Supplementary Material
Gauge Equivariant Convolutional Networks and the Icosahedral CNN

1. Recommended reading

For more information on manifolds, fiber bundles, connections, parallel transport, the exponential map, etc., we highly recommend the lectures by Schuller (2016), as well as the book Nakahara (2003) which explain these concepts very clearly and at a useful level of abstraction.

For further study, we recommend (Sharpe, 1997; Shoshichi Kobayashi, 1963; Husemöller, 1994; Steenrod, 1951; Wendl, 2008; Crane, 2014).

2. Mathematical Theory & Physics Analogy

From the perspective of the theory of principal fiber bundles, our work can be understood as follows. A fiber bundle E is a space consisting of a base space B (the manifold M in our paper), with at each point $p \in B$ a space F_p called the fiber at p. The bundle is defined in terms of a projection map $\pi : E \to B$, which determines the fibers as $F_p = \pi^{-1}(p)$. A principal bundle is a fiber bundle where the fiber F carries a transitive and free right action of a group G (the structure group).

One can think of the fiber F_p of a principal bundle as a (generalized) space of frames at p. Due to the free and transitive action of G on F_p, we have that F_p is isomoprhic to G as a G-space, meaning that it looks like G except that it does not have a distinguished origin or identity element as G does (i.e. there is no natural choice of frame).

A gauge transformation is then defined as a principal bundle automorphism, i.e. a map from $P \to P$ that maps fibers to fibers in a G-equivariant manner. Sometimes the automorphism is required to fix the base space, i.e. project down to the identity map via π. Such a B-automorphism will map each fiber onto itself, so it restricts to a G-space automorphism on each fiber.

Given a principal bundle P and a vector space V with representation ρ of G, we can construct the associated bundle $P \times_\rho V$, whose elements are the equivalence classes of the following equivalence relation on $P \times V$:

$$(p, v) \sim (pg, \rho(g^{-1})v). \tag{1}$$

The associated bundle is a fiber bundle over the same base space as P, with fiber isomorphic to V.

A (matter) field is described as a section of the associated bundle A, i.e. a map $\sigma : B \to A$ that satisfies $\pi \circ \sigma = 1_B$. Locally, one can describe a section as a function $B \to V$ (as we do in the paper), but globally this is not possible unless the bundle is trivial.

The group of automorphisms of P (gauge transformations) acts on the space of fields (sections of the associated bundle). It is this group that we wish to be equivariant to.

From this mathematical perspective, our work amounts to replacing the principal G bundle[1] $H \to H/G$ used in the work on regular and steerable G-CNNs of Cohen et al. (2018a;c), by another principal G bundle, namely the frame bundle of M. Hence, this general theory can describe in a unified way the most prominent and geometrically natural methods of geometrical deep learning (Masci et al., 2015; Boscaini et al., 2016), as well as all G-CNNs on homogeneous spaces.

Indeed, if we build a gauge equivariant CNN on a homogeneous space H/G (e.g. the sphereTo see this, note that the left action of H on itself (the total space of the principal G bundle) can be decomposed into an action on the base space H/G (permuting the fibers), and an action on the fibers (cosets) that factors through G (see e.g. Sec. 2.1 of (Cohen et al., 2018c)). The action on the base space preserves the local neighbourhoods from which we compute filter responses, and equivariance to the action of G is ensured by the kernel constraint. Since G-CNNs (Cohen et al., 2018a) and gauge equivariant CNNs employ the most general equivariant map, we conclude that they are indeed the same, for bundles $H \to H/G$. Thus, "gauge theory is all you need". (We plan to expand this argument in a future paper)

Most modern theories of physics are gauge theories, meaning they are based on this mathematical framework. In such theories, any construction is required to be gauge invariant (i.e. the coefficients must be gauge equivariant), for otherwise the predictions will depend on the way in which we choose to represent physical quantities. This logic applies not just to physics theories, but, as we have argued in the paper, also to neural networks and other models used in machine learning. Hence, it is only natural that the same mathematical framework is applicable in both fields.

[1] It is more common to use the letter G for the supergroup and H for the subgroup, but that leads to a principal H-bundle $G \to G/H$, which is inconsistent with the main text, where we use a principal G bundle. So we swap H and G here.

3. Deriving the kernel constraint

The gauge equivariant convolution is given by

$$(K \star f)(p) = \int_{\mathbb{R}^d} K(v)\rho_{\text{in}}(g_{p\leftarrow q_v})f(q_v)dv. \tag{2}$$

Under a gauge transformation, we have:

$$\begin{aligned} v &\mapsto g_p^{-1}v, & f(q_v) &\mapsto \rho_{\text{in}}(g_{q_v}^{-1})f(q_v), \\ w_p &\mapsto w_p g_p, & g_{p\leftarrow q_v} &\mapsto g_p^{-1} g_{p\leftarrow q_v} g_{q_v}. \end{aligned} \tag{3}$$

It follows that q_v is unchanged, because $q_v = \exp_p w_p v \mapsto \exp_p(w_p g_p)(g_p^{-1}v) = q_v$. Substituting the rest in the convolution equation, we find

$$\begin{aligned} &\int_{\mathbb{R}^d} K(g_p^{-1}v)\rho_{\text{in}}(g_p^{-1}g_{p\leftarrow q_v}g_{q_v})\rho_{\text{in}}(g_{q_v}^{-1})f(q_v)dv \\ &\quad = \int_{\mathbb{R}^d} K(g_p^{-1}v)\rho_{\text{in}}(g_p^{-1})\rho_{\text{in}}(g_{p\leftarrow q_v})f(q_v)dv \end{aligned} \tag{4}$$

Now if $K(g_p^{-1}v) = \rho_{\text{out}}(g_p^{-1})K(v)\rho_{\text{in}}(g_p)$ (i.e. K satisfies the kernel constraint), then we get

$$(K \star f)(p) \mapsto \rho_{\text{out}}(g_p^{-1})(K \star f)(p), \tag{5}$$

i.e. $K \star f$ transforms as a ρ_{out}-field under gauge transformations.

4. Additional information on experiments

4.1. MNIST experiments

Our main model consists of 7 convolution layers and 3 linear layers. The first layer is a scalar-to-regular gauge equivariant convolution layer, and the following 6 layers are regular-to-regular layers. These layers have $8, 16, 16, 24, 24, 32, 64$ output channels, and stride $1, 2, 1, 2, 1, 2, 1$, respectively.

In between convolution layers, we use batch normalization (Ioffe & Szegedy, 2015) and ReLU nonlinearities. When using batch normalization, we average over groups of 6 feature maps, to make sure the operation is equivariant. Any pointwise nonlinearity (like ReLU) is equivariant, because we use only trivial and regular representations realized by permutation matrices.

After the convolution layers, we perform global pooling over spatial and orientation channels, yielding an invariant representation. We map these through 3 FC layers (with $64, 32, 10$ channels) before applying softmax.

The other models are obtained from this one by replacing the convolution layers by scalar-to-regular + orientation pooling (S2R) or scalar-to-scalar (S2S) layers, or by disabling G-padding (NP) and/or kernel expansion (NE), always adjusting the number of channels to keep the number of parameters roughly the same.

The Spherical CNN (S2CNN) is obtained from the R2R model by replacing the S2R and R2R layers by spherical and $\mathrm{SO}(3)$ convolution layers, respectively, keeping the number of channels and strides the same. The Spherical CNN uses a different grid than the Icosahedral CNN, so we adapt the resolution / bandwidth parameter B to roughly match the resolution of the Icosahedral CNN. We use $B = 26$, to get a spherical grid of size $2B \times 2B = 52 \times 52$. Note that this grid has higher resolution at the poles, and lower resolution near the equator, which explains why the S2CNN performs a bit worse when trained on rotated data instead of digits projected onto the north-pole. To implement strides, we reduce the output bandwidth by 2 at each layer with stride.

The spherical convolution takes a scalar signal on the sphere as input, and outputs scalar signals on $\mathrm{SO}(3)$, which is analogous to a regular field over the sphere. $\mathrm{SO}(3)$ convolutions are analogous to $R2R$ layers. We note that this is a stronger Spherical CNN architecture than the one used by (Cohen et al., 2018b), which achieves only 96% accuracy on spherical MNIST.

The models were trained for 60 epochs, or 1 epoch of the $60\times$ augmented dataset (where each instance is transformed by each icosahedron symmetry $g \in \mathcal{I}$, or by a random rotation $g \in \mathrm{SO}(3)$).

4.2. Climate experiments

For the climate experiments, we used a U-net with regular-to-regular convolutions. The first layer is a scalar-to-regular convolution with 16 output channels. The downsampling path consists of 5 regular-to-regular layers with stride 2, and $32, 64, 128, 256, 256$ output channels. The downsampling path takes as input a signal with resolution $r = 5$ (i.e. 10242 pixels), and outputs one at $r = 0$ (i.e. 12 pixels).

The decoder is the reverse of the encoder in terms of resolution and number of channels. Upsampling is performed by bilinear interpolation (which is exactly equivariant), before each convolution layer (which uses stride 1). As usual in the U-net architecture, each layer in the upsampling path takes as input the output of the previous layer, as well as the output of the encoder path at the same resolution.

Each convolution layer is followed by equivariant batch-norm and ReLU.

The model was trained for 15 epochs with batchsize 15.

4.3. 2D-3D-S experiments

For the 2D-3D-S experiments, we used a residual U-Net with the following architecture.

The input layer is a scalar-to-regular layer with 8 channels, followed by batchnorm and relu. Then we apply 4 residual blocks with 16, 32, 64, 64 output channels, each of which

uses stride=2. In the upsampling stream, we use 32, 16, 8, 8 channels, for the residual blocks, respectively. Each upsampling layer receives input from the corresponding downsampling layer, as well as the previous layer. Upsampling is performed using bilinear interpolation, and downsampling by hexagonal max pooling.

The input resolution is $r = 5$, which is downsampled to $r = 1$ by the downsampling stream.

Each residual block consists of a convolution, batchnorm, skipconnection, and ReLU.

5. Computational complexity analysis of Spherical and Icosahedral CNNs

One of the primary motivations for the development of the Icosahedral CNN is that it is faster and more scalable than Spherical CNNs as originally proposed. The Spherical CNN as implemented by (Cohen et al., 2018b) uses feature maps on the sphere S^2 and rotation group $\mathrm{SO}(3)$ (the latter of which can be thought of a regular field on the sphere), sampled on the SOFT grids defined by (Kostelec & Rockmore, 2007), which have shape $2B \times 2B$ and $2B \times 2B \times 2B$, respectively (here B is the bandwidth / resolution parameter). Specifically, the grid points are:

$$\begin{aligned} \alpha_{j_1} &= \frac{2\pi j_1}{2B}, \\ \beta_k &= \frac{\pi(2k+1)}{4B}, \\ \gamma_{j_2} &= \frac{2\pi j_2}{2B}, \end{aligned} \tag{6}$$

where (α_{j_1}, β_k) form a spherical grid and $(\alpha_{j_1}, \beta_k, \gamma_{j_2})$ form an $\mathrm{SO}(3)$ grid (for $j_1, k, j_2 = 0, \dots 2B-1$). These grids have two downsides.

Firstly, because the SOFT grid consists of equal-lattitude rings with a fixed number of points (2B), the spatial density of points is inhomogeneous, with a higher concentration of points near the poles. To get a sufficiently high sampling near the equator, we are forced to oversample the poles, and thus waste computational resources. For almost all applications, a more homogeneous grid is more suitable.

The second downside of the SOFT grid on $\mathrm{SO}(3)$ is that the spatial resolution ($2B \times 2B$; α, β) and angular resolution ($2B$; γ) are both coupled to the same resolution / bandwidth parameter B. Thus, as we increase the resolution of the spherical image, the number of rotations applied to each filter is increased as well, which is undesirable.

The grid used in the Icosahedral CNN addresses both concerns. It is spatially very homogeneous, and we apply the filters in 6 orientations, regardless of spatial resolution.

The generalized FFT algorithm used by (Cohen et al., 2018a) only works on the SOFT grid. Generalized FFTs for other grids exist (Kunis & Potts, 2003), but are harder to implement. Moreover, although the (generalized) FFT can improve the asymptotic complexity of convolution for large input signals, the FFT-based convolution actually has worse complexity if we assume a fixed filter size. That is, the $\mathrm{SO}(3)$ convolution (used in most layers of a typical Spherical CNN) has complexity $O(B^3 \log B)$ which compares favorably to the naive $O(B^6)$ spatial implementation. However, if we use filters with a fixed (and usually small) size, the complexity of a naive spatial implementation reduces to $O(B^3)$, which is slightly better than the FFT-based implementation. Furthermore, because the Icosahedral CNN uses a fixed number of orientations per filter (i.e. 6), its computational complexity is even better: it is linear in the number of pixels of the grid, and so comparable to $O(B^2)$ for the SOFT grid.

The difference in complexity is clearly visible in Figures 1 and 2, below. On the horizonal axis, we show the grid resolution r for the icosahedral grid $\mathcal{H}_r$ (for the spherical CNN, we a SOFT grid with roughly the same number of spatial points). On the vertical axis, we show the amount of wallclock time (averaged over 100 runs) and memory required to run an $\mathrm{SO}(3)$ convolution (S2CNN) or a regular-to-regular gauge equivariant convolution (IcoNet) at that resolution. Note that since the number of grid pionts is exponential in r, and we use a logarithmic vertical axis, the figures can be considered log-log plots. Both plots were generated by running a single regular to regular convolution layer at the corresponding resolution r with 12 input and output channels. For a fair comparison with IcoCNNs we chose filter grid parameters so3_near_identity_grid(n_alpha=6, max_beta=np.pi/16, n_beta=1, max_gamma=2*np.pi, n_gamma=6) for the spherical convolution layer. To guarantee a full GPU utilization, results were measured on an as large as possible batch size per datapoint and subsequently normalized by that batch size.

As can be seen in Figure 1, the computational cost of running the S2CNN dramatically exceeds the cost of running the IcoCNN, particularly at higher resolutions. We did not run the spherical CNN beyond resolution $r = 6$, because the network would not fit in GPU memory even when using batch size 1.

As shown in Figure 2, the Spherical CNN at resolution $r = 6$ uses about 10GB of memory, whereas the Icosahedral CNN uses only about 1GB. Since we used the maximum batch size with subsequent normalization for each resolution the reported memory consumption mainly reflects the memory cost of the feature maps, not the constant memory cost of the filter banks.

Aside from the theoretical asymptotic complexity, the actual computational cost depends on important implementation

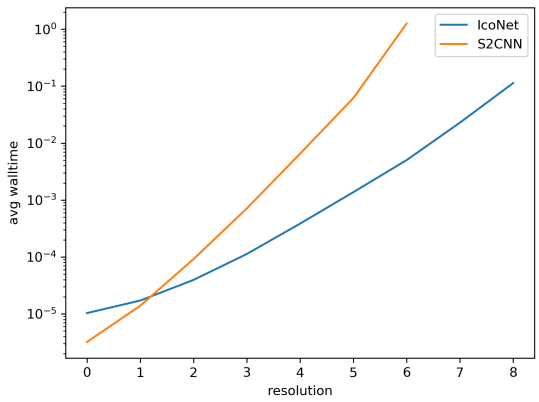

Figure 1. Comparison of computational cost (in wallclock time) of Icosahedral CNNs (IcoNet) and Spherical CNNs (S2CNN, (Cohen et al., 2018b)), at increasing grid resolution r.

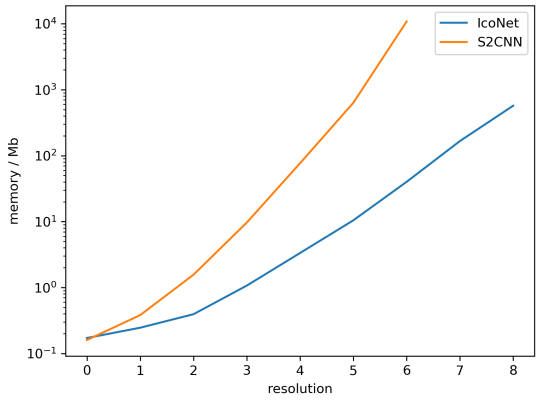

Figure 2. Comparison of memory usage of Icosahedral CNNs (IcoNet) and Spherical CNNs (S2CNN, (Cohen et al., 2018b)), at increasing grid resolution r.

details. Because the heavy lifting of the Icosahedral CNN is all done by a single conv2d call, our method benefits from the extensive optimization of, and hardware support for this operation. By contrast, the generalized FFT used in the original Spherical CNN uses a conventional FFT, as well as matrix multiplcations with spectral matrices of size $1, 3, 5, 7, \ldots, 2L+1$ (the $\mathrm{SO}(3)$ spectrum is matrix-valued, instead of the scalar valued spectrum for commutative groups). Implementing this in a way that is fast in practice is more challenging.

A final note on scalability. For some problems, such as the analysis of high resolution global climate or weather data, it is unlikely that even a single feature map will fit in memory at once on current or near-term future hardware. Hence, it may be useful to split large feature maps into local charts, and process each one on a separate compute node. For the final results to be globally consistent (so that each compute node makes equivalent predictions for points in the overlap of charts), gauge equivariance is indispensable.

6. Details on G-Padding

In a conventional CNN, one has to pad the input feature map in order to compute an output of the same size. Although the icosahedron itself does not have a boundary, the charts do, and hence require padding before convolution. However, in order to faithfully simulate convolution on the icosahedron via convolution in the charts, the padding values need to be copied from another chart instead of e.g. padding by zeros. In doing so, a gauge transformation may be required.

To see why, note that the conv2d operation, which we use to perform the convolution in the charts, implicitly assumes that the signal is expressed relative to a fixed global gauge in the plane, namely the frame defined by the x and y axes. This is because the filters are shifted along the x and y directions by conv2d, and as they are shifted they are not rotated. So the meaning of "right" and "up" doesn't change as we move over the plane; the local gauge at each position is aligned with the global x and y axes.

Hence, it is this global gauge that we must use inside the charts shown in Figure 4 (right) of the main paper. It is important to note that although all frames have the same numerical expression $e_1 = (1, 0), e_2 = (0, 1)$ relative to the x and y axes, the corresponding frames on the icosahedron itself are different for different charts. Since feature vectors are represented by coefficients that have a meaning only relative to a frame, they have a different numerical expression in different charts in which they are contained. The numerical representations of a feature vector in two charts are related by a gauge transformation.

To better understand the gauge transformation intuitively, consider a pixel p on a colored edge in Fig. 4 of the main paper, that lies in multiple charts. Now consider a vector attached at this pixel (i.e. in T_pM), pointing along the colored edge. Since the colored edge may have different orientations when pictured in different charts, the vector (which is aligned with this edge) will also point in different directions in the charts, when the charts are placed on the plane together as in Figure 4. More specifically, for the choice of charts we have made, the difference in orientation is always one "click", i.e. a rotation by plus or minus $2\pi/6$. This is the gauge transformation $g_{ij}(p)$, which describes the transformation at p when switching between chart i and j.

The transformation $g_{ij}(p)$ acts on the feature vector at p via the matrix $\rho(g_{ij}(p))$, where ρ is the representation of $G = C_6$ associated with the feature space under consideration. In this work we only consider two kinds of representations:

scalar features with $\rho(g) = 1$, and regular features with ρ equal to the regular representation:

$$\rho(2\pi/6) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{7}$$

That is, a cyclic permutation of 6 elements. Since $2\pi/6$ is a generator of C_6, the value of ρ at the other group elements is determined by this matrix: $\rho(k \cdot 2\pi/6) = \rho(2\pi/6)^k$. If the feature vector consists of multiple scalar or regular features, we would have a block-diagonal matrix $\rho(g_{ij}(p))$.

We implement G-padding by indexing operations on the feature maps. For each position p to be padded, we precompute $g_{ij}(p)$, which can be $+1 \cdot 2\pi/6$ or 0 or $-1 \cdot 2\pi/6$. We use these to precompute four indexing operations (for the top, bottom, left and right side of the charts).

References

Boscaini, D., Masci, J., Rodolà, E., and Bronstein, M. M. Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS*, 2016.

Cohen, T., Geiger, M., and Weiler, M. A General Theory of Equivariant CNNs on Homogeneous Spaces. 2018a.

Cohen, T. S., Geiger, M., Koehler, J., and Welling, M. Spherical CNNs. In *ICLR*, 2018b.

Cohen, T. S., Geiger, M., and Weiler, M. Intertwiners between Induced Representations (with Applications to the Theory of Equivariant Neural Networks). 2018c.

Crane, K. Discrete Differential Geometry: An Applied Introduction. 30664:1–6, 2014.

Husemöller, D. *Fibre Bundles*. Number 20 in Graduate Texts in Mathematics. Springer-Verlag, New York, 3rd ed edition, 1994. ISBN 978-0-387-94087-8.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167 [cs]*, Feb 2015. URL http://arxiv.org/abs/1502.03167. arXiv: 1502.03167.

Kostelec, P. J. and Rockmore, D. N. SOFT: SO(3) Fourier Transforms. 2007. URL https://www.cs.dartmouth.edu/~geelong/soft/soft20_fx.pdf.

Kunis, S. and Potts, D. Fast spherical Fourier algorithms. 161:75–98, 2003. ISSN 0377-0427.

Masci, J., Boscaini, D., Bronstein, M. M., and Vandergheynst, P. Geodesic convolutional neural networks on riemannian manifolds. *ICCVW*, 2015.

Nakahara, M. *Geometry, Topology, and Physics*. 2003. ISBN 978-0-7503-0606-5.

Schuller, F. Lectures on the Geometrical Anatomy of Theoretical Physics, 2016.

Sharpe, R. W. *Differential Geometry: Cartan's Generalization of Klein's Erlangen Program*. 1997.

Shoshichi Kobayashi, K. N. *Foundations of Differential Geometry (Volume 1)*. 1963.

Steenrod, N. *The Topology of Fibre Bundles*. 1951.

Wendl, C. *Lecture Notes on Bundles and Connections*. 2008. URL https://www.mathematik.hu-berlin.de/~wendl/connections.html.