

Nama : Andhika Aria Pratama N

NIM : 1103202121

3 PyTorch Computer Vision

- Perpustakaan visi komputer di PyTorch

Dalam konteks PyTorch, modul dan kelas yang disebutkan memiliki fungsi sebagai berikut:

Torchvision: Berisi fungsionalitas penting untuk tugas visi komputer. Ini mencakup dataset, arsitektur model, dan transformasi gambar yang umum digunakan dalam masalah visi komputer.

- `torchvision.datasets`: Modul ini menyediakan berbagai dataset contoh untuk berbagai tugas visi komputer, termasuk klasifikasi gambar, deteksi objek, captioning gambar, klasifikasi video, dan lainnya. Ini juga mencakup kelas dasar yang memudahkan pembuatan dataset kustom.
 - `torchvision.models`: Modul ini berisi arsitektur model visi komputer yang terkenal dan sering digunakan yang diimplementasikan di PyTorch. Pengguna dapat memanfaatkan model yang sudah diimplementasikan ini untuk masalah visi komputer mereka sendiri.
 - `torchvision.transforms`: Gambar sering perlu diubah, diproses, atau diperbesar sebelum digunakan dengan model. Transformasi gambar umum tersedia di modul ini, menawarkan cara yang nyaman untuk menyiapkan gambar untuk input model.
 - `torch.utils.data.Dataset`: Kelas ini berfungsi sebagai kelas dasar dataset dalam PyTorch. Ini menyediakan abstraksi untuk membuat dataset kustom, memungkinkan pengguna untuk mengorganisir dan menyusun data mereka untuk pelatihan atau evaluasi.
 - `torch.utils.data.DataLoader`: Kelas ini digunakan untuk membuat iterable Python atas suatu dataset, biasanya dataset yang dibuat dengan `torch.utils.data.Dataset`. Ini memungkinkan pemuatan dan pengelompokan data yang efisien selama proses pelatihan atau evaluasi.
- Mengambil dataset

```

# Setup training data
train_data = datasets.FashionMNIST(
    root="data", # where to download data to?
    train=True, # get training data
    download=True, # download data if it doesn't exist on disk
    transform=ToTensor(), # images come as PIL format, we want to turn into Torch tensors
    target_transform=None # you can transform labels as well
)

# Setup testing data
test_data = datasets.FashionMNIST(
    root="data",
    train=False, # get test data
    download=True,
    transform=ToTensor()
)

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
100%|#####| 26421880/26421880 [00:02:00:00, 9851480.88it/s]
Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
100%|#####| 29515/29515 [00:00:00:00, 177271.99it/s]
Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
100%|#####| 4422102/4422102 [00:01:00:00, 3278351.96it/s]
Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
100%|#####| 5148/5148 [00:00:00:00, 2237773.55it/s]
Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw

# See first training sample
image, label = train_data[0]
image, label
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.6078, 0.9255,
0.8118, 0.6980, 0.4196, 0.6118, 0.6314, 0.4275, 0.2510, 0.0902,
0.3020, 0.5090, 0.2824, 0.0588],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0039, 0.0000, 0.2706, 0.8118, 0.8745,
0.8549, 0.8471, 0.8471, 0.6392, 0.4980, 0.4745, 0.4784, 0.5725,
0.5529, 0.3451, 0.6745, 0.2588],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,

```

Kode di atas mengilustrasikan penggunaan PyTorch dalam mempersiapkan dan memuat data untuk pelatihan dan pengujian model pada dataset FashionMNIST. Pertama, dilakukan konfigurasi untuk dataset pelatihan dan pengujian menggunakan modul `torchvision.datasets`. Dataset FashionMNIST diunduh dan dilakukan transformasi menggunakan `ToTensor()`, sehingga gambar-gambar dalam dataset diubah menjadi tensor PyTorch.

Selanjutnya, dilakukan inisialisasi dataset pelatihan dan pengujian menggunakan objek `datasets.FashionMNIST`. Dataset pelatihan dan pengujian dipersiapkan dengan parameter-parameter seperti root (lokasi penyimpanan data), mode train atau test, dan transformasi yang diterapkan pada gambar.

Hasilnya, kita melihat contoh pertama dari dataset pelatihan yang menunjukkan tensor gambar dan labelnya. Pada contoh tersebut, gambar FashionMNIST pertama diubah menjadi tensor 3D, dan labelnya adalah 9, yang mengindikasikan kategori kelas "Ankle Boot". Seluruh proses ini merupakan langkah awal dalam mempersiapkan data untuk dilatih oleh model dalam tugas klasifikasi gambar.

- Bentuk masukan dan keluaran model visi computer

```
# What's the shape of the image?
image.shape

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

# How many samples are there?
len(train_data.data), len(train_data.targets), len(test_data.data), len(test_data.targets)

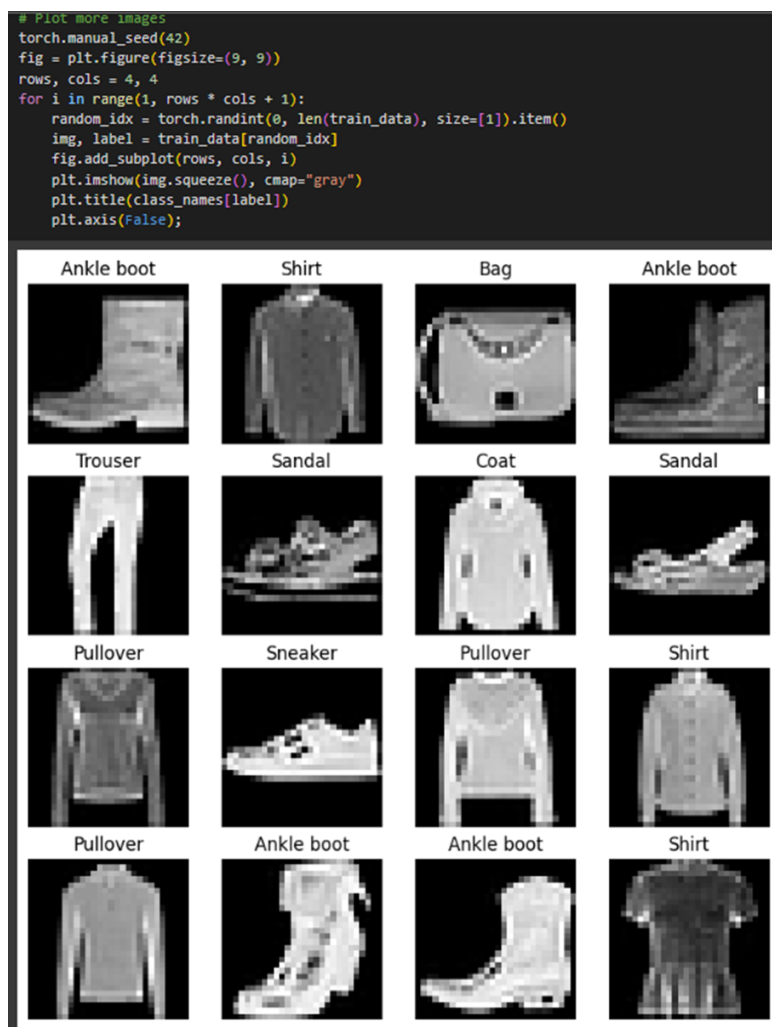
(60000, 60000, 10000, 10000)

# See classes
class_names = train_data.classes
class_names

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

Terdapat beberapa jenis pakaian yang telah diambil dari dataset. Total keseluruhan terdapat 7000 data dan terbagi menjadi 6000 data untuk train dan 1000 data untuk test.

- Visualisasi data



Memvisualisasikan data gambar dari beberapa jenis pakaian yang terdapat pada dataset serta mengubah warnanya.

- Mempersiapkan dataloader

```
# Turn datasets into iterables (batches)
train_dataloader = DataLoader(train_data, # dataset to turn into iterable
                              batch_size=BATCH_SIZE, # how many samples per batch?
                              shuffle=True # shuffle data every epoch?
                              )

test_dataloader = DataLoader(test_data,
                              batch_size=BATCH_SIZE,
                              shuffle=False # don't necessarily have to shuffle the testing data
                              )

# Let's check out what we've created
print(f"Dataloaders: {train_dataloader, test_dataloader}")
print(f"Length of train dataloader: {len(train_dataloader)} batches of {BATCH_SIZE}")
print(f"Length of test dataloader: {len(test_dataloader)} batches of {BATCH_SIZE}")

Dataloaders: (<torch.utils.data.dataloader.Dataloader object at 0x78a9037abc70>, <torch
Length of train dataloader: 1875 batches of 32
Length of test dataloader: 313 batches of 32

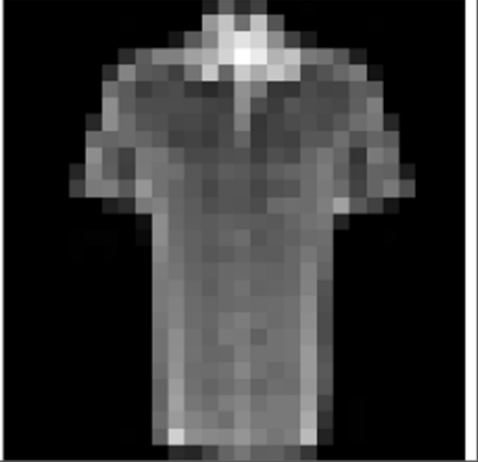
# Check out what's inside the training dataloader
train_features_batch, train_labels_batch = next(iter(train_dataloader))
train_features_batch.shape, train_labels_batch.shape

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

# Show a sample
torch.manual_seed(42)
random_idx = torch.randint(0, len(train_features_batch), size=[1]).item()
img, label = train_features_batch[random_idx], train_labels_batch[random_idx]
plt.imshow(img.squeeze(), cmap="gray")
plt.title(class_names[label])
plt.axis("off");
print(f"Image size: {img.shape}")
print(f"Label: {label}, label size: {label.shape}")

Image size: torch.Size([1, 28, 28])
Label: 6, label size: torch.Size([])

Shirt
```



Kode di atas menggunakan modul `torch.utils.data.DataLoader` untuk mengonversi dataset pelatihan dan pengujian ke dalam bentuk iterables (batch). Batch size atau ukuran batch didefinisikan sebagai 32. DataLoader ini memberikan iterator yang memungkinkan kita mengakses batch-batch dari dataset dengan mudah selama proses pelatihan dan pengujian. Dengan mengatur parameter shuffle pada DataLoader, data pelatihan diacak setiap epoch, sementara data pengujian tidak perlu diacak.

Hasilnya, kita mendapatkan dataloader untuk pelatihan dan pengujian yang masing-masing memiliki 1875 dan 313 batch, masing-masing dengan ukuran 32. Dengan menggunakan dataloader, kita dapat mengiterasi melalui batch-batch ini selama pelatihan dan pengujian model.

- Model 0: Membangun model dasar

```
# Create a flatten layer
flatten_model = nn.Flatten() # all nn modules function as a model (can do a forward pass)

# Get a single sample
x = train_features_batch[0]

# Flatten the sample
output = flatten_model(x) # perform forward pass

# Print out what happened
print(f"Shape before flattening: {x.shape} -> [color_channels, height, width]")
print(f"Shape after flattening: {output.shape} -> [color_channels, height*width]")

# Try uncommenting below and see what happens
#print(x)
#print(output)

Shape before flattening: torch.Size([1, 28, 28]) -> [color_channels, height, width]
Shape after flattening: torch.Size([1, 784]) -> [color_channels, height*width]

from torch import nn
class FashionMNISTModelV0(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.layer_stack = nn.Sequential(
            nn.Flatten(), # neural networks like their inputs in vector form
            nn.Linear(in_features=input_shape, out_features=hidden_units), # in_features
            nn.Linear(in_features=hidden_units, out_features=output_shape)
        )

    def forward(self, x):
        return self.layer_stack(x)

torch.manual_seed(42)

# Need to setup model with input parameters
model_0 = FashionMNISTModelV0(input_shape=784, # one for every pixel (28x28)
                               hidden_units=10, # how many units in the hidden layer
                               output_shape=len(class_names) # one for every class
)
model_0.to("cpu") # keep model on CPU to begin with

FashionMNISTModelV0(
  (layer_stack): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=784, out_features=10, bias=True)
    (2): Linear(in_features=10, out_features=10, bias=True)
  )
)
```

Pada kode di atas, kita membuat sebuah layer flatten menggunakan modul `nn.Flatten()` dari PyTorch. Layer ini digunakan untuk meratakan atau "flatten" inputnya, yang pada kasus ini adalah satu sampel gambar. Sample gambar tersebut memiliki bentuk awal `[color_channels, height, width]` dengan ukuran `[1, 28, 28]` (1 channel warna, tinggi 28 piksel, lebar 28 piksel).

Ketika kita menjalankan sampel gambar tersebut melalui layer flatten, bentuknya diubah menjadi `[color_channels, height * width]`, yang dalam hal ini menjadi `[1, 784]`. Proses ini melibatkan mengubah matriks 2D (28x28) menjadi vektor 1D (784).

Perlu dicatat bahwa kita dapat memeriksa perubahan ini dengan mencetak bentuk sebelum dan sesudah proses flattening. Kode tersebut memberikan keluaran yang menunjukkan perubahan bentuk tersebut. Jika ingin melihat nilai-nilai sebenarnya, bisa mencoba untuk mencetak nilai `x` sebelum flattening dan nilai `output` setelah flattening (uncomment print statements yang ada di kode).

- Metrik kerugian penyiapan, pengoptimal, dan evaluasi

```

import requests
from pathlib import Path

# Download helper functions from Learn PyTorch repo (if not already downloaded)
if Path("helper_functions.py").is_file():
    print("helper_functions.py already exists, skipping download")
else:
    print("Downloading helper_functions.py")
    # Note: you need the "raw" GitHub URL for this to work
    request = requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-deep-learning/main/helper_functions.py")
    with open("helper_functions.py", "wb") as f:
        f.write(request.content)

Downloading helper_functions.py

# Import accuracy metric
from helper_functions import accuracy_fn # Note: could also use torchmetrics.Accuracy(task = 'multiclass', num_classes=le

# Setup loss function and optimizer
loss_fn = nn.CrossEntropyLoss() # this is also called "criterion"/"cost function" in some places
optimizer = torch.optim.SGD(params=model_0.parameters(), lr=0.1)

```

Dalam kode tersebut, kita menggunakan `requests` untuk mendownload sebuah file utilitas Python, yaitu `helper_functions.py`, dari repositori Learn PyTorch di GitHub jika belum ada di sistem. Fungsi ini berisi utilitas bantuan, termasuk metrik akurasi (`accuracy_fn`). Setelah itu, kita mengimpor fungsi loss (`nn.CrossEntropyLoss()`) yang merupakan fungsi kerugian umum untuk tugas klasifikasi multikelas, dan optimizer Stochastic Gradient Descent (SGD) dari PyTorch dengan learning rate 0.1, yang akan digunakan selama pelatihan model (`torch.optim.SGD(params=model_0.parameters(), lr=0.1)`).

- Membuat fungsi untuk mengatur waktu percobaan

```

from timeit import default_timer as timer
def print_train_time(start: float, end: float, device: torch.device = None):
    """Prints difference between start and end time.

    Args:
        start (float): Start time of computation (preferred in timeit format).
        end (float): End time of computation.
        device ([type], optional): Device that compute is running on. Defaults to None.

    Returns:
        float: time between start and end in seconds (higher is longer).
    """
    total_time = end - start
    print(f"Train time on {device}: {total_time:.3f} seconds")
    return total_time

```

Fungsi `print_train_time` digunakan untuk mencetak waktu pelatihan model dengan menerima waktu awal (`start`) dan waktu akhir (`end`). Opsi tambahan mencakup perangkat tempat komputasi berlangsung, dan fungsi mengembalikan total waktu yang dicetak dalam detik.

- Membuat loop pelatihan dan melatih model pada kumpulan data

mengimplementasi loop pelatihan dan pengujian (testing) menggunakan model `model_0` pada dataset FashionMNIST selama tiga epoch. Dalam setiap epoch, model dilatih menggunakan data pelatihan, dan kemudian diuji menggunakan data pengujian. Metrik seperti loss dan akurasi dicetak untuk setiap epoch. Waktu total pelatihan juga dicetak menggunakan fungsi `print_train_time`. Pada akhir tiga

epoch, model mencapai akurasi pengujian sekitar 83.43%. Waktu total pelatihan pada CPU adalah sekitar 29.824 detik.

- Make predictions and get Model 0 results

mengimplementasi fungsi `eval_model` yang mengevaluasi model `model_0` pada dataset pengujian (test set) FashionMNIST. Hasil evaluasi, seperti nilai loss dan akurasi, kemudian disimpan dalam bentuk dictionary. Pada hasil evaluasi untuk `model_0`, diperoleh nilai loss sekitar 0.4766 dan akurasi sekitar 83.43%. Dictionary ini mencakup informasi seperti nama model, loss, dan akurasi.

- Siapkan kode agnostik perangkat

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

'cuda'
```

Mengecek device GPU pada device, jika tidak ada akan menghasilkan cpu.

- Model 1: Membangun model yang lebih baik dengan non-linearitas

Pada tahap ini mengimplementasikan pelatihan (training) dan evaluasi (testing) model neural network pada dataset FashionMNIST. Model yang digunakan adalah FashionMNISTModelV1, yang terdiri dari lapisan-lapisan non-linear dan linear. Proses pelatihan dilakukan selama tiga epoch dengan menggunakan fungsi pelatihan (train_step) dan evaluasi (test_step). Setelah pelatihan selesai, hasil evaluasi model, termasuk nilai loss dan akurasi, dicetak dan disimpan. Model ini kemudian dibandingkan dengan hasil model sebelumnya (model_0_results).

- Model 2: Membangun Jaringan Neural Konvolusional (CNN)

- Kode pertama mendefinisikan model `FashionMNISTModelV2`, sebuah arsitektur Convolutional Neural Network (CNN) yang terinspirasi dari TinyVGG dengan dua blok konvolusi dan lapisan klasifikasi linear. Blok konvolusi terdiri dari layer konvolusi, aktivasi ReLU, dan layer max-pooling. Lapisan klasifikasi mengubah dimensi dan melakukan klasifikasi.
- Kemudian, instance dari model (`model_2`) dibuat dan dipindahkan ke perangkat yang ditentukan (GPU jika tersedia).
- Kode selanjutnya menunjukkan contoh penggunaan layer konvolusi pada gambar dengan dimensi dan karakteristik yang sesuai dengan model (`images`).
- Selanjutnya, layer konvolusi (`conv_layer`) didefinisikan dan diterapkan pada `test_image`. Jika ada kesalahan dimensi, dimensi tambahan ditambahkan untuk sesuai dengan kebutuhan layer konvolusi.
- Kode juga menciptakan layer konvolusi lainnya (`conv_layer_2`) dengan parameter yang berbeda dan menunjukkan beberapa informasi tentang bobot dan bias yang terkandung di dalamnya.

Secara keseluruhan, pada tahap ini mendemonstrasikan pembuatan model CNN ('FashionMNISTModelV2'), penerapan layer konvolusi, serta pengecekan dan manipulasi dimensi pada penggunaan layer konvolusi pada gambar.

- Melangkah melalui nn.MaxPool2d()

```
# Print out original image shape without and with unsqueezed dimension
print(f"Test image original shape: {test_image.shape}")
print(f"Test image with unsqueezed dimension: {test_image.unsqueeze(dim=0).shape}")

# Create a sample nn.MaxPool2d() layer
max_pool_layer = nn.MaxPool2d(kernel_size=2)

# Pass data through just the conv layer
test_image_through_conv = conv_layer(test_image.unsqueeze(dim=0))
print(f"Shape after going through conv_layer(): {test_image_through_conv.shape}")

# Pass data through the max pool layer
test_image_through_conv_and_max_pool = max_pool_layer(test_image_through_conv)
print(f"Shape after going through conv_layer() and max_pool_layer(): {test_image_through_conv_and_max_pool.shape}")

Test image original shape: torch.Size([3, 64, 64])
Test image with unsqueezed dimension: torch.Size([1, 3, 64, 64])
Shape after going through conv_layer(): torch.Size([1, 10, 62, 62])
Shape after going through conv_layer() and max_pool_layer(): torch.Size([1, 10, 31, 31])

+ Code + Text

torch.manual_seed(42)
# Create a random tensor with a similar number of dimensions to our images
random_tensor = torch.randn(size=(1, 1, 2, 2))
print(f"Random tensor:\n{random_tensor}")
print(f"Random tensor shape: {random_tensor.shape}")

# Create a max pool layer
max_pool_layer = nn.MaxPool2d(kernel_size=2) # see what happens when you change the kernel_size value

# Pass the random tensor through the max pool layer
max_pool_tensor = max_pool_layer(random_tensor)
print(f"\nMax pool tensor:\n{max_pool_tensor} <- this is the maximum value from random_tensor")
print(f"Max pool tensor shape: {max_pool_tensor.shape}")

Random tensor:
tensor([[[[0.3367, 0.1288],
          [0.2345, 0.2303]]]])
Random tensor shape: torch.Size([1, 1, 2, 2])

Max pool tensor:
tensor([[[[0.3367]]]]) <- this is the maximum value from random_tensor
Max pool tensor shape: torch.Size([1, 1, 1, 1])
```

Dalam tahap ini menunjukkan proses pada perubahan bentuk gambar uji. Uji coba dilakukan baik sebelum maupun setelah dimensi tambahan ditambahkan menggunakan 'unsqueeze(dim=0)'. Selanjutnya, dibuat sebuah layer max-pooling dengan ukuran kernel 2x2. Gambar uji kemudian melewati layer konvolusi, diikuti dengan layer max-pooling, dan bentuk hasilnya dicetak. Kode terakhir menciptakan sebuah tensor acak, yang juga melewati layer max-pooling, dan hasilnya dicetak untuk menunjukkan efek dari operasi max-pooling pada tensor tersebut.

- Siapkan fungsi kerugian dan pengoptimal untuk model_2

```
# Setup loss and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(params=model_2.parameters(),
                             lr=0.1)
```

Dalam kode ini, fungsi loss yang digunakan adalah 'nn.CrossEntropyLoss()', yang umumnya digunakan dalam tugas klasifikasi multikelas. Optimizer yang digunakan adalah stochastic gradient descent (SGD) dengan laju pembelajaran (learning rate) sebesar 0.1. Kedua komponen ini bersama-sama membentuk bagian dari proses pelatihan (training) model jaringan saraf konvolusional (CNN) 'model_2' pada dataset FashionMNIST.

- Pelatihan dan pengujian model_2 menggunakan fungsi pelatihan dan pengujian


```

torch.manual_seed(42)

# Measure time
from timeit import default_timer as timer
train_time_start_model_2 = timer()

# Train and test model
epochs = 3
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n-----")
    train_step(data_loader=train_dataloader,
               model=model_2,
               loss_fn=loss_fn,
               optimizer=optimizer,
               accuracy_fn=accuracy_fn,
               device=device
    )
    test_step(data_loader=test_dataloader,
              model=model_2,
              loss_fn=loss_fn,
              accuracy_fn=accuracy_fn,
              device=device
    )

train_time_end_model_2 = timer()
total_train_time_model_2 = print_train_time(start=train_time_start_model_2,
                                             end=train_time_end_model_2,
                                             device=device)

100% ██████████ 3/3 [00:41<00:00, 13.20s/it]
Epoch: 0
-----
Train loss: 0.59448 | Train accuracy: 78.53%
Test loss: 0.37727 | Test accuracy: 86.61%

Epoch: 1
-----
Train loss: 0.35506 | Train accuracy: 87.20%
Test loss: 0.34734 | Test accuracy: 87.55%

Epoch: 2
-----
Train loss: 0.31861 | Train accuracy: 88.52%
Test loss: 0.31107 | Test accuracy: 88.85%

Train time on cuda: 41.023 seconds

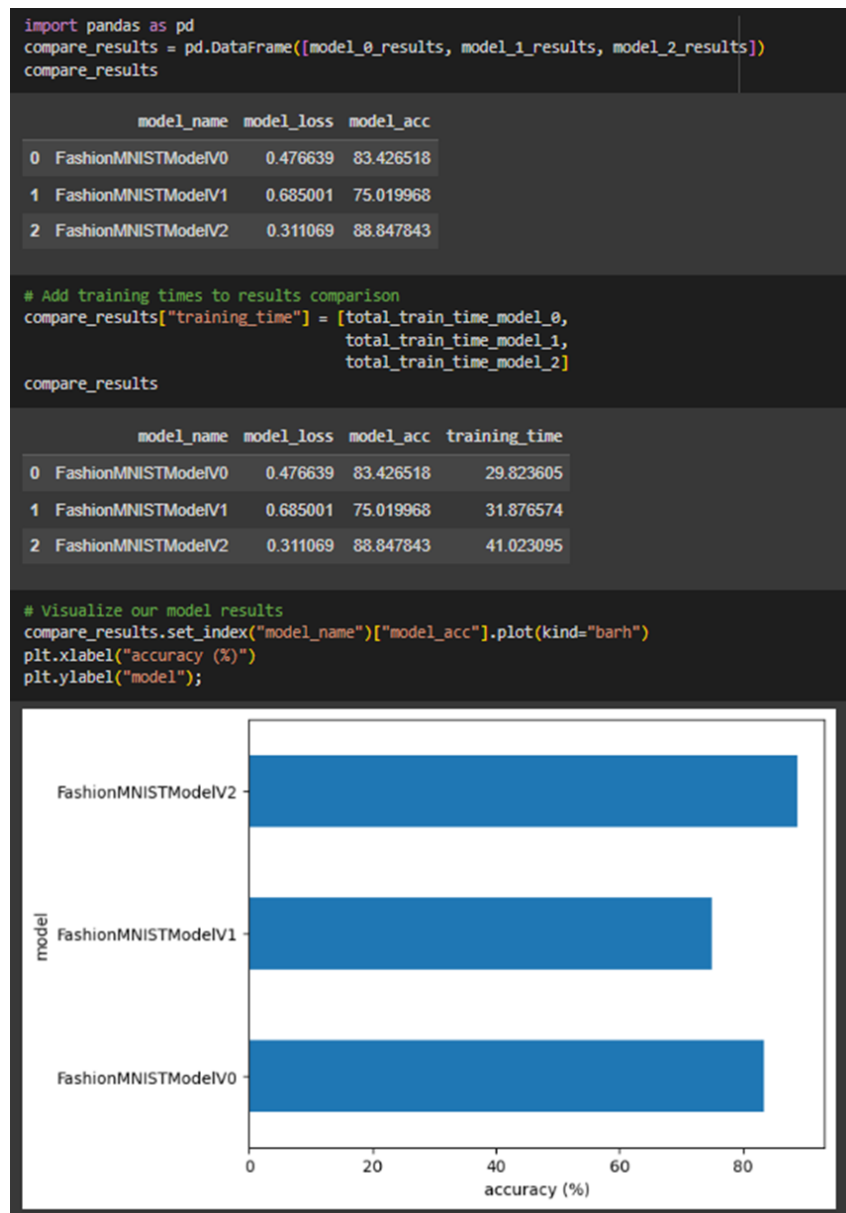
# Get model_2 results
model_2_results = eval_model(
    model=model_2,
    data_loader=test_dataloader,
    loss_fn=loss_fn,
    accuracy_fn=accuracy_fn
)
model_2_results

{'model_name': 'FashionMNISTModelV2',
 'model_loss': 0.31106942892074585,
 'model_acc': 88.84784345047923}

```

Dalam tahap ini, model jaringan saraf konvolusional (CNN) `model_2` dilatih dan diuji pada dataset FashionMNIST selama tiga epoch. Proses pelatihan dan pengujian dilakukan dengan memanggil fungsi `train_step` dan `test_step` yang telah didefinisikan sebelumnya. Pengukuran waktu juga dilakukan untuk melihat berapa lama waktu yang diperlukan untuk melatih model. Setelah pelatihan selesai, hasil evaluasi model, seperti nilai loss dan akurasi, dicetak. Hasilnya menunjukkan bahwa model mencapai loss sekitar 0.31 dan akurasi sekitar 88.85% di atas dataset pengujian.

- Bandingkan hasil model dan waktu pelatihan



Pada tahap ini, hasil evaluasi dari tiga model yang berbeda (`model_0`, `model_1`, dan `model_2`) telah digabungkan ke dalam suatu DataFrame menggunakan pustaka Pandas. DataFrame ini, `compare_results`, mencakup informasi seperti nama model, nilai loss, dan akurasi, serta waktu pelatihan untuk masing-masing model. Hasilnya menunjukkan perbandingan kinerja dan waktu pelatihan antara ketiga model tersebut. Model `FashionMNISTModelV2` memiliki loss yang rendah dan akurasi yang tinggi, tetapi waktu pelatihannya lebih lama dibandingkan dengan model lainnya.

- Membuat matriks konfusi untuk evaluasi prediksi

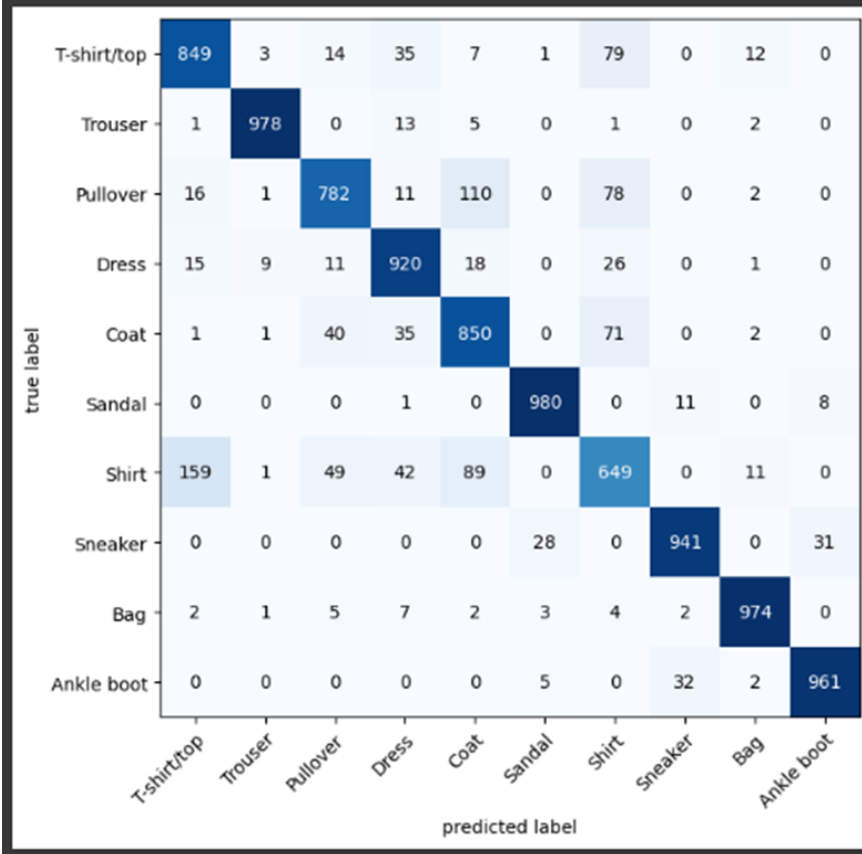
```

from torchmetrics import ConfusionMatrix
from mlxtend.plotting import plot_confusion_matrix

# 2. Setup confusion matrix instance and compare predictions to targets
confmat = ConfusionMatrix(num_classes=len(class_names), task='multiclass')
confmat_tensor = confmat(preds=y_pred_tensor,
                          target=test_data.targets)

# 3. Plot the confusion matrix
fig, ax = plot_confusion_matrix(
    conf_mat=confmat_tensor.numpy(), # matplotlib likes working with NumPy
    class_names=class_names, # turn the row and column labels into class names
    figsize=(10, 7)
);

```



Hasil evaluasi visual terhadap beberapa jenis pakaian.