Deep Learning on Meshes and Point Clouds

Ruben Wiersma

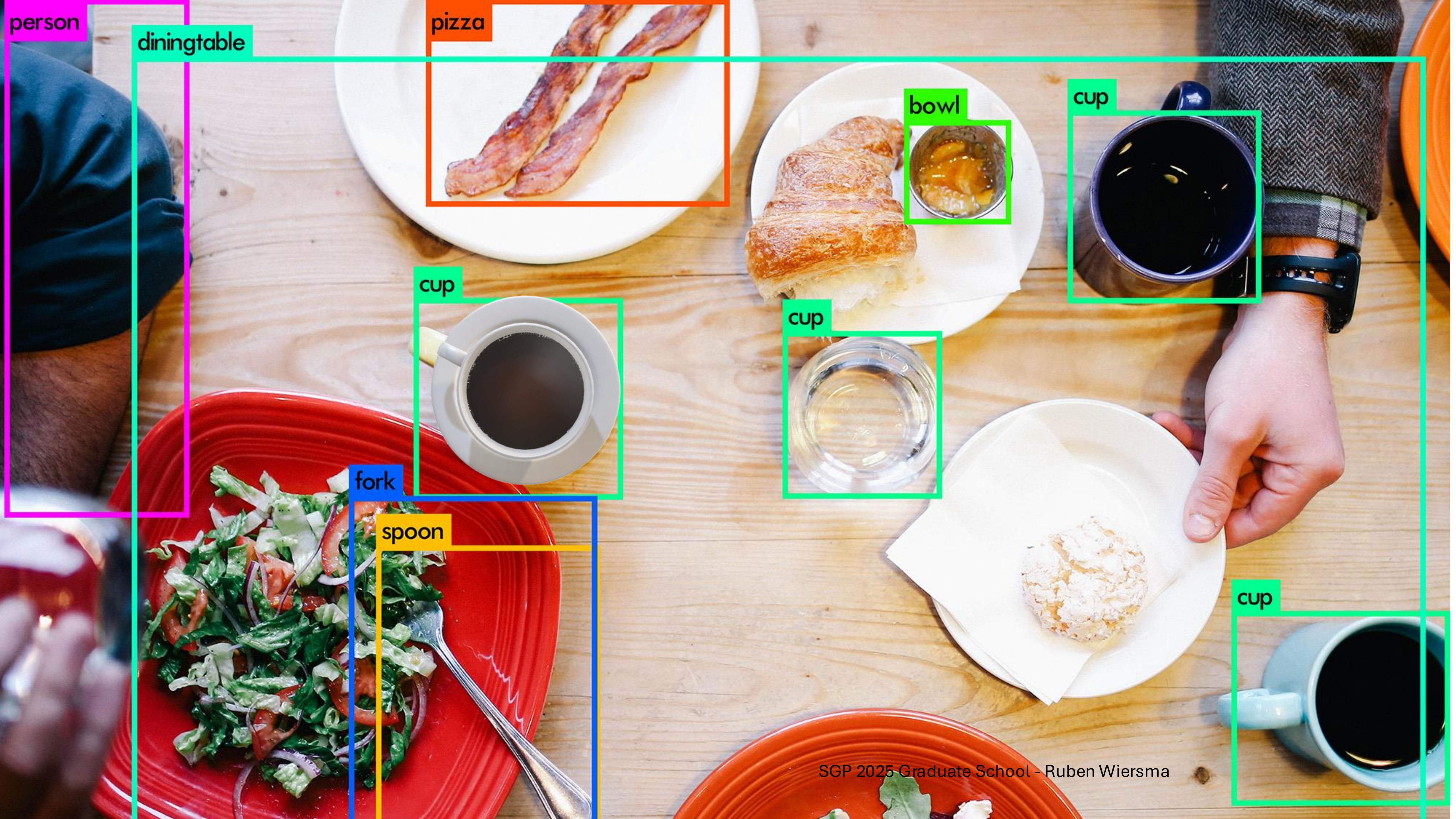
SGP 2025 Graduate School, Bilbao

Who are you?

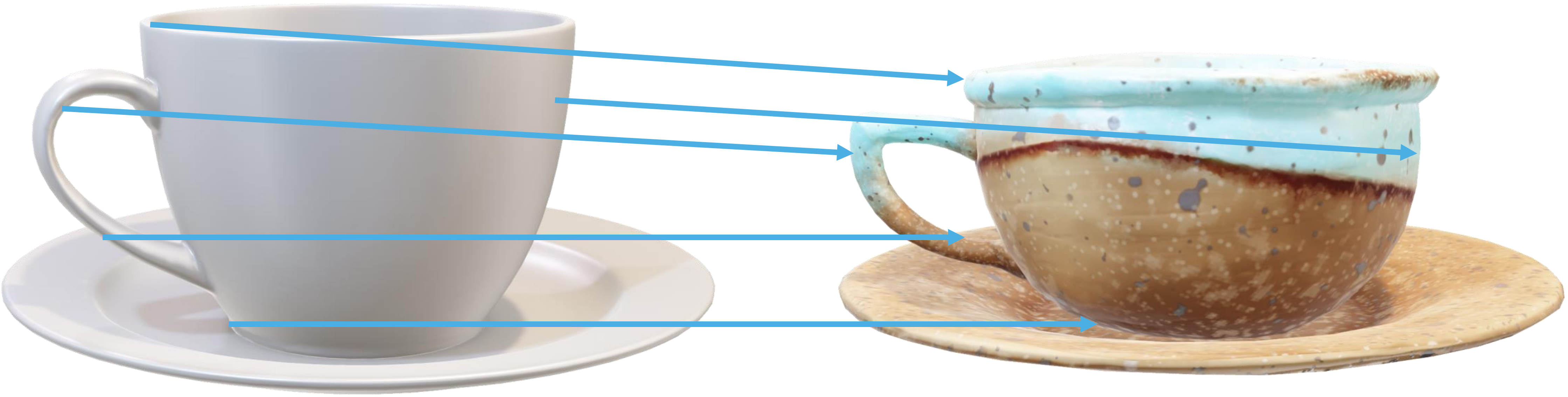
Goal

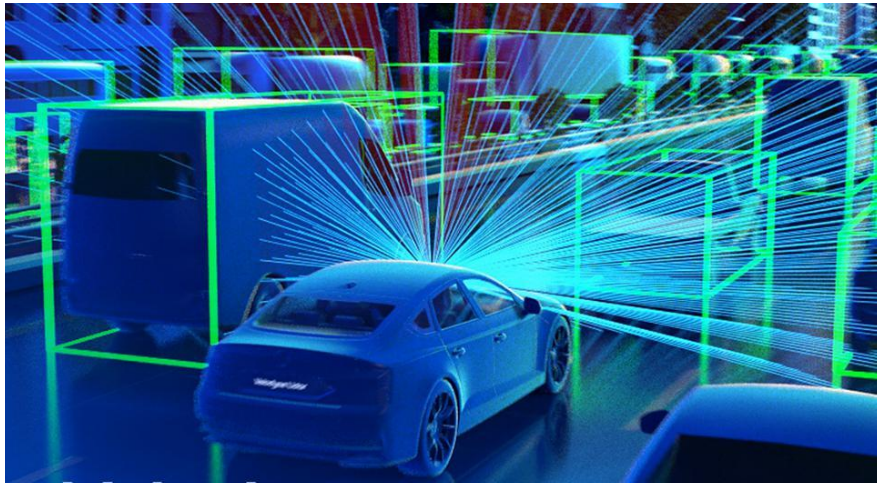
- Provide a ‘map’ of deep learning on 3D shapes
- Outcome
  - Applying deep learning to 3D tasks
  - Developing deep learning techniques for 3D tasks
- Audience
  - Some familiarity with optimization or machine learning

Why are we interested?

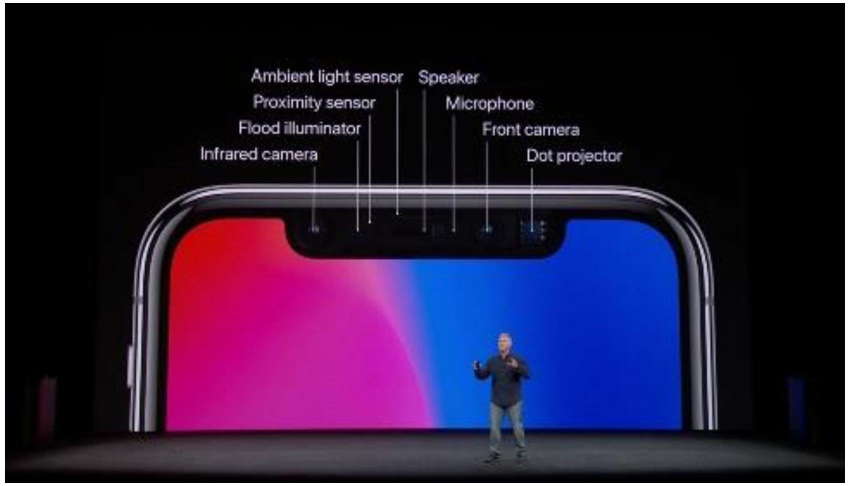

person
diningtable
pizza
bowl
cup
cup
cup
cup
fork
spoon


coffeecup
Cup
Handle
Plate

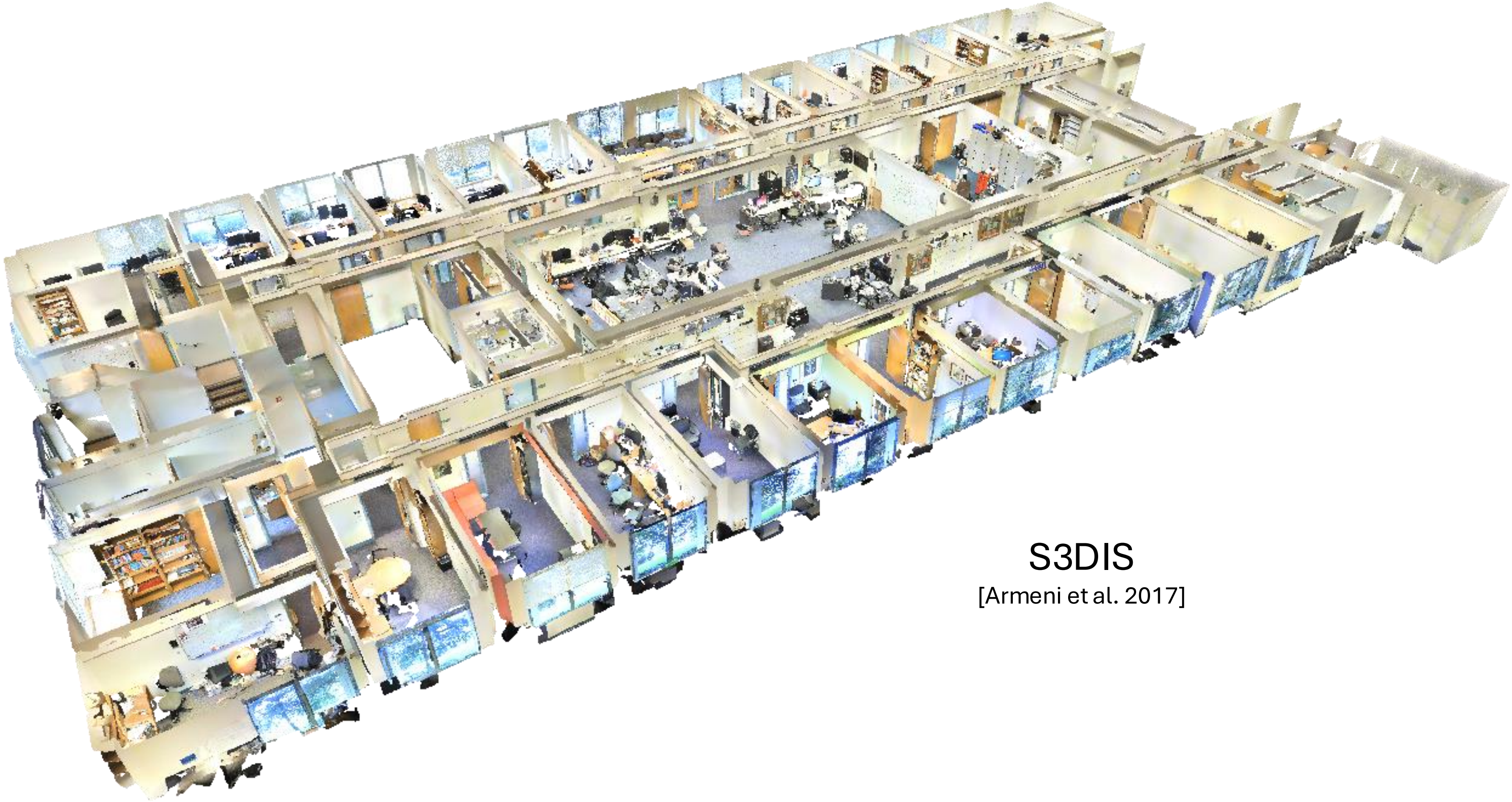





90 mm x 25 mm x 25 mm

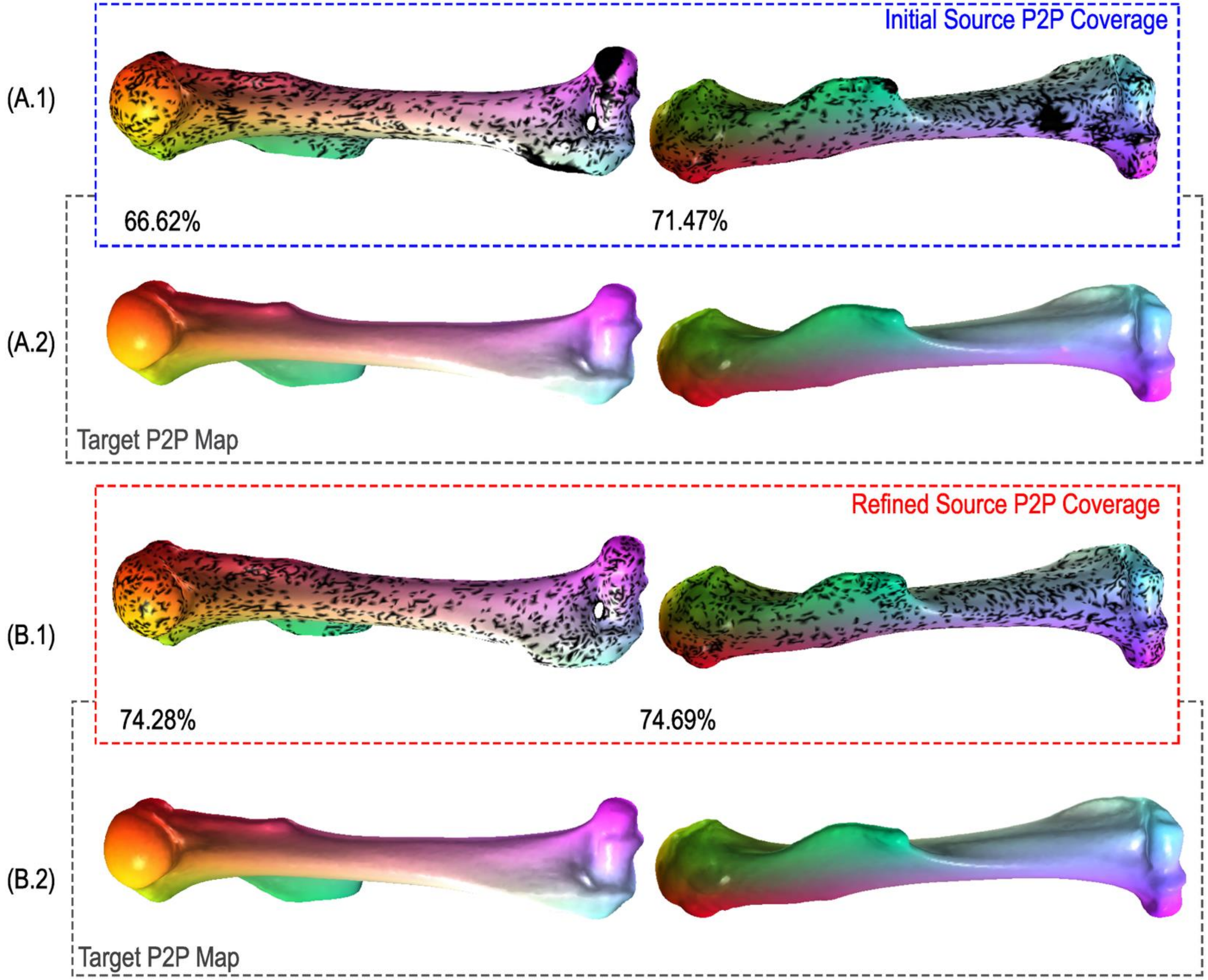

Ambient light sensor
Speaker
Proximity sensor
Microphone
Flood illuminator
Front camera
Infrared camera
Dot projector


From Bronstein

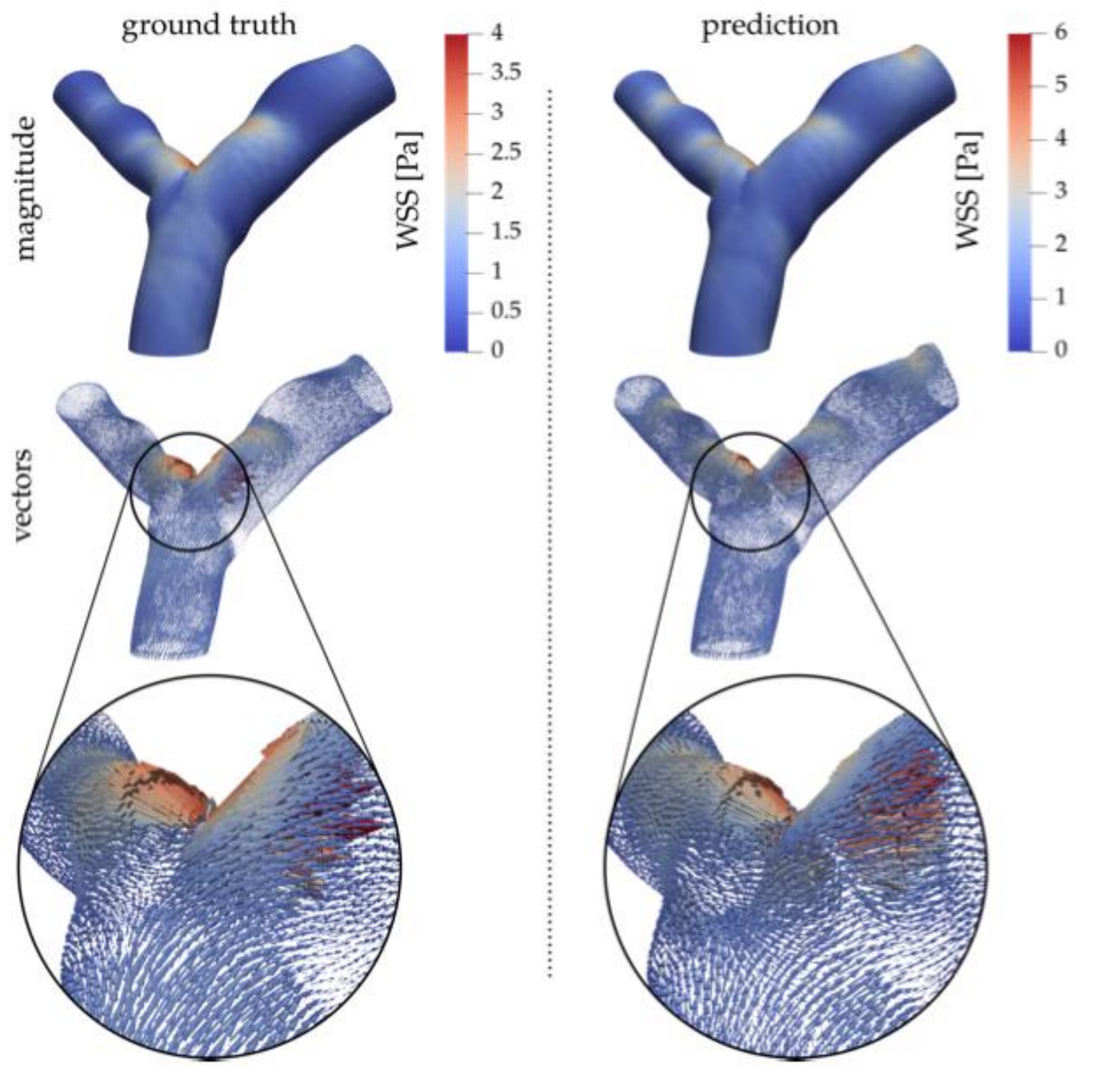


S3DIS
[Armeni et al. 2017]

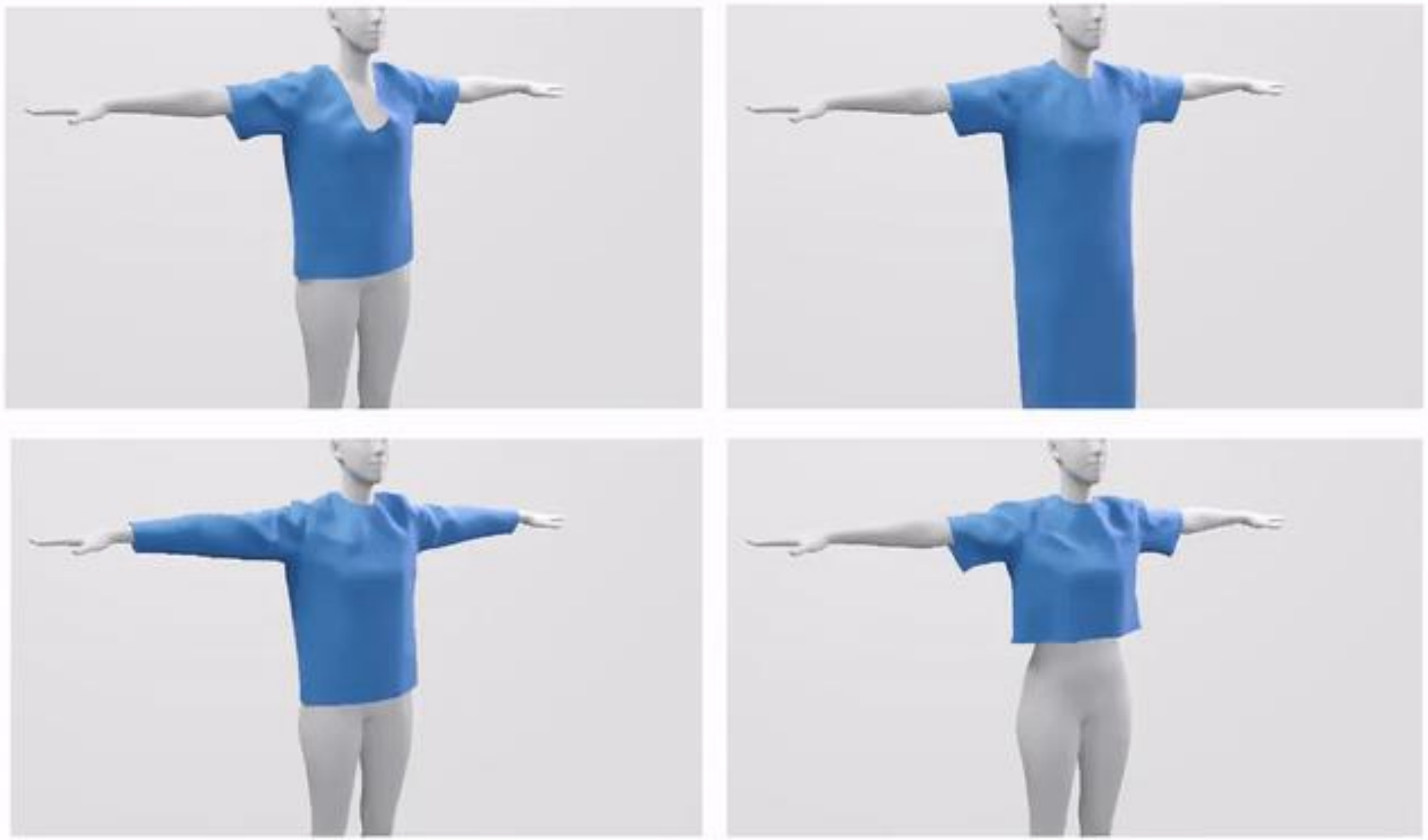
Understanding and Improving Features Learned in Deep Functional Maps, Attaiki and Ovsjanikov (2023)



Automated morphological phenotyping [...], O. Thomas et al. (2023)



Mesh Neural Networks for SE(3)-Equivariant Hemodynamics Estimation of the Artery Wall, Suk et al. (2022)

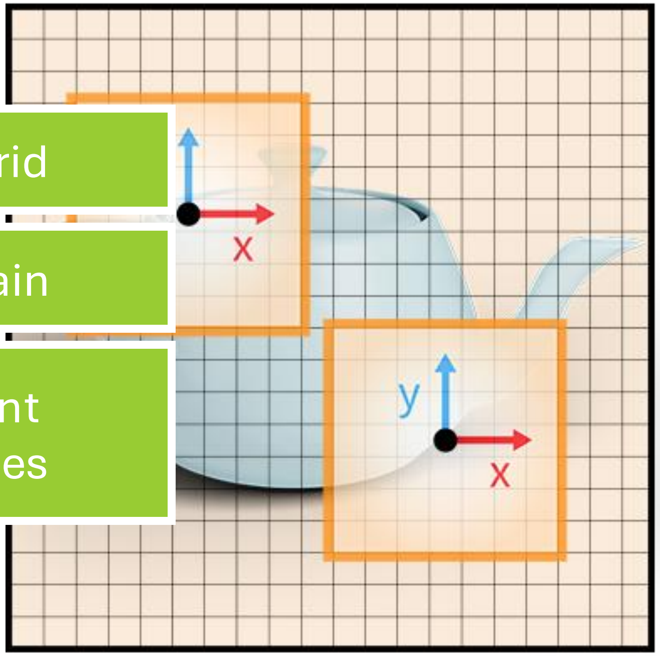


Fully Convolutional Graph Neural Networks for Parametric Virtual Try-On, Vidaurre et al. (2020)

Why are we interested?

- Classification
- Segmentation
- Registration, correspondence
- Surface reconstruction
- Speeding up classical geometry tasks (i.e., smart lookup table)
- Generative modeling

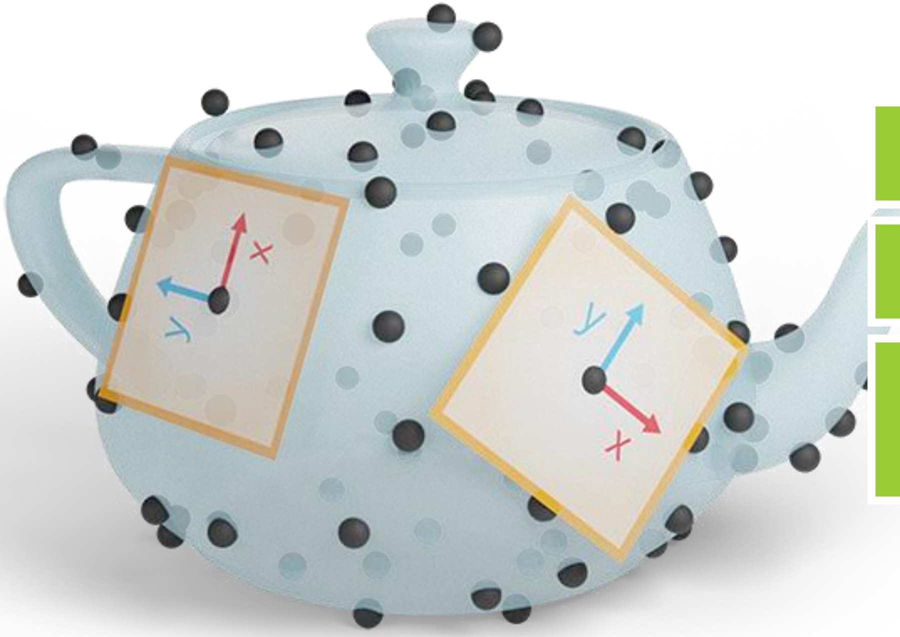
What’s the difference?



Regular grid

Flat domain

Consistent coordinates



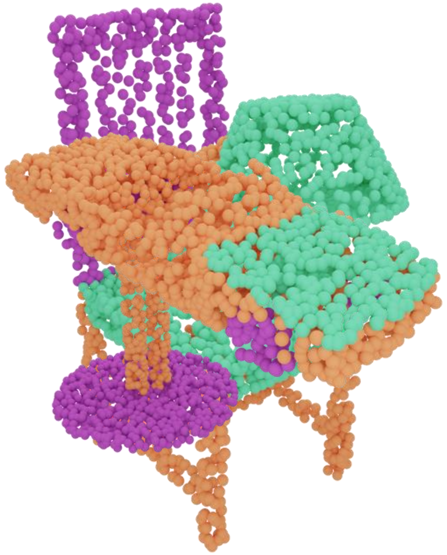
Irregular sampling

Curved domain

Local coordinates (charts)

Chapter 1: Basics

Deep learning is a machine learning method

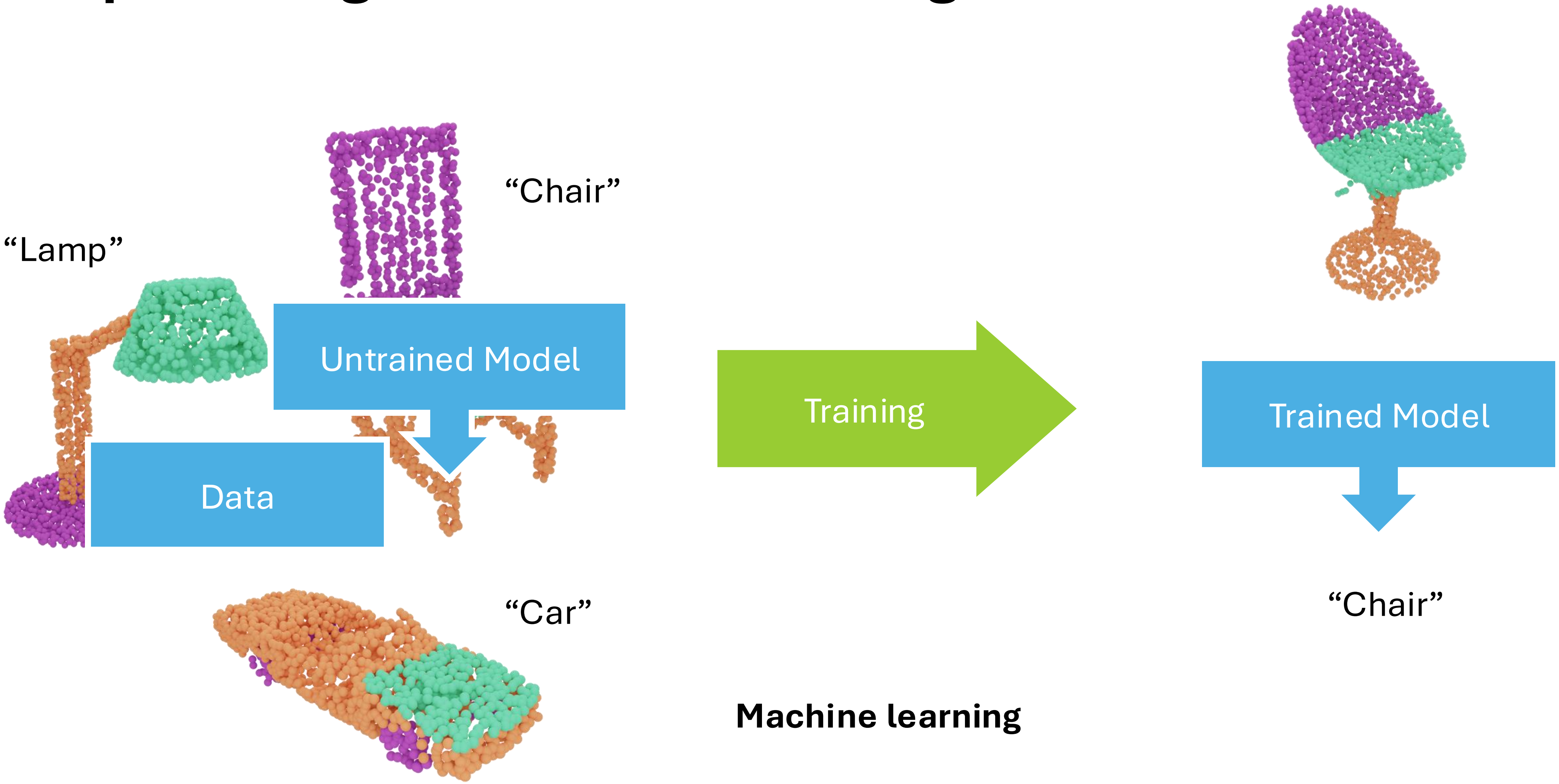


Algorithm

“Chair”
“Lamp”

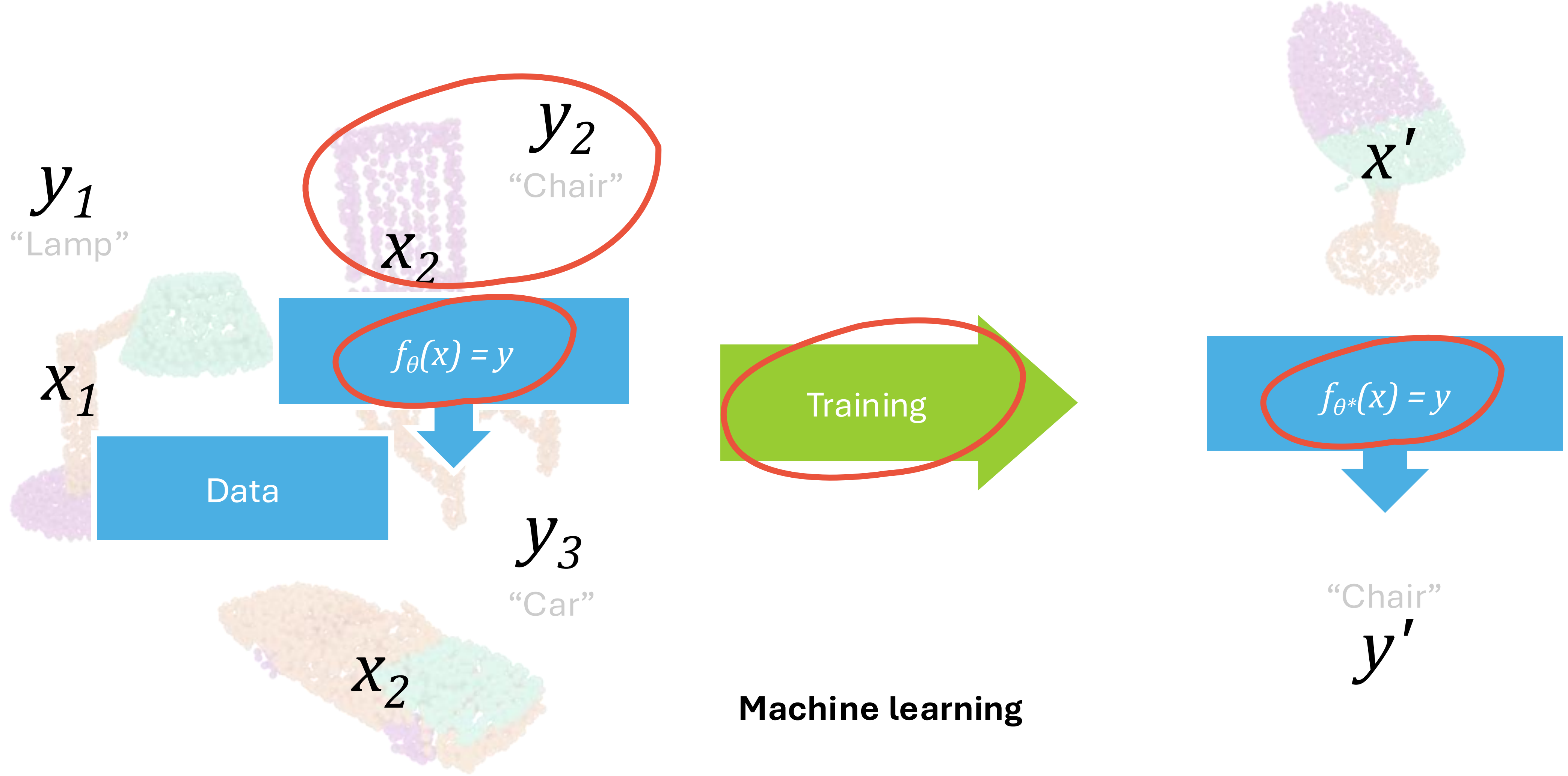
“Classic” approaches

Deep learning is a machine learning method



Machine learning

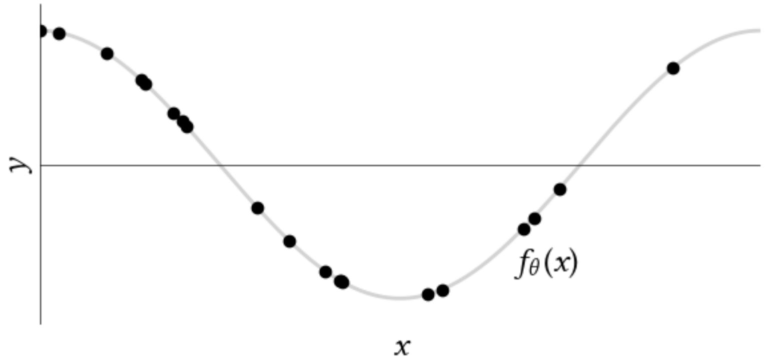
Deep learning is a machine learning method

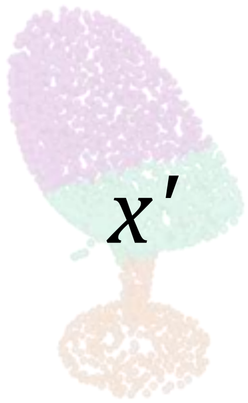


Machine learning

Machine learning basics

$f_\theta(x_i)$





$f_{\theta^*}(x) = y$

"Chair"

y'

Deep learning basics

θ are the unknowns, not x

$$\theta^* = \underbrace{\text{argmin}_\theta}_{\text{Training}} \underbrace{\sum_i^N \|\overbrace{f_\theta(x_i)}^{\text{Model}} - y_i\|_2^2}_{\text{Loss}}$$

- $f_\theta(x)$ is non-linear → numerical optimization
- θ is high-dimensional → gradient descent (instead of higher order methods)
- N is large → stochastic gradient descent (only a few x, y pairs at a time)

Characterization of deep learning

Data	Model	Optimizer	Hardware
Large datasets	Non-linear, layered	Gradient Descent	GPUs
	High # parameters	Backpropagation	
	End-to-end		

'Philosophy': Scaling (data, compute) beats algorithmic complexity

Is deep learning all you need?

Data	Model	Optimizer	Hardware
Small dataset	Non-linear, layered	Gradient Descent	GPUs
	High # parameters	Backpropagation	
	End-to-end		

Do you really need the complexity?

‘Philosophy’: Scaling (data, compute) beats algorithmic complexity

Is deep learning all you need?

Data	Model	Optimizer	Hardware
Small dataset	Linear model	Gradient Descent	GPUs

Better optimizers are available

Tip: Numerical Algorithms (Solomon)

‘Philosophy’: Scaling (data, compute) beats algorithmic complexity

“It’s all about the data” – Alexei Efros

e.g., https://www.youtube.com/watch?v=M1VHu1d4sGQ

Data	Model	Optimizer	Hardware
Large datasets	kNN		GPUs

‘Philosophy’: Scaling (data, compute) beats algorithmic complexity

Characterization of deep learning

Data	Model	Optimizer	Hardware
Large datasets	Non-linear, layered	Gradient Descent	GPUs
	High # parameters	Backpropagation	
	End-to-end		

‘Philosophy’: Scaling (data, compute) beats algorithmic complexity

Typical models

- MLP – Multi-layer perceptron
  - The Perceptron: A Probabilistic Model For Information Storage And Organization in the Brain – Rosenblatt (**1958**)
- CNN – Convolutional Neural Network
  - Gradient-based learning applied to document recognition – Lecun et al. (**1998**)
- Transformer (Attention)
  - Attention Is All You Need – Vaswani et al. (**2017**), attention was around before that

Multi-Layer Perceptron (geometric perspective)

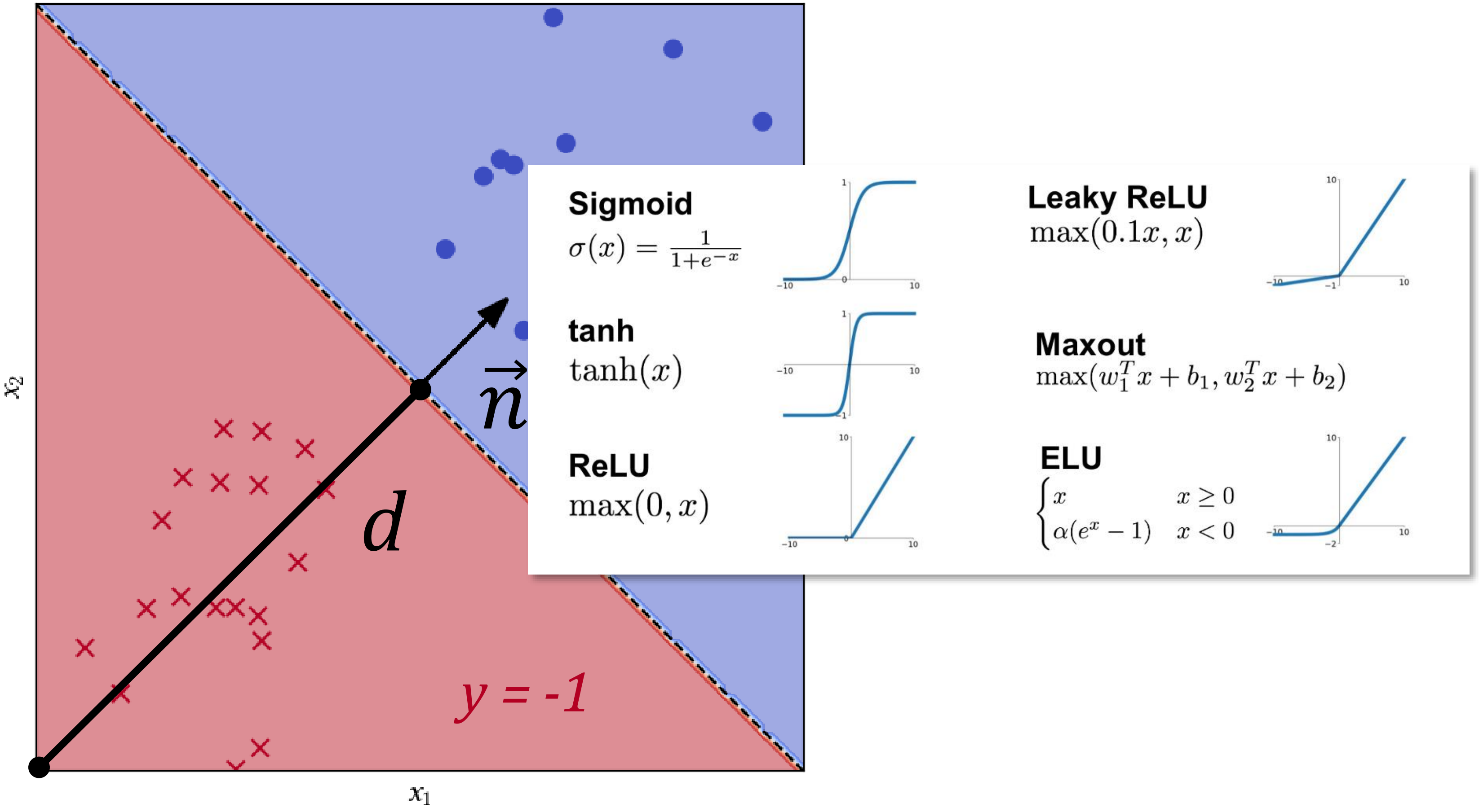
→ Separate these points

- Distance to line/plane

  $$\vec{x} \cdot \vec{n} - d$$

- Step function

  $$y = \sigma(\vec{x} \cdot \vec{n} - d)$$



https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092

Multi-Layer Perceptron (geometric perspective)

→ Separate these points

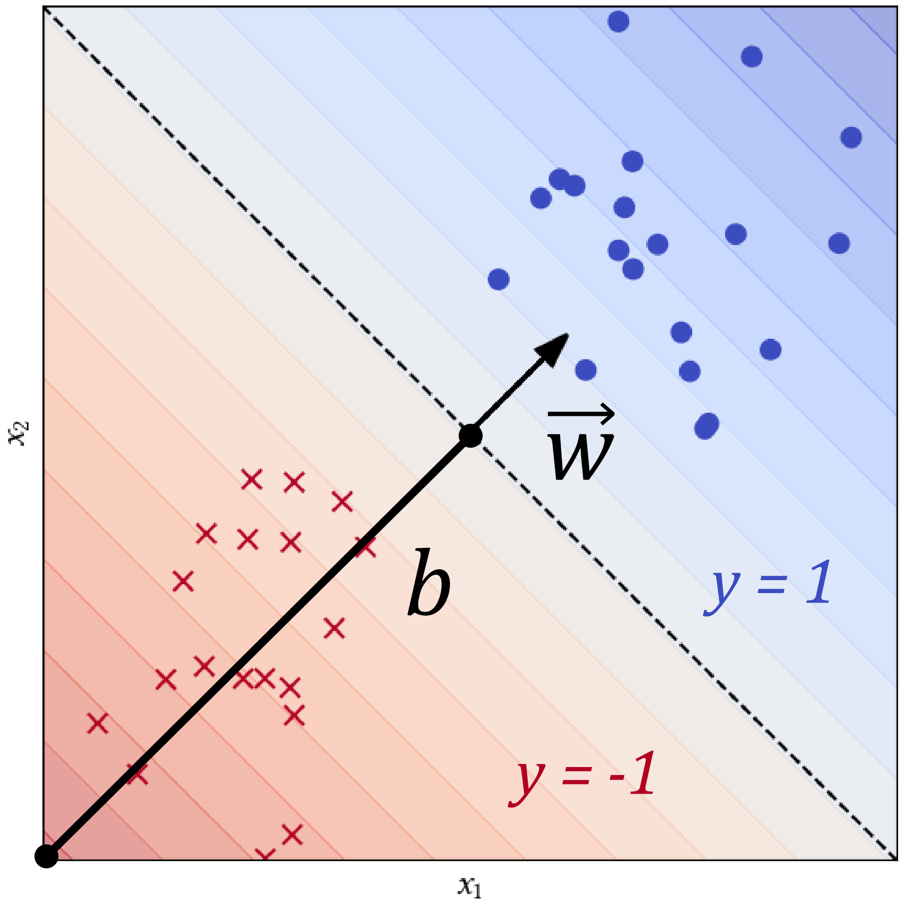
- Distance to line/plane

$$\vec{w}^T\vec{x} + b$$

- Step function

$$y = \sigma(\vec{w}^T\vec{x} + b)$$

- Weights, biases
- σ activation function
  - Non-linear!



Multi-Layer Perceptron

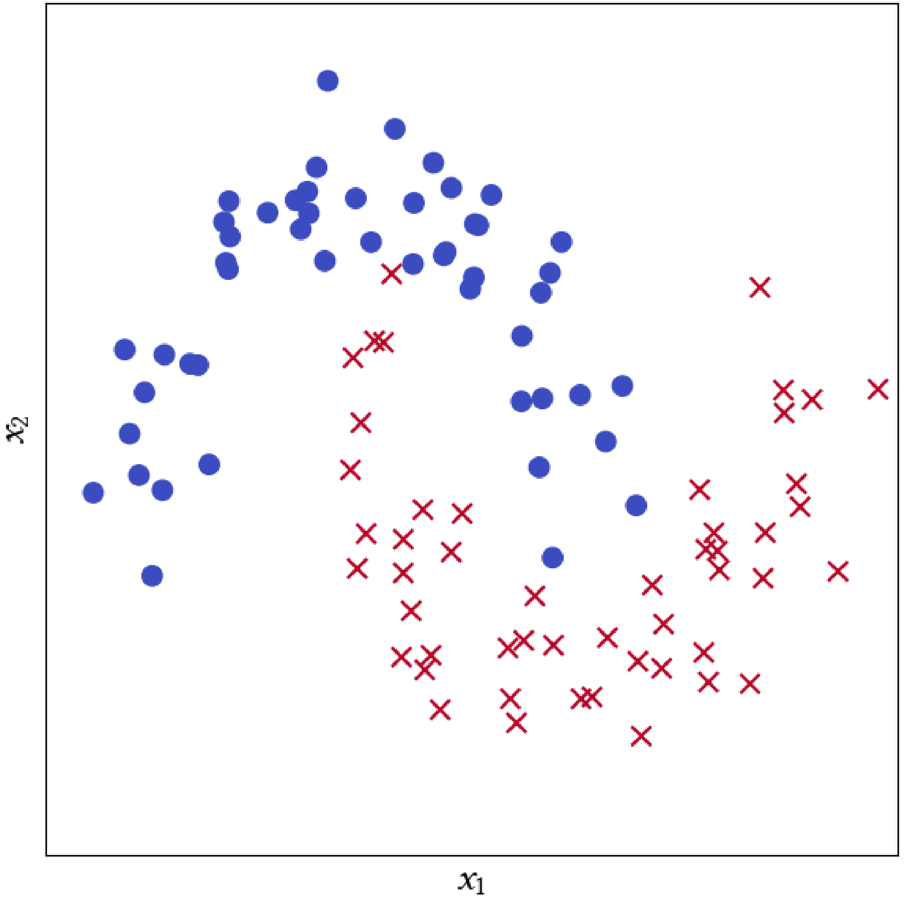
$$y = \sigma(\vec{w}^T \vec{x} + b)$$

→ Multiple outputs? | "Channels"

$$\vec{y} = \sigma(W\vec{x} + \vec{b})$$

→ Not linearly separable? | "Layers"

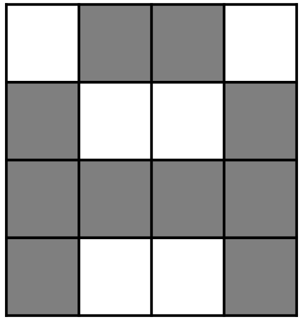
$$\vec{y} = \underbrace{\qquad\qquad\qquad}_{\text{Hidden layers}} \sigma\left(W_1\vec{x} + \vec{b}_1\right)$$



Multi-Layer Perceptron Notes

- Examples in 2D, but $\vec{x}$ can be of any dimension
- Interpretation: cutting up space with halfspaces
- "Multilayer Feedforward Networks [MLPs] are Universal Approximators"
  - Hornik, Stinchcombe and White (1989)
  - Non-linearity is necessary!
  - Compare, e.g., Fourier transform
- Higher complexity, more risk of overfitting

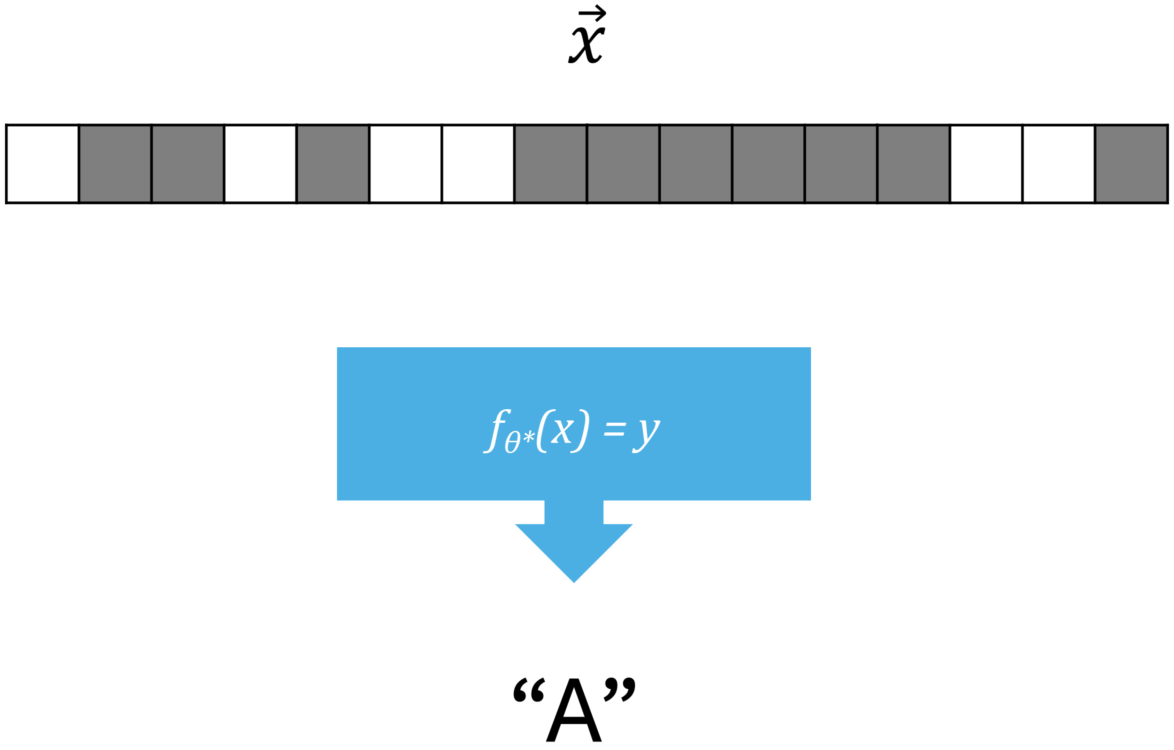
MLP on images



$f_{\theta^*}(x) = y$

“A”

MLP on images

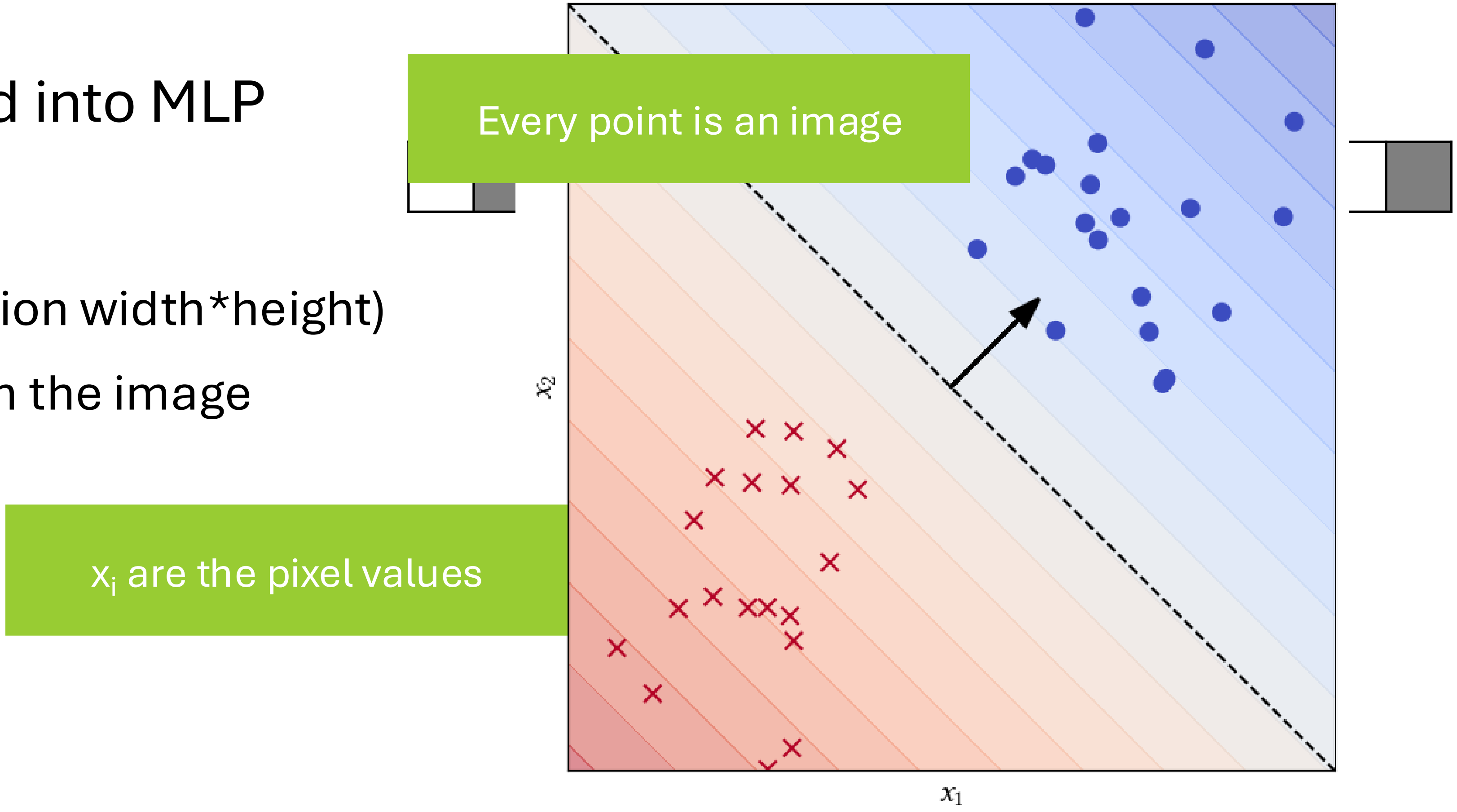


igl INTERACTIVE GEOMETRY LAB

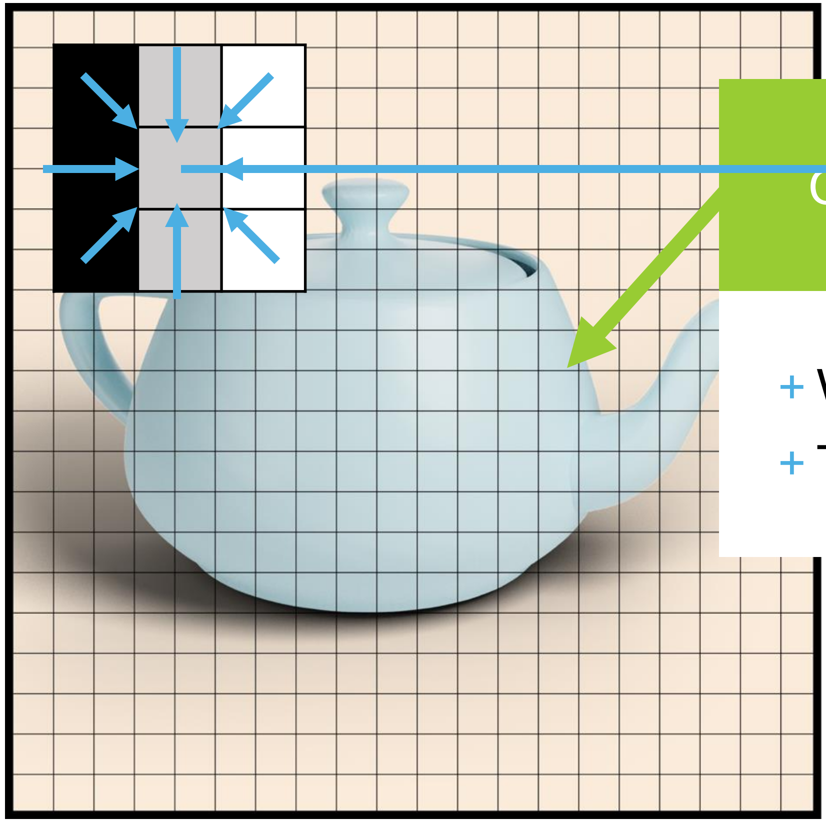
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

MLP on images

- Stack all pixel values, feed into MLP
- Problem?
  - Not efficient ($\vec{x}$ is of dimension width*height)
  - Patterns can be anywhere in the image

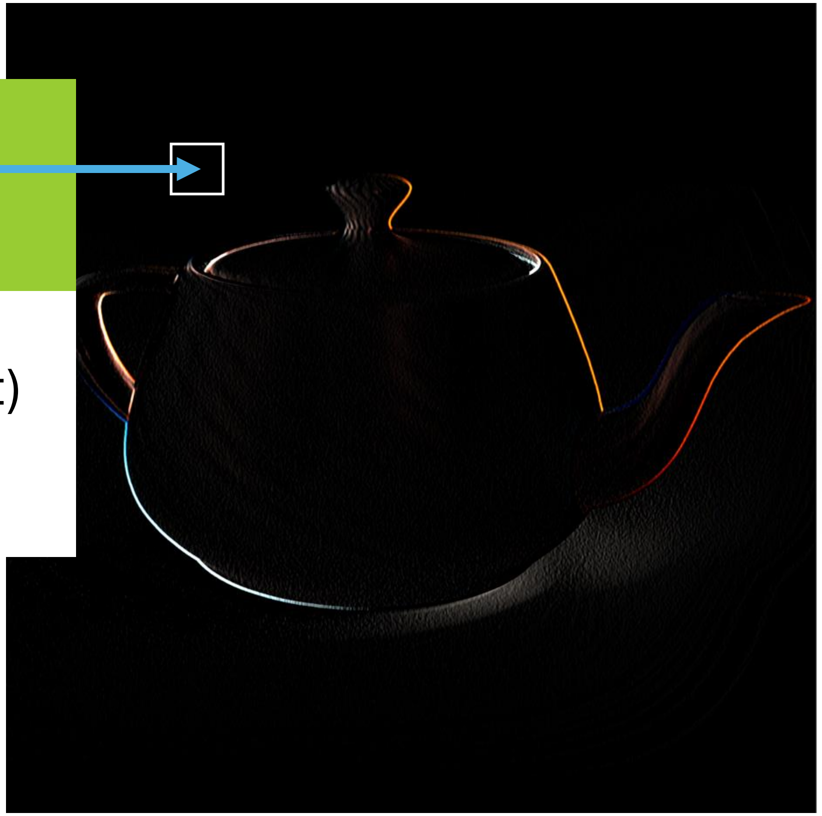


Convolutional Neural Network



Optimize kernel weights

+ Weight sharing (efficient)
+ Translation invariance

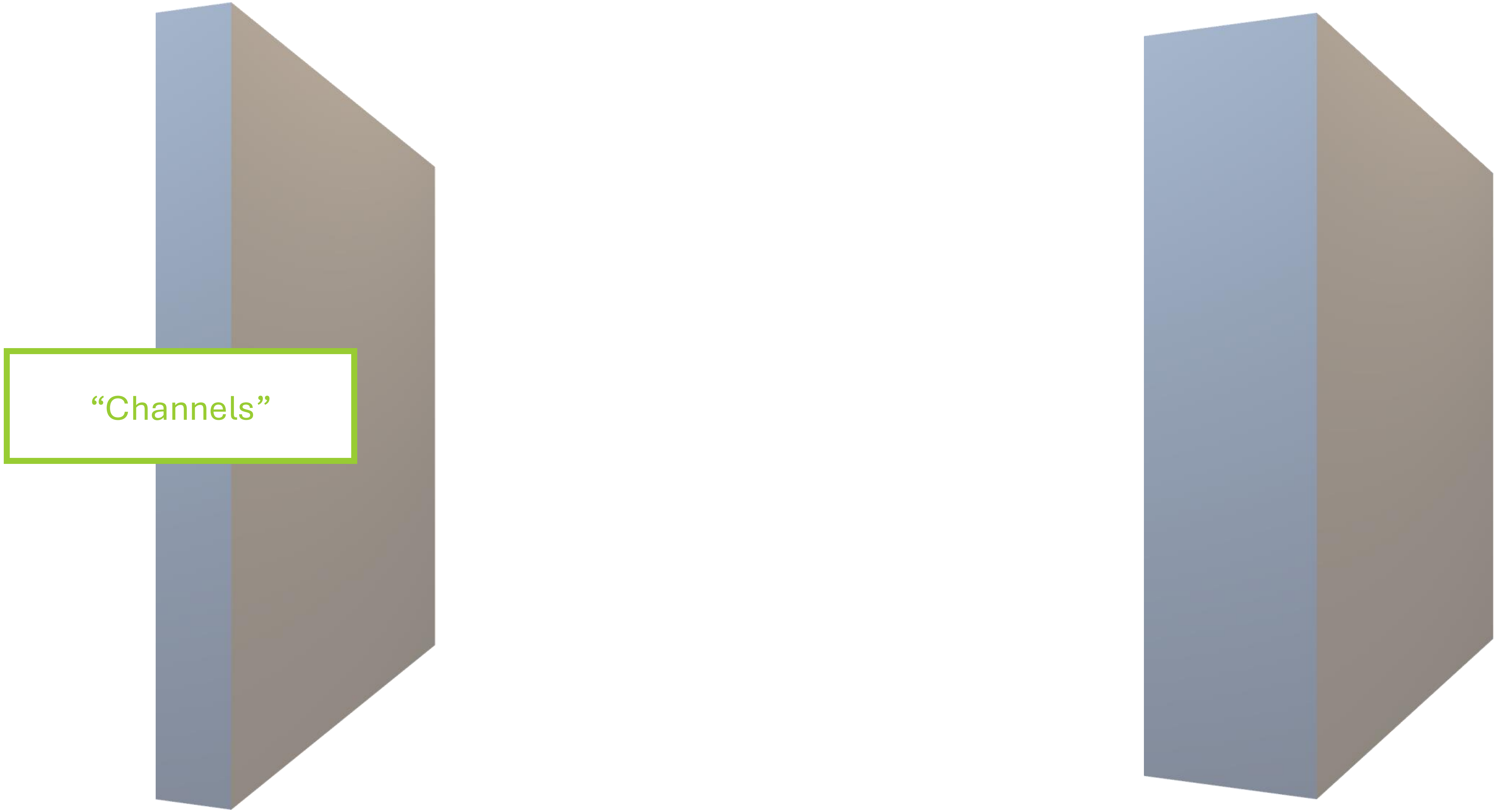


Schematics

INTERACTIVE GEOMETRY LAB

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

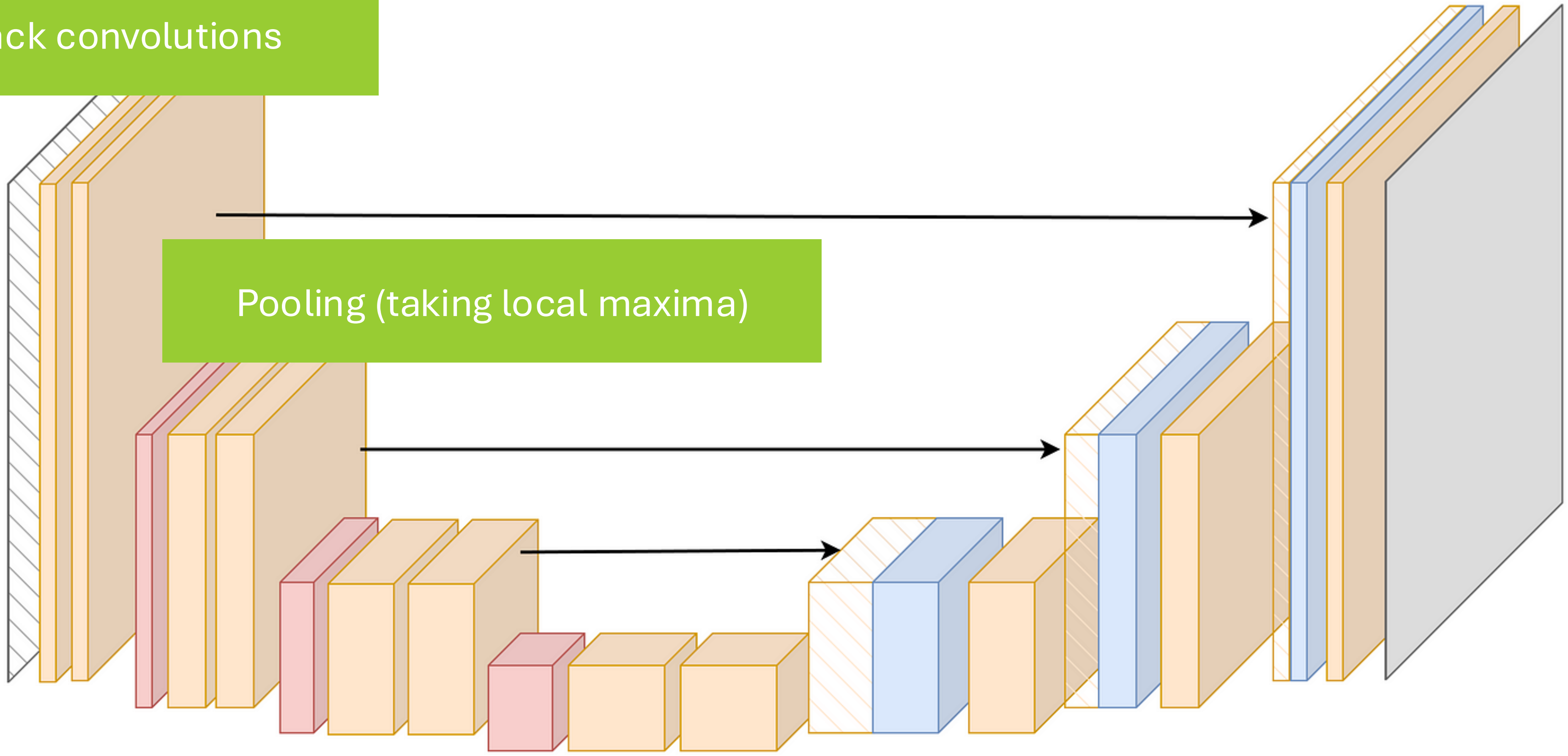
Schematics



Convolutional Neural Networks

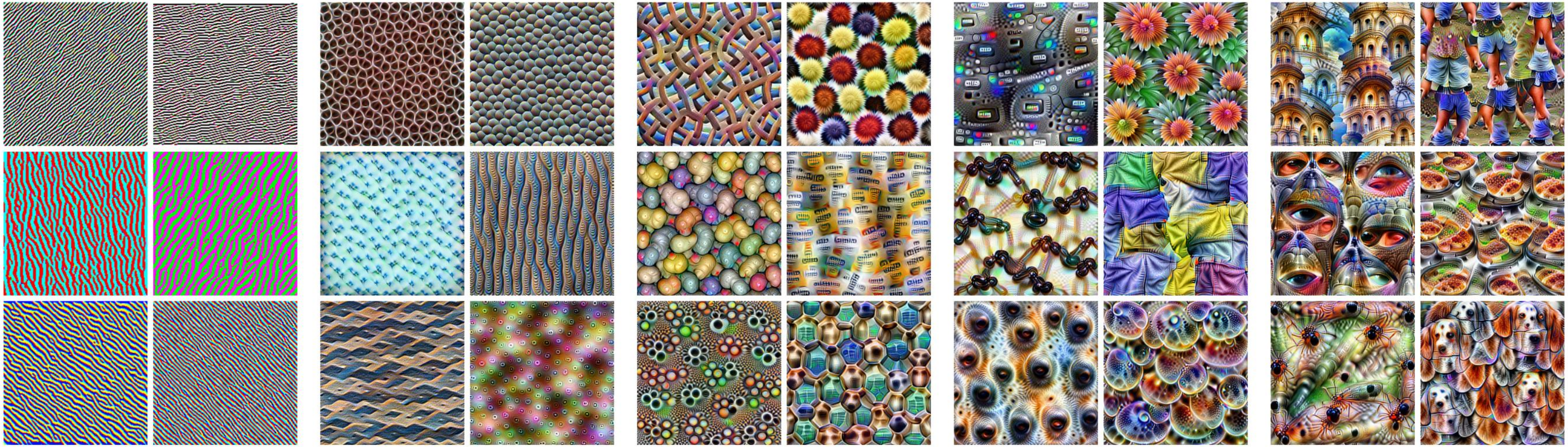
Stack convolutions

Pooling (taking local maxima)



https://medium.com/data-science/u-net-explained-understanding-its-image-segmentation-architecture-56e4842e313a

Hierarchies of Features in CNNs

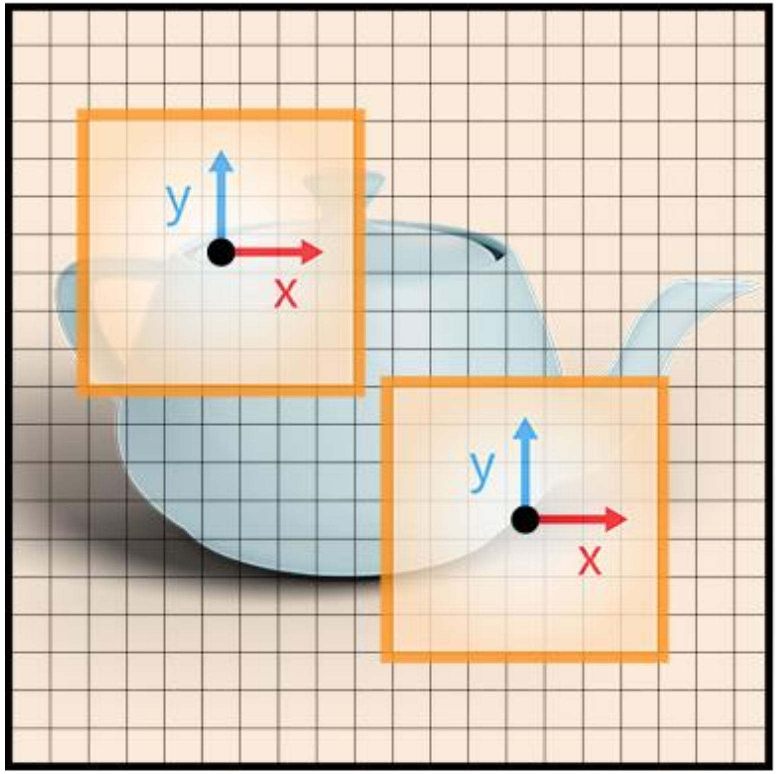


Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

Feature Visualization, Olah, Mordvintsev, Schubert (2017) – https://distill.pub/2017/feature-visualization/

Convolutional Neural Networks Notes

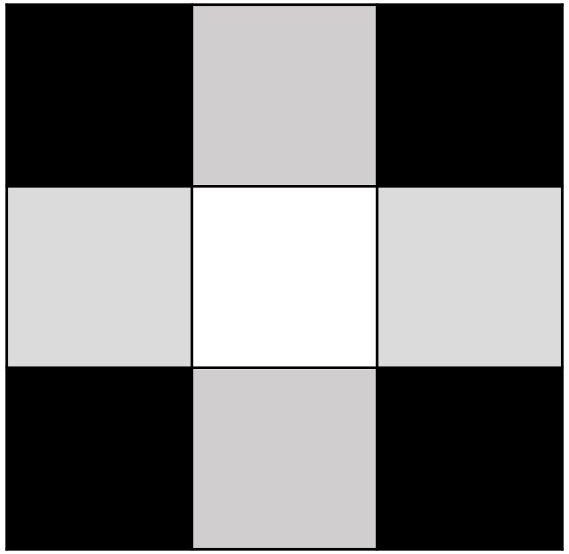
- Interpretation: MLP on patches
- Consistent pixel grid helps
  - Orientation
  - Maps to hardware well
- Downside (?) Template is fixed



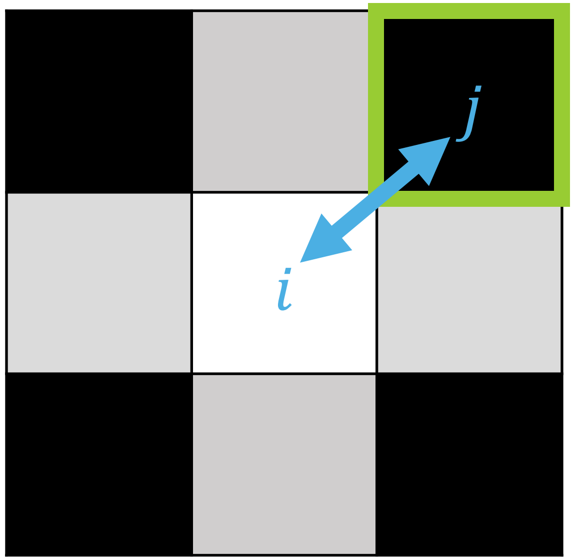
Limitation of Convolution Kernels

Kernel values based on location

Can we adapt to the signal?



Limitation of Convolution Kernels



Gaussian kernel g

$$x_i' = \sum_{j \in \mathcal{N}_i} \boxed{g\left(\underline{\|p_j - p_i\|}\right)} x_j$$

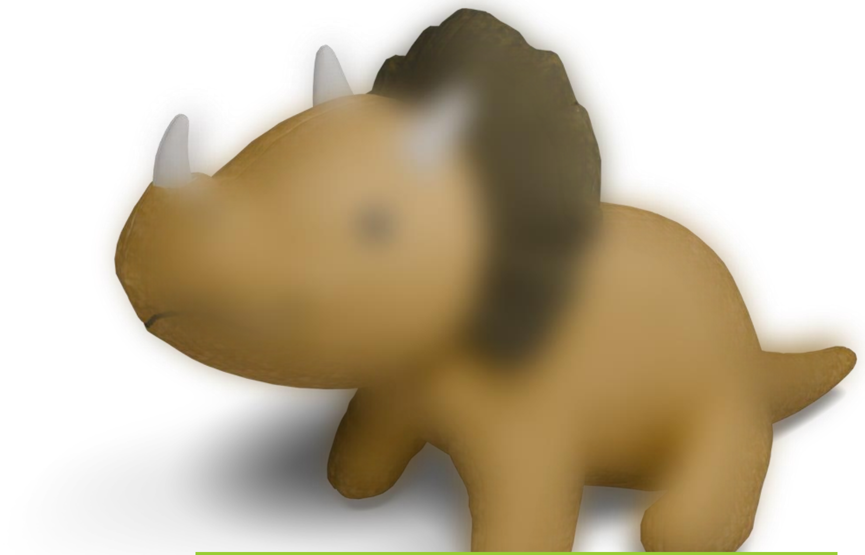
Based on distance

Denoising with Gaussian filter vs. Bilateral filter

$$x_i' = \sum_{j \in \mathcal{N}_i} g(\|p_j - p_i\|) x_j$$

$$x_i' = \sum_{j=0}^{N} \boxed{f(\|x_j - x_i\|)} g(\|p_j - p_i\|) x_j$$

Look at pixel value as well



Isotropic blurring

Preserves features

Self-attention (Transformers)

$$x_i' = \sum_{j=0}^{N} f\left(\boxed{\|x_j - x_i\|}\right) g(\|p_j - p_i\|) x_j$$

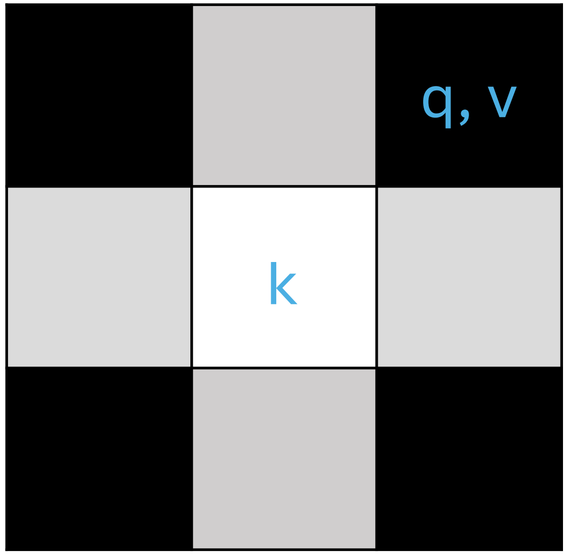
- Replace distance with cosine similarity

$$x'_i = \sum_{j=0}^{N} \boxed{\left(x_j^T x_i\right)} x_j$$

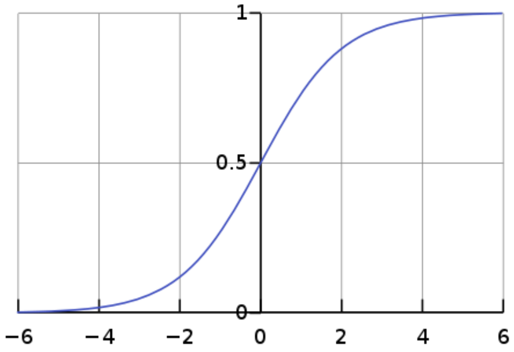
- Add flexibility with weight matrices and make sure weight 'behaves'

$$x_i' = \sum_{j=0}^{N} \mathrm{softmax}\left(\left(\mathrm{W}_q x_j\right)^T \mathrm{W}_k x_i\right) \mathrm{W}_v x_j$$

Self-attention (Transformers)



$$x_i' = \sum_{j=0}^{N} \underbrace{\text{softmax}}\Big(\underbrace{(W_q x_j)}_{\text{Query}}^T \underbrace{W_k x_i}_{\text{Key}} \Big) \underbrace{W_v x_j}_{\text{Value}}$$



Self-attention (Transformers)

$$x_i' = \sum_{j=0}^{N} \mathrm{softmax}\left(\left(\mathrm{W}_q x_j\right)^T \mathrm{W}_k x_i\right) \mathrm{W}_v x_j$$

- Transformers combine these blocks
- Positional encoding based on $p_j - p_i$

- Pro: Highly flexible, very effective (LLMs, Vision Transformers, etc.)
- Pro: ‘Global’ connections (vs. U-Net)
- Con: Computationally expensive (compared to CNN)
  - Solutions: work at coarse scale OR only apply locally

Summary

- **Multi-Layer Perceptrons (MLP)**
  - Linear combination, followed by non-linearity, repeated
  - Basic building block of Neural Networks
- **Convolutional Neural Network (CNN)**
  - Learn local kernel, convolve
  - Weight sharing, translation invariance
  - ‘Constructs’ localized features of increasing abstraction
- **Transformers**
  - Learn weights based on feature similarity
  - Highly expressive, but expensive; current SOTA for many tasks

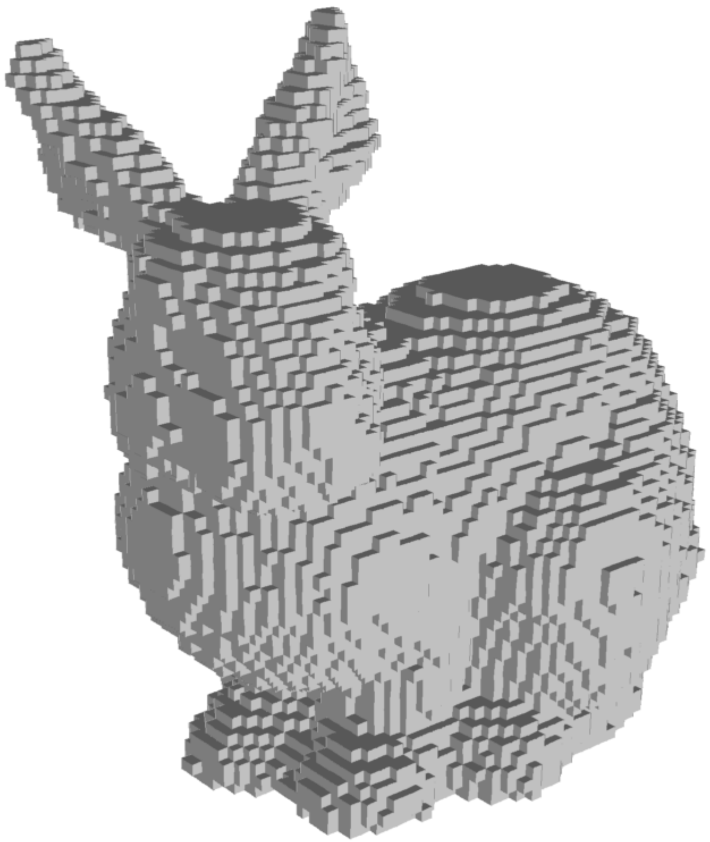
Chapter 2: 3D Data

Deep learning for 3D data

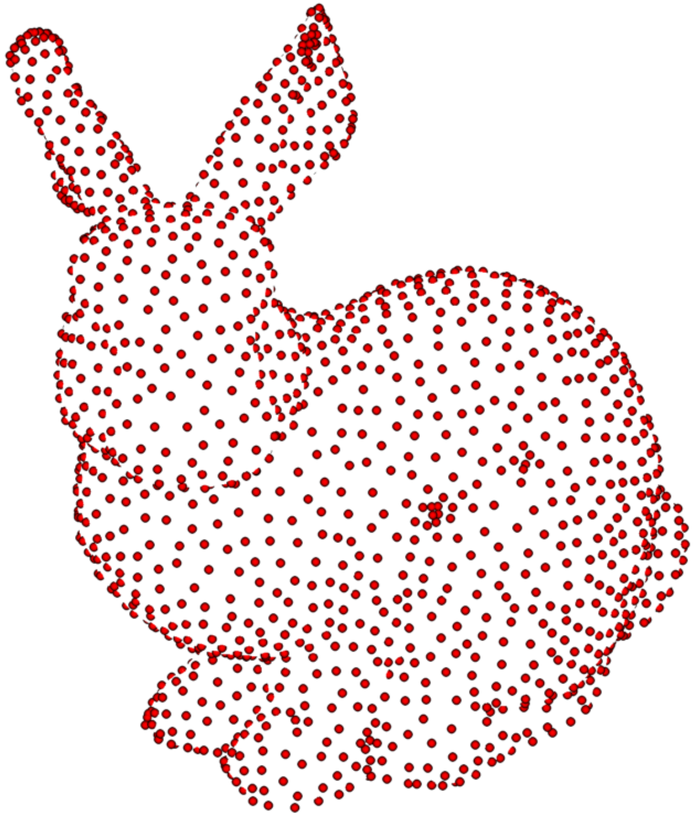
3D data	Model	Optimizer	Hardware
Large datasets	Layered, deep	Gradient Descent	GPUs
	High # parameters	Backpropagation	
	End-to-end		

'Philosophy': Scaling (data, compute) beats algorithmic complexity

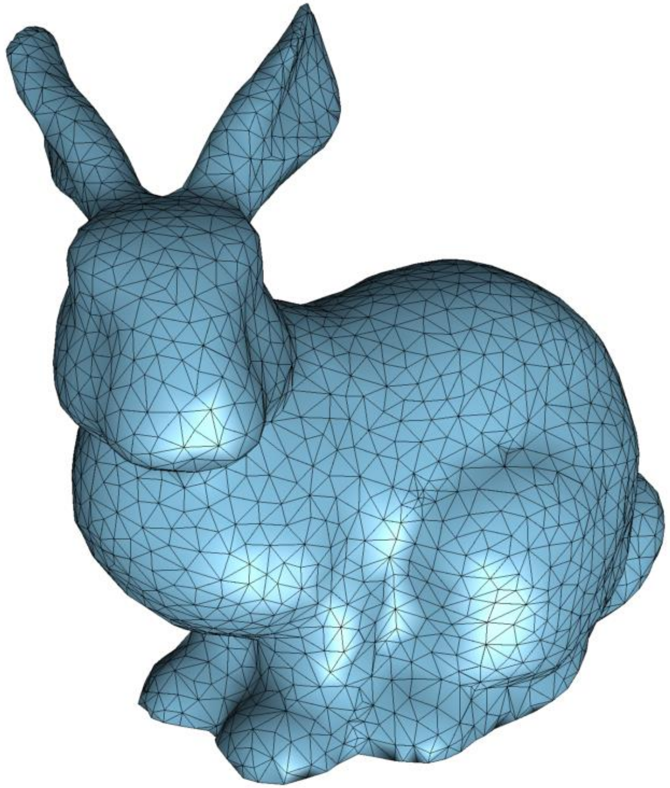
3D Data



Voxels



Point cloud



Mesh

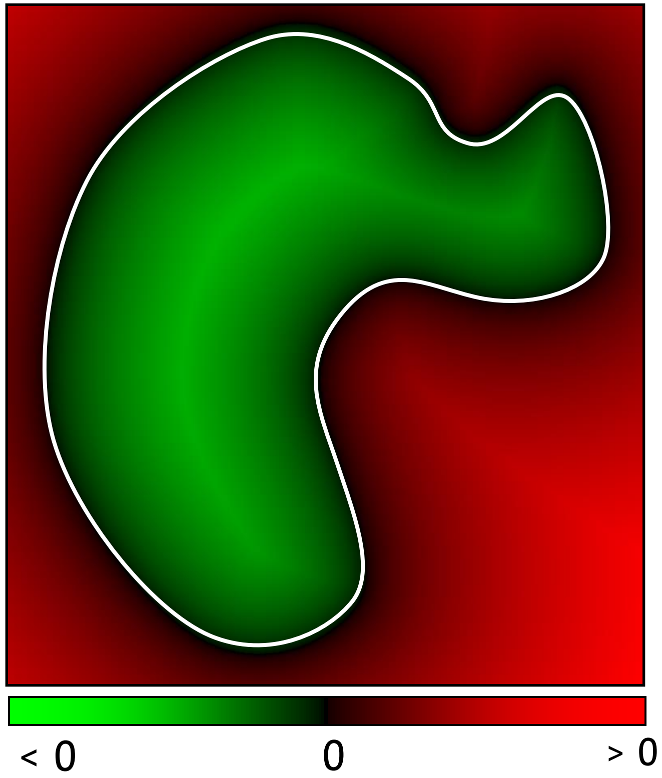
What about implicit functions?

$f: \mathbb{R}^3 \rightarrow \mathbb{R}$

Value > 0 outside shape, < 0 inside

In deep learning

- Use MLP to represent f
- Rarely used as input to MLP
  - Convert to voxels/mesh/point cloud

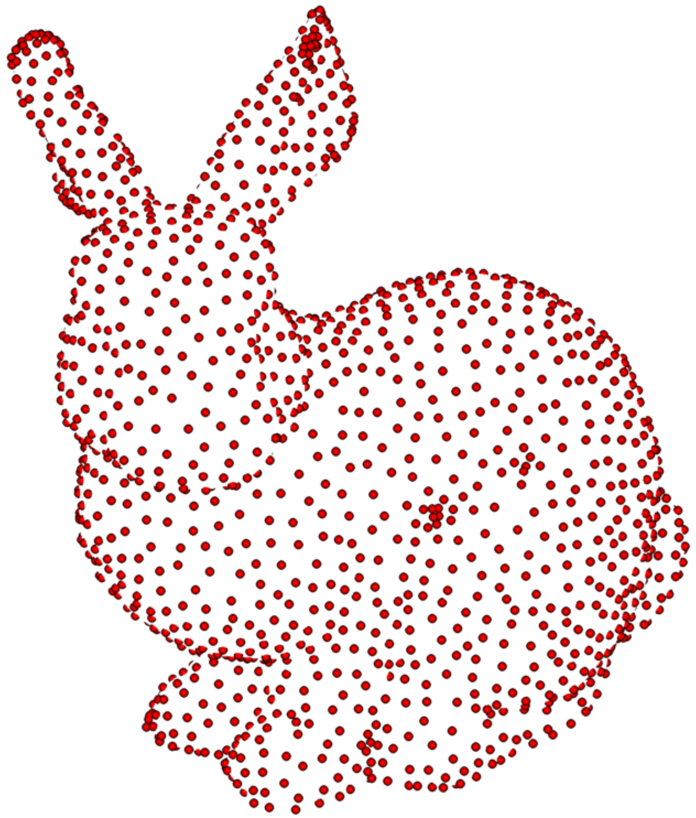


Based on slides by Olga Sorkine-Hornung

Point Cloud

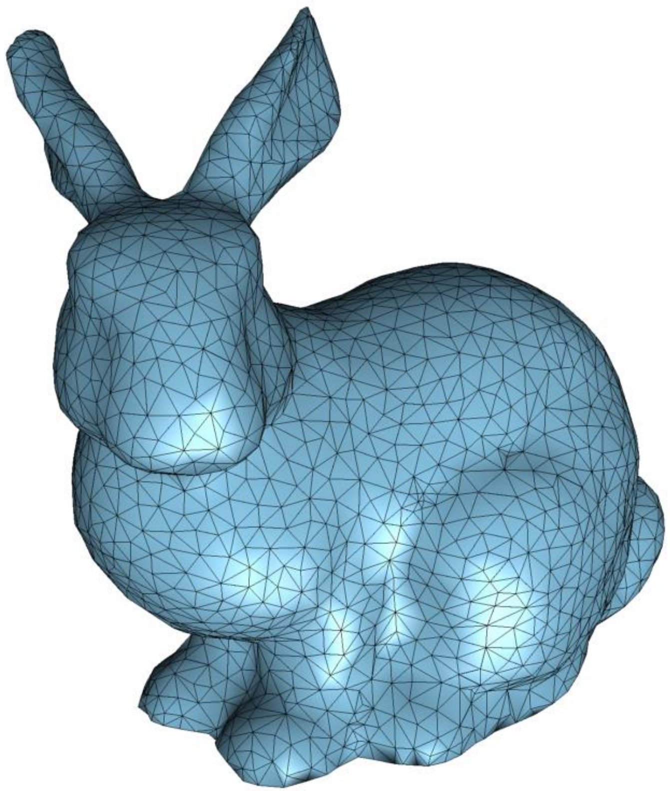
- Set of points in d-dimensional space

$$P \in \mathbb{R}^3$$



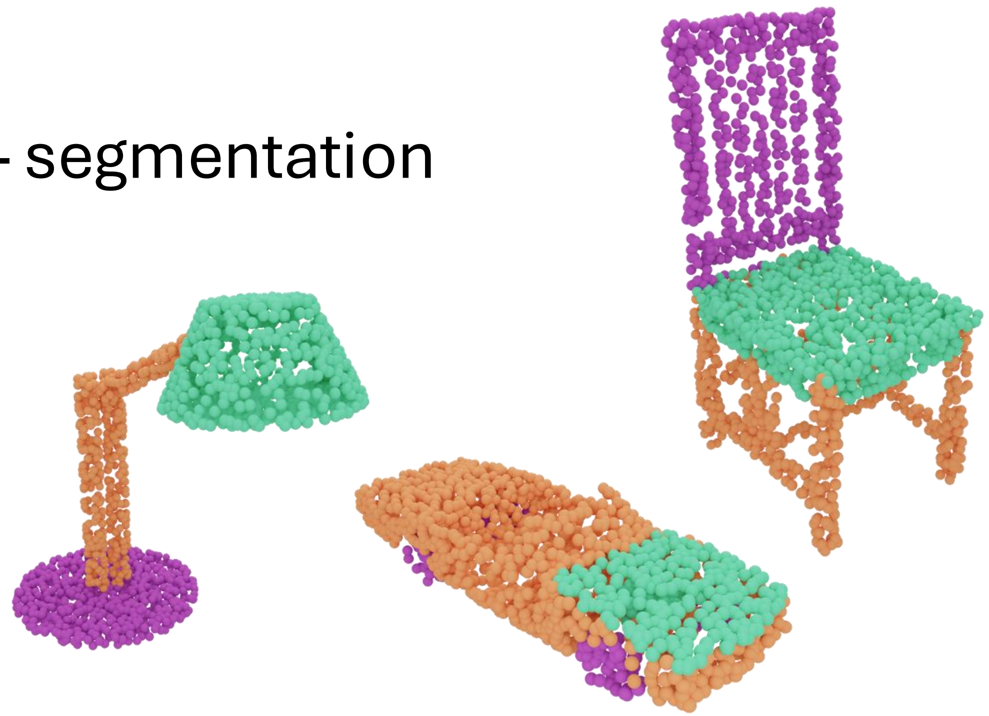
Mesh

- ‘Complements’ point cloud
  - connectivity and surface given by V, E, F
- Geometry can be on points, edges, etc.



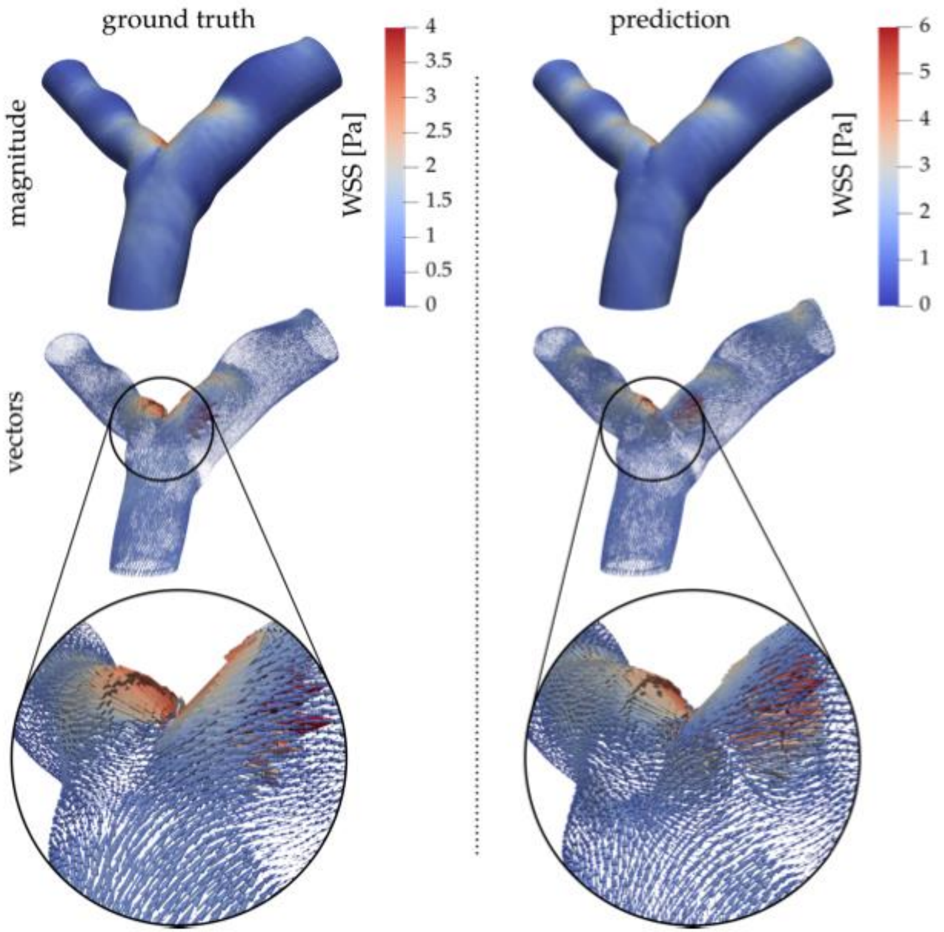
Datasets

- **ShapeNet** (meshes, point clouds) – segmentation, classification
- **ModelNet40** (meshes) – classification
- **ScanNet (v2, v3)** (RGB-D + reconstructions) – segmentation
- **S3DIS** (point clouds) – segmentation
- **Thingi10k** (meshes) – n/a
- **ABC** (meshes) – n/a
- Consider ethics, copyright!
  - E.g., https://huggingface.co/datasets/allenai/objaverse/discussions/18



More exotic inputs and outputs: simulation

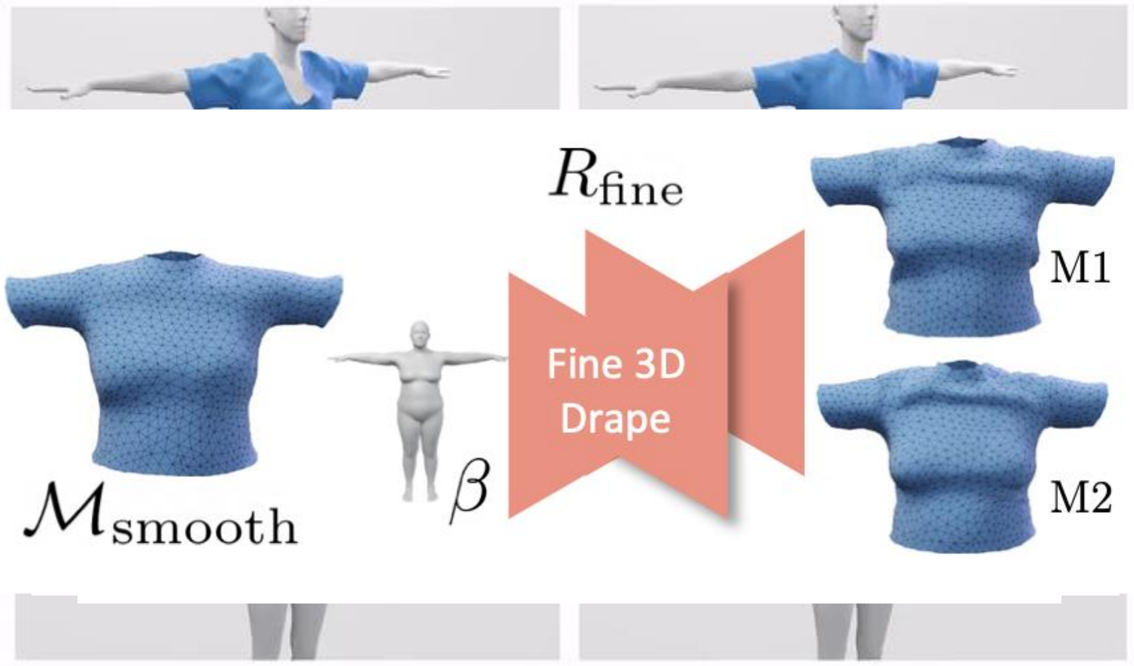
- **Input:** positions, distance to loops
- **Output:** tangent vector at each vertex



Mesh Neural Networks for SE(3)-Equivariant Hemodynamics Estimation of the Artery Wall, Suk et al. (2022)

More exotic inputs and outputs: simulation

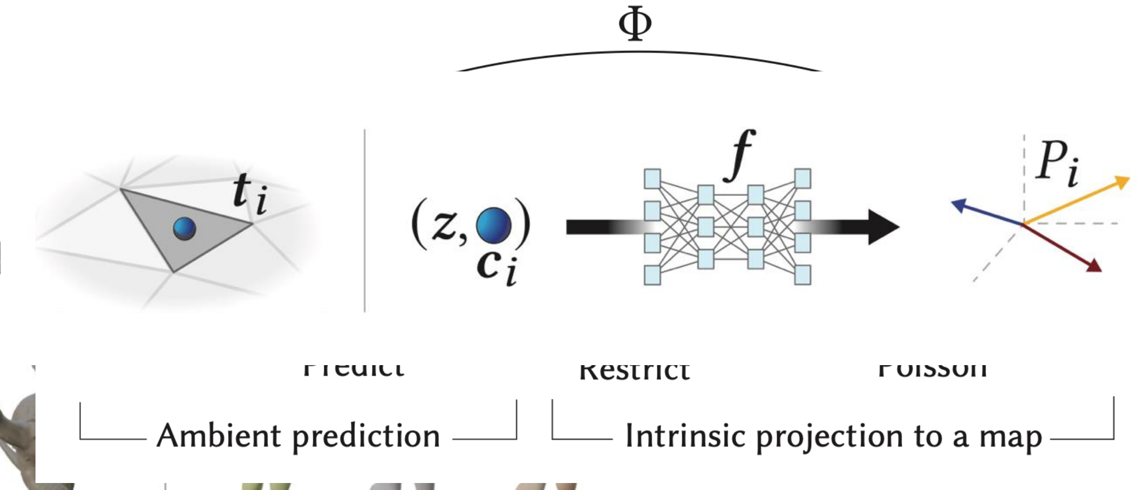
- Stage 1
  - **Input:** Garment parameters
  - **Output:** Mesh positions on mean shape
- Optimize topology
- Stage 2
  - **Input:** Mesh positions, target shape parameters
  - **Output:** Smooth mesh positions
- Stage 3
  - **Input:** Smooth mesh positions
  - **Output:** Fine mesh positions (wrinkles, etc.)

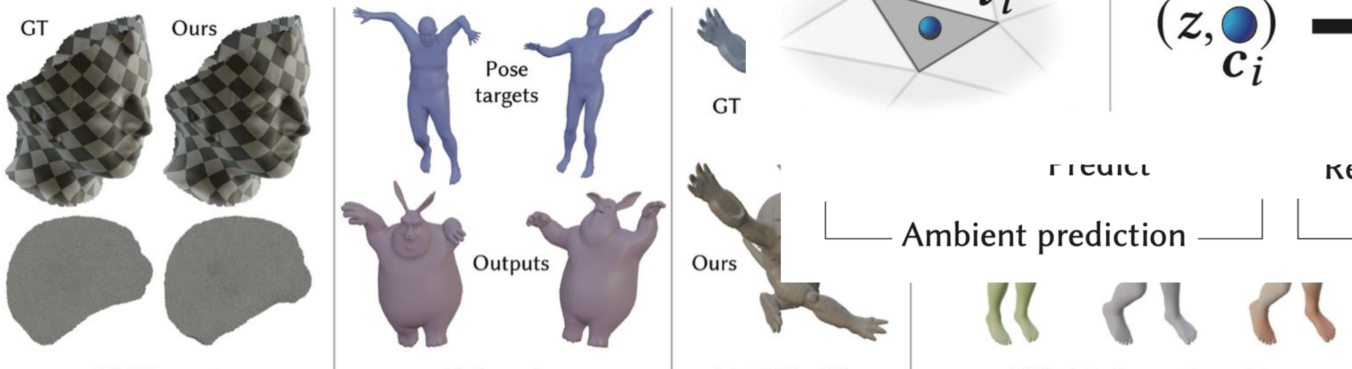


Fully Convolutional Graph Neural Networks for Parametric Virtual Try-On, Vidaurre et al. (2020)

More exotic inputs and outputs: Jacobians

- Neural Jacobian Fields
  - **Input:** Per-triangle centroid + global code
  - **Output:** Jacobian matrix
  - **Model:** MLP
- Post-proces
  - Restrict Jac
  - Solve Poiss



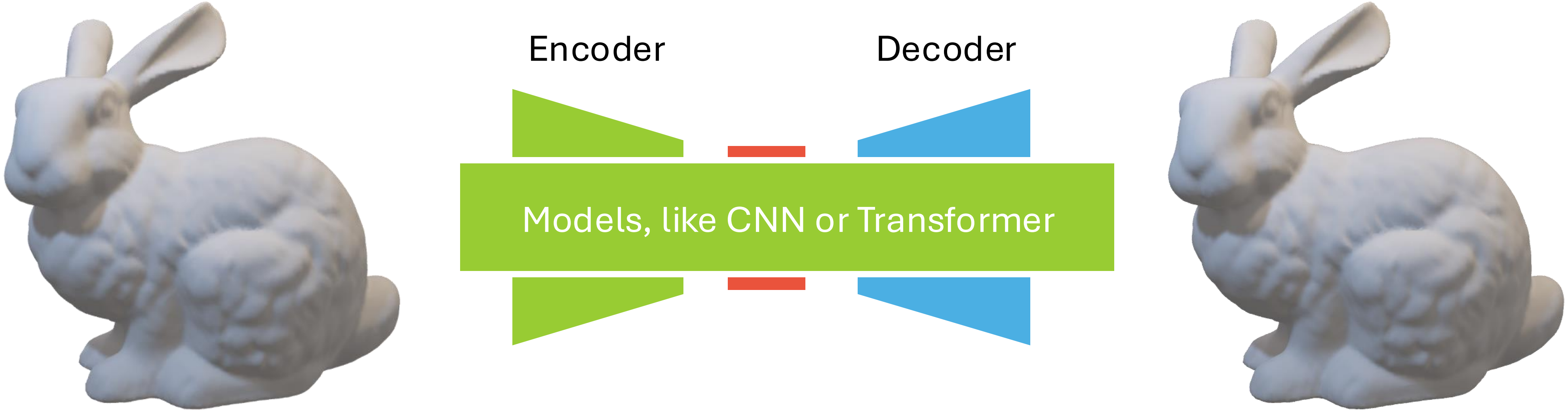


(a) UV mapping (b) Re-posing (c) ARAP editing (d) Registration and morphing

Neural Jacobian Fields: Learning Intrinsic Mappings of Arbitrary Meshes, Aigerman et al. (2022)

What if I have data, but no labels?

- Unsupervised learning
- For example: auto-encoders

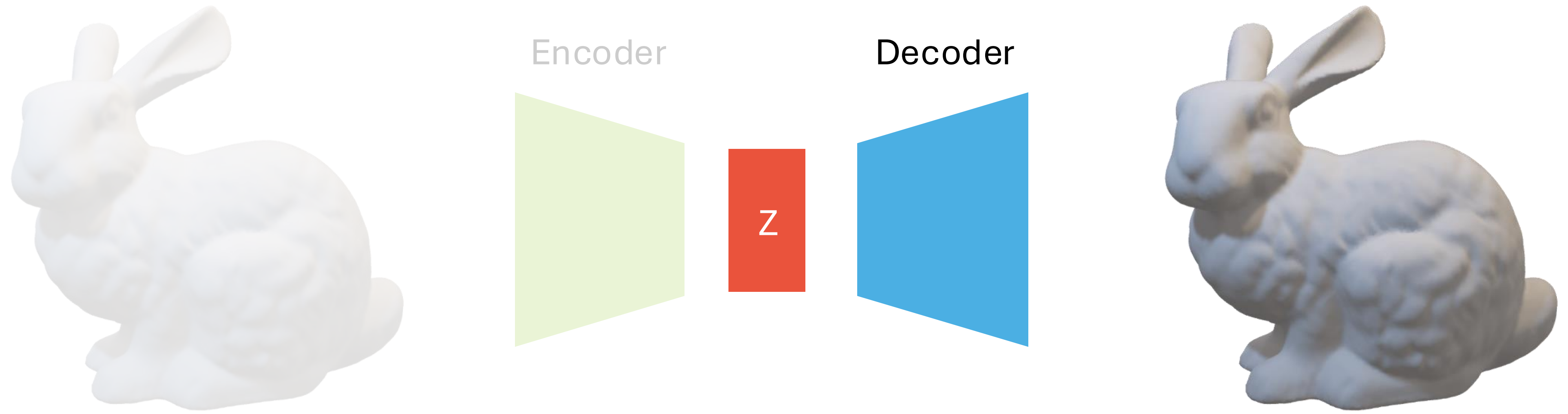


igl INTERACTIVE GEOMETRY LAB

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

What if I have data, but no labels?

- Unsupervised learning
- For example: auto-encoders



Chapter 3:
3D Deep Learning Models

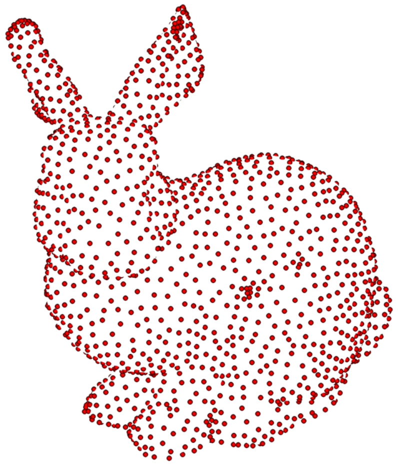
Deep learning for 3D data

3D data	Models for 3D	Optimizer	Hardware
Large datasets	Layered, deep	Gradient Descent	GPUs
	High # parameters	Backpropagation	
	End-to-end		

'Philosophy': Scaling (data, compute) beats algorithmic complexity

Let’s start simple

MLP



Stack all points?

Which order?

MLP

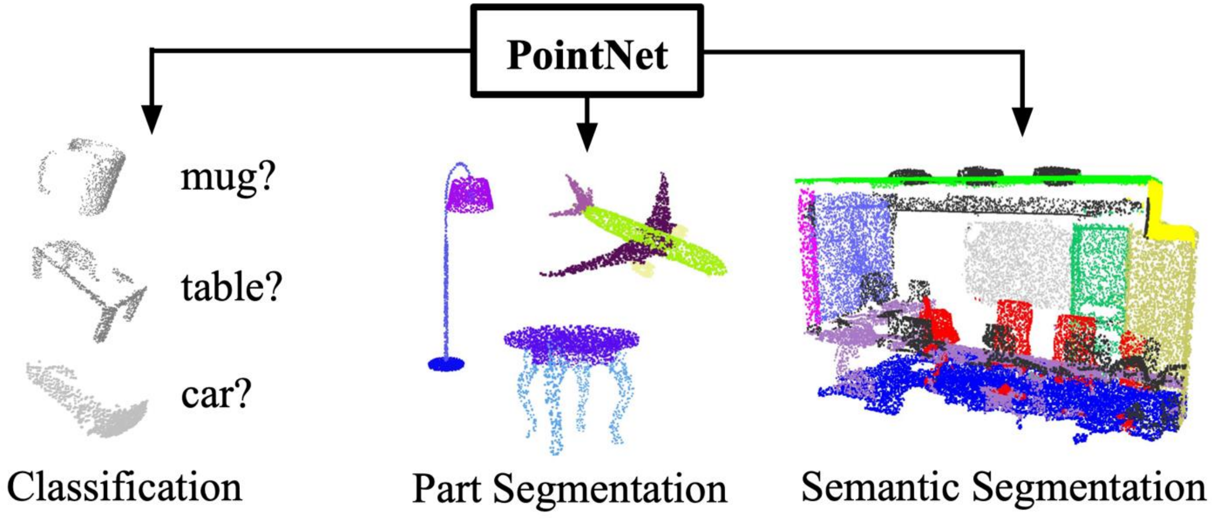
"Bunny"

PointNet

- MLP on each point
- Maximum over all points

$$y = \max_{j \in P} f_\theta(x_j)$$

Cannot learn from neighborhoods



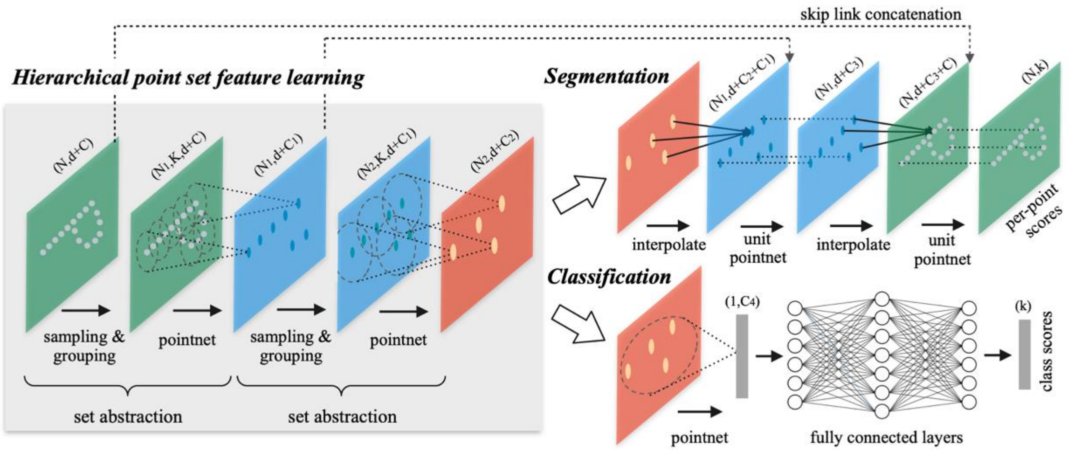
PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation, Qi, Su, Mo, Guibas (2016)

PointNet++

- MLP on each point
- Maximum over neighborhood
  - kNN – homogeneous
  - Radius – more robust to sampling
    - Geodesic/Euclidean?

$$x_i' = \max_{j \in N_i} f_\theta(x_j, p_j - p_i)$$

- Hierarchies with maximum pooling (like CNN)

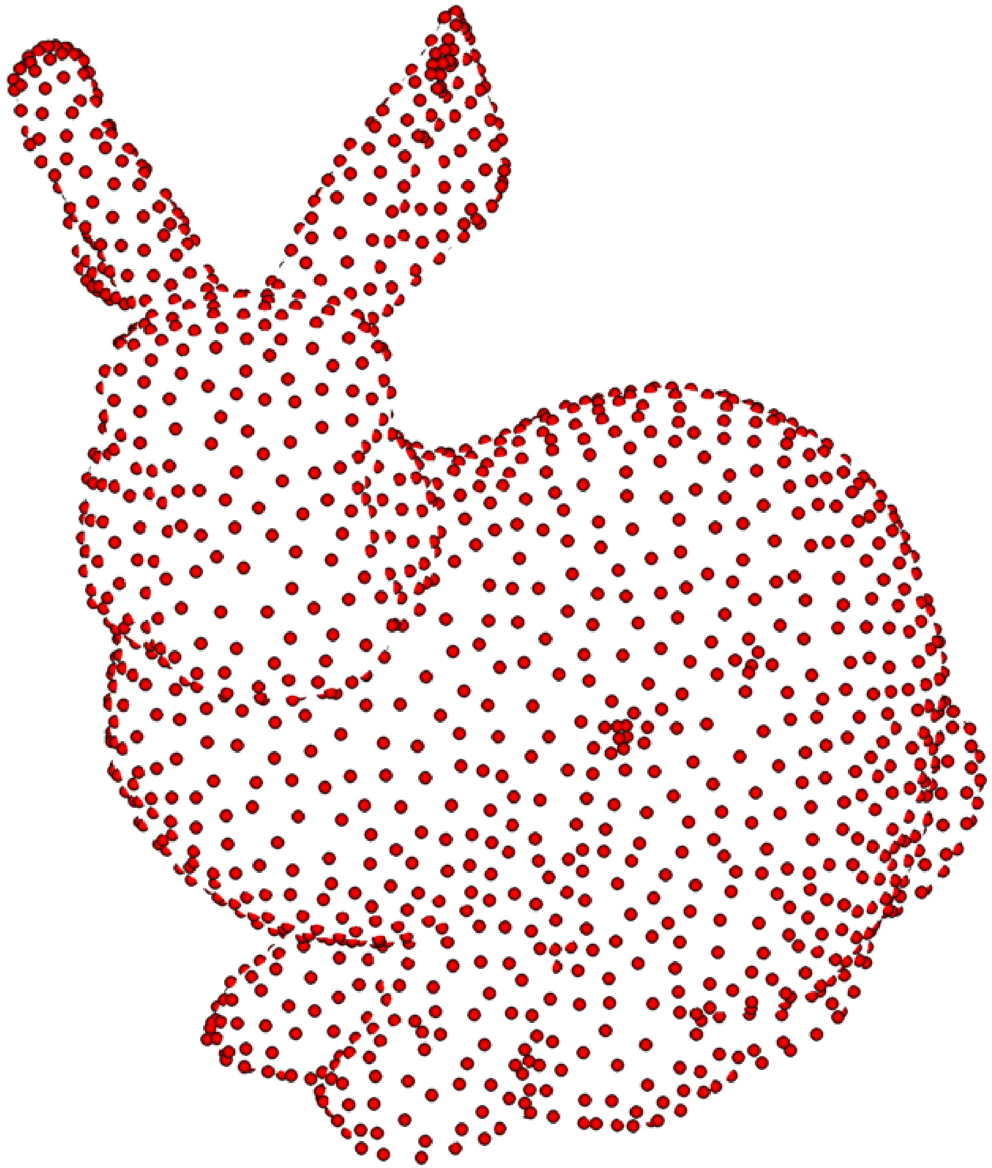


PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, Qi, Yi, Su, Guibas (2017)

Taking a step back

Surface as a graph

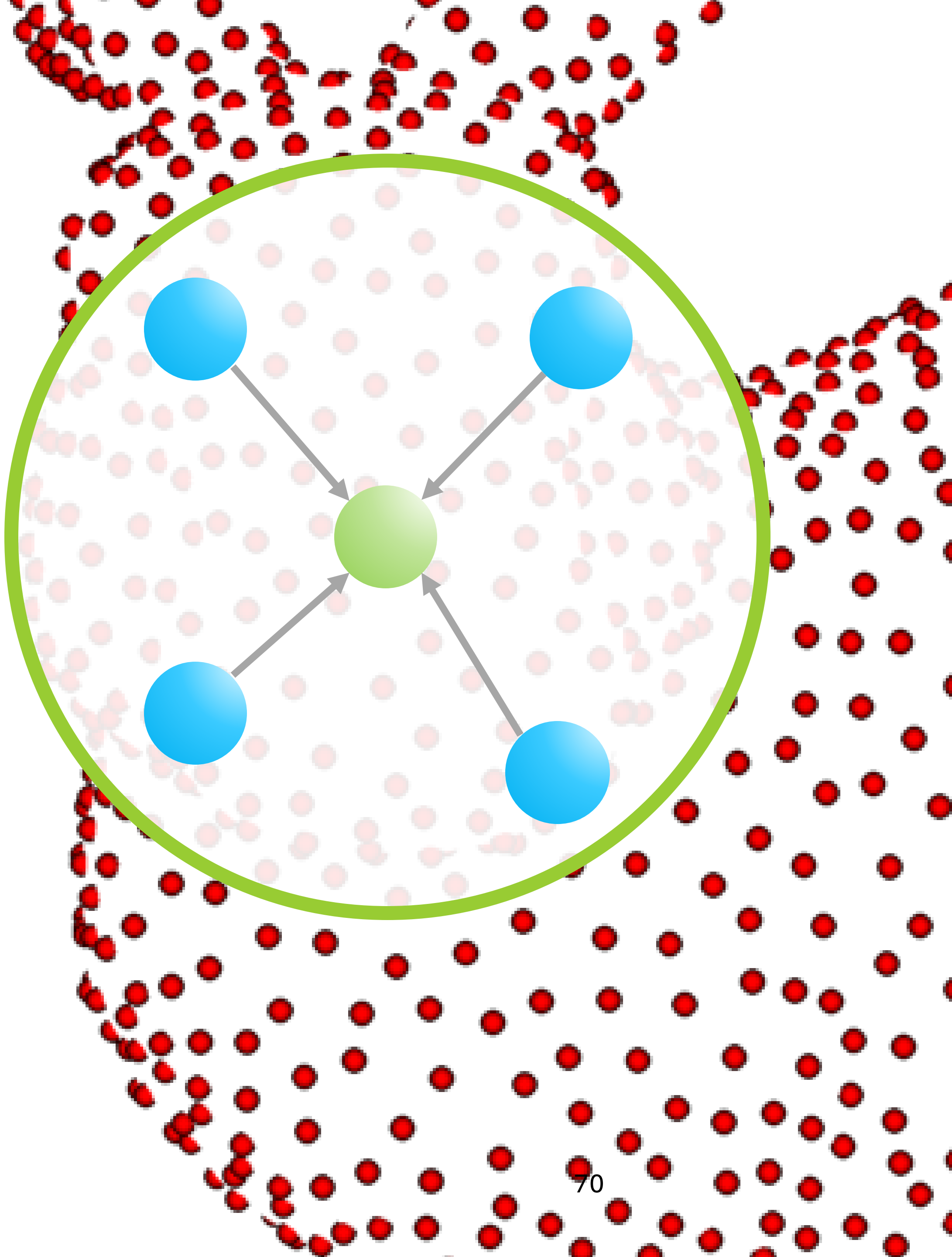
- Vertices or points are nodes



Surface as a graph

- Vertices or points are nodes
- Edges connect nearby points (radius graph, k-NN graph)
- For meshes: use mesh edges

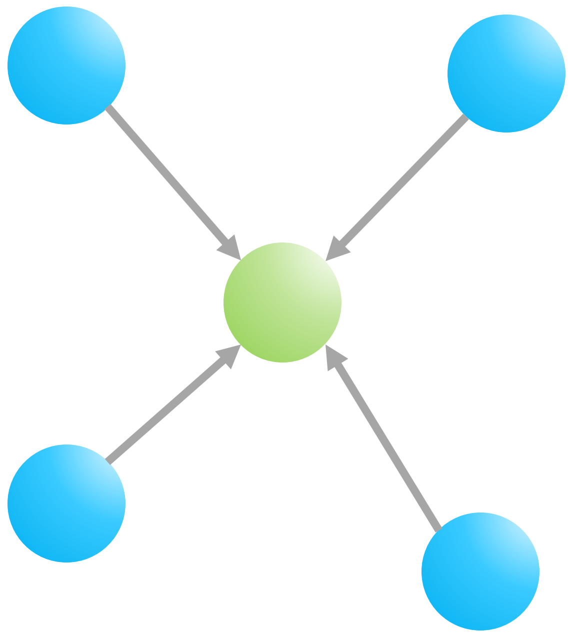
- What can meshes help with?
  - Geodesic neighborhoods (we know connectivity)
  - Encode geometry (e.g., MeshCNN)



igl
INTERACTIVE GEOMETRY LAB

Message passing

1. Compute ‘message’ on each node
2. Aggregate messages over edges

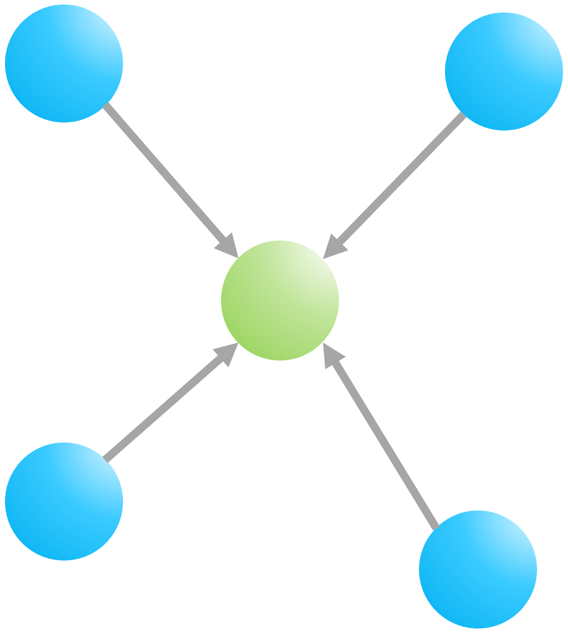


Neural Message Passing for Quantum Chemistry, Gilmer, Schoenholz, Riley, Vinayls, Dahl (2017)

https://pytorch-geometric.readthedocs.io/en/latest/tutorial/create_gnn.html

PointNet++ as Message Passing

1. **Message:** MLP on features + relative location
2. **Passing:** Maximum over neighbors



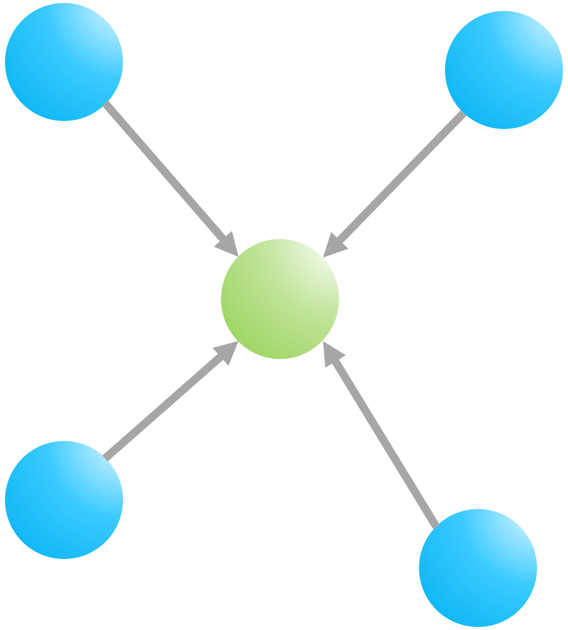
$$x_i' = \max_{j \in N_i} f_\theta(x_j, p_j - p_i)$$

PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, Qi, Yi, Su, Guibas (2017)

EdgeConv as Message Passing

1. **Message:** MLP on ‘relative features’ (edges)
2. **Passing:** Maximum over neighbors

$$x_i' = \max_{j \in N_i} f_\theta(x_i, x_j - x_i)$$

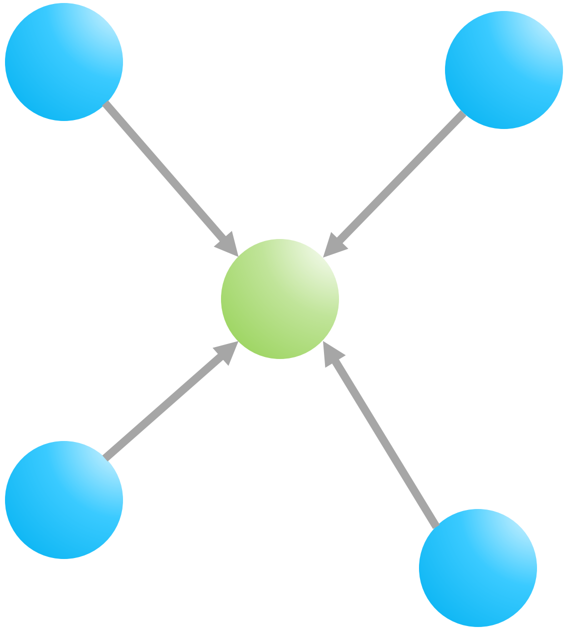


Dynamic Graph CNN for Learning on Point Clouds, Wang, Sun, Liu, Sarma, Bronstein, Solomon (2019)

GCN as Message Passing

1. **Message:** Linear transformation of features
2. **Passing:** Weight by degree, average

$$x_i' = \sigma\left(\mathrm{W}_0 x_i + \sum_{j \in N_i} \frac{1}{c_{ij}} \mathrm{W}_1 x_j\right)$$



Graph Convolutional Networks, Kipf, Welling (2016)

Laplacian in GCN and EdgeConv

- Laplacian: Sum of second derivatives
  - Discrete setting: Difference to average of neighbors

- GCN: graph Laplacian

$$x_i' = \sigma(\mathrm{W}_0 x_i + \sum_{j \in N_i} \frac{1}{c_{ij}} \mathrm{W}_1 x_j)$$

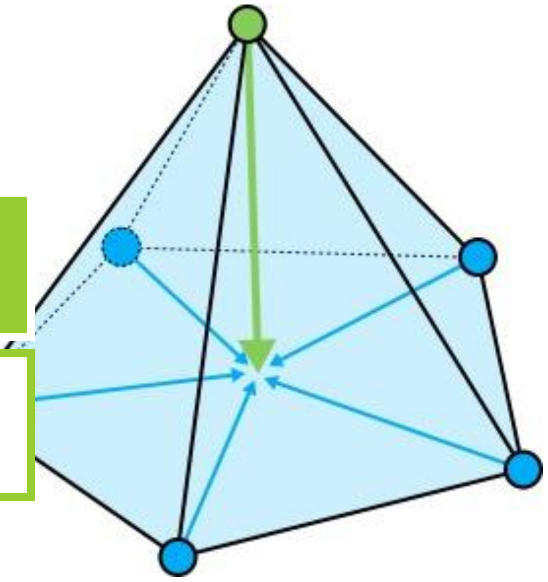
- EdgeConv

$$x_i' = \max_{j \in N_i} f_\theta(x_i, x_j - x_i)$$

Learning to Diffuse

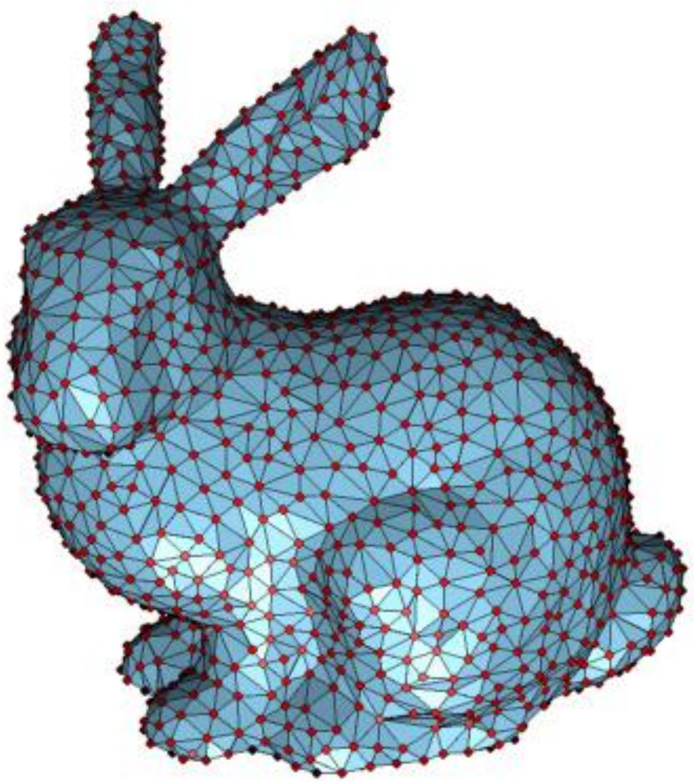
Smoothly

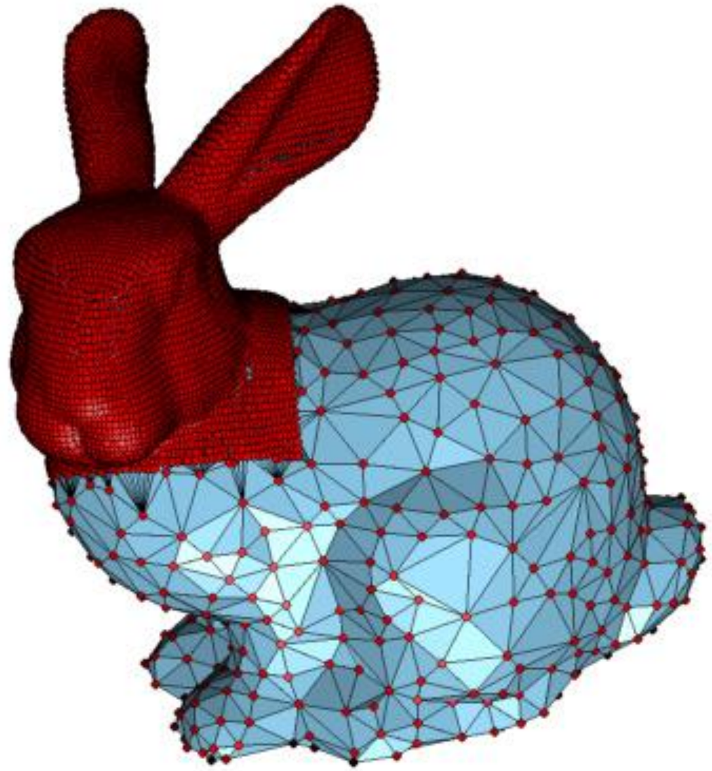
'Sharply'



More geometry

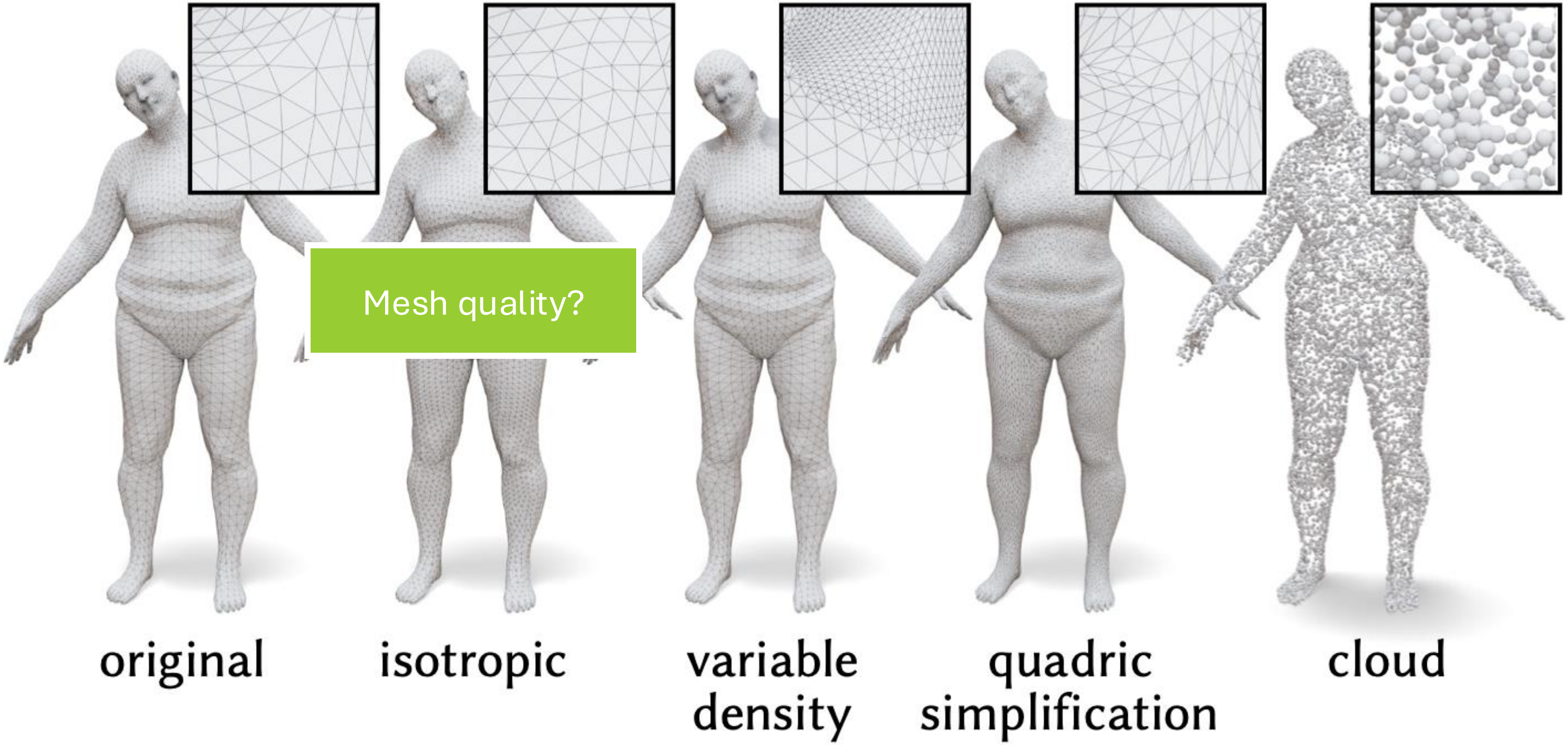
Why geometry?





Based on slides by Klaus Hildebrandt

Why geometry?

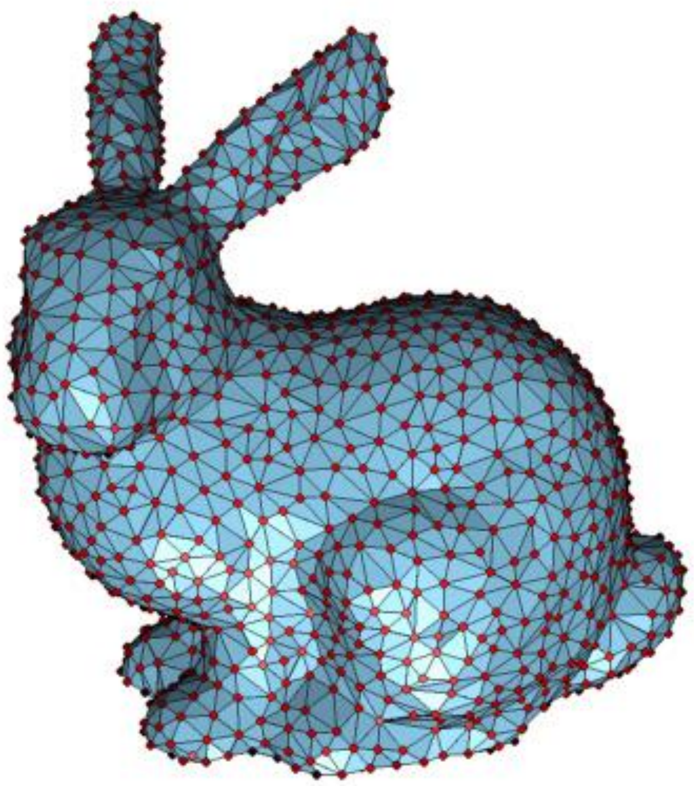


Mesh quality?

DiffusionNet: Discretization Agnostic Learning on Surfaces, Sharp et al. 2022

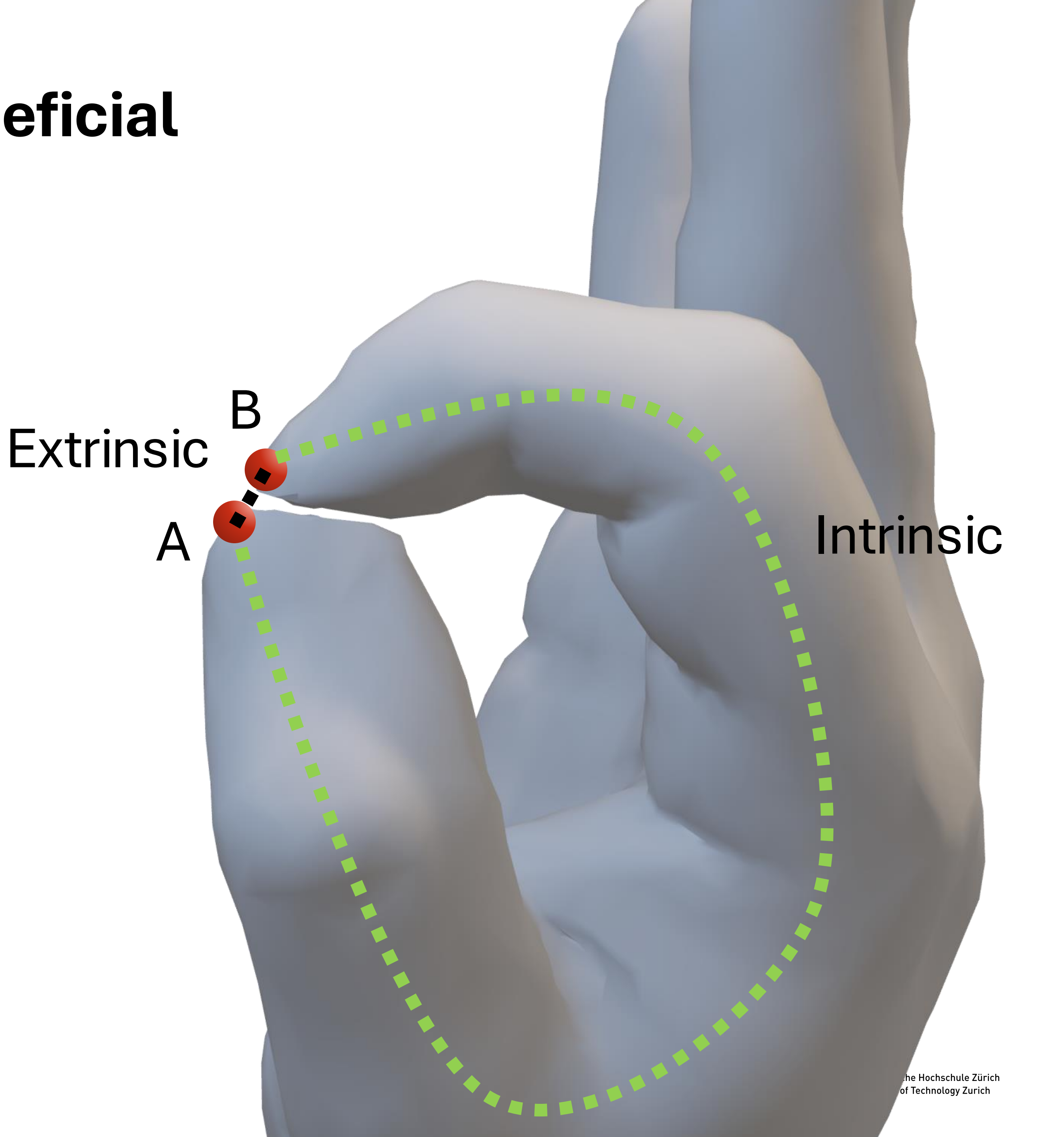
Why geometry?

- Invariances



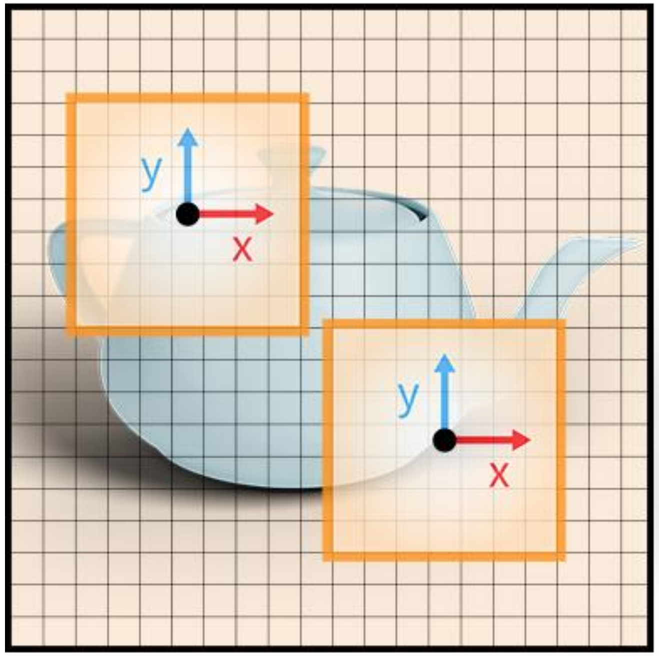
Intrinsic operations can be beneficial

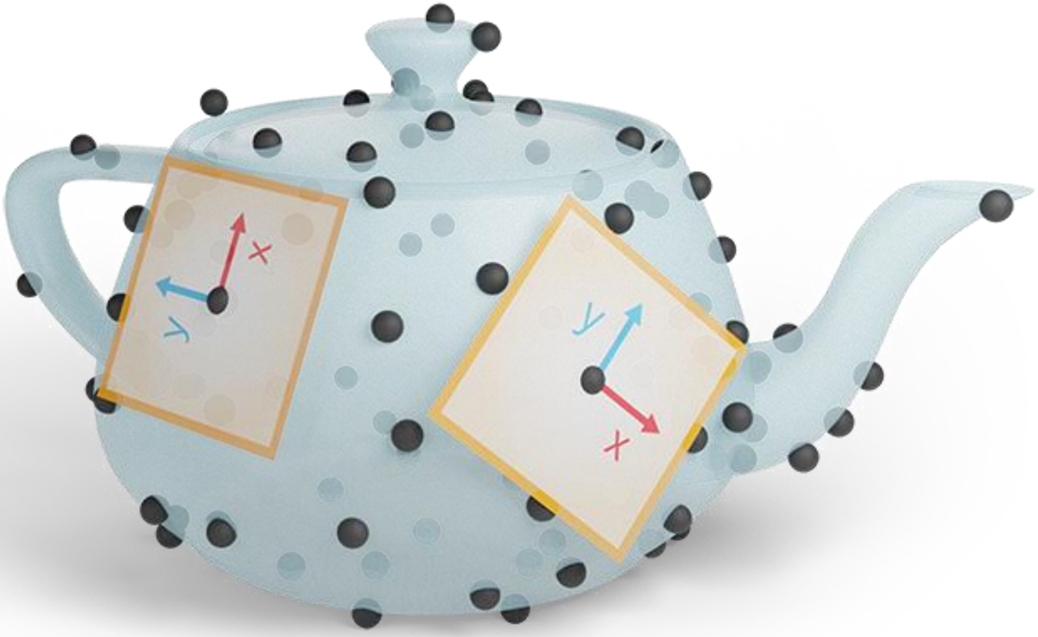
+ Robust to isometric deformations
+ 2D instead of 3D
+ No/less distortion or occlusion



Why Geometry?

- Invariances within an invariance

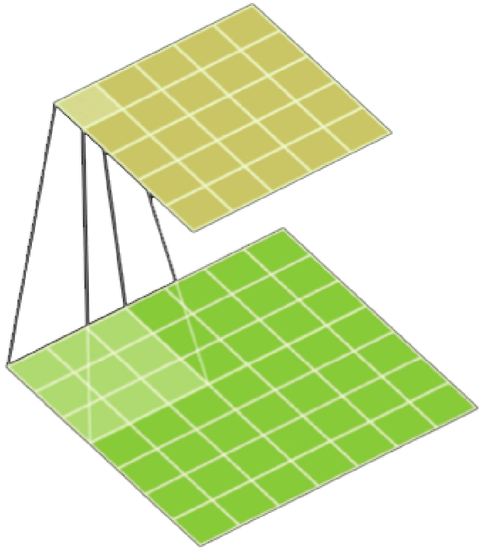


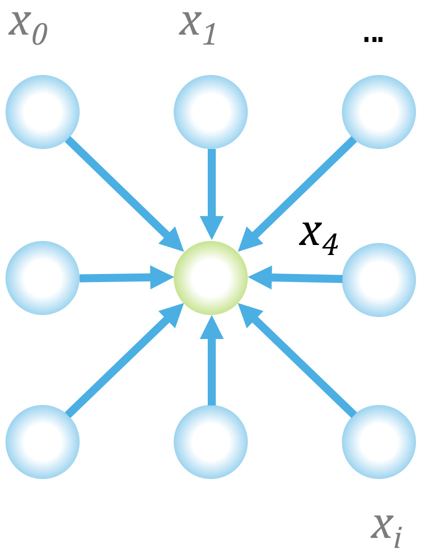


Back to CNNs

CNN on a Graph

Single CNN layer with 3x3 filter

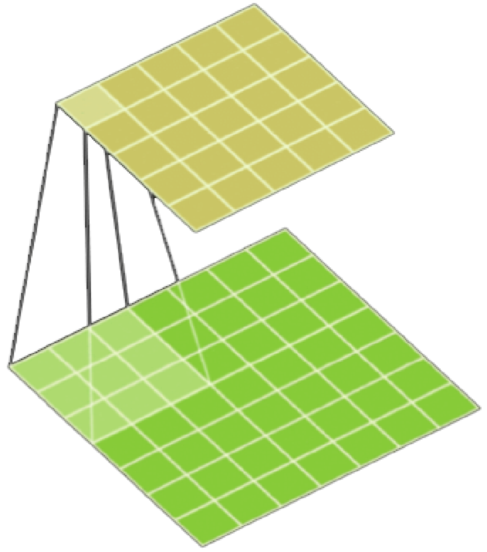


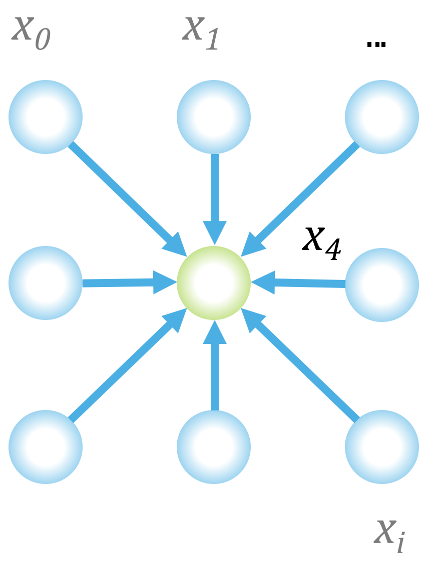


$$x'_4 = \sigma\Big(\sum_i \mathrm{W}_i x_i\Big)$$

CNN on a Graph

Single CNN layer with 3x3 filter





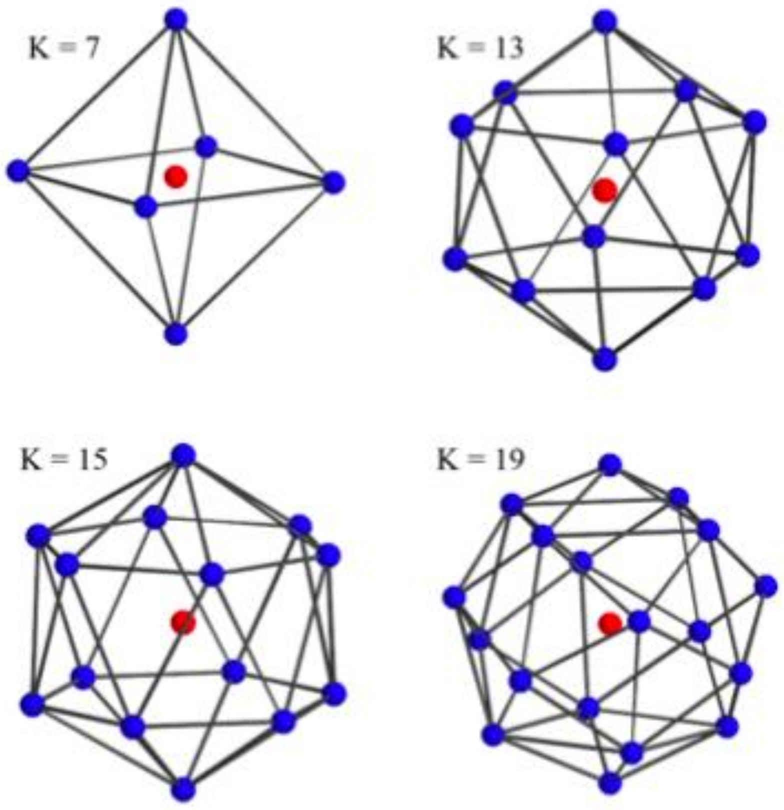
$$x_4' = \sigma\Big(\sum_i \boxed{W_i} x_i\Big)$$

Compare with GCN

$$x_i' = \sigma\Big(W_0 x_i + \sum_{j \in N_i} \frac{1}{c_{ij}} \boxed{W_1} x_j\Big)$$

CNNs for 3D

- Graph- and point based
  - GCN, PointNet++, EdgeConv
- 3D kernel (extrinsic)
  - KPConv, MinkowskiNet, SSCN

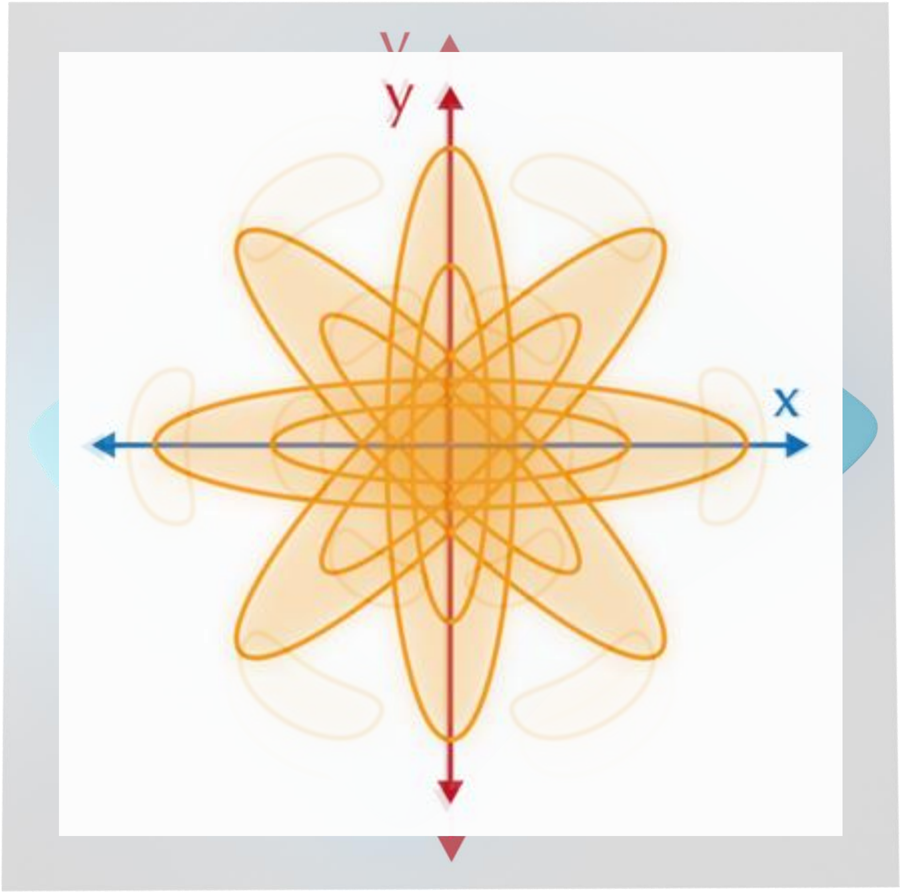


KPConv

[Thomas et al. 2019]

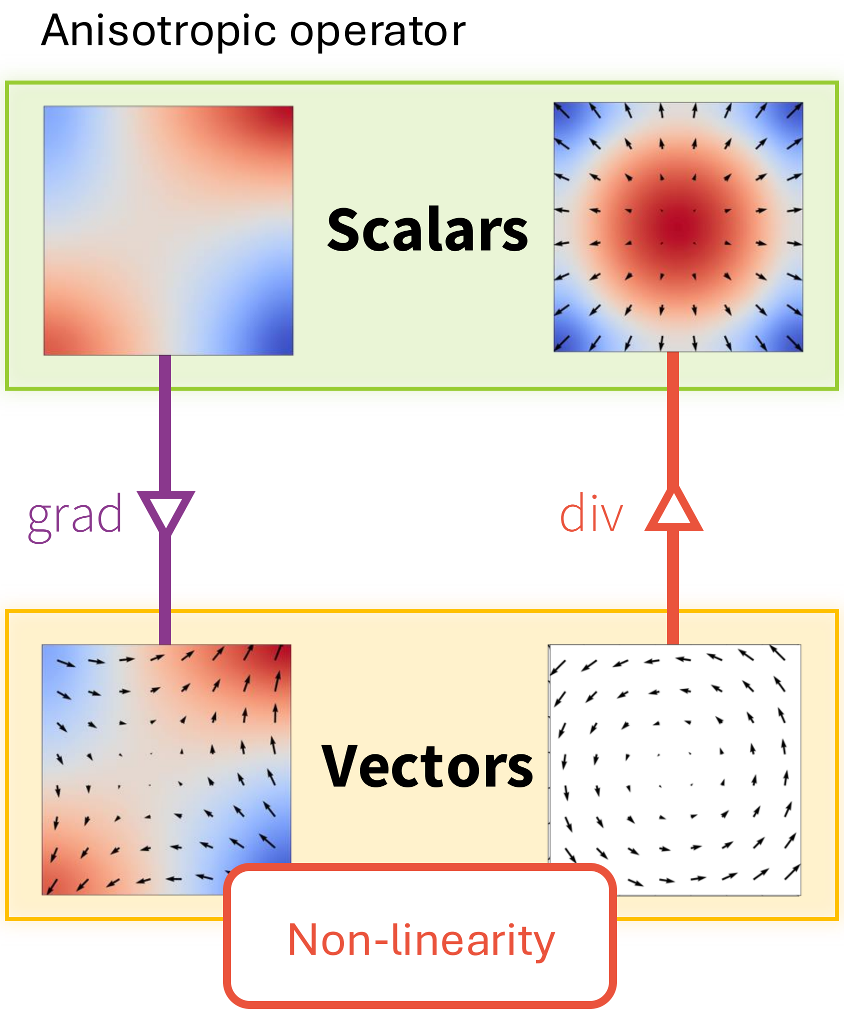
CNNs for 3D

- Graph- and point based
  - GCN, PointNet++, EdgeConv
- 3D kernel (extrinsic)
  - KPConv, MinkowskiNet, SSCN
- 2D kernels *on* surfaces (intrinsic)
  - GCNN, ACNN, MoNet, MDGCNN, HSN



CNNs for 3D

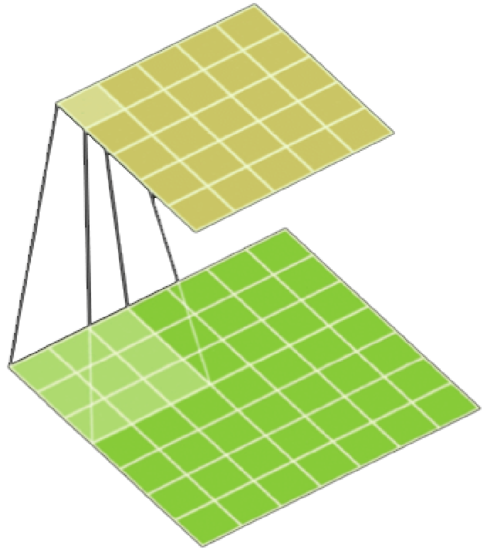
- Graph- and point based
  - GCN, PointNet++, EdgeConv
- 3D kernel (extrinsic)
  - KPConv, MinkowskiNet, SSCN
- 2D kernels *on* surfaces (intrinsic)
  - GCNN, ACNN, MoNet, MDGCNN, HSN
- Operator-based (e.g., Laplacian)
  - DeltaConv

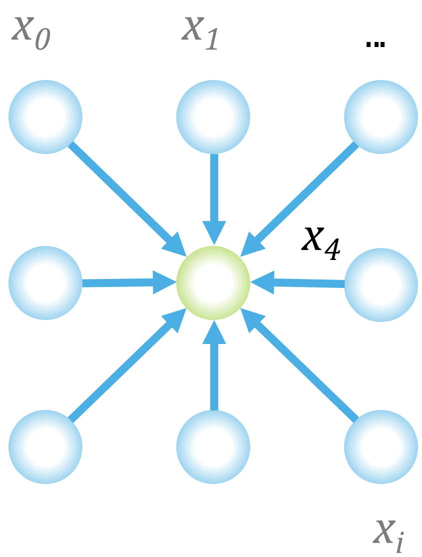


Transformers

Attention on a Graph

Single CNN layer with 3x3 filter

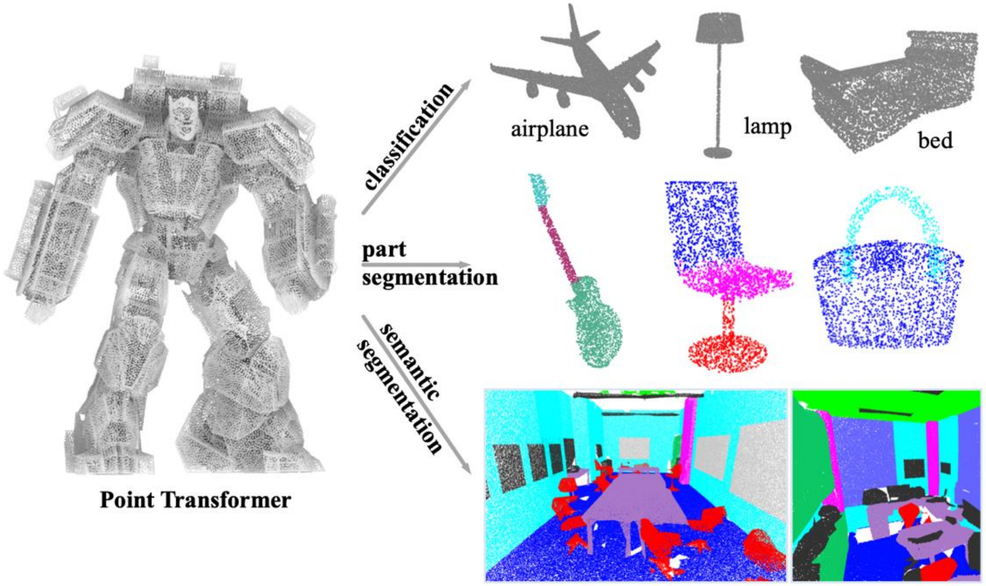




$$x_i' = \sum_{j=0}^{N} \mathrm{softmax}\left(\left(\mathrm{W}_q x_j\right)^T \mathrm{W}_k x_i\right) \mathrm{W}_v x_j$$

Transformers for 3D

- PointTransformer - Zhao et al. 2021
- Challenge: how to reduce cost?
  - Apply on local neighborhoods
  - Apply on serialized point clouds (space-filling curves, v3 2024)
- Challenge: positional encoding
  - PointNet++-like positional encoding, $\mathrm{MLP}(p_j - p_i)$
  - Rotation invariance?

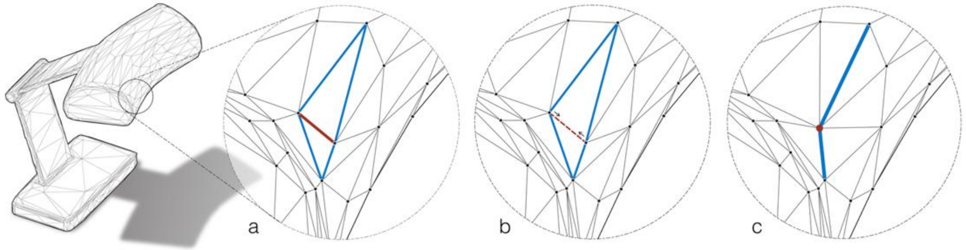


Current state-of-the-art

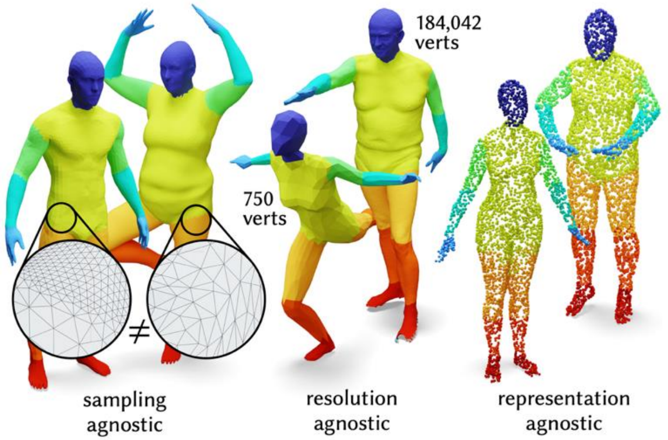
- Many approaches, *but* PointNet++ still seems to work quite well
  - Due to tasks/benchmarks?
- Other options often seen
  - DGCNN (EdgeConv), PointTransformer v3, Sparse Voxel Convolutions
- Which one to use depends on your task
  - Do you expect the network to understand curvature? Maybe not PointNet++/DGCNN?
  - Do you want efficiency, simplicity? – Try the simple networks first.

Methods you should know, but we didn't cover

- MeshCNN (Hanocka et al. 2019)
  - Operates on mesh edges
  - Geometric features
    - dihedral angle, two inner angles and two edge-length ratios for each face

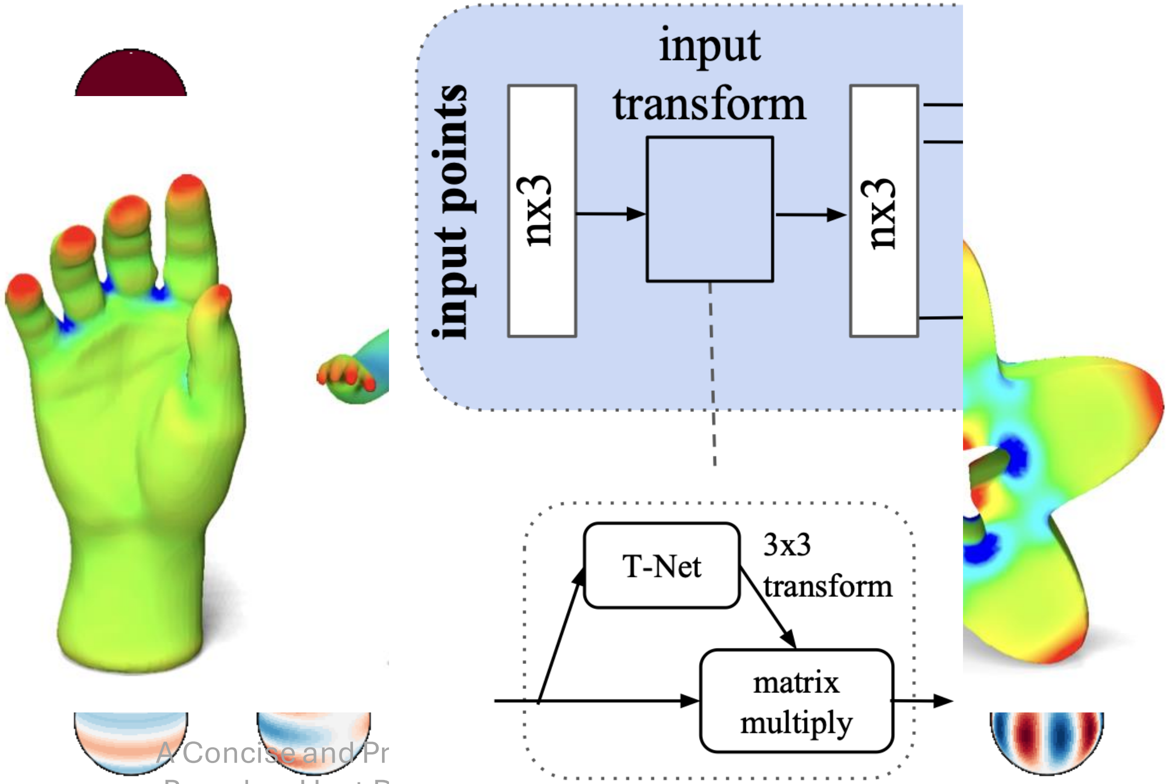


- DiffusionNet (Sharp et al. 2022)
  - Learning to diffuse
  - Accelerated in frequency domain (Eigenvectors of Laplacian)
  - Robust to discretizations (e.g., mesh → point cloud)



Rotation invariance, translation invariance?

- Data augmentation
  - Random rotation/translation
- ‘Learn’ the transformation
  - Spatial transformer, e.g., in PointNet
- Invariant input features
  - Heat-Kernel Signatures (DiffusionNet)
  - MeshCNN – Edges, intrinsic features
- Encode features in local frames
  - Point difference, normals (translation-invariant)
  - Distances (translation-rotation-invariant)
- Rotation-equivariance
  - E.g., Tensor Field Networks (Thomas et al. 2018), Spherical Harmonics (Poulenard et al. 2019)



A Concise and Pr
Based on Heat D
PointNet: Deep Learning on Point Sets for 3D Classification
and Segmentation, Qi, Su, Mo, Guibas (2016)
github.com/nvidia/torch-harmonics

igl INTERACTIVE GEOMETRY LAB
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Chapter 4: Hands-on with PyTorch Geometric

PyTorch Geometric

https://pytorch-geometric.readthedocs.io/en/latest/

- Implementation of message-passing paradigm
- Supports many convolution types
- Many helpful utilities

- Alternatives
  - Kaolin, PyTorch3D, GraphGym

Setting up the environment

Dataset, loading and visualization

Model

Training

Deep Learning on Meshes and Point Clouds

Ruben Wiersma

Graduate School - SGP 2025, Bilbao