Frustum VoxNet for 3D object detection from RGB-D or Depth images

Xiaoke Shen
The Graduate Center, CUNY
New York City, USA
xshen@gradcenter.cuny.edu

Ioannis Stamos
Hunter College & The Graduate Center, CUNY
New York City, USA
istamos@hunter.cuny.edu

Abstract

Recently, there have been a plethora of classification and detection systems from RGB as well as 3D images. In this work, we describe a new 3D object detection system from an RGB-D or depth-only point cloud. Our system first detects objects in 2D (either RGB, or pseudo-RGB constructed from depth). The next step is to detect 3D objects within the 3D frustums these 2D detections define. This is achieved by voxelizing parts of the frustums (since frustums can be really large), instead of using the whole frustums as done in earlier work. The main novelty of our system has to do with determining which parts (3D proposals) of the frustums to voxelize, thus allowing us to provide high resolution representations around the objects of interest. It also allows our system to have reduced memory requirements. These 3D proposals are fed to an efficient ResNet-based 3D Fully Convolutional Network (FCN). Our 3D detection system is fast, and can be integrated into a robotics platform. With respect to systems that do not perform voxelization (such as PointNet), our methods can operate without the requirement of subsampling of the datasets. We have also introduced a pipelining approach that further improves the efficiency of our system. Results on SUN RGB-D dataset show that our system, which is based on a small network, can process 20 frames per second with comparable detection results to the state-of-the-art [16], achieving a 2× speedup.

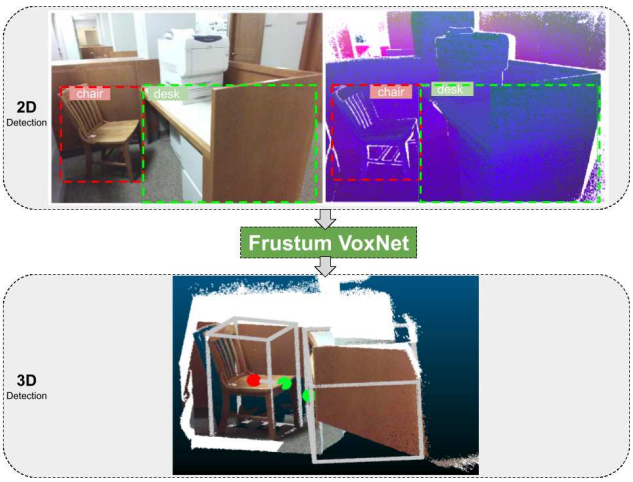

Figure 1. Overview of the whole system. Upper left: RGB image and detected 2D bounding boxes. Upper right: DHS (**D**epth **H**eight and **S**igned angle) image, and detected 2D bounding boxes. A DHS image is a pseudo-RGB image generated by a depth image (see text). Bottom: The final 3D detected objects from the associated 3D range image. The 3D detection not only provides an amodal bounding box but also an orientation. The red point is the center of the bounding box and the green one is the front center. The detected 2D bounding boxes from either and RGB or DHS image, generate 3D frustums (which are prisms having as apex the sensor location and extend through the 2D bounding boxes to the 3D space). They are then fed to our Frustum VoxNet network, which produces the 3D detections.

1. Introduction

Classification and object detection are significant problems in the fields of computer vision and robotics. 2D object detection systems from RGB images have been significantly improved in recent years due to the emergence of deep neural networks and large labeled image datasets. For applications related to robotics though, such as autonomous navigation, grasping, etc., a 2D object detection system is not adequate. Thus 3D object detection systems have been developed, with input coming from RGB-D or depth-only sensors. In this paper we describe a new 3D object detection system that incorporates mature 2D object detection methods as a first step. The 2D detector can run on an input RGB image, or pseuso-RGB image generated from a 3D point cloud. That 2D detection generates a 3D frustum (defined by the sensor and the 2D detected bounding box) where a search for a 3D object is performed. Our main contribution is the 3D object detection within such as frustum. Our method involves 3D voxelization, not of the whole frustum, but of a learned part of it. That allows for a higher resolution voxelization, lower memory requirements, and a more efficient detection.

978-1-7281-6553-0/20/$31.00 ©2020 IEEE

Figure 1 illustrates the overview of our system. In the upper left we see a 2D RGB image, along with the 2D detected bounded boxes (a chair and a desk). On the upper right we see a 2D pseudo-RGB image that was generated from the associated 3D range image (see [28]), along with similarly detected 2D bounded boxes. We call this pseudo-RGB image a **DHS** image, where D stands for **D**epth, H for **H**eight, and S for **S**igned angle. The depth is a normalized distance of the associated 3D point, height is a normalized height of the 3D point, and the signed angle is a normalized approximation of the normal at the 3D point (see [28]). We can apply traditional 2D detectors on this pseudo-RGB image, making our method applicable even when no RGB information is available. 3D frustums are then extracted from these 2D detections. A 3D frustum is a prism having as apex the sensor location and extending through the 2D bounding boxes into the 3D space. Learned parts of the 3D frustum are being voxelized. These voxelizations are fed to Frustum VoxNet, which is a 3D Fully Convolutional Neural Network (FCN).

The key contributions of our work can be summarized as follows:

- We demonstrate the power of using a 3D FCN approach based on volumetric data to achieve accurate 3D detection results efficiently.
- We provide a novel method for learning the parts of 3D space to voxelize. This allow us to provide high resolution representations around the objects of interest. It also allows our system to have reduced memory requirements and leads to its efficiency.
- Compared to systems that do not perform voxelization (such as [17, 16]), our methods can operate without the requirement of subsampling the datasets. Also, our approach is more efficient and can be used in robotics applications.
- Compared to systems that do voxelize (such as [29]), our system does not voxelize the whole space, and thus allows a higher-resolution object representation.
- We compare the 3D detection performance of using different input channels (RGB or DHS).
- We provide a more efficient variation of our method that involves pipelining, geared to robotics applications.
- The parameters of our network are much smaller than leading methods. That results in faster inference time.

We start by reviewing related work and then proceed with the description of our 3D detection system along with our experimental results. Since our final goal is indoor robotic navigation, our current system has been evaluated based on an indoor SUN-RGBD dataset[23].

2. Related Work

2D methods RGB-based approaches can be summarized as two-stage frameworks (proposal and detection stages) and one-stage frameworks (proposal and detection in parallel). Generally speaking, two-stage methods such as R-CNN [4], Fast RCNN [3], Faster RCNN [20], FPN [11] and mask R-CNN [6] can achieve a better detection performance while one-stage systems such as YOLO[18], YOLO9000[19] and RetinaNet [12] are faster at the cost of reduced accuracy. For deep learning based systems, as the size of network is increased, larger datasets are required. Labeled datasets such as PASCAL VOC dataset [2] and COCO (Common Objects in Context) [13] have played important roles in the continuous improvement of 2D detection systems.

3D methods Compared with detection based on 2D images, the detection based on 3D data is more challenging due to several reasons [22]: 1) Data representation itself is more complicated. 3D images can be represented by point clouds, meshes, or volumes, while 2D images have pixel grid representations. 2) Due to the extra dimension, there are increased computation and memory resource requirements. 3) 3D data is generally sparser and of lower resolution compared with the dense 2D images, making 3D objects more difficult to identify. Finally, 4) large sized labeled datasets, which are extremely important for supervised based algorithms, are still inferior compared with well-built 2D datasets. Below we summarize the basic approaches.

Project 3D data to 2D and then employ 2D methods There are different ways to project 3D data to 2D features. HHA was proposed in [5] where the depth image is encoded with three channels: Horizontal disparity, Height above ground, and the Angle of each pixels local surface normal with gravity direction. The signed angle feature described in[26] measures the elevation of the vector formed by two consecutive points and indicates the convexity or concavity of three consecutive points. Input features converted from depth images of normalized depth(D), normalized relative height(H), angle with up-axis(A), signed angle(S), and missing mask(M) were used in [28]. We are using DHS in this work to project 3D depth image to 2D since as shown in [28] adding more channels did not affect classification accuracy significantly. Keeping the number of total channels to three, allow us to use networks with pre-trained weights for starting our training.

2D-Driven 3D Object Detection from RGB-D Data Our proposed framework is mainly inspired by 2D-driven 3D object detection approaches as in [10, 16]. First a 2D detector is used to generate 2D detections. The differences

of our work with [10] are: 1) the 2D detector in [10] is only based on RGB images and our proposed system explores both RGB-D and Depth only data. 2) 3D detection in [10] uses a MLP regressor to regress the object boundaries based on histograms of points along x, y, and z directions. Converting raw point clouds to histograms results in a loss of information. The main differences of our system to Frustum PointNets [16] are the following: 1) in the 2D detection part, Frustum PointNets is based on RGB inputs, while our system can support both RGB-D and depth-only sensing. 2) in the 3D detection part, our system is using voxelized data, while Frustum PointNets is consuming raw point clouds via PointNet [17]. PointNet uses a fully connected neural network and max pooling, so it cannot support convolution/deconvolution operations well. We believe 3D convolution/deconvolution can play important roles in both 3D semantic segmentation and object detection. 3) PointNet's computation complexity is increased if more points are available as the framework's input is $N \times K$ where N is the number of points and K is the number of channels. 4) Random sampling is required in PointNet, but is not needed in our voxelization approach.

A recent method [15] that is based on PointNet and Hough Voting, achieves improved detection results without the use of RGB images. Our method is still more efficient in inference time, and thus more appropriate for robotics application. Also, our approach does not need to subsample the 3D point cloud as required by [15].

3D CNNs VoxelNet [29] uses 3D LiDAR data to detect 3D objects based on the KITTI outdoor dataset, and utilizes bird's eye view (BEV) features (such as MV3D [1] and AVOD [9])). The use of BEV is not helpful in indoor applications. Also, the use of the whole range image for voxelization lowers the resolution (and therefore the scale) of the objects of interest. Early influential 3D detection systems used two-stage approaches. The first stage generates proposals, while the second stage performs 3D detection. DeepSliding Shape[24] detects 3D objects based on the SUNRGB-D dataset and it uses directional Truncated Signed Distance Function (TSDF) to encode 3D shapes. The 3D space is divided into 3D voxels and the value in each voxel is defined to be the shortest distance between the voxel center and the surface from the input depth map. A fully convolutional 3D network extracts 3D proposals at two scales corresponding to large size objects and small size objects. For the final 3D detection, this method fuses the 3D voxel data and RGB image data by using 3D and 2D CNNs. Our approach, one the other hand, first focuses on the frustum to voxelize, and then selects the part to be voxelized based on training. That allows us to achieve higher resolution around the objects of interest.

We refer readers to [22] for latest, comprehensive comparisons of different 3D detection systems.

3. Dataset

We are focusing on the indoor SUN RGBD dataset[23]. SUN RGBD dataset splits the data into a training set which contains 5285 images and a testing set which contains 5050 images. For the training set, it further splits into a training only, which contains 2666 images and a validation set, which contains 2619 images. Similar to [24, 10], we are training our model based on the training only set and evaluate our system based on the validation set. We call the only training dataset as train2666 in the future description.

4. Frustum VoxNet System Overview

First, 2D detections on RGB or DHS image generate 2D bounding boxes of objects. The 2D detections generate 3D frustums (defined by the sensor and the 2D detected bounding box) where a search for a 3D object is performed. For each such frustum we know the class of the object to be detected by the 2D detection. Our system accurately localizes the amodal 3D bounding box and the orientation of the detected 3D object. To achieve this, we perform 3D voxelization, not of the whole frustum, but a learned part of it. That allows for a higher resolution voxelization, lower memory requirements, and a more efficient detection. We explain first how we decide which part of the frustum to use.

4.1. Frustum Voxelization

Given a 3D frustum (defined as a 3D prism from the sensor and the 2D detected bounding box into the 3D space), our goal is to voxelize only a part of it. We define that part as axis-aligned 3D bounding boxes enclosed in the frustum. We call that bounding box a **3D Cropped Box (3DCB** for short). Given a specific object class (for instance a table), an ideal 3DCB will be big enough to contain all the 3D points belonging to the object, but also small enough to achieve high resolution voxelization. In order to quantify the ability of a given 3DCB to tightly contain a given 3D object, we define the metric *3D Intersection over Itself (IoI)*. Suppose the object of interest lies in a bounding box 3DBBOX. Then the IoI of the 3DBBOX wrt to a given 3DCB is defined as the volume of intersection of the 3D bounding box with the 3DCB over the volume of the 3D bounding box itself. Therefore an IoI of 1.0 means that the 3DCB is perfectly enclosing the object in 3DBBOX, while as this number tends to 0.0 more and more empty space is included in the 3DCB.

The formula for 3D IoI is:

$$IoI^{3D} = \frac{volume^{3DBBOX} \cap volume^{3DCB}}{volume^{3DBBOX}}$$

From the definition, it is trivial to show that:

$$IoI^{3D} = IoI^{XY} * IoI^{Z}$$

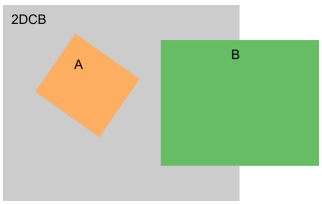

Figure 2. An example of 2DCB with two objects box A and box B. All these boxes are square. A has length 1, B has length 2 and 2DCB has length 3. Half of B is overlapped with 2DCB

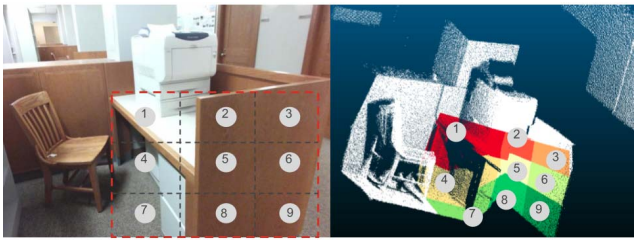

Figure 3. An example of equally subdividing a whole frustum into 3×3 subfrustums (best viewed in color). In this example, the object is a desk. The upper one shows the 2D bounding box of desk is equally divided into 9 small boxes. From each small box, a subfrustum is generated as shown in the bottom image.

where IoI^{XY} is the IoI in the XY plane and IoI^{Z} is the IoI along the Z axis.

$$IoI^{XY} = \frac{area^{3DBBOX^{XY}} \cap area^{3DCB^{XY}}}{area^{3DBBOX^{XY}}}$$

$$IoI^{Z} = \frac{length^{3DBBOX^{Z}} \cap length^{3DCB^{Z}}}{length^{3DBBOX^{Z}}}$$

$3DBBOX^{XY}$ and $3DCB^{XY}$ are 2D projections of 3D bounding box and 3DCB onto the XY plane. $3DBBOX^{Z}$ and $3DCB^{Z}$ are 1D projections of 3D bounding box and 3DCB onto the Z axis.

We use this metric to choose the optimal 3DCB size. A 2D example in Figure 2 is used to show the difference between IoI and IoU (Intersection over Union). From this example, box A is totally contained in 2DCB(XY plane projection of a 3DCB) while only half of box B is covered by 2DCB. If we use 2D IoU, we will get 0.11 for box A with 2DCB and 0.18 for box B with 2DCB.

Generating 3DCBs using an IoI metric During training, given a ground truth 2D bounding box of an object of a given class (for example table) and given the ground truth 3D bounding box of the same object, we would like to calculate the optimal 3DCB box. The 3DCB is represented by its center, and width, depth, and height. We are adding the constraint that width and depth are the same. This makes sure that the object can freely rotate within the 3DCB along the vertical axis. We proceed by equally dividing the 2D bounding box along the **R**ow and **C**olumn into $FR \times FC$ 2D boxes. Then we have $FR \times FC$ sub**F**frustums. We will generate $FR \times FC$ candidate centers of 3DCBs in that case. The center of each 3DCB is the centroid of the respective frustum. One example of 3×3 subfrustums of a desk is shown in Figure 3. If we set $FR = FC = 1$, then there is only one 3D frustum to consider (and therefore one 3DCB center). Our goal is to calculate the optimal sizes of respective 3DCBs for each object category.

A ground truth 3D bounding box will be recalled (i.e. enclosed into the 3DCB) if the *3D IoI* of this box is greater than a threshold. Formally, we define this recall as $recall^{volume}$:

$$recall^{volume} = \frac{|3DCB^{positive}|}{|3DCB|}$$

where $|3DCB^{positive}|$ is the cardinality of positive 3DCBs and $|3DCB|$ is the cardinality of all 3DCBs. A 3DCB is positive when $IoI^{3D} = IoI^{XY} * IoI^{Z} \geq threshold$. To make the parameter setting simple, we are exploring the recall of XY plane and Z axis separately. Similar to $recall^{volume}$, $recall^{XY}$ and $recall^{Z}$ are defined as: $recall^{XY} = \frac{|3DCB_{XY}^{positive}|}{|3DCB|}$, $recall^{Z} = \frac{|3DCB_{Z}^{positive}|}{|3DCB|}$, where $|3DCB_{XY}^{positive}|$ is the cardinality of positive 3DCBs in XY plane, $|3DCB_{Z}^{positive}|$ is the cardinality of positive 3DCBs in Z axis and $|3DCB|$ is the cardinality of all 3DCBs. A 3DCB is positive in XY plane when $IoI^{XY} \geq threshold_{XY}$ and a 3DCB is positive in Z axis when $IoI^{Z} \geq threshold_{Z}$.
Although, we can NOT naively have $recall^{volume} = recall^{XY} * recall^{Z}$, we have a nice inequality to guarantee a lower bound of $recall^{volume}$:

$$recall^{volume} \geq \max(0, \quad recall^{XY} + recall^{Z} - 1) \quad (1)$$

The proof of this inequality is given in the appendix.

Both of $threshold_{XY}$ and $threshold_{Z}$ are set as 0.90. We are generating both the average center and median center from subfrustums and pick up the best one from these $FR \times FC$ candidates to calculate the recall. The average recall based on different setups of width/depth and height are shown in Figure 4. From the results, we can observe: 1) the performance of the average center based 3DCB is better especially when 1×1 subfrustums are used compared with the median center. The reason for this might be the range of indoor depth sensor is limited and outliers will not have too much influence to the results. 2) The 3DCB generated from 1×1 is better than 3×3 and 5×5 ones. Based on these observations, we are choosing both 1×1 and 3×3 during training to generate more samples and make the training robust to the inaccurate bounding box predictions. During inference, 1×1 subfrustum

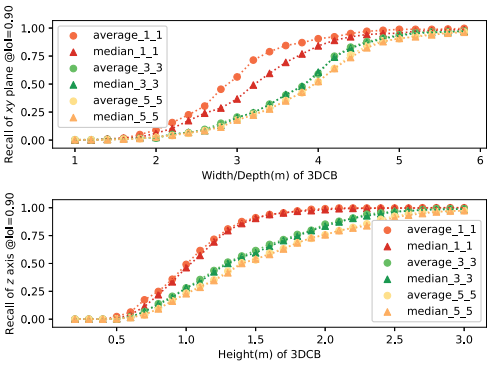

Figure 4. IoI^{XY} and IoI^{Z} with the widths/depths and heights. 3DCB are generated from average/median center based on $FR \times FC$ subfrustums with different widths/depths and heights. In this plot, *average/median_m_n* corresponds to recall based on average/median center in $m \times n$ subfrustums.

based 3DCB is used to speed up and get better performance.

Double Frustum Method To increase the accuracy of the center calculations, we developed a double Frustum framework. We use a smaller 2D bounding box to generate a smaller frustum for the calculation of the 3DCB center. The estimated center should now be more accurate since it will concentrate on the central part of the object and thus will avoid the use of other background objects. A 3DCB is then selected from a larger frustum in order to contain background context points and possible false negative points. The larger frustum is generated from a larger 2D bounding box. During training, we generate large frustum by randomly increasing the 2D bounding box width and height by 0% to 15% independently. For the small frustum, we randomly decrease the 2D bounding box width and height by 0% to 10% independently. During inference, the large frustum is generated by increasing the 2D bounding box width and height by 5%. Original 2D detection bounding boxes are used to calculate the 3DCB center.

Multiple Scale Networks In [24], two scales network were used for different categories concerning the 3D physical size. We are using 4 scales networks to voxelize the 3D objects corresponding to the average physical size of average height, maximum of average width and depth. The mapping of 3D object categories to different scales is shown in Table 1. We are calculating the $recall_{XY}$ and $recall_{Z}$ for different objects with the different setups for width/depth and heights. The curves of $recall_{XY}$ with width/depth and $recall_{Z}$ with height are plotted for four classes based on 3×3 subfrustums (sofa is from large short scale, chair is from medium short scale toilet is from small short scale and bookshelf is from median tall scale) are shown in Figure 5. From these curves, we can find out that

	Short ($h \leq 0.55$)	Tall ($h > 0.55$)
Small ($\max(w,h) \leq 0.3$)	toilet	N/A
Medium ($0.3 < \max(w,h) \leq 0.55$)	chair, nightstand, sofa chair, garbage bin,bathtub	bookshef
Large ($\max(w,h) > 0.55$)	table, desk, sofa, bed, dresser	N/A

Table 1: Objects are classified into 4 categories based on there average physical size. Voxelization is processed based on each category.

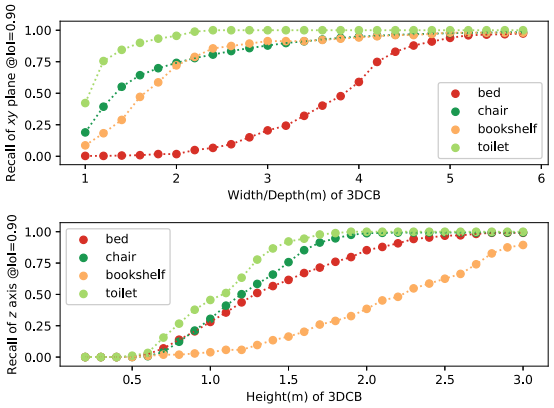

Figure 5. XY plane recall and Z axis recall for bed, chair bookshelf and toilet with the widths/depths and heights based on train2666 dataset.

medium tall scale category needs greater height and both the large short and medium short categories need more width/depth. We are selecting the minimum width/depth and height which can guarantee all objects within that scale network can meet the requirements of $recall^{XY} \geq 0.90$ and $recall^{Z} \geq 0.95$. This is based on 3×3 subfrustums. From the equation 1, we can have the lower bound of the $recall_{volume}$ of 0.85. Although 0.85 is not high enough, when based on 1×1 subfrustums, the lower bound of the $recall_{volume}$ can achieve 0.94 as $recall^{XY} \geq 0.95$ and $recall^{Z} \geq 0.99$ for 3DCBs generating from 1×1. Since we are using both 3DCBs from 1×1 and 3×3 subfrustums, the recall is good enough to support the training.

The physical sizes(width/depth/height) of 4 scale networks are shown in Table 2 based on the principles described above. 3DCB are further voxalized(counting the number of cloud points within each voxel) into a 3D tensor with the shape of $W \times D \times H$. The $W \times D \times H$ for each scale network are selected to make it having a better resolution as compared with [24]. The comparison of physical size, resolution, tensor shape of the RPN and detection networks of [24] and ours are also shown in Table 2.

Method	Network	3DCB physical size (m)	3DCB Shape	Resolution (cm)
DSS [24]	RPN	$2.5 \times 2.5 \times 2.5$	$208 \times 208 \times 100$	$5.2 \times 6.0 \times 2.5$
	Detection (bed)	$6.7 \times 6.7 \times 3.2$	$30 \times 30 \times 30$	$2.0 \times 2.0 \times 0.95$
	Detection (trash can)	$1.0 \times 1.0 \times 1.2$	$30 \times 30 \times 30$	$0.3 \times 0.3 \times 0.5$
Ours	small short	$1.6 \times 1.6 \times 1.5$	$198 \times 198 \times 102$	$0.8 \times 0.8 \times 1.5$
	medium short	$3.2 \times 3.2 \times 1.7$	$198 \times 198 \times 102$	$1.6 \times 1.6 \times 1.7$
	large short	$4.8 \times 4.8 \times 2.2$	$198 \times 198 \times 102$	$2.4 \times 2.4 \times 2.2$
	medium tall	$2.8 \times 2.8 \times 3.0$	$134 \times 134 \times 134$	$2.1 \times 2.1 \times 2.2$

Table 2: Resolution and shape comparison between DeepSliding Shape[24] and ours. Anchors of the bed and trash can from[24] are used as examples of proposal's physical size to make the comparison with ours.

4.2. 3D Object Detection

3D Bounding Box Encoding Similar to [24], we are using the orientation, center, width, depth and height to encode the 3D bounding box.

Network architecture We are using 3D FCN networks to build the 3D detection network by adapting the network structure of ResNet[7] and Fully Convolutional Network(FCN)[14]. We propose a fast 6 layer fully convolutional 3D CNN model as shown in Figure 6.

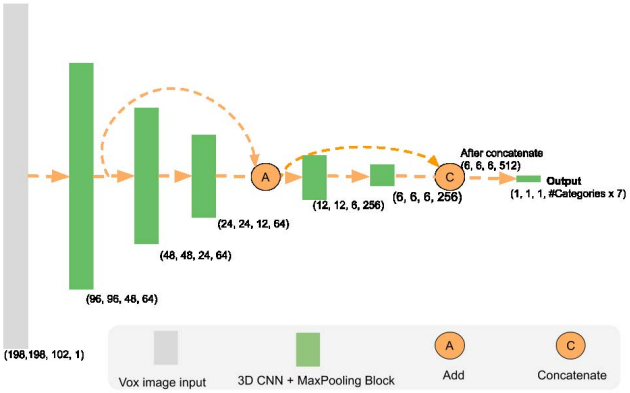

Figure 6. ResnetFCN6 architecture (used for large short scale). Every 3D CNN layer will be followed by a dropout layer. The tensor shape shown here is the output shape of each block. It provides the (width, depth, height, channel) information of the network. The rest three scale networks have the same structure with different input size as shown in Table 2. The architecture of ResnetFCN35 will be provided in the Supplementary Material.

Inputs of our networks are voxelized images. Our network will have $C * 7$ outputs, where C is the number of classes within the corresponding scale network, and 7 is the orientation, center xyz and size(width/depth/height) predictions. The 2D prediction info is implicitly encoded in the system since the prediction is based on each category.

Loss Function We are generating loss function for detection by adjusting the loss function from YOLO9000[19]. Similar to [19], we use simple $L2$ distance instead of KullbackLeibler divergence to evaluate the difference of predited category probability distributions and the ground truth distributions. For the regression part, for centers, we normalize the x, y, z values to 0 and 1 and then use a sigmoid function to make the prediction. For width(w), depth(d) and height(h), we use anchor to support the prediction. For each category, we set the anchor as the average value of the train2666 samples for objects within this category. The ratio of the bounding box to the related anchors are used to drive the network to make the correct prediction. The formal definition of the loss is given in the formulas below.

$$L^{3D}_{detection} = \lambda_1 L_{orientation} + \lambda_2 L_{xyz} + \lambda_3 L_{wdh}$$

Where $L_{xyz} = L_x + L_y + L_z$, $L_{wdh} = L_w + L_d + L_h$, $L_x = (x - x^\star)^2$, $L_y = (y - y^\star)^2$, $L_z = (z - z^\star)^2$, $L_w = (\log \frac{w}{a_w} - \log \frac{w^\star}{a_w})^2$, $L_d = (\log \frac{d}{a_d} - \log \frac{d^\star}{a_d})^2$, $L_h = (\log \frac{h}{a_h} - \log \frac{h^\star}{a_h})^2$. a_w, a_d, a_h are width/depth/height of anchors. $\lambda_1, \lambda_2, \lambda_3$ are used to balance losses.

5. Training Process

For the 2D detection, we are using ResNet[7] 101 layer as the backbone and using the feature pyramid layers proposed by[11] which is based on Faster RCNN[20] approach. The loss is the same as[11]. For the 2D detection, the network is pretrained on COCO dataset. Then it is retrained on SUN-RGBD dataset based on RGB or DHS images. Although, the DHS images are different to the RGB images, we find the pretrained weights can still speed up the whole training process and improve the detection results. Data is augmented by adding gaussian blur, random cropping and image translating up to 10% of the original images.

For the 3D detection, we use the stochastic gradient descent(SGD) with learning rate of 0.01 and a scheduled decay of 0.00001. For regulation we use batch normalization[8]. The cloud points are randomly rotated around z-axis and jittered during the voxelization process before feeding them to the network.

6. Efficiency boost by Pipelining

Pipelining instructions is a technology used in central processing units to speed up the computing. An instruction pipeline reads an instruction from the memory while previous instructions are being executed in other steps of the pipeline. Thus multiple instructions can be executed simultaneously. Pipelining can be perfectly used in our system as we have two stages, one is 2D detection and one is

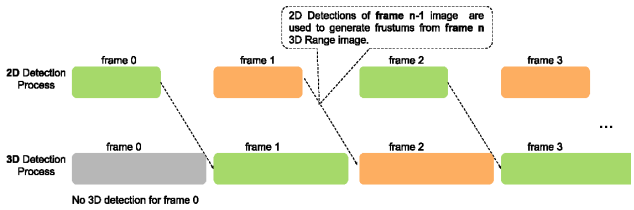

Figure 7. Illustration of using pipelining to speedup the whole detection framework.

3D detection. In the 3D detection, instead of using the 2D detection of frame n, we can use the 2D detection results of frame n-1 and generate frustums based on that. By using pipelining, our system can be sped up from $t_{2D} + t_{3D}$ to $\max(t_{2D}, t_{3D})$, where t_{2D} and t_{3D} are the 2D and 3D detection time, respectively. The disadvantage of using pipelining is frustums generated from the previous 2D image maybe not accurate under the fast movement of the sensor of an object of interest. However, our system will not suffer significantly as our results show, due to robustness on frustum location. We use multiple candidates with different centers during training to make it robust. Meanwhile, the double frustum method used in our system makes our 3D detections robust to slightly moved 2D detections. The illustration of the pipelining method is shown in figure 7. By using pipelining, our system can be sped up to 48 ms (this is about $2.5\times$ speedup to the state-of-the-art [16]) when use YOLO v3 and ResNetFCN6. It can achieve 21 frames per second which can well support real time 3D object detection.

7. Experiments Results

7.1. Effects of Batch Normalization [8], Group Normalization [27] and Dropout[25]

Dropout is a powerful tool to prevent neural networks from overfitting. Batch Normalization(BN) [8] is another method we can use to speed up the training and prevent overfitting. However, BN performs better when the batch size is large enough. Since Frustum VoxNet is using 3D CNNs, large batch sizes are not well supported when single GPU is used. Some new technologies are introduced to address the small batch size problem such as Group Normalization(GN) [27]. We explore the performance of different combinations of these methods by evaluating the performance of center and orientation predictions. Results are shown in Figure 8. We do not use BN as our batch size is small and the using of BN will lead to inconsistencies between training and inference. Although when using the GN, there are no inconsistencies between training and inference, the performance of center prediction is worse compared with not using any normalization. Therefore, our final model does not use any normalization. However, dropout is used in our final model as the performance of center prediction is improved.

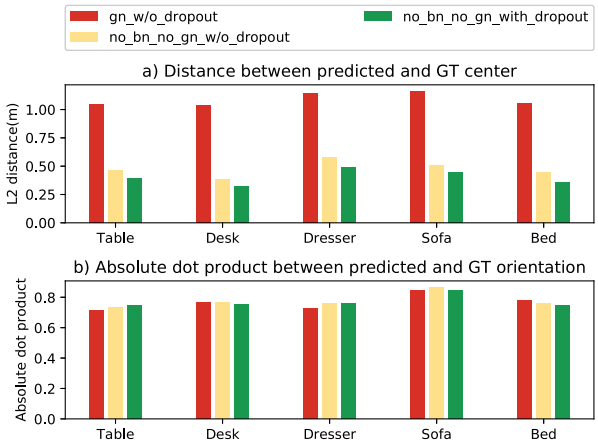

Figure 8. Performance comparison of different combinations on using BN, GN and dropout. "gn_w/o_dropout" means using GN without dropout. "no_bn_no_gn_w/o_dropout" means using none. "no_bn_no_gn_with_dropout" means not using BN/GN, however, the Dropout is used.

7.2. Evaluation of the whole system

First we evaluate the 2D detector in Table 3. The evaluation is based on the standard mAP metric with IoU threshold of 0.5. Comparing our RGB-based and depth-based (DHS image) 2D detection, we see that in most cases RGB performs better, but the depth-based 2D detector is competitive. For few classes such as bathtubs, DHS results are slightly better. The reason might be that some classes such as bathtubs have special geometric shapes and they are easier to be detected by depth sensors. Comparing with state-of-the-art methods, our 2D detector performs better in some categories, and we are also introducing new categories. We are on par with most other categories, except for bathtub, desk, and bookshelf.

Full 3D detection results are shown in Table 4. We provide various variations in our system. First two variations include RGB 2D detector, and the last two include depth only (DHS) 2D detector. In all cases, we use a FPN for the 2D detector. For the 3D detection we have experimented with ResNetFCN6 and ResNetFCN35. As in the 2D case, our 3D detector is on par in most categories with the state-of-the-art, and we have also incorporated more classes. Looking at the computational performance of the 3D detector only, we see that our implementation using ResNetFCN6 provides significant improvements on inference time. Since the architecture is modular (i.e. we can swap out our 2D detector with one from the reported as state-of-the-art),

	bed	toilet	night stand	bathtub	chair	dresser	sofa	table	desk	bookshelf	sofa chair	kitchen counter	kitchen cabinet	garbage bin	microwave	sink
RGB-D RCNN[5](RGB-D)	76.0	69.8	37.1	49.6	41.2	31.3	42.2	43.0	16.6	34.9	N/A	N/A	N/A	46.8	N/A	41.9
2D-driven[10](RGB)	74.5	86.2	49.5	45.5	53.0	29.4	49.0	42.3	22.3	45.7	N/A	N/A	N/A	N/A	N/A	N/A
Frustum PointNets[16](RGB)	56.7	43.5	37.2	81.3	64.1	33.3	57.4	49.9	77.8	67.2	N/A	N/A	N/A	N/A	N/A	N/A
OURS(RGB)	**81.0**	**89.5**	35.1	50.0	52.4	21.9	53.1	37.7	18.3	40.4	**47.8**	**22.0**	**29.8**	**52.8**	**39.7**	31.0
OURS(D)	**78.7**	77.6	34.2	51.9	51.8	16.5	48.5	34.9	14.2	19.2	48.7	19.1	18.5	30.3	22.2	30.1

Table 3: **2D detection results based on SUN-RGBD validation set**. Evaluation metric is average precision with 2D IoU threshold of 0.5.

	bed	toilet	night stand	bathtub	chair	dresser	sofa	table	desk	bookshelf	sofa chair	garbage bin	frustum proposal runtime	3D detection runtime	Total runtime
DSS[24](RGB-D)	78.8	78.9	15.4	44.2	61.2	6.4	53.5	50.3	20.5	11.9	N/A	N/A	N/A	N/A	19.55s
COG[21](RGB-D)	63.7	70.1	27.4	58.3	62.2	15.5	51.0	51.3	45.2	31.8	N/A	N/A	N/A	N/A	10-30min
2D-driven[10](RGB-D)	64.5	80.4	41.9	43.5	48.3	15.5	50.4	37.0	27.9	31.4	N/A	N/A	N/A	N/A	4.15s
Frustum PointNets[16](RGB-D)	81.1	90.0	58.1	43.3	64.2	32.0	61.1	51.1	24.7	33.3	N/A	N/A	**60ms**	60ms	**0.12s**
OURS RGB-D (FPN+3D ResNetFCN6)	78.5	84.5	34.5	42.4	47.2	18.2	40.3	30.4	12.4	18.0	47.1	47.6	110ms	**48ms**	0.16s
OURS RGB-D (FPN+3D ResNetFCN35)	79.5	84.6	36.2	44.6	49.1	19.6	40.8	27.5	12.5	19.1	47.9	**48.2**	110ms	128ms	0.24s
OURS Depth only (FPN+3D ResNetFCN6)	77.1	76.1	32.4	42.0	45.9	14.1	35.8	25.3	11.7	16.8	48.5	35.0	110ms	**48ms**	0.16s
OURS Depth only (FPN+3D ResNetFCN35)	77.4	76.8	33.1	43.7	45.8	15.2	37.3	25.5	11.8	17.4	**48.8**	35.4	110ms	148ms	0.24s

Table 4: **3D detection results on SUN-RGBD validation set.** Evaluation metric is average precision with IoU threshold of 0.25 as proposed by[23]. Both COG[21] and 2D-driven[10] are using room layout context to boost performance while ours, DSS[24] and Frustum PointNets[16] are not. Frustum PointNets[16] is using the 3D segmentation information to train the network to boost the 3D detection, while our system and DSS[24] are not.

we see that our approach can lead to significant efficiency improvements, without a significant drop in detection accuracy. That will lead to a system geared to real-time robotics applications.

We have also evaluated the efficiency and accuracy of our system when a very fast 2D detector (Yolo v3) is being used. Table 5 shows the decrease in detection accuracy as expected. Finally Table 6 provides a detailed analysis of multiple network combinations in terms of efficiency, along with the number of parameters to tune. As mentioned before we can achieve faster inference times in 3D detection, and can thus lead to a faster system overall if we swap our 2D detector with the ones reported as state-of-the-art. Using Yolo and pipelining approach, we can provide a significant boost in total efficiency, with accuracy loss though.

	2D network	3D network	bed	toilet	chair	sofa	table
2D Detection	FPN		81.0	89.5	52.4	53.1	37.7
	YOLO v3		71.8	73.7	38.5	51.4	22.1
3D Detection	FPN	3D ResNetFCN6	78.5	84.5	47.2	40.3	30.4
	YOLO v3	3D ResNetFCN6	66.9	69.8	30.1	37.9	18.8

Table 5: 2D/3D detection results based on YOLO v3 V.S. FPN. 2D detection is based on RGB images. 3D detection is based on RGB-D images.

8. Conclusion and Future Work

We presented a 2D-based 3D detection system by using 2D/3D CNNs. Our method can operate in both Depth only and RGB-D sensor modalities. We provide comparable results to state-of-the-art, but with significantly more efficient 3D detection. This is due to the use of networks with fewer number of parameters than competing methods. It is also due to our ability to voxelize only parts of the 3D frustums. This leads to decreased memory requirements and improved resolution around the objects of interest. In future work we will be integrating segmentation that we believe will further boost the detection accuracy of our system.

Methods	# parameters		Runtime (ms)		
	Frustum proposal	3D detection	Frustum proposal	3D detection	Total
Frustum PointNets (FPN+Pointnet v1)	**28M**	19M	60	60	120
Frustum PointNets (FPN+Pointnet v2)	**28M**	22M	60	107	167
Ours w/o Pipeline (FPN+3D ResNetFCN6)	42M	**2.5M**	110	**48**	158
Ours w/o Pipeline (FPN+3D ResNetFCN35)	42M	23.5M	110	149	259
Ours w/o Pipeline (YOLO v3+3D ResNetFCN6)	N/A	**2.5M**	**29**	**48**	**77**
Ours with Pipeline (YOLO v3+3D ResNetFCN6)	N/A	**2.5M**	**29**	**48**	**48**

Table 6: Number of parameters and inference time comparison between Frustum Pointnet and our system. For YOLO v3, input resolution is 416 by 416 and the model FLOPS is 65.86 Bn.

9. Acknowledgement

This work was partially supported by NSF Award CNS-1625843 and Google Faculty Research Award 2017 (special thanks to Aleksey Golovinskiy, Tilman Reinhardt and Steve Hsu for attending to all of our needs). We acknowledge the support of NVIDIA with the donation of the Titan-X GPU used for this work. We thank Jaspal Singh for data preparation and earlier discussion. We also would like to thank Allan Zelener, James Kluz, Jaime Canizales and Bradley Custer for helpful comments and advice.

References

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.

[2] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[3] R. B. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1440–1448. IEEE Computer Society, 2015.

[4] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[5] S. Gupta, R. B. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. *CoRR*, abs/1407.5736, 2014.

[6] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[9] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. *ArXiv e-prints*, Dec. 2017.

[10] J. Lahoud and B. Ghanem. 2d-driven 3d object detection in rgb-d images. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[11] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[12] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[13] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[14] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[15] C. R. Qi, O. Litany, K. He, and L. J. Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[16] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017.

[17] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[18] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[19] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[20] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 91–99, 2015.

[21] Z. Ren and E. B. Sudderth. Three-dimensional object detection and layout prediction using clouds of oriented gradients. pages 1525–1533, 06 2016.

[22] X. Shen. A survey of Object Classification and Detection based on 2D/3D data. *arXiv e-prints*, page arXiv:1905.12683, May 2019.

[23] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[24] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in RGB-D images. *CoRR*, abs/1511.02300, 2015.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[26] I. Stamos, O. Hadjiliadis, H. Zhang, and T. Flynn. Online algorithms for classification of urban objects in 3d point clouds. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, pages 332–339, Oct 2012.

[27] Y. Wu and K. He. Group normalization. *CoRR*, abs/1803.08494, 2018.

[28] A. Zelener and I. Stamos. Cnn-based object segmentation in urban lidar with missing points. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 417–425, Oct 2016.

[29] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, June 2018.