Neighborhood Normalization for Robust Geometric Feature Learning – Supplementary Material

Xingtong Liu *[1], Benjamin D. Killeen *[1], Ayushi Sinha[1], Masaru Ishii[2], Gregory D. Hager[1], Russell H. Taylor[1], and Mathias Unberath[1]

[1]Johns Hopkins University
[2]Johns Hopkins Medical Institutions
{xingtongliu, unberath}@jhu.edu

1. Transposed NHN-Conv and B-NHN-Conv

Since it is not straightforward to describe transposed sparse 3D convolution in mathematical terms, we described it in words here instead. For equations in Sec. 3.1 of the main content, $\sum_{\boldsymbol{v}\in\mathcal{N}(\boldsymbol{u})}$ indicates a generalized sparse convolution operation in actual implementation. We used the *MinkowskiConvolutionFunction* in the python package *Minkowski Engine* [1] for this purpose. To implement a transposed version of the NHN-Conv and B-NHN-Conv, we simply replaced all *MinkowskiConvolutionFunction* with *MinkowskiConvolutionTransposeFunction*.

2. Architectures of comparison methods

2.1. MinkowskiNet with standalone normalization

The architecture is shown in Fig. 1. In the main content, we evaluated this architecture with normalization BatchNorm [4], InstanceNorm [11], and Batch-Instance Norm [7]. These are abbreviated as Mink.+BN, Mink.+IN, and Mink.+BIN in the main content. Mink.+BN, Mink.+IN, and Mink.+BIN all have around 8.80 million learnable parameters. In addition, Mink.+NHN and Mink.+B-NHN also have around 8.80 million learnable parameters.

2.2. FCGF

The architecture is shown in Fig. 2. Please find the mathematical definition of the 3DConv layer in the main content. Tr-3DConv is simply a transposed version of 3DConv. All numbers mean the same as the ones in the Network Architecture section of the main content. For 3DConv and Tr-3DConv, the three numbers mean kernel size along one spatial dimension, stride size, and output channel size. The number in BatchNorm and ResBlock represents the output channel size. The total number of learnable parameters is 8.76 million.

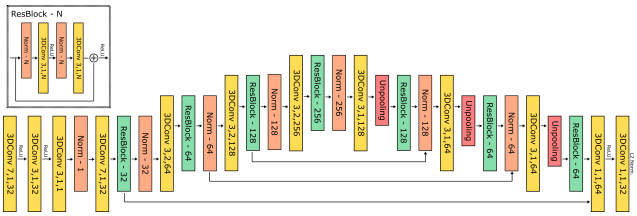

Figure 1. **Network architecture for MinkowskiNet (Mink.) with standalone normalization.** Norm can be any choice of standalone normalization. The number after Norm is the output channel size of this module. For the transposed version of this architecture that was used in the standard 3DMatch benchmark, we simply replaced the combination of the 3DConv and Unpooling with a transposed 3DConv with stride size 2. Note all skipping connections in this supplementary material is concatenation-style.

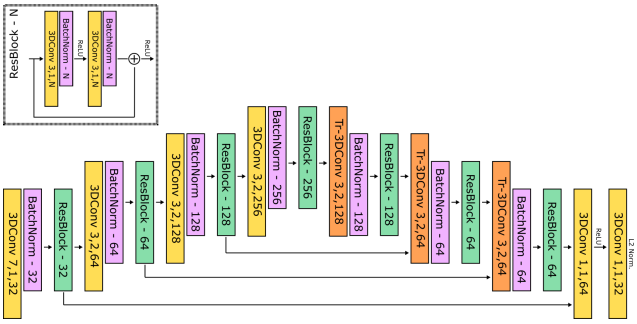

Figure 2. **Network architecture for FCGF [2].** Note that the architecture used in the actual state-of-the-art model in [2] is different from the one they have in the paper.

2.3. KPConv

The architecture is shown in Fig. 3. We changed the hyperparameter setting in the original work [10] for the

*These authors contributed equally to this work

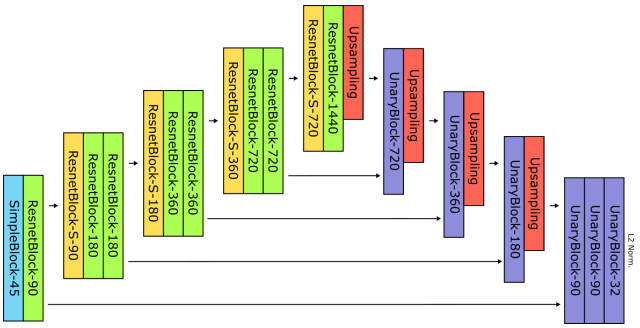

Figure 3. **Network architecture for KPConv [2].** Note we modified the original architecture for the task of 3D descriptor learning. The number besides the module name is the size of the output channel. Please refer to github repo https://github.com/HuguesTHOMAS/KPConv-PyTorch for the implementation details of all the modules in the figure. In the figure, SimpleBlock stands for the *SimpleBlock* module; ResnetBlock stands for the *ResnetBottleneckBlock* module; ResnetBlock-S stands for the *ResnetBottleneckBlock* module with striding enabled; Upsampling stands for the *NearestUpsampleBlock* module; UnaryBlock stands for the *UnaryBlock* module.

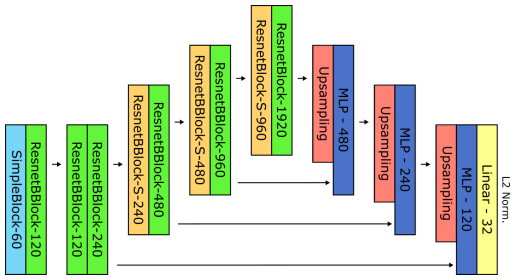

Figure 4. **Network architecture for PPNet [5].** Note we modified the original architecture for the task of 3D descriptor learning. The number besides the module name is the size of the output channel. We use the provided PPNet modules in the *PyTorch Points 3D* python package. In the figure, SimpleBlock stands for the *SimpleBlock* module; ResnetBBlock stands for the *ResnetBBlock* module; ResnetBBlock-S stands for the *ResnetBBlock* module with striding enabled. The combination of Upsamling and MLP modules stands for the *FPModule_PD* module. The Linear module at the end is a simple linear transform.

3DMatch dataset and the task of 3D descriptor learning. First, all the parameters are kept the same. The number of kernel points per filter is 15. The first subsampling grid size is set to 5 cm for a fair comparison with other methods in the 3DMatch benchmark. The first radius, *i.e.* number of grid cells, of convolution is 2.5. The radius of the area under influence for each kernel point is 1.2 grid cell. The type of KPConv influence is linear. The aggregation mode is summing in the standard benchmark and averaging in the resolution-mismatch one. The centered 3D spatial locations of all points are used for neighbor searching and downsampling inside the architecture. For what is changed, the channel size of the input feature is 1. The input features are all constant one. The channel dimension of the filter base is 90. The training setting, such as batch size, optimizer, and loss function, etc, is the same as the Mink. architecture described in the main content. The total number of learnable parameters is 9.08 million.

2.4. PPNet

The architecture is shown in Fig. 4. Some hyperparameter settings and the architecture is changed, compared with the original work [5]. The number of input feature is 3, which are all constant number one. The input grid size is set to 5 cm for a fair comparison. After each downsampling layer, the grid size is multiplied by 2. The channel dimension of the filter base is 60. The maximum number of neighbors is set to 27. The position embedding type is "XYZ" and the reduction type for local aggregation is averaging. The upsampling modules are the nearest upsampling. The total number of learnable parameters is 9.07 million.

2.5. PointNet++

The architecture is shown in Fig. 5. The input vertex features are a concatenation of centered point XYZ location and constant one. In MSGD, as opposed to the original design where the point cloud is downsampled to a fixed number, we use a fixed ratio of the points instead to account for the varying sample size. The downsample ratios for the four MSGD modules are 1.0, 0.25, 0.25, and 0.25. For all MSGD and GDBM modules, an additional 3 channels of point locations are concatenated with the feature map. The maximum number of neighbors is set to 27. The initial neighborhood radius is 12.5 cm. The radius is multiplied by 2 or divided by 2 whenever the point cloud is downsampled or upsampled, respectively. The total number of learnable parameters is 8.93 million.

2.6. DCM-Net

Because the form of input data in the experiments is a point cloud, the architecture in the original work [9] that uses only K-nearest neighbors for message propagation is used. The input vertex features point locations. The filters of the encoder part are [16, 96, 256, 384] with the number of propagation steps per graph layer as 4. The filters of the decoder part are the same as the encoder part, which is the original design in [9]. The pooling and aggregation modes are set to "max" and "mean", respectively. The channel size of the output feature description is 32, the same as all other comparison methods. The total number of learnable parameters is 7.29 million.

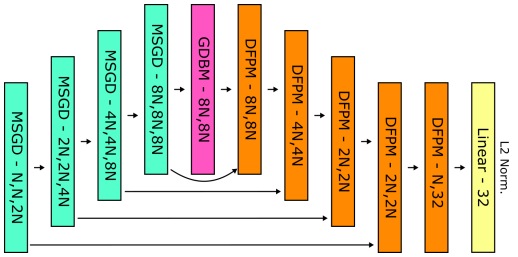

Figure 5. **Network architecture for PointNet++ [8]** Note we modified the original architecture for the task of 3D descriptor learning. We use the provided PointNet2 modules in the *PyTorch Points 3D* python package. In the figure, MSGD stands for the *PointNetMSGDown* module; GDBM stands for the *GlobalDenseBaseModule* module; DFPM stands for the *DenseFPModule* module. The Linear module at the end is a simple linear transform. MSGD consists of point downsampling and three Linear layers, and the three numbers after the module name are the output channel sizes of these Linear layers. The two numbers after GDBM and DFPM are the output channel sizes of the two Linear layers within the module. The number after Linear is the output channel size of the module. N, as the filter base, is set to 112.

3. Visualization of feature embeddings of resolution-mismatch sample pairs

The output feature embeddings from Mink.+B-NHN are visualized in Fig. 6, Fig. 7, and Fig. 8 for the 3DMatch [12], KITTI odometry [3], and the clinical datasets, respectively. The models of Mink.+B-NHN are trained with the resolution-mismatch settings described in the experiment section of the main content. UMAP [6] is used to reduce 32-dimension output feature descriptions to scalar values. These are then displayed with the JET colormap. To better visualize the embeddings of the 3DMatch and clinical datasets, we display the meshes instead of the input point clouds. The vertices of a displayed mesh get the embeddings of the spatially closest point in the corresponding input point cloud. All sample pairs displayed in these figures have a resolution mismatch. The mesh edges of the samples from 3DMatch and clinical datasets are displayed to make the resolution mismatch easier to observe. If the displayed colors of feature embeddings are similar, the L2 distances between the original feature embeddings are probably small.

References

[1] C. Choy, J. Gwak, and S. Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 1

[2] C. Choy, J. Park, and V. Koltun. Fully convolutional geometric features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8958–8966, 2019. 1, 2

[3] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. 3, 5

[4] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 1

[5] Z. Liu, H. Hu, Y. Cao, Z. Zhang, and X. Tong. A closer look at local aggregation operators in point cloud analysis. *ECCV*, 2020. 2

[6] L. McInnes, J. Healy, N. Saul, and L. Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. 3

[7] H. Nam and H.-E. Kim. Batch-instance normalization for adaptively style-invariant neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 2558–2567. Curran Associates, Inc., 2018. 1

[8] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5099–5108. Curran Associates, Inc., 2017. 3

[9] J. Schult, F. Engelmann, T. Kontogianni, and B. Leibe. Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3d meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 2

[10] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 1

[11] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. 1

[12] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1802–1811, 2017. 3, 4

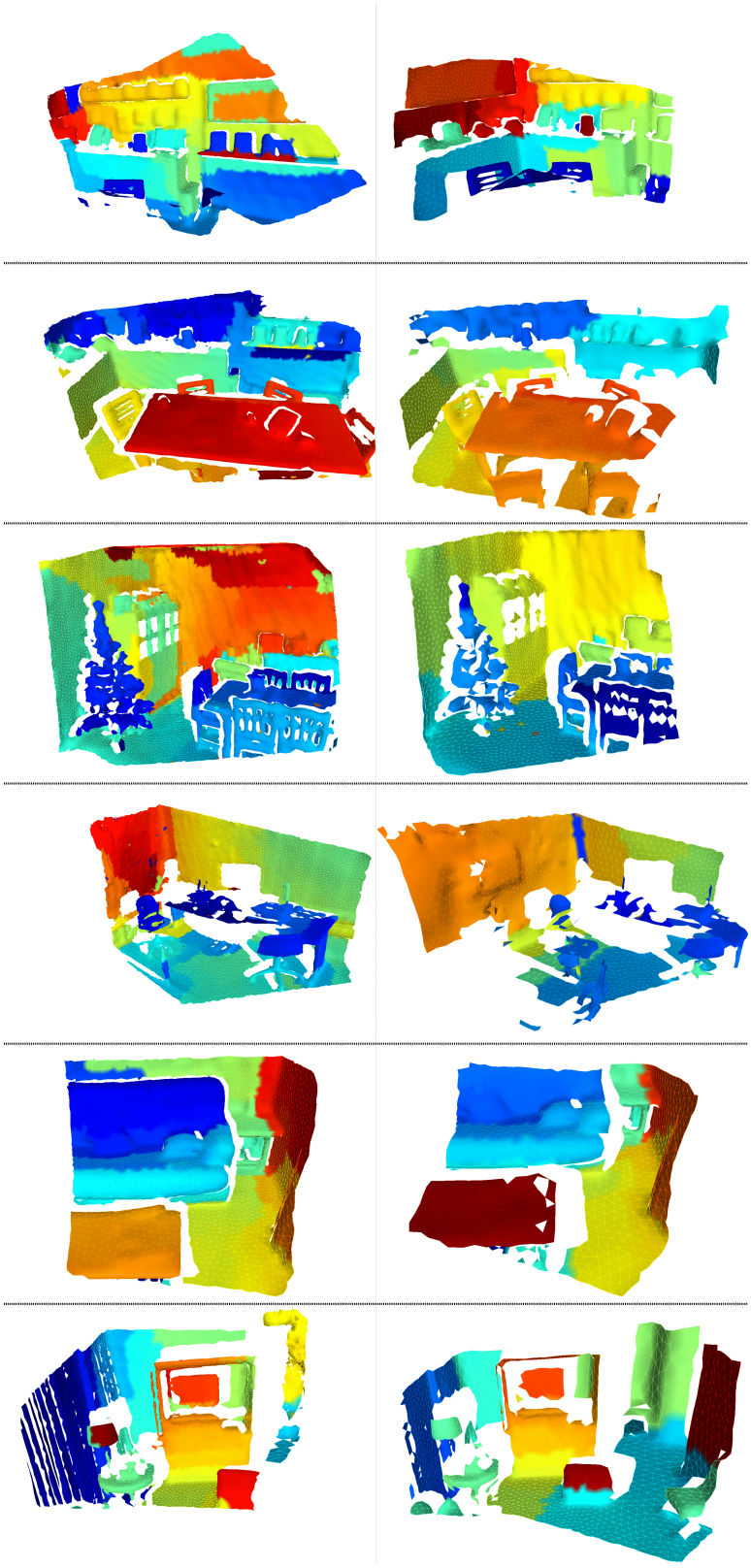

Figure 6. **Visualization of output feature embeddings for the 3DMatch dataset [12].** Matching colors indicate closely aligned feature representations; i.e. it is desirable for the same objects to have similar colors in each sample pair. The 1st and 2nd columns form sample pairs. The procedures taken for visualization are described in Sec. 3.

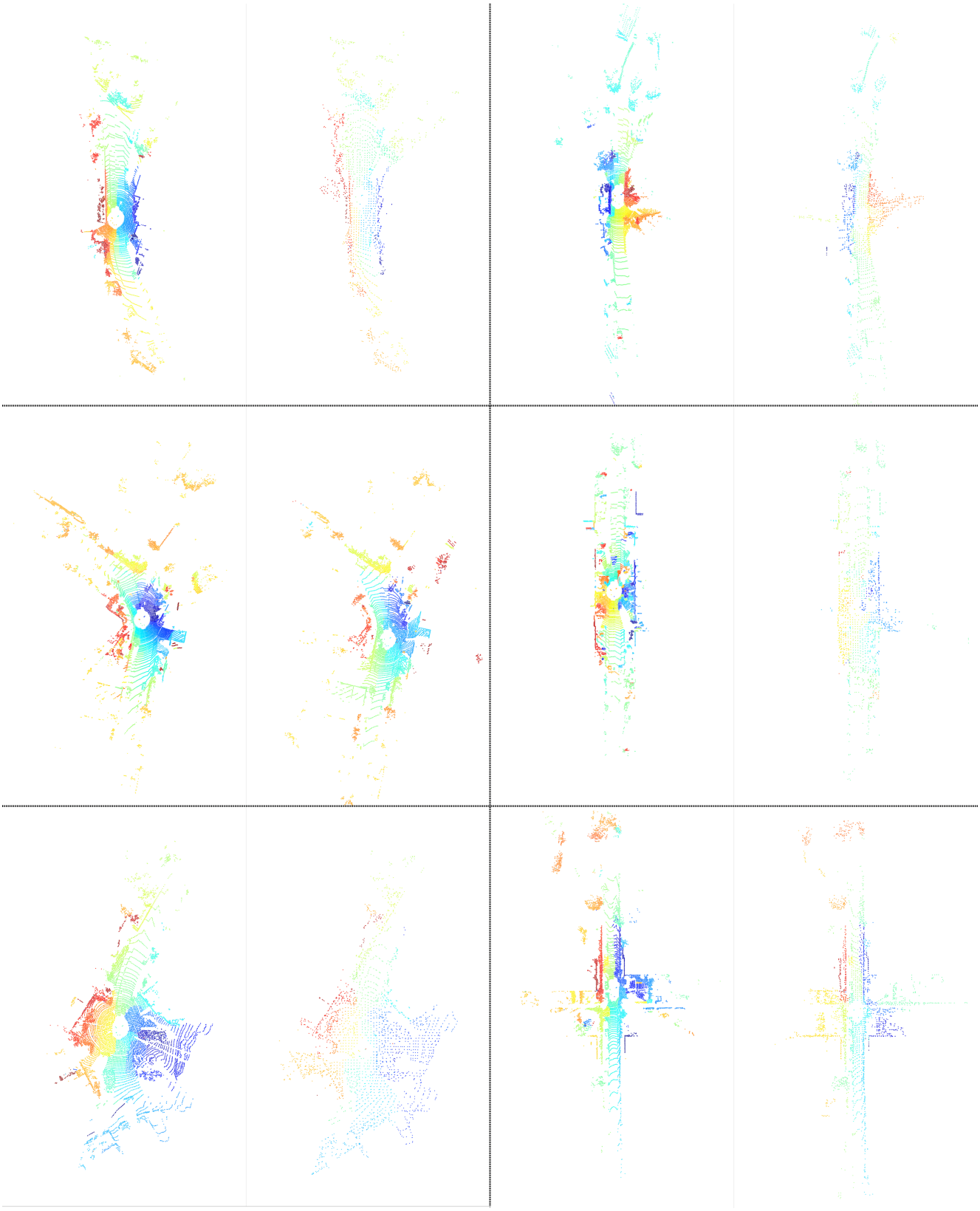

Figure 7. **Visualization of output feature embeddings for the KITTI dataset [3].** Matching colors indicate closely aligned feature representations. The 1st and 2nd columns form sample pairs, the same with the 3rd and 4th columns. The procedures taken for visualization are described in Sec. 3.

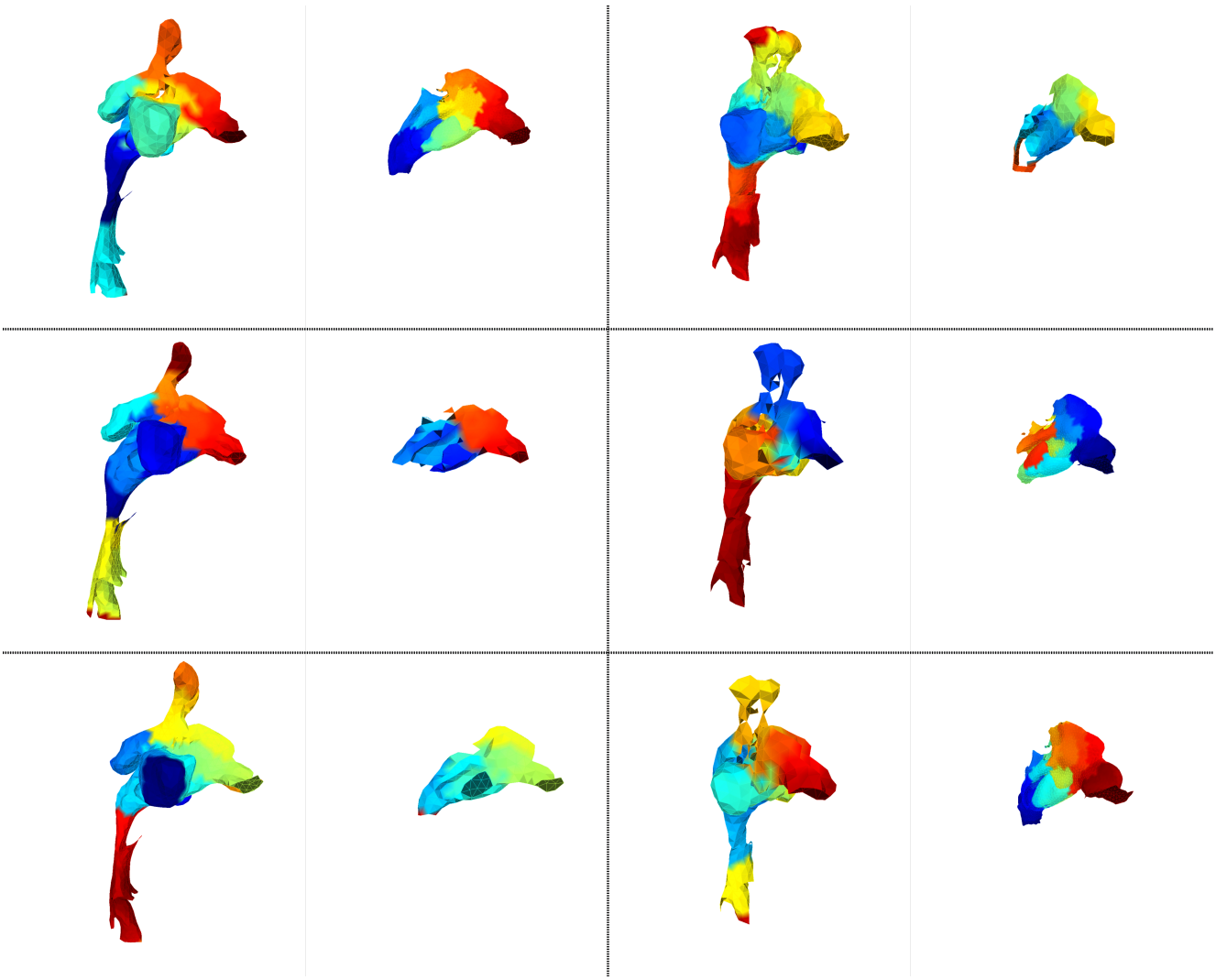

Figure 8. **Visualization of output feature embeddings for the clinical dataset of nasal cavities.** Matching colors indicate closely aligned feature representations. The 1st and 2nd columns form sample pairs, the same with the 3rd and 4th columns. The 1st and 3rd columns display the entire nasal cavity, while the 2nd and 4th columns display the nasal passage. The procedures taken for visualization are described in Sec. 3.