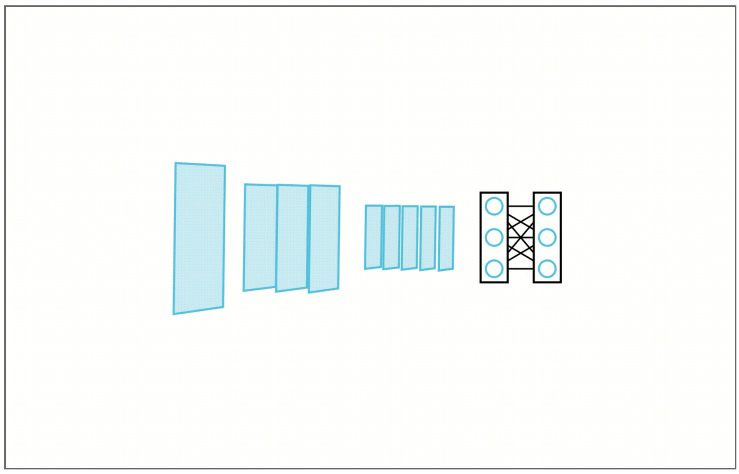
🔥 Deep Learning Workshop



Aron van de Pol

Leiden University Centre For Digital Humanities

2024-03-05

Images as data

📅 Today

1. Loading images into tensors
2. Batching & Dataloaders
3. Creating model architectures for image data.

Images as data

- Humanities we have lots of potential data
- Most often DH works with text
- Digitized texts not always *machine-readable*

Images as data

- Art Historians
- Visual history
- Print shops
- Advertisements
- Etc

👔 FashionMNIST

- Very common dataset to practice on
- Produced from Zalando product images
- 70,000 grayscale images
  - 28*28px
- 9 Classes

👔 FashionMNIST

0. T-shirt/top
1. Trouser
2. Pullover
3. Dress
4. Coat
5. Sandal
6. Shirt
7. Sneaker
8. Bag
9. Ankle boot

FashionMNIST

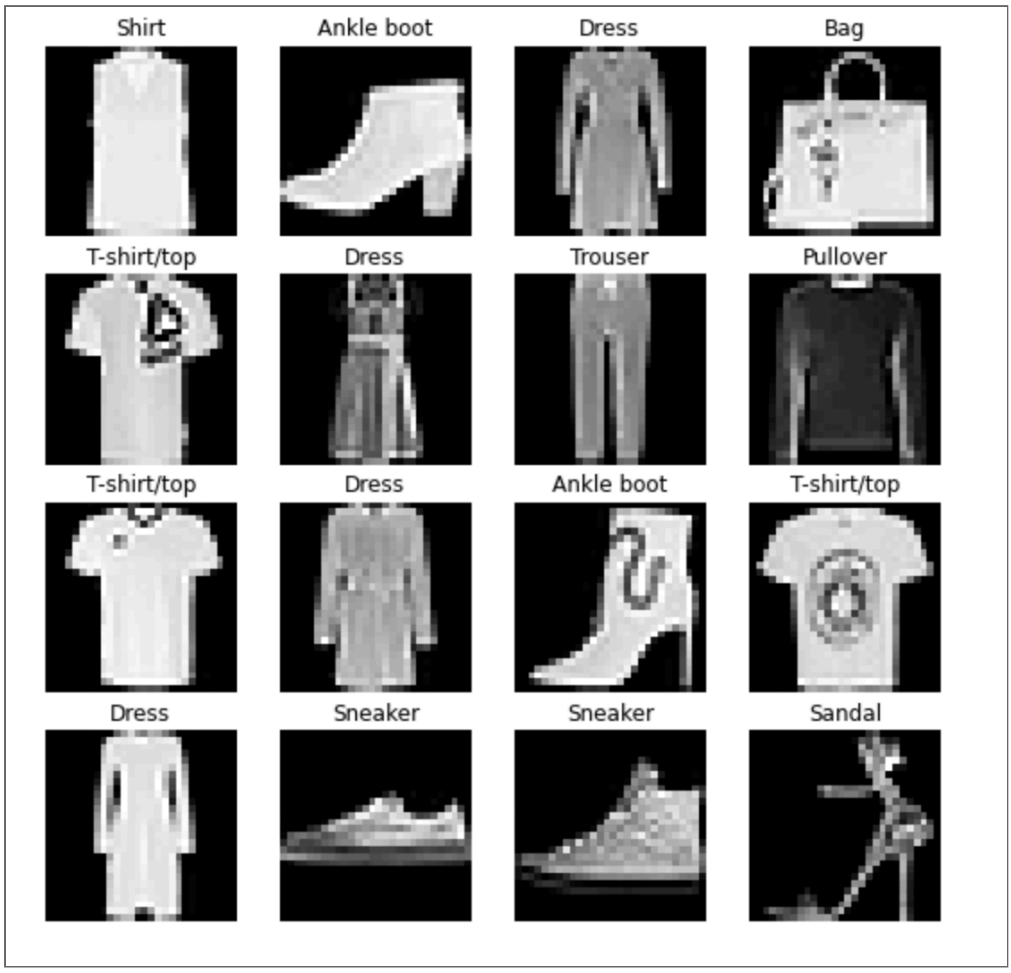


Figure 1: Random sampled images from the FashionMNIST dataset

FashionMNIST

We can load it from pytorch:

```
train_data = datasets.FashionMNI
    root="data", # where to down
    train=True, # get training d
    download=True, # download da
    transform=ToTensor(), # turn
    target_transform=None # you
)

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

📚 Batches and Dataloaders

📚 Batches

- Single number data points are light.
- Images as data are heavy.
- Computationally inefficient to push the entire dataset through the model.

📚 Batches

- Batching as a solution
- Split up the data into *batches*
- push batches through the model and every batch update parameters.

📚 Batches

1. **Sample** - smallest unit, represents one data point.
2. **Batch size** - The hyperparameter that defines the number of samples per batch.
   - Often this is a power of 2 (2, 4, 8 16 etc…)
3. **Batch** - *N* amount of samples.
4. **Epoch** - Full pass over all the data (thus all batches)

Batches in PyTorch

```
from torch.utils.data import Dat

# Setup the batch size hyperpara
BATCH_SIZE = 32

# Turn datasets into iterables (
train_dataloader = DataLoader(tr
                              ba
                              sh
)

test_dataloader = DataLoader(tes
                             bat
                             shu
)
```

🔧 Model Architectures for image data

🏗️ Baseline model

- We start of with a baseline model
  - `nn.Linear()`
  - `nn.ReLU()`
- But, how can we input data?

◾ `nn.Flatten()`

- Flatten changes our square image tensors into vectors.

Figure 2: `nn.Flatten()` visualized

nn.Flatten()

```
# Create a flatten layer
flatten_model = nn.Flatten()

# Get a single sample
x = train_features_batch[0]

# Flatten the sample
output = flatten_model(x)

# Print out what happened
print(f"Shape before flattening:
print(f"Shape after flattening:

#returns
Shape before flattening: torch.S
Shape after flattening: torch.Si
```

🏗️ Baseline model

```
# Create a model with non-linear
class FashionModelV0(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer_stack = nn.Se
            nn.Flatten(), # flat
            nn.Linear(in_feature
            nn.ReLU(),
            nn.Linear(in_feature
            nn.ReLU()
        )

    def forward(self, x: torch.T
        return self.layer_stack(
```

🏋️ Training loop differences

```
epochs = 10

for epoch in range(epochs): # lo
    print(f'epoch:{epoch}')
    train_loss = 0 # tracking th
    for batch, (X, y) in enumera
        X, y = X.to(device), y.t
        model_1.train()
        # 1. Forward pass
        y_pred = model_1(X)
        loss = loss_fn(y_pred, y
        train_loss += loss # acc
        # rest of train loop

    # (average loss per batch pe
    train_loss /= len(train_data
    # rest of the test loop...
```

🖥️ Let's code!

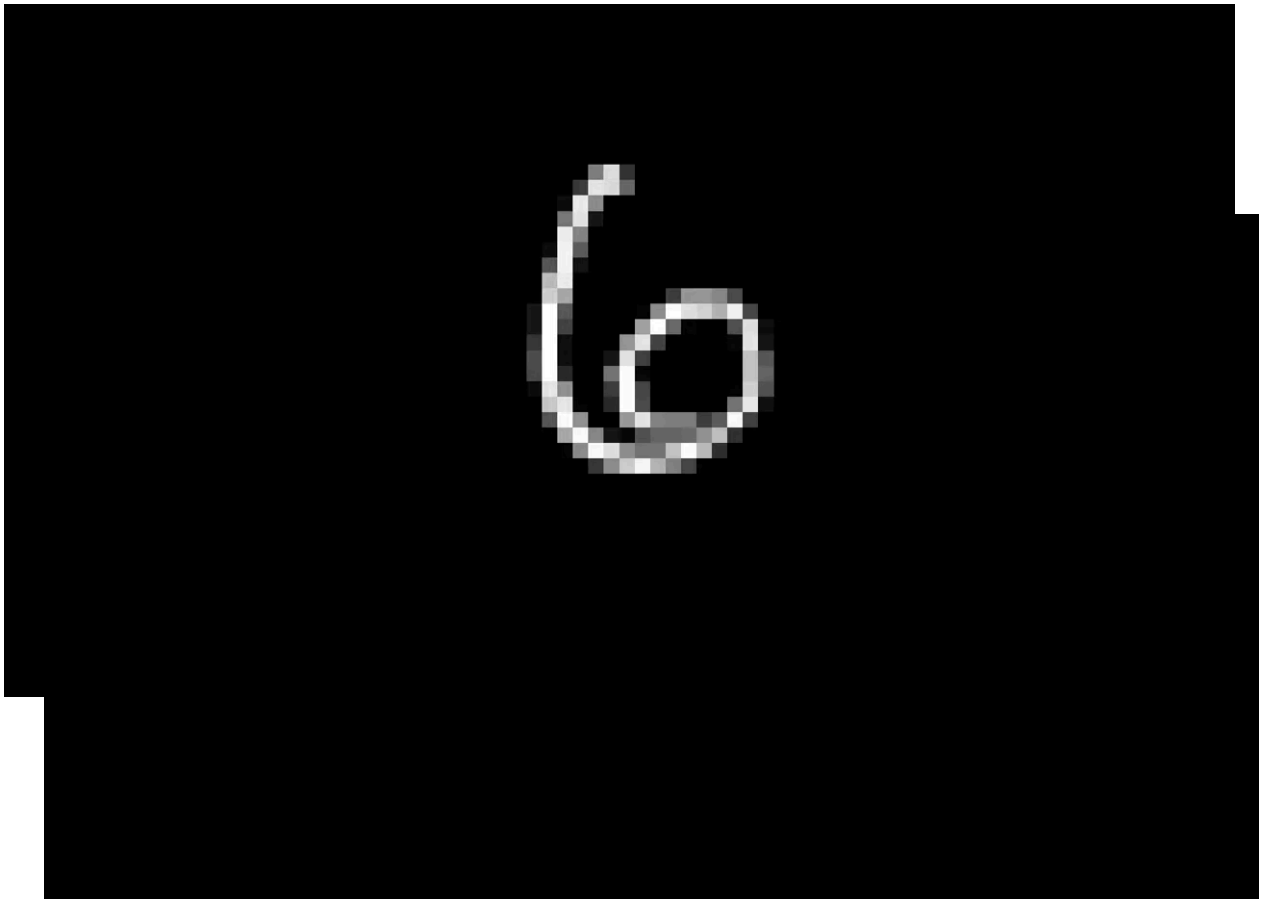
🎑 Convolutional Layers

🧮 Convolutional Layers

- Also known as CNN or ConvNet
- Good at finding patterns in visual data
- Makes use of so-called ‘convolutions’

🧮 A Convolution

1. We have our image
2. We have a kernel that applies some calculation to the pixels it ‘slides’ over. In this case the kernel is 3*3.
3. The result of the calculation is a single pixel value for a ‘new’ image.
4. Slide over all the pixels to get a ‘compressed’ image.



 A Convolution

CNN-explainer website

🏗️ TinyVGG Network

1. 1st Conv2d layer (the convolution layer)
2. ReLU activation function
3. 2nd Conv2d layer
4. ReLU activation function
5. MaxPool2d layer (compressing the image even further)

🏗️ TinyVGG Network

```
# Create a convolutional neura
class FashionModelV1(nn.Module
    """
    Model architecture copying
    """
    def __init__(self):
        super().__init__()
        self.block_1 = nn.Sequ
            nn.Conv2d(in_chann
                      out_chan
                      kernel_s
                      stride=1
                      padding=
            nn.ReLU(),
            nn.Conv2d(in_chann
                      out_chan
                      kernel_s
                      stride=1
```

🖥️ Let's Code

🟦 Confusion Matrix

Figure 3: Confusion Matrix of TinyVGG architecture on FashionMNIST

📚 Works Cited