pada data pengujian secara keseluruhan dan untuk setiap kelas. Model juga disimpan ke dalam file "cnn.pth" untuk penggunaan selanjutnya. Model CNN yang dilatih menggunakan dataset CIFAR-10 mencapai akurasi sebesar 47.35%. Klasifikasi yang paling baik terjadi pada kelas "horse" dengan akurasi 70.5%,

sedangkan kelas "cat" memiliki akurasi yang rendah sebesar 12.8%. Dalam pengembangan model selanjutnya, dapat dilakukan penyesuaian seperti peningkatan jumlah epoch, penyesuaian learning rate, atau penggunaan arsitektur yang lebih kompleks untuk meningkatkan performa klasifikasi.



p. transfer_learning

Dimplementasikan transfer learning menggunakan model ResNet-18 pada dataset Hymenoptera. Pada tahap pertama, model ResNet-18 pretrained digunakan dan layer fully connected (fc) diganti dengan linear layer sesuai dengan jumlah kelas pada dataset. Model tersebut dilatih menggunakan fungsi `train_model` dengan menggunakan criterion CrossEntropyLoss dan optimizer SGD. Pada tahap kedua, model ResNet-18 pretrained digunakan, namun kali ini semua parameter kecuali layer fc di-freeze. Model tersebut juga dilatih dengan menggunakan fungsi `train_model` dengan criterion dan optimizer yang sama. Transfer learning memungkinkan untuk memanfaatkan pengetahuan yang telah dipelajari oleh model pada tugas lain untuk meningkatkan kinerja & efisiensi pelatihan pada tugas berbeda.

q. Tensorboard

Digunakan TensorBoard untuk memantau dan memvisualisasikan pelatihan dan evaluasi model jaringan saraf pada dataset MNIST. Fungsi-fungsi yang digunakan meliputi SummaryWriter untuk membuat objek penulis TensorBoard, add_image untuk menampilkan gambar-gambar dataset ke TensorBoard, add_graph untuk menambahkan representasi grafik arsitektur model, add_scalar untuk mencatat skalar seperti loss dan akurasi pelatihan, serta add_pr_curve untuk menampilkan kurva presisi-recall pada TensorBoard. Dengan menggunakan TensorBoard, informasi dan visualisasi kinerja model dapat disimpan dan dianalisis secara interaktif, memudahkan pemahaman tentang bagaimana model bekerja.

r. Saveload

Penjelasan tentang penggunaan fungsi-fungsi terkait penyimpanan dan pengambilan model menggunakan PyTorch. Pertama, definisi kelas Model dan fungsi forward untuk model jaringan saraf sederhana. Kemudian, model tersebut disimpan ke dalam file menggunakan torch.save. Model yang disimpan kemudian dapat dimuat kembali menggunakan torch.load. Terdapat juga contoh penggunaan state_dict() untuk menyimpan dan memuat parameter model secara terpisah. Fungsi optimasi juga disimpan dan dimuat bersama dengan model. Contoh terakhir menjelaskan penggunaan untuk menyimpan dan memuat model di antara perangkat CPU dan GPU.