Particle Trajectory Reconstruction and Euclidian Equivariant Neural Networks

Omar Alterkait

Neutrino Physics and Machine Learning

8/22/23

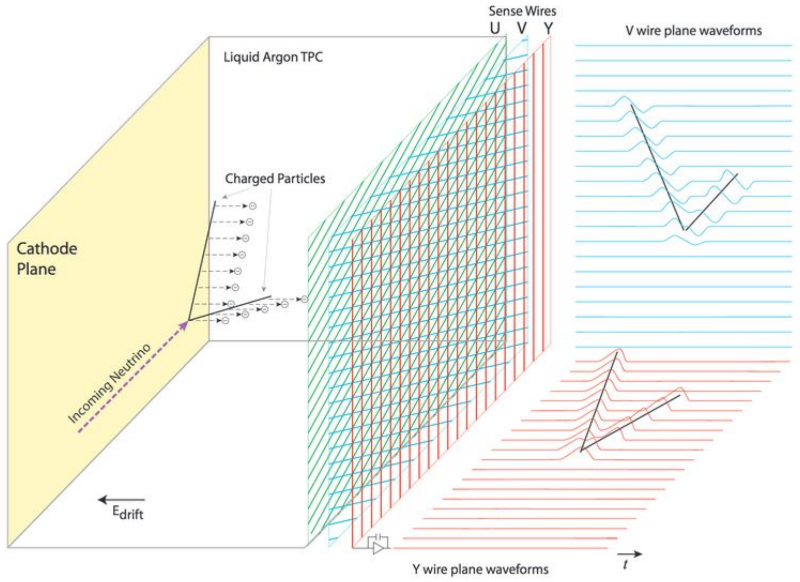
Outline

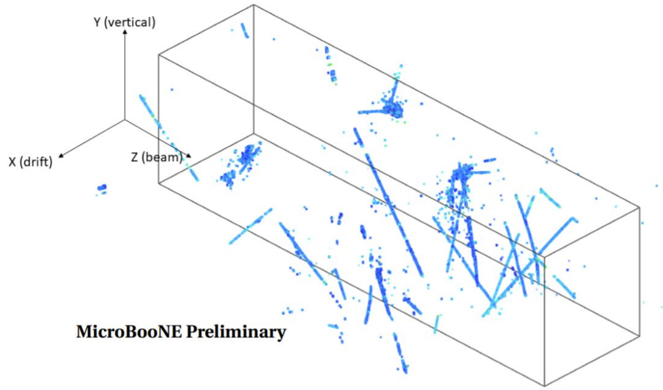
- **Introduce Track Reconstruction**
- **Start by giving a background on equivariant neural networks**
  - What is equivariance? Why is it useful?
- **Discuss Sparse Euclidean Equivairant CNNs**
  - What parts differ from normal CNNs?
- **Conclude about the next parts of the project**

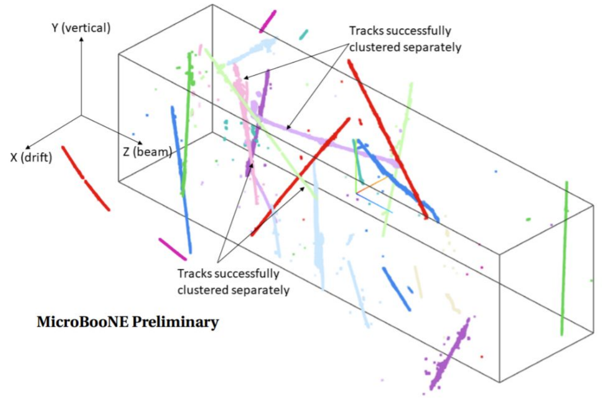
Track Reconstruction

LArTPC Track Reconstruction

- As particles move through the detector, they leave charge behind.
- This charge is detected by wire planes
- Current ML/ non-ML tools exist to go from 2D to 3D.
- Following that, we would like to reconstruct, segment, and classify all the particles in the event.

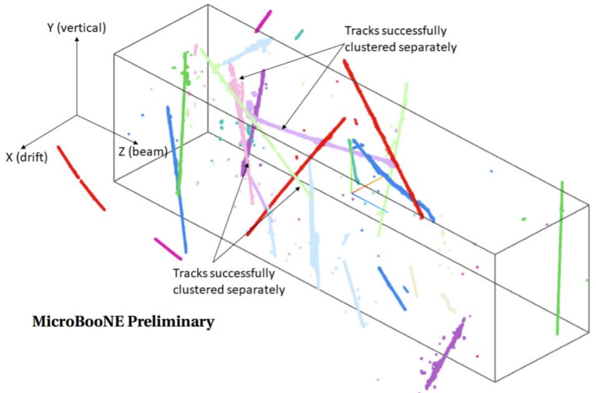




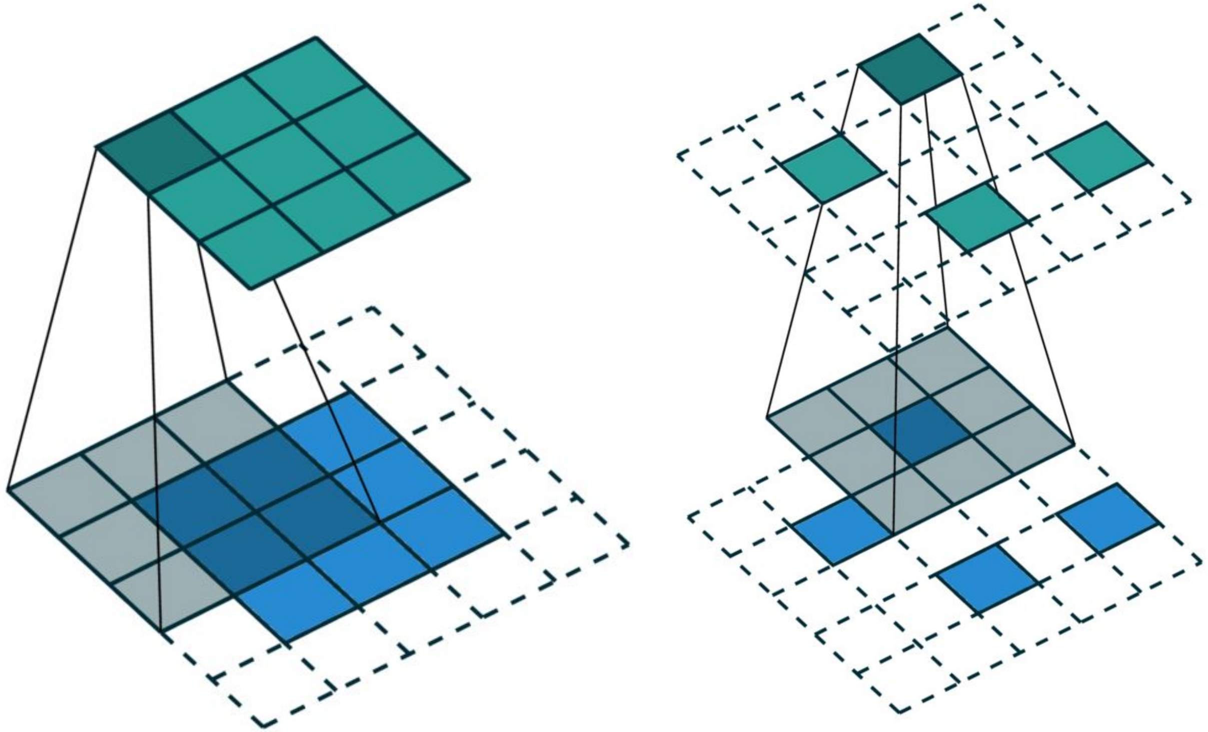


MICROBOONE-NOTE-1040-PUB

Submanifold Convolutions



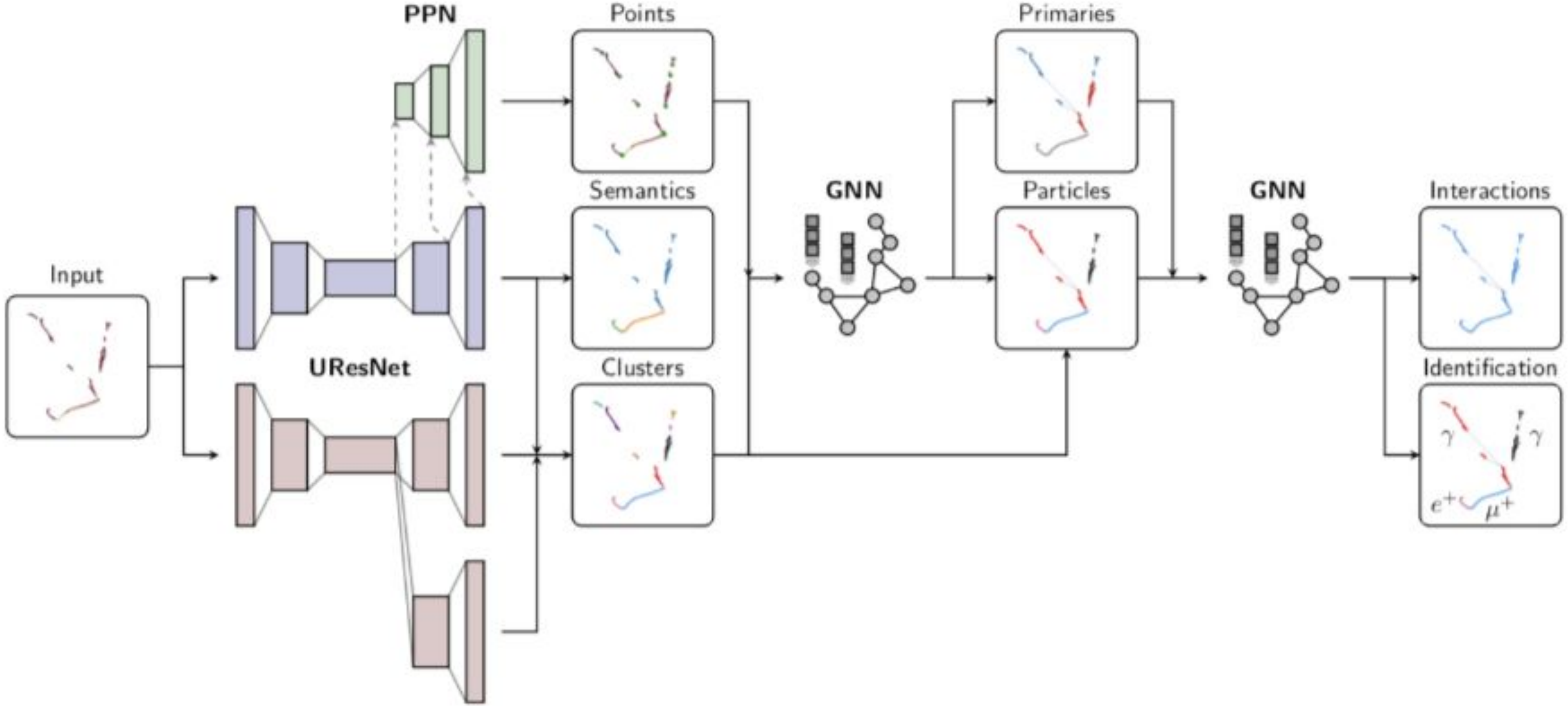
- For sparse but locally dense data, we can reduce computation by using submanifold convolutions
- This skips all empty spots during training/inference
- A package by NVIDIA called MinkowskiEngine is already employed in ML reconstruction toolchains



MinkowskiEngine

Reconstruction Pipeline

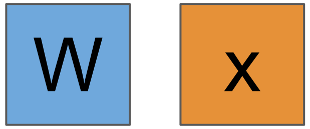
- Current ML tools are used for end-to-end reconstruction.
- These models are big and slightly difficult to train.
- Could we make them more efficient?



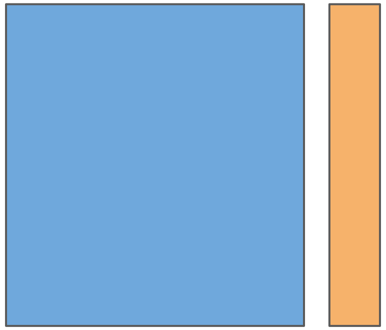
DeepLearnPhysics/lartpc_mlreco3d

Equivariant Neural Networks

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

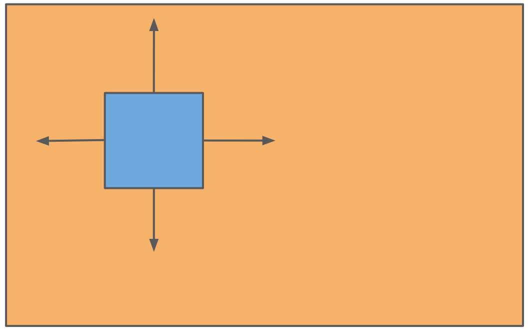


Arrays ➪ Dense NN



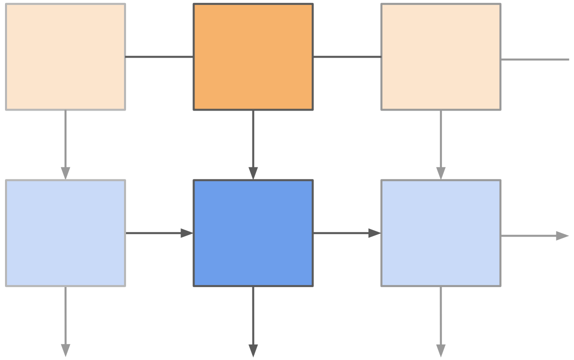
Components are independent.

2D images ➪ Convolutional NN



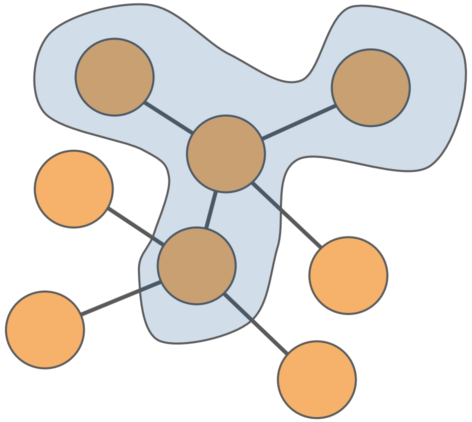
The same features can be found anywhere in an image. Locality.

Text ➪ Recurrent NN



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

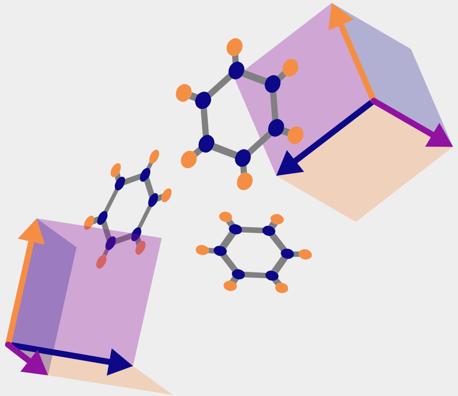
Graph ➪ Graph (Conv.) NN



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

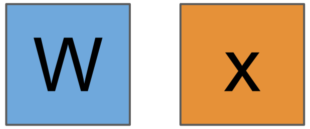
3D physical data
➪ Euclidean NN

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

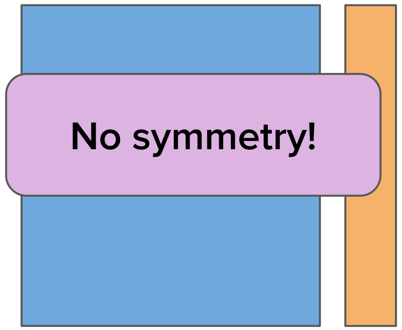


Smidt - e3nn

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

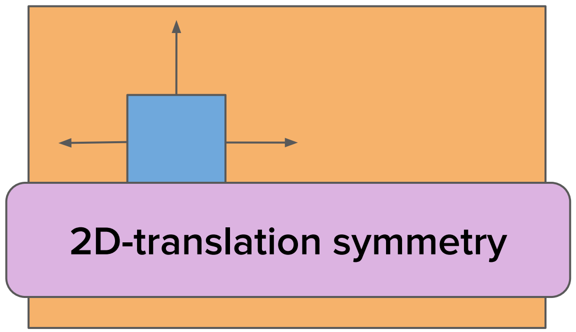


Arrays ➪ Dense NN



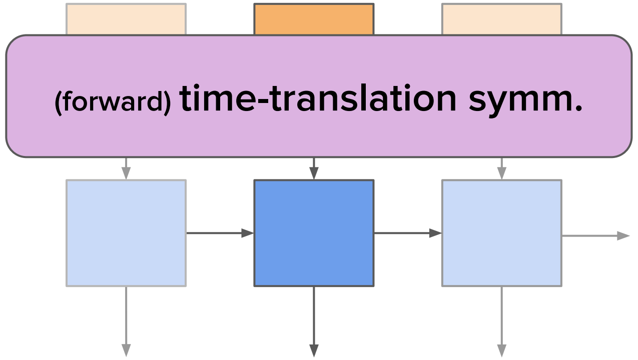
Components are independent.

2D images ➪ Convolutional NN



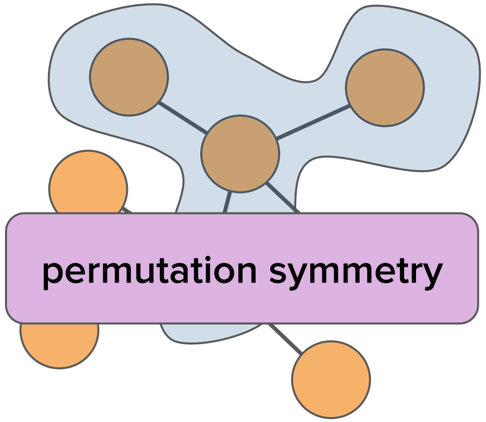
The same features can be found anywhere in an image. Locality.

Text ➪ Recurrent NN



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

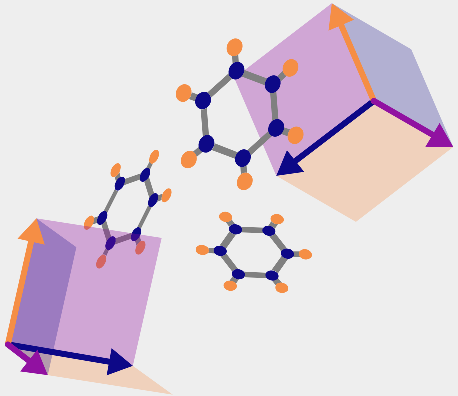
Graph ➪ Graph (Conv.) NN



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

3D physical data
➪ Euclidean NN

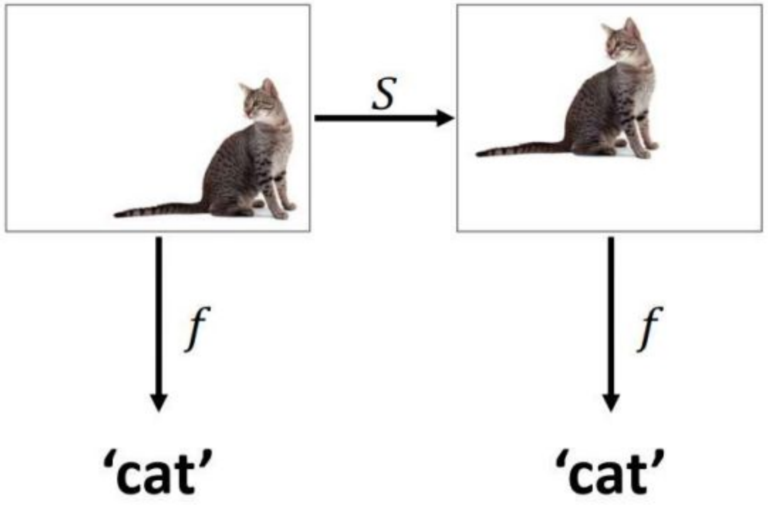
3D Euclidean symmetry E(3): 3D rotations, translations, and inversion



[Smidt - e3nn]

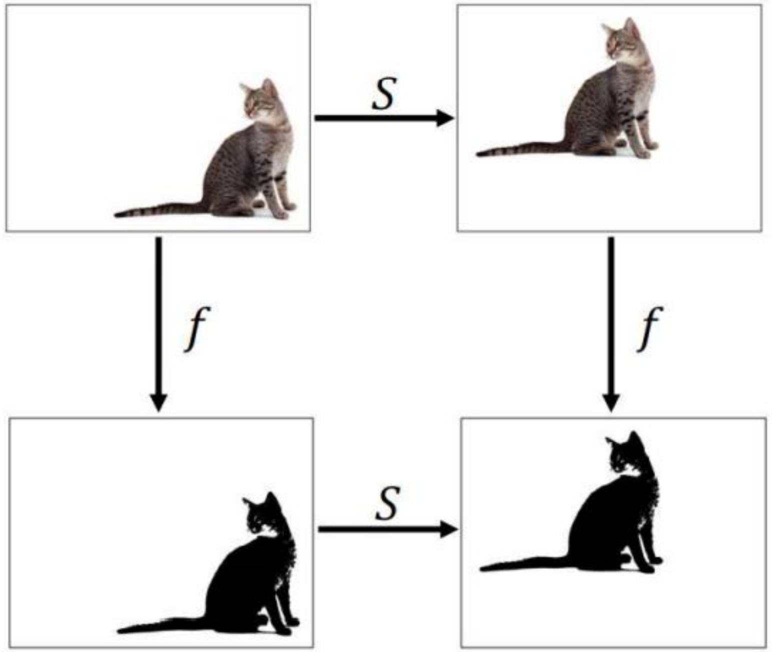
Invariance vs Equivariance

Invariance



$$f(x) = f(Sx)$$

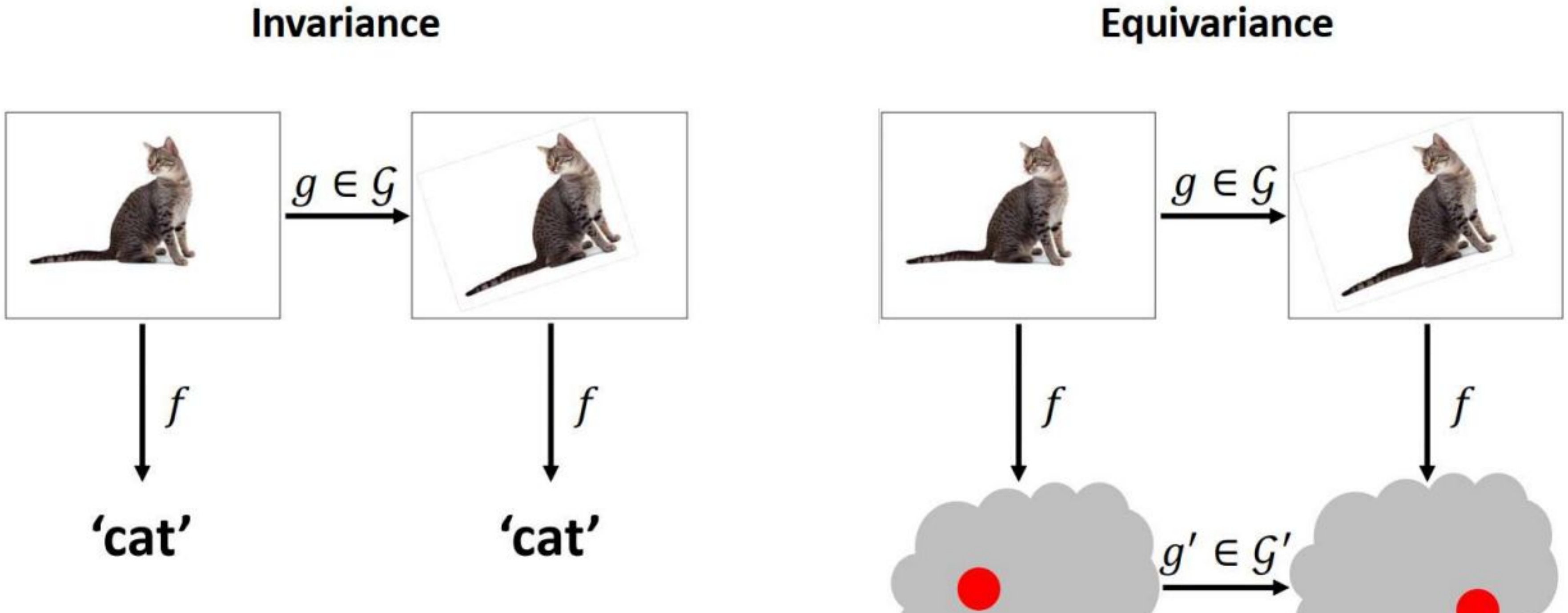
Equivariance



$$Sf(x) = f(Sx)$$

[Kainz - DL Notes]

Invariance vs Equivariance



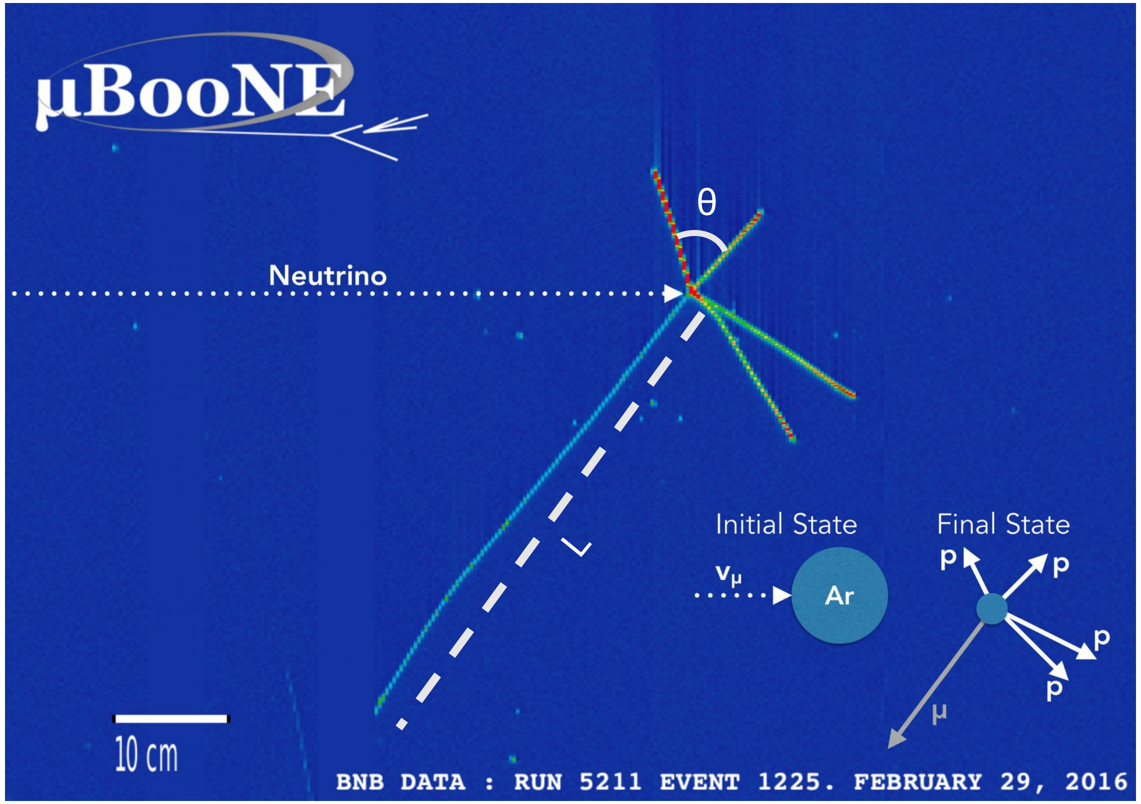
$$f(x) = f(gx)$$

$$g' f(x) = f(gx)$$

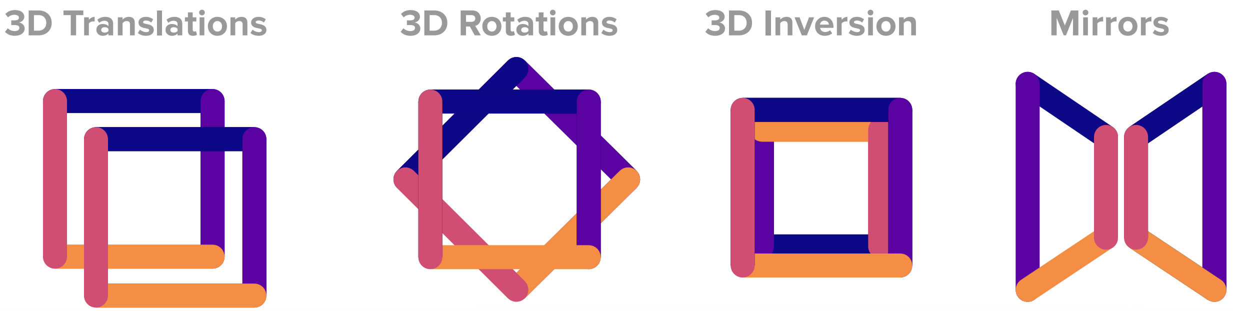
[Kainz - DL Notes]

Making Models Symmetry-Aware

- Approach 1: Data Augmentation
  - Brute Force
- Approach 2: Invariant Models
  - Most current models
- Approach 3: Equivariant Models
  - Goal of today's talk



Euclidean Transformations

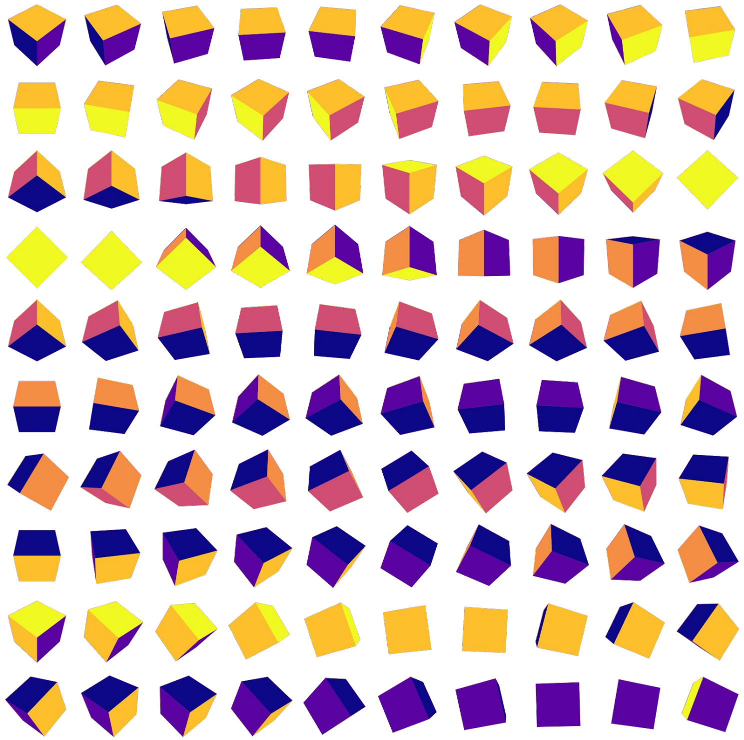


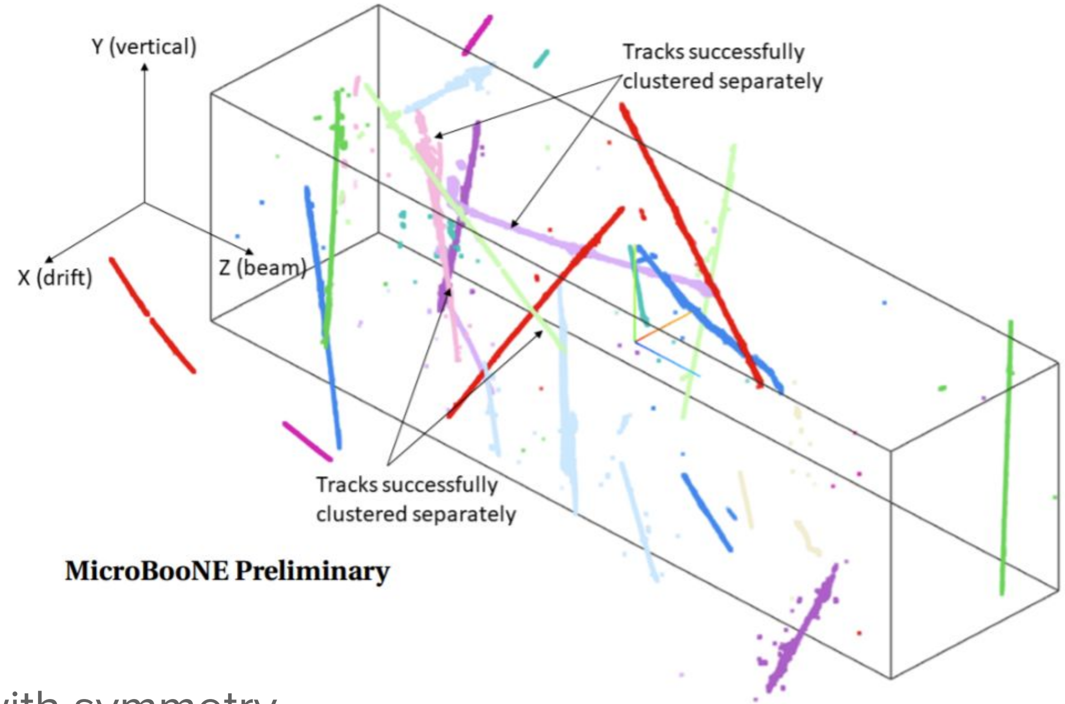
[MicroBooNE Article]

Approach 1: Data Augmentation

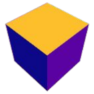
~500 Fold increase in training

training without rotational symmetry





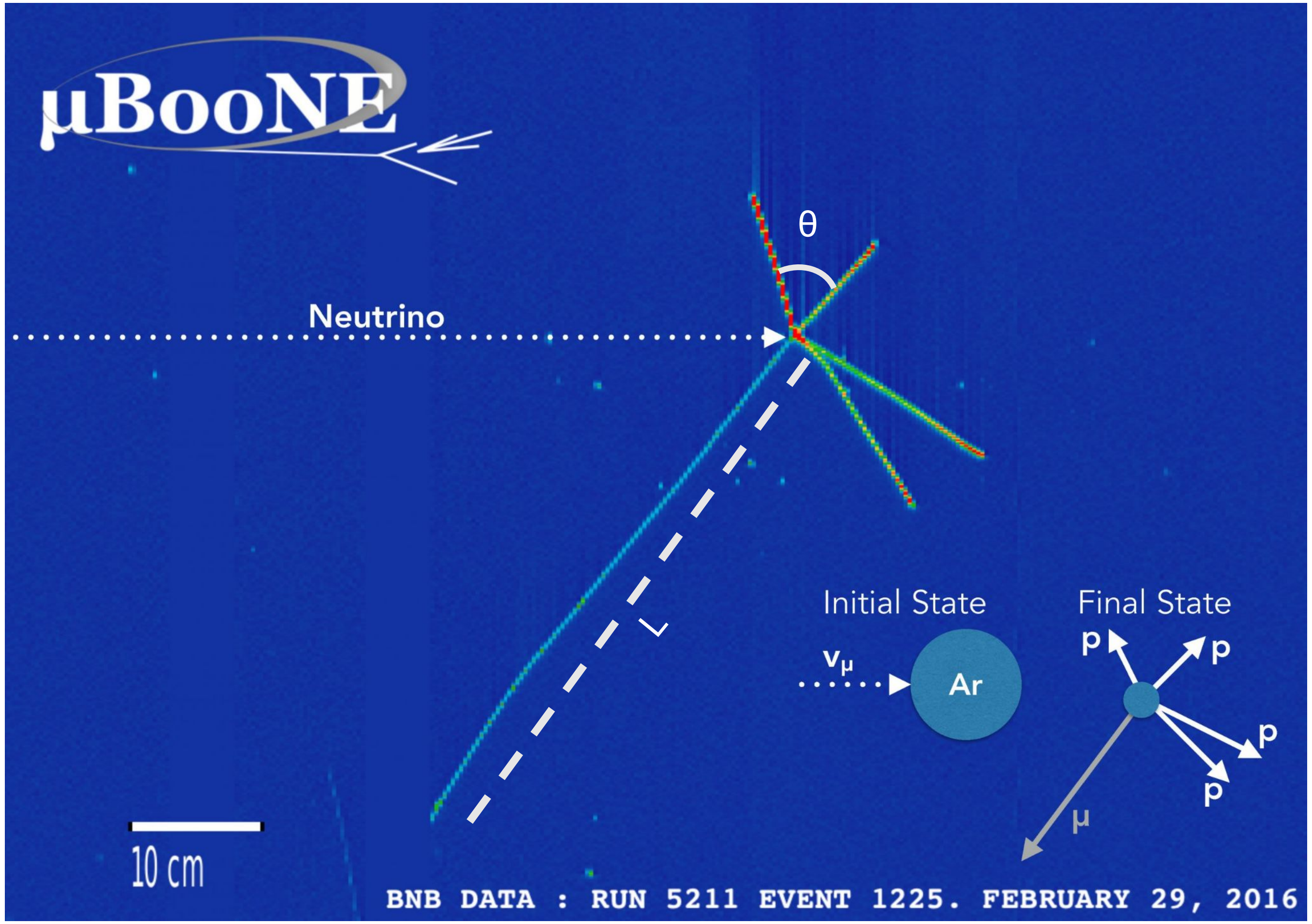
training with symmetry



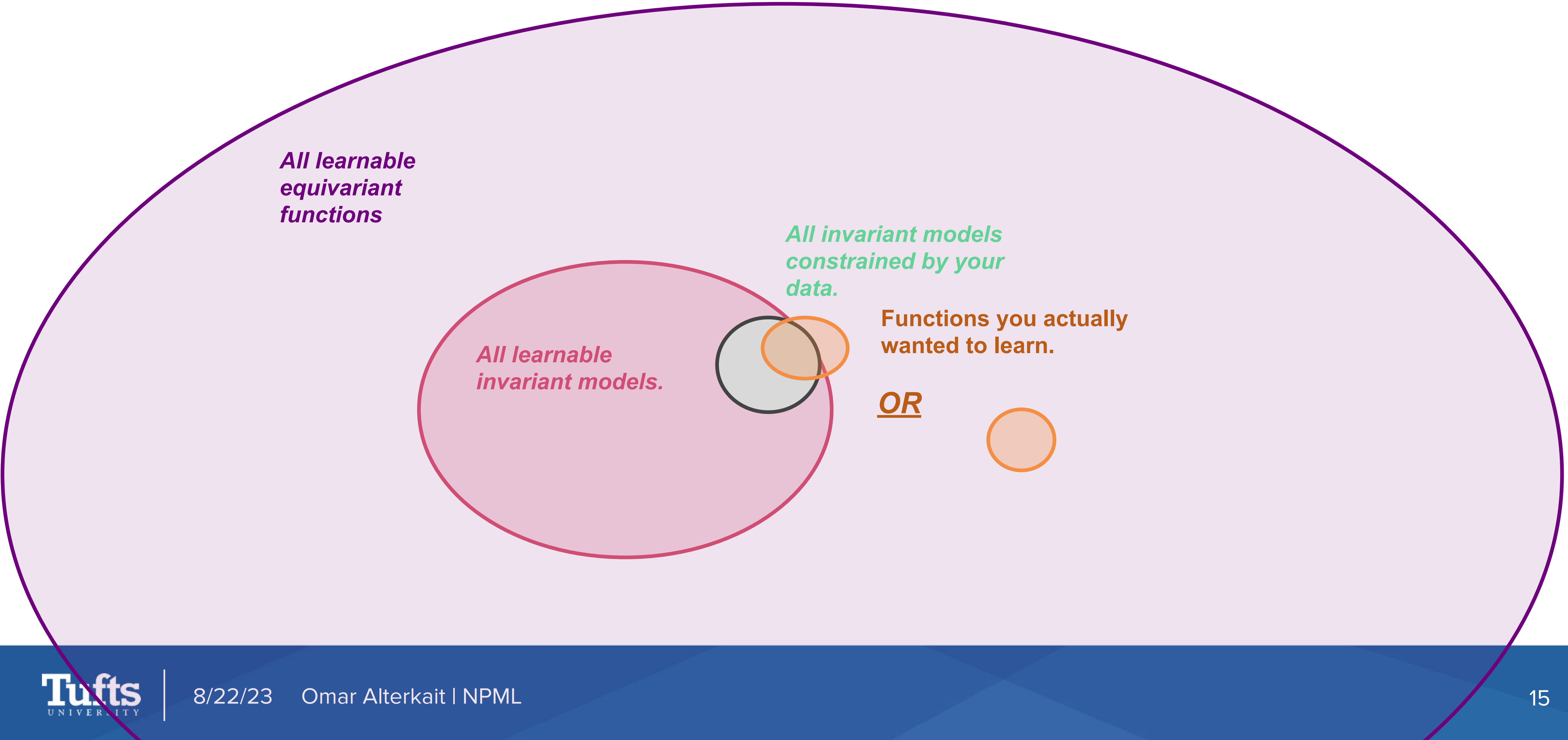
[Smidt - e3nn]

Approach 2: Invariant Models

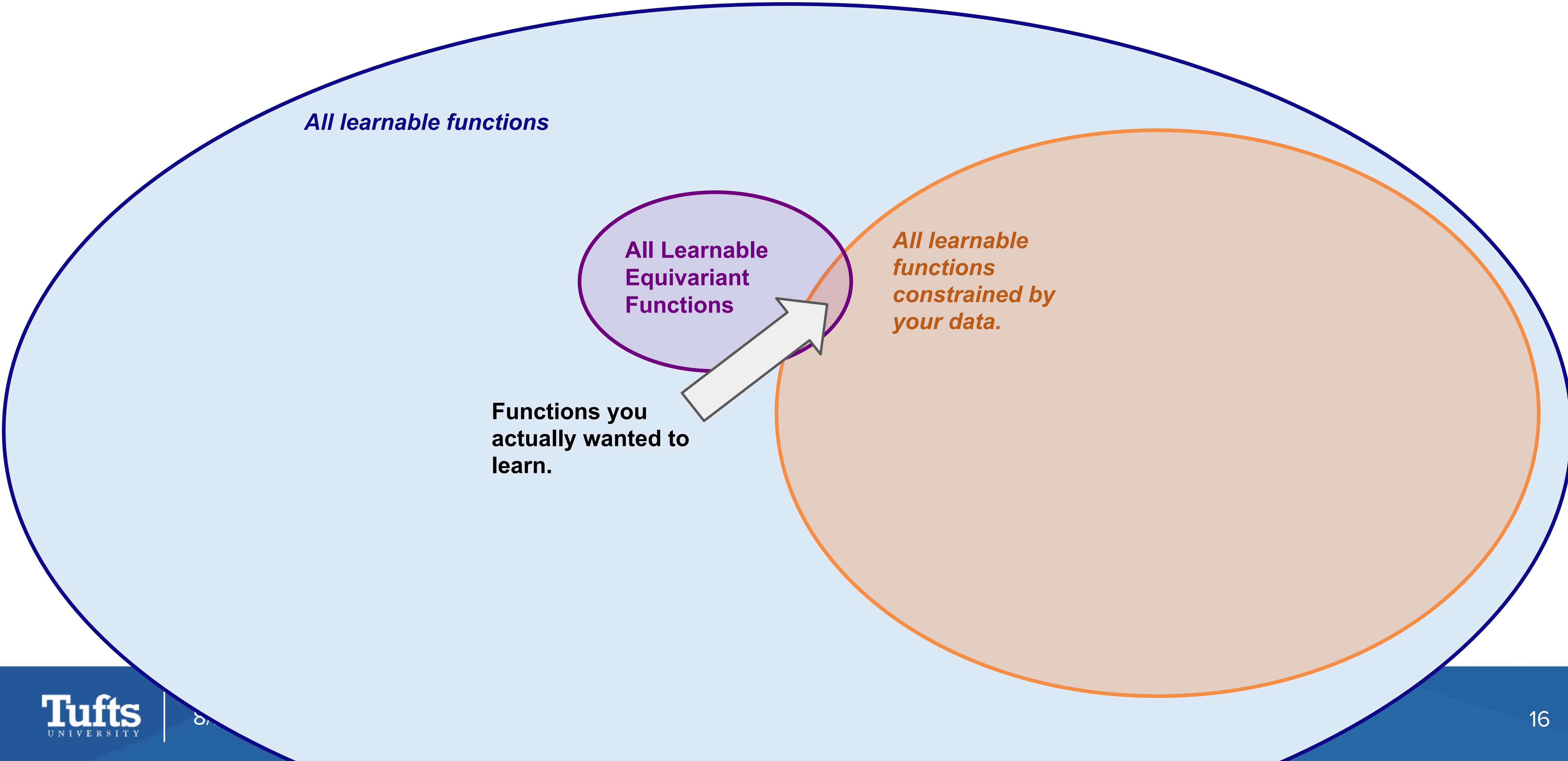
Invariant models pre-compute invariant features and throw away the coordinate system. Equivariant models keep the coordinate system AND if the coordinate system changes, the outputs change accordingly.



if you only use invariant models (only use scalar multiplication),
you have to guarantee that your input features already
contain any necessary equivariant interactions (e.g. cross-products).

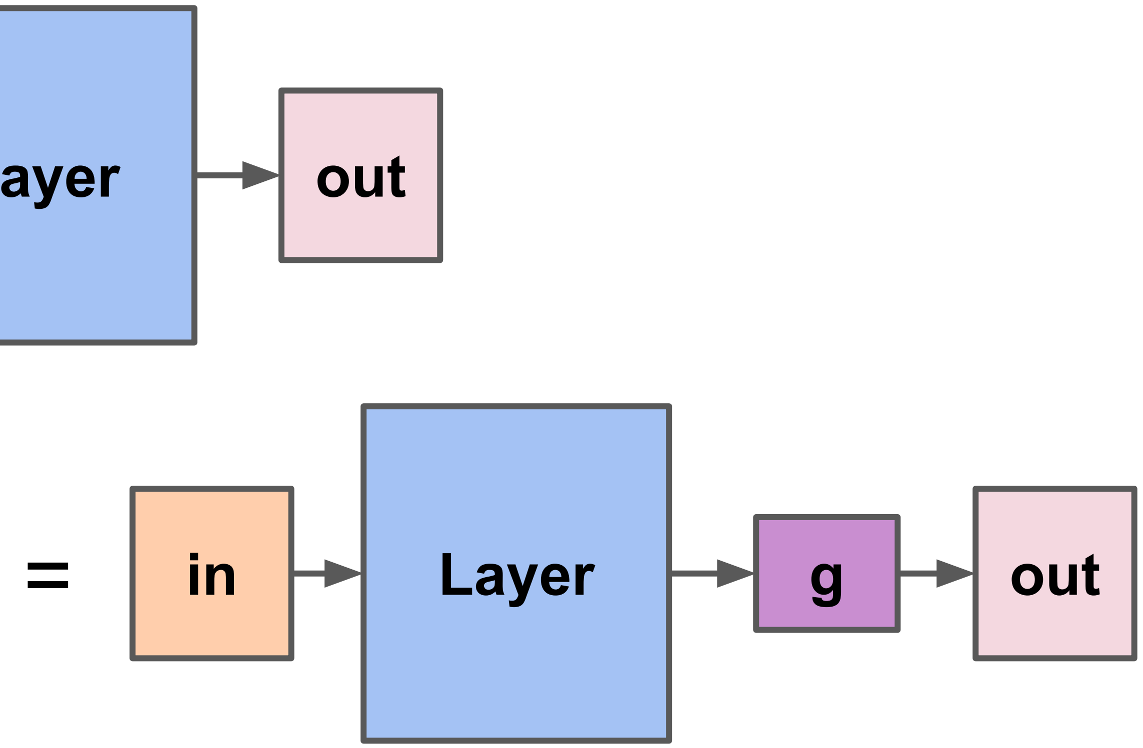


Equivariant Functions substantially shrink the space of learnable functions



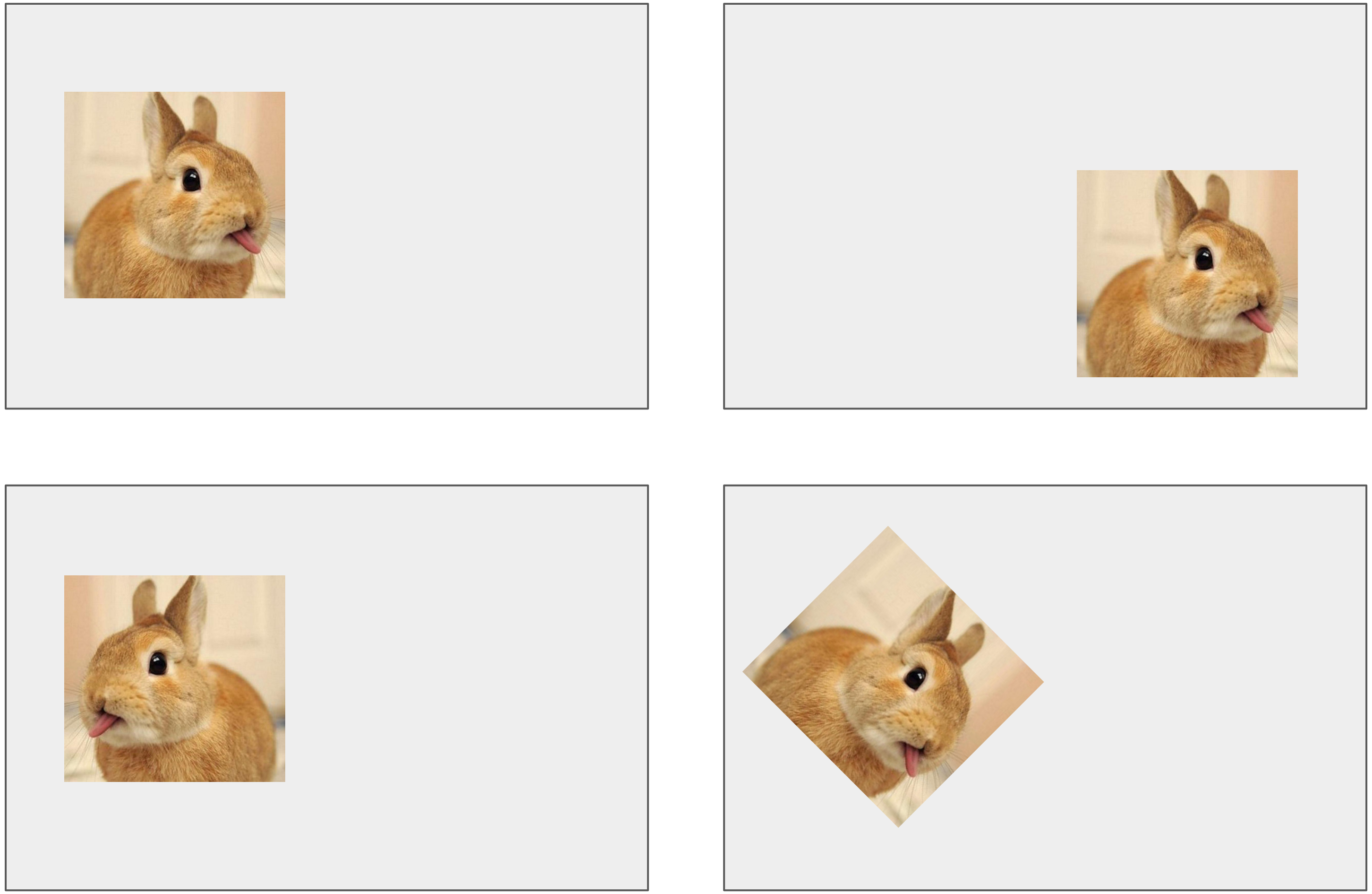
Approach 3: Equivariant Models

A g equivariant model is one where applying a group action g (in our case rotation) can be done before or after the layer is applied



If our model learns one instance of the model, it can learn it for all different orientations

if ✓
then...



✓ **translations**

✓ **mirrors**
(rotation +
inversion)

✓ **rotations**

[Smidt - e3nn]

Equivariant Neural Networks

input — learnable parameters — predicted output

$$f(x, w) = y$$

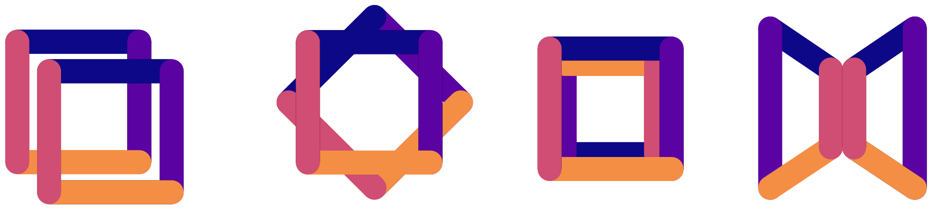
g is an element of Euclidean symmetry

$$f(D(g)x, w) = D(g)f(x, w)$$

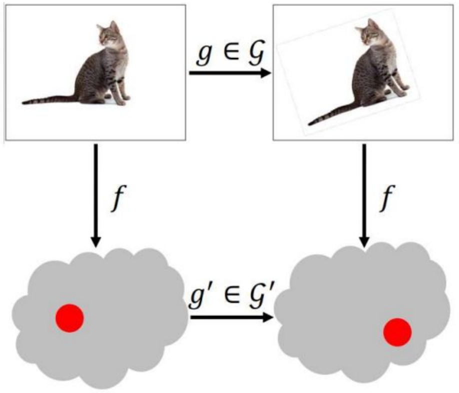
This restricts the possible functions to be tensors, and the only operations to be tensor algebra

Euclidean Transformations

3D Translations | 3D Rotations | 3D Inversion | Mirrors



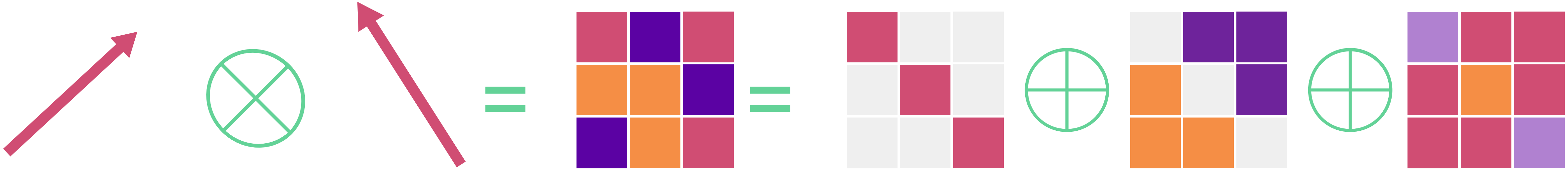
Equivariance



$$g' f(x) = f(gx)$$

invariant objects interact through scalar multiplication.

equivariant objects interact through tensor products



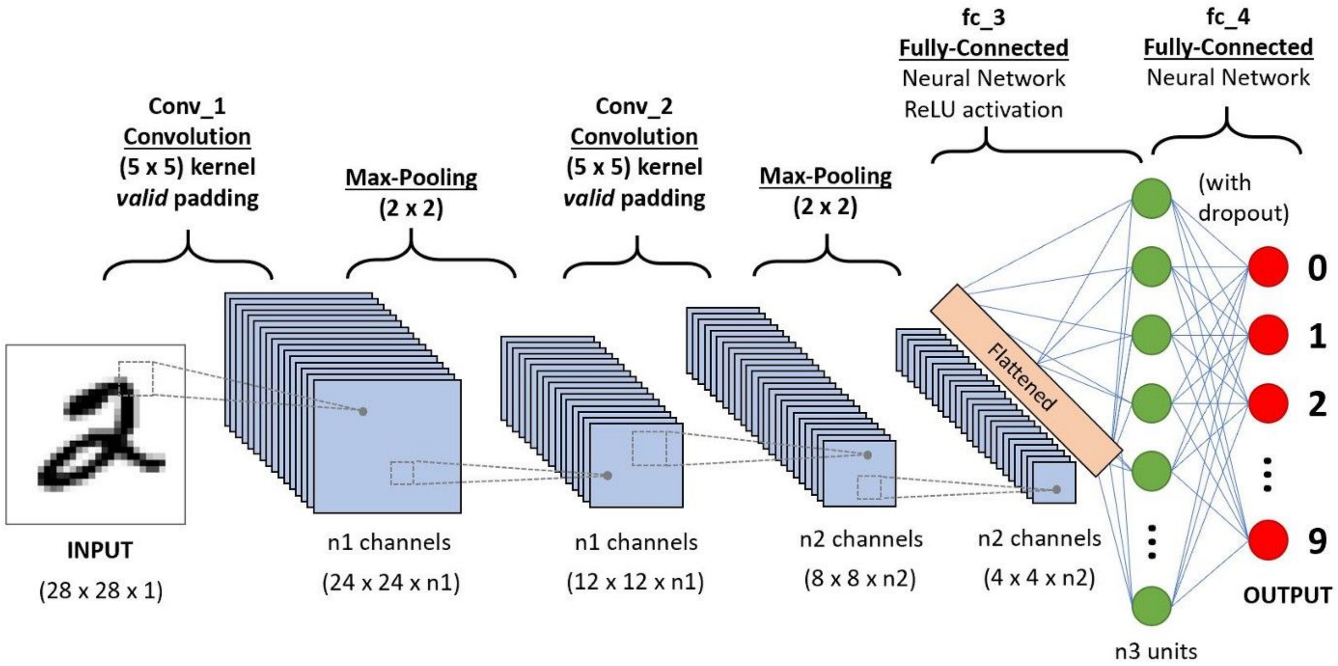
dot product	cross-product	symmetric
trace	antisymmetric	traceless
invariant	equivariant	equivariant
L=0	L=1	L=2
1 degree of freedom	3 degrees of freedom	5 degrees of freedom

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

In standard image convolutions, filter depends on coordinate system.

convolutional neural networks:

Used for images. In each layer, scan over image with learned filters.

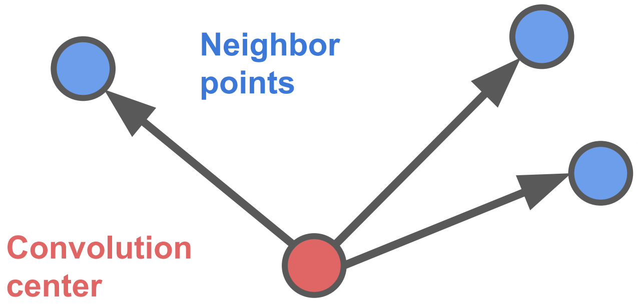


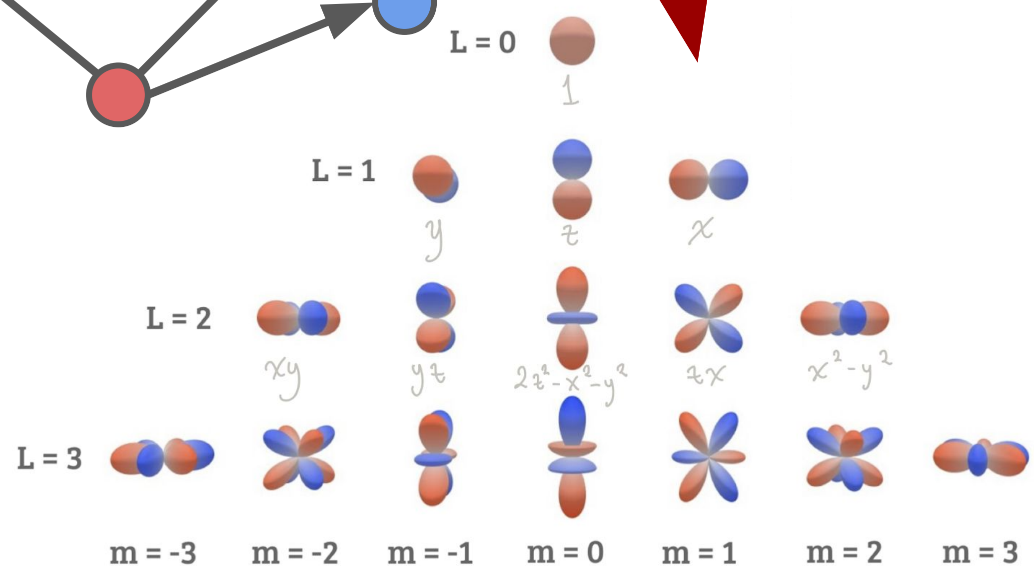
http://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/

… and we require the convolutional filter to be equivariant.

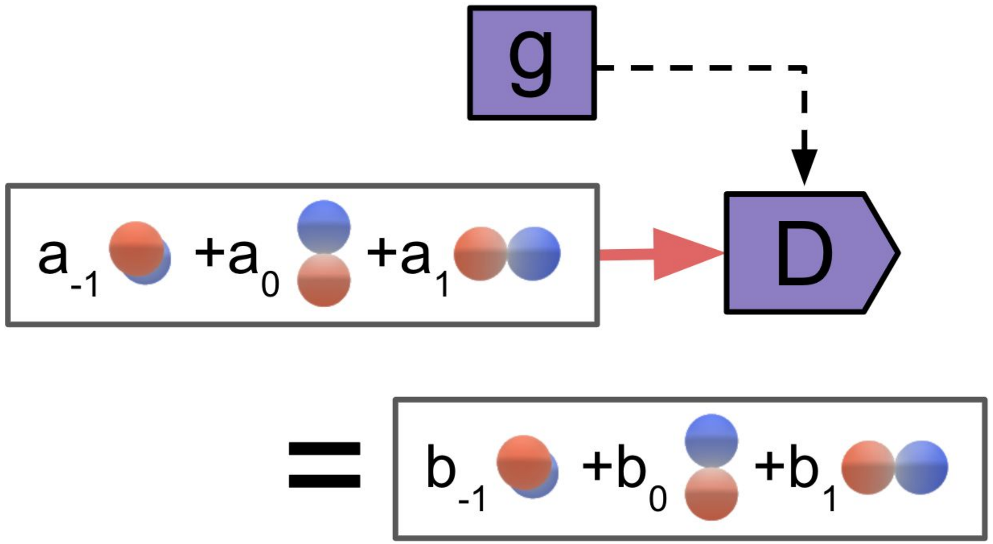
Rotation equivariant convolutional filters are based on learned radial functions and spherical harmonics…

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





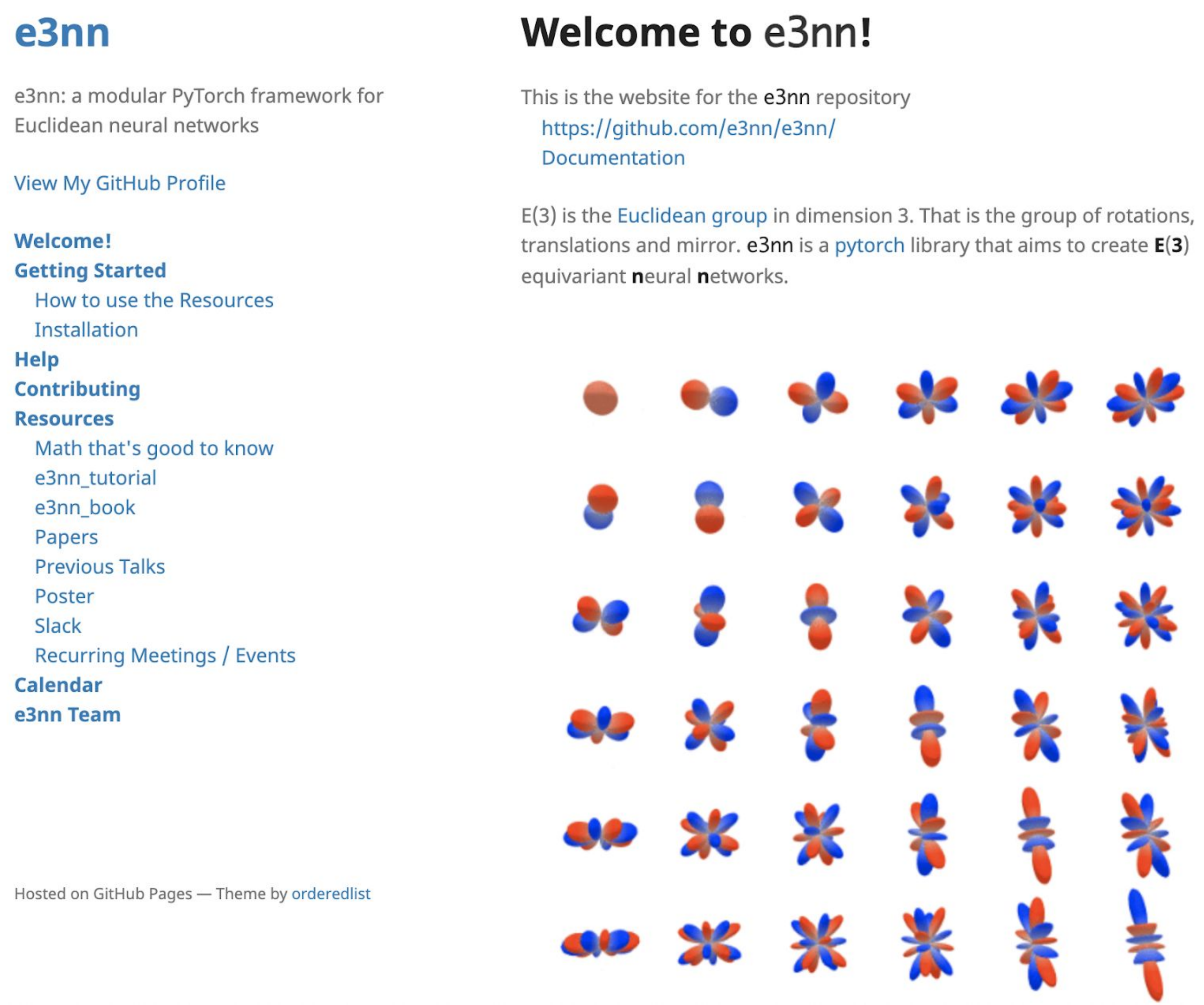
Spherical harmonics of the same L transform together under rotation **g**.



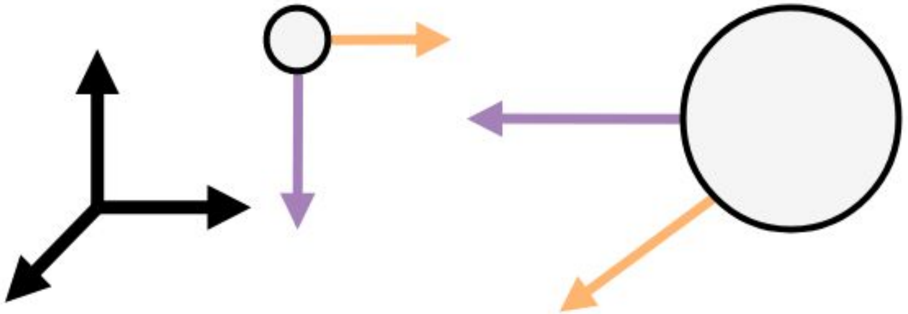
Spherical harmonics transform in the same manner as the irreducible representations of SO(3).

e3nn

- A euclidean equivariant neural network package in jax/pytorch
- https://github.com/e3nn/e3nn | https://e3nn.org | https://docs.e3nn.org



The input to our network is geometry and features on that geometry.
We categorize our features by how they transform under rotation and parity as *irreducible representations of O(3).*



```
geometry = [[x0, y0, z0],[x1, y1, z1]]
features = [
    [m0, v0x, v0y, v0z, a0x, a0y, a0z]
    [m1, v1x, v1y, v1z, a1x, a1y, a1z]
]
scalar = e3nn.o3.Irrep("0e")  # L=0, even p
vector = e3nn.o3.Irrep("1o")  # L=1, odd p
irreps = 1 * scalar + 1 * vector + 1 * vector
```

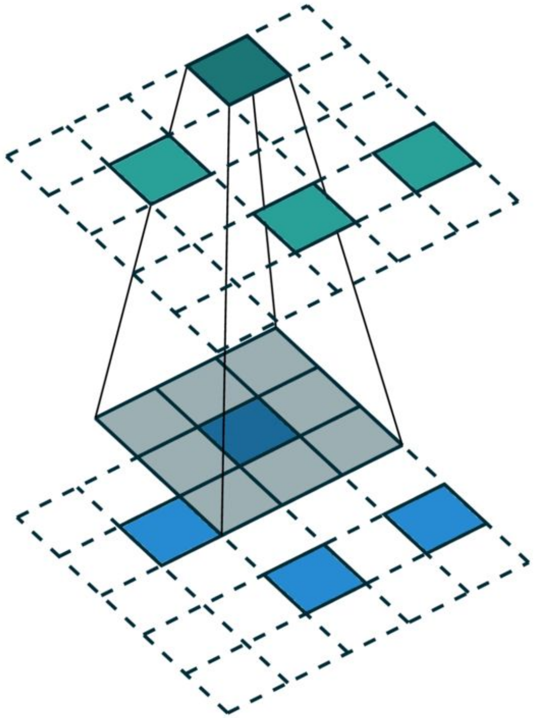
- All input, intermediate, and output data is "typed" by its transformation properties.
- All operations must respect this "typing".

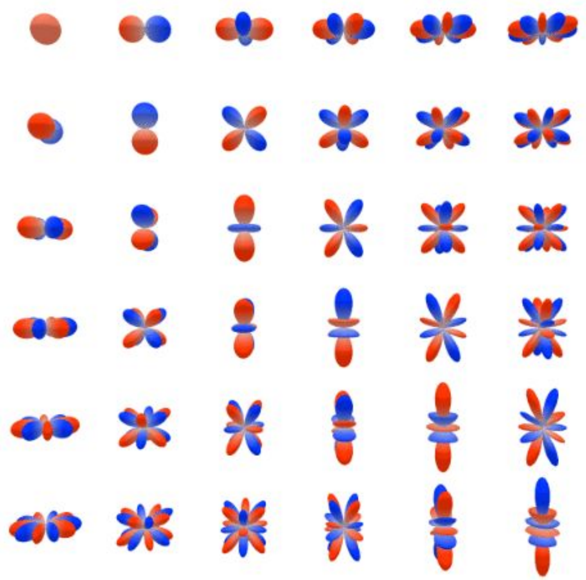
Sparse Equivariant Neural Networks

Sparse Euclidean Equivariant CNNs

Could we combine submanifold convolutions with euclidean equivariant NNs?

Yes, we can!





Recipe to make a Sparse Euclidean Equivariant CNN

- Equivariant Model Layers

Convolution

Batch Norm

Activation

Downsampling

Recipe to make a Sparse Euclidean Equivariant CNN

- To do convolutions, we embed the spherical harmonics on a set grid (our kernel shape). We then take the spherical harmonic with that. Scalar weights turn into irrep weights

Convolution

- For the batch normalization, we don't do anything different

Batch Norm

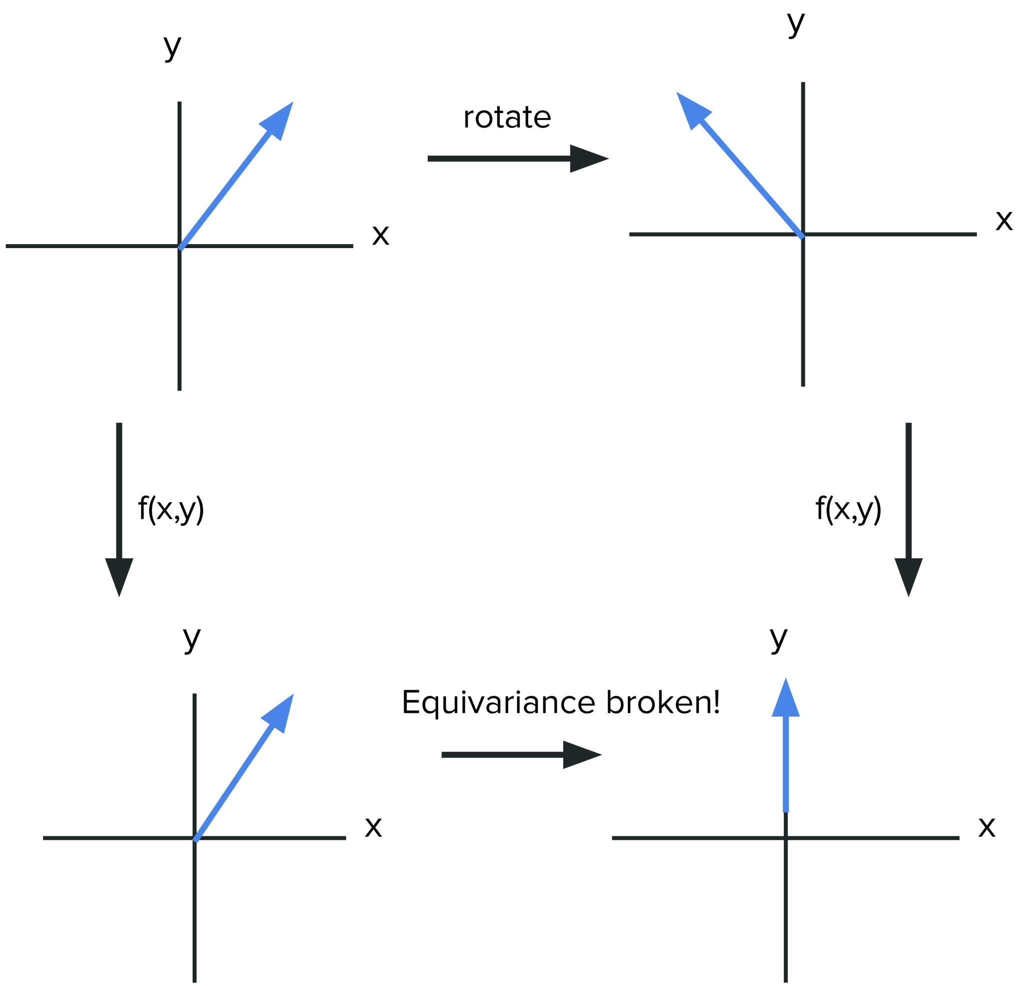
Gate Activation

- We cannot simply take the non-linearity of an equivariant function as that breaks the equivariance
- We can take non-linearity of scalars but not anything higher order
- To solve this, we take the non-linearity of a scalar times the higher order irrep

Activation

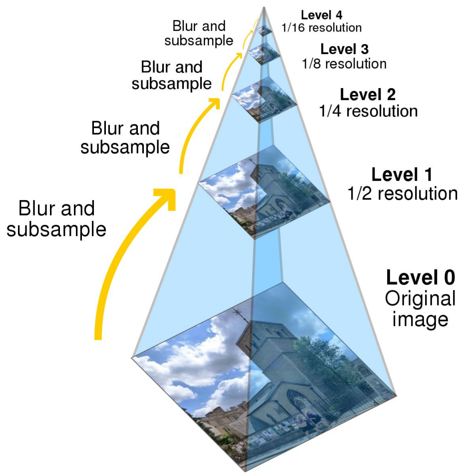
$$\left(\bigoplus_i \phi_i(x_i)\right) \oplus \left(\bigoplus_j \phi_j(g_j) y_j\right)$$

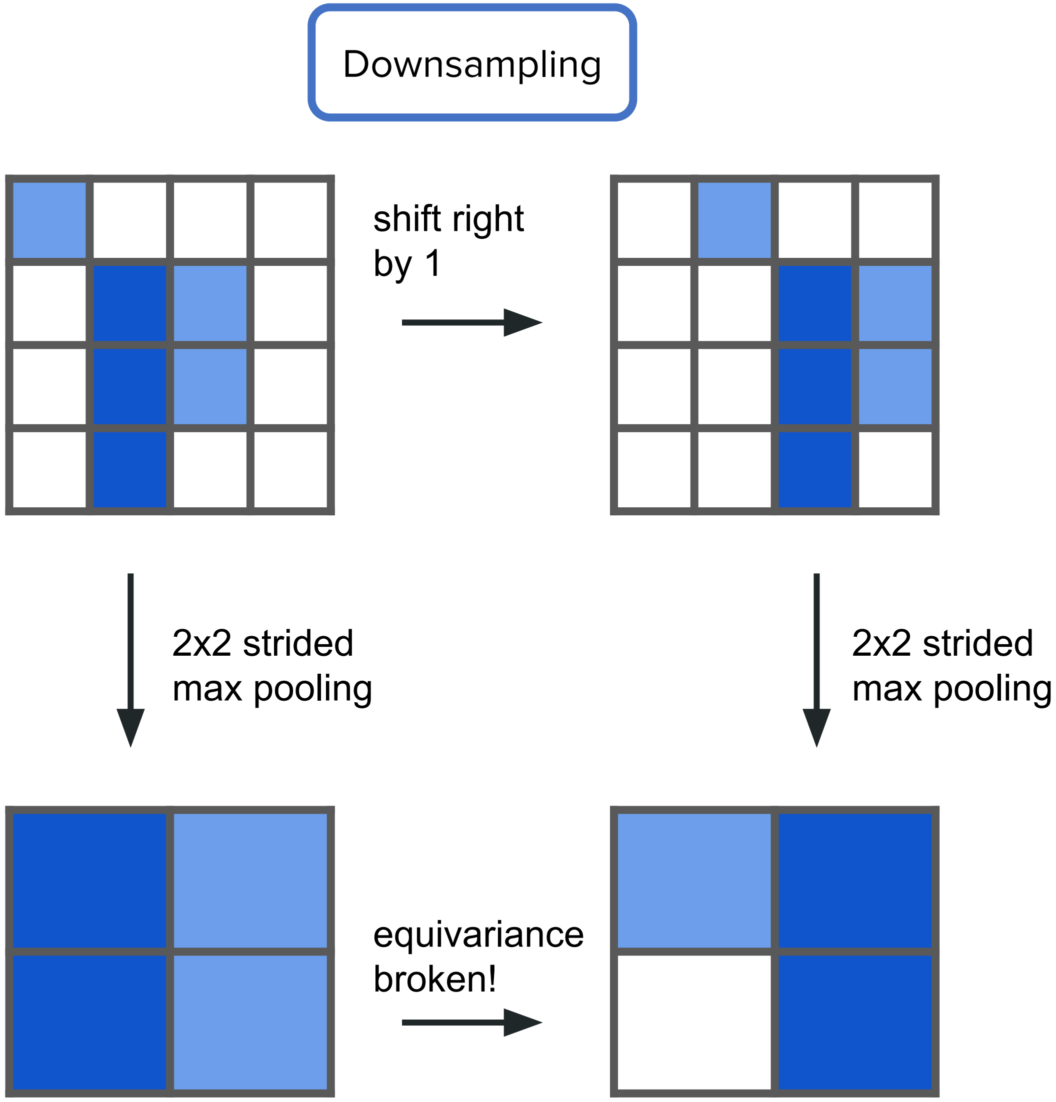
f(x,y) = [ReLU(x) , y]



Downsampling

- Downsampling breaks equivariance. CNNs are not fully shift equiavariant because of downsampling!
- Can be solved by downsampling using gaussian pyramid
- Need special care for downsampling non-scalar irreps





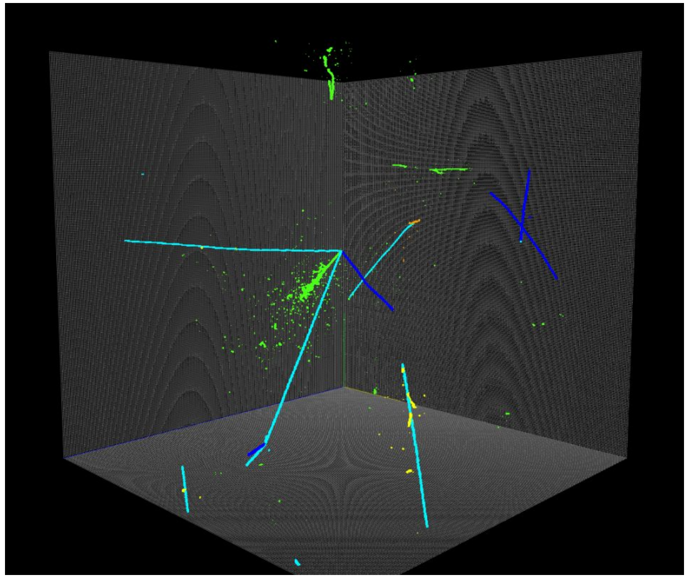
Recipe to make a Sparse Euclidean Equivariant CNN

- With all the parts in place. We can make a full model. All the layers don't break equivariance by construction ($\Delta<10^{-6}$) except the downsampling.
- The downsampling breaks the equivariance minimally ($\Delta<10^{-2}$)
- I was able to overfit the model on a batchsize of 8. The model stays equivariant before and after training

$$\Delta = g' f(x) - f(gx)$$

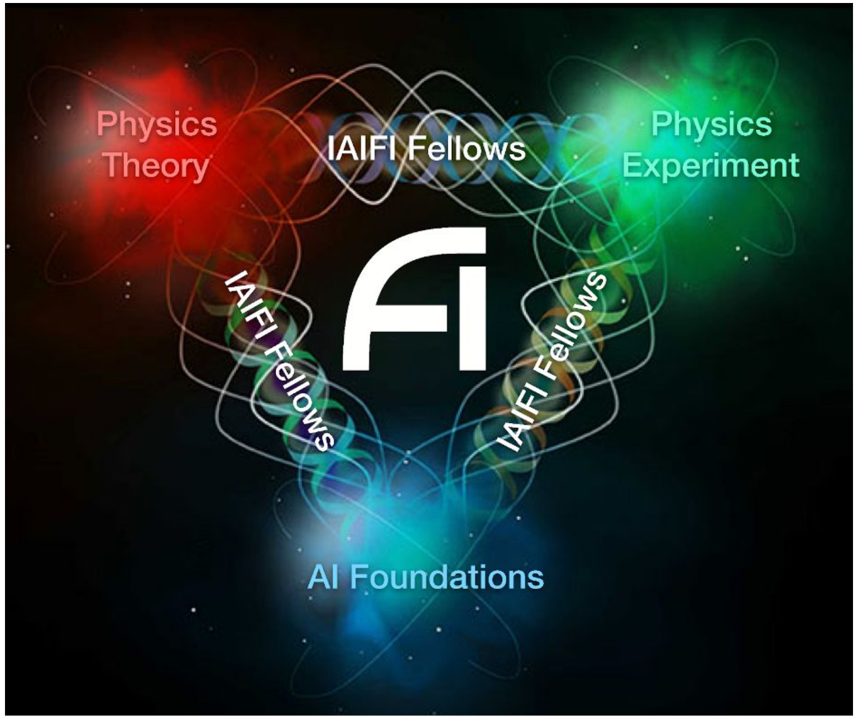
Next Steps

- With a working model, the next steps are to compare it to a baseline.
- A public dataset (PILArNet) is used for the training/test data
- I am starting off with a simpler task of single particle classification and seeing what improvements we get from training speed, inference speed, model size, …
- Following that, we can implement the model for harder tasks such as segmentation, momentum reconstruction, keypoint/vertex finding, …



PILArNet, Adams, et al

IAIFI collaboration



Taritree Wongjirad

Tess Smidt

Mario Geiger

Conclusion/ Outlook

- Track Reconstruction is a difficult problem that could use new innovation
- Equivariant Neural Networks is a surging field with many possible implementations/improvements
- I have successfully built a sparse euclidean equivariant convolutional neural networks, and I am currently working on scaling up the model training to compare to the baseline.

Questions