Supplementary

A. Overview

This document provides additional quantitative results, technical details and more qualitative test examples to the main paper.

In Sec B we extend the robustness test to compare PointNet with VoxNet on incomplete input. In Sec C we provide more details on neural network architectures, training parameters and in Sec D we describe our detection pipeline in scenes. Then Sec E illustrates more applications of PointNet, while Sec F shows more analysis experiments. Sec G provides a proof for our theory on PointNet. At last, we show more visualization results in Sec H.

B. Comparison between PointNet and VoxNet (Sec 5.2)

We extend the experiments in Sec 5.2 Robustness Test to compare PointNet and VoxNet [17] (a representative architecture for volumetric representation) on robustness to missing data in the input point cloud. Both networks are trained on the same train test split with 1024 number of points as input. For VoxNet we voxelize the point cloud to $32 \times 32 \times 32$ occupancy grids and augment the training data by random rotation around up-axis and jittering.

At test time, input points are randomly dropped out by a certain ratio. As VoxNet is sensitive to rotations, its prediction uses average scores from 12 viewpoints of a point cloud. As shown in Fig 8, we see that our PointNet is much more robust to missing points. VoxNet's accuracy dramatically drops when half of the input points are missing, from 86.3% to 46.0% with a 40.3% difference, while our PointNet only has a 3.7% performance drop. This can be explained by the theoretical analysis and explanation of our PointNet – it is learning to use a collection of *critical points* to summarize the shape, thus it is very robust to missing data.

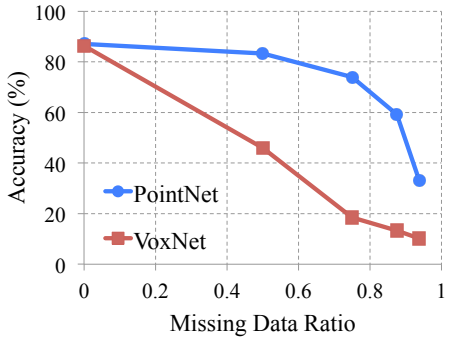

Figure 8. **PointNet v.s. VoxNet [17] on incomplete input data.** Metric is overall classification accurcacy on ModelNet40 test set. Note that VoxNet is using 12 viewpoints averaging while PointNet is using only one view of the point cloud. Evidently PointNet presents much stronger robustness to missing points.

C. Network Architecture and Training Details (Sec 5.1)

PointNet Classification Network As the basic architecture is already illustrated in the main paper, here we provides more details on the joint alignment/transformation network and training parameters.

The first transformation network is a mini-PointNet that takes raw point cloud as input and regresses to a 3×3 matrix. It's composed of a shared $MLP(64, 128, 1024)$ network (with layer output sizes 64, 128, 1024) on each point, a max pooling across points and two fully connected layers with output sizes 512, 256. The output matrix is initialized as an identity matrix. All layers, except the last one, include ReLU and batch normalization. The second transformation network has the same architecture as the first one except that the output is a 64×64 matrix. The matrix is also initialized as an identity. A regularization loss (with weight 0.001) is added to the softmax classification loss to make the matrix close to orthogonal.

We use dropout with keep ratio 0.7 on the last fully connected layer, whose output dimension 256, before class score prediction. The decay rate for batch normalization starts with 0.5 and is gradually increased to 0.99. We use adam optimizer with initial learning rate 0.001, momentum 0.9 and batch size 32. The learning rate is divided by 2 every 20 epochs. Training on ModelNet takes 3-6 hours to converge with TensorFlow and a GTX1080 GPU.

PointNet Segmentation Network The segmentation network is an extension to the classification PointNet. Local point features (the output after the second transformation network) and global feature (output of the max pooling) are concatenated for each point. No dropout is used for segmentation network. Training parameters are the same as the classification network.

As to the task of shape part segmentation, we made a few modifications to the basic segmentation network architecture (Fig 2 in main paper) in order to achieve best performance, as illustrated in Fig 9. We add a one-hot vector indicating the class of the input and concatenate it with the max pooling layer's output. We also increase neurons in some layers and add skip links to collect local point features in different layers and concatenate them to form point feature input to the segmentation network.

While [27] and [29] deal with each object category independently, due to the lack of training data for some categories (the total number of shapes for all the categories in the data set are shown in the first line), we train our PointNet across categories (but with one-hot vector input to indicate category). To allow fair comparison, when testing

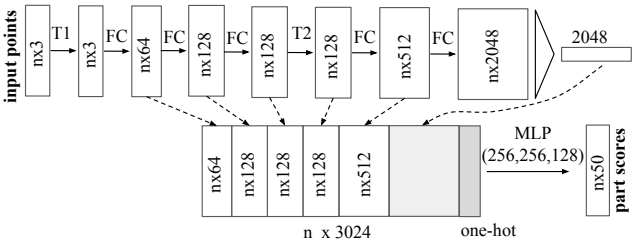

Figure 9. **Network architecture for part segmentation.** T1 and T2 are alignment/transformation networks for input points and features. FC is fully connected layer operating on each point. MLP is multi-layer perceptron on each point. One-hot is a vector of size 16 indicating category of the input shape.

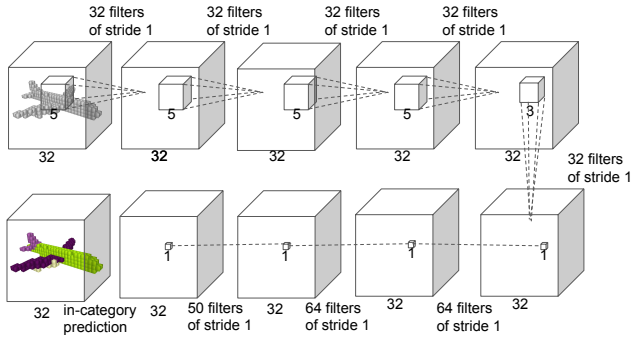

Figure 10. **Baseline 3D CNN segmentation network.** The network is fully convolutional and predicts part scores for each voxel.

these two models, we only predict part labels for the given specific object category.

As to semantic segmentation task, we used the architecture as in Fig 2 in the main paper.

It takes around six to twelve hours to train the model on ShapeNet part dataset and around half a day to train on the Stanford semantic parsing dataset.

Baseline 3D CNN Segmentation Network In ShapeNet part segmentation experiment, we compare our proposed segmentation version PointNet to two traditional methods as well as a 3D volumetric CNN network baseline. In Fig 10, we show the baseline 3D volumetric CNN network we use. We generalize the well-known 3D CNN architectures, such as VoxNet [17] and 3DShapeNets [28] to a fully convolutional 3D CNN segmentation network.

For a given point cloud, we first convert it to the volumetric representation as a occupancy grid with resolution $32 \times 32 \times 32$. Then, five 3D convolution operations each with 32 output channels and stride of 1 are sequentially applied to extract features. The receptive field is 19 for each voxel. Finally, a sequence of 3D convolutional layers with kernel size $1 \times 1 \times 1$ is appended to the computed feature map to predict segmentation label for each voxel. ReLU and batch normalization are used for all layers except the last one. The network is trained across categories, however, in order to compare with other baseline methods where object category is given, we only consider output scores in the given object category.

D. Details on Detection Pipeline (Sec 5.1)

We build a simple 3D object detection system based on the semantic segmentation results and our object classification PointNet.

We use connected component with segmentation scores to get object proposals in scenes. Starting from a random point in the scene, we find its predicted label and use BFS to search nearby points with the same label, with a search radius of 0.2 meter. If the resulted cluster has more than 200 points (assuming a 4096 point sample in a 1m by 1m area), the cluster's bounding box is marked as one object proposal. For each proposed object, it's detection score is computed as the average point score for that category. Before evaluation, proposals with extremely small areas/volumes are pruned. For tables, chairs and sofas, the bounding boxes are extended to the floor in case the legs are separated with the seat/surface.

We observe that in some rooms such as auditoriums lots of objects (e.g. chairs) are close to each other, where connected component would fail to correctly segment out individual ones. Therefore we leverage our classification network and uses sliding shape method to alleviate the problem for the chair class. We train a binary classification network for each category and use the classifier for sliding window detection. The resulted boxes are pruned by non-maximum suppression. The proposed boxes from connected component and sliding shapes are combined for final evaluation.

In Fig 11, we show the precision-recall curves for object detection. We trained six models, where each one of them is trained on five areas and tested on the left area. At test phase, each model is tested on the area it has never seen. The test results for all six areas are aggregated for the PR curve generation.

E. More Applications (Sec 5.1)

Model Retrieval from Point Cloud Our PointNet learns a global shape signature for every given input point cloud. We expect geometrically similar shapes have similar global signature. In this section, we test our conjecture on the shape retrieval application. To be more specific, for every given query shape from ModelNet test split, we compute its global signature (output of the layer before the score prediction layer) given by our classification PointNet and retrieve similar shapes in the train split by nearest neighbor search. Results are shown in Fig 12.

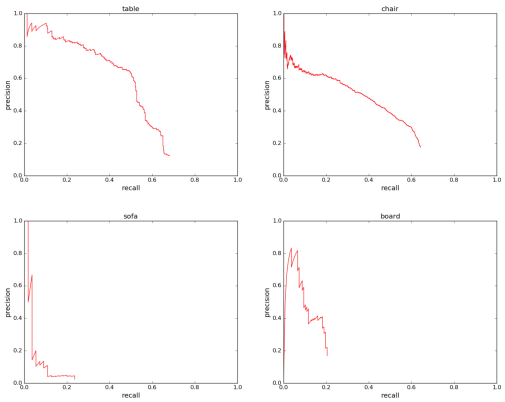

Figure 11. **Precision-recall curves for object detection in 3D point cloud.** We evaluated on all six areas for four categories: table, chair, sofa and board. IoU threshold is 0.5 in volume.

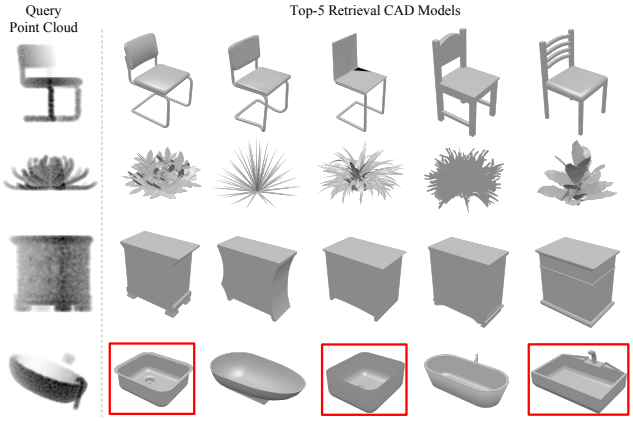

Figure 12. **Model retrieval from point cloud.** For every given point cloud, we retrieve the top-5 similar shapes from the ModelNet test split. From top to bottom rows, we show examples of chair, plant, nightstand and bathtub queries. Retrieved results that are in wrong category are marked by red boxes.

Shape Correspondence In this section, we show that point features learnt by PointNet can be potentially used to compute shape correspondences. Given two shapes, we compute the correspondence between their *critical point sets* C_S's by matching the pairs of points that activate the same dimensions in the global features. Fig 13 and Fig 14 show the detected shape correspondence between two similar chairs and tables.

F. More Architecture Analysis (Sec 5.2)

Effects of Bottleneck Dimension and Number of Input Points Here we show our model's performance change with regard to the size of the first max layer output as well as the number of input points. In Fig 15 we see that performance grows as we increase the number of points however it saturates at around 1K points. The max layer size plays an important role, increasing the layer size from

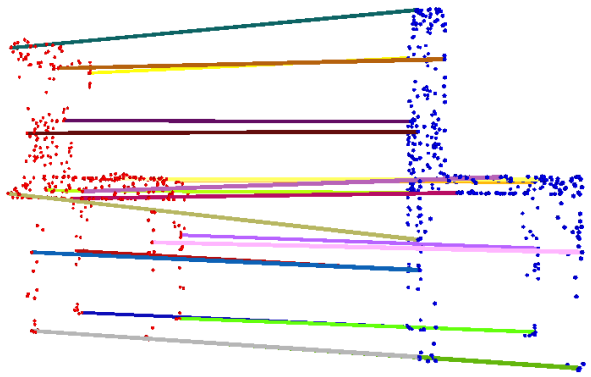

Figure 13. **Shape correspondence between two chairs.** For the clarity of the visualization, we only show 20 randomly picked correspondence pairs.

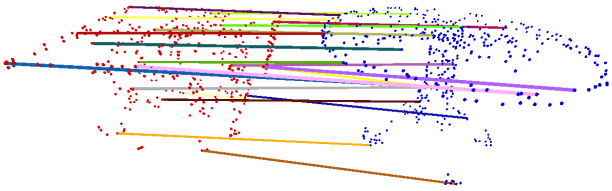

Figure 14. **Shape correspondence between two tables.** For the clarity of the visualization, we only show 20 randomly picked correspondence pairs.

64 to 1024 results in a $2-4\%$ performance gain. It indicates that we need enough point feature functions to cover the 3D space in order to discriminate different shapes.

It's worth notice that even with 64 points as input (obtained from furthest point sampling on meshes), our network can achieve decent performance.

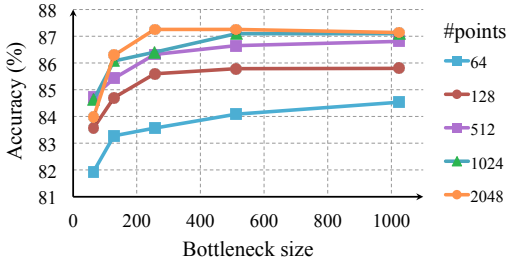

Figure 15. **Effects of bottleneck size and number of input points.** The metric is overall classification accuracy on ModelNet40 test set.

MNIST Digit Classification While we focus on 3D point cloud learning, a sanity check experiment is to apply our network on a 2D point clouds - pixel sets.

To convert an MNIST image into a 2D point set we threshold pixel values and add the pixel (represented as a

point with (x, y) coordinate in the image) with values larger than 128 to the set. We use a set size of 256. If there are more than 256 pixels int he set, we randomly sub-sample it; if there are less, we pad the set with the one of the pixels in the set (due to our max operation, which point to use for the padding will not affect outcome).

As seen in Table 7, we compare with a few baselines including multi-layer perceptron that considers input image as an ordered vector, a RNN that consider input as sequence from pixel (0,0) to pixel (27,27), and a vanilla version CNN. While the best performing model on MNIST is still well engineered CNNs (achieving less than 0.3% error rate), it's interesting to see that our PointNet model can achieve reasonable performance by considering image as a 2D point set.

	input	error (%)
Multi-layer perceptron [22]	vector	1.60
LeNet5 [12]	image	0.80
Ours PointNet	point set	0.78

Table 7. **MNIST classification results.** We compare with vanilla versions of other deep architectures to show that our network based on point sets input is achieving reasonable performance on this traditional task.

Normal Estimation In segmentation version of PointNet, local point features and global feature are concatenated in order to provide context to local points. However, it's unclear whether the context is learnt through this concatenation. In this experiment, we validate our design by showing that our segmentation network can be trained to predict point normals, a local geometric property that is determined by a point's neighborhood.

We train a modified version of our segmentation PointNet in a supervised manner to regress to the ground-truth point normals. We just change the last layer of our segmentation PointNet to predict normal vector for each point. We use absolute value of cosine distance as loss.

Fig. 16 compares our PointNet normal prediction results (the left columns) to the ground-truth normals computed from the mesh (the right columns). We observe a reasonable normal reconstruction. Our predictions are more smooth and continuous than the ground-truth which includes flipped normal directions in some region.

Segmentation Robustness As discussed in Sec 5.2 and Sec B, our PointNet is less sensitive to data corruption and missing points for classification tasks since the global shape feature is extracted from a collection of *critical points* from the given input point cloud. In this section, we show that the robustness holds for segmentation tasks too. The per-point part labels are predicted based on the combination of per-point features and the learnt global shape feature. In Fig 17, we illustrate the segmentation results for the given input point clouds S (the left-most column), the *critical point sets* $\mathcal{C}_S$ (the middle column) and the *upper-bound shapes* $\mathcal{N}_S$.

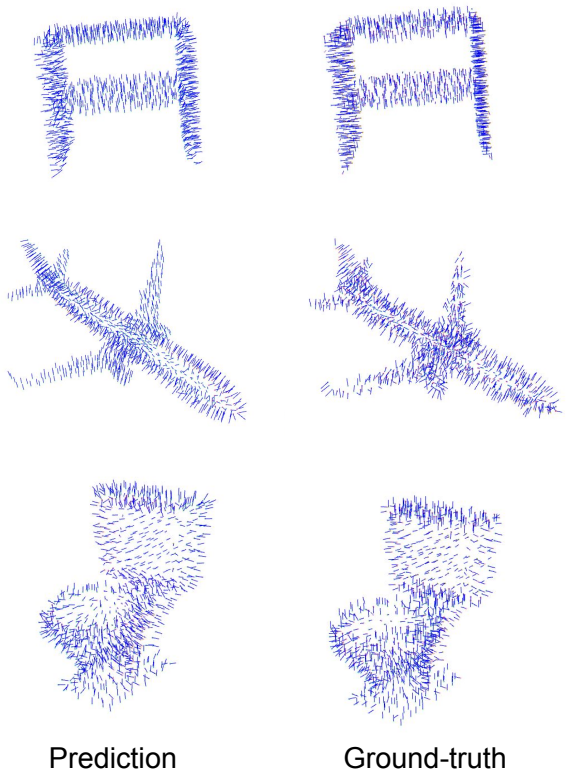

Figure 16. **PointNet normal reconstrution results.** In this figure, we show the reconstructed normals for all the points in some sample point clouds and the ground-truth normals computed on the mesh.

Network Generalizability to Unseen Shape Categories In Fig 18, we visualize the *critical point sets* and the *upper-bound shapes* for new shapes from unseen categories (face, house, rabbit, teapot) that are not present in ModelNet or ShapeNet. It shows that the learnt per-point functions are generalizable. However, since we train mostly on man-made objects with lots of planar structures, the reconstructed upper-bound shape in novel categories also contain more planar surfaces.

G. Proof of Theorem (Sec 4.3)

Let $\mathcal{X} = \{S : S \subseteq [0, 1] \text{ and } |S| = n\}$.

$f : \mathcal{X} \to \mathbb{R}$ is a continuous function on $\mathcal{X}$ w.r.t to Hausdorff distance $d_H(\cdot, \cdot)$ if the following condition is satisfied:

$\forall \epsilon > 0, \exists \delta > 0$, for any $S, S' \in \mathcal{X}$, if $d_H(S, S') < \delta$, then $|f(S) - f(S')| < \epsilon$.

We show that f can be approximated arbitrarily by composing a symmetric function and a continuous function.

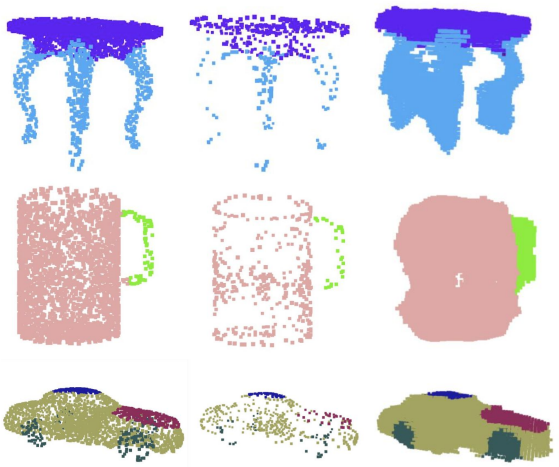

Figure 17. **The consistency of segmentation results.** We illustrate the segmentation results for some sample given point clouds S, their *critical point sets* $\mathcal{C}_S$ and *upper-bound shapes* $\mathcal{N}_S$. We observe that the shape family between the $\mathcal{C}_S$ and $\mathcal{N}_S$ share a consistent segmentation results.

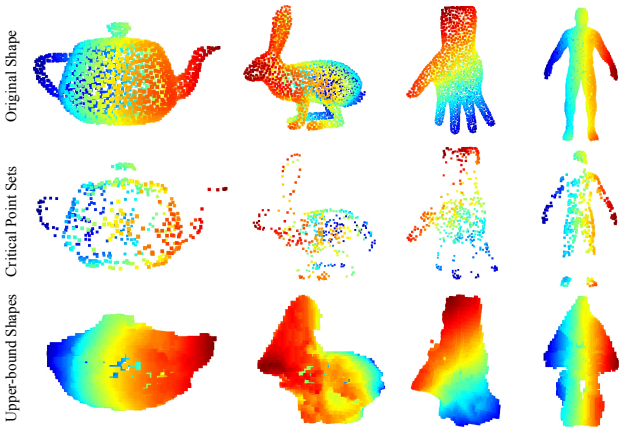

Figure 18. **The critical point sets and the upper-bound shapes for unseen objects.** We visualize the *critical point sets* and the *upper-bound shapes* for teapot, bunny, hand and human body, which are not in the ModelNet or ShapeNet shape repository to test the generalizability of the learnt per-point functions of our PointNet on other unseen objects. The images are color-coded to reflect the depth information.

Theorem 1. *Suppose $f : \mathcal{X} \to \mathbb{R}$ is a continuous set function w.r.t Hausdorff distance $d_H(\cdot,\cdot)$. $\forall \epsilon > 0$, $\exists$ a continuous function h and a symmetric function $g(x_1,\dots,x_n) = \gamma \circ MAX$, where γ is a continuous function, MAX is a vector max operator that takes n vectors as input and returns a new vector of the element-wise maximum, such that for any $S \in \mathcal{X}$,*

$$|f(S) - \gamma(MAX(h(x_1),\dots,h(x_n)))| < \epsilon$$

where $x_1,\dots,x_n$ are the elements of S extracted in certain order,

Proof. By the continuity of f, we take δ_ϵ so that $|f(S) - f(S')| < \epsilon$ for any $S, S' \in \mathcal{X}$ if $d_H(S,S') < \delta_\epsilon$.

Define $K = \lceil 1/\delta_\epsilon \rceil$, which split $[0,1]$ into K intervals evenly and define an auxiliary function that maps a point to the left end of the interval it lies in:

$$\sigma(x) = \frac{\lfloor Kx \rfloor}{K}$$

Let $\tilde{S} = \{\sigma(x) : x \in S\}$, then

$$|f(S) - f(\tilde{S})| < \epsilon$$

because $d_H(S,\tilde{S}) < 1/K \le \delta_\epsilon$.

Let $h_k(x) = e^{-d(x,[\frac{k-1}{K},\frac{k}{K}])}$ be a soft indicator function where $d(x,I)$ is the point to set (interval) distance. Let $\mathbf{h}(x) = [h_1(x);\dots;h_K(x)]$, then $\mathbf{h} : \mathbb{R} \to \mathbb{R}^K$.

Let $v_j(x_1,\dots,x_n) = \max\{\tilde{h}_j(x_1),\dots,\tilde{h}_j(x_n)\}$, indicating the occupancy of the j-th interval by points in S. Let $\mathbf{v} = [v_1;\dots;v_K]$, then $\mathbf{v} : \underbrace{\mathbb{R}\times\dots\times\mathbb{R}}_{n} \to \{0,1\}^K$ is a symmetric function, indicating the occupancy of each interval by points in S.

Define $\tau : \{0,1\}^K \to \mathcal{X}$ as $\tau(v) = \{\frac{k-1}{K} : v_k \ge 1\}$, which maps the occupancy vector to a set which contains the left end of each occupied interval. It is easy to show:

$$\tau(\mathbf{v}(x_1,\dots,x_n)) \equiv \tilde{S}$$

where $x_1,\dots,x_n$ are the elements of S extracted in certain order.

Let $\gamma : \mathbb{R}^K \to \mathbb{R}$ be a continuous function such that $\gamma(\mathbf{v}) = f(\tau(\mathbf{v}))$ for $v \in \{0,1\}^K$. Then,

$$\begin{aligned}&|\gamma(\mathbf{v}(x_1,\dots,x_n)) - f(S)|\\ =&|f(\tau(\mathbf{v}(x_1,\dots,x_n))) - f(S)| < \epsilon\end{aligned}$$

Note that $\gamma(\mathbf{v}(x_1,\dots,x_n))$ can be rewritten as follows:

$$\begin{aligned}\gamma(\mathbf{v}(x_1,\dots,x_n)) =&\gamma(\text{MAX}(\mathbf{h}(x_1),\dots,\mathbf{h}(x_n)))\\ =&(\gamma\circ\text{MAX})(\mathbf{h}(x_1),\dots,\mathbf{h}(x_n))\end{aligned}$$

Obviously $\gamma \circ \text{MAX}$ is a symmetric function. □

Next we give the proof of Theorem 2. We define $\mathbf{u} = \underset{x_i\in S}{\text{MAX}}\{h(x_i)\}$ to be the sub-network of f which maps a point set in $[0,1]^m$ to a K-dimensional vector. The following theorem tells us that small corruptions or extra noise points in the input set is not likely to change the output of our network:

Theorem 2. *Suppose $\mathbf{u} : \mathcal{X} \to \mathbb{R}^K$ such that $\mathbf{u} = \underset{x_i\in S}{MAX}\{h(x_i)\}$ and $f = \gamma \circ \mathbf{u}$. Then,*

(a) $\forall S, \exists\, \mathcal{C}_S, \mathcal{N}_S \subseteq \mathcal{X}$, $f(T) = f(S)$ *if* $\mathcal{C}_S \subseteq T \subseteq \mathcal{N}_S$;

(b) $|\mathcal{C}_S| \leq K$

Proof. Obviously, $\forall S \in \mathcal{X}$, $f(S)$ is determined by $\mathbf{u}(S)$. So we only need to prove that $\forall S, \exists\, \mathcal{C}_S, \mathcal{N}_S \subseteq \mathcal{X}, f(T) = f(S)$ if $\mathcal{C}_S \subseteq T \subseteq \mathcal{N}_S$.

For the jth dimension as the output of $\mathbf{u}$, there exists at least one $x_j \in \mathcal{X}$ such that $h_j(x_j) = \mathbf{u}_j$, where h_j is the jth dimension of the output vector from h. Take $\mathcal{C}_S$ as the union of all x_j for $j = 1, \ldots, K$. Then, $\mathcal{C}_S$ satisfies the above condition.

Adding any additional points x such that $h(x) \leq \mathbf{u}(S)$ at all dimensions to $\mathcal{C}_S$ does not change $\mathbf{u}$, hence f. Therefore, $\mathcal{T}_S$ can be obtained adding the union of all such points to $\mathcal{N}_S$.

□

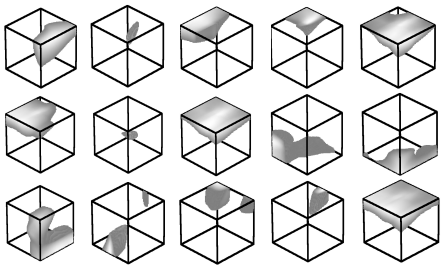

Figure 19. **Point function visualization.** For each per-point function h, we calculate the values $h(p)$ for all the points p in a cube of diameter two located at the origin, which spatially covers the unit sphere to which our input shapes are normalized when training our PointNet. In this figure, we visualize all the points p that give $h(p) > 0.5$ with function values color-coded by the brightness of the voxel. We randomly pick 15 point functions and visualize the activation regions for them.

H. More Visualizations

Classification Visualization We use t-SNE[15] to embed point cloud global signature (1024-dim) from our classification PointNet into a 2D space. Fig 20 shows the embedding space of ModelNet 40 test split shapes. Similar shapes are clustered together according to their semantic categories.

Segmentation Visualization We present more segmentation results on both complete CAD models and simulated Kinect partial scans. We also visualize failure cases with error analysis. Fig 21 and Fig 22 show more segmentation results generated on complete CAD models and their simulated Kinect scans. Fig 23 illustrates some failure cases. Please read the caption for the error analysis.

Scene Semantic Parsing Visualization We give a visualization of semantic parsing in Fig 24 where we show input point cloud, prediction and ground truth for both semantic segmentation and object detection for two office rooms and one conference room. The area and the rooms are unseen in the training set.

Point Function Visualization Our classification PointNet computes K (we take $K = 1024$ in this visualization) dimension point features for each point and aggregates all the per-point local features via a max pooling layer into a single K-dim vector, which forms the global shape descriptor.

To gain more insights on what the learnt per-point functions h's detect, we visualize the points p_i's that give high per-point function value $f(p_i)$ in Fig 19. This visualization clearly shows that different point functions learn to detect for points in different regions with various shapes scattered in the whole space.

Figure 20. **2D embedding of learnt shape global features.** We use t-SNE technique to visualize the learnt global shape features for the shapes in ModelNet40 test split.

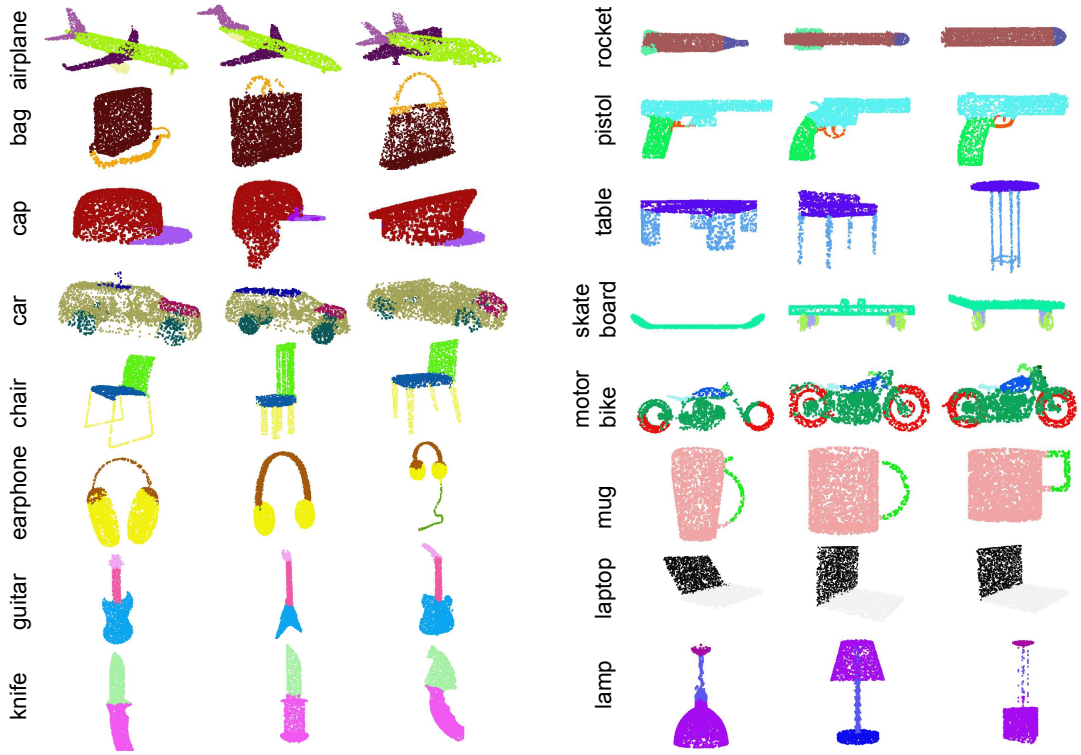

Figure 21. **PointNet segmentation results on complete CAD models.**

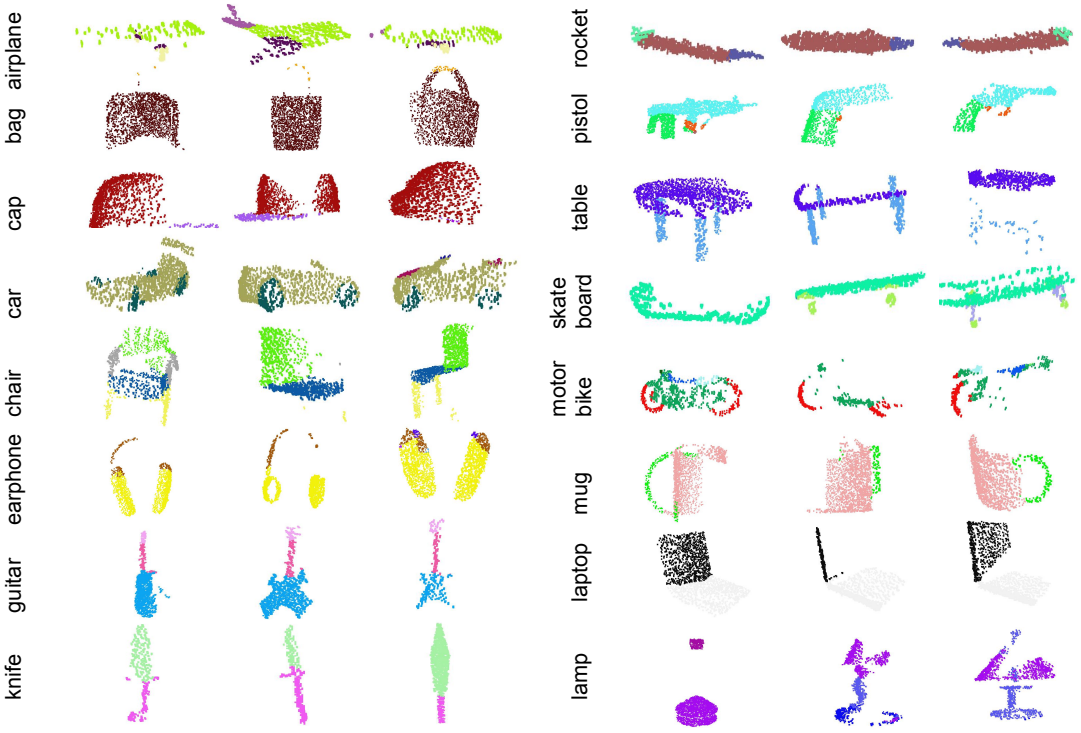

Figure 22. **PointNet segmentation results on simulated Kinect scans.**

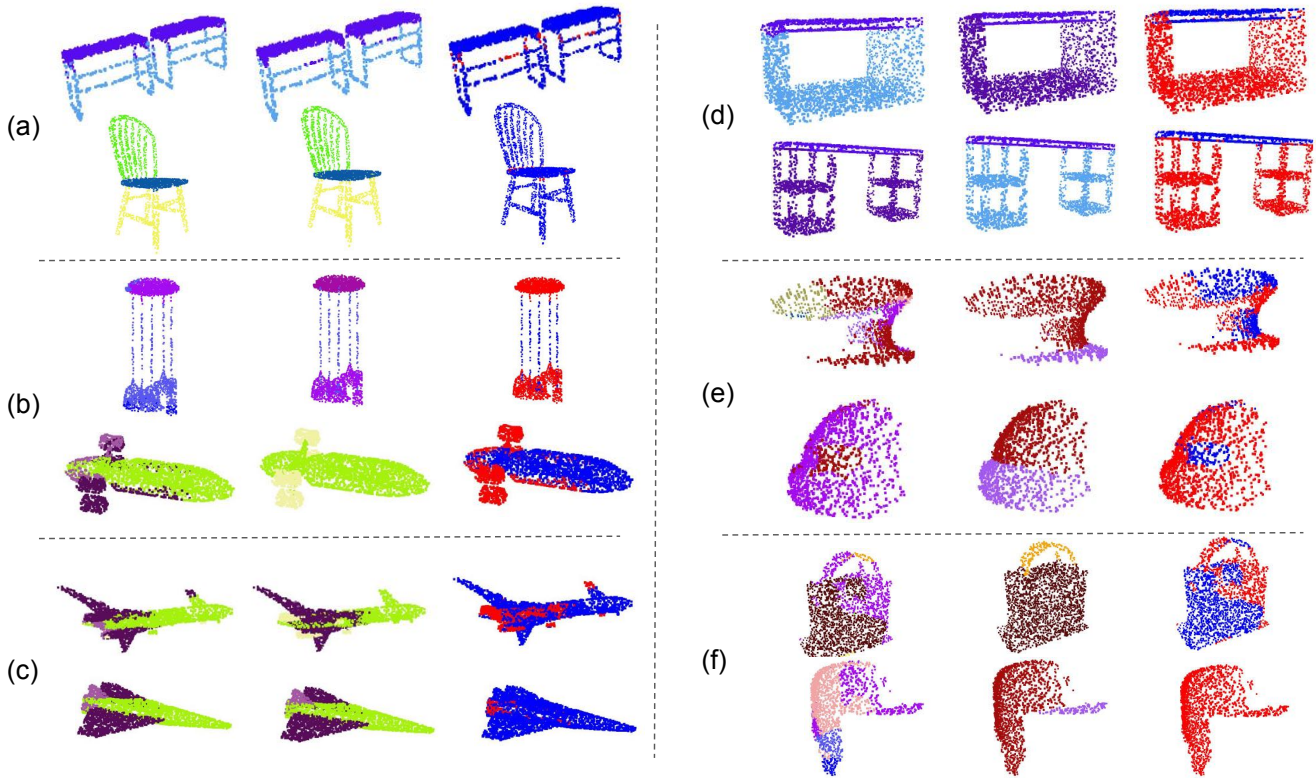

Figure 23. **PointNet segmentation failure cases.** In this figure, we summarize six types of common errors in our segmentation application. The prediction and the ground-truth segmentations are given in the first and second columns, while the difference maps are computed and shown in the third columns. The red dots correspond to the wrongly labeled points in the given point clouds. (a) illustrates the most common failure cases: the points on the boundary are wrongly labeled. In the examples, the label predictions for the points near the intersections between the table/chair legs and the tops are not accurate. However, most segmentation algorithms suffer from this error. (b) shows the errors on exotic shapes. For examples, the chandelier and the airplane shown in the figure are very rare in the data set. (c) shows that small parts can be overwritten by nearby large parts. For example, the jet engines for airplanes (yellow in the figure) are mistakenly classified as body (green) or the plane wing (purple). (d) shows the error caused by the inherent ambiguity of shape parts. For example, the two bottoms of the two tables in the figure are classified as table legs and table bases (category *other* in [29]), while ground-truth segmentation is the opposite. (e) illustrates the error introduced by the incompleteness of the partial scans. For the two caps in the figure, almost half of the point clouds are missing. (f) shows the failure cases when some object categories have too less training data to cover enough variety. There are only 54 bags and 39 caps in the whole dataset for the two categories shown here.

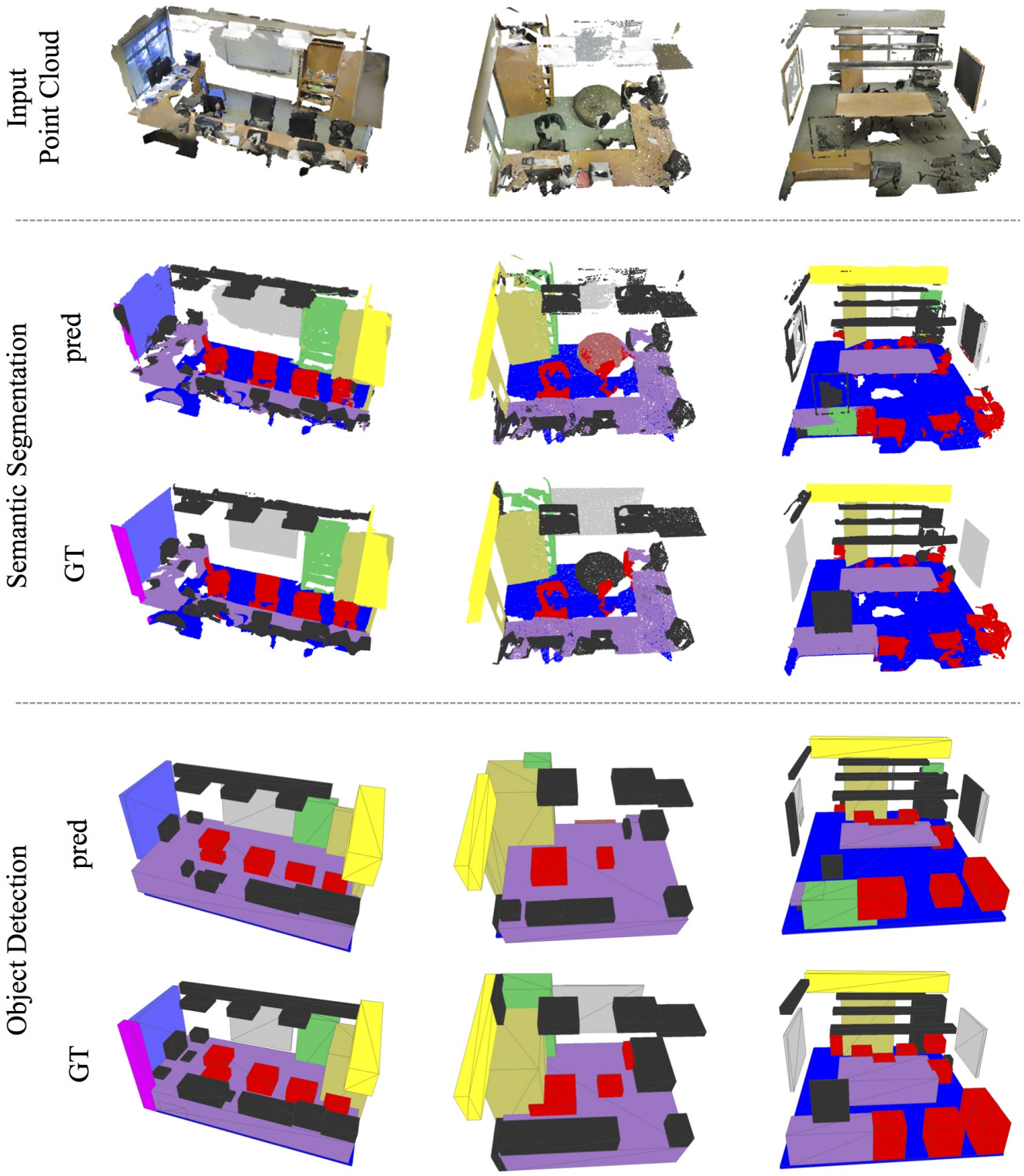

Figure 24. **Examples of semantic segmentation and object detection.** First row is input point cloud, where walls and ceiling are hided for clarity. Second and third rows are prediction and ground-truth of semantic segmentation on points, where points belonging to different semantic regions are colored differently (chairs in red, tables in purple, sofa in orange, board in gray, bookcase in green, floors in blue, windows in violet, beam in yellow, column in magenta, doors in khaki and clutters in black). The last two rows are object detection with bounding boxes, where predicted boxes are from connected components based on semantic segmentation prediction.