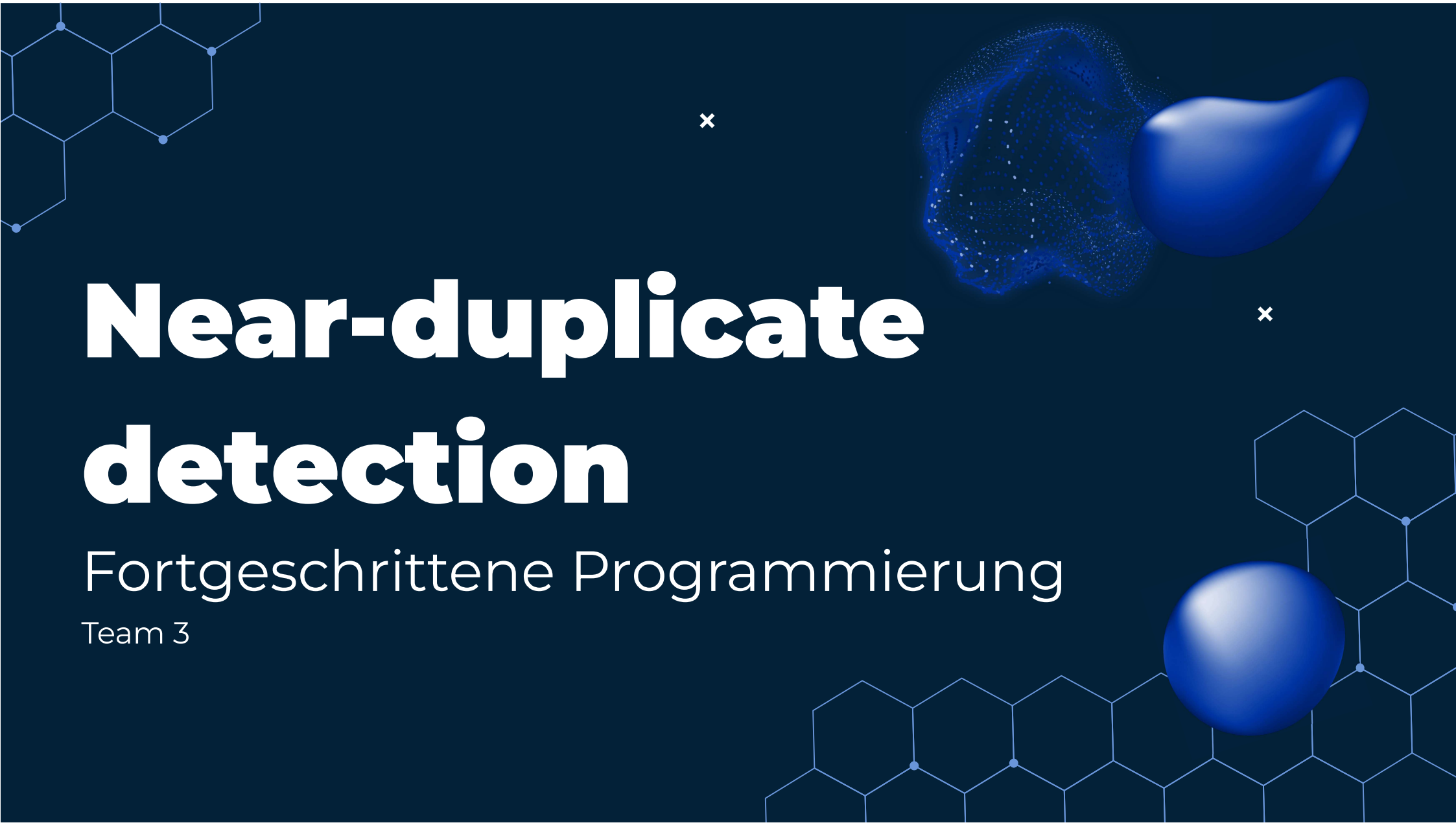
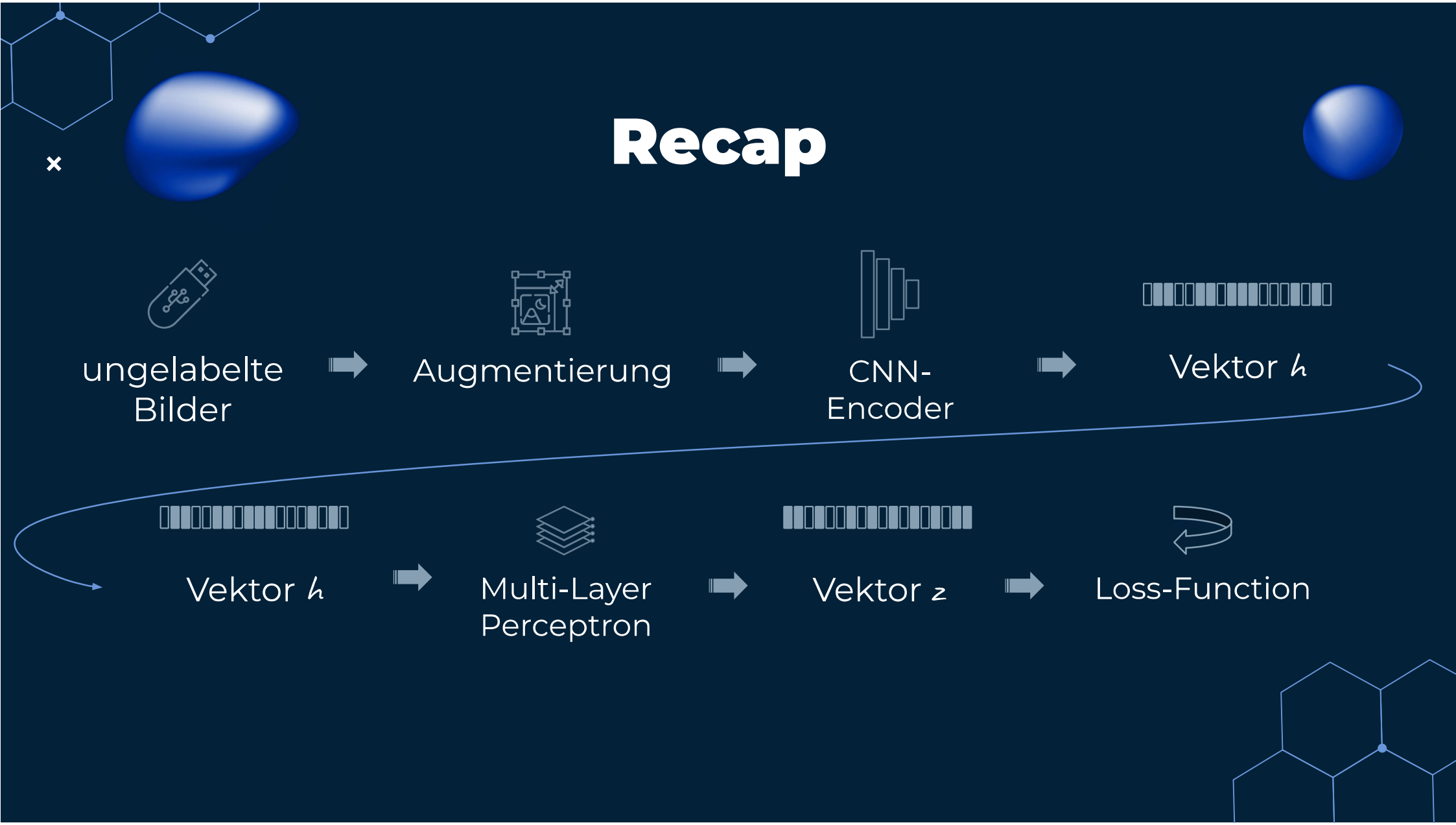
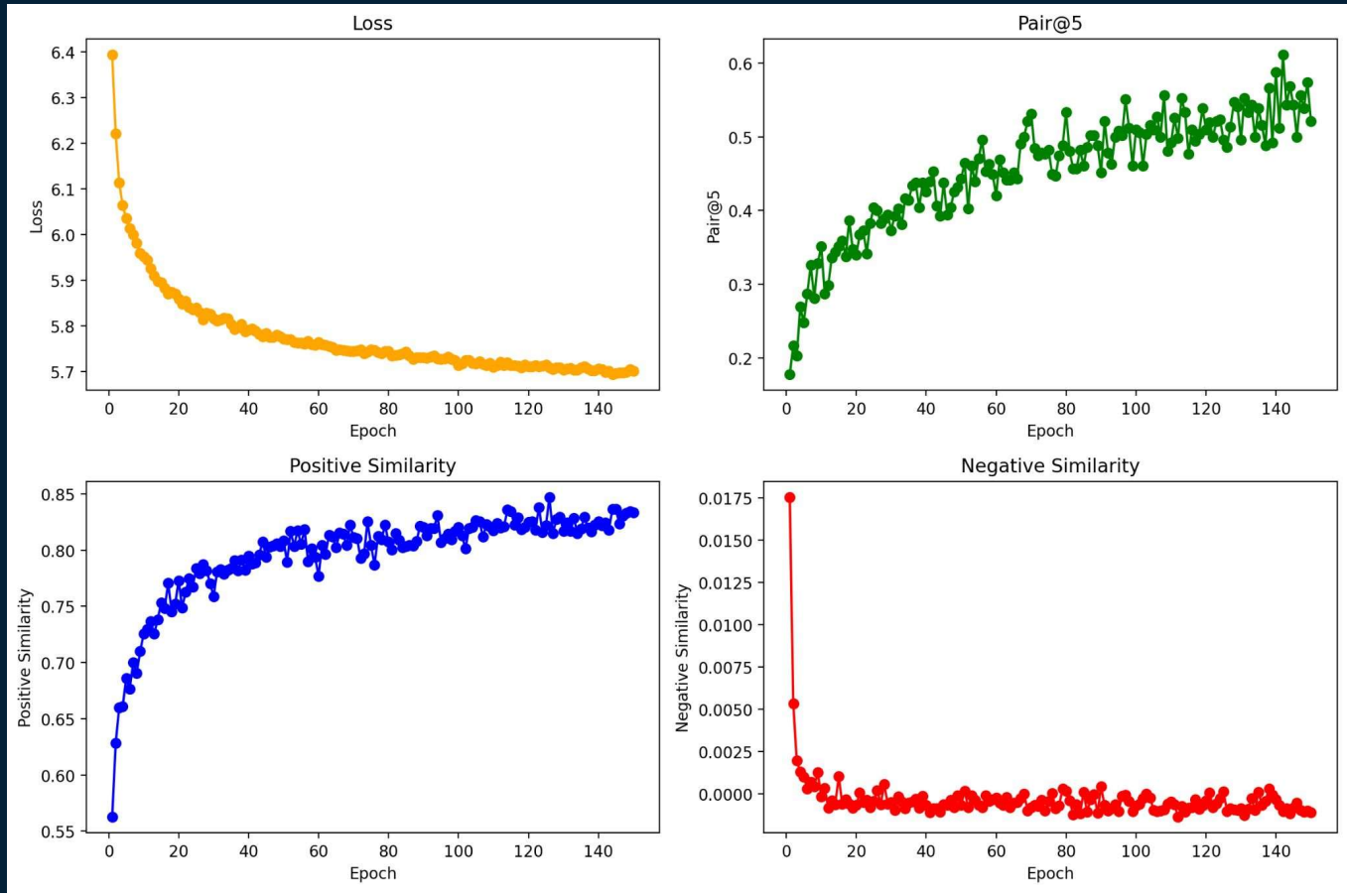

Near-duplicate detection
Fortgeschrittene Programmierung
Team 3

Recap


ungelabelte Bilder
Augmentierung
CNN-Encoder
Vektor h
Vektor h
Multi-Layer Perceptron
Vektor z
Loss-Function

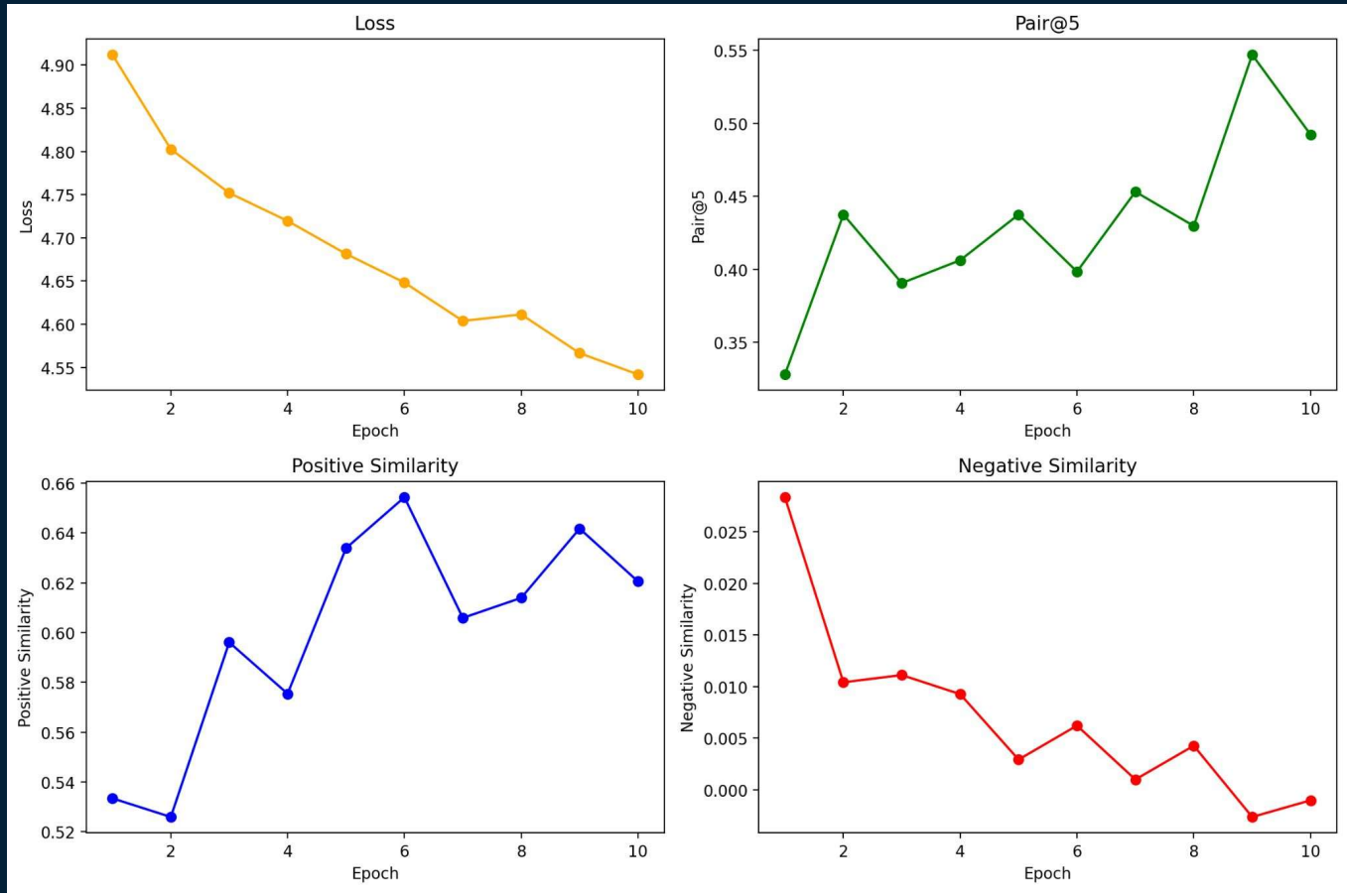
Training

Parameter	
Image Size	**144x144**
Batch Size	**512**
Learning Rate	**0.003**
Feature Dimension	**256**



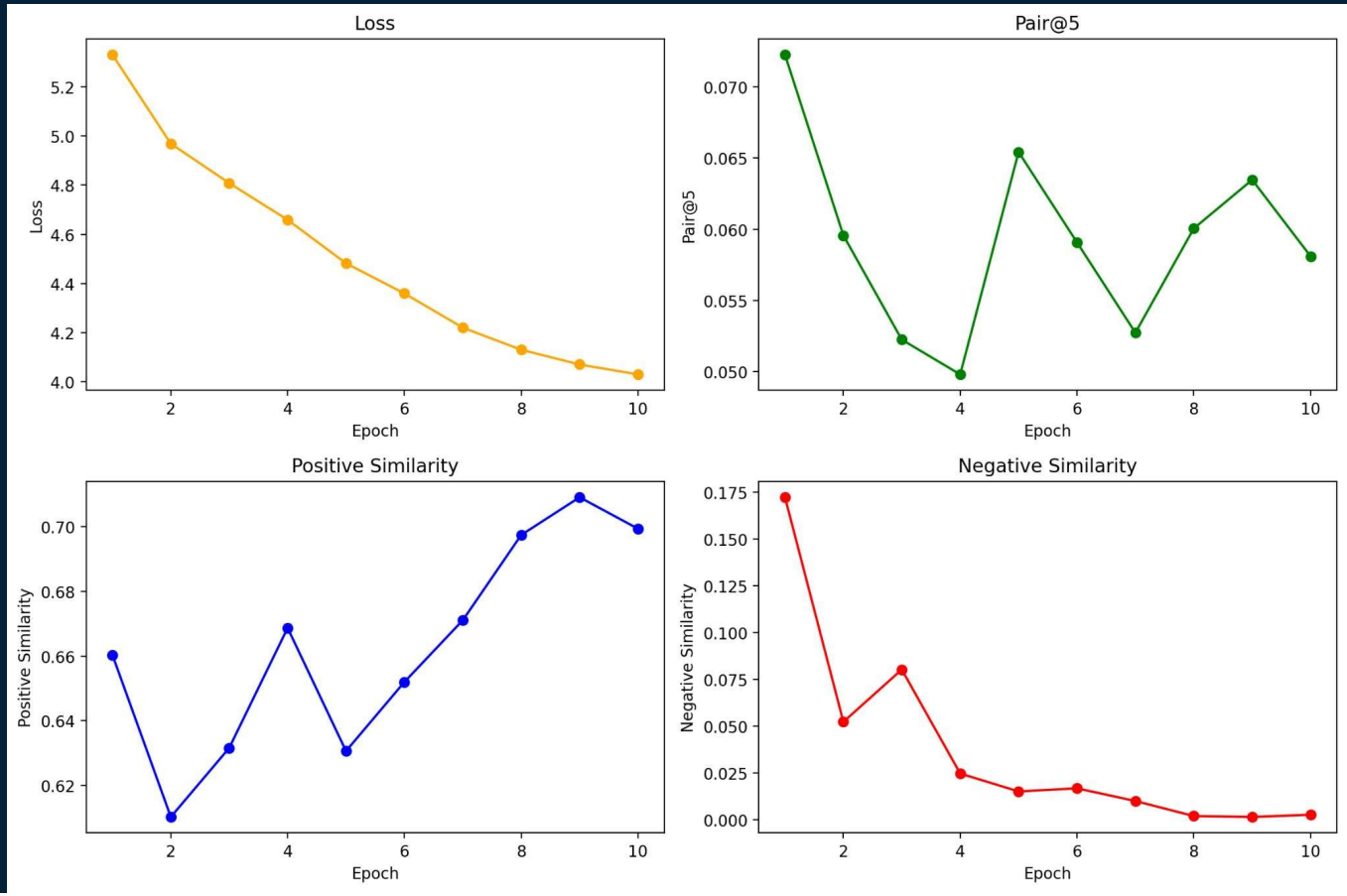
Training

Parameter	
Image Size	**224x224**
Batch Size	**128**
Learning Rate	**0.001**
Feature Dimension	**384**



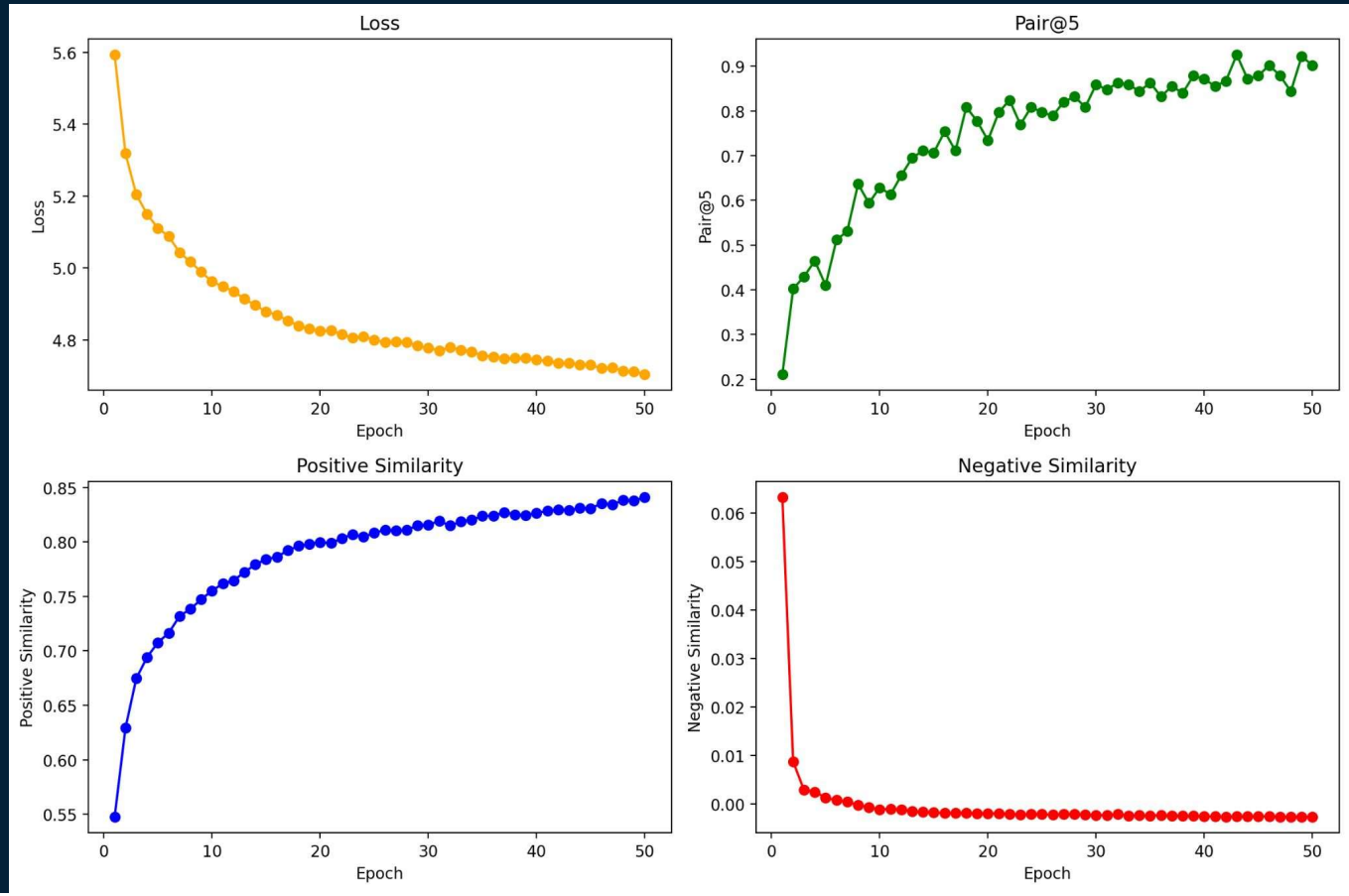
Training

Parameter	
Image Size	**192x192** ↓
Batch Size	**256** ↑
Learning Rate	**0.01** ↑
Feature Dimension	**256** ↓



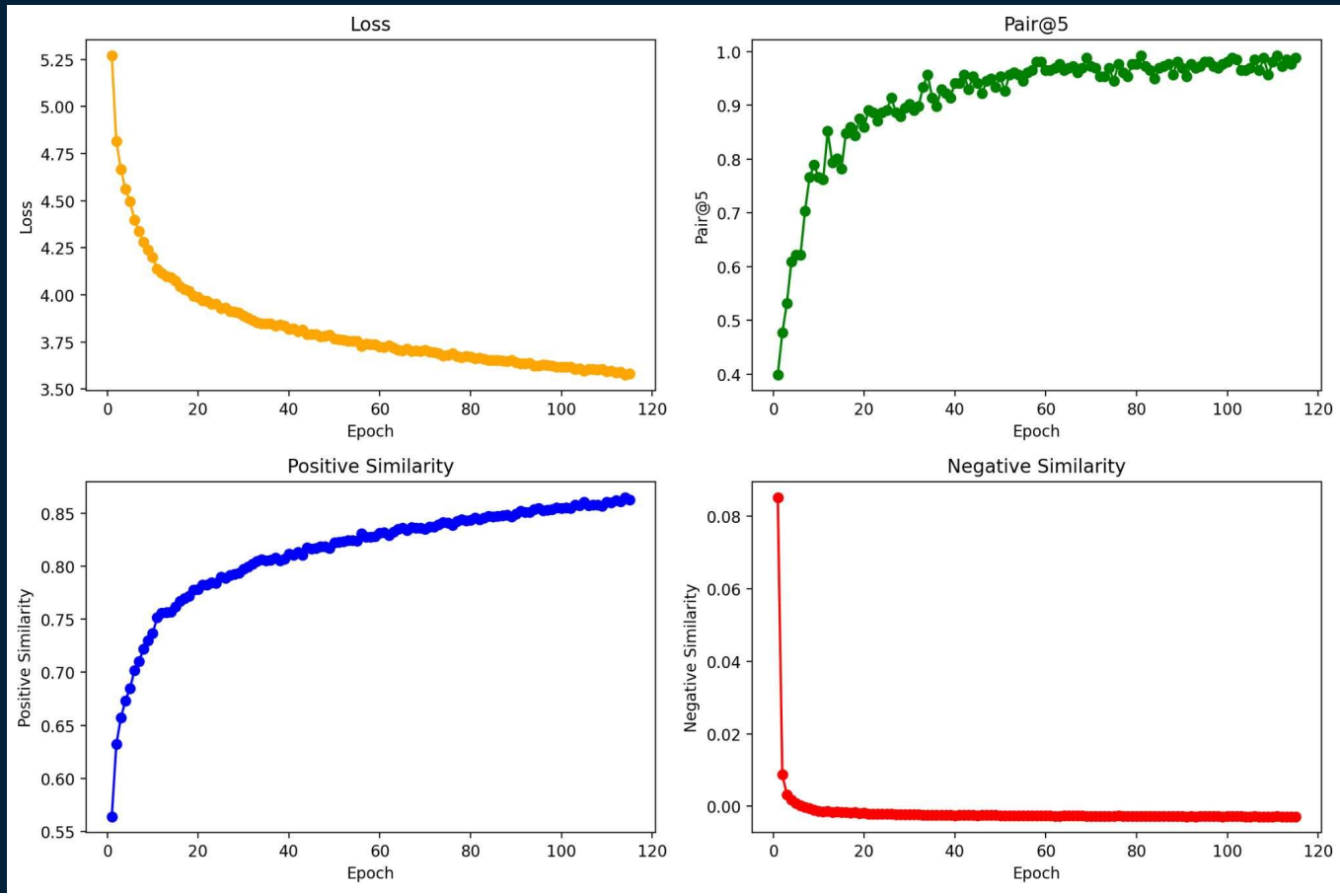
Training

Parameter	
Image Size	**224x224**
Batch Size	**256**
Learning Rate	**0.003**
Feature Dimension	**256**



Training Final

Parameter	
Image Size	**224x224**
Batch Size	**256**
Learning Rate	**0.003**
Feature Dimension	**512**



Encoder Versuch 1

Layer	Kanäle	Größe (H×W)	Neuronen
Input	3	224×224	150.528
Stem	64	112x112	802.816
Layer 1	128	56×56	401.408
Layer 2	256	28×28	200.704
Layer 3	512	14×14	100.352
Convo Out	512	14×14	100.352
GAP + FC	512	1×1	512

Stem:

➔ Kantenerkennung

Layer 1, 2 & 3:

➔ Conv2D
➔ GroupNorm
➔ ReLu

GAP:

➔ Global Average Pooling

FC:

➔ Fully Connected Layer

Encoder Final

Layer	Kanäle	Größe (H×W)	Neuronen
Input	3	224×224	150.528
Stem	64	56×56	200.704
Layer 1	64	56×56	200.704
Layer 2	128	28×28	100.352
Layer 3	256	14×14	50.176
Layer 4	512	7×7	25.088
GAP + FC	512	1×1	512

Stem:

- ➔ Cov2D + MaxPooling

Layer 1, 2 & 3:

- ➔ 2 Blöcke a 2 Conv2D
- ➔ BatchNorm
- ➔ ReLu
- ➔ Skip-Connection

Encoder

```
# Stem
self.conv1 = nn.Conv2d(3, self.in_ch, kernel_size=7, stride=2, padding=3, bias=False)
self.bn1   = nn.BatchNorm2d(self.in_ch)
self.relu  = nn.ReLU(inplace=True)
self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

# Stages (2,2,2,2)
self.layer1 = self._make_layer( BasicBlock, 64,  blocks=2, stride=1)
self.layer2 = self._make_layer( BasicBlock, 128, blocks=2, stride=2)
self.layer3 = self._make_layer( BasicBlock, 256, blocks=2, stride=2)
self.layer4 = self._make_layer( BasicBlock, 512, blocks=2, stride=2)

# Global Average Pooling
self.gap = nn.AdaptiveAvgPool2d((1, 1))
```

BasicBlock

```
self.conv1 = conv3x3(in_ch, out_ch, stride)
self.bn1   = nn.BatchNorm2d(out_ch)
self.relu  = nn.ReLU(inplace=True)
self.conv2 = conv3x3(out_ch, out_ch)
self.bn2   = nn.BatchNorm2d(out_ch)
self.downsample = downsample  # passt die Skip-Connection an (Form/Stride)
```

Skip-Connection:

- ➔ eigene Identität kann gelernt werden
- ➔ Gradientenfluss bleibt stabil

Stem:

- ➔ low-level Features (Kanten & Texturen)

GAP:

- ➔ 1 Wert pro Kanal

Python-Pakete

torch / torch.nn / torch.nn.functional

➔ Modelle und Layer (Convolution, Linear, Batchnorm)

torchvision / torchvision.transform

➔ Datenvorbereitung & Augmentierung

torch.utils.data

➔ DataLoader, Batch-Bildung

torch.optim

➔ Gewichte von Encoder & Projektionskopf anpassen

torch.load / torch.save

➔ Modell speichern & laden

torch.cuda.amp

➔ Speicher sparen & Trainingsbeschleunigung

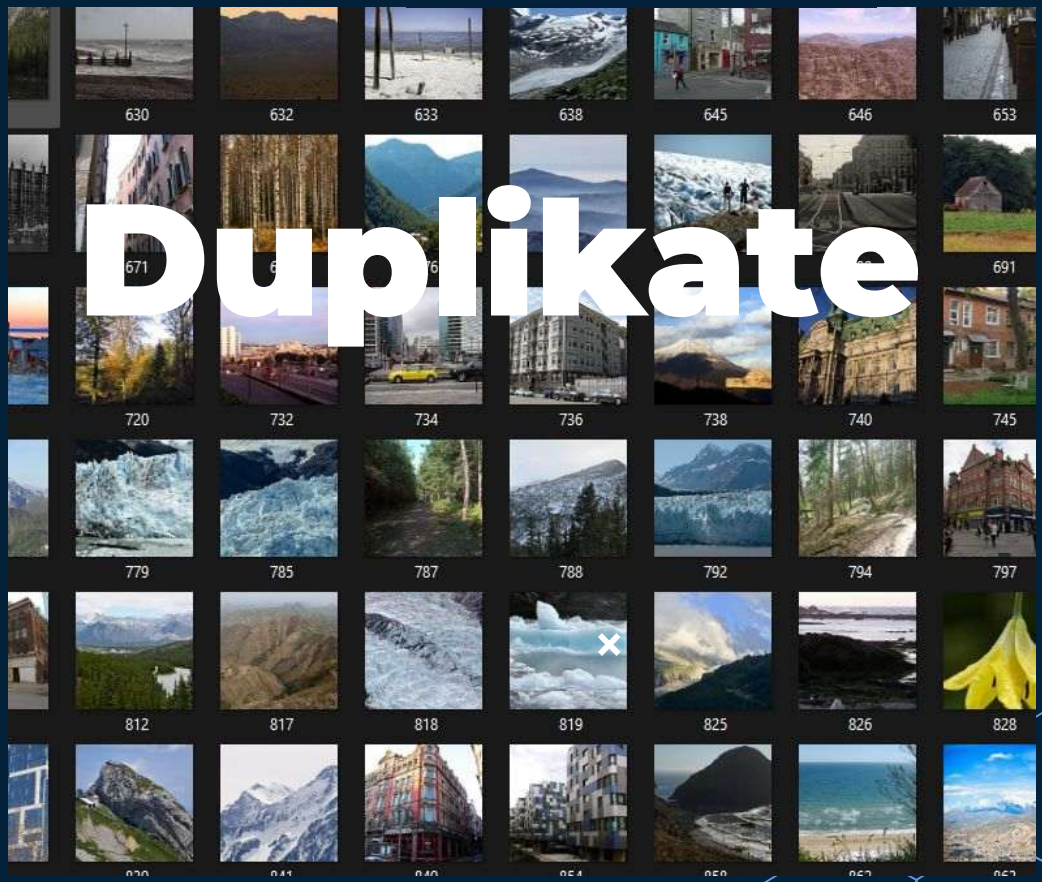
UL vs. SSL

Unsupervised Learning (Autoencoder)	Self-Supervised Learning (SimCLR)
Fokus auf Pixelgenaue Ähnlichkeit	robustere Repräsentationen
empfindlich gegenüber Augmentierungen	explizit invariant gegenüber Augmentierungen
Limitiert in Computer Vision (Farben, Kanten & Texturen)	State-of-the-Art in Computer Vision

[1], [2]

Anwendung

Duplikate


630
632
633
638
645
646
653
671
691
720
732
734
736
738
740
745
779
785
787
788
792
794
797
812
817
818
819
825
826
828

Wir brauchten

Kleine App

- ➔ Bildervergleich
- ➔ Duplikate angezeigt
- ➔ Möglichkeit gibt, Bilder zu behalten oder zu entfernen

DuplicAIt

Lokal verwendbar
➔ Offline & Nativ

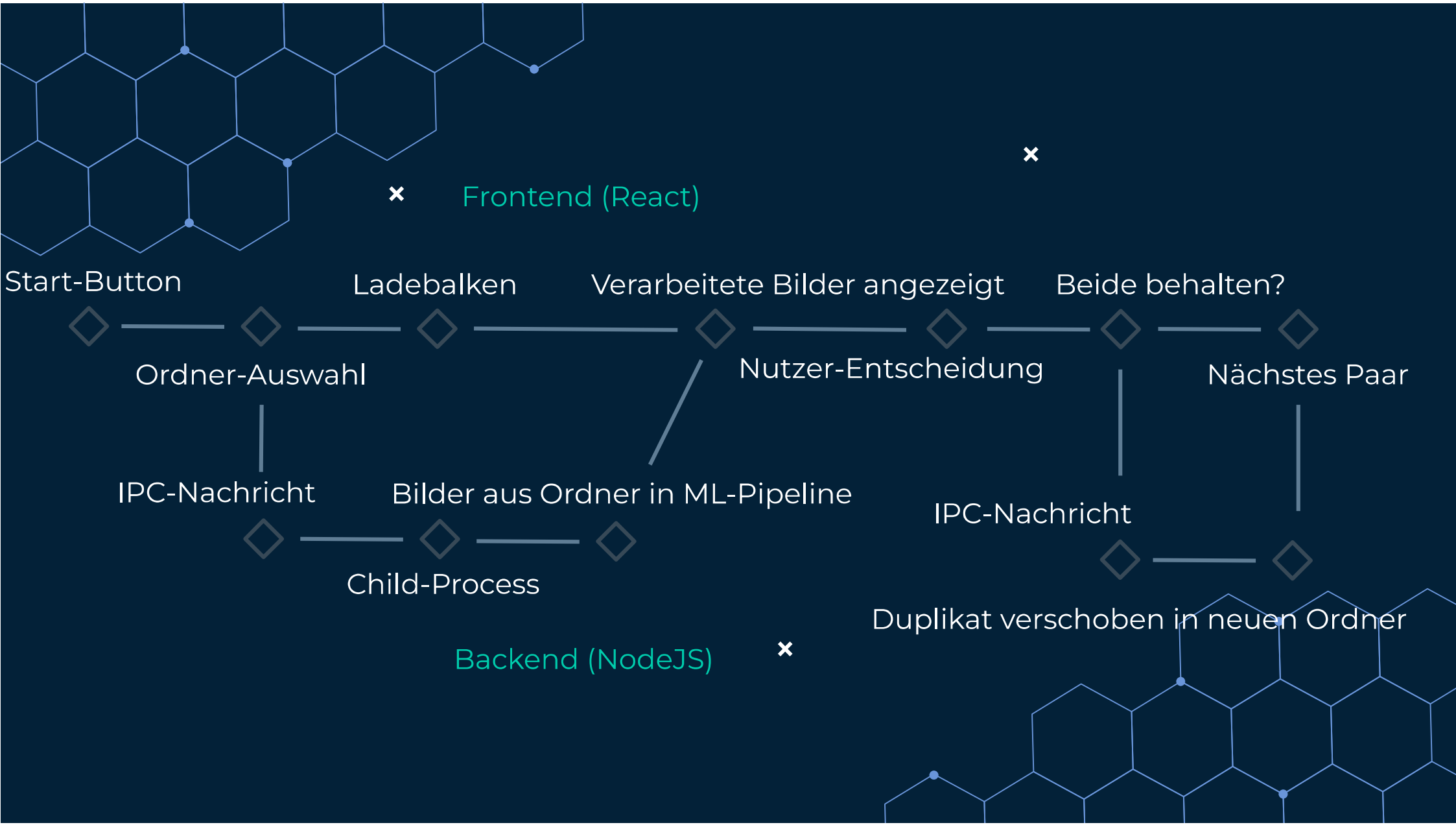
Einfach und intuitiv
➔ Minimalistisches Design

Ordnerauswahl
➔ Kompletter Foto-Ordner

Direkter Vergleich
➔ Gegenüberstellung der Duplikate

Lightweight
➔ Nur das Notwendigste

Backups
➔ Fotos nicht gelöscht, sondern verschoben


Frontend (React)
Start-Button
Ladebalken
Verarbeitete Bilder angezeigt
Beide behalten?
Ordner-Auswahl
Nutzer-Entscheidung
Nächstes Paar
IPC-Nachricht
Bilder aus Ordner in ML-Pipeline
IPC-Nachricht
Child-Process
Duplikat verschoben in neuen Ordner
Backend (NodeJS)

Finaler Build

Dependencies

- Verschiedene Komponenten
- Vollständige Python-Integration
- Aufblasen einer kompakten App

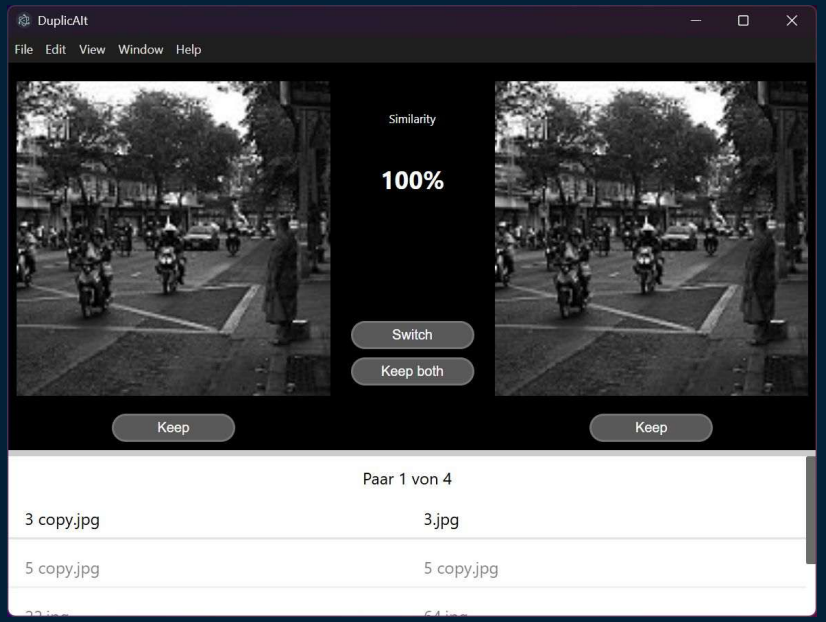
Distribution

- Unabhängige Anwendung
- Native Erfahrung im Zentrum
- Electron Forge bündelt alles

Finaler Build

Darstellung und UI

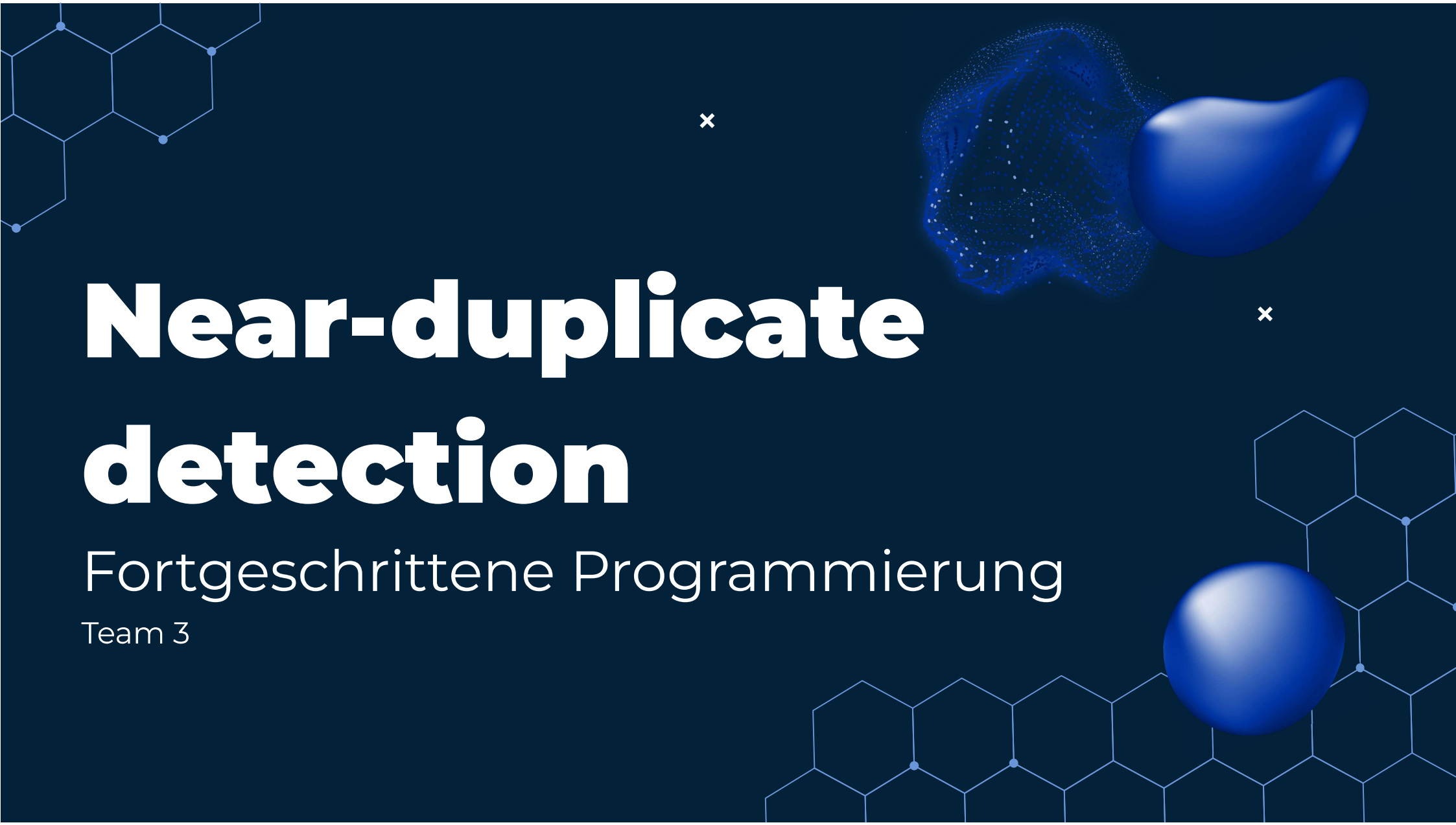
- Modernisierte Elemente
- Dunkles UI für hohen Kontrast
- Keine Überladung, organisiert
- Wichtige Kernpositionen



Demo

Literatur

[1] Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020). *A simple framework for contrastive learning of visual representations.* In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (Vol. 33, pp. 1597–1607). Curran Associates, Inc. https://arxiv.org/abs/2002.05709
[2] Le, Q. V., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G. S., Dean, J., & Ng, A. Y. (2012). *Building high-level features using large scale unsupervised learning*. In *Proceedings of the 29th International Conference on Machine Learning* (pp. 507–514). Omnipress.


Near-duplicate detection
Fortgeschrittene Programmierung
Team 3