

Part 2: Group Convolution

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.*

A convolutional layer reuses a detector at every position. A group convolution reuses it at every position *and pose*. We will implement this idea for quarter-turn rotations before giving the general equivariance proof. The required background is ordinary CNNs, matrix multiplication, and the meaning of an inverse transformation.

2.1 Start with the convolution you already use

Let an image be a function $f : \mathbb{Z}^2 \rightarrow \mathbb{R}^{C_{\text{in}}}$. Thus $f(x)$ is the column vector of input channels at pixel x . A kernel $\psi : \mathbb{Z}^2 \rightarrow \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$ acts by

$$(C_\psi f)(t) = \sum_{\delta \in \mathbb{Z}^2} \psi(\delta) f(t + \delta) = \sum_{x \in \mathbb{Z}^2} \psi(x - t) f(x). \quad (2.1)$$

Only finitely many offsets δ are used in a practical kernel. Channel summation is the matrix–vector multiplication in this equation. We omit the bias until discussing complete networks.

The CNN convention. Equation (2.1) is mathematically cross-correlation: it uses $f(t + \delta)$ rather than the $f(t - \delta)$ of mathematical convolution. CNN papers and libraries, including PyTorch [2], usually call this operation *convolution*. We follow that terminology: from here on, *group convolution* denotes C_ψ with the relative argument $g^{-1}h$. The two mathematical conventions differ by kernel inversion (a spatial flip for translations). Rotating a kernel and indexing a group must follow the chosen convention consistently; our formulas need no extra flip in `conv2d`.

If $(T_u f)(x) = f(x - u)$ shifts an image, then $C_\psi T_u f = T_u C_\psi f$: the same detector moves with the image. A generic learned filter does *not* similarly commute with image rotation. A horizontal edge detector and a vertical edge detector give different responses. The solution is to retain the responses of all rotated copies of a detector and let a rotation permute those responses.

Why retain orientation? Taking the maximum over orientations immediately can detect that an edge exists, but discards which way it points. Later layers may need that information to recognize a corner or a relative arrangement of parts. Group convolutions keep orientation as a feature-map index and share weights across its possible values. The construction of Cohen and Welling [1] provides this extension of ordinary CNNs.

2.2 A working example: translations and quarter-turns

2.2.1 The two rules needed to index the layer

Write $C_4 = \{0, 1, 2, 3\}$ for quarter-turn rotations, with addition modulo 4. Let R_r rotate a planar vector counterclockwise by $r\pi/2$. A position–rotation pair $g = (t, r)$ acts on a point by

$$gx = t + R_r x.$$

Rotating and translating in opposite orders usually give different results. Composing the corresponding actions therefore gives

$$(t, r)(u, s) = (t + R_r u, r + s), \quad (t, r)^{-1} = (-R_{-r} t, -r). \quad (2.2)$$

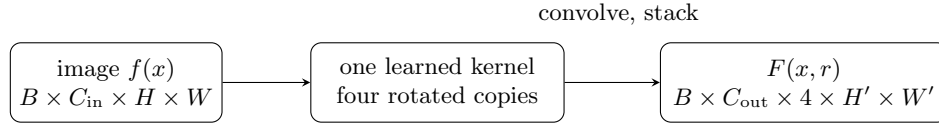
The rightmost transformation acts first. Rotation indices are always interpreted modulo 4. The collection of these pairs is the group $p4 = \mathbb{Z}^2 \rtimes C_4$. The symbol $\rtimes$ records that rotating a translation changes its direction; no further group theory is needed to implement the formulas below.

2.2.2 First layer: lift an image to position and orientation

For a learned spatial kernel ψ , define

$$(\mathcal{L}_\psi f)(t, r) = \sum_{x \in \mathbb{Z}^2} \psi(R_{-r}(x - t))f(x) = \sum_{\delta} \psi(R_{-r}\delta)f(t + \delta). \quad (2.3)$$

For each r , this is one ordinary CNN convolution with a rotated copy of the same kernel. Stack the four outputs. The resulting feature map $F(t, r)$ has position *and* orientation indices.



The orientation index is not four independently learned channels. It indexes four poses of each learned feature type. If a detector strongly responds at $(t, 0)$, rotating the input by one quarter-turn moves the corresponding response to $(R_1 t, 1)$. More precisely, for $a = (u, q)$,

$$(L_a F)(t, r) = F(R_{-q}(t - u), r - q). \quad (2.4)$$

This transformation combines a spatial warp with a cyclic permutation of the orientation axis. Rotating only the image plane of F is incorrect.

Check the permutation at the rotation center. At $t = 0$, a pure quarter-turn maps

$$[F(0, 0), F(0, 1), F(0, 2), F(0, 3)] \mapsto [F(0, 3), F(0, 0), F(0, 1), F(0, 2)].$$

Away from the center, read these values at the inverse-rotated spatial coordinate as well. This small example is useful when checking the sign of a cyclic shift in code.

What is being lifted? The image lives on the plane, while the output lives on $p4$, the space of position–rotation pairs. In group notation the plane is the quotient $p4/C_4$: rotations fixing the origin are ignored when specifying only a point. The output does not live on C_4 alone; it still has two spatial coordinates.

2.2.3 Later layers: mix relative positions and orientations

After lifting, a hidden feature map is $F : p4 \rightarrow \mathbb{R}^{C_{\text{in}}}$. The kernel now has an orientation argument, $\psi : p4 \rightarrow \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$. Use a relative transformation to index it:

$$(C_\psi F)(g) = \sum_{h \in p4} \psi(g^{-1}h)F(h). \quad (2.5)$$

The product $g^{-1}h$ describes input pose h in the coordinate frame of output pose g . Applying (2.2) gives

$$(t, r)^{-1}(x, s) = (R_{-r}(x - t), s - r),$$

and hence the implementable formula

$$(C_\psi F)(t, r) = \sum_{\delta \in \mathbb{Z}^2} \sum_{s=0}^3 \psi(R_{-r}\delta, s - r)F(t + \delta, s). \quad (2.6)$$

One learned kernel specifies which nearby features and *relative* orientations fit together. It is reused for every output orientation. For example, when $r = 1$, relative orientation 0 receives the input at $s = 1$, relative orientation 1 receives $s = 2$, and so on.

Why ordinary convolution on flattened channels fails. Flattening $C_{\text{in}} \times 4$ into ordinary channels is a useful computational step, but unconstrained weights for the $4C_{\text{out}}$ outputs generally break equivariance. The weights must be generated by the coupled spatial rotation *and* orientation permutation in (2.6).

2.3 An implementation recipe

Use channel-first tensors and a square, odd-sized $k \times k$ kernel centered at offset zero. With unit stride and suitable padding, $H' = H$ and $W' = W$. For quarter-turns, kernel rotation is an exact array permutation.

Layer	Input tensor	Learned base kernel tensor
Lifting	$[B, C_{\text{in}}, H, W]$	$[C_{\text{out}}, C_{\text{in}}, k, k]$
Group-to-group	$[B, C_{\text{in}}, 4, H, W]$	$[C_{\text{out}}, C_{\text{in}}, 4, k, k]$

1. **Generate the tied kernel bank.** For lifting, construct $K_r(\delta) = \psi(R_{-r}\delta)$. For a group-to-group layer, construct

$$K_r[c_{\text{out}}, c_{\text{in}}, s, \delta] = \psi[c_{\text{out}}, c_{\text{in}}, s - r, R_{-r}\delta].$$

Implement these operations as a spatial quarter-turn and a cyclic orientation-index shift. Verify the direction against (2.4); array row coordinates point down while usual Cartesian vertical coordinates point up.

2. **Call the existing convolution routine.** Stack output indices as (c_{out}, r) . For lifting, reshape the bank to $[4C_{\text{out}}, C_{\text{in}}, k, k]$. For a hidden layer, flatten input indices as (c_{in}, s) and reshape the bank to $[4C_{\text{out}}, 4C_{\text{in}}, k, k]$. Use this same ordering when reshaping the input and the output.
3. **Restore the orientation axis.** Reshape the result to $[B, C_{\text{out}}, 4, H', W']$. Add a bias of shape $[C_{\text{out}}]$, broadcast over orientation and position. Apply an orientation-compatible nonlinearity.
4. **Learn the base weights only.** The rotated bank is a differentiable function of the base tensor. Autodifferentiation sums gradients from all tied copies. Do not register four separate trainable filters and merely initialize them as rotations.

What does the sharing cost? Ignoring biases, lifting has $C_{\text{out}}C_{\text{in}}k^2$ parameters; a group-to-group layer has $4C_{\text{out}}C_{\text{in}}k^2$. An unconstrained convolution between $4C_{\text{in}}$ and $4C_{\text{out}}$ channels would have $16C_{\text{out}}C_{\text{in}}k^2$ parameters. Thus the symmetry reduces parameters relative to the same tensor sizes. It does not make four orientation maps free: activation storage and arithmetic increase relative to an ordinary CNN with only C channels. When comparing models, report channel counts, parameter counts, and compute separately.

2.4 The main guarantee: equivariance for every learned kernel

The practical index rules above are instances of one short result. The *left action* on a group feature map is $(L_a F)(h) = F(a^{-1}h)$: to find a transformed value, look up where it came from. This inverse is essential, especially when transformations do not commute.

Theorem 2.1 (Equivariance of group convolution) *Let G be a discrete group, and define*

$$(C_\psi F)(g) = \sum_{h \in G} \psi(g^{-1}h)F(h), \quad \psi(h) \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}.$$

Assume the sums converge absolutely at the arguments used. Then for every $a \in G$, $C_\psi L_a F = L_a C_\psi F$. The same statement holds on a locally compact group when the sum is replaced by an absolutely convergent integral with respect to a left Haar measure.

Proof: For the discrete case, substitute $h = az$. Left multiplication is a bijection, so it just relabels the terms:

$$\begin{aligned} (C_\psi L_a F)(g) &= \sum_h \psi(g^{-1}h)F(a^{-1}h) \\ &= \sum_z \psi(g^{-1}az)F(z) \\ &= \sum_z \psi((a^{-1}g)^{-1}z)F(z) \\ &= (C_\psi F)(a^{-1}g) = (L_a C_\psi F)(g). \end{aligned}$$

For the continuous case, the same substitution preserves a left Haar measure: $d\mu(az) = d\mu(z)$. Replace each sum by an integral. ■

What the theorem gives a practitioner. No symmetry loss or special training data are required for the layer identity. It holds for every value of the learned kernel. Weight sharing enforces this identity; it does not guarantee that the chosen symmetry is appropriate for the task, or that the resulting network has better accuracy.

The lifting layer obeys the corresponding identity. For images use $(T_a f)(x) = f(a^{-1}x)$, where the inverse acts on a point. For the lifted output use L_a on position–rotation pairs. Then $\mathcal{L}_\psi T_a = L_a \mathcal{L}_\psi$. Indeed, substituting $x = ay$ gives

$$(\mathcal{L}_\psi T_a f)(g) = \sum_y \psi(g^{-1}ay)f(y) = (\mathcal{L}_\psi f)(a^{-1}g).$$

Here the discrete planar action permutes pixels. The analogous continuous argument needs an invariant measure on the *image domain*, which is planar area for rigid motions.

2.5 Build a whole network without losing the guarantee

The network is equivariant if each layer respects the same prescribed feature transformation. For the regular, orientation-indexed features above:

- **Pointwise nonlinearities.** ReLU or another shared scalar nonlinearity commutes with the permutation of positions and orientations. This argument applies to these permutation features; it does not license componentwise ReLU on arbitrary vector or tensor representations.
- **Bias and normalization.** A separate bias for every orientation breaks the sharing. Tie bias and affine normalization parameters across orientations. For example, compute normalization statistics per learned channel over batch, space, and orientation, and use one scale and offset per learned channel. Ordinary batch normalization on $4C$ independently flattened channels does not enforce these constraints.
- **Residual connections.** Add features only when their group actions and orientation conventions agree. A shortcut changing the feature type must itself be equivariant.

- **Pooling.** Reducing orientation alone, $PF(t) = \sum_r F(t, r)$ or $\max_r F(t, r)$, removes the orientation axis but leaves a spatial map:

$$P(L_{(u,q)}F)(t) = (PF)(R_{-q}(t - u)).$$

This map is spatially equivariant, not an invariant scalar. A subsequent sum, mean, or maximum over a domain permuted by the transformation gives an invariant readout. For classification, pool at the end; for dense prediction, retain the required spatial output action.

2.5.1 What exact equivariance means on a finite tensor

The infinite-grid theorem and a finite implementation have different domains. The following checks separate algebraic mistakes from numerical and boundary effects.

1. **Boundary conditions.** Cropping an image after a translation discards information, so arbitrary shifts of a fixed crop need not commute with the layer. Circular padding gives a clean exact translation test on a periodic grid. For quarter-turn tests, use square grids and consistently centered rotations; symmetric zero padding with unit stride can preserve these rotations exactly. Rectangular crops generally are not preserved by quarter-turns.
2. **Sampling and stride.** A stride of d retains only one sampling phase. Generic one-pixel translations no longer act as integer shifts on the output. Equivariance can remain for translations by multiples of d , with a compatible output action. Anti-aliasing can reduce instability but does not restore the discarded phases by itself.
3. **Allowed rotations.** Quarter-turns and square-grid reflections permute pixels. A 45° rotation does not map $\mathbb{Z}^2$ to itself. Consequently a planar C_8 implementation obtained by interpolating rotated filters is generally approximate: composition of resampling operations need not equal resampling by the composed rotation. A continuous equivariance theorem does not remove this discretization error.

2.5.2 An equivariance unit test

Choose random inputs and weights; put the network in deterministic evaluation mode. For each allowed transformation a , compare two independent forward passes:

$$y_{\text{left}} = \Phi(T_a x), \quad y_{\text{right}} = T'_a \Phi(x), \quad \epsilon_{\text{eq}} = \frac{\|y_{\text{left}} - y_{\text{right}}\|_2}{\max(\|y_{\text{left}}\|_2, \|y_{\text{right}}\|_2, 10^{-12})}.$$

Use the appropriate input and output actions: the first layer receives an ordinary image, while a hidden layer receives spatially rotated *and orientation-permuted* features. Test all four rotations, several translations, and their compositions. In an exact finite-group setup the residual should be at numerical precision, with a tolerance appropriate to the dtype and accumulated sums. Also check gradients through the tied kernel bank. A low training loss is not a substitute for this test.

2.6 Continuous rotations: what replaces the sum?

For $SE(2) = \mathbb{R}^2 \rtimes SO(2)$, replace the four rotation indices by an angle $\theta \in [0, 2\pi)$. The action is still $(t, \theta)x = t + R_\theta x$. An integral requires a measure that assigns the same volume before and after a group transformation.

Haar measure is symmetry-compatible volume. A left Haar measure μ satisfies $\mu(aA) = \mu(A)$. Locally compact groups possess such a nonzero measure, unique up to a positive scale. The property actually used in the proof is

$$\int_G q(ah) d\mu(h) = \int_G q(h) d\mu(h).$$

Counting measure gives ordinary sums on discrete groups. On $SO(2)$, $d\theta/(2\pi)$ is normalized angular measure. On $SE(2)$ one may use $d^2t d\theta/(2\pi)$: translations preserve planar area, rotations have determinant one, and adding an angle preserves angular measure. The spatial factor has infinite total volume, so this is not a probability measure on all of $SE(2)$.

The continuous group convolution is

$$(C_\psi F)(g) = \int_G \psi(g^{-1}h)F(h) d\mu(h). \quad (2.7)$$

Theorem 2.1 applies directly. It needs only a *left* Haar measure; it does not require the group to be unimodular (a term meaning that left and right Haar measures agree).

Continuous lifting instead integrates over the image plane:

$$(\mathcal{L}_\psi f)(t, \theta) = \int_{\mathbb{R}^2} \psi(R_{-\theta}(x - t))f(x) d^2x. \quad (2.8)$$

Its output is indexed by position and angle even though the integral is over position only. In implementation, angle sampling and spatial quadrature approximate these integrals. Increasing the number of sampled orientations improves angular resolution but also increases memory and compute; exact continuous equivariance requires more than simply storing many rotated arrays. The following notes develop steerable kernels, which encode the transformation law in a basis rather than relying only on repeated raster rotations.

Implementation takeaway. Choose the desired input and output actions first. Generate all transformed filters from one base tensor, keep the orientation axis explicit, and test the complete layer under those actions. The group notation is a compact way of specifying the correct weight sharing; the computation is still a bank of ordinary CNN convolutions.

References

- [1] T. S. Cohen and M. Welling. *Group Equivariant Convolutional Networks*. ICML, 2016. Paper and supplementary material. The original construction and implementation of discrete G-CNNs.
- [2] PyTorch contributors. *Conv2d documentation*. Official API reference. Defines the channel, kernel, stride, padding, and cross-correlation conventions used by the underlying primitive.