



# Intro til Machine Learning: FashionMNIST

## Opgavebeskrivelse

I denne uge skal I arbejde med machine learning og få praktisk erfaring med, hvordan man kan træne en model til at genkende billeder. Formålet er at lære principperne bag dataforberedelse, modeltræning, evaluering og gemning af modeller

Vi bruger datasættet FashionMNIST, som indeholder billeder af forskellige typer tøj (sko, trøjer, bukser osv.). Datasættet er designet, så det minder om det klassiske [MNIST-datasæt](#) med håndskrevne cifre, men med mere realistiske og komplekse billeder.

Denne opgave er tænkt som en introduktion til praktisk brug af machine learning. Hvis du allerede har erfaring, vil du muligvis savne et dybere statistisk lag — det kommer i næste opgave, hvor du får mulighed for selv at vælge metode og fokusområde.

---

## Afleveringsformat

Afleveringen til denne uges opgave består af:

- **En individuel præsentation på fredag:**
  - Varighed: 10-15 minutter
  - Fokus på følgende områder:
    - Arbejdsproces og workflow
      - Hvordan gik du til opgaven?
      - Hvad var din proces og hvilke valg tog du? F.eks. antal epochs, batch size, learning rate ...
      - Eventuelle små ændringer eller justeringer, der afviger fra standardvejledningen (hvis relevant)
    - Trænings- og testresultater:
      - Brug gerne visuelle eksempler, fx. før og efter billeder, graf over loss eller accuracy over epochs



- Erfaringer og refleksion
    - Hvad var nemt/svært?
    - Hvad ville du gøre anderledes, hvis du havde mere tid?
    - Hvilke idéer har du til at forbedre modellen eller eksperimentere med data?
  - **Link til jeres Github-repository indeholdende:**
    - Kildekode med kommentarer
    - En README.md-fil, der kort og udførligt beskriver hvordan man kører jeres kode, trænings- og testresultater og andet I vurderer som relevant.
- 

## Casebeskrivelse

Du er blevet hyret ind som konsulent af firmaet “BigStyles”, som ønsker at automatisere sorteringen af tøj på deres varelager. Firmaet har allerede installeret et X-ray kamera, som tager billeder af kasser med tøj, og du vælger at benytte denne datakilde som fundament for at lave en machine learning-algoritme der kan hjælpe med at sortere kasserne automatisk.

Firmaet stiller et datasæt til rådighed, som består af gråtonede billeder af tøjet samt tilhørende labels, der angiver typen af tøj. Billederne har en størrelse på 28×28 pixels — små nok til at gøre træningen effektiv, men stadig tilstrækkelige til, at modellen kan lære at skelne mellem forskellige tøjtyper.

Datasættet er opdelt i trænings- og testfiler:

- `train-images-idx3-ubyte`
- `train-labels-idx1-ubyte`
- `t10k-images-idx3-ubyte`
- `t10k-labels-idx1-ubyte`

Filerne kan tilgås direkte via PyTorchs `torchvision.datasets.FashionMNIST`, hvilket gør det nemt at hente og arbejde med dataene

---

## Opgavevejledning

### Opsætning

- Start med at lave den mappe struktur du ønsker, f.eks. som nedestående. Dog skal datamappen have præcis denne struktur for at pytorch kan læse data ind:

```
your_project_folder
├── .gitignore
├── requirements.txt
├── README.md
├── .venv
├── data
│   └── FashionMNIST
│       └── raw
│           ├── train-images-idx3-ubyte
│           ├── train-labels-idx1-ubyte
│           ├── t10k-images-idx3-ubyte
│           └── t10k-labels-idx1-ubyte
└── src
    └── main.py
```

- Opret en config fil. Du vælger selv om det er json, .env eller lignende. Hvis man foretrækker det, kan man lave commandline interfaces. I config filen skal du specificere tre variabler:
  - Epochs:
    - Antal gange systemet træner.
    - Sæt den til **5**.
  - Learning\_rate:
    - Den værdi systemet kan ændre sine vægtning med
    - Sæt den til **1e-3**.
  - Batch\_size:
    - Den mængde data systemet læser af gangen.
    - Sæt den til **64**.
- Opret en main.py fil, der loader config filerne.

## Data loading og præparation

- Pytorch har en specifik klasse til at loade data fra dette datasæt ind:  
`torchvision.datasets.FashionMNIST`.
  - Brug klassen som beskrevet for at definere to datasæt. Et til træning og et andet til testning.
- Klassen tager imod to forskellige transform argumenter, *transform* og *target\_transform*. Der findes mange forskellige former for transformationer, der kan gives som disse argumenter. Begge argumenter er optional og vi bruger kun det første, *transform*.
  - Specificér i begge datasæt, at de skal bruge ***torchvision.transforms.ToTensor()***. På den måde bliver billederne transformeret til tensorer.
- Datasættene er nu læst ind. Dog skal der præciseres en loader, der definerer hvor meget data netværket skal håndtere ad gangen. Lav derfor en dataloader til begge datasæt. Som *batch\_size*, skal den bruge den værdi du definerede i opsætningen.

## Det neurale netværk

- I din src mappe skal du oprette en fil der hedder `network.py`.
  - Inde i filen skal du lave en klasse der repræsenterer dit neurale netværk.
  - Opret en klasse der hedder `neural_network` eller lignende. Det er vigtigt at den nedarver fra `torch.nn.Module`. Dette kan gøres ved:

```
class neural_network(nn.Module):
```

- Klassen skal have mindst to funktioner. En ***init*** og en ***forward*** funktion.
  - Init funktionen sørger for at sætte klassen op.
  - Forward funktionen fortæller hvordan systemet skal behandle data.
  - ***Init*** funktionen skal se ud som følgende:

```
def __init__(self) -> none:
    super().__init__()
    self.flatten: Flatten = nn.Flatten()
    self.network_stack: Sequential = nn.Sequential(
        nn.Linear(in_features = 28*28, out_features = 512),
        nn.ReLU(),
        nn.Linear(in_features = 512, out_features = 10)
```



)

- **`super.__init__()`** sørger for at klassen **`nn.Module`** bliver initialiseret korrekt. **`Flatten`** sørger for at tensoren bliver 1 dimensionel. **`Network_stack`** er selve strukturen af dit neurale netværk. Den består af følgende:
  - **`nn.Linear`** er et lineært lag, der tager billedet og reducerer det ned til 512 værdier
  - **`nn.ReLU`** er activation funktionen Rectified Linear Unit
  - Endnu et **`nn.Linear`** lag, der yderligere reducerer data ned til 10 grupper
- **`Forward`** funktionen skal se ud som følgende:

```
def forward(self, x):
    x = self.flatten()
    output = self.network_stack(x)
    return output
```

- Det vil sige at hver gang din model bliver kaldt med et batch af data, vil den først flatten, og derefter køre den igennem netværket.
- 

### Gør systemet klar til test

I din `main.py` skal vi definere 4 objekter.

- Systemet skal have en enhed den kan træne på, ofte en GPU. Det er dog ikke altid der er en GPU til rådighed (specielt ikke på disse bærbare). Man kan derfor definere en fall-back mulighed til CPU'en, som følgende:

```
device = torch.accelerator.current_accelerator().type if
torch.accelerator.is_available() else "cpu" # skal stå på en linie
```

- Systemet skal have din model. Den kan importeres og instantieres via:

```
model = neural_network.to(device)
```

- Systemet skal have en optimizer. For nu bruger vi en stochastic gradient descent (SGD) algoritme. Lav den ved:

```
optimizer = torch.optim.SGD(params = model.parameters(), lr = learning_rate)
```

- Til sidst skal systemet kende til en loss function. For nu bruger vi cross entropy loss. Lav den ved:

```
loss_fn = nn.CrossEntropyLoss()
```

### Lav trænings og test loops

- Kig [her](#) for opsætning

### Træn og test

- Vi kan nu træne og teste modellen. Lav derfor et loop der kører epochs antal gange. Det kan eventuelt se således ud:

```
for t in range(epochs):
    print(f"Epoch {t+1}\n-----")
    train_loop(
        train_dataloader,
        model=model,
        loss_fn=loss_fn,
        optimizer=optimizer
    )
    test_loop(
        test_dataloader,
        model=model,
        loss_fn=loss_fn
    )
```

### Gem trænet model

- Når modellen er blevet trænet og testet vil man typisk gemme sin model, således at man kan bruge den senere. Dette kan gøres på to måder:
  - Gem kun vægtene
  - Gem hele modellen
- Hvilken man bruger varierer. Dog vil det mest fordelagtige være kun at gemme vægtene
- Gemning og loading af vægtene kan gøres på denne måde:

```
model_file = "model.pt"
model = neural_network()
if os.path.exists(model_file):
    model.load_state_dict(torch.load(model_file, weights_only=True))
```



```
    model.eval  
.  
.  
torch.save(model.state_dict,model_file)
```

- Dog vil vi bemærke, at modelfilen ikke har en bestemt type extension. Her bruger vi .pt for pytorch
- Gemning og loading af hele modellen kan gøres på denne måde

```
model_file = "model.pt"  
if os.path.exist(model_file):  
    model = torch.load(model_file,weights_only=False)  
else:  
    model = neural_network()  
.  
.  
torch.save(model,model_file)
```

---

## Pensum og Ressourcer

- [Machine learning](#)

---

God arbejdslyst!