General Nonlinearities in SO(2)-Equivariant CNNs – Supplementary Material –

Daniel Franzen
Institute of Computer Science
Johannes Gutenberg University Mainz
Staudingerweg 9,
55122 Mainz, Germany
dfranz@uni-mainz.de

Michael Wand
Institute of Computer Science
Johannes Gutenberg University Mainz
Staudingerweg 9,
55122 Mainz, Germany
wandm@uni-mainz.de

In this supplementary file, we give detailed information on our architectures and training procedures, show additional results, and provide extended proofs and derivations that we could not fit in the limited space of our main paper.

Contents

A Architectures and training details **2**

A.1 SE(2)-Equivariant Networks for 2D Point Clouds 2

A.1.1 Data and preprocessing 2

A.1.2 Architecture 2

A.1.3 Training and evaluation protocol 2

A.1.4 Experiments 2

A.2 SE(3)-Equivariant Surfel Networks 6

A.2.1 Local reference frames and feature interaction 6

A.2.2 Data and preprocessing 6

A.2.3 Architecture 7

A.2.4 Training and evaluation protocol 7

A.2.5 Experiments 8

B Proofs and derivations **9**

B.1 Equivariance of the joint convolution operation 9

B.2 Convolution in Fourier basis 9

B.3 Calculations for point clouds 10

B.4 Learnable filters and multiple feature channels 11

B.5 Practical considerations 11

C NeurIPS 2021 Paper Checklist **12**

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

A Architectures and training details

In this section, we explain in detail how we performed our experiments for our SE(2)- and SE(3)-equivariant models, including the datasets, preprocessing steps, architectural designs as the training and evaluation protocols we used.

A.1 SE(2)-Equivariant Networks for 2D Point Clouds

A.1.1 Data and preprocessing

For simplicity, our 2D implementation operates directly on point clouds (this is not a fundamental limitation; as the point cloud code is more general, we can use it to handle pixel grids, too). As the MNIST-rot dataset consists of images, we first convert them to point clouds by placing the values on a regular grid with a spacing of 1, i.e. we generate a *coordinate* tensor containing the point coordinates and a *data* tensor containing the pixel values for each pixel in the image. The *data* tensor of the input is real-valued with rotation order 0, but in general, its values in the equivariant part of the network are complex Fourier coefficients of various rotation order, representing the network activations as band-limited angular functions.

A.1.2 Architecture

Architecture-wise, we stay as close as possible to the best performing rotation-equivariant MNIST-rot architecture by Weiler and Cesa [5], i.e. we use the same number, type and width of layers, as well as identical filter parameters for the equivariant convolutional layers. A detailed view of our architecture is shown in Table 1.

Our network consists of 6 rotation-equivariant convolutional layers, after which an *invariant map* is applied, followed by 3 rotation-invariant linear (fully-connected) layers. After each layer, we perform Batch Normalization [3], followed by a nonlinearity, with exception of the final linear layer. When working on complex Fourier coefficients, we apply a custom Batch Normalization layer which operates directly on the coefficients (as discussed in the main paper), otherwise, we use the standard implementation from *PyTorch*. Furthermore, following the design of Weiler and Cesa [5], we also apply Dropout [4] with $p = 0.3$ to the inputs of each linear layer.

Contrary to Weiler and Cesa [5], who use spatial max-pooling in their best performing architectures, we opt for $2 * 2$ spatial average pooling instead, which has the advantage that it can be applied directly on the Fourier coefficients without requiring any transformation. After each pooling step, we divide the *coordinate* tensor by 2, so the spacing between grid points is kept at 1 at all times.

A.1.3 Training and evaluation protocol

We train out networks on the training set of MNIST-rot, containing 12,000 rotated images from the original MNIST dataset, and evaluate them on the corresponding test set, containing 50,000 images, using the same data loading pipeline and hyperparameters as Weiler and Cesa [5]. Network weights are initialized randomly using He initialization [2], and then trained using the Adam optimizer with *PyTorch* default parameters (betas $= (0.9, 0.999)$, eps $= 10^{-8}$) and a mini-batch size of 64 to optimize the log-softmax cross entropy of the network output. Training is done for a total of 40 epochs, using a fixed learning rate of 0.015 in the first 16 epochs, and then decayed exponentially by applying a factor of 0.8 before each following epoch. To get meaningful results, we calculate the mean test error and its standard deviation from 10 independent training runs.

Between training and testing, we perform an additional pass on the training set without changing trainable weights to calculate the exact statistics on the full training set in the Batch Normalization layers. This gives more representative values than using an expontial average for determining the mean and variance values during the optimization process (which can lag behind when the parameters of the network change).

A.1.4 Experiments

We train different variations of our model on MNIST-rot (Table 2) and measure accuracy and training time per epoch with varying precision of the angular representation by using a different number of

Table 1: Detailed architecture for experiments on the MNIST-rot dataset

no.	layer type	spatial reduction	output size $x * y *$ channels $*$ coeffs	filters (r, σ, frequencies)	followed by
0	Input	-	$28 * 28 * 1 * 1$	-	–
1	Conv2D	crop(4)	$20 * 20 * 24 * c_{\text{max}}$	$(0, 0.005, 0)$ $(1, 0.600, -2\ldots2)$ $(2, 0.600, -3\ldots3)$ $(3, 0.600, -6\ldots6)$ $(4, 0.400, -2\ldots2)$	BN + nonlinearity
2	Conv2D	pool(2)	$10 * 10 * 32 * c_{\text{max}}$	$(0, 0.005, 0)$ $(1, 0.600, -2\ldots2)$ $(2, 0.600, -3\ldots3)$ $(3, 0.400, -2\ldots2)$	BN + nonlinearity
3	Conv2D	–	$10 * 10 * 36 * c_{\text{max}}$	$(0, 0.005, 0)$ $(1, 0.600, -2\ldots2)$ $(2, 0.600, -3\ldots3)$ $(3, 0.400, -2\ldots2)$	BN + nonlinearity
4	Conv2D	pool(2)	$5 * 5 * 36 * c_{\text{max}}$	$(0, 0.005, 0)$ $(1, 0.600, -2\ldots2)$ $(2, 0.600, -3\ldots3)$ $(3, 0.400, -2\ldots2)$	BN + nonlinearity
5	Conv2D	–	$5 * 5 * 64 * c_{\text{max}}$	$(0, 0.005, 0)$ $(1, 0.600, -2\ldots2)$ $(2, 0.600, -3\ldots3)$ $(3, 0.400, -2\ldots2)$	BN + nonlinearity
6	Conv2D	crop(2)	$1 * 1 * 96 * c_{\text{final}}$	$(0, 0.005, 0)$ $(1, 0.600, -2\ldots2)$ $(2, 0.400, -2\ldots2)$	BN + nonlinearity
7	Linear	–	96	–	BN + nonlinearity
8	Linear	–	96	–	BN + nonlinearity
9	Linear	–	40	–	CrossEntropyLoss

Table 2: Variations of our MNIST-rot model using ReLU and its polynomial approximations

model	group	repre-sentation	num. coeff.	FFT pad	activation function	invariant map	model param.	sec / epoch	test error (%) mean	std
Ours	SO(2)	Fourier	3	127	ReLU	*norm*	365,338	21	0.985	0.035
Ours	SO(2)	Fourier	5	127	ReLU	*norm*	708,634	25	0.768	0.021
Ours	SO(2)	Fourier	9	0	ReLU	*norm*	1,394,986	30	0.710	0.025
Ours	SO(2)	Fourier	9	7	ReLU	*norm*	1,394,986	30	0.689	0.019
Ours	SO(2)	Fourier	9	8	Poly(2)–ReLU	*norm*	1,394,986	32	0.690	0.015
Ours	SO(2)	Fourier	9	24	Poly(4)–ReLU	*norm*	1,394,986	36	0.690	0.024
Ours	SO(2)	Fourier	9	127	ReLU	*norm*	1,394,986	36	0.685	0.026
Ours	SO(2)	Fourier	9	127	ReLU	*conv2triv*	891,178	36	0.719	0.018
Ours	SO(2)	Fourier	17	8	Poly(2)–ReLU	*norm*	2,729,098	61	0.691	0.022
Ours	SO(2)	Fourier	17	24	Poly(4)–ReLU	*norm*	2,729,098	64	0.694	0.025
Ours	SO(2)	Fourier	17	127	ReLU	*norm*	2,729,098	64	0.699	0.033

Table 3: Comparison of nonlinearities on MNIST-rot

model	group	repre-sentation	num. coeff.	FFT pad	activation function	invariant map	model param.	sec / epoch	test error (%) mean	std
Ours	SO(2)	Fourier	9	–	$\mathbb{C}$–ReLU	*norm*	1,396,138	30	0.980	0.031
Ours	SO(2)	Fourier	9	–	$\mathbb{C}$–$\mathrm{sigmoid}$	*norm*	1,396,138	30	1.500	0.034
Ours	SO(2)	Fourier	9	127	ReLU	*norm*	1,394,986	36	0.685	0.026
Ours	SO(2)	Fourier	9	127	$\mathrm{LeakyReLU}$	*norm*	1,394,986	36	0.690	0.028
Ours	SO(2)	Fourier	9	127	SiLU	*norm*	1,394,986	36	0.705	0.026
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	$\mathrm{sigmoid}$	*norm*	1,394,986	36	0.809	0.022

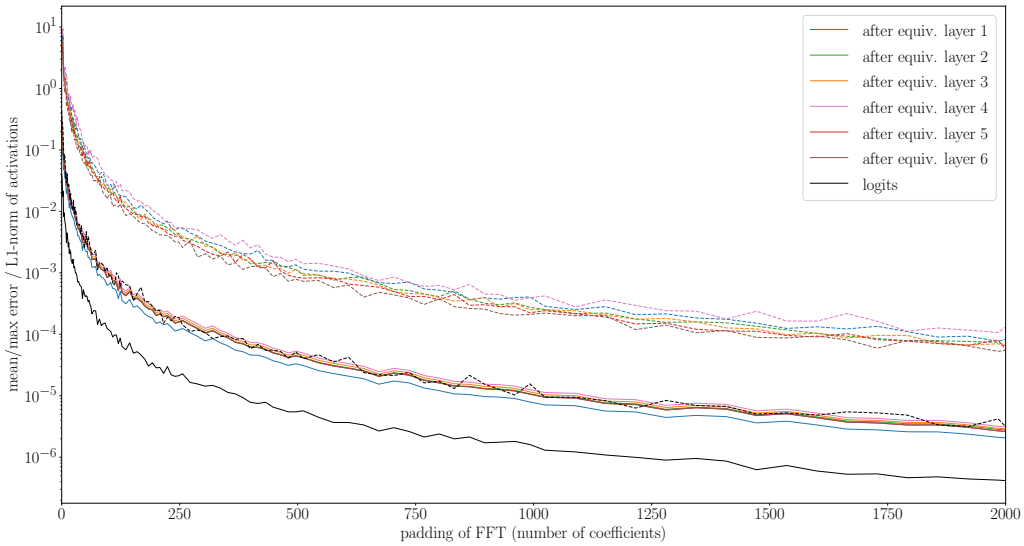

Figure 1: Relative error of ReLU activations for our 2D architecture measured on MNIST-rot for randomly rotated vs. unrotated inputs (architecture as in Table 1, 9 Fourier coefficients, *norm*-map). solid: mean absolute error, dashed: maximum error, relative to the layer-wise L1-norm measured for batches of 32 images, with 36 random rotations applied to each image

Fourier coefficients (3, 5, 9, or 17), as well as precision of the nonlinearity application by using different padding sizes for the FFT algorithm.

Invariant map: Here, we consider two possible choices. One method to produce invariant activations, *conv2triv*, is to output only the rotation invariant coefficient z_0 in the last convolutional layer (i.e. to set $c_{\text{final}} = 1$ in Table 2). This coefficient is the average value of the angular function, therefore this method is equivalent to average pooling over different rotations of the output. However, as strong activations may appear only for some specific rotation of the input, this method may not be optimal, as can be also seen from the results. As an alternative, we set the last convolutional layer to full sized output ($c_{\text{final}} = c_{\text{max}}$) and take the *norm* of each complex coefficient after the nonlinearity has been applied.

Nonlinearities: Having found a well-performing architecture, we use it to investigate different nonlinearities (Table 3). We evaluate the ReLU, $\mathrm{LeakyReLU}$, SiLU, ELU, $\tanh$ and $\mathrm{sigmoid}$ nonlinearities with the FFT algorithm using a padding of 127 to guarantee a good precision, with ReLU and $\mathrm{LeakyReLU}$ resulting in the best accuracy. We also include the polynomial approximations of ReLU [1], where we set the FFT padding depending on the degree of the polynomial. These approximations were designed for an input range of $[-5; 5]$. As polynomial approximations diverge quickly outside of the intended range of the approximation, we clamp the ℓ_1-norm of each channel's Fourier coefficients (which is an upper limit to the maximum absolute value of the angular function) to a maximum value of 5 before applying the nonlinearity, to make sure we avoid problems with exploding activations or gradients.

An alternative approach is to use norm-only nonlinearities, which do not require a Fourier transform as they act exclusively on the norm of the complex coefficients [7; 5]. This is only suitable for functions with strictly positive output (like ReLU and $\mathrm{sigmoid}$). In accordance with Weiler and Cesa [5], we find that norm-based nonlinearities lead to a worse accuracy than classic ReLU on MNIST-rot.

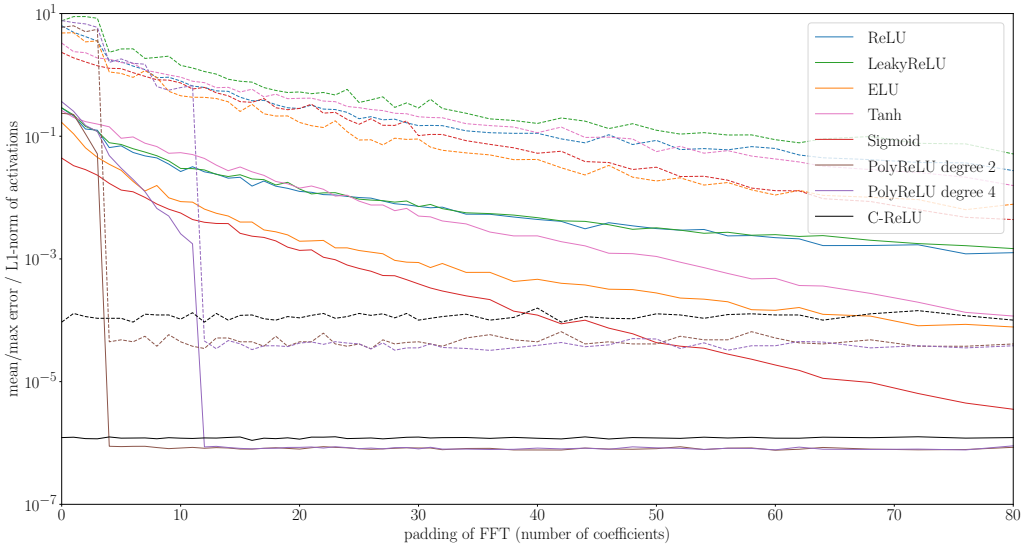

Figure 2: Relative error for various nonlinearities for our 2D architecture measured on MNIST-rot for randomly rotated vs. unrotated inputs. Errors are measured as in Figure 1 after the fifth (penultimate) equivariant layer. Polynomials show the expected sharp decline with increasing FFT padding. $\mathbb{C}$–ReLU included as reference (no FFT used).

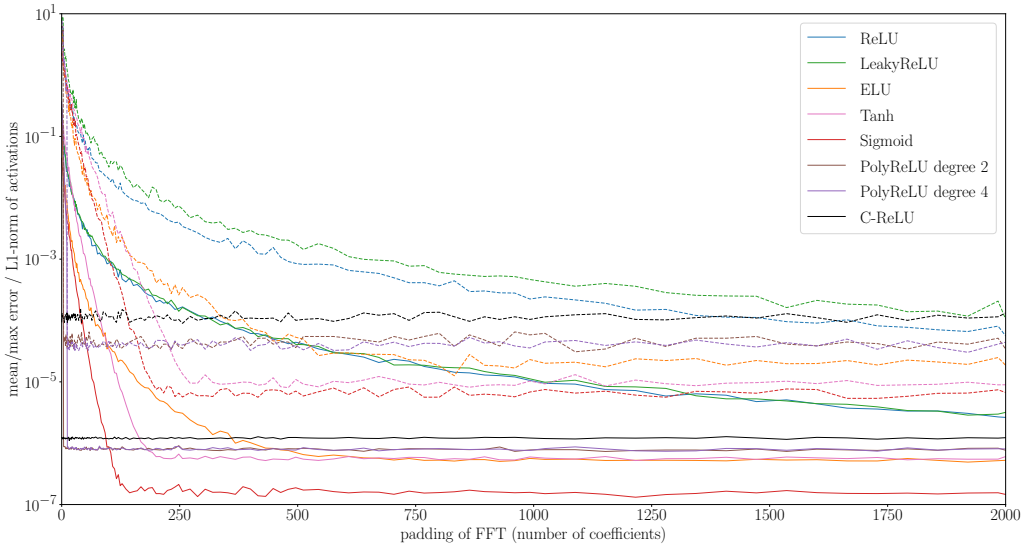

Figure 3: Same plot as Figure 2, but extended to very large FTT paddings. ReLU and LeakyReLU converge slowest and still show improvements up to very large padding values of 2000.

Table 4: Detailed architecture for experiments on the ModelNet-40 dataset

no.	layer type	layer output			layer filters			followed by
		channels	coeffs	sampling level	radius	levels	σ	
0	Input	1	1	1	–	–	–	–
1	SurfelConv	16	9	2	0.1	$(-0.1, 0.1, 0.1)$	0.0424	BN + nonlinearity
2	SurfelConv	32	9	3	0.2	$(-0.2, 0.2, 0.2)$	0.0849	BN + nonlinearity
3	SurfelConv	48	9	3	0.4	$(-0.4, 0.4, 0.4)$	0.1699	BN + nonlinearity
4	SurfelConv	64	9	3	0.4	$(-0.4, 0.4, 0.4)$	0.1699	BN + nonlinearity
5	SurfelConv	96	9	3	0.8	$(-0.8, 0.8, 0.8)$	0.3397	BN + nonlinearity
6	SurfelConv	40	1	3	0.8	$(-0.8, 0.8, 0.8)$	0.3397	BN + nonlinearity
7	PointAvgPool	40	1	(1 value)	–	–	–	CrossEntropyLoss

A.2 SE(3)-Equivariant Surfel Networks

A.2.1 Local reference frames and feature interaction

Our SE(3) based surfel network is inspired by the work of Wiersma et al. [6] and transfers the concept of angular-dependent activations, which previously lived on a flat surface of an image, to the curved surface of a 3D object. This poses some difficulties, as the implicit assumption that all features are aligned in an identical way (which we made for the 2D case of flat images) does not hold for generally curved 3D surfaces, where the normal vectors can have varying directions. Therefore, this approach requires to assign a *local reference coordinate frame* ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{n}$) to each point feature, consisting of the normal vector $\mathbf{n}$, an arbitrarily chosen tangential vector $\mathbf{x} \perp \mathbf{n}$, and a second tangential vector $\mathbf{y} = \mathbf{n} \times \mathbf{x}$ (calculated by taking a cross product, to obtain a right-handed coordinate system). The angular features $x(\alpha)$ can then be mapped to directions $\mathbf{v}$ in the tangential $\mathbf{x}$,$\mathbf{y}$-plane:

$$\mathbf{v} = \mathbf{x} \cos \alpha - \mathbf{y} \sin \alpha \tag{1}$$

For two features to interact in an equivariant fashion, we need to take their local alignment into account. Wiersma et al. [6] solve this problem by using parallel transport along the surface of the object to align the two local coordinate systems in a reliable manner.

We chose to take a different approach. We first find a common tangent vector $\mathbf{t}$ which lies in the tangential $\mathbf{x}$,$\mathbf{y}$-plane of both (input and output) local coordinate systems and rotate all input Fourier features to this common coordinate system. We then multiply the imaginary part of all coefficients by the dot product of the normal vectors $\langle \mathbf{n}_1, \mathbf{n}_2 \rangle$. Finally, we rotate the coefficients to the coordinate system of the output. For coefficients of rotation order 1 (z_1), where activations can be represented as tangential vectors fixed to the object that point in the direction of maximal activation according to Eq. 1, this approach is identical to projecting these vectors to the tangential plane of the output point, as depicted in Figure 3 in the main paper.

A.2.2 Data and preprocessing

For the 3D surfel case, we use ModelNet-40 [8] as benchmark. We rescale all models to fit in a bounding cube (coordinate range $[-1, 1]$) while keeping the aspect ratio and convert the polygonal data to point clouds by z-Buffer rasterization with high resolution from 50 random view points. This method keeps only points on the surface of the model which are visible from the outside, which we found to slightly improve performance, as some models in ModelNet-40 contain a lot of internal elements. Normals are estimated from a PCA-fit to 20 nearest neighbors at a sample spacing of 0.005, with some small random dithering value added to avoid numerical instabilities in case when two normals have the exact same direction, and always oriented to point away from origin of the coordinate system.

The final input to our network is obtained by a by Poisson disc sampling with a sample spacing of 0.05 for sampling level 1. Subsequent sampling levels are created by iteratively, choosing a quarter of the points of the previous sampling level in each step. For this process, we use the sample elimination algorithm described by Yuksel [9]. Note that the final Poisson disc sampling process is done individually for each training epoch, generates different samplings for each epoch, which is important to avoid overfitting to a specific sampling in the training process. We use external tools for the preprocessing of the dataset, therefore, these processing steps are not included in our published code.

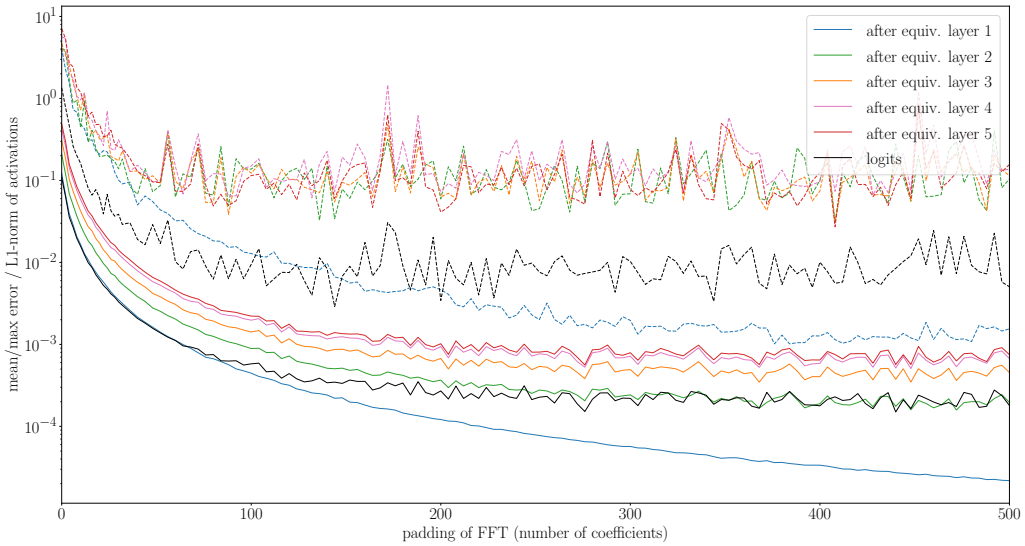

Figure 4: Relative error of ReLU activations for our 3D surfel architecture measured on ModelNet-40 for randomly rotated vs. unrotated inputs (architecture as in Table 4). Note the different scale on the axes compared to the MNIST-rot plots. solid: mean absolute error, dashed: maximum error, relative to the layer-wise L1-norm measured for batches of 32 images where a random SO(3) rotation is applied individually for each image

As our preprocessing only gives us a set of input points and corresponding normals, but not any initial feature vectors associated with these points, we use a single-channels input tensor with the rotation-invariant Fourier coefficient set to 1 for each point as input.

A.2.3 Architecture

Our detailed architecture is shown in Table 4. Experiments have shown that convolutional layers with a filter stack of 3 equidistant levels (Figure 2 in the main paper), where each level consists of a single filter ring with a radius equal to the spacing between adjacent levels, work well in practice. The FWHM (full width a half maximum) of the Gaussian filter profile is also chosen equal to to the level spacing, resulting in the σ values given in Table 4 ($\sigma = \mathrm{FWHM}/\sqrt{8 \ln 2}$). We use 9 coefficients to represent our angular dependent functions and we do not impose limits on filter frequencies, i.e. allow interactions between all coefficients, which results in filter frequencies of a rotation order of up to 8.

In total, our network consists of 6 equivariant convolutional layers. To generate rotation invariant output, we output only the invariant coefficients in the final convolutional layer (*conv2triv*) and apply average pooling over all remaining points. As in the MNIST-rot scenario, we use Batch Normalization (directly on Fourier coefficients, if applicable) and a nonlinearity after each convolutional layer except the last layer.

A.2.4 Training and evaluation protocol

We train out networks on the training set of ModelNet-40, containing 9,843 3D meshes of 40 different object classes. There is a significant imbalance of the number of objects per class between training and testing set, however, we chose not to apply any measures to counteract this, to allow a fair comparison of our results to those of other papers.

We initialize the network weights randomly before training using He initialization [2], and train using the Adam optimizer with *PyTorch* default parameters (betas $= (0.9, 0.999)$, eps $= 10^{-8}$) and a mini-batch size of 32 to optimize the log-softmax cross entropy of the network output. Training is done for a total of 30 epochs, using a fixed learning rate of 0.015 in the first 10 epochs, and then

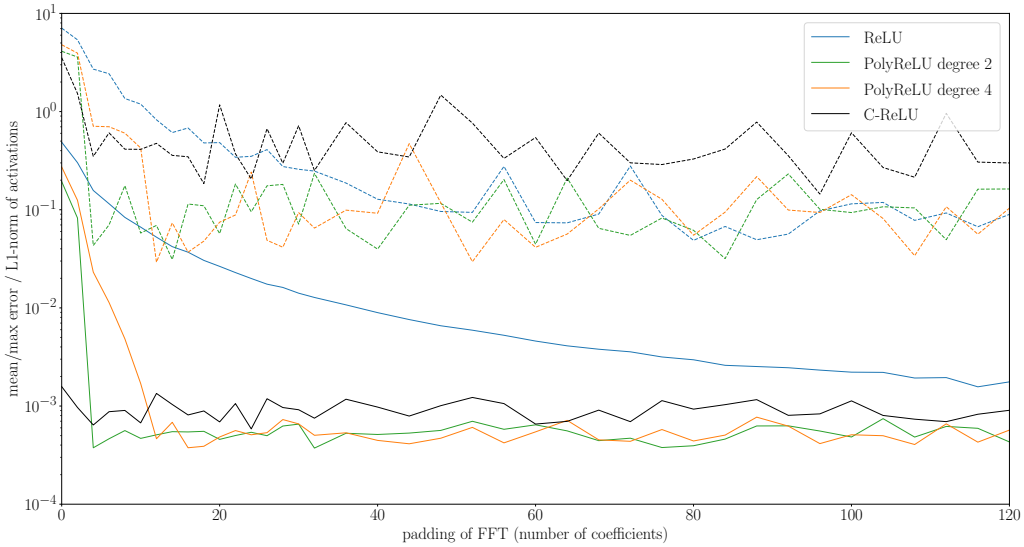

Figure 5: Relative error for various nonlinearities for our 3D surfel architecture measured on ModelNet-40 for randomly rotated vs. unrotated inputs. Errors are measured as in Figure 4 after the fifth (penultimate) convolutional layer. $\mathbb{C}$–ReLU included as reference (no FFT used).

decayed linearly to zero over the next 20 epochs. Batch statistics are calculated over the full training set, as outlined in Section A.1.3.

We test our architecture on the testing set of ModelNet-40, containing 2,468 3D models. As common in the literature, we perform runs with different rotational augmentation during training and testing,

- N/SO(3), referring to training on the original (non-augmented) dataset and testing with random $SO(3)$ rotations,
- z/SO(3), denoting random rotations around the z-axis during training and $SO(3)$ rotations during testing, and
- SO(3)/SO(3), using random SO(3) augmentation during training and testing.

A.2.5 Experiments

Our experiments and results for ModelNet-40, including runtimes (measured without taking the preprocessing steps into account), can be found in Table 2 in the main paper. Rotation error measurements for random rotations can be found in Figure 4 for ReLU and Figure 5 for other nonlinearities. The error is generally higher than for our 2D architecture, as can be seen from the layerwise plot in Figure 4. It also increases for deeper layers, probably due to cumulative effects. However, the network is still mostly invariant in the classification task, as can be seen from Table 2 in the main paper.

The higher error is most likely not caused by the FFT algorithm for applying nonlinearities, as $\mathbb{C}$–ReLU (which is equivariant by construction, and thus does not use FFT calculations at all) produces a similar error level (see Figure 5). Therefore, we assume this is a general drawback of the 3D surfel architecture used in this test, probably due to the much more complex calculations in comparison to the 2D case and thus higher amplification of noise. As the relative error is the same (polynomials) or easily approaches the same level (ReLU) as the fully-equivariant baseline ($\mathbb{C}$–ReLU), this does not directly weaken the result of our paper (that more general nonlinearities can be handled at error rates comparable to specialized solutions with provable equivariance without additional measures). However, it indicates that other sources of numerical error amplification might be a separate issue to pay attention to if strong invariances need to be maintained.

B Proofs and derivations

B.1 Equivariance of the joint convolution operation

In the main paper, we derive our equivariant neural network layer, which performes a joint convolution operation on $\mathbb{R}^2$ and SO(2). We now to prove that this layer is equivariant under the transformation T_γ:

$$T_\gamma \, x^{(l)}(\mathbf{t}, \alpha) = x^{(l)}(\Theta_\gamma \mathbf{t}, \alpha - \gamma) \tag{2}$$

In the following, we will show that the network layer commutes with transformation $T\gamma$. Therefore, we will perform series of conversions on our convolution operation:

$$\hat{x}^{(l)}\left(\mathbf{t}, \alpha\right) = \int_{\mathbb{R}^2} d\mathbf{t}' \int_0^{2\pi} d\alpha' \; w^{(l)}\left(\Theta_\alpha(\mathbf{t}' - \mathbf{t}), \alpha' - \alpha\right) x^{(l-1)}\left(\mathbf{t}', \alpha'\right) \tag{3}$$

We start by replacing the integration variables $\mathbf{t}' \rightarrow \Theta_\gamma \mathbf{t}'$ and $\alpha' \rightarrow \alpha' - \gamma$. This can be done without changing the limits of the integrals, as we integrate over infinite or periodic domains.

$$\hat{x}^{(l)}\left(\mathbf{t}, \alpha\right) = \int_{\mathbb{R}^2} d\mathbf{t}' \int_0^{2\pi} d\alpha' \; w^{(l)}\left(\Theta_\alpha(\Theta_\gamma \mathbf{t}' - \mathbf{t}), (\alpha' - \gamma) - \alpha\right) x^{(l-1)}\left(\Theta_\gamma \mathbf{t}', \alpha' - \gamma\right) \tag{4}$$

Reordering some elements yields:

$$\hat{x}^{(l)}\left(\mathbf{t}, \alpha\right) = \int_{\mathbb{R}^2} d\mathbf{t}' \int_0^{2\pi} d\alpha' \; w^{(l)}\left(\Theta_{\alpha+\gamma}(\mathbf{t}' - \Theta_{-\gamma}\mathbf{t}), \alpha' - (\alpha + \gamma)\right) x^{(l-1)}\left(\Theta_\gamma \mathbf{t}', \alpha' - \gamma\right) \tag{5}$$

We now perform a similar replacement for the output parameters $\mathbf{t} \rightarrow \Theta_\gamma \mathbf{t}$ and $\alpha \rightarrow \alpha - \gamma$.

$$\hat{x}^{(l)}\left(\Theta_\gamma \mathbf{t}, \alpha - \gamma\right) = \int_{\mathbb{R}^2} d\mathbf{t}' \int_0^{2\pi} d\alpha' \; w^{(l)}\left(\Theta_\alpha(\mathbf{t}' - \mathbf{t}), \alpha' - \alpha\right) x^{(l-1)}\left(\Theta_\gamma \mathbf{t}', \alpha' - \gamma\right) \tag{6}$$

Comparing with our equivariance transformation (Eq. 2), we can substitute with T_γ:

$$T_\gamma \, \hat{x}^{(l)}\left(\mathbf{t}, \alpha\right) = \int_{\mathbb{R}^2} d\mathbf{t}' \int_0^{2\pi} d\alpha' \; w^{(l)}\left(\Theta_\alpha(\mathbf{t}' - \mathbf{t}), \alpha' - \alpha\right) T_\gamma \, x^{(l-1)}\left(\mathbf{t}', \alpha'\right) \tag{7}$$

We can now see that applying T_γ to the input $x^{(l-1)}$ results in an identical transformation of the output $\hat{x}^{(l)}$, which satisfies the definition of equivariance under the transformation T_γ.

B.2 Convolution in Fourier basis

To derive our convolution operation in the Fourier representation, we start from the continuous convolution operation:

$$\hat{x}^{(l)}\left(\mathbf{t}, \alpha\right) = \int_{\mathbb{R}^2} d\mathbf{t}' \int_0^{2\pi} d\alpha' \; w^{(l)}\left(\Theta_\alpha(\mathbf{t}' - \mathbf{t}), \alpha' - \alpha\right) x^{(l-1)}\left(\mathbf{t}', \alpha'\right) \tag{8}$$

Plugging in the Fourier representations for x

$$x^{(l)}\left(\mathbf{t}, \alpha\right) = \sum_{k=-K}^{K} z_k^{(l)}(\mathbf{t}) \, e^{ik\alpha} \tag{9}$$

and w

$$w^{(l)}\left(\mathbf{t},\alpha\right)=\rho^{(l)}\left(\|\mathbf{t}\|\right)\sum_{u,v\in\mathbb{Z}}q_{u,v}^{(l)}\,e^{iu\alpha}\,e^{iv\angle(\mathbf{t})} \tag{10}$$

gives us:

$$\begin{aligned}\hat{x}^{(l)}\left(\mathbf{t},\alpha\right)=&\int_{\mathbb{R}^2}d\mathbf{t}'\,\rho^{(l)}\left(\|\mathbf{t}'-\mathbf{t}\|\right)\\&\cdot\int_0^{2\pi}d\alpha'\sum_{u,v\in\mathbb{Z}}q_{u,v}^{(l)}\,e^{iu(\alpha'-\alpha)}\,e^{iv\left(\angle(\mathbf{t}'-\mathbf{t})+\alpha\right)}\sum_{k'=-K}^{K}z_{k'}^{(l-1)}(\mathbf{t}')\,e^{ik'\alpha'}\end{aligned} \tag{11}$$

First, we perform some reordering of the equation:

$$\begin{aligned}\hat{x}^{(l)}\left(\mathbf{t},\alpha\right)=&\int_{\mathbb{R}^2}d\mathbf{t}'\,\rho^{(l)}\left(\|\mathbf{t}'-\mathbf{t}\|\right)\sum_{k'=-K}^{K}\sum_{u,v\in\mathbb{Z}}q_{u,v}^{(l)}\,z_{k'}^{(l-1)}(\mathbf{t}')\,e^{iv\angle(\mathbf{t}'-\mathbf{t})}\,e^{i(v-u)\alpha}\\&\cdot\int_0^{2\pi}d\alpha'\,e^{i(u+k')\alpha'}\end{aligned} \tag{12}$$

As $\int_0^{2\pi}d\alpha'\,e^{i(u+k')\alpha'}=\delta_{u,-k'}$, we can drop the sum over u, replacing $u\to-k'$:

$$\hat{x}^{(l)}\left(\mathbf{t},\alpha\right)=\int_{\mathbb{R}^2}d\mathbf{t}'\,\rho^{(l)}\left(\|\mathbf{t}'-\mathbf{t}\|\right)\sum_{k'=-K}^{K}\sum_{v\in\mathbb{Z}}q_{-k',v}^{(l)}\,z_{k'}^{(l-1)}(\mathbf{t}')\,e^{i(v-k')\angle(\mathbf{t}'-\mathbf{t})}\,e^{i(v+k')\alpha} \tag{13}$$

Next, we perform a sum shift of our infinite sum over $\mathbb{Z}$, replacing $v\to v-k'$:

$$\hat{x}^{(l)}\left(\mathbf{t},\alpha\right)=\int_{\mathbb{R}^2}d\mathbf{t}'\,\rho^{(l)}\left(\|\mathbf{t}'-\mathbf{t}\|\right)\sum_{k'=-K}^{K}\sum_{v\in\mathbb{Z}}q_{-k',v-k'}^{(l)}\,z_{k'}^{(l-1)}(\mathbf{t}')\,e^{iv\angle(\mathbf{t}'-\mathbf{t})}\,e^{iv\alpha} \tag{14}$$

This gives $e^{iv\alpha}$ on the right hand side, which allows extract the output Fourier coefficients after band-limiting. We rename $v\to k$ for consistency, yielding our convolution operation in the Fourier basis, as shown in the main paper:

$$\hat{z}_k^{(l)}\left(\mathbf{t},\alpha\right)=\int_{\mathbb{R}^2}d\mathbf{t}'\,\rho^{(l)}\left(\|\mathbf{t}'-\mathbf{t}\|\right)\sum_{k'=-K}^{K}q_{-k',k-k'}^{(l)}\,z_{k'}^{(l-1)}(\mathbf{t}')\,e^{i(k-k')\angle(\mathbf{t}'-\mathbf{t})} \tag{15}$$

B.3 Calculations for point clouds

To allow for continuous rotations and translations in the spatial domain, we use point clouds and consider our activations to be Dirac delta functions δ located at specific points $\mathbf{p}_1^{(l)},...,\mathbf{p}_{N^{(l)}}^{(l)}$ for each layer l:

$$x(l)\left(\mathbf{t},\alpha\right)=\sum_{n=1}^{N^{(l)}}\delta_{\mathbf{p}_n^{(l)}}(\mathbf{t})\,x_n^{(l)}(\alpha) \tag{16}$$

Using the Fourier representation from Eq. 9 from the previous section, we associate a set of Fourier coefficients $z_{k,n}^{(l)}$ (instead of angular-dependent functions $x_n^{(l)}$) with each point $p_n^{(l)}$:

$$z_k^{(l)}(\mathbf{t}) = \sum_{n=1}^{N^{(l)}} \delta_{\mathbf{p}_n^{(l)}}(\mathbf{t})\; z_{k,n}^{(l)} \tag{17}$$

Plugging in the point-based Fourier representation in our convolution operation (Eq. 15), the integration over $\mathbb{R}^2$ becomes a sum over all points in the previous layer:

$$\hat{z}_k^{(l)}(\mathbf{t}) = \sum_{n'=1}^{N^{(l-1)}} \rho^{(l)}\left(\|\mathbf{p}_{n'}^{(l-1)} - \mathbf{t}\|\right) \sum_{k'=-K}^{K} q_{-k',k-k'}^{(l)}\; z_{k',n'}^{(l-1)}\; e^{i(k-k')\angle(\mathbf{p}_{n'}^{(l-1)} - \mathbf{t})} \tag{18}$$

To discretize the result, we sample the output at specific points in the next layer. This gives our convolution operation for point clouds:

$$\hat{z}_{k,n}^{(l)} = \sum_{n'=1}^{N^{(l-1)}} \rho^{(l)}\left(\|\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)}\|\right) \sum_{k'=-K}^{K} q_{-k',k-k'}^{(l)}\; z_{k',n'}^{(l-1)}\; e^{i(k-k')\angle(\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)})} \tag{19}$$

B.4 Learnable filters and multiple feature channels

In general, both parts of our convolutional filter ρ and q can be learned by the neural network. For the angular function q, the straightforward approach is to directly use the Fourier coefficients $q_{u,v}$ as trainable parameters of the architecture. The radial profile ρ could be similarly constructed from basis functions and using trainable coefficients. Alternatively (and this is the approach we use in our paper), a sum over multiple pre-defined (fixed) radial profiles ρ_d, $d \in \{1, ..., D^{(l)}\}$ can be used, each associated with a different angular function, making the coefficients $q_{u,v}$ additionally depend on d:

$$\hat{z}_{k,n}^{(l)} = \sum_{n'=1}^{N^{(l-1)}} \sum_{d=1}^{D^{(l)}} \rho_d^{(l)}\left(\|\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)}\|\right) \sum_{k'=-K}^{K} q_{-k',k-k',d}^{(l)}\; z_{k',n'}^{(l-1)}\; e^{i(k-k')\angle(\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)})} \tag{20}$$

In the general case, we can have multiple unstructured feature channels per layer. As customary in regular CNNs, we can express this by using feature vectors $\mathbf{z}$ and a weight matrix $\mathbf{Q}$ instead of single-valued coefficients z and q. In our convolution operation, we then perform a matrix-vector multiplication:

$$\hat{\mathbf{z}}_{k,n}^{(l)} = \sum_{n'=1}^{N^{(l-1)}} \sum_{d=1}^{D^{(l)}} \rho_d^{(l)}\left(\|\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)}\|\right) \sum_{k'=-K}^{K} \mathbf{Q}_{-k',k-k',d}^{(l)}\; \mathbf{z}_{k',n'}^{(l-1)}\; e^{i(k-k')\angle(\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)})} \tag{21}$$

B.5 Practical considerations

Note that if input and output positions are identical ($\mathbf{p}_{n'}^{(l-1)} = \mathbf{p}_n^{(l)}$), the angle $\angle(\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)})$ becomes undefined. In this case, we evaluate the sum over k only for $k = k'$, where we assume $e^{i(k-k')\angle(\mathbf{p}_{n'}^{(l-1)} - \mathbf{p}_n^{(l)})} = e^0 = 1$. This only allows interactions between coefficients of the identical order (which is unproblematic, as in this case, only rotation-invariant spatial filters are used).

C NeurIPS 2021 Paper 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 [5]. The ModelNet-40 case discusses important aspects explicitly.
   (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]

References

[1] V. Gottemukkula. Polynomial activation functions. In *8th International Conference on Learning Representations (ICLR), retracted paper*, 2020. URL https://openreview.net/forum?id=rkxsgkHKvH.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *IEEE International Conference on Computer Vision, (ICCV)*, 2015. URL https://doi.org/10.1109/ICCV.2015.123.

[3] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

[4] 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(1):1929–1958, 2014. URL http://dl.acm.org/citation.cfm?id=2670313.

[5] M. Weiler and G. Cesa. General E(2)-equivariant Steerable CNNs. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/45d6637b718d0f24a237069fe41b0db4-Abstract.html.

[6] R. Wiersma, E. Eisemann, and K. Hildebrandt. CNNs on surfaces using rotation-equivariant features. *ACM Trans. Graph.*, 39(4):92, 2020. URL https://doi.org/10.1145/3386569.3392437.

[7] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic Networks: Deep translation and rotation equivariance. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. URL https://doi.org/10.1109/CVPR.2017.758.

[8] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. URL https://doi.org/10.1109/CVPR.2015.7298801.

[9] C. Yuksel. Sample elimination for generating poisson disk sample sets. *Comput. Graph. Forum*, 34(2): 25–32, 2015. URL https://doi.org/10.1111/cgf.12538.