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

A. Import Tensorflow

cnn.py

```
#=============== Import Tensorflow ===============
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

```

B. 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.

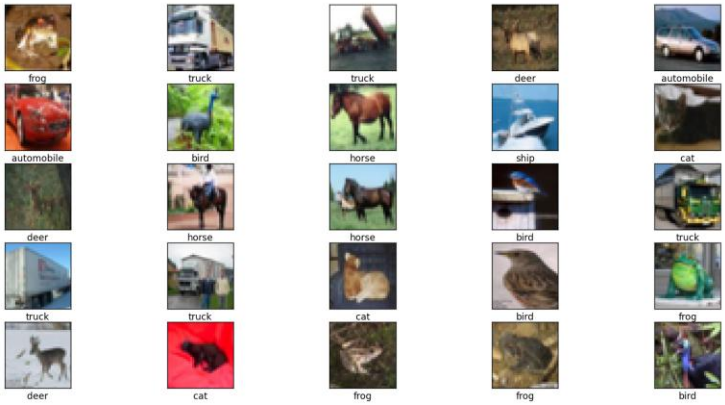
```
#=============== Download and prepare the CIFAR10 dataset ===============
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

#Normalize pixel values to between 0 dan 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
PS C:\KP\MATERI\2.4 Mofifikasi Model> python cnn.py
2025-09-02 08:49:47.770562: I tensorflow/core/util/port.cc:153] oneDNN custom operat
ults due to floating-point round-off errors from different computation orders. To tu
NN_OPTS=0`.
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 35s 0us/step
PS C:\KP\MATERI\2.4 Mofifikasi Model>
```

C. 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.

```
#=============== Verify The Data ===============
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
               'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(120,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



D. 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.

```
8
9   #=============== Create the convolutional base ===============
0   model = models.Sequential()
1   model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
2   model.add(layers.MaxPooling2D((2, 2)))
3   model.add(layers.Conv2D(64, (3, 3), activation='relu'))
4   model.add(layers.MaxPooling2D((2, 2)))
5   model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
NN_OPTS=0'.
C:\Users\ASUS ROG\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\convolutional\base_conv.py:113: UserWarning: Do
not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first l
ayer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-09-02 09:15:06.611100: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU ins
tructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler
flags.
PS C:\KP\MATERI\2.4 Mofifikasi Model>
```

Mari kita tampilkan arsitektur model Anda sejauh ini:

```
37  model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928

```
Total params: 56,320 (220.00 KB)
Trainable params: 56,320 (220.00 KB)
Non-trainable params: 0 (0.00 B)
PS C:\KP\MATERI\2.4 Mofifikasi Model>
```

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.

E. 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.

```
41  #=============== Add  Dense layers on Top ===============
42  model.add(layers.Flatten())
43  model.add(layers.Dense(64, activation='relu'))
44  model.add(layers.Dense(10))
45
46  model.summary()
```

```
flags.
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65,600
dense_1 (Dense)	(None, 10)	650

```
Total params: 122,570 (478.79 KB)
Trainable params: 122,570 (478.79 KB)
Non-trainable params: 0 (0.00 B)
PS C:\KP\MATERI\2.4 Mofifikasi Model>
```

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

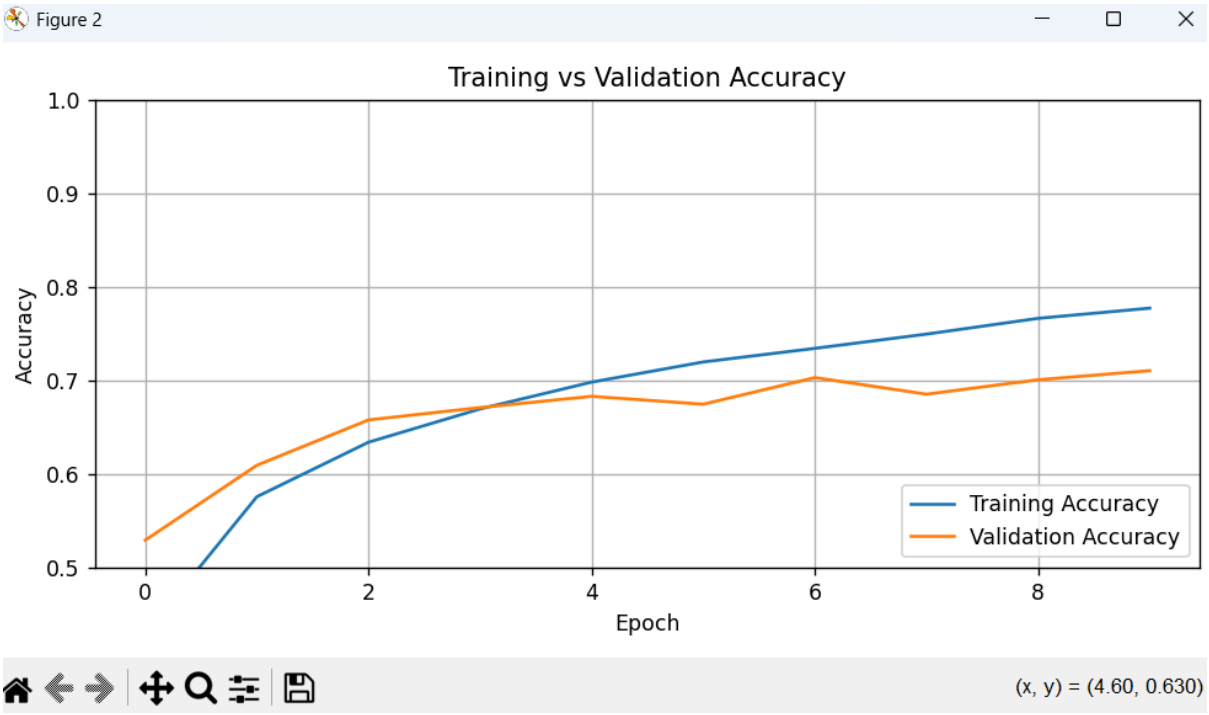
F. Compile and train the model

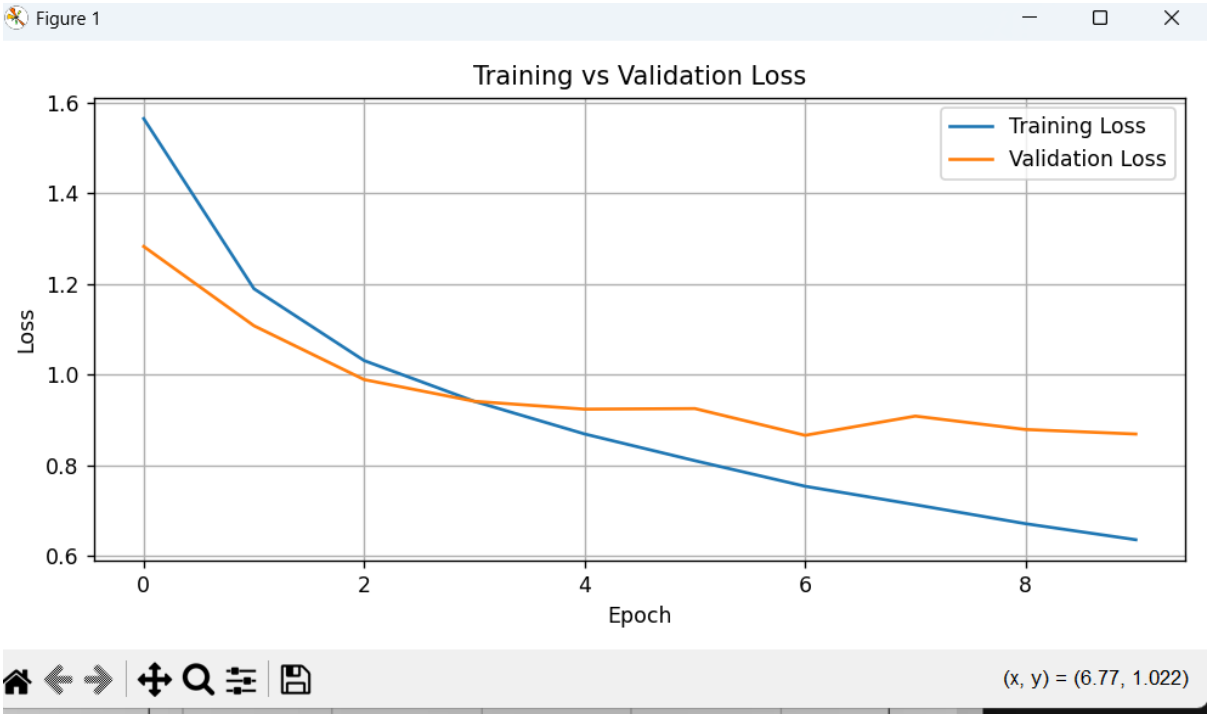
```python
#=============== Compile and Train the model ===============
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy']
              )
history=model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate c
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 19s 11ms/step - accuracy: 0.4499 - loss: 1.5118 - val_accuracy: 0.5475 - val_loss: 1.2757
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 17s 11ms/step - accuracy: 0.5915 - loss: 1.1450 - val_accuracy: 0.6153 - val_loss: 1.0781
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 17s 11ms/step - accuracy: 0.6515 - loss: 0.9931 - val_accuracy: 0.6478 - val_loss: 1.0069
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.6890 - loss: 0.8894 - val_accuracy: 0.6876 - val_loss: 0.8955
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - accuracy: 0.7141 - loss: 0.8158 - val_accuracy: 0.6797 - val_loss: 0.9202
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 17s 11ms/step - accuracy: 0.7369 - loss: 0.7543 - val_accuracy: 0.6913 - val_loss: 0.8848
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 11ms/step - accuracy: 0.7524 - loss: 0.7078 - val_accuracy: 0.7008 - val_loss: 0.8743
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.7692 - loss: 0.6595 - val_accuracy: 0.7054 - val_loss: 0.8602
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 17s 11ms/step - accuracy: 0.7837 - loss: 0.6206 - val_accuracy: 0.7178 - val_loss: 0.8593
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 17s 11ms/step - accuracy: 0.7930 - loss: 0.5877 - val_accuracy: 0.7180 - val_loss: 0.8592
PS C:\KP\MATERI\2.4 Mofifikasi Model>
```

G. Evaluate the model

```python
#=============== Evaluate the model ===============
# =============== Plot Accuracy ===============
plt.figure(figsize=(8, 4))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training vs Validation Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()

# =============== Plot Loss ===============
plt.figure(figsize=(8, 4))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training vs Validation Loss')
plt.legend(loc='upper right')
plt.grid(True)
plt.tight_layout()
plt.show()

# =============== Evaluate Model ===============
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```






```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 16s 10ms/step - accuracy: 0.7773 - loss: 0.6366 - val_acc
313/313 - 1s - 5ms/step - accuracy: 0.7106 - loss: 0.8695
Test Accuracy: 0.7106
Test Loss: 0.8695
PS C:\KP\MATERI\2.4 Mofifikasi 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.

```
#============== Preparing The Input Data ==============

# This value only applies to EMNIST dataset, consider choosing appropriate
# values if switching to other datasets.
MAX_CLIENT_DATASET_SIZE = 418

CLIENT_EPOCH_PER_ROUND = 1
CLIENT_BATCH_SIZE = 20
TEST_BATCH_SIZE = 500

emnist_train, emnist_test = tff.simulation.datasets.emnist.load_data(
    only_digits=True
)

def reshape_emnist_element(element):
    return (tf.expand_dims(element['pixels'], axis=-1), element['label'])

def preprocess_train_dataset(dataset):
    """Preprocessing function for the EMNIST training dataset."""
    return (dataset
            # Shuffle according to the largest client dataset
            .shuffle(buffer_size=MAX_CLIENT_DATASET_SIZE)

            # Repeat to do multiple local epochs
            .repeat(CLIENT_EPOCH_PER_ROUND)

            # Batch to a fixed client batch size
            .batch(CLIENT_BATCH_SIZE, drop_remainder=False)

            # Preprocessing step
            .map(reshape_emnist_element)
            )
emnist_train = emnist_train.preprocess(preprocess_train_dataset)

```

2. 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.

```
41
42  #=============== Defini a model ===============
43  def create_original_fedavg_cnn_model(only_digits=True) :
44      """The CNN model used in https://arxiv.org/abs/1602.05629."""
45      data_format = 'channel_last'
46
47      max_pool = functools.partial(
48          tf.keras.layers.MaxPooling2D,
49          pool_size=(2,2),
50          padding='same',
51          data_format=data_format)
52
53      conv2d = functools.partial(
54          tf.keras.layers.Conv2D,
55          kernel_size = 5,
56          padding='same',
57          data_format=data_format,
58          activation=tf.nn.relu)
59
60      model = tf.keras.models.Sequential([
61          tf.keras.layers.InputLayer(input_shape=(28,28,1)),
62          conv2d(filters=32),
63          max_pool(),
64          conv2d(filters=64),
65          max_pool(),
66          tf.keras.layers.Flatten(),
67          tf.keras.layers.Dense(512, activation=tf.nn.relu),
68          tf.keras.layers.Dense(10 if only_digits else 62),
69          tf.keras.layers.Softmax(),
70      ])
71
72      return model
73
74  # Gets the type information of the input data. TFF is a strongly typed
75  # functional programming framework, and needs type information about inputs to
76  # the model.
77  input_spec = emnist_train.create_tf_dataset_for_client(
78      emnist_train.client_ids[0]).element_spec
79
80  def tff_model_fn():
81      keras_model = create_original_fedavg_cnn_model()
82      return tff.learning.models.from_keras_model(
83          keras_model = keras_model,
84          input_spec = input_spec,
85          loss=tf.keras.losses.SparseCategoricalCrossentropy(),
86          metrics=[tf.keras.metrics.SparseCategoricalAccuracy()]
87      )
```

3. Training the model and outputting training metrics

Pertama, kita perlu membangun algoritma Federated Averaging menggunakan API tff.learning.algorithms.build_weighted_fed_avg.

```
92  #=============== Training the model and outputting training metrics ===============
93  federated_averaging = tff.learning.algorithms.build_weighted_fed_avg(
94      model_fn=tff_model_fn,
95      client_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=0.02),
96      server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0)
97  )
```

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.

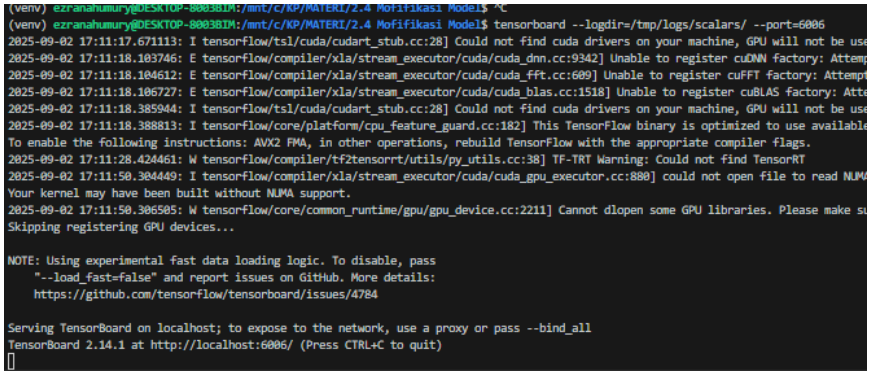
- Jalankan algoritma secara bertahap. Pada setiap tahap, keadaan server baru akan dikembalikan sebagai hasil dari setiap klien yang melatih model pada datanya. Biasanya pada satu tahap:
    - Server menyiarkan model ke semua klien yang berpartisipasi.
    - Setiap klien melakukan pekerjaan berdasarkan model dan datanya sendiri.
    - Server menggabungkan semua model untuk menghasilkan keadaan server yang berisi model baru.

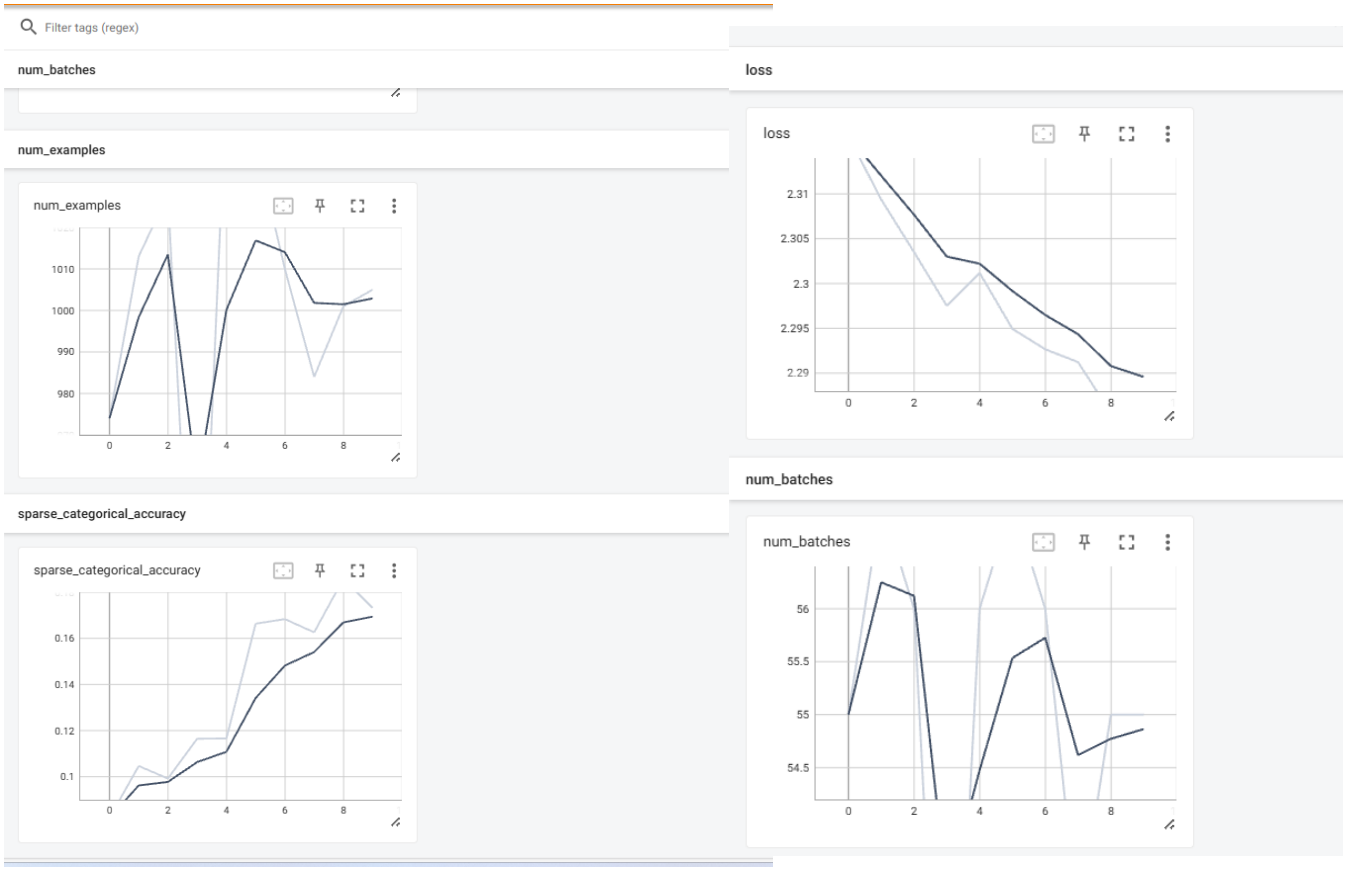
Metrik pelatihan ditulis ke direktori Tensorboard untuk ditampilkan setelah proses pelatihan selesai.

```python
#=============== Training the model and outputting training metrics ==============
federated_averaging = tff.learning.algorithms.build_weighted_fed_avg(
    model_fn=tff_model_fn,
    client_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=0.02),
    server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0)
)



def train(federated_averaging_process, num_rounds, num_clients_per_round, summary_wrtiter):
    """Trains the federated averaging process and output metrics."""

    #Initialize the Federated Averaging algorithm to get the initial server state.
    state = federated_averaging_process.initialize()

    with summary_wrtiter.as_default():
        for round_num in range(num_rounds):
            # Sample the clients parcitipated in this round.
            sampled_clients = np.random.choice(
                emnist_train.client_ids,
                size=num_clients_per_round,
                replace=False
            )

            # Create a list of `tf.Dataset` instances from the data of sampled clients.
            sampled_train_data = [
                emnist_train.create_tf_dataset_for_client(client)
                for client in sampled_clients
            ]

            # Round one round of the algorithm based on the server state and client data
            # and output the new state and metrics.
            result = federated_averaging_process.next(state, sampled_train_data)
            state = result.state
            train_metrics = result.metrics['client_work']['train']

            # === Print hasil setiap round ===
            print(f"Round {round_num}, metrics={train_metrics}")

            # Add metrics to Tensorboard.
            for name, value in train_metrics.items():
                tf.summary.scalar(name,value, step=round_num)
            summary_wrtiter.flush()

# Clean the log directory to avoid conflicts.
try :
    tf.io.gfile.rmtree('/tmp/logs/scalar')
except tf.errors.OpError as e :
    pass # Path doesn't exist

# Set up the log directory and writer for Tensorboard.
logdir = "/tmp/logs/scalars/original/"
summary_writter = tf.summary.create_file_writer(logdir)

train(federated_averaging_process=federated_averaging, num_rounds=10,
      num_clients_per_round=10, summary_wrtiter=summary_writter)
```

```
Round 0, metrics=OrderedDict([('sparse_categorical_accuracy', 0.08213552), ('loss', 2.3164458), ('num_examples', 974), ('num_batches', 55)])
Round 1, metrics=OrderedDict([('sparse_categorical_accuracy', 0.10463969), ('loss', 2.3093743), ('num_examples', 1013), ('num_batches', 57)])
Round 2, metrics=OrderedDict([('sparse_categorical_accuracy', 0.09922179), ('loss', 2.303509), ('num_examples', 1028), ('num_batches', 56)])
Round 2, metrics=OrderedDict([('sparse_categorical_accuracy', 0.09922179), ('loss', 2.303509), ('num_examples', 1028), ('num_batches', 56)])
Round 3, metrics=OrderedDict([('sparse_categorical_accuracy', 0.11646137), ('loss', 2.2975066), ('num_examples', 893), ('num_batches', 50)])
Round 4, metrics=OrderedDict([('sparse_categorical_accuracy', 0.116587676), ('loss', 2.3011804), ('num_examples', 1055), ('num_batches', 56)])
Round 5, metrics=OrderedDict([('sparse_categorical_accuracy', 0.16634615), ('loss', 2.2949393), ('num_examples', 1040), ('num_batches', 57)])
Round 6, metrics=OrderedDict([('sparse_categorical_accuracy', 0.16831683), ('loss', 2.2926638), ('num_examples', 1010), ('num_batches', 56)])
Round 7, metrics=OrderedDict([('sparse_categorical_accuracy', 0.16260162), ('loss', 2.2912276), ('num_examples', 984), ('num_batches', 53)])
Round 8, metrics=OrderedDict([('sparse_categorical_accuracy', 0.18581419), ('loss', 2.2855632), ('num_examples', 1001), ('num_batches', 55)])
Round 9, metrics=OrderedDict([('sparse_categorical_accuracy', 0.17313433), ('loss', 2.2877924), ('num_examples', 1005), ('num_batches', 55)])
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.4 Mofifikasi Model$
```

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.





4. 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.

```
#============== Build a custom aggregation function ==============
compression_aggregator = tff.learning.compression_aggregator()
print(isinstance(compression_aggregator, tff.aggregators.WeightedAggregationFactory))
```

```
True
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.4 Mofifikasi Model$
```

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.

```
federated_averaging_with_compression = tff.learning.algorithms.build_weighted_fed_avg(
    tff_model_fn,
    client_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=0.02),
    server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0),
    model_aggregator=compression_aggregator
)
```

5. Training the model again

Sekarang mari kita jalankan algoritma Federated Averaging yang baru.

```
#============== Training the model again ==============]
logdir_for_compression = "/tmp/logs/scalars/compression/"
summary_writter_for_compression = tf.summary.create_file_writer(
    logdir_for_compression
)

train(federated_averaging_process=federated_averaging_with_compression,
      num_rounds=10,
      num_clients_per_round=10,
      summary_writter=summary_writter_for_compression)
```

```
Round 0, metrics=OrderedDict([('sparse_categorical_accuracy', 0.08814888), ('loss', 2.319201), ('num_examples', 1021), ('num_batches', 57)])
Round 1, metrics=OrderedDict([('sparse_categorical_accuracy', 0.08933718), ('loss', 2.3106072), ('num_examples', 1041), ('num_batches', 59)])
Round 2, metrics=OrderedDict([('sparse_categorical_accuracy', 0.08442211), ('loss', 2.3068802), ('num_examples', 995), ('num_batches', 53)])
Round 3, metrics=OrderedDict([('sparse_categorical_accuracy', 0.08372978), ('loss', 2.3055167), ('num_examples', 1051), ('num_batches', 58)])
Round 4, metrics=OrderedDict([('sparse_categorical_accuracy', 0.12513034), ('loss', 2.2985919), ('num_examples', 959), ('num_batches', 52)])
Round 5, metrics=OrderedDict([('sparse_categorical_accuracy', 0.10132576), ('loss', 2.302568), ('num_examples', 1056), ('num_batches', 57)])
Round 6, metrics=OrderedDict([('sparse_categorical_accuracy', 0.123667374), ('loss', 2.295725), ('num_examples', 938), ('num_batches', 50)])
Round 7, metrics=OrderedDict([('sparse_categorical_accuracy', 0.13157895), ('loss', 2.2974281), ('num_examples', 1026), ('num_batches', 56)])
Round 8, metrics=OrderedDict([('sparse_categorical_accuracy', 0.12820514), ('loss', 2.2971292), ('num_examples', 1014), ('num_batches', 56)])
Round 9, metrics=OrderedDict([('sparse_categorical_accuracy', 0.11657559), ('loss', 2.295844), ('num_examples', 1098), ('num_batches', 60)])
True
(venv) ezranahumury@DESKTOP-8003BIM:/mnt/c/KP/MATERI/2.4 Mofifikasi Model$ python TFF_WRAPPER.py
```

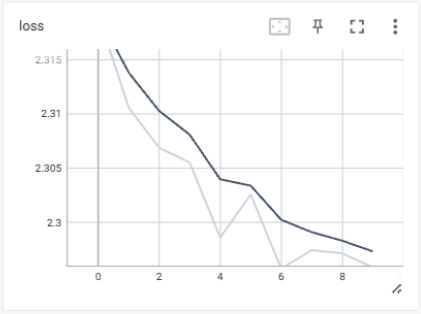
Jalankan TensorBoard kembali untuk membandingkan metrik pelatihan antara dua kali eksekusi.

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.

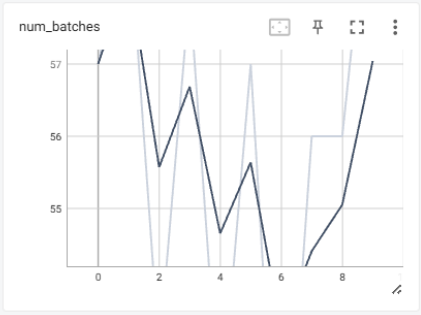
Kesimpulannya, kami telah mengimplementasikan algoritma kompresi yang dapat mencapai kinerja serupa dengan algoritma Federated Averaging asli, sementara biaya komunikasi berkurang secara signifikan.

Filter tags (regex)

loss



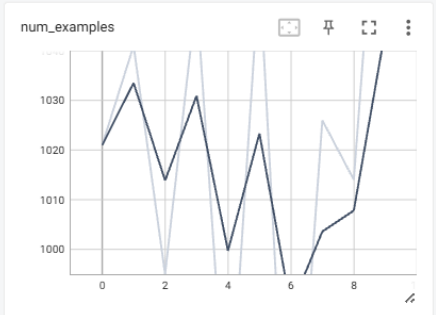
num_batches



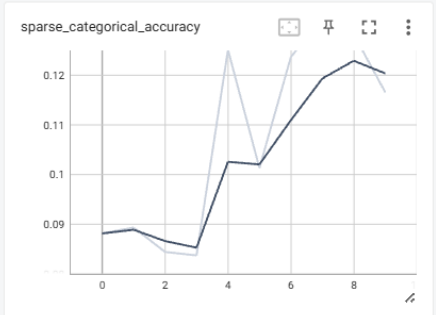
Filter tags (regex)

num_batches

num_examples



sparse_categorical_accuracy



Task : Custom Sederhana

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

```python
#tambahan
def create_simple_cnn_model():
    """A smaller CNN as the 'custom sederhana' baseline."""
    model = tf.keras.models.Sequential([
        tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),
        tf.keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(10),           # digits: 10 classes
        tf.keras.layers.Softmax(),
    ])
    return model
```

```python
def make_tff_model_fn(keras_model_fn):
    def tff_model_fn():
        keras_model = keras_model_fn()
        return tff.learning.models.from_keras_model(
            keras_model=keras_model,
            input_spec=input_spec,
            loss=tf.keras.losses.SparseCategoricalCrossentropy(),
            metrics=[tf.keras.metrics.SparseCategoricalAccuracy()]
        )
    return tff_model_fn
```

```python
# ---------------- Build FedAvg process ----------------------
def build_fedavg_process(keras_model_fn, model_aggregator=None):
    return tff.learning.algorithms.build_weighted_fed_avg(
        model_fn=make_tff_model_fn(keras_model_fn),
        client_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=0.02),
        server_optimizer_fn=tff.learning.optimizers.build_sgdm(learning_rate=1.0),
        model_aggregator=model_aggregator
    )
```

```python
def train(fedavg_process, *, num_rounds, num_clients_per_round, writer, label):
    state = fedavg_process.initialize()
    client_ids = np.array(emnist_train.client_ids)

    with writer.as_default():
        for round_num in range(1, num_rounds + 1):
            sampled_clients = np.random.choice(
                client_ids, size=num_clients_per_round, replace=False
            )
            sampled_train_data = [
                emnist_train.create_tf_dataset_for_client(cid) for cid in sampled_clients
            ]

            result = fedavg_process.next(state, sampled_train_data)
            state = result.state
            mt = result.metrics['client_work']['train']

            acc = float(mt.get('sparse_categorical_accuracy', 0.0))
            loss = float(mt.get('loss', 0.0))

            #Tambahan 1: print ringkas setiap round
            print(f"[{label}] Round {round_num:02d} | acc={acc:.4f} | loss={loss:.4f}")

            #Tambahan 2: logging scalar dengan prefix label
            tf.summary.scalar(f"{label}/train/accuracy", acc, step=round_num)
            tf.summary.scalar(f"{label}/train/loss", loss, step=round_num)

            writer.flush()
    return state
```

```python
# 2) Custom Simple CNN
proc_simple = build_fedavg_process(create_simple_cnn_model)
logdir_simple = "/tmp/logs/scalars/simple/"
writer_simple = tf.summary.create_file_writer(logdir_simple)

train(fedavg_process=proc_simple,
      num_rounds=10,
      num_clients_per_round=10,
      writer=writer_simple,
      label="simple")
```

```
Skipping registering GPU devices...
[simple] Round 01 | acc=0.0709 | loss=2.3525
[simple] Round 02 | acc=0.0815 | loss=2.3287
[simple] Round 03 | acc=0.0951 | loss=2.3151
[simple] Round 04 | acc=0.0921 | loss=2.3171
[simple] Round 05 | acc=0.0923 | loss=2.3115
[simple] Round 06 | acc=0.1194 | loss=2.3012
[simple] Round 07 | acc=0.1286 | loss=2.2940
[simple] Round 08 | acc=0.1219 | loss=2.2911
[simple] Round 09 | acc=0.1254 | loss=2.2885
[simple] Round 10 | acc=0.1280 | loss=2.2924
[info] compression_aggregator is WeightedAggregationFactory: True
2025-09-03 01:16:04.695862: I tensorflow/compiler/xla/stream_executor/cuda/cuda
```

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.

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

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