3D Shape Completion and Canonical Pose Estimation with Structured Neural Networks

Wentao Yuan
May, 2019

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Martial Hebert
David Held
Brian Okorn

CMU-RI-TR-19-22

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

To my parents.

Abstract

3D point cloud is an efficient and flexible representation of 3D structures and the raw output of many 3D sensors. The ability to learn useful representations from 3D point cloud data is thus of great value to autonomous agents that perceive and interact with the surrounding 3D environment. Recently, neural networks operating on point clouds have shown superior performance on various 3D understanding tasks, thanks to their power to extract task-specific semantic features directly from points. In this thesis, we study how to incorporate geometric and algorithmic structures into the design of neural networks in order to achieve more effective and efficient learning from raw point clouds.

In particular, we investigate two problems – 3D shape completion and canonical pose estimation – that address two essential characteristics of 3D data in the wild: incompleteness and misalignment. We show that the structured neural networks we propose outperform alternative approaches that do not incorporate structural priors on synthetic benchmarks and demonstrate the potential of our networks to operate on challenging real world data such as LiDAR scans collected from an autonomous vehicle.

Acknowledgments

My two years at CMU are invaluable. Two years ago, I could not imagine that I would be sharing my original research with researchers from around the world or writing up a sixty-page thesis. All of these would not be possible without the help of many fabulous people I met at CMU.

I am very fortunate to have Prof. Martial Hebert as my advisor, who opened my eyes to the emerging field of 3D computer vision. His experience and insights are my source of inspiration and his conscientious attitude towards research has deeply influenced me and encouraged me to tackle to frontiers of human knowledge.

I am grateful to Tejas Khot for being an amazing labmate. The countless late-night discussions we had in the lab has made the lonely journey of research much more enjoyable.

I would also like to thank Christoph Mertz for graciously providing his labspace and funding to support my research, and David Held, Leonid Keselman, Chen-Hsuan Lin, Brian Okorn, Chaoyang Wang, Rui Zhu for helpful discussions and suggestions.

Now, let the new journey begin.

Contents

Chapter 1

Introduction

1.1 Motivation

3D point cloud is the raw output of many 3D sensors and a widely used representation for 3D structures in 3D reconstruction [2], tracking [27] and localization [35]. Due to its efficiency and flexibility, there is a growing interest in using point clouds directly for high level understanding tasks, skipping the need for meshing or other post-processing. These tasks require an understanding of the semantic concept represented by the points. On other modalities like images, deep neural networks [28] have proven to be a powerful model for extracting semantic information from raw sensor data, and have gradually replaced hand-crafted features. A similar trend is happening on point clouds. With the introduction of deep learning architectures that operate on point clouds [38], it is possible to train powerful feature extractors that outperform traditional geometric descriptors on tasks such as shape classification and object part segmentation.

However, existing benchmark datasets [11, 55] that are used to evaluate performance on these tasks make two simplifying assumptions: first, the 3D shapes are sampled from CAD models with complete geometry; second, the shapes are aligned in a canonical coordinate system defined by human modelers. These assumptions are rarely met in real world scenarios. First, due to occlusions and limitations of sensors, real world 3D scans often contain missing regions. Second, real world 3D data are often obtained in the sensors coordinates, which do not align with the canonical

coordinates of object models. In other words, real 3D point cloud data are *partial* and *misaligned*.

This thesis provides a step towards learning from partial, misaligned 3D data by proposing two structured neural networks models that operate on raw point clouds: Point Completion Network (PCN) and Iterative Transformer Networks (IT-Net). We will show how these networks address the problems of 3D shape completion and canonical pose estimation and how they exploit structural priors that lead to more effective learning.

1.2 Overview of Contributions

This thesis offers the following contributions to the research community:

1. A learning-based shape completion method that operates directly on 3D point clouds without intermediate voxelization and is end-to-end trainable on pairs of partial and complete shapes without additional annotations;
2. A novel multistage encoder-decoder architecture that generates a dense point cloud in a coarse-to-fine fashion, drawing upon patch-based representation of 3D surfaces;
3. A demonstration of how shape completion can aid downstream tasks such as point cloud registration;
4. A new 3D transformer network estimates rigid transformations from point clouds in an iterative fashion;
5. An anytime[1] pose estimator that can also be trained jointly with point-based shape classification and segmentation networks to improve their performance;
6. A dataset for pose estimation, shape classification and part segmentation consisting of partial, unaligned point clouds.

[1]The result of an *anytime* predictor can be gradually refined until the test-time computational budget is depleted.

1.3 Thesis Outline

This thesis will be organized as follows. In Chapter 2, we introduce Point Completion Network [63], a learning-based shape completion method on 3D point clouds, and show improved completion results over existing completion methods, robustness against noise and sparsity and generalization to real-world data. In Chapter 3, we propose Iterative Transformer Network [62], a differentiable network module that predicts rigid transformations from point clouds, and demonstrate its efficacy on canonical pose estimation, shape classification and part segmentation. In Chapter 4, we provide concluding remarks and highlight some important open problems for future work.

Chapter 2

3D Shape Completion

Shape completion, the problem of estimating the complete geometry of objects from partial observations, lies at the core of many vision and robotics applications. In this chapter, we introduce Point Completion Network (PCN), a novel learning-based approach for shape completion. PCN takes a partial point cloud as input and generates a dense, complete point cloud in a single forward pass with a structured multi-stage decoder. Unlike existing learning-based shape completion methods, PCN directly operates on raw point clouds without any discretization. Our experiments show that PCN produces dense, complete point clouds with realistic structures in the missing regions in a fraction of a second. In addition, trained on pairs of partial and complete point clouds from a synthetic dataset, our model is able to generalize to challenging real world 3D data such as car point clouds extracted from sparse LiDAR scans in the KITTI dataset [17].

2.1 Motivation

With the rapid development of 3D sensors such as depth cameras and LiDARs, the availability of 3D data has increased significantly over the years. However, due to occlusion, limited sensor resolution and sensor failures, real-world 3D scans are often contain large missing regions, causing loss in geometric and semantic information. For example, the cars in the LiDAR scan shown in Figure 2.1 are hardly recognizable due to the sparsity of data points and missing regions caused by occlusion.

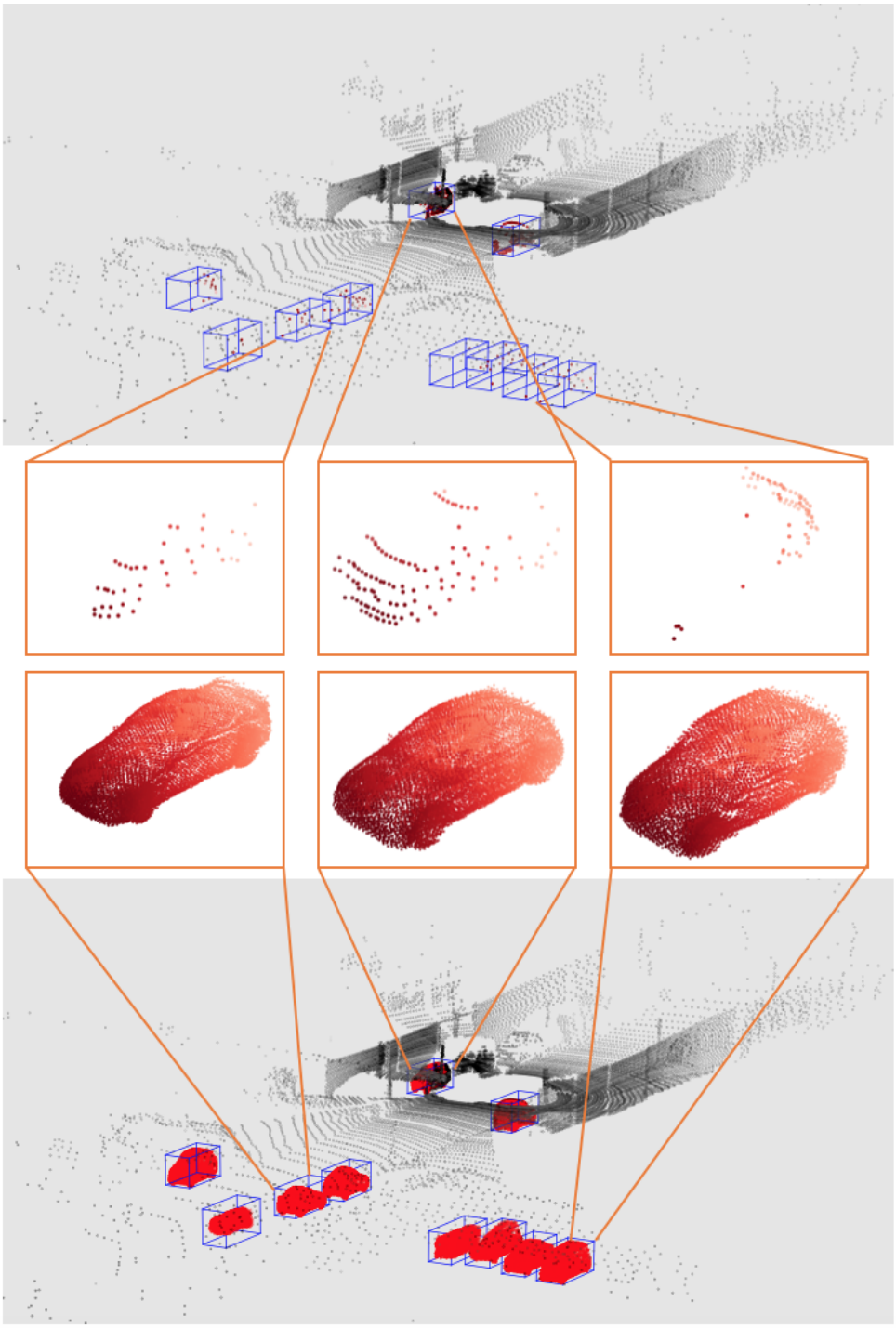


Figure 2.1: **(Top) Raw LiDAR scan from KITTI** [17]. Note how the cars are barely recognizable due to incompleteness of the data. **(Bottom) Completed scan generated by PCN** on individual car point clouds segmented from the scene.

Psychological experiments [18, 40] show that completion of occluded structures is performed frequently by humans, where we leverage prior experience to infer unobserved parts of objects. Recently, several large-scale 3D shape repositories [11, 55] containing CAD models of common objects have been constructed. Taking inspiration from human vision, a number of recent works [14, 46] utilize these repositories to train deep neural networks that encode implicit shape priors. Once trained, these networks are able to use the learned shape prior to predict complete geometry from partial views in an extremely efficient fashion, mimicking human vision.

However, in order to leverage the power of convolutional networks, existing learning-based shape completion methods voxelize the 3D data into occupancy grids or distance fields, which leads to several disadvantages. First, the cubically growing memory cost of 3D voxel grids limits the output resolution. Second, detailed geometry of the shapes is often lost in the process of discretization.

To address these shortcomings, we propose a learning-based shape completion method that use point clouds as the representation for 3D geometry. The point cloud representation prevents the high memory cost and loss of geometric details caused by voxelization and allows our method to generate higher quality completions.

The key to our approach is the design of a neural network that consumes and generates 3D point clouds, which involves a number of challenges. First, a point cloud is an unordered set, which means permutations of the points do not change the geometry they represent. This necessitates the design of a feature extractor and a loss function that are permutation invariant. Second, there is no clear definition of local neighbourhoods in point clouds, making it difficult to apply any convolutional operation. Lastly, existing point cloud generation networks only generate a small set of points, which is not sufficient to capture enough detail in the output shape. Our proposed model tackles these challenges by combining a permutation invariant, non-convolutional feature extractor and a coarse-to-fine point set generator in a single network that is trained end-to-end.

The rest of this chapter will be organized as follows. In Section 2.2, we describe our formulation of the shape completion problem. In Section 2.3, we illustrate details of our proposed encoder-decoder network. In Section 2.4, we show extensive experiment results including improved completion quality over strong baselines, robustness against noise and sparsity, generalization to real-world data and application of completion to

downstream tasks. In Section 2.5, we cover previous works related to our approach. In Section 2.6, we discuss failure modes of our method and possible avenues for improvement.

2.2 Problem Statement

Let X be a set of 3D points lying on the observed surfaces of an object obtained by a single observation or a sequence of observations from a 3D sensor. Let Y be a dense set of 3D points uniformly sampled from the observed and unobserved surfaces of the object. We define the shape completion problem as predicting Y given X. Note that under this formulation, X is not necessarily a subset of Y and there is no explicit correspondence between points in X and points in Y, because they are independently sampled from the underlying object surfaces.

We tackle this problem using supervised learning. Leveraging a large-scale synthetic dataset where samples of X and Y can be easily acquired, we train a neural network to predict Y directly from X. The network is generic across multiple object categories and does not assume anything about the structure of underlying objects such as symmetry or planarity.

2.3 Point Completion Network

In this section, we describe the architecture of our proposed model, the Point Completion Network (PCN). As shown in Figure 2.2, PCN is an encoder-decoder network. The encoder takes the input point cloud X and outputs a k-dimensional feature vector. The decoder takes this feature vector and produces a coarse output point cloud Y_{coarse} and a detailed output point cloud Y_{detail}. The loss function L is computed between the ground truth point cloud Y_{gt} and the outputs of the decoder, which is used to train the entire network through backpropagation.

Note that, unlike an auto-encoder, we do not explicitly enforce the network to retain the input points in its output. Instead, it learns a projection from the space of partial observations to the space of complete shapes. In what follows, we describe the specific design of the encoder, decoder and the loss function used.

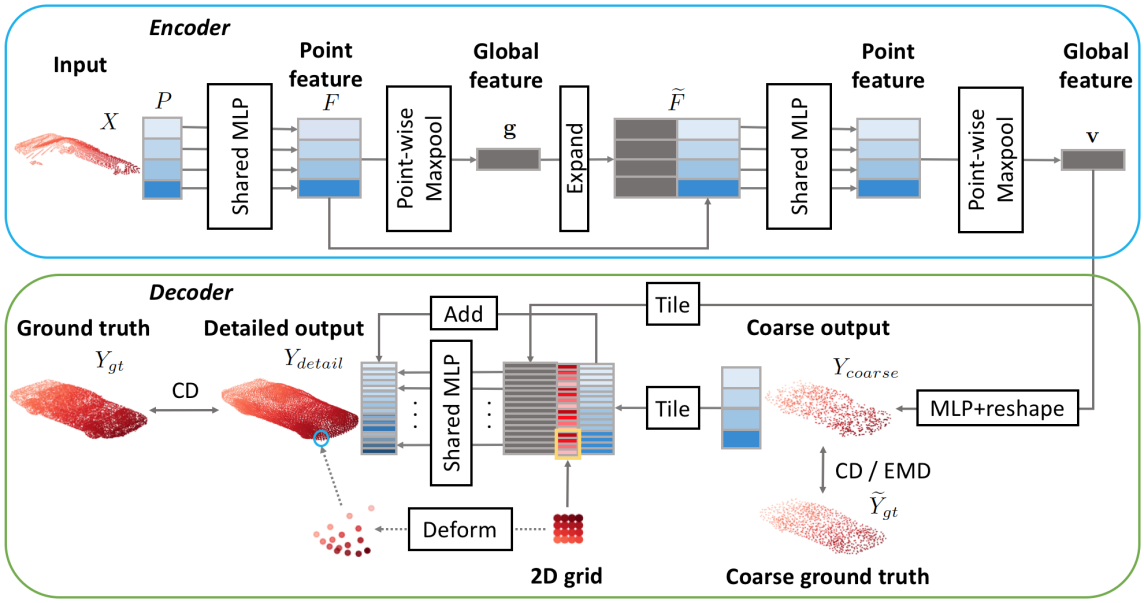


Figure 2.2: **PCN Architecture**. The encoder abstracts the input point cloud X as a feature vector $\mathbf{v}$. The decoder uses $\mathbf{v}$ to first generate a coarse output Y_{coarse} followed by a detailed output Y_{detail}. Each colored rectangle denotes a matrix row. Same color indicates same content.

2.3.1 Point Feature Encoding

The encoder is in charge of summarizing the geometric information in the input point cloud as a feature vector $\mathbf{v} \in \mathbb{R}^k$ where $k = 1024$. Specifically, the encoder consists of two stacked PointNet (PN) layers. The first layer consumes m input points represented as an $m \times 3$ matrix P where each row is the 3D coordinate of a point $\mathbf{p}_i = (x, y, z)$ (note that out network can handle input of various sizes). A shared multi-layer perceptron (MLP) consisting of two linear layers with ReLU activation is used to transform each $\mathbf{p}_i$ into a point feature vector $\mathbf{f}_i$. This gives us a feature matrix F whose rows are the learned point features $\mathbf{f}_i$. Then, a point-wise maxpooling is performed on F to obtain a k-dimensional global feature $\mathbf{g}$, where $\mathbf{g}_j = \max_{i=1,\dots,m}\{F_{ij}\}$ for $j = 1, \dots, k$. The second PN layer takes F and $\mathbf{g}$ as input. It first concatenates $\mathbf{g}$ to each $\mathbf{f}_i$ to obtain an augmented point feature matrix $\widetilde{F}$ whose rows are the concatenated feature vectors $[\mathbf{f}_i\ \mathbf{g}]$. Then, $\widetilde{F}$ is passed through another shared MLP and point-wise max pooling similar to the ones in the first layer, which gives the final feature vector $\mathbf{v}$.

There are three key properties of our proposed encoder. First, it is invariant to permutations of the input points. This follows from the fact that every operation in our proposed encoder is either executed in parallel with respect to each point (e.g. the shared MLP) or a symmetric operation (e.g. max pooling). This property is essential in learning useful features from point clouds because changing the order of points in a point cloud does not change the underlying surfaces the points are sampled from. Second, our encoder is tolerant to noise in the input data. As proven in [38], small disturbances of the input points will not affect the learned global feature vector. This is especially important for our model to generalize to real-world data which can be noisy. Third, our encoder is able to combine local and global geometry via the chaining of PN layers. With a single PN layer, the point feature vector $\mathbf{f}_i$ produced by the shared MLP depends solely on the point coordinates, which implies that all $\mathbf{f}_i$ lie on a manifold with only 3 degrees of freedom. By passing the concatenated global feature and point feature through another PN layer which blends the local and global information, the network can learn much more complex point features. We found that this helps the network achieve better performance in practice.

2.3.2 Multistage Point Generation

The decoder is responsible for generating the output point cloud from the feature vector $\mathbf{v}$. Our proposed decoder combines the advantages of the fully-connected decoder [1] and the folding-based decoder [59] in a multistage point generation pipeline. In our experiments, we show that our decoder outperforms either the fully-connected or the folding-based decoder on its own.

Our key observation is that the fully-connected decoder is good at predicting a sparse set of points which represents the global geometry of a shape. Meanwhile, the folding-based decoder is good at approximating a smooth surface which represents the local geometry of a shape. Thus, we divide the generation of the output point cloud into two stages. In the first stage, a coarse output Y_{coarse} of s points is generated by passing $\mathbf{v}$ through a fully-connected network with $3s$ output units and reshaping the output into a $s \times 3$ matrix. In the second stage, for each point $\mathbf{q}_i$ in Y_{coarse}, a patch of $t = u^2$ points is generated in the local coordinates centered at $\mathbf{q}_i$ via the local folding operation (see Section 2.3.3), and transformed into the global coordinates by

adding $\mathbf{q}_i$ to the output. Combining all s patches gives the detailed output Y_{detail} consisting of $n = st$ points. This multistage process allows our network to generate a dense output point cloud with fewer parameters than fully-connected decoder (see Table 2.1) and more flexibility than folding-based decoder.

2.3.3 Local Folding Operation

As shown in Figure 2.3, the local folding operation takes a point $\mathbf{q}_i$ in the coarse output Y_{coarse} and the k-dimensional global feature $\mathbf{v}$ as inputs, and generates a patch of $t = u^2$ points in local coordinates centered at $\mathbf{q}_i$ by deforming a $u \times u$ grid. It first takes points on a zero-centered $u \times u$ grid with side length r (r controls the scale of the output patch) and organize their coordinates into a $t \times 2$ matrix G. Then, it concatenates each row of G with the coordinates of the center point $\mathbf{q}_i$ and the global feature vector $\mathbf{v}$, and passes the resulting matrix through a shared MLP that generates a $t \times 3$ matrix Q, i.e. the local patch centered at $\mathbf{q}_i$. This shared MLP can be interpreted as a non-linear transformation that deforms the 2D grid into a smooth 2D manifold in 3D space. Note that the same MLP is used in the local patch generation for each $\mathbf{q}_i$ so the number of parameters in the local folding operation does not grow with the output size.

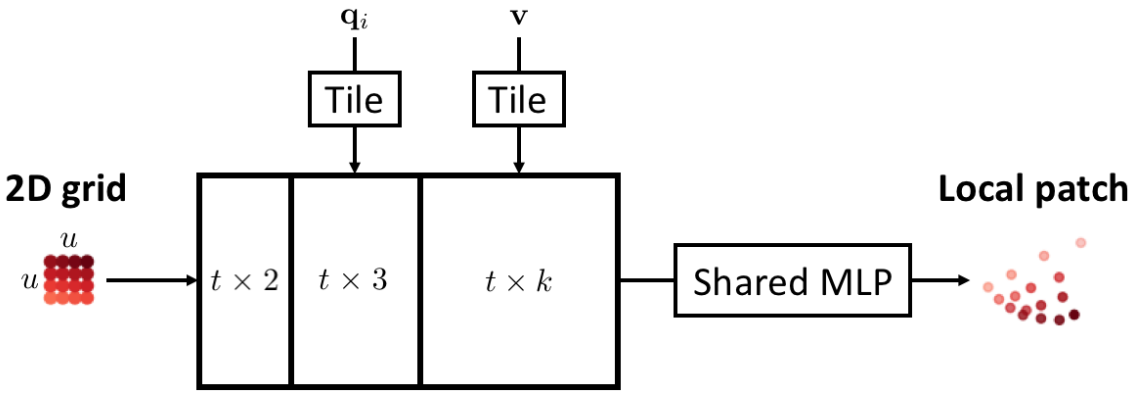


Figure 2.3: The local folding operation

2.3.4 Loss Function

The loss function measures the difference between the output point cloud and the ground truth point cloud. Since both point clouds are unordered, the loss needs to be invariant to permutations of the points. Two candidates of permutation invariant functions are introduced by [16] – Chamfer Distance (CD) and Earth Mover's Distance (EMD).

$$CD(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2 + \frac{1}{|S_2|} \sum_{y \in S_2} \min_{x \in S_1} \|y - x\|_2 \tag{2.1}$$

CD (2.1) calculates the average closest point distance between the output point cloud S_1 and the ground truth point cloud S_2. We use the symmetric version of CD where the first term forces output points to lie close to ground truth points and the second term ensures the ground truth point cloud is covered by the output point cloud. Note that S_1 and S_2 need not be the same size to calculate CD.

$$EMD(S_1, S_2) = \min_{\phi: S_1 \to S_2} \frac{1}{|S_1|} \sum_{x \in S_1} \|x - \phi(x)\|_2 \tag{2.2}$$

EMD (2.2) finds a bijection $\phi : S_1 \to S_2$ which minimizes the average distance between corresponding points. In practice, finding the optimal ϕ is too expensive, so we use an iterative $(1 + \epsilon)$ approximation scheme [6]. Unlike CD, EMD requires S_1 and S_2 to be the same size.

$$L(Y_{coarse}, Y_{detail}, Y_{gt}) = d_1(Y_{coarse}, \widetilde{Y}_{gt}) + \alpha \; d_2(Y_{detail}, Y_{gt}) \tag{2.3}$$

Our proposed loss function (2.3) consists of two terms, d_1 and d_2, weighted by hyperparameter α. The first term is the distance between the coarse output Y_{coarse} and the subsampled ground truth $\widetilde{Y}_{gt}$ which has the same size as Y_{coarse}. The second term is the distance between the detailed output Y_{detail} and the full ground truth Y_{gt}.

In our experiments, we use both CD and EMD for d_1 but only CD for d_2. This is because the $O(n^2)$ complexity of the EMD approximation scheme makes it too expensive to compute during training when n is large, while CD can be computed with $O(n \log n)$ complexity using efficient data structure for nearest neighbour search such as KDTree.

2.4 Experimental Evaluation

In this section, we present experimental results of our point-based shape completion method on synthetic as well as real-world data. First, we describe how we generate a large-scale, multi-category dataset to train our model in Section 2.4.1. Next, we provide detailed ablation studies and show superior performance of our method on synthetic point clouds from ShapeNet [11] in Section 2.4.2. Then, we demonstrate completion results on real-world point clouds from LiDAR scans in the KITTI dataset [17] in Section 2.4.3. Finally, we show how our completion results can help downstream tasks such as point cloud registration in Section 2.4.4.

2.4.1 Data Generation and Model Training

To train our completion model, we created a large-scale, multi-category dataset containing pairs of partial and complete point clouds (X, Y) from synthetic CAD models in ShapeNet [11]. Specifically, we take 30,974 models from 8 categories: airplane, cabinet, car, chair, lamp, sofa, table, vessel. The complete point clouds are created by sampling 16384 points uniformly on the mesh surfaces and the partial point clouds are generated by back-projecting 2.5D depth images into 3D. We use back-projected depth images for partial inputs instead of subsets of the complete point cloud in order to bring the input distribution closer to real-world sensor data. For each model, 8 partial point clouds are generated from 8 randomly distributed viewpoints. Note that the partial point clouds can have different sizes.

We choose to use a synthetic dataset to generate training data because it contains complete, detailed 3D models of objects that are not available in real-world datasets. Despite the fact that recent datasets such as ScanNet [13] or S3DIS [3] have very high quality 3D reconstructions, these reconstructions have missing regions due to the limitations of the scanner's view, and thus are not good enough to use as ground truth for our model.

We reserve 100 models for validation and 150 models for testing. The rest is used for training. All our models are trained using the Adam [26] optimizer with an initial learning rate of 0.0001 for 50 epochs and a batch size of 32. The learning rate is decayed by 0.7 every 50K iterations.

2.4.2 Completion Results on ShapeNet

In this section, we compare our method against several strong baselines, including a representative volumetric network and modified versions of our model, on synthetic point clouds from ShapeNet [11]. We also test the generalizability of these methods to novel shapes and the robustness of our model against occlusion and noise.

Baselines Previous point-based completion methods either assume more complete inputs than we have [23] or prior knowledge of the shape such as semantic class, symmetry and part segmentation [50], and thus are not directly comparable to our method. Here, we compare our model against four strong baselines which, like our method, work on objects from multiple categories with different levels of incompleteness.

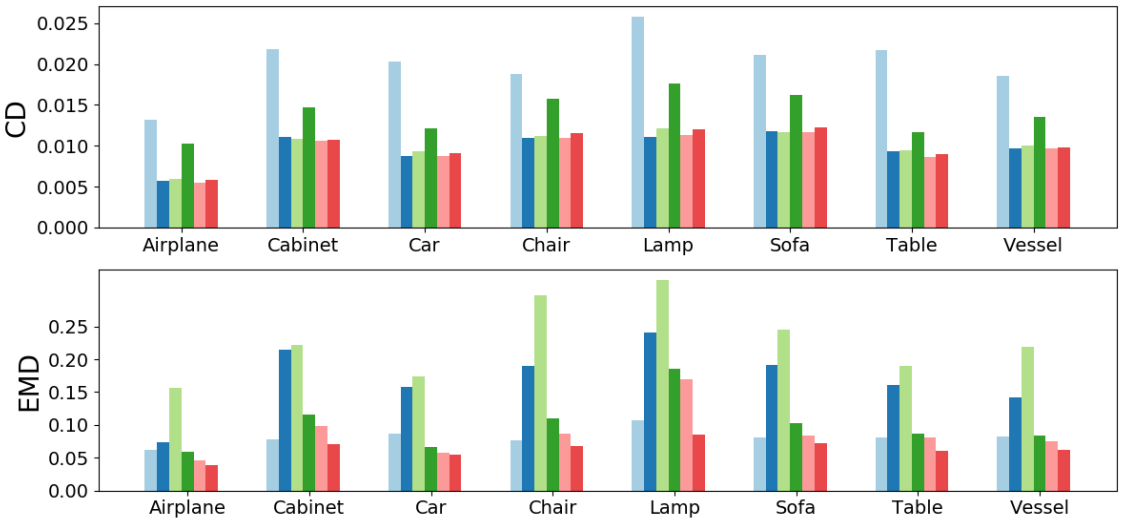
1) **3D-EPN** [14]: This is a representative of the class of volumetric completion methods that is also trained end-to-end on large synthetic dataset. To compare the distance field outputs of 3D-EPN with the point cloud outputs of PCN, we convert the distance fields into point clouds by extracting the isosurface at a small value d and uniformly sampling 16384 points on the resulting mesh. To ensure fair comparison, we also convert the point cloud outputs of PCN into distance fields by calculating the distance from grid centers to the closest point in the output.

2) **FC**: This is a network that uses the same encoder as PCN but the decoder is a 3-layer fully-connected network which directly outputs the coordinates of 16384 points.

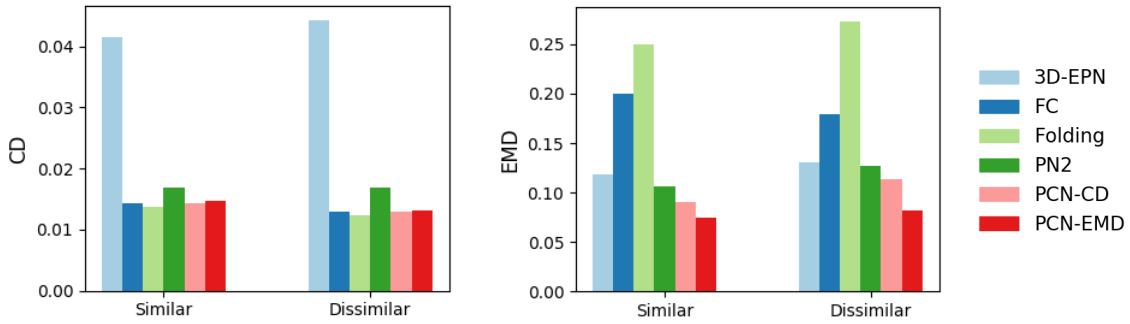
3) **Folding**: This is a network that also uses the same encoder as PCN but the decoder is purely folding-based [59], which deforms a 128-by-128 2D grid into a 3D point cloud.

4) **PN2**: This is a network that uses the same decoder as our proposed model but the encoder is PointNet++ [39].

We provide two versions of our model for comparison, PCN-CD and PCN-EMD. The number of points in the coarse and detailed outputs are $s = 1024$ and $n = 16384$ respectively. For the loss on coarse output, PCN-CD uses CD and PCN-EMD uses EMD. Note that both models use CD for the loss on detailed output due to the computational complexity of EMD.



(a) Results on trained categories



(b) Results on novel categories

Figure 2.4: **Quantitative comparison on ShapeNet**. (a) shows results for test instances from the same categories used in training. (b) shows results for test instances from categories not included in training, which are divided into similar (bus, bed, bookshelf, bench) and dissimilar (guitar, motorbike, skateboard, pistol). For both CD (top) and EMD (below), lower is better.

Test Set We created two test sets: one consists of 150 reserved shapes from the 8 object categories on which the models are trained; the other consists of 150 models from 8 novel categories that are not in the training set. We divide the novel categories into two groups: one that is visually similar to the training categories – bed, bench, bookshelf and bus, and another that is visually dissimilar to the training categories – guitar, motorbike, pistol and skateboard. The quantitative comparisons are shown in Figure 2.4 and some qualitative examples are shown in Figure 2.5.

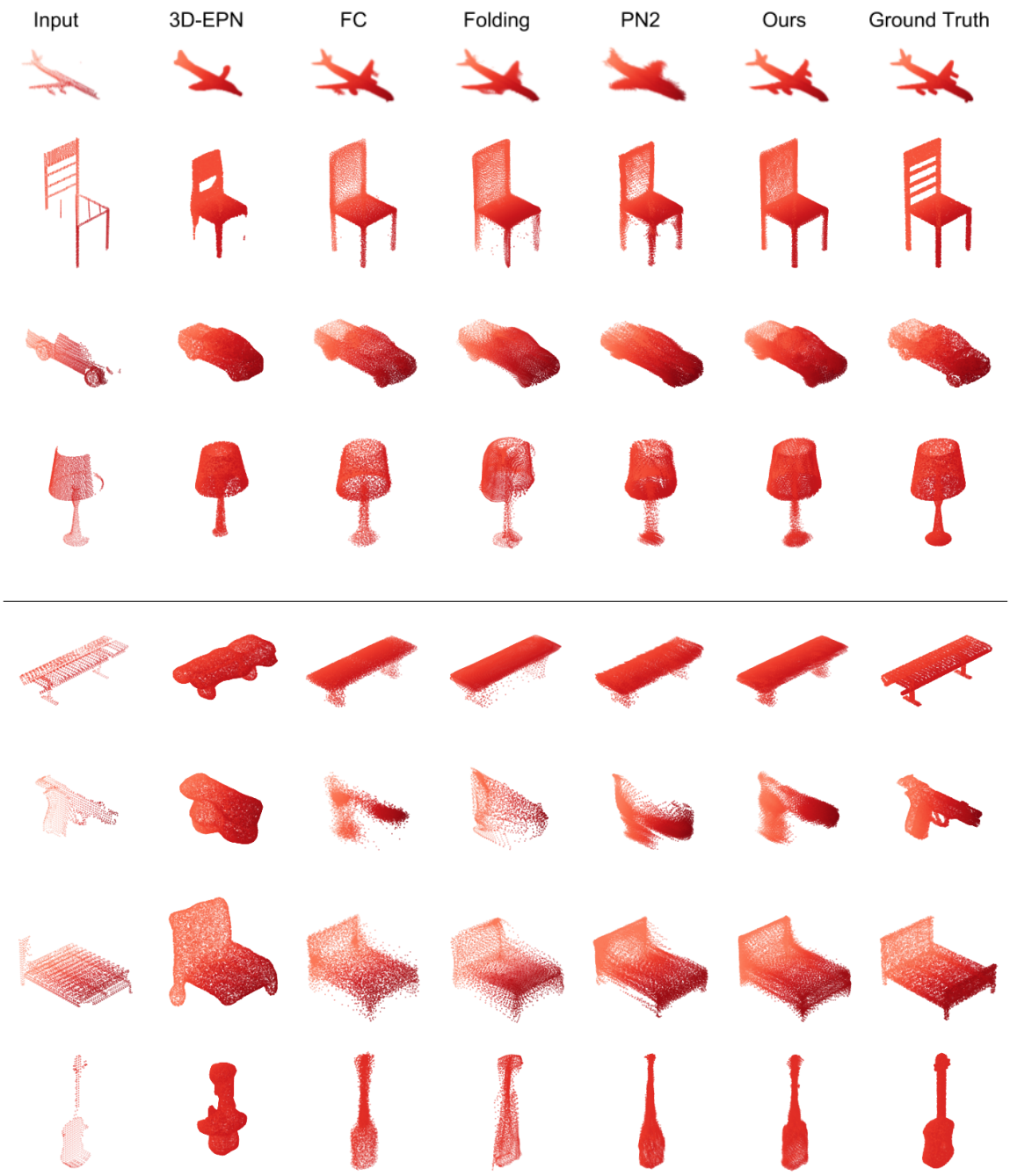


Figure 2.5: **Qualitative completion results on ShapeNet**. Top four rows are results on categories used during training. Bottom four rows are results on categories not seen during training.

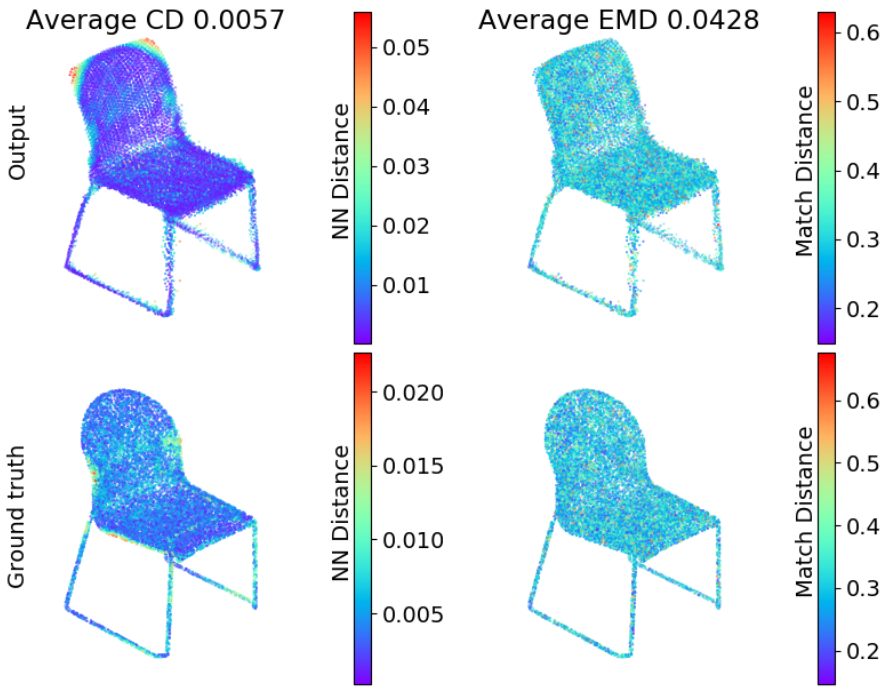
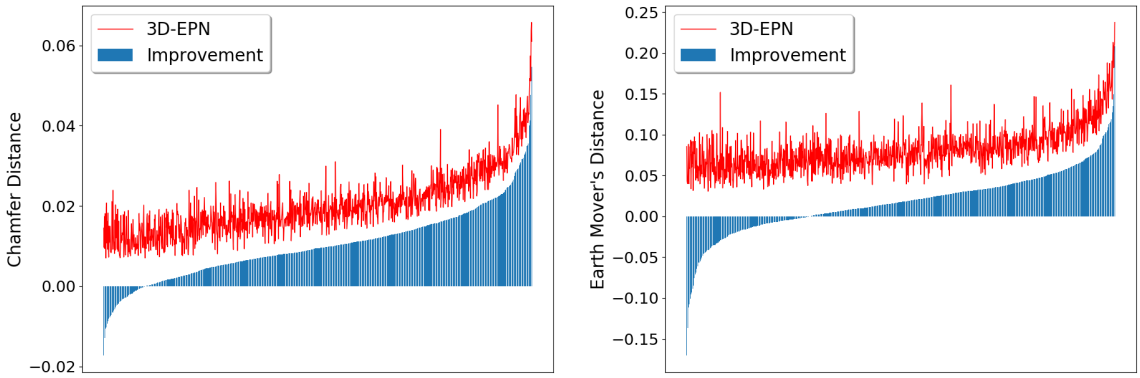


Figure 2.6: **Illustration of CD (left) and EMD (right)**. The top row shows the output of our model and the bottom row shows the ground truth. The points on the left are colored by their distance to the closest point in the other point cloud (nearest neighbor (NN) distance). The points on the right are colored by their distance to the corresponding point under the optimal bijection (match distance). Average CD is the mean of the NN distances and average EMD is the mean of the match distances.

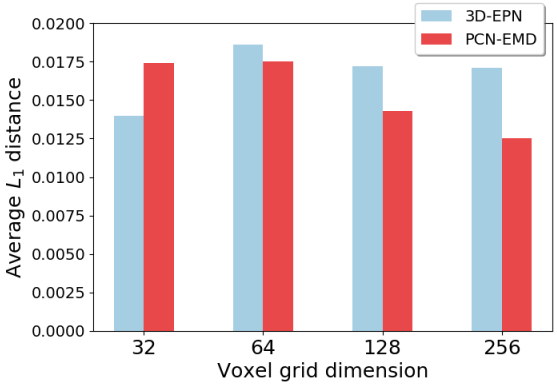
Metrics The metrics we use on point clouds are CD and EMD between the output and ground truth point clouds, as defined in 2.3.4. An illustration of the difference between the two metrics is shown in Figure 2.6. We can see that CD is high where the global structure is different, e.g. around the corners of the chair back. On the other hand, EMD is more evenly distributed, as it penalizes density difference between the two point clouds. Note that on average, EMD is much higher than CD. This is because EMD requires one-to-one correspondences between the points, whereas the point correspondences used by CD can be one-to-many.

The metric we use on distance fields is the L_1 distance between the output and ground truth distance fields, same as in [13]. To have comparable numbers across different dimensions, we convert the error from the voxel distance to distance in the model's metric space.



(a) Chamfer Distance (CD)

(b) Earth Mover's Distance (EMD)



(c) L_1 distance

Figure 2.7: **Comparison between PCN-EMD and 3D-EPN**. (a) and (b) shows comparison of point cloud outputs. The x-axis represents different object instances. The height of the blue bar indicates the amount of improvement of PCN-EMD over 3D-EPN. The red curve is the error of 3D-EPN and the difference between the red curve and the blue bar is the error of PCN-EMD. PCN-EMD improves on the majority of instances. (c) shows comparison of distance field outputs. On the y-axis is the average L_1 distance on occluded voxels between output and ground truth distance fields. PCN-EMD achieves lower L_1 distance on higher resolutions.

Comparison to Volumetric Method It can be seen that our method outperforms 3D-EPN by a large margin on both CD and EMD. To better interpret the numbers, in Figures 2.7a, 2.7b, we show the amount of improvement of our completion results over that of 3D-EPN on CD and EMD for each instance in the test set. The results of our method improve on the majority of instances. Further, they improve the most on examples where the error of 3D-EPN is high, indicating its ability to handle challenging cases where previous methods fail.

In Figure 2.7c, we show that our method achieves lower L_1 distance when its outputs are converted to a distance field. Moreover, the improvement of our method over 3D-EPN is more significant at higher resolutions.

Decoder Comparison The results in Figure 2.4 show how our proposed decoder compares with existing decoder designs. Our multistage design which combines the advantages of fully-connected and folding-based decoders outperforms either design on its own. From the qualitative results, we observe that the fully-connected decoder does not have any constraints on the local density of the output points, and thus the output points are often over-concentrated in areas such as table top, which results in high EMD. On the other hand, the folding-based decoder often produces points that are floating in space and not consistent with the global geometry of the ground truth shape, which results in high CD. This is because the shapes in our dataset contain many concavities and sharp edges, which makes globally folding a 2D plane into a 3D shape very challenging. FoldingNet [59] addresses this by chaining two folding operations. However, by only doing the folding operation locally, our decoder is able to achieve better performance with only one folding operation.

Encoder Comparison Another pair of comparison shown in Figure 2.4 is between the stacked PN and PN2 [39] as the encoder. PN2 is a representative of the class of hierarchical feature extraction networks that aggregate local information before global pooling. Our results show that it is outperformed by our stacked PN encoder which uses only global pooling. We believe this is because local pooling is less stable than global pooling due to suboptimality in the selection of local neighbourhoods for the partial data we are dealing with. Thus, we argue that stacking PN layers instead of doing local pooling is a better way of mixing local and global information.

Generalizability to Novel Objects As shown in Figure 2.4b, our method outperforms all baselines on object from novel categories. More importantly, our model's performance is not significantly affected even on visually dissimilar categories (e.g. the pistol in Figure 2.5). This shows that the shape prior learned by our model is general across various object categories.

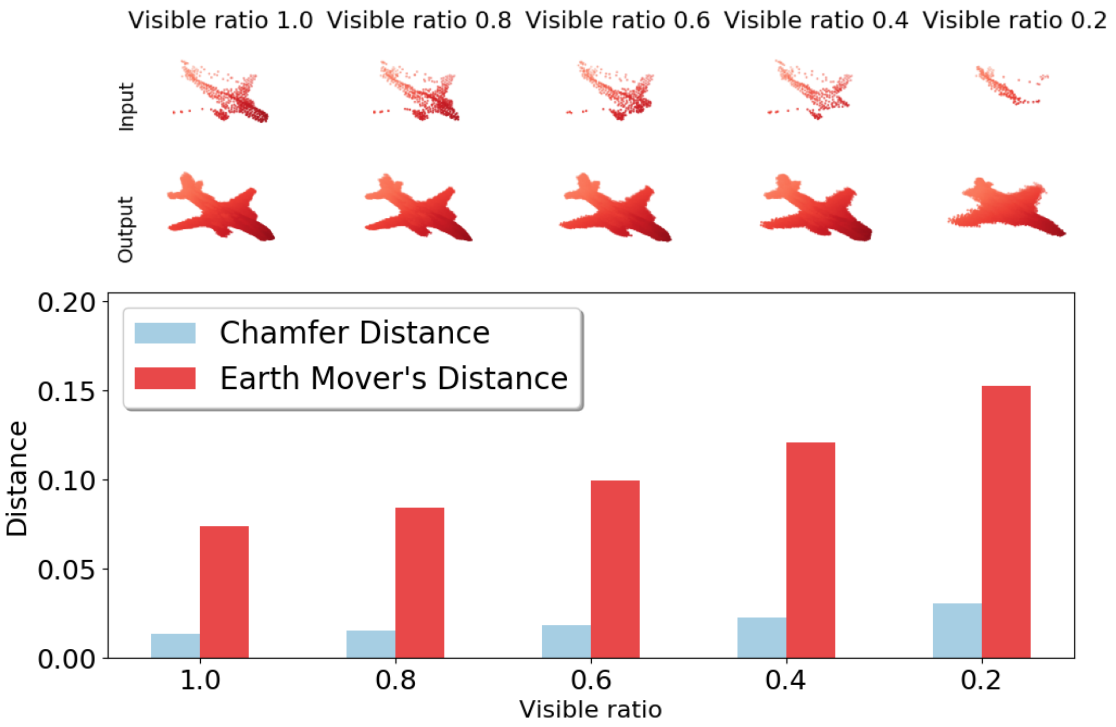


Figure 2.8: Qualitative (top) and quantitative (bottom) results on noisy inputs with different level of visibility.

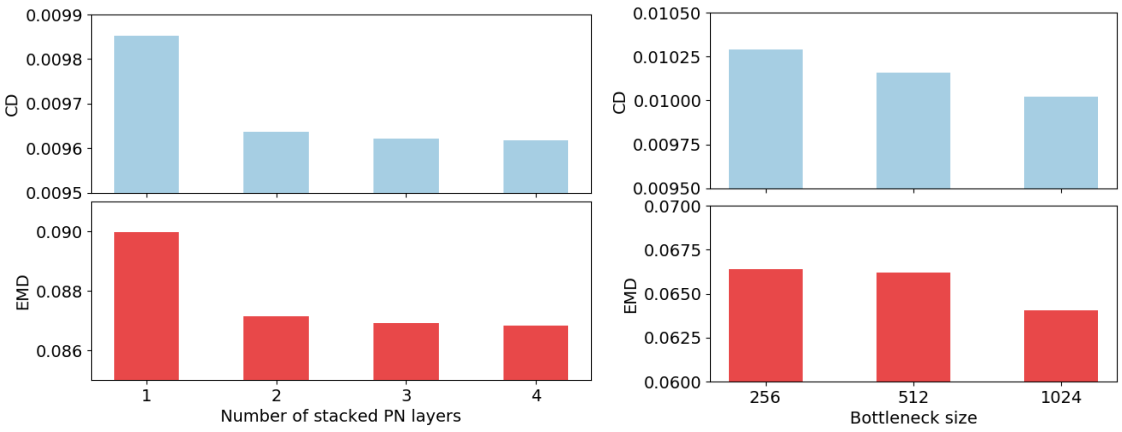
Robustness to occlusion and noise To test the robustness of our method to sensor noise and large occlusions, we perturbed the depth map with Gaussian noise whose standard deviation is 0.01 times the scale of the depth measurements, and occluded it with a mask that covers p percent of points, where p ranges from 0% to 80%. Additionally, we randomly set 1% of the measurements to $d_{max} = 1.6$.

As we can see from Figure 2.8, the errors (CD and EMD) increase only gradually as more and more regions are occluded. Note that our model is *not* trained with these occluded and noisy examples, but it is still robust to them. The strong shape prior that the model has learned helps it to ignore the noisy points and predict reasonable outputs under occlusions. This in part explains its strong generalizability to real-world data, as we will show in the following section.

Number of Parameters As shown in Table 2.1, our model has an order of magnitude fewer parameters than 3D-EPN and FC while achieving significantly better performance.

Table 2.1: Number of trainable model parameters

Method	3D-EPN	FC	Folding	PN2	Ours
# Params	52.4M	53.2M	2.40M	6.79M	6.85M



(a) Test error vs. number of PN layers

(b) Test error vs. bottleneck sizes

Figure 2.9: Analysis of variations in the model architecture. For both CD and EMD, lower is better.

Effect of stacked PN layers We evaluated variants of PCN-CD with different number of stacked PN layers in the encoder. The mean CD and EMD on the ShapeNet test set are shown in Figure 2.9a. It can be seen that the advantage of using 2 stacked PN layers over 1 is quite apparent, whereas the benefit of using more stacked PN layers is almost negligible. This shows that 2 stacked PN layers are sufficient for mixing the local and global geometry information in the input. Thus, we keep the number of stacked PN layers as 2 in our experiments, even though PCN's performance can be further improved by using more stacked PN layers.

Effect of bottleneck size We also tested variants of PCN-EMD with different bottleneck size, i.e. the length of the global feature vector $\mathbf{v}$. The mean CD and EMD on the ShapeNet test set are shown in Figure 2.9b. It can be seen that PCN's performance improves as the bottleneck size increases. In our experiments, we choose the bottleneck size to be 1024 because a larger bottleneck of size 2048 cannot fit into the memory of a single GPU. This implies that if multiple GPUs are used for training, PCN's performance can further improve with a larger bottleneck.

2.4.3 Completion Results on KITTI

In this section, we test our completion method on partial point clouds from real-world LiDAR scans. Specifically, we take a sequence of Velodyne scans from the KITTI dataset [17]. For each frame, we extract points within the object bounding boxes labeled as cars, which results in 2483 partial point clouds. Each point cloud is then transformed to the box's coordinates, completed with a PCN trained on cars from ShapeNet, and transformed back to the world frame. The process is illustrated in Figure 2.1 and several scans with cars completed by our model are shown in Figure 2.11. We use a model trained specifically on cars here to incorporate prior knowledge of the object class. Having such prior knowledge is not necessary for our method but will help the model achieve better performance.

We do not have the complete ground truth point clouds for KITTI. Thus, we propose three alternative metrics to evaluate the performance of our model:

1. **Fidelity** is the average distance from each point in the input to its nearest neighbour in the output. This measures how well the input is preserved. Note that since the input can be corrupted by noise, output with 0 fidelity error may not be actually desirable;
2. **Minimal Matching Distance (MMD)** is the Chamfer Distance (CD) between the output and the car point cloud from ShapeNet that is closest to the output point cloud in terms of CD. This measures how much the output resembles a typical car;
3. **Consistency** is the average CD between the completion outputs of the same instance in consecutive frames. This measures how consistent the network's outputs are against variations in the inputs. We also compute the average CD between the inputs in consecutive frames, denoted as **Consistency (input)**.

These metrics are reported in Table 2.2. We observe that in most cases, our model produces a valid car shape that matches the input while being different from the matched model in ShapeNet, as the one shown in the top row of Figure 2.10. However, we also observe some failure cases, e.g. the one shown in the bottom row of Figure 2.10, caused by extra points from the ground or nearby objects that are within the car's bounding box. This problem can potentially be resolved by adding a segmentation step before passing the partial point cloud to PCN.

Table 2.2: Quantitative results on KITTI.

Fidelity	MMD	Consistency	Consistency (input)
0.02800	0.01850	0.01163	0.05121

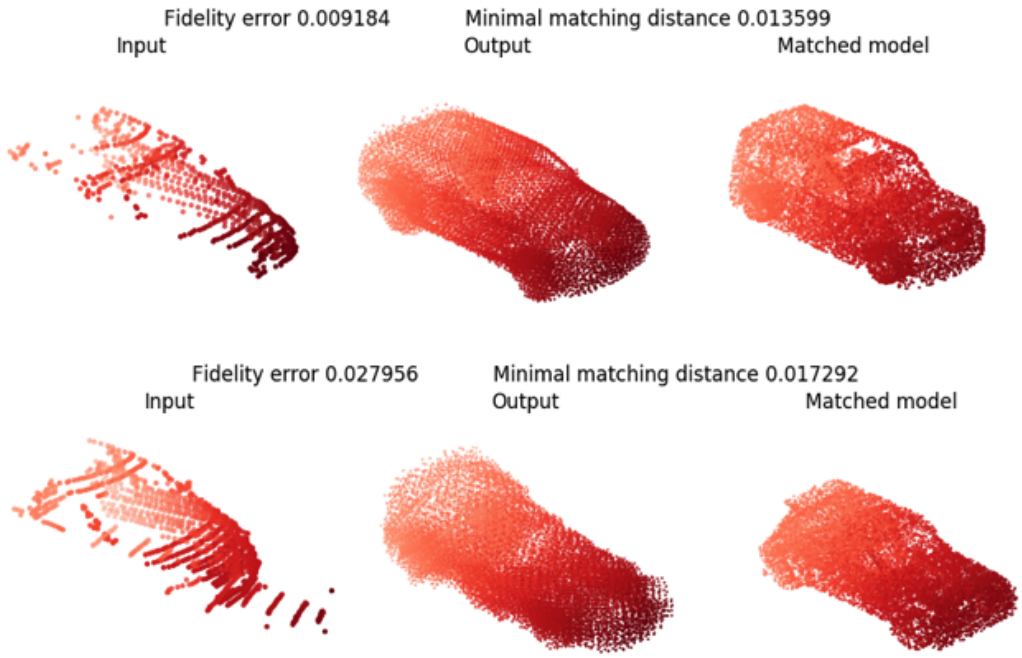


Figure 2.10: **Completion (middle) and matched model (right) for cars in KITTI (left)**. The matched model is the car point cloud from ShapeNet that is closest to the completion output in Chamfer Distance. Top row shows a successful completion and bottom row shows a failure case caused by incorrect segmentation.

Unlike point clouds back-projected from 2.5D images, point clouds from LiDAR scans are very sparse. The 2483 partial point clouds here contain 440 points on average, with some having fewer than 10 points. In contrast, point clouds from 2.5D images used in training usually contain more than 1000 points. In spite of this, our model is able to transfer easily between the two distributions without any fine tuning, producing consistent completions from extremely partial inputs. This can be attributed to the use of point-based representation, which is less sensitive to input density than volumetric representations. In addition, each prediction with our model takes only 0.0012s on a Nvidia GeForce 1080Ti GPU and 2s on a 3.60GHz Intel Core i7-7700 CPU, making it suitable for real-time applications.

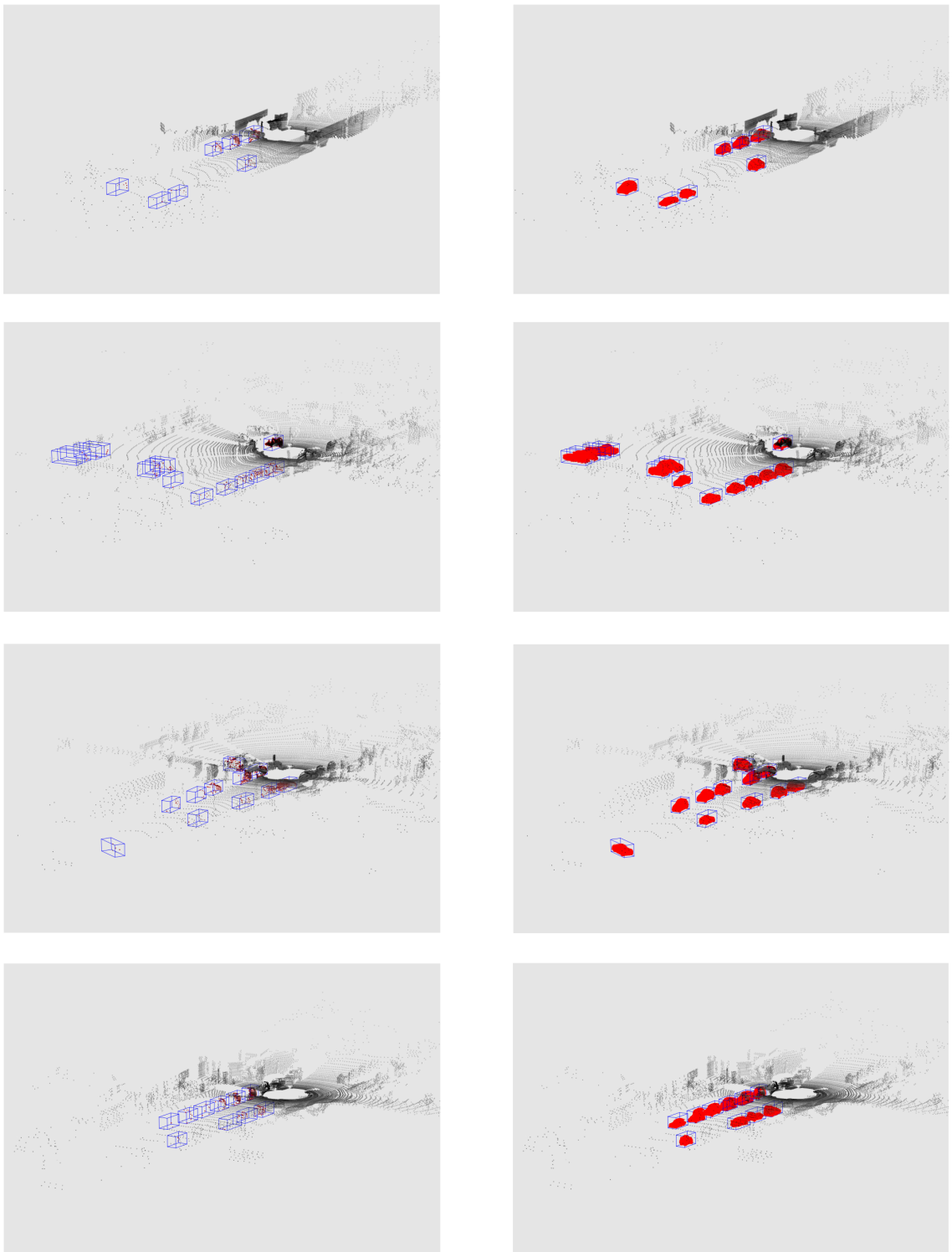
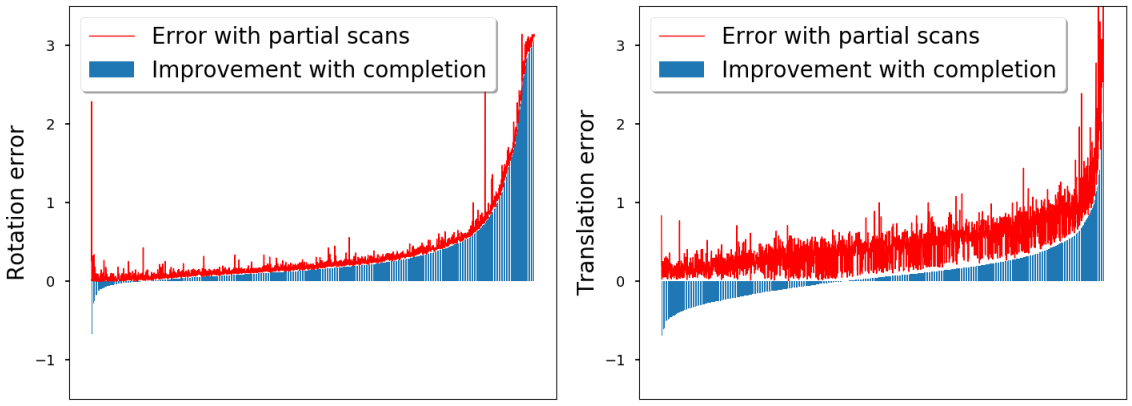
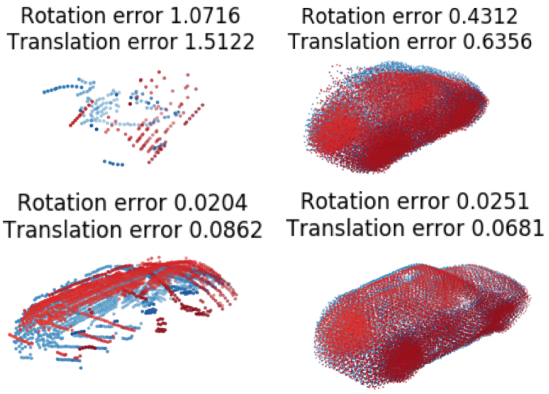


Figure 2.11: Qualitative completion results on KITTI



(a) Rotation error (b) Translation error



(c) Qualitative example

Figure 2.12: **Improvements on point cloud registration**. In (a) and (b), the x-axis represents different registration instances. The red curve is the registration error with partial point clouds and the blue bar indicates the amount of improvement resulting from doing registration with complete point clouds produced by PCN. In (c), registered partial point clouds are shown on the left and registered complete point clouds of the same instances are shown on the right.

2.4.4 Point Cloud Registration with Completion

Many common tasks on point clouds can benefit from a more complete and denser input. Here, as an example of such applications, we show that the output of our network can improve the results of point cloud registration. Specifically, we take the same Velodyne sequence from Section 2.4.3, we extract the car point clouds and perform registration between cars from neighboring frames, which results in 2396 registration instances. We use a simple point-to-point ICP [7] algorithm implemented

in PCL [42] for registration. We provide two kinds of inputs to the registration algorithm – one is partial point clouds from the original scans, another is completed point clouds by a PCN trained on cars from ShapeNet. We compare the rotational and translational error on the registration results with partial and complete inputs. The rotational error is computed as $2\cos^{-1}(2\langle q_1, q_2\rangle^2 - 1)$, where q_1 and q_2 are the quaternion representations of the ground truth rotation and the rotation computed by ICP. This measures the angle between q_1 and q_2. The translational error is computed as $\|t_1 - t_2\|_2$, where t_1 is the ground truth translation and t_2 is the translation computed by ICP.

As shown in Figure 2.12a and 2.12b, both rotation and translation estimations are more accurate with complete point clouds produced by PCN, and the improvement is most significant when the error with partial point clouds is large. Figure 2.12c shows some qualitative examples. We can see that the point clouds from LiDAR scans are very sparse and there is little overlap between the points even in neighbouring frames, which makes the registration very challenging. On the other hand, the completed point clouds from our network are much denser and contain large overlaps, which significantly simplifies the registration problem. Note that the improvement brought by our completion results is not specific to ICP, but can be applied to any registration algorithm. Since inference of our network is fast, it can potentially benefit many real time vision and robotic applications which requires point cloud registration.

2.5 Related Work

3D Shape Completion Existing methods for 3D shape completion can be roughly categorized into geometry-based, alignment-based and learning-based approaches.

Geometry-based approaches complete shapes using geometric cues from the partial input without any external data. For example, surface reconstruction methods [15, 23, 65] generate smooth interpolations to fill holes in locally incomplete scans. Symmetry-driven methods [34, 50, 52] identify symmetry axes and repeating regular structures in the partial input in order to copy parts from observed regions to unobserved regions. These approaches assume moderately complete inputs where the geometry of the missing regions can be inferred directly from the observed regions. This assumption does not hold on most incomplete data from the real world.

Alignment-based approaches complete shapes by matching the partial input with template models from a large shape database. Some [29, 36, 37] retrieve the complete shape directly while some [22, 25, 45] retrieve object parts and then assemble them to obtain the complete shape. Other works [19, 32, 41, 61] deform the retrieved model to synthesize shapes that are more consistent with the input. There are also works [12, 44] that use geometric primitives such as planes and quadrics in place of a shape database. These methods require expensive optimization during inference, making them impractical for online applications. They are also sensitive to noise.

Learning-based approaches complete shapes with a parameterized model (often a deep neural network) that directly maps the partial input to a complete shape, which offers fast inference and better generalization. Our method falls into this category. While most existing learning-based methods [14, 47, 53, 57] represents shapes using voxels, which are convenient for convolutional neural networks, our method uses point clouds, which preserve complete geometric information about the shapes while being memory efficient. One recent work [32] also explores deformable meshes as the shape representation. However, their method assumes all the shapes are in correspondence with a common reference shape, which limits its applicability to certain shape categories such as humans or faces.

Deep Learning on Point Clouds Our method is built upon several recent advances in deep neural networks that operates on point clouds. PointNet and its extension [38, 39] is the pioneer in this area and the state-of-the-art while this work was developed. It combines pointwise multi-layer perceptrons with a symmetric aggregation function that achieves invariance to permutation and robustness to perturbation, which are essential for effective feature learning on point clouds. Several alternatives [30, 48, 51, 54] have been proposed since then. Any of these can be incorporated into our proposed model as the encoder.

There are relatively fewer works on decoder networks which generates point sets from learned features. [1] uses a simple fully-connected decoder, while [16] proposes a multi-branch decoder combining fully-connected and deconvolution layers. Recently, [59] introduces an interesting decoder design which mimics the deformation of a 2D plane into a 3D surface. Our model combines the advantages of these designs to generate higher resolution outputs in an efficient manner.

2.6 Discussion

We have presented a new approach to shape completion using point clouds without any voxelization. To this end, we have designed a deep learning architecture which combines advantages from existing architectures to generate a dense point cloud in a coarse-to-fine fashion, enabling high resolution completion with much fewer parameters than voxel-based models. Our method is effective across multiple object categories and works with inputs from different sensors. In addition, it shows strong generalization performance on unseen objects and real-world data. Our point-based completion method is more scalable and robust than voxel-based methods, which makes it a better candidate for completion of more complex data such as scenes.

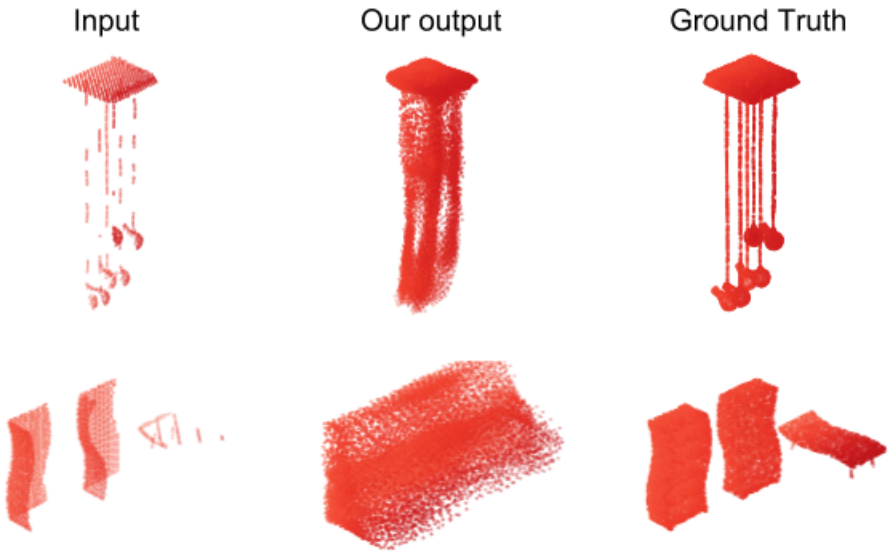


Figure 2.13: **Failure modes**: thin structures (top) and disconnected parts (bottom).

Although our method produces high-quality completion results in most cases, we have identified two prominent failure modes of our model. First, there are some object instances consisting of multiple disconnected parts. Our model fails to recognize this and incorrectly connects the parts. This is likely a result of the strong priors learned from the training dataset where almost all objects are connected. Second, some objects contain very thin structures such as wires. Our model is occasionally unable to recover these structures. There are two possible reasons. First, the points from

these structures are often sparse since they have small surface areas, which makes the 3D feature extraction more difficult. Second, unlike most object surfaces the local geometry of thin structures does not resemble the 2D grid, making it challenging for our model to deform a 2D grid into these thin structures. Some visualizations of these failures are shown in Figure 2.13.

Chapter 3

Canonical Pose Estimation

Estimating the canonical pose of an object from partial views is a longstanding challenge in computer vision and robotics. A key challenge in canonical pose estimation is how to learn features that are equivariant with respect to geometric transformations. To address this challenge, we propose the Iterative Transformer Network (IT-Net), a network module that canonicalizes the pose of an input object point cloud with a series of 3D rigid transformations predicted in an iterative fashion. We demonstrate the efficacy of IT-Net as an anytime pose estimator that predicts 6DoF poses from partial point clouds without knowledge of complete object models. In addition, we show that IT-Net can be trained end-to-end with point-based shape classification and part segmentation networks to improve their performance.

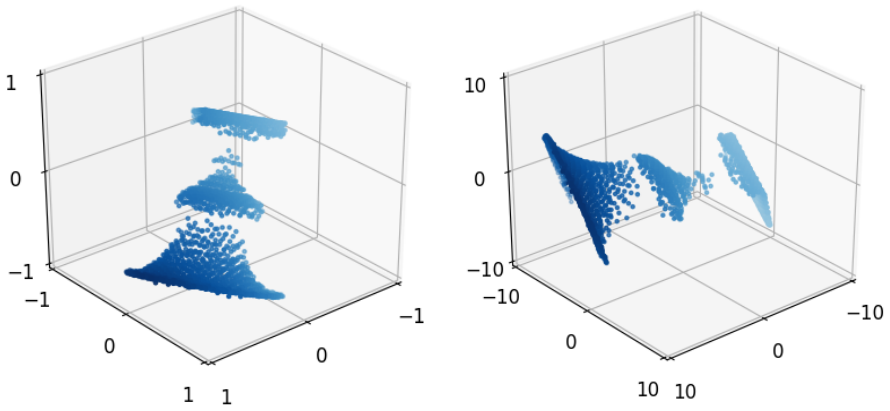
3.1 Motivation

Evidences from cognitive psychology [20] show that in human perception system, shapes are internally represented by description relative to a perceptual reference frame, which we will call the *canonical* frame. In this chapter, we are concerned with the problem of estimating this canonical frame from partial observations of an object represented as a point cloud. This problem is equivalent to estimating the transformation from the canonical frame and to the sensor's coordinate frame where the observations are taken. We will call this transformation the *canonical pose* of an object. Estimating the canonical pose can help us obtain a viewpoint-independent

description the object, which can be beneficial to various downstream tasks such as object recognition and part segmentation. In practice, we can define the canonical frame either explicitly or implicitly as any coordinate frame that makes subsequent tasks easier.

We take a learning-based approach to the pose estimation problem, i.e. we would like to learn the mapping from observations to the canonical pose from data using a parametrized model such as a neural network. The learning-based approach allows us to utilize existing large-scale shape repositories to train a generic pose estimator that is efficient during inference.

A key challenge in learning canonical poses from data is to extract features that are equivariant with respect to input transformations, that is, we want the features to vary accordingly if a transformation is applied to the input. Spatial Transformer Networks (STN) [21] provide a way to learn equivariant features from images by predicting a transformation that is applied to the input before feature extraction for subsequent tasks. Inspired by the idea of STN, we propose a transformer network that operates on 3D point clouds, named Iterative Transformer Network (IT-Net).



(a) Input (b) Transformed input

Figure 3.1: T-Net [38] scales and distorts the input shape (a vase). Note the different scales on the plots.

IT-Net has two major features that distinguish it from existing 3D transformer networks such as T-Net [38]. First, IT-Net outputs a rigid transformation instead of an affine transformation, which prevents undesirable deformation of the inputs (see Figure 3.1) and allows the outputs to be used directly as estimates for object

poses. Second, instead of predicting the transformation in a single step, IT-Net takes advantage of an iterative refinement scheme which decomposes a large transformation into smaller ones that are easier to predict. The iterative refinement scheme not only increases accuracy of the predicted transformation, but also allows anytime prediction, where the prediction result can be gradually refined until the test-time computational budget is depleted.

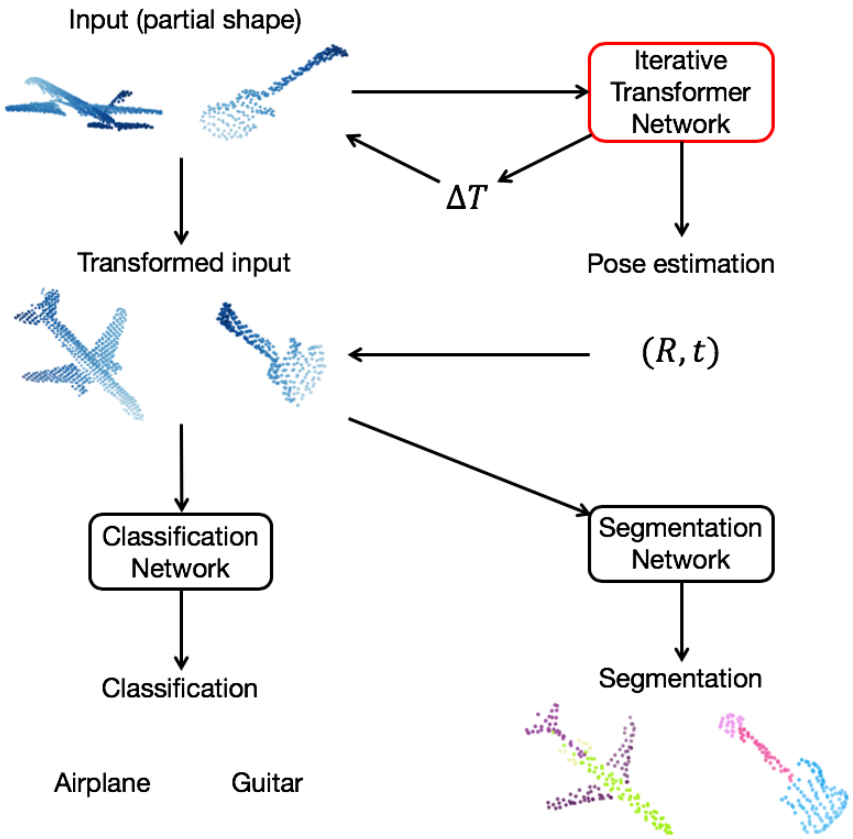


Figure 3.2: Iterative Transformer Network (IT-Net) predicts rigid transformations from partial point clouds in an iterative fashion. It can be used independently as a pose estimator or jointly with classification and segmentation networks.

We evaluate IT-Net on three point cloud learning tasks – pose estimation, shape classification and part segmentation (Figure 3.2) – with partial, unaligned inputs from synthetic as well as real world 3D data. We demonstrate that IT-Net can be used as an anytime object pose estimator and show that IT-Net outperforms existing 3D transformer networks when trained jointly with various state-of-the-art classification

or segmentation networks. We also contribute a dataset consisting of partial point clouds generated from virtual scans of CAD models in ModelNet [55] and ShapeNet [11] as well as real world scans from ScanNet [13]. Our dataset contains challenging inputs with arbitrary 3D rotation, translation and realistic self-occlusion patterns.

The rest of this chapter will be organized as follows. In Section 3.2, we illustrate the architecture of the iterative transformer network. In Section 3.3, we describe how we construct a dataset of partial, misaligned object point clouds. In Section 3.4, we evaluate IT-Net on various point cloud learning tasks and demonstrate its advantage over existing 3D transformer networks. In Section 3.5, we cover previous works related to our approach. In Section 3.6, we provide concluding remarks for the chapter.

3.2 Iterative Transformer Network

Iterative Transformer Network (IT-Net) (Figure 3.3 takes a 3D point cloud and produces a transformation that can be used directly as a pose estimate or applied to the input before feature extraction for subsequent tasks. In what follows, we introduce two features that differentiate IT-Net from existing 3D transformer networks: the rigid transformation prediction and the iterative refinement scheme.

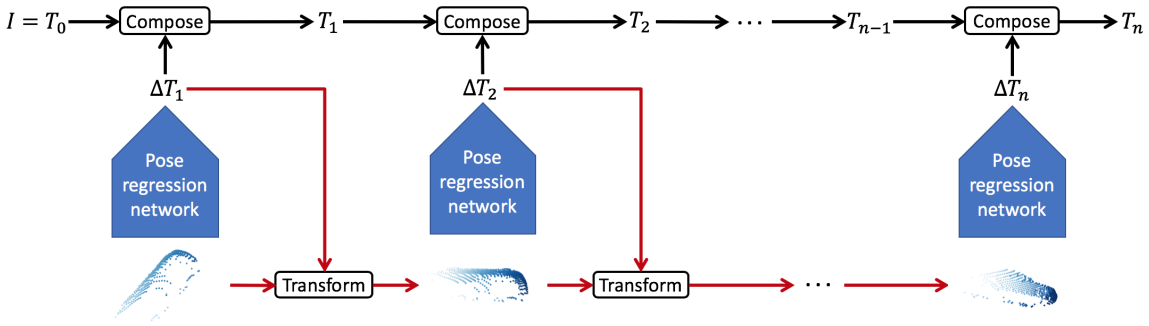


Figure 3.3: Illustration of the iterative scheme employed by IT-Net. At each iteration, the output of the pose regression network is used to transform the input for the next iteration. The parameters of the pose regression network shown in blue arrows are shared across iterations. The final output is a composition of the transformations predicted at each iteration. Arrows colored in red indicate places where the gradient flow is stopped to decorrelate the inputs at different iterations (see Section 3.2.3).

3.2.1 Rigid Transformation Prediction

A single iteration of IT-Net is a pose regression network that takes a point cloud and outputs a 3D rigid transformation T, consisting of a rotation R and translation $\mathbf{t}$ where R is a 3×3 matrix satisfying $RR^T = I, \det(R) = 1$ and $\mathbf{t}$ is a 3×1 vector. Due to the constraints on R, it is inconvenient to represent the rotation as a 3×3 matrix during optimization. As a result, many classical [7] as well as modern deep learning methods [24] parametrize 3D rotations with unit quaternions, which allows us to map an arbitrary 4D vector to a valid 3D rotation.

Therefore, we let the pose regression network output 7 numbers parametrizing the 3D rigid transformation. The first 4 numbers are normalized into a unit quaternion $\mathbf{q}$ and the last 3 are treated as a 3D translation vector $\mathbf{t}$. Then, $\mathbf{q}$ and $\mathbf{t}$ are assembled into a 4×4 matrix $T = \begin{bmatrix} R(\mathbf{q}) & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ where $R(\mathbf{q})$ is the rotation matrix corresponding to $\mathbf{q}$. The matrix representation turns the composition of two rigid transformations into a matrix multiplication, which is convenient for composing the outputs from multiple iterations. In our experiments, we use PointNet [38] as the regression network for its simplicity, but other point cloud-based networks can be used as well.

In contrast to the affine transformation produced by T-Net [38], the rigid transformation predicted by IT-Net can be directly interpreted as a 6D pose, making it possible to use IT-Net independently for pose estimation. More importantly, rigid transformations preserve scales and angles. As a result, the appearance of a point cloud will not vary drastically if it is transformed by the output of IT-Net. This makes it possible to apply the same network iteratively to obtain a more accurate estimation of the transformation.

We note that it is possible to add a regularization term $\|AA^T - I\|$ that forces an affine matrix A to be orthogonal in order to achieve similar effects of predicting a rigid transformation[1]. However, the constraint, no matter how close to satisfied, cannot produce a truly rigid transformation that prevents the deformation of inputs. As shown in Section 3.4.2, the results of the regularized network are not as good as the network that directly outputs rigid transformations.

[1]In [38], this regularization is added to the feature transformation, but not to the input transformation.

3.2.2 Iterative Alignment

The idea of using an iterative scheme for predicting geometric transformations goes back to the classical Lucas-Kanade (LK) algorithm [33] for estimating dense alignment between images. The key insight of LK is that the complex non-linear mapping from image appearance to geometric transformations can be estimated iteratively using simple linear predictors. Specifically, at each iteration, a warp transformation Δp is predicted with a linear function that takes a source and a target image as inputs. Then, the source image is warped by Δp and the process is repeated. The final transformation is a composition of Δp at each step. Later, [5] shows that the parameters used to predict Δp can remain constant across iterations while achieving the same effect as non-constant predictors.

The same idea is employed in the Iterative Closest Point (ICP) algorithm [7] for the alignment of 3D point clouds. At each iteration of ICP, a corresponding set is identified and a rigid transformation ΔT is produced to align the corresponding points. Then, the source point cloud is transformed by ΔT and the process is repeated. Again, the final output is a composition of ΔT at each step. The effectiveness of ICP shows that the iterative refinement framework applies not only to images, but also to 3D point clouds.

The multi-iteration IT-Net can be viewed as an instantiation of this iterative framework. Specifically, the forward pass of IT-Net is unfolded into multiple iterations. At the i-th iteration, a rigid transformation ΔT_i is predicted as described in Section 3.2.1. Then, the input is transformed by ΔT_i and the process is repeated. The final output after n iterations is a composition of the transformations predicted at each iteration, which can be written as a simple matrix product $T_n = \prod_{i=1}^{n} \Delta T_i$.

We use a fixed predictor (i.e. share the network's parameters) across iterations following [5]. In addition to reduction in the number of parameters, the fixed predictor allows us to use different numbers of unfolded iterations in training and testing. As will be shown in Section 3.4.1, once trained, IT-Net can be used as an anytime predictor where increasingly accurate pose estimates can be obtained as the network is applied for more iterations.

The iterative scheme can be interpreted as a way to automatically generate a curriculum, which breaks down the original task into a set of simpler pieces. In

earlier iterations, the network learns to predict large transformations that bring the input near its canonical pose. In later iterations, the network learns to predict small transformations that adjusts the estimate from previous iterations. Note that the curriculum is not manually defined but rather generated by the network itself to optimize the end goal. It will be empirically shown in Section 3.4.1 that this curriculum emerges from the training of IT-Net.

3.2.3 Implementation Details

In addition to the key ingredients above, there are a couple of details that are important for the training of IT-Net. First, we initialize the network to predict the identity transformation $\mathbf{q} = [1\ 0\ 0\ 0]$, $\mathbf{t} = [0\ 0\ 0]$. In this way, the default behavior of each iteration is to preserve the transformation predicted by previous iterations. This identity initialization helps prevent the network from producing large transformations which cancel each other. Second, we stop the gradients propagating through input transformations (red arrows in Figure 3.3) and let the gradients propagate through the output composition only (black arrows in Figure 3.3). This removes dependency among inputs at different iterations which leads to gradient explosion. Empirical evaluations for these design choices can be found in Section 3.4.1.

	Partial ModelNet40	ShapeNet Part	ShapeNet Pose	ScanNet Objects
Source	ModelNet40	ShapeNet	ShapeNet	ScanNet
# classes	40	16	2	33
# train	78,744	13,937	22,000	8,098
# test	2,468	2,874	2,000	1,024
Annotation	Category	Object part	Canonical pose	Category
Synthetic/Real	Synthetic	Synthetic	Synthetic	Real
# scans	1-4	1	1	-
Scan size	64×64	50×50	128×128	-
Focal length	57	∞	64	-
Distance to object center	2-4	∞	1-2	-

Table 3.1: Statistics and generation parameters of our partial point cloud dataset for shape classification, part segmentation and canonical pose estimation.

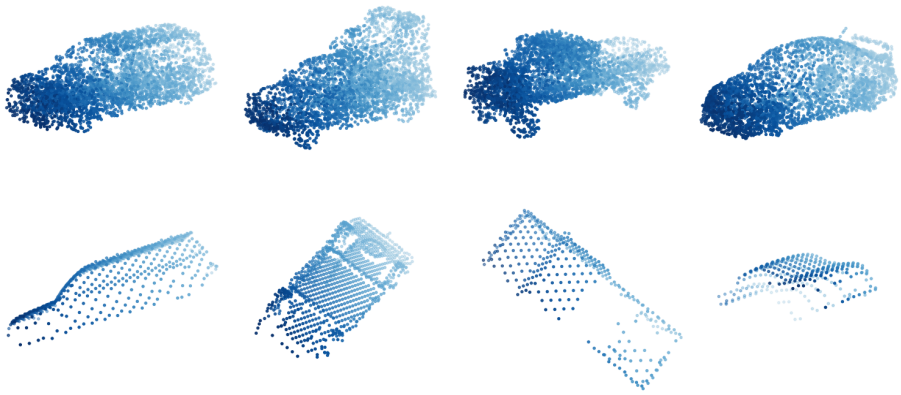


Figure 3.4: Complete point clouds aligned in a canonical frame from ShapeNet [11] (top row) versus partial, unaligned point clouds from our dataset (bottom row).

3.3 Dataset

To evaluate the performance of point cloud learning tasks under a more realistic setting, we build a dataset of object point clouds which captures the incomplete and misaligned nature of real world 3D data. The dataset consists of four parts: ShapeNet Pose, ShapeNet Part, Partial ModelNet40 and ScanNet Objects. The first three parts are generated from virtual depth scans of synthetic objects and the last part is created from real world RGB-D scans. A summary of the dataset statistics is shown in Table 3.1 and a comparison between point clouds in our dataset and the uniformly sampled point clouds used in prior works [38, 39, 54] are shown in Figure 3.4. It can be seen that our dataset contains much more challenging inputs with various poses, densities and levels of incompleteness. Details on the generation of each part will be covered below.

In **Partial ModelNet40**, each point cloud is generated by fusing a sequence of depth scans of CAD models from the ModelNet40 dataset [55] (the models are normalized into the unit sphere following [38]). Specifically, we create a trajectory with up to 4 cameras. The initial camera's orientation is uniformly random and the distance between the camera center and the object center is varied between 2 and 4 units to create point clouds with different density. The subsequent camera positions are generated by rotating the camera around the object center for an angle of up

to 30 degrees. Then, we use Blender [8] to render 64×64 depth scans of the model and back-project them into point clouds. At last, we combine the point clouds by transforming them into the initial camera's coordinates and concatenating the points. We generate 8 partial point clouds for each model in the training split and 1 for each model in the test split of the ModelNet40 dataset [55].

The point clouds in **ShapeNet Pose** are generated in a similar way, except that the CAD models come from the chair and car category in ShapeNet [11] and each point cloud is labeled with the transformation between the camera's coordinates and the model's coordinates. In order to maintain uniqueness of the annotation, we only use 1 scan per instance. Note that the test set and the training set are created from different object models.

Each point cloud in **ShapeNet Part** is labeled with 2-6 parts using the labels provided by [60]. Since the provided part labels are on sampled point clouds instead of the mesh models, we use an approximate rendering procedure that mimics an orthographic depth camera to create the partial point clouds. Specifically, we randomly rotate the complete point cloud and project the points onto a virtual image with pixel size 0.02 in the xy-plane. For each pixel of the virtual image, we keep the point with the smallest z value and discard all other points that project onto the same pixel as they are considered as occluded by the selected point. This procedure creates partial point clouds that look much like those generated from depth scans rendered by Blender.

The point clouds in **ScanNet Objects** are created from 1,512 real world RGB-D scans of indoor scenes in ScanNet [13]. In total, we collect 9,122 object points clouds by cropping the room scans with labeled bounding boxes, where realistic sensor noise and background clutter in the original scans are kept in the resulting point clouds. We then normalize the point clouds into the unit sphere.

3.4 Experimental Evaluation

In this section, we demonstrate the ability of IT-Net to estimate the canonical pose of an object from partial views (Section 3.4.1), and show how this ability leads to performance gains in various point cloud learning tasks, including shape classification (Section 3.4.2) and object part segmentation (Section 3.4.3).

3.4.1 Pose Estimation

In this section, we investigate the efficacy of the iterative refinement scheme on the task of estimating the canonical pose of an object from a partial observation. Specifically, we use IT-Net to predict the transformation that aligns the input shape to a canonical frame defined across all models in the same category (see the top row in Figure 3.4). Unlike most existing works on pose estimation, we do not assume knowledge of the complete object model and we train a single network that generalizes to different objects in the same category.

Architecture and training The architecture of IT-Net is described in Section 3.2 where the pose regression network is a PointNet [38]. The pose regression network shown in Figure 3.5 consists of three parts. The first part is a multi-layer perceptron (MLP) that is applied to each point independently, which takes the $N \times 3$ coordinate matrix and produces a $N \times 1024$ feature matrix. The second part is a max-pooling function which aggregates the features in to a 1×1024 global feature vector. The third part is another MLP that regresses $M = 7$ pose parameters from the global feature vector.

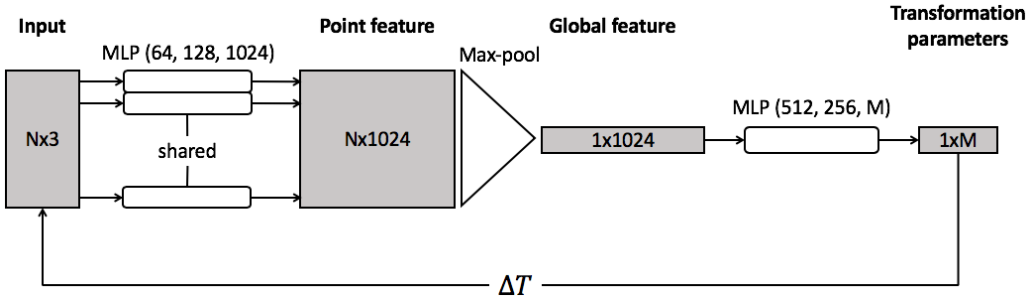


Figure 3.5: Architecture of the pose regression network. Numbers in the parenthesis indicate the number of neurons in each MLP layer.

An explicit loss is applied to the output transformation. For the loss function, we use a variant of PLoss proposed in [56], which measures the average distance between the same set of points under the estimated pose and the ground truth pose. Compared to the L2 loss used in earlier works [24], this loss has the advantage of automatically handling the tradeoff between small rotations and small translations.

The loss can be written as

$$L((R, \mathbf{t}), (\widetilde{R}, \tilde{\mathbf{t}})) = \frac{1}{|X|} \sum_{\mathbf{x} \in X} \|(R\mathbf{x} + \mathbf{t}) - (\widetilde{R}\mathbf{x} + \tilde{\mathbf{t}})\|_2^2, \tag{3.1}$$

where $R, \mathbf{t}$ are the ground truth pose and $\widetilde{R}, \tilde{\mathbf{t}}$ are the estimated pose and X is the set of input points.

We trained IT-Nets under different settings and evaluated their performance on the car point clouds in ShapeNet Pose. In what follows, we provide detailed analysis of the results.

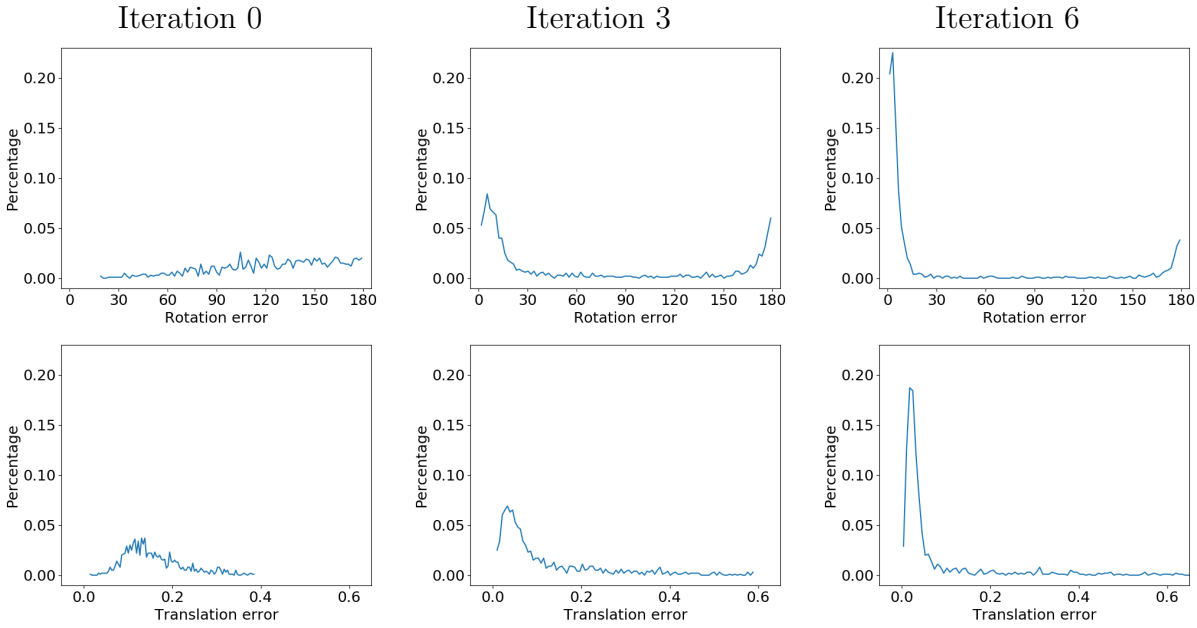
Number of unfolded iterations The number of unfolded iterations during training can be treated as a hyperparameter that controls the iteration at which the loss is applied. We trained IT-Net with different number of unfolded iterations and, as shown in Table 3.2, IT-Net trained with 5 unfolded iterations gives the best performance.

In Figure 3.6a, we visualized the distribution of input poses at different iterations during training by measuring errors with respect to the canonical pose. It can be observed that the distribution of poses skews towards the canonical pose in later iterations. This is evidence that the network generates a curriculum as mentioned in Section 3.2.2. We observe that with too few unfolded iterations, the network does not see enough examples with small errors and thus fails to predict accurate refinements when the shape is near its canonical pose. With too many unfolded iterations, the network sees too many examples with small errors and becomes overly conservative in its prediction. Empirically, 5 iterations turns out to strike a good balance and generate an appropriate distribution of examples in the curriculum which leads to good performance.

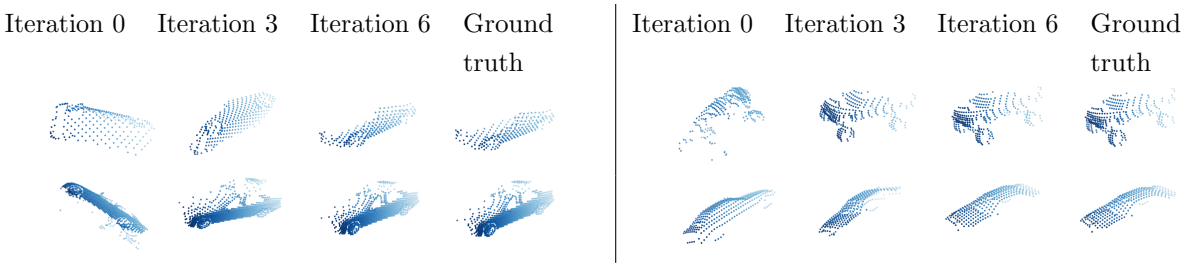
Ablation studies We conducted ablation studies to validate our design choices described in Section 3.2.3, i.e. initializing the network's prediction with the identity transformation and stopping the gradient flow through input transformations. The results are summarized in Table 3.2. It can be seen that the performance degrades significantly without either identity initialization or gradient stopping, which indicates that both are crucial for the iterative refinement scheme to achieve desired behavior.

Unfolded iterations	1	2	3	4	5	6	7
no init	17.1	0.5	18.4	7.3	6.2	41.1	13.1
no stop	36.0	5.0	0.0	0.0	4.1	4.1	0.0
ours	36.9	41.7	47.7	61.1	67.6	62.6	48.2

Table 3.2: Pose accuracy (%) with error threshold $10°, 0.1$ of IT-Nets with different number of unfolded iterations during training. No init means the output is *not* initialized as the identity transformation. No stop means the gradient is *not* stopped during input transformations.



(a) Distribution of rotation and translation errors at different iterations



(b) Qualitative examples

Figure 3.6: **(a)** The distribution (PDF) of rotation and translation error of 2,400 test instances at different iterations. Note how the error distribution skews towards 0 in later iterations. The peak at 180 degrees for rotation error is caused by symmetries in the car models. **(b)** Qualitative results at corresponding iterations.

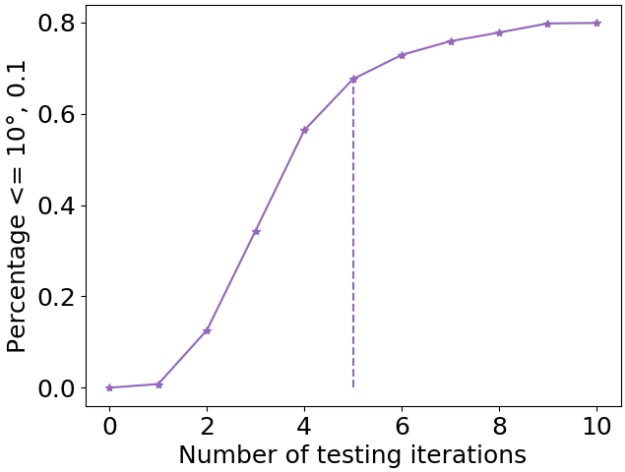


Figure 3.7: Pose accuracy (%) against the number of iterations applied during inference. The dotted line corresponds to the number of unfolded iterations in training. Note how pose accuracy keeps improving with more testing iterations.

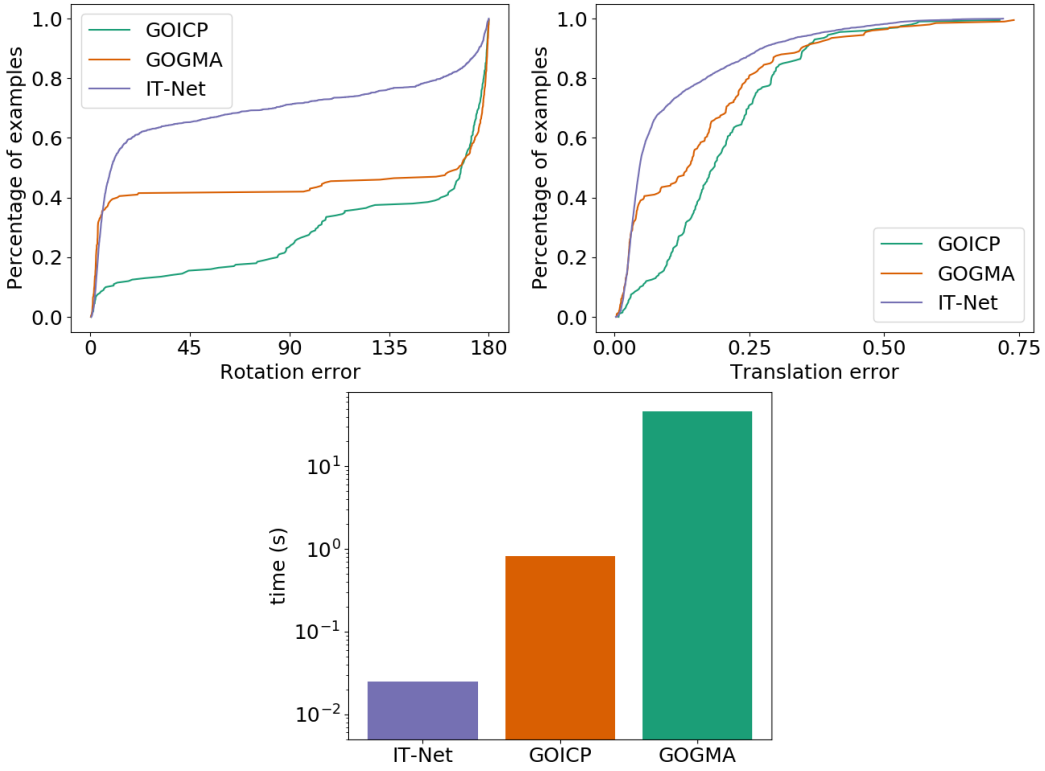


Figure 3.8: Comparison with non-learning baselines on point cloud alignment. The two plots on the left show the CDF of rotation and translation errors over 1000 test instances. The plot on the right shows the average running time per instance.

Anytime pose estimation As noted in Section 3.2.2, sharing weights across iterations allows us to use IT-Net as an anytime pose estimator. Figure 3.7 shows the pose accuracy of an IT-Net against the number of iterations applied during inference. It can be seen that the performance keeps increasing as the number of iteration increases. As a result, during inference, we can keep applying the trained IT-Net to obtain increasingly accurate pose estimates until the time budget runs out (each iteration takes about 0.025s on a 3.60GHz Intel Core i7 CPU).

Comparison with non-learning baselines We applied our pose estimation network to the problem of point cloud alignment and compared the results with classical baselines that is not learning-based. Specifically, for each pair of shapes in the test set, we computed their relative transformation using the poses predicted by IT-Net. The results are compared against two state-of-the-art, non-learning-based alignment methods, GOICP [58] and GOGMA [9]. Unlike classical ICP which only works with good initialization, these baseline methods can estimate alignment from arbitrary initialization. The results are shown in Figure 3.8. Note that IT-Net not only produces more accurate alignment, but is also several orders of magnitude faster on average. The running times are measured on a 3.60GHz Intel Core i7 CPU.

3.4.2 3D Shape Classification

In this section, we demonstrate the ability of IT-Net to improve the performance of point-based classification networks by aligning the inputs into a canonical frame.

Specifically, we trained IT-Net with two state-of-the-art shape classification networks, PointNet [38] and Dynamic Graph CNN (DGCNN) [54]. The classifier takes the point cloud transformed by the output of IT-Net and outputs a score for each class. The entire network is trained with cross-entropy loss on the class scores and no explicit supervision is applied on the predicted transformation. and tested their performance on two datasets, Partial ModelNet40 and ScanNet Objects.

We compared IT-Net with two baselines, T-Net and regularized T-Net (T-Net reg), which share the same architecture as IT-Net except for the last output layer of the pose regression network. While IT-Net outputs 7 numbers for rotation (quaternion) and translation, T-Net and T-Net reg outputs 9 numbers to form a 3×3 affine

transformation matrix A. For T-Net reg, a regularization term $\|AA^T - I\|$ is added to the loss with weight 0.001. We compared the performance of different transformer-classifier combinations on two datasets, Partial ModelNet40 and ScanNet Objects, and summarize the results in Table 3.3 and Table 3.4 respectively.

Classifier	PointNet								
Transformer	None	T-Net		T-Net reg		IT-Net (ours)			
# Iterations	0	1	2	1	2	1	2	3	4
Accuracy	59.97	66.04	35.13	65.84	67.06	68.72	**69.94**	69.61	68.35
Classifier	DGCNN								
Transformer	None	T-Net		T-Net reg		IT-Net (ours)			
# Iterations	0	1	2	1	2	1	2	3	4
Accuracy	65.60	70.38	16.61	71.15	72.69	72.57	**74.15**	73.46	72.85

Table 3.3: Classification accuracy on Partial ModelNet40.

Classifier	PointNet						
Transformer	None	T-Net		T-Net reg		IT-Net (ours)	
# Iterations	0	1	2	1	2	1	2
Accuracy	62.11	63.09	30.86	62.99	61.82	63.67	**66.02**
Classifier	DGCNN						
Transformer	None	T-Net		T-Net reg		IT-Net (ours)	
# Iterations	0	1	2	1	2	1	2
Accuracy	66.02	72.75	18.55	74.12	70.80	76.36	**76.66**

Table 3.4: Classification accuracy on ScanNet Objects.

Unlike T-Net, IT-Net preserves the shape of the input without introducing any scaling or shearing. Figure 3.9 shows that the output of T-Net is on a very different scale than the original input. This explains why the performance of T-Net drops significantly if we try to apply the same iterative scheme directly: as the network sees inputs on vastly different scales from different iterations, the training fails to converge. The regularized T-Net resolves this issue to some extent, but its performance is still not as good as IT-Net.

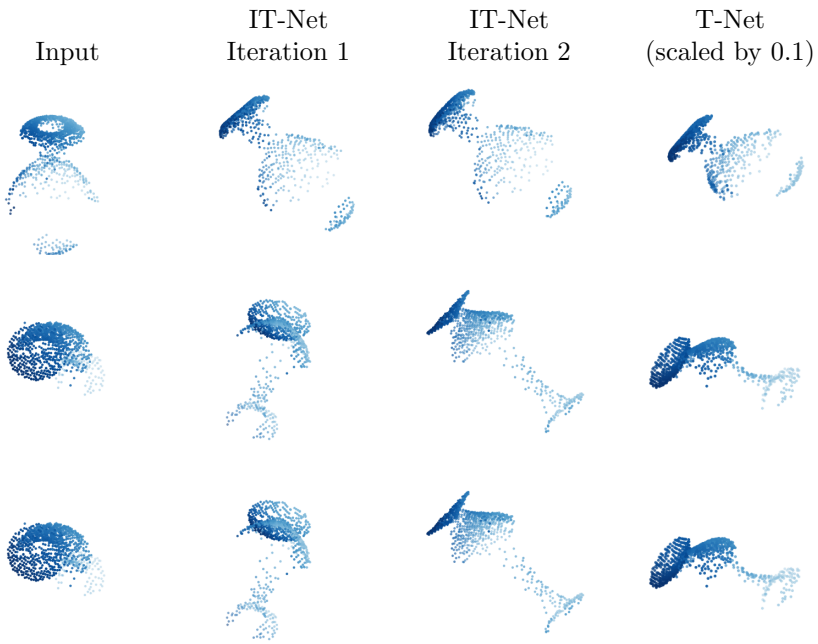
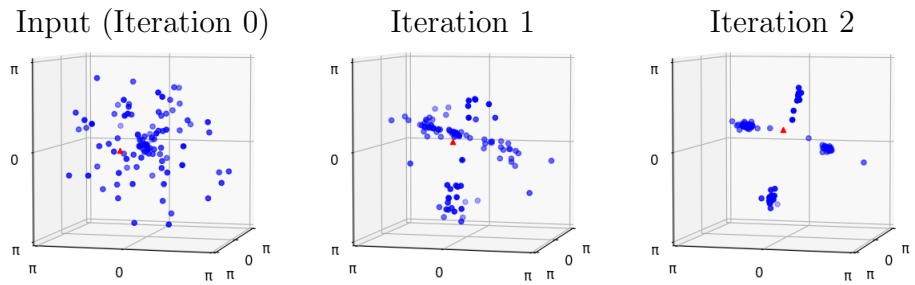


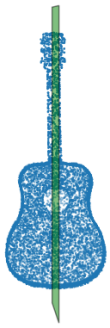
Figure 3.9: Inputs transformed by IT-Net and T-Net trained jointly with DGCNN. Note that T-Net's outputs are on a much different scale (10 times bigger) than the original inputs.

It can be seen that IT-Net consistently outperforms baseline transformers when trained with different classifiers. This is evidence that the advantage of IT-Net is agnostic to the classifier architecture. Further, the advantage of IT-Net over baselines on real data matches that on synthetic data. This demonstrates IT-Net's ability to process point clouds with realistic sensor noise and clutter and its potential to be incorporated in detection/pose estimation pipelines in the wild.

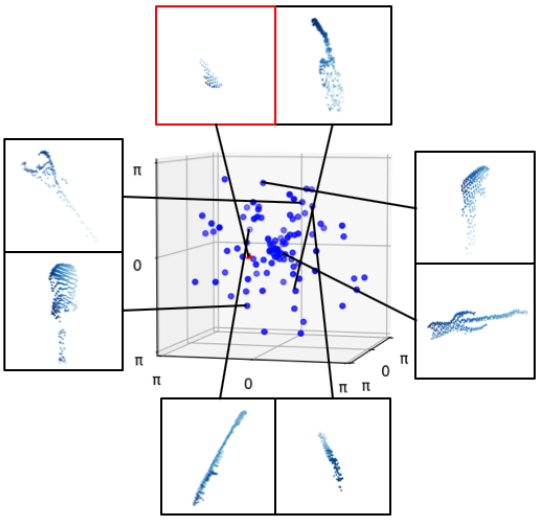
We observe that without explicit supervision, IT-Net learns to transform the inputs into a set of canonical poses which we call "pose clusters". The learned transformations removes pose variations among the inputs, which simplifies the classification problem. We note that transforming the inputs into one of the pose clusters so that they can be recognized by the classifiers is a simpler task than producing precise alignments as in Section 3.4.1. Therefore, the performance gain diminishes as the number of unfolded iterations becomes larger than 2 (see Table 3.3).



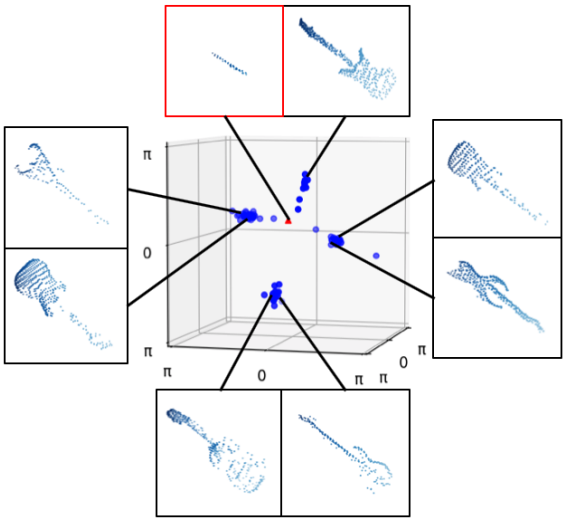
(a) Distribution of orientations at different iterations.



(b) Reflection symmetry of guitars.

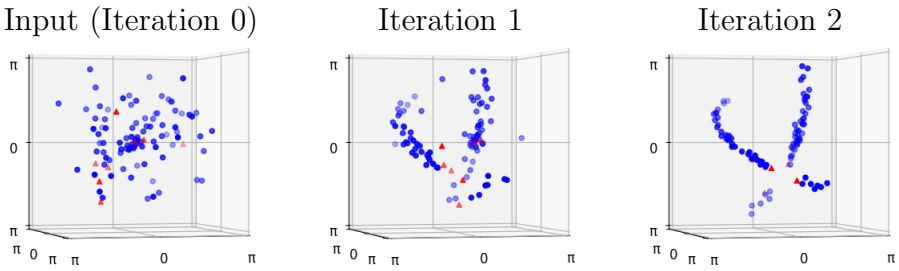


(c) Examples of original inputs (Iteration 0).

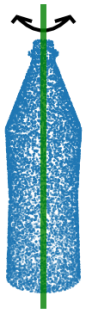


(d) Examples of transformed inputs (Iteration 2).

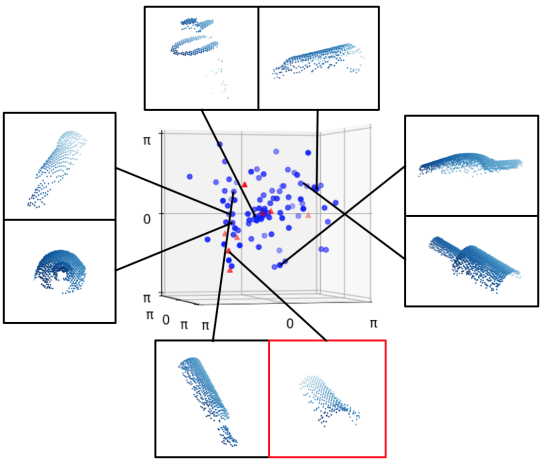
Figure 3.10: Pose cluster visualization for guitars. **(a)** Distribution of axis-angle representation of orientations of all test examples at different iterations. Note how clusters emerge from uniformly distributed poses. Correctly classified examples are shown in blue and incorrectly classified examples are shown in red. **(b)** The reflection symmetry present in most guitars. **(c)** Examples of original inputs at iteration 0. The object orientations are uniformly distributed. **(d)** Examples of transformed inputs at iteration 2. Note that these are the inputs received by the classifier. The object orientations are grouped into 4 clusters, but visually there seems to be only 2 major orientations due to the reflection symmetry shown in **(b)**. A failure case caused by heavy occlusion is shown in the red box.



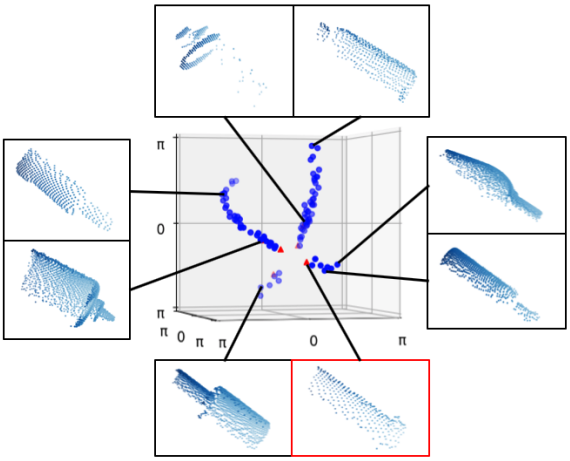
(a) Distribution of orientations at different iterations.



(b) Rotational symmetry of bottles.



(c) Examples of original inputs (Iteration 0).



(d) Examples of transformed inputs (Iteration 2).

Figure 3.11: Pose cluster visualization for bottles. **(a)** Distribution of axis-angle representation of orientations of all test examples at different iterations. Note how clusters emerge from uniformly distributed poses. Correctly classified examples are shown in blue and incorrectly classified examples are shown in red. **(b)** The rotational symmetry present in most bottles. **(c)** Examples of original inputs at iteration 0. The object orientations are uniformly distributed. **(d)** Examples of transformed inputs at iteration 2. Note that these are the inputs received by the classifier. The object orientations after transformation are grouped into 2 clusters. The clusters have semicircle shapes since any orientation in these semicircles are in fact indistinguishable due to the rotational symmetry shown in **(b)**. A failure case is shown in the red box. In this case the model misclassifies the bottle as a vase.

In Figure 3.10 and Figure 3.11, we visualize the "pose clusters" learned by a 2-iteration IT-Net on the the guitar and bottle category of Partial ModelNet40 respectively. To visualize the pose clusters, we calculate the difference between the canonical orientation of the input shape and the orientation of the transformed input at different iterations. Then, we convert the orientation difference into axis-angle representation, which is a 3D vector, and plot these vectors for all test examples in a particular category.

We observe that although the object poses are uniformly distributed initially, clusters of poses emerge after applying the transformations predicted by IT-Net (Figure 3.10a, 3.11a). This is evidence that IT-Net discover a canonical space to align the inputs with no explicit supervision. Interestingly, there are usually more than one cluster and the shapes of the clusters are related to the symmetries of the object (Figure 3.10b, 3.11b). Further, we note that sometimes even objects across different categories are aligned after being transformed by IT-Net (Figure 3.10d, 3.11d). Nevertheless, the pose clusters are less apparent for certain categories where the shapes are nearly spherical, e.g. flower pots.

3.4.3 Object Part Segmentation

In this section, we show that the advantage resulting from the canonical frame learned by IT-Net is not specific to the classification task. More specifically, we replaced the classification network in Sec. 3.4.2 with a segmentation network and evaluated the part segmentation performance on ShapeNet Part. We treat part segmentation as a per-point classification problem and train the network with a per-point cross entropy loss. Similar to Section 3.4.2, no explicit supervision is applied on the transformations.

Table 3.5 summarizes the quantitative result and Figure 3.12 shows some qualitative examples. As in the case of classification, IT-Net reduces pose variations in the inputs, which leads to performance gains over the base model without transformer as well as models trained with alternative 3D transformers. Note that the architecture of IT-Net here is identical to the ones in Section 3.4.2, which demonstrates the potential of IT-Net as a plug-in module without task-specific adjustments to the model architecture.

	mean	table	chair	air plane	lamp	car	guitar	laptop	knife
# shapes		5271	3758	2690	1547	898	787	451	392
# parts		3	4	4	4	4	3	2	2
None	67.9	71.6	75.2	68.8	56.9	48.2	82.4	58.0	68.5
T-Net	71.1	73.7	77.5	73.6	60.2	53.0	85.8	**63.2**	73.6
IT-Net-1	72.3	74.5	**78.7**	75.9	60.6	**57.7**	85.1	58.3	**78.6**
IT-Net-2	**72.6**	**75.1**	78.3	**76.3**	**62.1**	56.3	**86.8**	58.9	74.5
		pistol	motor cycle	mug	skate board	bag	ear phone	rocket	cap
# shapes		283	202	184	152	76	68	66	55
# parts		3	6	2	3	2	3	3	2
None		61.7	39.0	65.6	49.6	41.9	43.5	28.1	**50.9**
T-Net		65.4	48.5	70.3	57.7	15.9	41.8	41.7	48.5
IT-Net-1		67.9	**51.5**	70.3	61.6	31.6	**53.9**	35.2	45.3
IT-Net-2		**68.6**	46.4	**70.6**	**65.9**	**43.5**	51.6	**42.6**	45.9

(a) Results where the base segmentation model is PointNet [38]

	mean	table	chair	air plane	lamp	car	guitar	laptop	knife
# shapes		5271	3758	2690	1547	898	787	451	392
# parts		3	4	4	4	4	3	2	2
None	76.9	78.8	82.6	77.3	71.3	52.3	90.1	76.8	80.0
T-Net	77.1	79.2	82.5	78.0	70.1	55.7	89.1	73.1	81.5
IT-Net-1	78.2	79.9	84.3	78.2	**72.9**	54.9	91.0	**78.7**	78.1
IT-Net-2	**79.1**	**80.2**	**84.7**	**79.9**	72.1	**62.6**	**91.1**	76.4	**82.8**
		pistol	motor cycle	mug	skate board	bag	ear phone	rocket	cap
# shapes		283	202	184	152	76	68	66	55
# parts		3	6	2	3	2	3	3	2
None		70.1	40.4	**86.1**	67.6	71.0	66.7	53.1	76.9
T-Net		73.0	39.1	81.1	69.1	**74.1**	71.1	51.4	74.6
IT-Net-1		71.8	**44.6**	84.8	66.6	71.2	**72.7**	**55.0**	77.9
IT-Net-2		**76.9**	44.0	84.4	**71.8**	68.1	66.8	54.2	**80.4**

(b) Results where the base segmentation model is DGCNN [54]

Table 3.5: Part segmentation results on ShapeNet Part. The number appending IT-Net indicates the number of iterations. The metric is mIoU(%) on points. The mean is calculated as the average of per-category mIoUs weighted by the number of shapes. We order the categories by number of shapes since the performance is more unstable for categories with fewer shapes.

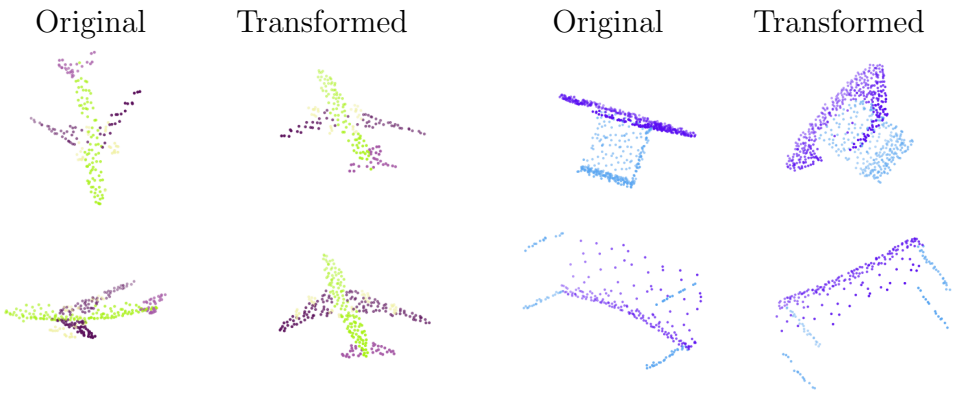


Figure 3.12: Inputs transformed by IT-Net trained with DGCNN [54] on part segmentation. The colors indicate predictions of the segmentation network.

3.5 Related Work

Feature Learning on Point Clouds Traditional point feature descriptors [4, 43, 49] rely on geometric properties of points such as curvatures. They do not encode semantic information and it is non-trivial to find the combination of features that is optimal for specific tasks.

PointNet [38] introduces a way to extract semantic and task-specific features from point clouds using a neural network, which outperforms competing methods on several shape analysis tasks like shape classification. Subsequent works [30, 39, 54] further improves the performance of point cloud-based networks by accounting for interactions among local neighborhoods of points.

Most datasets [55, 60] used to evaluate feature learning on point clouds consist of complete point clouds. A few works [38, 63] have investigated feature learning from partial point clouds. However, these works assume that the inputs are aligned in a canonical coordinate system. In this work, we show how to remove this assumption using a transformer network.

Spatial Transformer Network Spatial Transformer Network (STN) [21] is a network module that performs explicit geometric transformations on the input image in a differentiable way, which introduces invariance to geometric transformations and can be trained jointly with various task-specific networks to improve their performance. STN can be thought of as a geometry predictor which models the com-

plicated non-linear relationship between the appearance of the image and geometric transformations.

IC-STN [31] is an extension of STN that makes use of an iterative scheme inspired by the Lucas-Kanade algorithm [33]. Our network utilizes a similar iterative scheme to predict accurate geometric transformations.

Iterative Error Feedback The idea of using iterative error feedback (IEF) in neural networks have been studied in the context of 2D human pose estimation [10] and taxonomic prediction [64]. Under the IEF framework, instead of trying to directly predict the target in a feed-forward fashion, the network predicts the error in the current estimate and corrects it iteratively. While our proposed network falls under this general framework, unlike previous works, it does not use intermediate supervision or separate stages of training. Rather, the loss is applied at a certain iteration during training and the gradient is propagated through the composition of outputs from previous iterations.

3.6 Summary

We propose a new transformer network on 3D point clouds named Iterative Transformer Network (IT-Net). In an iterative fashion, IT-Net outputs a rigid transformation that can be used to estimate object pose or transform the input for subsequent tasks. The effectiveness of IT-Net in various tasks shows that the classical idea of iterative refinement still applies in the context of deep learning.

IT-Net can be easily integrated with existing deep learning architectures for shape classification and segmentation, and improve the performance on these tasks with partial, unaligned inputs by introducing invariance to geometric transformations. This opens up many avenues for future research on using neural networks to extract semantic information from real world point cloud data.

Chapter 4

Conclusion and Open Problems

In this thesis, we have introduced two approaches to incorporate prior knowledge into the design of neural networks that operate on 3D point clouds and demonstrated their efficacy on two important 3D understanding tasks – 3D shape completion and canonical pose estimation. First, we show that leveraging the geometric structure of 3D surfaces as collections of embedded 2D patches, we can design a point generation network that has fewer parameters but better representational power, which is useful for completing unobserved regions of partial scans. Next, we illustrate that algorithmic structures such as iterative refinement can help us build networks that estimate quantities like object poses more accurately without increasing the network capacity, and that possess desired properties such as anytime prediction.

While these works are important steps towards developing autonomous systems that understand 3D environments from point cloud data, there are still many problems that remain open. We will highlight a number of these problems below.

1. Most of existing works on point cloud understanding assumes the inputs are isolated objects. While these inputs can be obtained from instance segmentation methods, instance segmentation of point clouds is itself an open problem. Moreover, in tasks such as shape completion, it is often desirable to ground the objects in context. Thus, moving towards holistic scene understanding is an important direction, which requires models that can process large-scale point clouds more efficiently.
2. Not all points carry an equal amount of information. For example, dropping a

point on a planar region on a wall is probably not as important as dropping a point on a sharp corner of a table. Although point cloud is very memory-efficient compared to voxels, it is still desirable to have some hierarchical model that allows us to put more capacity in regions with more interesting structures.

3. Another assumption that is often made in existing 3D understanding research is that the 3D scene is static, but in fact, we can acquire much richer information if we look at the dynamics of a scene. For instance, given a video with associated depth information, we can easily segment a moving object from a static background. How to incorporate the time dimension and extract semantics from dynamic 3D data is an exciting and largely unexplored area of research.

Bibliography

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017. 2.3.2, 2.5

[2] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M Seitz, and Richard Szeliski. Building rome in a day. In *2009 IEEE 12th international conference on computer vision*, pages 72–79. IEEE, 2009. 1.1

[3] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017. 2.4.1

[4] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1626–1633. IEEE, 2011. 3.5

[5] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004. 3.2.2

[6] Dimitri P Bertsekas. A distributed asynchronous relaxation algorithm for the assignment problem. In *Decision and Control, 1985 24th IEEE Conference on*, pages 1703–1704. IEEE, 1985. 2.3.4

[7] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992. 2.4.4, 3.2.1, 3.2.2

[8] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2018. URL http://www.blender.org. 3.3

[9] Dylan Campbell and Lars Petersson. Gogma: Globally-optimal gaussian mixture alignment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5685–5694, 2016. 3.4.1

[10] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human

pose estimation with iterative error feedback. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4733–4742, 2016. 3.5

[11] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 1.1, 2.1, 2.4, 2.4.1, 2.4.2, 3.1, 3.4, 3.3

[12] Anne-Laure Chauve, Patrick Labatut, and Jean-Philippe Pons. Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1261–1268. IEEE, 2010. 2.5

[13] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2017. 2.4.1, 2.4.2, 3.1, 3.3

[14] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 3, 2017. 2.1, 2.4.2, 2.5

[15] James Davis, Stephen R Marschner, Matt Garr, and Marc Levoy. Filling holes in complex surfaces using volumetric diffusion. In *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, pages 428–441. IEEE, 2002. 2.5

[16] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 38, 2017. 2.3.4, 2.5

[17] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32 (11):1231–1237, 2013. 2, 2.1, 2.4, 2.4.3

[18] Walter Gerbino and D Salmaso. The effect of amodal completion on visual matching. *Acta psychologica*, 65(1):25–46, 1987. 2.1

[19] Saurabh Gupta, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Aligning 3d models to rgb-d images of cluttered scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4731–4740, 2015. 2.5

[20] Glyn W Humphreys. Reference frames and shape perception. *Cognitive Psychology*, 15(2):151–196, 1983. 3.1

[21] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015. 3.1, 3.5

[22] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics (TOG)*, 31(4):55, 2012. 2.5

[23] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. 2.4.2, 2.5

[24] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015. 3.2.1, 3.4.1

[25] Vladimir G Kim, Wilmot Li, Niloy J Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and Thomas Funkhouser. Learning part-based templates from large collections of 3d shapes. *ACM Transactions on Graphics (TOG)*, 32(4):70, 2013. 2.5

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2.4.1

[27] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10. IEEE Computer Society, 2007. 1.1

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1.1

[29] Yangyan Li, Angela Dai, Leonidas Guibas, and Matthias Nießner. Database-assisted object retrieval for real-time 3d reconstruction. In *Computer Graphics Forum*, volume 34, pages 435–446. Wiley Online Library, 2015. 2.5

[30] 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*, pages 820–830, 2018. 2.5, 3.5

[31] Chen-Hsuan Lin and Simon Lucey. Inverse compositional spatial transformer networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2568–2576, 2017. 3.5

[32] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. *arXiv preprint arXiv:1712.00268*, 2017. 2.5

[33] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981. 3.2.2, 3.5

[34] Niloy J Mitra, Leonidas J Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Transactions on Graphics (TOG)*, 25(3):560–568, 2006. 2.5

[35] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 1.1

[36] Liangliang Nan, Ke Xie, and Andrei Sharf. A search-classify approach for cluttered indoor scene understanding. *ACM Transactions on Graphics (TOG)*, 31(6):137, 2012. 2.5

[37] Mark Pauly, Niloy J Mitra, Joachim Giesen, Markus H Gross, and Leonidas J Guibas. Example-based 3d scan completion. In *Symposium on Geometry Processing*, number EPFL-CONF-149337, pages 23–32, 2005. 2.5

[38] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 1.1, 2.3.1, 2.5, 3.1, 3.2.1, 1, 3.3, 3.4.1, 3.4.2, 3.5a, 3.5

[39] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5105–5114, 2017. 2.4.2, 2.4.2, 2.5, 3.3, 3.5

[40] Ronald A Rensink and James T Enns. Early completion of occluded objects. *Vision research*, 38(15-16):2489–2505, 1998. 2.1

[41] Jason Rock, Tanmay Gupta, Justin Thorsen, JunYoung Gwak, Daeyun Shin, and Derek Hoiem. Completing 3d object shape from one depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2484–2493, 2015. 2.5

[42] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011. 2.4.4

[43] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217, 2009. 3.5

[44] Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and reconstruction with primitive shapes. In *Computer Graphics Forum*, volume 28, pages 503–512. Wiley Online Library, 2009. 2.5

[45] Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. Structure recovery by part assembly. *ACM Transactions on Graphics (TOG)*, 31(6):180, 2012. 2.5

[46] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 190–198. IEEE, 2017. 2.1

[47] David Stutz and Andreas Geiger. Learning 3d shape completion from laser scan data with weak supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1955–1964, 2018. 2.5

[48] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018. 2.5

[49] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Computer graphics forum*, volume 28, pages 1383–1392. Wiley Online Library, 2009. 3.5

[50] Minhyuk Sung, Vladimir G Kim, Roland Angst, and Leonidas Guibas. Data-driven structural priors for shape completion. *ACM Transactions on Graphics (TOG)*, 34(6):175, 2015. 2.4.2, 2.5

[51] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018. 2.5

[52] Sebastian Thrun and Ben Wegbreit. Shape from symmetry. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1824–1831. IEEE, 2005. 2.5

[53] Jacob Varley, Chad DeChant, Adam Richardson, Joaquín Ruales, and Peter Allen. Shape completion enabled robotic grasping. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 2442–2447. IEEE, 2017. 2.5

[54] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. 2.5, 3.3, 3.4.2, 3.5b, 3.12, 3.5

[55] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 1.1, 2.1, 3.1, 3.3, 3.5

[56] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn:

A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017. 3.4.1

[57] Bo Yang, Stefano Rosa, Andrew Markham, Niki Trigoni, and Hongkai Wen. 3d object dense reconstruction from a single depth view. *arXiv preprint arXiv:1802.00411*, 2018. 2.5

[58] Jiaolong Yang, Hongdong Li, and Yunde Jia. Go-icp: Solving 3d registration efficiently and globally optimally. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1457–1464, 2013. 3.4.1

[59] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215, 2018. 2.3.2, 2.4.2, 2.4.2, 2.5

[60] Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, Leonidas Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016. 3.3, 3.5

[61] Kangxue Yin, Hui Huang, Hao Zhang, Minglun Gong, Daniel Cohen-Or, and Baoquan Chen. Morfit: interactive surface reconstruction from incomplete point clouds with curve-driven topology and geometry control. *ACM Trans. Graph.*, 33(6):202–1, 2014. 2.5

[62] Wentao Yuan, David Held, Christoph Mertz, and Martial Hebert. Iterative transformer network for 3d point cloud. *arXiv preprint arXiv:1811.11209*, 2018. 1.3

[63] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pages 728–737. IEEE, 2018. 1.3, 3.5

[64] Amir R Zamir, Te-Lin Wu, Lin Sun, William B Shen, Bertram E Shi, Jitendra Malik, and Silvio Savarese. Feedback networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 1808–1817. IEEE, 2017. 3.5

[65] Wei Zhao, Shuming Gao, and Hongwei Lin. A robust hole-filling algorithm for triangular mesh. *The Visual Computer*, 23(12):987–997, 2007. 2.5