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.

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) [4, 37], alternates between estimating the rigid motion based...
Deep Closest Point (DCP), a learning-based method that
addresses 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
[30] and DGCNN [48] for this step); then, (2) an attention
based module combining pointer network [45, 43] 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 [50] 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 [16]. 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 trans-
formation 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 code1, to facilitate reproducibility and
future research.

2. Related Work

Traditional point cloud registration methods: ICP [4]
is the best-known algorithm for solving rigid registration
problems; it alternates between finding point cloud corre-
spondences and solving a least-squares problem to update
the alignment. ICP variants [34, 37, 5] consider issues with
the basic method, like noise, partiality, and sparsity; prob-
abilistic models [12, 13, 17] also can improve resilience to
uncertain data. ICP can be viewed as an optimization algo-
rithm searching jointly for a matching and a rigid align-ment; hence, [11] propose using the Levenberg–Marquardt algo-
rithm to optimize the objective directly, which can yield a
better solution. For more information, [29, 34] 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 in polynomial time, Go-
ICP [53] uses a branch-and-bound (BnB) method to search
the motion space SE(3). It outperforms local ICP meth-
ods 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 opti-
imization [33], convex relaxation [24], and mixed-integer
programming [19].

Learning on graphs and point sets: A broad class of
deep architectures for geometric data termed geometric deep
learning [6] includes recent methods learning on graphs
[49, 56, 10] and point clouds [30, 31, 48, 54].

The graph neural network (GNN) is introduced in [36];
similarly, [9] defines convolution on graphs (GCN) for
molecular data. [22] uses renormalization to adapt to graph
structure and applies GCN to semi-supervised learning on
graphs. MoNet [25] learns a dynamic aggregation function
based on graph structure, generalizing GNNs. Finally, graph
attention networks (GATs) [44] incorporate multi-head at-
tention into GCNs. DGCNN [48] can be regarded as graph
neural network applied to point clouds with dynamic edges.

Another branch of geometric deep learning includes Point-
Net [30] 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++
[31] applies a shared PointNet to \(k\)-nearest neighbor clusters
to learn local features. As an alternative, DGCNN [48]
explicitly recovers the graph structure in both Euclidean
space and feature space and applies graph neural networks
to the result. PCNN [2] uses an extension operator to define
convolution on point clouds, while PointCNN [23] applies
Euclidean convolution after applying a learned transforma-
tion. Finally, SPLATNet [40] 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 ma-

---

1https://github.com/WangYueFt/dcp
Machine translation, language modeling, and question answering, can be formulated as sequence-to-sequence problems (seq2seq). [42] 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 transportation [38, 28], combinatorial optimization [18], and graph matching [52]. To address this issue, in our registration pipeline, we use a related method to Pointer Networks [46], 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.

**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 \( X \) and \( Y \) to denote two point clouds, where \( X = \{ x_1, \ldots, x_i, \ldots, x_N \} \subset \mathbb{R}^3 \) and \( Y = \{ y_1, \ldots, y_j, \ldots, y_M \} \subset \mathbb{R}^3 \). 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.

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

\[
E(R_{XY}, t_{XY}) = \frac{1}{N} \sum_{i=1}^{N} \|R_{XY}x_i + t_{XY} - y_i\|^2. \tag{1}
\]

Define centroids of \( X \) and \( Y \) as

\[
\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \quad \text{and} \quad \overline{y} = \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{2}
\]

Then the cross-covariance matrix \( H \) is given by

\[
H = \sum_{i=1}^{N} (x_i - \overline{x})(y_i - \overline{y})^T. \tag{3}
\]

We can use the singular value decomposition (SVD) to decompose \( H = USV^T \). Then, the alignment minimizing \( E(\cdot, \cdot) \) in (1) is given in closed-form by

\[
R_{XY} = UV^T \quad \text{and} \quad t_{XY} = -R_{XY}\overline{x} + \overline{y}. \tag{4}
\]

Here, we take the convention that \( U, V \in SO(3) \), while \( S \) is diagonal but potentially signed; this accounts for orientation-reversing choices of \( H \). This classic orthogonal Procrustes problem assumes that the point sets are matched to each
We evaluate two possible choices of learnable embedding. We embed point clouds into a high-dimensional space using a function $E$ where the algorithm gets stuck. Our goal is to use learned modules, PointNet [30] and DGCNN [48] to compute a rigid transformation, which we will detail in section 4.

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 [30] or DGCNN [48] (§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 $X$ and $Y$ 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 [30] and DGCNN [48].

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 $F_X = \{x_1^I, x_2^I, \ldots, x_N^I\}$ and $F_Y = \{y_1^I, y_2^I, \ldots, y_N^I\}$, 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 $x_i^l$ be the embedding of point $i$ in the $l$-th layer, and let $h^l_{\theta}$ be a nonlinear function in the $l$-th layer parameterized by a shared multilayer perceptron (MLP). Then, the forward mechanism is given by

$$x_i^l = h^l_{\theta}(x_i^{l-1}).$$

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

$$x_i^l = f(\{h^l_{\theta}(x_i^{l-1}, x_j^{l-1}) \forall j \in \mathcal{N}_i\}),$$

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 $X$ and $Y$ together rather than embedding $X$ and $Y$ 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 [8], non-local neural networks [47], and relational networks [35] using attention-based models, we design a module to learn co-contextual information by capturing self-attention and conditional attention.

Take $F_X$ and $F_Y$ 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.
\[
\Phi_X = F_X + \phi(F_X, F_Y) \\
\Phi_Y = F_Y + \phi(F_Y, F_X)
\]  

(8)

Notice we treat \( \phi \) as a residual term, providing an additive change to \( F_X \) and \( F_Y \) depending on the order of its inputs. The idea here is that the map \( F_X \mapsto \Phi_X \) modifies the features associated to the points in \( \mathcal{X} \) in a fashion that is knowledgeable about the structure of \( \mathcal{Y} \): the map \( F_Y \mapsto \Phi_Y \) serves a symmetric role. We choose \( \phi \) as an asymmetric function given by a Transformer \([43,2]\), since 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 simple distance on a set of neural network weights, which must be learned during a training phase. We employ a fairly straightforward strategy for training, measuring deviation of \([R_{XY}, t_{XY}]\) from ground truth for synthetically-generated pairs of point clouds.

To backpropagate gradients through the networks, we need to differentiate the SVD. \([26]\) describes a standard means of computing this derivative; version of this calculation are included in PyTorch \([27]\) and TensorFlow \([1]\). Note we need to solve only \( 3 \times 3 \) 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 \([R_{XY}, t_{XY}]\) that aligns them to each other. The initial feature module \((\S 4.1)\) and the attention module \((\S 4.2)\) 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 deviation of \([R_{XY}, t_{XY}]\) 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:

\[
\text{Loss} = \| R_{XY}^T R_{XY}^g - I \|^2 + \| t_{XY} - t_{XY}^g \|^2 + \lambda \| \theta \|^2
\]

(11)

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

5. Experiments

We compare our models to ICP, Go-ICP \([53]\), Fast Global Registration (FGR) \([57]\), and the recently-proposed PointNetLK deep learning method \([16]\). We denote our model without attention \((\S 4.2)\) 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 [58]. For PointNetLK, we adapt the code partially released by the authors. Notice that FGR [57] uses additional geometric features.

The architecture of DCP is shown in Figure 2. We use 5 EdgeConv (denoted as DGCNN [48]) 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 [3] without Dropout [39]. Adam [21] is used to optimize the network parameters, with initial learning rate 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 $\Phi_X = F_X$ and $\Phi_Y = F_Y$.

We experiment on the ModelNet40 [50] 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 [30], 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 $X$. 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 $[-0.5, 0.5]$. $X$ and a transformation of $X$ by the rigid motion are used as input to the network, which is evaluated against the known ground truth using (11).

Table 1 evaluates 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. Figure 4 shows results of DCP-v2 on some objects.

5.2. ModelNet40: Category Split

To test the generalizability of different models, we split ModelNet40 evenly by category into training and testing sets.

| Model | MSE(R) | RMSE(R) | MAE(R) | MSE(t) | RMSE(t) | MAE(t) |
|-------|--------|---------|--------|--------|---------|--------|
| ICP   | 894.97139 | 29.914835 | 23.544817 | 0.084643 | 0.290935 | 0.248755 |
| Go-ICP [53] | 140.477325 | 11.852313 | 2.588463 | 0.00659 | 0.025665 | 0.007092 |
| FGR [57] | 87.61491 | 9.367277 | 1.992920 | 0.00194 | 0.013939 | 0.002369 |
| PointNetLK [16] | 227.470331 | 15.955374 | 4.223094 | 0.00487 | 0.020965 | 0.005484 |

| DCP-v1 (ours) | 6.480572 | 0.549569 | 1.505548 | 0.000663 | 0.001763 | 0.001451 |
| DCP-v2 (ours) | 1.307329 | 1.143388 | 0.770573 | 0.000063 | 0.001786 | 0.001195 |

Table 1. ModelNet40: Test on unseen point clouds

| Model | MSE(R) | RMSE(R) | MAE(R) | MSE(t) | RMSE(t) | MAE(t) |
|-------|--------|---------|--------|--------|---------|--------|
| ICP   | 892.601135 | 29.876431 | 23.626110 | 0.086005 | 0.293266 | 0.251916 |
| Go-ICP [53] | 192.258636 | 13.865736 | 2.914169 | 0.000491 | 0.022154 | 0.006219 |
| FGR [57] | 97.002747 | 9.584899 | 1.445460 | 0.000162 | 0.015303 | 0.002211 |
| PointNetLK [16] | 306.132975 | 17.302133 | 5.280545 | 0.000784 | 0.028007 | 0.007263 |

| DCP-v1 (ours) | 19.201385 | 4.819398 | 2.680408 | 0.000025 | 0.04950 | 0.000597 |
| DCP-v2 (ours) | 9.923701 | 3.150191 | 2.007210 | 0.000025 | 0.005039 | 0.003703 |

Table 2. ModelNet40: Test on unseen categories

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 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 $\mathcal{N}(0, 0.01)$, clip the noise to $[-0.05, 0.05]$, and add it to $X$ 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.

#### 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 [48], PointNet [30] learns a global descriptor of the whole shape while DGCNN [48] 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 [50], using DCP-v1 and DCP-v2. Table 5. Models perform consistently better with DGCNN that 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

[30] 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 [30].

### 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 [48] 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 prob-
lems 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 reward to learn a policy that controls when to stop iterating. Finally, we can incorporate our method into larger pipelines to enable high-accuracy Simultaneous Localization and Mapping (SLAM) or Structure from Motion (SFM).

8. Acknowledgement

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, and from the Skoltech-MIT Next Generation Program. Yue Wang wants to thank David Palmer for helpful discussion.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 5
[2] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. ACM Trans. Graph., 37(4):71:1–71:12, July 2018. 2
[3] L. J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. CoRR, abs/1607.06450, 2016. 6
[4] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. IEEE Trans. Pattern Anal. Mach. Intell., 14(2):239–
