Nama : Nabilah Salwa

NIM : 1103204060

UAS Machine Learning

## 03. PyTorch Computer Vision Exercise

```
# Check for GPU
!nvidia-smi
```

Kode di atas menggunakan perintah !nvidia-smi untuk mengecek ketersediaan dan informasi GPU pada sistem. Perintah ini biasanya digunakan untuk menampilkan informasi tentang GPU, termasuk spesifikasi dan penggunaan sumber daya. Jika sistem memiliki GPU yang terhubung, hasil dari perintah ini akan menampilkan detail GPU seperti nama, penggunaan memori, dan informasi lainnya.

```
# Import torch
import torch

# Exercises require PyTorch > 1.10.0
print(torch.__version__)

# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

Kode di atas memiliki dua tujuan utama. Pertama, untuk menunjukkan versi PyTorch yang digunakan dengan mencetak versi PyTorch menggunakan `print(torch.__version__)`. Kedua, untuk menentukan perangkat yang akan digunakan untuk komputasi, apakah itu CPU atau GPU.

Pertama-tama, versi PyTorch dicetak ke layar menggunakan `print(torch.__version__)` untuk memberikan informasi tentang versi PyTorch yang sedang digunakan. Setelah itu, menggunakan `torch.cuda.is_available()`, kode mengecek ketersediaan GPU. Jika GPU tersedia, variabel `device` diatur sebagai "cuda" (menunjukkan penggunaan GPU); jika tidak, variabel `device` diatur sebagai "cpu" (menunjukkan penggunaan CPU). Variabel `device` kemudian dicetak untuk memberikan informasi tentang perangkat yang akan digunakan untuk komputasi.

1. Load the torchvision.datasets.MNIST() train and test datasets.

   ```
   import torchvision
   from torchvision import datasets

   from torchvision import transforms
   ```

   Kode di atas melakukan impor modul-modul yang diperlukan dari pustaka `torchvision` dalam lingkungan PyTorch. Pertama, `import torchvision` mengimpor seluruh pustaka `torchvision`,

yang menyediakan berbagai alat dan fungsi untuk pengolahan data gambar dan penggunaan dataset pada PyTorch. Selanjutnya, `from torchvision import datasets` memungkinkan akses ke modul `datasets`, yang berisi fungsi-fungsi untuk mendownload dan mengelola dataset umum dalam visi komputer, seperti MNIST dan CIFAR-10. Terakhir, `from torchvision import transforms` memungkinkan penggunaan modul `transforms`, yang menyediakan berbagai transformasi yang dapat diterapkan pada gambar, seperti pemotongan, perputaran, dan normalisasi. Dengan impor modul-modul ini, pengguna dapat lebih mudah memanfaatkan berbagai dataset dan melakukan prapemrosesan data gambar secara efisien.

```
# Get the MNIST train dataset
train_data = datasets.MNIST(root=".",
                            train=True,
                            download=True,
                            transform=transforms.ToTensor())

# Get the MNIST test dataset
test_data = datasets.MNIST(root=".",
                           train=False,
                           download=True,
                           transform=transforms.ToTensor())
```

Kode di atas menggunakan pustaka `torchvision` untuk mengakses dataset MNIST, yang merupakan dataset gambar digit tulisan tangan. Pertama, `train_data = datasets.MNIST(...)` mendefinisikan variabel `train_data` yang berisi dataset MNIST untuk pelatihan model. Parameter `root="."` menunjukkan bahwa dataset akan disimpan di direktori saat ini. Parameter `train=True` menunjukkan bahwa kita menginginkan dataset untuk pelatihan. Jika dataset belum diunduh, `download=True` akan mengunduhnya secara otomatis. Transformasi `transforms.ToTensor()` digunakan untuk mengubah gambar menjadi tensor PyTorch. Selanjutnya, `test_data = datasets.MNIST(...)` mendefinisikan variabel `test_data` untuk dataset MNIST yang akan digunakan dalam pengujian model. Parameter `train=False` menunjukkan bahwa ini adalah dataset pengujian. Seperti sebelumnya, `download=True` akan mengunduh dataset jika belum ada, dan transformasi `transforms.ToTensor()` digunakan untuk mengonversi gambar menjadi tensor PyTorch. Dengan mengimplementasikan kode ini, kita dapat dengan mudah mengakses dan mempersiapkan dataset MNIST untuk melatih dan menguji model pembelajaran mesin.

```
train_data, test_data
```

Baris kode train_data, test_data adalah pernyataan yang mencetak dua variabel, yaitu train_data dan test_data. Variabel ini sebelumnya telah didefinisikan untuk menyimpan dua dataset MNIST yang berbeda: train_data digunakan untuk melatih model, sementara test_data digunakan untuk menguji model.

```
len(train_data), len(test_data)
```

Baris kode `len(train_data), len(test_data)` bertujuan untuk mengecek jumlah sampel atau gambar yang terdapat dalam dataset pelatihan (`train_data`) dan dataset pengujian (`test_data`). Fungsi `len()` digunakan untuk menghitung jumlah elemen atau panjang dari setiap dataset. Dengan menjalankan baris kode ini, kita mendapatkan dua nilai: jumlah sampel dalam dataset pelatihan dan jumlah sampel dalam dataset pengujian. Angka ini mencerminkan banyaknya gambar digit tulisan tangan yang tersedia untuk melatih dan menguji model. Informasi ini berguna untuk memastikan bahwa dataset memiliki ukuran yang memadai untuk melatih dan menguji model dengan baik.

```
# Data is in tuple form (image, label)
img = train_data[0][0]
label = train_data[0][1]
print(f"Image:\n {img}")
print(f"Label:\n {label}")
```

Dalam blok kode di atas, kita menggunakan indeks `[0]` untuk mengakses satu sampel pertama dari dataset pelatihan MNIST. Setiap sampel dalam dataset ini berbentuk tuple, dengan elemen pertama berisi tensor PyTorch yang merepresentasikan gambar digit tulisan tangan, dan elemen kedua berisi label yang menunjukkan digit yang sesuai. Variabel `img` menyimpan tensor gambar dari sampel pertama, sementara variabel `label` menyimpan label yang sesuai. Setelah mengakses elemen-elemen tersebut, kita mencetak informasi tersebut untuk mengamati struktur dan nilai-nilai dalam tensor gambar, serta label yang bersesuaian. Proses ini membantu kita memahami representasi data di dalam dataset dan memberikan wawasan awal tentang format gambar dan label yang akan digunakan dalam pelatihan model.

```
# Check out the shapes of our data
print(f"Image shape: {img.shape} -> [color_channels, height, width] (CHW)")
print(f"Label: {label} -> no shape, due to being integer")
```

Dalam blok kode di atas, kita menggunakan perintah `print` untuk menampilkan informasi tentang bentuk (shape) dari data gambar dan label dari satu sampel dalam dataset pelatihan MNIST. Pertama, kita mencetak bentuk dari tensor gambar (`img`). Tensor gambar ini memiliki tiga dimensi yang mewakili saluran warna (color channels), tinggi (height), dan lebar (width). Representasi tersebut disusun dalam format CHW (Color-Height-Width), dengan saluran warna sebagai dimensi pertama, diikuti oleh tinggi dan lebar. Selanjutnya, kita mencetak label (`label`). Label ini merupakan bilangan bulat yang menunjukkan digit yang terdapat pada gambar. Karena label hanya merupakan satu nilai bulat, tidak ada bentuk (shape) yang terkait dengan label tersebut. Informasi ini membantu untuk memahami struktur data dan memastikan bahwa data dalam dataset sesuai dengan harapan, sehingga dapat digunakan dengan benar dalam proses pelatihan model.

```
# Get the class names from the dataset
class_names = train_data.classes
class_names
```

Dalam blok kode di atas, kita menggunakan atribut `classes` dari objek dataset pelatihan MNIST (`train_data`) untuk mendapatkan daftar nama kelas (class names). Dataset MNIST menyediakan metode ini untuk mengambil kumpulan nama kelas yang sesuai dengan digit 0 hingga 9. Hasilnya, variabel `class_names` akan berisi daftar nama kelas, di mana setiap nama kelas merepresentasikan digit yang sesuai. Dalam dataset MNIST, nama kelas akan berurutan dari "0" hingga "9", sesuai dengan digit yang dapat muncul pada gambar tulisan tangan dalam dataset ini. Informasi ini berguna untuk mengetahui korespondensi antara label numerik dan representasi teks dari kelas-kelas digit dalam dataset, memudahkan interpretasi hasil prediksi model nantinya.

2. Visualize at least 5 different samples of the MNIST training dataset.

```
import matplotlib.pyplot as plt
for i in range(5):
  img = train_data[i][0]
  print(img.shape)
  img_squeeze = img.squeeze()
  print(img_squeeze.shape)
  label = train_data[i][1]
  plt.figure(figsize=(3, 3))
  plt.imshow(img_squeeze, cmap="gray")
  plt.title(label)
  plt.axis(False);
```

Dalam blok kode di atas, kita menggunakan perulangan `for` untuk mengambil lima sampel pertama dari dataset pelatihan MNIST (`train_data`). Untuk setiap sampel, kita mengakses tensor gambar (`img`) dan label (`label`) menggunakan indeks. Kemudian, kita mencetak bentuk (shape) dari tensor gambar (`img`) untuk melihat struktur data. Kita kemudian menggunakan metode `squeeze()` untuk menghilangkan dimensi yang memiliki panjang 1, karena gambar pada dataset MNIST berupa gambar grayscale, sehingga hanya memiliki satu saluran warna. Setelah itu, kita mencetak kembali bentuk dari tensor gambar yang sudah di-"squeeze" (`img_squeeze`).

Selanjutnya, kita menggunakan `matplotlib` untuk menampilkan gambar dalam bentuk matriks piksel dan memberi judul sesuai dengan label digit yang sesuai. Langkah-langkah tersebut diulang untuk lima sampel pertama, dan hasilnya ditampilkan dalam bentuk gambar menggunakan fungsi `imshow` dari `matplotlib`. Informasi ini membantu kita untuk memahami struktur data gambar dalam dataset dan memberikan visualisasi awal terhadap dataset MNIST.

3. Turn the MNIST train and test datasets into dataloaders using `torch.utils.data.DataLoader`, set the `batch_size=32.

```
# Create train dataloader
from torch.utils.data import DataLoader

train_dataloader = DataLoader(dataset=train_data,
                              batch_size=32,
                              shuffle=True)

test_dataloader = DataLoader(dataset=test_data,
                             batch_size=32,
                             shuffle=False)
```

Dalam blok kode di atas, kita menggunakan modul `DataLoader` dari PyTorch untuk membuat dataloader untuk dataset pelatihan dan pengujian MNIST. Dataloader digunakan untuk mempermudah pengolahan data dalam bentuk batch selama pelatihan dan evaluasi model.
Pertama, kita membuat dataloader untuk dataset pelatihan (`train_data`). Dalam konteks ini, kita mengatur ukuran batch menjadi 32, yang berarti bahwa 32 sampel akan dimuat bersamaan dalam setiap iterasi pelatihan. Pengaturan `shuffle=True` digunakan untuk mengacak urutan sampel setiap epoch agar model tidak mengingat pola urutan data. Kemudian, kita membuat dataloader untuk dataset pengujian (`test_data`). Pengaturan `shuffle=False` digunakan karena pada fase pengujian, kita ingin mempertahankan urutan data asli untuk evaluasi yang konsisten. Dengan menggunakan dataloader, kita dapat dengan mudah mengakses dan memproses data dalam batch selama pelatihan dan evaluasi model, meningkatkan efisiensi dalam penggunaan sumber daya dan mempercepat proses pelatihan.

```
train_dataloader, test_dataloader
```

Dalam blok kode tersebut, kita membuat dua objek DataLoader terpisah untuk dataset pelatihan (train_data) dan dataset pengujian (test_data). Objek-objek ini, yaitu train_dataloader dan test_dataloader, digunakan untuk mempermudah proses iterasi dan pengolahan data selama pelatihan dan evaluasi model.

```
for sample in next(iter(train_dataloader)):
  print(sample.shape)
```

Dalam blok kode tersebut, kita menggunakan iter(train_dataloader) untuk mendapatkan iterator dari train_dataloader. Kemudian, dengan menggunakan next(), kita mengambil satu batch data pertama dari pelatihan dataloader. Setiap batch data memiliki bentuk yang dapat dicetak dengan menggunakan pernyataan print(sample.shape).

```
len(train_dataloader), len(test_dataloader)
```

Dalam blok kode tersebut, kita menggunakan fungsi len() untuk menghitung jumlah batch dalam train_dataloader dan test_dataloader. Kedua dataloader ini telah disiapkan sebelumnya untuk memproses data pelatihan dan pengujian dari dataset MNIST. Pada umumnya, panjang

dataloader mencerminkan jumlah batch yang dapat digunakan selama satu epoch dari dataset yang diberikan.

4. Recreate model_2 used in notebook 03 (the same model from the CNN Explainer website, also known as TinyVGG) capable of fitting on the MNIST dataset.

```python
from torch import nn
class MNIST_model(torch.nn.Module):
  """Model capable of predicting on MNIST dataset.
  """
  def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
    super().__init__()
    self.conv_block_1 = nn.Sequential(
      nn.Conv2d(in_channels=input_shape,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.Conv2d(in_channels=hidden_units,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2)
    )
    self.conv_block_2 = nn.Sequential(
      nn.Conv2d(in_channels=hidden_units,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.Conv2d(in_channels=hidden_units,
                out_channels=hidden_units,
                kernel_size=3,
                stride=1,
                padding=1),
      nn.ReLU(),
      nn.MaxPool2d(kernel_size=2)
    )
    self.classifier = nn.Sequential(
      nn.Flatten(),
      nn.Linear(in_features=hidden_units*7*7,
                out_features=output_shape)
    )

  def forward(self, x):
    x = self.conv_block_1(x)
    # print(f"Output shape of conv block 1: {x.shape}")
    x = self.conv_block_2(x)
    # print(f"Output shape of conv block 2: {x.shape}")
    x = self.classifier(x)
    # print(f"Output shape of classifier: {x.shape}")
    return x
```

Blok kode tersebut mendefinisikan sebuah kelas model PyTorch yang disebut `MNIST_model`. Kelas ini mewarisi dari kelas `torch.nn.Module`. Model ini dirancang untuk melakukan prediksi pada dataset MNIST, yang terdiri dari gambar digit tulisan tangan.

Dalam konstruktor (`__init__`), model ini dibangun dengan menggunakan dua blok konvolusi (`conv_block_1` dan `conv_block_2`) yang terdiri dari lapisan-lapisan konvolusi 2D, aktivasi ReLU, dan lapisan pooling maksimum. Setelah itu, terdapat lapisan classifier yang terdiri dari lapisan flatten (untuk meratakan output dari blok konvolusi menjadi vektor), dan lapisan linear yang menghubungkan input ke output. Jumlah fitur masukan dan keluaran serta jumlah unit tersembunyi dapat diatur melalui parameter konstruktor. Metode `forward` mendefinisikan aliran maju (forward pass) dari model. Input melewati kedua blok konvolusi dan kemudian melalui lapisan classifier. Output akhir dari model adalah tensor yang berisi prediksi kelas untuk setiap gambar dalam batch. Model ini mencoba untuk memahami fitur-fitur hierarki

dari gambar MNIST melalui konvolusi dan memprosesnya untuk prediksi kelas akhir melalui lapisan-lapisan linear.

```
device
```

Blok kode tersebut mendefinisikan variabel `device`, yang digunakan untuk menentukan perangkat (device) tempat tensor PyTorch akan ditempatkan. Variabel ini diatur untuk menggunakan perangkat GPU ("cuda") jika tersedia, dan jika tidak, akan menggunakan perangkat CPU ("cpu"). Hal ini memungkinkan fleksibilitas dalam menentukan di mana tensor-tensor PyTorch akan disimpan dan diolah, dengan preferensi untuk menggunakan GPU jika tersedia untuk percepatan perhitungan.

```
model = MNIST_model(input_shape=1,
                    hidden_units=10,
                    output_shape=10).to(device)
model
```

Blok kode tersebut membuat instance dari model `MNIST_model` dengan spesifikasi sebagai berikut:
- `input_shape=1`: Menandakan bahwa masukan ke model adalah gambar grayscale dengan satu saluran warna.
- `hidden_units=10`: Jumlah filter atau unit tersembunyi yang digunakan dalam setiap lapisan konvolusi.
- `output_shape=10`: Jumlah kelas output, sesuai dengan jumlah kelas digit pada dataset MNIST (0 hingga 9).

Model tersebut kemudian dipindahkan ke perangkat yang ditentukan sebelumnya dengan menggunakan `.to(device)`. Ini memastikan bahwa model dan semua parameternya berada pada perangkat yang sama dengan `device` yang telah ditentukan sebelumnya (misalnya, GPU atau CPU). Hasilnya adalah model yang siap untuk dilatih dan dievaluasi pada perangkat yang ditentukan. Model ini dirancang untuk melakukan klasifikasi digit pada dataset MNIST.

```
# Try a dummy forward pass to see what shapes our data is
dummy_x = torch.rand(size=(1, 28, 28)).unsqueeze(dim=0).to(device)
# dummy_x.shape
model(dummy_x)
```

Blok kode tersebut melakukan penerapan maju (forward pass) pada model MNIST dengan menggunakan data contoh yang dihasilkan secara acak (`dummy_x`). `dummy_x` dihasilkan dengan ukuran (1, 28, 28), yang sesuai dengan format gambar grayscale MNIST. Proses `.unsqueeze(dim=0)` dilakukan untuk menambahkan dimensi batch sehingga menjadi (1, 1, 28, 28), yang sesuai dengan format yang diperlukan oleh model.

Selanjutnya, data contoh tersebut dipindahkan ke perangkat yang ditentukan sebelumnya dengan `.to(device)` untuk memastikan konsistensi perangkat. Kemudian, forward pass

dijalankan pada model menggunakan data contoh, dan keluarannya dicetak. Ini memberikan kita gambaran tentang bentuk keluaran dari setiap lapisan di model.

```
dummy_x_2 = torch.rand(size=([1, 10, 7, 7]))
dummy_x_2.shape
```

Dalam blok kode tersebut, dibuat sebuah tensor acak yang disebut `dummy_x_2` menggunakan fungsi `torch.rand`. Tensor ini memiliki ukuran (1, 10, 7, 7), yang mencerminkan dimensi yang mungkin muncul dalam pengolahan gambar melalui lapisan-lapisan konvolusi. Dalam konteks model konvolusi, dimensi (1, 10) menunjukkan adanya 10 saluran (channel) atau fitur hasil dari penggunaan beberapa filter. Dimensi (7, 7) mencerminkan tinggi dan lebar gambar hasil dari operasi konvolusi dan pooling.

```
flatten_layer = nn.Flatten()
flatten_layer(dummy_x_2).shape
```

Dalam blok kode tersebut, sebuah objek `flatten_layer` dari kelas `nn.Flatten()` dibuat. Fungsi ini bertujuan untuk "membentangkan" (flatten) tensor, yaitu mengubah tensor multi-dimensi menjadi tensor satu dimensi. Pada tahap ini, `flatten_layer` diaplikasikan pada `dummy_x_2`, yang memiliki ukuran awal (1, 10, 7, 7).

5. Train the model you built in exercise 8. for 5 epochs on CPU and GPU and see how long it takes on each.

```
%%time
from tqdm.auto import tqdm

# Train on CPU
model_cpu = MNIST_model(input_shape=1,
                        hidden_units=10,
                        output_shape=10).to("cpu")

# Create a loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_cpu.parameters(), lr=0.1)

### Training loop
epochs = 5
for epoch in tqdm(range(epochs)):
  train_loss = 0
  for batch, (X, y) in enumerate(train_dataloader):
    model_cpu.train()

    # Put data on CPU
    X, y = X.to("cpu"), y.to("cpu")

    # Forward pass
    y_pred = model_cpu(X)

    # Loss calculation
    loss = loss_fn(y_pred, y)
    train_loss += loss

    # Optimizer zero grad
    optimizer.zero_grad()
```

```
  # Loss backward
  loss.backward()

  # Step the optimizer
  optimizer.step()

# Adjust train loss for number of batches
train_loss /= len(train_dataloader)

### Testing loop
test_loss_total = 0

# Put model in eval mode
model_cpu.eval()

# Turn on inference mode
with torch.inference_mode():
  for batch, (X_test, y_test) in enumerate(test_dataloader):
    # Make sure test data on CPU
    X_test, y_test = X_test.to("cpu"), y_test.to("cpu")
    test_pred = model_cpu(X_test)
    test_loss = loss_fn(test_pred, y_test)

    test_loss_total += test_loss

  test_loss_total /= len(test_dataloader)

# Print out what's happening
print(f"Epoch: {epoch} | Loss: {train_loss:.3f} | Test loss: {test_loss_total:.3f}")
```

Dalam blok kode tersebut, sebuah model CNN (Convolutional Neural Network) untuk klasifikasi dataset MNIST diimplementasikan dan dilatih pada CPU. Model tersebut menggunakan lapisan-lapisan konvolusi dan lapisan-lapisan linear untuk mengolah citra digit dari dataset MNIST. Proses pelatihan dilakukan selama beberapa epoch menggunakan metode gradien turun stokastik (Stochastic Gradient Descent - SGD) sebagai pengoptimalkan dan fungsi kerugian CrossEntropyLoss.

Pada setiap epoch pelatihan, model dievaluasi pada data pelatihan menggunakan batch yang diambil dari dataloader. Kemudian, gradien dari fungsi kerugian dihitung menggunakan backpropagation, dan parameter model diperbarui dengan metode SGD. Selama proses pelatihan, fungsi kerugian untuk data pelatihan dihitung dan dicetak sebagai indikator performa model.

Setelah setiap epoch pelatihan, model dievaluasi pada data uji menggunakan batch yang diambil dari dataloader uji. Fungsi kerugian pada data uji juga dihitung dan dicetak untuk memantau performa model pada data yang tidak terlihat selama pelatihan. Proses ini diulang selama beberapa epoch, dan hasilnya dicetak untuk setiap epoch. Metode tqdm digunakan untuk menambahkan bar kemajuan untuk memantau kemajuan pelatihan.

```
%%time
from tqdm.auto import tqdm

device = "cuda" if torch.cuda.is_available() else "cpu"

# Train on GPU
model_gpu = MNIST_model(input_shape=1,
                        hidden_units=10,
                        output_shape=10).to(device)

# Create a loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_gpu.parameters(), lr=0.1)

# Training loop
epochs = 5
for epoch in tqdm(range(epochs)):
  train_loss = 0
  model_gpu.train()
  for batch, (X, y) in enumerate(train_dataloader):
    # Put data on target device
    X, y = X.to(device), y.to(device)

    # Forward pass
    y_pred = model_gpu(X)

    # Loss calculation
    loss = loss_fn(y_pred, y)
    train_loss += loss

    # Optimizer zero grad
    optimizer.zero_grad()
```

```
  # Loss backward
  loss.backward()

  # Step the optimizer
  optimizer.step()

# Adjust train loss to number of batches
train_loss /= len(train_dataloader)

### Testing loop
test_loss_total = 0
# Put model in eval mode and turn on inference mode
model_gpu.eval()
with torch.inference_mode():
  for batch, (X_test, y_test) in enumerate(test_dataloader):
    # Make sure test data on target device
    X_test, y_test = X_test.to(device), y_test.to(device)

    test_pred = model_gpu(X_test)
    test_loss = loss_fn(test_pred, y_test)

    test_loss_total += test_loss

  # Adjust test loss total for number of batches
  test_loss_total /= len(test_dataloader)

# Print out what's happening
print(f"Epoch: {epoch} | Loss: {train_loss:.3f} | Test loss: {test_loss_total:.3f}")
```

Dalam blok kode tersebut, model CNN untuk klasifikasi dataset MNIST diimplementasikan dan dilatih pada GPU jika tersedia, atau pada CPU jika tidak. Model tersebut menggunakan lapisan-lapisan konvolusi dan lapisan-lapisan linear untuk mengolah citra digit dari dataset MNIST. Proses pelatihan dilakukan selama beberapa epoch menggunakan metode gradien turun stokastik (Stochastic Gradient Descent - SGD) sebagai pengoptimalkan dan fungsi kerugian CrossEntropyLoss.

Selama setiap epoch pelatihan, model dievaluasi pada data pelatihan menggunakan batch yang diambil dari dataloader. Gradien dari fungsi kerugian dihitung menggunakan backpropagation, dan parameter model diperbarui dengan metode SGD. Pada akhir setiap epoch, model dievaluasi pada data uji menggunakan batch yang diambil dari dataloader uji. Fungsi kerugian pada data uji dihitung dan dicetak untuk memantau performa model pada data yang tidak terlihat selama pelatihan.

Proses pelatihan ini diulang selama beberapa epoch, dan hasilnya dicetak untuk setiap epoch. Metode tqdm digunakan untuk menambahkan bar kemajuan yang membantu memantau kemajuan pelatihan. Model dievaluasi pada perangkat target yang sesuai (CPU atau GPU) sesuai dengan ketersediaan perangkat keras.

6. Make predictions using your trained model and visualize at least 5 of them comparing the prediciton to the target label.

```
# Make predictions with the trained model
plt.imshow(test_data[0][0].squeeze(), cmap="gray")
```

Dalam blok kode tersebut, kita membuat prediksi dengan model yang telah dilatih pada dataset uji MNIST. Kita menampilkan gambar dari dataset uji menggunakan matplotlib, dan kemudian menggunakan model yang telah dilatih untuk membuat prediksi terhadap gambar tersebut. Gambar ditampilkan menggunakan warna abu-abu (cmap="gray") karena dataset MNIST berisi citra digit biner yang direpresentasikan dalam skala abu-abu.

```
# Logits -> Prediction probabilities -> Prediction labels
model_pred_logits = model_gpu(test_data[0][0].unsqueeze(dim=0).to(device)) # make sure image is right shape + on right device
model_pred_probs = torch.softmax(model_pred_logits, dim=1)
model_pred_label = torch.argmax(model_pred_probs, dim=1)
model_pred_label
```

Dalam blok kode tersebut, kita melakukan langkah-langkah untuk mengonversi output model, yang disebut logits, menjadi prediksi akhir. Pertama, kita menggunakan model yang telah dilatih untuk membuat logits terhadap suatu gambar dari dataset uji MNIST. Selanjutnya, kita menerapkan fungsi softmax pada logits untuk menghasilkan distribusi probabilitas. Dengan distribusi probabilitas ini, kita kemudian menentukan label prediksi dengan memilih indeks dengan probabilitas tertinggi menggunakan fungsi argmax. Hasilnya adalah prediksi label yang dihasilkan oleh model untuk gambar tersebut dalam dataset uji MNIST.

```
num_to_plot = 5
for i in range(num_to_plot):
  # Get image and labels from the test data
  img = test_data[i][0]
  label = test_data[i][1]

  # Make prediction on image
  model_pred_logits = model_gpu(img.unsqueeze(dim=0).to(device))
  model_pred_probs = torch.softmax(model_pred_logits, dim=1)
  model_pred_label = torch.argmax(model_pred_probs, dim=1)

  # Plot the image and prediction
  plt.figure()
  plt.imshow(img.squeeze(), cmap="gray")
  plt.title(f"Truth: {label} | Pred: {model_pred_label.cpu().item()}")
  plt.axis(False);
```

Dalam blok kode tersebut, kita melakukan iterasi sebanyak `num_to_plot` untuk menampilkan beberapa contoh gambar dari dataset uji MNIST beserta prediksi yang dihasilkan oleh model. Untuk setiap iterasi, kita mengambil gambar dan label asli dari data uji. Kemudian, kita menggunakan model yang telah dilatih untuk membuat prediksi terhadap gambar tersebut. Hasil prediksi diperoleh dalam bentuk logits, yang kemudian diubah menjadi distribusi probabilitas melalui fungsi softmax. Label prediksi kemudian ditentukan dengan memilih indeks dengan probabilitas tertinggi menggunakan fungsi argmax. Akhirnya, gambar asli dan label prediksi ditampilkan dalam sebuah plot dengan judul yang mencakup label asli dan label prediksi tersebut.

7. Plot a confusion matrix comparing your model's predictions to the truth labels.

```
# See if torchmetrics exists, if not, install it
try:
    import torchmetrics, mlxtend
    print(f"mlxtend version: {mlxtend.__version__}")
    assert int(mlxtend.__version__.split(".")[1]) >= 19, "mlxtend verison should be 0.19.0 or higher"
except:
    !pip install -q torchmetrics -U mlxtend
    import torchmetrics, mlxtend
    print(f"mlxtend version: {mlxtend.__version__}")
```

Dalam blok kode tersebut, kita mencoba untuk mengimpor dua pustaka, yaitu `torchmetrics` dan `mlxtend`. Jika kedua pustaka tersebut sudah terpasang dan versi `mlxtend`nya setidaknya 0.19.0, maka kita akan mencetak versi `mlxtend`. Jika salah satu atau kedua pustaka tersebut belum terpasang atau versi `mlxtend`nya kurang dari 0.19.0, maka kita akan menginstal atau mengupgrade keduanya. Hasil dari versi `mlxtend` yang telah terpasang kemudian dicetak.

```
# Import mlxtend upgraded version
import mlxtend
print(mlxtend.__version__)
assert int(mlxtend.__version__.split(".")[1]) >= 19
```

Dalam blok kode tersebut, kita mengimpor pustaka `mlxtend` dan mencetak versinya. Setelah itu, kita memastikan bahwa versi `mlxtend` yang terpasang setidaknya 0.19.0 dengan

menggunakan pernyataan `assert`. Jika versinya memenuhi syarat, maka kode akan berjalan tanpa masalah. Jika tidak, akan muncul pesan kesalahan yang menunjukkan bahwa versi `mlxtend` tidak memenuhi persyaratan yang dibutuhkan.

```
# Make predictions across all test data
from tqdm.auto import tqdm
model_gpu.eval()
y_preds = []
with torch.inference_mode():
  for batch, (X, y) in tqdm(enumerate(test_dataloader)):
    # Make sure data on right device
    X, y = X.to(device), y.to(device)
    # Forward pass
    y_pred_logits = model_gpu(X)
    # Logits -> Pred probs -> Pred label
    y_pred_labels = torch.argmax(torch.softmax(y_pred_logits, dim=1), dim=1)
    # Append the labels to the preds list
    y_preds.append(y_pred_labels)
  y_preds=torch.cat(y_preds).cpu()
len(y_preds)
```

Dalam blok kode tersebut, kita menggunakan model `model_gpu` yang telah dilatih untuk melakukan prediksi pada seluruh dataset uji (`test_data`). Langkah pertama adalah mengubah model ke mode evaluasi menggunakan `model_gpu.eval()`. Selanjutnya, kita menggunakan loop `for` untuk mengiterasi melalui dataloader uji (`test_dataloader`). Setiap iterasi melibatkan langkah-langkah berikut:

1. Memastikan bahwa data berada di perangkat yang benar (`device`).
2. Melakukan *forward pass* dengan mengirim data melalui model dan memperoleh keluaran logits.
3. Mengonversi logits menjadi probabilitas prediksi menggunakan fungsi softmax.
4. Menentukan label prediksi dengan mengambil argumen dari nilai probabilitas tertinggi.
5. Menambahkan label prediksi ke dalam daftar prediksi (`y_preds`).

Setelah selesai mengiterasi, seluruh prediksi digabungkan menjadi satu tensor menggunakan `torch.cat`, dan tensor tersebut dipindahkan kembali ke CPU agar dapat diakses lebih lanjut. Jumlah prediksi dihitung dengan menggunakan `len(y_preds)`. Proses ini membantu kita memperoleh prediksi model untuk seluruh dataset uji.

```
test_data.targets[:10], y_preds[:10]
```

Dalam blok kode tersebut, kita mencetak sepuluh label target sebenarnya (`test_data.targets[:10]`) dari dataset uji dan sepuluh label prediksi yang dihasilkan oleh model (`y_preds[:10]`). Proses ini memberikan kita gambaran singkat tentang sejauh mana model telah berhasil dalam memprediksi kelas untuk sepuluh sampel pertama dari dataset uji. Dengan membandingkan label target sebenarnya dengan label prediksi, kita dapat mengevaluasi performa model pada sebagian kecil data uji tersebut.

```
from torchmetrics import ConfusionMatrix
from mlxtend.plotting import plot_confusion_matrix

# Setup confusion matrix
confmat = ConfusionMatrix(task="multiclass", num_classes=len(class_names))
confmat_tensor = confmat(preds=y_preds,
                         target=test_data.targets)

# Plot the confusion matrix
fix, ax = plot_confusion_matrix(
    conf_mat=confmat_tensor.numpy(),
    class_names=class_names,
    figsize=(10, 7)
)
```

Dalam blok kode tersebut, kita menggunakan `ConfusionMatrix` dari pustaka `torchmetrics` untuk menghitung matriks kebingungan (confusion matrix) berdasarkan prediksi yang dihasilkan oleh model (`y_preds`) dan label target sebenarnya dari dataset uji (`test_data.targets`). Matriks kebingungan ini memberikan wawasan tentang sejauh mana model dapat mengklasifikasikan instance dari setiap kelas. Selanjutnya, kita menggunakan fungsi `plot_confusion_matrix` dari pustaka `mlxtend` untuk memvisualisasikan matriks kebingungan dalam bentuk grafik. Grafik ini memberikan gambaran yang lebih intuitif tentang seberapa baik model memprediksi setiap kelas pada dataset uji.

8. Create a random tensor of shape [1, 3, 64, 64] and pass it through a nn.Conv2d() layer with various hyperparameter settings (these can be any settings you choose), what do you notice if the kernel_size parameter goes up and down?

```
random_tensor = torch.rand([1, 3, 64, 64])
random_tensor.shape
```

Dalam blok kode di atas, kita membuat tensor acak dengan menggunakan fungsi `torch.rand([1, 3, 64, 64])`. Tensor ini memiliki dimensi `(1, 3, 64, 64)`, yang dapat diartikan sebagai satu batch dari gambar dengan tiga saluran warna (RGB) berukuran 64x64 piksel. Secara lebih spesifik, dimensi pertama adalah dimensi batch dengan ukuran 1, dimensi kedua adalah saluran warna RGB dengan ukuran 3, dan dimensi ketiga dan keempat adalah tinggi dan lebar gambar masing-masing.

```
conv_layer = nn.Conv2d(in_channels=3,
                       out_channels=64,
                       kernel_size=3,
                       stride=2,
                       padding=1)

print(f"Random tensor original shape: {random_tensor.shape}")
random_tensor_through_conv_layer = conv_layer(random_tensor)
print(f"Random tensor through conv layer shape: {random_tensor_through_conv_layer.shape}")
```

Dalam blok kode di atas, kita menggunakan modul convolutional dari PyTorch (`nn.Conv2d`) untuk membuat lapisan konvolusi dengan konfigurasi tertentu. Lapisan konvolusi ini

diterapkan pada tensor acak `random_tensor` yang telah kita buat sebelumnya. Pengaturan konvolusi termasuk jumlah saluran masukan (`in_channels=3`, sesuai dengan saluran warna RGB), jumlah saluran keluaran (`out_channels=64`), ukuran kernel (`kernel_size=3`), langkah konvolusi (`stride=2`), dan padding (`padding=1`).

Hasilnya, `random_tensor` yang awalnya berdimensi `(1, 3, 64, 64)` melalui lapisan konvolusi tersebut berubah menjadi `(1, 64, 32, 32)`. Perubahan dimensi ini disebabkan oleh penggunaan kernel dengan ukuran 3x3 dan langkah konvolusi sebesar 2, yang menghasilkan pengurangan setengah dari tinggi dan lebar gambar. Padding sebesar 1 juga diterapkan untuk menjaga dimensi gambar setelah konvolusi.

9. Use a model similar to the trained model_2 from notebook 03 to make predictions on the test

```
# Download FashionMNIST train & test
from torchvision import datasets
from torchvision import transforms

fashion_mnist_train = datasets.FashionMNIST(root=".",
                                            download=True,
                                            train=True,
                                            transform=transforms.ToTensor())

fashion_mnist_test = datasets.FashionMNIST(root=".",
                                           train=False,
                                           download=True,
                                           transform=transforms.ToTensor())

len(fashion_mnist_train), len(fashion_mnist_test)
```

Dalam blok kode di atas, kita menggunakan modul `datasets` dari torchvision untuk mengunduh dataset FashionMNIST. Pertama, kita mendownload dan membuat dataset FashionMNIST untuk pelatihan dengan parameter `train=True`. Selanjutnya, kita membuat dataset FashionMNIST untuk pengujian dengan parameter `train=False`. Dua dataset tersebut kemudian disimpan dalam variabel `fashion_mnist_train` dan `fashion_mnist_test`. Untuk setiap dataset, kita mengonversi data gambar menjadi tensor menggunakan transformasi `transforms.ToTensor()`. Terakhir, kita menghitung panjang dataset pelatihan dan pengujian menggunakan fungsi `len()` dan mencetak hasilnya.

```
# Get the class names of the Fashion MNIST dataset
fashion_mnist_class_names = fashion_mnist_train.classes
fashion_mnist_class_names
```

Pada blok kode di atas, kita menggunakan atribut `classes` dari dataset FashionMNIST untuk mendapatkan daftar nama kelas yang tersedia. `fashion_mnist_class_names` kemudian berisi daftar ini, yang mencakup kategori pakaian yang terdapat dalam dataset tersebut. Dengan melakukan ini, kita mendapatkan pemahaman awal tentang jenis pakaian yang mungkin terdapat dalam dataset FashionMNIST ini.

```
# Turn FashionMNIST datasets into dataloaders
from torch.utils.data import DataLoader

fashion_mnist_train_dataloader = DataLoader(fashion_mnist_train,
                                            batch_size=32,
                                            shuffle=True)

fashion_mnist_test_dataloader = DataLoader(fashion_mnist_test,
                                           batch_size=32,
                                           shuffle=False)

len(fashion_mnist_train_dataloader), len(fashion_mnist_test_dataloader)
```

Blok kode di atas digunakan untuk mengonversi dataset FashionMNIST ke dalam dataloader menggunakan modul `torch.utils.data.DataLoader`. Dataloaders ini memungkinkan kita untuk memuat data secara efisien dalam bentuk batch selama pelatihan atau pengujian model. `fashion_mnist_train_dataloader` dan `fashion_mnist_test_dataloader` mewakili dataloader untuk data pelatihan dan pengujian FashionMNIST, masing-masing. Pengaturan seperti ukuran batch dan pengacakan telah ditentukan dalam proses pembuatan dataloaders ini. Panjang dari masing-masing dataloader juga dihitung, memberikan kita informasi tentang jumlah batch yang akan digunakan dalam proses pelatihan dan pengujian.

```
# model_2 is the same architecture as MNIST_model
model_2 = MNIST_model(input_shape=1,
                      hidden_units=10,
                      output_shape=10).to(device)
model_2
```

Blok kode di atas membuat instance dari model yang diberi nama `model_2`. Model ini memiliki arsitektur yang sama dengan `MNIST_model`, yang telah didefinisikan sebelumnya. `MNIST_model` adalah model untuk mengklasifikasikan gambar-gambar dari dataset MNIST. Model tersebut memiliki input shape sebesar 1 (karena gambar MNIST adalah gambar grayscale), hidden units sebanyak 10, dan output shape sebanyak 10 sesuai dengan jumlah kelas pada dataset MNIST. Model tersebut kemudian diinisialisasi pada perangkat yang telah ditentukan (dalam hal ini, apakah menggunakan CPU atau GPU). Informasi mengenai struktur dan konfigurasi model dapat dilihat dengan mencetak `model_2`.

```
# Setup loss and optimizer
from torch import nn
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_2.parameters(), lr=0.01)
```

Blok kode di atas menyiapkan fungsi loss dan optimizer untuk model `model_2`. Fungsi loss yang digunakan adalah `CrossEntropyLoss`, yang umumnya digunakan untuk tugas klasifikasi dengan lebih dari dua kelas. Optimizer yang digunakan adalah Stochastic Gradient Descent (SGD) dengan laju pembelajaran (learning rate) sebesar 0.01. Loss function dan

optimizer ini nantinya akan digunakan selama proses pelatihan model untuk menghitung nilai loss dan melakukan optimisasi parameter model.

```
# Setup metrics
from tqdm.auto import tqdm
from torchmetrics import Accuracy

acc_fn = Accuracy(num_classes=len(fashion_mnist_class_names)).to(device)

# Setup training/testing loop
epochs = 5
for epoch in tqdm(range(epochs)):
  train_loss, test_loss_total = 0, 0
  train_acc, test_acc = 0, 0

  ### Training
  model_2.train()
  for batch, (X_train, y_train) in enumerate(fashion_mnist_train_dataloader):
    X_train, y_train = X_train.to(device), y_train.to(device)

    # Forward pass and loss
    y_pred = model_2(X_train)
    loss = loss_fn(y_pred, y_train)
    train_loss += loss
    train_acc += acc_fn(y_pred, y_train)

    # Backprop and gradient descent
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
# Adjust the loss/acc (find the loss/acc per epoch)
train_loss /= len(fashion_mnist_train_dataloader)
train_acc /= len(fashion_mnist_train_dataloader)

### Testing
model_2.eval()
with torch.inference_mode():
  for batch, (X_test, y_test) in enumerate(fashion_mnist_test_dataloader):
    X_test, y_test = X_test.to(device), y_test.to(device)

    # Forward pass and loss
    y_pred_test = model_2(X_test)
    test_loss = loss_fn(y_pred_test, y_test)
    test_loss_total += test_loss

    test_acc += acc_fn(y_pred_test, y_test)

  # Adjust the loss/acc (find the loss/acc per epoch)
  test_loss /= len(fashion_mnist_test_dataloader)
  test_acc /= len(fashion_mnist_test_dataloader)

# Print out what's happening
print(f"Epoch: {epoch} | Train loss: {train_loss:.3f} | Train acc: {train_acc:.2f} | Test loss: {test_loss_total:.3f} | Test acc: {test_acc:.2f}")
```

Blok kode di atas menyiapkan metrik akurasi (`Accuracy`) dan melakukan proses pelatihan dan pengujian (testing) pada model `model_2` menggunakan dataset Fashion MNIST. Loop pelatihan dan pengujian tersebut berlangsung selama lima epoch. Pada setiap epoch, model

diatur ke mode pelatihan (`model_2.train()`) untuk proses pelatihan dan ke mode evaluasi (`model_2.eval()`) untuk pengujian.

Dalam loop pelatihan (`for batch, (X_train, y_train) in enumerate(fashion_mnist_train_dataloader)`), dilakukan iterasi melalui setiap batch dari dataloader pelatihan. Setiap batch dioperasikan pada model (`y_pred = model_2(X_train)`) dan nilai loss dihitung menggunakan fungsi loss `CrossEntropyLoss`. Selain itu, metrik akurasi (`train_acc`) juga dihitung menggunakan objek fungsi akurasi (`acc_fn`). Setelah itu, dilakukan backpropagation dan pengoptimalan menggunakan optimizer SGD (`optimizer`). Hasil akhir dari loss dan akurasi pelatihan per epoch kemudian dihitung.

Proses yang serupa dilakukan pada loop pengujian (`for batch, (X_test, y_test) in enumerate(fashion_mnist_test_dataloader)`), dengan memperhitungkan nilai loss dan akurasi pengujian. Hasil dari loss dan akurasi pelatihan dan pengujian kemudian dicetak untuk setiap epoch.

```
# Make predictions with trained model_2
test_preds = []
model_2.eval()
with torch.inference_mode():
  for X_test, y_test in tqdm(fashion_mnist_test_dataloader):
    y_logits = model_2(X_test.to(device))
    y_pred_probs = torch.softmax(y_logits, dim=1)
    y_pred_labels = torch.argmax(y_pred_probs, dim=1)
    test_preds.append(y_pred_labels)
test_preds = torch.cat(test_preds).cpu() # matplotlib likes CPU
test_preds[:10], len(test_preds)
```

Blok kode di atas digunakan untuk membuat prediksi dengan model `model_2` yang telah dilatih. Proses ini dilakukan pada dataset pengujian Fashion MNIST (`fashion_mnist_test_dataloader`). Selama iterasi melalui dataloader pengujian, dilakukan prediksi pada setiap batch dengan menggunakan model `model_2`.

Output dari model berupa logits diubah menjadi probabilitas prediksi dengan fungsi softmax, dan kemudian dilakukan pemilihan label prediksi dengan mengambil indeks argmax dari probabilitas tersebut. Hasil prediksi tersebut kemudian disusun menjadi satu tensor menggunakan `torch.cat`. Hasil akhir berupa tensor `test_preds` yang berisi label prediksi untuk seluruh dataset pengujian.

```
# Get wrong prediction indexes
import numpy as np
wrong_pred_indexes = np.where(test_preds != fashion_mnist_test.targets)[0]
len(wrong_pred_indexes)
```

Blok kode di atas digunakan untuk mendapatkan indeks dari prediksi yang salah pada dataset Fashion MNIST. Ini dilakukan dengan membandingkan label prediksi (`test_preds`) dengan label sebenarnya dari dataset pengujian Fashion MNIST (`fashion_mnist_test.targets`).

Menggunakan fungsi `np.where`, kita mendapatkan indeks di mana prediksi tidak sesuai dengan label sebenarnya. Hasilnya adalah array `wrong_pred_indexes` yang berisi indeks-indeks dari prediksi yang salah pada dataset pengujian Fashion MNIST. Panjang array tersebut memberikan jumlah total prediksi yang salah.

```
# Select random 9 wrong predictions and plot them
import random
random_selection = random.sample(list(wrong_pred_indexes), k=9)

plt.figure(figsize=(10, 10))
for i, idx in enumerate(random_selection):
  # Get true and pred labels
  true_label = fashion_mnist_class_names[fashion_mnist_test[idx][1]]
  pred_label = fashion_mnist_class_names[test_preds[idx]]

  # Plot the wrong prediction with its original label
  plt.subplot(3, 3, i+1)
  plt.imshow(fashion_mnist_test[idx][0].squeeze(), cmap="gray")
  plt.title(f"True: {true_label} | Pred: {pred_label}", c="r")
  plt.axis(False);
```

Blok kode di atas digunakan untuk memilih secara acak 9 prediksi yang salah dari dataset Fashion MNIST dan menampilkannya dalam bentuk gambar dengan label yang benar dan label prediksi. `random_selection` mengandung indeks-indeks dari prediksi yang salah yang dipilih secara acak dari array `wrong_pred_indexes`. Selanjutnya, menggunakan loop, setiap prediksi yang salah diproses untuk ditampilkan dalam subplot 3x3 pada gambar dengan menampilkan gambar tersebut, label yang sebenarnya, dan label prediksi yang diberi warna merah sebagai penanda bahwa prediksinya salah.