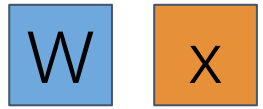
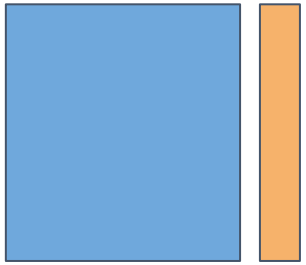
Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds

Nathanie Thomas，Tess Smidt，Steven Kearnes .etc.

Neural networks are specially designed for different data types.
Assumptions about the data type are built into how the network operates.

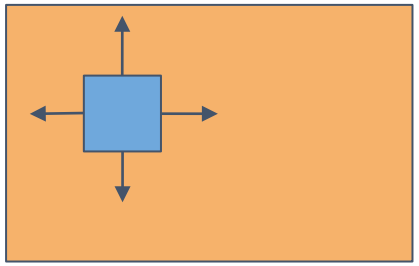


Arrays ⇨ Dense NN



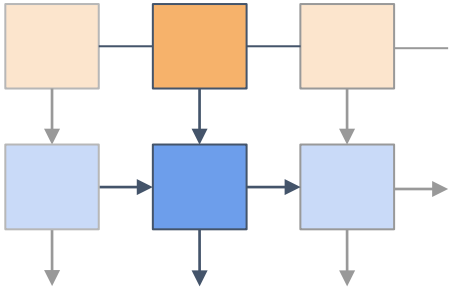
Components are independent.

2D images
⇨ Convolutional NN



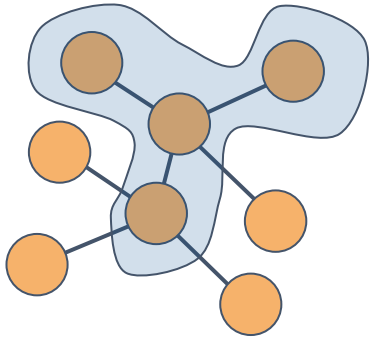
The same features can be found anywhere in an image. Locality.

Text ⇨ Recurrent NN



Sequential data. Next input/output depends on input/output that has come before.

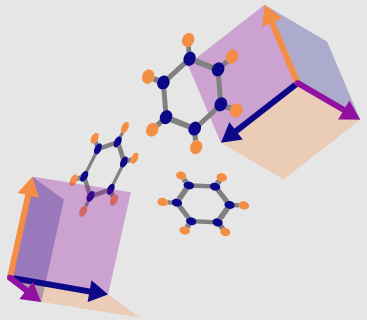
Graph ⇨ Graph (Conv.) NN



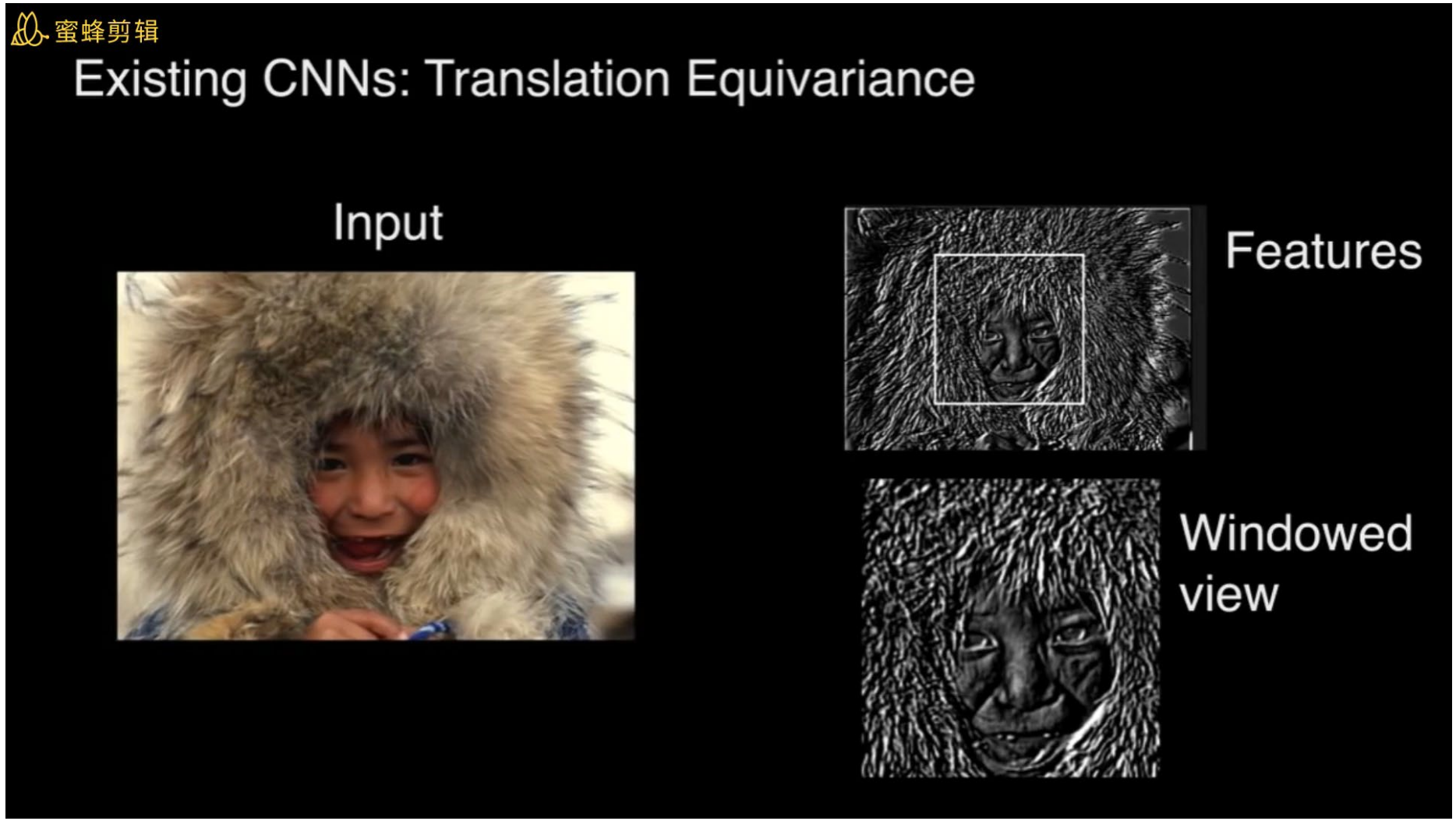
Topological data. Nodes have features and network passes messages between nodes connected via edges.

3D physical data
⇨ Euclidean NN

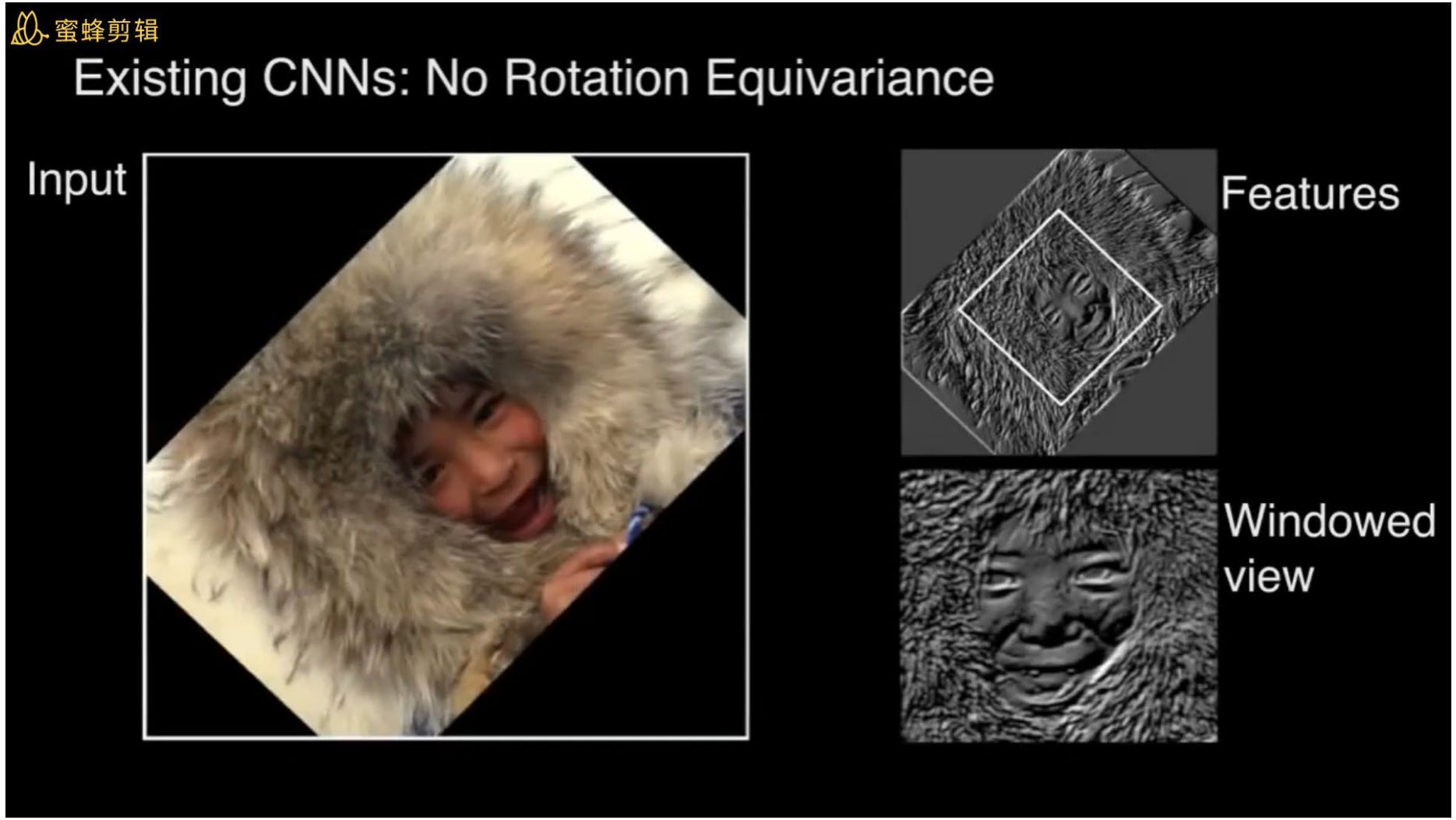
Data in 3D Euclidean space. Freedom to choose coordinate system.



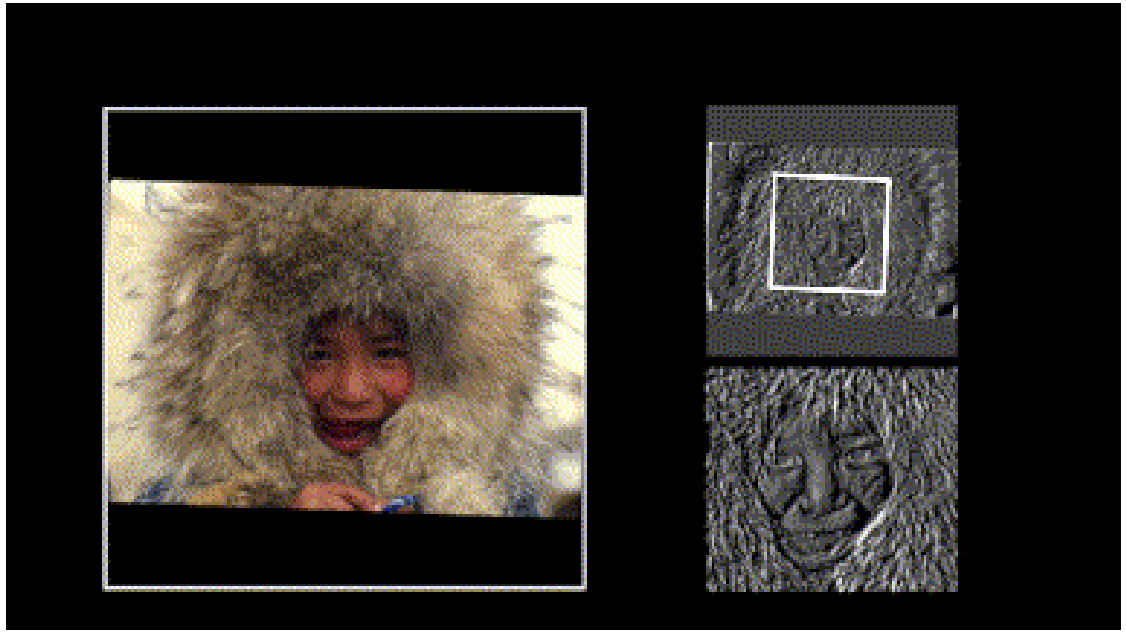
CNN


蜜蜂剪辑
Existing CNNs: Translation Equivariance
Input
Features
Windowed view

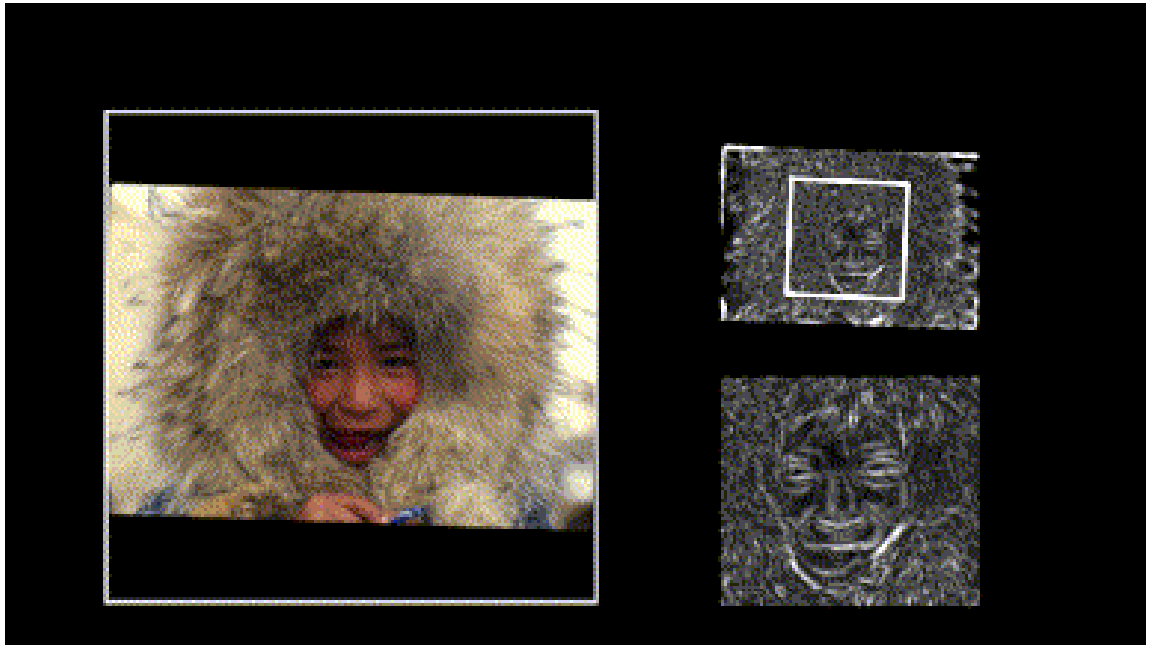
CNN-LACK OF KNOWN ROTATION EQUIVARIANCE:



CNN vs Hnet

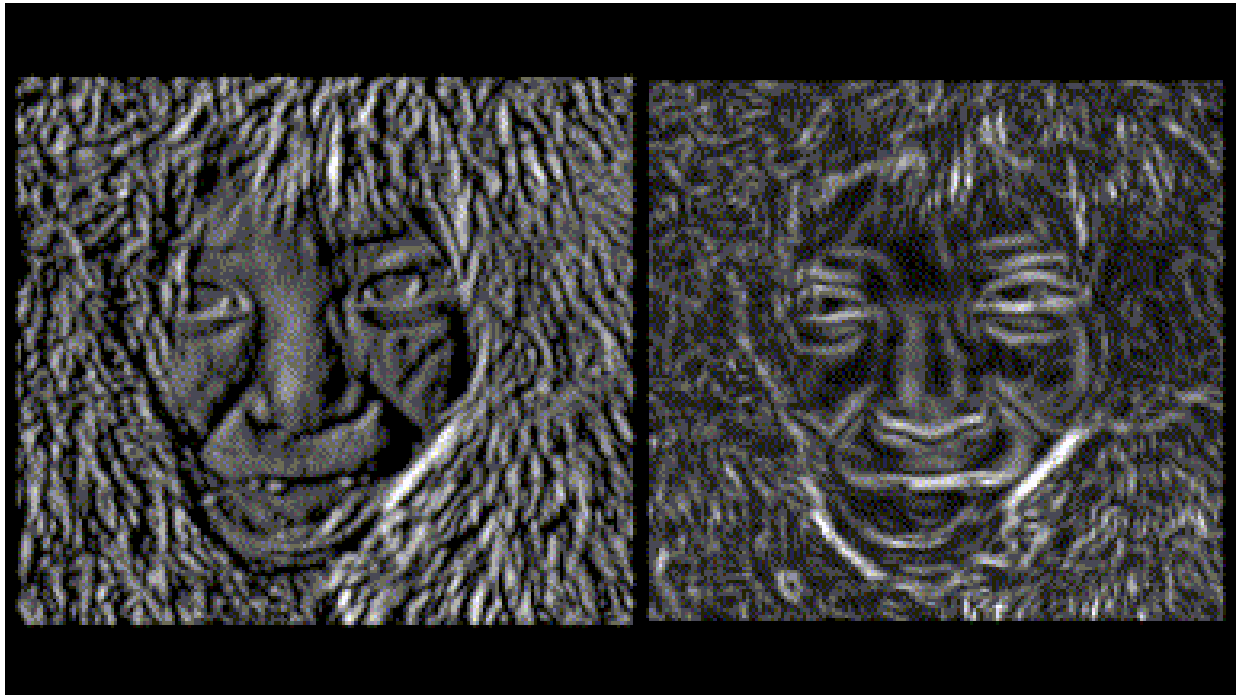


Regular CNN



Regular HNet

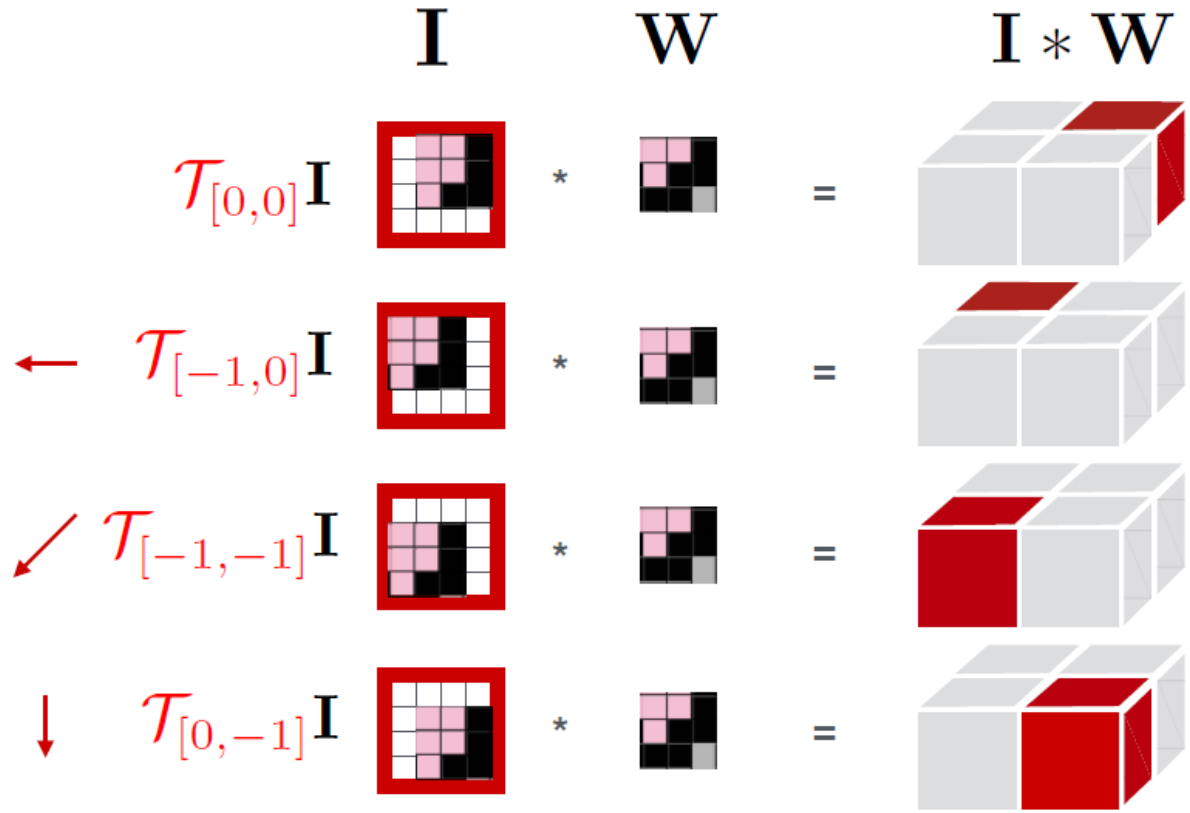
Equivalent



CONVOLUTIONS ON TRANSLATION GROUP

$$[\mathbf{I} * \mathbf{W}](\mathbf{y}) = \sum_{\mathbf{x} \in \mathbb{Z}^2} \mathbf{I}(\mathbf{x}) \mathbf{W}(\mathbf{x} - \mathbf{y})$$

Translating kernel



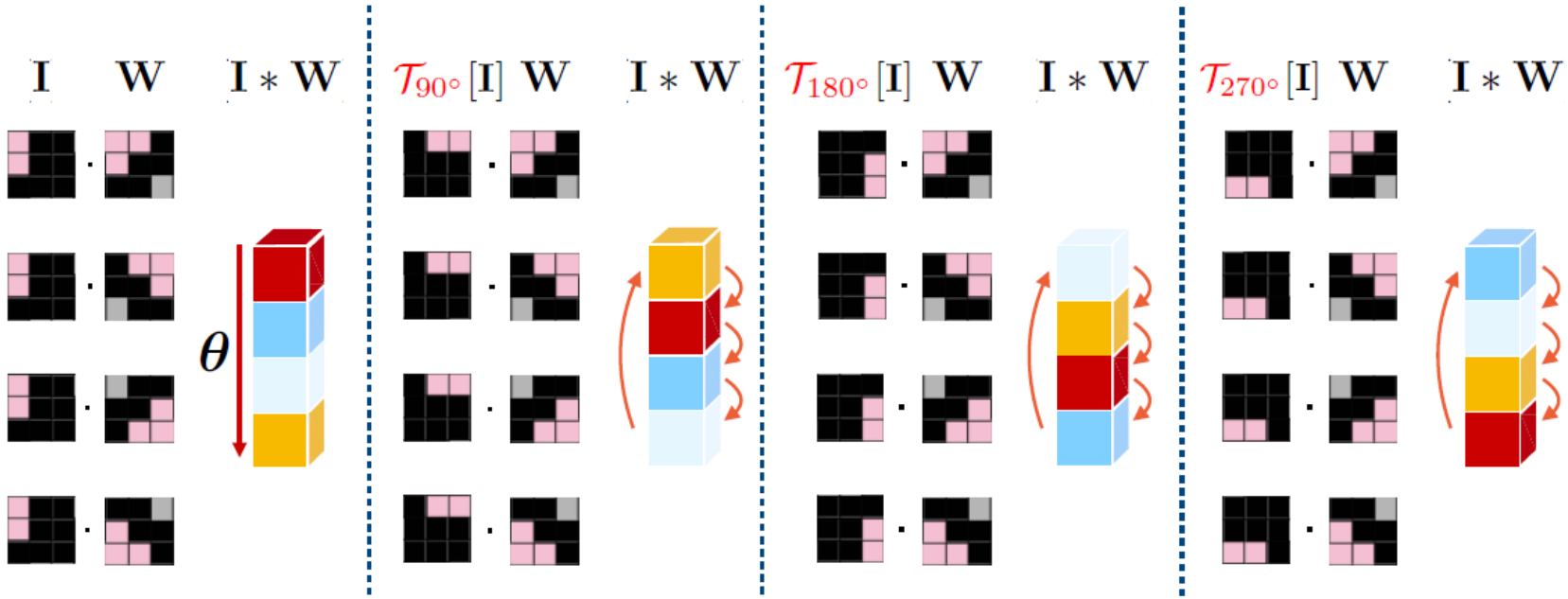
CONVOLUTIONS ON ROTATION GROUP angle

Group convolution **Notice index of convolution**

Ref.
$$[\mathbf{I} * \mathbf{W}](\mathbf{y}) = \sum_{\mathbf{x} \in \mathbb{Z}^2} \mathbf{I}(\mathbf{x}) \mathbf{W}(\mathbf{x} - \mathbf{y})$$

$$[\mathbf{I} * \mathbf{W}](\boldsymbol{\theta}) = \sum_{\mathbf{x} \in \mathbb{Z}^2} \mathbf{I}(\mathbf{x}) \mathbf{W}(\mathbf{R}_{\theta}^{-1} \mathbf{x})$$

Rotating kernel



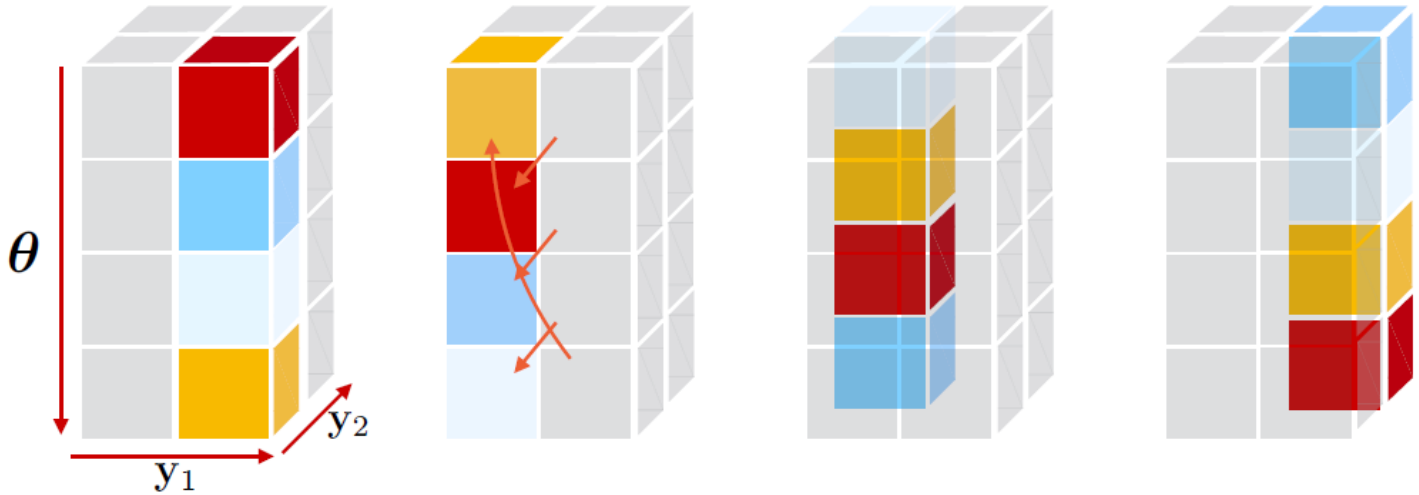
@ hidden layer $$[\mathbf{I} * \mathbf{W}](\psi) = \sum_{\theta \in \Theta} \mathbf{I}(\theta) \mathbf{W}(\theta - \psi)$$

CONVOLUTIONS ON ROTO-TRANSLATION

Group convolution

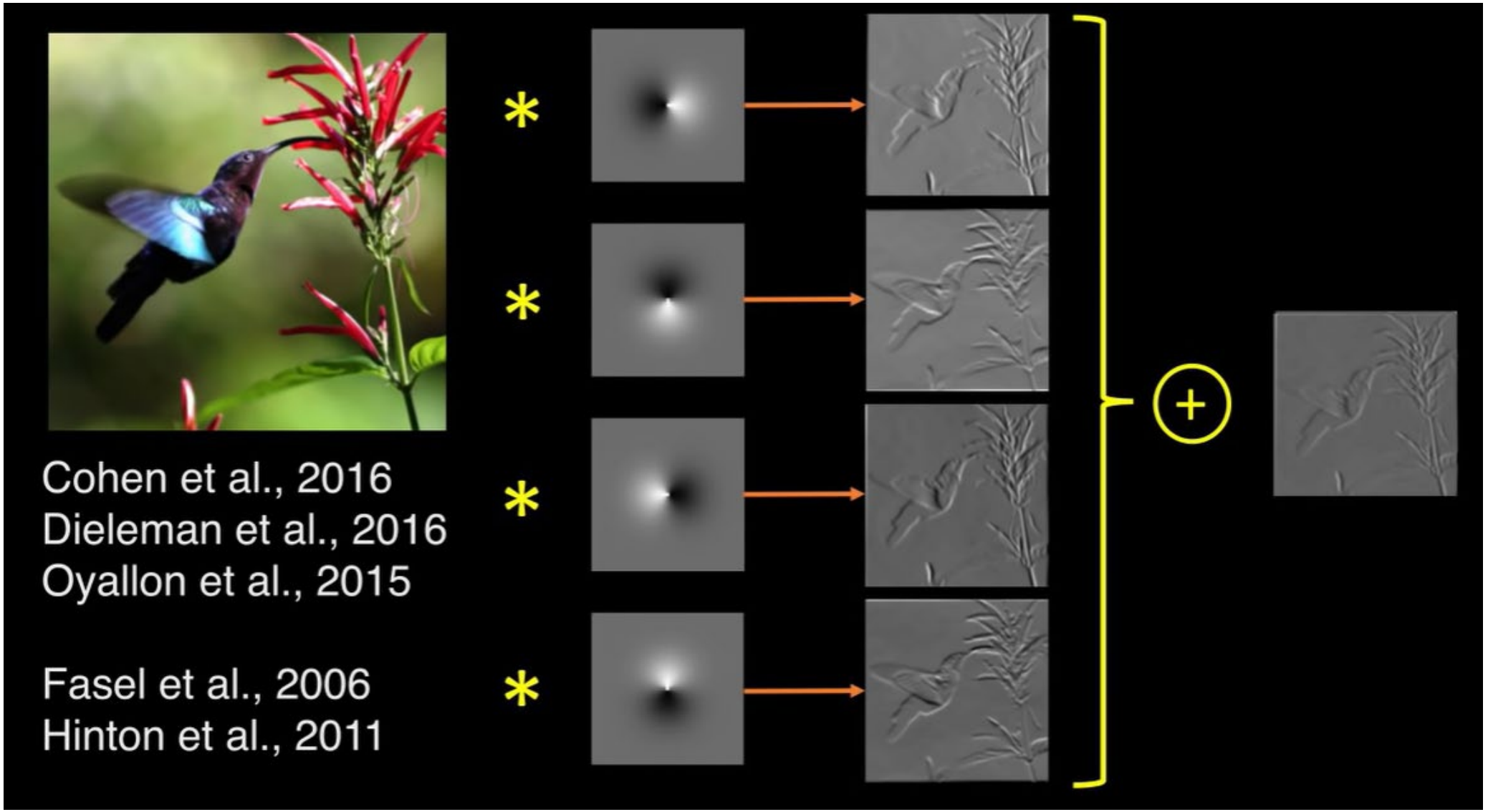
$$[\mathbf{I} * \mathbf{W}](\boldsymbol{\theta}, \mathbf{y}) = \sum_{\mathbf{x} \in \mathbb{Z}^2} \mathbf{I}(\mathbf{x}) \mathbf{W}(\mathbf{R}_{\theta}^{-1}\mathbf{x} - \mathbf{y})$$

Roto-translating kernel

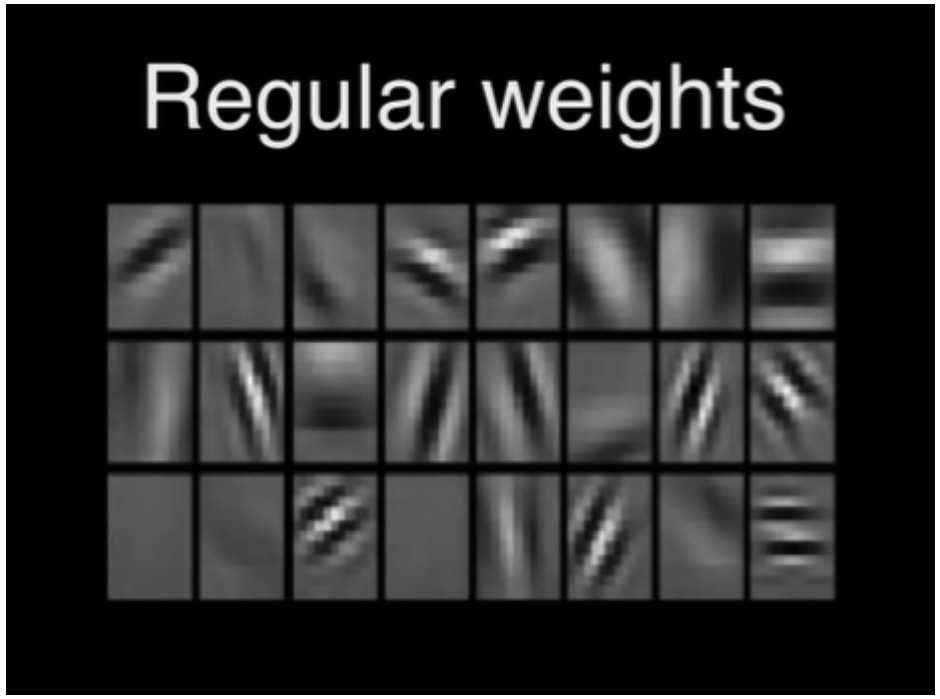


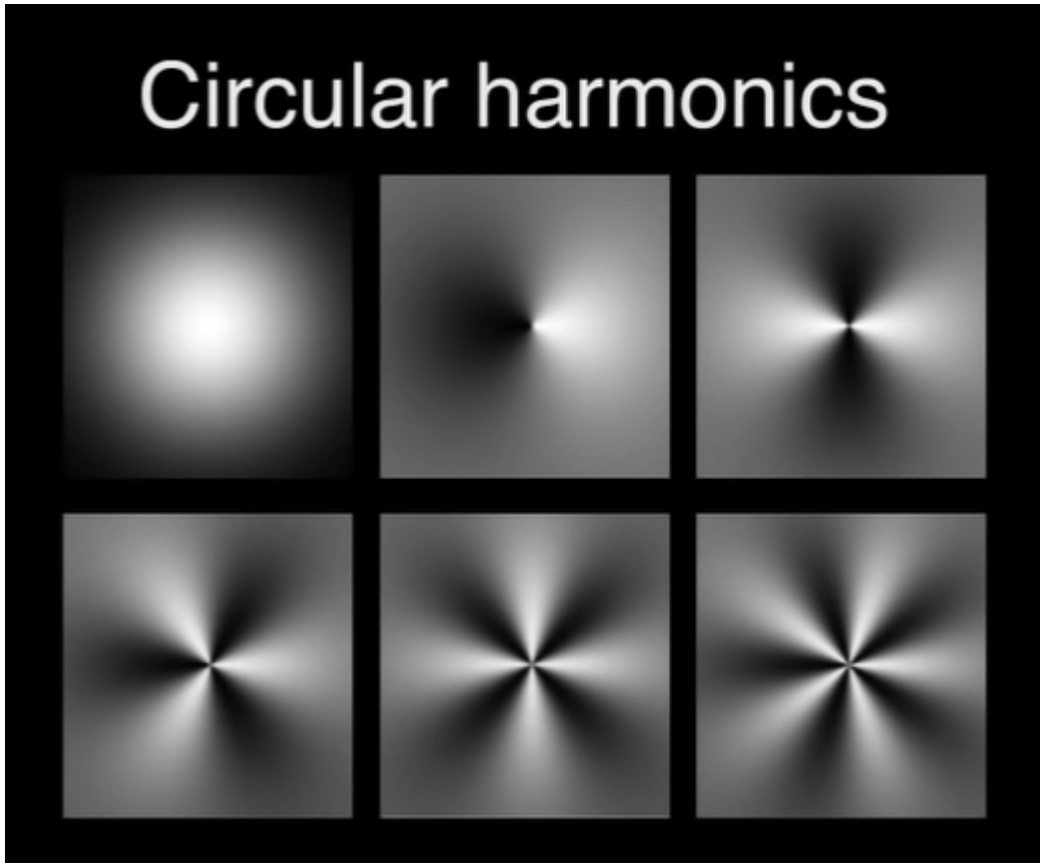
@ hidden layer $[\mathbf{I} * \mathbf{W}](\boldsymbol{\psi}, \mathbf{z}) = \sum_{(\boldsymbol{\theta}, \mathbf{Z}) \in (\Theta \times \mathbb{Z}^2)} \mathbf{I}(\boldsymbol{\theta}, \mathbf{y}) \mathbf{W}(\mathbf{R}_{\psi}^{-1}\mathbf{y} - \mathbf{z}, \boldsymbol{\theta} - \boldsymbol{\psi})$

CONVOLUTIONS ON ROTO-TRANSLATION-Harmonic Networks-discrete



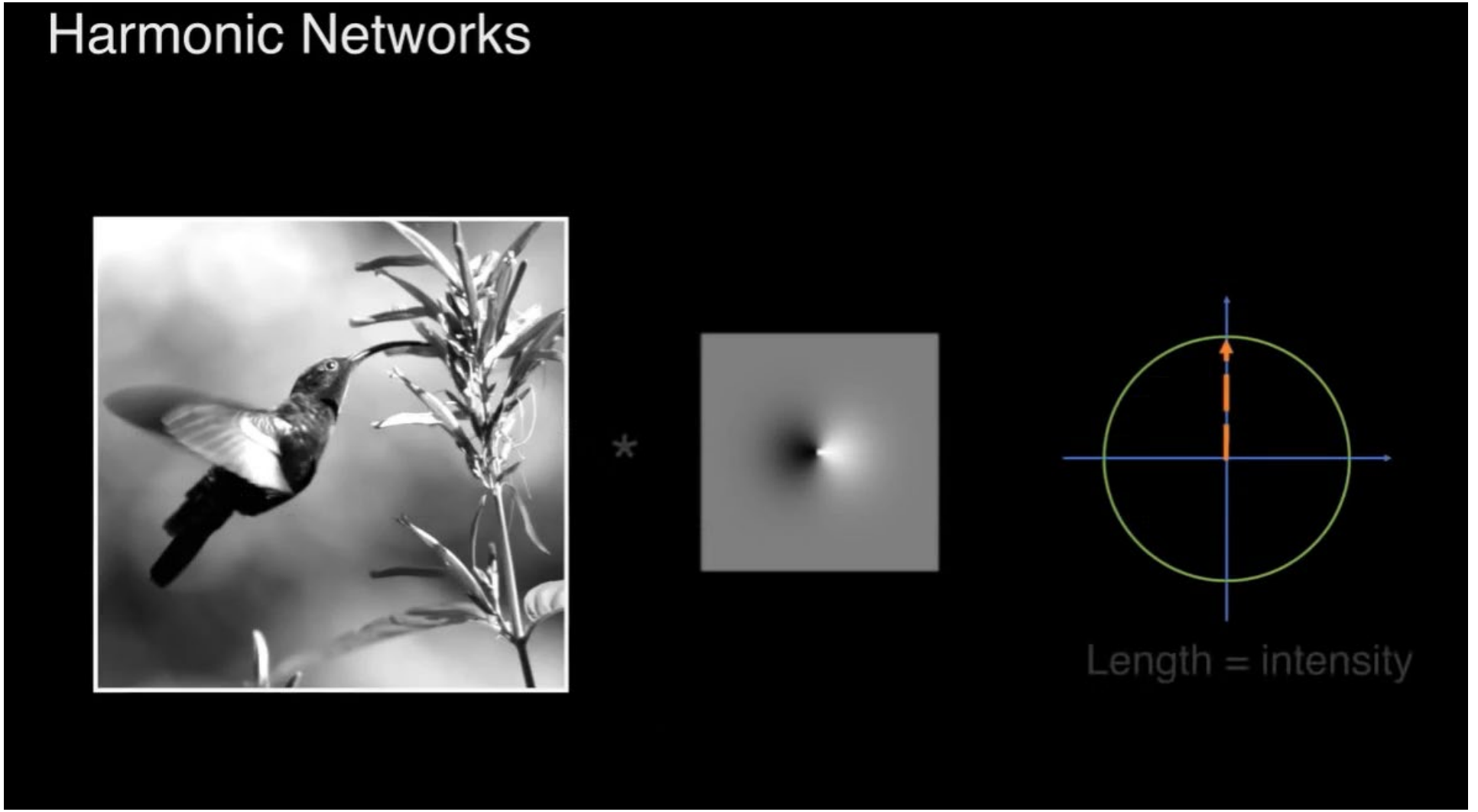
CONVOLUTIONS ON ROTO-TRANSLATION-Harmonic Networks



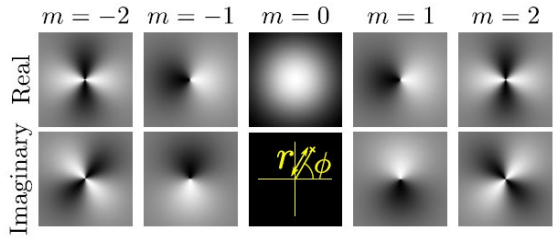


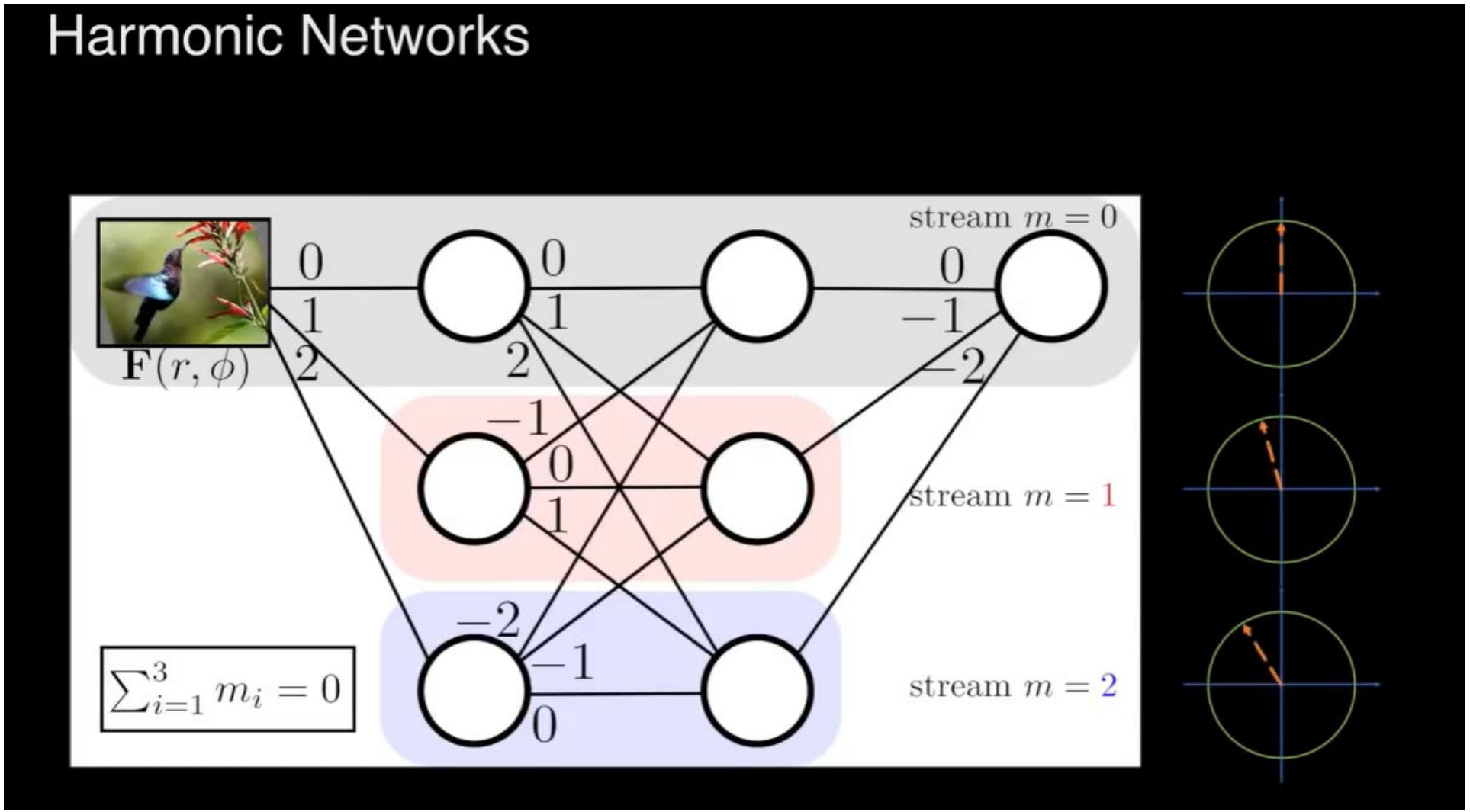
continuous

CONVOLUTIONS ON ROTO-TRANSLATION-Harmonic Networks

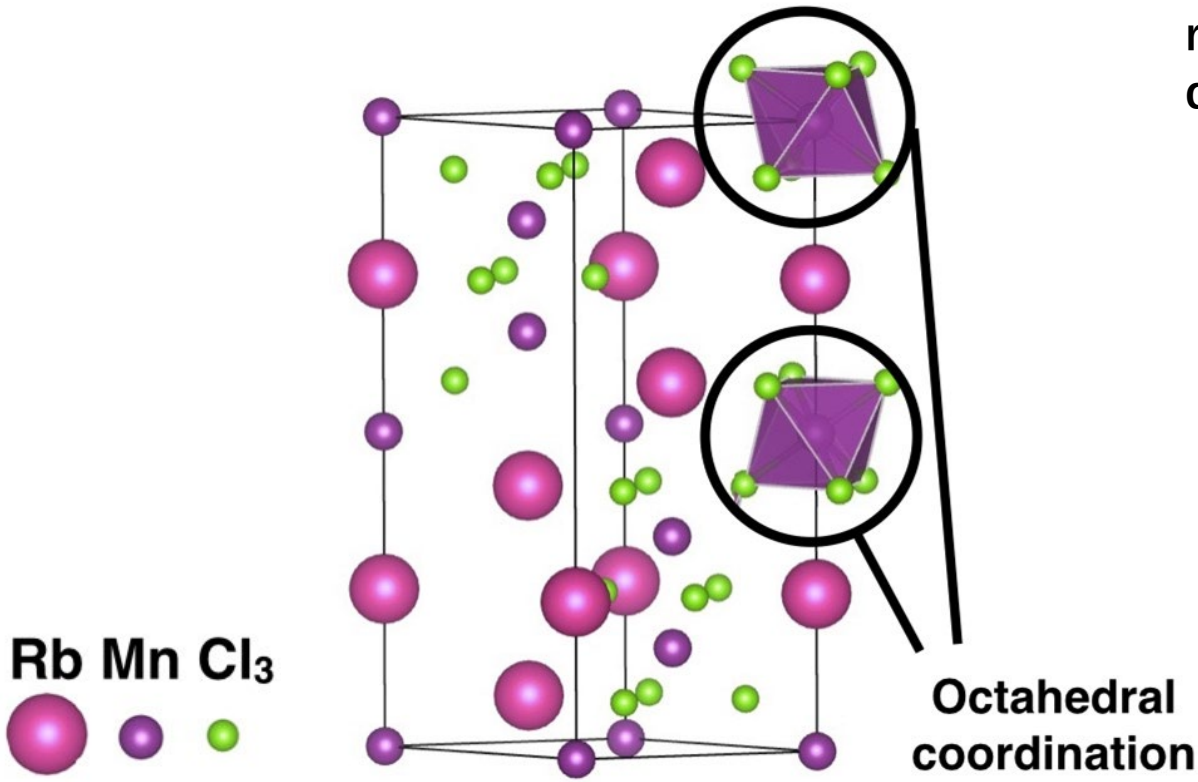


CONVOLUTIONS ON ROTO-TRANSLATION-Harmonic Networks



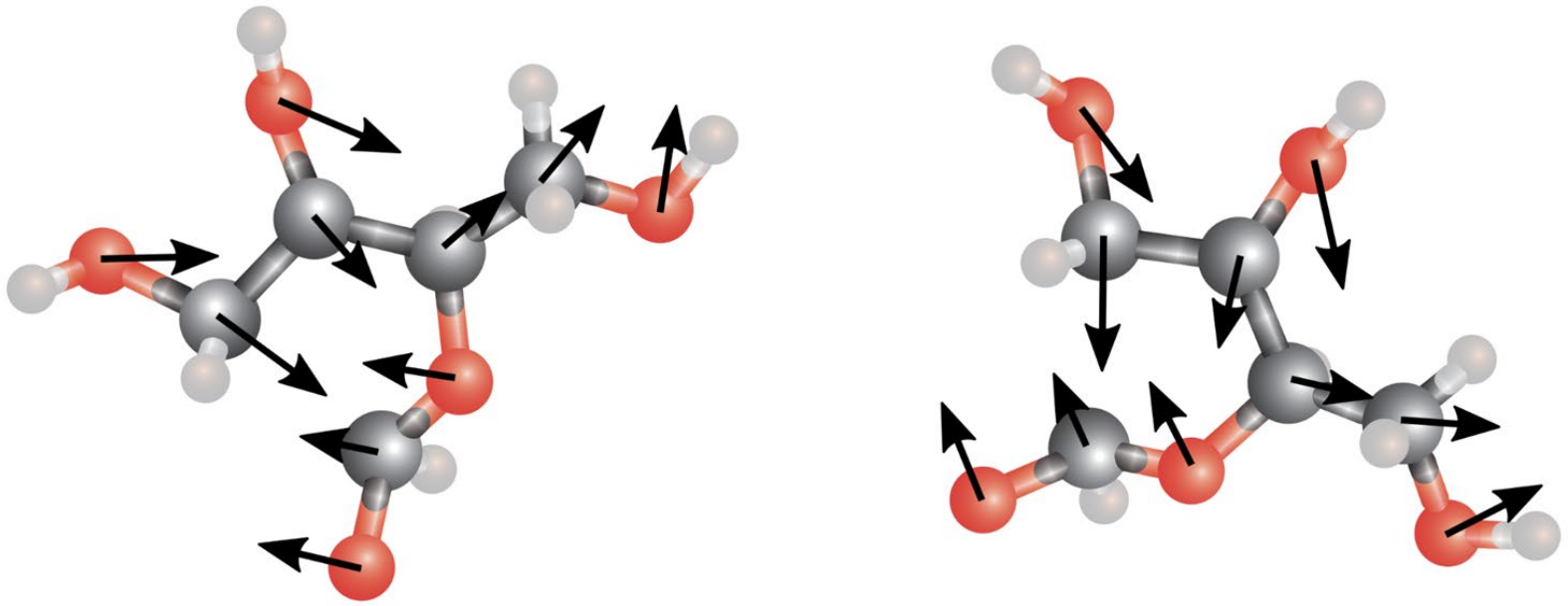


What assumptions do we want “built in” to our neural networks (for materials data)?



These networks can recognize equivalent recurring geometric patterns that appear in different locations and orientations.

Given a molecule and a rotated copy,
predicted forces are the same up to rotation.
(Predicted forces are equivariant to rotation.)



Previous work on 2D rotation-equivariance uses filters based on circular harmonics.

To extend to 3D, we use spherical harmonics.

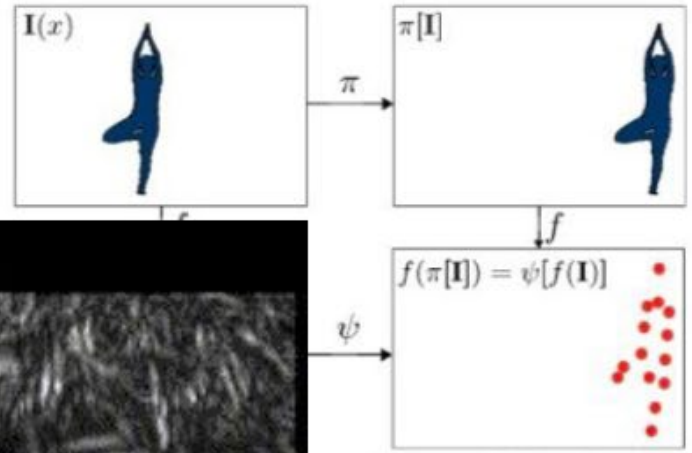
Harmonic Networks: Deep Translation and Rotation Equivariance

Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov and Gabriel J. Brostow
{d.worrall, s.garbin, d.turmukhambetov, g.brostow}@cs.ucl.ac.uk
University College London*

Apr 2017

Abstract

Translating or rotating an input image should not affect the results of many computer vision tasks. Convolutional neural networks (CNNs) are already translation equivariant: input image translations produce proportionate feature map translations.



on equivariance in CNNs arises from
hat a translation π of the input image **I**,

Rotated image **CNN filter output** **Harmonic filter output**

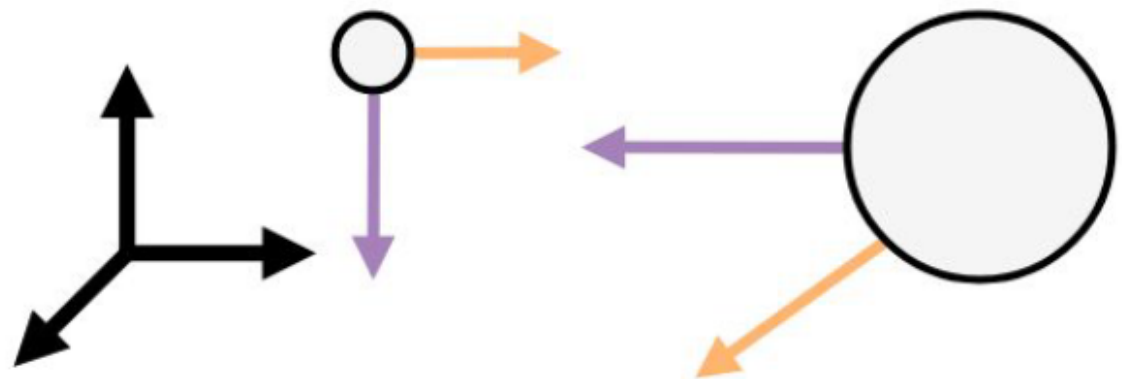
http://visual.cs.ucl.ac.uk/pubs/harmonicNets/

The input and output of our network is represented as tensors with **point** (or atom), **channel**, and **representation** indices organized by irreducible representation.

$$V^{(l)}_{acm} =$$

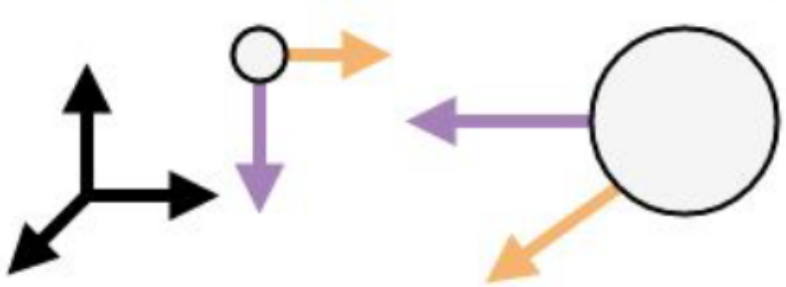
```
{0: [[[m0]],[[m1]]],
 1: [[[v0x, v0y, v0z],[a0x, a0y, a0z]],
     [[v1x, v1y, v1z],[a1x, a1y, a1z]]]}
```

`l` : dictionary key, l
[ ] point index, a
[ ] channel index, c
[ ] representation index, m



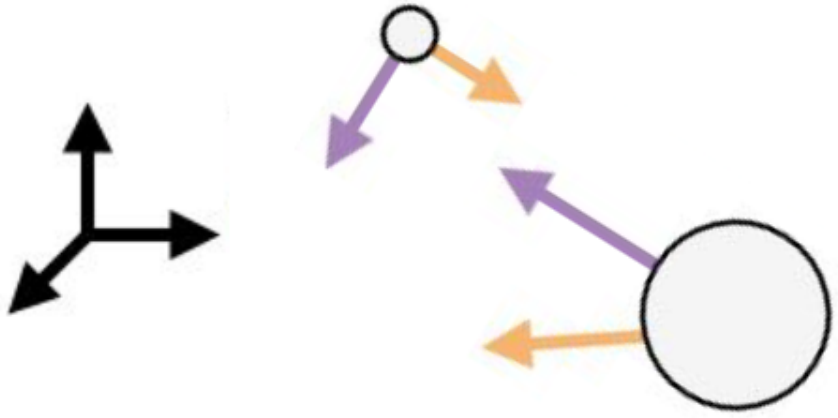
Geometric tensors transform predictably under 3D rotation.

Two point **masses** with **velocity** and **acceleration.**



```
geometry = [[x0, y0, z0],[x1, y1, z1]]
features = [
    [m0, v0y, v0z, v0x, a0y, a0z, a0x]
    [m1, v1y, v1z, v1x, a1y, a1z, a1x]
```

Same system, with rotated coordinates.



Convolutional kernels...

Learned Parameters

with no symmetry:

$$W(\vec{r})$$

with 2D circular harmonics:

$$R(r)e^{im\phi} \longrightarrow [-\pi,\pi]$$

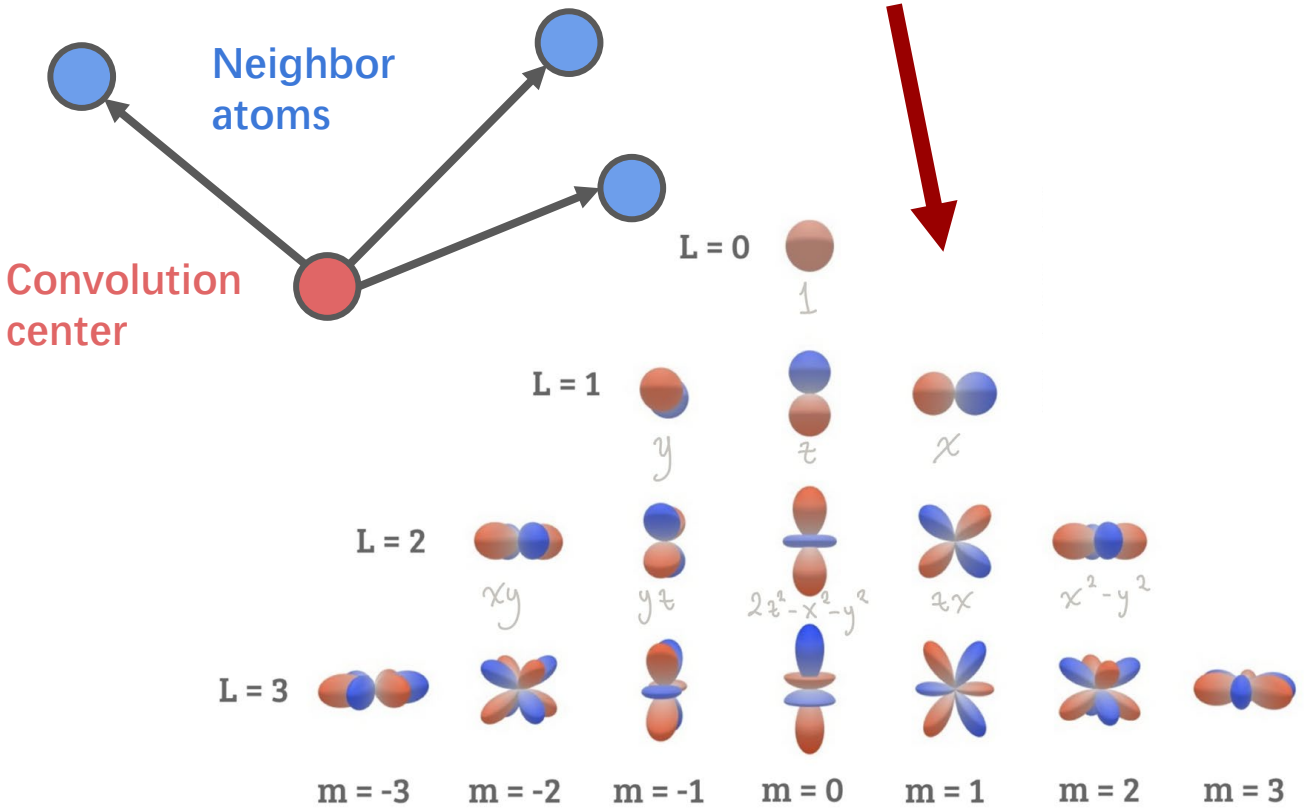
with 3D spherical harmonics:

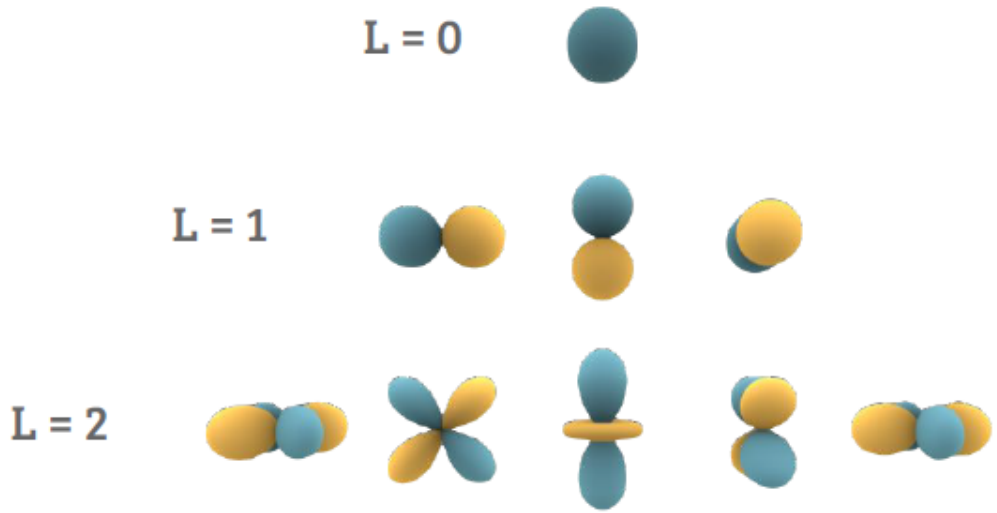
$$R(r)Y_l^m(\hat{r})$$

Euclidean Neural Networks are similar to convolutional neural networks…

Equivariant convolutional filters are based on learned radial functions and spherical harmonics…

$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$





Irreducible representations

Scalars fields $l = 0$

Vectors fields $l = 1$

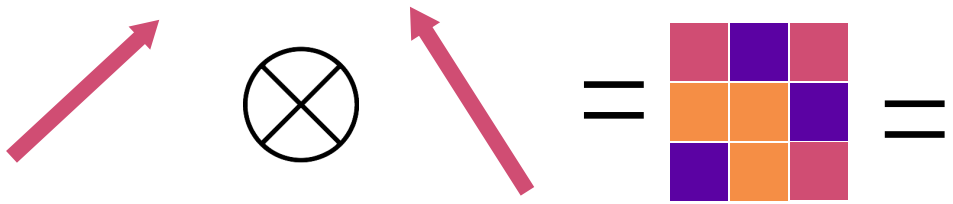
3x3 Matrix fields $l = 0 \oplus 1 \oplus 2$

```
Rs_s_orbital = o3.Irrep("0e")

Rs_p_orbital = o3.Irrep("1o")

Rs_d_orbital = o3.Irrep("2e")

Rs_f_orbital = o3.Irrep("3o")

scalar = e3nn.o3.Irrep("0e")  # L=0, even
vector = e3nn.o3.Irrep("1o")  # L=1, odd

irreps = 1 * scalar + 1 * vector + 1 * vector
```

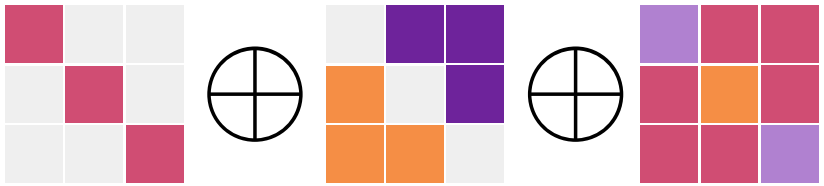
Interactions in equivariant models are more complex.

How do we interact invariant objects? Scalar multiplication.

How do we interact equivariant objects?
Tensor products!



Generalizes to higher orders. Same mathematics that describes atomic interactions, e.g. selection rules in spectroscopy.



dot product
trace
invariant
L=0
1 degree of freedom

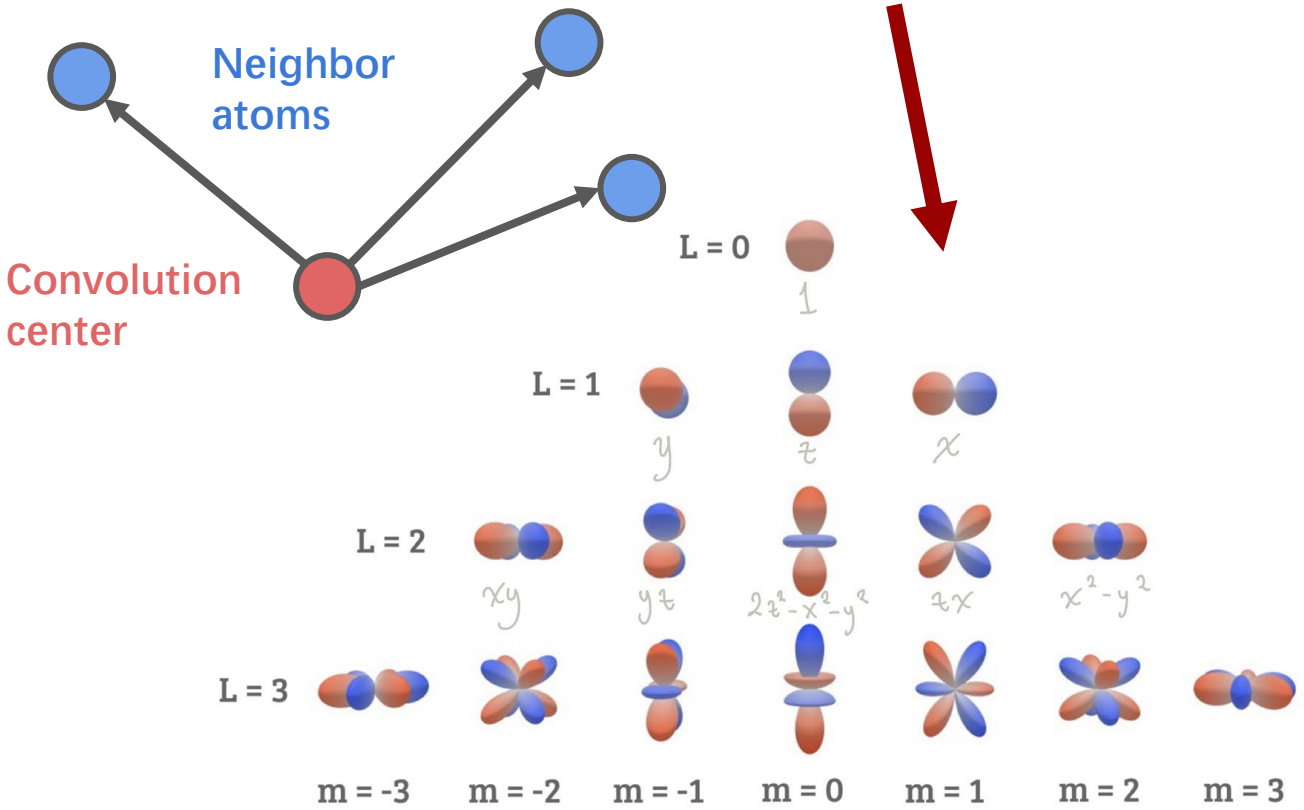
cross-product
antisymmetric
equivariant
L=1
3 degrees of freedom

symmetric
traceless
equivariant
L=2
5 degrees of freedom

Euclidean Neural Networks are similar to convolutional neural networks…

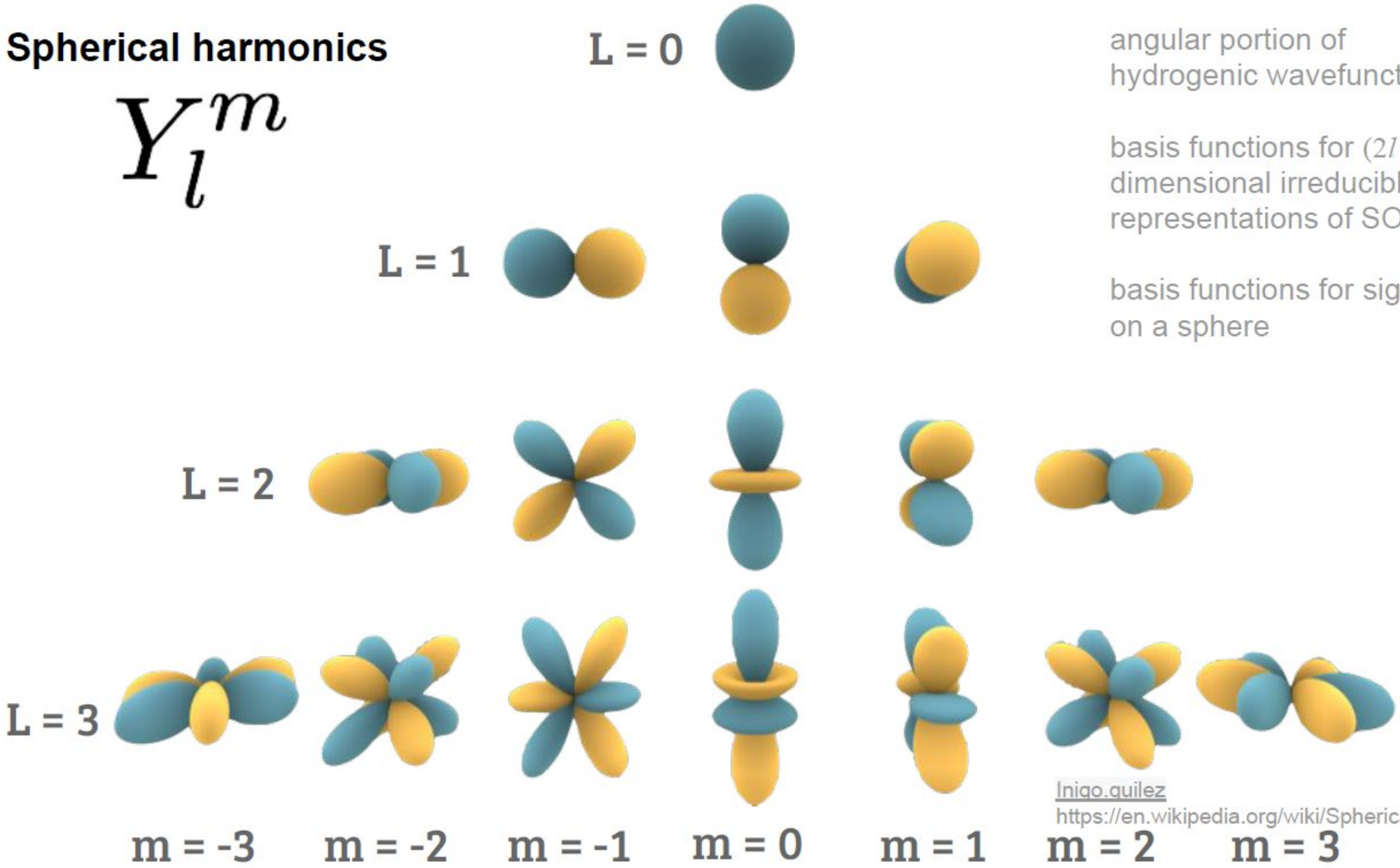
Equivariant convolutional filters are based on learned radial functions and spherical harmonics…

$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



Spherical harmonics

$$Y_l^m$$



angular portion of hydrogenic wavefunctions

basis functions for $(2l + 1)$ dimensional irreducible representations of SO(3)

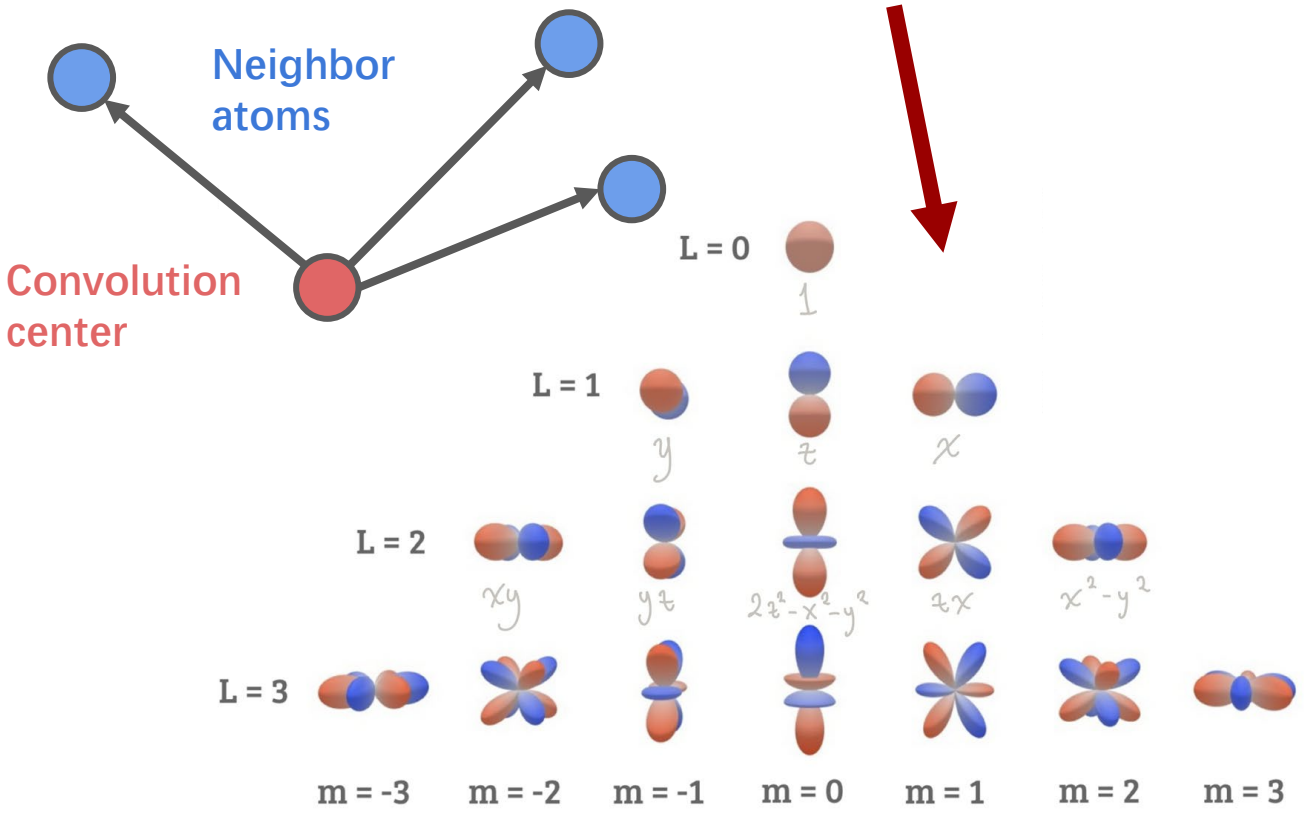
basis functions for signals on a sphere

Inigo.quilez
https://en.wikipedia.org/wiki/Spherical_harmonics

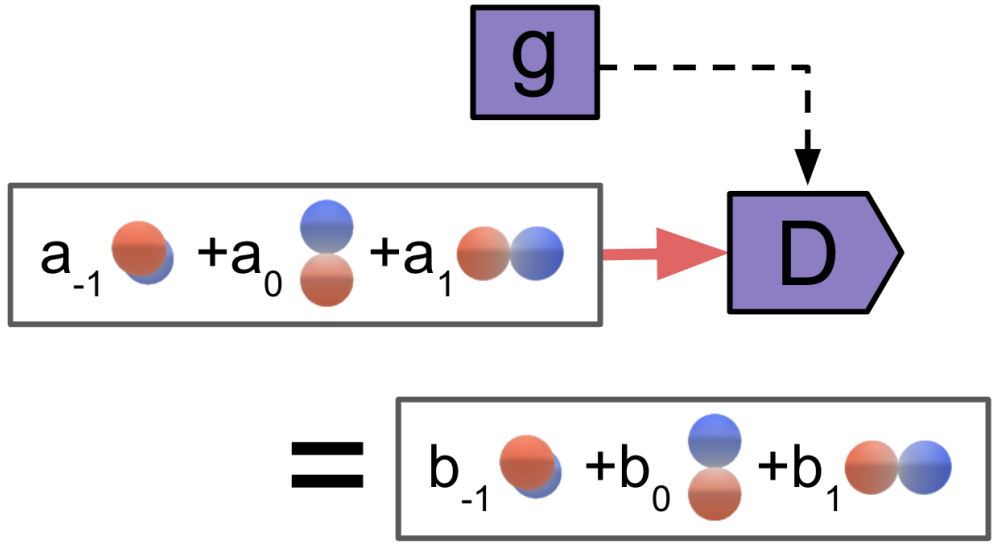
Euclidean Neural Networks are similar to convolutional neural networks…

Equivariant convolutional filters are based on learned radial functions and spherical harmonics…

$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



Spherical harmonics of the same L transform together under rotation **g**.

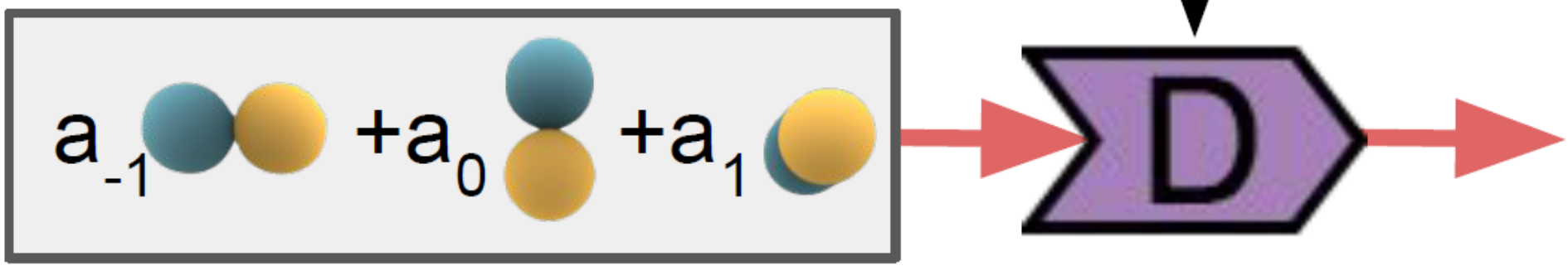


Spherical harmonics transform in the same manner as the irreducible representations of SO(3).

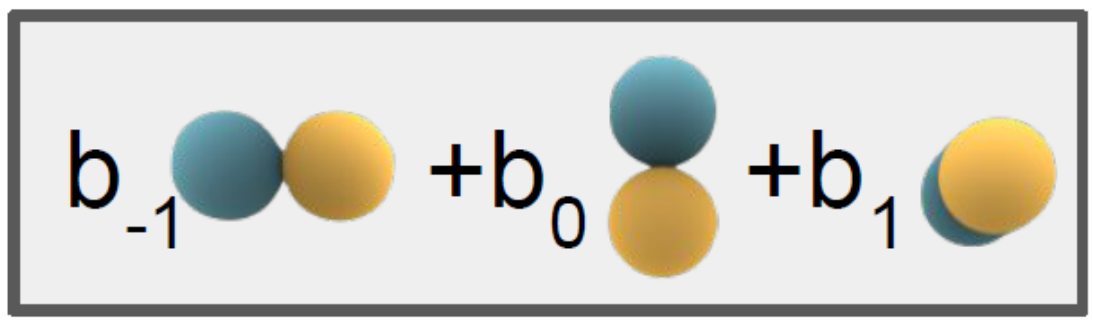
Spherical harmonics of a given L transform together under rotation.

Let **g** be a 3d rotation matrix.

D is the Wigner-D matrix.
It has shape $[2l+1, 2l+1]$
and is a function of **g**.



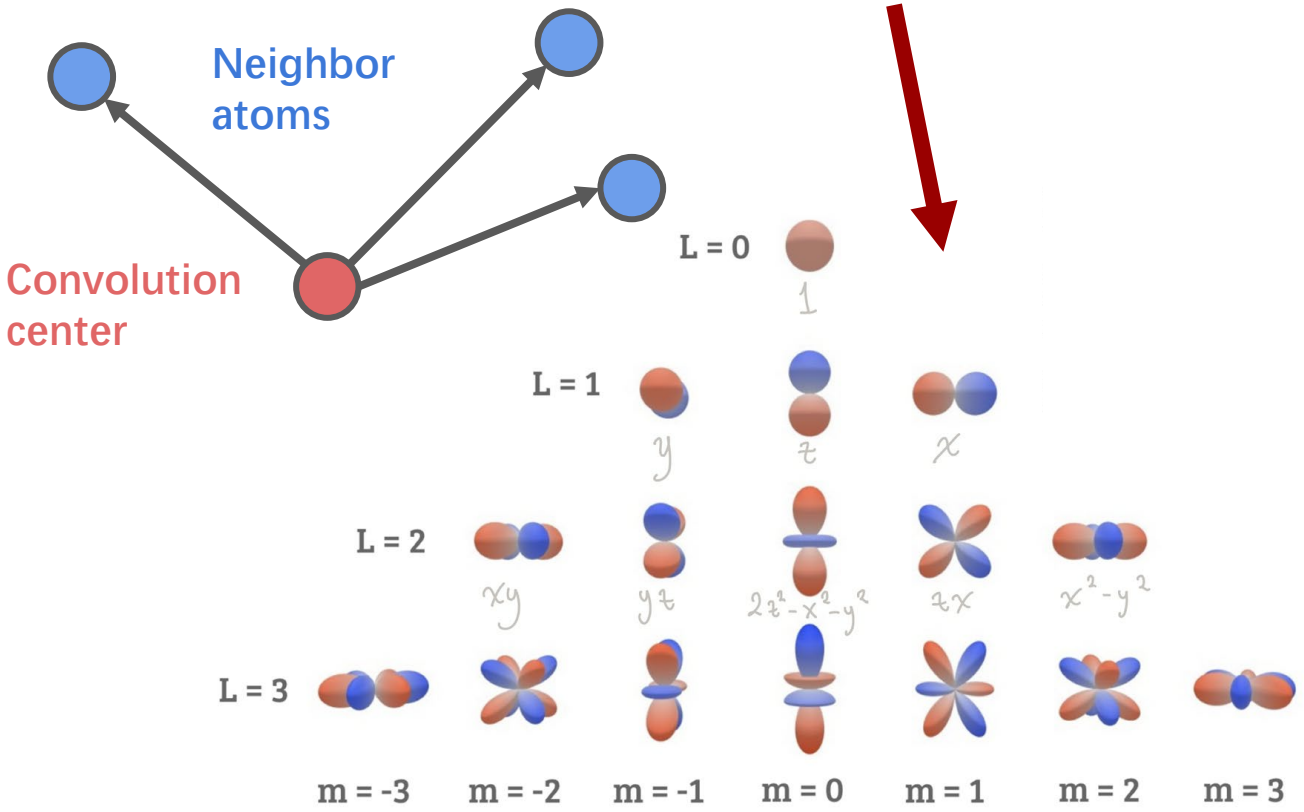
=



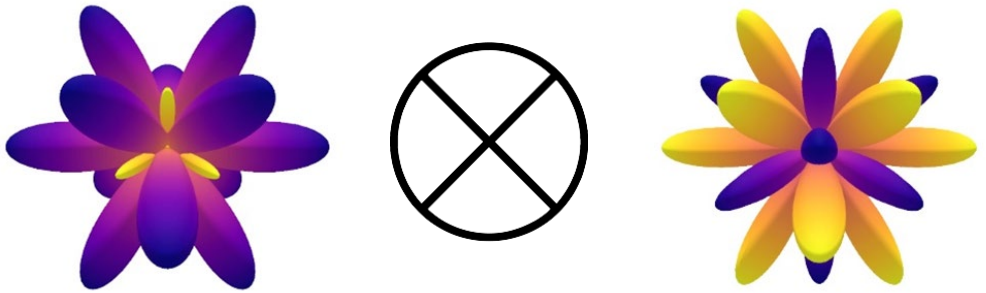
Euclidean Neural Networks are similar to convolutional neural networks...

Equivariant convolutional filters are based on learned radial functions and spherical harmonics...

$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



...and geometric tensor algebra allow us to generalize scalar operations to more complex geometric tensors.



e.g. How to multiply two vectors?

$$\vec{a} \cdot \vec{b} = c$$ scalar

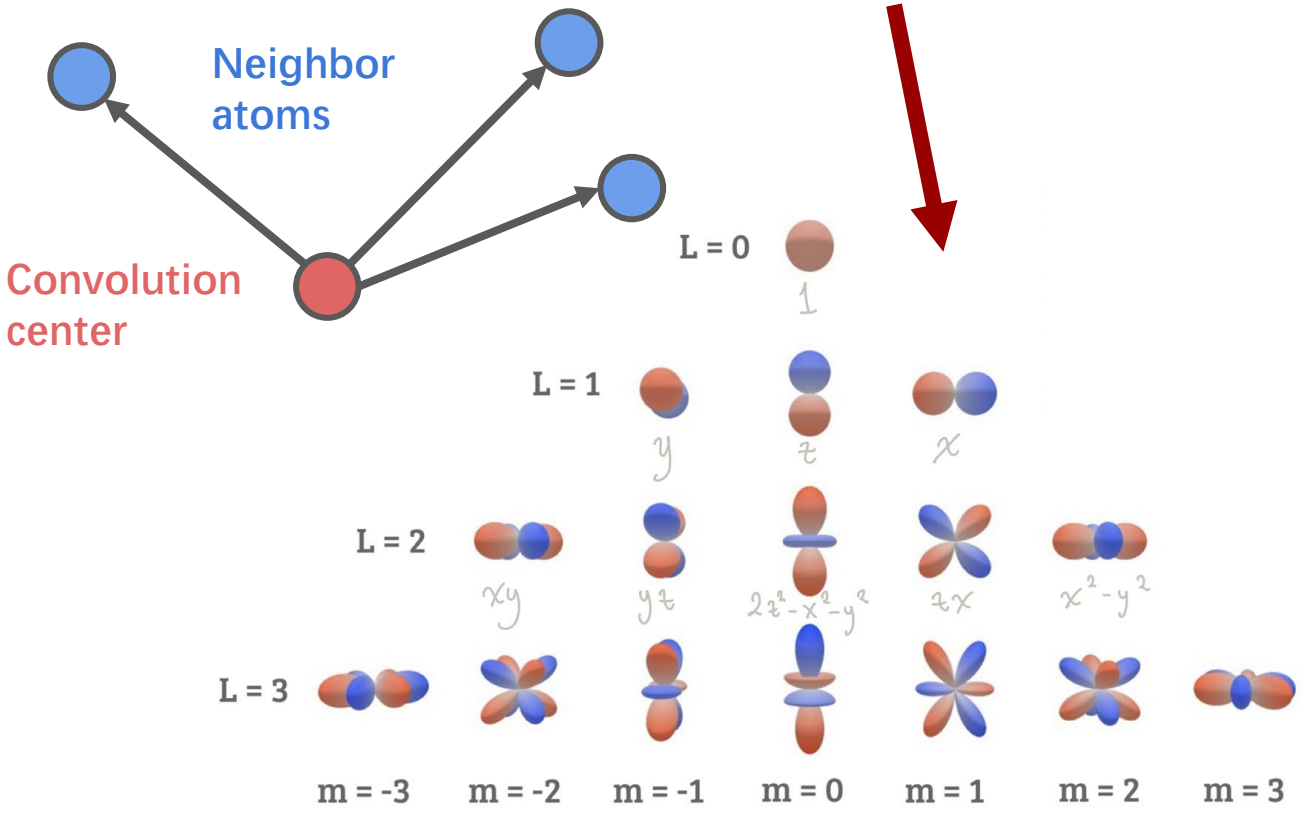
$$\vec{a} \times \vec{b} = \vec{c}$$ vector

$$\vec{a} \otimes \vec{b} = C$$ 3x3 matrix

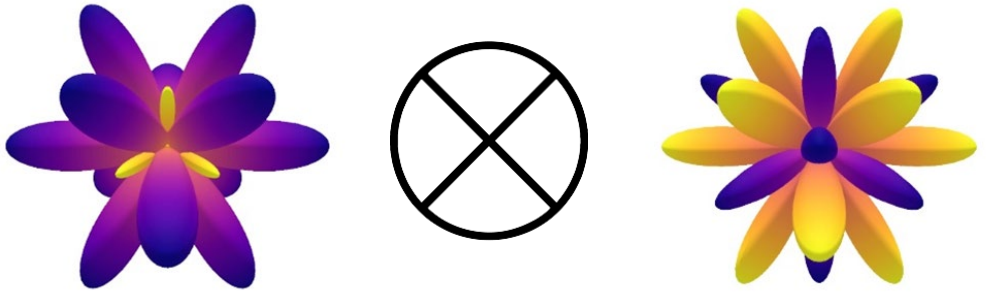
Euclidean Neural Networks are similar to convolutional neural networks...

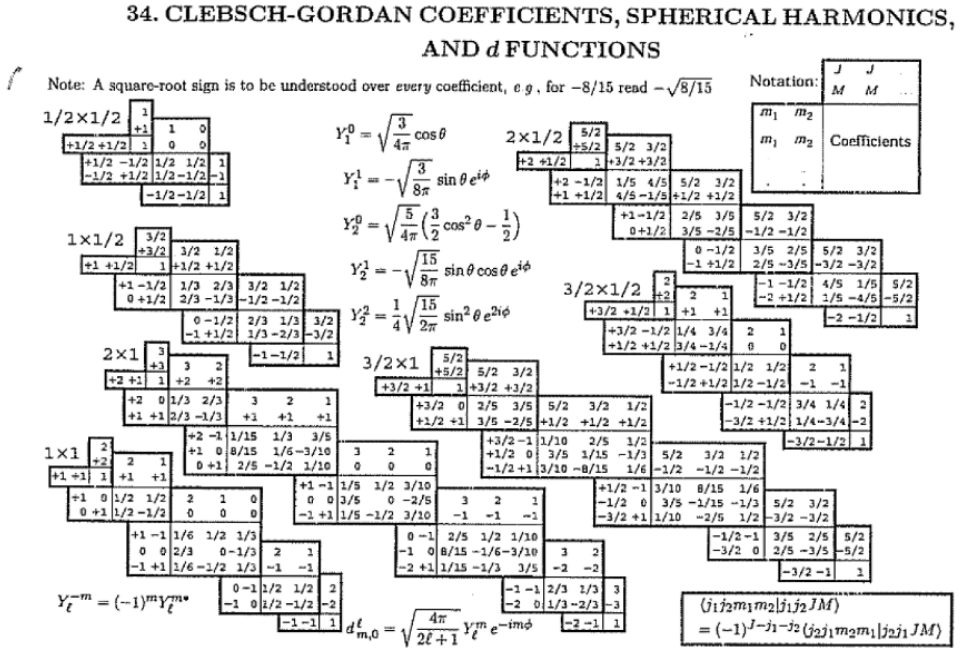
Equivariant convolutional filters are based on learned radial functions and spherical harmonics...

$$W(\vec{r}) = R(r)Y_l^m(\hat{r})$$



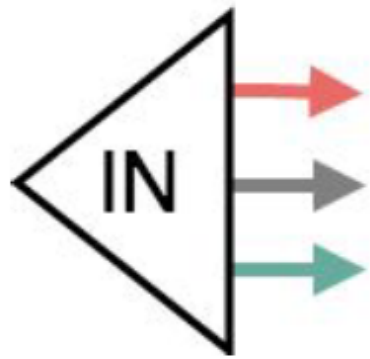
...and geometric tensor algebra allow us to generalize scalar operations to more complex geometric tensors.





The input and output of our network is represented as tensors with **point** (or atom), **channel**, and **representation** indices organized by irreducible representation.

$$V^{(l)}_{acm} =$$



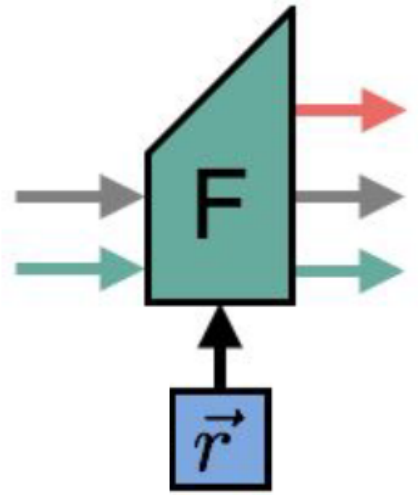
Representation →

Points →

Channels →

Filters contribute a **representation** index due to use of spherical harmonics.

$$R(r)Y_l^m(\hat{r})$$



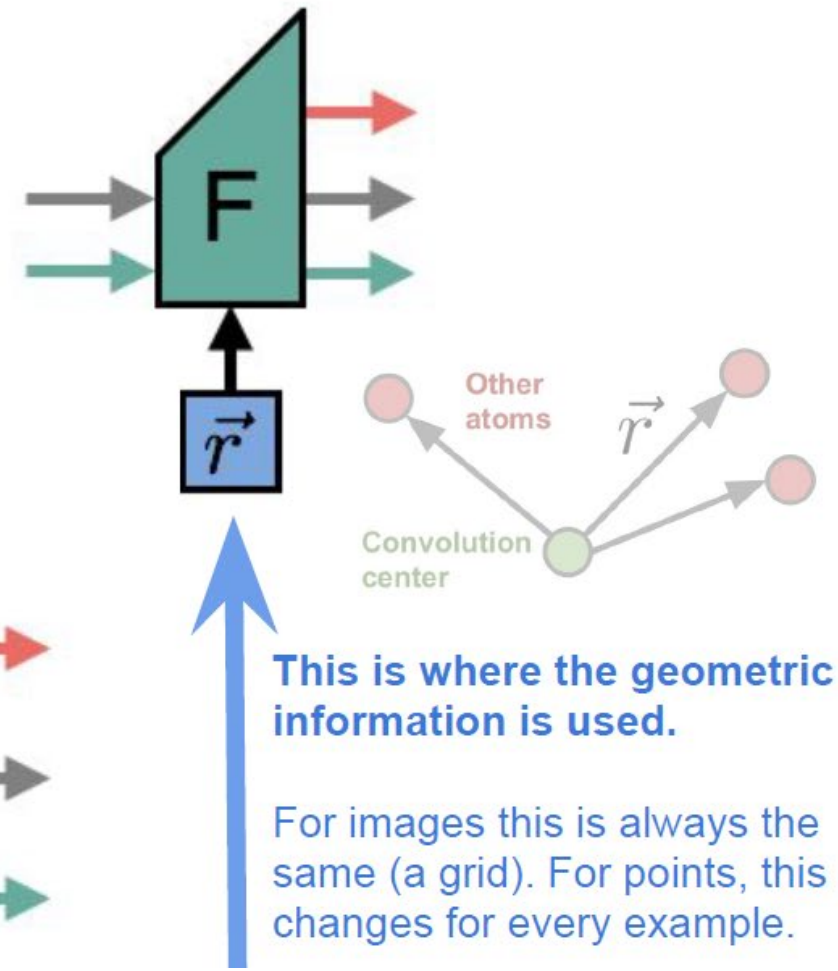
Representation →

Points →

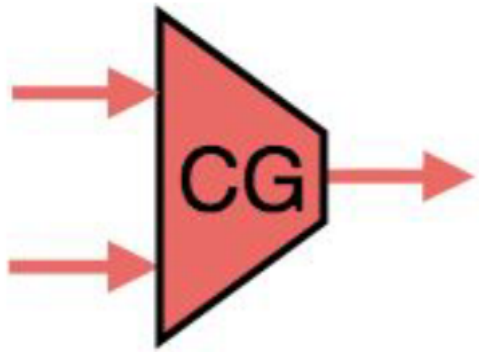
Channels →

Filters contribute a **representation** index due to use of spherical harmonics.

$$R(r)Y_l^m(\hat{r})$$



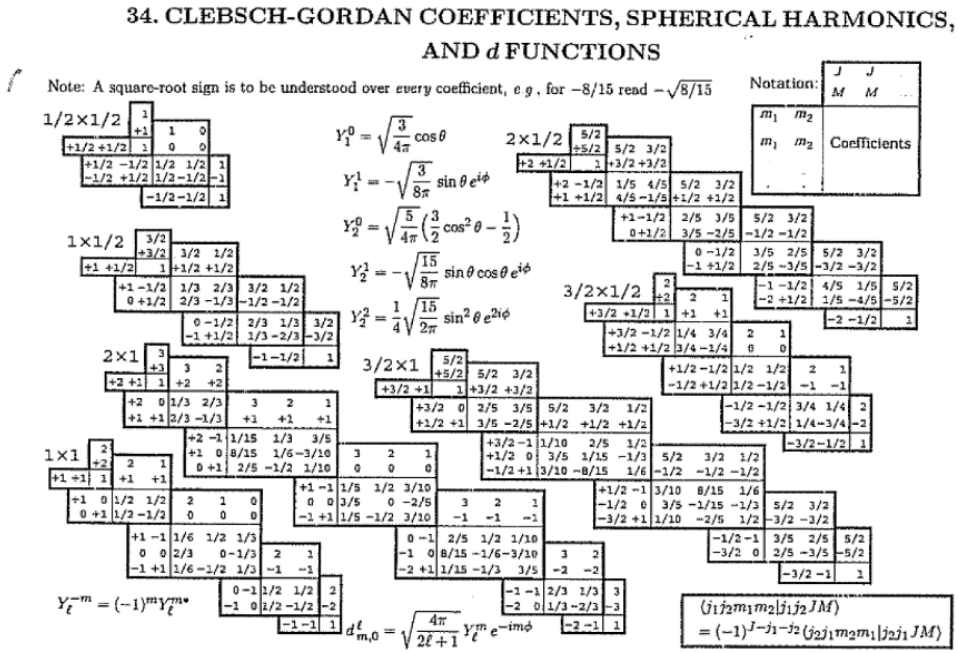
To combine two tensors to create one tensor, we uses Clebsch-Gordan coefficients.



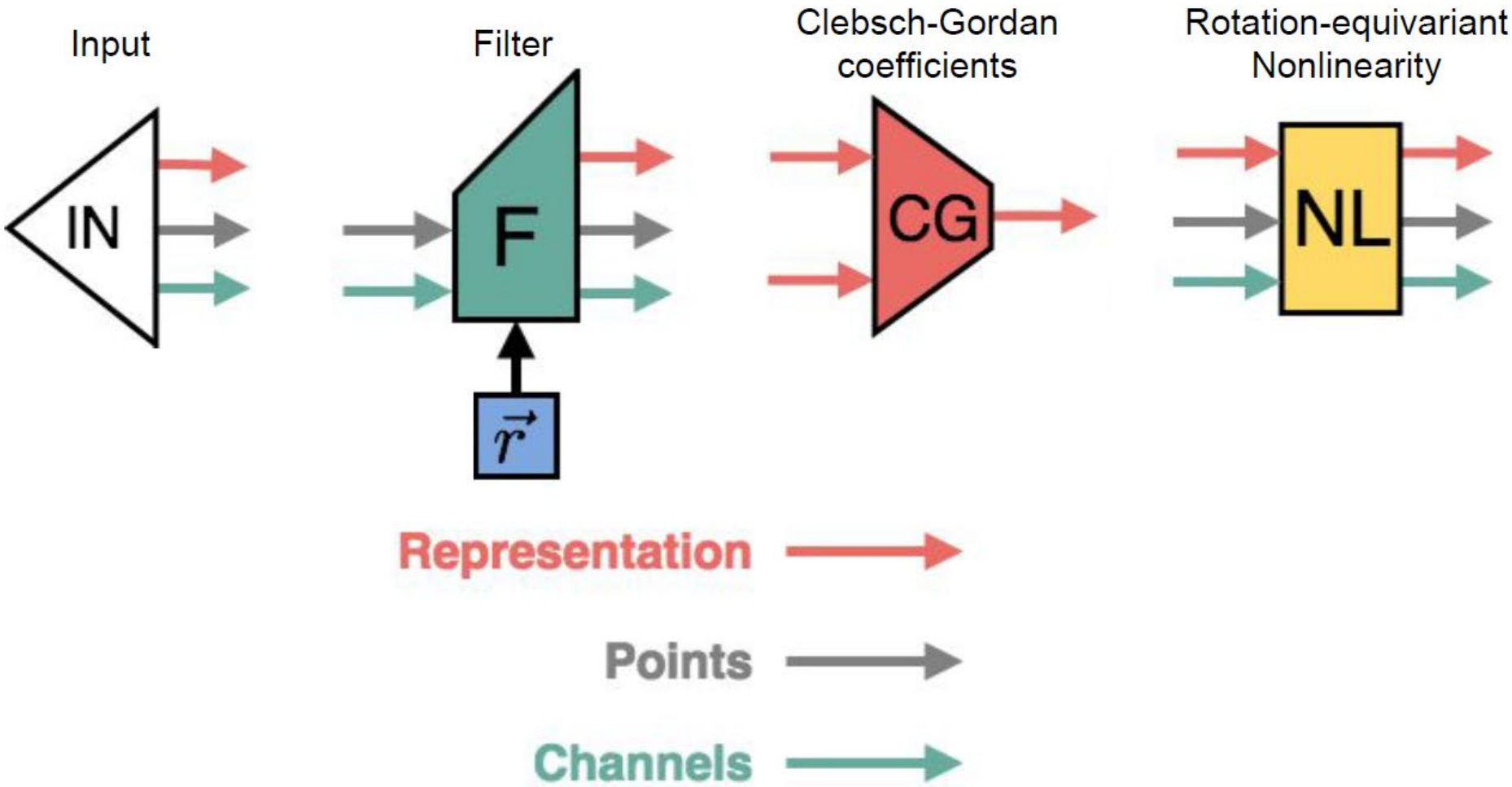
Representation →

Points →

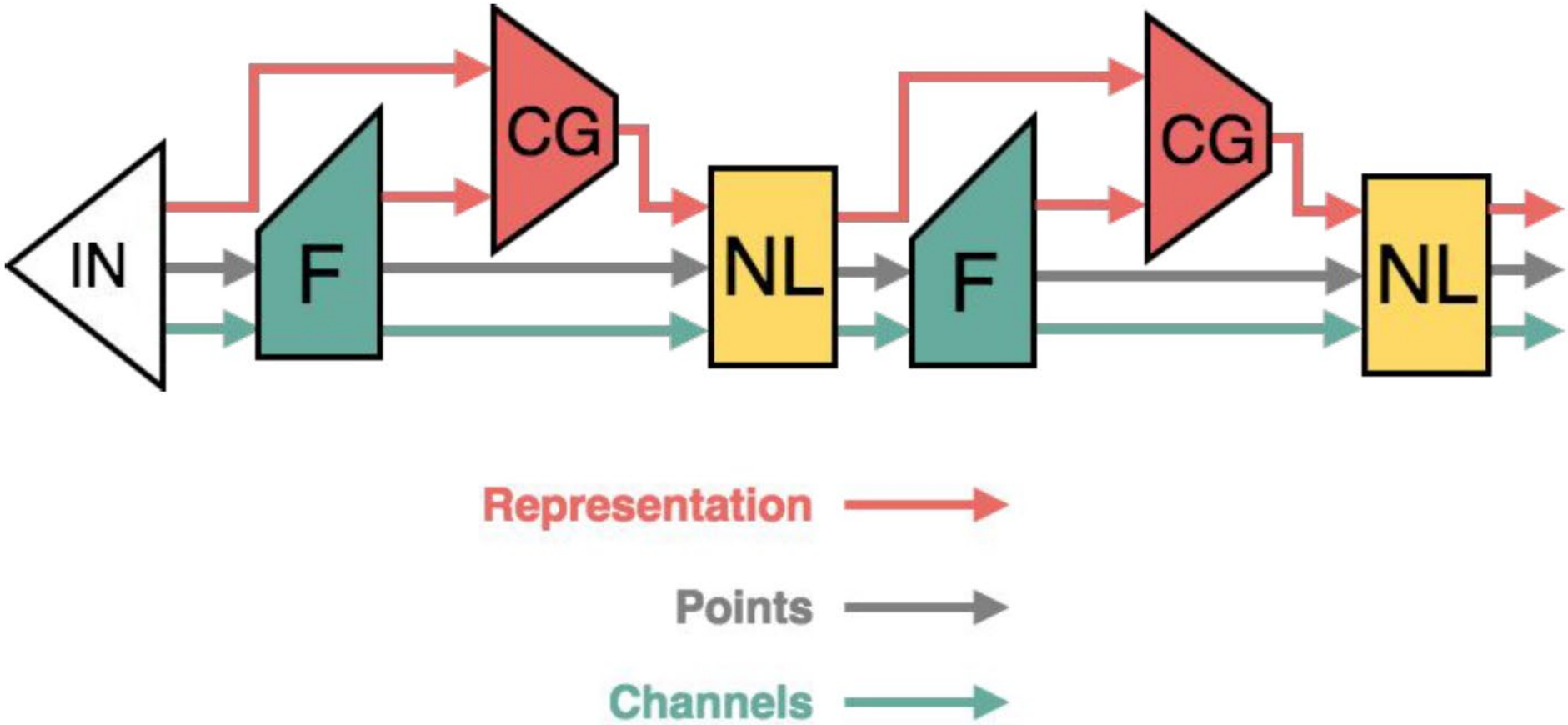
Channels →



These are components of **tensor field networks**



This is what a two-layer tensor field network looks like:



… and this is what one layer (without NL) looks like with all the indices written out:

$$\mathcal{L}^{(l_O)}_{acm_O}\left(\vec{r}_a, V^{(l_I)}_{acm_I}\right) := \sum_{m_F, m_I} C^{(l_O,m_O)}_{(l_F,m_F)(l_I,m_I)} \sum_{b \in S} F^{(l_F,l_I)}_{cm_F}(\vec{r}_{ab}) V^{(l_I)}_{bcm_I}$$

… and this is what one layer (without NL) looks like with all the indices written out:

$$\mathcal{L}^{(l_O)}_{acm_O}\left(\vec{r}_a, V^{(l_I)}_{acm_I}\right) := \sum_{m_F,m_I} C^{(l_O,m_O)}_{(l_F,m_F)(l_I,m_I)} \sum_{b\in S} F^{(l_F,l_I)}_{cm_F}(\vec{r}_{ab}) V^{(l_I)}_{bcm_I}$$

$$F^{(l_F,l_I)}_{cm_F}(\vec{r}_{ab}) = R^{(l_F,l_I)}_{c}(r_{ab}) Y^{(l_F)}_{m_F}(\hat{r}_{ab})$$

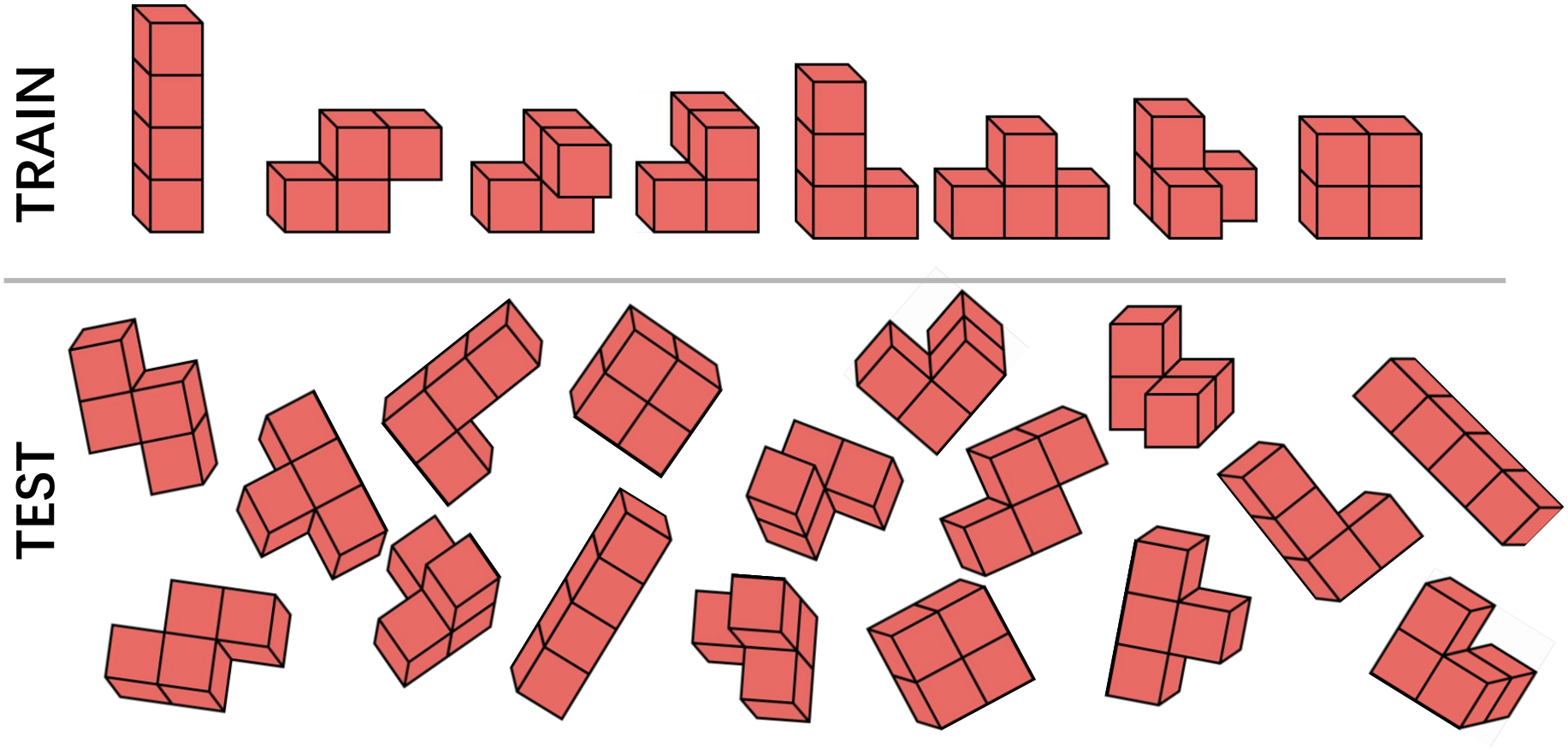
… and this is what one layer (without NL) looks like with all the indices written out:

$$\mathcal{L}^{(l_O)}_{acm_O}\left(\vec{r}_a, V^{(l_I)}_{acm_I}\right) := \sum_{m_F,m_I} C^{(l_O,m_O)}_{(l_F,m_F)(l_I,m_I)} \sum_{b\in S} F^{(l_F,l_I)}_{cm_F}(\vec{r}_{ab}) V^{(l_I)}_{bcm_I}$$

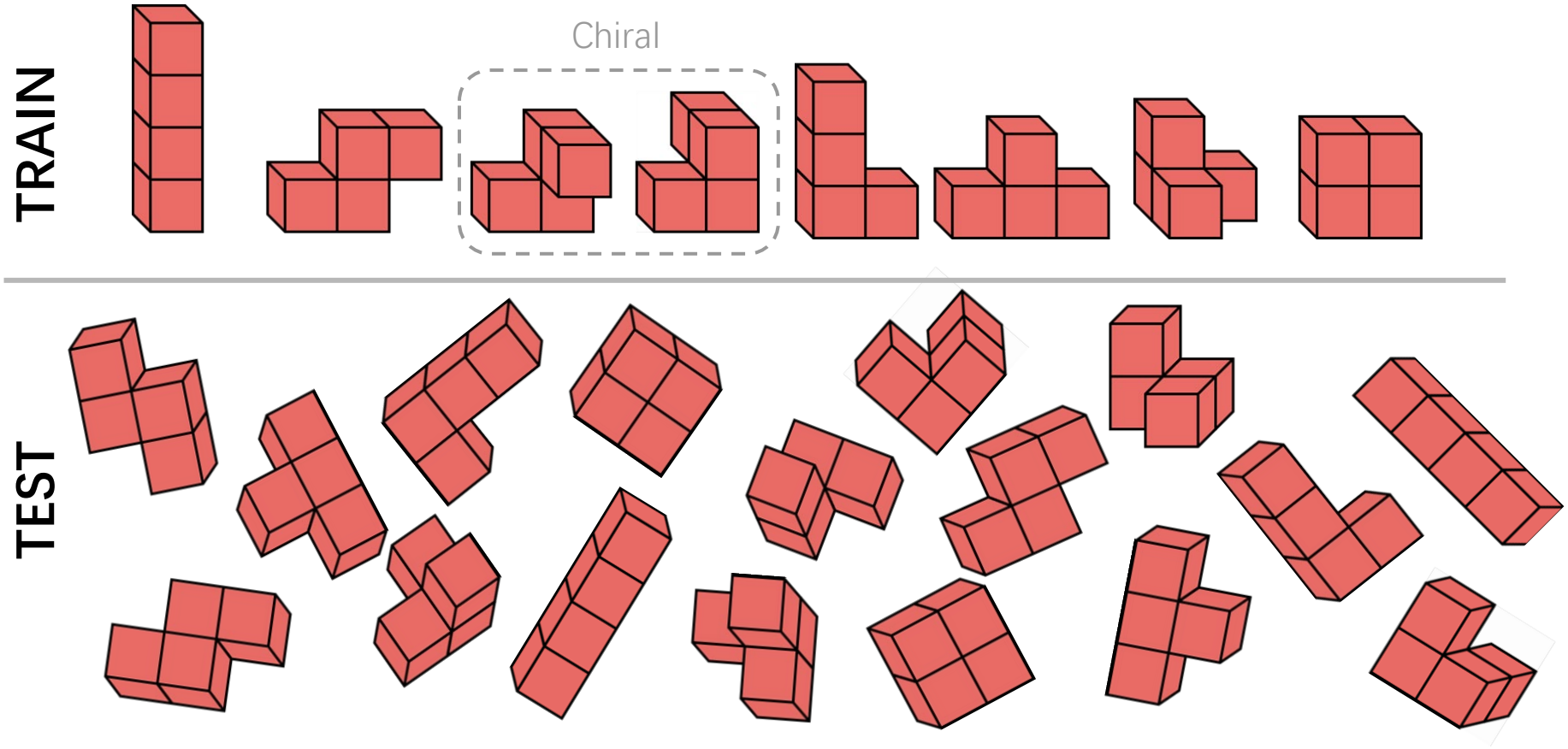
$$F^{(l_F,l_I)}_{cm_F}(\vec{r}_{ab}) = R^{(l_F,l_I)}_{c}(r_{ab}) Y^{(l_F)}_{m_F}(\hat{r}_{ab})$$

$$R^{(l_F,l_I)}_{c}(r_{ab}) = \sum_h W_{ch}\ \mathrm{NL}\Big(\sum_j W_{hj} B_j(r_{ab}) + b_h\Big) + b_c$$

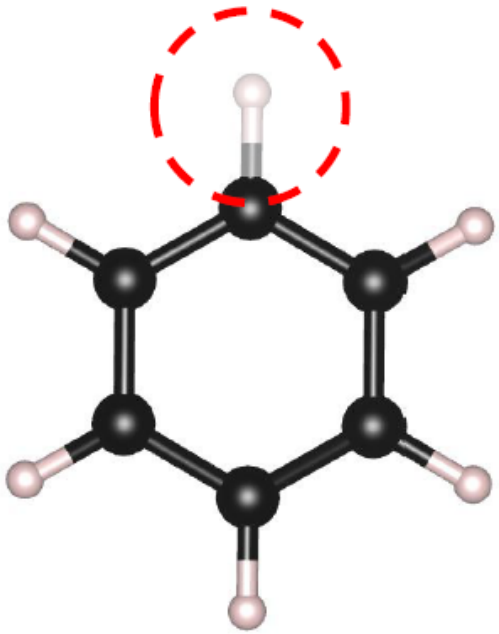
**Our unit test: Trained on 3D Tetris shapes in one orientation,
these network can perfectly identify these shapes in any orientation.**



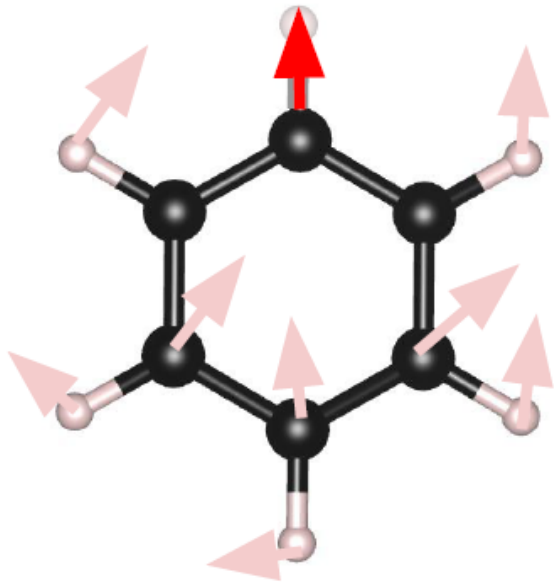
**Our unit test: Trained on 3D Tetris shapes in one orientation,
these network can perfectly identify these shapes in any orientation.**



Given a small organic molecule with an atom removed, replace the correct element at the correct location in space.



Input coordinates with missing atom.



Network outputs
(N-1) atom type features (scalars),
(N-1) displacement vectors, and
(N-1) scalars indicating confidence probability used for "voting".

DATASET
QM9: *http://www.quantum-machine.org/datasets/*
134k molecules with 9 or less heavy atoms (non-hydrogen) and elements H, C, N, O, F.

TRAIN
1,000 molecules with 5-18 atoms

TEST
1,000 molecules with 19 atoms
1,000 molecules with 23 atoms
1,000 molecules with 25-29 atoms

Atoms	Number of predictions	Accuracy (%) ($\leq$ 0.5 Å and atom type)	Distance MAE in Å
5-18 (train)	15 947	92.6	0.16
19	19 000	94.7	0.15
23	23 000	96.9	0.14
25-29	25 404	97.8	0.17

Learns to replace atoms with over 90% accuracy across train and test by seeing the same 1,000 molecules 200 times.