Appendix for Deep Iterative Surface Normal Estimation

1. Overview

The supplemental materials contain details about the graph neural network in Section 2, information about the implementation in Section 3, a short discussion about the spatial transformer in 4, and an additional analysis of accuracy over re-weighting iterations in Section 5. Further, we show results for transferring models between different neighborhood sizes in Section 6 and qualitative results for the whole PCPNet test set in Section 7.

2. Architecture Details

The graph neural network for the deep kernel parameterization follows a general message passing scheme [1] with edge update function

$$\mathbf{f}_e(i,j) = h\big(\mathbf{f}(i) \,|\, \mathbf{d}_{i,j} \,|\, \mathbf{prf}(i,j)\big) \tag{1}$$

and node update function

$$\mathbf{f}(i) = \gamma\Big(\frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} \mathbf{f}_e(i,j)\Big), \tag{2}$$

consisting of 6 MLPs, h_i and γ_i for $i \in \{1,2,3\}$. Together with the kernel MLP ψ, all functions are detailed in Table 1 The h_i and ψ networks are shared over all edges in the neighborhood graph while the γ_i are shared over all points. Additionally, all MLPs are shared over the iterations of the algorithm. Each MLP consists of two linear layers, seperated by a ReLU non-linearity. Layer sizes are given in Table 1. All in all, the networks contain 7981 parameters and fulfill the following properties.

Permutation Invariance Neighborhood aggregation is performed using an average operator, which is invariant regarding the order of points. Since there are no other functions over sets of points, the resulting network is permutation invariant. We refer to [2] for further discussion. It should be noted that PointNet can also be expressed in the same message passing scheme and is permutation invariant for the same reasons.

Varying neighborhood sizes For the cases in which we decide to use a radius graph instead of a k-nn graph, the network allows differently sized neighborhoods in one graph, since all parameters are shared over edge or nodes and the only operation over the whole neighborhood, the average, is agnostic to the neighborhood size.

Network	Architecture		
h_1	$L(32)$,	$ReLU$,	$L(16)$
γ_1	$L(32)$,	$ReLU$,	$L(8)$
h_2	$L(32)$,	$ReLU$,	$L(16)$
γ_2	$L(32)$,	$ReLU$,	$L(8)$
h_3	$L(32)$,	$ReLU$,	$L(16)$
γ_3	$L(32)$,	$ReLU$,	$L(12)$
ψ	$L(64)$,	$ReLU$,	$L(1)$

Table 1: Details of the used graph neural network for iterative re-weighting. $L(x)$ stands for a fully-connected layer with x output neurons.

Locality Due to using only local operators, the presented algorithm can be applied on partial point clouds, which is of importance for many practical applications.

3. Implementation Details

The implementation of the proposed algorithm is based on the *Pytorch Geometric* library [1] and uses the provided scheme consisting of scattering and gathering between node and edge feature space. Therefore, varying neighborhood sizes (*e.g.* varying node degree) can still be handled in parallel on the GPU by parallelization in graph edge space.

For parallel eigendecomposition of a large number of symmetric 3×3 matrices and for the parallel quaternion to rotation matrix map, we provide our own Pytorch extensions which will be made available online. We provide efficient forward and backward steps on GPU and CPU.

4. Rotational Spatial Transformer

Our spatial transformer learns to bring the point sets in canonical orientation, which leads to equivariant behaviour, as our results show. Directly parameterizing 3×3 matrices for the spatial transformer would lead to arbitrary affine transformations which can easily collapse or diverge during training. Thus, parameterizing the rotation group $SO(3)$ is

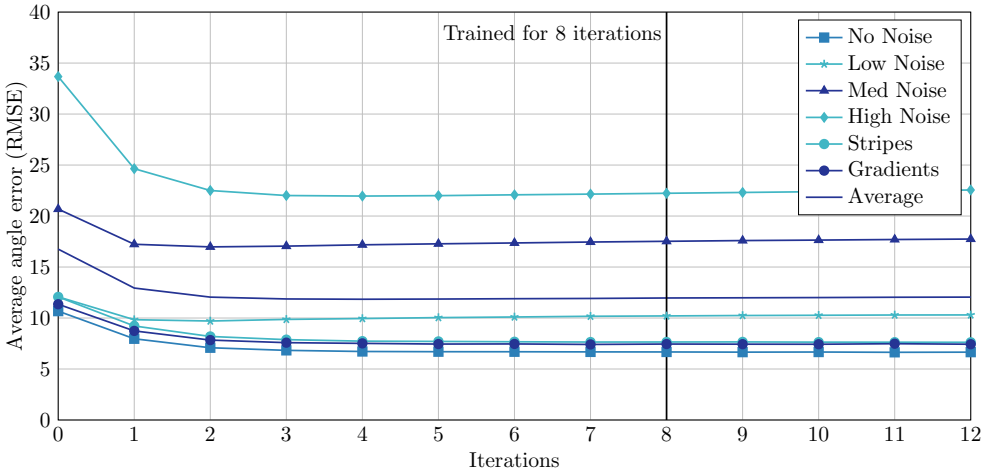

Figure 1: Test errors (RMSE) over iterations of the proposed algorithm. Iteration 0 shows results for unweighted PCA only. The network was trained on the training set for 8 iterations. For evaluation, we perform four additional iterations to evaluate stability.

	Trained on $k^{\text{train}} = 32$					Trained on $k^{\text{train}} = 64$					Trained on $k^{\text{train}} = 128$				
k^{test}	32	48	64	96	128	32	48	64	96	128	32	48	64	96	128
No noise	6.09	6.96	7.43	8.25	8.77	6.13	6.47	6.72	7.10	7.27	6.66	7.01	7.24	7.29	7.35
Noise ($\sigma = 0.00125$)	10.22	10.01	10.09	10.37	10.62	10.19	9.93	9.95	10.18	10.35	9.89	9.57	9.50	9.50	9.64
Noise ($\sigma = 0.006$)	18.17	17.44	17.22	17.08	17.05	18.28	17.43	17.18	17.01	16.94	20.98	18.40	17.63	17.07	16.90
Noise ($\sigma = 0.012$)	25.17	22.97	22.33	21.91	21.80	25.20	22.53	21.96	21.69	21.67	30.99	24.94	23.20	22.34	22.13
Density (Stripes)	7.22	7.92	8.51	9.43	9.90	7.21	7.55	7.73	8.16	8.34	7.80	8.14	8.37	8.61	8.67
Density (Gradients)	6.84	7.46	8.06	8.80	9.21	6.89	7.17	7.51	8.04	8.03	7.48	7.75	8.11	8.39	8.49
Average	12.28	12.12	12.27	12.64	12.89	12.31	11.85	11.84	12.00	12.10	13.97	12.63	12.34	12.20	12.20

Table 2: Results for transferring models between different neighborhood sizes k. Shown are RMSE values for models trained with $k^{\text{train}} \in \{32, 64, 128\}$, each tested with $k^{\text{test}} \in \{32, 48, 64, 96, 128\}$.

	Trained on $k^{\text{train}} = 32$					
k^{test}	2	4	8	16	24	32
No noise	17.26	7.23	5.63	5.36	5.77	6.09
Noise ($\sigma = 0.00125$)	54.02	49.66	33.65	13.80	10.74	10.22
Noise ($\sigma = 0.006$)	61.08	60.91	55.32	28.17	19.78	18.17
Noise ($\sigma = 0.012$)	61.29	61.26	58.89	41.37	28.99	25.17
Density (Stripes)	19.50	8.14	6.53	6.36	6.71	7.22
Density (Gradients)	22.89	8.44	6.51	6.23	6.57	6.84
Average	39.34	32.59	27.75	16.88	13.09	12.28

Table 3: Results for transferring the model trained on $k^{\text{train}} = 32$ to even smaller $k^{\text{test}} \in \{2, 4, 8, 16, 24, 32\}$ until the method breaks down. Note that $k^{\text{test}} = 2$ means 2 neighbors, excluding point i, so there are still 3 points in total for each neighborhood, avoiding underdefined plane fitting problems.

the more fitting choice for the given task. Unit quaternions are a good representation choice because they cover $SO(3)$ (twice) without any discontinuities, as exist in e.g. Euler angles or axis-angle representations. Discontinuities in the $SO(3)$ representation would force the network to sometimes predict very different values for $SO(3)$ elements that lie next to each other on the Lie group manifold, which can lead to unstable gradients.

5. Behaviour over iterations

The algorithm is trained for $L = 8$ (performing 8 iterations of re-weighting), where we compute a loss and perform an optimization step after each iteration. It produces normal vector estimations after each iteration, which can be analyzed quantitatively. The RMSE results for the PCPNet test set over algorithm iterations are shown in Figure 1. It can be seen that after iteration 4, further iterations do not lead to significant improvements. Also, the algorithm be-

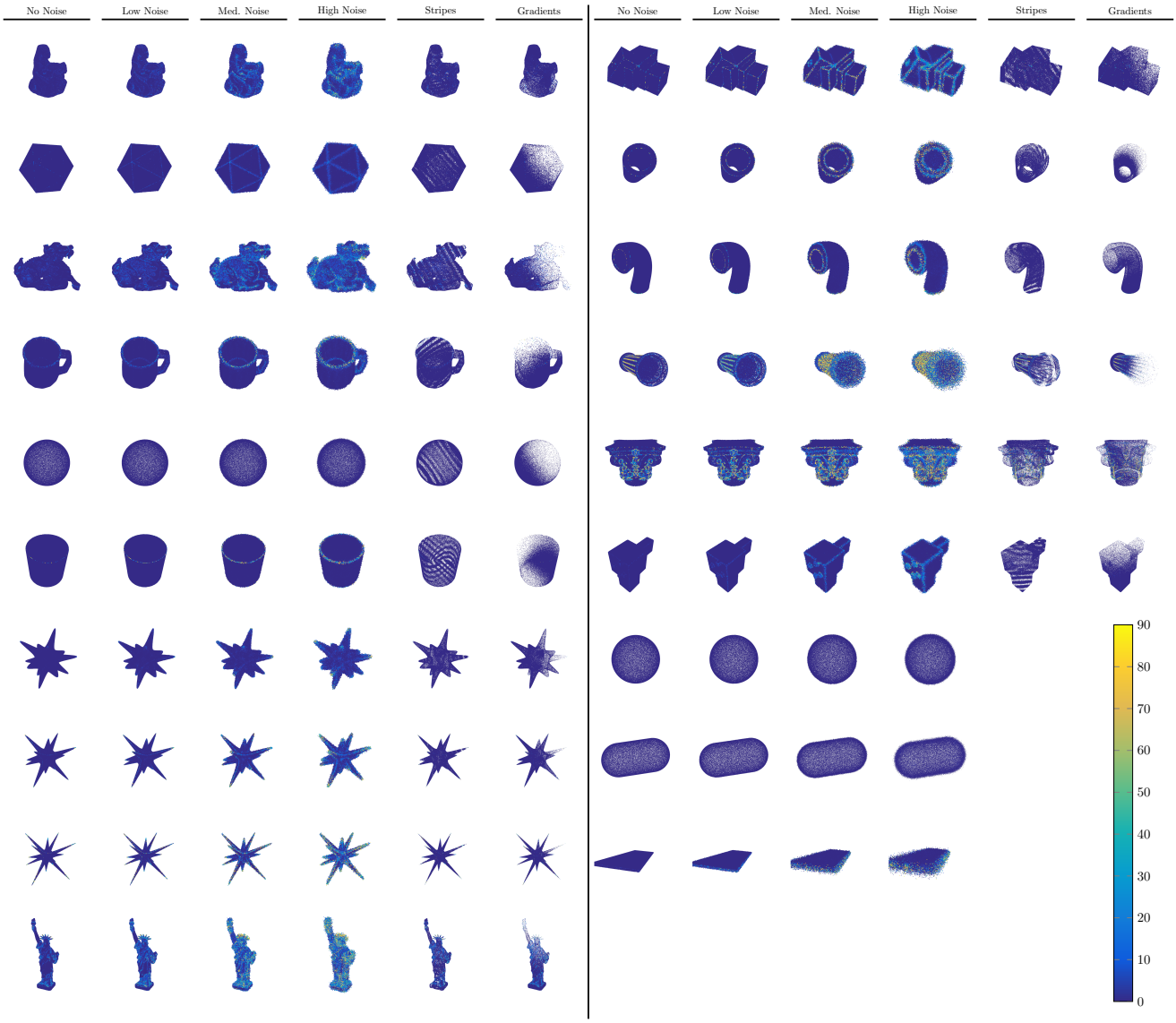

Figure 2: Qualitative results for all examples of the test set. Colors encode the RMSE in degree for each point. Best viewed in the digital version.

haves reasonable stable, not diverging immediately after we pass the iterations for which the network was trained. However, we observe a small drift in favor of low-noise datasets over the iterations. Errors for the test sets with no noise or variable density still decrease further while errors for data with higher noise levels slightly increase. Meanwhile, the average error stays nearly constant.

6. Transfer between neighborhood sizes

As stated in the main paper, the proposed algorithm generalizes reasonably well between neighborhood sizes, meaning that a model trained using neighborhood size k^{train} can be applied using a different neighborhood size k^{test} while producing good results. For verification, we report RSME errors for different combinations of k^{train} and k^{test} in Table 2. It can be seen that if the difference in neighborhood size is not too big, transferred models often only perform slightly worse than models trained directly for the appropriate k. However, transferring over very large difference like from 128 to 32 or the other way around, leads to a significant decrease in performance. The model trained on the balanced $k = 64$ performs very well on all other neighborhood sizes.

Additionally, Table 3 provides results for applying the model on even smaller neighborhood sizes, to evaluate the minimum k before the method breaks down. We found that

when using a $k^{\text{train}} <\approx 30$, the training becomes unstable, which is why we transfer the model from $k^{\text{train}} = 32$ to smaller $k^{\text{test}} = 32$. Results show that the algorithm provides good results for noise-free data down to $k = 4$. For noisy data, the approach breaks down quite fast when lowering k, as expected: At least $k = 24$ is required to provide reliable results. For lower k, the results approach the accuracy of random normals.

7. Further qualitative results

Last, we provide qualitative results for the whole PCPNet test set in Figure 2. For point clouds with varying density, the point size is reduced in order to better visualize the densities. Similar to examples shown in the paper, we can see that the method produces very sharp normal vectors, which usually resemble the plane normal of one of the plausible planes in the neighborhood. The abstract objects are good examples to show equivariance, as all edges show similar errors, independent of orientation. Sometimes, points are assigned to a false plane, leading to high error normal vectors. Compared to other approaches, we do not observe heavy smoothing around edges.

References

[1] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. *CoRR*, abs/1903.02428, 2019. 1

[2] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems 30*, pages 5099–5108. 2017. 1