Topik : 2.4. Modifikasi Model

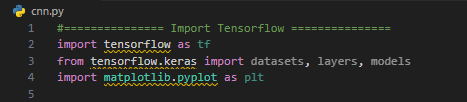
Objective : Ganti model dengan CNN custom sederhana

Task : Bandingkan performanya dengan model default

Source : <https://www.tensorflow.org/tutorials/images/cnn>’

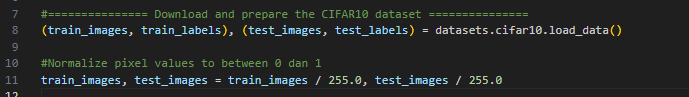
**Tensorflow CNN model**

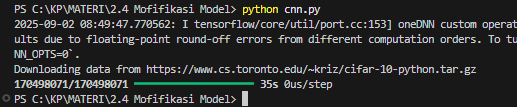
1. Import Tensorflow



1. Download and prepare the CIFAR10 Dataset

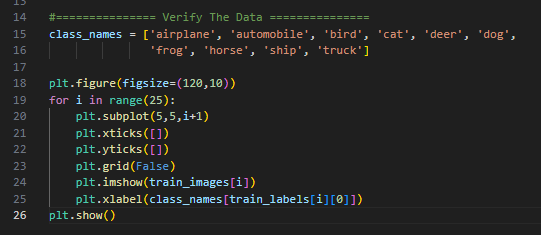
Kumpulan data CIFAR10 berisi 60.000 gambar berwarna dalam 10 kelas, dengan 6.000 gambar dalam setiap kelas. Kumpulan data ini dibagi menjadi 50.000 gambar pelatihan dan 10.000 gambar pengujian.

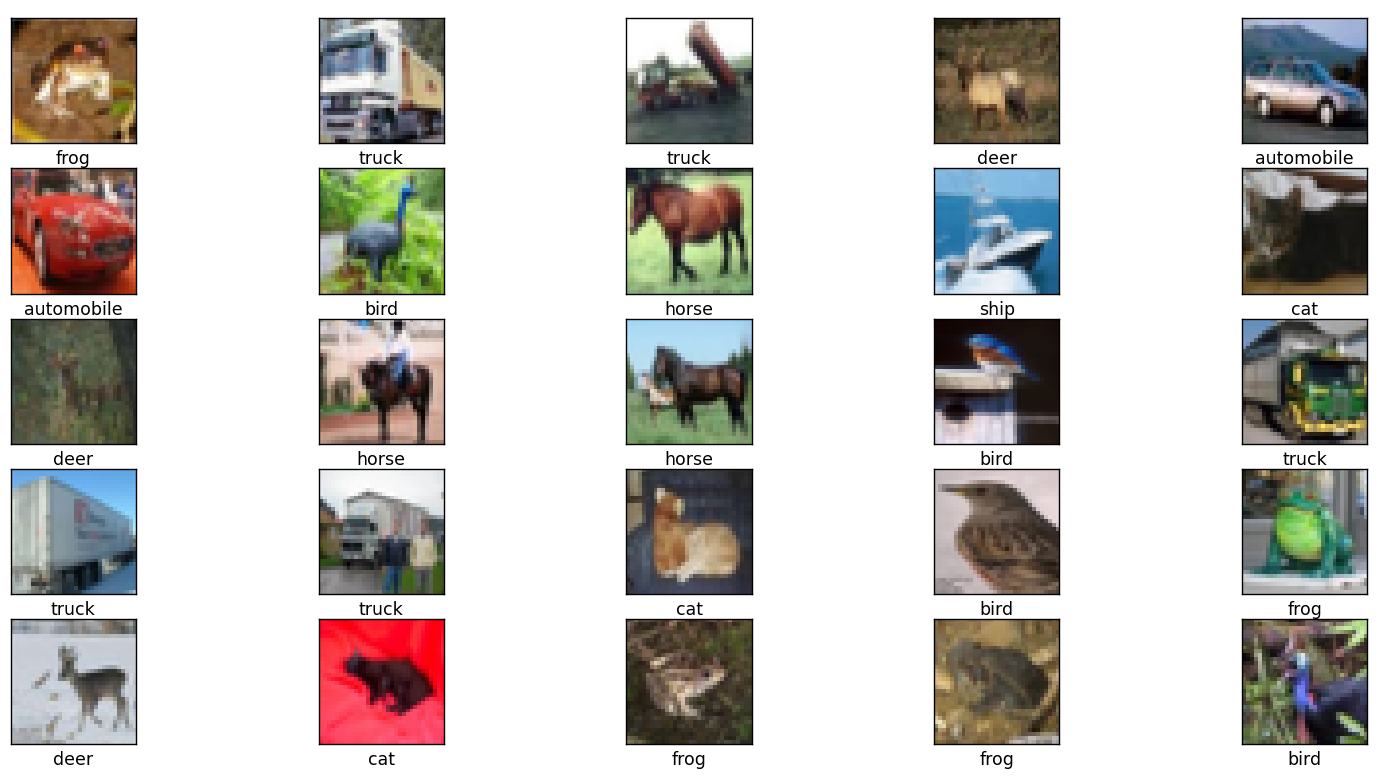




1. Verify the Data

Untuk memverifikasi bahwa dataset terlihat benar, mari kita plot 25 gambar pertama dari set pelatihan dan tampilkan nama kelas dibawah setiap gambar.

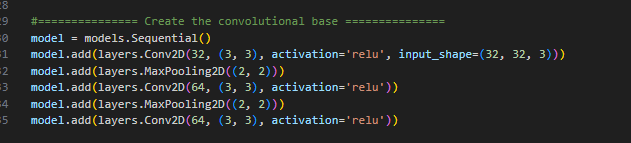


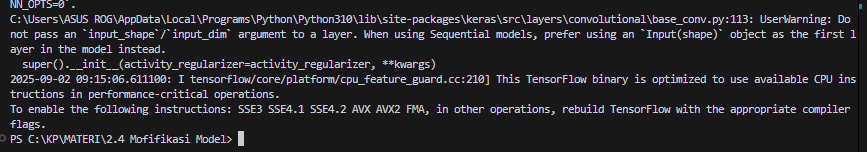


1. Create the convolutional base

Enam baris kode dibawah ini mendefinisikan basis konvolusi menggunakan pola umum: tumpukan lapisan Conv2D dan MaxPooling2D.

CNN menerima tensor dengan bentuk (image\_height, image\_width, color\_channels), dan mengabaikan ukuran batch serta saluran warna merujuk kepada (R,G,B). dalam contoh ini, anda akan mengonfigurasi CNN untuk memproses input dengan bentuk (32,32,3), yang merupakan format gambar CIFAR. Kita dapat melakukannya dengan memberikan argument input\_shape ke lapisan pertama.





Mari kita tampilkan arsitektur model Anda sejauh ini:

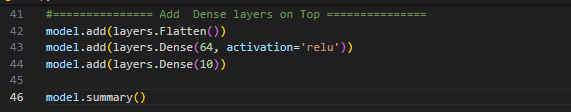


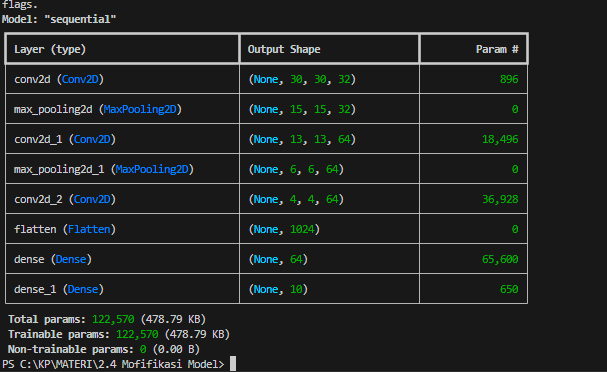


Output dari setiap lapisan Conv2D dan MaxPooling2D Adalah tensor 3D dengan bentuk (tinggi, lebar, saluran). Dimensi lebar dan tinggi cenderung menyusut seiring anda masuk lebih dalam ke dalam jaringan. Jumlah saluran keluaran untuk setiap lapisan Conv2D dikendalikan oleh argument pertama (misalnua, 32 atau 64). Secara umum, seiring lebar dan tinggi menyusut, anda dapat secara komputasi menambahkan lebih banyak saluran keluaran di setiap lapisan Conv2D.

1. Add Dense Layers on Top

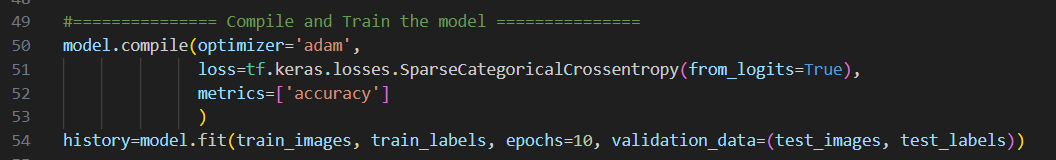
Untuk menyelesaikan model, maka kita akan memasukkan tensor output terakhir dari lapisan konvolusi dasar yang berbentuk (4, 4, 64) ke dalam satu atau lebih lapisan Dense untuk melakukan klasifikasi. Lapisan Dense menerima vector sebagai masukan ( yang berdimensi 1D), sementara output saat ini adalah tensor 3D, maka kita akan meratakan (atau menggulung) output 3D menjadi 1D, lalu menambahkan satu atau lebih lapisan Dense diatasnya. CIFAR memiliki 10 kelas output, jadi kita menggunakan lapisan Dense akhir dengan 10 output.

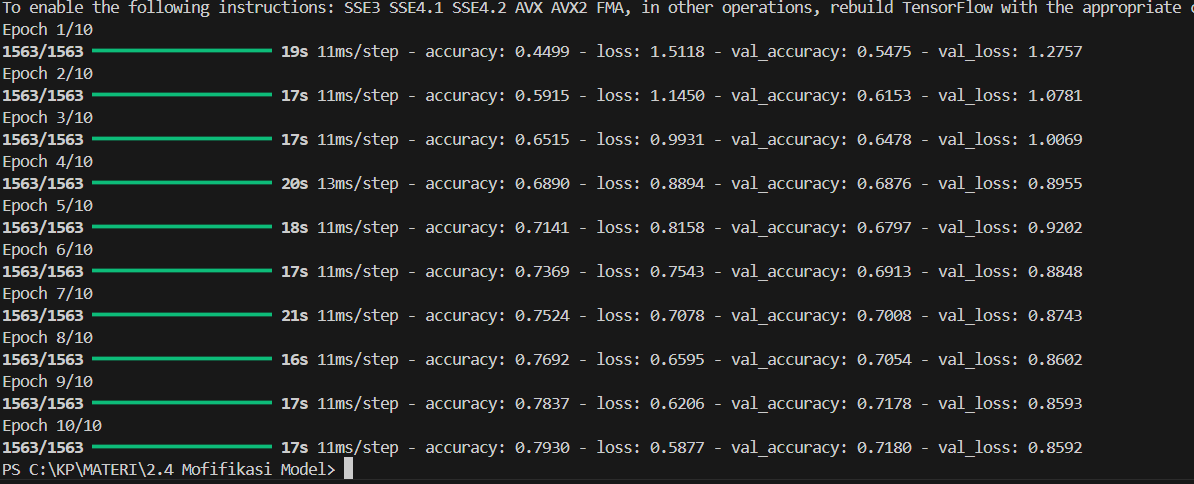




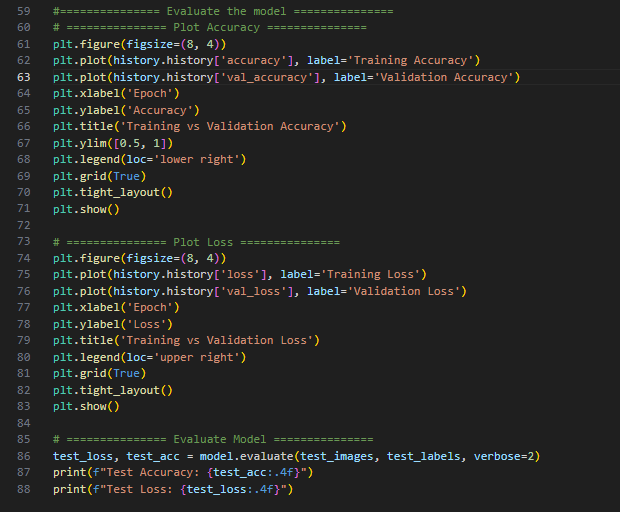
Ringkasan jaringan menunjukkan bahwa keluaran (4,4,64) diubah menjadi vector dengan bentuk (1024) sebelum melewati dua lapisan Dense.

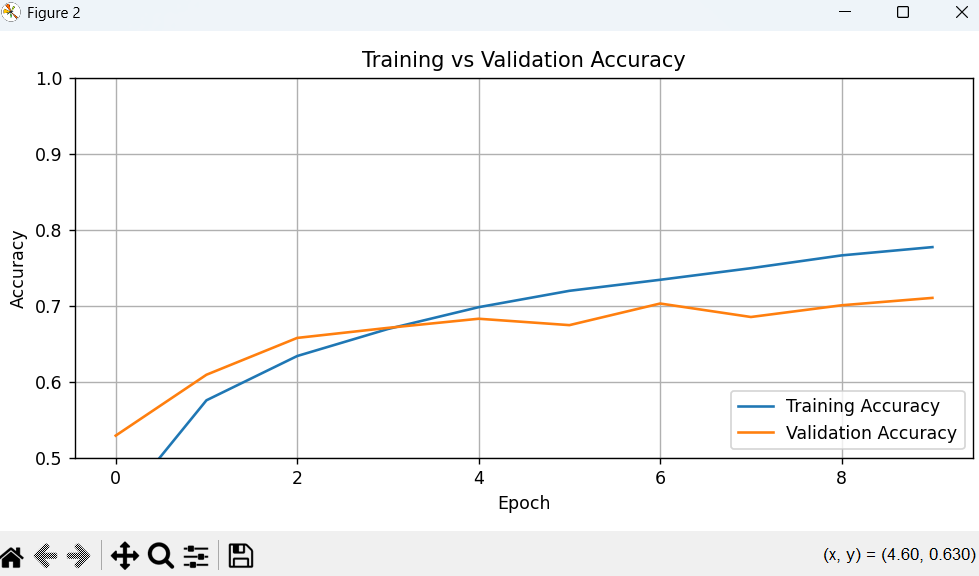
1. Compile and train the model



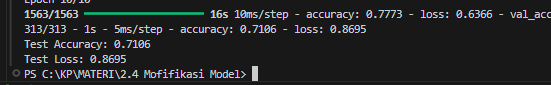


1. Evaluate the model







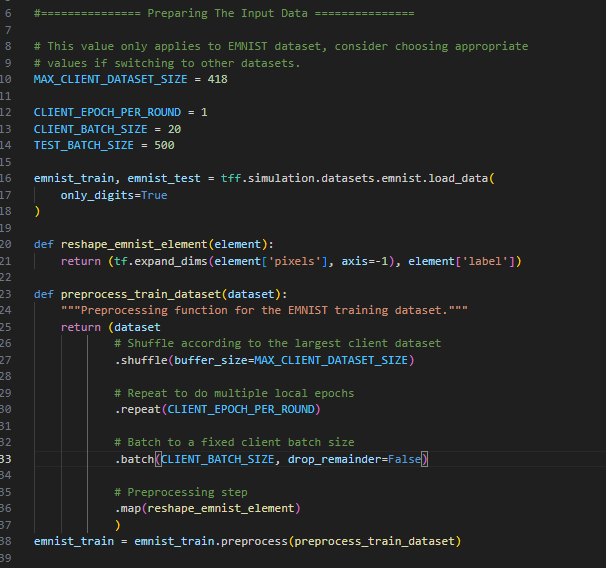


Source : <https://www.tensorflow.org/federated/tutorials/tff_for_federated_learning_research_compression>

**TFF Model Wrapper**

1. Preparing the input data

Memuat dan melakukan prapemrosesan dataset EMNIST yang termasuk dalam TFF.



1. Defining a model

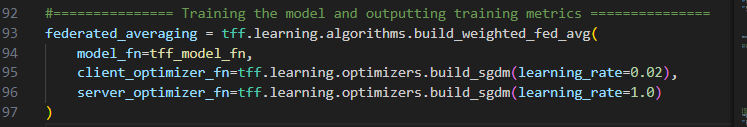
Mendefinisikan model keras berdasarkan FedAvg CNN asli, lalu membungkus model keras tersebut dalam instance tff.learning.models.VariableModel agar dapat digunakan oleh TFF.

Kita memerlukan fungsi yang menghasilkan model, bukan sekadar model secara langsung. Selain itu, fungsi tersebut tidak boleh hanya menangkap model yang sudah dibangun sebelumnya, melainkan harus membangun model dalam konteks di mana fungsi tersebut dipanggil. Hal ini karena TFF dirancang untuk dijalankan di perangkat, dan memerlukan kontrol atas waktu pembentukan sumber daya agar dapat ditangkap dan dikemas.



1. Training the model and outputting training metrics

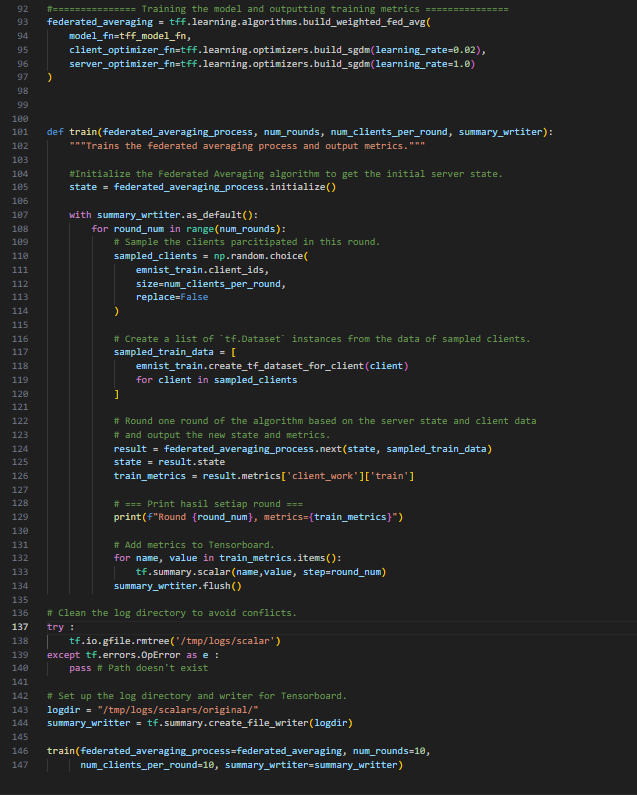
Pertama, kita perlu membangun algoritma Federated Averaging menggunakan API tff.learning.algorithms.build\_weighted\_fed\_avg.

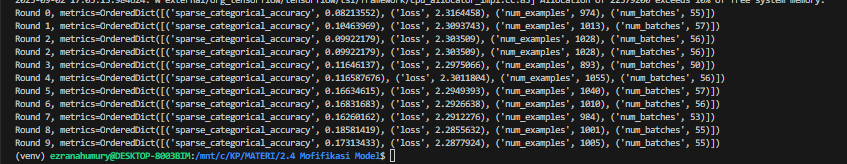


Menjalankan algoritma Federated Averaging. Pelaksanaan algoritma Federated Averaging Learning dari perspektif TFF terlihat seperti ini :

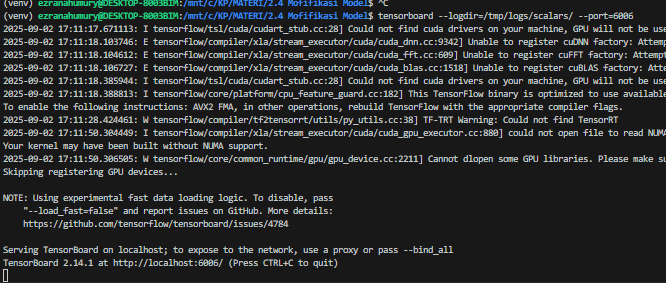
* Inisialisasi algoritma yang didapatkan keaadaan server awal. eadaan server berisi informasi yang diperlukan untuk menjalankan algoritma. Ingat, karena TFF bersifat fungsional, keadaan ini mencakup baik keadaan optimizer yang digunakan algoritma (misalnya istilah momentum) maupun parameter model itu sendiri—ini akan diteruskan sebagai argumen dan dikembalikan sebagai hasil dari perhitungan TFF.
* Jalankan algoritma secara bertahap. Pada setiap tahap, keadaan server baru akan dikembalikan sebagai hasil dari setiap klien yang melatih model pada datanya. Biasanya pada satu tahap:
  + Server menyiarkan model ke semua klien yang berpartisipasi.
  + Setiap klien melakukan pekerjaan berdasarkan model dan datanya sendiri.
  + Server menggabungkan semua model untuk menghasilkan keadaan server yang berisi model baru.

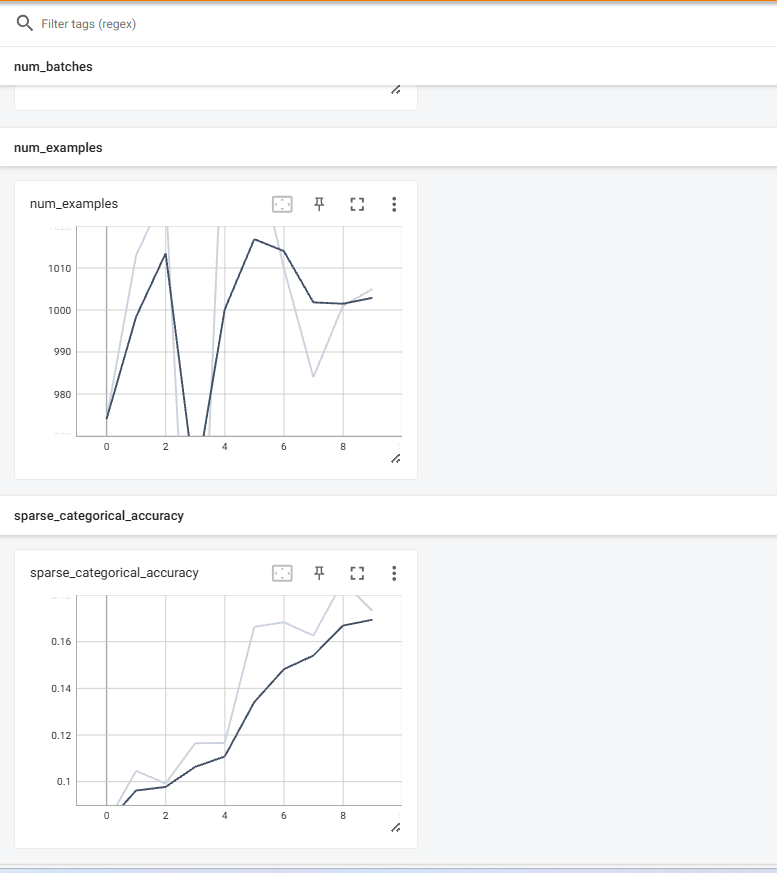
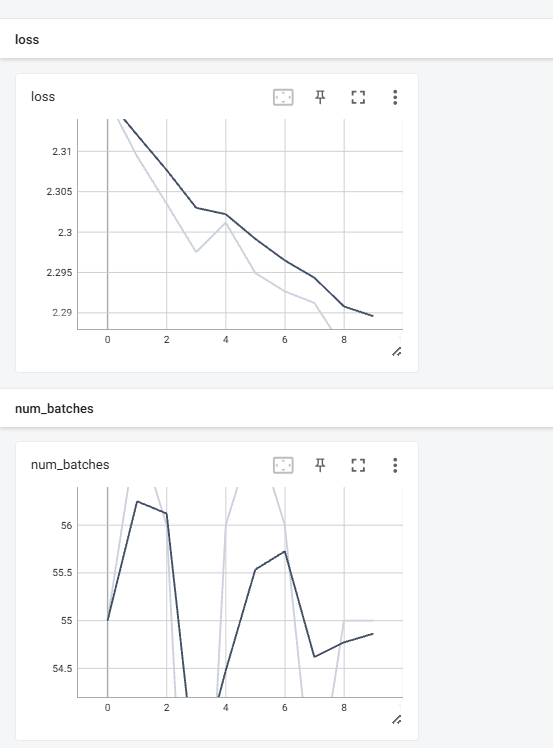
Metrik pelatihan ditulis ke direktori Tensorboard untuk ditampilkan setelah proses pelatihan selesai.





Jalankan TensorBoard dengan direktori log akar yang ditentukan di atas untuk menampilkan metrik pelatihan. Proses pemuatan data mungkin memakan waktu beberapa detik. Selain Loss dan Accuracy, kami juga menampilkan jumlah data yang dikirimkan dan dikumpulkan. Data yang dikirimkan merujuk pada tensor yang dikirimkan server ke setiap klien, sedangkan data yang dikumpulkan merujuk pada tensor yang dikembalikan setiap klien ke server.

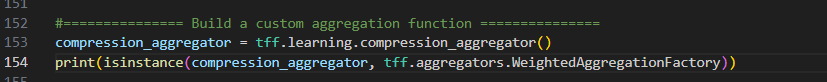




1. Build a custom aggregation function

Mengimplementasikan fungsi untuk menggunakan algoritma kompresi lossy pada data yang telah di agregasi. Kita akan menggunakan API TFF untuk membuat tff.aggregators.AggregationFactory untuk tujuan ini. Kita akan menggunakan metode bawaan untuk melakukannya, yaitu tff.learning.compression\_aggregator.

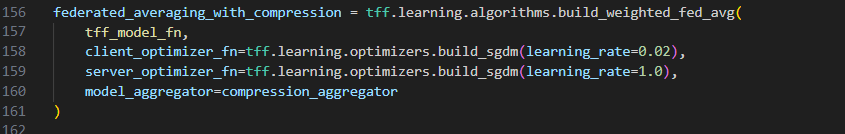
Agregator ini tidak menerapkan kompresi pada seluruh model sekaligus. Sebaliknya, kompresi hanya diterapkan pada variabel-variabel dalam model yang cukup besar. Secara umum, variabel-variabel kecil seperti bias lebih sensitif terhadap ketidakakuratan, dan karena ukurannya relatif kecil, potensi penghematan komunikasi juga relatif kecil.





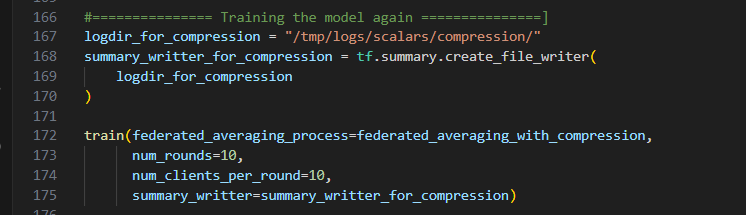
Di atas, Anda dapat melihat bahwa kompresor agregator adalah pabrik agregasi berberat, yang berarti melibatkan agregasi berberat (berbeda dengan agregator yang dirancang untuk privasi diferensial, yang seringkali tidak berberat).

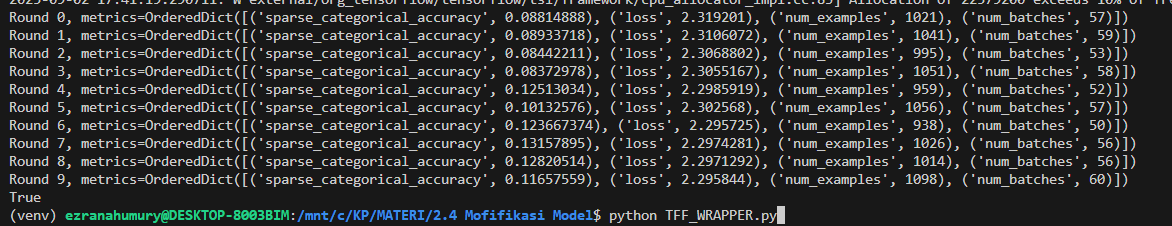
Pabrik agregasi ini dapat langsung diintegrasikan ke FedAvg melalui argumen model\_aggregator-nya.



1. Training the model again

Sekarang mari kita jalankan algoritma Federated Averaging yang baru.

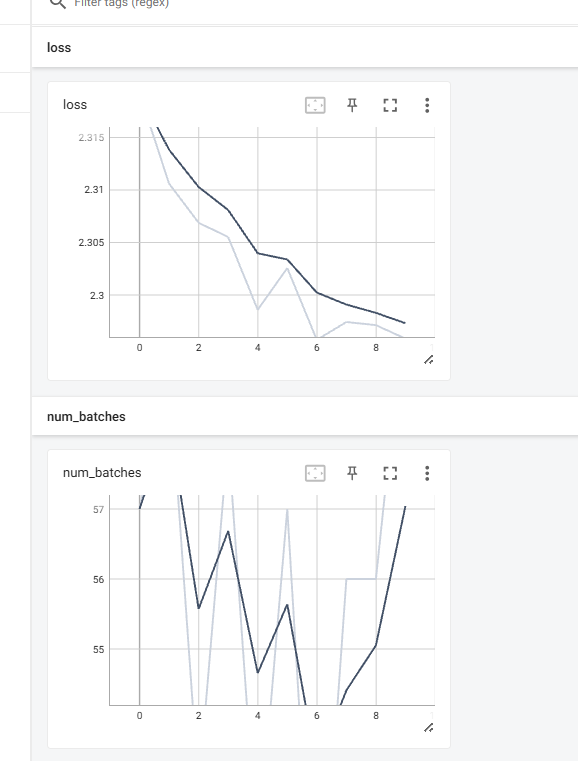


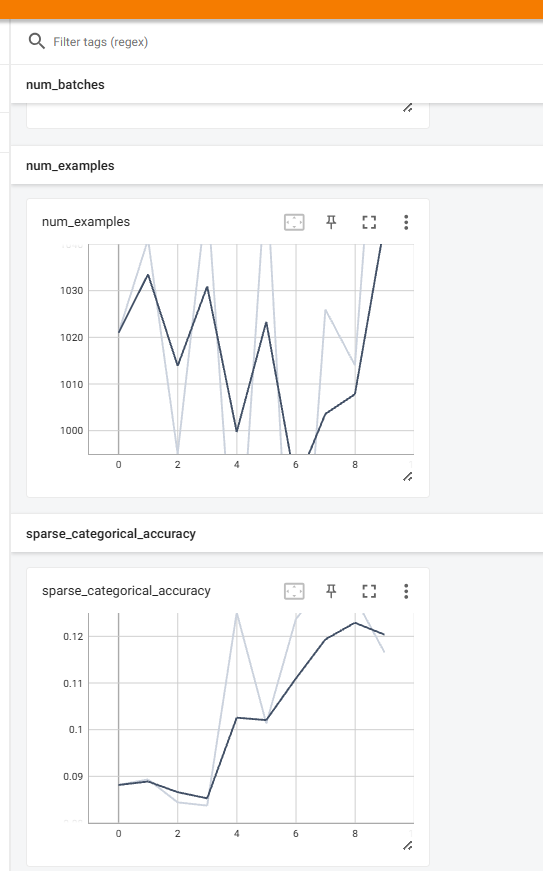


Jalankan TensorBoard kembali untuk membandingkan metrik pelatihan antara dua kali eksekusi.

Seperti yang dapat Anda lihat di TensorBoard, terdapat penurunan yang signifikan antara kurva asli dan kurva kompresi pada grafik aggregated\_bits, sementara pada grafik loss dan sparse\_categorical\_accuracy, kedua kurva tersebut cukup mirip.

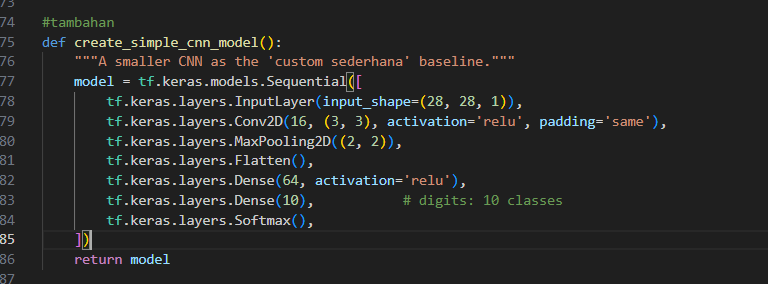
Kesimpulannya, kami telah mengimplementasikan algoritma kompresi yang dapat mencapai kinerja serupa dengan algoritma Federated Averaging asli, sementara biaya komunikasi berkurang secara signifikan.

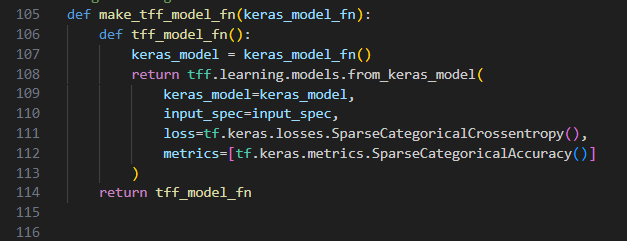


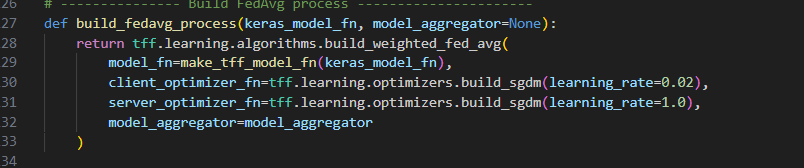


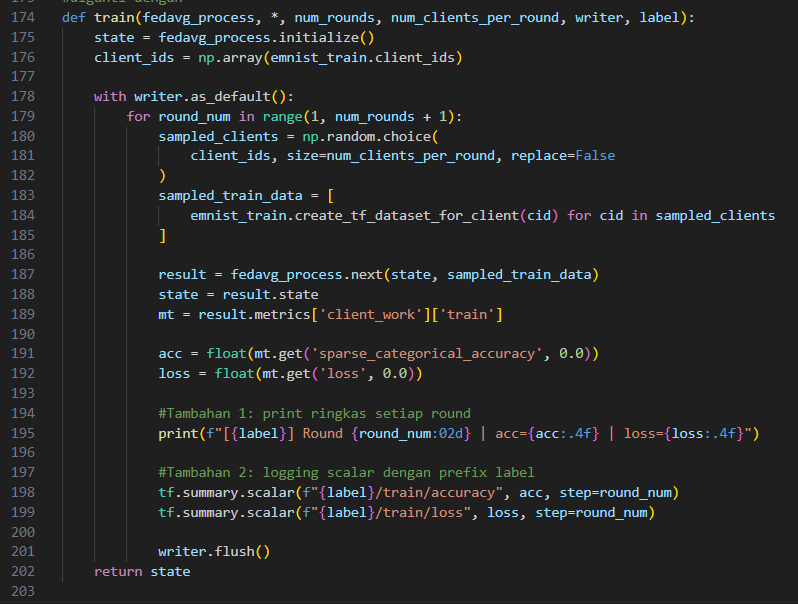
Task : Custom Sederhana

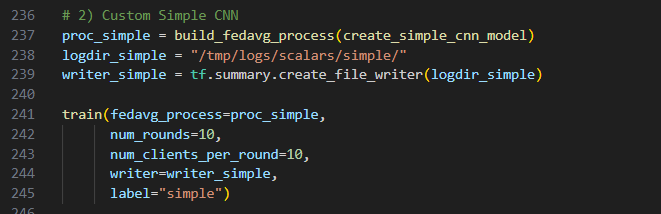
Dari model TFF WRAPPER diatas , saya melakukan beberapa custom diantaranya :

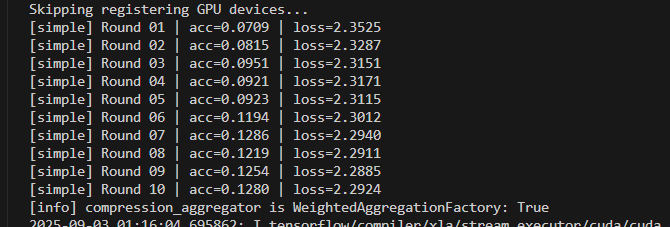












Perbandingan :

|  |  |
| --- | --- |
| Model Default | Model Custom |
| * **Round 0 → Round 9** * Akurasi awal: **0.0881 (8.8%)** * Akurasi akhir: **0.1167 (11.6%)** * Loss menurun sedikit: **2.3190 → 2.2958** | * **Round 1 → Round 10** * Akurasi awal: **0.0709 (7.0%)** * Akurasi akhir: **0.1280 (12.8%)** * Loss menurun lebih konsisten: **2.3525 → 2.2924** |

Analisis :

1. Akurasi

* Model custom lebih baik: akhir di 12.8% vs default 11.6%.
* Walaupun gap-nya kecil, tren custom naik lebih stabil per round.

1. Loss

* Kedua model menunjukkan penurunan loss dengan kecepatan hampir sama.
* Custom sedikit lebih rendah di akhir (2.2924 vs 2.2958).

1. Stabilitas Training

* Custom model menghasilkan output yang lebih konsisten per ronde.
* Default model terlihat agak fluktuatif (naik-turun tipis).