3D Graph Neural Networks for RGBD Semantic Segmentation

Xiaojuan Qi[†] Renjie Liao[‡,§] Jiaya Jia[†,♭] Sanja Fidler[‡] Raquel Urtasun[§,‡]
[†] The Chinese University of Hong Kong [‡] University of Toronto
[§] Uber Advanced Technologies Group [♭] Youtu Lab, Tencent
{xjqi, leojia}@cse.cuhk.edu.hk {rjliao, fidler, urtasun}@cs.toronto.edu

Abstract

RGBD semantic segmentation requires joint reasoning about 2D appearance and 3D geometric information. In this paper we propose a 3D graph neural network (3DGNN) that builds a k-nearest neighbor graph on top of 3D point cloud. Each node in the graph corresponds to a set of points and is associated with a hidden representation vector initialized with an appearance feature extracted by a unary CNN from 2D images. Relying on recurrent functions, every node dynamically updates its hidden representation based on the current status and incoming messages from its neighbors. This propagation model is unrolled for a certain number of time steps and the final per-node representation is used for predicting the semantic class of each pixel. We use back-propagation through time to train the model. Extensive experiments on NYUD2 and SUN-RGBD datasets demonstrate the effectiveness of our approach.

1. Introduction

Advent of depth sensors makes it possible to perform RGBD semantic segmentation along with many applications in virtual reality, robotics and human-computer interaction. Compared with the more common 2D image setting, RGBD semantic segmentation can utilize the real-world geometric information by exploiting depth infromation. For example, in Fig. 1(a), given the 2D image alone, the local neighborhood of the red point located on the *table* inevitably includes *microwave* and *counter* pixels. However, in 3D, there is no such confusion because these points are distant in the 3D point cloud, as shown in Fig. 1(b).

Several methods [11, 29, 21, 7] treat RGBD segmentation as a 2D segmentation problem where depth is taken as another input image. Deep convolutional neural networks (CNNs) are applied separately to the color and depth images to extract features. These methods need two CNNs, which double computation and memory consumption. Possible errors stem from missing part of the geometric context information since 2D pixels are associated with 3D ones in the real world. For example, in Fig. 1(c), the two-network model [29] classifies the *table* as part of the *counter*.

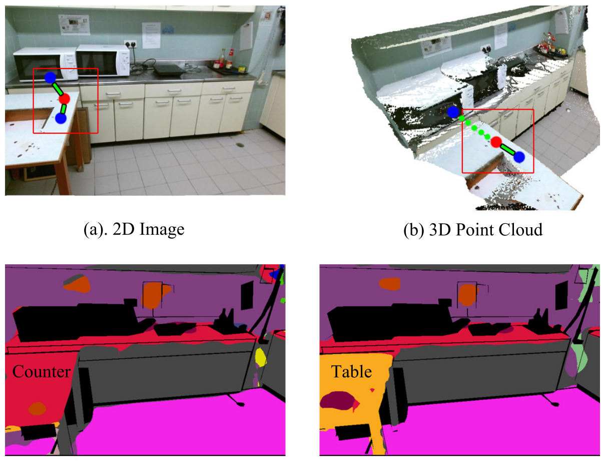

Figure 1. **2D and 3D context**. The solid lines indicate neighbors in 3D while the dotted lines are for neighbors in 2D but not in 3D. (a) Input image (b) 2D image projected into 3D point cloud. (c) Prediction by the two-stream CNN with HHA encoding [29]. (d) Our 3DGNN prediction.

An alternative is to use 3D CNNs [37] in voxelized 3D space. This type of methods has the potential to extract more geometric information. However, since 3D point clouds are quite sparse, effective representation learning from such data is challenging. In addition, 3D CNNs are computationally more expensive than their 2D version, thus it is difficult to scale up these systems to deal with a large number of classes. Anisotropic convolutional neural network [1, 30] provides a promising way to learn filters in non-euclidean space for shape analysis. Yet it faces the same difficulty of scaling-up to perform large-scale RGBD dense semantic segmentation due to complex association of points.

To tackle the challenges above, we propose an end-to-end 3D graph neural network, which directly learns its representation from 3D points. We first cast the 2D pixels into

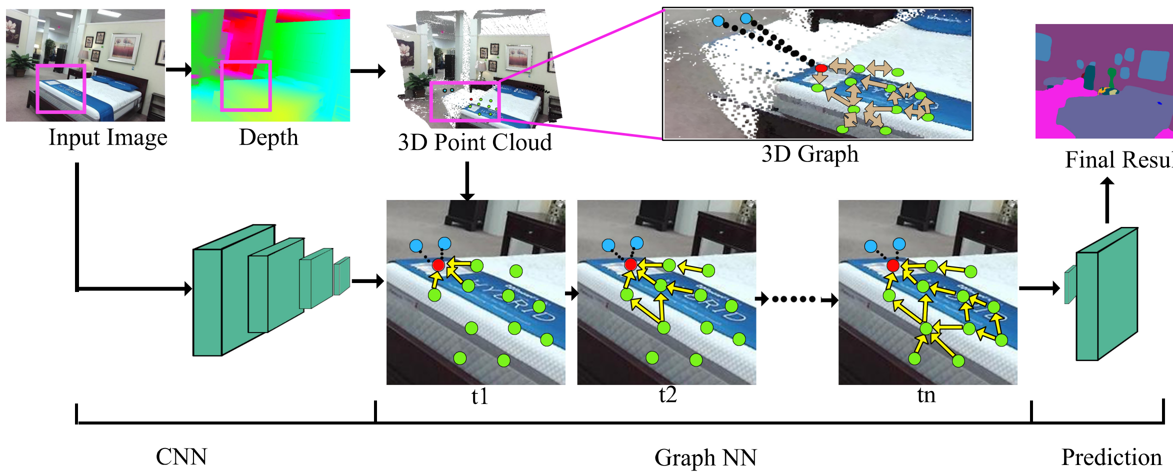

Figure 2. Overview of our 3D graph neural network. The top part of the figure shows the 3D point cloud and a close-up of the constructed graph based on the point cloud. Blue points and the associated black dotted lines represent nodes and edges which exist in the graph constructed from 2D image. It is clear that a graph built from the 3D point cloud encodes geometric information which is hard to be inferred from the 2D image. In the bottom part, we show the sub-graph connected to the red point as an example to illustrate the propagation process. We highlight the source of messages the red point receives in different time steps using yellow edges.

3D based on depth information and associate with each 3D point a unary feature vector, i.e., an output of a 2D segmentation CNN. We then build a graph whose nodes are these 3D points, and edges are constructed by finding the nearest neighbors in 3D. For each node, we take the image feature vector as the initial representation and iteratively update it using a recurrent function. The key idea of this dynamic computation scheme is that the node state is determined by its history state and the messages sent by its neighbors, while taking both appearance and 3D information into consideration.

We use the final state of each node to perform per-node classification. The back-propagation through time (BPTT) algorithm is adopted to compute gradients of the graph neural network. Further, we pass the gradients to the unary CNN to facilitate end-to-end training. Our experimental results show state-of-the-art performance on the challenging NYUD2 and SUN-RGBD datasets.

2. Related Work

2D Semantic Segmentation. Fully convolutional networks (FCN) [29] have demonstrated effectiveness in performing semantic segmentation. In fact, most of the following work [3, 45, 44, 26, 25, 46, 28, 43, 4] is built on top of FCN. Chen *et al.* [3] used dilated convolutions to enlarge the receptive field of the network while retaining dense prediction. Conditional random fields (CRFs) have been applied as post-processing [3] or have been integrated into the network [45] to refine boundary of prediction. Recently, global and local context is modeled in scene parsing [42, 27]. In [44, 27], the context is incorporated with pyramid pooling [12]. Liang *et al.* [22, 23] tackled semantic segmentation as sequence prediction and used LSTM to capture local and global dependencies. Graph LSTM [22] was used to model structured data. However, the update procedure and sequential processing make it hard to scale up the system to large graphs.

RGBD Semantic Segmentation. Compared to the 2D setting, RGBD semantic segmentation has the benefit of exploiting more geometric information. Several methods have encoded the depth map as an image. In [11, 29, 21], depth information forms three channels via HHA encoding: horizontal disparity, height above ground and norm angle. In [7], the depth image is simply treated as a one-channel image. FCNs were then applied to extract semantic features directly on the encoded images.

In [11, 24], a set of 2.5D region proposals are first generated. Each proposal is then represented by its RGB image and the encoded HHA image. Two CNNs were used to extract features separately, which are finally concatenated and passed as input to SVM classification. Besides high computation, separate region proposal generation and label assignment make these systems fragile. The final classification stage could be influenced by errors produced in the proposal stage. Long *et al.* [29] applied FCN to RGB and HHA images separately and fused scores for final prediction. Eigen *et al.* [7] proposed a global-to-local strategy to combine different levels of prediction, which simply extracts features via CNNs from the depth image. The extracted feature is again concatenated with the image one for prediction. Li *et al.* [21] used LSTM to fuse the HHA image and color information. These methods all belong to the category that uses 2D CNNs to extract depth features.

Alternatively, several approaches deal with 2.5D data using 3D voxel networks [37, 41, 36]. Song *et al.* [37] used a 3D dilated voxel convolutional neural network to learn the semantics and occupancy of each voxel. These methods take better advantage of 3D context. However, scaling up to deal with high-resolution and complex scenes is challenging since 3D voxel networks are computationally expensive. Further, quantization of 3D space can lead to additional errors. Other methods [1, 30] learned non-euclidean filters for shape analysis. They typically rely on well-defined point association, e.g., meshes, which are not readily available for complex RGBD segmentation data.

Graph neural networks. In terms of the structure of neural networks, there has been effort to generalize neural networks to graph data. One direction is to apply Convolutional Neural Networks (CNNs) to graphs. In [2, 5, 18], CNNs are employed in the spectral domain relying on the graph Laplacian. While [6] used hash functions so that CNN can be applied to graphs. Another direction is to recurrently apply neural networks to every node of the graph [9, 33, 20, 39], producing "Graph Neural Networks". This model includes a propagation process, which resembles message passing of graphical models [8]. The final learning process of such a model can be achieved by the back-propagation through time (BPTT) algorithm.

3. Graph Neural Networks

In this section, we briefly review Graph Neural Networks (GNN) [9, 33] and their variants, e.g., gated GNN [20], and discuss their relationship with existing models.

For GNNs, the input data is represented as a graph $G = \{V, E\}$, where V and E are the sets of nodes and edges of the graph, respectively. For each node $v \in V$, we denote the input feature vector by $\mathbf{x}_v$ and its hidden representation describing the node's state at time step t by $\mathbf{h}_v^t$. We use Ω_v to denote the set of neighboring nodes of v.

A Graph Neural Network is a dynamic model where the hidden representation of all nodes evolve over time. At time step t, the hidden representation is updated as

$$\begin{aligned} \mathbf{m}_v^t &= \mathcal{M}\left(\{\mathbf{h}_u^t | u \in \Omega_v\}\right) \\ \mathbf{h}_v^{t+1} &= \mathcal{F}\left(\mathbf{h}^t, \mathbf{m}_v^t\right), \end{aligned} \tag{1}$$

where $\mathbf{m}_v^t$ is a vector, which indicates the aggregation of messages that node v receives from its neighbors Ω_v. $\mathcal{M}$ is a function to compute the message and $\mathcal{F}$ is the function to update the hidden state. Similar to a recurrent neural network, $\mathcal{M}$ and $\mathcal{F}$ are feedforward neural networks that are shared among different time steps. Simple $\mathcal{M}$ and $\mathcal{F}$ can be an element-wise summation function and a fully connected layer, respectively. Note that these update functions specify a propagation model of information inside the graph. It is also possible to incorporate more information from the graph with different types of edges using multiple $\mathcal{M}$.

Inference is performed by executing the above propagation model for a certain number of steps. The final prediction can be at the node or at the graph level depending on the task. For example, one can feed the hidden representation (or aggregation of it) to another neural network to perform node (or graph) classification.

Graph Neural Networks are closely related to many existing models, such as conditional random fields (CRFs) and recurrent neural networks (RNNs). We discuss them next. We focussed on pairwise CRFs but note that the connection extends to higher-order models.

Loopy Belief Propagation Inference. We are given a pairwise (often cyclic in practice) CRF whose conditional distribution factorizes as $\log P(Y|I) \propto -\sum_{i \in V} \phi_u(y_i|I) - \sum_{(i,j) \in E} \phi_p(y_i, y_j|I)$ with $Y = \{y_i | i \in V\}$ the set of all labels, I the set of all observed image pixels, and ϕ_u and ϕ_p the unary and pairwise potentials, respectively. One fundamental algorithm to approximate inference in general MRFs/CRFs is loopy belief propagation (BP) [31, 8]. The propagation process is denoted as

$$\beta_{i \to j} = \sum_{y_i} \exp\{-\phi_u(y_i) - \phi_p(y_i, y_j)\} \prod_{k \in \Omega_i / j} \beta_{k \to i} \tag{2}$$

where β is the recursively defined message and the subscription $i \to j$ means the message is sent from node i to j. The message is a vector defined in the same space of output label y_i. The message update is performed either until convergence or when reaching the maximum number of iterations. One can then construct the final marginal probability as $P(y_i) \propto \exp\{-\phi_u(y_i)\} \prod_{j \in \Omega_i} \beta_{j \to i}$. It can be deemed as a special case of the graph neural network built on top of the factor graph. In this case, we only associate hidden representation with the factor node and treat other nodes as dummy in GNN. Then the message β in BP corresponds to the hidden state of the factor node. The message function $\mathcal{M}$ is the product and the update function $\mathcal{F}$ takes the form of Eq. (2).

Mean Field Inference. Mean field inference defines an approximate distribution $Q(Y) = \prod_i Q(y_i)$ and minimizes the KL-divergence $KL(Q||P)$. The fixed-point propagation equations characterize the stationary points of the KL-divergence as

$$Q(y_i) = \frac{1}{Z_i} \exp\left\{ -\phi_u(y_i) - \sum_{j \in \Omega_i} \mathbb{E}_{Q(y_j)}\left[\phi_p(y_j, y_i)\right] \right\}, \tag{3}$$

where Z_i is a normalizing constant and Ω_i is the neighbor of node i. This fixed-point iteration converges to a local minimum [8]. From Eqs. (1) and (3), it is clear that the mean field propagation is a special case of graph neural networks. The hidden representation of node i is just the approximate distribution $Q(y_i)$. $\mathcal{M}$ and $\mathcal{F}$ is the negation of element-wise summation and softmax respectively. $\mathbb{E}_{Q(y_j)}\left[\phi_p(y_j, y_i)\right]$ is the message sent from node j to i. While the messages of CRFs lie in the space of output labels y, GNNs have messages $\mathbf{m}_v^t$ in the space of hidden representations. GNNs are therefore more flexible in terms of information propagation as the dimension of the hidden space can be much larger than that of the label space.

Connection to RNNs. GNNs can also be viewed as a generalization of RNNs from sequential to graph data. When the input data is chain-structured, the neighborhood Ω_v degenerates to the single parent of the current node v. Then a vanilla RNN, e.g., the one used in text modeling [38], is just a graph neural network with a particular instantiation of the message function $\mathcal{M}$ and the update function $\mathcal{F}$.

GNNs are also closely related to Tree-LSTM [39] when the input graph is a tree. In this case, message computation is the summation of hidden representations of all children nodes. The main difference from common graph neural networks is that each child node computes its own copy of the forget gate output – they are aggregated together in their parents' cell memory update.

4. 3DGNN for RGBD Semantic Segmentation

In this section, we propose a special GNN to tackle the problem of RGBD semantic segmentation.

4.1. Graph Construction

Given an image, we construct a directed graph based on the 2D position and the depth information of pixels. Let $[x, y, z]$ be the 3D coordinates of a point in the camera coordinate system and let $[u, v]$ be its projection onto the image according to the pinhole camera model. Geometry of perspective projection yields

$$\begin{aligned} x &= (u - c_x) * z / f_x \\ y &= (v - c_y) * z / f_y, \end{aligned} \tag{4}$$

where f_x and f_y are the focal length along the x and y directions, and c_x and c_y are coordinates of the principal point. To form our graph, we regard each pixel as a node and connect it via directed edges to K nearest neighbors (KNN) in the 3D space, where K is set to 64 in our experiments. Note that this process creates asymmetric structure, i.e., an edge from A to B does not necessarily imply existence of the edge from B to A. We visualize the graph in Fig. 2.

4.2. Propagation Model

After constructing the graph, we use a CNN as the unary model to compute the features for each pixel. We provide details in Section 5. Taking these features as the initial hidden representation of the corresponding node, we encode the appearance information. Given that the 3D graph already encodes the geometric context, this graph neural network exploits both appearance and geometry information. The propagation process is expressed as

$$\begin{aligned} \mathbf{m}_v^t &= \frac{1}{|\Omega_v|} \sum_{u \in \Omega_v} g\left(\mathbf{h}_u^t\right) \\ \mathbf{h}_v^{t+1} &= \mathcal{F}\left(\mathbf{h}^t, \mathbf{m}_v^t\right), \end{aligned} \tag{5}$$

where g is a multi-layer perceptron (MLP). Unless otherwise specified, all instances of MLP that we employ have one layer with ReLU [19] as the nonlinearity. At each time step, every node collects messages from its neighbors. The message is computed by first feeding hidden states to the MLP g and then taking the average over the neighborhood. Then every node updates its hidden state based on previous state and the aggregated message. This process is shown in Fig. 2. We consider two choices of the update function $\mathcal{F}$.

Vanilla RNN Update. We can use a MLP as the update function as

$$\mathbf{h}_v^{t+1} = q(\left[\mathbf{h}_v^t, \mathbf{m}_v^t\right]), \tag{6}$$

where we concatenate the hidden state and message before feeding it to the MLP q. This type of update function is common in vanilla RNN.

LSTM Update. Another choice is to use a long short-term memory (LSTM) [15] cell. This is more powerful since it maintains its own memory to help extract useful information from incoming messages.

4.3. Prediction Model

Assuming the propagation model in Eq. (5) is unrolled for T steps, we now predict the semantic label for each pixel

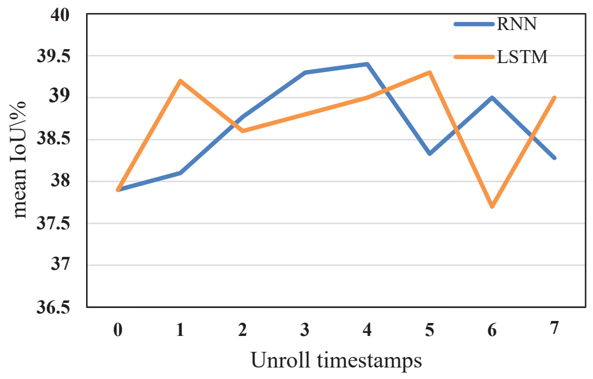

Figure 3. Performance of GNNs in different propagation steps. Blue and orange lines correspond to vanilla RNN and LSTM update respectively. Timestamp 0 represents the CNN baseline.

model	mean IoU%	mean acc%
Gupta *et al.* [11] (2014)	28.6	35.1
Long *et al.* [29] (2015)	34.0	46.1
Eigen *et al.* [7] (2015)	34.1	45.1
Lin *et al.* [26] + ms (2016)	40.6	53.6
HHA + ss	40.8	54.6
3DGNN + ss	39.9	54.0
3DGNN + ms	41.7	55.4
HHA-3DGNN + ss	42.0	55.2
HHA-3DGNN + ms	**43.1**	**55.7**

Table 1. Comparison with state-of-the-arts on NYUD2 test set in 40-class setting. HHA means combining HHA feature [29]. "ss" and "ms" indicate single- and multi-scale test.

model	mean IoU%	mean acc%
Silberman *et al.* [34] (2012)	-	17.5
Ren *et al.* [32] (2012)	-	20.2
Gupta *et al.* [10] (2015)	-	30.2
Wang *et al.* [40] (2015)	-	29.2
Khan *et al.* [17] (2016)	-	43.9
Li *et al.* [21] (2016)	-	49.4
3DGNN + ss	43.6	57.1
3DGNN + ms	**45.4**	**59.5**

Table 2. Comparison with state-of-the-art methods on NYUD2 test set in the 37-class setting. "ss" and "ms" indicate single- and multi-scale test.

in the score map. For node v corresponding to a pixel in the score map, we predict the probability over semantic classes y_v as follows:

$$p_{y_v} = s(\left[\mathbf{h}_v^T, \mathbf{h}_v^0\right]), \tag{7}$$

where s is a MLP with a softmax layer shared by all nodes. Note that we concatenate the initial hidden state, which is the output of the unary CNN to capture the 2D appearance information. We finally associate a softmax cross-entropy loss function for each node and train the model with the back-propagation through time (BPTT) algorithm.

5. Experiments

Datasets. We evaluate our method on two popular RGBD datasets: NYUD2 [34] and SUN-RGBD [35]. NYUD2 contains a total of 1,449 RGBD image pairs from 464 different scenes. The dataset is divided into 795 images from 249 scenes for training and 654 images from 215 scenes for testing. We randomly split 49 scenes from the training set as the validation set, which contains 167 images. The remaining 654 images from 200 scenes are used as the training set. SUN-RGBD consists of 10,335 images, which are divided into 5,285 RGBD image pairs for training and 5,050 for testing. All our hyperparameter search and ablation studies are performed on the NYUD2 validation set.

Unary CNN. For most of the ablation experiments, we use a modified VGG-16 network, i.e., deeplab-LargeFov [3] with dilated convolutions as our unary CNN to extract the appearance features from the 2D images. We use the *fc7* feature map. The output feature map is of size $H \times W \times C$ where H, W and C are the height, width and the channel size respectively. Note that due to the stride and pooling of this network, H and W are $1/8$ of the original input in terms of size. Therefore, our 3D graph is built on top of the downsampled feature maps.

To further incorporate contextual information, we use global pooling [27] to compute another C-dim vector from the feature map. We then append the vector to all spatial positions, which result in a $H \times W \times 2C$ feature map. In our experiment, $C = 1024$ and a 1×1 convolution layer is used to further reduce the dimension to 512. We also experimented by replacing the VGG-net with ResNet-101 [14] or by combining it with the HHA encoding.

Implementation Details. We initialize the unary CNN from the pre-trained VGG network in [3]. We use SGD with momentum to optimize the network and clip the norm of the gradients such that it is not larger than 10. The initial learning rates of the pre-trained unary CNN and GNN are 0.001 and 0.01 respectively. Momentum is set to 0.9. We initialize RNN and LSTM update functions of the Graph Neural Network using the MSRA method [13]. We randomly scale the image in scaling range $[0.5, 2]$ and randomly crop 425×425 patches. For the multi-scale testing, we use three scales 0.8, 1.0 and 1.2. In the ResNet-101 experiment, we modified the network by reducing the overall stride to 8 and by adding dilated convolutions to enlarge the receptive field.

We adopt two common metrics to evaluate our method: mean accuracy and mean intersection-over-union (IoU).

5.1. Comparison with State-of-the-art

In our comparison with other methods, we use the vanilla RNN update function in all our experiments due to its efficiency and good performance. We defer the thorough ablation study to Section 5.2.

NYUD2 dataset. We first compare with other methods in the NYUD2 40-class and 37-class settings. As shown in Tables 1 and 2 our model achieves very good performance in both settings. Note that Long *et al.* [29] and Eigen *et al.* [7] both used two-VGG networks with HHA image/depth encoding whereas we only use one VGG network to extract appearance features. The configuration of Lin *et al.* [26] is a bit different since it only takes the color image as input and builds a complicated model that involves several VGG networks to extract image features.

Results in these tables also reveal that by combining VGG with the HHA encoding features [29] as the unary model, our method further improves in performance.

model	mean IoU%	mean acc%
Song *et al.* [35] (2015)	-	36.3
Kendall *et al.* [16] (2015)	-	45.9
Li *et al.* [21] (2016)	-	48.1
HHA + ss	41.7	52.3
ResNet-101 + ss	42.7	53.5
3DGNN + ss	40.2	52.5
3DGNN + ms	42.3	54.6
HHA-3DGNN + ss	42.0	55.2
HHA-3DGNN + ms	43.1	55.7
ResNet-101-3DGNN + ss	44.1	55.7
ResNet-101-3DGNN + ms	**45.9**	**57.0**

Table 3. Comparison with other methods on SUN-RGBD test set. “ResNet-101” exploits ResNet-101 as the unary model. “HHA” denotes a combination of the RGB image feature with HHA image feature [29]. “ss” and “ms” indicate single-scale and multi-scale.

Propagation Step	Unary CNN	2DGNN	3DGNN
0	37.9	-	-
1	-	37.8	38.1
3	-	**38.4**	39.3
4	-	38.0	**39.4**
6	-	38.1	39.0

Table 4. 2D VS. 3D graph. Performance with different propagation steps on NYUD2 validation set is listed.

Dataset	network	mean IoU%	mean acc%
NYUD2-40	Unary CNN	37.1	51.0
	2DGNN	38.7	52.9
	3DGNN	**39.9**	**54.0**
NYUD2-37	Unary CNN	41.7	55.0
	3DGNN	**43.6**	**57.0**
SUNRGBD	Unary CNN	38.5	49.4
	2DGNN	38.9	50.3
	3DGNN	**40.2**	**52.5**

Table 5. Comparison with the unary CNN on NYUD2 and SUN-RGBD test set.

SUN-RGBD dataset. We also compare these methods on SUN-RGBD in Table 3. The performance difference is significant. Note that Li *et al.* [21] also adopted the Deeplab-LargeFov network for extracting image feature and a separate network for HHA encoded depth feature extraction. Our single 3DGNN model already outperforms previous ones by a large margin. Combining HHA features or replacing VGG-net with ResNet-101 further boost the performance. These gains showcase that our method is effective in encoding 3D geometric context.

5.2. Ablation Study

In this section, we conduct an ablation study on our NYUD2 validation set to verify the functionality of different parts of our model.

Propagation Steps. We first calculate statistics of the constructed 3D graphs. The average diameter of all graphs is 21. It corresponds to the average number of propagation steps to traverse the graph. The average distance between any pair of nodes is 7.9. We investigate how the number of propagation steps affects the performance of our model in Fig. 3. The performance, i.e., mean IoU, gradually improves when the number of propagation step increases.

The oscillation when the propagation step is large might relate to the optimization process. We found that 3 to 6 propagation steps produce reasonably good results. We also show the segmentation maps using different propagation steps in Fig. 4. Limited by the receptive field size, the unary CNN often yields wrong predictions when the objects are too large. For example, in the first row of Fig. 4, the table is confused as the counter. With 4 propagation steps, our prediction of the table becomes much more accurate.

Update Equation. Here we compare the two update equations described in Section 4.2. As shown in Fig. 3, the vanilla RNN performs similarly to LSTM. The computation complexity of LSTM update is much larger than the vanilla RNN. According to this finding, we stick to the Vanilla RNN update in all our experiments.

2D VS. 3D Graph. To investigate how much improvement the 3D graph additionally brings, we compare with 2D graphs that are built on 2D pixel positions with the same KNN method. We conduct experiments using the same Graph Neural Network and show the performance of different propagation steps in Table 4. Results on the whole test set is shown in Table 5. They indicate that with 3DGNN, more 3D geometric context is captured, which in turn makes prediction more accurate. Another interesting observation is that even the simple 2DGNN still outperforms the method incorporating the unary CNN.

Performance Analysis. We now compare our 3DGNN to the unary CNN in order to investigate how GNN can be enhanced by leveraging 3D geometric information. The results based on the single-scale data input are listed in Table 5. Our 3DGNN model outperforms the unary and 2DGNN models, which again supports the fact that 3D context is important in semantic segmentation.

We further break down the improvement in performance for each semantic class in Fig. 5. The statistics show that our 3DGNN outperforms the unary CNN by a large margin for classes like cabinet, bed, dresser, and refrigerator. This is likely because these objects are easily misclassified as their surroundings in the 2D image. However, in 3D space, they typically have rigid shapes and the depth distribution

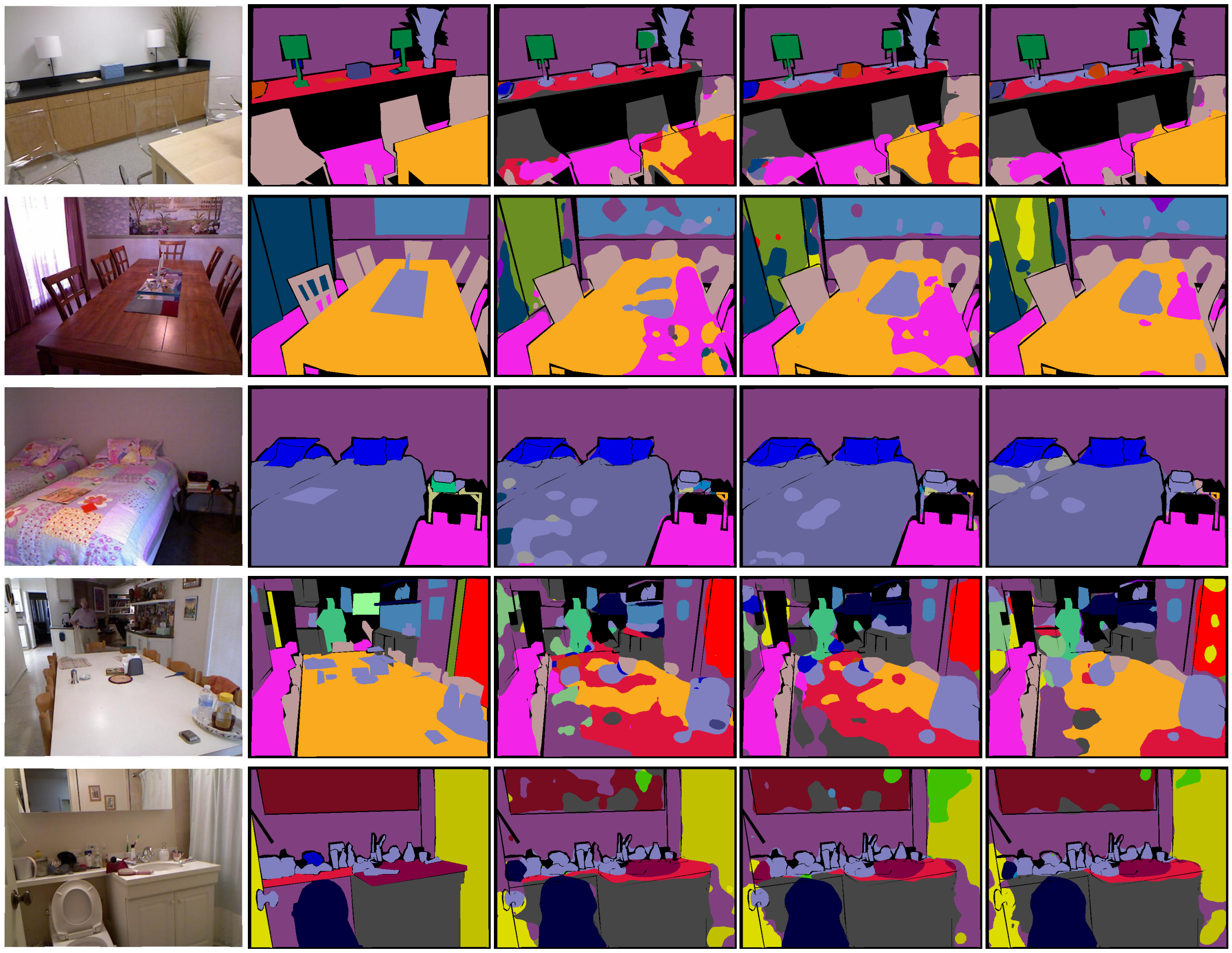
(a) Original Image (b) Ground Truth (c) Unary CNN (d) Propagation Step 1 (e) Propagation Step 4

Figure 4. Influence of different propagation steps on NYUD2 validation set.

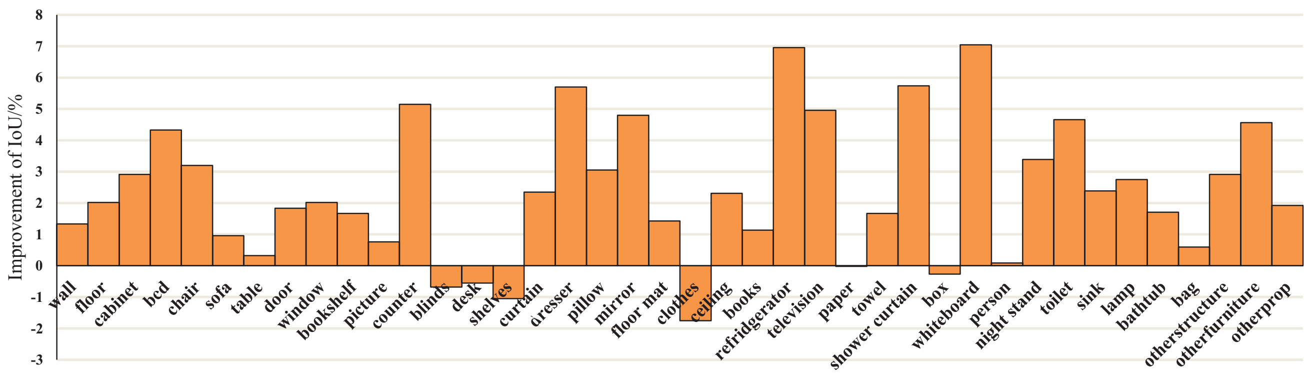

Figure 5. Per-class IoU improvement of 3DGNN over unary CNN.

is more consistent, which makes the classification task relatively easier to tackle.

To better understand what contributes to the improvement, we analyze how the performance gain varies for different sizes of objects. In particular, for each semantic class, we first divide the ground truth segmentation maps into a set of connected components where each component is regarded as one instance of the object within that class. We then count the sizes of object instances for all classes. The range of object sizes is up to $10,200$ different values in

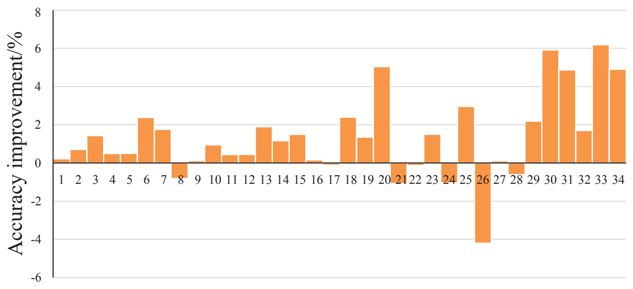

Figure 6. Performance gain as a function of sizes on the NYUD2 test set. Each bin is of size 3,000.

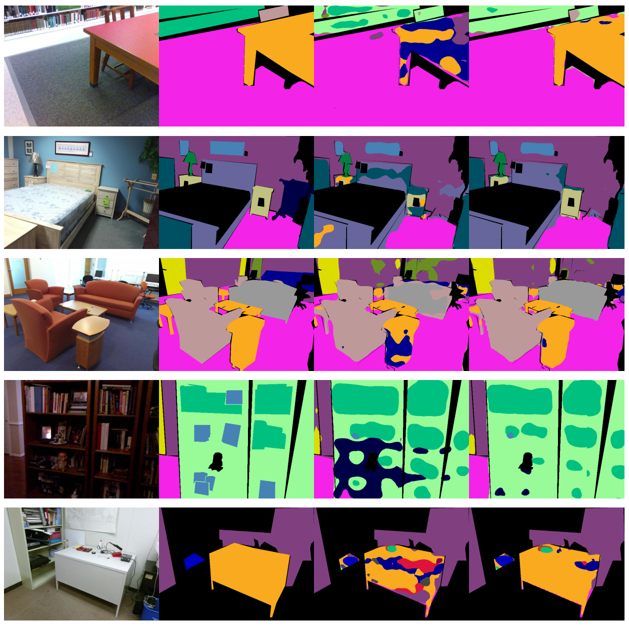
(a) Image (b) Ground Truth (c) Unary CNN (d) Ours

Figure 7. Unary CNN vs 3DGNN on SUNRGBD test set.

terms of the number of pixels. We divide them into 34 bins with bin-width of $3,000$. We record the average improvement of prediction accuracy for object instances in each bin.

As shown in Fig. 6, our 3DGNN handles best large- and middle-size objects rather than small ones. This result proves that our 3DGNN can overcome the limited size of the receptive field of unary CNN and captures the long-range dependency in images.

Qualitative Analysis. We show example results from our model on the SUN-RGBD dataset in Fig. 7 and compare it with the unary CNN. One can see that our 3DGNN exploits 3D geometric context information and learns a better representation for classification compared to the unary CNN. Results in Fig. 7 show that, for example, the segmentation of table, bed frame, night stand, sofa and book shelf have much better shapes and more accurate boundaries when using our method.

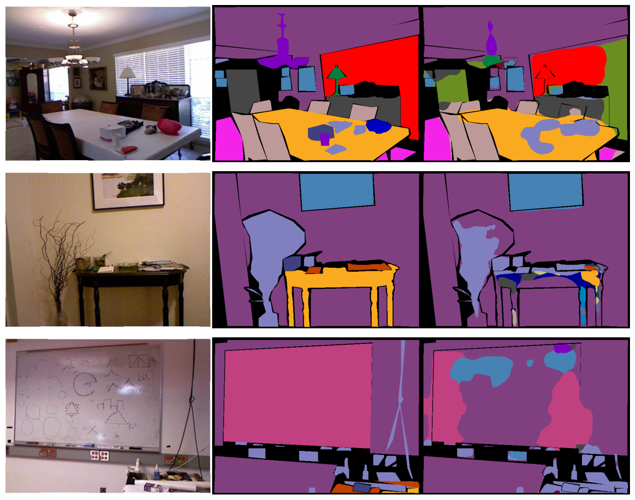
(a) Original Image (b) Ground Truth (c) Ours

Figure 8. Failure cases on NYUD2 test set

Failure Case Analysis. Finally, we show and analyze representative failure cases of our model. First, our model sometimes fails to make good prediction when objects have similar depth, especially for small objects. In the first row of Fig. 8, the lamp is misclassified as the blinds because of this fact. The second type of failure case is due to objects with complex shapes. Discontinuous depth can make 3D neighborhood quite noisy. As shown in the second row of Fig. 8, the table is recognized as pieces of other objects. Moreover, when both the 2D appearance and the 3D context of two objects are similar, our method does not work well. An example is the white board confused as the white wall, shown in the third row of Fig. 8.

6. Conclusion

We presented a novel 3D Graph Neural Network for RGBD semantic segmentation. The Graph Neural Network is built on top of points with color and depth extracted from RGBD images. Our 3DGNN leverages both the 2D appearance information and 3D geometric relations. It is capable of capturing the long range dependencies within images, which has been difficult to model in traditional methods. A variety of empirical results show that our model achieves good performance on standard RGBD semantic segmentation benchmarks. In the future, we plan to investigate feedback to adjust the structure of the constructed graphs.

7. Acknowledgements

This work is in part supported by NSERC, CFI, ORF, ERA, CRC as well as Research Grants Council of the Hong Kong SAR (project No. 413113). We also acknowledge GPU donations from NVIDIA.

References

[1] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *NIPS*, pages 3189–3197, 2016. 1, 3

[2] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *ICLR*, 2014. 3

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *ICLR*, 2015. 2, 5

[4] J. Dai, K. He, and J. Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *ICCV*, 2015. 2

[5] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016. 3

[6] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2015. 3

[7] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015. 1, 2, 5

[8] N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 2004. 3

[9] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IJCNN*, 2005. 3

[10] S. Gupta, P. Arbeláez, R. Girshick, and J. Malik. Indoor scene understanding with rgb-d images: Bottom-up segmentation, object detection and semantic segmentation. *IJCV*, 2015. 5

[11] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from rgb-d images for object detection and segmentation. In *ECCV*, 2014. 1, 2, 5

[12] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014. 2

[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 5

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997. 4

[16] A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv*, 2015. 6

[17] S. H. Khan, M. Bennamoun, F. Sohel, R. Togneri, and I. Naseem. Integrating geometrical context for semantic labeling of indoor scenes using rgbd images. *IJCV*, 2016. 5

[18] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017. 3

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 4

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *ICLR*, 2016. 3

[21] Z. Li, Y. Gan, X. Liang, Y. Yu, H. Cheng, and L. Lin. Lstm-cf: Unifying context modeling and fusion with lstms for rgb-d scene labeling. In *ECCV*, 2016. 1, 2, 5, 6

[22] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan. Semantic object parsing with graph lstm. In *ECCV*, 2016. 2

[23] X. Liang, X. Shen, D. Xiang, J. Feng, L. Lin, and S. Yan. Semantic object parsing with local-global long short-term memory. In *CVPR*, 2016. 2

[24] D. Lin, S. Fidler, and R. Urtasun. Holistic scene understanding for 3d object detection with rgbd cameras. In *ICCV*, 2013. 2

[25] G. Lin, A. Milan, C. Shen, and I. Reid. Refinenet: Multi-path refinement networks with identity mappings for high-resolution semantic segmentation. *arXiv*, 2016. 2

[26] G. Lin, C. Shen, A. van den Hengel, and I. Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *CVPR*, 2016. 2, 5

[27] W. Liu, A. Rabinovich, and A. C. Berg. Parsenet: Looking wider to see better. *ICLR Workshop*, 2016. 2, 5

[28] Z. Liu, X. Li, P. Luo, C.-C. Loy, and X. Tang. Semantic image segmentation via deep parsing network. In *ICCV*, 2015. 2

[29] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1, 2, 5, 6

[30] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *ICCV Workshops*, pages 37–45, 2015. 1, 3

[31] J. Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, 1988. 3

[32] X. Ren, L. Bo, and D. Fox. Rgb-(d) scene labeling: Features and algorithms. In *CVPR*, 2012. 5

[33] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE TNN*, 2009. 3

[34] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 5

[35] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR*, 2015. 5, 6

[36] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *CVPR*, 2016. 3

[37] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. *arXiv*, 2016. 1, 3

[38] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011. 4

[39] K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *ACL*, 2015. 3, 4

[40] A. Wang, J. Lu, J. Cai, G. Wang, and T.-J. Cham. Unsupervised joint feature learning and encoding for rgb-d scene labeling. *TIP*, 2015. 5

[41] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015. 3

[42] J. Yao, S. Fidler, and R. Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation. In *CVPR*, 2012. 2

[43] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv*, 2015. 2

[44] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *arXiv*, 2016. 2

[45] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015. 2

[46] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. 2