Nonlinearities in Steerable SO(2)-Equivariant CNNs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Invariance under symmetry is an important problem in machine learning. Our paper looks specifically at equivariant neural networks where transformations of inputs yield homomorphic transformations of outputs. Here, steerable CNNs have emerged as the standard solution. An inherent problem of steerable representations is that general nonlinear layers break equivariance, thus restricting architectural choices. Our paper applies harmonic distortion analysis to illuminate the effect of nonlinearities on Fourier representations of $SO(2)$. We develop a novel FFT-based algorithm for computing representations of non-linearly transformed activations while maintaining band-limitation. It yields exact equivariance for polynomial (approximations of) nonlinearities, as well as approximate solutions with tunable accuracy for general functions. We apply the approach to build a fully $E(3)$-equivariant network for sampled 3D surface data. In experiments with 2D and 3D data, we obtain results that compare favorably to the state-of-the-art in terms of accuracy while permitting continuous symmetry and exact equivariance.

1 Introduction

Modeling of symmetry in data, i.e., the invariance of properties under classes of transformations, is a cornerstone of machine learning: Invariance of statistical properties over samples is the basis of any form of generalization, and the prior knowledge of additional symmetries can be leveraged for performance gains. Aside from data efficiency prospects, some applications require exact symmetry. For example, in computational physics, symmetry of potentials and force fields is directly linked to conservation laws, which are important for example for the stability of simulations.

In deep neural networks, (discrete) translational symmetry over space and/or time is exploited in many architectures and is the defining feature of convolutional neural networks (CNNs) and their successors. In most applications, we are typically interested in invariance (e.g., classification remains unchanged) or co-variance (e.g., predicted geometry is transformed along with the input). Formally, this goal is captured under the more general umbrella of equivariance [5]:

Let $f : X \to Y$ be a function (e.g., a network layer) that maps between vector spaces X, Y (e.g., feature maps in a CNN). Let G be a group and let (in slight abuse of notation) $g \circ v$ denote the application of the action of group element g on a vector v. f is called *equivariant*, iff:

$$\forall g \in G : f(g \circ v) = h(g) \circ f(v), \tag{1}$$

where $h : G \mapsto G'$ is a group homomorphism mapping into a suitable group G'. Informally speaking, the effect of a transformation on the input should have an effect on the output that has (at least) the same algebraic structure. Invariance ($h \equiv 1_{G'}$) and covariance ($h = id_{G \to G'}$) are special cases.

G-CNNs: To make current CNN architectures (consisting of linear layers and nonlinearities) equivariant, the standard (and also most general, see [32]) approach are *group convolutional networks* (G-CNNs) [5], which conceptually boil down to just applying all transformations $g \in G$ to filters,

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

correlating the result with the data, and storing the results. Typically, G will be a continuous, compact Lie group such as $SO(d)$ (our paper focuses on $SO(2)$). To avoid infinite costs, results are band-limited, stored as coefficients of a truncated Fourier basis on G. Simultaneously, the Fourier coefficients provide a linear representation of a subgroup of G and thus exact equivariance in the sense of Eq. 1. Using such a basis to directly construct sets of filters yields steerable filter banks [6], were each filter outputs a whole vector of coefficients that represent functions on G.

Nonlinearities: Unfortunately, band-limiting interferes in non-trivial ways with network architecture, as an application of standard nonlinearities such as $\mathrm{ReLU}, \tanh$, or even simple nonlinear polynomials to the Fourier coefficients will break equivariance. Multiple solutions to this problem have been proposed [31]: Multiplicative non-linearities [32; 18] as in tensor networks keep equivariance but behave differently from traditional non-linearities and therefore cannot be used as a drop-in in classical CNN architectures. Complex nonlinearities such as $\mathbb{C}$-ReLU that only act nonlinearly on the magnitude of Fourier coefficients [35; 34] also keep perfect equivariance but are less expressive as they do not permit non-linear operations on the phase information. A recent study Weiler and Cesa [31] shows that simple discretized rotations [5], which do not require architectural adaptations but provide only approximate equivariance, yield the best practical results in image classification tasks.

SO(2)-equivariance: The goal of our paper is to clarify the effect of non-linearities on Fourier-domain representations. We restrict ourselfs to the case of $SO(2)$, which permits the application of standard harmonic distortion analysis [1]. Our goal is to maintain a band-limited representation of a function on $SO(2)$ (corresponding to a fixed angular resolution) and efficiently compute the a band-limited Fourier-representation after application of a non-linearity. We obtain an exact algorithm for polynomial nonlinearities with a computational overhead of $\mathcal{O}(D \log D)$ for degree D. For general nonlinearities, we can formally motivate a quick convergence, which we validate in numerical experiments. The effect of this overhead in practice is usually minor.

SE(3)-equivariant surface networks: While the limitation to $SO(2)$ might appear restrictive, it is still important for many problems in processing image and geometric data: Adding translational invariance ($E(2)$-equivariance) is easy, and we also apply our representation to surface data with normal information, extending ideas of Wiersma et al. [34] to a fully $E(3)$-equivariant network

We evaluate our networks on example benchmarks for 2D image and 3D object recognition. We obtain invariant results and equivariant intermediate representations up to numerical precision for polynomial nonlinearities (double checking the formal guarantees) and low-error approximations in with general nonlinearities such as $\tanh$,ReLU,ELU [4] at reasonable overhead (less than 20%, depending on approximation quality). Classification accuracy on MNIST-rot and ModelNet-40 [36] is on par with the state-of-the-art for ReLU and slightly reduced for low-degree polynomials [12].

Our main contributions are (i) a simple analytical model for the effect of non-linearities on Fourier representations in $SO(2)$-equivariant networks and (ii) an efficient algorithm for applying nonlinearities. It is provably exact for polynomials and empirically yields good approximations for common non-polynomial functions. This permits, for the first time, a the usage of standard CNN architectures with common nonlinearities without compromising equivariance.

2 Related Work

Equivariant networks: The are various approaches to achieving equivariance or invariance to rotations and other transformations in CNNs. The first approaches focused on exact C_4-equivariance of rotated images [19] by applying the same network to rotated copies of the input (with shared weights), followed by some invariant operation (e.g. max-pooling), after which the feature maps become invariant to input rotations. More advanced architectures allow the network to "see" filter responses of the previous layer to different rotations of the inputs by adding weights for those inputs in a way that does not break equivariance [5; 6; 7]. More fine-grained equivariance can be achieved by representing the image on a hexagonal grid [15], or by using *steerable filters* [33] and Fourier representations [35]. A comprehensive study that puts these different approaches into a common framework is provided by Weiler and Cesa [31].

3D data: While early neural network models for 3D data operated on regular grids similar to their 2D counterparts, alternative methods for processing 3D data with neural networks have quickly emerged to avoid the overhead. Some methods rely on data reduction through lower dimensional projections,

such as generating rendered images (sometimes from multiple perspectives to improve performance and approximate equivariance) and feeding them into into classical 2D-CNNs [27]. Other methods employ projections on spherical surfaces, on which convolution operations can be defined. These surfaces can be represented by a sampling (e.g. CSGNN[10] uses an icosahedral sampling) or a spherical harmonics basis [8].

Graph-based networks perform convolutions on a connectivity graph. For example, SchNet [25] is a popular architecture for predicting molecular properties, operating on the graph of molecular bonds. Another example is MeshCNN [13], which defines convolution operations on vertices of a 3D triangle mesh and uses a vertex merging as pooling operation. Other methods work directly on the intrinsic geometry of manifolds[34]. Such methods often have the advantage of being naturally equivarient to rotations or translations, as they work on intrinsic properties of an object. However, they often need to be tailored to a specific type of data.

Point based architectures have become a popular alternative. Early models like PointNet [23] and follow-ups [24; 20] directly used the coordinates of 3D data as network input. Later, Point Convolutional Neural Networks [2] emerged as an attempt to generalize the grid-based CNNs architecture to a spatial convolution on point clouds, using a radial Gaussian filter basis with fixed or trainable offset values. Kernel Point Convolutions (KPConv [29]) improve on this by introducing correlation functions which define the interaction of nearby points and limiting the distance of interactions to reduce overhead.

There are various ways of transforming these architectures to rotationally invariant models. One way is use layers that produce rotationally invariant features, which can then be processed further without restrictions, for example by aligning the inputs to the convolutional filters. This approach is taken by GCANet [37], and by MA-KPConv [28], which extends the KPConv model by using multiple alignments for the filters. Another approach is to apply an *invariant map* as the final function of each layer, as in Spherical Harmonics Networks (SPHnet) [22], which calculate activations in a spherical harmonic basis and produce invariant output by taking the norm over coefficients with identical degree. Various papers generalize the notion of steerable filters to 3-dimensional data [32; 30].

Our $SE(3)$-architecture is probably closest to the method of Wiersma et al. [34], which differs from the methods named above by giving each feature vector its own local reference frame, which is aligned to the surface normal and one arbitrarily chosen perperdicular direction. Equivariance is guaranteed by using parallel transport along the surface to align the reference frames of different features. In contrast to this, we skip the parallel transport step and simply rotate the normal vectors onto each other to find a suitable alignment of the local coordinate systems.

Our main analytical tool is harmonic distortion analysis. Originally developed in physics and engineering [9], Ali Mehmeti-Göpel et al. [1] have recently applied it to the problem of understanding the trainability of deep networks. In our case, we study how our linear representations of $SO(2)$ are affected by using a similar transformation with respect to an angular variable α.

3 SO(2)-Equivariant Networks

We start with a very brief recap of $SO(2)$-equivariant steerable networks [6]. We formulate the approach in terms of band-limited angular functions rather than steerable filter banks. This is merely a transposed view but facilitates the discussion in Section 4.1.

The starting point is a network layer $f^{(l)}, l \in \{1...L\}$ that maps functions $x^{(l-1)}$ to functions $x^{(l)}$, both having the domain $SO(2)$ and vector-valued output. We use an arc-length parametrization

$$x^{(l)} : \mathbb{R} \to \mathbb{R}^{d_l} \quad \text{with periodicity} \quad x^{(l)}(\alpha + 2\pi) = x^{(l)}(\alpha) \text{ for all } \alpha \in \mathbb{R} \tag{2}$$

To make layer activation functions representable in finite memory, we assume that each is band-limited to a maximum frequency of $K^{(l)}$ and thus can be represented by a complex Fourier series

$$x^{(l)}(\alpha) = \sum_{k=-K_l}^{K_l} \mathbf{z}_k^{(l)} \cdot e^{ik\alpha}, \qquad \mathbf{z}_k^{(l)} \in \mathbb{C}^{d_l}, \;\; \mathbf{z}_k^{(l)} = \overline{\mathbf{z}}_{-k}^{(l)}. \tag{3}$$

The conjugation symmetry holds because we are representing real functions. In a concrete implementation, we therefore store only half of the coefficients. According to the sampling theorem [11], such

functions can be represented exactly by a uniform sampling with $2K_l + 1$ samples, and the mapping between the two discrete representations of $2K_l + 1$ coefficients each is a unitary bijection. The layer function $f^{(l)}$ is, as usual, a concatenation of a linear function $W^{(l)}$ and a point-wise non-linearity $\varphi : \mathbb{R} \to \mathbb{R}$ that is applied to all angles and all output dimensions:

$$\big[[f^{(l)}(x^{(l-1)})](\alpha)\big]_i = \varphi\big(\big[[W^{(l)}(x^{(l-1)})](\alpha)\big]_i\big) \tag{4}$$

Above, we use "$[\,\cdot\,]_i$" to denote indexing of vector-valued function outputs.

We now construct linear $W^{(l)}$ that are equivairant under rotations. In our representation, these are cyclic shifts, i.e., equivariance can be expressed as $W(x(\alpha + T)) = (W \circ x)(\alpha + T))$ for all $T \in \mathbb{R}$. This means that W is a shift-invariant linear operator. Signal theory [11] tells us that these correspond do convolution of the input function with a kernel $w_i^{(l)} : [0, 2\pi] \to \mathbb{R}^{d_{l-1}}$. A subtlety: Considering input angles α and output angles β, we can introduce a dependency of $w_i^{(l)} = w_{i,\beta}^{(l)}$ on the output parameter while maintaining equivariance wrt. α. Weiler and Cesa [31] show that this is not only the most case and but also provides non-trivial performance benefits in practice:

$$[W^{(l)}(x^{(l-1)})]_i(\beta) := \int_0^{2\pi} \langle x^{(l-1)}(\alpha), w_{i,\beta}^{(l)}(\alpha - \beta)\rangle d\alpha \tag{5}$$

Due to the convolution theorem, this can be more conveniently written as a point-wise multiplication of the output Fourier coefficients $\mathbf{z}_k^{(l-1)}$ with the coefficients $\mathbf{q}_{k,k',i,j}^{(l)}$ of the a Fourier-series of $w_i^{(l)}$

$$z_{k',i}^{(l)} = \sum_{k=1}^{K_{l-1}} \sum_{j=1}^{d_{l-1}} q_{k,k',i,j}^{(l)} \cdot z_{k,j}^{(l-1)}. \tag{6}$$

The index k refers to the input and k' to the output frequency, and i and j the input/output feature indices. We use the $K_l \times K_{l-1} \times d_l \times d_{l-1}$-tensor $\mathbf{q}^{(l)}$ directly as trainable parameters for each layer. Overall, the application of a W to an x is structurally simple, requiring only a (complex) linear map of the input Fourier coefficients. Issues, however, arise when trying to apply the nonlinearity φ.

4 Nonlinearities in $SO(2)$-Equivariant Networks

In the continuous case, Eq. 5 settles the problem: Equivariance is still holds (trivially) if the non-linearity φ is applied point-wise, at every angle, as defined in Eq. 4. The issue is that this requires not only a transformation back from the frequency into the angular domain, but also a re-encoding, which involves a continuous integral that requires a potentially expensive numerical calculation: Unlike linear mappings, a non-linear can create higher-order frequencies, called "harmonics".

4.1 Non-Linearities Create Harmonics

In order to understand this effect, we apply harmonic distortion analysis [9; 1]. To simplify the notation, we will in the following drop the layer and feature index and denote by $z_k, k \in \{-K, ...K\}$ the Fourier coefficients of the *preactivation* $[W^{(l)}(x^{(l)})]_i$ of a single, 1D feature channel at a fixed layer. We denote the whole series as $\mathrm{z} = [z_{-K}, ..., z_K]$. For the analysis, we also start by assuming that φ is a polynomial of finite degree D:

$$\varphi(x) = \sum_{j=0}^{D} t^j x^j \tag{7}$$

Plugging in the corresponding Fourier series yields:

$$f(\alpha) \;=\; \varphi\left(\sum_{k=-K}^{K} z_k e^{ik\alpha}\right) = \sum_{j=1}^{D} t_j \cdot \sum_{k_1,\ldots,k_j=-K}^{K} z_{k_1} \cdots z_{k_j} e^{i(k_1+\cdots+k_j)\alpha}, \tag{8}$$

The convolution theorem now converts the point-wise multiplication into a convolution of the spectra in the Fourier domain: The series $\mathrm{z}' \in \mathbb{C}^{\mathbb{Z}}$ of output Fourier coefficients is given by

$$\mathrm{z}' = t_0 \;+\; t_1\mathrm{z} \;+\; t_2(\mathrm{z} \otimes \mathrm{z}) \;+\; t_3(\mathrm{z} \otimes \mathrm{z} \otimes \mathrm{z}) \;+\; \cdots \;+\; t_D(\mathrm{z} \otimes \cdots \otimes \mathrm{z}) \tag{9}$$

where "$\otimes$" denotes the discrete convolution

$$[\mathrm{z} \otimes \mathrm{w}]_k := \sum_{m \in \mathbb{Z}} z_m \cdot w_{k-m}. \tag{10}$$

As expected, an application of a non-linear function could potentially spread the spectrum towards higher frequencies. Inputs band-limited to frequency K yield outputs will band-limited to KD.

We can use this observation to construct efficient algorithms for computing z' from z. Our goal is to compute the first K_l Fourier coefficients efficiently from the K_{l-1} Fourier coefficients of layer $l-1$. The linear layer corresponds to a simple complex matrix-vector multiplication. However, applying the activation function is not trivial, due to the presence of negative frequencies in the Fourier series, that lead to a mixing of high and low frequency components. Further, a naive computation on only K coefficients would introduce aliasing effects that break equivariance.

4.2 Exact Equivariance: Polynomial NonLinearities

Direct Convolution: The simplest correct solution is to directly and iteratively evaluate the discrete convolution in Eq. 9 D-times, requiring $\mathcal{O}(\sum_{j=1}^{D} K_{l-1} \cdot j K_{l-1}) = \mathcal{O}(K_{l-1}^2 D^2)$ time and $\mathcal{O}(DK_{l-1})$ temporary memory, with only $\mathcal{O}(K_l)$ values being kept in the end.

FFT-Based Algorithm: The direct algorithm becomes inefficient for higher-order polynomials. The alternative is to evaluate $\varphi(f(\alpha))$ directly in the angluar domain, which requires an inverse Fast Fourier Transformation (IFFT), application of φ and an forward FFT. For large D, the point-wise application of φ in the anbular domain is obviously more efficient than the Fourier-domain convolution. However, in order to maintain equivariance, we need to make sure to sample adequately.

Counting the non-zero coefficients in Eq. 10 shows us immediately that $2DK_{l-1} + 1$ Fourier coefficients, i.e., a Fourier expansion up to frequency DK_{l-1}, is sufficient for an exact evaluation of a degree D polynomial φ: Arising from a D-fold convolution in the Fourier domain, the signal is band-limited accordingly. The sampling theorem then translates this to equidistant sampling at $2DK_{l-1} + 1$ discretization points in the angular domain (the corresponding discrete FFT becomes a bijection). This gives us an asymptotically more efficient algorithm with run-time $\mathcal{O}(DK_g \log DK_g)$ and memory $\mathcal{O}(DK_g)$ that is still exact. As the discrete Fourier transformation is also a unitary map, we can also expect favorable numerical properties. For low orders D, direct covolution might be slightly more efficient than the FFT-based approach. Nonetheless, our current implementation uses only the FFT variant, for simplicity, and also because we observe that, anyways, the overhead of applying the non-linearities is minor in the overall computational costs in our networks.

Practical application: In practice, we use polynomial non-linearities of moderate degree, as high-order polynomials become unstable [12] (computational costs of increasing D are not a limiting factor). A problem of is that any polynomial of degree $D \geq 1$ diverges asymptotically, thus making training unstable. We address this by clip the ℓ_1-norm of the Fourier coefficients at a maximum value, keeping $||z||_1 \leq c$ with c chosen bounding the range the polynomial was designed for. The ℓ_1-norm is an upper bound of the maximum value $x(\alpha)$ for all α that is also tight, as can be seen by an example Fourier series with aligned complex phases such that $|x(0)| = \sum_{k=-K}^{K} |z_k|$.

4.3 Approximation: Oversampled FFT

For non-polynomial nonlinearities φ, the FFT-based algorithm can still be used for approximate evaluation. We still map to the angular domain and back via FFT, using D-fold oversampling, but D now becomes a user-parameter. In the general case, the spectrum will not be band-limited, but decaying (the Fourier series converges for functions of bounded variation, which certainly applies to post-activations in all practical networks). Thus the truncated FFT will create (non-equivariant) aliasing artifacts that should vanish with increasing D.

The convergence behavior depends on the fall-off of the Fourier spectrum for a non-linearity that is not bounded in polynomial degree, which is hard to quantify. Even if we assume a polynomial approximation, we would need large degrees and results would depend on the decay rate of the t_j employed to gain a realistically tight bounds. Empirically, we do observe an exponential increase of precision with oversampling, as shown experimentally in Section 6. As the run-time costs of the FFT are moderate, this still a practical algorithm.

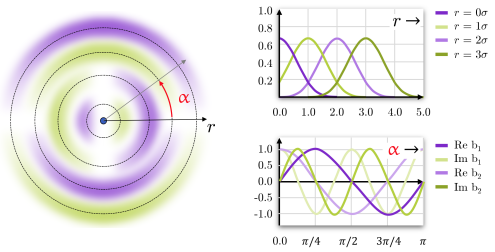

Figure 1: $SE(2)$-*equivariant point cloud networks:* Each point is associated with a set of concentric $SO(2)$-steerable filters (Fourier basis), modulated by equidistant gaussians in radial direction (for translational band-limiting).

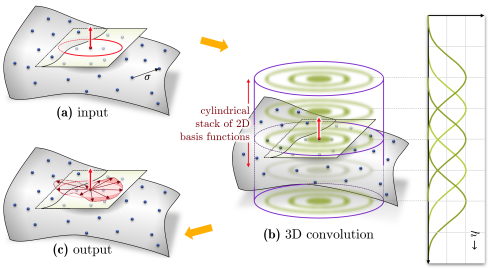

Figure 2: $SE(3)$-*equivariant surfel networks*: (a) for oriented surfels, we (b) perform 3D convolutions with unknown rotation around the normal direction, outputing (c) a collection of functions that assign scalar to tangent vectors.

5 Application to Geometric Data

In the following, we apply the $SO(2)$-equivariant network layer to 2D and 3D data.

5.1 SE(2)-Equivariant Networks for 2D Point Clouds

We construct an SE(2)-equivariant network for a set of input points $\mathbf{p}_1, ..., \mathbf{p}_n$ that carry arbitrary attribute vectors $\mathbf{y}_i$. These points can be a regular pixel grid of an image or a sampling of a geometric object. We equip each point $\mathbf{p}_i$ with a set of radial filters, in polar coordinates (r, φ) such that a 2D filter corresponds to a set of concentric circles of radius $r > 0$, on each of which a Fourier basis in φ is used to linearly represent $SO(2)$ (Fig. 1). Radially, we discretize by simple equidistant FIR low-pass filters — our current implementation uses equidistant Gaussians $\omega_m(r) := \exp(-(\frac{r}{2\sigma} - m)^2)$:

$$b^{(l)}_{k,m}(\varphi, r) := \omega_m(r) \cdot e^{2ik\varphi}, \;\; m = 0, 1, ..., M^{(l)}, \; k = -r^{(l)} \cdot m, ..., r^{(l)} \cdot m \tag{11}$$

The whole construction is illustrated in Figure 1. The input points for each layer are treated as Dirac functions $x^{(0)}(\mathbf{t}) = \sum_{i=1}^{n} \mathbf{y}_i \delta_{\mathbf{p}_i}(\mathbf{t})$.

Further layer types: To build complete networks, we employ spatial average pooling layers, which averages Fourier coefficients (which is possible due to linearity of the Fourier transform). For the projection to invariant features, we either output only the scalar Fourier coefficient z_0 in the last convolutional layer (*conv2triv*) or take the norm of all complex outputs of the last convolutional layer after the nonlinearity has been applied.
Batch-normalization [16] of Fourier-representations follows the obvious route of obtaining the mean via the z_0-coefficients and the variances via the power spectrum $||\mathrm{z}||_2^2$ of the Fourier coefficients. Instead of tracking a running mean during training for batch normalization, we calculate the exact training set statistics after training is done in one extra pass, while not changing other network weights.

Non-linearities: We evaluate our network for various general nonlinearities (ReLU, $\tanh$ and ELU [4]), as well as polynomial approximation of the ReLU function of degrees 2 and 4 (see Figure 4) taken from Gottemukkula [12], computed by FFT algorithm outlined in section 4.1. In case of the polynomial activations, we clamp the L1-norm of each channel's Fourier coefficients to the range $[-5, 5]$ before applying the nonlinearity to avoid problems with exploding activations or gradients. We also include the $\mathbb{C}$-ReLU function in our experiments (which acts on the norm of the activations only and therefore requires no Fourier transformation) to better estimate the performance penalties of the FFT method.

Architecture: Our concrete network design follows construction of Weiler and Cesa [31] for their best MNIST-rot-models, we use the same number of equivariant and linear layers with the same output channel count and filter properties (radii, rotation orders and width) and also apply Dropout [26] with $p = 0.3$ before each linear layer. We train our models for 40 epochs with at batch size of 64 images. We use the Adam [17] optimizer, starting with a learning rate of 0.015, with an exponential decay

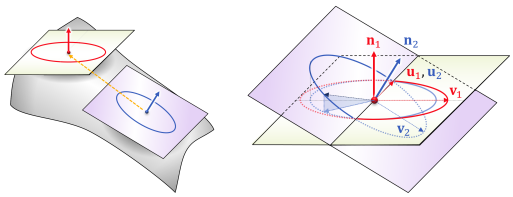

Figure 3: The output feature functions, which are tangent vectors, are aligned by projecting into the tangent-space of the target point.

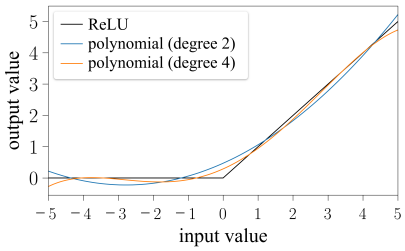

Figure 4: Polynomial approximations of the ReLU function [12].

Table 1: Results on the MNIST-rot dataset

model	group	repre-sentation	num. coeff.	FFT pad	activation function	invariant map	model param.	sec / epoch	test error (%) mean	test error (%) std
E2CNN [31]	C_{16}	regular	16	-	ELU	*maxpool*	2,692,690	38	0.716	0.028
E2CNN [31]	C_{16}	quotient	16	-	ELU	*maxpool*	2,749,686	49	0.705	0.045
E2CNN [31]	$D_{16\|5}C_{16}$	regular	16	-	ELU	*maxpool*	3,069,353	76	0.682	0.022
Ours	$SO(2)$	Fourier	9	-	$\mathbb{C}$-ReLU	*norm*	1,396,138	30	0.980	0.031
Ours	$SO(2)$	Fourier	9	127	ELU	*norm*	1,394,986	36	0.729	0.029
Ours	$SO(2)$	Fourier	9	127	tanh	*norm*	1,394,986	36	0.768	0.024
Ours	$SO(2)$	Fourier	9	127	ReLU	*norm*	1,394,986	36	0.685	0.026
Ours	$SO(2)$	Fourier	9	7	ReLU	*norm*	1,394,986	30	0.689	0.019
Ours	$SO(2)$	Fourier	17	127	ReLU	*norm*	2,729,098	64	0.699	0.033
Ours	$SO(2)$	Fourier	9	127	ReLU	*conv2triv*	891,178	36	0.719	0.018
Ours	$SO(2)$	Fourier	9	8	Poly(2)	*norm*	1,394,986	32	0.690	0.015
Ours	$SO(2)$	Fourier	9	24	Poly(4)	*norm*	1,394,986	36	0.690	0.024

factor of 0.8 per epoch starting after epoch 16. We calculate the mean test error and its standard deviation from 10 independent training runs with He-initialized [14] weights.

5.2 $SE(3)$-Equivariant Surfel Networks

Following the concept of Wiersma et al. [34], we can apply $SO(2)$-equivariant layers for building $SE(3)$-equivariant surface networks (our construction differs slightly: it uses extrinsic rather than intrinsic operations, thus actually creating exact $SE(3)$-equivariance but not having isometric ("bending") invariance). The main idea is that we only consider point-sampled surfaces with oriented normals at every point ("surfels", [21]). We then perform the same construction as in the 2D point cloud case, equipping each individual point with vertical stack of radial filters in its tangent plane (see Fig. 2a,b). In vertical direction, we use the same Gaussian filters as in radial direction (for consistent band-limiting). The filters are convolved against the point clouds, with uniform features $\mathbf{y}_i = 1$ for all i. The resulting angular functions can be interpreted as a vector-valued function in the tangent plane (Fig. 2c).

When performing a second level of convolution on already complex-valued input coefficients, we need to relate these angular feature functions computed by different surfels, which live in different reference frames. Here, we project the tangent vectors into the tangent plane, see Fig. 3. This construction is purely geometric and thus covariant under rigid transformations (the whole geometry is just rotated/translated together). Note: Aiming at different applications, Wiersma et al. [34] use intrinsic parallel transport here, which is more invariant but requires manifold input.

6 Results

We have implemented the $SO(2)$-equivariant layers and the corresponding $SE(2)$- and $SE(3)$-equivariant point cloud / surfel networks in PyTorch using PyKeOps [3] for computing the sparse matrix-vector multiplications of the general point clouds efficiently. The source code is provided as supplementary material.

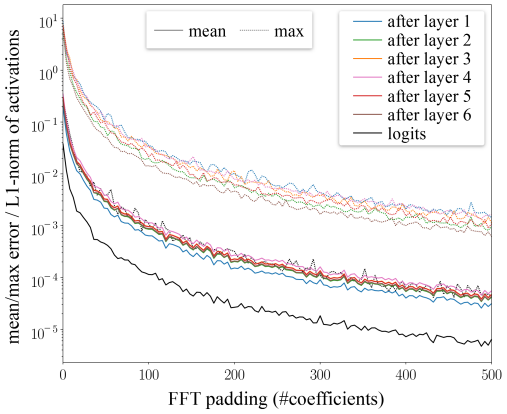

Figure 5: Relative error of ReLU activations (basic MNIST-rot network, 9 Fourier coeffs., *norm*-map) for random rotations vs. unrotated input. solid: mean absolute error, dashed: maximum error, relative to the layer-wise L1-norm for batches of 32 images.

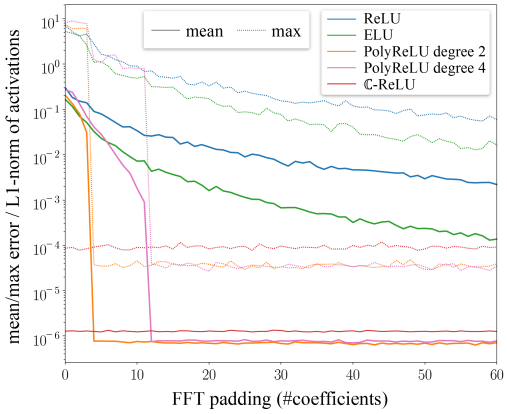

Figure 6: Same errors as in Fig. 5 after the fifth (penultimate) equivariant layer for various nonlinearities. Polynomials show the expected sharp decline at with increasing FFT padding. $\mathbb{C}$-ReLU included as reference, (no FFT used). Note the different scale on the axes.

Table 2: Results on the ModelNet-40 dataset

method	training time	test accuracy (%)		
		N/$SO(3)$	z/$SO(3)$	$SO(3)$/$SO(3)$
Spherical CNNs [8]	-	-	-	81.3
SPHnet [22]	1.5 hrs (RTX 2080 Ti)	86.6	-	87.6
CSGNN [10]	6.8 hrs (Tesla P100)	-	86.2	88.9
MA-KPConv [28]	7.5 hrs (Tesla V100)	89.1	89.1	89.1
GCANet [37]	-	-	89.1	89.2
Ours (Poly(2), FFT pad 8)	55 min (RTX 2080 Ti) (w/o preprocessing)	88.0	88.0	88.0
Ours (Poly(4), FFT pad 24)	60 min (RTX 2080 Ti) (w/o preprocessing)	88.7	88.7	88.7
Ours (ReLU, FFT pad 127)	60 min (RTX 2080 Ti) (w/o preprocessing)	89.1	89.1	89.1

Data sets: We test our implementation on image and 3D data. In the image case, we replicate the architecture used by Weiler and Cesa [31] on the MNIST-rot dataset from their recent survey of $E(2)$-equivariant image processing networks. For simplicity, we convert all image data into point clouds. This comes with some preprocessing overhead, but the absolute training times are still comparable to the pixel-based implementation used in [31], see Table 1. The point-based representation can be rotated exactly, facilitating the measurement of accuracy in terms of equivariance (images require supersampling to the remove aliasing of the pixel grid after rotations).

For the 3D surfel case, we use ModelNet-40 [36] as benchmark. We rescale all models to a unit bounding cube and convert the polygonal data into point clouds by z-Buffer rasterization from 50 random view points. Normals are estimated from a PCA-fit to 20 nearest neighbors at a sample spacing of 0.005, and oriented to point away from the center of mass. The final input is obtained by a reduction by Poisson-disc sampling with a sample spacing of 0.05.

Accuracy of Equivariance Figure 5 shows the error for the FFT-sampled ReLU for various amount of oversampling (padding applied to the Fourier basis) for all layers of the MNIST-rot-network. The different equivariant layers show a similar relative error, with the error on the final invariant (logits) layer being lower. All experiments use 32-bit floating-point GPU computations.

We compare the error approximations of different nonlinearities in Figure 6. The error for the polynomial nonlinearities drops sharply when a specific amount of oversampling is applied, which is in accordance with our theoretical considerations from Section 4.2. From this point on, further oversampling does not improve equivariance, which suggests that the remaining small fluctuations are due to the numerical limitations of the 32-bit floating point representation. This is supported by the observation that $\mathbb{C}$-ReLU, which should be perfectly equivariant as it only operates on the norm of the Fourier coefficients, produces a similar level of fluctuations.

The error for the approximations of ReLU and ELU drop continuously with increasing oversampling, with ELU dropping significantly faster that ReLU. The convergence behavior is good enough to reach reasonable accuracies (mean output errors of 10^{-5}) with feasible oversampling (up to 512 coefficients). For high accuracy requirements, the polynomial approach appears to be favorable (with an order of magnitude higher accuracy, at the limits of 32-bit floating point).

Prediction Accuracy: Table 1 lists our results on the MNIST-rot dataset, together with those obtained by Weiler and Cesa [31], which we managed to reproduce using their published code. Our models using ReLU or its polynomial approximations reach comparable performance to the best rotation-only (C_{16}) equivariant models evaluated this paper, while having a lower overall parameter count. Interestingly, the amount of oversampling (if we compare the ReLU-models with padding 7 and 127) does not seem to have a large impact on the accuracy score, while models using the ELU or $\tanh$ nonlinearities produce slightly worse results.

The results on ModelNet-40 are compared to the literature in Table 2. As common in the literature, we refer to a model trained on the original (non-augmented) dataset and evaluated with random rotations as N/$SO(3)$, while we denote rotational augmentation during training and testing as $SO(3)$/$SO(3)$. We also include a benchmarks for z/$SO(3)$, denoting random rotations around the z-axis during training and full rotations during testing, which is used in some papers. We performed the evaluation for all 3 regimes to allow a fair comparison. The resulting accuracy of our surfel based model compares favorable with various other architecture. When using the ReLU nonlinearity, results are on par with other state-of-the-art architecture on this task, while polynominal activation functions perform slightly worse.

Computational Cost In Table 1, we list training time in seconds per epoch obtained on a single *Nvidia RTX 2080Ti* graphics card. We can estimate the compuational cost of the FFT by comparing the runtimes with those of the $\mathbb{C}$-ReLU, which can be quickly computed and does not require performing an FFT. This suggests the the overhead of computing the FFT is low (smaller than 20%, depending on the amount of oversampling) compared to the other operations performed. Note that the times per epoch given in Table 1 also reflect total training time, as all models (E2CNN [31] and Ours) are trained for 40 epochs.

7 Conclusions & Future Work

In this paper, we have presented an analysis of the effect of nonlinear activation functions on the Fourier representations used by $SO(2)$-equivariant networks. The main insight is that the nonlinearity creates high frequency harmonics; thus applying the nonlinearity to an oversampled angular domain representation can maintain equivariance. For polynomial nonlinearities, this construction is provably exact. This theoretical prediction is also observed in real numerical implementation. In the general case, we empirically observe rapid convergence, which is also plausible from an analytical perspective. As a sanity check, we have applied our method to shape and image classification, and reach performance on par with state-of-the-art equivariant architectures while providing full, continuous equivariance.

Our main result, the oversampling algorithm for applying nonlinearities closes a small, but important gap in the literature on equivariant networks for iamge and geometric data processing: It removes most architectural restictions, making the design of such networks significantly easier. The algorithm employed is easy to understand and implement; the only (but, as our experiments show, crucial) departure from base-line angular-domain evaluation is a resolution increase.

The method provides continuous equivariance up to numerical precision. In crictical applications, such as physical simulations, polynomial non-linearities can optionally provide an a priori guarantee; however, with empirical calibration of oversampling factor D, general non-linearities can be used. In both cases, when integrated into a point cloud network, the performance penalty is very minor in relation to the overall costs.

The biggest conceptual limitation of this approach is probably the restriction to $SO(2)$. While similar construction could probably be applied to more complex compact Lie groups like $SO(3)$ (without using auxiliary orientation information), performing harmonic distortion analysis in non-commutative symmetry groups becomes significantly more involved (requiring for example a matrix-valued convolution theorem).

References

[1] C. H. Ali Mehmeti-Göpel, D. Hartmann, and M. Wand. Ringing ReLUs: Harmonic distortion analysis of nonlinear feedforward networks. In *International Conference on Learning Representations (ICLR).*, 2021. URL `https://openreview.net/forum?id=TaYhv-q1Xit`.

[2] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *ACM Transactions on Graphics (TOG)*, 37:1 – 12, 2018.

[3] B. Charlier, J. Feydy, J. A. Glaunès, F.-D. Collin, and G. Durif. Kernel operations on the gpu, with autodiff, without memory overflows. *Journal of Machine Learning Research*, 22(74):1–6, 2021. URL `http://jmlr.org/papers/v22/20-275.html`.

[4] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[5] T. S. Cohen and M. Welling. Group equivariant convolutional networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[6] T. S. Cohen and M. Welling. Steerable cnns. In *International Conference on Learning Representations (ICLR)*, 2017.

[7] S. Dieleman, J. D. Fauw, and K. Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks, 2016.

[8] C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis. 3d object classification and retrieval with spherical cnns. *ArXiv*, abs/1711.06721, 2017.

[9] R. P. Feynman, R. B. Leighton, and M. Sands. The feynman lectures on physics; vol. i. *American Journal of Physics*, 50(8), 1965.

[10] J. Fox, B. Zhao, S. Rajamanickam, R. Ramprasad, and L. Song. Concentric spherical GNN for 3d representation learning. *CoRR*, abs/2103.10484, 2021. URL `https://arxiv.org/abs/2103.10484`.

[11] A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers, 1995.

[12] V. Gottemukkula. Polynomial activation functions. In *International Conference on Learning Representations (retracted paper)*, 2020. URL `https://openreview.net/forum?id=rkxsgkHKvH`.

[13] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38:1 – 12, 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1026–1034, USA, 2015. IEEE Computer Society.

[15] E. Hoogeboom, J. W. T. Peters, T. S. Cohen, and M. Welling. Hexaconv, 2018.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[18] R. Kondor, T. S. Hy, H. Pan, B. M. Anderson, and S. Trivedi. Covariant compositional networks for learning graphs. In *ICLR 2018 Workshop Track*, 2018. URL `https://openreview.net/forum?id=S1TgE7WR-`.

[19] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks, 2016.

[20] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on X-transformed points. *arXiv: Computer Vision and Pattern Recognition*, 2018.

[21] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *ACM SIGGRAPH 2000 Conference Proceedings, Annual Conference Series*, 2000.

[22] A. Poulenard, M.-J. Rakotosaona, Y. Ponty, and M. Ovsjanikov. Effective rotation-invariant point cnn with spherical harmonics kernels. *2019 International Conference on 3D Vision (3DV)*, pages 47–56, 2019.

[23] C. Qi, H. Su, K. Mo, and L. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[24] C. Qi, L. Yi, H. Su, and L. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

[25] K. Schütt, P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K. Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *NIPS*, 2017.

[26] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[27] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 945–953, 2015.

[28] H. Thomas. Rotation-invariant point convolution with multiple equivariant alignments. *2020 International Conference on 3D Vision (3DV)*, pages 504–513, 2020.

[29] H. Thomas, C. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. Guibas. Kpconv: Flexible and deformable convolution for point clouds. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6410–6419, 2019.

[30] N. Thomas, T. Smidt, S. M. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219, 2018. URL `http://arxiv.org/abs/1802.08219`.

[31] M. Weiler and G. Cesa. General E(2)-equivariant steerable cnns. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 14334–14345. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/45d6637b718d0f24a237069fe41b0db4-Paper.pdf`.

[32] M. Weiler, W. Boomsma, M. Geiger, M. Welling, and T. S. Cohen. 3d steerable cnns, learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[33] M. Weiler, F. A. Hamprecht, and M. Storath. Learning steerable filters for rotation equivariant cnns, 2018.

[34] R. Wiersma, E. Eisemann, and K. Hildebrandt. CNNs on surfaces using rotation-equivariant features. *Transactions on Graphics*, 39(4), July 2020. doi: 10.1145/3386569.3392437.

[35] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance, 2017.

[36] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. doi: 10.1109/CVPR.2015.7298801.

[37] Z. Zhang, B.-S. Hua, W. Chen, Y. Tian, and S. Yeung. Global context aware convolutions for 3d point cloud understanding. *2020 International Conference on 3D Vision (3DV)*, pages 210–219, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] The paper discusses the effect of nonlinearities in equivariant deep networks with steerable filters. The paper provides additional insights on how such networks can be improved (more flexibility in the architecture / more accurate equivariant behavior). As a very generic low-level insight, we cannot foresee any specific social impacts other than those associated with general progress in machine learning with deep networks.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] In the theoretical analysis, we operate under the assumption of all functions being band-limited. This removes most of the technical difficulties of Fourier analysis (as there are no issues concerning convergence).

(b) Did you include complete proofs of all theoretical results? [No] The main theoretical result is that our algorithm is exact for polynomial non-linearities. The paper show this in a formally conclusive way (see also comments above). For the general case of non-polynomial non-linearities, we only provide empirical evidence, as stated clearly in the paper.

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] The SE(2)-equivariant image classification test follows the protocol of [31]. The ModelNet-40 case discusses important aspects explicitly. We provide a more extensive discussion in the supplementary material.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We do provide standard deviations over multiple runs for the accuracy comparisons in Table 1, where the numbers are very close and statistical errors could easily lead to wrong conclusions. For the accuracy plots, we already have an average over a large network, and the effect is non-stochastic; therefore, we do not perform multiple runs.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We only use desktop systems; the components are specified.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We use standard benchmarks (MNIST-rot, ModelNet-40)

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]