Introduction to Group Equivariant Neural Networks!

Part 0: Motivation

Imagine you're working with the MNIST database of digits and you want to build a classifier. Intuitively, you'd like an 8 rotated 45 degrees still to be classified as an 8.

What is one way of solving this problem without worrying about designing a new architecture or altering our model? (open question)

- Data Augmentation

This works pretty well for small datasets with only a few possible symmetries, but what about for more complicated problems? How can we build these kinds of symmetries into networks as inductive biases?

Part 1: The Mathematical Language of Symmetry

A few definitions:

Def: A group G is a set G with a law of composition that obeys the following

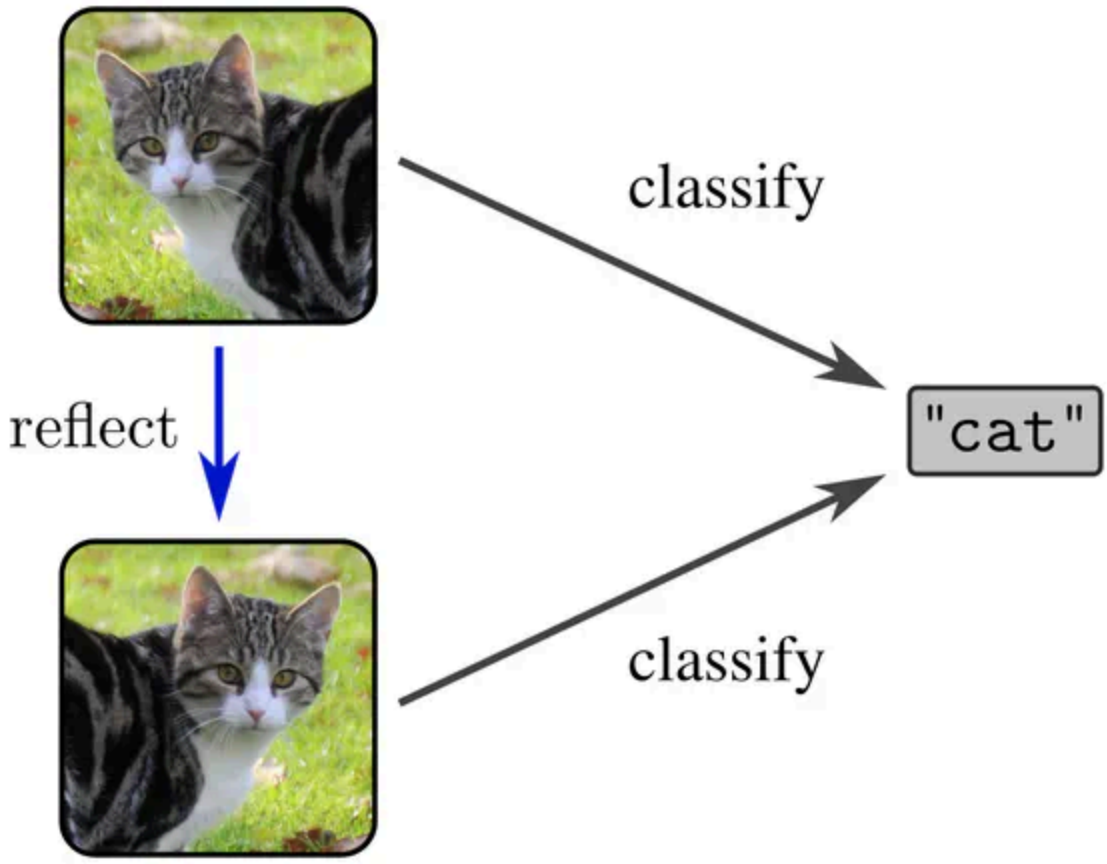
- Closure For all $(a, b \in G)$, the result of the operation $(a * b)$ is also in (G):
- Associativity: For all $(a, b, c \in G)$, the operation * satisfies: $(a * b) * c = a * (b * c)$
- Identity Element: There exists an element $(e \in G)$ such that for every $a \in G$: $e * a = a * e = a$
- Inverse Element: For each $(a \in G)$, there exists an element $(a^{-1} \in G)$ such that: $a * a^{-1} = a^{-1} * a = e$

Def: A group action α of the group G on the set X is a map from $(G \times S) \rightarrow S$ that has an identity such that $\alpha(e, x) = x$ and that the group action is compatible with the law of composition in the group $\alpha(g, \alpha(h, x)) = \alpha(gh, x)$

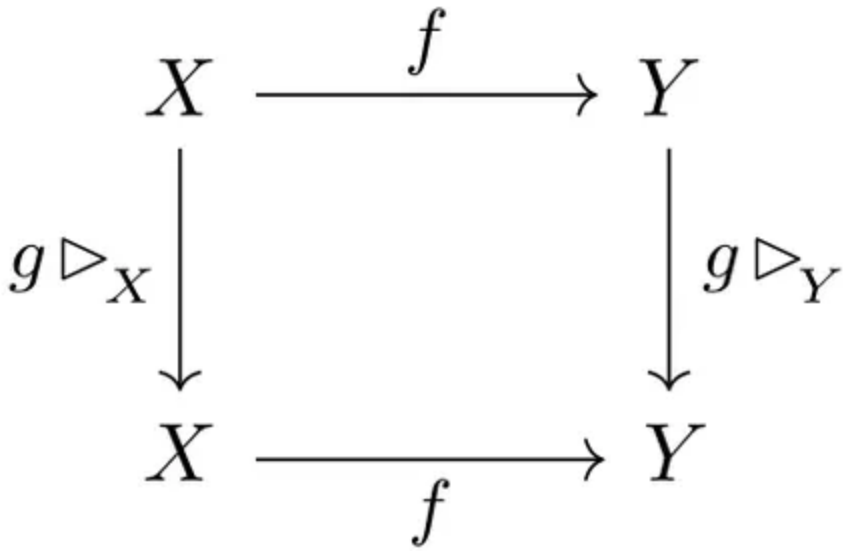
Why should we care? Well, we need a way to formalize the kinds of symmetries we're interested in. All that language comes from the language of group theory. In particular, we define invariance and equivariance as follows:

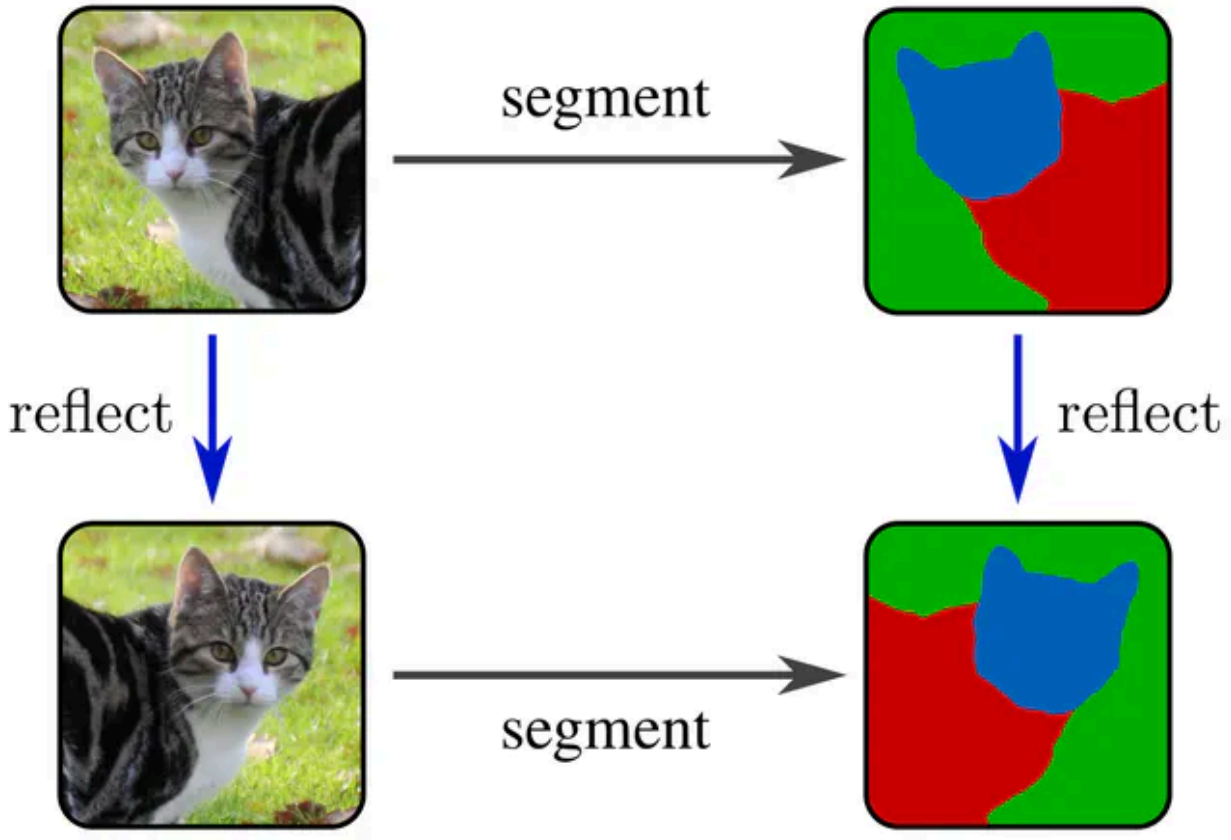
Let G be a group and X be a set and $f : X \to Y$ be a function (here our function is our model)

Def: f is invariant with respect to the action of $g \in G$ if $f(g \curvearrowright x) = f(x)$



Def: f is **equivariant** with respect to the action of $g \in G$ if $f(g \curvearrowright x) = g \curvearrowright f(x)$

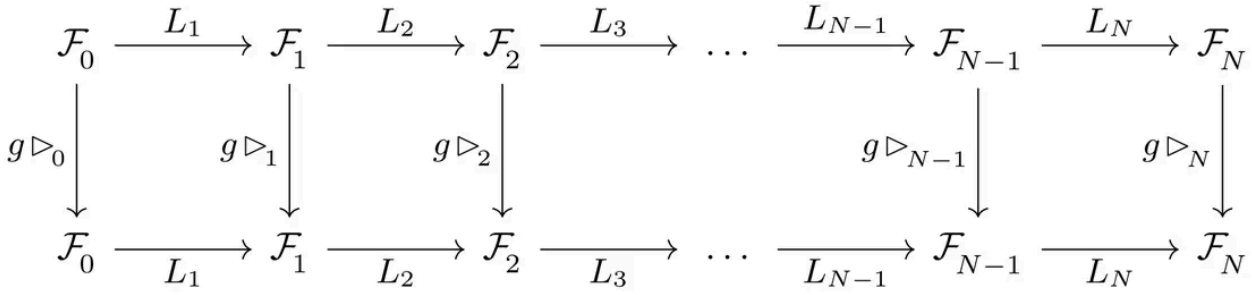




Practically speaking, equivariance can save a lot of time and prevent you from having to do a lot of tedious data augmentation. It also means that you need many fewer parameters (we'll return to this later, but intuitively, enforcing equivariance as a constraint is the same as restricting the parameter search space to a lower dimensional subspace).

Let's think about how we can formalize the notion of equivariance for a neural network. Recall that $f = L_1 \circ L_2 \circ L_3 \cdots \circ L_n$ describes an equivariant neural network, where each L_i describes a trainable layer mapping between adjacent feature spaces. We want a

sequence of equivariant layers to make this model equivariant. The group action is a "hyper parameter" chosen before training.



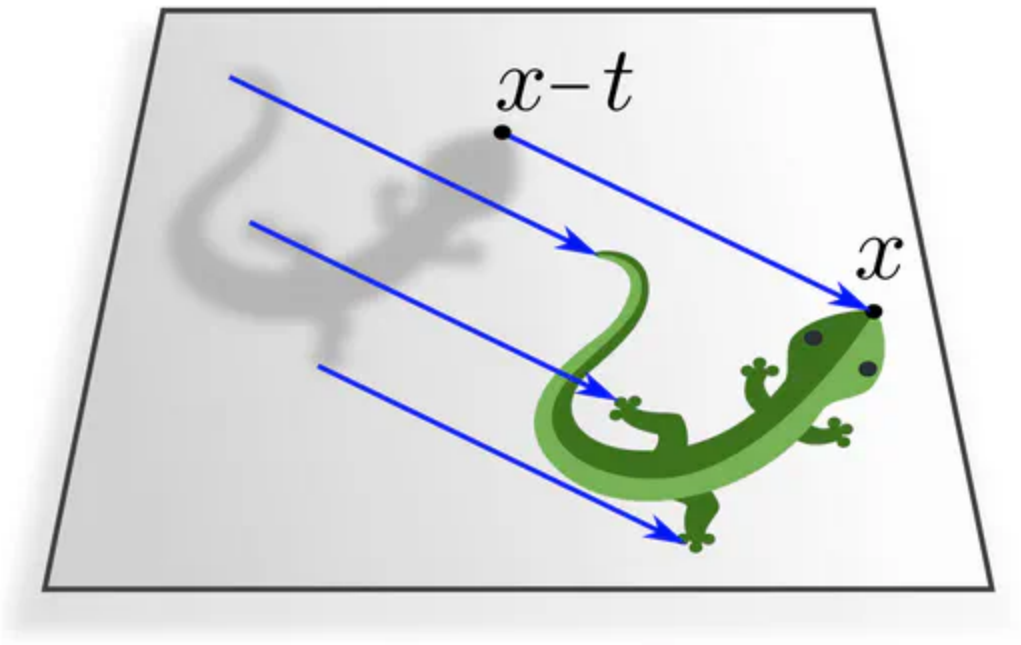
A few words of warning: anecdotally, people who work in industry with less constrained compute/data are often skeptical of building in strong inductive biases like this.

You may also not know what inductive biases are appropriate to your data situation -- building the wrong inductive bias into your model could be very bad for performance.

If you work with convolutional neural networks, you're already working with a translation invariant neural network. CNNs are so useful as classifiers for spatially structured data because they are translation invariant -- a dog in the upper left quadrant is just as much a dog as one in the lower left quadrant.

Part 2: Convolutional Neural Networks and Translation Invariance

Here, we're trying to enforce translational equivariance, so we'll be working with the group of translations $\mathbb{R}^d$ with the law of composition given by addition. The action of $(\mathbb{R}^d, +)$ on $\mathbb{R}^{\mathbb{d}}$ is just a shift. Consider $t \in G$. t acts on points $x \in \mathbb{R}^d$ by $(t, x) \mapsto x + t$ and acts on feature maps by $[t \curvearrowright F](x) = F(x - t)$



It turns out that you can prove that the only way to enforce this constraint that $\mathcal{L}(t \curvearrowright F) = t \curvearrowright \mathcal{L}(F)$ is by using convolutions.

Theorem: Translation equivariant linear functions between feature maps are necessarily convolutions KF

This I hope demonstrates the power and value of thinking of equivariance as the result of obeying some constraint imposed by a group action.

Part 3: Group-Equivariant Convolutional Neural Networks

But what about more fun groups, like the rotation group promised at the beginning of this talk? We can easily discuss finite rotation groups $G =< r >$ or the set of all r, 2r, 3r, ... nr degree rotations etc.

Constraint Equation for Invariant GCNNs

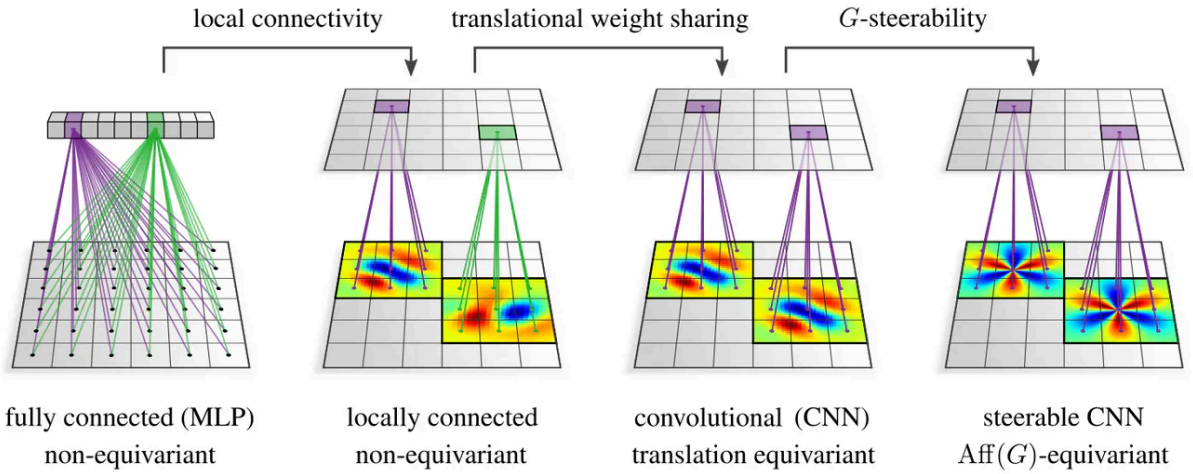
$$\mathcal{H}_{\text{inv}} := \{\mathcal{M} : \mathcal{F}_{\text{in}} \to \mathcal{F}_{\text{out}} \mid \mathcal{M}(g \triangleright_{\text{in}} x) = \mathcal{M}(x) \quad \forall g \in G,\, x \in \mathcal{F}_{\text{in}}\} \subseteq \mathcal{H}_{\text{full}}$$

- Intuition: An invariant GCNN would be useful for tasks where the object's identity, not its orientation, is important. For example, a network recognizing a circular object like a clock should give the same output regardless of the clock's orientation.

Constraint Equation for Equivariant GCNNs

$$\mathcal{H}_{\text{equiv}} := \{\mathcal{M} : \mathcal{F}_{\text{in}} \to \mathcal{F}_{\text{out}} \mid \mathcal{M}(g \triangleright_{\text{in}} x) = g \triangleright_{\text{out}} \mathcal{M}(x) \quad \forall g \in G,\, x \in \mathcal{F}_{\text{in}}\} \subseteq \mathcal{H}_{\text{full}}$$

- **Intuition**: In an equivariant GCNN, if an image is rotated, the output feature map is also rotated correspondingly. This property is crucial for tasks like pose estimation, where the orientation of an object matters.
  This diagram shows the hierarchy of subspace constraints from right to left. The Aff(G)-Equivariant subspace is far more constrained than even the translation equivariant model.
  Reflection steerable convolution between scalar fields via symmetric kernels



  Big Idea: Similar to translational equivariance achieved through weight sharing in conventional CNNs, steerable CNNs enforce equivariance using structured weight sharing. The filter weights are tied together in a way that ensures the response transforms according to the group's rules.

Example: $G = \{e, r, r^2, r^3\}$
In a standard CNN, a single convolutional filter scans the image using the same set of weights, achieving translational equivariance. In a GCNN invariant to the 90-degree rotation group, we extend this idea to handle rotations by sharing weights in a structured way. Here's how it works:

1. Rotational Weight Sharing

Instead of having separate filters for each orientation, a GCNN enforces **weight sharing** across rotated versions of the same filter. Specifically, if we have a base filter W, we create rotated versions of W for each element in the rotation group G. The network ensures that the responses of these rotated filters are tied together, respecting the rotational symmetry.

2. Mathematical Representation

Let W be a base filter, and let R be a rotation operator that applies 90-degree rotations. We generate rotated filters as:

$W_r = R(W), \quad W_{r^2} = R^2(W), \quad W_{r^3} = R^3(W)$ where W_r, W_{r^2} and W_{r^3} are the filters obtained by rotating W by 90, 180, and 270 degrees, respectively.

The **structured weight sharing** constraint ties these filters together, ensuring that they are not learned independently. Instead, the parameters of W determine the parameters of W_r, W_{r^2}, W_{r^3}

Suppose our base filter W is a 3x3 matrix: $W = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

$W_r = \begin{bmatrix} g & d & a \\ h & e & b \\ i & f & c \end{bmatrix}$

$W_{r^2} = \begin{bmatrix} i & h & g \\ f & e & d \\ c & b & a \end{bmatrix}$

$W_{r^3} = \begin{bmatrix} c & f & i \\ b & e & h \\ a & d & g \end{bmatrix}$

Part 4: Next Steps

So far, I've only talked in detail about how to think about invariance and equivariance for finite groups, like D_8 or S_4. This takes us to roughly 2016 in the literature. However, there is a rich world of literature beyond simple finite groups, including networks equivariant to continuous groups (infinite sized groups), steerable GCNNS, Gauge-equivariant CNNs and more.

If you want to play around with these models in real life try the ESCNN library, from UVA, which has lots of these models built in: https://quva-lab.github.io/escnn/

Part 4: Bibliography

Maurice Weiler's Textbook and Blog Posts are FANTASTIC on this subject:
https://maurice-weiler.gitlab.io/blog_post/cnn-book_1_equivariant_networks/
https://maurice-weiler.gitlab.io/blog_post/cnn-book_2_conventional_cnns/