

Part 5: Building and Testing G-CNNs

Lecturer: Renjie Liao

Revised: Sep. 29, 2026

Note: *LaTeX template courtesy of UC Berkeley EECS dept.***Disclaimer:** *These notes were revised with assistance from GPT-6 Astra.*

The preceding parts supply the building blocks. This part explains how to assemble, train, and test a complete model. Its two examples are a small quarter-turn CNN and a scalar-to-vector layer on 3D points. A final Fourier appendix connects orientation channels to irreducible representations. The companion file `group_equivariance_examples.py` contains the executable examples and tests.

5.1 Specify the transformation contract before the architecture

Write down the input action, hidden feature types, and desired output action. These choices determine which operations are allowed.

Setting	Suitable starting point	Key design choice
Square images, quarter-turns	$p4$ lifting and group layers	Keep an explicit orientation axis
Images, arbitrary planar rotations	Steerable spatial kernels	Choose field types; measure sampling error
3D points or atoms	Typed geometric message passing	Use relative positions and scalar/vector or higher-order features

For a classifier, the last output action is trivial. For a force predictor, each output is a polar vector. Reflection symmetry is an additional choice: $SO(3)$ requires rotations, while $O(3)$ also constrains reflections. This distinction matters for handedness and chiral targets.

Theorem 5.1 (Composition preserves equivariance) *Suppose every layer $F_\ell : V_{\ell-1} \rightarrow V_\ell$ satisfies*

$$F_\ell T_g^{\ell-1} = T_g^\ell F_\ell \quad \text{for every } g.$$

Then the composition $F_L \circ \dots \circ F_1$ is equivariant from the input action to the final output action. If $PT_g^L = P$, applying the readout P makes the prediction invariant.

Proof: Move the transformation through each layer in order. This gives $F_L \dots F_1 T_g^0 = T_g^L F_L \dots F_1$. Apply P and its stated identity for the second claim. ■

Why this result is useful. One can check small components independently, but *every* operation must respect the contract, including bias, normalization, activation, residual paths, pooling, and coordinate pre-processing.

5.2 A minimal quarter-turn network in PyTorch

Use images of shape $[B, C_{\text{in}}, H, W]$ with $H = W$, and hidden features $[B, C, 4, H, W]$. As in Part 2, output orientation r uses a spatially rotated kernel and, for hidden layers, an orientation shift by r . The following code uses odd kernels, unit stride, and circular padding. On this finite periodic grid, integer translations and quarter-turns are exact permutations. Circular padding is chosen to make the algebra testable; whether wraparound is appropriate for real images is a separate modeling decision.

```

import torch
import torch.nn.functional as F

def spatial_corr(x, w):
    p = w.shape[-1] // 2
    return F.conv2d(F.pad(x, (p,p,p,p), mode="circular"), w)

def lift(x, w):
    # w: [Cout,Cin,k,k]; result: [B,Cout,4,H,W]
    return torch.stack([
        spatial_corr(x, torch.rot90(w, r, (-2,-1)))
        for r in range(4)], dim=2)

def group_layer(x, w):
    # w: [Cout,Cin,4,k,k]
    B, Cin, _, H, W = x.shape
    flat = x.reshape(B, Cin*4, H, W)
    ys = []
    for r in range(4):
        wr = torch.rot90(w, r, (-2,-1)).roll(r, dims=2)
        wr = wr.reshape(w.shape[0], Cin*4, *w.shape[-2:])
        ys.append(spatial_corr(flat, wr))
    return torch.stack(ys, dim=2)

def rotate_hidden(x, r):
    return torch.rot90(x, r, (-2,-1)).roll(r, dims=2)

```

The two rotations of w and x use the same array convention. The positive `roll(r)` reads orientation $s - r$ at destination s . The group layer flattens (C_{in}, s) in the same order as its weights. Four separate convolution calls make this example easy to audit; stacking the weight bank permits a single call in an optimized implementation. PyTorch's `conv2d` uses Part 2's CNN convolution convention, so no extra kernel flip is applied [1].

5.2.1 Add trainable parameters and a readout

Declare the base kernels as `torch.nn.Parameter` tensors in a module. Only those base kernels belong in the optimizer; the rotated banks are differentiable views or derived tensors, not independent parameters. A simple model is

$$z = \text{ReLU}(\text{lift}(x, w_1)), \quad y = \text{group_layer}(z, w_2).$$

Componentwise ReLU is valid here because the group action only permutes locations and orientation channels. A bias must be shared over orientation:

$$y_{b,c,r,u,v} \leftarrow y_{b,c,r,u,v} + b_c.$$

For invariant classification, average over *both* orientation and space and apply an ordinary classifier:

$$z_{b,c}^{\text{cls}} = \frac{1}{4HW} \sum_{r,u,v} y_{b,c,r,u,v}, \quad \text{scores} = \text{MLP}(z^{\text{cls}}).$$

Pooling orientation alone leaves a spatial feature map that still rotates. For segmentation, retain spatial coordinates and construct an equivariant output head instead of global pooling.

Training is familiar. Initialize the base weights, run the forward pass, compute the task loss, backpropagate, and update the coefficients with the usual optimizer. The parameterization preserves the symmetry for every update; no separate equivariance penalty is required for these exact discrete layers. Augmentation can still help with variations not covered by the chosen group.

5.3 A 3D layer without spherical-harmonic machinery

Let $p_i \in \mathbb{R}^3$ be node positions and $h_i \in \mathbb{R}$ be scalar features. Define

$$v_i = \sum_j a(\|p_i - p_j\|^2) h_j (p_i - p_j). \quad (5.1)$$

The radial function a may be a learned MLP. Distances and h_j are unchanged by rigid motion; the displacement is a vector. Thus $p_i \mapsto Rp_i + t$ gives $v_i \mapsto Rv_i$, including reflections if $R \in O(3)$. The $j = i$ term is zero.

```
def scalar_to_vector(pos, h):
    # pos: [N,3]; h: [N]; fixed radial function for this demo
    delta = pos[:, None, :] - pos[None, :, :] # xi - xj
    d2 = (delta * delta).sum(-1)
    weight = torch.exp(-d2) * h[None, :]
    return (weight[..., None] * delta).sum(dim=1)
```

This is the scalar-to-vector steerable kernel from Part 4 with a smooth radial profile and no division at zero. It is also a geometric message-passing layer. The dense demonstration costs $O(N^2)$ pair operations; a distance-based neighbor list reduces it to $O(|E|)$ message operations, plus neighbor-search cost. Rebuild or transform the graph consistently. A radius graph is invariant under rigid motion in exact arithmetic; index-based tie breaking or arbitrary neighbor caps can break permutation equivariance.

A valid vector nonlinearity. For an orthogonal representation, a scalar function of the norm is invariant, so

$$\text{gate}(v) = \text{sigmoid}(\|v\|) v$$

is equivariant. Componentwise ReLU is not: for $v = (1, -1, 0)^\top$ and $R = \text{diag}(-1, -1, 1)$, $\text{ReLU}(Rv) = (0, 1, 0)^\top$ while $R \text{ReLU}(v) = (-1, 0, 0)^\top$. An operation familiar from scalar CNNs can violate a vector feature type.

5.3.1 Extending to typed geometric networks

Part 4 generalizes (5.1) using spherical harmonics and fixed tensor-product couplings:

$$m_{ij} = \sum_{\text{allowed paths}} a_{\text{path}}(\|p_i - p_j\|) \mathcal{C}_{\text{path}}(h_j \otimes Y_J(\widehat{p_i - p_j})), \quad h'_i = \sum_j m_{ij}.$$

Here a path specifies compatible input, angular, and output types; $\mathcal{C}_{\text{path}}$ is a fixed change of basis, while the radial weights are learned. The implementation pattern is: build edges, evaluate distances and directions, evaluate the radial network and angular basis, form typed messages, sum by destination, then apply a compatible nonlinearity. The `e3nn` convolution guide implements this pattern and provides tensor-product primitives [2, 3]. A practitioner can use these primitives without implementing Clebsch–Gordan coefficients.

Read type specifications as tensor shapes. In `e3nn`, `16x0e + 8x1o` denotes 16 even scalar copies and eight odd-parity vector copies, hence $16 + 8 \cdot 3 = 40$ coordinates. The letters describe inversion parity; an ordinary polar vector is odd under inversion. A hidden type’s *multiplicity* is a channel-width choice; its $2l + 1$ coordinates must transform together. For planar images, `escnn` provides typed fields and equivariant convolution blocks [4]. Check a library’s declared group and normalization conventions before connecting its tensors to custom code.

5.4 Test transformation rules before measuring accuracy

The defining equation gives a direct test. For an input x and group element g , compute

$$a = F(T_g^{\text{in}} x), \quad b = T_g^{\text{out}} F(x), \quad \varepsilon_{\text{rel}} = \frac{\|a - b\|_2}{\max(\|a\|_2, \|b\|_2, \epsilon)}.$$

Also record the maximum absolute error; relative error is unstable when both outputs are nearly zero. Libraries such as `e3nn` provide utilities for this comparison [5].

```
torch.manual_seed(576)
x = torch.randn(2, 2, 9, 9, dtype=torch.float64)
w = torch.randn(3, 2, 3, 3, dtype=torch.float64)
for r in range(4):
    before = lift(torch.rot90(x, r, (-2,-1)), w)
    after = rotate_hidden(lift(x, w), r)
    torch.testing.assert_close(before, after, atol=1e-10, rtol=1e-10)
```

Test protocol.

1. Verify the transformation laws: identity, inverse, and composition. Test several nonzero angles or translations, not only the identity.
2. Test layers and then the full network, including readouts. Use nonconstant, nonsymmetric inputs and multiple random weights.
3. Start with CPU float64 and exact permutations. Then test the actual dtype, device, padding, interpolation, and stride used for training.
4. For point data, transform positions and every typed feature, and also test node permutations. For mixed scalar/vector output, compare each block using its own action.
5. Disable stochastic layers or transform their randomness consistently. Testing two unrelated dropout masks does not test deterministic equivariance.
6. Backpropagate a scalar loss. Check that all intended coefficient tensors receive finite, nonzero gradients.

The companion tests cover all four quarter-turns and several periodic shifts, hidden group layers, invariant pooling, rigid motions and permutations of 3D points, and gradient flow. They also include deliberate failures for orientation-dependent biases and vector ReLU. Small numerical errors support an implementation check; they do not replace a proof for all inputs.

5.4.1 Common ways a valid layer becomes an invalid network

Operation	Required condition	Typical failure
Bias	Fixed by the output representation	Independent biases for vector components or orientations
Normalization	Statistics and affine parameters respect the type	Separate running statistics for permuted orientations
Residual addition	Both branches have the same action	Adding equally sized tensors with different feature types
Subsampling	Output lattice and action are compatible	Stride two assumed equivariant to every one-pixel shift
Pooling	Reduced indices form the intended symmetry orbit	Calling an orientation-pooled spatial map invariant
Dropout	Masks coupled under the action for samplewise equality	Independent masks for coordinates of one vector

For regular orientation features, a simple normalization shares statistics and affine parameters across orientations. For orthogonal vector features, use invariant norms or type-aware normalization; arbitrary coordinatewise centering and scaling are not generally valid.

5.5 Choose the symmetry budget, then compare fairly

Equivariance trades expressiveness for a specified structural assumption. It is most useful when that assumption is reliable and data do not already cover every relevant transformation. More symmetry is not always better: overly strong invariance can remove information needed by the target.

Count parameters and operations separately. With A orientations and $C_{\text{in}}, C_{\text{out}}$ learned feature copies per orientation, a $k \times k$ lifting kernel has $C_{\text{out}}C_{\text{in}}k^2$ learned parameters and produces AC_{out} channels. A dense group kernel has $AC_{\text{out}}C_{\text{in}}k^2$ parameters, while direct group convolution costs on the order of $A^2C_{\text{out}}C_{\text{in}}k^2HW$ multiply-adds. The factor depends on which widths are held fixed. An assertion that every G-CNN layer costs only A times an ordinary layer is therefore incomplete. Steerable bases share parameters, but basis expansion, tensor products, and larger feature dimensions still cost time and memory.

A small experimental plan. Compare an ordinary network, that network with augmentation, and an equivariant network. Report parameter count, runtime, training-set size, ordinary test accuracy, transformed-input accuracy, and equivariance error. Include inputs where the proposed symmetry is only approximate. Keep the data split and tuning budget comparable; a gain should not be attributed to symmetry when it comes from a larger model or a different training budget.

Appendix: why the Fourier view leads to feature types

This appendix preserves the spectral connection without making it a prerequisite for the implementations above.

Orientation channels and frequency channels encode the same data

For A orientation responses z_r , define the discrete Fourier transform

$$\hat{z}_m = \sum_{r=0}^{A-1} z_r e^{-2\pi i m r / A}, \quad z_r = \frac{1}{A} \sum_{m=0}^{A-1} \hat{z}_m e^{2\pi i m r / A}.$$

If a rotation cycles the responses by q , $z'_r = z_{r-q}$, then $\hat{z}'_m = e^{-2\pi i m q / A} \hat{z}_m$. A permutation in the orientation basis becomes a phase in the frequency basis. The full transform is only a change of coordinates; dropping frequencies is an additional restriction.

An irreducible representation is a smallest nonzero invariant block. Finite-dimensional representations of finite groups, and continuous finite-dimensional representations of compact groups, decompose into such blocks. Over $\mathbb{C}$, irreducible representations of an abelian compact group are one-dimensional. Over $\mathbb{R}$, nonzero frequencies of $SO(2)$ are two-dimensional sine/cosine pairs, as in Part 3. For $SO(3)$ the degree- l block has $2l + 1$ coordinates. Changing basis can simplify rotations, but arbitrary nonlinearities in the new basis need not preserve the original transformation law.

General compact-group transform and the order of multiplication

Let G be compact with normalized Haar measure $\int_G dg = 1$. Let ρ_λ range over inequivalent unitary irreducible representations of dimension d_λ . For a scalar complex signal, use the convention

$$\hat{f}(\lambda) = \int_G f(g) \rho_\lambda(g)^\dagger dg, \quad f(g) = \sum_\lambda d_\lambda \text{Tr}(\hat{f}(\lambda) \rho_\lambda(g)). \quad (5.2)$$

The inverse is an L^2 expansion for $f \in L^2(G)$, not a claim of pointwise convergence for every function. For a finite group, integration means the uniform average and the inverse is exact with finitely many blocks. This is the matrix-valued counterpart of a Fourier series; matrix coefficients supply a complete orthogonal basis [6].

For $f, k \in L^2(G)$, define mathematical convolution as $(f * k)(g) = \int_G f(h)k(h^{-1}g) dh$. With the adjoint convention in (5.2), the order is

$$\widehat{f * k}(\lambda) = \widehat{k}(\lambda)\widehat{f}(\lambda). \quad (5.3)$$

Indeed, substituting $g = hu$ gives $\rho_\lambda(hu)^\dagger = \rho_\lambda(u)^\dagger \rho_\lambda(h)^\dagger$, which yields the two factors in that order. Alternative Fourier conventions give the opposite order; the definitions must be changed together. Noncommuting matrix blocks make the distinction essential.

The group convolution C_ψ taught in Part 2 uses the CNN cross-correlation convention. With the same integration measure in both operators, its scalar form satisfies $C_\psi f = f * k$ with $k(u) = \psi(u^{-1})$; do not silently replace this with $k = \psi$. On a finite group, Part 2 uses counting measure: its unnormalized sum is $|G|$ times the normalized-Haar operator used here. The companion code checks (5.3) on the noncommutative group of permutations of three objects. A cyclic-group test alone cannot expose a reversed product of scalar Fourier blocks.

Scope. The normalized compact-group formula applies to $SO(2)$, $SO(3)$, and finite rotation groups. The translation groups and $SE(d)$ are noncompact and have no normalized Haar probability measure. Their harmonic analysis requires different integration and representation machinery; the spatial constructions in Parts 2–4 avoid that prerequisite.

References

- [1] PyTorch contributors. *Conv2d documentation*. Official API documentation.
- [2] M. Geiger and T. Smidt. *e3nn: Euclidean Neural Networks*. 2022. arXiv:2207.09453.
- [3] e3nn contributors. *Convolution*. Official implementation guide.
- [4] escnn contributors. *escnn documentation*. Official documentation.
- [5] e3nn contributors. *Equivariance Testing*. Official testing guide.
- [6] D. Vogan. *Harmonic Analysis on Compact Groups*. MIT lecture notes.