

Syifa Wanda Isnaini

1103201248

Machine Learning

03. PyTorch Computer Vision Exercise

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

Sat Apr 16 03:23:02 2022

NVIDIA-SMI 460.32.03 Driver Version: 460.32.03 CUDA Version: 11.2									
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC		
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.	MIG M.		
0	Tesla P100-PCIE...	Off	00000000:00:04.0	Off			0		
N/A	39C	P0	29W / 250W	0MiB / 16280MiB	0%	Default	N/A		

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	
ID	ID					Usage	
No running processes found							

Kode `!nvidia-smi` digunakan untuk memeriksa informasi tentang perangkat GPU yang tersedia di sistem. Ini biasanya digunakan di lingkungan yang mendukung GPU, seperti Google Colab atau mesin yang memiliki GPU Nvidia.

Jika perangkat memiliki GPU dan perangkat lunak Nvidia System Management Interface (`nvidia-smi`) terpasang, perintah ini akan menampilkan informasi tentang GPU yang terpasang, termasuk penggunaan memori, temperatur, dan utilitas GPU lainnya.

Namun, perintah ini mungkin tidak berfungsi di semua lingkungan atau sistem yang tidak memiliki GPU Nvidia, dan bisa menghasilkan pesan kesalahan jika perangkat atau perangkat lunak yang diperlukan tidak tersedia.

```
# Import torch
import torch

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

# TODO: Setup device agnostic code
```

1.10.0+cu111
'cuda'

Kode di atas memeriksa versi PyTorch yang diinstal pada sistem dengan mencetak versi menggunakan `torch.version`.

Selanjutnya, perlu dilakukan pengaturan kode yang bersifat agnostik perangkat (device agnostic). Kode agnostik perangkat biasanya mengidentifikasi apakah GPU (cuda) tersedia atau tidak, dan kemudian memilih perangkat yang sesuai (cuda jika tersedia, dan cpu jika tidak). Kode di atas menggunakan `torch.cuda.is_available()` untuk memeriksa ketersediaan GPU. Jika GPU tersedia, variabel device diatur sebagai "cuda", jika tidak, diatur sebagai "cpu". Hal ini memungkinkan kode untuk berjalan di GPU jika tersedia, atau kembali ke CPU jika tidak.

1. What are 3 areas in industry where computer vision currently being used?

Visi komputer adalah bidang yang berkembang pesat dengan berbagai aplikasi di berbagai industri. Berikut adalah tiga area di mana visi komputer saat ini digunakan dalam industri:

Kesehatan:

Pemrosesan Citra Medis: Visi komputer digunakan untuk menganalisis citra medis seperti sinar-X, MRI, dan CT scan. Ini membantu dalam deteksi dan diagnosis penyakit, tumor, dan kelainan. Diagnostik: Sistem visi komputer digunakan untuk tujuan diagnostik, termasuk mengidentifikasi kondisi kulit, mendeteksi retinopati diabetik, dan membantu dalam analisis patologi. Manufaktur:

Kontrol Kualitas: Visi komputer diterapkan untuk memeriksa dan memastikan kualitas produk yang diproduksi di jalur produksi. Ini dapat mendeteksi cacat, mengukur dimensi, dan mengidentifikasi penyimpangan dari spesifikasi. Robotika: Sistem visi memungkinkan robot untuk menangkap dan menjelajahi lingkungan mereka. Hal ini penting dalam tugas seperti operasi pilih dan letak, perakitan, dan robotika kolaboratif. Kendaraan Otonom:

Mobil Otonom: Visi komputer memainkan peran sentral dalam kendaraan otonom, membantu mereka memahami lingkungan dan membuat keputusan real-time. Ini melibatkan tugas seperti deteksi objek, pelacakan jalur, dan menghindari rintangan. Sistem Transportasi: Selain kendaraan otonom, visi komputer digunakan dalam pemantauan lalu lintas, pengenalan pelat nomor, dan sistem manajemen lalu lintas pintar. Aplikasi ini hanya mewakili sebagian kecil dari dampak luas yang dimiliki visi komputer di berbagai industri. Sektor lainnya, termasuk ritel, pertanian, keamanan, dan hiburan, juga memanfaatkan teknologi visi komputer untuk berbagai tujuan.

2. Search 'what is overfitting in machine learning' and write down a sentence about what you find

Overfitting dalam machine learning terjadi ketika model yang telah dihasilkan terlalu sesuai dengan data pelatihan, sehingga kinerjanya menurun saat diuji dengan data baru karena model tersebut telah "menghafal" data pelatihan daripada memahami pola umum yang dapat diterapkan pada data baru.

3. Search "ways to prevent overfitting in machine learning", write down 3 of the things you find and a sentence about each.

Penggunaan Data Validasi: Menggunakan dataset validasi terpisah selama pelatihan untuk memonitor kinerja model pada data yang tidak digunakan selama proses pelatihan, sehingga dapat mendeteksi tanda-tanda overfitting dan mengambil tindakan pencegahan.

Pengurangan Kompleksitas Model: Mengurangi kompleksitas model dengan mengurangi jumlah parameter atau lapisan dalam arsitektur, sehingga model tidak terlalu "menghafal" data pelatihan dan lebih mampu menggeneralisasi pola umum.

Augmentasi Data: Menggunakan teknik augmentasi data seperti rotasi, pembesaran, atau pemangkasan untuk menciptakan variasi dalam dataset pelatihan, membantu model memahami variasi yang mungkin terjadi pada data baru.

4. Spend 20-minutes reading and clicking through the [CNN Explainer website](#).

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

```
import torch
import torchvision
from torchvision import datasets
from torchvision import transforms
import matplotlib.pyplot as plt

# Get the MNIST train dataset
train_data = datasets.MNIST(root=".",
                             train=True,
                             download=True,
                             transform=transforms.ToTensor()) # do we want to transform the data as we download
it?

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

train_data, test_data
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./MNIST/raw/train-images-idx3-ubyte.gz
100%|#####| 9912422/9912422 [00:00<00:00, 254834329.84it/s]Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./MNIST/raw/train-labels-idx1-ubyte.gz
100%|#####| 28881/28881 [00:00<00:00, 58547942.88it/s]
Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./MNIST/raw/t10k-images-idx3-ubyte.gz
100%|#####| 1648877/1648877 [00:00<00:00, 171130363.91it/s]Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|#####| 4542/4542 [00:00<00:00, 19927331.35it/s]
Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw

(Dataset MNIST
  Number of datapoints: 60000
  Root location: .
  Split: Train
  StandardTransform
  Transform: ToTensor(),
Dataset MNIST
  Number of datapoints: 10000
  Root location: .
  Split: Test
  StandardTransform
  Transform: ToTensor())
```

torch: Library utama PyTorch.

torchvision: Library PyTorch khusus untuk pengolahan citra dan pemrosesan data dalam machine learning.

datasets: Sub-modul di torchvision untuk mengakses berbagai dataset.

transforms: Sub-modul di torchvision untuk melakukan transformasi data pada citra.

matplotlib.pyplot as plt: Library untuk plotting grafik, digunakan untuk menampilkan beberapa gambar dari dataset.

datasets.MNIST: Fungsi untuk mengakses dataset MNIST.

root=".": Lokasi di mana dataset akan diunduh atau sudah ada.

train=True: Mengambil bagian dari dataset yang digunakan untuk pelatihan.

download=True: Mendownload dataset jika belum ada di lokasi yang ditentukan.

transform=transforms.ToTensor(): Mengubah gambar menjadi representasi tensor PyTorch.

Parameter dan argumen mirip dengan train_data, namun train=False menunjukkan bahwa ini adalah dataset pengujian.

Baris ini hanya menampilkan objek dataset pelatihan dan pengujian. Ketika kode ini dijalankan, kita akan melihat cuplikan dari dataset pelatihan dan pengujian.

```
# 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")
```

```
Image shape: torch.Size([1, 28, 28]) -> [color_channels, height, width] (CHW)
Label: 5 -> no shape, due to being integer
```

```
print(f"Image shape: {img.shape} -> [color_channels, height, width] (CHW)");
```

`img.shape`: Mencetak bentuk tensor gambar `img`.

`[color_channels, height, width]`: Ini memberikan informasi tentang dimensi atau bentuk dari gambar.

`"(CHW)"`: Menandakan bahwa urutan dimensi yang dicetak adalah Channel-Height-Width, yang merupakan format umum untuk data citra dalam PyTorch. Artinya, gambar direpresentasikan sebagai tensor 3D dengan dimensi warna (channel), tinggi, dan lebar.

```
print(f"Label: {label} -> no shape, due to being integer");
```

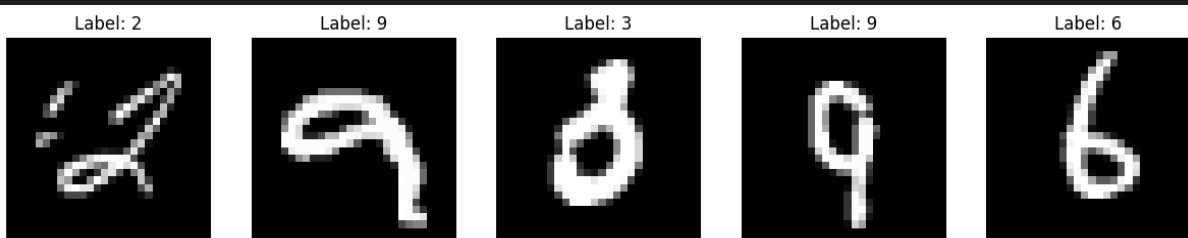
`label`: Mencetak label yang terkait dengan gambar.

`"no shape, due to being integer"`: Label adalah bilangan bulat tunggal yang menunjukkan kelas atau kategori dari gambar tersebut. Karena label hanya merupakan bilangan bulat, tidak ada bentuk (shape) yang terkait dengan label, dan oleh karena itu, pesan ini memberi tahu bahwa label tidak memiliki dimensi.

Dengan mencetak bentuk gambar dan label, ini membantu untuk memahami struktur dasar dari dataset MNIST. Gambar direpresentasikan sebagai tensor dengan tiga dimensi (CHW), sementara labelnya hanya merupakan bilangan bulat

6. 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);
```



```
train_dataset = MNIST(root='./data', train=True, transform=transform, download=True):
```

Menginisialisasi dataset pelatihan MNIST.

`indices = torch.randint(0, len(train_dataset), (5,))`: Memilih 5 indeks acak dari dataset.

Membuat subplot dan loop melalui indeks untuk menampilkan gambar dan labelnya.

Kode ini akan menampilkan lima sampel gambar dari dataset pelatihan MNIST beserta labelnya.

```
# 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)

train_dataloader, test_dataloader
```

(<torch.utils.data.dataloader.Dataloader at 0x7b447d6c4cd0>,
<torch.utils.data.dataloader.Dataloader at 0x7b447d6c4e80>)

`Dataloader`: Kelas dari PyTorch yang menyediakan fungsionalitas untuk memuat data dalam batch.

`dataset=train_data`: Menentukan dataset yang akan digunakan, yaitu `train_data` yang telah diinisialisasi sebelumnya.

`batch_size=32`: Menentukan ukuran batch, yaitu jumlah sampel data yang akan diproses dalam satu iterasi. Dalam hal ini, ukuran batchnya adalah 32.

`shuffle=True`: Jika diatur ke `True`, data dalam dataset akan diacak setiap kali iterasi dimulai. Ini membantu model untuk melihat variasi yang berbeda dalam setiap epoch selama pelatihan.

`dataset=test_data`: Menentukan dataset yang akan digunakan untuk pengujian, yaitu `test_data`.

`batch_size=32`: Seperti sebelumnya, menentukan ukuran batch untuk data pengujian.

`shuffle=False`: Tidak perlu mengacak data pada dataset pengujian karena urutan data tidak mempengaruhi evaluasi model. Pengacakan data umumnya hanya dilakukan pada dataset pelatihan.

Setelah dibuat, kedua data loader (`train_dataloader` dan `test_dataloader`) dapat digunakan untuk iterasi melalui batch-batch data saat melatih dan menguji model, membuat proses tersebut lebih efisien dan mudah dikelola.

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

torch.Size([32, 1, 28, 28])
torch.Size([32])

Kode di bawah ini digunakan untuk mengiterasi melalui satu batch dari dataloader pelatihan dan mencetak bentuk (`shape`) setiap sampel

Hasil runningnya akan mencetak bentuk (`shape`) dari setiap sampel dalam satu batch dari dataloader pelatihan. Perlu diingat bahwa setiap sampel dalam batch akan memiliki bentuk yang sama, dan bentuknya akan mencerminkan dimensi dari data gambar dan label.

Dalam kasus ini, `torch.Size([32, 1, 28, 28])` menunjukkan bahwa batch tersebut terdiri dari 32 gambar dengan 1 channel warna (grayscale) masing-masing, dan setiap gambar memiliki ukuran 28x28 piksel.

`torch.Size([32])` menunjukkan bahwa ada 32 label yang sesuai dengan setiap gambar dalam batch.

```
len(train_dataloader), len(test_dataloader)
(1875, 313)
```

Kode `len(train_dataloader)` dan `len(test_dataloader)` digunakan untuk mengukur jumlah batch (kelompok) data dalam masing-masing data loader (`train_dataloader` dan `test_dataloader`)

Ini mengembalikan jumlah batch data dalam data loader pelatihan (`train_dataloader`).

Angka ini akan bergantung pada jumlah data pelatihan, ukuran batch yang digunakan, dan apakah pengacakan dilakukan (`shuffle=True`).

Hasilnya memberikan informasi tentang seberapa banyak iterasi pelatihan yang akan dilakukan pada satu epoch.

Ini mengembalikan jumlah batch data dalam data loader pengujian (`test_dataloader`).

Seperti sebelumnya, angka ini tergantung pada jumlah data pengujian, ukuran batch yang digunakan, dan pengaturan lainnya.

Hasilnya memberikan informasi tentang seberapa banyak iterasi pengujian yang akan dilakukan.

Dalam konteks pembelajaran mesin, lebih umum jika jumlah batch dalam satu epoch didefinisikan sebagai "langkah" atau "iterasi". Oleh karena itu, jumlah batch dalam satu epoch dapat dihitung sebagai `len(train_dataloader)` atau `len(test_dataloader)` tergantung pada apakah sedang dilakukan pelatihan atau pengujian.

8. Recreate model_2 used in notebook 03

```
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

```

conv_block_1 dan conv_block_2 (Convolutional Blocks):

Dua blok konvolusi yang terdiri dari lapisan-lapisan konvolusi 2D, aktivasi ReLU, dan max pooling. Masing-masing blok bertujuan untuk mengekstraksi fitur-fitur penting dari gambar.

classifier (Fully Connected Classifier):

Blok yang terdiri dari lapisan flatten untuk meratakan output dari blok konvolusi menjadi vektor satu dimensi, diikuti oleh lapisan linear untuk menghasilkan output kelas.

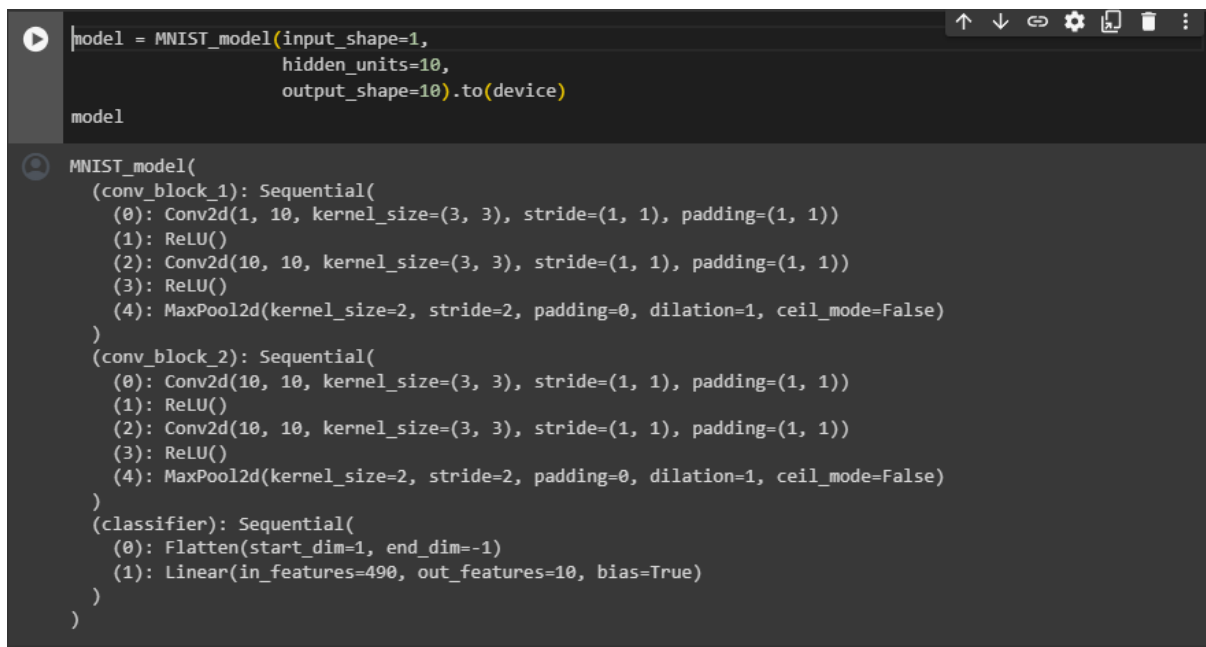
forward method:

Metode yang mendefinisikan urutan proses pengolahan data saat melewati model.

Input x melewati conv_block_1, conv_block_2, dan classifier secara berurutan.

Output dari model adalah hasil klasifikasi.

Model ini menggunakan dua blok konvolusi untuk mengekstraksi fitur-fitur citra, diikuti oleh sebuah lapisan linear untuk melakukan klasifikasi pada dataset MNIST. Model tersebut dirancang untuk menerima gambar dengan bentuk (input_shape, height, width) dan menghasilkan output kelas dengan bentuk (output_shape).



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

model

MNIST_model(
  (conv_block_1): Sequential(
    (0): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=490, out_features=10, bias=True)
  )
)
```

Kode tersebut membuat sebuah objek model dari kelas MNIST_model dengan parameter-parameter tertentu dan memindahkan model ke perangkat yang ditentukan (biasanya CPU atau GPU).

MNIST_model(input_shape=1, hidden_units=10, output_shape=10):

Membuat objek model dari kelas MNIST_model dengan parameter:

input_shape: Jumlah saluran warna (channel) pada gambar. Dalam konteks MNIST, ini biasanya 1 karena gambar grayscale.

hidden_units: Jumlah unit dalam lapisan-lapisan konvolusi dan fully connected.

output_shape: Jumlah kelas yang ingin diprediksi. Dalam kasus MNIST, ada 10 kelas digit (0-9).

.to(device):

Memindahkan model ke perangkat yang ditentukan. device di sini harus telah ditentukan sebelumnya, misalnya, dengan menggunakan `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`. Ini memungkinkan model untuk diletakkan di GPU jika GPU tersedia atau di CPU jika tidak.

model:

Ini adalah objek model yang telah dibuat dan disiapkan untuk pelatihan atau pengujian.

Dengan langkah-langkah ini, model siap digunakan dan dapat digunakan untuk pelatihan atau pengujian pada dataset MNIST. Model ini akan dapat memproses data input, melakukan propagasi maju (forward pass), dan menghasilkan prediksi.

```
[ ] # 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)

tensor([[ 0.0169, -0.0080,  0.0213, -0.0463, -0.0163,  0.0157, -0.0268,  0.0421,
         -0.0124, -0.0211]], device='cuda:0', grad_fn=<AddmmBackward0>)

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

torch.Size([1, 10, 7, 7])

[ ] flatten_layer = nn.Flatten()
flatten_layer(dummy_x_2).shape

torch.Size([1, 490])
```

`dummy_x`: Membuat tensor acak dengan ukuran (1, 28, 28), yang mencirikan gambar dummy dengan satu saluran warna (grayscale) dan ukuran 28x28 piksel.

`.unsqueeze(dim=0)`: Menambahkan dimensi batch ke tensor, sehingga bentuknya menjadi (1, 1, 28, 28).

`.to(device)`: Memindahkan tensor ke perangkat yang ditentukan.

`model(dummy_x)`: Melakukan forward pass dengan model pada gambar dummy. Ini memberikan output dari model untuk gambar dummy yang diberikan.

`dummy_x_2`: Membuat tensor acak dengan ukuran (1, 10, 7, 7), yang mencirikan tensor dummy dengan satu batch, sepuluh saluran, dan ukuran 7x7 piksel untuk setiap saluran.

`dummy_x_2.shape`: Mengembalikan bentuk tensor `dummy_x_2`.

`flatten_layer(dummy_x_2)`: Menerapkan lapisan flatten ke tensor `dummy_x_2`. Lapisan ini meratakan tensor menjadi satu dimensi.

`.shape`: Mengembalikan bentuk tensor setelah melewati lapisan flatten.

Secara keseluruhan, kode tersebut berguna untuk memahami bentuk data pada beberapa tahap proses, terutama setelah melalui lapisan flatten. Ini dapat membantu dalam menentukan ukuran input dan output yang diperlukan oleh model saat pelatihan atau pengujian.

9. Train the model you built in exercise 8. For 5 epoch 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}")

```

```

100% ██████████ 5/5 [03:28<00:00, 41.05s/it]
Epoch: 0 | Loss: 0.367 | Test loss: 0.062
Epoch: 1 | Loss: 0.069 | Test loss: 0.059
Epoch: 2 | Loss: 0.053 | Test loss: 0.042
Epoch: 3 | Loss: 0.044 | Test loss: 0.035
Epoch: 4 | Loss: 0.039 | Test loss: 0.039
CPU times: user 3min 18s, sys: 3.32 s, total: 3min 21s
Wall time: 3min 28s

```

Inisialisasi Model dan Optimizer:

model_cpu: Model yang akan dilatih, dipindahkan ke perangkat CPU.

loss_fn: Fungsi kerugian (cross-entropy loss).

optimizer: Optimizer SGD (Stochastic Gradient Descent) untuk mengoptimalkan model.

Training Loop:

Melakukan loop sebanyak epochs (5 kali dalam contoh ini).

Untuk setiap epoch, melakukan loop melalui data pelatihan (train_dataloader).

Melakukan forward pass, menghitung loss, melakukan backpropagation, dan melakukan update parameter model menggunakan optimizer.

Mencetak loss pelatihan setiap epoch.

Testing Loop:

Setelah selesai epoch pelatihan, melakukan loop melalui data pengujian (test_dataloader).

Mengevaluasi model pada data pengujian dan menghitung loss pengujian.

Mencetak loss pengujian setiap epoch.

Tambahan:

Pengaturan model ke mode evaluasi (model_cpu.eval()) saat pengujian.

Penggunaan torch.inference_mode() untuk mematikan fitur yang tidak diperlukan saat inference, meningkatkan efisiensi.

Print Out:

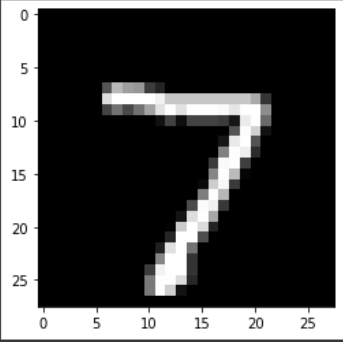
Mencetak informasi setiap epoch, termasuk nilai loss pelatihan dan pengujian.

Waktu Eksekusi:

Menggunakan %%time untuk mengukur waktu eksekusi dari seluruh cell kode.

10. Make prediction using your trained model and visualized at least 5 of them comparing the prediction to the target label

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

<matplotlib.image.AxesImage at 0x7f77f0022190>


[ ] # 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
model_pred_probs = torch.softmax(model_pred_logits, dim=1)
model_pred_label = torch.argmax(model_pred_probs, dim=1)
model_pred_label

tensor([7], device='cuda:0')
```

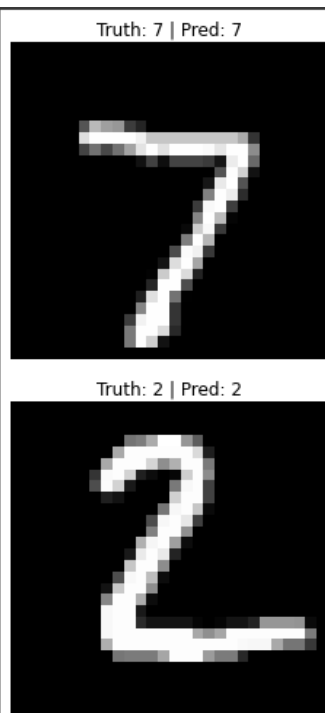
```

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);

```



`plt.imshow(test_data[0][0].squeeze(), cmap="gray")`: Menampilkan gambar pertama dari dataset pengujian menggunakan matplotlib. Gambar ini digunakan untuk memberikan contoh visual dari data.

`model_pred_logits = model_gpu(test_data[0][0].unsqueeze(dim=0).to(device))`: Melakukan prediksi pada gambar pertama dari dataset pengujian menggunakan model yang telah dilatih (`model_gpu`). Pastikan gambar memiliki bentuk yang sesuai dan berada pada perangkat yang benar (GPU).

`model_pred_probs = torch.softmax(model_pred_logits, dim=1)`: Mengonversi logits (keluaran model sebelum fungsi softmax) menjadi probabilitas menggunakan fungsi softmax.

`model_pred_label = torch.argmax(model_pred_probs, dim=1)`: Mengambil label prediksi dengan memilih kelas dengan probabilitas tertinggi.

`num_to_plot = 5`: Menentukan jumlah gambar dari dataset pengujian yang akan ditampilkan.

`for i in range(num_to_plot):` Melakukan loop sebanyak `num_to_plot`.

`img = test_data[i][0]:` Mendapatkan gambar ke-i dari dataset pengujian.

`label = test_data[i][1]:` Mendapatkan label yang sesuai dengan gambar.

`model_pred_label = torch.argmax(model_pred_probs, dim=1):` Mengambil label prediksi menggunakan model yang telah dilatih.

`plt.figure():` Membuat gambar (figure) baru untuk setiap iterasi.

`plt.imshow(img.squeeze(), cmap="gray"):` Menampilkan gambar dari dataset pengujian.

`plt.title(f"Truth: {label} | Pred: {model_pred_label.cpu().item()}"):` Menambahkan judul pada gambar yang berisi label sebenarnya (Truth) dan label prediksi (Pred).

`plt.axis(False):` Menghilangkan sumbu gambar untuk tampilan yang lebih bersih.

11. Plot a confusion matrix comparing your model's prediction 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 version should be 0.19.0 or higher"
except:
    !pip install -q torchmetrics -U mlxtend # <- Note: If you're using Google Colab, this may require restart
    import torchmetrics, mlxtend
    print(f"mlxtend version: {mlxtend.__version__}")

mlxtend version: 0.19.0

[ ] # Import mlxtend upgraded version
import mlxtend
print(mlxtend.__version__)
assert int(mlxtend.__version__.split(".")[1]) >= 19 # should be version 0.19.0 or higher

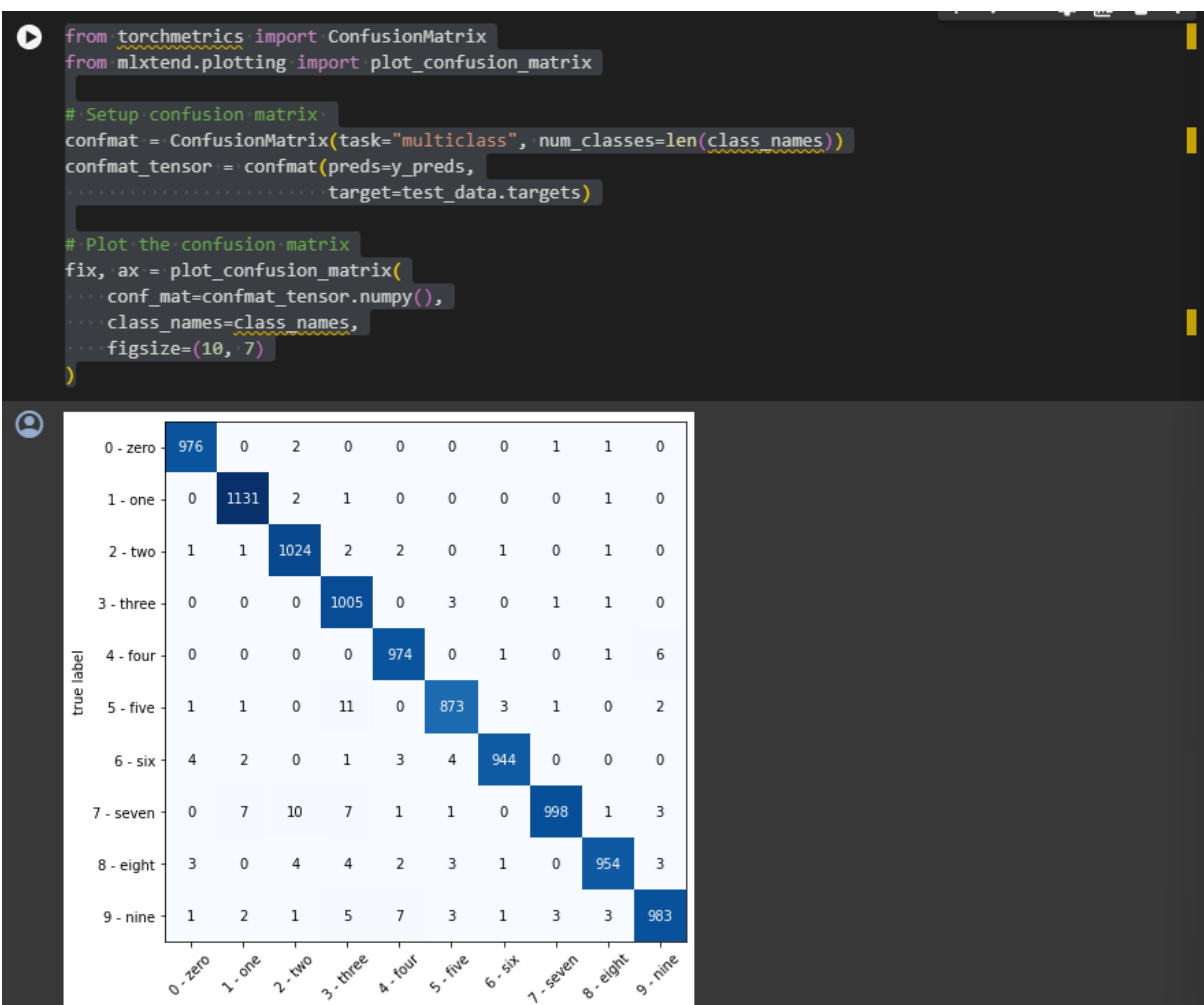
0.19.0
```

```
# 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)
```

313/? [00:01<00:00, 322.62it/s]
10000

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

(tensor([7, 2, 1, 0, 4, 1, 4, 9, 5, 9]),
tensor([7, 2, 1, 0, 4, 1, 4, 9, 5, 9]))



Blok ini mencoba mengimpor modul torchmetrics dan mlxtend. Jika torchmetrics atau mlxtend belum diinstal atau versinya kurang dari 0.19.0, maka instal modul-modul tersebut menggunakan perintah pip install.

Blok ini mengimpor modul `mlxtend` dan memeriksa versi. Jika versinya kurang dari 0.19.0, maka kode akan menampilkan pesan kesalahan. Ini memastikan bahwa versi yang digunakan dari `mlxtend` memenuhi persyaratan minimum.

Blok ini melakukan prediksi pada seluruh data pengujian menggunakan model yang telah dilatih (`model_gpu`).

Hasil prediksi (`y_pred_labels`) diambil sebagai argumen dengan nilai tertinggi dari fungsi `softmax` pada logits model.

Prediksi tersebut kemudian di-append ke dalam list `y_preds` untuk selanjutnya dikonkatenasi menjadi tensor satu dimensi.

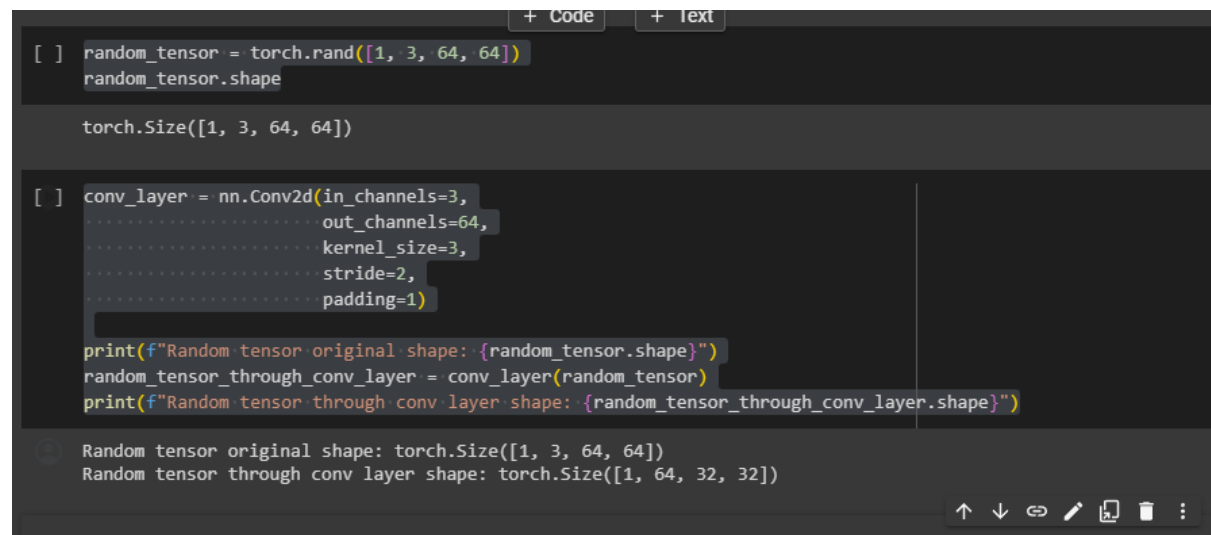
Panjang tensor `y_preds` dicetak, memberikan informasi tentang jumlah prediksi yang dihasilkan.

Blok ini mencetak 10 label sebenarnya (`test_data.targets[:10]`) dan 10 label prediksi pertama (`y_preds[:10]`) untuk memeriksa hasil prediksi secara langsung.

Blok ini mengimpor `ConfusionMatrix` dari modul `torchmetrics` untuk menghitung matriks kebingungan. Matriks kebingungan dihitung menggunakan label prediksi (`y_preds`) dan label sebenarnya (`test_data.targets`).

Matriks tersebut kemudian diplot menggunakan fungsi `plot_confusion_matrix` dari `mlxtend`. Dimensi dan label kelas digunakan untuk memberikan informasi visual yang lebih baik.

Hasilnya adalah matriks kebingungan yang memberikan gambaran tentang sejauh mana model dapat memprediksi dengan benar setiap kelas pada dataset pengujian MNIST.



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

    torch.Size([1, 3, 64, 64])

[ ] 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}")

    Random tensor original shape: torch.Size([1, 3, 64, 64])
    Random tensor through conv layer shape: torch.Size([1, 64, 32, 32])
```

Membuat tensor acak dengan bentuk (shape) `[1, 3, 64, 64]`. Ini menunjukkan tensor dengan batch size 1, 3 saluran warna (RGB), dan dimensi gambar 64x64 piksel.

Mendefinisikan lapisan konvolusi (`nn.Conv2d`) dengan parameter sebagai berikut:

`in_channels=3`: Menunjukkan jumlah saluran warna pada input, yang sesuai dengan jumlah saluran warna pada tensor acak (`random_tensor`).

`out_channels=64`: Menunjukkan jumlah saluran output atau filter yang akan diterapkan oleh konvolusi.

`kernel_size=3`: Menunjukkan ukuran kernel konvolusi (3x3 kernel).

stride=2: Menunjukkan langkah (stride) pergeseran kernel saat bergerak melintasi gambar.

padding=1: Menunjukkan jumlah padding yang ditambahkan di sekitar gambar.

Mencetak bentuk (shape) dari tensor acak sebelum melewati lapisan konvolusi. Ini memberikan informasi tentang bentuk input yang diberikan kepada lapisan konvolusi.

Meneruskan tensor acak melalui lapisan konvolusi menggunakan metode `conv_layer`. Ini akan menghasilkan tensor yang mengalami transformasi oleh lapisan konvolusi.

Mencetak bentuk (shape) dari tensor setelah melewati lapisan konvolusi. Ini memberikan informasi tentang bagaimana bentuk tensor berubah setelah konvolusi. Perubahan ini dipengaruhi oleh parameter seperti jumlah saluran output, ukuran kernel, langkah, dan padding yang diberikan pada lapisan konvolusi.

```
# 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)

(60000, 10000)

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

['T-shirt/top',
 'Trouser',
 'Pullover',
 'Dress',
 'Coat',
 'Sandal',
 'Shirt',
 'Sneaker',
 'Bag',
 'Ankle boot']

# 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)

(1875, 313)
```

```

# 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

MNIST_model(
  (conv_block_1): Sequential(
    (0): Conv2d(1, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=490, out_features=10, bias=True)
  )
)

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

```

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}")

```

100%  5/5 [01:14<00:00, 14.37s/it]

Epoch: 0	Train loss: 1.186	Train acc: 0.57	Test loss: 197.707	Test acc: 0.77
Epoch: 1	Train loss: 0.543	Train acc: 0.80	Test loss: 152.445	Test acc: 0.83
Epoch: 2	Train loss: 0.454	Train acc: 0.84	Test loss: 141.563	Test acc: 0.84
Epoch: 3	Train loss: 0.411	Train acc: 0.85	Test loss: 128.963	Test acc: 0.86
Epoch: 4	Train loss: 0.386	Train acc: 0.86	Test loss: 126.913	Test acc: 0.86

```
[ ] # 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)
```

100%  313/313 [00:00<00:00, 325.62it/s]

(tensor([9, 2, 1, 1, 6, 1, 4, 6, 5, 7]), 10000)

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

1425

```
# 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);
```



Mengunduh dan menyiapkan dataset FashionMNIST untuk pelatihan dan pengujian.

`transforms.ToTensor()` digunakan untuk mengubah gambar menjadi tensor.

Membuat `DataLoader` untuk dataset pelatihan dan pengujian, memungkinkan pemrosesan data dalam batch.

Membuat model (`model_2`) dengan arsitektur yang sama seperti `MNIST_model`.

Menggunakan `CrossEntropyLoss` sebagai fungsi kerugian dan `SGD` sebagai optimizer.

Melakukan pelatihan dan pengujian model selama beberapa epoch.

Mencetak hasil pelatihan dan pengujian seperti loss dan akurasi setiap epoch.

Melakukan prediksi pada dataset pengujian menggunakan model yang telah dilatih.

Memvisualisasikan beberapa prediksi yang salah dari dataset pengujian dalam bentuk gambar.

Menandai prediksi yang salah dengan warna merah pada judul gambar.

Kode ini menyajikan langkah-langkah utama untuk pelatihan, pengujian, dan visualisasi hasil dari model yang telah dilatih pada dataset FashionMNIST.