Deep Closest Point: Learning Representations for Point Cloud Registration

Yue Wang
Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA 02139
yuewangx@mit.edu

Justin M. Solomon
Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA 02139
jsolomon@mit.edu

Abstract

Point cloud registration is a key problem for computer vision applied to robotics, medical imaging, and other applications. This problem involves finding a rigid transformation from one point cloud into another so that they align. Iterative Closest Point (ICP) and its variants provide simple and easily-implemented iterative methods for this task, but these algorithms can converge to spurious local optima. To address local optima and other difficulties in the ICP pipeline, we propose a learning-based method, titled Deep Closest Point (DCP), inspired by recent techniques in computer vision and natural language processing. Our model consists of three parts: a point cloud embedding network, an attention-based module combined with a pointer generation layer to approximate combinatorial matching, and a differentiable singular value decomposition (SVD) layer to extract the final rigid transformation. We train our model end-to-end on the ModelNet40 dataset and show in several settings that it performs better than ICP, its variants (e.g., Go-ICP, FGR), and the recently-proposed learning-based method PointNetLK. Beyond providing a state-of-the-art registration technique, we evaluate the suitability of our learned features transferred to unseen objects. We also provide preliminary analysis of our learned model to help understand whether domain-specific and/or global features facilitate rigid registration.

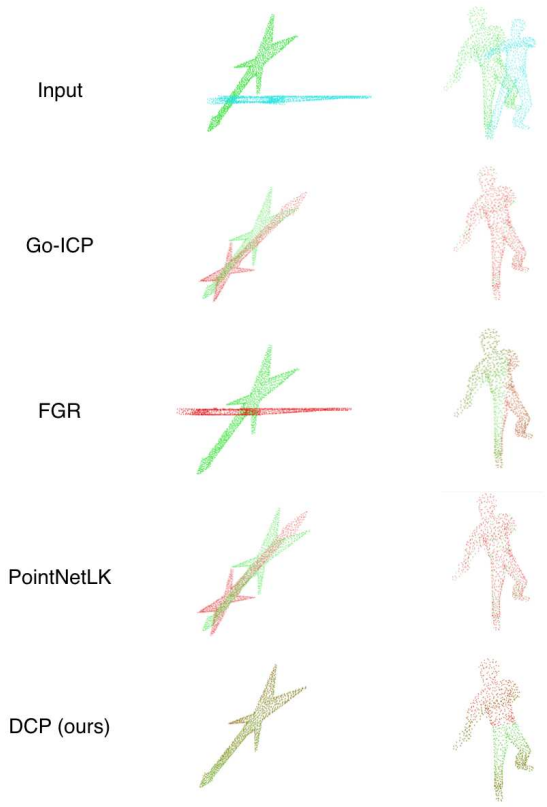


Figure 1. **Left**: a moved guitar. **Right**: rotated human. All methods work well with small transformation. However, only our method achieve satisfying alignment for objects with sharp features and large transformation.

1. Introduction

Geometric registration is a key task in many computational fields, including medical imaging, robotics, autonomous driving, and computational chemistry. In its most basic incarnation, registration involves the prediction of a rigid motion to align one shape to another, potentially obfuscated by noise and partiality.

Many modeling and computational challenges hamper the design of a stable and efficient registration method. Given exact correspondences, singular value decomposition yields the globally optimal alignment; similarly, computing matchings becomes easier given some global alignment information. Given these two observations, most algorithms alternate between these two steps to try to obtain a better result. The resultant iterative optimization algorithms, however, are prone to local optima.

The most popular example, Iterative Closest Point (ICP) [5, 40], alternates between estimating the rigid motion based

on a fixed correspondence estimate and updating the correspondences to their closest matches. Although ICP monotonically decreases a certain objective function measuring alignment, due to the non-convexity of the problem, ICP often stalls in suboptimal local minima. Many methods [37, 13, 55] attempt to alleviate this issue by using heuristics to improve the matching or by searching larger parts of the motion space $\mathrm{SE}(3)$. These algorithms are typically slower than ICP and still do not always provide acceptable output.

In this work, we revisit ICP from a deep learning perspective, addressing key issues in each part of the ICP pipeline using modern machine learning, computer vision, and natural language processing tools. We call our resulting algorithm *Deep Closest Point (DCP)*, a learning-based method that takes two point clouds and predicts a rigid transformation aligning them.

Our model consists of three parts: (1) We map the input point clouds to permutation/rigid-invariant embeddings that help identify matching pairs of points (we compare PointNet [33] and DGCNN [50] for this step); then, (2) an attention-based module combining pointer network [48, 46] predicts a soft matching between the point clouds; and finally, (3) a differentiable singular value decomposition layer predicts the rigid transformation. We train and test our model end-to-end on ModelNet40 [52] in various settings, showing our model is not only efficient but also outperforms ICP and its extensions, as well as the recently-proposed PointNetLK method [18]. Our learned features generalize to unseen data, suggesting that our model is learning salient geometric features.

Contributions: Our contributions include the following:

- We identify sub-network architectures designed to address difficulties in the classical ICP pipeline.
- We propose a simple architecture to predict a rigid transformation aligning two point clouds.
- We evaluate efficiency and performance in several settings and provide an ablation study to support details of our construction.
- We analyze whether local or global features are more useful for registration.
- We release our code to facilitate reproducibility and future research.

2. Related Work

Point cloud registration methods: ICP [5] is the best-known algorithm for solving rigid registration problems; it alternates between finding point cloud correspondences and solving a least-squares problem to update the alignment. ICP variants [37, 40, 6] consider issues with the basic method, like noise, partiality, and sparsity; probabilistic models [2, 15, 19] also can improve resilience to uncertain data. ICP can be viewed as an optimization algorithm searching jointly for a matching and a rigid alignment. Hence, [13] propose using the Levenberg–Marquardt algorithm to optimize the objective directly, which can yield a better solution. For more information, [32, 37] summarize ICP and its variants developed over the last 20 years.

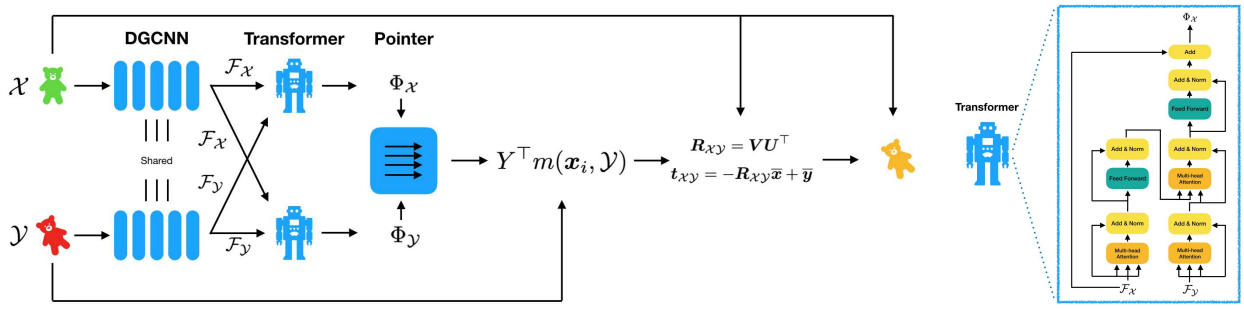
ICP-style methods are prone to local minima due to non-convexity. To find a good optimum, Go-ICP [55] uses a branch-and-bound (BnB) method to search the motion space $\mathrm{SE}(3)$. It outperforms local ICP methods when a global solution is desired but is several orders of magnitude slower than other ICP variants despite using local ICP to accelerate the search process. Other methods attempt to identify global optima using Riemannian optimization [36], convex relaxation [27], and mixed-integer programming [21].

Recently, descriptor learning methods have brought significant progress in point cloud registration: 3DMatch [59] proposes learning a local volumetric patch descriptor to establish correspondences; 3DFeatNet [56] takes similar approach to point cloud representation for local regions; PPFFoldNet [9] uses a folding-based autoencoder to learn a local descriptor; and 3DSmoothNet [14] employs a voxelized smoothed density value (SDV) representation for descriptor learning. The critical difference between our algorithm and these techniques is that we carry out end-to-end registration prediction while the others target descriptor learning. Also, these works rely on keypoint detection and outlier removal using RANSAC. Concurrent work [26] proposes an end-to-end pipeline for point cloud registration. A significant difference is that theirs computes loss for each point sample while ours optimizes the registration objective directly.

Learning on graphs and point sets: A broad class of deep architectures for geometric data termed *geometric deep learning* [7] includes recent methods learning on graphs [51, 60, 12] and point clouds [33, 34, 50, 57].

The *graph neural network* (GNN) is introduced in [39]; similarly, [11] defines convolution on graphs (GCN) for molecular data. [24] uses renormalization to adapt to the graph structure and applies GCN to semi-supervised learning on graphs. MoNet [28] learns a dynamic aggregation function based on the graph structure, generalizing GNN. Finally, graph attention networks (GAT) [47] incorporate multi-head attention into GCN. DGCNN [50] (discussed below) can be regarded as a graph neural network applied to point clouds with dynamic edges.

Another branch of geometric deep learning includes PointNet [33] and other algorithms designed to process point clouds. PointNet can be seen as applying GCN to graphs without edges, mapping points in $\mathbb{R}^3$ to high-dimensional space. PointNet only encodes global features gathered from the point cloud's embedding, impeding application to tasks involving local geometry. To address this issue, PointNet++ [34] applies a shared PointNet to k-nearest neighbor clus-



(a) Network architecture (b) Transformer module

Figure 2. Network architecture for DCP, including the Transformer module for DCP-v2.

ters to learn local features. As an alternative, DGCNN [50] explicitly recovers the graph structure in both Euclidean space and feature space and applies graph neural networks to the result. PCNN [3] uses an extension operator to define convolution on point clouds, while PointCNN [25] applies Euclidean convolution after applying a learned transformation. Finally, SPLATNet [43] encodes point clouds on a lattice and performs bilateral convolution. All these works aim to apply convolution-like operations to point clouds and extract local geometric features.

Sequence-to-sequence learning and pointer networks: Many tasks in natural language processing, including machine translation, language modeling, and question answering, can be formulated as sequence-to-sequence (seq2seq) problems. [45] first uses deep neural networks (DNN) to address seq2seq problems at large scale. Seq2seq, however, often involves predicting discrete tokens corresponding to positions in the input sequence. This problem is difficult because there is an exponential number of possible matchings between input and output positions. Similar problems can be found in optimal transport [41, 31], combinatorial optimization [20], and graph matching [54]. To address this issue, in our registration pipeline we use a related method to Pointer Networks [48], which use attention as a pointer to select from the input sequence. In each output step, a Pointer Network predicts a distribution over positions and uses it as a "soft pointer." The pointer module is fully differentiable, and the whole network can be trained end-to-end.

Non-local approaches: To denoise images, non-local means [8] leverages the simple observation that Gaussian noise can be removed by non-locally weighted averaging all pixels in an image. Recently, non-local neural networks [49] have been proposed to capture long-range dependencies in video understanding; [53] uses the non-local module to denoise feature maps to defend against adversarial attacks. Another instantiation of non-local neural networks, known as *relational networks* [38], has shown effectiveness in visual reasoning [38], meta-learning [44], object detection [17], and reinforcement learning [58]. Its counterpart in natural language processing, attention, is arguably the most fruitful recent advance in this discipline. [46] replaces recurrent neural networks [22, 16] with a model called the Transformer, consisting of several stacked multi-head attention modules. Transformer-based models [10, 35] outperform other recurrent models by a considerable amount in natural language processing. In our work, we also use a Transformer to learn contextual information of point clouds.

3. Problem Statement

In this section, we formulate the rigid alignment problem and discuss the ICP algorithm, highlighting key issues in the ICP pipeline. We use $\mathcal{X}$ and $\mathcal{Y}$ to denote two point clouds, where $\mathcal{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_i, \dots, \boldsymbol{x}_N\} \subset \mathbb{R}^3$ and $\mathcal{Y} = \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_j, \dots, \boldsymbol{y}_M\} \subset \mathbb{R}^3$. For ease of notation, we consider the simplest case, in which $M = N$. The methods we describe here extend easily to the $M \neq N$ case because DGCNN, Transformer, and Softmax treat inputs as unordered sets. None requires $\mathcal{X}$ and $\mathcal{Y}$ to have the same length or a bijective matching.

In the rigid alignment problem, we assume $\mathcal{Y}$ is transformed from $\mathcal{X}$ by an unknown rigid motion. We denote the rigid transformation as $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$ where $\boldsymbol{R}_{\mathcal{XY}} \in \mathrm{SO}(3)$ and $\boldsymbol{t}_{\mathcal{XY}} \in \mathbb{R}^3$. We want to minimize the mean-squared error $E(\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}})$, which—if $\mathcal{X}$ and $\mathcal{Y}$ are ordered the same way (meaning $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are paired)—can be written

$$E(\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}) = \frac{1}{N} \sum_{i}^{N} \|\boldsymbol{R}_{\mathcal{XY}} \boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{y}_i\|^2. \quad (1)$$

Define centroids of $\mathcal{X}$ and $\mathcal{Y}$ as

$$\overline{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_i \quad \text{and} \quad \overline{\boldsymbol{y}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y}_i. \quad (2)$$

Then the cross-covariance matrix $\boldsymbol{H}$ is given by

$$\boldsymbol{H} = \sum_{i=1}^{N} (\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{y}_i - \overline{\boldsymbol{y}})^\top. \tag{3}$$

We can use the singular value decomposition (SVD) to decompose $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$. Then, the alignment minimizing $E(\cdot,\cdot)$ in (1) is given in closed-form by

$$\boldsymbol{R}_{\mathcal{XY}} = \boldsymbol{V}\boldsymbol{U}^\top \qquad \text{and} \qquad \boldsymbol{t}_{\mathcal{XY}} = -\boldsymbol{R}_{\mathcal{XY}}\overline{\boldsymbol{x}} + \overline{\boldsymbol{y}}. \tag{4}$$

Here, we take the convention that $\boldsymbol{U}, \boldsymbol{V} \in \mathrm{SO}(3)$, while $\boldsymbol{S}$ is diagonal but potentially signed; this accounts for orientation-reversing choices of $\boldsymbol{H}$. This classic orthogonal Procrustes problem assumes that the point sets are matched to each other, that is, that $\boldsymbol{x}_i$ should be mapped to $\boldsymbol{y}_i$ in the final alignment for all i. If the correspondence is unknown, however, the objective function E must be revised to account for matching:

$$E(\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}) = \frac{1}{N}\sum_{i}^{N} \|\boldsymbol{R}_{\mathcal{XY}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{y}_{m(x_i)}\|^2. \tag{5}$$

Here, a mapping m from each point in $\mathcal{X}$ to its corresponding point in $\mathcal{Y}$ is given by

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \arg\min_j \|\boldsymbol{R}_{\mathcal{XY}}\boldsymbol{x}_i + \boldsymbol{t}_{\mathcal{XY}} - \boldsymbol{y}_j\|. \tag{6}$$

Equations (5) and (6) form a classic *chicken-and-egg* problem. If we know the optimal rigid transformation $[\boldsymbol{R}_{\mathcal{XY}}, \boldsymbol{t}_{\mathcal{XY}}]$, then the mapping m can be recovered from (6); conversely, given the optimal mapping m, the transformation can be computed using (4).

ICP iteratively approaches a stationary point of E in (5), including the mapping $m(\cdot)$ as one of the variables in the optimization problem. It alternates between two steps: finding the current optimal transformation based on a previous mapping m^{k-1} and finding an optimal mapping m^k based on the current transformation using (6), where k denotes the current iteration. The algorithm terminates when a fixed point or stall criterion is reached. This procedure is easy to implement and relatively efficient, but it is extremely prone to local optima; a distant initial alignment yields a poor estimate of the mapping m, quickly leading to a situation where the algorithm gets stuck. Our goal is to use learned embeddings to recover a better matching $m(\cdot)$ and to use this matching to compute a rigid transformation, as we will detail in the next section.

4. Deep Closest Point

Having established preliminaries about the rigid alignment problem, we are now equipped to present our Deep Closest Point architecture, illustrated in Figure 2. In short, we embed point clouds into high-dimensional space using PointNet [33] or DGCNN [50] (§4.1), encode contextual information using an attention-based module (§4.2), and finally estimate an alignment using a differentiable SVD layer (§4.4).

4.1. Initial Features

The first stage of our pipeline embeds the unaligned input point clouds $\mathcal{X}$ and $\mathcal{Y}$ into a common space used to find matching pairs of points between the two clouds. The goal is to find an embedding that quotients out rigid motion while remaining sensitive to relevant features for rigid matching. We evaluate two possible choices of learnable embedding modules, PointNet [33] and DGCNN [50].

Since we use per-point embeddings of the two input point clouds to generate a mapping m and recover the rigid transformation, we seek a feature *per point* in the input point clouds rather than one feature per cloud. For this reason, in these two network architectures, we use the representations generated before the last aggregation function, notated $\mathcal{F}_{\mathcal{X}} = \{\boldsymbol{x}_1^L, \boldsymbol{x}_2^L, ..., \boldsymbol{x}_i^L, ..., \boldsymbol{x}_N^L\}$ and $\mathcal{F}_{\mathcal{Y}} = \{\boldsymbol{y}_1^L, \boldsymbol{y}_2^L, ..., \boldsymbol{y}_i^L, ..., \boldsymbol{y}_N^L\}$, assuming a total of L layers.

In more detail, PointNet takes a set of points, embeds each by a nonlinear function from $\mathbb{R}^3$ into a higher-dimensional space, and optionally outputs a global feature vector for the whole point cloud after applying a channel-wise aggregation function f (e.g., $\max$ or $\sum$). Let $\boldsymbol{x}_i^l$ be the embedding of point i in the l-th layer, and let h_θ^l be a nonlinear function in the l-th layer parameterized by a shared multilayer perceptron (MLP). Then, the forward mechanism is given by $\boldsymbol{x}_i^l = h_\theta^l(\boldsymbol{x}_i^{l-1})$.

While PointNet largely extracts information based on the embedding of each point in the point cloud independently, DGCNN explicitly incorporates local geometry into its representation. In particular, given a set of points $\mathcal{X}$, DGCNN constructs a k-NN graph $\mathcal{G}$, applies a nonlinearity to the values at edge endpoints to obtain edgewise values, and performs vertex-wise aggregation ($\max$ or $\sum$) in each layer. The forward mechanism of DGCNN is thus

$$\boldsymbol{x}_i^l = f(\{h_\theta^l(\boldsymbol{x}_i^{l-1}, \boldsymbol{x}_j^{l-1}) \; \forall j \in \mathcal{N}_i\}), \tag{7}$$

where $\mathcal{N}_i$ denotes the neighbors of vertex i in graph $\mathcal{G}$. While PointNet features do not incorporate local neighborhood information, we find empirically that DGCNN's local features are critical for high-quality matching in subsequent steps of our pipeline (see §6.1).

4.2. Attention

Our transition from PointNet to DGCNN is motivated by the observation that the most useful features for rigid alignment are learned jointly from local and global information. We additionally can improve our features for matching

by making them *task-specific*, that is, changing the features depending on the particularities of $\mathcal{X}$ and $\mathcal{Y}$ together rather than embedding $\mathcal{X}$ and $\mathcal{Y}$ independently. That is, the task of rigidly aligning, say, organic shapes might require different features than those for aligning mechanical parts with sharp edges. Inspired by the recent success of BERT [10], non-local neural networks [49], and relational networks [38] using attention-based models, we design a module to learn co-contextual information by capturing *self-attention* and *conditional attention*.

Take $\mathcal{F}_\mathcal{X}$ and $\mathcal{F}_\mathcal{Y}$ to be the embeddings generated by the modules in §4.1; these embeddings are computed independently of one another. Our attention model learns a function $\phi : \mathbb{R}^{N\times P} \times \mathbb{R}^{N\times P} \to \mathbb{R}^{N\times P}$, where P is embedding dimension, that provides new embeddings of the point clouds as

$$\begin{aligned} \Phi_\mathcal{X} &= \mathcal{F}_\mathcal{X} + \phi(\mathcal{F}_\mathcal{X}, \mathcal{F}_\mathcal{Y}) \\ \Phi_\mathcal{Y} &= \mathcal{F}_\mathcal{Y} + \phi(\mathcal{F}_\mathcal{Y}, \mathcal{F}_\mathcal{X}) \end{aligned} \tag{8}$$

Notice we treat ϕ as a *residual* term, providing an additive change to $\mathcal{F}_\mathcal{X}$ and $\mathcal{F}_\mathcal{Y}$ depending on the order of its inputs. The idea here is that the map $\mathcal{F}_\mathcal{X} \mapsto \Phi_\mathcal{X}$ modifies the features associated to the points in $\mathcal{X}$ in a fashion that is knowledgeable about the structure of $\mathcal{Y}$; the map $\mathcal{F}_\mathcal{Y} \mapsto \Phi_\mathcal{Y}$ serves a symmetric role. We choose ϕ as an asymmetric function given by a Transformer [46]. The Transformer is a framework to solve sequence-to-sequence problems. It consists of several stacked encoder-decoder layers. The encoder takes one sequence/set ($\mathcal{F}_\mathcal{X}$) and encodes it to an embedding space by using a self-attention layer and shared multi-layer perceptron (MLP). The decoder has two parts: The first part takes another sequence/set ($\mathcal{F}_\mathcal{Y}$) and encodes it in the same way as the encoder, and the second part relates two embedded sequences/sets using co-attention. Therefore, the output embeddings ($\Phi_\mathcal{X}$ and $\Phi_\mathcal{Y}$) have contextual information from both sequences/sets ($\mathcal{F}_\mathcal{X}$ and $\mathcal{F}_\mathcal{Y}$). The matching problem we encounter in rigid alignment is analogous to the sequence-to-sequence problem that inspired its development, other than their use of positional embeddings to describe where words are in a sentence.

4.3. Pointer Generation

The most common failure mode of ICP occurs when the matching estimate m^k is far from optimal. When this occurs, the rigid motion subsequently estimated using (6) does not significantly improve alignment, leading to a spurious local optimum. As an alternative, our learned embeddings are trained specifically to expose matching pairs of points using a simple procedure explained below. We term this step *pointer generation*, again inspired by terminology in the attention literature introduced in §4.2.

To avoid choosing non-differentiable hard assignments, we use a probabilistic approach that generates a (singly-stochastic) "soft map" from one point cloud into the other. That is, each $\boldsymbol{x}_i \in \mathcal{X}$ is assigned a probability vector over elements of $\mathcal{Y}$ given by

$$m(\boldsymbol{x}_i, \mathcal{Y}) = \operatorname{softmax}(\Phi_\mathcal{Y} \Phi_{\boldsymbol{x}_i}^\top). \tag{9}$$

Here, $\Phi_\mathcal{Y} \in \mathbb{R}^{N\times P}$ denotes the embedding of $\mathcal{Y}$ generated by the attention module, and $\Phi_{\boldsymbol{x}_i}$ denotes the i-th row of the matrix $\Phi_\mathcal{X}$ from the attention module. We can think of $m(\boldsymbol{x}_i, \mathcal{Y})$ as a *soft pointer* from each $\boldsymbol{x}_i$ into the elements of $\mathcal{Y}$.

4.4. SVD Module

The final module in our architecture extracts the rigid motion from the soft matching computed in §4.3. We use the soft pointers to generate a matching averaged point in $\mathcal{Y}$ for each point in $\mathcal{X}$:

$$\hat{\boldsymbol{y}}_i = Y^\top m(\boldsymbol{x}_i, \mathcal{Y}) \in \mathbb{R}^3. \tag{10}$$

Here, we define $Y \in \mathbb{R}^{N\times 3}$ to be a matrix containing the points in Y. Then, $\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}$ and $\boldsymbol{t}_{\mathcal{X}\mathcal{Y}}$ are extracted using (4) based on the pairing $\boldsymbol{x}_i \mapsto \hat{\boldsymbol{y}}_i$ over all i.

To backpropagate gradients through the networks, we need to differentiate the SVD. [29] describes a standard means of computing this derivative; versions of this calculation are included in PyTorch [30] and TensorFlow [1]. Note we need to solve only 3×3 eigenproblems, small enough to be solved using simple algorithms or even (in principle) a closed-form formula.

4.5. Loss

Combined, the modules above map from a pair of point clouds $\mathcal{X}$ and $\mathcal{Y}$ to a rigid motion $[\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}]$ that aligns them to each other. The initial feature module (§4.1) and the attention module (§4.2) are both parameterized by a set of neural network weights, which must be learned during a training phase. We employ a fairly straightforward strategy for training, measuring the deviation of $[\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}, \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}]$ from ground truth for synthetically-generated pairs of point clouds.

We use the following loss function to measure our model's agreement to the ground-truth rigid motions:

$$\text{Loss} = \|\boldsymbol{R}_{\mathcal{X}\mathcal{Y}}^\top \boldsymbol{R}_{\mathcal{X}\mathcal{Y}}^g - I\|^2 + \|\boldsymbol{t}_{\mathcal{X}\mathcal{Y}} - \boldsymbol{t}_{\mathcal{X}\mathcal{Y}}^g\|^2 + \lambda\|\theta\|^2. \tag{11}$$

Here, g denotes ground-truth. The first two terms define a simple distance on $\mathrm{SE}(3)$. The third term denotes Tikhonov regularization of the DCP parameters θ, which serves to reduce the complexity of the network.

5. Experiments

We compare our models to ICP, Go-ICP [55], Fast Global Registration (FGR) [61], and the recently-proposed PointNetLK deep learning method [18]. We denote our model

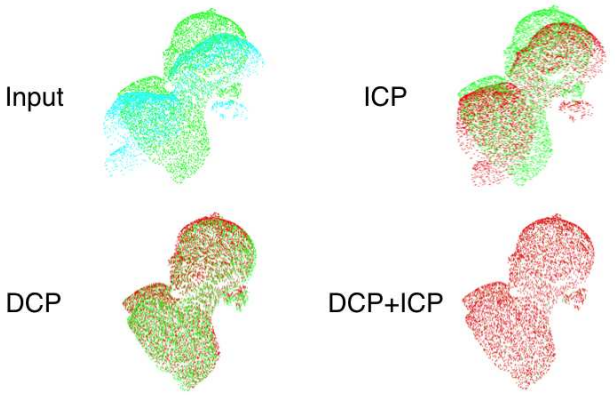


Figure 3. **Top left**: input. **Top right**: result of ICP with random initialization. **Bottom left**: initial transformation provided by DCP. **Bottom right**: result of ICP initialized with DCP. Using a good initial transformation provided by DCP, ICP converges to the global optimum.

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	894.897339	29.914835	23.544817	0.084643	0.290935	0.248755
Go-ICP [55]	140.477325	11.852313	2.588463	0.000659	0.025665	0.007092
FGR [61]	87.661491	9.362772	1.999290	0.000194	0.013939	0.002839
PointNetLK [18]	227.870331	15.095374	4.225304	0.000487	0.022065	0.005404
DCP-v1 (ours)	6.480572	2.545697	1.505548	**0.000003**	**0.001763**	**0.001451**
DCP-v2 (ours)	**1.307329**	**1.143385**	**0.770573**	**0.000003**	0.001786	**0.001195**

Table 1. ModelNet40: Test on unseen point clouds

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	892.601135	29.876431	23.626110	0.086005	0.293266	0.251916
Go-ICP [55]	192.258636	13.865736	2.914169	0.000491	0.022154	0.006219
FGR [61]	97.002747	9.848997	**1.445460**	0.000182	0.013503	0.002231
PointNetLK [18]	306.323975	17.502113	5.280545	0.000784	0.028007	0.007203
DCP-v1 (ours)	19.201385	4.381938	2.680408	**0.000025**	**0.004950**	**0.003597**
DCP-v2 (ours)	**9.923701**	**3.150191**	2.007210	**0.000025**	0.005039	0.003703

Table 2. ModelNet40: Test on unseen categories

without attention (§4.2) as **DCP-v1** and the full model with attention as **DCP-v2**. Go-ICP is ported from the authors' released code. For ICP and FGR, we use the implementations in Intel Open3D [62]. For PointNetLK, we adapt the code partially released by the authors. Notice that FGR [61] uses additional geometric features. In all experiments, the proposed method does *not* make an assumption that a good initial pose is given; the test point clouds are generated in the same way as the training point clouds. ICP and its variants are initialized with an identity transformation matrix.

The architecture of DCP is shown in Figure 2. We use 5 *EdgeConv* (denoted as DGCNN [50]) layers for both DCP-v1 and DCP-v2. The numbers of filters in each layer are $[64, 64, 128, 256, 512]$. In the Transformer layer, the number of heads in multi-head attention is 4 and the embedding dimension is 1024. We use LayerNorm [4] without Dropout [42]. Adam [23] is used to optimize the network parameters, with an initial learning rate of 0.001. We divide the learning rate by 10 at epochs 75, 150, and 200, training for a total of 250 epochs. DCP-v1 does not use the Transformer module but rather employs identity mappings $\Phi_{\mathcal{X}} = \mathcal{F}_{\mathcal{X}}$ and $\Phi_{\mathcal{Y}} = \mathcal{F}_{\mathcal{Y}}$.

We experiment on the ModelNet40 [52] dataset, which consists of 12,311 meshed CAD models from 40 categories. Of these, we use 9,843 models for training and 2,468 models for testing. We follow the experimental settings of PointNet [33], uniformly sampling 1,024 points from each model's outer surface. As in previous work, points are centered and rescaled to fit in the unit sphere, and no features other than (x, y, z) coordinates appear in the input.

We measure mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) between ground truth values and predicted values. Ideally, all of these error metrics should be zero if the rigid alignment is perfect. All angular measurements in our results are in units of degrees.

5.1. ModelNet40: Full Dataset Train & Test

In our first experiment, we randomly divide all the point clouds in the ModelNet40 dataset into training and test sets, with no knowledge of the category label; different point clouds are used during training and during testing. During training, we sample a point cloud $\mathcal{X}$. Along each axis, we randomly draw a rigid transformation; the rotation along each axis is uniformly sampled in $[0, 45^\circ]$ and translation is in $[-0.5, 0.5]$. $\mathcal{X}$ and a transformation of $\mathcal{X}$ by the rigid motion are used as input to the network, which is evaluated against the known ground truth using (11).

Table 1 evaluates the performance of our method and its peers in this experiment (vanilla ICP nearly fails). DCP-v1 already outperforms other methods under all the performance metrics, and DCP-v2 exhibits even stronger performance.

5.2. ModelNet40: Category Split

To test the generalizability of different models, we split ModelNet40 evenly by category into training and testing sets. We train DCP and PointNetLK on the first 20 categories, then test them on the held-out categories. ICP, Go-ICP, and FGR are also tested on the held-out categories. As shown in Table 2, on unseen categories, FGR behaves more strongly than other methods. DCP-v1 has much worse performance than DCP-v2, supporting our use of the attention module. Although the learned representations are task-dependent, DCP-v2 exhibits smaller error than others except for FGR, including the learning-based method PointNetLK.

5.3. ModelNet40: Resilience to Noise

We also experiment with adding noise to each point of the input point clouds. We sample noise independently from

Model	MSE($\boldsymbol{R}$)	RMSE($\boldsymbol{R}$)	MAE($\boldsymbol{R}$)	MSE($\boldsymbol{t}$)	RMSE($\boldsymbol{t}$)	MAE($\boldsymbol{t}$)
ICP	882.564209	29.707983	23.557217	0.084537	0.290752	0.249092
Go-ICP [55]	131.182495	11.453493	2.534873	0.000531	0.023051	0.004192
FGR [61]	607.694885	24.651468	10.055918	0.011876	0.108977	0.027393
PointNetLK [18]	256.155548	16.004860	4.595617	0.000465	0.021558	0.005652
DCP-v1 (ours)	6.926589	2.631841	1.515879	0.000003	0.001801	0.001697
DCP-v2 (ours)	**1.169384**	**1.081380**	**0.737479**	**0.000002**	**0.001500**	**0.001053**

Table 3. ModelNet40: Test on objects with Gaussian noise

# points	ICP	Go-ICP	FGR	PointNetLK	DCP-v1	DCP-v2
512	0.003972	15.012375	0.033297	0.043228	0.003197	0.007932
1024	0.004683	15.405995	0.088199	0.055630	0.003300	0.008295
2048	0.044634	15.766001	0.138076	0.146121	0.040397	0.073697
4096	0.044585	15.984596	0.157124	0.162007	0.039984	0.74263

Table 4. Inference time (in seconds)

$\mathcal{N}(0, 0.01)$, clip the noise to $[-0.05, 0.05]$, and add it to $\mathcal{X}$ during testing. In this experiment, we use the model from §5.1 trained on noise-free data from all of ModelNet40.

Table 3 shows the results of this experiment. ICP typically converges to a far-away fixed point, and FGR is sensitive to noise. Go-ICP, PointNetLK, and DCP, however, remain robust to noise.

5.4. DCP Followed By ICP

Since our experiments involve point clouds whose initial poses are far from aligned, ICP fails nearly every experiment we have presented so far. In large part, this failure is due to the lack of a good initial guess. As an alternative, we can use ICP as a *local* algorithm by initializing ICP with a rigid transformation output from our DCP model. Figure 3 shows an example of this two-step procedure; while ICP fails at the global alignment task, with better initialization provided by DCP, it converges to the global optimum. In some sense, this experiment shows how ICP can be an effective way to "polish" the alignment generated by DCP.

5.5. Efficiency

We profile the inference time of different methods on a desktop computer with an Intel I7-7700 CPU, an Nvidia GTX 1070 GPU, and 32G memory. Computational time is measured in seconds and is computed by averaging 100 results. As shown in Table 4, DCP-v1 is the fastest method among our points of comparison, and DCP-v2 is only slower than vanilla ICP.

6. Ablation Study

We conduct several ablation experiments in this section, dissecting DCP and replacing each part with an alternative to understand the value of our construction. All experiments are done in the same setting as the experiments in §5.1.

Metrics	PN+DCP-v1,	DGCNN+DCP-v1	PN+DCP-v2	DGCNN+DCP-v2
MSE($\boldsymbol{R}$)	17.008427	6.480572	49.863022	1.307329
RMSE($\boldsymbol{R}$)	4.124127	2.545697	7.061375	1.143385
MAE($\boldsymbol{R}$)	2.800184	1.505548	4.485052	0.770573
MSE($\boldsymbol{t}$)	0.000697	0.000003	0.000258	0.000003
RMSE($\boldsymbol{t}$)	0.026409	0.001763	0.016051	0.001786
MAE($\boldsymbol{t}$)	0.01327	0.001451	0.010546	0.001195

Table 5. Ablation study: PointNet or DGCNN?

Metrics	DCP-v1+MLP	DCP-v1+SVD	DCP-v2+MLP	DCP-v2+SVD
MSE($\boldsymbol{R}$)	21.115917	6.480572	9.923701	1.307329
RMSE($\boldsymbol{R}$)	4.595206	2.545697	3.150191	1.143385
MAE($\boldsymbol{R}$)	3.291298	1.505548	2.007210	0.770573
MSE($\boldsymbol{t}$)	0.000861	0.000003	0.000025	0.000003
RMSE($\boldsymbol{t}$)	0.029343	0.001763	0.005039	0.001786
MAE($\boldsymbol{t}$)	0.022501	0.001451	0.003703	0.001195

Table 6. Ablation study: MLP or SVD?

6.1. PointNet or DGCNN?

We first try to answer whether the localized features gathered by DGCNN provide value over the coarser features that can be measured using the simpler PointNet model. As discussed in [50], PointNet [33] learns a global descriptor of the whole shape while DGCNN [50] learns local geometric features via constructing the k-NN graph. We replace the DGCNN with PointNet (denoted as PN) and conduct the experiments in §5.1 on ModelNet40 [52], using DCP-v1 and DCP-v2. Table 5. Models perform consistently better with DGCNN than their counterparts with PointNet.

6.2. MLP or SVD?

While MLP is in principle a universal approximator, our SVD layer is designed to compute a rigid motion specifically. In this experiment, we examine whether an MLP or a custom-designed layer is better for registration. We compare MLP and SVD with both DCP-v1 and DCP-v2 on ModelNet40. Table 6 shows both DCP-v1 and DCP-v2 perform better with SVD layer than MLP. This supports our motivation to compute rigid transformation using SVD.

6.3. Embedding Dimension

[33] remarks that the embedding dimension is an important parameter affecting the accuracy of point cloud deep learning models up to a critical threshold, after which there is an insignificant difference. To verify our choice of dimensionality, we compare models with embeddings into spaces of different dimensions. We test models with DCP-v1 and v2, using DGCNN to embed the point clouds into $\mathbb{R}^{512}$ or $\mathbb{R}^{1024}$. The results in Table 7 show that increasing the embedding dimension from 512 to 1024 does marginally help DCP-v2, but for DCP-v1 there is small degeneracy. Our results are consistent with the hypothesis in [33].

Metrics	DCP-v1 (512)	DCP-v1 (1024)	DCP-v2 (512)	DCP-v2 (1024)
MSE($\boldsymbol{R}$)	6.480572	7.291216	1.307329	1.217545
RMSE($\boldsymbol{R}$)	2.545697	2.700225	1.143385	1.103424
MAE($\boldsymbol{R}$)	1.505548	1.616465	0.770573	0.750242
MSE($\boldsymbol{t}$)	0.000003	0.000001	0.000003	0.000003
RMSE($\boldsymbol{t}$)	0.001763	0.001150	0.001786	0.001696
MAE($\boldsymbol{t}$)	0.001451	0.000677	0.001195	0.001170

Table 7. Ablation study: Embedding dimension

7. Conclusion

In some sense, the key observation in our Deep Closest Point technique is that learned features greatly facilitate rigid alignment algorithms; by incorporating DGCNN [50] and an attention module, our model reliably extracts the correspondences needed to find rigid motions aligning two input point clouds. Our end-to-end trainable model is reliable enough to extract a high-quality alignment in a single pass, which can be improved by iteration or "polishing" via classical ICP.

DCP is immediately applicable to rigid alignment problems as a drop-in replacement for ICP with improved behavior. Beyond its direct usage, our experiments suggest several avenues for future inquiry. One straightforward extension is to see if our learned embeddings transfer to other tasks like classification and segmentation. We could also train DCP to be applied *iteratively* (or recursively) to refine the alignment, rather than attempting to align in a single pass; insight from reinforcement learning could help refine approaches in this direction, using mean squared error as a reward to learn a policy that controls when to stop iterating.

We are also interested in testing on scenes, which often have up to 300,000 points. Current deep networks, however, can only handle object-level point clouds (each usually has around 500 to 5,000 points); this is a common limitation of recent point cloud learning methods. Testing on scenes, no matter the task, requires designing an efficient scene-level point cloud encoding network, which is a promising but challenging direction for point cloud learning generally. Finally, we hope our method can be incorporated into larger pipelines to enable high-accuracy Simultaneous Localization and Mapping (SLAM) or Structure from Motion (SFM).

8. Acknowledgements

The authors acknowledge the generous support of Army Research Office grant W911NF-12-R-0011, of National Science Foundation grant IIS-1838071, from an Amazon Research Award, from the MIT-IBM Watson AI Laboratory, from the Toyota-CSAIL Joint Research Center, from Air Force Office of Scientific Research award FA9550-19-1-0319, from a gift from Adobe Systems, and from the Skoltech-MIT Next Generation Program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these organizations. Yue Wang wants to thank David Palmer for helpful discussion.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 5

[2] Gabriel Agamennoni, Simone Fontana, Roland Siegwart, and Domenico Sorrenti. Point clouds registration with probabilistic data association. In *Proceedings of The International Conference on Intelligent Robots and Systems (IROS)*, 2016. 2

[3] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Transactions on Graphics*, 37(4):71:1–71:12, July 2018. 3

[4] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 6

[5] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb. 1992. 1, 2

[6] Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. Sparse iterative closest point. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing*, SGP '13, pages 113–123, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association. 2

[7] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 2

[8] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '05, pages 60–65, Washington, DC, USA, 2005. IEEE Computer Society. 3

[9] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPF-FoldNet: Unsupervised learning of rotation invariant 3D local descriptors. *ArXiv*, abs/1808.10322, 2018. 2

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. 3, 5

[11] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. D. Lawrence,

D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2224–2232. Curran Associates, Inc., 2015. 2

[12] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 869–877, 2018. 2

[13] Andrew W. Fitzgibbon. Robust registration of 2D and 3D point sets. In *British Machine Vision Conference*, pages 662–670, 2001. 2

[14] Zan Gojcic, Caifa Zhou, Jan Dirk Wegner, and Wieser Andreas. The perfect match: 3D point cloud matching with smoothed densities. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[15] Timo Hinzmann, Thomas Stastny, Gianpaolo Conte, Patrick Doherty, Piotr Rudol, Marius Wzorek, Enric Galceran, Roland Siegwart, and Igor Gilitschenski. Collaborative 3D reconstruction using heterogeneous uavs: System and experiments. In *International Symposium on Experimental Robotics*, 2016. 2

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov. 1997. 3

[17] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. *arXiv:1711.11575*, 2017. 3

[18] Rangaprasad Arun Srivatsan Hunter Goforth, Yasuhiro Aoki and Simon Lucey. PointNetLK: Robust & efficient point cloud registration using PointNet. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, LA, USA, June 2019. 2, 5, 6, 7

[19] Dirk Hhnel and Wolfram Burgard. Probabilistic matching for 3D scan registration. In *Proceedings of the VDI Conference*, 2002. 2

[20] Ivelin Ivanov. Review of combinatorial optimization - theory and algorithms. *SIGACT News*, 33(2):14–16, June 2002. 3

[21] Gregory Izatt, Hongkai Dai, and Russ Tedrake. Globally optimal object pose estimation in point clouds with mixed-integer programming. In *International Symposium on Robotics Research*, 12 2017. 2

[22] Michael I. Jordan. Artificial neural networks. chapter Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, pages 112–127. IEEE Press, Piscataway, NJ, USA, 1990. 3

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 6

[24] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016. 2

[25] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on X-transformed points. In *Advances in Neural Information Processing Systems 31*, pages 828–838, 2018. 3

[26] Weixin Lu, Guowei Wan, Yao Zhou, Xiangyu Fu, P. Yuan, and Shiyu Song. DeepICP: An end-to-end deep neural network for 3D point cloud registration. *ArXiv*, abs/1905.04153, 2019. 2

[27] Haggai Maron, Nadav Dym, Itay Kezurer, Shahar Kovalsky, and Yaron Lipman. Point registration via efficient convex relaxation. *ACM Transactions on Graphics*, 35(4):73:1–73:12, July 2016. 2

[28] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *IEEE Conference on Computer Vision and Pattern Recognition CVPR*, pages 5425–5434, 2017. 2

[29] Théodore Papadopoulo and Manolis IA Lourakis. Estimating the Jacobian of the singular value decomposition: Theory and applications. In *European Conference on Computer Vision*, pages 554–570. Springer, 2000. 5

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017. 5

[31] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019. 3

[32] François Pomerleau, Francis Colas, and Roland Siegwart. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends in Robotics*, 4(1):1–104, May 2015. 2

[33] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85. IEEE Computer Society, 2017. 2, 4, 6, 7

[34] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas 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 30*, pages 5099–5108. Curran Associates, Inc., 2017. 2

[35] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 3

[36] David M. Rosen, Luca Carlone, Afonso S. Bandeira, and John J. Leonard. SE-Sync: A certifiably correct algorithm for synchronization over the Special Euclidean group. In *The Workshop on the Algorithmic Foundations of Robotics*, San Francisco, CA, December 2016. 2

[37] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, June 2001. 2

[38] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4967–4976. Curran Associates, Inc., 2017. 3, 5

[39] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Transactions on Neural Networks*, 20(1):61–80, Jan. 2009. 2

[40] Aleksandr Segal, Dirk Hhnel, and Sebastian Thrun. Generalized ICP. In Jeff Trinkle, Yoky Matsuoka, and Jos A. Castellanos, editors, *Robotics: Science and Systems*. The MIT Press, 2009. 1, 2

[41] Justin Solomon. Optimal transport on discrete domains. *CoRR*, abs/1801.07745, 2018. 3

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. 6

[43] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. SPLATNet: Sparse lattice networks for point cloud processing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2530–2539, 2018. 3

[44] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. 3

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. 2, 3, 5

[47] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. 2

[48] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems 28*, NIPS'15, pages 2692–2700, Cambridge, MA, USA, 2015. MIT Press. 2, 3

[49] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3, 5

[50] Yue Wang, Yongbin Sun, Sanjay E. Sarma Ziwei Liu, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics, to appear*, 2019. 2, 3, 4, 6, 7, 8

[51] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. 2

[52] Zhirong Wu, Shuran Song, Aditya Khosla, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, USA, June 2015. 2, 6, 7

[53] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L. Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. *CoRR*, abs/1812.03411, 2018. 3

[54] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, ICMR '16, pages 167–174, New York, NY, USA, 2016. ACM. 3

[55] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. Go-ICP: A globally optimal solution to 3D ICP point-set registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2241–2254, Nov. 2016. 2, 5, 6, 7

[56] Zi Jian Yew and Gim Hee Lee. 3DFeat-Net: Weakly supervised local 3D features for point cloud registration. In *European Conference on Computer Vision*, 2018. 2

[57] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3391–3401. Curran Associates, Inc., 2017. 2

[58] Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Relational deep reinforcement learning. *CoRR*, abs/1806.01830, 2018. 3

[59] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3DMatch: Learning local geometric descriptors from RGB-D reconstructions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[60] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018. 2

[61] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision*, pages 766–782, 2016. 5, 6, 7

[62] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 6