Supplementary Materials for Fully Convolutional Geometric Features

1. Network Architecture

We use U-networks with skip connections for feature extraction. The first half of the network consists of strided convolutions that downsample the points and increase the receptive field size exponentially; and the latter half of the network consists of transposed convolutions that upsample the points and concatenate the generated features with the skipped features with the same stride from the lower layers. The general form of the network can be succinctly represented as

$$f = f_1 \circ (\mathbf{1}, f_2' \circ (\mathbf{1}, f_3' \circ (\mathbf{1}, f_4) \circ f_3) \circ f_2) \circ f_5 \quad (1)$$

where f_i is short for $f_i(\cdot; w_i)$ which is a block of operations with parameter w_i. For the base architecture, we use one strided convolution followed by a batch normalization and a ReLU. f_i' is the same block with a transposed convolution. $\mathbf{1}$, $(\cdot, \cdot)$ are identity and concatenation respectively. For strided and transposed convolutions, we use stride=2 and create downsampled/upsampled sparse coordinates that have half/double the resolution of the previous sparse coordinates. We visualize the base network and a residual variant in Fig. 1.

2. Network Analysis

In this section, we present some directions in network architectural search that might provide useful guidelines in designing better networks for geometric features. We used the validation set for all experiments.

Kernel size. First, we found that the kernel size of the first convolution layer affects the performance significantly. We conjecture that that low-level geometries can be extracted more accurately with a larger kernel. For instance, if the input point cloud is noisy and the density varies drastically, the normal direction of a 3D patch gets more accurate if we extract the normal from a larger 3D patch.

Number of strided convolutions. Second, using more or fewer strided convolutions to increase or decrease the receptive field size do not increase the performance. We created several variants of the base network and replaced f_4 with $(\mathbf{1}, f_4 \circ (1, f_5))$ or removed the f_4 all together. However, these changes lead to worse performance on the 5cm voxel grid. This is probably due to the fact that geometric features could benefit from a larger context, but would lose specificity if the receptive field size gets too large.

Transposed convolution channels. Third, the number of channels in the last half of the network affects the performance greatly as well. We progressively increase the output channel sizes of the transposed convolutions and named them A, B, C, and D with output channels (64, 64, 32), (64, 64, 64), (128, 64, 32), and (128, 64, 64) respectively. The network variant C outperformed all the other networks. One outlier is the network D, which has more channels, but it does not lead to better performance. We conjecture that compressing the right information early is important as the fully-convolutional geometric features is very compact (32-dimensional).

Getting Subvoxel Accuracy Noe that our networks achieve sub-voxel level accuracy for registration even with voxel downsampling on all our experiments. This is due to the fact that we use voxel downsampling which returns continuous coordinates.

3. Hash-based Filtering Algorithm

One of the most time-consuming parts in the hardest-contrastive loss and the hardest-triplet loss is computing $I(i, j_i, d_t)$, the indicator function that filters out false hard negatives. In this section, we present an efficient hash-based hard contrastive algorithm for fully-convolutional feature learning. First, we create a matrix P that contains the indices of positive pairs (i, j) as well as additional matrix P_{d_t} that contains all pairs of indices that fall within a certain distance threshold d_t. Next, we find the hardest negatives for each positive pair and filter out the hardest negatives that fall within the vicinity of positive pairs using hash keys (Alg. 1).

The hash $\setminus$ can be implemented efficiently using sorted lists. The Alg. 1 is efficient if the distance threshold d_t is small. If the distance threshold d_t is large, use the coordinates of the associated features instead.

3.1. On-the-fly data augmentation

We apply various data augmentation on the data while loading fragment pairs. We apply random 360-degree rotation along an arbitrary vector to both fragments. Also, we normalize all inputs and add Gaussian noise on all chan-

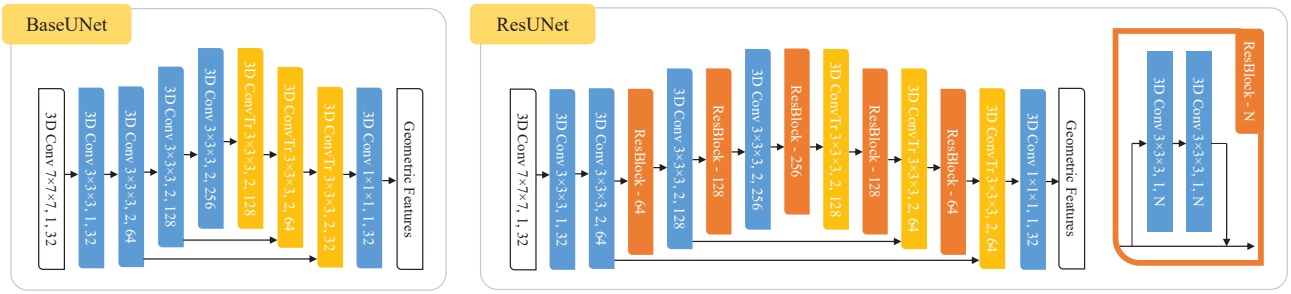

Figure 1: Base network structures used for analysis. The white blocks indicate input and output layers. Each block has three numbers: the kernel size, stride and channel size. All convolutions except the last layer have a batch normalization followed by a nonlinearity (ReLU).

Algorithm 1 Hardest Negative Mining with Hashing

Inputs: $P_{d_t} \in \mathbb{Z}_+^{N_P \times 2}$, $F_1 \in \mathbb{R}^{N_1 \times D}$, $F_2 \in \mathbb{R}^{N_2 \times D}$
Sample $\{\mathbf{S}_1 \subseteq \{1, ..., N_1\} | \text{card}(\mathbf{S}_1) = n\}$
Sample $\{\mathbf{S}_2 \subseteq \{1, ..., N_2\} | \text{card}(\mathbf{S}_2) = n\}$
$J_i = \text{argmin}_{j \in S_2} D(F_1[i], F_2[j])$
$I_j = \text{argmin}_{i \in S_1} D(F_2[j], F_1[i])$
$P_{\text{neg},1} = [i, J_i]_{\{i=1..N_1\}}$, $P_{\text{neg},2} = [I_j, j_i]_{\{j=1..N_2\}}$
$\mathbf{p} = \text{hash}(P_{d_t})$, $\mathbf{n}_1 = \text{hash}(P_{\text{neg},1})$, $\mathbf{n}_2 = \text{hash}(P_{\text{neg},2})$
$\mathbf{k}_1 = \text{index}(\mathbf{n}_1 \setminus \mathbf{p} \text{ from } \mathbf{n})$, $\mathbf{k}_2 = \text{index}(\mathbf{n}_2 \setminus \mathbf{p} \text{ from } \mathbf{n})$
return P_{d_t}, $P_{\text{neg},1}[\mathbf{k}_1]$, $P_{\text{neg},2}[\mathbf{k}_2]$

nels. Finally, we find all nearest neighbors within 1.5 times the voxel size across fragments with KDTree and generate positive pairs $\mathbf{P}$. The final on-the-fly data augmentation algorithm is on Alg. 2. The (X'_i, F'_i) forms a sparse tensor (coordinates and features) which is an input to our network. The KDTree function returns all pair of indices that are within a certain distance, which is used as the positive pairs $\mathbf{P}$ and $\text{A}(\cdot)$ is the data augmentation function. Note that we voxel-downsample point clouds after rotation.

Algorithm 2 On-the-fly Rotation Augmentation and Positive Pair

Point Clouds X_1, X_2 with ground truth $T_{\text{gt}} \in \text{SE}(3)$
Sample $T_1, T_2 \in \text{SO}(3)$
$X_1 \leftarrow T_1 T_{\text{gt}} X_1$, $X_2 \leftarrow T_2 X_2$
$X'_1, F'_1 \leftarrow \text{A}(\text{VoxelDownsample}(X_1, F_1))$
$X'_2, F'_2 \leftarrow \text{A}(\text{VoxelDosnwample}(X_2, F_2))$
return (X'_1, F'_1), (X'_2, F'_2), $\text{KDTree}(X'_1, T_1 T_2^{-1} X'_2)$

4. 3DMatch Benchmark Results

We present the full 3D Match benchmark results on Tab. 1 and Tab. 2. Note that the FCGF consistently outperformed most of the categories.

5. Effect of Margin for Triplet Loss

We analyze FCGFs trained with the hardest-negative triplet loss with various margins and report the numbers on Tab. 3. As we increase the margin, the feature-match recall increases, but plateau quickly after 1. For all hardest-triplet losses, we used 1024 random triplets and 512 hardest-negative per scene (increases proportionally with the batch size).

References

[1] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPF-FoldNet: Unsupervised learning of rotation invariant 3d local descriptors. In *ECCV*, 2018. 3

[2] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPFNet: Global context aware local features for robust 3d point matching. In *CVPR*, 2018. 3

[3] Haowen Deng, Tolga Birdal, and Slobodan Ilic. 3D local features for direct pairwise registration. In *CVPR*, 2019. 3

[4] Zan Gojcic, Caifa Zhou, Jan Dirk Wegner, and Wieser Andreas. The perfect match: 3D point cloud matching with smoothed densities. In *CVPR*, 2019. 3

[5] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *PAMI*, 21(5), 1999. 3

[6] Marc Khoury, Qian-Yi Zhou, and Vladlen Koltun. Learning compact geometric features. In *ICCV*, 2017. 3

[7] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 3

[8] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3D registration. In *ICRA*, 2009. 3

[9] S. Salti, F. Tombari, and L. di Stefano. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125, 2014. 3

[10] F. Tombari, S. Salti, and L. Di Stefano. Unique shape context for 3d data description. In *ACM Workshop on 3D Object Retrieval*, 2010. 3

	Spin [5]	SHOT [9]	FPFH [8]	USC [10]	PointNet [7]	CGF [6]	3DMatch [12]	Folding [11]	PPFNet [2]	PPF-Fold [1]	Direct [3]	Capsule [13]	PerfectMatch [4]	FCGF (2.5cm)
Kitchen	0.195	0.178	0.306	0.557	0.712	0.605	0.575	0.595	0.897	0.787	0.796	0.850	0.970	**0.986**
Home 1	0.397	0.373	0.583	0.321	0.551	0.712	0.737	0.718	0.558	0.763	0.808	0.853	0.955	**0.962**
Home 2	0.365	0.337	0.466	0.308	0.539	0.572	0.707	0.606	0.591	0.615	0.697	0.769	0.894	**0.933**
Hotel 1	0.181	0.208	0.261	0.535	0.407	0.572	0.571	0.655	0.580	0.681	0.726	0.814	0.965	**0.978**
Hotel 2	0.203	0.221	0.327	0.192	0.289	0.538	0.442	0.423	0.577	0.712	0.673	0.760	0.933	**0.942**
Hotel 3	0.316	0.389	0.500	0.315	0.333	0.833	0.630	0.611	0.611	0.944	0.944	0.926	**0.982**	**0.982**
Study	0.056	0.073	0.154	0.507	0.432	0.387	0.562	0.712	0.534	0.620	0.699	0.760	**0.935**	**0.935**
Lab	0.105	0.130	0.273	0.468	0.507	0.455	0.546	0.584	0.636	0.623	0.623	0.727	**0.935**	0.896
Average	0.227	0.238	0.359	0.400	0.471	0.585	0.596	0.613	0.623	0.718	0.746	0.807	0.947	**0.952**
STD	0.1143	0.1087	0.1344	0.1246	0.1265	0.140	0.088	0.087	0.108	0.105	0.094	0.062	0.027	0.029
Feat. Dim.	153	352	33	1980	256	**32**	512	512	64	512	512	512	**32**	**32**
Time (ms)	0.1331	0.2785	0.0325	3.7115	0.1712	1.4627	3.2136	0.3092	2.2573	0.7938	0.794	1.208	5.515	0.009158

Table 1: Feature-match recall at $\tau_1 = 0.1$, $\tau_2 = 0.05$ on the 3DMatch dataset [12].

	Spin [5]	SHOT [9]	FPFH [8]	3DMatch [12]	CGF [6]	PPFNet [2]	Folding [11]	PPF-Fold [1]	PerfectMatch [4]	FCGF (2.5cm)
Kitchen	0.1779	0.1779	0.2905	0.0040	0.4466	0.0020	0.0178	0.7885	0.972	**0.9783**
Home 1	0.4487	0.3526	0.5897	0.0128	0.6667	0.0000	0.0321	0.7821	0.962	**0.9744**
Home 2	0.3413	0.3365	0.4712	0.0337	0.5288	0.0144	0.0337	0.6442	0.909	**0.9183**
Hotel 1	0.1814	0.2168	0.3009	0.0044	0.4425	0.0044	0.0133	0.6770	0.965	**0.9735**
Hotel 2	0.1731	0.2404	0.2981	0.0000	0.4423	0.0000	0.0096	0.6923	0.923	**0.9712**
Hotel 3	0.3148	0.3333	0.5185	0.0096	0.6296	0.0000	0.0370	0.9630	**0.982**	**0.9815**
Study	0.0582	0.0822	0.1575	0.0000	0.4178	0.0000	0.0171	0.6267	**0.945**	**0.9452**
MIT Lab	0.1169	0.1299	0.2857	0.0260	0.4156	0.0000	0.0260	0.6753	**0.935**	0.8831
Average	0.2265	0.2337	0.3640	0.0113	0.4987	0.0026	0.0233	0.7311	0.949	**0.9532**
STD	0.1214	0.0947	0.1364	0.0116	0.0927	0.0047	0.0096	0.1035	0.0238	0.0332

Table 2: Feature-match recall on the augmented 3DMatch benchmark [1]. Features are extracted on the transformed point clouds.

margin	Feature Match Recall	STD
0.1	0.8635	0.0499
0.2	0.8963	0.0438
0.4	0.9008	0.0485
1	0.9029	0.0511
2	0.9070	0.0384

Table 3: Feature match recall of FCGFs (5cm voxel downsampling) on the 3DMatch benchmark with various margin m with 512 hardest-negatives and 1024 random triplets.

[11] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. FoldingNet: Interpretable unsupervised learning on 3d point clouds. In *CVPR*, 2017. 3

[12] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3Dmatch: Learning the matching of local 3D geometry in range scans. In *CVPR*, 2017. 3

[13] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3D point capsule networks. In *CVPR*, 2019. 3