Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

inovex

AI Labor - Sommersemester 2020

Computer Vision
“Sprintwechsel”

Robin Baumann, Matthias Richter

Karlsruhe, 3. April 2020

Agenda für Heute / die nächsten 2 Wochen

- Theorie:
  - Remote Arbeit Best Practices
  - Keras Sequence
  - Keras Functional API
  - Transfer Learning
  - Regularisierung
- Praxis: Wie löse ich ein Problem?
  - PointNet: Paper nachimplementieren
  - Benchmark: reduced ModelNet10
  - Praxis: Retraining mit neuen Klassen

Remote Best Practices

Remote (Meeting) Best Practices

- Kamera einschalten: ermöglicht nonverbale Kommunikation
  - Ausnahme: schwache Leitung
- Mikro aus, wenn Ihr nicht sprecht
  - Google Meets shortcut: Strg+D
- Benutzt Kopfhörer oder Headset
- Generell: Blogpost mit Tipps und Tools

Keras Sequence

Keras Sequence

Lazy, paralleles Laden von Datensätzen

- Große Datensätze passen nicht immer in den RAM
- Beim Training wird immer nur ein Batch benötigt
- `Sequence` abstrahiert Zugriff auf Datensatz:
  - Lädt nur den benötigten Batch
  - Kann eigenes shuffling implementieren
  - Augmentierung und Vorverarbeitung “on the fly”
- Was wir machen müssen:
  - Tracken der File-IDs
  - Auswahl und Shuffling auf Index

Keras Sequence

```
class ModelNetProvider(Sequence):
    """
    Lazily load point clouds and annotations from filesystem and prepare it for model trainin
    """

    def __init__(self, dataset, batch_size, n_classes, sample_size):...

    def __len__(self):...

    def __getitem__(self, index):...

    def __generate_data(self, batch_samples):...

    def on_epoch_end(self):...
```

Länge := Anzahl an “steps” in Epoche

index := Step in Epoche

Ausgliederung des Loading & Preprocessing

Am Ende jeder Epoche ausgeführt

Keras Functional API

Keras Functional API

- Mehr Flexibilität beim Erstellen der Modell-Architektur
  - z.B. Multi-Input/-Output Modelle/ Shared Layer etc.
- Layer-Instanzen sind als Funktion aufrufbar
  - erlaubt beliebige Verschachtelung
- Modell wird am Ende über inputs/outputs definiert
  - `Model(inputs=[...], outputs=[...])`

Keras Functional API

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
output_1 = Dense(64, activation='relu')(inputs)
output_2 = Dense(64, activation='relu')(output_1)
predictions = Dense(10, activation='softmax')(output_2)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```


https://keras.io/getting-started/functional-api-guide/ (24.10.2019)

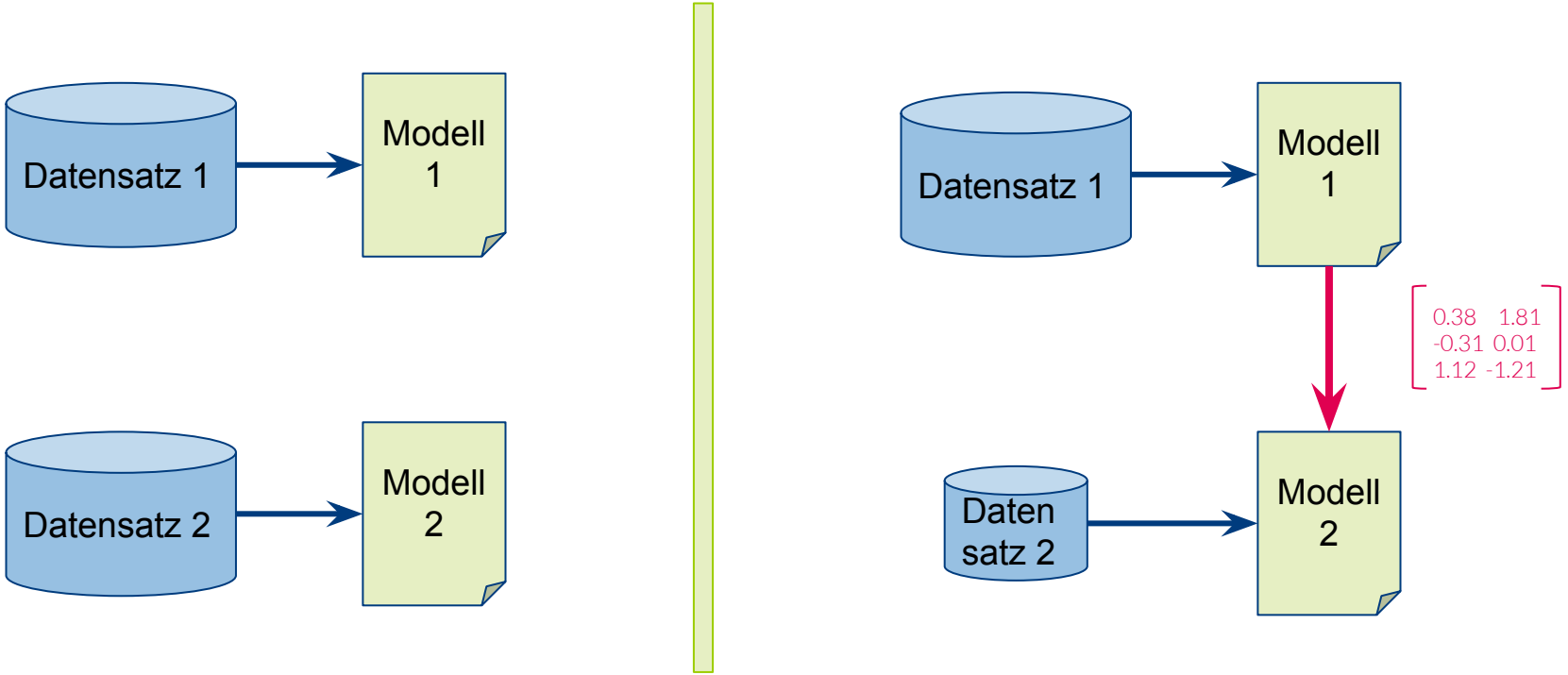
Transfer Learning

“After supervised learning - Transfer Learning will be the next driver of ML commercial success” — Andrew Ng (2016)

Motivation

- Deep (Supervised) Learning benötigt viele annotierte Trainingsdaten
- Datensätze wie ImageNet gibt es aber nicht für jede beliebige Anwendungsdomäne
- Vielleicht aber für eine verwandte Domäne?

Learn from Scratch vs Transfer Learning



inovex

Transfer Learning Strategien

“Wann kann / sollte ich Transfer Learning nutzen?”

1. Induktives Transfer Learning
   - Gleiche Domäne, unterschiedliche Tasks

2. Unsupervised Transfer Learning
   - Ähnlich zu 1., aber nur ein Task ist unsupervised

3. Transduktives Transfer Learning
   - Ähnlichkeit der Tasks, aber in unterschiedlichen Domänen

Transfer Learning Strategien

“Wie kann ich Information transferieren?”

- Instanz-Transfer
  - Direkte Übernahme von Trainingsbeispielen
  - Bsp.: Aggregation von gemeinsamen Subklassen
- Transfer von Merkmals-Repräsentationen
  - Wiederverwendung eines bereits gelernten Merkmalsextraktors
- Parameter-Transfer
  - Übernehme Parameter eines gesamten Modells als Ausgangspunkt

Transfer Learning in Keras

Parameter-Transfer; Neue Anzahl an Klassen

```
old_model = get_old_model()
old_model.load_weights('pretrained.h5')

new_model = get_new_model()
embedding = old_model.layers[-2]
outputs = Dense(num_classes,
                activation='softmax')(embedding.output)

new_model = Model(inputs=old_model.inputs,
                  outputs=outputs)
```

Regularisierung

Regularisierung

- Singularitäten → Modell unbrauchbar

$$\mathbf{w} = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \rightarrow \text{alle Features} = \mathbf{0} \rightarrow \text{Klassifikation unmöglich}$$

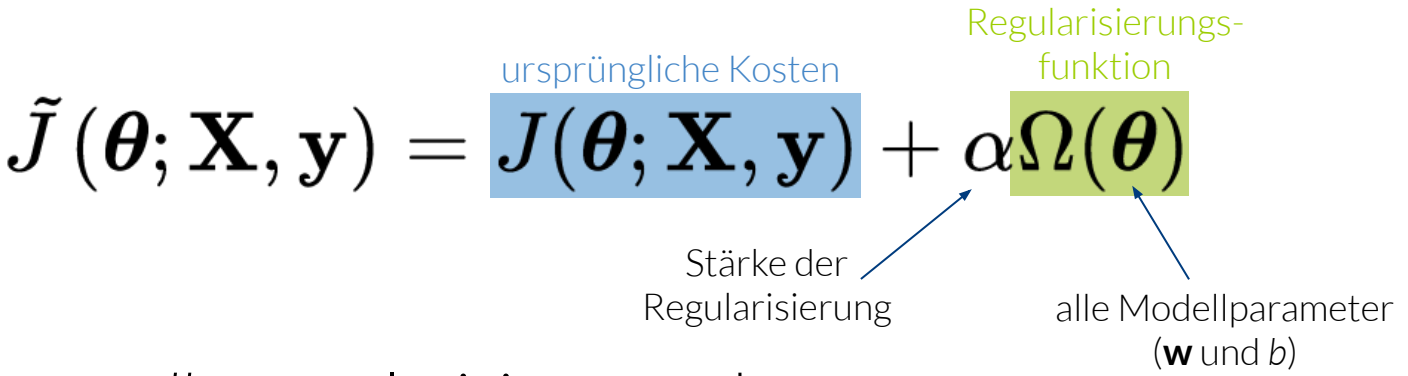
- Idee: Unerwünschte Parameter beim Lernen bestrafen

  → Teil der Loss-Funktion

Regularisierung

Allgemeine Notation

- Regularisierte Kostenfunktion:



- Es kann *alles* regularisiert werden
  - Gewichte (**w**), Bias (*b*), Zwischenergebnisse, Output (**y**), ...
  - Meistens werden nur die Gewichte (**w**) regularisiert

Typische Regularisierungen

Gegen Overfitting

- L^2 (aka “Weight Decay”, “Ridge”): $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{w}\|_2^2$

  → Keine großen Gewichte:

  $$\|\mathbf{w}\|_2^2 = w_{11}^2 + w_{12}^2 + \ldots + w_{nm}^2$$

- L^1 (aka“LASSO”, “Basis Pursuit”): $\Omega(\boldsymbol{\theta}) = \|\mathbf{w}\|_1$

  → Viele Gewichte = 0 (**w** ist dünn besetzt / *sparse*):

  $$\|\mathbf{w}\|_1 = |w_{11}| + |w_{12}| + \ldots + |w_{nm}|$$

Regularisierung

In Tensorflow

- Eigene Loss-Funktion definieren

```
model.compile(..., loss=loss_with_regularization)
```

- Besser: Layer-Parameter `kernel_regularizer`

```
def douglas_reg(w):
  return K.abs(K.sum(w) - 42)

net = Dense(23, kernel_regularizer=douglas_reg)(net)
```


tensorflow.org/api_docs/python/tf/keras/layers/Dense und tensorflow.org/api_docs/python/tf/keras/regularizers

PointNet

Motivation

- Typische Anforderungen von Deep Learning Modellen
  - Feste Größe / Dimensionalität der Eingaben
  - Feste Ordnung / Reihenfolge der Werte
  - Gilt für Punktwolken nicht immer
- Hotfix: 3D Occupancy bzw. Voxel-Grid samplen
  - → Input hat feste Größe und Ordnung
- Problem:
  - Wie Quantisieren (Informationsverlust vs Speicher)?
  - Unnötiger Rechenaufwand

PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

Charles R. Qi* Hao Su* Kaichun Mo Leonidas J. Guibas
Stanford University

Abstract

Point cloud is an important type of geometric data structure. Due to its irregular format, most researchers transform such data to regular 3D voxel grids or collections of images. This, however, renders data unnecessarily voluminous and causes issues. In this paper, we design a novel type of neural network that directly consumes point clouds, which well respects the permutation invariance of points in the input. Our network, named PointNet, provides a unified architecture for applications ranging from object classification, part segmentation, to scene semantic parsing. Though simple, PointNet is highly efficient and effective. Empirically, it shows strong performance on par or even better than state of the art. Theoretically, we provide analysis towards understanding of what the network has learnt and why the network is robust with respect to input perturbation and corruption.

1. Introduction

In this paper we explore deep learning architectures capable of reasoning about 3D geometric data such as point clouds or meshes. Typical convolutional architectures require highly regular input data formats, like those of image grids or 3D voxels, in order to perform weight sharing and other kernel optimizations. Since point clouds

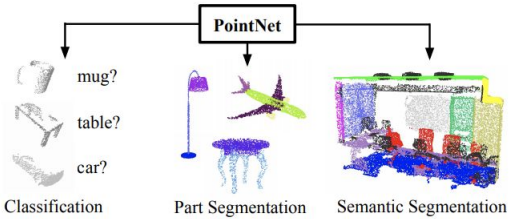
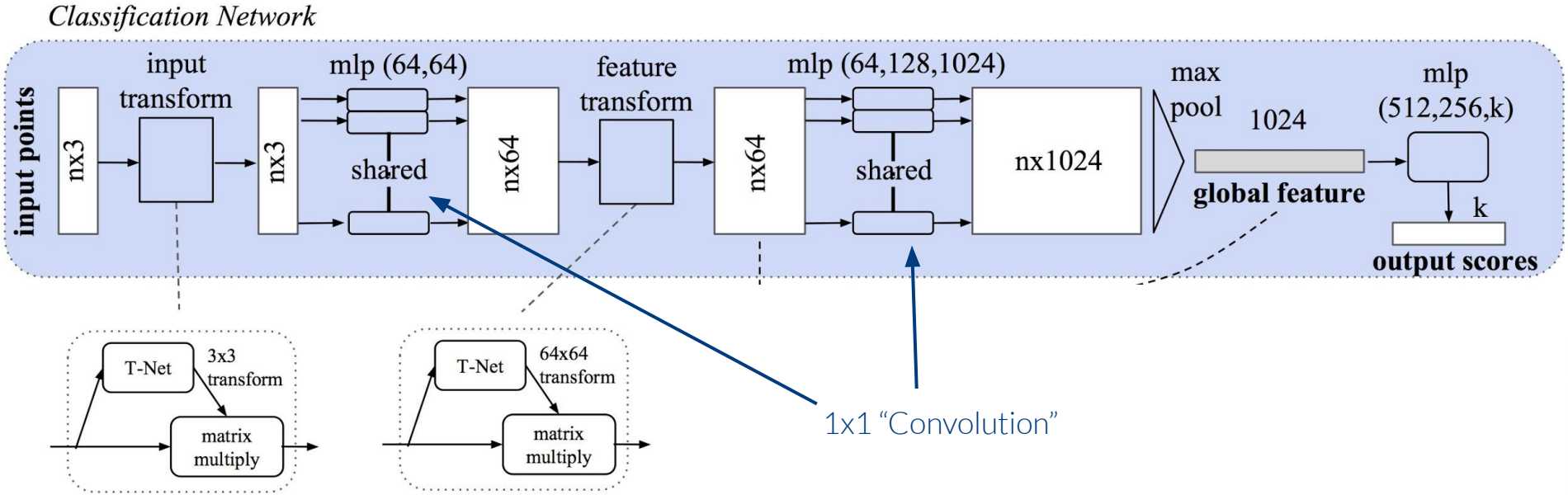


Figure 1. **Applications of PointNet.** We propose a novel deep net architecture that consumes raw point cloud (set of points) without voxelization or rendering. It is a unified architecture that learns both global and local point features, providing a simple, efficient and effective approach for a number of 3D recognition tasks.

still has to respect the fact that a point cloud is just a set of points and therefore invariant to permutations of its members, necessitating certain symmetrizations in the net computation. Further invariances to rigid motions also need to be considered.

Our PointNet is a unified architecture that directly takes point clouds as input and outputs either class labels for the entire input or per point segment/part labels for each point of the input. The basic architecture of our network is surprisingly simple as in the initial stages each point is processed identically and independently. In the basic setting each point is represented by just its three coordinates (x, y, z). Additional dimensions may be added

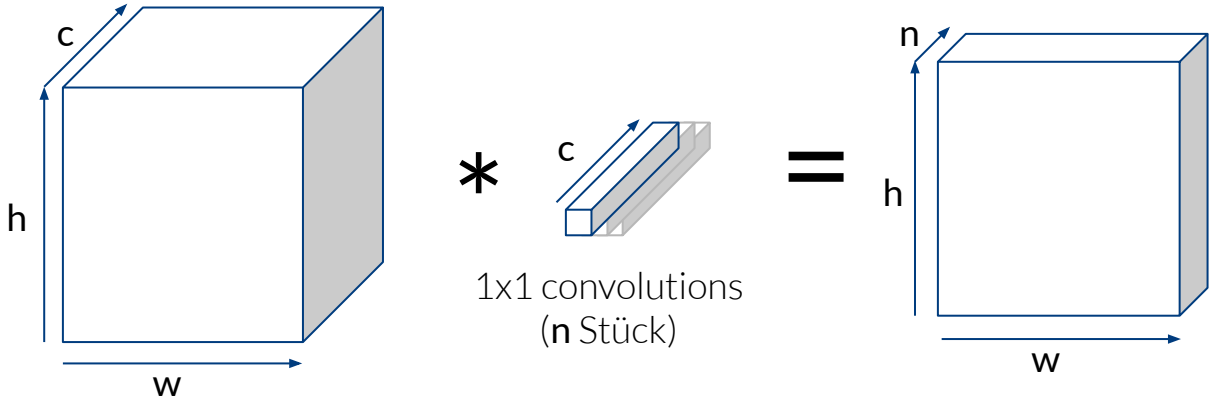


PointNet Architektur

inovex

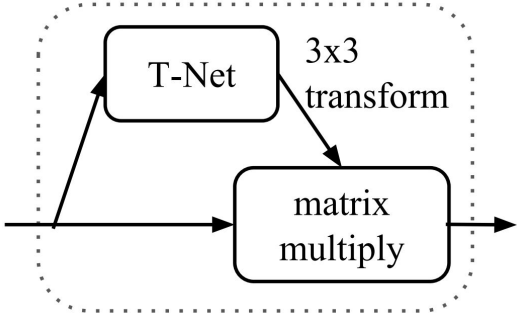
1x1 “Convolution”

- Fully Connected Layer → Bildgröße bleibt gleich
- Ein Pixel, aber viele Kanäle
  - Ein Output-Kanal ist Kombination aus allen Input-Kanälen



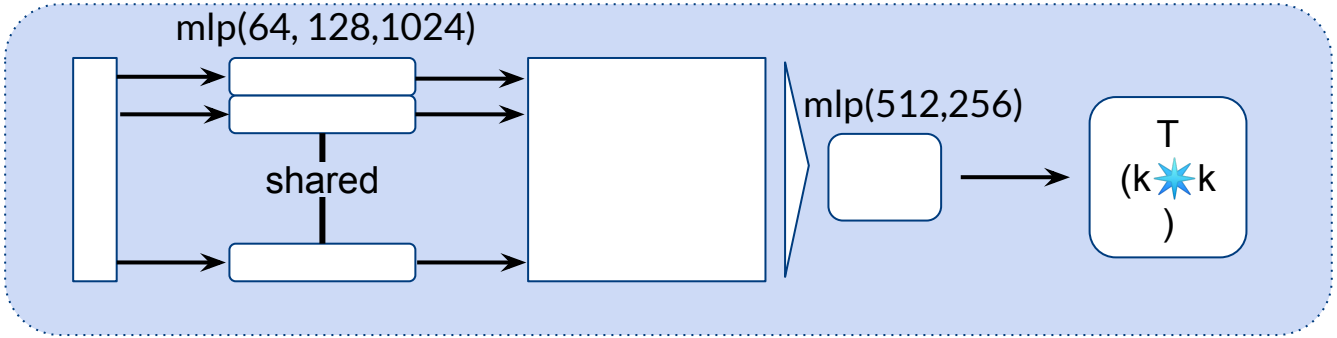
“In Convolutional Nets, there is no such thing as ‘fully-connected layers’. There are only convolution layers with 1x1 convolution kernels and a full connection table.” – Yann LeCun (2015)

T-Net?



- “Mini-PointNet”
- Gelernte orthogonale Rotationsmatrix

  → Normalisierung der Daten
- Regularisierung: $L_{reg} = \|I - AA^T\|_F^2$

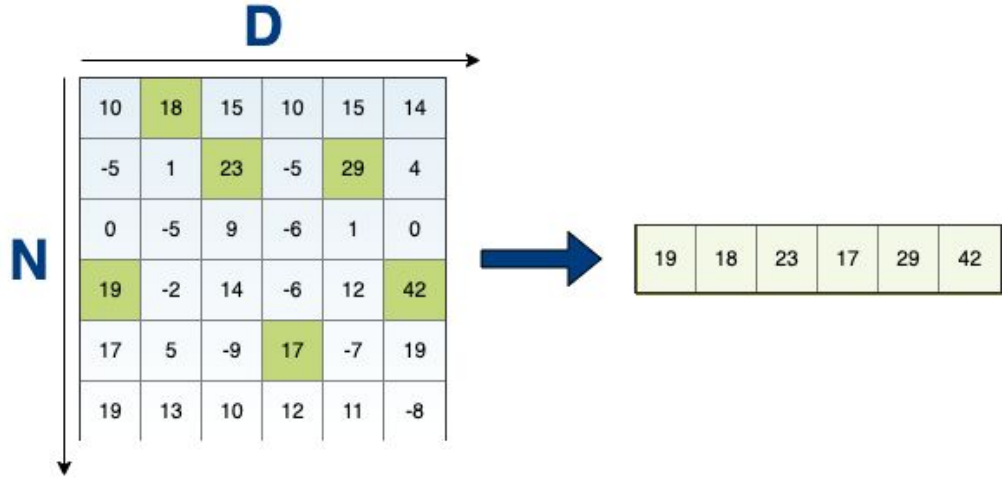


inovex

Der Vollständigkeit halber...

(1D) Max Pooling

- Input: N x D dimensionaler Tensor
- Output: 1 x D dimensionaler Tensor
- Symmetrische Funktion: Erzeugt “globale Signatur” der Daten

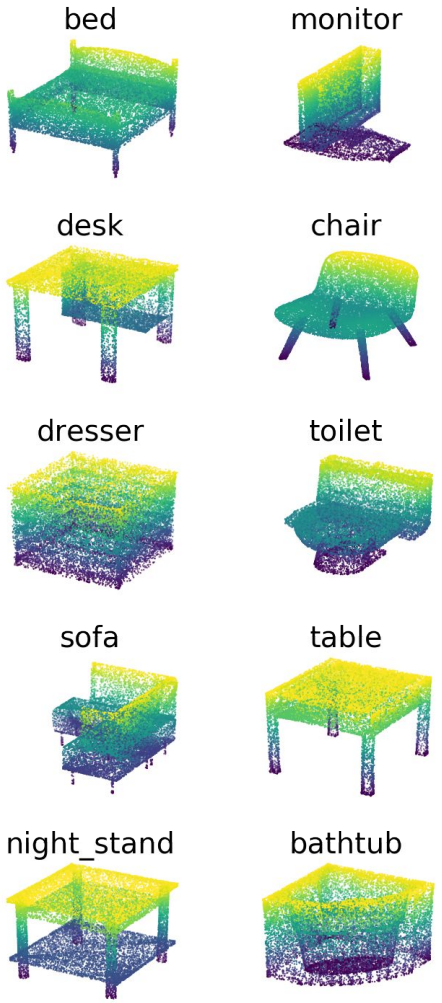


inovex

Datensatz:

ModelNet10

- 3D Meshes
- 10 unterschiedliche Klassen
- Training / Testing Split:
  - 3991 Training Samples
  - 908 Testing Samples
- https://modelnet.cs.princeton.edu



inovex

Aufgaben

TODOs implementieren:

1. PointNet-Architektur implementieren
2. Vortrainierte Gewichte laden
3. Merkmalsextraktor freezen und Classifier neu trainieren
4. Ergebnisse validieren

Feedback

https://forms.gle/DCbbKicvUZsDMc9y7

Vielen Dank

Robin Baumann
rbaumann@inovex.de

Matthias Richter
mrichter@inovex.de

inovex GmbH
Ludwig-Erhard-Allee 6
76131 Karlsruhe