Chapter 6

Geometric Features

6.1 Introduction

Finding geometric correspondences is a key step in many 3D registration, tracking, and scene flow estimation pipelines. A substantial body of work has thus focused on designing 3D features that can capture discriminative local geometry for correspondence establishment [16, 29, 26, 24, 23, 36, 17, 7, 6].

Learning-based 3D features have recently gained popularity due to their robustness and superior performance. Existing learning-based features rely on low-level geometric characteristics as input: e.g., angular deviation [7, 6, 26, 29, 23], point distributions [17, 16, 21], or volumetric distance functions [36, 11]. Then, a 3D patch is extracted at each point of interest and mapped to a low-dimensional feature space through a multi-layer perceptron or 3D convolutions. This process is computationally expensive and features are extracted only at downsampled interest points, thus lowering the spatial resolution for subsequent registration steps.

Such patch-based processing is inefficient because intermediate network activations are not reused across adjacent patches. If we use a 2D analogy, extracting 3D patches for feature learning is akin to extracting small 2D patches around each pixel for semantic segmentation. Furthermore, current pipelines limit spatial context by focusing on patches with restricted spatial extent.

Instead, we could apply 3D convolutions on the entire input without cropping out sections by simply transforming convolutions to fully-convolutional counterparts. Similarly, we could convert all fully-connected layers in a multi-layer perceptron into a series of convolutional layers with kernel size $1 \times 1 \times 1$. This is known as fully-convolutional processing and has been widely used in image analysis [20, 35, 5, 3]. Fully-convolutional networks can capture broad context, and are faster and more memory-efficient than non-fully-convolutional counterparts since intermediate activations are reused across overlapping regions.

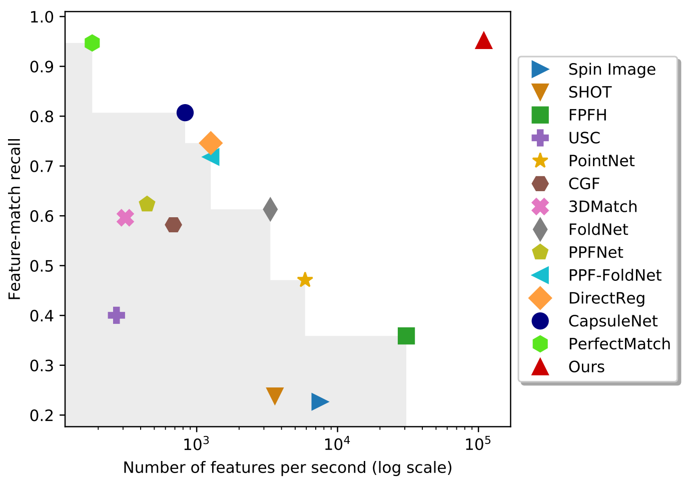

Figure 6.1: Feature-match recall [6, 7] and speed in log scale on the 3DMatch benchmark [36]. Our approach is the most accurate and the fastest. The gray region shows the Pareto frontier of the prior methods.

Despite these advantages, fully-convolutional networks have not been widely used for 3D geometric feature extraction due to the characteristics of 3D data. A standard input representation for convolutional networks on 3D data is a dense 4D tensor: three spatial dimensions and one feature dimension. This representation has a massive memory footprint, even though most 3D voxels are empty.

In this work, we adopt a sparse tensor representation, introduced in recent work on 3D semantic segmentation [12, 2]. We also introduce new losses for fully-convolutional metric learning, based on the observation that the characteristics of fully-convolutional features differ from traditional independent identically distributed (i.i.d.) features traditionally assumed in metric learning. Our approach does not require low-level preprocessing or 3D patches as input, and can rapidly generate high-resolution features with state-of-the-art discriminative power.

We validate fully-convolutional geometric features (FCGF) on both indoor and outdoor 3D datasets [36, 10]. Our approach achieves state-of-the-art performance on the 3DMatch benchmark [36], while being nine times faster than the fastest learning-based method and 290 times faster than the current state of the art (Fig. 6.1).

6.2 Related Work

Hand-crafted 3D features. Early work on 3D feature description focused on hand-crafting descriptors that can discriminatively characterize local geometry. Spin Images [16] use a projection of adjacent points onto the tangent plane. USC [29] uses covariance matrices of point pairs. SHOT [26] creates a 3D histogram of normal vectors. PFH [24] and FPFH [23] build an oriented histogram using pairwise geometric properties. Guo et al. [13] provide a comprehensive review of such hand-crafted descriptors.

Learning-based 3D features. More recently, attention has shifted to learning-based 3D features. Zeng et al. [36] use a siamese convolutional network to learn 3D patch descriptors. Khoury et al. [17] map 3D oriented histograms to a low-dimensional feature space using multi-layer perceptrons. Deng et al. [7, 6] adapt the PointNet architecture for geometric feature description. Yew and Lee [34] use a PointNet to extract features in outdoor scenes.

Our work addressed a number of limitations in the prior work. First, all prior approaches extract a small 3D patch or a set of points and map it to a low-dimensional space. This not only limits the receptive field of the network but is also computationally inefficient since all intermediate representations are computed separately even for overlapping 3D regions. Second, using expensive low-level geometric signatures as input can slow down feature computation. Lastly, limiting feature extraction to a subset of interest points results in lower spatial resolution for subsequent matching stages and can thus reduce registration accuracy.

Fully-convolutional networks. Fully-convolutional networks for images were introduced by Long et al. [20]. In 3D space, fully-convolutional networks have been used for semantic segmentation [4, 12, 22, 2]. The broad adoption of fully-convolutional networks can be attributed to three factors. First, fully-convolutional networks are efficient and fast because they share intermediate activations across neurons with overlapping receptive fields. Second, neurons in fully-convolutional networks can have bigger receptive fields because they are not constrained by operating on separately extracted and processed patches. Third, fully-convolutional networks produce dense output, which is well-suited for tasks that call for detailed characterization of scenes.

Deep metric learning. Deep metric learning combines deep networks and metric learning to produce compact embeddings. The contrastive loss formulates the objective in terms of pairwise constraints [14]. There has also been significant interest in higher-order loss terms, including triplet [32], quadruplet [18], and histogram losses [30]. Due to the polynomial growth in complexity that accompanies high-order losses, many recent papers focus on triplets with hard-negative mining within a batch. Lifted structure [28] and N-pair losses [27] proposed using a softmax for mining hard negatives within a batch.

In this work, we study fully-convolutional metric learning, where the basic assumption that features are independent and identically distributed (i.i.d.) within a batch no longer holds. To address this, we develop new losses for fully-convolutional feature learning and show that they are more effective than traditional ones.

6.3 Sparse Tensors and Convolutions

The 3D data of interest in this work consists of 3D scans of surfaces. In such datasets, most of the 3D space is empty. To handle this sparsity, we use sparse tensors: high-dimensional equivalents of sparse matrices. Mathematically, we can represent a sparse tensor for 3D data as a set of coordinates C and associated features F:

$$C = \begin{bmatrix} x_1 & y_1 & z_1 & b_1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & b_N \end{bmatrix}, \ F = \begin{bmatrix} \mathbf{f}_1^T \\ \vdots \\ \mathbf{f}_N^T \end{bmatrix} \tag{6.1}$$

where $x_i, y_i, z_i \in \mathbb{Z}$ is the i-th 3D coordinate and b_i is the batch index which provides an additional dimension for batch processing. $\mathbf{f}_i$ is the feature associated with the i-th coordinate.

Convolutions on sparse tensors (also known as sparse convolutions) require a somewhat different definition from conventional (dense) convolutions. In discrete dense 3D convolution, we extract input features and multiply with a dense kernel matrix. Denote a set of offsets in n-dimensional space by $\mathcal{V}^n(K)$, where K is the kernel size. For example, in 1D, $\mathcal{V}^1(3) = \{-1, 0, 1\}$. The dense convolution can be defined as in Eq. 6.2, where $W_\mathbf{i}$ denotes the kernel value at offset $\mathbf{i}$:

$$\mathbf{x}_\mathbf{u}^{\text{out}} = \sum_{\mathbf{i} \in \mathcal{V}^3(K)} W_\mathbf{i} \mathbf{x}_{\mathbf{u}+\mathbf{i}}^{\text{in}} \text{ for } \mathbf{u} \in \mathbb{Z}^3. \tag{6.2}$$

In contrast, a sparse tensor has a feature at location $\mathbf{u}$ only if the corresponding coordinates are present in the set C. Thus it suffices to evaluate the convolution terms only over a subset $\mathcal{N}^n(\mathbf{u}, K, C) = \{\mathbf{i} | \mathbf{i} \in \mathcal{V}^n(K), \mathbf{i} + \mathbf{u} \in C\}$. (I.e., the set of offsets $\mathbf{i}$ for which $\mathbf{i} + \mathbf{u}$ is present in the set of coordinates C.) If we make the sets of input and output coordinates different (C^{in} and C^{out}, respectively), we arrive at the generalized convolution [2], summarized in Eq. 6.3:

$$\mathbf{x'}_\mathbf{u}^{\text{out}} = \sum_{\mathbf{i} \in \mathcal{N}^3(\mathbf{u}, K, C^{\text{in}})} W_\mathbf{i} \mathbf{x'}_{\mathbf{u}+\mathbf{i}}^{\text{in}} \text{ for } \mathbf{u} \in C^{\text{out}}. \tag{6.3}$$

Sparse fully-convolutional features. Fully-convolutional networks consist purely of translation-invariant operations, such as convolutions and elementwise nonlinearities. Similarly, if we apply a sparse convolutional network to a sparse tensor, we get a sparse output tensor. We refer to the contents of this output tensor as fully-convolutional features. We use a UNet structure with skip connections and residual blocks to extract such sparse fully-convolutional features. Our network architecture is visualized in Fig. 6.2.

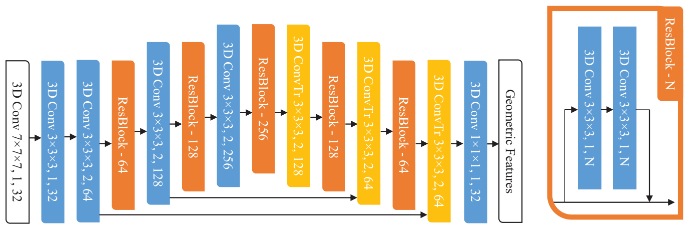

Figure 6.2: We use a ResUNet architecture. The white blocks indicate input and output layers. Each block is characterized by three parameters: kernel size, stride, and channel dimensionality. All convolutions except the last layer are accompanied by batch normalization followed by a nonlinearity (ReLU).

6.4 Fully-convolutional Metric Learning

In this section, we briefly go over a few standard metric learning losses and negative-mining techniques. Then, we characterize metric learning in the fully-convolutional setting and propose variants for fully-convolutional features that integrate negative-mining into the contrastive and triplet losses. We refer to these new losses as “hardest-contrastive” and “hardest-triplet”.

Metric learning begins with two constraints: similar features have to be close to each other – $D(\mathbf{f}_i, \mathbf{f}_j) \to 0 \quad \forall (i, j) \in \mathcal{P}$ – and dissimilar features have to be at least a margin away: $D(\mathbf{f}_i, \mathbf{f}_j) > m$ $\forall\ (i, j) \in \mathcal{N}$, where $D(\cdot, \cdot)$ is a distance measure. We square the violation and get a standard contrastive loss. Lin et al. [19] showed that the constraints for positive pairs could lead to overfitting and proposed a margin-based loss for positive pairs:

$$L(\mathbf{f}_i, \mathbf{f}_j) = I_{ij} \left[D(\mathbf{f}_i, \mathbf{f}_j) - m_p \right]_+^2 + \bar{I}_{ij} \left[m_n - D(\mathbf{f}_i, \mathbf{f}_j) \right]_+^2$$

where $I_{ij} = 1$ if $(i, j) \in \mathcal{P}$ and 0 otherwise, and $\bar{\cdot}$ is the NOT operator. m_p and m_n are margins for positive and negative pairs. Similarly, we can convert the ranking constraint $m + D(\mathbf{f}, \mathbf{f}_+) < D(\mathbf{f}, \mathbf{f}_-)$ into a triplet loss:

$$L(\mathbf{f}, \mathbf{f}_+, \mathbf{f}_-) = \left[m + D(\mathbf{f}, \mathbf{f}_+) - D(\mathbf{f}, \mathbf{f}_-) \right]_+^2 \tag{6.4}$$

For both contrastive and triplet losses, the sampling strategy affects the performance greatly as the decision boundary is defined by very few hardest negatives.

6.4.1 Characteristics of Fully-convolutional Features

Traditional metric learning assumes that the features are independent and identically distributed (i.i.d.) since a batch is constructed by random sampling [14, 32, 28, 27]. However, in fully-convolutional feature extraction, adjacent features are locally correlated. Thus, hard-negative mining

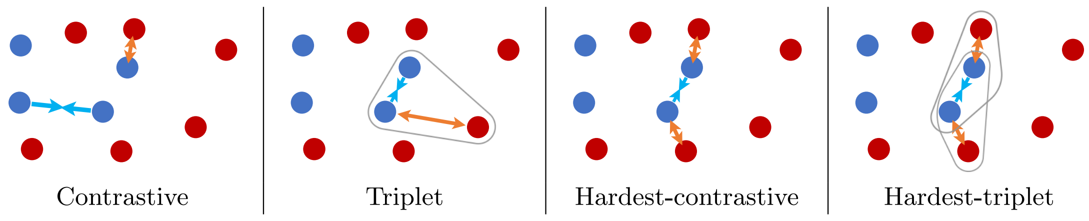

Figure 6.3: Sampling and negative-mining strategy for each method. Traditional contrastive and triplet losses use random sampling. Our hardest-contrastive and hardest-triplet losses use the hardest negatives.

could find features adjacent to anchors, and they are false negatives. Thus, filtering out the false negatives is crucial, and Choy et al. [3] used a distance threshold.

Also, the number of features used in the fully-convolutional setting is orders of magnitude larger than in standard metric learning algorithms [27, 28]. For instance, FCGF generates ~40k features for a pair of scans (this increases proportionally with the batch size) while a minibatch in traditional metric learning has around 1k features. Thus, it is not feasible to use all pairwise distances within a batch as in standard metric learning.

6.4.2 Hardest-contrastive and Hardest-triplet Losses

In this section, we propose metric learning losses for fully-convolutional feature learning. Like many algorithms in metric learning, we focus on efficient hard-negative mining. First, we sample anchor points and a set for mining per scene. Then, we mine the hardest negatives $\mathbf{f}_i^-, \mathbf{f}_j^-$ for both $\mathbf{f}_i$ and $\mathbf{f}_j$ in a positive pair $(\mathbf{f}_i, \mathbf{f}_j)$ (Fig. 6.3) and remove false negatives that fall within a certain radius from the corresponding anchor. Then, we use the pairwise loss for the mined quadruplet $(\mathbf{f}_i, \mathbf{f}_j, \mathbf{f}_i^-, \mathbf{f}_j^-)$ and form the fully-convolutional contrastive loss:

$$
\begin{aligned}
L_C = \sum_{(i,j)\in\mathcal{P}} \Bigg\{ & \left[D(\mathbf{f}_i, \mathbf{f}_j) - m_p\right]_+^2 / |\mathcal{P}| \\
& + \lambda_n I_i \left[m_n - \min_{k\in\mathcal{N}} D(\mathbf{f}_i, \mathbf{f}_k)\right]_+^2 / |\mathcal{P}_i| \\
& + \lambda_n I_j \left[m_n - \min_{k\in\mathcal{N}} D(\mathbf{f}_j, \mathbf{f}_k)\right]_+^2 / |\mathcal{P}_j| \Bigg\}
\end{aligned} \tag{6.5}
$$

where $\mathcal{P}$ is a set of all positive pairs in fully-convolutionally extracted features in a minibatch and $\mathcal{N}$ is a random subset of fully-convolutional features in a minibatch that will be used for negative mining. I_i is short for $I(i, k_i, d_t)$, which is an indicator function that returns 1 if the feature k_i is located outside a sphere with diameter d_t centered at feature i in the physical domain and 0 otherwise, where $k_i = \arg\min_{k\in\mathcal{N}} D(\mathbf{f}_i, \mathbf{f}_k)$. $|\mathcal{P}_i| = \sum_{(i,j)\in\mathcal{P}} I(i, k_i, d_t)$ is the number of valid mined negatives for the first item ($|\mathcal{P}_j|$ for the second item) in a pair. The indicator function removes the hardest negative loss if the hardest negative is physically close to the ground truth correspondence. The normalization term for negative pairs is simply averaging all valid negative pairs equally. λ_n is a weight for negative losses and we simply used 0.5 to weight positives and negatives equally. Similarly, we can form a triplet loss with hard negatives mined within a batch:

$$\begin{aligned} L_T = \frac{1}{Z} \sum_{(i,j)\in\mathcal{P}} \Bigg(& I(i,k_i) \left[m + D(\mathbf{f}_i, \mathbf{f}_j) - \min_{k\in\mathcal{N}} D(\mathbf{f}_i, \mathbf{f}_k) \right]_+ \\ & + I(j,k_j) \left[m + D(\mathbf{f}_i, \mathbf{f}_j) - \min_{k\in\mathcal{N}} D(\mathbf{f}_j, \mathbf{f}_k) \right]_+ \Bigg) \end{aligned} \tag{6.6}$$

where $Z = \sum_{(i,j)\in\mathcal{P}} (I(i,k_i) + I(j,k_j))$, a normalization constant. The above equation finds the hardest negatives for both $(i,j) \in \mathcal{P}$ (Fig. 6.3). Here $\mathcal{P}$ is the set of all positive pairs in the batch. Note that we followed Hermans et al. [15] and used non-squared loss to mitigate features from collapsing into a single point. Experimentally, we still observed that the fully-convolutional hardest triplet loss is prone to collapse (all features converge to a single point). Instead, we mix the above triplet loss with randomly sampled triplets to mitigate the collapse. Similar to Eq. 6.6, we weigh both randomly subsampled triplets and hard-negative triplets equally.

6.5 Implementation

We use the Minkowski Engine [2], an auto-differentiation library for sparse tenors, for sparse convolution and other essential layers. As the input to the network requires unique coordinates C and corresponding features F, we first downsample the input point cloud using a fast GPU-based voxel downsampling function. All these preprocessing steps can be parallelized in data-loading parallel processes and consume a fraction of the training time.

Hash-based negative filtering. One of the most time-consuming parts in both Eqs. 6.5 and 6.6 is computing $I(i, j_i, d_t)$, the indicator function that filters out false hard negatives. We use hash-based filtering to efficiently remove false negatives from the hard negative mining step to implement $I(i, j_i)$. First, we create a matrix P that contains the indices of positive pairs (i,j) as well as an 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 all positive pairs and filter out the hardest negatives that fall within the vicinity of positive pairs by comparing the hash keys. Filtering out hash keys can be implemented efficiently using two sets of sorted lists.

6.6 Experiments

We validate our fully-convolutional geometric features (FCGF) on both indoor and outdoor datasets. We show that FCGF outperform all state-of-the-art methods in both accuracy and speed, and analyze the proposed hardest-contrastive and hardest-triplet losses.

6.6.1 Datasets and Training

For indoor data, we use the standard 3D Match dataset [36]. For outdoor experiments, we use the KITTI odometry dataset [10]. We followed the official data split for 3D Match. For KITTI, we use the odometry training set because it provides GPS ground truth. This training set contains 11 sequences, which we split into train/val/test sets as follows: sequence 0 to 5 for training, sequence 6 to 7 for validation, and sequence 8 to 10 for testing. For all LIDAR scans, we used the first scan that is taken at least 10m apart within each sequence to create a pair. We found the GPS "ground truth" to be very noisy and used the Iterative Closest Point algorithm (ICP) to refine the alignment. If ICP fails or the number of overlapping voxels is less than 1k, we removed the pair from the dataset. This procedure yielded 1358 pairs for training, 180 for validation, and 555 for testing.

We train the networks for 100 epochs using Stochastic Gradient Descent starting with learning rate 0.1 with a Exponential learning rate schedule with $\gamma = 0.99$. We used batch size 4 for all experiments and analysis. We applied data augmentation including random scaling $\in [0.8, 1.2]$ to a pair, and different random rotation augmentation $\in [0^\circ, 360^\circ)$ along an arbitrary 3D direction for both scans in a pair. We found rotation augmentation to be a simple (SO(3) multiplication) and effective way to make FCGF invariant to relative camera pose change.

Since a sparse tensor is defined as a pair of coordinates and associated features, we tried to use a few different features such as color and normal for input sparse tensor features. However, the dataset was not diverse or large enough, even with data augmentation, to prevent the network from overfitting when color was provided in the input. Also, as FCGF was trained to capture the underlying geometry, using normal directions did not make a meaningful difference. In the end, we create an input sparse tensor with coordinates from a scan and 1-vectors as features for all experiments. However, we suspect that color could boost the performance if used with a large and diverse dataset.

6.6.2 Evaluation Metrics

For the 3D Match benchmark, we use two standard metrics to measure the quality of features under registration: feature-match recall and registration recall. For the outdoor dataset, we use the Relative Translation Error and the Relative Rotation Error.

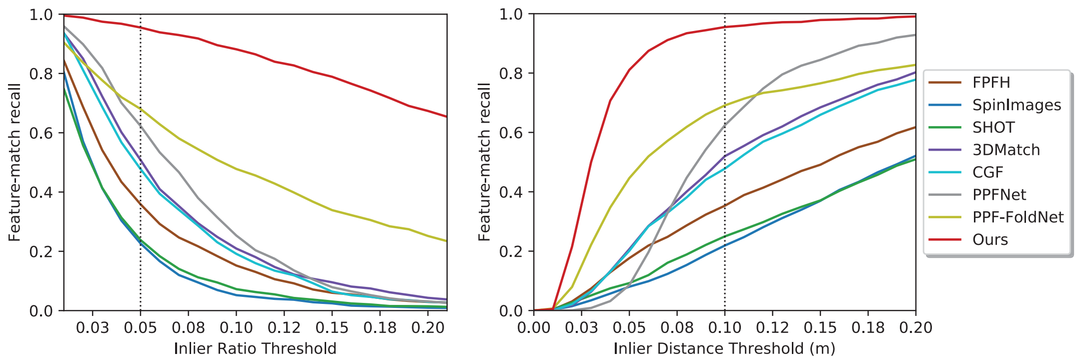

Figure 6.4: Feature-match recall with respect to inlier ratio threshold τ_2 (left) and inlier distance accuracy tolerance τ_1 (right). The vertical lines are $\tau_2 = 0.05$ (left) and $\tau_1 = 0.1$ (right), following [6, 7].

Feature-match recall. The feature-match recall [7] measures the percentage of fragment pairs that can recover the pose with high confidence. Mathematically, it is

$$R = \frac{1}{M}\sum_{s=1}^{M} \mathbb{1}\left(\left[\frac{1}{|\Omega_s|}\sum_{(i,j)\in\Omega_s}\mathbb{1}\left(\|\mathbf{T}^*\mathbf{x}_i - \mathbf{y}_j\| < \tau_1\right)\right] > \tau_2\right) \tag{6.7}$$

where M is the number of fragment pairs, Ω_s is a set of correspondences between a fragment pair s, $\mathbf{x}$ and $\mathbf{y}$ are 3D coordinates from the first and second fragment, and $\mathbf{T}^* \in \mathrm{SE}(3)$ is the ground-truth pose. $\tau_1 = 0.1m$ is the inlier distance threshold and $\tau_2 = 0.05$ or 5% is the inlier recall threshold, following [6, 7].

Registration recall. The feature-match recall measures the quality of feature under pairwise registration. However, it does not measure the quality of feature when used within a reconstruction system. Instead, the registration recall [1] takes a set of overlapping fragments with a ground-truth pose and measures how many overlapping fragments a matching algorithm can correctly recover. Specifically, the registration recall uses the following error metric between estimated fragments $\{i, j\}$, and corresponding pose estimation $\hat{\mathbf{T}}_{i,j}$ to define a true positive:

$$E_{RMSE} = \sqrt{\frac{1}{\Omega^*}\sum_{(\mathbf{x}^*,\mathbf{y}^*)\in\Omega^*}\|\hat{\mathbf{T}}_{i,j}\mathbf{x}^* - \mathbf{y}^*\|^2} \tag{6.8}$$

where Ω^* is a set of corresponding ground-truth pairs in fragments $\{i, j\}$, and $\mathbf{x}^*$ and $\mathbf{y}^*$ are the 3D coordinates of the ground-truth pair. For fragments $\{i, j\}$ with at least 30% overlap, the registration is evaluated as a correct pair if $E_{RMSE} < 0.2m$. As noted in several works [1, 7, 6, 17], recall is more important than precision since it is possible to improve precision with better pruning.

Relative translation and rotation error. The Relative Translation Error (RTE) and Relative Rotation Error (RRE) measure the registration errors of features used for RANSAC. Thus, they are indirect measures, but we follow Yew and Lee [34] for outdoor dataset evaluation. RTE and RRE are defined as RTE=$|\hat{T} - T^*|$ where $\hat{T}$ is the estimated translation after registration and RRE=$\arccos((\text{Tr}(\hat{R}^T R^*) - 1)/2)$ where $\hat{R}$ is the estimated rotation matrix and R^* is the ground-truth rotation matrix.

6.6.3 3D Match Benchmark

We compare FCGF with hand-crafted features and recent state-of-the-art methods on the 3DMatch benchmark [36] using feature-match recall and registration recall.

Tab. 6.1 lists the feature-match recall for all methods at $\tau_1 = 10cm$ and $\tau_2 = 0.05$ (following [6, 7]), the feature dimension, and the feature extraction time. FCGF outperforms all hand-crafted features and PointNet-based methods by a large margin and marginally outperforms a recent 3D-convolution-based method [11]. FCGF is the fastest feature extraction method and is 600 times faster than [11]. Please refer to Fig. 6.1 for visualization of the performance and speed of each method, and Sec. 6.6.7 for more details on timing. Note that FCGF is 32-dimensional while most state-of-the-art methods have higher dimensionality. Standard deviation (STD) of feature-match recall is computed across room types following [11]. Fig. 6.4 shows the sensitivity of feature-match recall to the inlier distance threshold τ_1 and inlier recall threshold τ_2.

Method	3DMatch		with Rot. Aug.		Feat.	Time
	FMR	STD	FMR	STD	Dim.	(ms)
Spin [16]	0.227	0.114	0.227	0.121	153	0.133
SHOT [26]	0.238	0.109	0.234	0.095	352	0.279
FPFH [23]	0.359	0.134	0.364	0.136	33	0.032
USC [29]	0.400	0.125	-	-	1980	3.712
PointNet [21]	0.471	0.127	-	-	256	0.171
CGF [17]	0.582	0.142	0.585	0.140	**32**	1.463
3DMatch [36]	0.596	0.088	0.011	0.012	512	3.210
Folding [33]	0.613	0.087	0.023	0.010	512	0.352
PPFNet [7]	0.623	0.108	0.003	0.005	64	2.257
PPF-Fold [6]	0.718	0.105	0.731	0.104	512	0.794
DirectReg [8]	0.746	0.094	-	-	512	0.794
CapsuleNet [37]	0.807	0.062	0.807	0.062	512	1.208
PerfectMatch [11]	0.947	0.027	0.949	0.024	**32**	5.515
Ours	**0.952**	0.029	**0.953**	0.033	**32**	**0.009**

Table 6.1: Feature-match recall at $\tau_1 = 0.1$, $\tau_2 = 0.05$ [6] on 3DMatch [33]. FMR and STD indicate the Feature Match Recall and its standard deviation. Feat. Dim. indicates feature dimensionality and Time is in milliseconds consumed per feature.

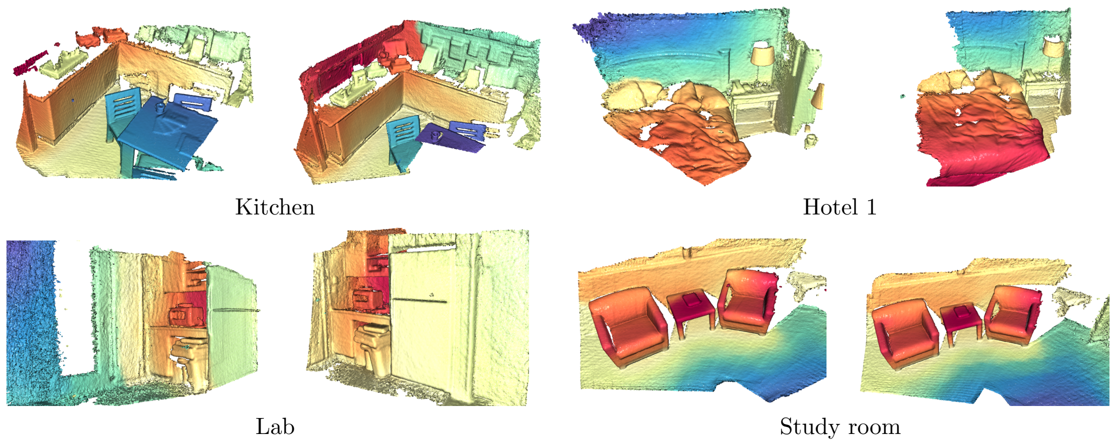

Figure 6.5: Color-coded features overlaid on selected fragment pairs. The 32-dimensional FCGF features for each pair of point clouds are mapped to a scalar space using t-SNE [31] and colorized with the Spectral color map.

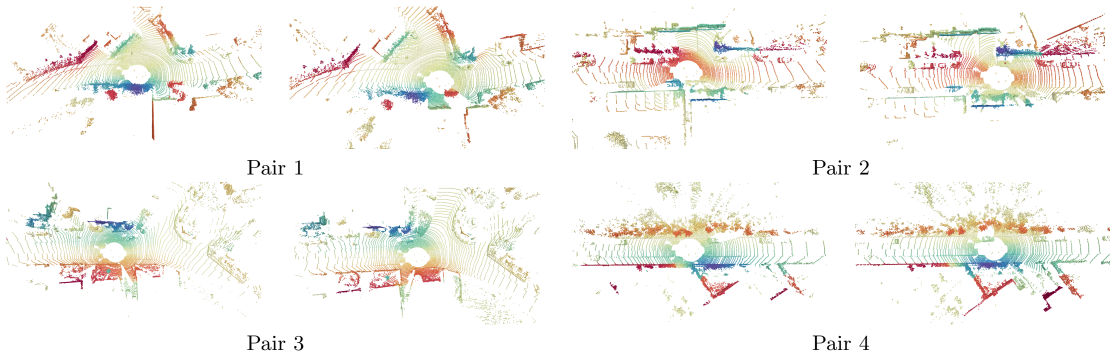

Figure 6.6: Color-coded FCGF features for pairs of KITTI LIDAR scans that are 10m apart. FCGF features from downsampled LIDAR scans are mapped to a scalar space using t-SNE [31] and colorized with the Spectral color map.

Overall, FCGF has better registration recall and feature-match recall across different scenes, a wide range of distance thresholds, and inlier recall thresholds. To visualize the general quality of FCGF, we use t-SNE [31] to project all FCGFs in a pair of scans to a color-coded one-dimensional space and visualize them in Fig. 6.5.

Rotation and translation invariance. One of the most important characteristics of good geometric features is rotation and translation invariance. Some of the baseline methods achieve this by aligning 3D patches along the normal direction of a patch. Instead, FCGF learns the rotation invariance through on-the-fly data augmentation (details in the supplement). Translation invariance is an inherent property of a sparse tensor as the translation does not affect convolutions. Tab. 6.1 shows the feature-match recall of FCGF and other features on the augmented 3DMatch dataset [6].

Feat. Dimensions	16	32	48	64
Feat. Match Recall (5cm)	0.9011	0.9242	0.9235	0.9343
STD (5cm)	0.0328	0.0439	0.0429	0.0365
Feat. Match Recall (2.5cm)	0.9412	0.9535	0.9489	OOM
STD (2.5cm)	0.0336	0.0334	0.0393	OOM

Table 6.2: Hardest-contrastive loss feature match recall with different feature dimensionality on 3DMatch. OOM denotes Out Of Memory under the same hyperparameters.

Hardest-Triple Num. HN / RT	Feature Match Recall	STD
1024 / 512	COLLAPSE	
768 / 512	COLLAPSE	
768 / 768	0.8866	0.0377
512 / 768	0.8935	0.0393
512 / 1024	0.9022	0.0399
128 / 2048	0.9087	0.0458
0 / 2048	0.7903	0.0494
Contrastive	0.7309	0.0245
Contrastive (norm.)	0.8493	0.0489
Triplet	0.7903	0.0494
Triplet (norm.)	0.6935	0.0446
Hardest-Contrastive	**0.9344**	0.0365

Table 6.3: Feature match recall of the hardest-triplet loss with various hardest-negative and random triplet ratios (Hardest-Negative triplets (HN) and Random Triplets (RT) per a pair of scans with 5cm voxel downsampling), contrastive, triplet, and hardest-contrastive loss. The hardest-contrastive loss outperforms random triplets, hardest-triplet, and contrastive loss. ”norm.” denotes the normalized feature.

Note that FCGF maintains similar performance on the augmented dataset without any explicit mechanism.

6.6.4 Hardest-contrastive and Hardest-triplet Losses

We compare the traditional contrastive and triplet losses with the proposed hardest-contrastive and hardest-triplet losses in Tab. 6.3. As the hardest-triplet loss tends to collapse easily, we varied the number of hardest-negatives and random triplets per pair (increase proportionally to the batch size) and reported the feature-match recall on Tab. 6.3.

For the contrastive loss, we use both normalized (denoted norm.) and unnormalized features. We used L2 normalization to project features to the surface of a hypersphere and pass the gradient from the loss through the normalization layer to train the network with normalization. For the

Neg. margin	Pos. margin	Feature Match Recall	STD
4	0.1	0.9169	0.0478
3	0.1	0.9206	0.0398
2	0.1	**0.9344**	0.0362
1.4	0.1	0.9242	0.0439
1	0.1	0.9249	0.0403
0.5	0.1	0.8832	0.0433
4	0.2	0.9158	0.0450
2	0.2	0.9110	0.0438
1	0.2	0.9116	0.0527
1	0.4	0.9013	0.0464

Table 6.4: Feature-match recall of FCGF (5cm voxel downsampling) on 3DMatch with various positive margins m_p and negative margins m_n.

normalized features, we used positive margin 0.1, negative margin 1.4 ($\approx \sqrt{2}$); for the unnormalized features, positive margin 0.1 and negative margin 2. Similarly, we used both normalized (denoted norm.) and unnormalized features for the random triplet loss. (Note that the hardest-negative triplet with 0 hardest negatives becomes the random triplet.) However, for the random triplets with the normalized feature, higher margins lead to performance degradation, and at margin 0.5, the random triplets fail to achieve reasonable performance. For all triplets with unnormalized features, we use the margin 1.4. Please refer to the supplement for an analysis of the margin for the hardest-triplet loss.

Interestingly, as we increase the number of random triplets, feature-match recall also increases (the top section of Tab. 6.3). However, if we remove the hardest negatives altogether, the performance drops significantly (the row 0/2048). Surprisingly, we did not observe any degeneration for the hardest-negative contrastive loss, which uses the same mining technique, and does not require random negatives.

6.6.5 Effect of Margins for Hardest-contrastive

The hardest-contrastive loss in Eq. 6.5 requires two hyper-parameters: positive margin and negative margin. We trained networks with various margins with 5cm voxel downsampling and report the result in Tab. 6.4. In general, the ratio between negative margin and positive margin (m_n/m_p) plays a significant role: the larger the ratio, the higher the performance. However, the absolute value of the negative margin is also critical since 1/0.2 (negative margin / positive margin) has the same ratio as 0.5/0.1, but 1/0.2 yields better results. Please refer to the supplement for an analysis of the margin for the hardest-triplet loss.

	FPFH [23]	USC [29]	CGF [17]	3DMatch [36]	PPFNet [7]	Ours
Kitchen	0.36	0.52	0.72	0.85	0.90	**0.93**
Home 1	0.56	0.35	0.69	0.78	0.58	**0.91**
Home 2	0.43	0.47	0.46	0.61	0.57	**0.71**
Hotel 1	0.29	0.53	0.55	0.79	0.75	**0.91**
Hotel 2	0.36	0.20	0.49	0.59	0.68	**0.87**
Hotel 3	0.61	0.38	0.65	0.58	**0.88**	0.69
Study	0.31	0.46	0.48	0.63	0.68	**0.75**
Lab	0.31	0.49	0.42	0.51	0.62	**0.80**
Average	0.40	0.43	0.56	0.67	0.71	**0.82**

Table 6.5: Registration recall on 3DMatch [36].

DS voxel size	RTE (cm)	STD(cm)	RRE(°)	STD(°)	Succ. rate
3DFeat [34]	25.9	26.2	0.57	0.46	95.97%
FCGF 20cm	4.881	5.338	0.170	0.175	97.83%
FCGF 25cm	6.066	8.730	0.213	0.291	98.56%
FCGF 30cm	6.466	6.067	0.228	0.229	98.92%
FCGF 35cm	6.978	5.332	0.254	0.240	98.92%
FCGF 40cm	8.025	5.935	0.273	0.251	98.92%

Table 6.6: Results on the KITTI dataset. Relative Translation Error (RTE) and Relative Rotation Error (RRE) after RANSAC on FCGF trained with the hardest-contrastive loss with various downsampling voxel sizes. Success if RTE < 2m and RRE $< 5°$ [34].

Registration recall. We used the 3DMatch registration set [36] to evaluate the registration recall of FCGF. The results are reported in Tab. 6.5. For all experiments, we used RANSAC [9] with early termination [38].

6.6.6 Outdoor Experiment: KITTI

We trained FCGF on the KITTI registration dataset with various voxel-downsampling sizes and report Relative Translation Error (RTE) and Relative Rotation Error (RRE) with RANSAC in Tab. 6.6. Registration is considered successful if RTE < 2m and RRE $< 5°$ (following [34]). Note that the translation error and success rate increase as the voxel size increases. This is because a high-resolution point cloud increases the specificity of the registration, which leads to lower translation error. Fig. 6.6 visualizes the distribution and stability of the computed features on pairs of scans.

6.6.7 Runtime

We compare the runtimes of all different methods on 3DMatch in Fig. 6.1 and Tab. 6.1. The reported times include data preprocessing and feature extraction. We used an Intel i7 10-core 3.0GHz CPU (i7-6950) and an Nvidia Titan-X Pascal GPU to measure FCGF runtime. ([7, 6, 36] used an Intel i7 8-core 3.2GHz CPU and an Nvidia Titan-X Pascal.) We ran other baselines on the same workstation using PCL 1.8 [25] to test SHOT, USC, and Spin Image, and Open3D [38] to test FPFH. The reported times include both data preprocessing and feature extraction. FCGF is about 87 times faster than PPF-FoldNet [6], 350 times faster than 3DMatch [36], and 600 times faster than PerfectMatch [11]. We ascribe this speed to the fully-convolutional network that takes the point cloud directly without heavy preprocessing such as creating a volumetric function or searching neighboring points and grouping. On average, our approach takes about 0.164 seconds to extract features for a single fragment on 3DMatch with 2.5cm voxel size. However, with batch processing, the effective speed can be faster.

6.7 Conclusion

We presented fully-convolutional geometric features (FCGF): fast and compact metric features for geometric correspondence. Experimentally, we showed that FCGF is more accurate and faster than state-of-the-art methods. An interesting avenue for future work is to extend the FCGF methodology to end-to-end registration.

Bibliography

[1] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *CVPR*, 2015.

[2] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4D spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019.

[3] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *NIPS*, 2016.

[4] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017.

[5] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

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

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

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

[9] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 1981.

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *CVPR*, 2012.

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

[12] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018.

[13] Y. Guo, M. Bennamoun, F. A. Sohel, M. Lu, J. Wan, and N. M. Kwok. A comprehensive performance evaluation of 3d local feature descriptors. *IJCV*, 116(1), 2016.

[14] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

[15] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv:1703.07737*, 2017.

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

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

[18] M. T. Law, N. Thome, and M. Cord. Quadruplet-wise image similarity learning. In *ICCV*, 2013.

[19] Jie Lin, Olivier Morere, Vijay Chandrasekhar, Antoine Veillard, and Hanlin Goh. Deephash: Getting regularization, depth and fine-tuning right. *arXiv:1501.04711*, 2015.

[20] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[21] 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.

[22] Dario Rethage, Johanna Wald, Jurgen Sturm, Nassir Navab, and Federico Tombari. Fully-convolutional point networks for large-scale point clouds. In *ECCV*, 2018.

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

[24] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. In *IROS*, 2008.

[25] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *ICRA*, 2011.

[26] 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.

[27] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *NIPS*, 2016.

[28] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016.

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

[30] Evgeniya Ustinova and Victor Lempitsky. Learning deep embeddings with histogram loss. In *NIPS*, 2016.

[31] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-SNE. *JMLR*, 2008.

[32] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In *CVPR*, 2014.

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

[34] Zi Jian Yew and Gim Hee Lee. 3DFeat-Net: Weakly supervised local 3D features for point cloud registration. In *ECCV*, 2018.

[35] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

[36] 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.

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

[38] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.