CS348n: Neural Representations and Generative Models for 3D Geometry

Leonidas Guibas
Computer Science Department
Stanford University
Last Time: Learning Vector and Mesh/CAD Representations
Vector (2D) and Mesh/CAD (3) Shape Models

Fonts

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

Autodesk Fusion 360 Dataset
Im2Vec Inference Architecture

4 step process.

Raster input 

Encoder 

Global latent code 

RNN 

Path Decoder 

Path Decoder 

Path Decoder 

Rasterizer 

Rasterizer 

Rasterizer 

Compositing 

Rasterized output 

Path Decoder 

Sampling 

Latent code 

Sampled control points 

Adaptive resampler 

Deformed control points 

Decoder 

Decoded Bézier path 

(circular 1D CNN) 

(circular 1D CNN)
Input images are 128x128
Reconstruction

Input images are 128x128
Multiple spline curves with variable numbers of control points

• Whole model
Attention Results
Irregular
Unordered
&
Unoriented
Goal: CNN directly on the irregular mesh elements

Classification

Segmentation

MeshCNN

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

\[ p = \text{avg}(a, b, e) \]

\[ q = \text{avg}(c, d, e) \]
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. Lambourne¹, Karl D.D. Willis¹, Pradeep Kumar Jayaraman¹, Aditya Sanghi¹, Peter Meltzer², Hooman Shayani¹

Autodesk Research¹, UCL, Computer Science²
Boundary representation models
Topological walks defining neighboring entities
BRepNet convolution

Concatenate feature vectors

MLP
Layer 1
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
• Gap: converting sketches to a CAD model?
Sketch2CAD

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

Protocol:
AddPolyhedron: <plane 3, length 0.25>
Results

Step 1

Step 2

Step 3

Step 4

Step 5
Results
3D Equivariance and Invariance
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
Why Equivariance / Invariance?

• What neural networks can do?

• But what is this?
Gestalt theory – how human perceive objects

- Emergence
- Reification
- Multistability
- Invariance

https://en.wikipedia.org/wiki/Gestalt_psychology#Properties
• Many datasets are **aligned** (shapes under canonical poses)
• Networks trained aligned data **cannot generalize** to arbitrary poses
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
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
✓ “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

So We’re Looking for...

classification

segmentation

reconstruction

airplane

invariant

equivariant

(or invariant, depending on data representation)

equivariant encoder

invariant decoder


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

\[
f(xR; \theta) = f(x; \theta)D(R)
\]

- \( D(R) \) should be independent of \( x \)
- **Special case:** when \( D(R) = R \) is the identity mapping, it is the common-sense “equivariance”
- **Special case:** when \( D(R) = I \) 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]
Classical (scalar) feature $z = [z_1, z_2, \cdots, z_C]^\top \in \mathbb{R}^C$, with $z_i \in \mathbb{R}$

Vector-list feature $V = [v_1, v_2, \cdots, v_C]^\top \in \mathbb{R}^{C \times 3}$, with $v_i \in \mathbb{R}^3$

- For pointcloud with $N$ points $\mathcal{V} = \{V_1, V_2, \cdots, V_N\} \in \mathbb{R}^{N \times C \times 3}$

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 $R \in \text{SO}(3)$:

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

(classical) scalar neurons

vector neurons
VN Features

**Classical**: scalar channels

\[ C \times 1 \text{ feature} \]

**VN**: 3D vector channels

\[ C \times 3 \text{ feature} \]
VN Features (for Point Cloud)

Classical:

\[ N \times C \times 1 \text{ feature} \]

VN:

\[ N \times C \times 3 \text{ feature} \]
VN Linear Layer

**Linear operator:** left multiply by the learnable weight matrix

\[
\begin{bmatrix}
C' 	imes C & \text{weight} \\
\end{bmatrix}
\times
\begin{bmatrix}
C & \text{3 feature} \\
\end{bmatrix}
=
\begin{bmatrix}
C' & \text{3 feature} \\
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
C' 	imes C & \text{weight} \\
\end{bmatrix}
\times
\begin{bmatrix}
C & \text{3 feature} \\
\end{bmatrix}
= \begin{bmatrix}
C' & \text{3 feature} \\
\end{bmatrix}
\times
\begin{bmatrix}
\end{bmatrix}
\]
VN Linear Layer

Vector-list feature \( \mathbf{V} \in \mathbb{R}^{C\times 3} \)

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

\[
\mathbf{V}' = f_{\text{lin}}(\mathbf{V}; \mathbf{W}) = \mathbf{WV} \in \mathbb{R}^{C'\times 3}
\]

**Equivariance** to rotation \( R \in \text{SO}(3) \):

\[
f_{\text{lin}}(\mathbf{VR}; \mathbf{W}) = \mathbf{WVR} = f_{\text{lin}}(\mathbf{V}; \mathbf{W})R = \mathbf{V}'R
\]

- \( \mathbf{W} \)- left multiplication, \( R \)- right multiplication
- Note the absence of a bias term
VN Non-Linearity

ReLU Non-Linearity

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

Learn a feature $q = WV \in \mathbb{R}^{1 \times 3}$
Learn a direction $k = UV \in \mathbb{R}^{1 \times 3}$

For each output vector neuron $v' \in V'$

$$v' = \begin{cases} q & \text{if } \langle q, k \rangle \geq 0 \\ q - \langle q, \frac{k}{\|k\|} \rangle \frac{k}{\|k\|} & \text{otherwise} \end{cases}$$
VN Non-Linearity

ReLU Non-Linearity

- **Non-linear layer** (with built-in linear layer)
  
  \[ q + k \]
  
- A single non-linear layer detached from linear layer: \( q = v \)
  
  - introduce additional network depth!

- Other non-linearities

overall structure
VN Pooling

✓ Mean pooling

? Max pooling
  • (Similar to non-linearity)
  • argmax alone learned directions

= argmax
Mean pooling

Max pooling

- similar to non-linearities → use learnable directions

Given a set of vector-lists \( \mathcal{V} = \{ V_1, V_2, \cdots, V_N \} \in \mathbb{R}^{N \times C \times 3} \)

Learn data-dependent directions \( \mathcal{K} = \{ W V_n \}_{n=1}^{N} \in \mathbb{R}^{N \times C \times 3} \) with learnable weights \( W \in \mathbb{R}^{C \times C} \)

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

\[
f_{\text{MAX}}(\mathcal{V})[c] = V_{n^*}[c] \quad \text{where} \quad n^* = \arg\max \langle W V_n[c], V_n[c] \rangle
\]
VN Normalizations

✓ LayerNorm
✓ InstanceNorm
✓ Dropout

? BatchNorm

averaging across arbitrarily rotated inputs would not necessarily be meaningful

(classical) scalar neurons

Vector neurons
VN Normalizations

BatchNorm

element-wise 2-norm = ElementWiseNorm

... = BatchNorm

output = ElementWiseNorm × ...
VN Normalizations

BatchNorm

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

\[ N_b = \text{ElementWiseNorm}(V_b) \in \mathbb{R}^{N \times 1} \]

\[ \{N'_b\}^B_{b=1} = \text{BatchNorm} \left( \{N_b\}^B_{b=1} \right) \]

\[ V'_b = V_b[c] \frac{N'_b[c]}{N_b[c]} \quad \forall c \in [C] \]

• Element-wise norm: 2-norm for each vector \( v_c \in V_b \)
VN Invariant Layer

\((\text{equivariant feature}) \times (\text{equivariant feature})^T = (\text{invariant feature})\)
VN Invariant Layer

Specifically...

\[ C \times 3 \times 3 \]

equivariant coordinate system

\[ C' \times 3 \]

output canonicalized feature
For pointcloud: combine global information with local features.

- VN Invariant Layer
- Concatenate with global mean
- $C \times 3$ input
- $3 \times 3$ equivariant coordinate system
VN Invariant Layer

• Product of an equivariant signal \( \mathbf{V} \in \mathbb{R}^{C \times 3} \) by the transpose of another equivariant signal \( \mathbf{T} \in \mathbb{R}^{C' \times 3} \) → invariant signal

• **Special case:** \( \mathbf{T} \in \mathbb{R}^{3 \times 3} \) - an equivariant coordinate system

• For pointcloud, concatenate local feature \( \mathbf{V} \in \mathbb{R}^{C \times 3} \) with global mean \( \overline{\mathbf{V}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{V}_n \in \mathbb{R}^{C \times 3} \)

**Invariant layer:**

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

\[
\text{VN-In}(\mathbf{V}_n) = \mathbf{V}_n \mathbf{T}_n^\top
\]
Build VN Networks: VN-DGCNN

DGCNN

Edge feature: \[ E_{nm}' = \text{ReLU}(\Theta(m - n) + \Phi_n) \]

Aggregation: \[ E_n = \text{Pool}_{m:(n,m) \in \mathcal{E}}(E_{nm}') \]

VN-DGCNN

Edge feature: \[ E_{nm}' = \text{VN-ReLU}(\Theta(m - n) + \Phi_n) \]

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

[Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, J. M. Solomon, TOG 2019]
Convolution in latent spaces, graph edges not embedded in $\mathbb{R}^3$

**DGCNN:**

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

- **Edge conv**
  
  $$e'_{nm} = \text{ReLU}(\Theta(x_m - x_n) + \Phi x_n)$$

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

**VN-DGCNN:**

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

- **Edge conv**
  
  $$E'_{nm} = \text{VN-ReLU}(\Theta(V_m - V_n) + \Phi V_n)$$

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

---

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

\[ x' = \text{Pool}_{x_n \in X} (h(x_1), h(x_2), \ldots, h(x_N)) \]

VN-PointNet

\[ v' = \text{VN-Pool}_{v_n \in V} (f(v_1), f(v_2), \ldots, f(v_N)) \]

[C. R. Qi, H. Su, K. Mo, L. J. Guibas, CVPR 2017]
No convolutions, only point-wise feature transformations

**PointNet:** \[ x' = \text{Pool}_{x_n \in x}(h(x_1), \cdots, h(x_N)) \]

**VN-PointNet:** \[ V' = \text{VN-Pool}_{V_n \in V}(f(V_1), \cdots, f(V_N)) \]

- \( h \) MLP, \( f \) VN-MLP
- First layer single channel \( \rightarrow \) 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

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

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

Results on ShapeNet (Examples)

<table>
<thead>
<tr>
<th>OccNet</th>
<th>VN-OccNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/I</td>
<td>I/I</td>
</tr>
<tr>
<td>I/\text{SO}(3)</td>
<td>\text{SO}(3)/\text{SO}(3)</td>
</tr>
</tbody>
</table>

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

**VN-OccNet:**
Consistent across rotations
Neural Implicit Reconstruction

Results on ShapeNet (Examples)

<table>
<thead>
<tr>
<th>OccNet</th>
<th>VN-OccNet</th>
<th>1/1</th>
<th>1/\text{SO}(3)</th>
<th>\text{SO}(3)/\text{SO}(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla OccNet:</td>
<td>Blurry shapes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Neural Implicit Reconstruction

Results on ShapeNet (Examples)

<table>
<thead>
<tr>
<th>OccNet</th>
<th>VN-OccNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/I</td>
<td>I/\text{SO}(3)</td>
</tr>
</tbody>
</table>

Vanilla OccNet: Averaged shapes
Neural Implicit Reconstruction

Results on ShapeNet (Examples)

<table>
<thead>
<tr>
<th>OccNet</th>
<th>VN-OccNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I/I</td>
</tr>
</tbody>
</table>

**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
A type $\ell \in \mathbb{N}$ equivariant feature map $f^\ell : \mathbb{R}^{n \times 3} \to \mathbb{R}^{2\ell+1}$ satisfying $f^\ell(X R^\top) = D^\ell(R) f^\ell(X)$ for $R \in \text{SO}(3)$, where $D^\ell(R) \in \text{SO}(2\ell + 1)$ is the type $\ell$ Wigner matrix.
Tensor Field Networks (TFN)

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

  \[ f^\ell (X)_{i,c,:} \quad \text{Pointwise features} \]
  \[ g^\ell (X)_{c,:} \quad \text{Global features} \]
Examples of Type 0 Features

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

\[
f^0(X)_{i,c,:} = f^0(X R^\top)
\]

Segmentation

Classification

\[
g^0(\text{Chair}) = \text{Chair}
\]

\[
g^0(\text{Chair}) = \text{Chair}
\]
Examples of type 1 Features

- We have $D^1(R) = R$, therefore 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 $S_2$ form an orthonormal basis of $L^2(S_2)$.

- Just like type $\ell$ equivariant features the vector of degree $\ell$ spherical harmonics $Y^\ell(x) \in \mathbb{R}^{2\ell+1}$, satisfies $Y^\ell(Rx) = D^\ell(R)Y^\ell(x)$. 

![Diagram showing spherical harmonics](image)
Spherical Harmonics & Higher Degree Features

- Spherical harmonics rotating around the $z$ axis:
• Dot products of type $\ell$ 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.

$$\text{TFN} =$$

[Diagram showing the structure of TFN with convolution and subsampling layers]
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 $\mathbb{Z}^d$ ($d = 2, 3$).
• 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:

\[
\text{Conv}_X(f, \kappa, W)_k := \sum_{ij} W_{kij} f_i * X \kappa_j.
\]
Convolutions and Rotations

• For any rotation $R$ we have:

$$f \ast_{XR^T} \kappa_j(X_i) = \sum_k f_k \kappa_j(R(X_k - X_i)).$$

• Problem what is the relation between $(f \ast_X \kappa_j)_j$ and $(f \ast_{XR^T} \kappa_j)_j$?
Steerable Kernel Basis

- A steerable kernel basis, is a kernel basis \((\kappa_k)_k\) such that, the rotation of any kernel \(\kappa_j\) 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:

\[
f \ast_{XR^\top} \kappa_j(x) = \sum_j D(R)_{jk} f \ast_X \kappa_k(x)
\]
3D Steerable Basis

- For each $\ell$ we have a steerable basis of the form:

$$\kappa_{r,m}^{\ell}(x) := \varphi_r(\|x\|_2) Y_m^\ell \left( \frac{x}{\|x\|_2} \right)$$

- Where $\varphi_r$ 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 $\ell$ 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 $D^\ell(t) = e^{i\ell t}$.

<table>
<thead>
<tr>
<th>$r \setminus \ell$</th>
<th>0 Re</th>
<th>1 Re 1 Im</th>
<th>2 Re 2 Im</th>
<th>3 Re 3 Im</th>
<th>4 Re 4 Im</th>
<th>\ldots</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td>\ldots</td>
</tr>
<tr>
<td>2</td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td>\ldots</td>
</tr>
<tr>
<td>1</td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
<td><img src="image15.png" alt="Image" /></td>
<td>\ldots</td>
</tr>
<tr>
<td>0</td>
<td><img src="image16.png" alt="Image" /></td>
<td><img src="image17.png" alt="Image" /></td>
<td><img src="image18.png" alt="Image" /></td>
<td><img src="image19.png" alt="Image" /></td>
<td><img src="image20.png" alt="Image" /></td>
<td>\ldots</td>
</tr>
</tbody>
</table>

Example of steerable kernels, \(\ldots\) (source Learning Steerable Filters with Rotation Equivariant CNNs)
• We consider a collection of steerable kernel bases \((\kappa_r^\ell)_{r \ell}\) over \(\mathbb{R}^d\) indexed by an equivariance type \(\ell\) such that for any rotation \(R \in \text{SO}(d)\) we have:

\[
\kappa^\ell(Rx) = D^\ell(R)\kappa^\ell(x)
\]

• For each \(\ell\) we define the associated steerable convolution operator:

\[
\text{SConv}_X^\ell(f, \kappa_r^\ell, W)_{km} := \sum_{ir} W_{kir} f_i \ast_X \kappa_r^\ell.
\]
Comparing Regular and Steerable Convolution

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

\[
\text{Conv}_X(f, \kappa, W)_k := \sum_{ij} W_{ki}f_i \ast_X \kappa_j.
\]

- Steerable convolution does not take linear combination over the "steerable" index \(m\) in order to preserve equivariance:

\[
\text{SConv}^\ell_X(f, \kappa^\ell_r, W)_{km} := \sum_{ir} W_{kir}f_i \ast_X \kappa^\ell_{rm}.
\]
A rotation $R$ of the pointcloud $X$ induces a linear transform $D^\ell(R)$ which is independent of $X$:

$$\text{SConv}^\ell_{XR^\top}(f, \kappa_r, W)_k = \sum_{ir} W_{kir} f_i *_{XR^\top} \kappa_r^\ell$$

$$= \sum_{ir} W_{kir} f_i *_X D^\ell(R) \kappa_r^\ell$$

$$= D^\ell(R) \text{SConv}^\ell_X(f, \kappa_r^\ell, W)_k$$
Why not Mix Different Types?

- In 2D a type $\ell$ features makes $\ell$ 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.
• 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 $\ell$ 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 invari-ant:

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