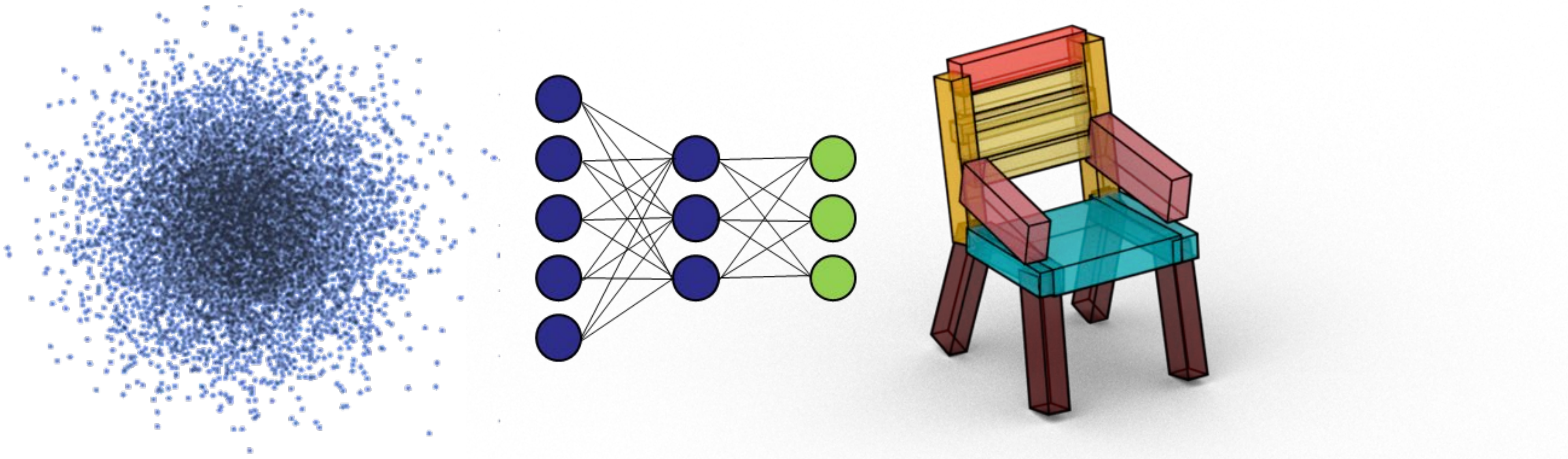
CS348n: Neural Representations and Generative Models for 3D Geometry

Leonidas Guibas
Computer Science Department
Stanford University

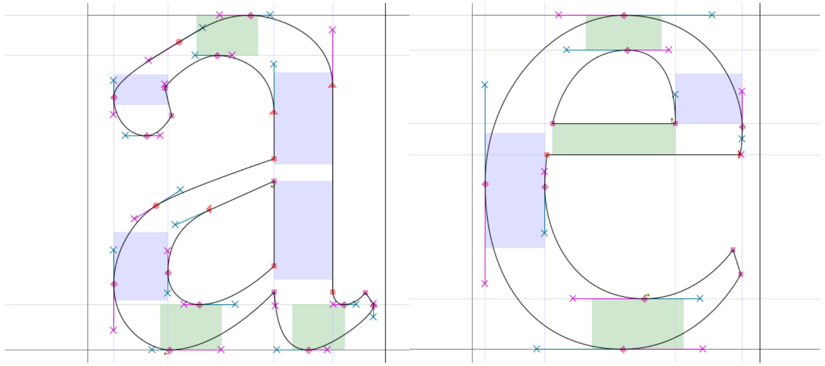


Leonidas Guibas Laboratory

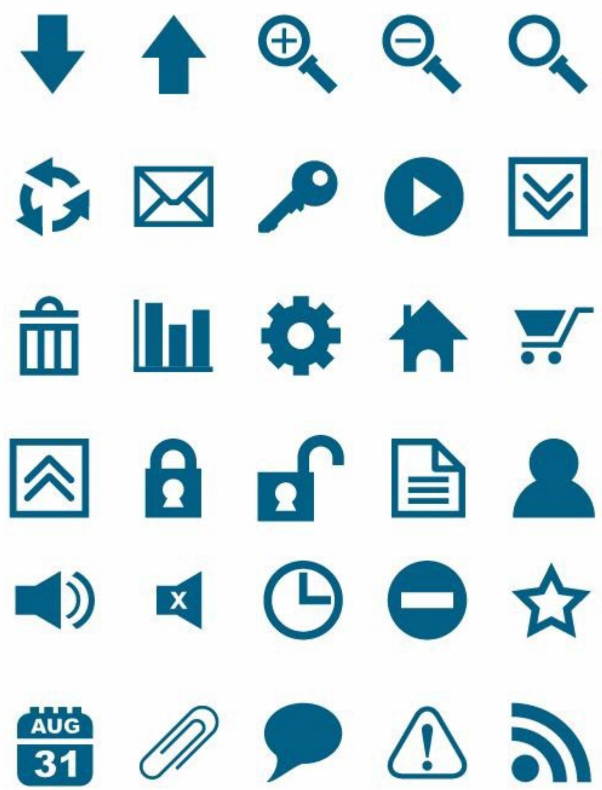
Geometric Computing

Last Time: Learning Vector and Mesh/CAD Representations

Vector (2D) and Mesh/CAD (3) Shape Models



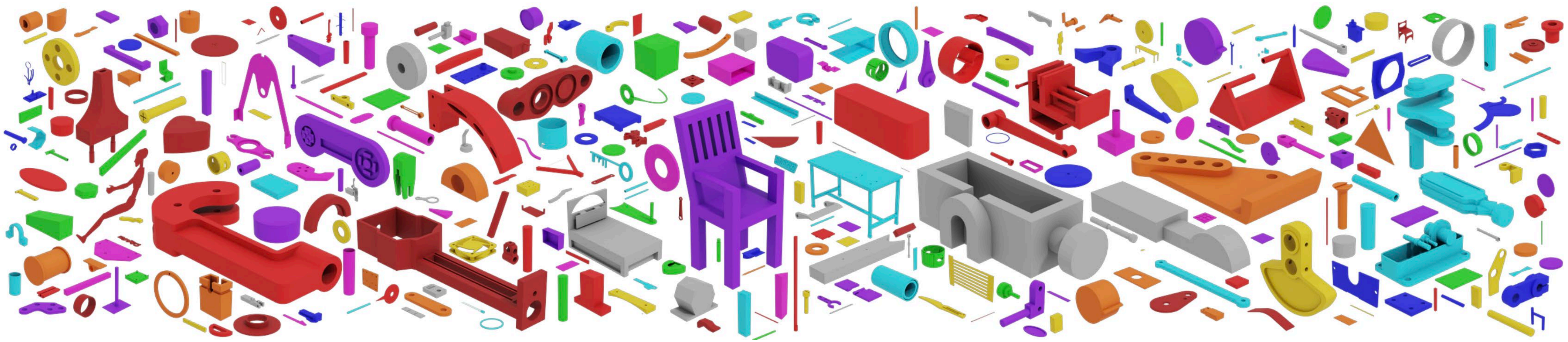
Fonts



Clip Art

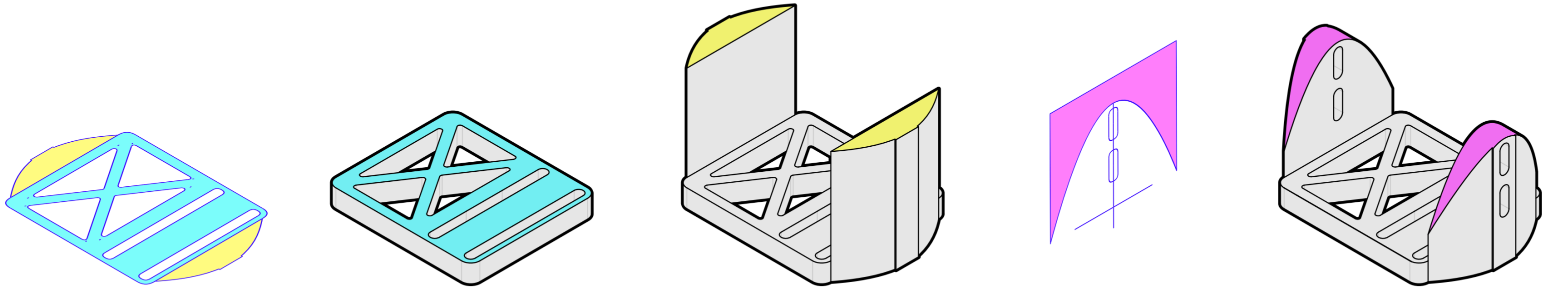
2D

Vector (2D) and Mesh/CAD (3D) Shape Models



Autodesk Fusion 360 Dataset

3D



Sketch 1 → Extrude 1 → Extrude 2 → Sketch 2 → Extrude 3

Sketch 1

```
s1 = add_sketch('XZ')
add_line(.6, .8, .6, 2.4)
add_arc(.6, 2.4, .8, 2.4, 90)
add_line(.8, 2.6, 1, 2.6)
...
p1 = add_line(-1, 2, -1, .8)
```

Extrude 1

```
add_extrude(
    sketch=s1,
    profile=p1[1],
    distance=.8,
    operation='NewBody')
```

Extrude 2

```
add_extrude(
    sketch=s1,
    profile=p1[0::2],
    distance=5,
    operation='Join')
```

Sketch 2

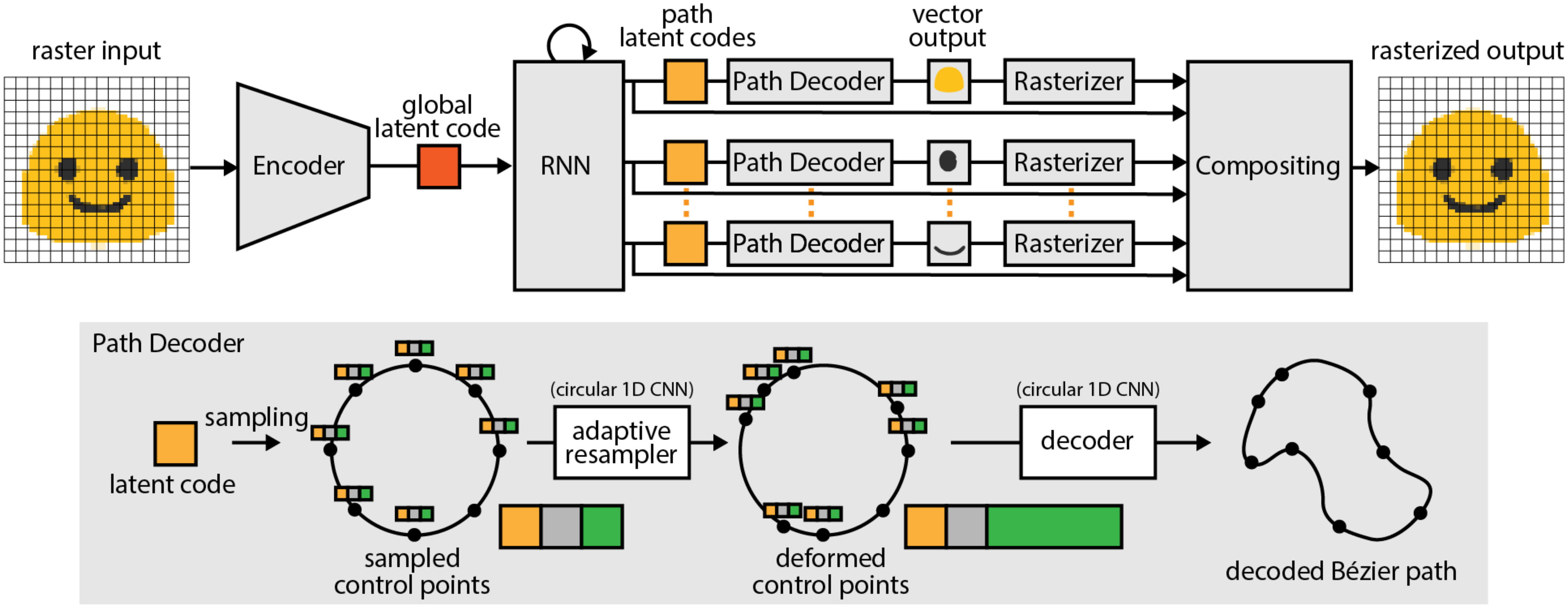
```
s2 = add_sketch('YZ')
add_line(.8, 2.8, 5.8, 2.8)
add_line(5.8, 2.8, 5.8, -2.8)
add_line(5.8, -2.8, .8, -2.8)
...
p2 = add_line(2, .2, 2.8, .2)
```

Extrude 3

```
add_extrude(
    sketch=s2,
    profile=p2[0],
    distance=14,
    operation='Cut')
```

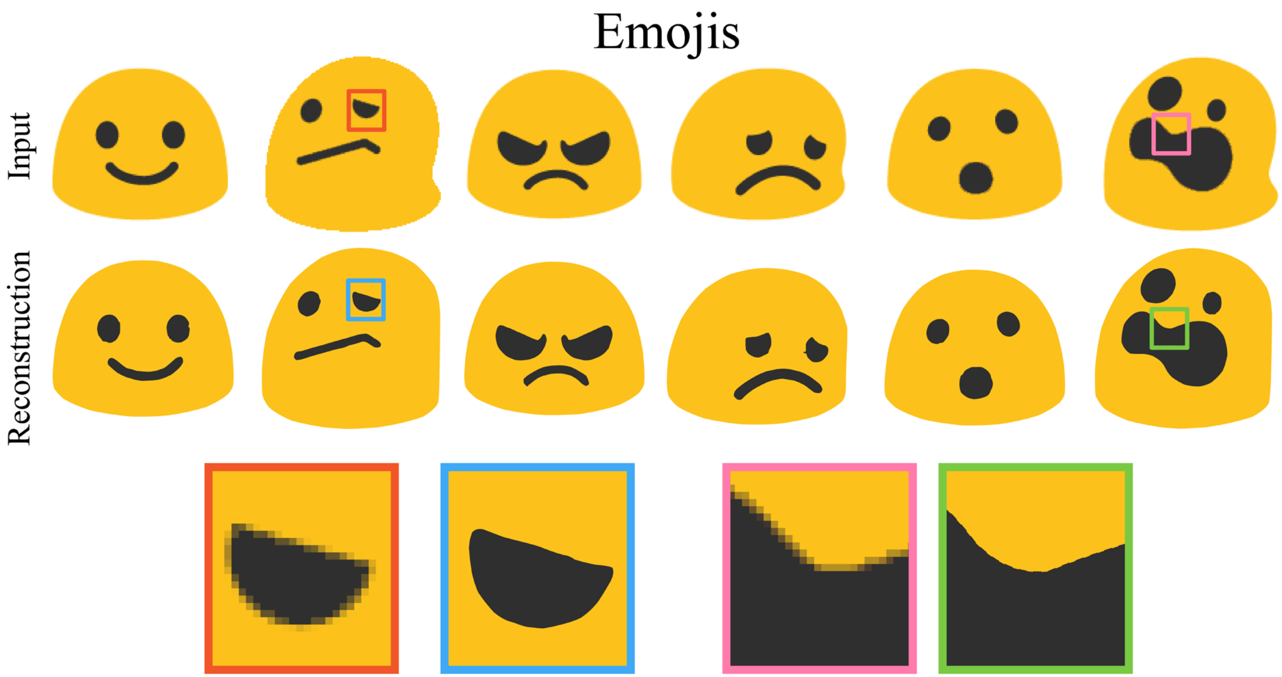
Im2Vec Inference Architecture

4 step process.



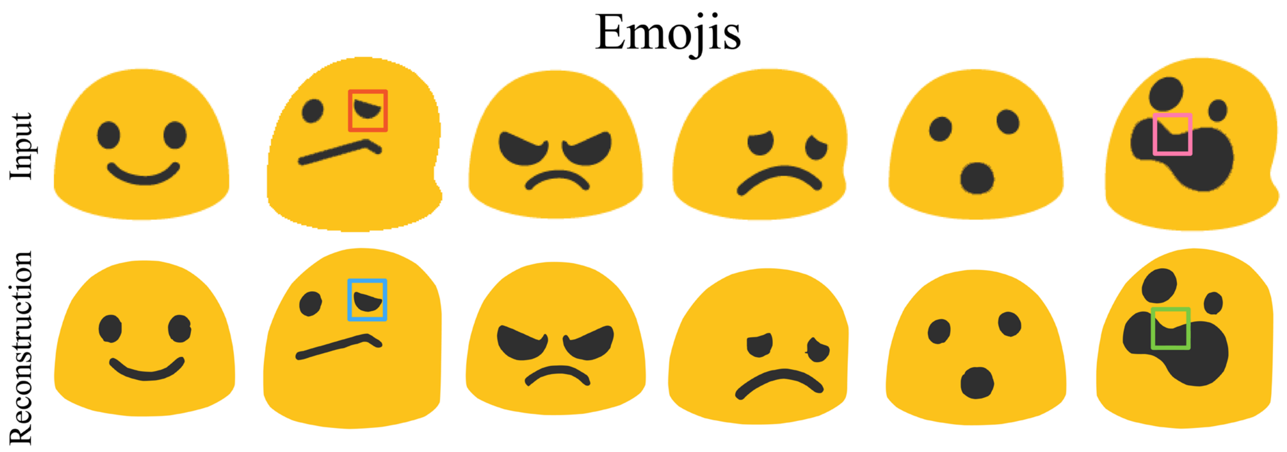
Reconstruction

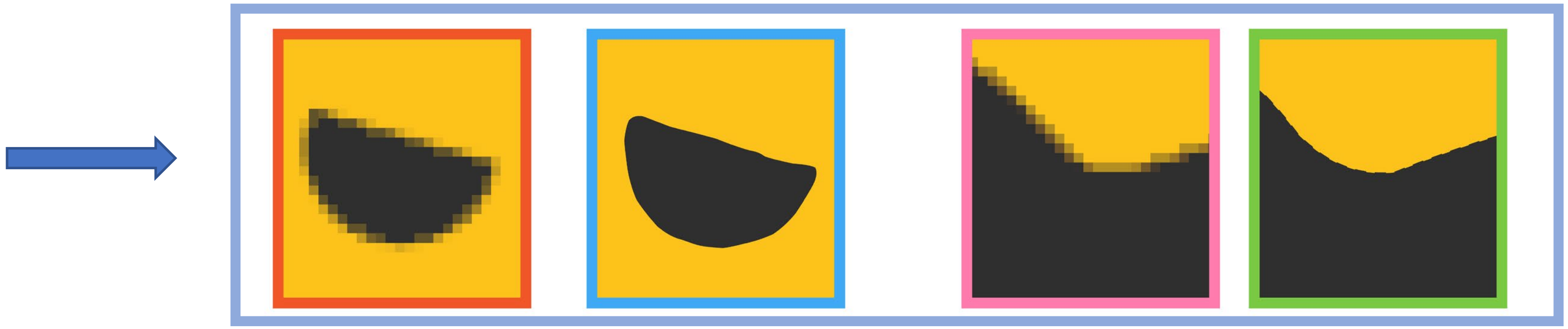
Input images are 128x128



Reconstruction

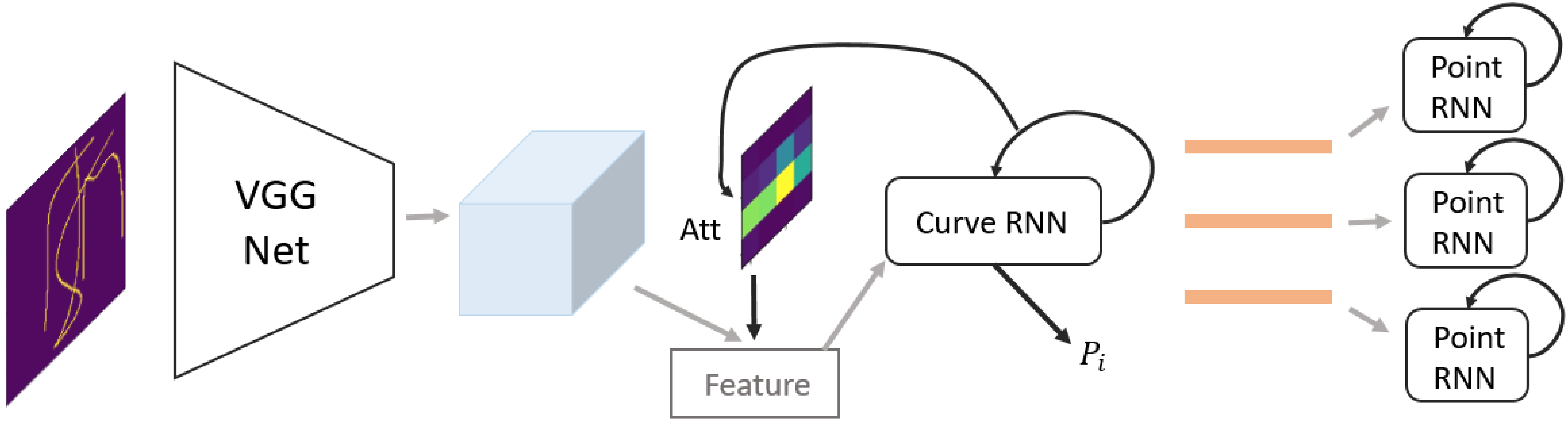
Input images are 128x128



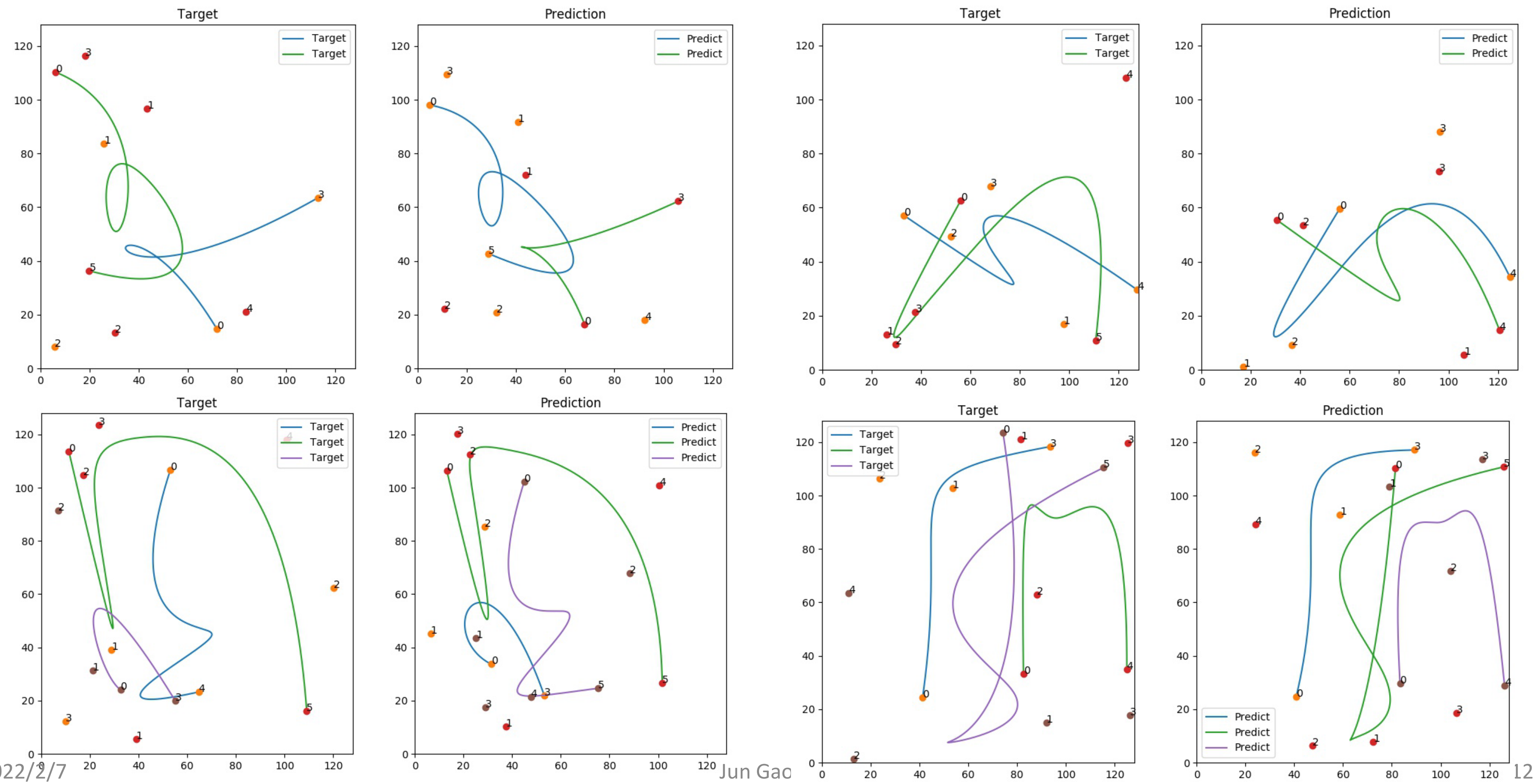


Multiple spline curves with variable numbers of control points

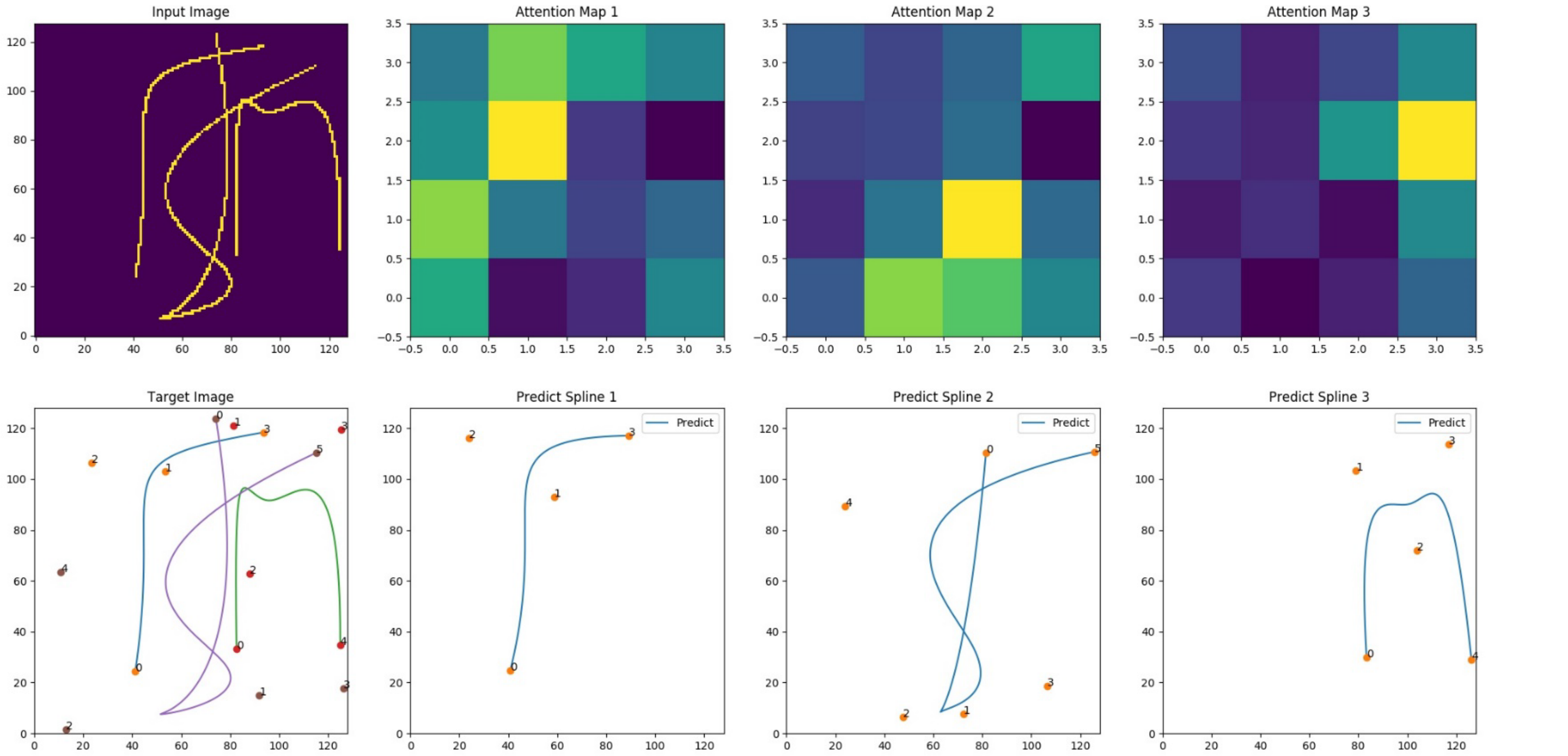
- Whole model

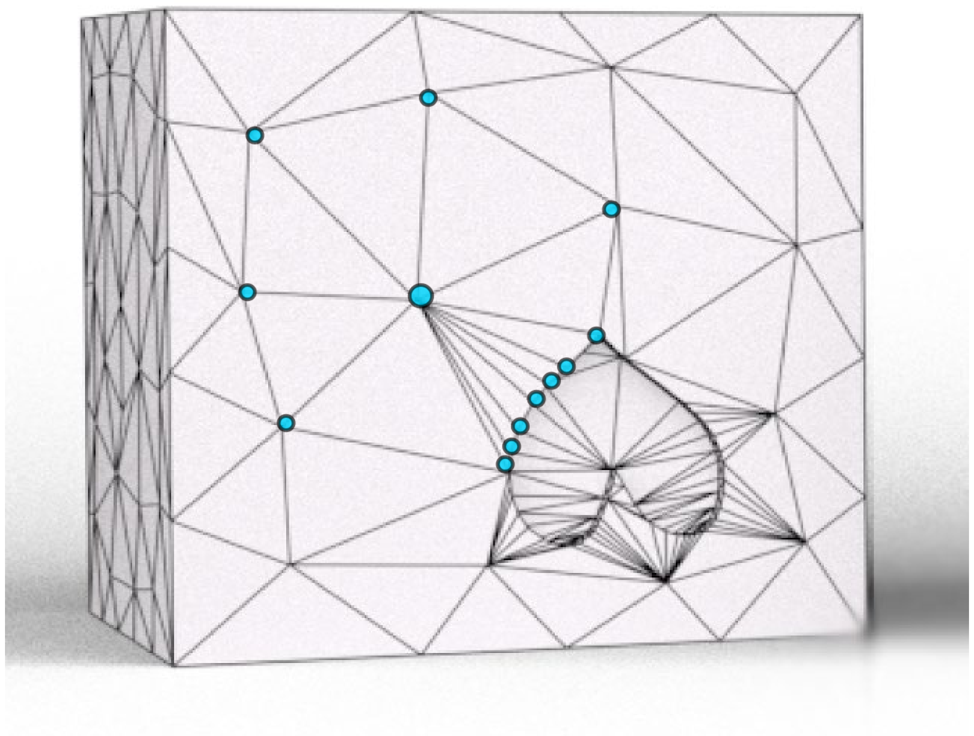


Results


Target
Target
Target
Prediction
Predict
Predict
Target
Target
Target
Prediction
Predict
Predict
Target
Target
Target
Target
Prediction
Predict
Predict
Predict
Target
Target
Target
Target
Prediction
Predict
Predict
Predict

Attention Results



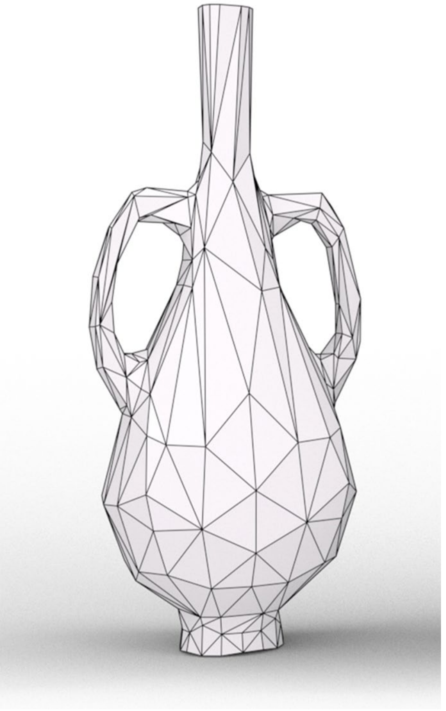


Irregular
Unordered
&
Unoriented

Goal: CNN directly on the irregular mesh elements

Classification

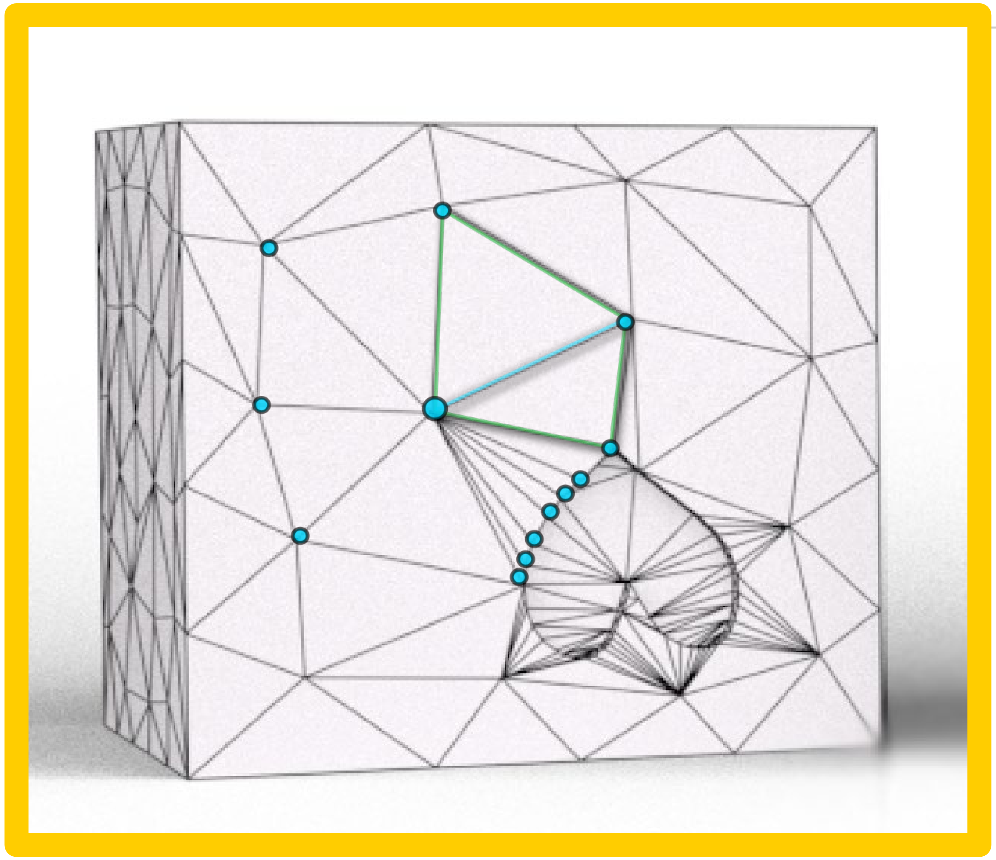
Segmentation



Vase

Fixed Size

Neighborhood



Vertices

<x, y, z>

Edges

<vi, vj>

Faces

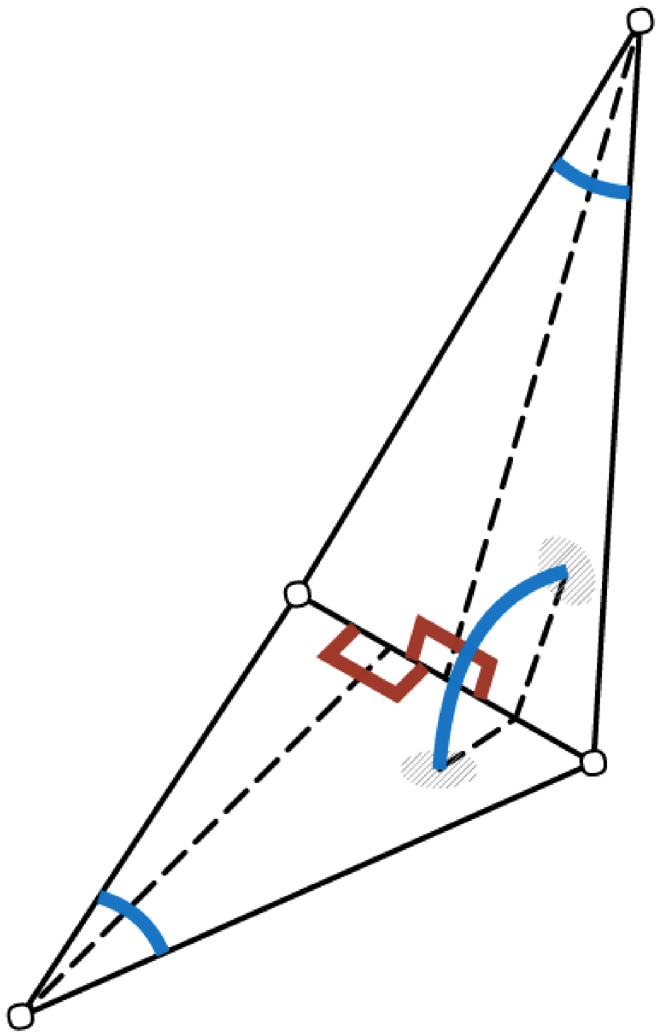
<vi, vj, vk>

Input Edge Features

Relative Geometric Features

→ Invariant to *similarity* transformations

5-dimensional vector



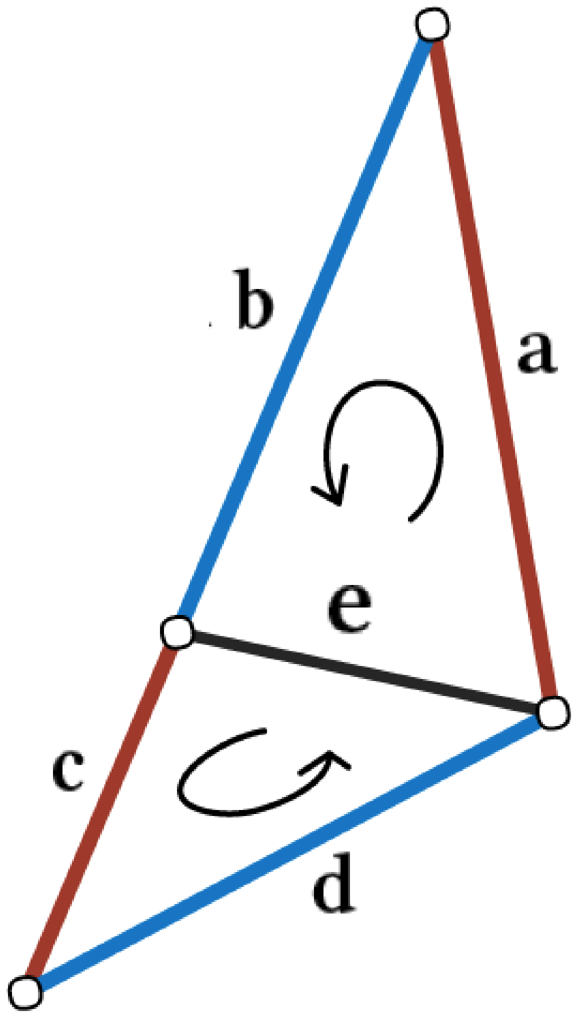
Mesh Convolution Order

Face normal

- ➔ Consistent ordering in each face
- ➔ Two *valid* orderings

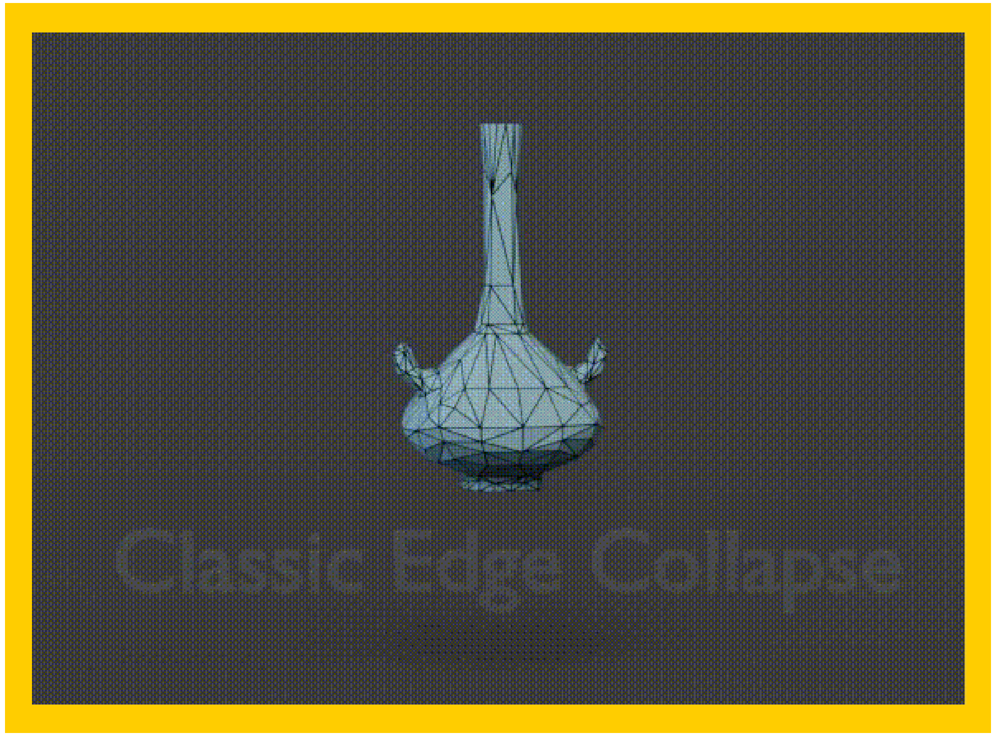
Solution: build symmetric features

- ➔ e-> (a+c, |a-c|, b+d, |b-d|)



Learned Edge Collapse

- Network decides collapse
- Strengthens the learned representation
- Visual insights from network

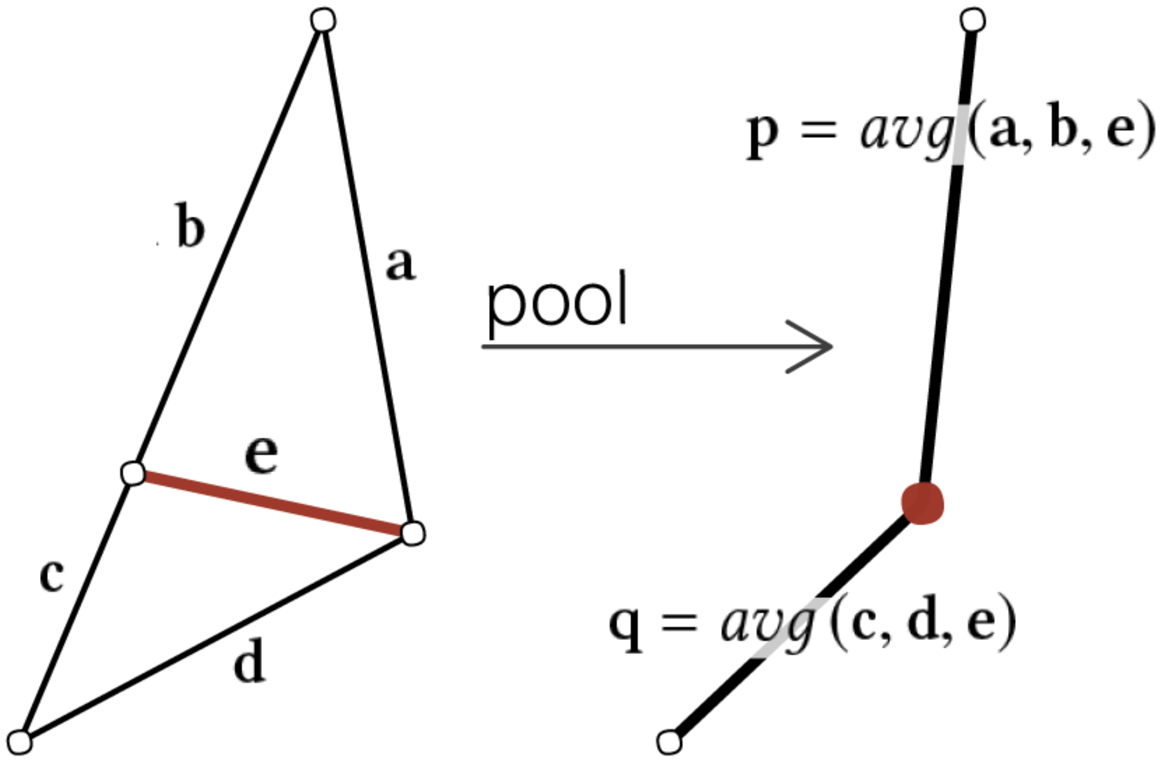


Mesh Pooling

Delete edge with smallest feature activations

→ Aggregate features

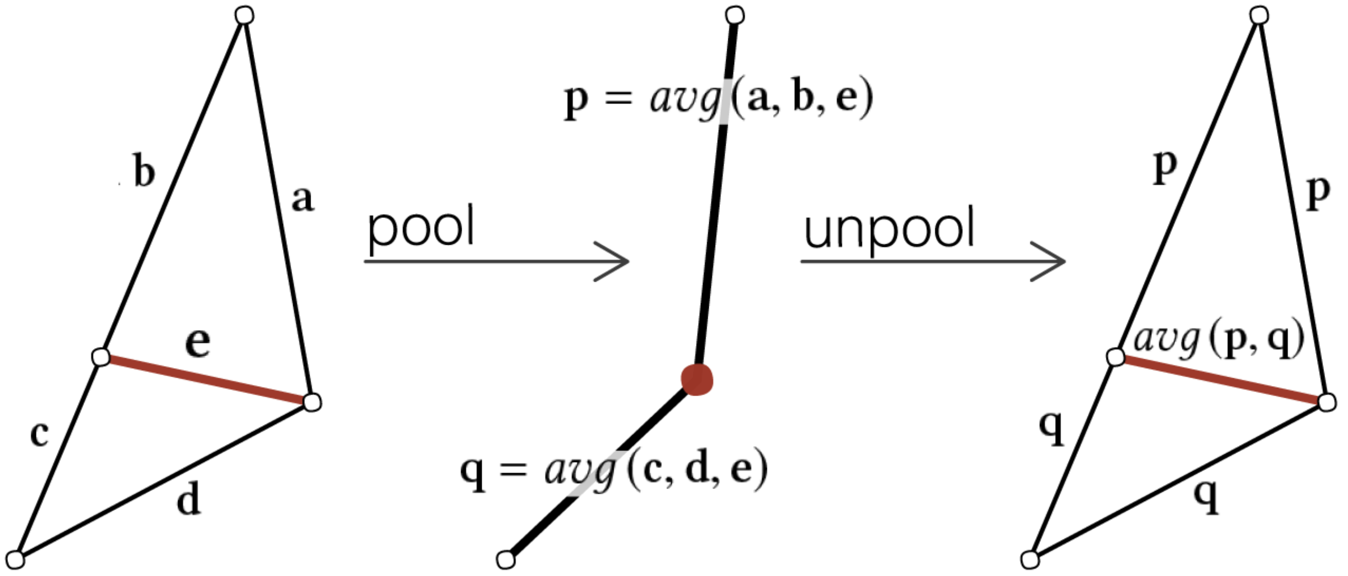
→ Update topology



Mesh Unpooling

Partial Inverse of Pooling

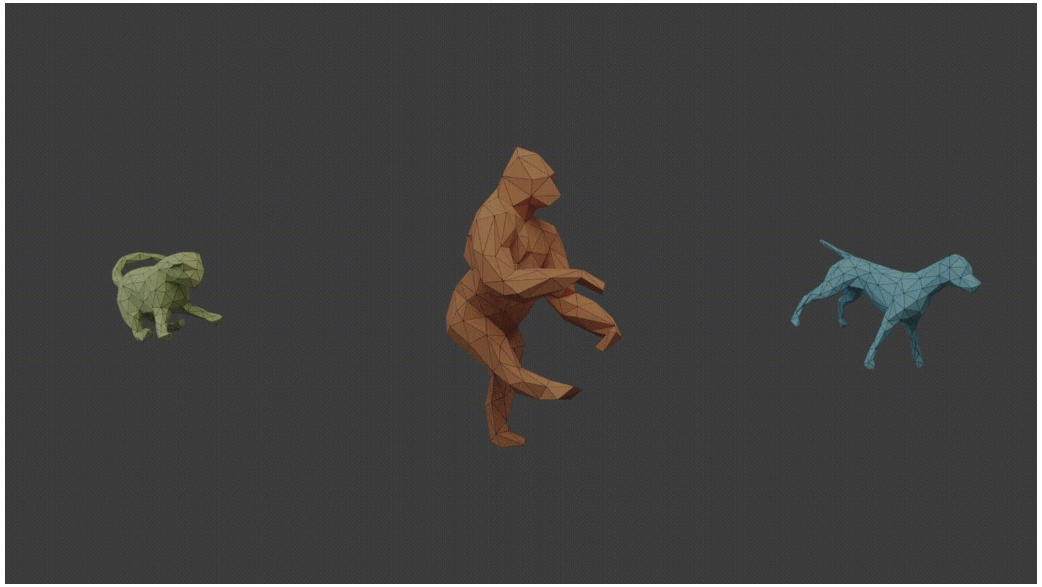
- Restores upsampled topology (reversible)
- Unpooled features weighted combination of pooled features



Applications of MeshCNN

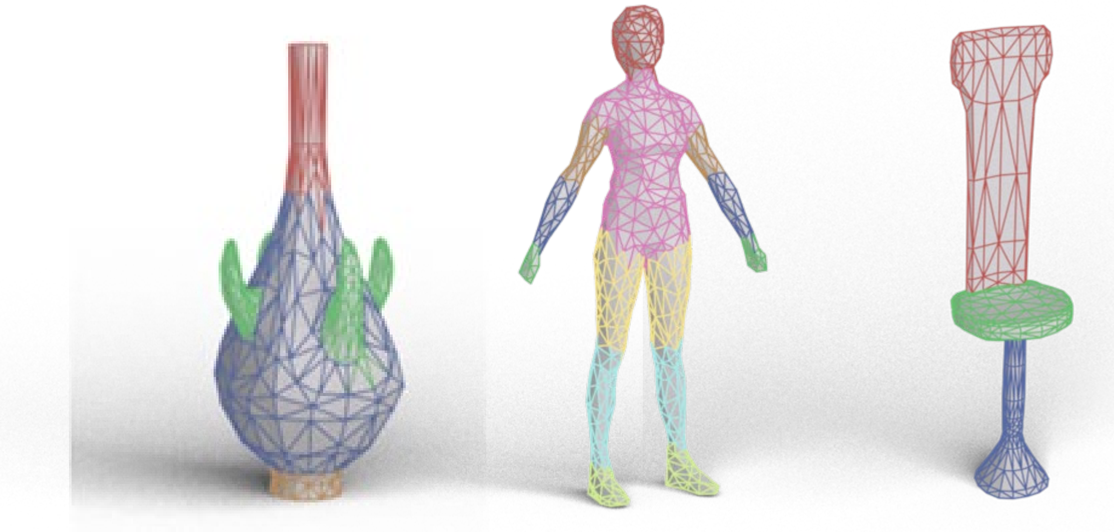
Classification

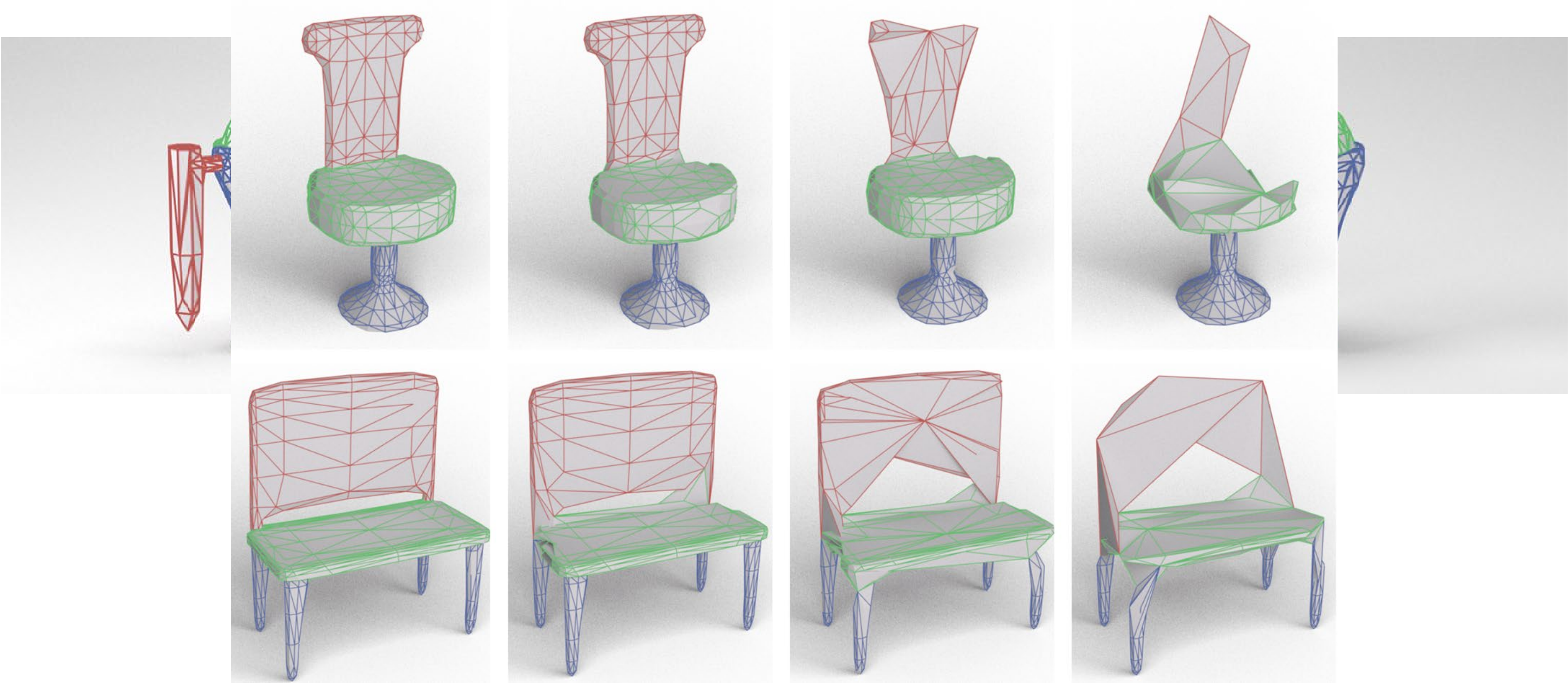
- Conv & Pooling Layers
- Fully-Connected Layers



Segmentation

- Fully convolutional
- Conv & Pooling & Unpooling



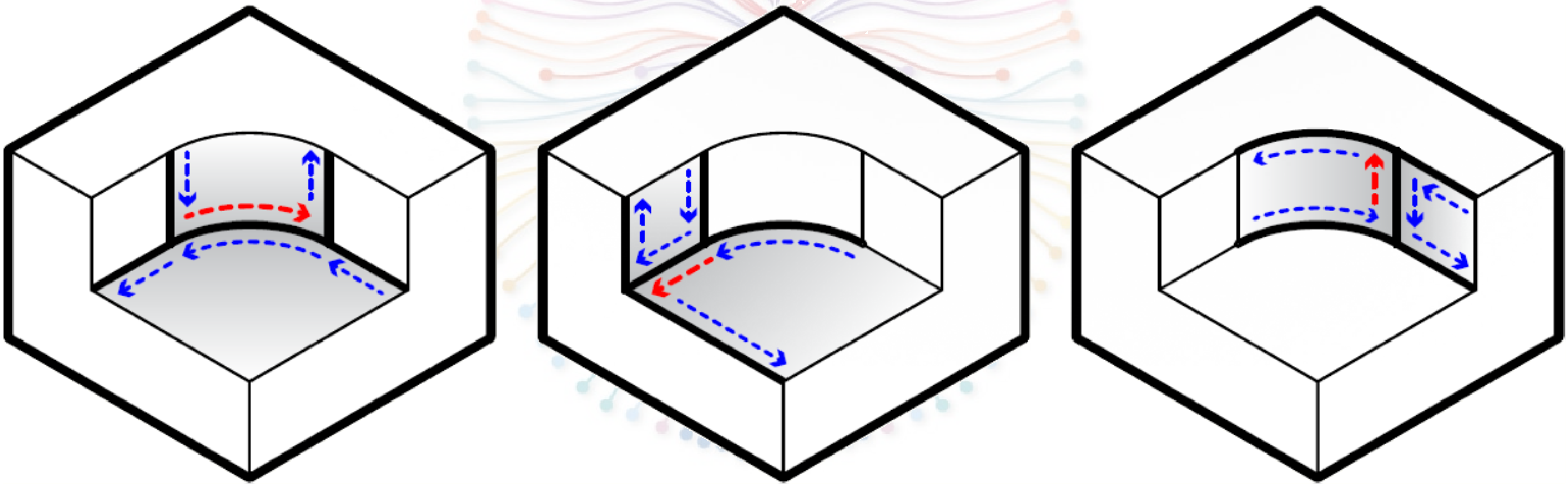


Segmentation

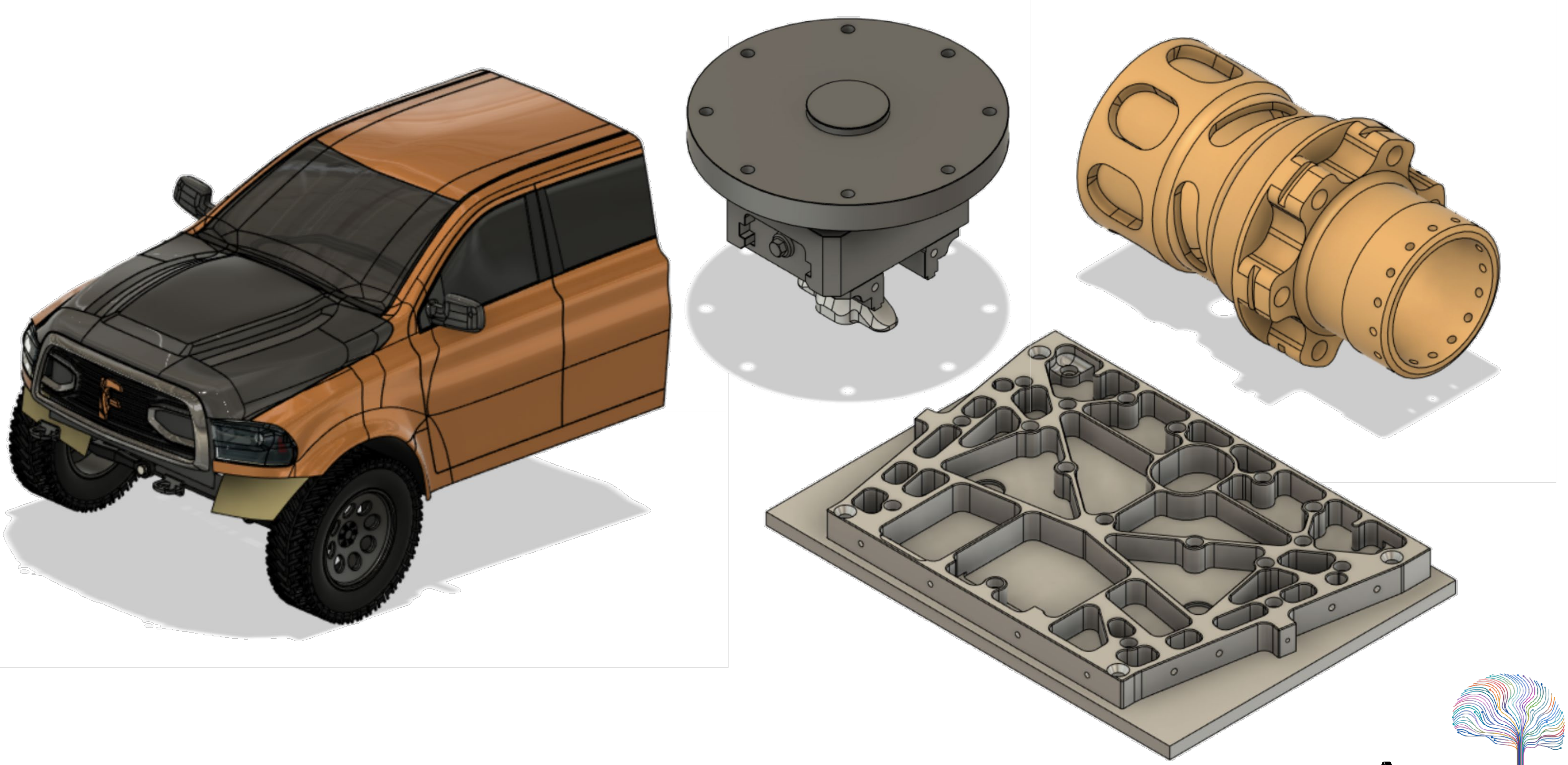
BRepNet: A topological message passing system for solid models

Joseph G. Lambourne[1], Karl D.D. Willis[1], Pradeep Kumar Jayaraman[1], Aditya Sanghi[1], Peter Meltzer[2], Hooman Shayani[1]

Autodesk Research[1], UCL, Computer Science[2]

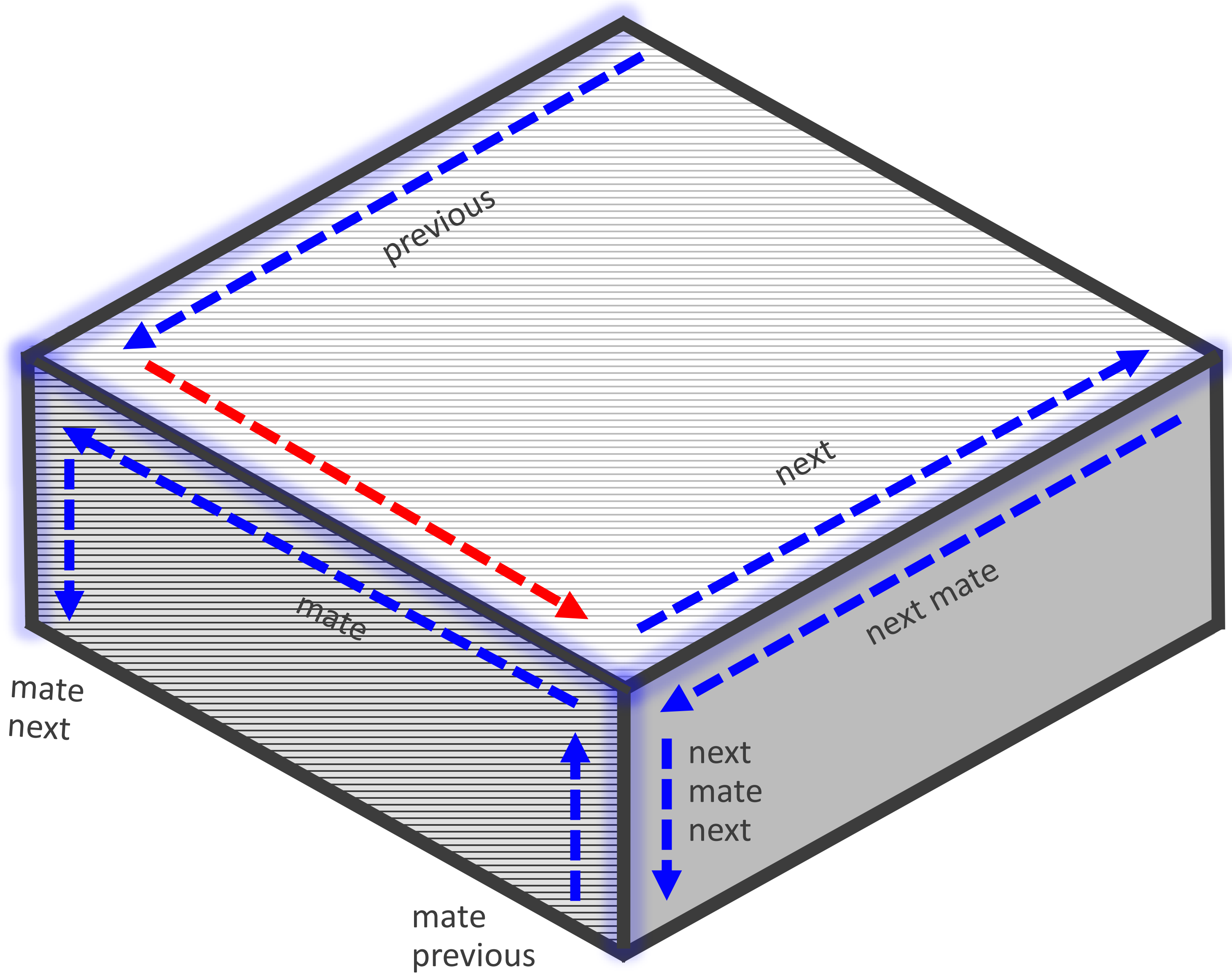


Boundary representation models



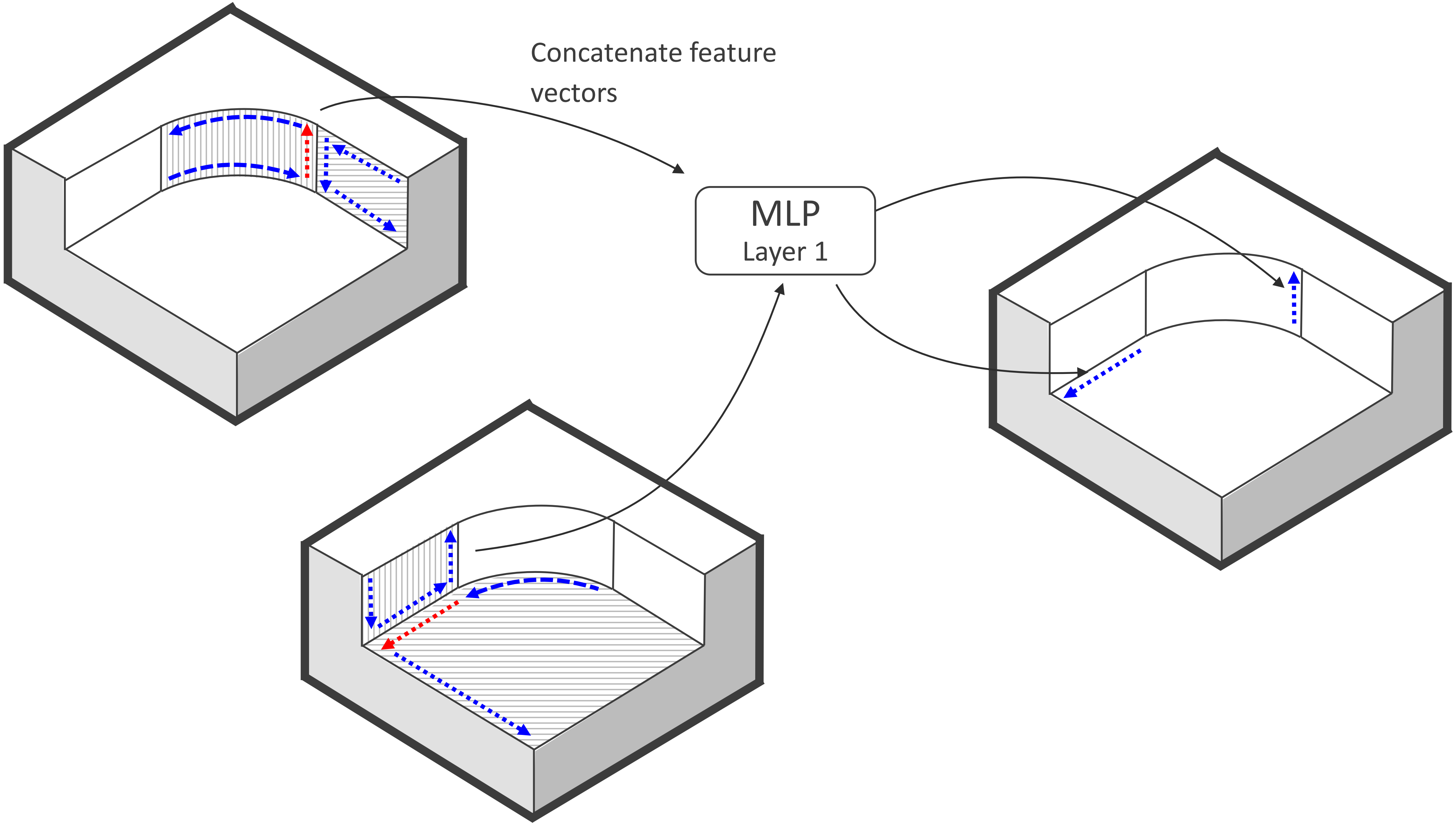
AUTODESK AI LAB

Topological walks defining neighboring entities



AUTODESK AI LAB

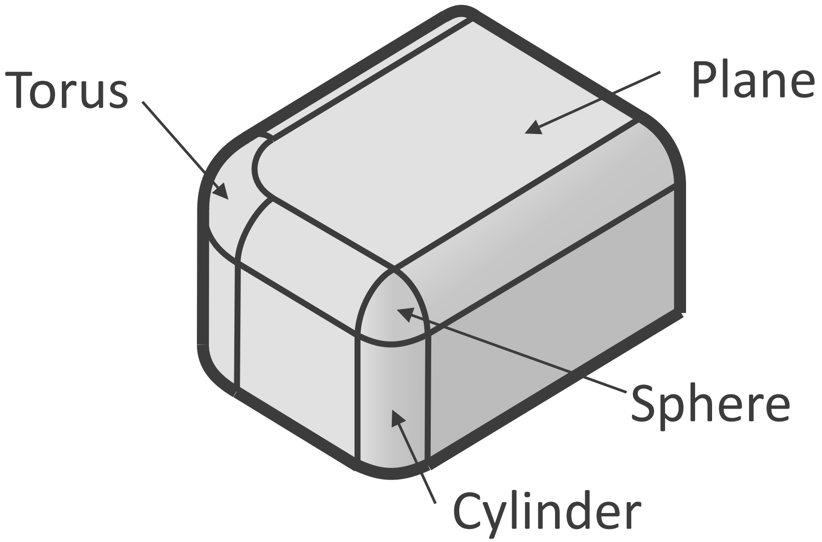
BRepNet convolution

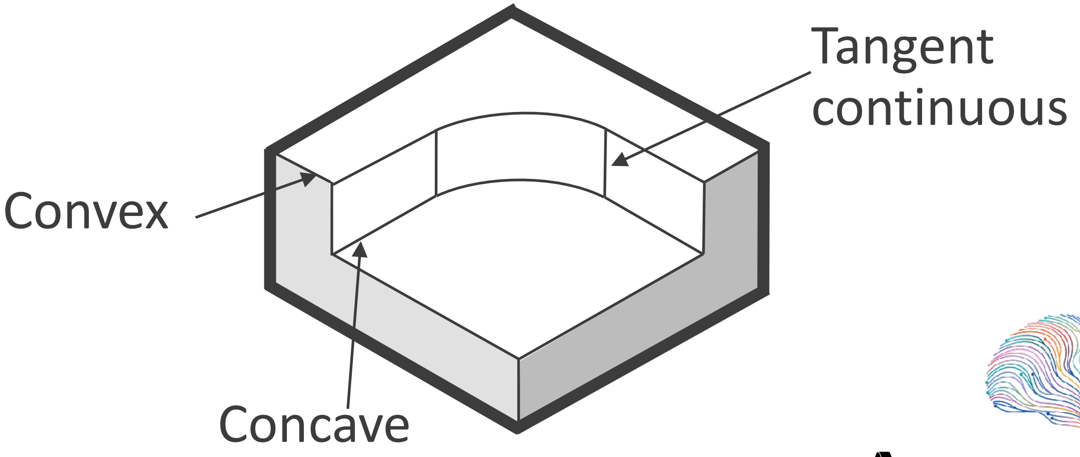


AUTODESK AI LAB

Input features

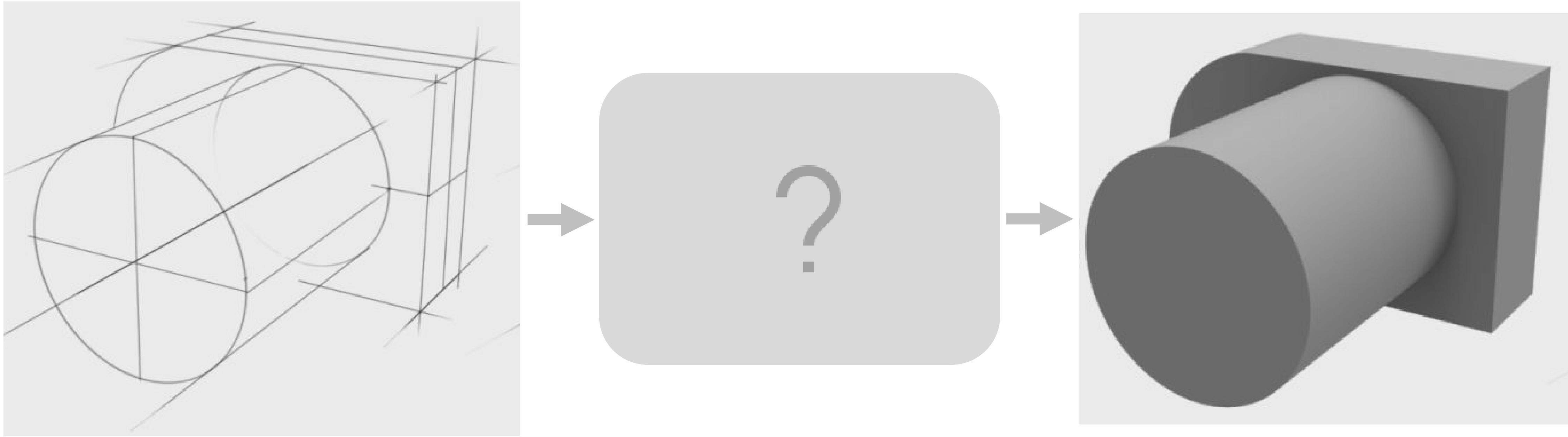
- Face features
  - Surface type (plane, cylinder, cone, sphere, torus, spline)
  - Face area
- Edge features
  - Curve type (line, circle, ellipse, intersection curve, spline)
  - Edge convexity (concave, convex, tangent plane continuous)
  - Edge length





Sketch2CAD Goal

- Gap: converting sketches to a CAD model?

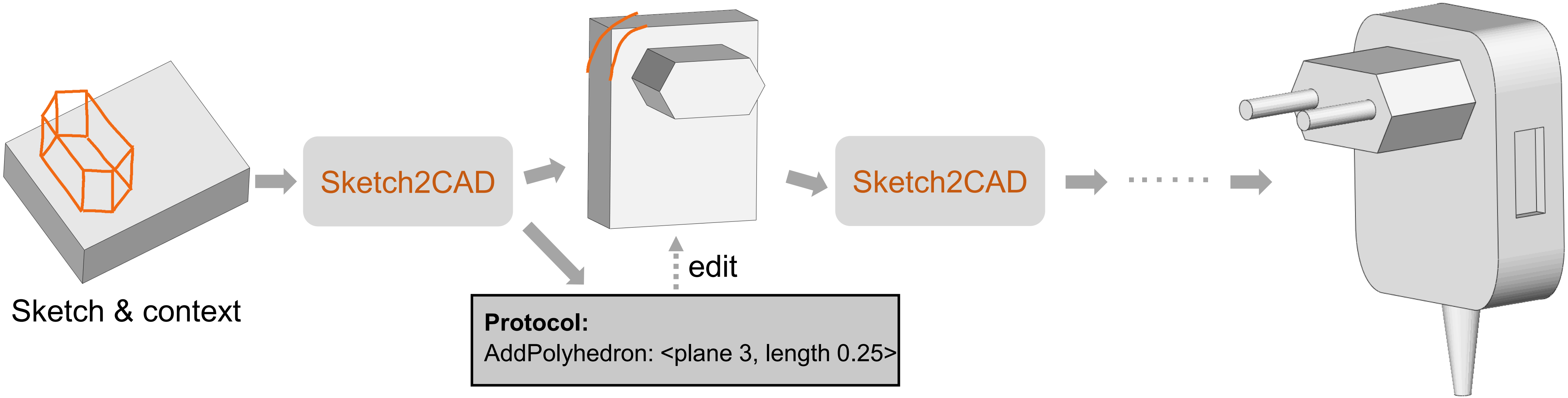


User sketches

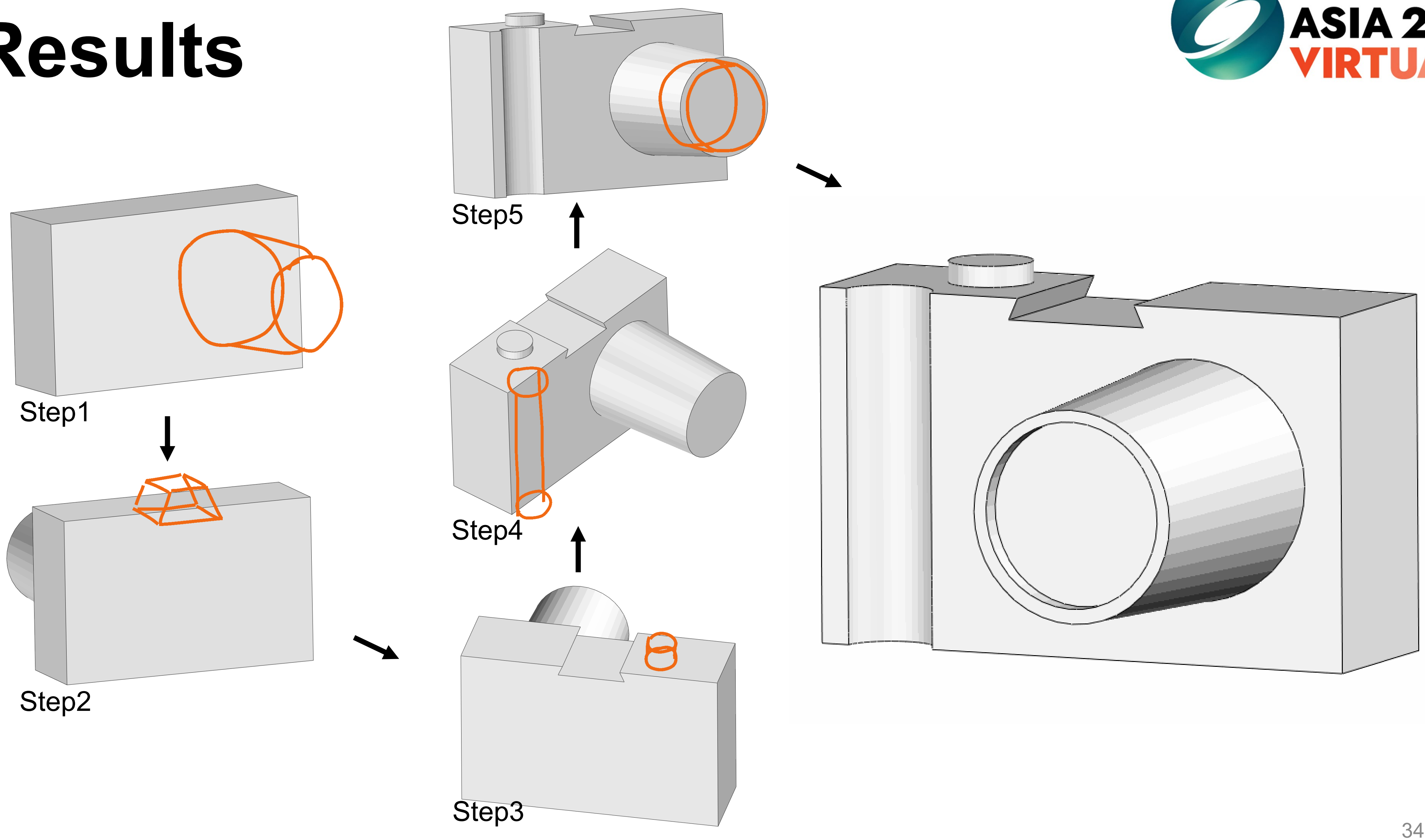
3D model

Sketch2CAD

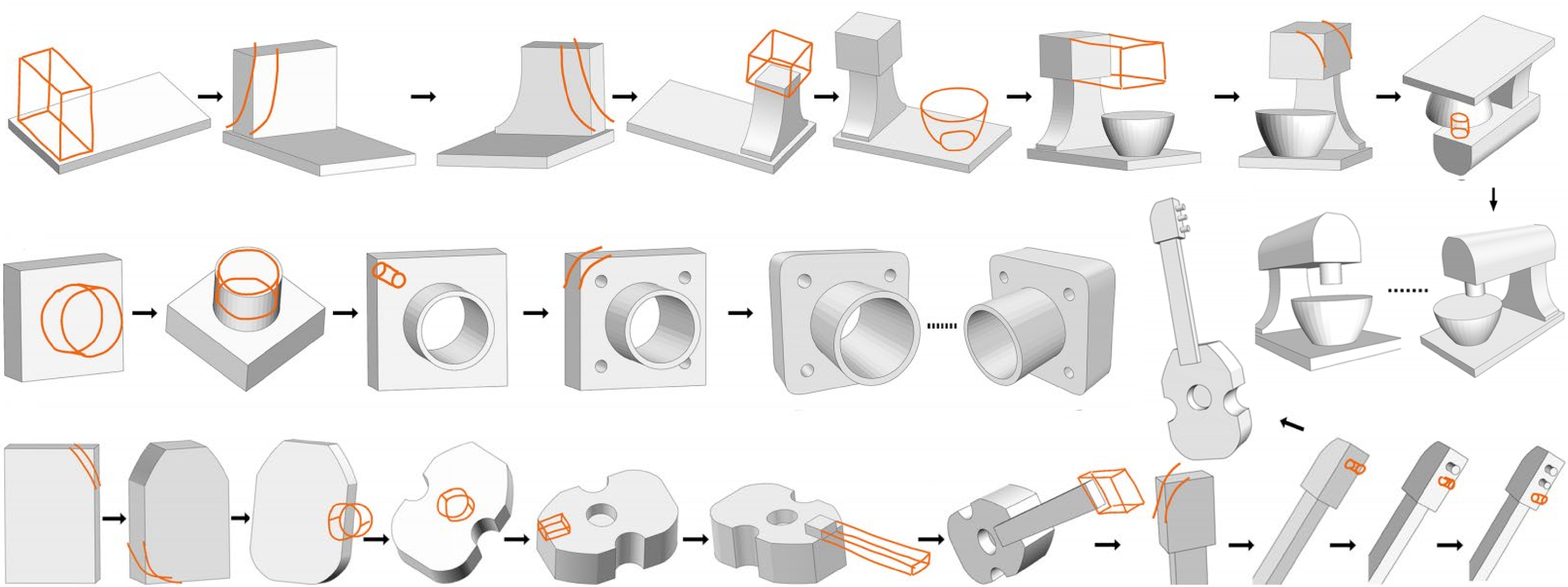
- A learning-based modeling system that translates sketching operations to their corresponding CAD operations, along with the associated parameters.



Results



Results



3D Equivariance and Invariance

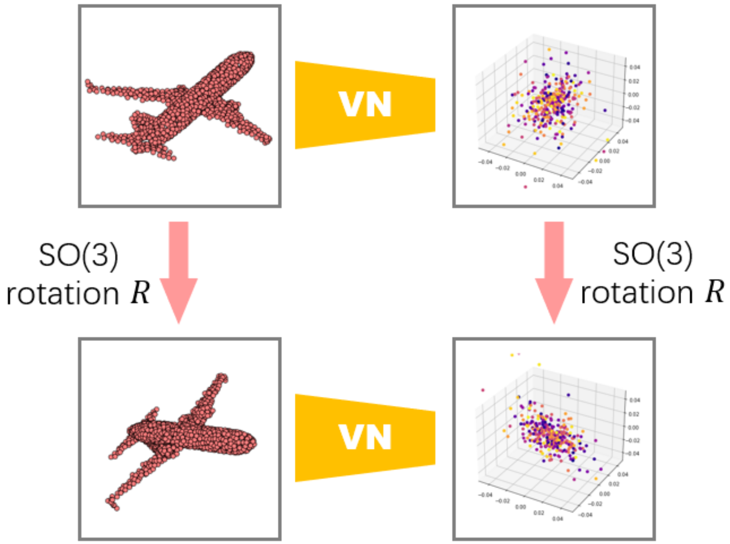
Class Questionnaire

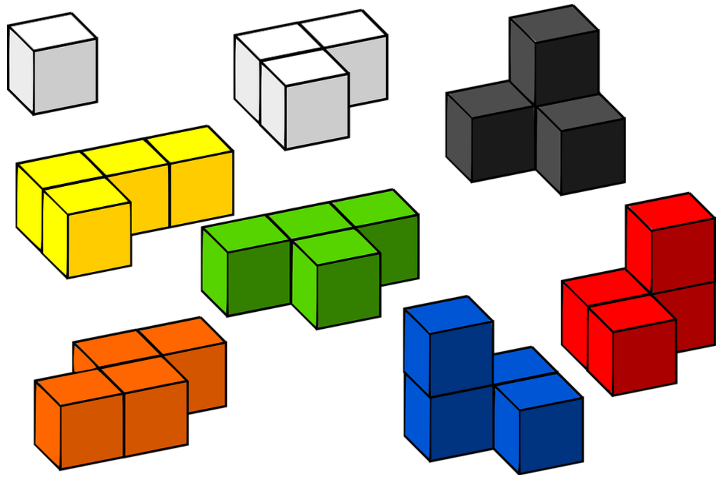
Class Questionnaire

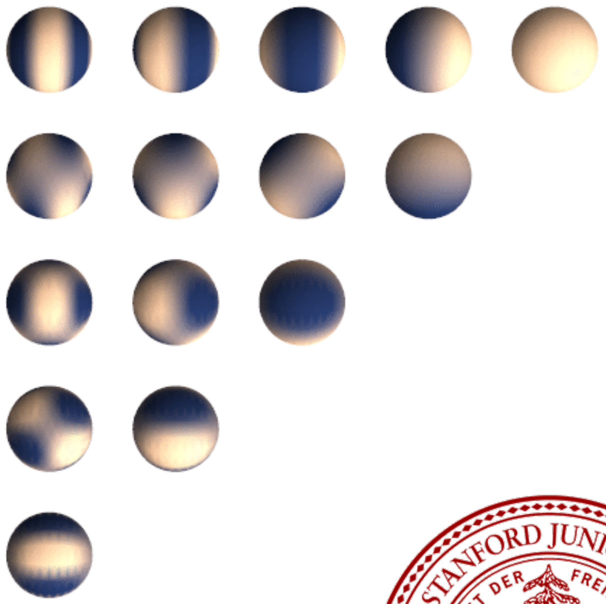
- Please take the questionnaire below to provide us with feedback on the class:

- https://forms.gle/igFFpmnWaWL11Tfw9

CS348N Lecture 10:
3D Rotation Equivariance and Invariance







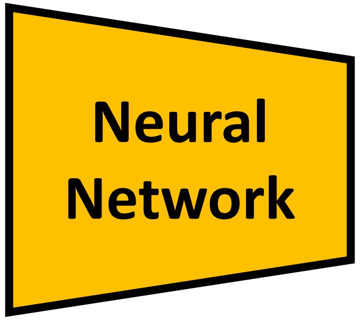
Leonidas Guibas Laboratory

Geometric Computing

Stanford University

Why Equivariance / Invariance?

- What neural networks can do?



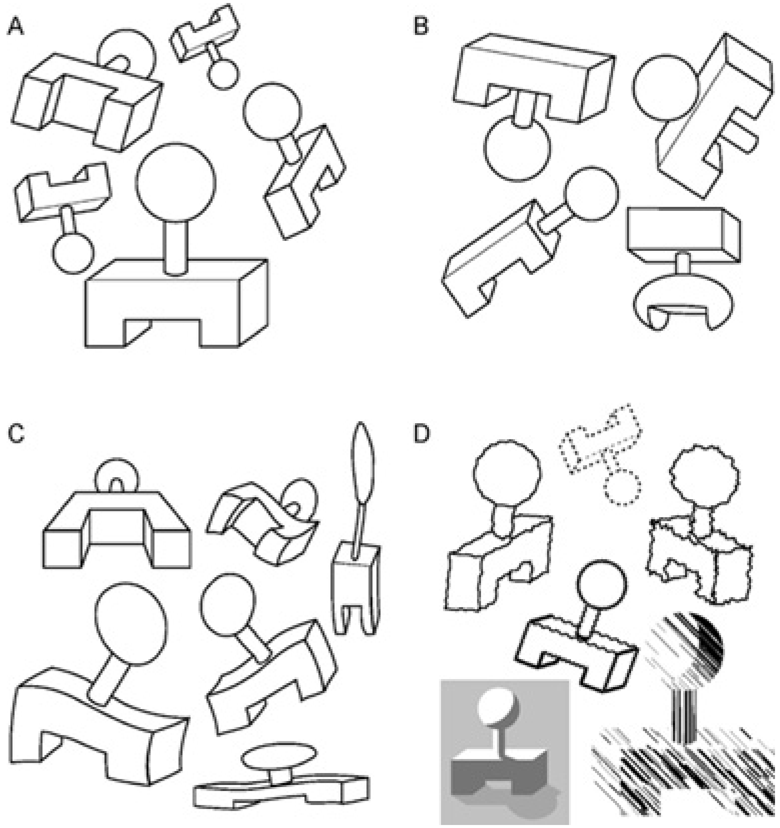
“chair”

- But what is this?

Not too Hard for a Human…

Gestalt theory – how human perceive objects

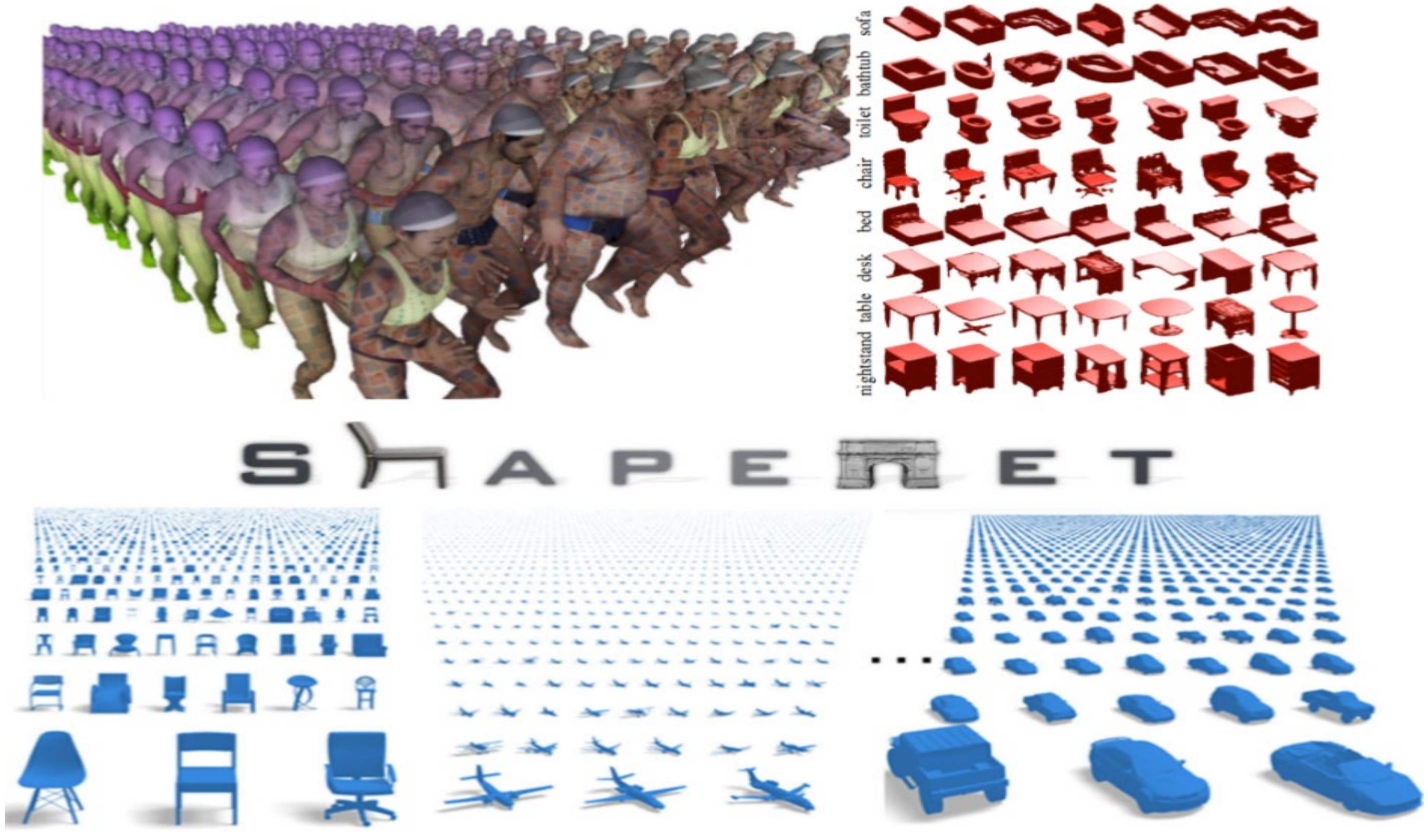
- Emergence
- Reification
- Multistability
- **Invariance**



https://en.wikipedia.org/wiki/Gestalt_psychology#Properties

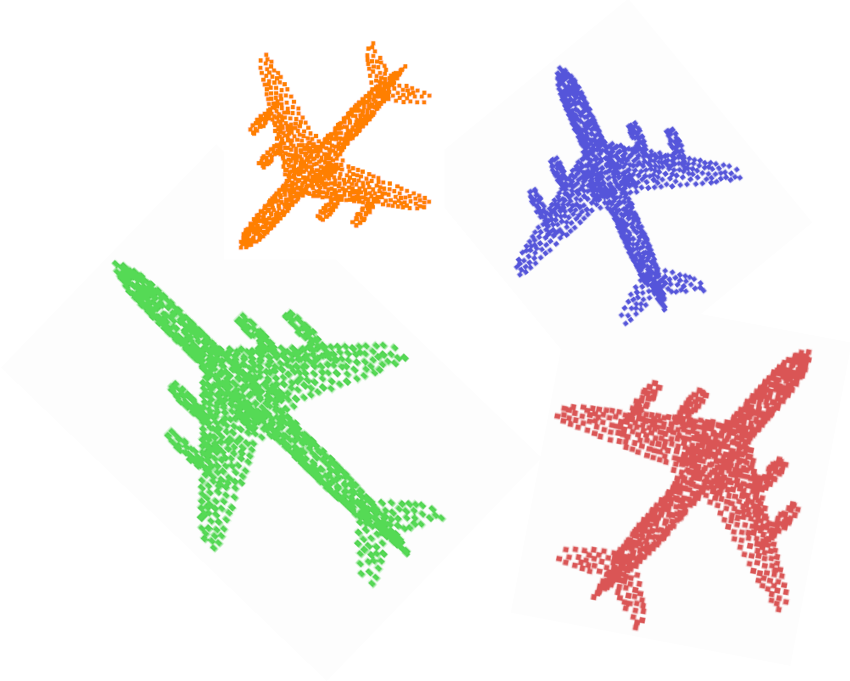
But Neural Networks Struggle…

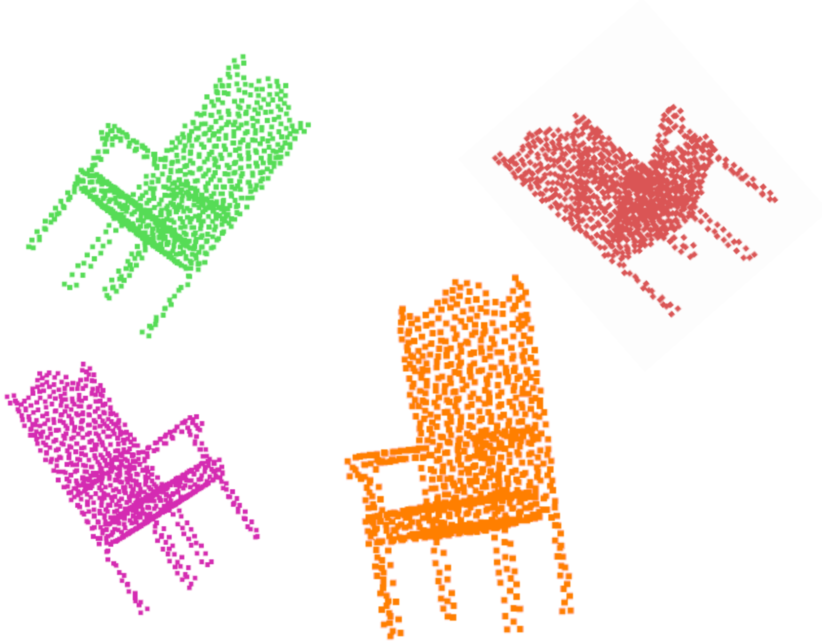
- Many datasets are **aligned** (shapes under canonical poses)
- Networks trained aligned data **cannot generalize** to arbitrary poses



But Neural Networks Struggle…

Identical objects coming in different poses, scales, ratios, colors…
– viewed as totally irrelevant entities by classical neural networks



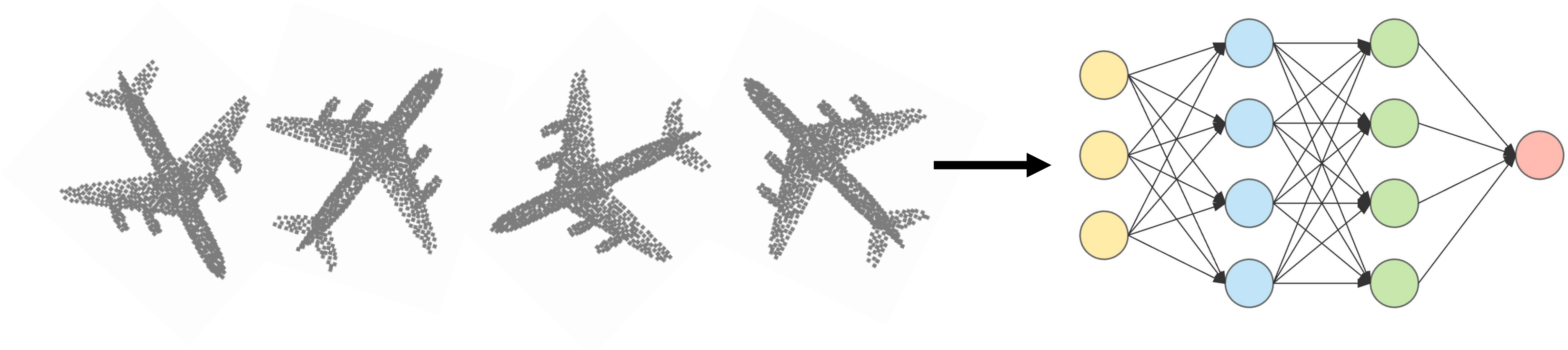


A Naïve Solution: Data Augmentation

Apply random rotation to the training data

So we let the network “see” and learn from all possible poses

- Reducing the generalization gap – but not eliminating it
- Sacrificing data-efficiency – longer training time
- Statistically equivariant/invariant – not guaranteed

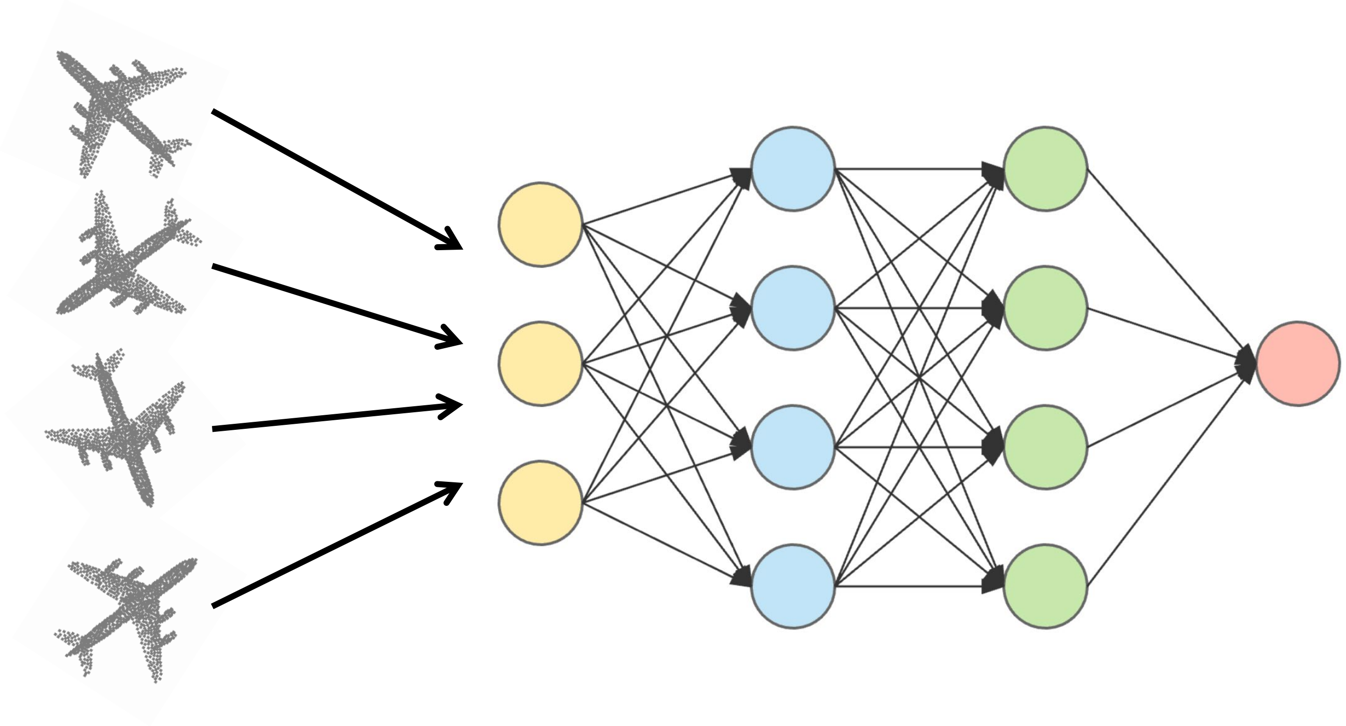


A Less Naïve Solution: Multi-view Approach

Feed multiple poses of the same object to the network **at once**

“If we don’t know what pose to look at, why not just look at **all** poses!”

- Can reproduce fairly good results
- Sacrifice data-efficiency – more memory consumption
- Theoretically equivariant – up to precision errors caused by discretization

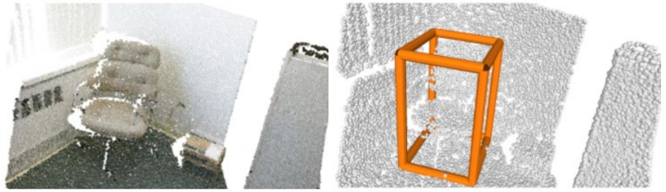


Summary: Why Equivariance / Invariance?

✓**"Intelligence":** More aligned with human perception

✓**Generalization:** Eliminate the prior that all shapes in a dataset (e.g., ShapeNet) are aligned

✓**Real-world applications:** Shapes (3D scans) in the wild may not have or come with canonical poses

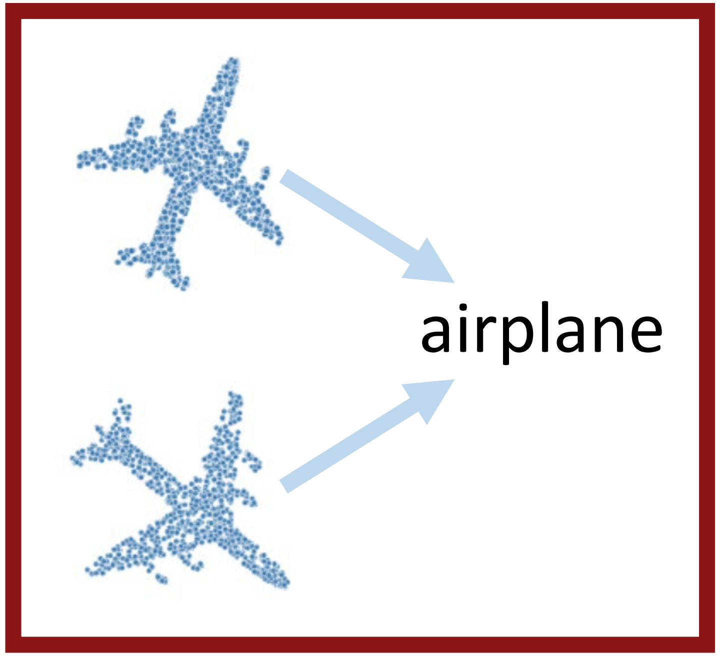


✓**Data-efficiency:** Avoid exhaustive data augmentation

[A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, ... F. Yu, arXiv:1512.03012 (2015)]

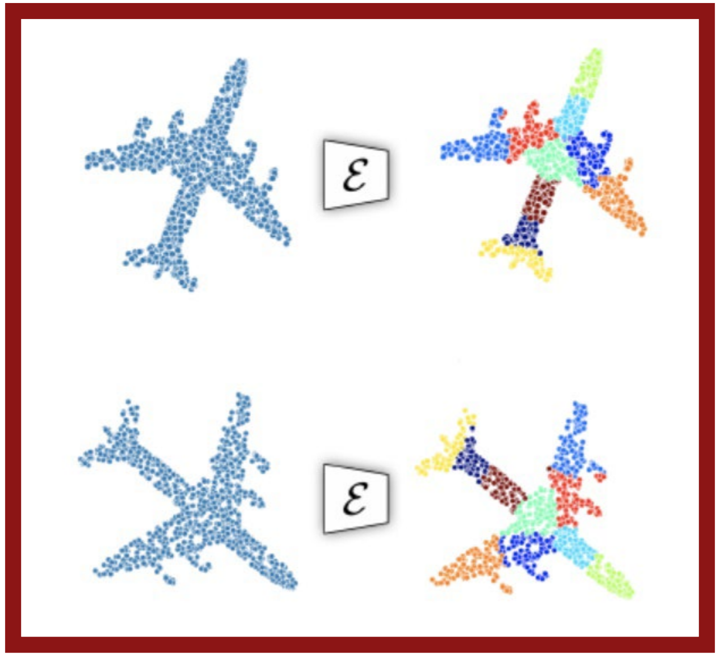
So We're Looking for…

classification



invariant

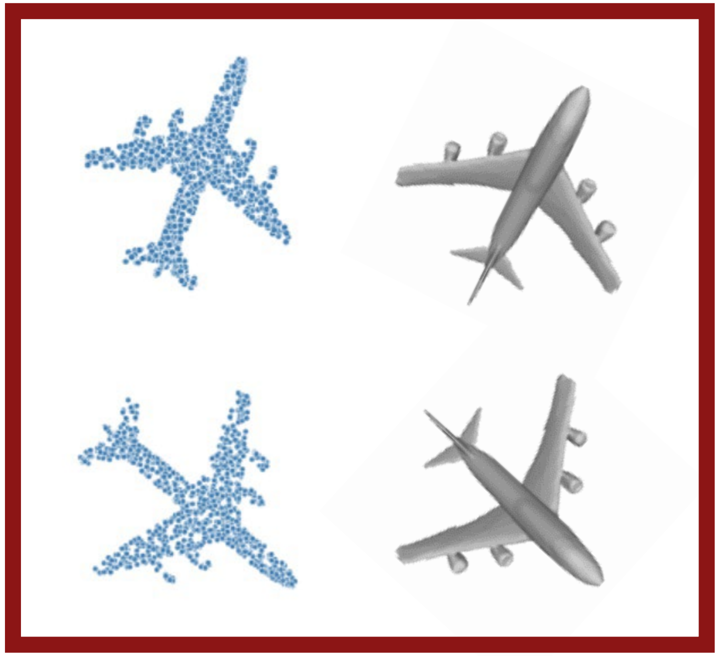
segmentation



equivariant

(or **invariant**, depending on data representation)

reconstruction



equivariant encoder

invariant decoder

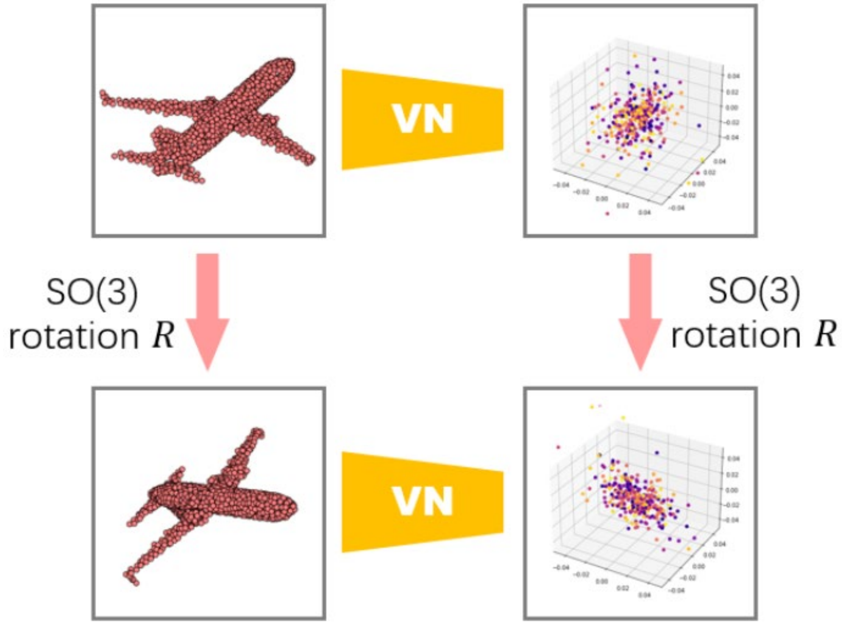
[W. Sun, A. Tagliasacchi, B. Deng, S. Sabour, S. Yazdani, G. Hinton, K. M. Yi, arXiv:2012.04718 (2020)]
[J. J. Park, P. Florence, J. Straub, R. Newcombe, S. Lovegrove, CVPR 2019]

Equivariance!

We say a neural network $f(\cdot\,;\theta)$ is rotation equivariant, if for any 3D rotation $R \in \mathrm{SO}(3)$ applied to its input $\mathbf{x}$, it is explicitly related to a transformation $D(R)$ on the network output satisfying

$$f(\mathbf{x}R;\theta) = f(\mathbf{x};\theta)D(R)$$

- $D(R)$ should be independent of $\mathbf{x}$
- **Special case:** when $D(R) = R$ is the identity mapping, it is the common-sense “equivariance”
- **Special case:** when $D(R) = \mathrm{I}$ is the constant mapping, it is invariance



Vector Neurons (VN)

[C. Deng, O. Litany, Y. Duan, A. Poulenard, A. Tagliasacchi, L. J. Guibas, ICCV 2021]

Vector Neurons

Classical (scalar) feature $\boldsymbol{z} = [z_1, z_2, \cdots, z_C]^\top \in \mathbb{R}^C$, **with** $z_i \in \mathbb{R}$

Vector-list feature $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_C]^\top \in \mathbb{R}^{C\times 3}$, **with** $\boldsymbol{v}_i \in \mathbb{R}^3$

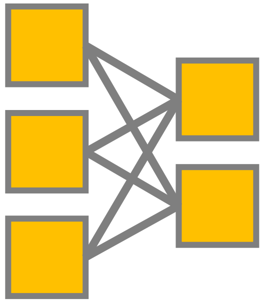
- For pointcloud with N points $\mathcal{V} = \{\boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_N\} \in \mathbb{R}^{N\times C\times 3}$

Mapping between network layers:

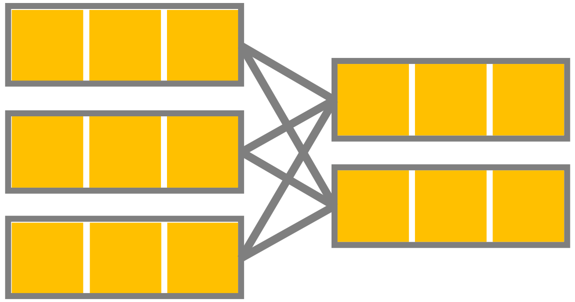
$$f(\cdot;\theta) : \mathbb{R}^{N\times C^{(d)}\times 3} \to \mathbb{R}^{N\times C^{(d+1)}\times 3}$$

? Equivariance to rotation $R \in \mathrm{SO}(3)$:

$$f(\mathcal{V}R;\theta) = f(\mathcal{V};\theta)R$$



(classical) scalar neurons

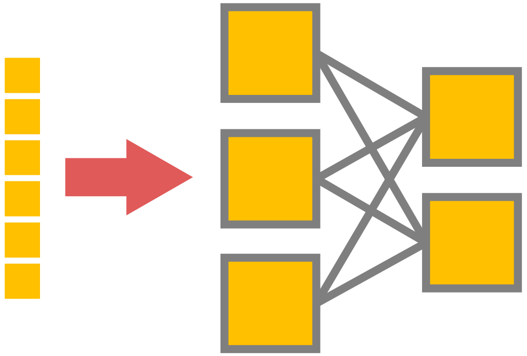


vector neurons

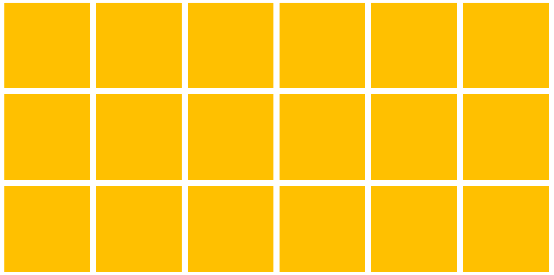
VN Features

Classical: scalar channels

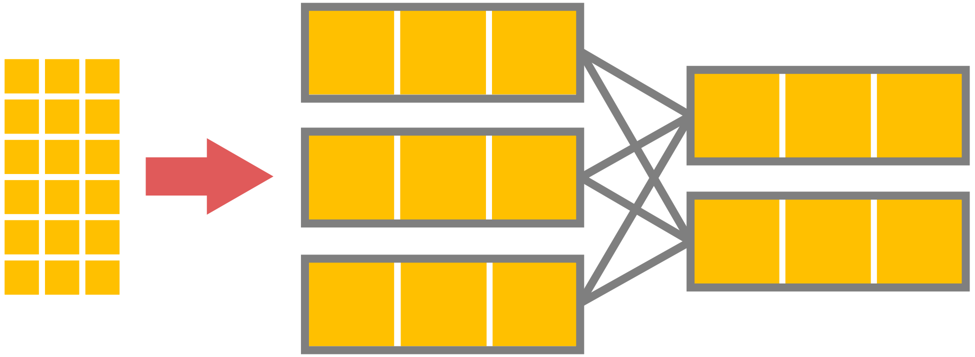
$C \times 1$ feature



VN: 3D vector channels

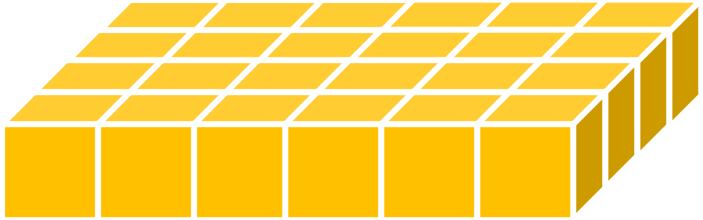


$C \times 3$ feature



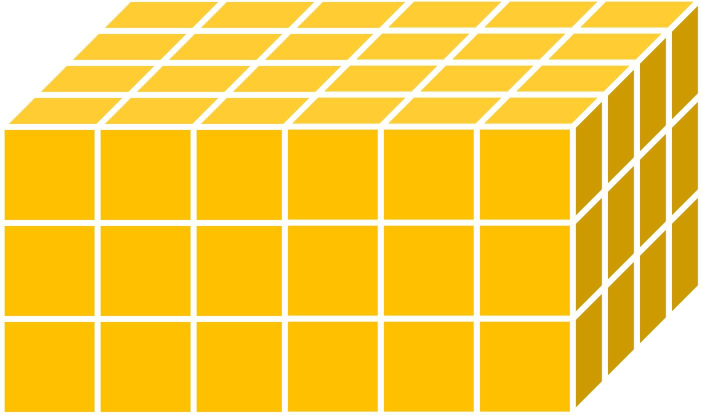
VN Features (for Point Cloud)

Classical:

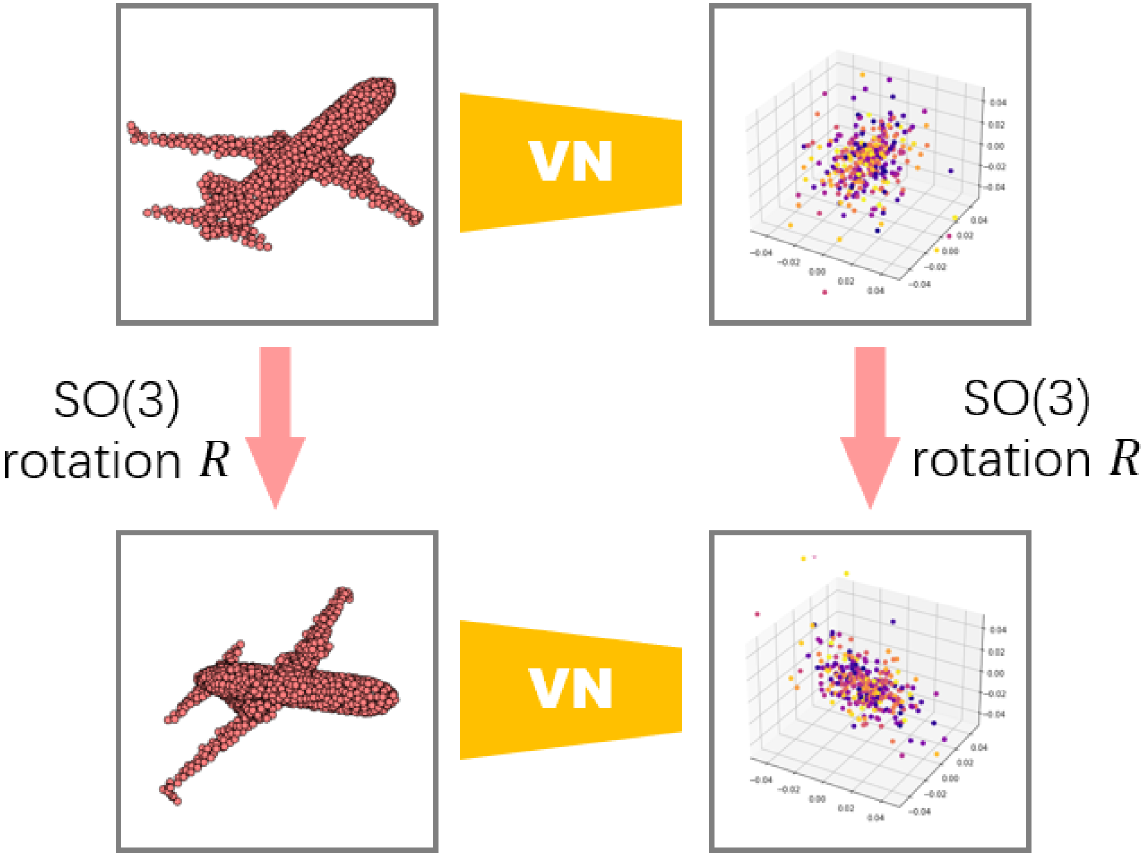


$N \times C \times 1$ feature

VN:

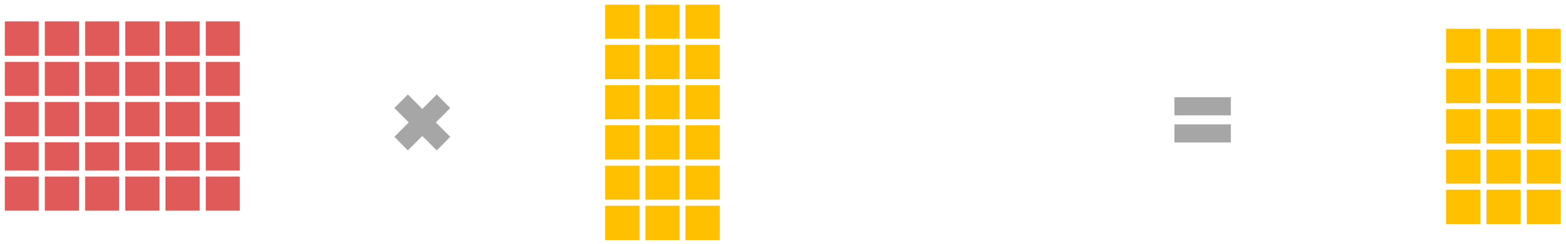


$N \times C \times 3$ feature



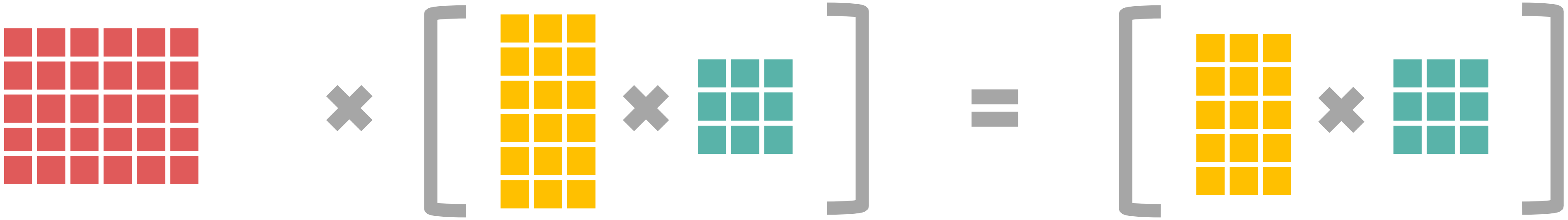
VN Linear Layer

Linear operator: left multiply by the learnable weight matrix



$C' \times C$ weight $C \times 3$ feature $C' \times 3$ feature

Equivariance: right multiply by the SO(3) rotation matrix



$C' \times C$ weight $C \times 3$ feature $C' \times 3$ feature

VN Linear Layer

Vector-list feature $\boldsymbol{V} \in \mathbb{R}^{C\times 3}$

Linear operator $f_{\mathrm{lin}}(\,\cdot\,;\mathbf{W})$ with learnable weights $\mathbf{W} \in \mathbb{R}^{C'\times C}$:

$$\boldsymbol{V}' = f_{\mathrm{lin}}(\boldsymbol{V};\mathbf{W}) = \mathbf{W}\boldsymbol{V} \in \mathbb{R}^{C'\times 3}$$

Equivariance to rotation $R \in \mathrm{SO}(3)$:

$$f_{\mathrm{lin}}(\boldsymbol{V}R;\mathbf{W}) = \mathbf{W}\boldsymbol{V}R = f_{\mathrm{lin}}(\boldsymbol{V};\mathbf{W})R = \boldsymbol{V}'R$$

- $\mathbf{W}$- left multiplication, R- right multiplication
- Note the absence of a bias term

VN Non-Linearity

ReLU Non-Linearity

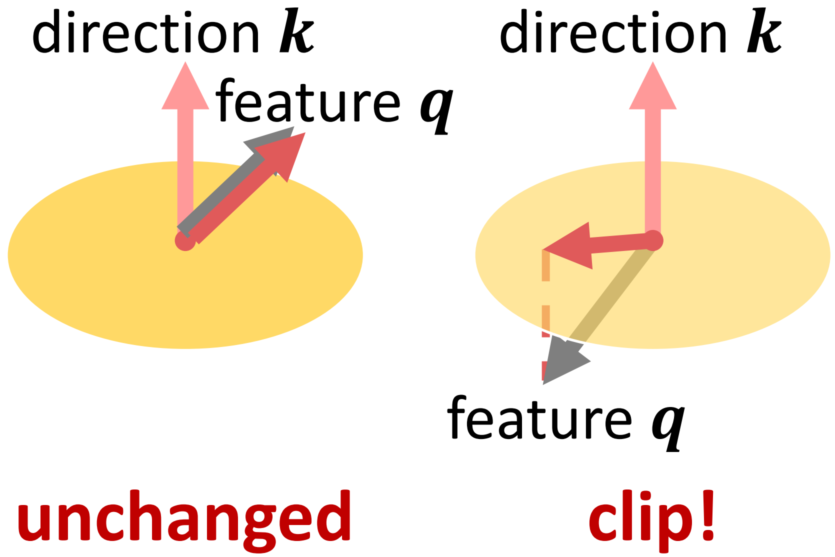
Weights $\mathbf{W} \in \mathbb{R}^{1\times C}$ and $\mathbf{U} \in \mathbb{R}^{1\times C}$

Learn a feature $\boldsymbol{q} = \mathbf{W}\boldsymbol{V} \in \mathbb{R}^{1\times 3}$

Learn a direction $\boldsymbol{k} = \mathbf{U}\boldsymbol{V} \in \mathbb{R}^{1\times 3}$

For each output vector neuron $\boldsymbol{v}' \in \boldsymbol{V}'$

$$\boldsymbol{v}' = \begin{cases} \boldsymbol{q} & \text{if } \langle \boldsymbol{q}, \boldsymbol{k} \rangle \geqslant 0 \\ \boldsymbol{q} - \langle \boldsymbol{q}, \frac{\boldsymbol{k}}{\|\boldsymbol{k}\|} \rangle \frac{\boldsymbol{k}}{\|\boldsymbol{k}\|} & \text{otherwise} \end{cases}$$



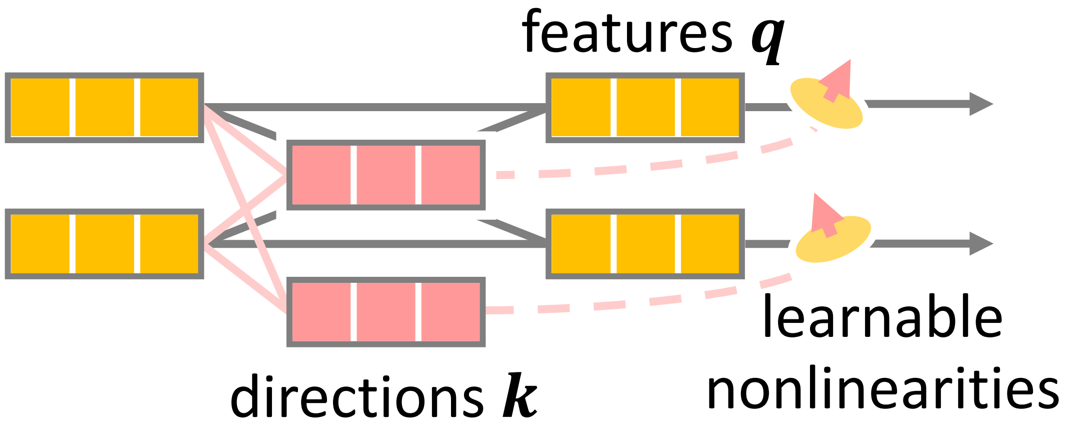
VN Non-Linearity

ReLU Non-Linearity

- **Non-linear layer** (with built-in linear layer)
  = **input linear transformation q + non-linearity k**

- A single non-linear layer detached from linear layer: $\boldsymbol{q} = \boldsymbol{v}$
  - introduce additional network depth!

- Other non-linearities



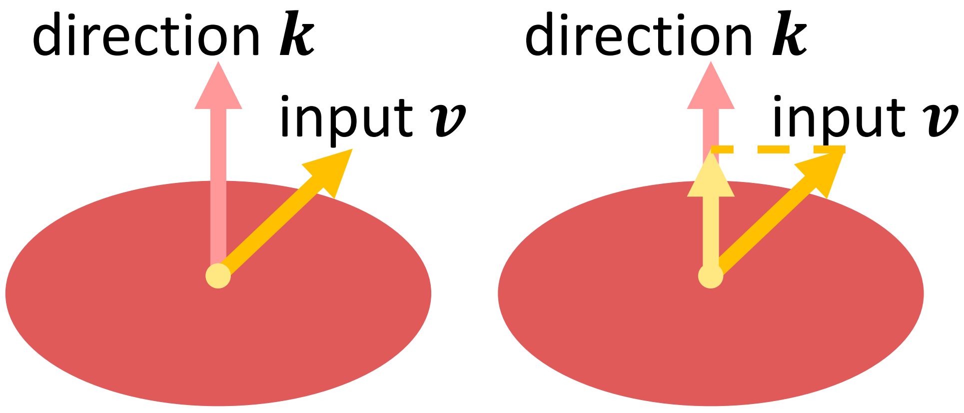
overall structure

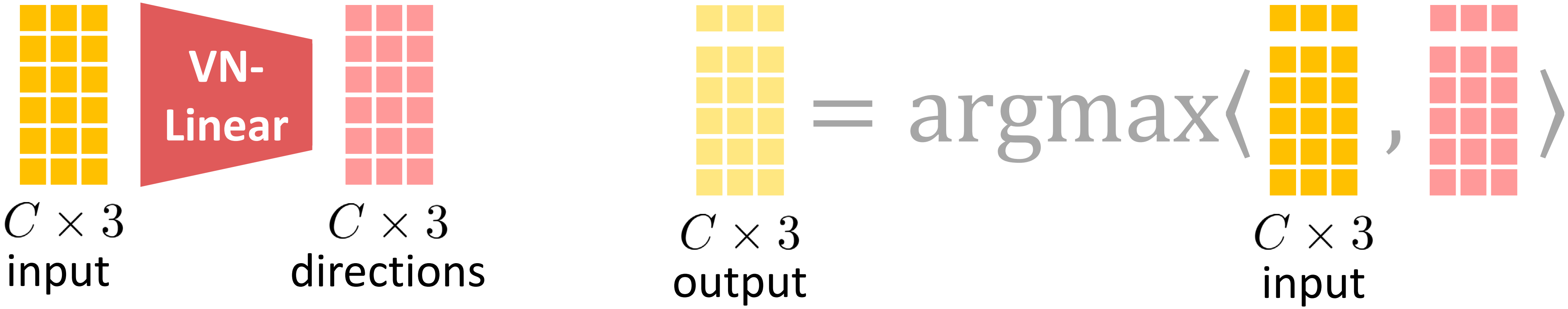
VN Pooling

✓ **Mean pooling**

? Max pooling

- (Similar to non-linearity)
- argmax alone learned directions





VN Pooling

✓**Mean pooling**

? Max pooling

- similar to non-linearities → use learnable directions

Given a set of vector-lists $\mathcal{V} = \{\boldsymbol{V}_1, \boldsymbol{V}_2, \cdots, \boldsymbol{V}_N\} \in \mathbb{R}^{N \times C \times 3}$

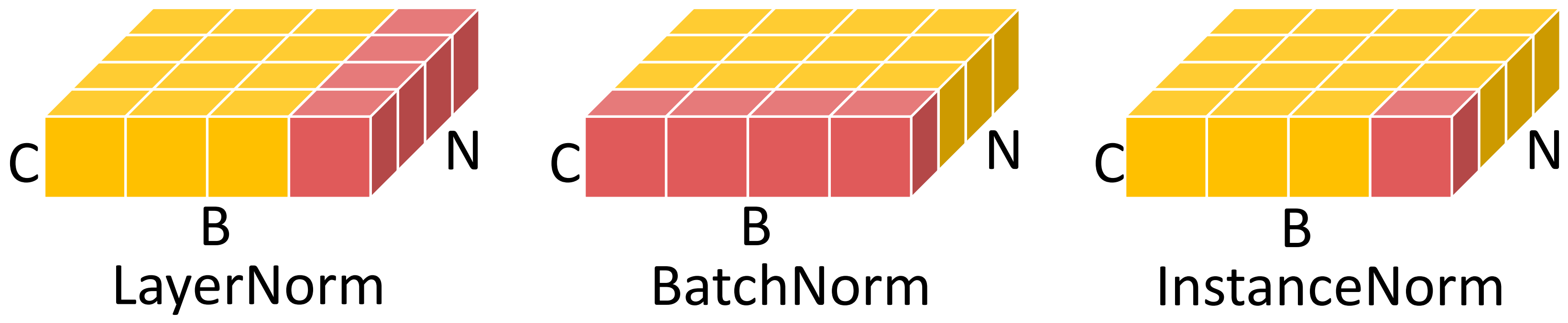
Learn data-dependent directions $\mathcal{K} = \{\mathbf{W}\boldsymbol{V}_n\}_{n=1}^{N} \in \mathbb{R}^{N \times C \times 3}$ with learnable weights $\mathbf{W} \in \mathbb{R}^{C \times C}$

Max pool along $\mathcal{K}$ directions: for each channel $c \in [C]$

$$f_{\mathrm{MAX}}(\mathcal{V})[c] = \boldsymbol{V}_{n^*}[c] \quad \text{where } n^* = \operatorname{argmax}\langle \mathbf{W}\boldsymbol{V}_n[c], \boldsymbol{V}_n[c] \rangle$$

VN Normalizations

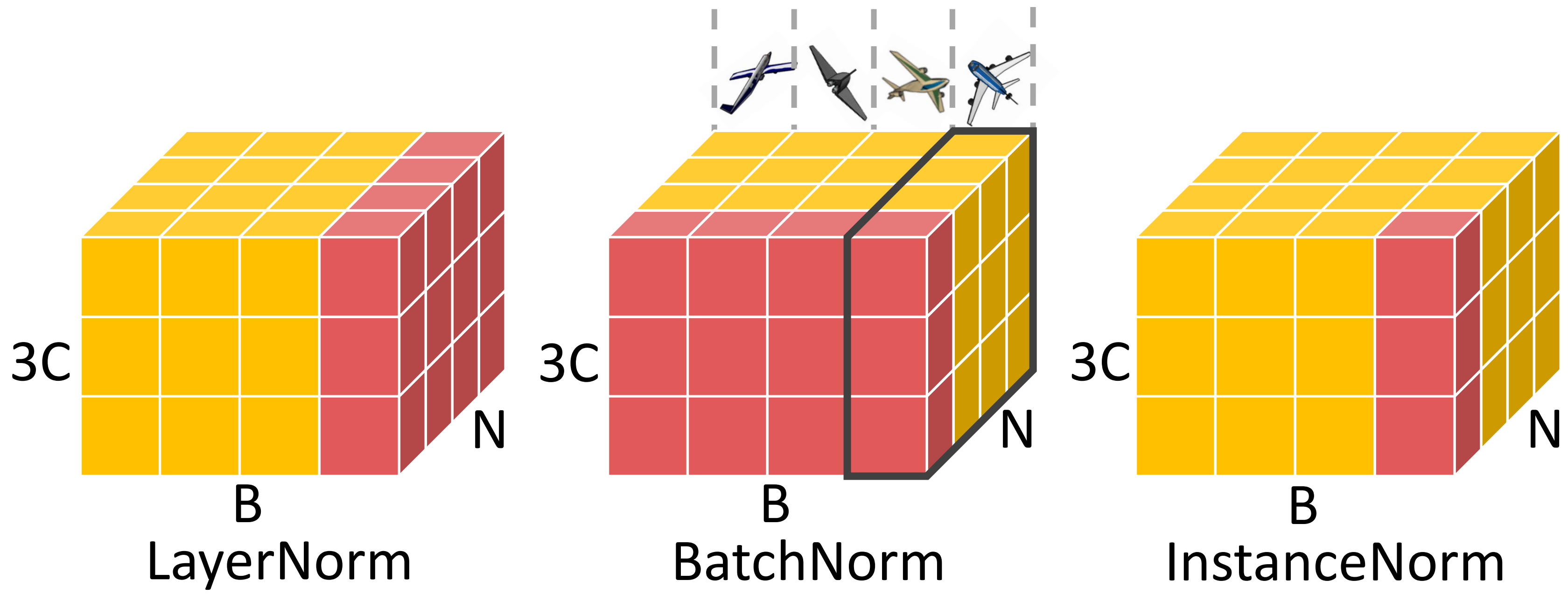
- ✓ **LayerNorm**
- ✓ **InstanceNorm**
- ✓ **Dropout**



(classical) scalar neurons

? BatchNorm

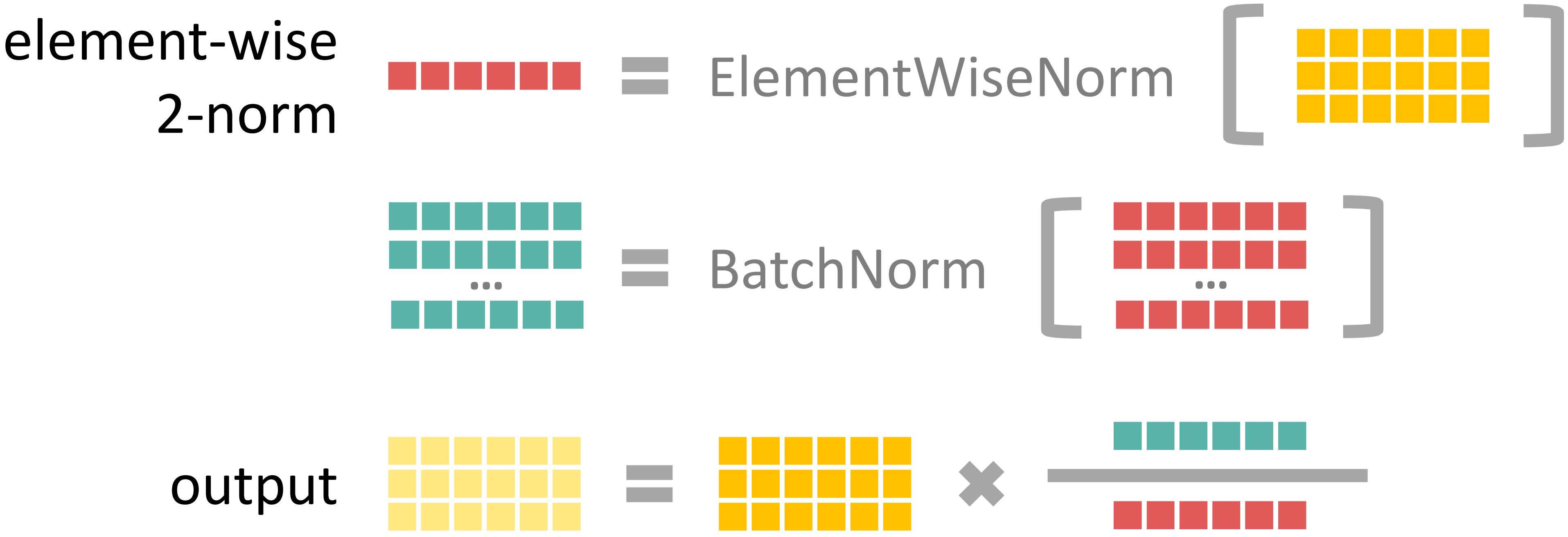
averaging across arbitrarily rotated inputs would not necessarily be meaningful



Vector neurons

VN Normalizations

BatchNorm



VN Normalizations

BatchNorm

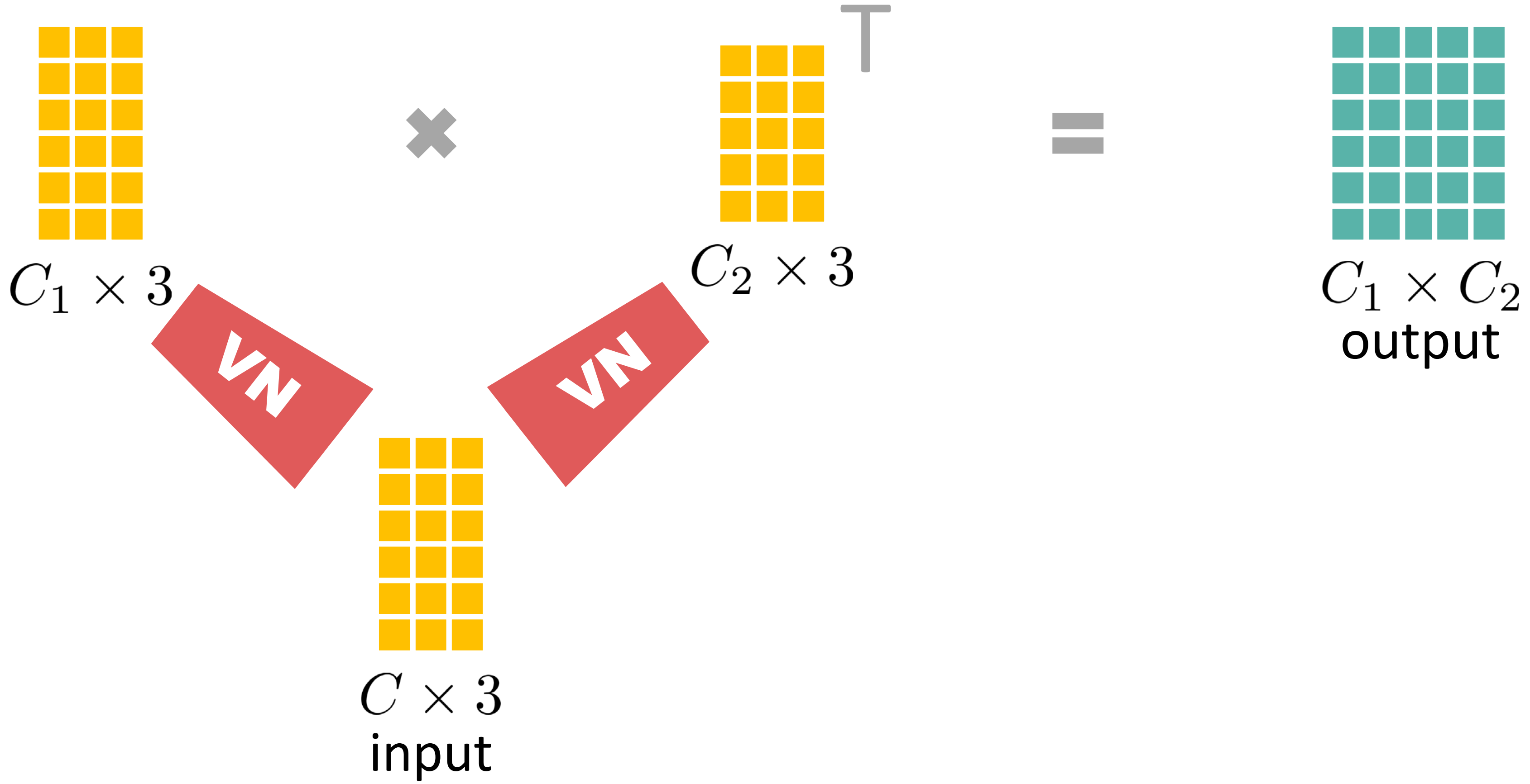
- Normalize the **2-norm (invariant component)** of the vector-list feature

$$\boldsymbol{N}_b = \mathrm{ElementWiseNorm}(\boldsymbol{V}_b) \in \mathbb{R}^{N\times 1}$$
$$\{\boldsymbol{N}'_b\}_{b=1}^{B} = \mathrm{BatchNorm}\left(\{\boldsymbol{N}_b\}_{b=1}^{B}\right)$$
$$\boldsymbol{V}'_b = \boldsymbol{V}_b[c]\frac{\boldsymbol{N}'_b[c]}{\boldsymbol{N}_b[c]} \quad \forall c \in [C]$$

- Element-wise norm: 2-norm for each vector $\boldsymbol{v}_c \in \boldsymbol{V}_b$

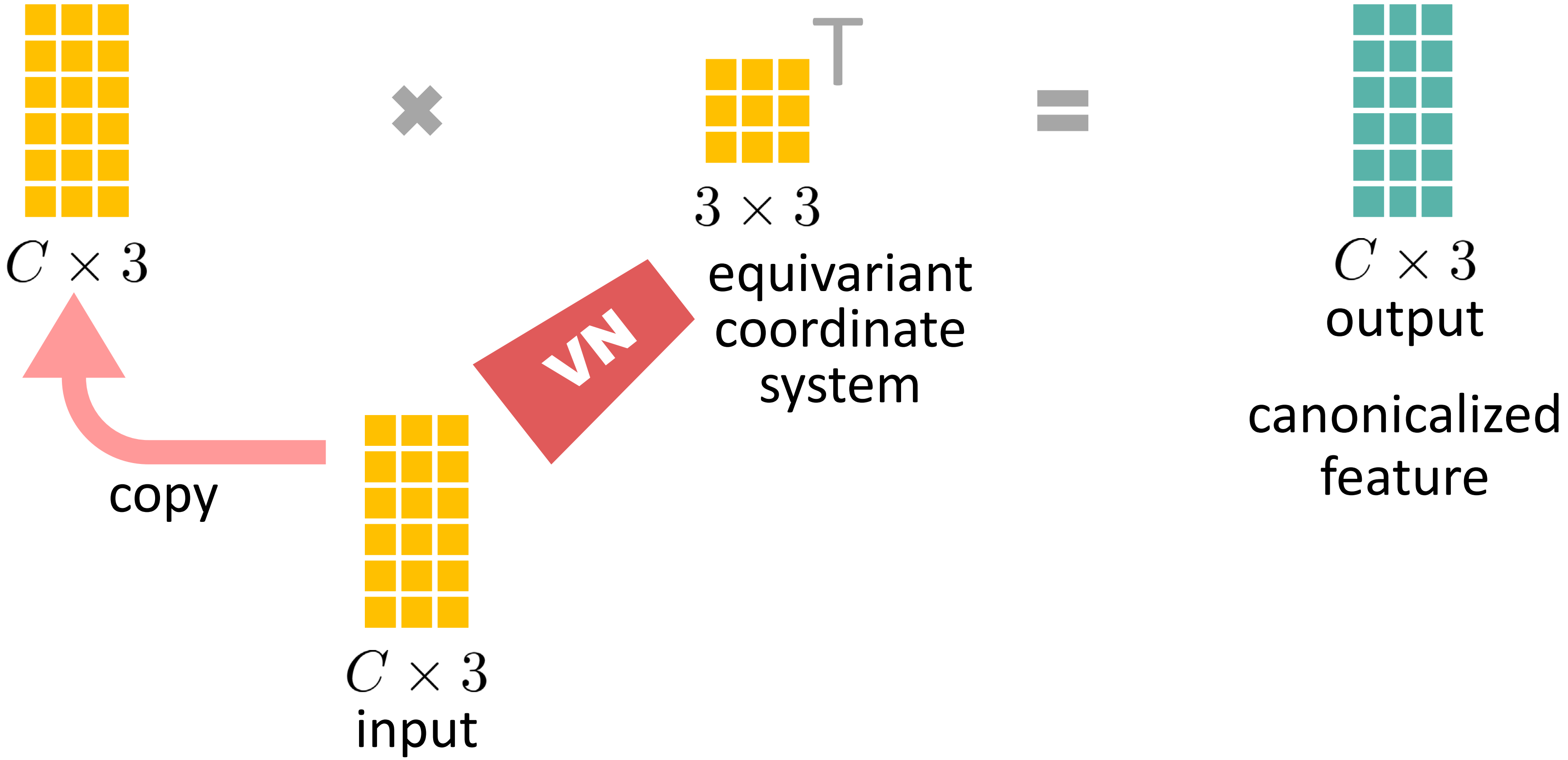
VN Invariant Layer

(equivariant feature) × (equivariant feature)$^{\top}$ = (invariant feature)



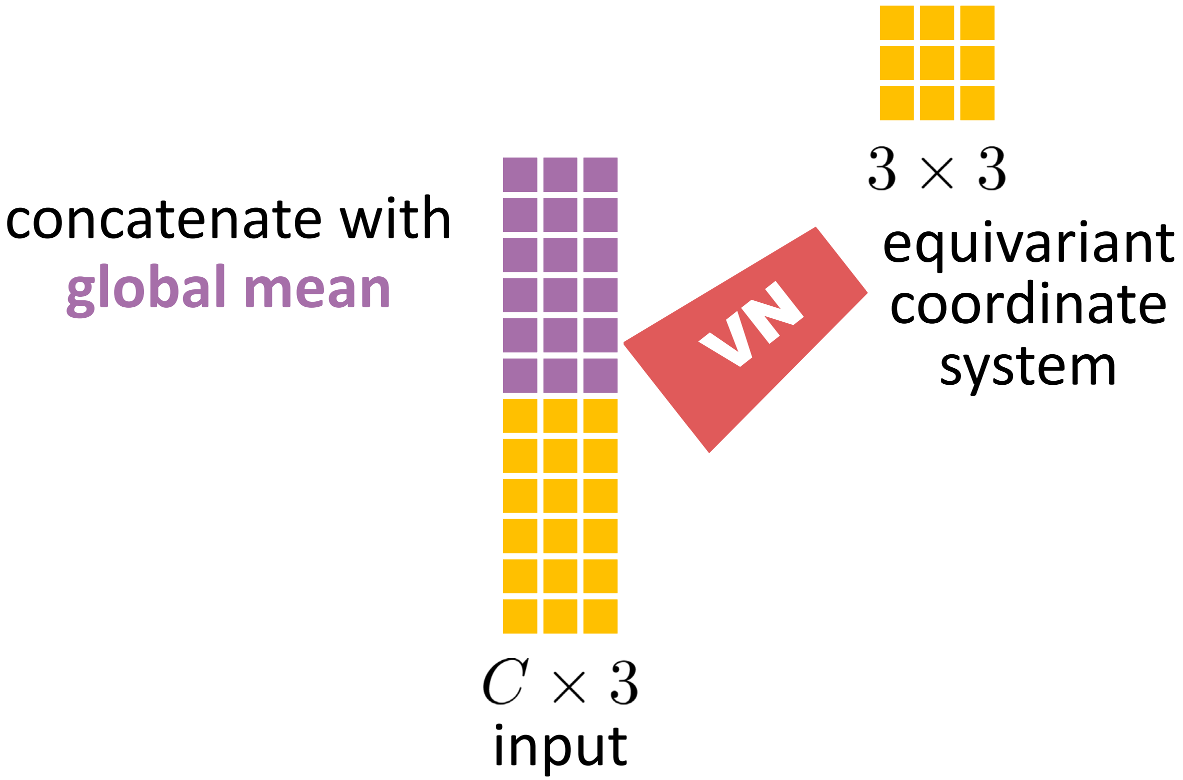
VN Invariant Layer

Specifically…



VN Invariant Layer

For pointcloud: combine **global information** with local features



VN Invariant Layer

- Product of an equivariant signal $\boldsymbol{V} \in \mathbb{R}^{C\times 3}$ by the transpose of another equivariant signal $\boldsymbol{T} \in \mathbb{R}^{C'\times 3}$ → invariant signal

- **Special case:** $\boldsymbol{T} \in \mathbb{R}^{3\times 3}$ - an equivariant coordinate system
- For pointcloud, concatenate local feature $\boldsymbol{V} \in \mathbb{R}^{C\times 3}$ with global mean $\overline{\boldsymbol{V}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{V}_n \in \mathbb{R}^{C\times 3}$

Invariant layer:

$$\boldsymbol{T}_n = \text{VN-MLP}([\boldsymbol{V}_n, \overline{\boldsymbol{V}}])$$

$$\text{VN-In}(\boldsymbol{V}_n) = \boldsymbol{V}_n \boldsymbol{T}_n^\top$$

Build VN Networks: VN-DGCNN

DGCNN

Edge feature: $${}'_{nm} = \mathrm{ReLU}(\Theta({}_m - {}_n) + \Phi\,{}_n)$$

Aggregation: $${}'_{n} = \mathrm{Pool}_{m:(n,m)\in\mathcal{E}}({}'_{nm})$$

VN-DGCNN

Edge feature: $${}'_{nm} = \text{VN-ReLU}(\Theta({}_m - {}_n) + \Phi\,{}_n)$$

Aggregation $${}'_{n} = \text{VN-Pool}_{m:(n,m)\in\mathcal{E}}({}'_{nm})$$

[Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, TOG 2019]

Build VN Networks: VN-DGCNN

Convolution in latent spaces, graph edges not embedded in $\mathbb{R}^3$

DGCNN: $\boldsymbol{x} \in \mathbb{R}^{C}, \boldsymbol{e}'_n, \boldsymbol{x}'_n \in \mathbb{R}^{C'}$ scalar features

Edge conv $\boldsymbol{e}'_{nm} = \mathrm{ReLU}(\Theta(\boldsymbol{x}_m - \boldsymbol{x}_n) + \Phi \boldsymbol{x}_n)$

Aggregation $\boldsymbol{x}'_n = \mathrm{Pool}_{m:(n,m)\in\mathcal{E}}(\boldsymbol{e}'_{nm})$

VN-DGCNN: $\boldsymbol{V} \in \mathbb{R}^{C\times 3}, \boldsymbol{E}'_n, \boldsymbol{V}'_n \in \mathbb{R}^{C'\times 3}$ vector-list features

Edge conv $\boldsymbol{E}'_{nm} = \text{VN-ReLU}(\Theta(\boldsymbol{V}_m - \boldsymbol{V}_n) + \Phi \boldsymbol{V}_n)$

Aggregation $\boldsymbol{V}'_n = \text{VN-Pool}_{m:(n,m)\in\mathcal{E}}(\boldsymbol{E}'_{nm})$

[Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, TOG 2019]

Build VN Networks: VN-PointNet

PointNet

$$' = \mathrm{Pool}_{\boldsymbol{x}_n \in \mathcal{X}}\left(h(\ _1), h(\ _2) \cdots, h(\ _N)\right)$$

VN-PointNet

$$' = \text{VN-Pool}_{\boldsymbol{V}_n \in \mathcal{V}}\left(f(\ _1), f(\ _2), \cdots, f(\ _N)\right)$$

[C. R. Qi, H. Su, K. Mo, L. J. Guibas, CVPR 2017]

Build VN Networks: VN-PointNet

No convolutions, only point-wise feature transformations

PointNet: $$\boldsymbol{x}' = \mathrm{Pool}_{\boldsymbol{x}_n \in \mathcal{X}}(h(\boldsymbol{x}_1), \cdots, h(\boldsymbol{x}_N))$$

VN-PointNet: $$\boldsymbol{V}' = \text{VN-Pool}_{\boldsymbol{V}_n \in \mathcal{V}}(f(\boldsymbol{V}_1), \cdots, f(\boldsymbol{V}_N))$$

- h MLP, f VN-MLP
- First layer single channel → edge conv to lift dimensionality
  - “Cannot apply per-channel transformation to gray-scale images”

[C. R. Qi, H. Su, K. Mo, L. J. Guibas, CVPR 2017]

Classification

Classification results on ModelNet40

VN Networks

Rotation sensitive methods

Rotation robust methods

Methods	z/z	$z/\mathrm{SO}(3)$	$\mathrm{SO}(3)/\mathrm{SO}(3)$
Point / mesh inputs			
PointNet [25]	85.9	19.6	74.7
DGCNN [35]	90.3	33.8	88.6
VN-PointNet	77.5	77.5	77.2
VN-DGCNN	89.5	**89.5**	**90.2**
PCNN [2]	92.3	11.9	85.1
ShellNet [40]	**93.1**	19.9	87.8
PointNet++ [26]	91.8	28.4	85.0
PointCNN [20]	92.5	41.2	84.5
Spherical-CNN [11]	88.9	76.7	86.9
a^3S-CNN [21]	89.6	87.9	88.7
SFCNN [27]	91.4	84.8	90.1
TFN [32]	88.5	85.3	87.6
RI-Conv [39]	86.5	86.4	86.4
SPHNet [24]	87.7	86.6	87.6
ClusterNet [6]	87.1	87.1	87.1
GC-Conv [41]	89.0	89.1	89.2
RI-Framework [18]	89.4	89.4	89.3

Classification

Classification results on ModelNet40

- VN networks are **robust to (seen & unseen) rotations**

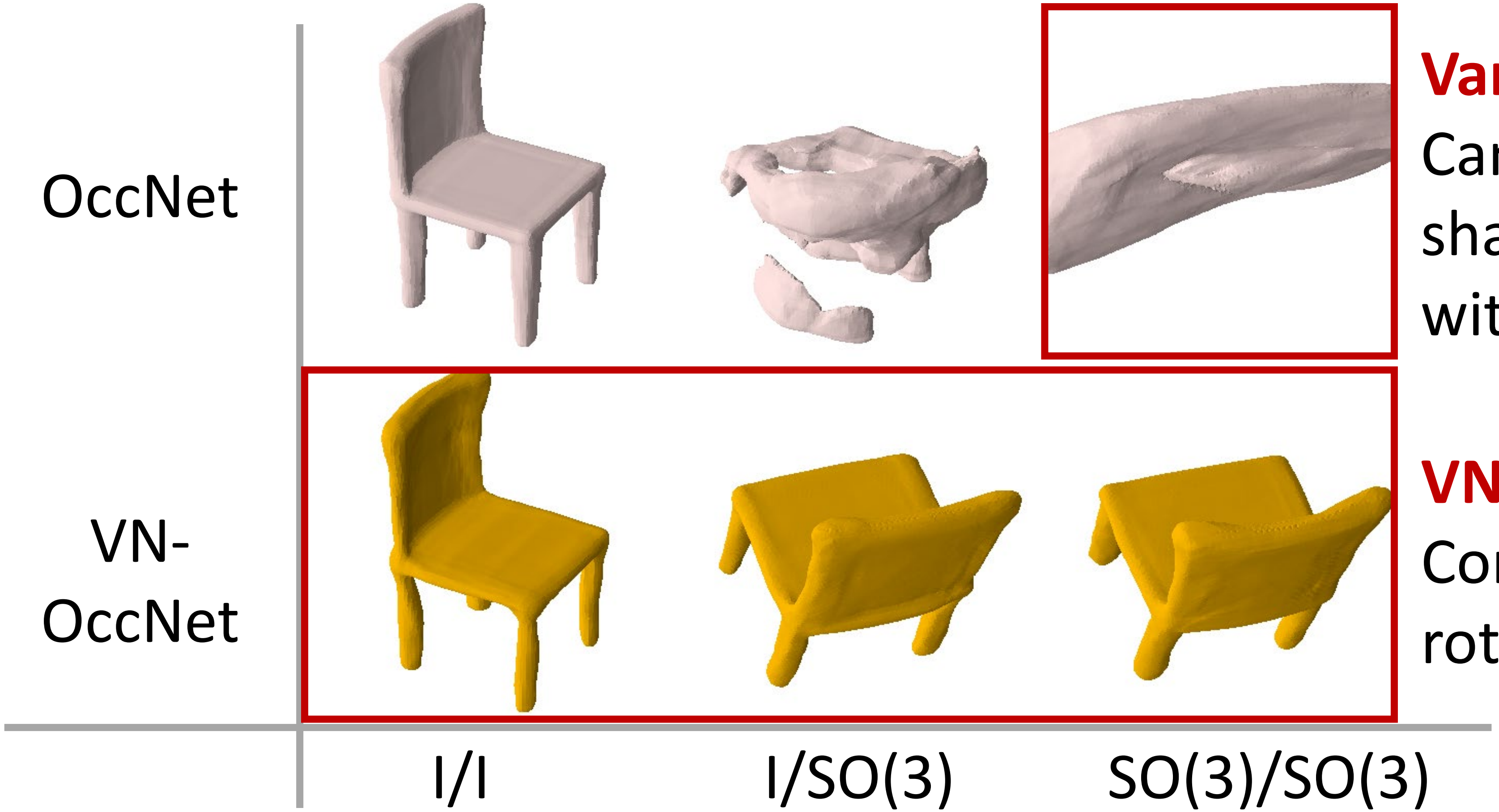
- Excellent performance compared with other methods

- **SO(3)/SO(3):** equivariance by construction is better than rotation augmentation

Methods	z/z	$z/\mathrm{SO}(3)$	$\mathrm{SO}(3)/\mathrm{SO}(3)$
Point / mesh inputs			
PointNet [25]	85.9	19.6	74.7
DGCNN [35]	90.3	33.8	88.6
VN-PointNet	77.5	77.5	77.2
VN-DGCNN	89.5	**89.5**	**90.2**
PCNN [2]	92.3	11.9	85.1
ShellNet [40]	**93.1**	19.9	87.8
PointNet++ [26]	91.8	28.4	85.0
PointCNN [20]	92.5	41.2	84.5
Spherical-CNN [11]	88.9	76.7	86.9
a^3S-CNN [21]	89.6	87.9	88.7
SFCNN [27]	91.4	84.8	90.1
TFN [32]	88.5	85.3	87.6
RI-Conv [39]	86.5	86.4	86.4
SPHNet [24]	87.7	86.6	87.6
ClusterNet [6]	87.1	87.1	87.1
GC-Conv [41]	89.0	89.1	89.2
RI-Framework [18]	89.4	89.4	89.3

Neural Implicit Reconstruction

Results on ShapeNet (Examples)

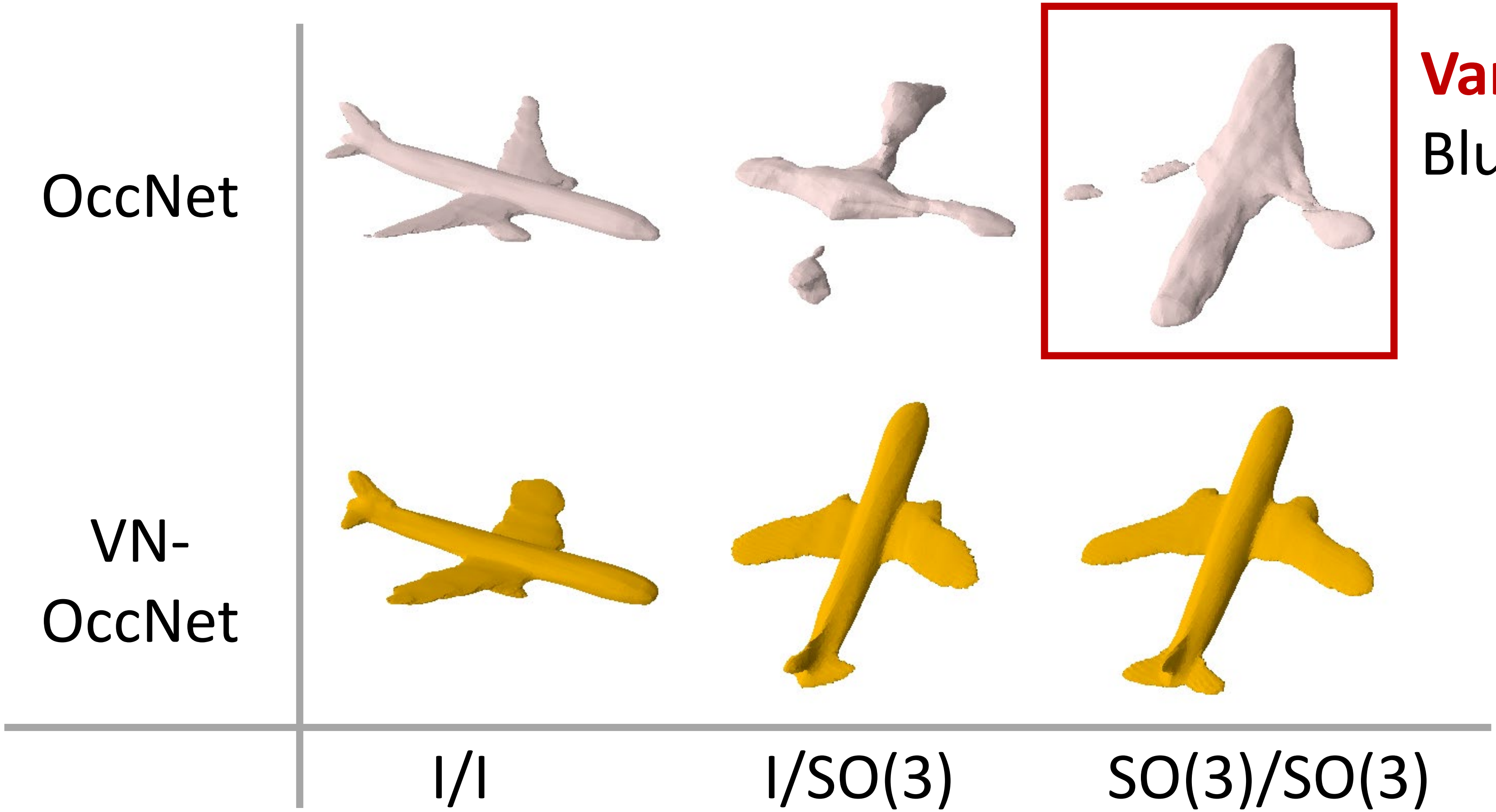


Vanilla OccNet:
Can’t learn rotated shapes even trained with augmentation

VN-OccNet:
Consistent across rotations

Neural Implicit Reconstruction

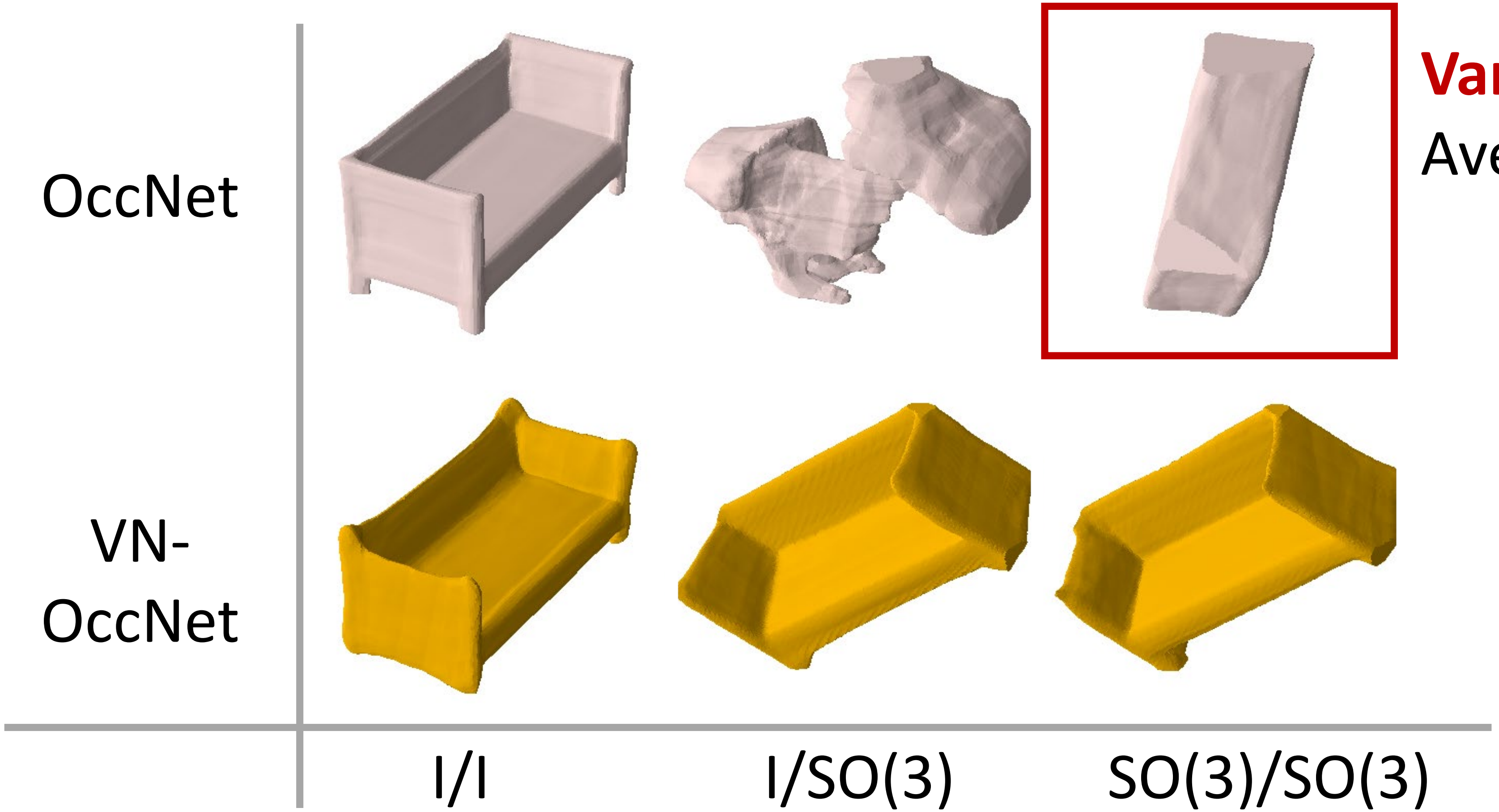
Results on ShapeNet (Examples)



Vanilla OccNet:
Blurry shapes

Neural Implicit Reconstruction

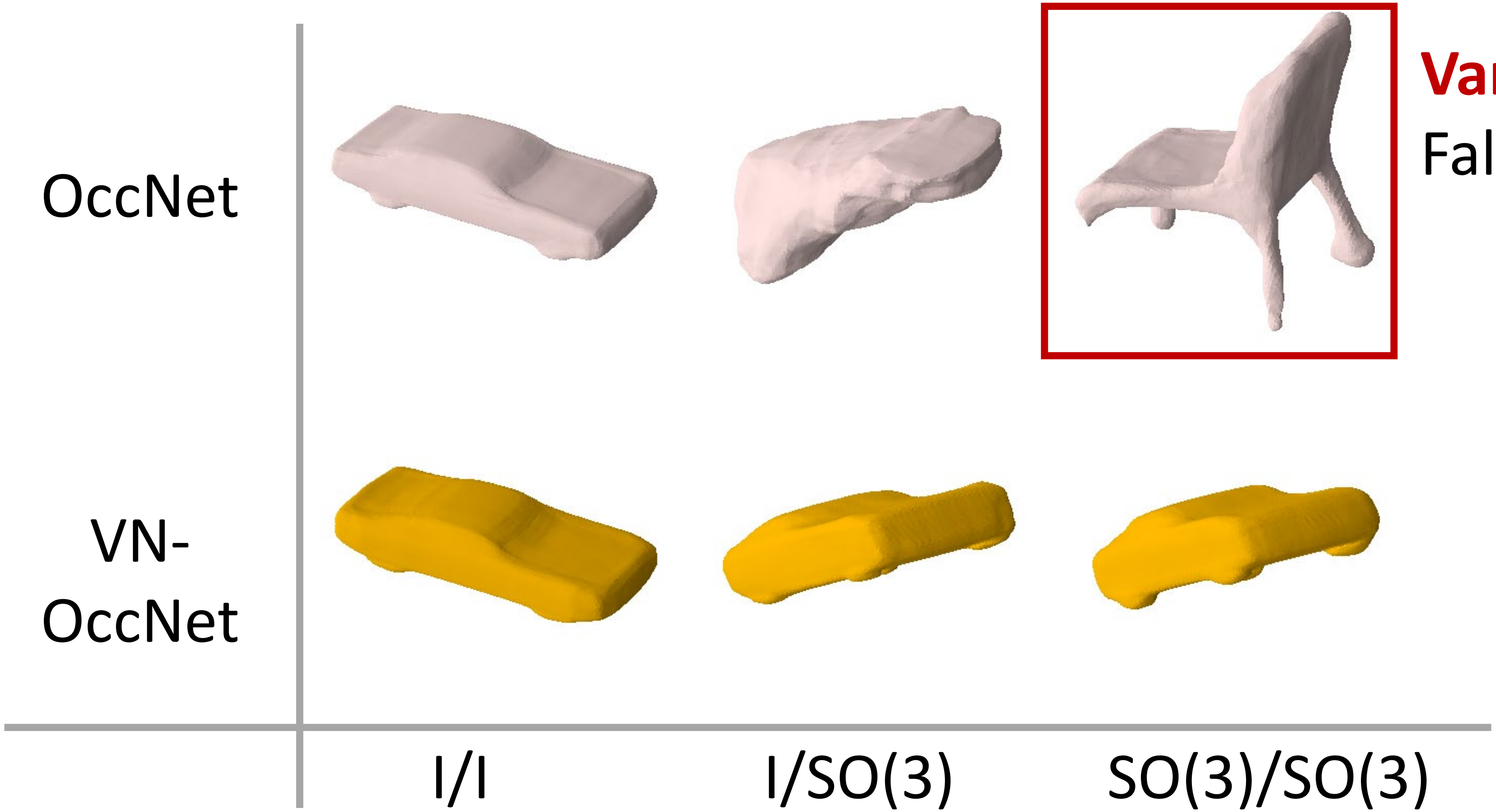
Results on ShapeNet (Examples)



Vanilla OccNet:
Averaged shapes

Neural Implicit Reconstruction

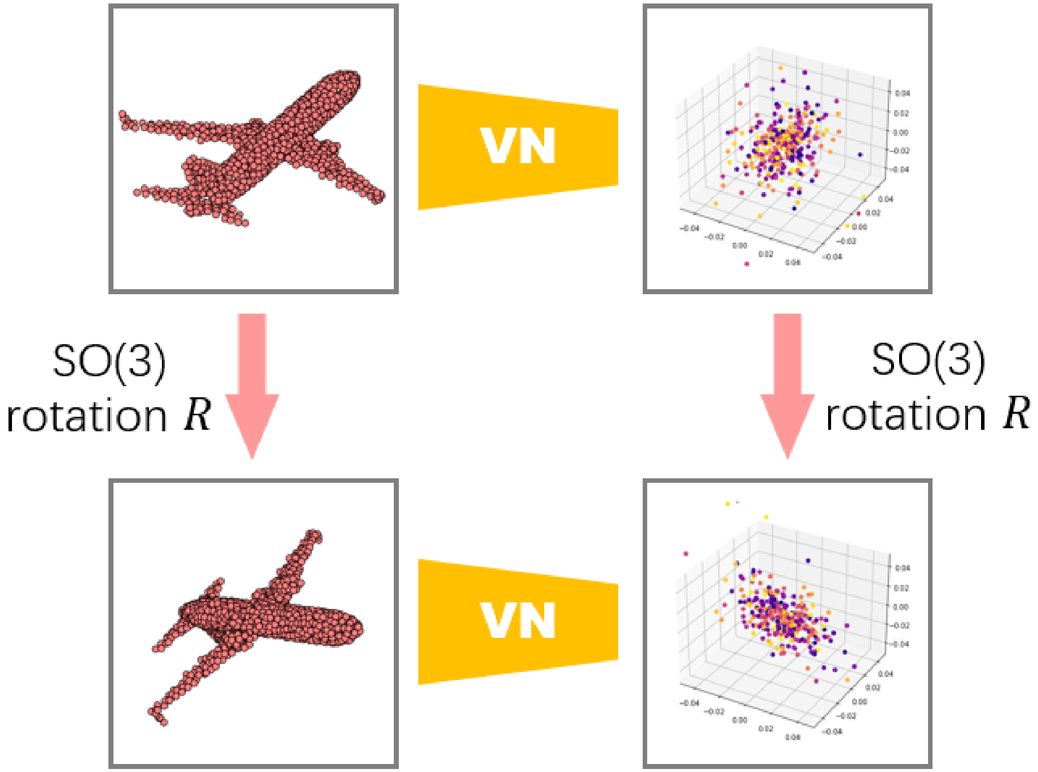
Results on ShapeNet (Examples)



Vanilla OccNet:
False shape priors

Summary: Vector Neurons

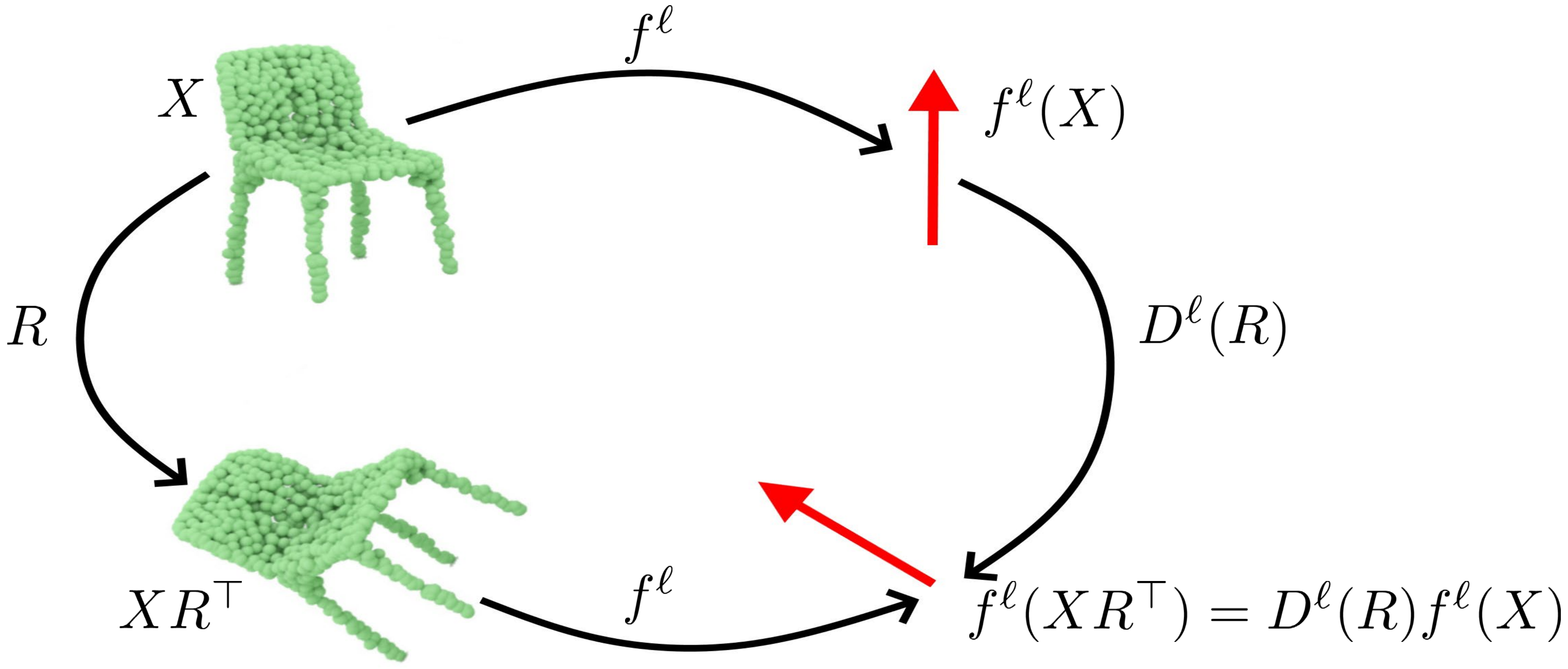
- **Vector Neurons:**
  Lift latent features to 3D vector lists
- **Building blocks:**
  - Linear layer
  - Non-linearity (ReLU)
  - Pooling (MaxPool)
  - Normalizations (BatchNorm)
  - Invariance
- **Network examples:**
  - VN-DGCNN
  - VN-PointNet



Tensor Field Networks and Equivariant CNNs

SO(3) Equivariant Features

- A type $\ell \in \mathbb{N}$ equivariant feature map $f^\ell : \mathbb{R}^{n\times 3} \to \mathbb{R}^{2\ell+1}$ satsfying $f^\ell(XR^\top) = D^\ell(R) f^\ell(X)$ for $R \in \mathrm{SO}(3)$, where $D^\ell(R) \in \mathrm{SO}(2\ell+1)$ is the type ℓ wigner matrix.



Tensor Field Networks (TFN)

- Given a pointcloud X a TFN can produce pointwise or global equivariant features of different types ($\ell \in \mathbb{N}$):

Pointwise features

$$\underbrace{f^{\ell}(X)_{i,c,:}}$$

$$f^{\ell}(X)_{\underset{\text{point}}{i},\underset{\text{channel}}{c},:}$$

global features

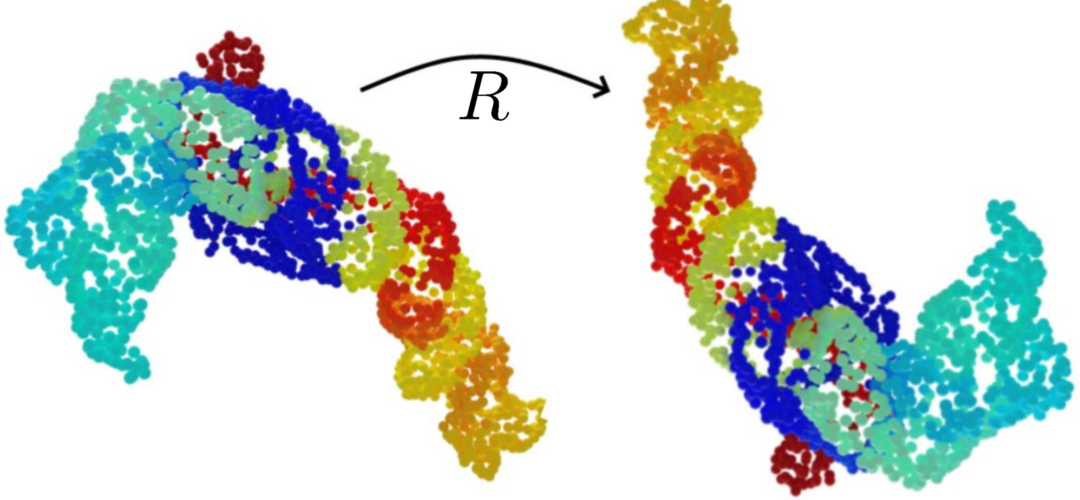
$$g^{\ell}(X)_{\underset{\text{channel}}{c},:}$$

Examples of Type 0 Features

- Type 0 features are rotation invariant as $D^0(R) = 1$:

Segmentation

$$f^0(X)_{i,c,:} = f^0(XR^\top)$$



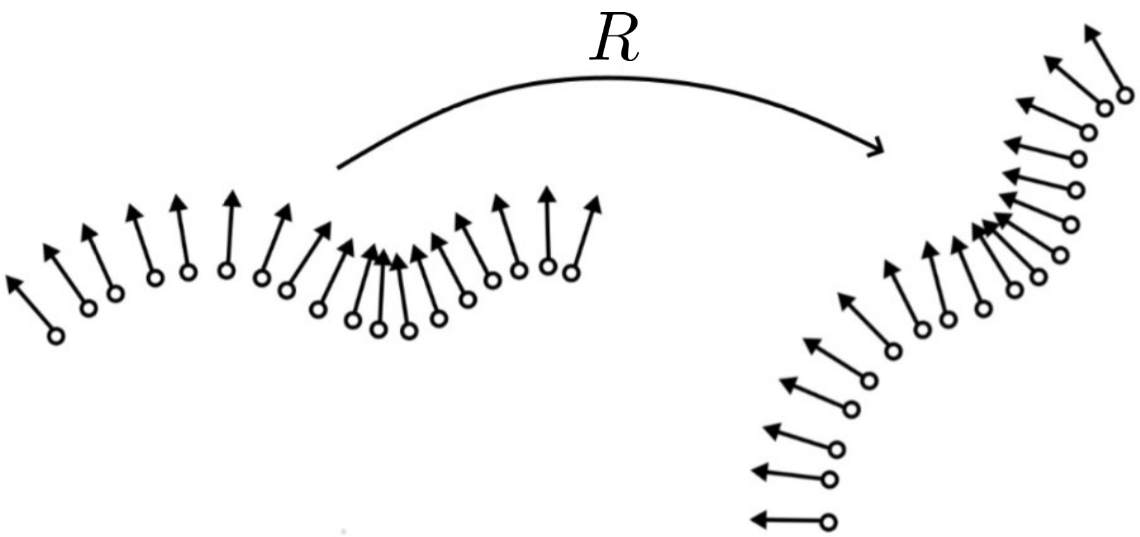
Classification

$g^0($  $) = \text{Chair}$

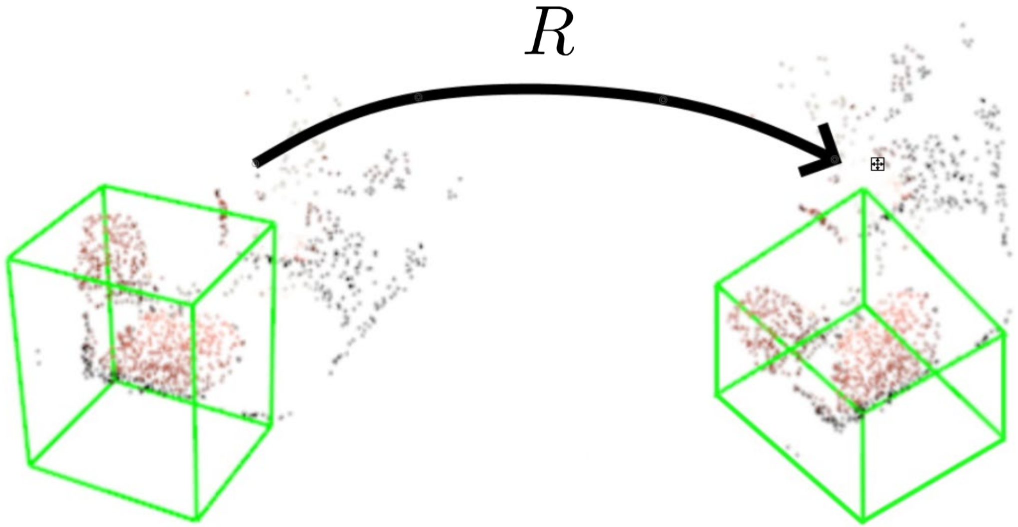
$g^0($  $) = \text{Chair}$

Examples of type 1 Features

- We have $D^1(R) = R$, thefore type 1 features are 3D vectors rotating with the pointcloud X.



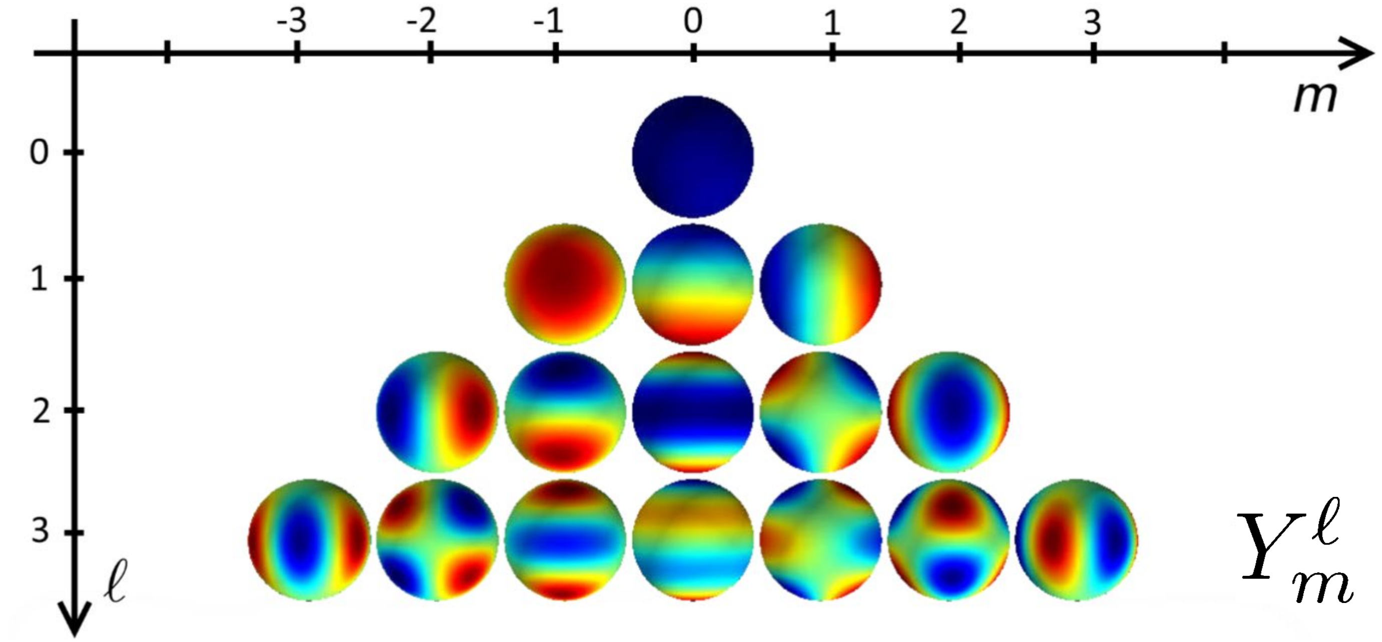
Pointcloud normals are type 1 features.



Bounding box center and principal directions are type 1 features, lengths are type 0 features

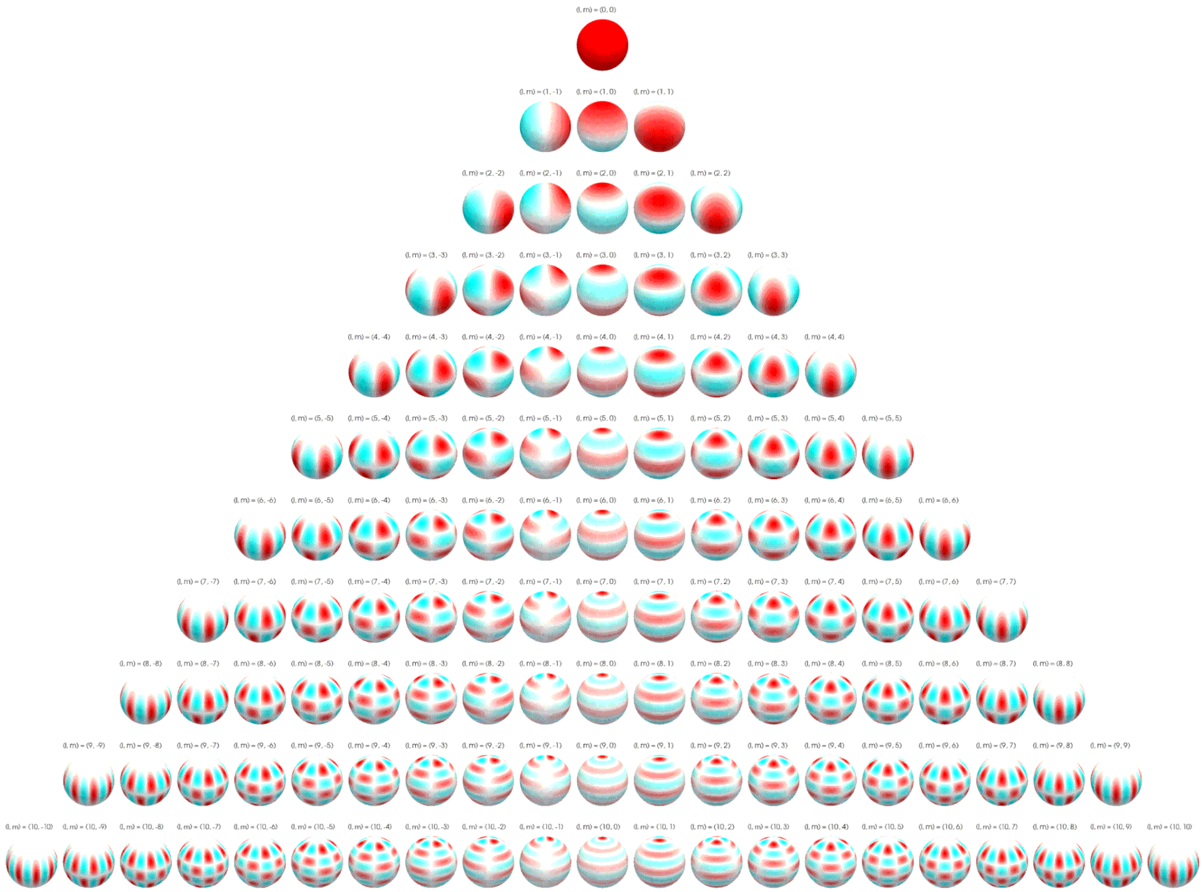
Spherical Harmonics & Higher Degree Features

- Spherical harmonics are homogeneous polynomials on $\mathbb{R}^3$, their restriction to $\mathcal{S}_2$ form an orthonormal basis of $L^2(\mathcal{S}_2)$.
- Just like type ℓ equivariant features the vector of degree ℓ spherical harmonics $Y^\ell(x) \in \mathbb{R}^{2\ell+1}$, satisfies $Y^\ell(Rx) = D^\ell(R)Y^\ell(x)$.



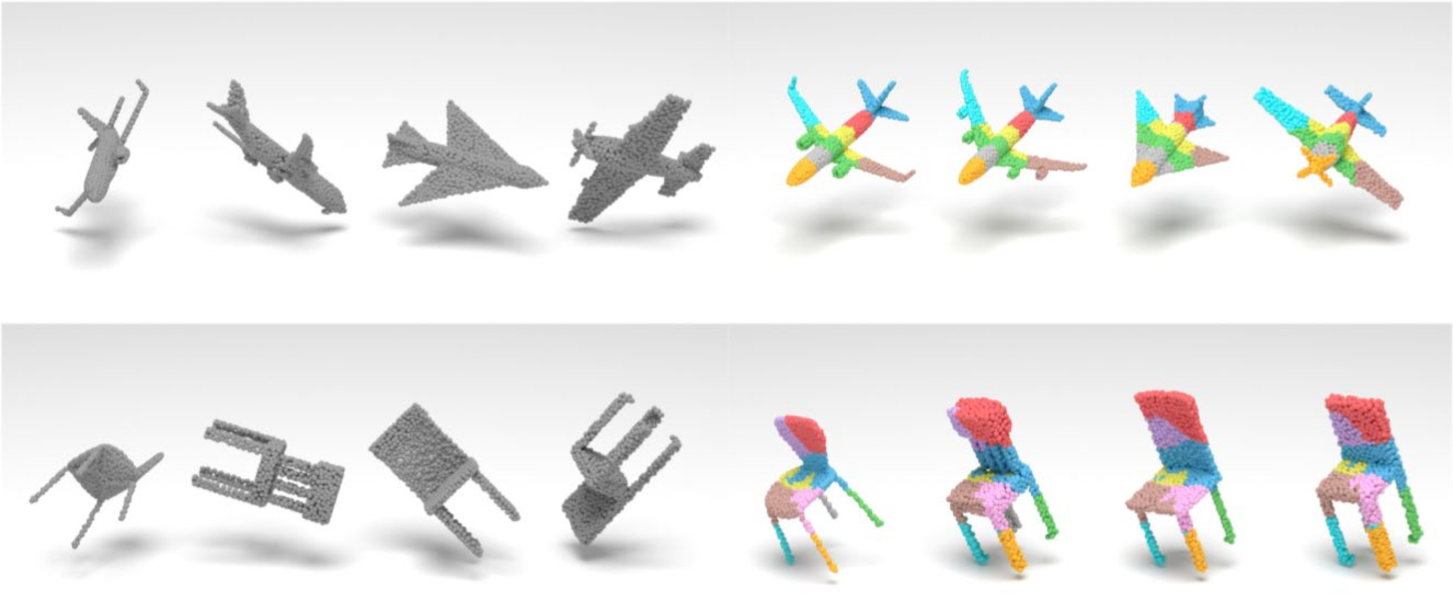
Spherical Harmonics & Higher Degree Features

- Spherical harmonics rotating arround the z axis:



SH & TFN Feature Applications (ConDor)

- Dot products of type ℓ features are invariant.
- We can view TFN features as coefficients of functions in the SH basis.
- The dot prods $\langle f^{\ell}(X)_{ic}, Y^{\ell}(X_i) \rangle$ give an invariant embedding of X.



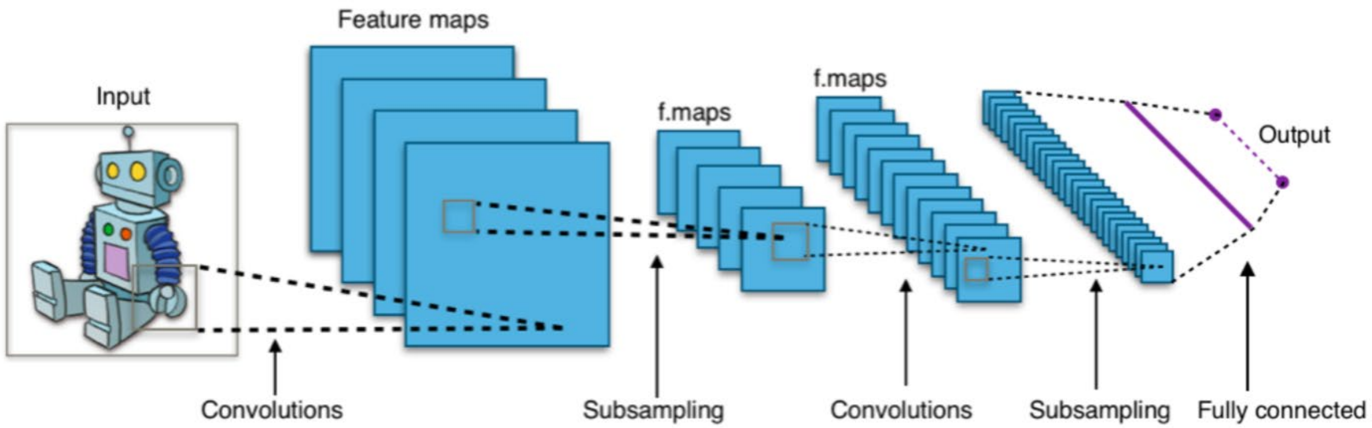
- Invariant embeddings can be used to:
  1. canonicalize shape pose.
  2. segment the shape

(see ConDor).

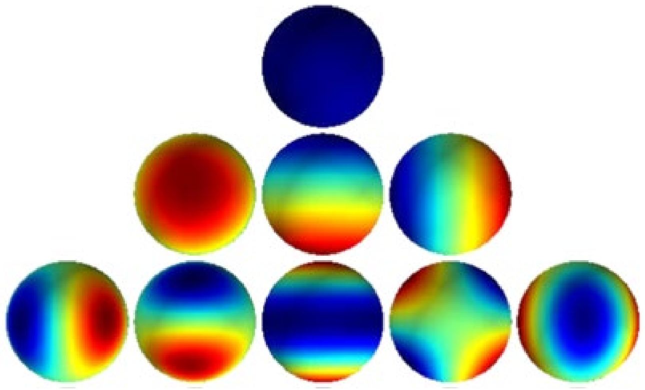
How does TFN work ?

- TFN is a convolutional architecture.
- It inherits its equivariance properties from SH kernels.

$$\mathrm{TFN} =$$



$$+$$



Convolutions on Different Domains

- The convolution between a function $f : \mathbb{R}^d \to \mathbb{R}$ and a kernel $\kappa : \mathbb{R}^d \to \mathbb{R}$ is defined by:

$$f * \kappa(x) := \int_{\mathbb{R}^d} f(t)\kappa(t - x)dt.$$

- Similarly the convolution a function f over a pointcloud X and a kernel $\kappa : \mathbb{R}^d \to \mathbb{R}$ is defined by:

$$f *_X \kappa(X_i) := \sum_j f_j \kappa(X_j - X_i).$$

- We can view images as functions over the pointcloud $\mathbb{Z}^d$ $(d = 2, 3)$.

Convolutional Layers in Neural Nets

- Typically, we are given a collection of functions $(f_i)_i$ on a pointcloud X and a collection of kernel filters $(\kappa_j)_j$ and a weight tensor W_{kij}.
- A convolutional layer outputs a collection of functions by taking linear combinations of convolutions:

$$\mathrm{Conv}_X(f, \kappa, W)_k := \sum_{ij} W_{kij} f_i *_X \kappa_j.$$

Convolutions and Rotations

- For any rotation R we have:

$$f *_{XR^\top} \kappa_j(X_i) = \sum_k f_k \kappa_j(R(X_k - X_i)).$$

- Problem what is the relation between $(f *_X \kappa_j)_j$ and $(f *_{XR^\top} \kappa_j)_j$?

Steerable Kernel Basis

- A steerable kernel basis, is a kernel basis $(\kappa_k)_k$ such that, the rotation of any kernel κ_j linearly decomposes onto the basis by a rotation matrix $D(R)$:

$$\kappa_j(Rx) = \sum_k D(R)_{jk}\kappa_k(x).$$

- Using a steerable basis we can relate convolutions on X and on $XR^\top$ by a simple linear relation:

$$f *_{XR^\top} \kappa_j(x) = \sum_j D(R)_{jk} f *_X \kappa_k(x)$$

3D Steerable Basis

- For each ℓ we have a sterrable basis of the form:

$$\kappa^{\ell}_{rm}(x) := \varphi_r(\|x\|_2) Y^{\ell}_m\left(\frac{x}{\|x\|_2}\right)$$

- Where φ_r is any radial function e.g. a gaussian shell:

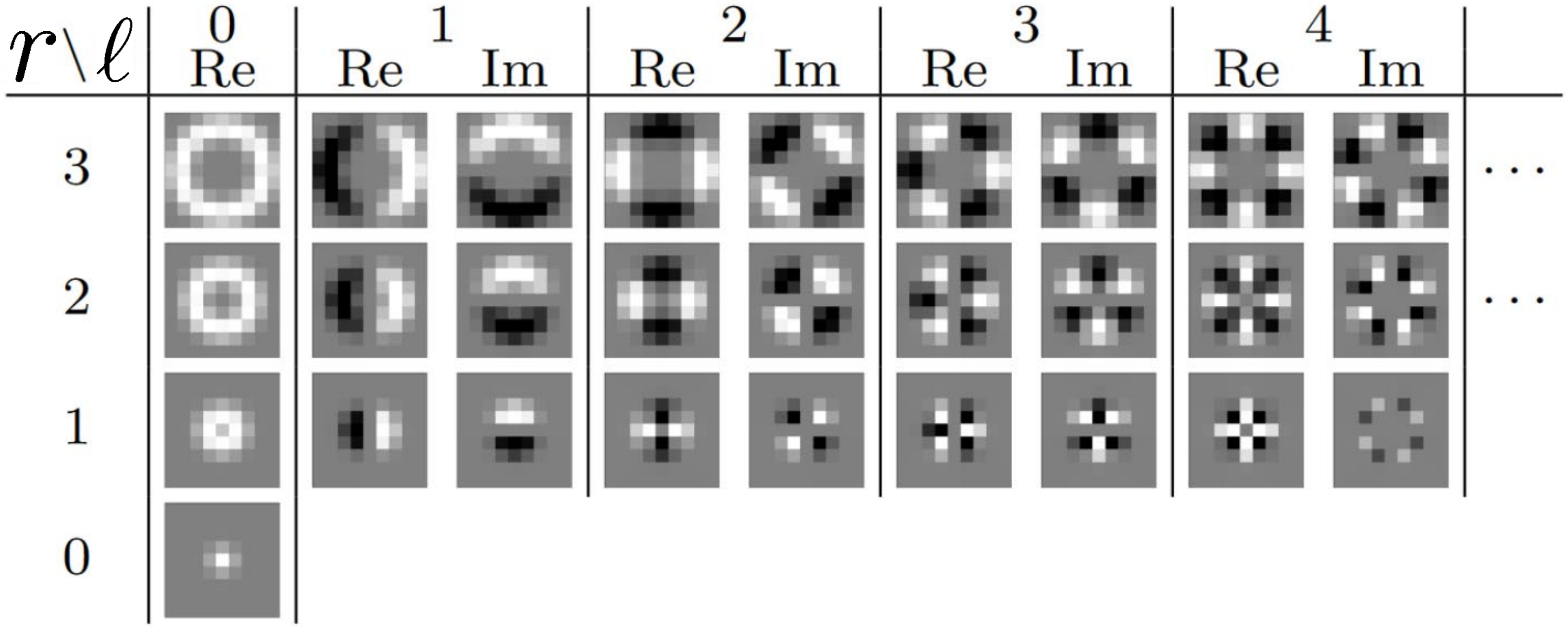
$$\varphi_r(y) := \exp\left(\frac{-(y-\rho_r)^2}{2\sigma^2}\right)$$

Example of Steerable Basis (2D)

- For each ℓ in $\mathbb{Z}$ we can build a steerable kernel basis on $\mathbb{R}^2$ using polar coordinates:

$$\kappa_r^\ell(\theta, \rho) := \varphi_r(\rho) e^{i\ell\theta}$$

  here we have $D^\ell(t) = e^{i\ell t}$.



Example of steerable kernels, (source Learning Steerable Filters with Rotation Equivariant CNNs)

Steerable Convolution

- We consider a collection of steerable kernel bases $(\kappa_r^\ell)_{\ell r}$ over $\mathbb{R}^d$ indexed by an equivariance type ℓ such that for any rotation $R \in \mathrm{SO}(d)$ we have:

$$\kappa^\ell(Rx) = D^\ell(R)\kappa^\ell(x)$$

- For each ℓ we define the associated steerable convolution operator:

$$\mathrm{SConv}_X^\ell(f, \kappa_r^\ell, W)_{km} := \sum_{ir} W_{kir} f_i *_X \kappa_{rm}^\ell.$$

Comparing Regular and Steerable Convolution

- Regular convolution takes linear combination over all indices of the kernel basis:

$$\mathrm{Conv}_X(f, \kappa, W)_k := \sum_{ij} W_{kij} f_i *_X \kappa_j.$$

- Steerable convolution does not take linear combination over the "steer-able" index m in order to preserve equivariance:

$$\mathrm{SConv}^{\ell}_X(f, \kappa^{\ell}_r, W)_{km} := \sum_{ir} W_{kir} f_i *_X \kappa^{\ell}_{rm}.$$

Steerable Convolution is Equivariant

- A rotation R of the pointcloud X induces a linear transform $D^\ell(R)$ which is independent of X:

$$\begin{aligned}
\mathrm{SConv}^\ell_{XR^\top}(f, \kappa^\ell_r, W)_k &= \sum_{ir} W_{kir} f_i *_{XR^\top} \kappa^\ell_r \\
&= \sum_{ir} W_{kir} f_i *_X D^\ell(R)\kappa^\ell_r \\
&= D^\ell(R)\mathrm{SConv}^\ell_X(f, \kappa^\ell_r, W)_k
\end{aligned}$$

Why not Mix Different Types?

- In 2D a type ℓ features makes ℓ turns when the input makes one turn:

$$v(e^{i\theta}X) = e^{i\ell\theta}v(X).$$

- There is no single rotation that rotates different types, or a linear combination of different types.

Stacking Steerable Convolutions

- We have described steerable convolution of rotation invariant pointwise features with a steerable basis.
- TFN extends steerable convolution to equivariant inputs, by linearly decomposing the resulting convolutional features into equivariant features of different types.

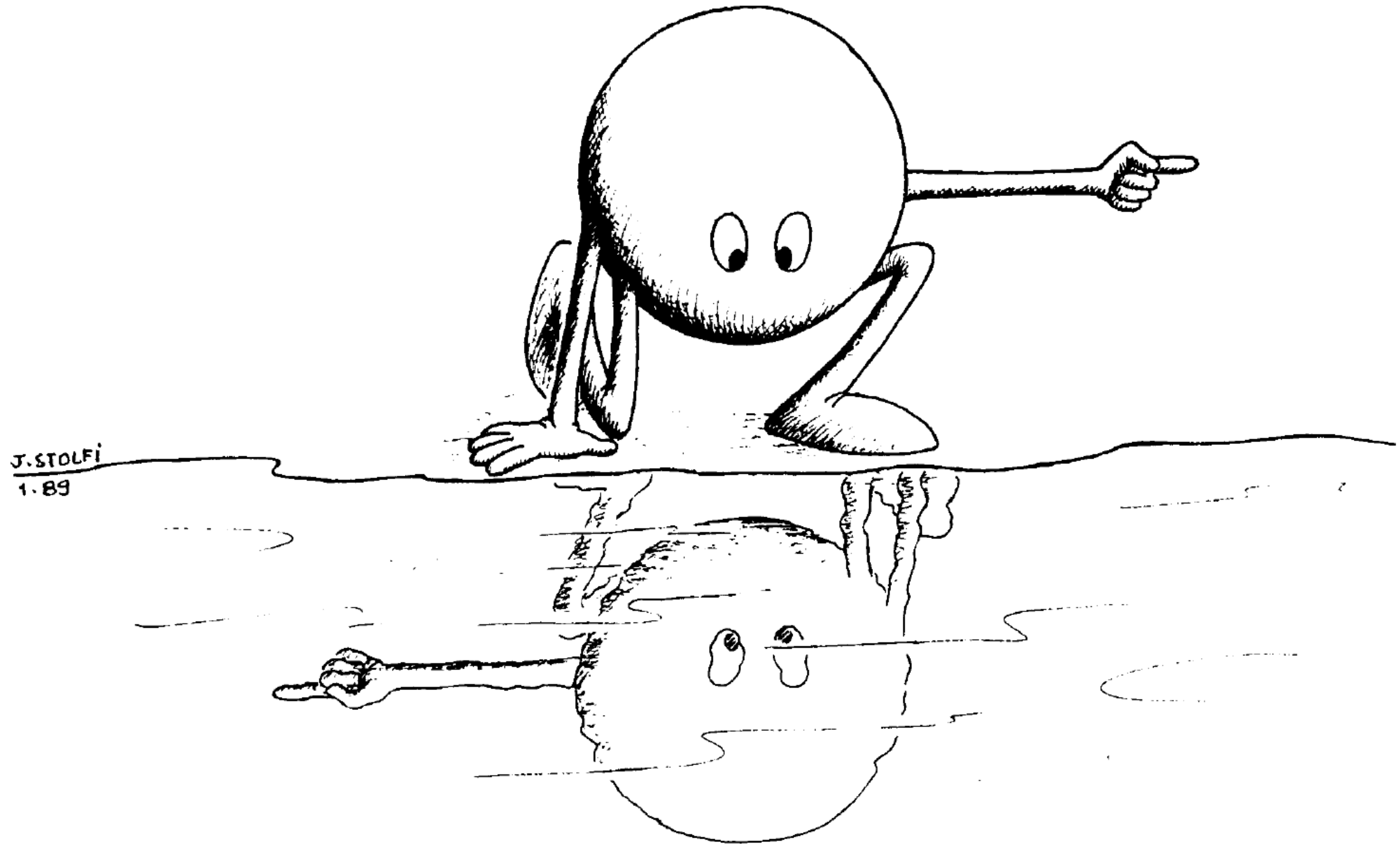
Equivariant non-Linearities

- Dot products and norms of type ℓ features are rotation invariant:

$$\langle f(R.X), g(R.X)\rangle = \langle D^\ell(R) f(X), D^\ell(R) g(X)\rangle = \langle f(X), g(X)\rangle.$$

- Non linearities can by applied to the norms of features as they are invariant:

$$\xi(f^\ell(X), b) = \xi(\|f^\ell(X)\|_2 + b) f^\ell(X).$$


J.STOLFI
1.89