SPHERICAL CNNs ON UNSTRUCTURED GRIDS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present an efficient convolution kernel for Convolutional Neural Networks (CNNs) on unstructured grids using parameterized differential operators while focusing on spherical signals such as panorama images or planetary signals. To this end, we replace conventional convolution kernels with linear combinations of differential operators that are weighted by learnable parameters. Differential operators can be efficiently estimated on unstructured grids using one-ring neighbors, and learnable parameters can be optimized through standard back-propagation. As a result, we obtain extremely efficient neural networks that match or outperform state-of-the-art network architectures in terms of performance but with a significantly lower number of network parameters. We evaluate our algorithm in an extensive series of experiments on a variety of computer vision and climate science tasks, including shape classification, climate pattern segmentation, and omnidirectional image semantic segmentation. Overall, we present (1) a novel CNN approach on unstructured grids using parameterized differential operators for spherical signals, and (2) we show that our unique kernel parameterization allows our model to achieve the same or higher accuracy with significantly fewer network parameters.

1 INTRODUCTION

A wide range of machine learning problems in computer vision and related areas require processing signals in the spherical domain; for instance, omnidirectional RGBD images from commercially available panorama cameras, such as Matterport (Chang et al., 2017), panoramic videos coupled with LIDAR scans from self-driving cars (Geiger et al., 2013), or planetary signals in scientific domains such as climate science (Racah et al., 2017). Unfortunately, naively mapping spherical signals to planar domains results in unwanted distortions. Specifically, projection artifacts near polar regions and handling of boundaries makes learning with 2D convolutional neural networks (CNNs) particularly challenging and inefficient. Very recent work, such as Cohen et al. (2018) and Esteves et al. (2018), propose network architectures that operate natively in the spherical domain, and are invariant to rotations in the $SO(3)$ group. Such invariances are desirable in a set of problems – e.g., machine learning problems of molecules – where gravitational effects are negligible and orientation is arbitrary. However, for yet a different class of problems at large, assumed orientation information is crucial to the predictive capability of the network. A good example of such problems is the MNIST digit recognition problem, where orientation plays the dominant role in distinguishing digits “6” and “9”. Other examples include omnidirectional images, where images are naturally oriented by gravity; and planetary signals, where planets are naturally oriented by their axis of rotation.

In this work, we present a new convolutional kernel for CNNs on arbitrary manifolds and topologies, discretized by an unstructured grid (i.e., mesh), and focus on its applications towards the spherical domain approximated by an icosahedral spherical mesh. We propose and evaluate the use of a new parameterization scheme for CNN convolution kernels, which we call Parameterized Differential Operators (PDOs), which is easy to implement on unstructured grids. This parameterization scheme utilizes only 4 parameters for each kernel, and achieves significantly better performance than competing methods, with a much smaller parameter budget. In particular, we illustrate its use towards various machine learning problems in computer vision and climate science.
Figure 1: Illustration for the MeshConv operator using parameterized differential operators to replace conventional learnable convolutional kernels.

In summary, our contributions are as follows:

• We present a general approach for orientable CNNs on unstructured grids using parameterized differential operators.
• We show that our spherical model achieves significantly higher parameter efficiency compared to state-of-the-art network architectures for 3D classification tasks and spherical image semantic segmentation.

We organize the structure of the paper as follows. We first provide an overview of related studies in the literature in Sec.2, we then introduce details of our methodology in Sec.3, followed by an empirical assessment of the effectiveness of our model in Sec.4. Finally, we evaluate the design choices of our kernel parameterization scheme in Sec.5.

2 BACKGROUND

Spherical CNNs A first and foremost concern for processing spherical signals is regarding distortions introduced by the curved spherical surface. Su & Grauman (2017) process equirectangular images with regular convolutions with increased kernel sizes near polar regions where greater distortions are introduced by the planar mapping. Coors et al. (2018) and Zhao et al. (2018) use a constant kernel that samples points on the tangent plane of the spherical image to reduce distortions. A slightly different line of literature explores rotational-equivariant implementations of spherical CNNs. Cohen et al. (2018) proposed spherical convolutions with intermediate feature maps in SO(3) that are rotational-equivariant. Esteves et al. (2018) used spherical harmonic basis to achieve similar results.

Reparameterized Convolutional Kernel Related to our approach in using parameterized differential operators, several works utilize the diffusion kernel for efficient Machine Learning and CNNs. Kondor & Lafferty (2002) was among the first to suggest the use of diffusion kernel on graphs. Atwood & Towsley (2016) propose Diffusion-Convolutional Neural Networks (DCNN) for efficient convolution on graph structured data. Boscaini et al. (2016) introduce a generalization of classic CNNs to non-Euclidean domains by using a set of oriented anisotropic diffusion kernels. Ruthotto & Haber (2018) explore the reparameterization of convolutional kernels using parabolic and hyperbolic differential basis with regular grid images.

Non-Euclidean Convolutions Related to our work on performing convolutions on manifolds represented by an unstructured grid (i.e., mesh), works in geometric deep learning address similar problems (Bronstein et al. 2017). Other methods perform graph convolution by parameterizing the convolution kernels in the spectral domain, thus converting the convolution step into a spectral dot product (Bruna et al. 2014; Defferrard et al. 2016; Kipf & Welling 2017; Yi et al. 2017). Masci et al. (2015) perform convolutions directly on manifolds using cross-correlation based on geodesic distances and Maron et al. (2017) use an optimal surface parameterization method (seamless toric
covers) to parameterize genus-zero shapes into 2D signals for analysis using conventional planar CNNs.

**Image Semantic Segmentation**  Image semantic segmentation is a classical problem in computer vision, and there has been an impressive body of literature studying semantic segmentation of planar images [Ronneberger et al. (2015); Badrinarayanan et al. (2015); Long et al. (2015); Jégou et al. (2017); Wang et al. (2018a)]. Song et al. (2017) study semantic segmentation of equirectangular omnidirectional images, but in the context of image inpainting, where only a partial view is given as input. Armeni et al. (2017); Chang et al. (2017) provide benchmarks for semantic segmentation of 360 panorama images. In the 3D learning literature, researchers have looked at 3D semantic segmentation on point clouds or voxels [Dai et al., 2017; Qi et al., 2017a; Wang et al., 2018b; Tchapmi et al., 2017; Dai et al., 2017b]. Our method also targets the application domain of image segmentation by providing a more efficient convolutional operator for spherical domains, for instance, focusing on panoramic images [Chang et al., 2017].

## 3 Method

### 3.1 Parameterized Differential Operators

We present a novel scheme for efficiently performing convolutions on manifolds approximated by a given underlying mesh, using what we name as Parameterized Differential Operators. To this end, we reparameterize the learnable convolution kernel as a linear combination of differential operators. Such reparameterization provides two distinct advantages: first, we can drastically reduce the number of parameters per given convolution kernel, allowing for an efficient and lean learning space; second, as opposed to the cross-correlation type convolution on mesh surfaces [Masci et al., 2015], which requires large amounts of geodesic computations and interpolations, first and second order differential operators can be efficiently estimated using only the one-ring neighborhood.

In order to illustrate the concept of PDOs, we draw comparisons to the conventional $3 \times 3$ convolution kernel in the regular grid domain. The $3 \times 3$ kernel parameterized by parameters $\theta: G^3_{3 \times 3}$ can be written as a linear combination of basis kernels which can be viewed as delta functions at constant offsets:

$$G^3_{3 \times 3}(x, y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} \delta(x - i, y - j)$$  \hspace{1cm} (1)

Due to the linearity of the cross-correlation operator ($\ast$), the output feature map can be expressed as a linear combination of the input function cross-correlated with different basis functions. Defining the linear operator $\Delta_{ij}$ to be the cross-correlation with a basis delta function, we have:

$$\Delta_{ij} \mathcal{F}(x, y) := \mathcal{F}(x, y) \ast \delta(x - i, y - j)$$  \hspace{1cm} (2)

$$\mathcal{F}(x, y) \ast G^3_{3 \times 3}(x, y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} \theta_{ij} \Delta_{ij} \mathcal{F}(x, y)$$  \hspace{1cm} (3)

In our formulation of PDOs, we replace the cross-correlation linear operators $\Delta_{ij}$ with differential operators of varying orders. Similar to the linear operators resulting from cross-correlation with basis functions, differential operators are linear, and approximate local features. In contrast to cross-correlations on manifolds, differential operators on meshes can be efficiently computed using Finite Element basis, or derived by Discrete Exterior Calculus. In the actual implementation below, we choose the identity ($I$, 0th order differential, same as $\Delta_{00}$), derivatives in two orthogonal spatial dimensions ($\nabla_x, \nabla_y$, 1st order differential), and the Laplacian operator ($\nabla^2$, 2nd order differential):

$$\mathcal{F}(x, y) \ast G^d_{\theta} = \theta_0 I \mathcal{F} + \theta_1 \nabla_x \mathcal{F} + \theta_2 \nabla_y \mathcal{F} + \theta_3 \nabla^2 \mathcal{F}$$  \hspace{1cm} (4)

The identity ($I$) of the input function is trivial to obtain. The first derivative ($\nabla_x, \nabla_y$) can be obtained by first computing the per-face gradients, and then using area-weighted average to obtain per-vertex
3.2 Icosahedral Spherical Mesh

The icosahedral spherical mesh (Baumgardner & Frederickson, 1985) is among the most uniform and accurate discretizations of the sphere. A spherical mesh can be obtained by progressively subdividing each face of the unit icosahedron into four equal triangles and reprojecting each node to unit distance from the origin. Apart from the uniformity and accuracy of the icosahedral sphere, the subdivision scheme for the triangles provides a natural coarsening and refinement scheme for the grid that allows for easy implementations of pooling and unpooling routines associated with CNN architectures. See Fig. 1 for a schematic of the level-3 icosahedral spherical mesh.

For the ease of discussion, we adopt the following naming convention for mesh resolution: starting with the unit icosahedron as the level-0 mesh, each progressive mesh resolution is one level above the previous. Hence for a level-\( l \) mesh:

\[
\begin{align*}
    n_f &= 20 \cdot 4^l; \\
    n_e &= 30 \cdot 4^l; \\
    n_v &= n_e - n_f + 2
\end{align*}
\]

3.3 Model Architecture Design

A detailed schematic for the neural architectures in this study is presented in Fig. 2. The schematic includes architectures for both the classification and regression network, which share a common en-
Table 1: Results on the MNIST dataset for validating the use of Parameterized Differential Operators. Our model achieves state-of-the-art performance with comparable number of training parameters.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy(%)</th>
<th>Number of Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2CNN (Cohen et al., 2018)</td>
<td>96.00</td>
<td>58k</td>
</tr>
<tr>
<td>SphereNet (Coors et al., 2018)</td>
<td>94.41</td>
<td>196k</td>
</tr>
<tr>
<td>Ours</td>
<td>99.23</td>
<td>62k</td>
</tr>
</tbody>
</table>

Minor adjustments are made for different tasks, mainly surrounding adjusting the number of input and output layers to process signals at varied resolutions. A detailed breakdown for model architectures, as well as training details for each task in the Experiment section (Sec. 4), is provided in the appendix (Appendix Sec. A).

4 EXPERIMENTS

4.1 SPHERICAL MNIST

To validate the use of parameterized differential operators to replace conventional convolution operators, we implemented such neural networks towards solving the classic computer vision benchmark task: the MNIST digit recognition problem (LeCun, 1998).

Experiment Setup We follow Cohen et al. (2018) by projecting the pixelated digits onto the surface of the unit sphere. We further move the digits to the equator to prevent coordinate singularity at the poles. We benchmark our model against two other implementations of spherical CNNs: a rotational-invariant model by Cohen et al. (2018) and an orientable model by Coors et al. (2018). All models are trained and tested with non-rotated digits to illustrate the performance gain from orientation information.

Results and Discussion Our model outperforms its counterparts by a significant margin, achieving the best performance among comparable algorithms, with comparable number of parameters. We attribute the success in our model to the gain in orientation information, which is indispensable for many vision tasks. In contrast, S2CNN (Cohen et al., 2018) is rotational-invariant, and thus has difficulties distinguishing digits “6” and “9”.

4.2 3D OBJECT CLASSIFICATION

We use the ModelNet40 benchmark (Wu et al., 2015), a 40-class 3D classification problem, to illustrate the applicability of our spherical method to a wider set of problems in 3D learning. For this study, we look into two aspects of our model: peak performance and parameter efficiency.

Experiment Setup To use our spherical CNN model for the object classification task, we preprocess the 3D geometries into spherical signals. We follow Cohen et al. (2018) for preprocessing the 3D CAD models. First, we normalize and translate each mesh to the coordinate origin. We then encapsulate each mesh with a bounding level-5 unit sphere and perform ray-tracing from each point to the origin. We record the distance from the spherical surface to the mesh, as well as the $\sin$, $\cos$ of the incident angle. The data is further augmented with the 3 channels corresponding to the convex hull of the input mesh, forming a total of 6 input channels. An illustration of the data preprocessing process is presented in Fig. 5. For peak performance, we compare the best performance achievable by our model with other 3D learning algorithms. For the parameter efficiency study, we progressively reduce the number of feature layers in all models without changing the overall model architecture. Then, we evaluate the models after convergence in 250 epochs. We benchmark our
Figure 3: Parameter efficiency study on ModelNet40, benchmarked against representative 3D learning models consuming different input data representations: PointNet++ using point clouds as input, VoxNet consuming binary-voxel inputs, S2CNN consuming the same input structure as our model (spherical signal). The abscissa is drawn based on log scale.

Results and Discussion Fig. 3 shows a comparison of model performance versus number of parameters. Our model achieves the best performance across all parameter ranges. In the low-parameter range, our model is able to achieve approximately 60% accuracy for the 40-class 3D classification task with a mere 2000+ parameters. Table 2 shows a comparison of peak performance between models. At peak performance, our model is on-par with comparable state-of-the-art models, and achieves the best performance among models consuming spherical input signals.

We use the author’s open-source implementations: PointNet++, VoxNet (Qi et al., 2016), and S2CNN (Cohen et al., 2018).

Table 2: Results on ModelNet40 dataset. Our method compares favorably with state-of-the-art, and achieves best performance among networks utilizing spherical input signals.

<table>
<thead>
<tr>
<th>Model</th>
<th>Input</th>
<th>Accu. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DShapeNets (Wu et al., 2015)</td>
<td>voxels</td>
<td>84.7</td>
</tr>
<tr>
<td>VoxNet (Maturana &amp; Sehery, 2015)</td>
<td>voxels</td>
<td>85.9</td>
</tr>
<tr>
<td>PointNet (Qi et al., 2017a)</td>
<td>points</td>
<td>89.2</td>
</tr>
<tr>
<td>PointNet++ (Qi et al., 2017b)</td>
<td>points</td>
<td>91.9</td>
</tr>
<tr>
<td>DGCNN (Wang et al., 2018b)</td>
<td>points</td>
<td>92.2</td>
</tr>
<tr>
<td>S2CNN (Cohen et al., 2018)</td>
<td>spherical</td>
<td>85.0</td>
</tr>
<tr>
<td>SphericalCNN (Esteves et al., 2018)</td>
<td>spherical</td>
<td>88.9</td>
</tr>
<tr>
<td>Ours</td>
<td>spherical</td>
<td>90.5</td>
</tr>
</tbody>
</table>

Figure 4: Parameter efficiency study on 2D3DS semantic segmentation task. Our spherical segmentation model outperforms the planar and point-based counterparts by a significant margin across all parameter regimes.

Figure 5: Illustration of spherical signal rendering process for a given 3D CAD model.
4.3 Omnidirectional Image Segmentation

We illustrate the semantic segmentation capability of our network on the omnidirectional image segmentation task. We use the Stanford 2D3DS dataset (Armeni et al., 2017) for this task. The 2D3DS dataset consists of 1,413 equirectangular images with RGB+depth channels, as well as semantic labels across 13 different classes. The panoramic images are taken in 6 different areas, and the dataset is officially split for a 3-fold cross validation. While we are unable to find reported results on the semantic segmentation of these omnidirectional images, we benchmark our spherical segmentation algorithm against classic 2D image semantic segmentation networks as well as a 3D point-based model, trained and evaluated on the same data.

Experiment Setup First, we preprocess the data into a spherical signal by sampling the original rectangular images at the latitude-longitudes of the spherical mesh vertex positions. Input RGB-D channels are interpolated using bilinear interpolation, while semantic labels are acquired using nearest-neighbor interpolation. We input and output spherical signals at the level-5 resolution. We use the official 3-fold cross validation to train and evaluate our results. We benchmark our semantic segmentation results against two classic semantic segmentation networks: the U-Net (Ronneberger et al., 2015) and FCN8s (Long et al., 2015). We also compared our results with a 3D point-based method, PointNet++ (Qi et al., 2017b) using \((x, y, z, r, g, b)\) inputs reconstructed from panoramic RGBD images. We evaluate the network performance under two standard metrics: mean Intersection-over-Union (mIoU), and pixel-accuracy. Similar to Sec. 4.2, we evaluate the models under two settings: peak performance and a parameter efficiency study by varying model parameters. We progressively decimate the number of feature layers uniformly for all models to study the effect of model complexity on performance.

Results and Discussion Fig. 6 compares our model against state-of-the-art baselines. Our spherical segmentation outperforms the planar baselines for all parameter ranges, and more significantly so compared to the 3D PointNet++. We attribute PointNet++’s performance to the small sample size of the training data. Fig. 6 shows a visualization of our semantic segmentation performance compared to the ground truth and the planar baselines.
Table 3: We achieve better accuracy compared to our baseline for climate pattern segmentation.

<table>
<thead>
<tr>
<th>Model</th>
<th>Background (%)</th>
<th>TC (%)</th>
<th>AR (%)</th>
<th>Mean (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mudigonda et al. (2017)</td>
<td>97</td>
<td>74</td>
<td>65</td>
<td>78.67</td>
</tr>
<tr>
<td>Ours</td>
<td>97</td>
<td>94</td>
<td>93</td>
<td>94.67</td>
</tr>
</tbody>
</table>

Table 4: Results for the ablation study. The choice of kernel that includes all differential operator components achieves the best accuracy, validating our choice of kernel in Eqn.4.

<table>
<thead>
<tr>
<th>Convolution kernel</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I + \frac{\partial}{\partial y} + \nabla^2$</td>
<td>0.8748</td>
</tr>
<tr>
<td>$I + \frac{\partial}{\partial x} + \nabla^2$</td>
<td>0.8809</td>
</tr>
<tr>
<td>$I + \nabla^2$</td>
<td>0.8801</td>
</tr>
<tr>
<td>$I + \frac{\partial}{\partial x} + \frac{\partial}{\partial y}$</td>
<td>0.8894</td>
</tr>
<tr>
<td>$I + \frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \nabla^2$</td>
<td><strong>0.8979</strong></td>
</tr>
</tbody>
</table>

Figure 7: Visualization of segmentation for Atmospheric River (AR). Plotted in the background is Integrated Vapor Transport (IVT), whereas red masks indicates the existence of AR.

4.4 CLIMATE PATTERN SEGMENTATION

To further illustrate the capabilities of our model, we evaluate our model on the climate pattern segmentation task. We follow Mudigonda et al. (2017) for preprocessing the data and acquiring the ground-truth labels for this task. This task involves the segmentation of Atmospheric Rivers (AR) and Tropical Cyclones (TC) in global climate model simulations. Following Mudigonda et al. (2017), we analyze outputs from a 20-year run of the Community Atmospheric Model v5 (CAM5) (Neale et al., 2010). We benchmark our performance against Mudigonda et al. (2017) for the climate segmentation task to highlight our model performance. We preprocess the data to level-5 resolution.

Results and Discussion Segment accuracy is presented in Table 3. Our model achieves better segmentation accuracy as compared to the baseline models. The baseline model (Mudigonda et al., 2017) trains and tests on random crops of the global data, whereas our model inputs the entire global data and predicts at the same output resolution as the input. Processing full global data allows the network to acquire better holistic understanding of the information, resulting in better overall performance.

5 ABLATION STUDY

We further perform an ablation study for justifying the choice of differential operators for our convolution kernel (as in Eqn. 4). We use the ModelNet40 classification problem as a toy example and use a 250k parameter model for evaluation. We choose various combinations of differential operators, and record the final classification accuracy. Results for the ablation study is presented in Table 4. Our choice of differential operator combinations in Eqn. 4 achieves the best performance among other choices, and the network performance improves with increased differential operators, thus allowing for more degrees of freedom for the kernel.

6 CONCLUSION

We have presented a novel method for performing convolution on unstructured grids using parameterized differential operators as convolution kernels. Our results demonstrate its applicability to machine learning problems with spherical signals and show significant improvements in terms of overall performance and parameter efficiency. We believe that these advances are particularly relevant with the widespread of omnidirectional signals, for instance, as captured by real-world 3D or LIDAR panorama sensors.
REFERENCES


APPENDIX

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshConv($a, b$)</td>
<td>Mesh convolution layer with $a$ input channels and producing $b$ output channels</td>
</tr>
<tr>
<td>MeshConv($a, b$)$^T$</td>
<td>Mesh transpose convolution layer with $a$ input channels and producing $b$ output channels</td>
</tr>
<tr>
<td>BN</td>
<td>Batch Normalization.</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit activation function</td>
</tr>
<tr>
<td>DownSamp</td>
<td>Downsampling spherical signal at the next resolution level.</td>
</tr>
<tr>
<td>ResBlock($a, b, c$)</td>
<td>As illustrated in Fig. 1, where $a, b, c$ stands for input channels, bottleneck channels, and output channels.</td>
</tr>
<tr>
<td>$[\cdot]_{L_i}$</td>
<td>The layers therein is at a mesh resolution of $L_i$.</td>
</tr>
<tr>
<td>Concat</td>
<td>Concatenate skip layers of same resolution.</td>
</tr>
</tbody>
</table>

Table 5: Network architecture notation list

A NETWORK ARCHITECTURE AND TRAINING DETAILS

In this section we provide detailed network architecture and training parameters for reproducing our results in Sec. 4. We use Fig. 2 as a reference.

A.1 SPHERICAL MNIST

Architecture Since the input signal for this experiment is on a level-4 mesh, the input pathway is slightly altered. The network architecture is as follows:

$[\text{MeshConv}(1, 16) + \text{BN} + \text{ReLU}]_{L_4} + [\text{DownSamp} + \text{ResBlock}(16, 16, 64)]_{L_3} + [\text{DownSamp} + \text{ResBlock}(64, 64, 256)]_{L_2} + \text{AvgPool} + \text{MLP}(256, 10)$

Total number of parameters: 61658

Training details We train our network with a batch size of 16, initial learning rate of $1 \times 10^{-2}$, step decay of 0.5 per 10 epochs, and use the Adam optimizer. We use the cross-entropy loss for training the classification network.

A.2 3D OBJECT CLASSIFICATION

Architecture The input signal is at a level-5 resolution. The network architecture closely follows that in the schematics in Fig. 2. We present two network architectures, one that corresponds to the network architecture with the highest accuracy score (the full model), and another that scales well with low parameter counts (the lean model). The full model:

$[\text{MeshConv}(6, 32) + \text{BN} + \text{ReLU}]_{L_5} + [\text{DownSamp} + \text{ResBlock}(32, 32, 128)]_{L_4} + [\text{DownSamp} + \text{ResBlock}(128, 128, 512)]_{L_3} + [\text{DownSamp} + \text{ResBlock}(512, 512, 2048)]_{L_2} + \text{AvgPool} + \text{MLP}(2048, 40)$

Total number of parameters: 3737160

The lean model:

$[\text{MeshConv}(6, 8) + \text{BN} + \text{ReLU}]_{L_5} + [\text{DownSamp} + \text{ResBlock}(8, 8, 16)]_{L_4} + [\text{DownSamp} + \text{ResBlock}(16, 16, 64)]_{L_3} + [\text{DownSamp} + \text{ResBlock}(64, 64, 256)]_{L_2} + \text{AvgPool} + \text{MLP}(256, 40)$

Total number of parameters: 70192

Training details We train our network with a batch size of 16, initial learning rate of $5 \times 10^{-3}$, step decay of 0.7 per 25 epochs, and use the Adam optimizer. We use the cross-entropy loss for training the classification network.
A.3 Omnidirectional Image Segmentation

Architecture. Input signal is sampled at a level-5 resolution. The network architecture is identical to the segmentation network in Fig.2. Encoder parameters are as follows:

\[[\text{MeshConv}(4,32)]_{L5} + [\text{DownSamp} + \text{ResBlock}(32, 32, 64)]_{L4} + [\text{DownSamp} + \text{ResBlock}(64, 64, 128)]_{L3} + [\text{DownSamp} + \text{ResBlock}(128, 128, 256)]_{L2} + [\text{DownSamp} + \text{ResBlock}(256, 256, 512)]_{L1} + [\text{DownSamp} + \text{ResBlock}(512, 512, 512)]_{L0}\]

Decoder parameters are as follows:

\[[\text{MeshConv}^T(512,512) + \text{Concat} + \text{ResBlock}(1024, 256, 256)]_{L1} + [\text{MeshConv}^T(256,256) + \text{Concat} + \text{ResBlock}(512, 128, 128)]_{L2} + [\text{MeshConv}^T(128,128) + \text{Concat} + \text{ResBlock}(256, 64, 64)]_{L3} + [\text{MeshConv}^T(64,64) + \text{Concat} + \text{ResBlock}(128, 32, 32)]_{L4} + [\text{MeshConv}^T(32,32) + \text{Concat} + \text{ResBlock}(64, 32, 32)]_{L5} + [\text{MeshConv}(32,15)]_{L5}\]

Total number of parameters: 5180239

Training details. Note that the number of output channels is 15, since the 2D3DS dataset has two additional classes (invalid and unknown) that are not evaluated for performance. We train our network with a batch size of 16, initial learning rate of \(1 \times 10^{-2}\), step decay of 0.7 per 20 epochs, and use the Adam optimizer. We use the weighted cross-entropy loss for training. We weight the loss for each class using the following weighting scheme:

\[w_c = \frac{1}{1.02 + \log(f_c)}\]  \hspace{1cm} (7)

where \(w_c\) is the weight corresponding to class \(c\), and \(f_c\) is the frequency by which class \(c\) appears in the training set. We use zero weight for the two dropped classes (invalid and unknown).

A.4 Climate Pattern Segmentation

Architecture. We use the same network architecture as the Omnidirectional Image Segmentation task in Sec. A.3. Minor difference being that all feature layers are cut by 1/4.

Total number of parameters: 328339

Training details. We train our network with a batch size of 256, initial learning rate of \(1 \times 10^{-2}\), step decay of 0.4 per 20 epochs, and use the Adam optimizer. We train using weighted cross-entropy loss, using the same weighting scheme as in Eqn.7.

B Detailed Statistics for 2D3DS Segmentation

We provide detailed statistics for the 2D3DS semantic segmentation task. We evaluate our model’s per-class performance against the benchmark models. All statistics are mean over 3-fold cross validation.

<table>
<thead>
<tr>
<th>Model</th>
<th>Mean</th>
<th>beam</th>
<th>board</th>
<th>bookcase</th>
<th>ceiling</th>
<th>chair</th>
<th>clutter</th>
<th>column</th>
<th>door</th>
<th>floor</th>
<th>sofa</th>
<th>table</th>
<th>wall</th>
<th>window</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNet</td>
<td>0.5880</td>
<td>0.1777</td>
<td>0.4038</td>
<td>0.5914</td>
<td>0.9180</td>
<td>0.5088</td>
<td>0.4603</td>
<td>0.0875</td>
<td>0.4398</td>
<td>0.9480</td>
<td>0.2823</td>
<td>0.6855</td>
<td>0.7117</td>
<td>0.3481</td>
</tr>
<tr>
<td>FCNs</td>
<td>0.4842</td>
<td>0.1439</td>
<td>0.4413</td>
<td>0.3952</td>
<td>0.8971</td>
<td>0.5244</td>
<td>0.5799</td>
<td>0.0564</td>
<td>0.5962</td>
<td>0.9661</td>
<td>0.0322</td>
<td>0.6614</td>
<td>0.7359</td>
<td>0.2682</td>
</tr>
<tr>
<td>PointNet++</td>
<td>0.3349</td>
<td>0.1928</td>
<td>0.2942</td>
<td>0.3177</td>
<td>0.5488</td>
<td>0.4145</td>
<td>0.2246</td>
<td>0.3110</td>
<td>0.2701</td>
<td>0.4596</td>
<td>0.3391</td>
<td>0.4976</td>
<td>0.3535</td>
<td>0.1413</td>
</tr>
<tr>
<td>Ours</td>
<td>0.5463</td>
<td>0.1964</td>
<td>0.4556</td>
<td>0.4904</td>
<td>0.9356</td>
<td>0.6382</td>
<td>0.4309</td>
<td>0.2798</td>
<td>0.6321</td>
<td>0.9638</td>
<td>0.2103</td>
<td>0.6996</td>
<td>0.7457</td>
<td>0.3897</td>
</tr>
</tbody>
</table>

Table 6: Per-class accuracy comparison with baseline models

<table>
<thead>
<tr>
<th>Model</th>
<th>Mean</th>
<th>beam</th>
<th>board</th>
<th>bookcase</th>
<th>ceiling</th>
<th>chair</th>
<th>clutter</th>
<th>column</th>
<th>door</th>
<th>floor</th>
<th>sofa</th>
<th>table</th>
<th>wall</th>
<th>window</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNet</td>
<td>0.3587</td>
<td>0.0853</td>
<td>0.2721</td>
<td>0.3072</td>
<td>0.7857</td>
<td>0.3551</td>
<td>0.2883</td>
<td>0.0487</td>
<td>0.3377</td>
<td>0.8911</td>
<td>0.0817</td>
<td>0.3851</td>
<td>0.5878</td>
<td>0.2392</td>
</tr>
<tr>
<td>FCNs</td>
<td>0.3560</td>
<td>0.0572</td>
<td>0.3139</td>
<td>0.2804</td>
<td>0.7981</td>
<td>0.5623</td>
<td>0.2973</td>
<td>0.0153</td>
<td>0.4081</td>
<td>0.8884</td>
<td>0.0263</td>
<td>0.3809</td>
<td>0.5849</td>
<td>0.1859</td>
</tr>
<tr>
<td>PointNet++</td>
<td>0.2312</td>
<td>0.0909</td>
<td>0.1503</td>
<td>0.2210</td>
<td>0.4775</td>
<td>0.2981</td>
<td>0.1610</td>
<td>0.0782</td>
<td>0.1866</td>
<td>0.4426</td>
<td>0.1844</td>
<td>0.3332</td>
<td>0.3061</td>
<td>0.0755</td>
</tr>
<tr>
<td>Ours</td>
<td>0.3829</td>
<td>0.0869</td>
<td>0.3268</td>
<td>0.3544</td>
<td>0.8216</td>
<td>0.4197</td>
<td>0.2562</td>
<td>0.1012</td>
<td>0.4159</td>
<td>0.8702</td>
<td>0.0783</td>
<td>0.4170</td>
<td>0.6167</td>
<td>0.2349</td>
</tr>
</tbody>
</table>

Table 7: Mean IoU comparison with baseline models