3D Object Recognition and Semantic Segmentation

Jianbo Shi, Professor, CIS, GRASP Lab
Ty Nguyen, PhD, CIS, GRASP Lab
Haoyuan Zhang, MSE, Robotcis, GRASP Lab

December 21, 2017

Abstract

This is the latest project report version of the academic proposal project, our topic is 3D object recognition and semantic segmentation. There will be three sections,

1. Brief introduction of proposal projects.
2. The network pipeline and detailed approaches.
3. The completed tasks and corresponding results.
4. Next steps and future work.

1 Brief Introduction

Over the last decade, there have been significant achievements on 2D image segmentation thanks to the emergence of deep neural networks. However, segementation on 3D scene is still a challenging problem due to the complexity of 3D object representations as well as occlusions and cluttered scenes.

In this work, we aim to address the 3D object segmentation using both 3D point cloud and 2D views. These inputs can be obtained from a single RGBD camera or a fusion of a RGB camera and a depth sensor. These settings are more and more ubiquitous in robotics thanks to the decrease in price of sensors such as RGB, RGBD cameras, lidar and stereo cameras.

The input of our raised network is a 3D raw point cloud, we first convert that into a voxel data format which contains higher resolution information and less noise, then use a series of camera models with different poses to capture multiple 2D views of the voxel objects, we input those images into our Faster-RCNN based network for the classification and 2D bounding box estimation tasks. Finally, given multiple 2D bboxes and corresponding camera pose information, we build a 3D bounding box in the original 3D object and showing the predicted label as well.

2 Approaches

Here we propose an architecture which handles the 3D raw point cloud as input and the outputs are detection class label as well as 3D bounding box estimation. Below figure shows the diagram of the pipeline.

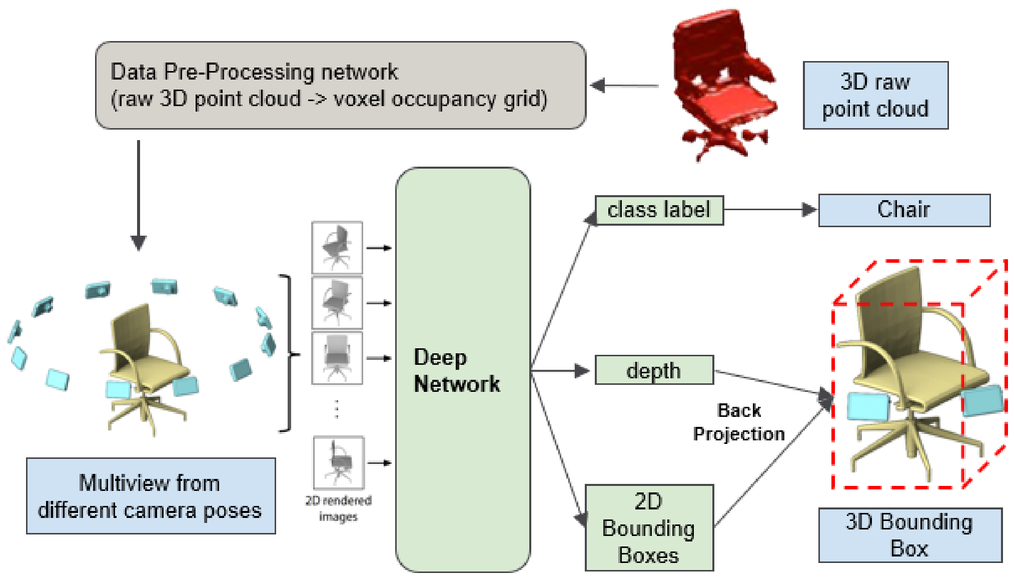


Figure 1: 3D Object Segmentation Pipeline

Accordingly, we separate the whole task into three sub-sections:

1. 3D Data Pre-processing, including the Voxel generation and 2D multiple views generation.
2. The deep network construction to estimate the object class label, 2D bounding boxes and corresponding depth maps based on multiple views input.
3. 3D Bounding box estimation.

Below show more detailed explanations about strategies that we use to achieve the performance.

2.1 3D Data Pre-processing

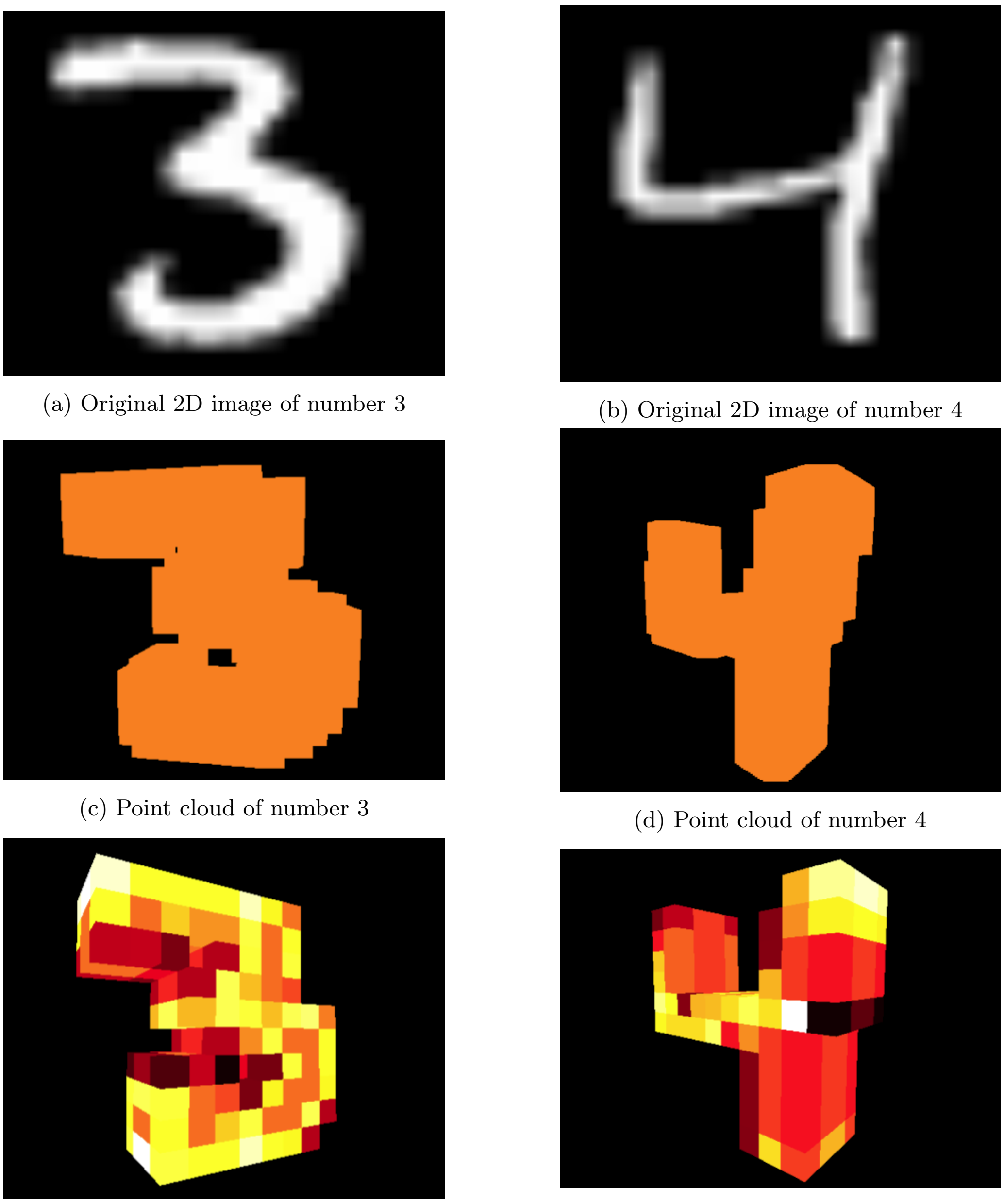
The input of our model is the 3D raw point cloud while the deep network is aimed to take multiple 2D views as input. Therefore, it's necessary to convert the 3D raw point cloud into 2D views representations. This task is achieved by two steps:

1. Voxel grid cell generation.
2. 2D views capture and generation.

2.1.1 Voxel Generation

The reason why we use the Voxel grid cell to represent the 3D raw point cloud is that there contains much noise of the original dataset, however, the Voxel will be much cleaner which can do a lot of favor to the following deep network training such as increasing the estimation accuracy as well as speeding up the process.

So as to faciliate the computation as well as make it easier to deal with bounding boxes, we voxelize the point clouds. Figure 2 shows some example of the generated voxels.



(a) Original 2D image of number 3

(b) Original 2D image of number 4

(c) Point cloud of number 3

(d) Point cloud of number 4

(e) Constructed voxel of number 3

(f) Constructed voxel of number 4

Figure 2: An example of the generated voxel of numbers 3 and 4

2.1.2 2D Views Generation

In order to generate 2D views, we use pinhole camera model with the intrinsics parameters as follows:

$$K = \begin{bmatrix} 200 & 0 & 30 \\ 0 & 200 & 30 \\ 0 & 0 & 1 \end{bmatrix}$$

The image size is set to be $(128, 128)$ and can be changed later. The extrinsics camera parameters can be represented as $[R|T]$, where R is the rotation matrix and T is the transition matrix. To obtain different 2D views, we just need to choose different set of these parameters. To simplify the problem, we chose a fix transition why changing rotation matrices.
The rotation matrix can be broken down into $R = R_z R_y R_x$, where R_z, R_y, R_x are rotation matrices around z, y and x axes respectively.

Given the above camera parameters, a point $A = [X, Y, Z, 1]^T$ in 3D space can be projected onto the camera coordinate according to the following equation:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \lambda K[R\ T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where λ is the scale factor. To obtain the 2D image, we project all points in the voxel grid into the camera plane and do interpolation.

As a consequence, we can also obtain the 2D bounding box from 3D bounding box for training data by projecting corners of the 3D bounding box into 2D image plane and find the convex box that contains of these corners on 2D plane.

Figure 3 shows an example of 2D projection, from different camera angles, of the voxel of number 5. Here $Pi = (Roll, Pitch, Yaw)$ in degree unit, those angles are respected to the **red cross** which is the center of voxel grid.

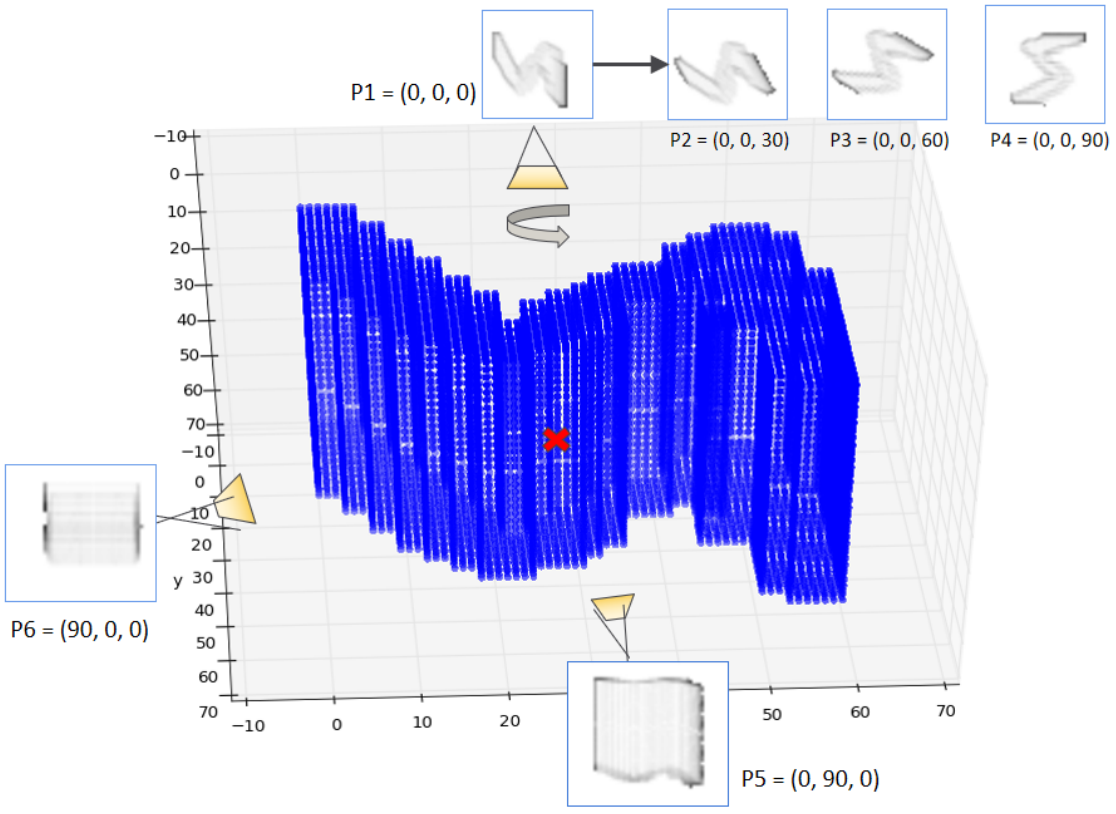


Figure 3: 2D multiple views generation

Here we capture total six 2D views from each converted Voxel model due to the computational complexity concerns. In addition, four of them only rotate yaw angles of camera, one in roll domain and the other one in pitch domain. The reason why we choose this kind of view pose combination is that, the 3D object lies down on the xy plane such that the up-top cameras can capture more information of the object which can do a favor to the object classification, and two side views are essential to estimate the 3D bounding box.

2.2 Deep Network Architecture

Here we propose an deep network, taking those captured 2D views as input, and the outputs include the estimated object class label and 3D bounding box. Below diagram shows the detailed architecture of our deep network.

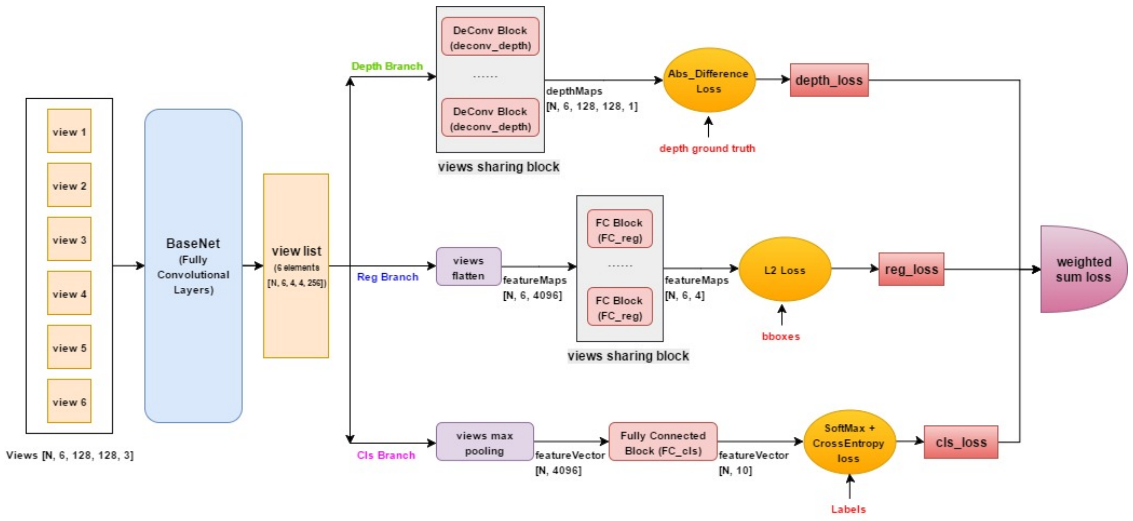


Figure 4: Deep network architecture

More specifically, all 2D views will share the same BaseNet(Fully Convolutional Layers) which is used to extract features. After feed forwarding the BaseNet, we can get a list of views. For the next step, we separate the generated feature maps into three branches:

- **cls**: the object classification branch
- **reg**: the region regression branch to estimate the 2D bbox
- **depth**: the depth map estimation branch

Below three subsections will cover more detailed explanations for those branches.

2.2.1 Classification Branch (cls)

The key point of this cls branch is called the **Views Max Pooling**. For all feature maps in the *view_list* (in our case the number is 6), we first flatten them to become feature vectors, then concatenate together in the view number dimension, finally implement the max view pooling to make it into a single feature vector for the following Fully Connected Block *FC_cls*. The output of this branch is a feature vector with 10 dimension which represents the predicted probability of each class (from digit 0 to 9). Here we use the *softmax* and *Cross-entropy* to compute **cls_loss**.

2.2.2 Regression Branch (Reg)

In the regression branch, our strategy is pretty straightforward. We flatten those feature maps of different views into feature vector form since we use 4d vector to represent the 2D bbox information, $[min\_rowind, max\_rowind, min\_colind, max\_colind]$. Then we feed all feature vectors into a fully connected block which is *FC_reg*. Here, the *FC_reg* block will be shared by all feature vectors which can be simply achieved by using the for loop. This view sharing strategy is well used to handle multiple views issue since the functional block can learn from the all views information. Finally, we use the *L2_loss* to compute the **reg_loss**.

2.2.3 Depth Branch (Depth)

We still use the views sharing block in this depth branch. Inn order to keep the spatial information of the original object structure, we take a reference to the semantic segmentation network, use the *Deconvolutional Block* to recover the feature maps into the same shape as the ground truth depth map (here the shape is $[128, 128, 1]$), then we implement the pixelwise loss computation and use the *Abs_difference* function to get **dept_loss**. In addition, we only consider the values that locate inside the 2D bounding box region and ignore backgrounds, which can make the algorithm efficient and also bring better performance.

In the output side, we utilize the multi-task loss function, sum up three branches losses with different weight to train the whole structure in a single stage.

2.3 3D Bounding Box Estimation

There are two main steps to estimate 3D bounding box.

1. Back-project 2D estimated boxes into 3D.
2. Finding the minimal 3D bounding box.

Actually, given the depth and 2D bounding box prediction results for each captured view, combined with corresponding camera intrinsic and extrinsic matrices information, we can back project those bounding boxes into the physical world frame for sure. The below figure shows more intuitive back-project operation.

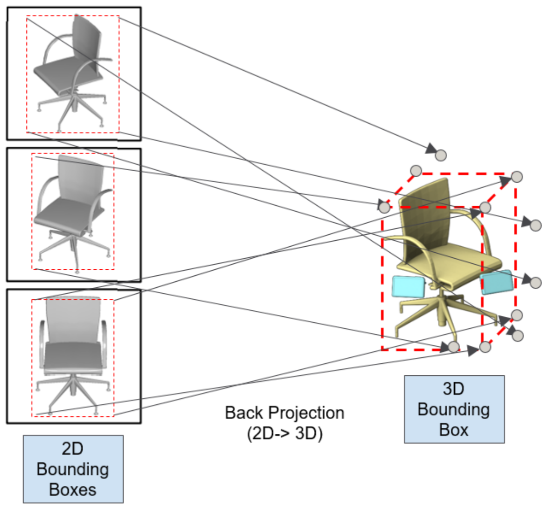


Figure 5: 2D boxes back-projection

However, the challenging of the back-projection is, when capturing those 2D views, we also lose some geometric information such like the physical distance and real angle. Therefore, the 4 back-projected corners of each 2D box may not lie on the object such that their depth estimations are not correct.

One alternative solution is, we use the depth of the closest point on the circle that cover 70% area of the object, and generate a plane at that depth distance which is also tangent to the object, then we arrange those 4 back-projected points on the plane as our estimation.

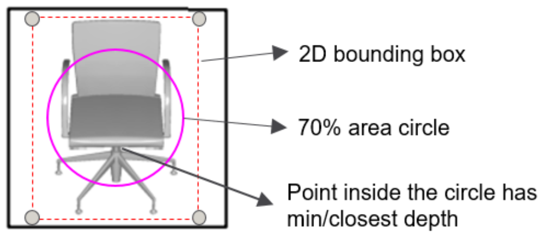


Figure 6: Find the minimal depth value

After achieving back-projection for all views' boxes, we need to generate a 3D bounding box to wrap the original 3D objects. Here we use a straightforward algorithm:

1. Find a convex hull of all corner points.
2. Find minimum-volume box supported by this convex hull.

Feel free to check the below link to get more ideas of the 'Min-Volume Convex Hull Estimation' algorithm. *https://www.geometrictools.com/Documentation/MinimumVolumeBox.pdf*

3 Achieved Tasks and Results

We have succeed to construct the whole pipeline and use the 3D MNIST dataset to train the deep network, finally, we can complete the object classification and 3D bounding box estimation tasks.

More specifically, currently our training dataset contains total **1472** 3D raw digits, and use **448** digits for testing, those can provide reasonable training and test results so far. Later we will try to generate more data to help the network training. In addition, we capture 6 views of each 3D object, later we can modify the number and pose of cameras in order to handle more complicated dataset. Below figures show our training process and test results for each branch (cls, reg, depth) respectively.

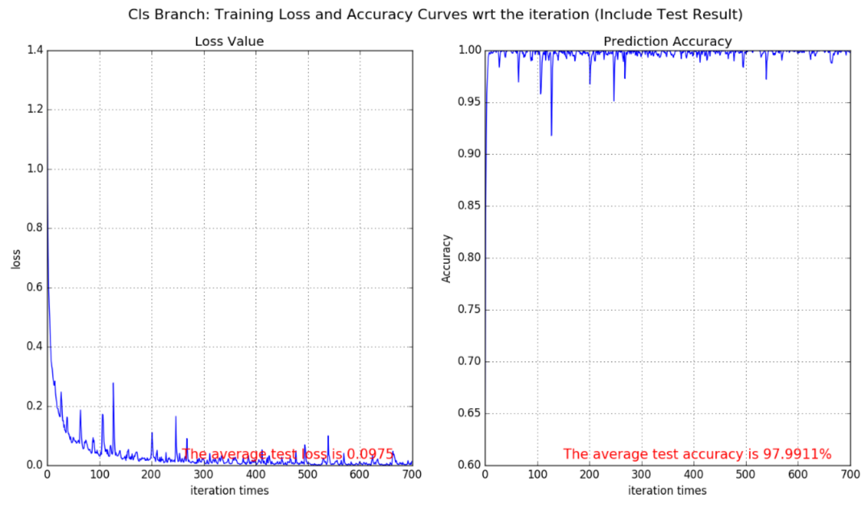


Figure 7: classification branch accuracy and loss

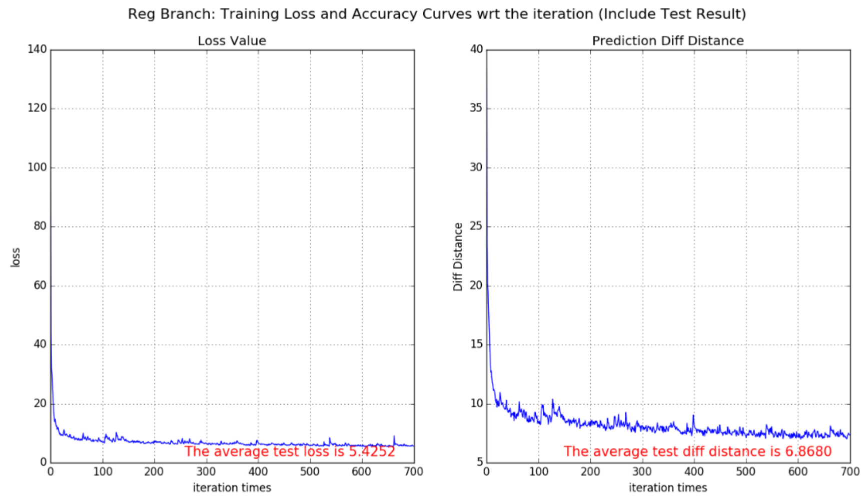


Figure 8: 2D bbox regression branch accuracy and loss

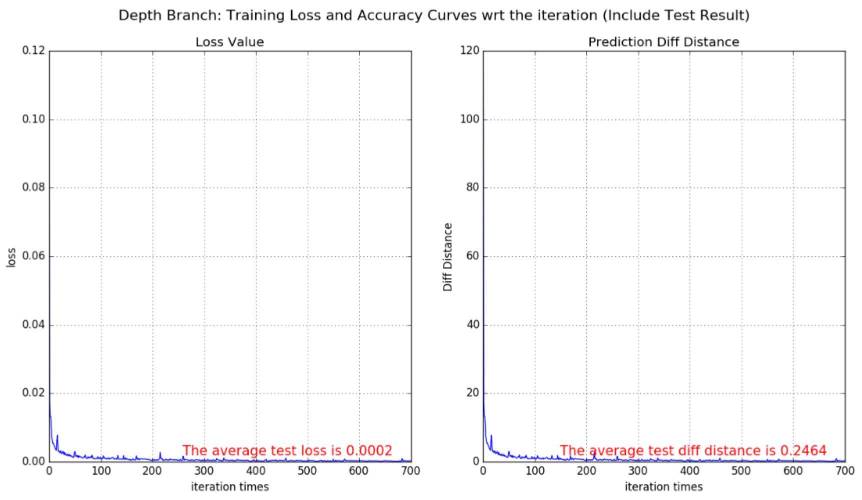
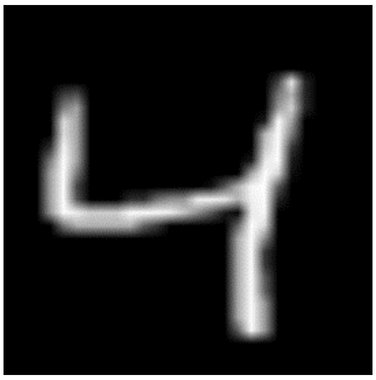


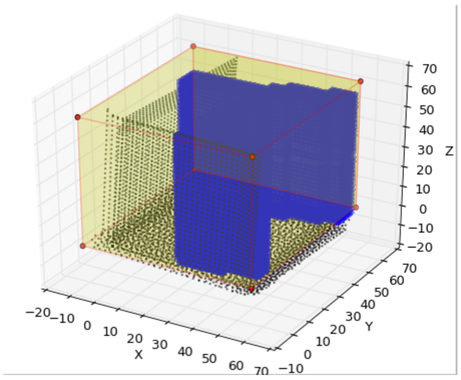
Figure 9: depth estimation branch accuracy and loss

Overall, all three branches are trained reasonable. In the classification branch, the curve will fluctuate a lot over iterations, while the regression and depth branch keeps more stable, the reason should be that the latter two branches dominate the multi-task loss. It's better to try different weights combinations for three losses to figure out which mode should be more optimal.

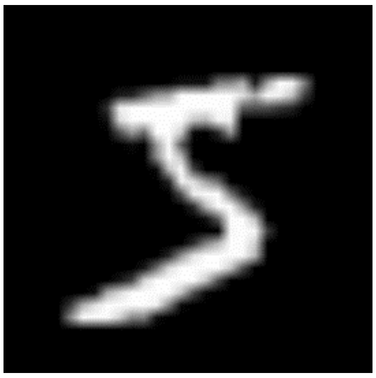
Below figures are some samples we generate to show the 3D bounding box estimation results.



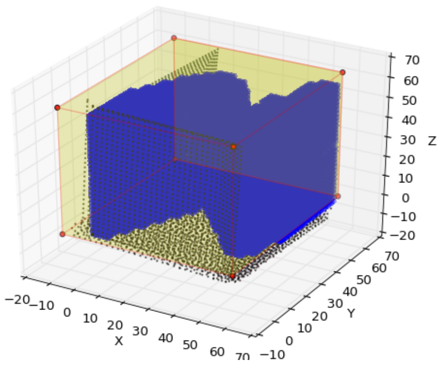
(a) Original 2D image of number 4



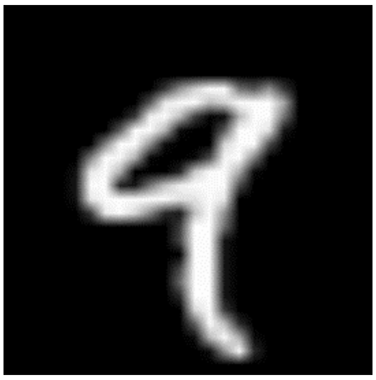
(b) 3D bbox estimation of digit 4



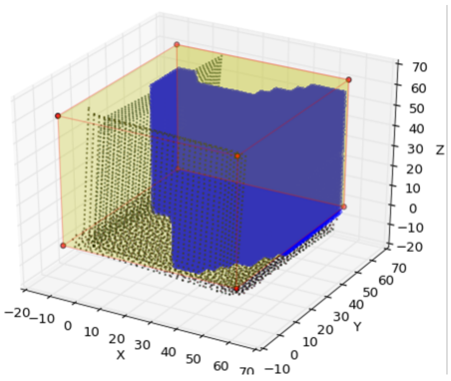
(c) Original 2D image of number 5



(d) 3D bbox estimation of digit 5



(e) Original 2D image of number 9



(f) 3D bbox estimation of digit 9

Figure 10: An example of the generated 3D bbox

4 Future Work

Although our model can work well for the 3D MNIST dataset currently, there still exist a lot of tasks to be achieved for a better and robust 3D object detection and segmentation algorithm.

4.1 More Advanced Deep Network

The current deep network can truly work and make all three branches converge, however, there still exist problems, such as the fluctuation in the classification branch, and the loss of region regression branch is not good enough. Therefore, we can consider below several points to make our deep network more robust.

- Update the training strategy instead of the simple one-stage training with the multi-tasks loss, and it's also necessary to figure out the optimal weighted combinations of three branch losses.

- Update the 2D bounding box regression branch to include the spatial information instead of flattening the feature map directly. (Similar idea as RPN which users the mask and anchor strategies to compute the regression loss).
- Use the more powerful and stable Convolutional blocks such as the ResNet.

4.2 More Complicated Scenarios

Currently we only use the 3D MNIST data to train the network which is quite simple, and also exclude background in our scenario. In order to make our model general and robust to more complicated cases.

- Collect more data with diverse view information for each single 3D object.
- Use the physical real 3D objects to train the network such as chair, table or cars.
- Handle more complicated detection scenario which contains multiple 3D objects, also including some occlusion cases.