3D Object Detection with Sparse Sampling Neural Networks

by

Ryan Goy

A thesis submitted in partial satisfaction of the requirements for the degree of **Master of Science, Plan II** in Electrical Engineering and Computer Science in the Graduate Division of the University of California, Berkeley

Committee in charge:

Professor Avideh Zakhor, Research Advisor

Date

Professor John Chuang, Second Reader

Date

Fall 2018

Abstract. The advent of inexpensive 3D sensors has resulted in an abundance of 3D point-clouds and datasets. For instance, RGB-D sensors such as Kinect can result in 3D point clouds by projecting 2D pixels into 3D world coordinate using depth and pose information. Recent advancements in deep learning techniques appear to result in promising solutions to 2D and 3D recognition problems including 3D object detection. Unlike 3D classification, 3D object detection has received less attention in the research community. In this thesis, we propose a novel approach to 3D object detection, the Sparse Sampling Neural Network (SSNN), which takes large, unordered point clouds as input. We overcome the challenges of processing three dimensional data by convolving a collection of "probes" across a point cloud input which then feeds into a 3D convolutional neural network. This approach allows us to efficiently and accurately infer bounding boxes and their associated classes without discritizing the volumetric space into voxels. We demonstrate that our network performs well on indoor scenes, achieving mean Average Precision (mAP) of 54.48% on the Matterport3D dataset, 62.93% on the Stanford Large-Scale 3D Indoor Spaces Dataset, and 48.4% on the SUN RGB-D dataset.

1 Introduction

Processing 3D data poses additional challenges as compared to 2D. In 2D, pixels are commonly organized by coordinate in regular 2D arrays so convolutions can easily take advantage of locality [1]. Due to the lack of structure in 3D point clouds, the current state of the art point cloud processing systems often begin with discretizing 3D space into voxels or other regular formats to feed into further processing using machine learning methods [2]. While this approach has been proven to work well for classification tasks [3], it suffers from discretization issues, and loses details past the resolution of the voxel grid. On the other hand, fine voxel resolution results in excess memory usage and hence is computationally intractable. For detection tasks, the input space is often much larger and more complex, so smaller objects cannot be expressed within the voxel grid unless the voxel resolution is extremely high thus resulting in excessive memory requirements. Standard voxel-based approaches are not feasible for 3D object detection because discretization issues would only be magnified for larger input spaces, making current 3D techniques impractical [2].

Deep learning methods commonly encode 2D images into a smaller latent vector which can be processed by fully connected layers. For 3D images in deep learning pipelines, each step of this encoding process requires an extra dimension and therefore requires significantly more processing and memory. Because the size of the convolutional intermediate within the deep learning pipeline increases cubically with respect to density or resolution of voxelization step, developing efficient algorithms for processing raw point clouds is imperative.

With recent advances and increase in availability of 3D sensors such as LIDAR, point cloud data has become more prevalent [4]. For scene understanding, 3D data is usually processed using RGB-D data frame by frame, i.e. depth data is overlaid onto 2D imagery for a given time step to create "2.5D" data [5]. These approaches rely primarily on 2D object detection approaches, and the depth information merely provides information on distance from the camera in 3D space. Using 3D geometry directly provides new information that may not be available in the 2.5D data. However, directly using point clouds for 3D object detection is a difficult task and is thus much less explored as compared to 2D object detection. The difficulty primarily lies in 3D scene's varying length, width, and height dimensions. Most existing machine learning pipelines such as those used for processing 2D imagery require input data to have constant dimensions.

Object detection based solely on 3D geometry is not as prevalent for a number of reasons. First, point cloud data is highly irregular compared to 2D image data. Each point has three floating point values representing coordinates, and cannot be easily discretized. This characteristic of point

clouds creates difficulty in preserving geometry, especially since convolutional neural networks require constant input and output sizes. Second, 3D scenes are generally much more sparse than 2D images. For example, if scenes from the Matterport3D dataset is voxelized into $1m^3$ voxels, around ninety percent of voxels contain no points. This means any algorithm applied to these scenes must be extremely robust to empty space and sensitive to objects much smaller than $1m^3$.

In this work, we propose a Sparse Sampling Neural Network pipeline to overcome some of the above issues. Specifically, we focus on efficiently discritizing large, unordered point cloud data without information loss commonly seen in existing methods. We propose an operation for creating a regular structure from point clouds that can be directly fed into a deep neural network for object detection. This operation is combined with a feed-forward convolutional network that infers object categories as well as associated bounding boxes. The pipeline is trained end-to-end and directly processes point clouds for detection.

We test our pipeline on indoor datasets, where training and testing examples are rooms of various dimensions. We run benchmarks on the Matterport3D Research Dataset [6] as well as the Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [7]. We also provide a rough comparison between our benchmarks and existing results on the SUNRGB-D 2.5D datasets.

2 Related Work

2.1 2D Object Detection

Object Detection first became popular in the 2D image domain. Region-proposal based methods have been a common and effective approach to object detection in 2D imagery [1]. Faster R-CNN, for example, uses a single network for object detection and classification and surpasses previous benchmarks on common datasets [8]. Since the introduction of Faster R-CNN, other prominent detectors have been developed. These include Single Shot Multibox Detector (SSD) [9] which uses classic 2D object recognition frameworks and ResNet [10] which infers region proposals with associated class predictions after each max pooling step. This latter method allows region proposals at varying scales and aspect ratios while adding minimal computational overhead. These region proposals are concatenated and non-maximum suppression is applied to create a set of object proposals for an entire scene. Similar to SSD, our proposed method also generates region proposals at multiple feature layers in order to detect objects at different scales while keeping our model structure simple.

2.2 3D Object Detection and Recognition

3D object detection methods have used various forms of 3D data in order to create region proposals. Some methods such as Deep Sliding Shapes (DSS) [11], 2D Driven [5], and COG [12] employ RGB-D data. Others, such as Sliding Shapes [13], directly process point cloud data by using geometry-encoded features or by voxelize point clouds, such as Vote3Deep [14]. Others still use sensor fusion from RGB data and point clouds, such as Frustum PointNet [15] and Multi-View 3D object detection network (MV3D) [16].

Current methods for processing point clouds are limited by the difficulty of hand-selecting features and by the loss of information introduced by naively discretizing 3D space. To overcome these challenges, different methods of processing point clouds have been adopted for differing tasks. For example, O-CNN [2] uses an octree-based approach for efficiently representing 3D shapes for object classification and shape segmentation. PointNet++ [17] and its predecessor PointNet [4] directly process unordered sets of points from point clouds for object classification, part segmentation, and scene segmentation. For object detection, VoxelNet [18] provides an innovation on the conventional discretization of 3D space by using a voxel feature encoding (VFE) layer to learn important information that might otherwise be lost from discretizing voxels. This approach has been demonstrated to be effective in 3D point clouds for outdoor tasks. Recently, Zhou *et al.* [18] proposed VoxelNet, which addresses 3D detection for autonomous vehicles, such as KITTI. Our work can be applied to outdoor datasets as well, however in this thesis we primarily focus on indoor datasets.

Field Probing Neural Networks (FPNN) [19] work on the 3D object recognition domain, and use "probes" to understand the 3D geometry of pointcloud objects. The locations of the probes are trainable and arrange themselves into various patterns during the training process. The advantage of this approach over other 3D object recognition approaches is that FPNN skips the voxelization pre-processing step that other approaches use, maintaining fine detail throughout the pipeline. As seen shortly, the probes from FPNN have inspired the probe operation of our proposed method, however we have made several changes to adapt it to object detection. For example, unlike [19], we convolve our probes along the entire input pointcloud. Although in our proposed method called Sparse Sampling Neural Networks (SSNN) we do not train the positions of the probes, we do use a similar nearest neighbor algorithm to FPNN.

3 The Sparse Sampling Neural Network (SSNN)

This section describes our proposed SSNN model in Section 3.1 as well as its associated loss function in Section 3.2 and training pipeline in Section 3.3.

3.1 Model

The SSNN model consists of three parts: probe layer, dot product layer, and a multi-scale feed-forward 3D convolutional network as shown in Figures 1 and 2. The probe layer directly consumes 3D point clouds and discretizes the input space. The dot product layer then associates trainable weights with the probe layer. This is then passed into a 3D convolutional network that uses hook layers at multiple scales similar to Single Shot MultiBox Detector [9], described later in this section. Our model predicts the coordinates of bounding boxes and classification probabilities shown in Figures 3 and 1. We choose this architecture because the hook layers allow for different scale bounding box predictions as well end-to-end training of the network. A diagram of the hook architecture can be seen in Figure 4.

3.1.1 Probe Layer The primary goal of this layer is to convert the unordered point cloud input into a form a neural network can process. This is the first step in the SSNN pipeline. This layer is similar to FPNN's sensor layer [19] where probes are used to gather information about 3D space. Since FPNN attempts to solve 3D object recognition rather than 3D object detection, its input is point clouds of individual objects rather than entire scenes. As described shortly, we extend this idea to larger 3D scenes by initializing multiple sets of probes and sliding these probes across the entire scene.

To be more specific, as seen in Figure 5, we initialize N randomly distributed "probes" in a cube of side length d. Each probe is a single point in 3D. The probes are the red points, the cube of side length d is the smaller cube, and the input pointcloud is padded with zeros or truncated to fit inside the larger cube with side length l. This truncation is necessary for scenes with dimensions larger than l. However, truncation is only necessary for less than 1% of scenes in all three datasets. For example, long hallways and large auditoriums are truncated. Missed objects due to truncation are uncommon even for these edge cases.

Each of the N probes can be represented by a vector $\boldsymbol{p}$ of length three corresponding to its location in 3D space. Let $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_M} \in X$ be the set of points in the input pointcloud, also each a vector of length three containing its xyz coordinate. Each probe runs a nearest neighbors search

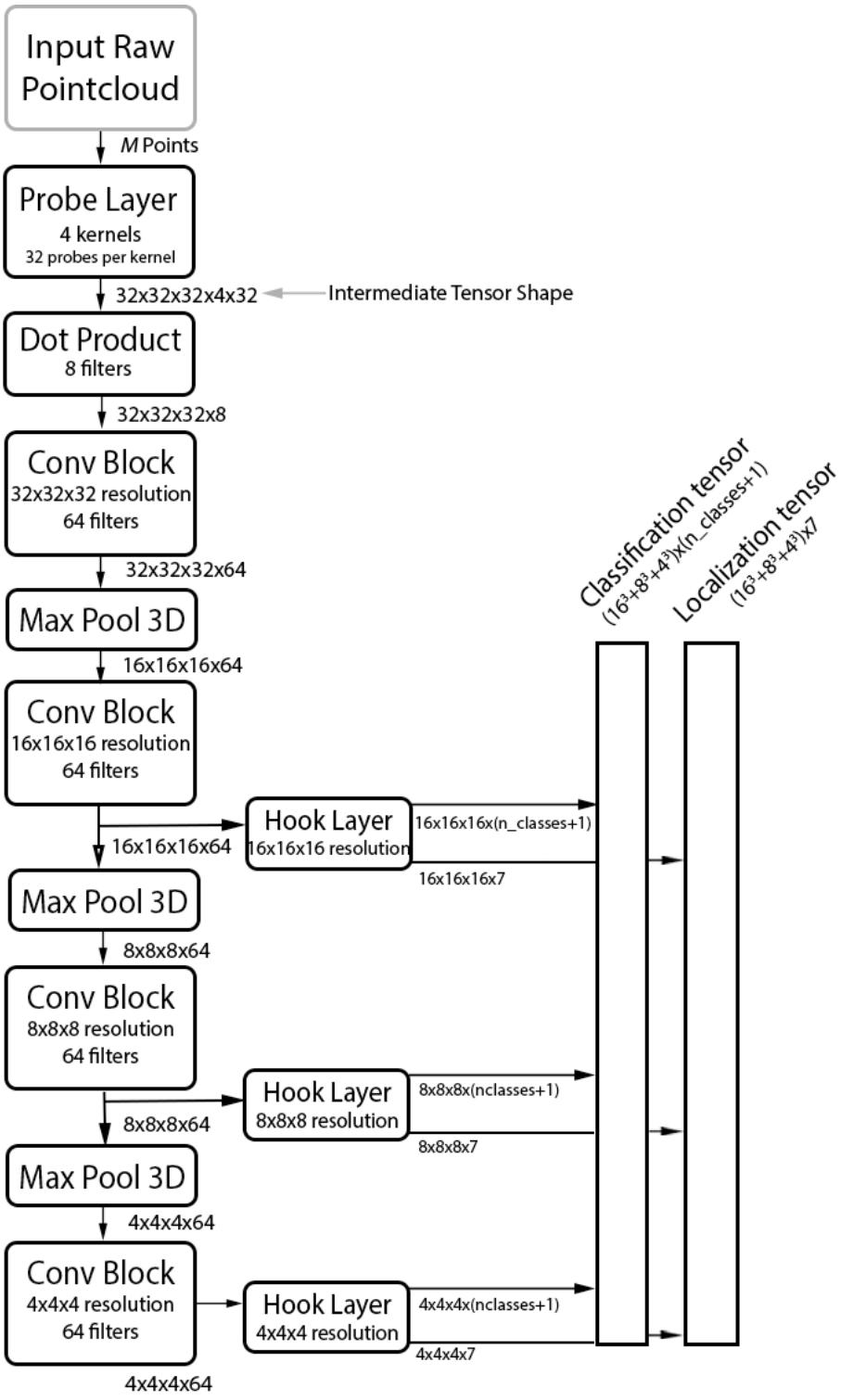


Fig. 1: SSNN model architecture with constants.

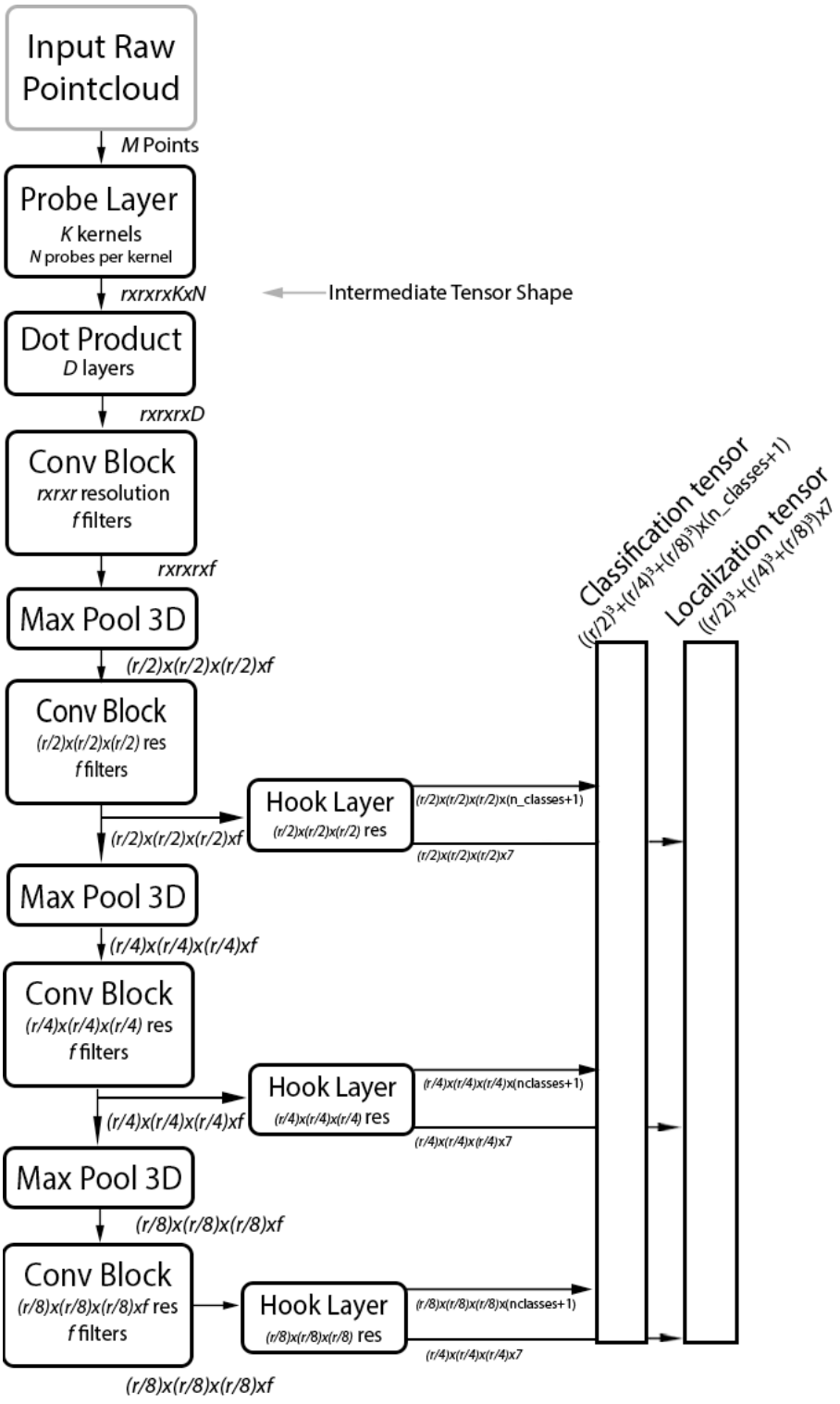


Fig. 2: SSNN model architecture with symbols.

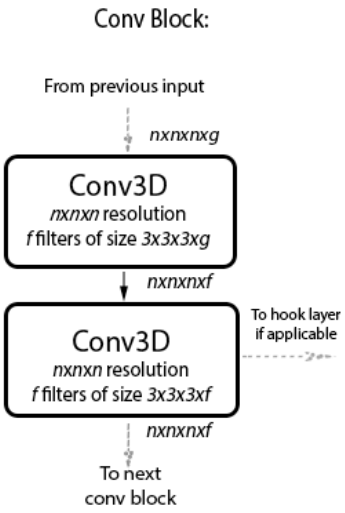


Fig. 3: A convolutional block consisting of two 3D convolution operations, used in Figure 1.

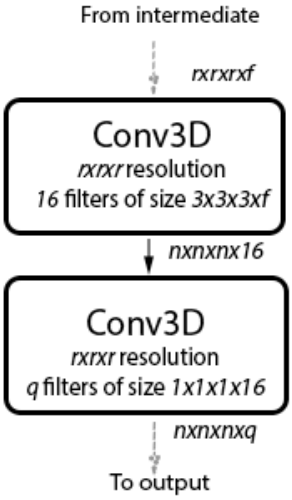


Fig. 4: A hook layer consisting of two 3D convolution operations, used in Figure 1. q is $n_{classes} + 1$ for a classification hook output and 7 for a localization hook output.

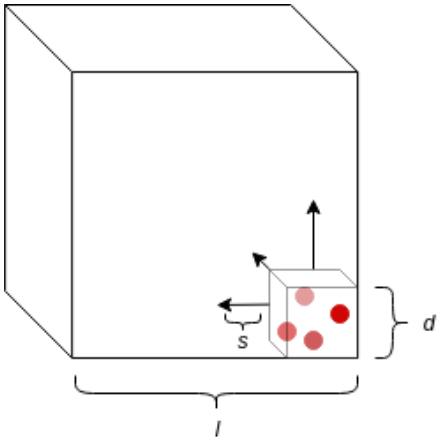


Fig. 5: Illustration of the probe operation.

on the input point cloud and returns a value corresponding to the proximity of nearby points in the entire pointcloud using a truncated linear function:

$$f_p(\boldsymbol{x_1}, \ldots, \boldsymbol{x_M}) = \max(d, c - \min_{1 \geq i \geq M}(\|\boldsymbol{x_i} - \boldsymbol{p}\|)) \tag{1}$$

where m and c are a constants chosen by cross validation.

We call this set of N probes a "kernel." In order for a kernel to process information from the entire point cloud, the kernel slides at discrete increments of step size s, where $s < d$. A slide operation consists of adding s to the same dimension for each probe in a kernel. After each slide, each probe runs the nearest neighbors search described above in Equation 1. Therefore, the entire kernel runs the nearest neighbor search at r^3 locations, where $r = \frac{l}{s}$ is defined to be the input resolution for the probing layer. Finer input resolution means more detail but more memory usage and slower training. Courser input resolution means less detail but less memorry usage and quicker training. This operation is meant to create information from the entire pointcloud with fine granularity. We refer to this sliding operation as convolution since we are gathering the probes' response as we slide the kernel across the pointcloud. Due to the fact that point clouds do not have any natural ordering, we convolve across physical dimensions rather than pixels, as is the case with 2D convolutions. This way, there is no need for prior ordering of the point cloud input. K different kernels are convolved over the entire pointcloud, which has width, height, and depth of size l. Thus, the output from the probe layer generates a tensor of shape $r^3 \times K \times N$. The probe layer allows us to scale this operation to larger spaces without information loss common in other voxelization approaches. The variables N, s, d, and K are all determined by 5-fold cross-validation.

Unlike FPNN [19], the positions of the probes in the probe layer are fixed rather than trained after they are randomly initialized. This probe layer allows for arbitrary pointcloud input size because it converts continuous space into discretized space while still retaining granular information from the pointcloud. Pointclouds can be sparse or dense and the probe layer can extract the same information regardless.

To clarify the symbols used in this section, refer to Table 1.

Symbol	Definition
d	Kernel width, height, and depth
l	Input grid width, height, and depth
r	Resolution of probing layer
s	Kernel step size
$\boldsymbol{p}$	3D coordinate of a probe
$\boldsymbol{q}$	Output tensor of probe layer
$\boldsymbol{t}$	Output tensor of dot product layer
$\boldsymbol{w}$	Weights of dot product layer
D	Number of dot product layers
K	Number of kernels
N	Number probes per kernel
M	Number of points in an input pointcloud

Table 1: Mathematical symbol definitions.

3.1.2 Dot Product Layer Unlike FPNN [19], the probes in each kernel have fixed positions. Thus, we need trainable weights associated with each probe to learn which patterns are more significant than others. We solve this problem by adding a dot product layer after the probe layer. The dot product layer associates a trainable weight represented by a weight vector $\boldsymbol{w}$ for every probe p and applies the dot product operation as seen in Equation 2. For example, for one kernel with N probes, we initialize N weights, one for each probe. The probe response, i.e. the result of the nearest neighbor algorithm, is multiplied by its respective weight. Then, the multiplied weights for this kernel is summed across all probes in a kernel as seen in Equation 2.

$$\boldsymbol{t}_{(a,b,c)} = (\boldsymbol{q}^{\top}_{(a,b,c)}\boldsymbol{w}_{p,1}, \dots \boldsymbol{q}^{\top}_{(a,b,c)}\boldsymbol{w}_{p,i}, \dots \boldsymbol{q}^{\top}_{(a,b,c)}\boldsymbol{w}_{p,D}) \tag{2}$$

$\boldsymbol{w}$ is a weight vector, $w_{p,i}$ is the ith weight associated with probe p, $\boldsymbol{q}_{(a,b,c)}$ is the probe responses for a kernel at voxel grid coordinate (a, b, c), $\boldsymbol{d}_{(a,b,c)}$ is the output from the dot product layer at

voxel grid coordinate (a, b, c) and D is the number of dot product layers. The number of discrete steps for the output of the probing layer is $(\frac{l}{s})^3$ as described in the probe layer section. We refer to this structure as a "voxel grid".

We assign multiple dot product layers per kernel to lower the number of kernels needed for training. Combined with the probe layer, the dot product layer learns basic 3D geometries such as flat planes and edges as seen in Figure 6. We initialize multiple dot product layers per kernel since the different patterns can be learned from the same kernel. Figure 6 shows a visualization of a kernel with 32 probes. Higher weighed probes are indicated with darker points. In most cases, higher weighted probes are clustered near each other.

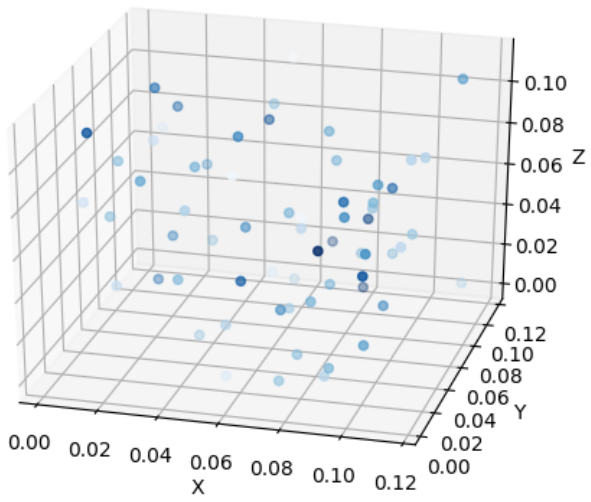


Fig. 6: Visualization of a randomly initialized kernel with 32 probes. Darker shades indicate higher weights. In most cases higher weighted probes are clustered near each other.

Our hypothesis is that if a sufficiently large density of probes are initialized, then it is sufficient to tune the dot product layer exclusively and skip training the probing layer. This saves the extra computation during training and reduces training complexity by reducing the total number of trainable weights in the pipeline.

Multi-scale 3D CNN As is the case for any region proposal machine learning algorithm, direct inference of the position and size of bounding boxes with classical regression approaches is impractical. To handle this, we use a similar approach detailed in Single Shot MultiBox Detector

[9] extended to the 3D setting: our multi-scale 3D CNN consists of a series of 3D convolutions and max pooling layers as seen in Figure 1, where the last three convolution blocks of the network are appended with hook layers. The data passed between intermediate layers, is a 3D voxel grid with a varying number of features per voxel depending on the number of filters in the convolutional block. The input for a hook layer is a voxel grid, which allows the training process to have constant dimensions between training examples. For every voxel, the output proposes likelihood scores for each class as well as the relative offset, dimension, and rotation of a region proposal as seen in Figure 7. We call these outputs the classification tensor (cls) and the localization tensor (loc), respectively. The classification tensor has length $(n_{classes} + 1)$, where $n_{classes}$ is the number of classes in the dataset, and the extra unit of length representing the null class. The localization tensor has dimension 7 with values $loc = (x, y, z, w, h, d, \theta)$, where x, y, z are offset, w, h, d are dimensions, and θ is yaw. Classification and localization are predicted separately to create more structure within our pipeline. We found that separating classification and localization resulted in more effective training. Specifically, overfitting, convergence, and overall accuracy are all improved in doing so. For our datasets, roll and pitch are fixed to be axis aligned. A hook layer classifies and generates likely region proposals for an entire voxel grid at different scales by processing the input at different resolutions. Finer resolutions produce smaller region proposals and coarser resolutions produce larger region proposals. Utilizing multiple resolutions allows our network to achieve higher accuracy for objects of various scales. The hook layer does this by applying two 3D convolutional layers of the same size as the hook layer's input, one with $(n_{classes} + 1)$ filters and the other with 7 filters. Since the filters predict values for each voxel, we can directly train the hook layer without any fully connected layer. For example, for a 3D convolution intermediate of size $\frac{r}{2} \times \frac{r}{2} \times \frac{r}{2}$, the hook layer would generate $\frac{r}{2}^3$ region proposals and infer the most likely object category for each region proposal as well. In most cases, this will predict the null class. When it predicts a specific object category with higher than 50% confidence, we add the region proposal to our final list of object predictions. Because each hook layer predicts region proposals at a different scale, this allows the network to naturally produce predictions at different scales in one feed-forward pass.

We use three hook layer resolutions: 16x16x16, 8x8x8, and 4x4x4 as seen in the hook layers in Figure 1. Smaller objects are detected in the higher resolution outputs such as 16x16x16, while larger objects are detected by lower resolution outputs such as 4x4x4.

Figure 8 shows a bird's eye view of the the output of the final hook layer with a 4x4x4 resolution looking down the z-axis. Row (a) depicts the classification predictions while row (b) depicts classification labels. The labels in row (b) are constructed by calculating the intersection over union of each

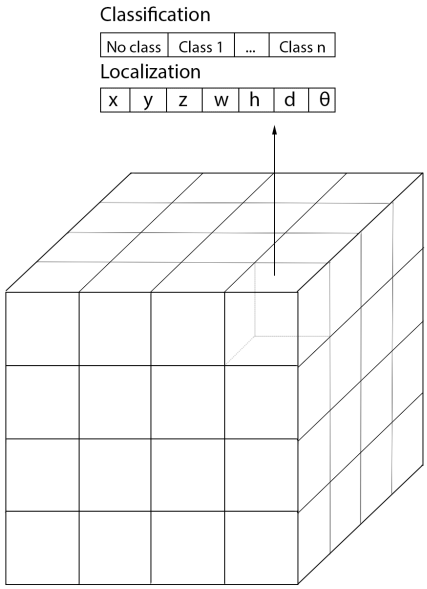


Fig. 7: Visualization of the output of a hook layer. Each hook layer outputs a grid with classification and localization vectors at each voxel. The resolution of this grid is 4x4x4, and correspond to the last hook layer in Figure 1 which has length 4^3, or length 64. In the localization vector, x, y, z are offset, w, h, d are dimensions, and θ is yaw. For our datasets, roll and pitch are fixed to be axis aligned.

label bounding box to each voxel over the 4x4x4 resolution. This is done in a two-step process. First, for each label bounding box, the voxel with the highest intersection over union is found. The class label for that voxel is set to the corresponding one-hot vector of the object and the relative offset, dimension, and rotation of the bounding box is associated with that voxel. Second, for each voxel, if the intersection over union of the bounding box is greater than 0.25 and it does not already contain a label, the class and bounding box label is also associated. This second matching phase not only increases the number of positive examples in a relatively sparse space, but also allows prediction for any shape from any scale hook layer. With this method of approximate matching, this second matching stage removes the requirement for the model to automatically learn a form of non-maximum suppression. Rather, it can propose bounding boxes from multiple hook layers during test time, and non-maximum suppression can be applied afterwards. For non-maximum suppression, we find each pair of overlapping bounding boxes in descending order of confidence and delete the bounding box with lower confidence. Classification predictions generally generate more positives than actually exist in the labels as shown in the leftmost column of Figure 8(b) because the same object can be predicted from more than one voxel. In other words, for the left most column, five voxels registered

a confidence value of greater than 50%. These all predicted a box corresponding to the ground truth label and were later suppressed into one bounding box prediction after non-maximum suppression.

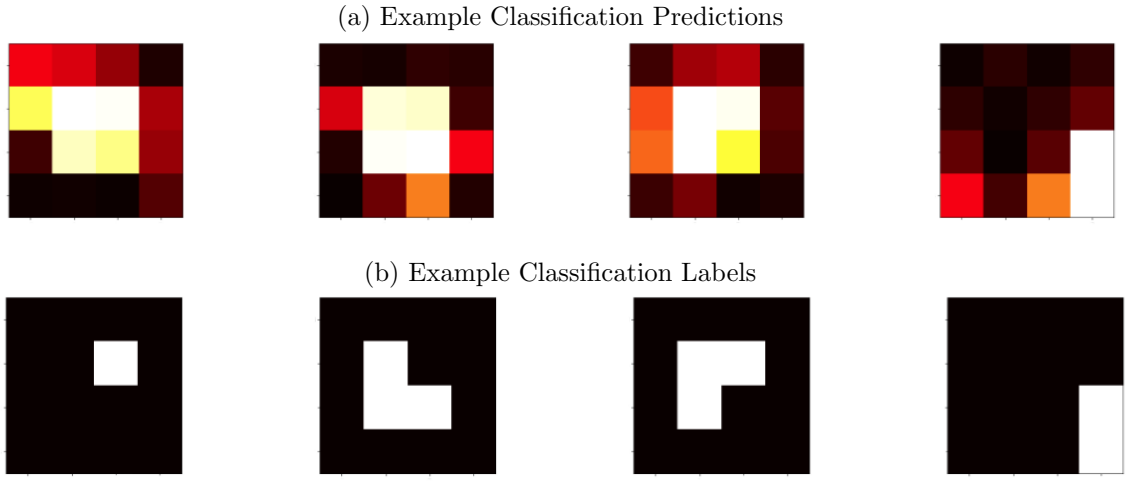


Fig. 8: Four different example of classification prediction in (a) and corresponding label visualization in (b). This is a bird's eye view of a 3D voxel grid from the output of the last hook layer, which has a resolution of 4x4x4 as seen at the bottom of Figure 1. For both the top and bottom row, the brighter a voxel is, the more likely an object exists at that location.

After prediction, the output must be converted into bounding box format. For each voxel grid with a prediction confidence higher than 50%, we add the voxel's grid position to its predicted offset, multiply the voxel resolution by its predicted dimension, and apply its predicted rotation. As seen in Figure 9(b). The last step is to apply non-maximum suppression and to calculate the mean Average Precision (mAP) score.

3.2 Training

To avoid overfitting, we employ several data augmentation techniques in our training process. Each input point cloud was rotated along its vertical axis by multiples of 45°. The points in the input point clouds were also perturbed by adding a tensor of Gaussian noise. This data augmentation proved extremely important for objects with few training examples, and increased mAP by up to 20% in some cases. We also used dropout [20] in the hook layers and between the dot product layer and the first 3D convolution. We found that regularization methods such as dropout combined with low learning rates significantly reduced overfitting. We then set parameters using 5-fold cross validation, optimizing for an overlap threshold of 25% for all datasets, i.e. the intersection over union for predicted and labeled bounding boxes is greater than 25%.

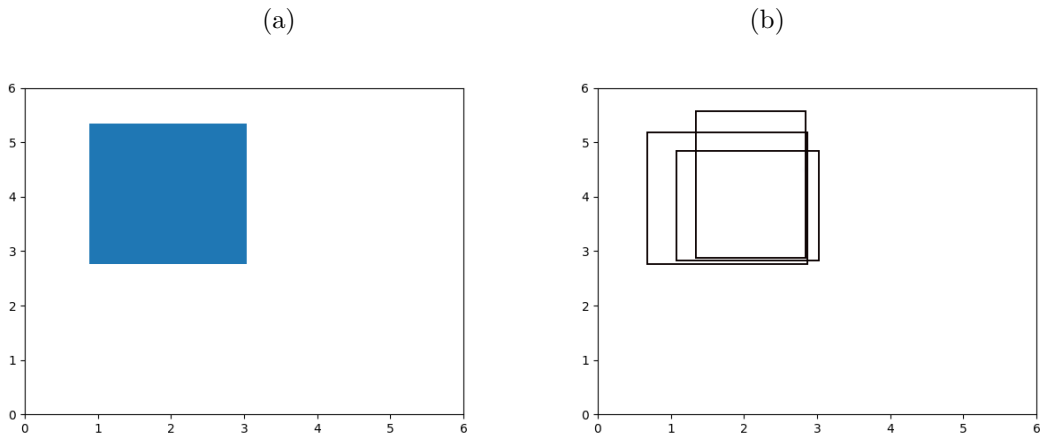


Fig. 9: Bounding box predictions after relative location is applied to the classification predictions. (a) is the label and (b) is the predictions. Non-maximum suppression is applied afterward to (b) in order to remove duplicate detections.

We use our model from Figure 1 for all training and testing. The probe operation has a step size of s, a kernel size of d, and an input resolution of r, where $s = \frac{l}{32}$, $d = \frac{3l}{32}$, and $r = \frac{l}{s}$. 4 kernels are used with 32 probes each, and 8 dot product layers are initialized per kernel. All weights are initialized with Xavier initialization [21]. We converted all datasets into colored pointclouds and converted the RGB component of our pointcloud to HSV. The multi-scale 3D CNN consists of four sets of "convolutional modules", with each module consisting of two 3D convolutional layers followed by a max pooling layer as shown in Figure 1. Hook layers are appended to the last three modules. We use the Adam optimizer [22] with a learning rate of 10^{-5} and a batch size of 4. We train all datasets for 30 epochs. The total training time takes around 4 hours for Matterport and SUNRGB-D and about 2 hours for the Stanford dataset on a NVIDIA Titan X (Pascal) with a Samsung SSD 960 EVO M.2 NVMe for streaming data.

3.3 Loss formulation

The loss formulation comes in two parts: classification and localization. Classification (cls) determines which class a particular voxel is likely to be, and localization (loc) determines the offset and dimension of the region proposal relative to a given voxel. We structure our network such that there is no more than one region proposal per voxel at the same scale, so each voxel can only predict one region proposal. Classification is represented as a softmax one-hot vector $\mathbf{c}$ with an extra category for the null class. We use cross-entropy loss for classification, weighted differently between positive examples and negative examples. We define cross entropy loss to be

$$\mathrm{L}_{CE}(\mathbf{v}, \hat{\mathbf{v}}) = -\sum_i v_i \log\, \hat{v_i} \tag{3}$$

where $\mathbf{v}$ is a label vector and $\hat{\mathbf{v}}$ is a prediction vector. This prevents negative examples from skewing the results since around 80% of examples are negative. However, predicting the relative offset and dimension for a voxel without an identifiable object, or "empty" voxel, has no meaning. This empty voxel situation comprises of a large majority of the voxels. In order to circumvent this problem, the relative offset and dimension is only trained when an object is present for a given label voxel. Let N_{pos} and N_{neg} denote the number of positive and negative examples, respectively. The one-hot classification vectors are defined as $\mathbf{c}$ and $\hat{\mathbf{c}}$ for classification labels and classification prediction, respectively. Then, the expression for classification loss is as follows:

$$L_{cls} = \frac{1}{N_{pos}} \sum_{i \in Pos} L_{CE}(\mathbf{c_i}, \hat{\mathbf{c}}_\mathbf{i}) + \frac{\alpha}{N_{neg}} \sum_{j \in Neg} L_{CE}(\mathbf{c_j}, \hat{\mathbf{c}}_\mathbf{j}) \tag{4}$$

where $\alpha \in \mathbb{R}$ and is chosen with 5-fold cross validation.

Localization vectors are represented $\boldsymbol{loc} = (x, y, z, w, h, d, \theta)$ for each voxel; the first three elements represent the x, y, and z offset from the current voxel center, the next three represent the width, height, and depth dimensions of the predicted bounding box with respect to the voxel resolution, and theta represents the yaw of the bounding box. We use L_2 loss, or mean-squared error, for localization:

$$L_{loc} = \sum_{i \in Pos} \|\boldsymbol{loc_i} - \hat{\boldsymbol{loc_i}}\|_2^2 \tag{5}$$

Where $\boldsymbol{loc_i}$ and $\hat{\boldsymbol{loc_i}}$ denote the ith localization label and localization prediction, respectively.

Each hook layer outputs both classification scores and localization adjustments. Since our model is trained end-to-end, we have a single objective loss, which is a linear combination of the classification and localization losses.

$$L_{obj} = L_{cls} + \lambda L_{loc} \tag{6}$$

$\lambda \in \mathbb{R}$ is a constant chosen by 5-fold cross validation.

3.4 GPU implementation

Due to the large number of nearest neighbor operations required for our probing layer, efficient utilization of GPU resources is crucial. The majority of our pipeline uses Tensorflow methods, however the probe operation was coded from scratch and integrated into the Tensorflow API. Therefore,

an important part of this work has to do with optimizing our probe operation for running on the GPU. Because we need to run a k-nearest neighbors (k-NN) algorithm millions of times per scene for the probe operation, we make the following approximation: probes do not require information from points that lie outside the dimensions of its kernel, i.e. points that lie outside the $d \times d \times d$ sliding box. This allows each probe operation to only search a small subset of the point cloud. In addition, this approximation does not affect performance due to the sliding nature of our kernels.

This is reflected in Equation 1 by the constant d, allowing us to restrict our k-NN search to points a distance d or less from the probe. In order to run this operation on the GPU, k-NN must be run in parallel. Therefore, we created a "Grid-list" data structure, where points are organized by voxel grid. This allows many k-NN operations to be run in parallel. The voxel grid's dimensions are defined by the number of steps, or resolution r, of our probing layer. These organized points are then flattened into a single one-dimensional array with an associated mapping from voxel position to index. For each probe in a kernel, a nearest neighbor query is run using the mapping for the points in a given voxel position.

4 Experiments

We tested our pipeline on three datasets: the Matterport3D Research Dataset, the Stanford 3D Indoor Spaces Dataset, and the SUNRGB-D Dataset. In all the experiments, we used the model in Figure 1.

4.1 Matterport3D Research Dataset

The Matterport3D Research Dataset contains 10,800 3D scenes generated from RGB-D images using Matterport's Pro 3D Camera. We set aside 10% of the dataset for testing and used the rest for training: specifically, 1,080 scenes were used for testing and 9720 for training. The object distribution can be seen in Figure 10. The Matterport3D dataset is in mesh format, so we converted the meshes into point clouds by sub-sampling as seen in Figure 11. Then, we took the minimum and maximum x, y, and z coordinates from the segmentation labels and converted these into bounding boxes. We opted to test on this dataset because we wanted to see how our model performed on a dataset where points are relatively evenly distributed across all objects. Most 3D detection datasets focus more on amodal detection.

Table 2 shows our results on 10 categories in the dataset and the associated ROC curve in Figure 12. For the probing layer, we used parameters $l = 7.50, s = 0.23, d = 0.70, r = 32$. A visualization

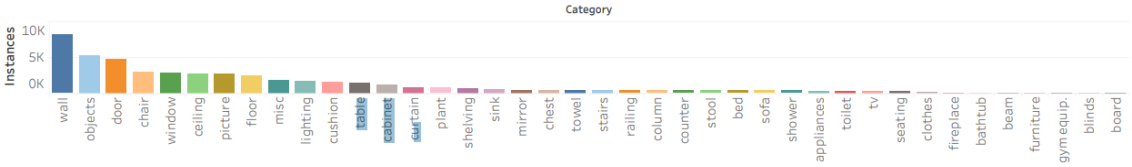


Fig. 10: Object distribution for the Matterport3D dataset.

Fig. 11: Example scenes from the Matterport dataset.

of predictions can be seen in Figure 13. Our network performed well on objects with less in-class variance, such as bed and bathtub. These objects often have the same appearance throughout the dataset. Objects such as table, dresser, and desk have widely varying colors, shapes, and dimensions from example to example. Our network seemed to overfit heavily on these types of objects.

bathtub	bed	bookshelf	chair	desk	dresser	nightstand	sofa	table	toilet	mAP
65.9	92.1	43.4	66.0	31.0	37.2	72.9	45.2	39.3	51.8	54.48

Table 2: **3D object detection AP**. mAP performance of ten objects trained on the Matterport3D dataset using 0.25 IoU.

4.2 Stanford Large-Scale 3D Indoor Spaces Dataset

The S3DIS dataset is collected in 6 large-scale indoor areas [7]. There are 270 total scenes in the dataset. Two example scenes are shown in Figure 14. The dataset contains semantic labels for 13 classes: ceiling, floor, wall, beam, column, door, window, table, chair, sofa, bookcase, board, and clutter. Of these, we opt to test our methods on 4 the same classes as previous work [7] [4]: table, chair, sofa, and board seen in Table 3.

An illustration of the label generation and prediction generation process for three objects in the Stanford dataset can be seen in Figures 15 through 17. Label generation starts with a bounding

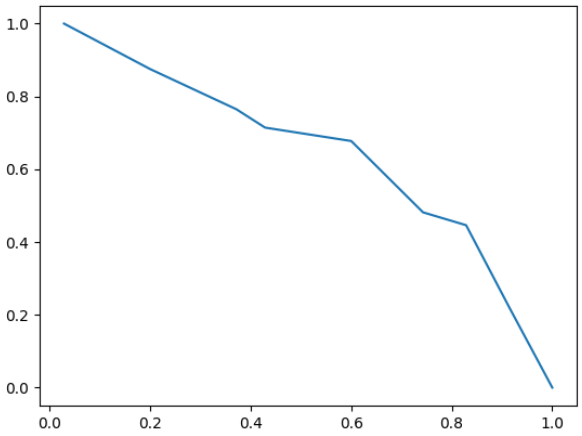


Fig. 12: ROC curve for all categories in the Matterport3D dataset.

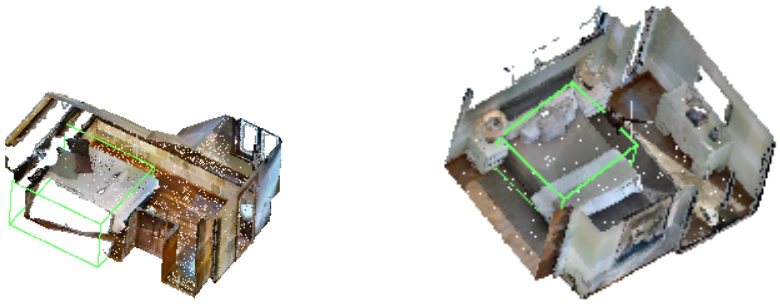


Fig. 13: Example bed detections on the Matterport dataset.

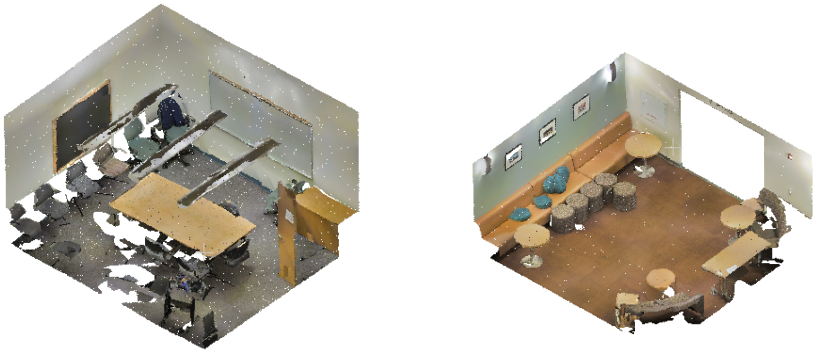


Fig. 14: Example scenes from the Stanford dataset.

	table	chair	sofa	board	mAP
Armeni et al. 0.5 IoU [7]	46.02	16.15	6.78	3.91	18.22
PointNet 0.5 IoU [4]	46.67	33.80	4.76	11.72	24.24
Ours 0.5 IoU	24.02	10.94	3.48	1.63	10.02
Ours 0.25 IoU	79.83	89.79	39.22	42.88	62.93

Table 3: **3D object detection AP**. Results on S3DIS dataset.

box and is converted into classification and localization format. Proposal generation is the reverse process of transferring the classification and localization format into bounding boxes. In Figure 15 Example 1 for the table object, the proposed box has an IoU of greater than 0.25 with the label and is considered a true positive. In fact, there are two accurate bounding boxes proposed, and one box is suppressed by the box with the higher confidence value. This is an example of successful detection. In Example 2 of Figure 15 which is also for the table object, multiple boxes survive non-maximum suppression. This is due to a high confidence prediction in the wrong voxels, seen in the middle resolution of the classification column. This is an example of unsuccessful detection. In Figure 16 Example 3 for the sofa object, the proposed box has an IoU of greater than 0.25 with the label and is considered a true positive. This is a successful detection. In Figure 16 Example 2 for the sofa object, there is a missed detection. This is an unsuccessful detection. In Example 5 for the board object, the proposed box has an IoU of greater than 0.25 with the label and is considered a true positive. This is a successful detection. In Example 2 for the board object, there is a missed detection.

The S3DIS dataset is split into six "Areas". Of the 6 Areas in the S3DIS dataset, we set aside Area 6, which contains 48 scenes, for testing and use Areas 1-5, which contains 222 scenes, for training and validation. For the probing layer, we used parameters $l = 7.0, s = 0.22, d = 0.66, r = 32$. The number of objects per category is shown in Table 4, and an example table detection can be seen in Figure 18. Due to the low number of training and test examples, variance in mAP performance for the Stanford dataset is high, especially for the sofa and board classes. Specifically, the number of test examples for sofa is only 10 which makes the results for sofa in Table 3 unreliable. For most papers, mAP is reported with 0.25 IoU loss [6] [15] for 3D object detection. For this dataset, only 0.5 IoU loss is reported from other papers. Our network's strength is in detection rather than tight bounding box proposals, so we perform poorly using the 0.5 IoU metric. This can be seen in our 52.91% mAP improvement between 0.5 IoU and 0.25 IoU. On the other hand, [4] performs on a per-point basis so finding tighter bounding boxes is more feasible. When we decrease the IoU threshold, our mAP

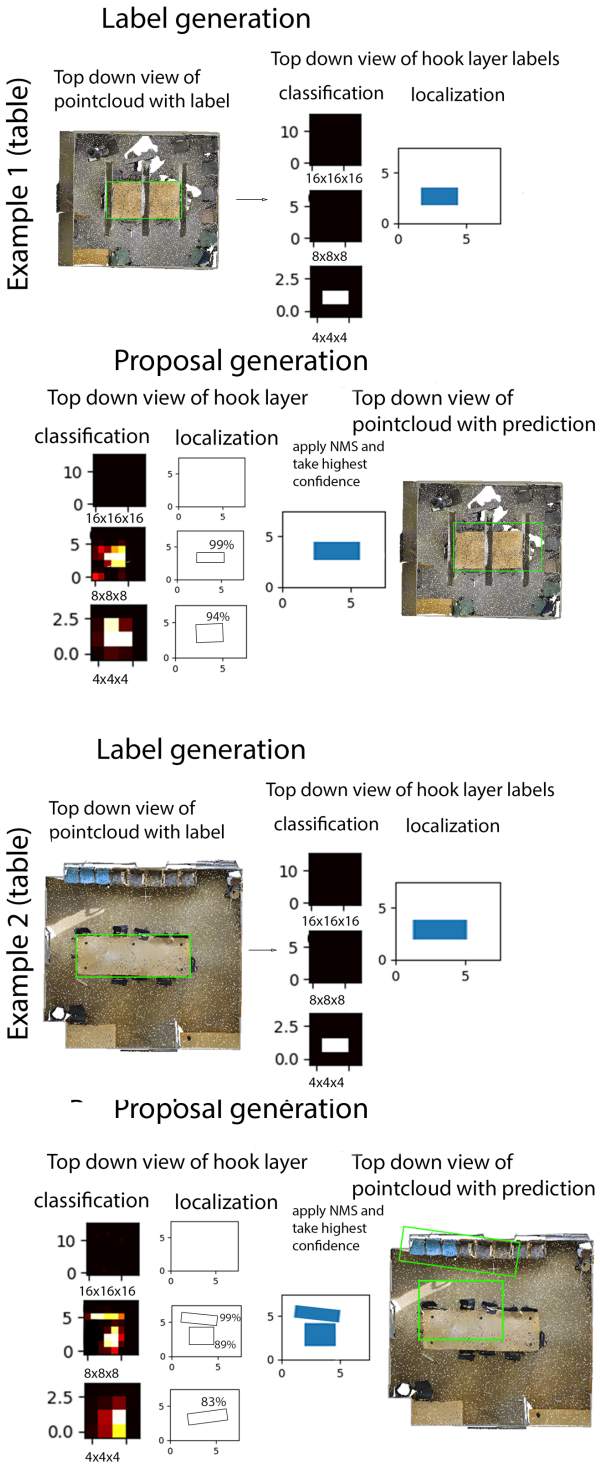


Fig. 15: An illustration of the label generation and prediction generation process for a table in the Stanford dataset. In Example 1 the proposed box has an IoU of greater than 0.25 with the label and is considered a true positive. This is a successful detection. In Example 2, there is one false positive and one true positive. This is an unsuccessful detection.

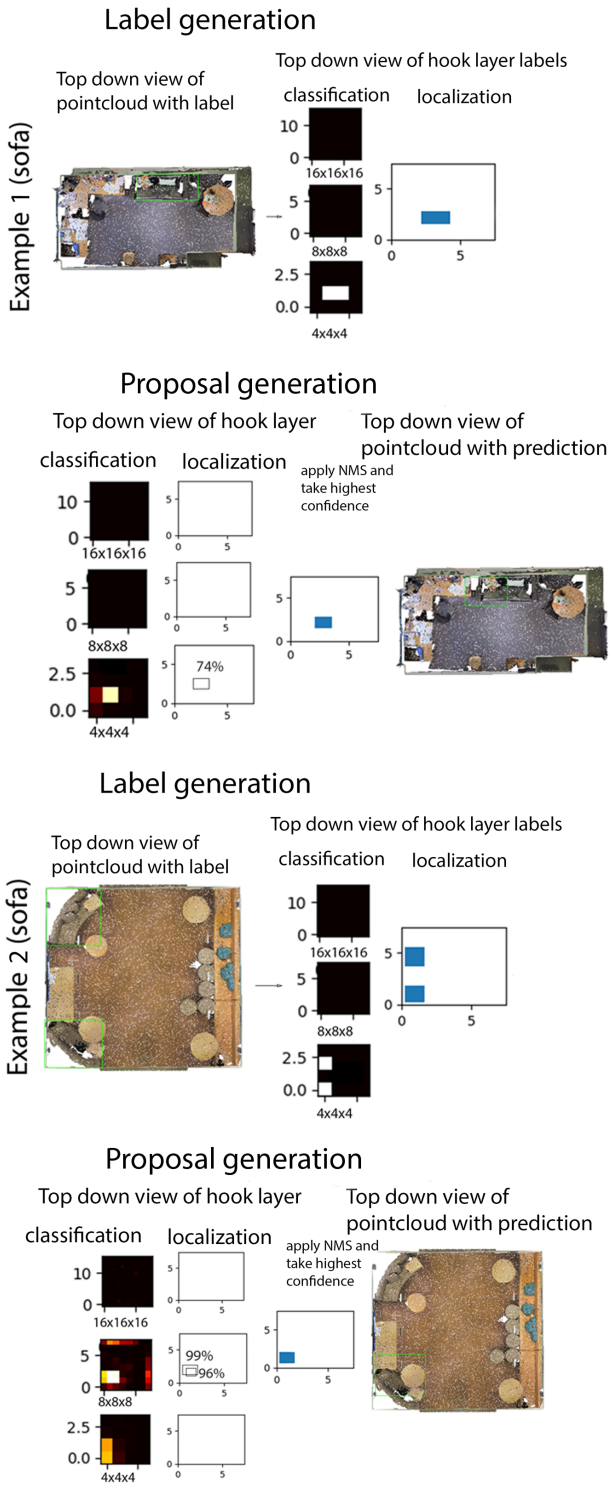


Fig. 16: An illustration of the label generation and prediction generation process for a sofa in the Stanford dataset. In Example 1, the proposed box has an IoU of greater than 0.25 with the label and is considered a true positive. This is a successful detection. In Example 2 there is a missed detection. This is an unsuccessful detection

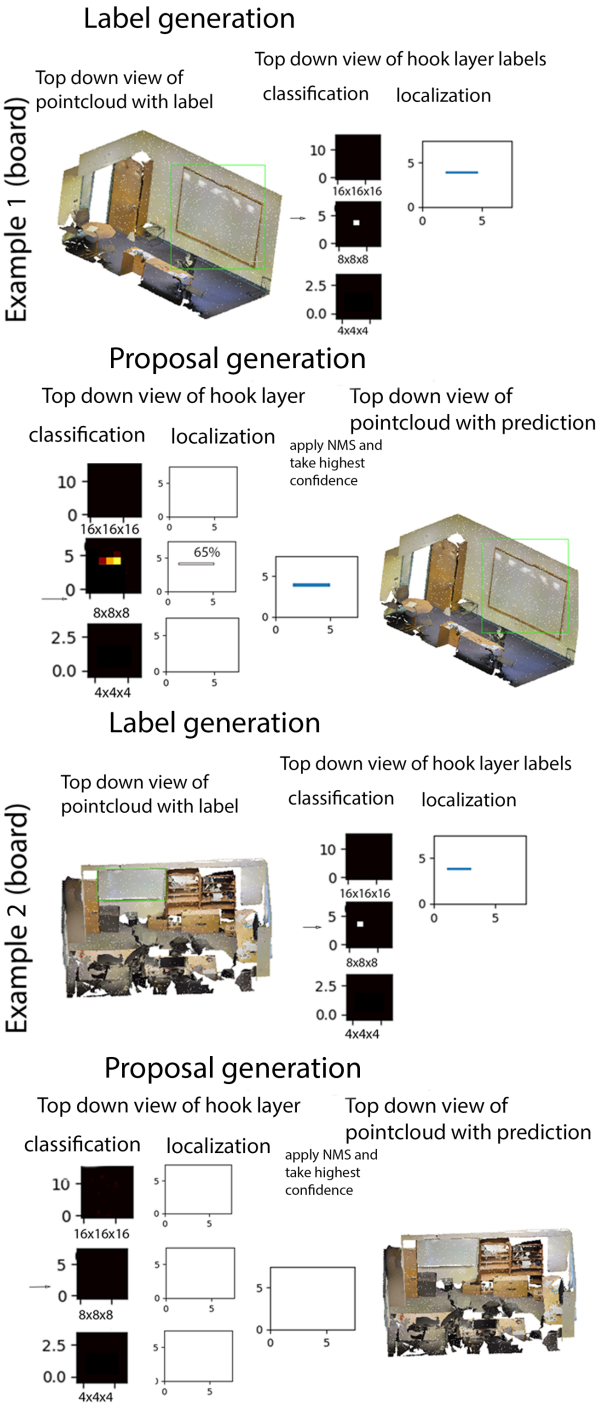


Fig. 17: An illustration of the label generation and prediction generation process for a board the Stanford dataset. In Example 1 the proposed box has an IoU of greater than 0.25 with the label and is considered a true positive. This is a successful detection. In Example 2, there is a missed detection.

increases significantly. Since our pipeline directly infers bounding boxes instead of segments points, performing well on objects such as board is difficult due to the extremely small depth.

	table	chair	sofa	board
Area 1	70	156	7	28
Area 2	47	547	7	18
Area 3	31	68	10	13
Area 4	80	160	15	11
Area 5	155	259	12	43
Area 6	78	180	10	30

Table 4: Object class statistics for the Stanford 3D Indoor Spaces dataset.

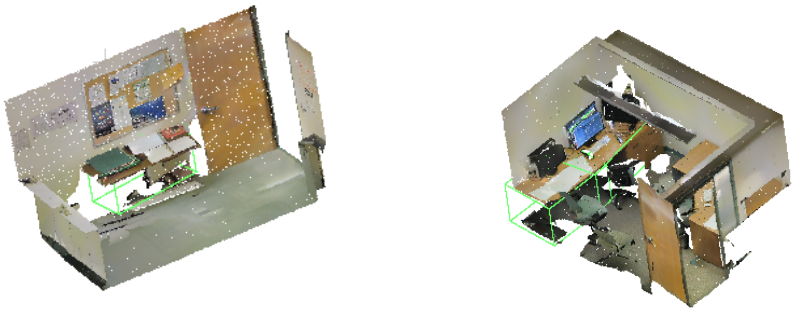


Fig. 18: Example table detections from the Stanford dataset.

4.3 SUN RGB-D Scene Understanding Benchmark Suite

The SUNRGB-D dataset is an RGB-D (2.5D) dataset for indoor scenes that contains 10,000 RGB-D images, and three example scenes are shown in Figure 19. The object distribution can be seen in Figure 20. Since our proposed pipeline primarily relies on 3D geometry, the 2.5D information from a single view is not well suited to our algorithm. Amodal perception does not provide full 3D geometry of any objects in a given scene. Nonetheless, the SSNN pipeline can perform reasonably well compared to other state of the art 2.5D approaches, achieving a mAP similar to Frustum Pointnet [15] seen in Table 5.

In order to run the SSNN pipeline on this dataset, we backproject all pixel coordinates to 3D coordinates using depth information along with camera intrinsic and extrinsic information. This

Fig. 19: Example scenes from the SUNRGB-D dataset projected into 3D space.

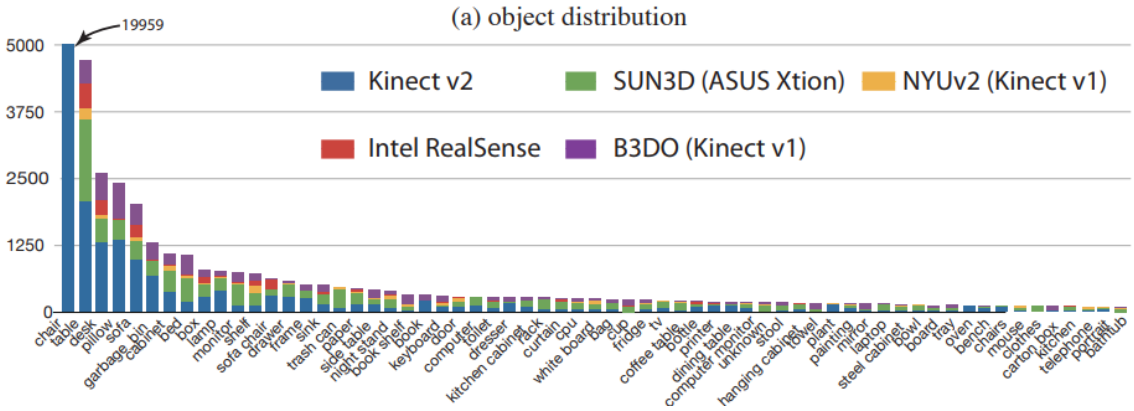


Fig. 20: Object distribution for the SUNRGB-D dataset.

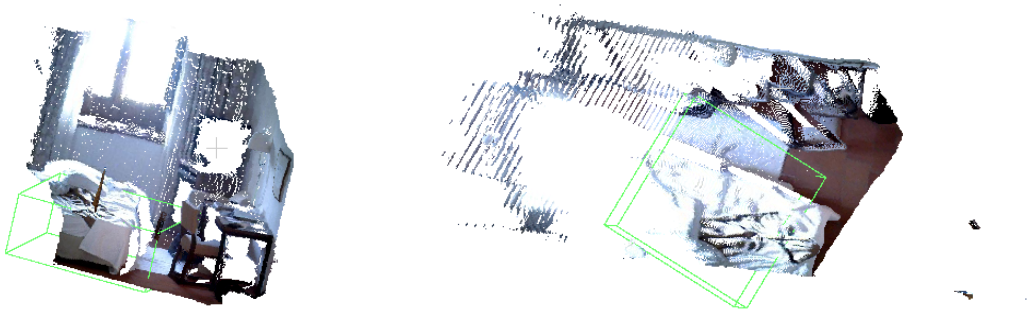
	bathtub	bed	bookshelf	chair	desk	dresser	nightstand	sofa	table	toilet	mAP
DSS	44.2	78.8	11.9	61.2	20.5	6.4	15.4	53.5	50.3	78.9	42.1
COG	58.3	63.7	31.8	62.2	45.2	15.5	27.4	51.0	51.3	70.1	47.6
2D-driven	43.5	64.5	31.4	48.3	27.9	25.9	41.9	50.4	37.0	80.4	45.1
Frustum PointNet	43.3	81.1	33.3	64.2	24.7	32.0	58.1	61.1	51.1	90.9	54.0
Ours	42.5	80.7	22.9	59.7	30.7	12.0	60.5	45.7	48.7	80.1	48.4
Ours w/ 2D	52.0	86.6	15.2	24.3	39.0	14.5	52.1	45.5	58.3	80.0	46.7

Table 5: **3D object detection AP**. mAP performance of ten objects trained on the SUNRGBD dataset for amodel object detection using 0.25 IoU.

generates a pointcloud with a point corresponding to each pixel in the 2.5D image. These pointclouds are relatively sparse compared to the pointclouds from the Stanford and Matterport datasets. As seen in Figure 21(b), certain viewpoints make it hard even for humans to recognize objects in the scene due to the limited amount of information available as well as inaccurate depth readings for many pixels. The viewpoint that makes object recognition easiest is the original view from which the RGB-D frame is taken, yet the SSNN pipeline performs reasonably well despite not being optimized for this task as seen in Table 5. We used 1,000 images for testing and 9,000 images for training. For the probing layer, we used parameters $l = 8.5, s = 0.27, d = 0.80, r = 32$.

Three objects from the SUNRGB-D dataset are analyzed in Figures 22 through 24. In Figure 22 Example 1 for the sofa object, the proposed box has an IoU of greater than 0.25 with the label and is considered a true positive. This is an example of successful detection. In Figure 22 Example 2, two boxes are suppressed from non-maximum suppression. However, the final bounding box has an IoU of less than 0.25 and is considered a false positive. The other two boxes had an IoU of less than 0.25 with the label as well. This is an example of unsuccessful detection. In Figure 23 Example 1, the IoU of the proposals with the label is greater than 0.25 and is considered a true positive. This is a successful detection. In Figure 23 Example 2 the IoU overlap is less than 0.25 and is considered a false positive. This is an unsuccessful detection. In Figure 24 Example 1 , the IoU of the proposal with the label is greater than 0.25 and is considered a true positive. This is a successful detection. In Figure 24 Example 2, there is both a successful detection and a missed detection.

We also take advantage of the availability of 2D RGB information to further improve our prediction process. First, we run a 2D object detection pipeline on RGB imagery such as Single Shot Multibox Detector [9]. We then project our 3D object detection proposals from our pipeline and find which proposals have an IoU greater than 0.5 with any of the proposals from the 2D pipeline. For the intersection proposals, we increase the associated confidence value corresponding to the confidence of the 2D proposal. This effectively changes the mAP score by rearranging proposal when sorted by



(a) (b)

Fig. 21: Example bed detection from the SUNRGB-D dataset. (a) the view similar to the 2D image. (b) top down view of the same scene and detection.

confidence value. Some examples of this approach can be seen in Figure 25. The 2D prediction is in red, the 3D predictions with overlap less than 0.75 IoU in blue and 3D predictions with overlap greater than 0.75 IoU in green. This approach worked well for the validation set which was subsampled from the training set but performed poorly on the test set seen in Table 5 for several reasons. First, the issue of dataset bias was compounded by both 2D and 3D pipelines, i.e. the effects of overfitting were magnified by combining two sets of predictions. Second, while true positives from the 2D pipeline help the overall mAP score, we were unable to come up with a method to negate the adverse affects of false positives and false negatives on our results. Lastly, there was no trivial solution for combining 3D boxes with the frustum generated from a 2D proposal, and so objects not aligned optimally with the frustum had unnecessarily large bounding boxes which gave incorrect results when combined with 3D bounding box proposals. We experimented with several other overlap strategies, without much success. This approach would work well if two pipelines with similar outputs were combined, i.e. if they both generated 3D axis-aligned bounding boxes. Instead, combining 2D and 3D pipelines proved to cause more problems than it solved and is reflected in the overall mAP score seen in Table 5.

4.4 Design Analysis

SSNN has many tunable hyperparameters, so we ran several visualizations and controlled experiments to better understand how different parameters and characteristics of datasets affect performance. We examined parameters that affected our overall mAP by greater than 3%. Upon running our pipeline with different random seeds, we found that mAP scores varying by $\pm 3\%$ can be

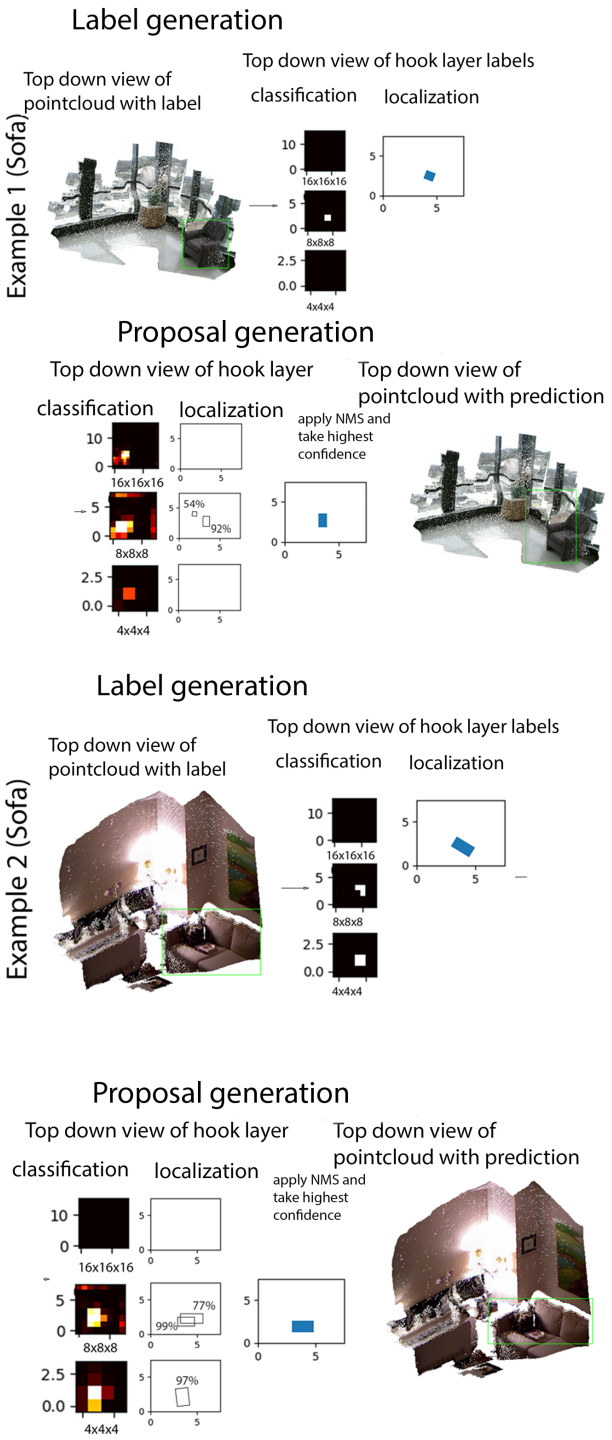


Fig. 22: An illustration of the label generation and prediction generation process for a sofa in the SUNRGB-D dataset. In Example 1, the IoU of the proposals with the label is greater than 0.25 and is considered a true positive. This is a successful detection. In Example 2 the IoU overlap is less than 0.25 and is considered a false positive. This is an unsuccessful detection.

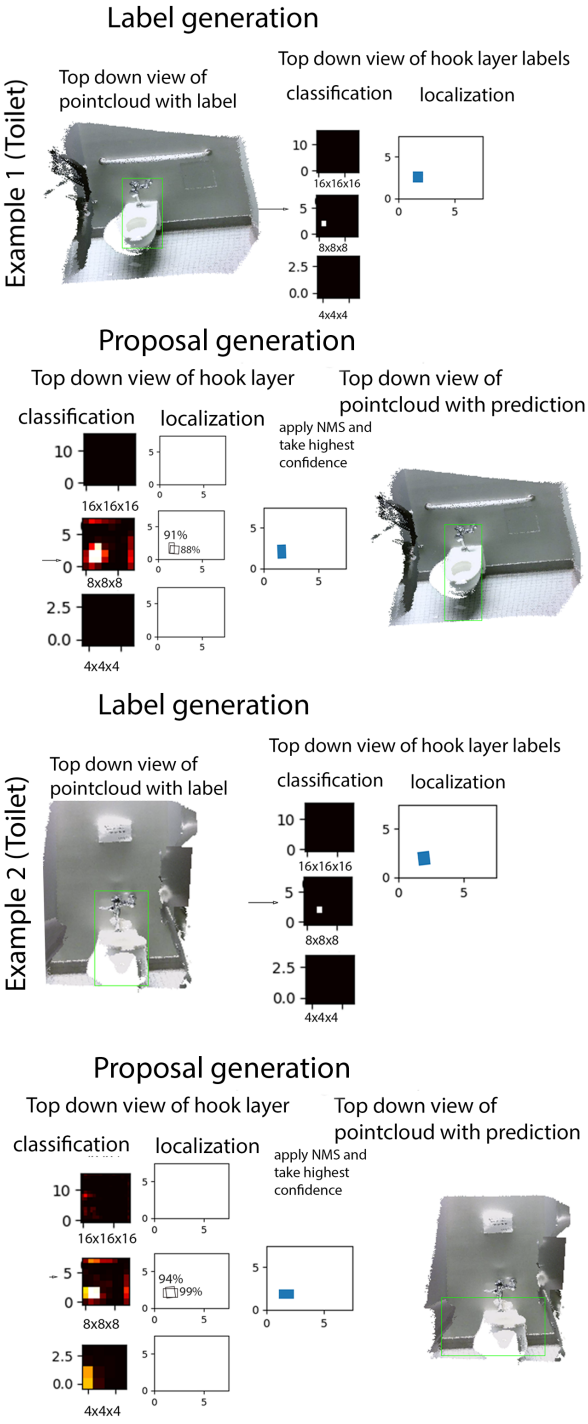


Fig. 23: An illustration of the label generation and prediction generation process for a toilet in the SUNRGB-D dataset. In Example 1, the IoU of the proposals with the label is greater than 0.25 and is considered a true positive. This is a successful detection. In Example 2 the IoU overlap is less than 0.25 and is considered a false positive. This is an unsuccessful detection.

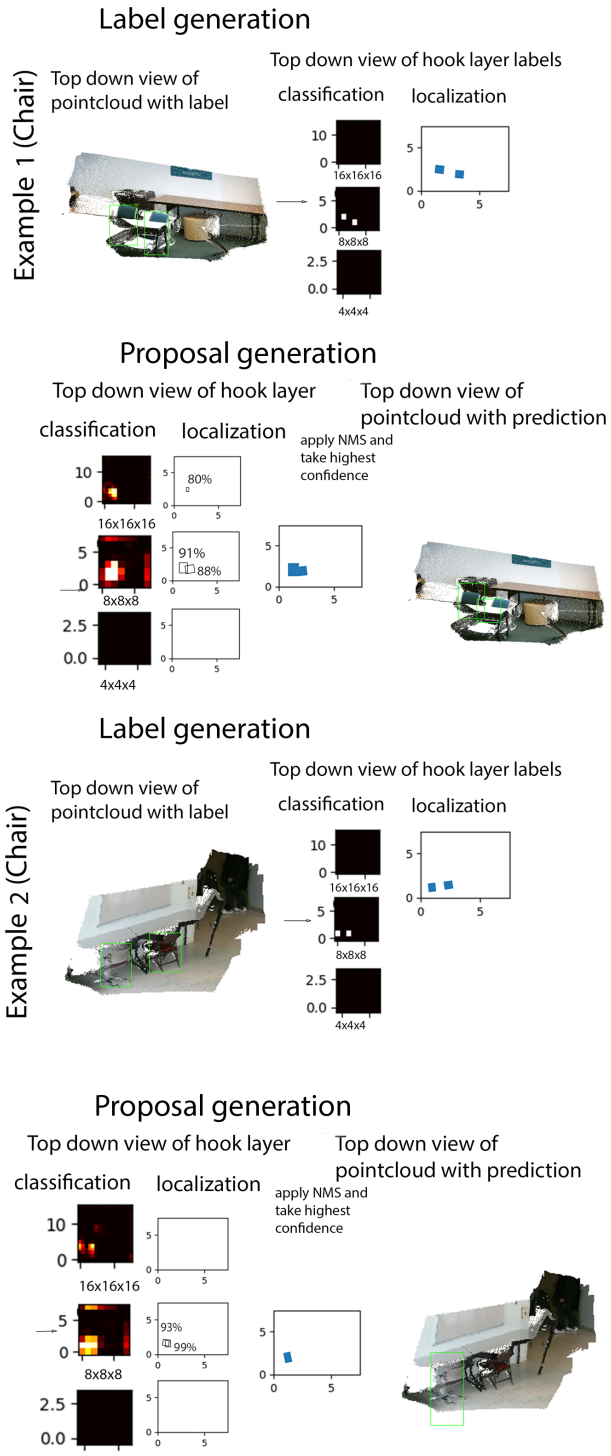
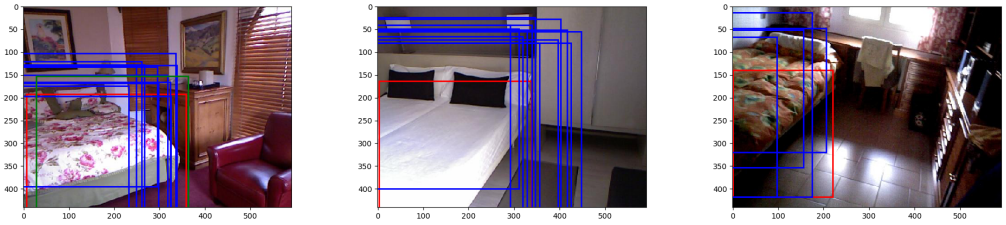


Fig. 24: An illustration of the label generation and prediction generation process for a chair in the SUNRGB-D dataset. In Example 1, the IoU of the proposal with the label is greater than 0.25 and is considered a true positive. This is a successful detection. In Example 2, there is both a successful detection and a missed detection.



(a) (b) (c)

Fig. 25: Example 2D and 3D prediction for bed. The 2D prediction is in red, the 3D predictions with overlap less than 0.75 IoU in blue and 3D predictions with overlap greater than 0.75 IoU in green. (a) An example where the combined approach works well. (b) An example where the labels for 2D do not include the bed frame while 3D labels do, so the predicted boxes do not overlap well. (c) An example where the labels for 2D are not good since there are many high confidence predictions, so the output confidence for these predictions are relatively low even though some bound the bed quite well.

attributed to noise. Unlike other state-of-the-art work in 3D detection, our model is much more interpretable due to our intuitive approach to sampling point clouds. We are able to visualize intermediates to see information flow through our system. Throughout this section, we use the network structure detailed in Fig. 1 on the Matterport3D dataset.

Hyperparameter tuning and loss We found that our model is sensitive to loss formulation. Usually around 80% of the entries in our voxel grid are empty, so the model predicts many false negatives. To circumvent this, we normalize loss by the number of positive examples per label, which allows our model to become more robust to imbalanced ratio between positive and negative training examples. Certain hyperparameters, such as number of filters per layer of the 3D CNN, did not have much effect on test accuracy as long as training time was chosen via maximum validation mAP, rather than minimum validation loss (L_{obj}).

Hyperparameter tuning for the SSNN pipeline is important for improving accuracy. We found that even small changes to certain hyperparameters greatly affected our results depending on the data set and the object category. There are many tunable parameters in our pipeline; however, the parameters for the probing layer affects our performance the most. For example, the number of probes per kernel is important because it directly correlates to the resolution in the very first processing step in our pipeline. If the probes are too far apart, then the model cannot represent small objects, and if they are too close together then the speed of the pipeline is slowed down and memory issues arise. In addition, if too many probes are initialized, training takes longer to converge and in some cases never converges. This is due to the exponential increase in parameters per probe. We found that initializing the probes randomly within the kernel greatly reduces the number of total

probes. On the other hand, initializing probes randomly not only allows for a decreased number of probes, but also causes the pipeline to train in a more stable manner.

The next hyperparameter to consider is the number of kernels. Here, we ran into similar issues as the number of probes per kernel. If we initialize too many kernels, it becomes harder to train the model; conversely, if we initialize too few kernels, not all the information from a scene can be extracted properly. Our approach to decrease the number of probes per kernel is to associate multiple dot product layers per probe. Associated with each kernel is a set of trainable weights; if multiple dot product layers per kernel are initialized, we can reduce redundant computation. For example, after convolving one kernel across an entire scene, if two dot product layers are associated with this kernel, one dot product layer looks for straight edges and the other looks for corners. This step also reduces the total number of parameters within our model and results in more efficient training with less overfitting. This is visualized in Figure 26 where different dot product weights have emphasis on different probes for the same kernel, depicted by darker colors. On the other hand, lighter dots have smaller weights and are less significant. Further research might explore pruning extraneous probes for test time to offset this computational and memory cost.

Point cloud density and data augmentation For the Matterport3D dataset, input is in mesh format, and point clouds are created by sampling the mesh. Higher density point clouds improve test accuracy slightly, but increase train and test time significantly due to memory bottlenecks. Adding data augmentation such as point jittering, rotations, and translations also improved test accuracy. We also resampled more times for object categories with fewer objects so that the number of occurrences for each category were similar. This increased our overall performance due to label imbalances.

Volume Prediction One of the challenges in moving from 2D to 3D predictions is estimating volume accurately. Compared to 2D, in 3D, each prediction contains two extra dimensions: the z-axis as well as rotation. This makes achieving the same IoU benchmarks as 2D extremely difficult. This is the reason 0.25 IoU is a more common benchmark for 3D. Nonetheless, 0.25 IoU is a challenging threshold to meet especially for objects with small dimensions such as the board object from the Stanford dataset. Figure 27 shows the relationship between predicted and actual volumes of bounding boxes. Certain categories such as the bed object are easier and have less variance compared to others such as the dresser object. The mAP difference for the Matterport3D dataset for these two categories is 54.9%. Harder categories to bound require a larger λ parameter in Equation (6).

Input dimension analysis Certain objects, such as chair have volumetric distributions with high variance relative to the rest of the dataset. In order to increase mAP for chair specifically, we

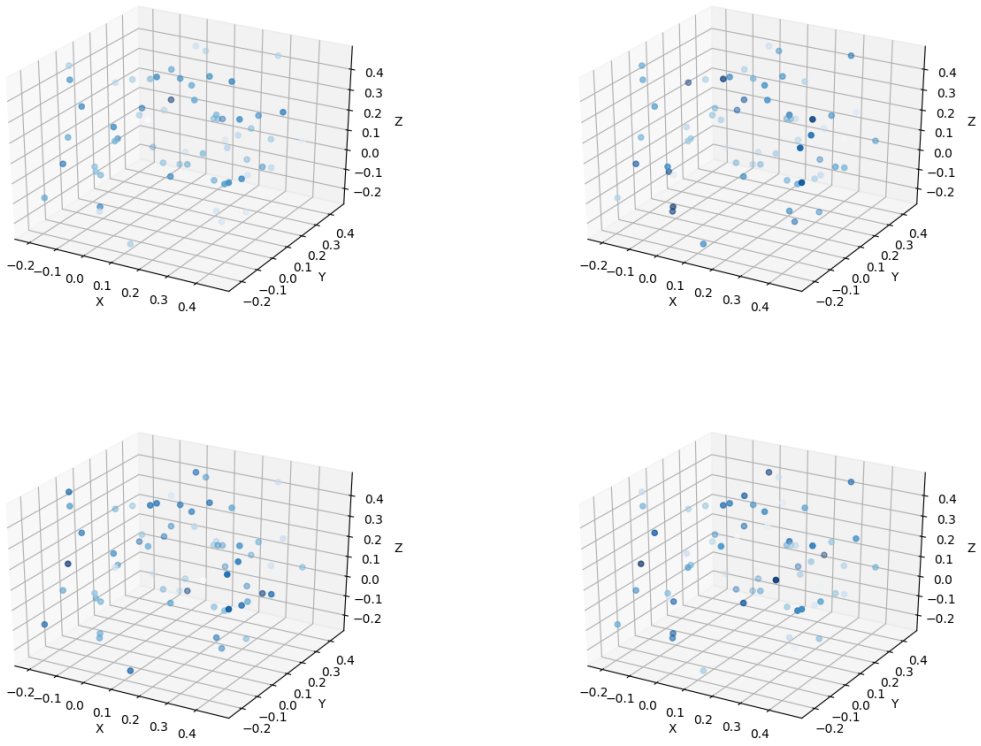


Fig. 26: Visualization of a single kernel with different dot product weights. Different dot product weights have emphasis on different probes for the same kernel, depicted by darker colors. On the other hand, lighter dots have smaller weights and are less significant..

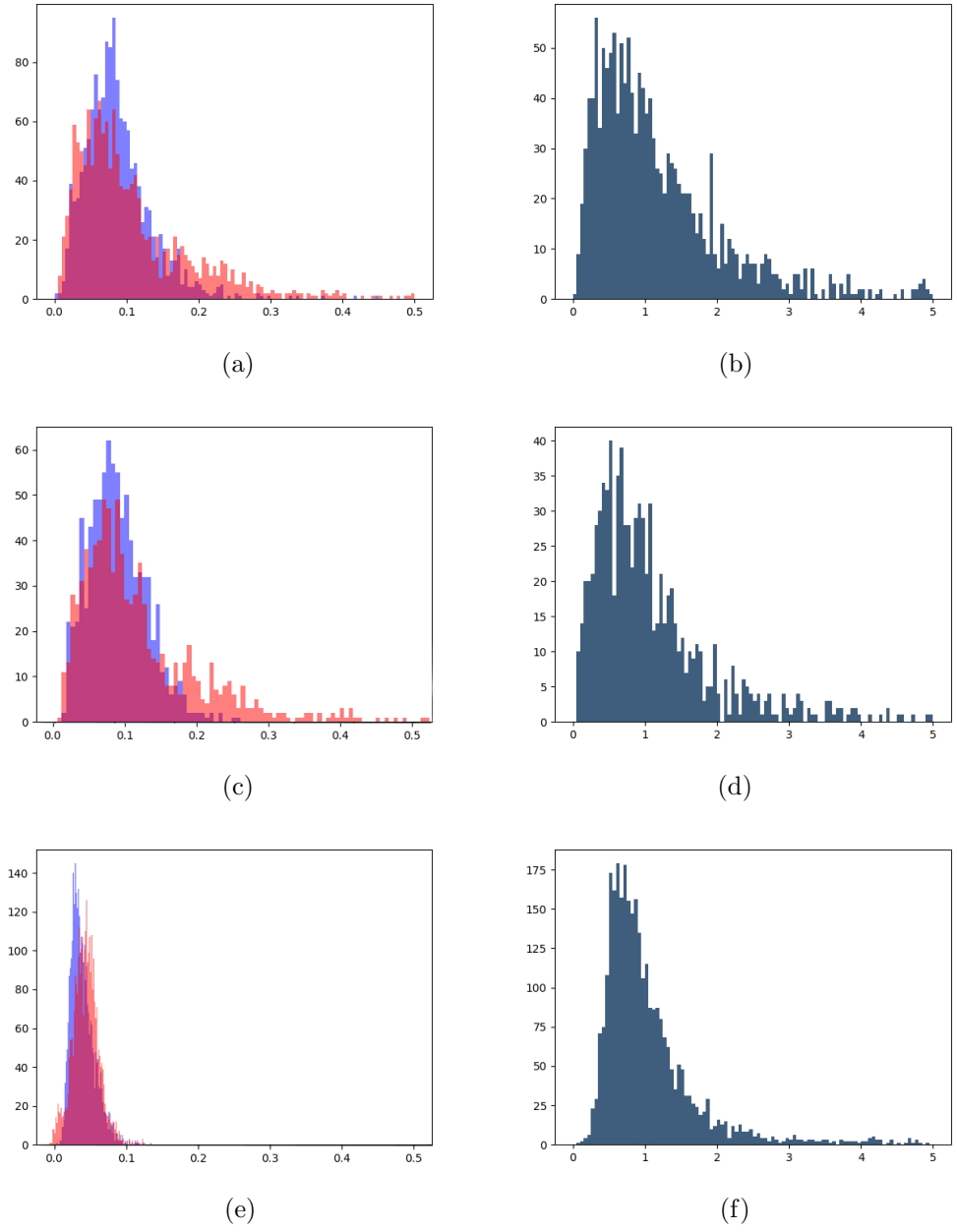


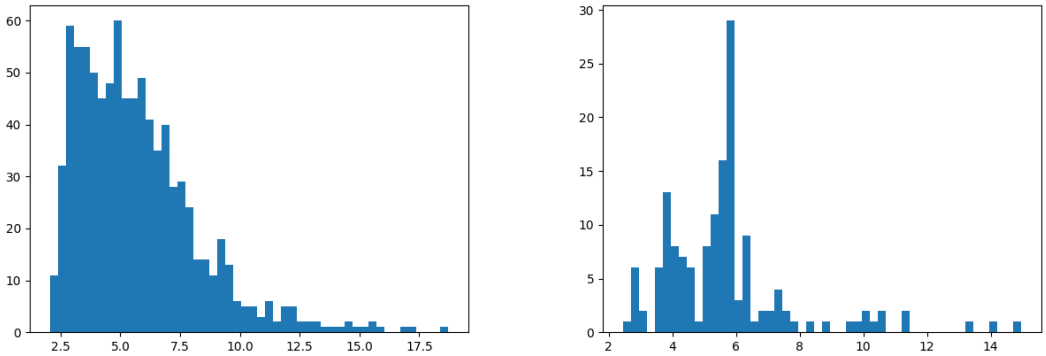
Fig. 27: Relative volumes of categories for the SUNRGB-D dataset. (a), (c), (e) Volumes of predictions (red) and volumes of ground truth (blue) for all categories, table category, and chair category, respectively. (b), (d), (f) Ratio of predicted volume to ground truth volume for all categories, table category, and chair category, respectively.

can increase λ in Equation 6. However, if we increase λ, it will affect all objects. Thus, increased performance for chair causes decreased performance for other objects.

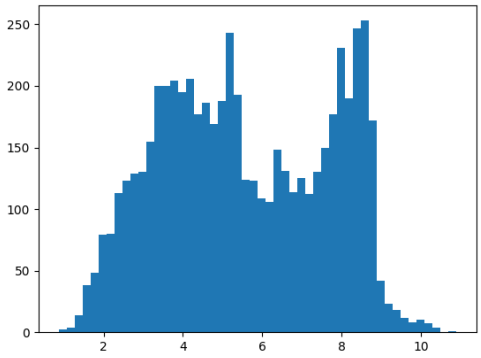
Figure 28 shows the histogram of maximum dimensions of scenes for all three datasets. The Matterport dataset has the least variance, while SUNRGB-D has many scenes with large maximum dimensions. The Stanford datasets has several outliers, but most scenes are normal rooms. For the Matterport dataset, 9.5% of scenes were above 7.5m. For the stanford dataset, 15.2% of scenes were above 7.5m. For the SUNRGB-D dataset, 26.8% of scenes were above 7.5m. SUNRGB-D contains a large number of scenes above 7.5m due to outlier depth readings, not because the scenes themselves are large. Thus, the points that are truncated are often not part of the scene. We also considered the input dimension of the entire room. Since we use a maximum input size of 7.5m, small rooms leave most the input space empty while rooms larger than 7.5m are truncated. One approach is to normalize all scenes to a constant dimension such that the input resolution is fixed to 32^3. The results for this normalization experiment can be seen in Table 6. The mAP for the Matterport dataset improved the most from input normalization, perhaps due to the even distribution of the maximum dimensions shown in Figure 28(a). Meanwhile, the mAP difference from the other datasets are likely to be within noise. We also analyzed the effect of normalization factor on each category in the SUNRGB-D dataset seen in Figure 29. The x-axis is the normalization scale, i.e. the normalized dimensions divided by original dimensions. The y-axis is the relative performance factor, i.e. the normalized mAP result divided by the unnormalized mAP result. Normalization can decrease performance for extremely large scenes, i.e. those with small normalization scale. Normalizing large scenes can shrink normal objects past the probing layer's capabilities, indicated by the low mAP performance in bookshelf, drawer, and table when the scene experiences a normalization factor of around 0.125. However, the percentage of occurrences for these extremely large scenes is a very small: less than one percent for all datasets. Figure 29 seems to imply that the more an object is magnified, the higher mAP it has, but this trend could also be within noise.

Even though normalization of input does not preserve metric scale it does facilitate feeding a high range of input dimensions into the network as this prevents truncation and excess zero padding of input. Meanwhile, if input is not normalized, a voxel resolution must be chosen such that a high percentage of scenes are not truncated while still having a fine enough resolution to detect small objects in the scene.

Non-maximum suppression (NMS) Non-maximum suppression is the last step in the prediction pipeline. For non-maximum suppression, we find each pair of overlapping bounding boxes and delete the bounding box with lower confidence. Since it is not part of training, understanding



(a) (b)



(c)

Fig. 28: (a) the distribution of maximum dimensions of the Matterport3D dataset, (b) the distribution of maximum dimension of the Stanford 3D Indoor Spaces dataset, and (c) the distribution of maximum dimension of the SUNRGB-D dataset.

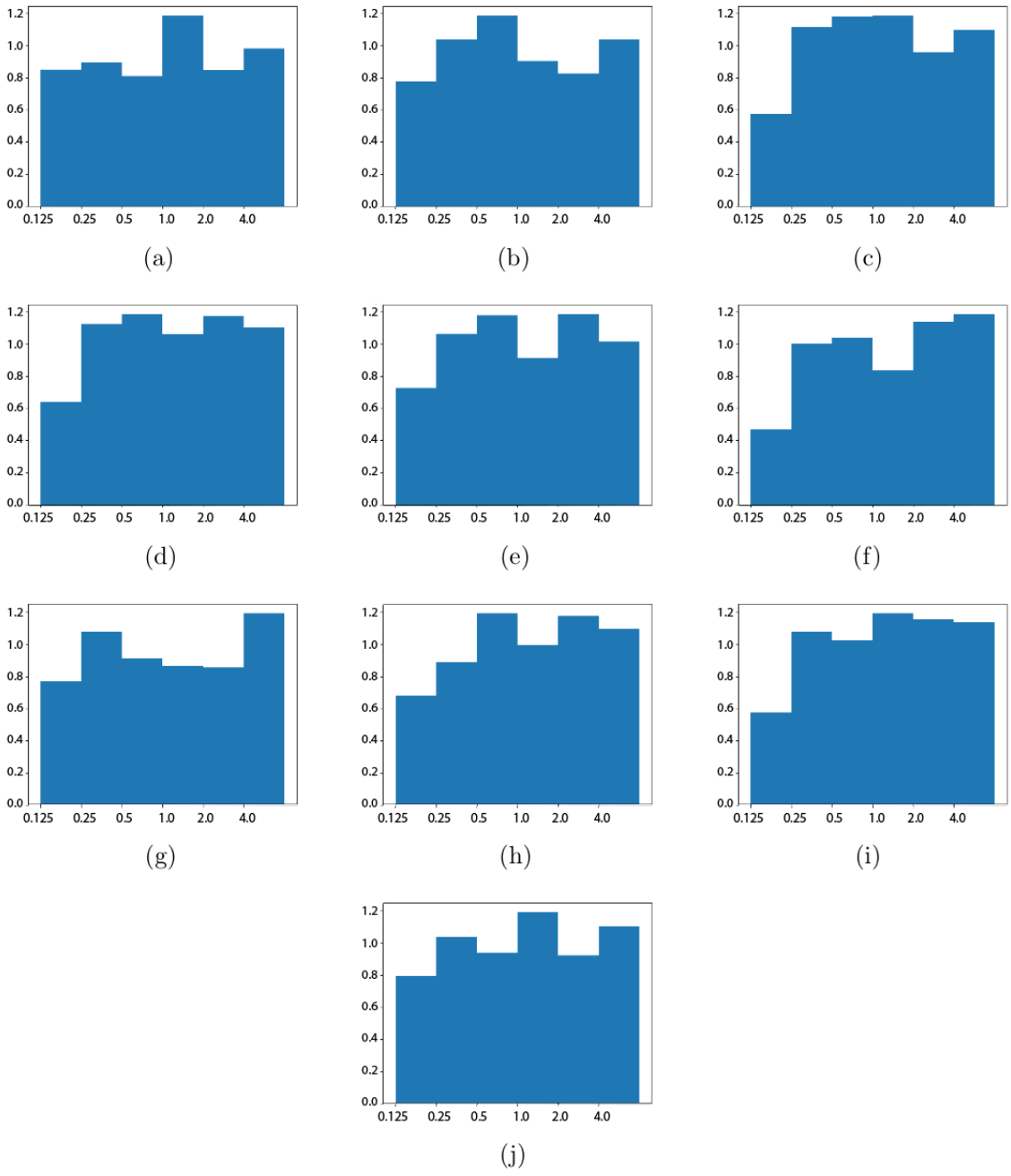


Fig. 29: The effect of normalization factor on each category in SUNRGB-D. (a) bathtub, (b) bed, (c) bookshelf, (d) chair, (e) desk, (f) drawer, (g) nightstand, (h) sofa, (i) table, (j) toilet. The x-axis is the normalization scale, i.e. the result dimensions divided by original dimensions. The y-axis is the relative performance factor, i.e. the normalized mAP result divided by the unnormalized mAP result. The graphs seems to imply that the more an object is magnified, the higher mAP it has, but this trend could also be within noise.

	bathtub	bed	bookshelf	chair	desk	dresser	nightstand	sofa	table	toilet	board	mAP
S3DIS Non-normalized	-	-	-	89.8	-	-	-	39.2	79.8	-	42.9	62.9
S3DIS Normalized	-	-	-	91.4	-	-	-	41.1	77.8	-	41.2	62.9
Matterport3D Non-normalized	65.9	92.1	43.4	66.0	31.0	37.2	72.9	45.2	39.3	51.8	-	54.48
Matterport3D Normalized	66.9	94.8	42.3	66.1	29.9	40.7	72.7	54.2	32.6	56.4	-	57.18
SUNRGB-D Non-normalized	42.5	80.7	22.9	59.7	30.7	12.0	60.5	45.7	48.7	80.1	-	48.4
SUNRGB-D Normalized	41.6	83.9	19.3	59.8	32.8	7.2	52.4	52.7	56.4	82.3	-	48.8

Table 6: Normalized input results for all three datasets.

the effects of NMS on overall performance is important. We evaluated the performance of our model accuracy without NMS, counting duplicate detections as true positives. On all datasets, the mAP increased only slightly by between 1 and 3 percentage points. This means that the boxes that are suppressed often have low confidence values regardless. This also means that NMS is not likely to be degrading the performance of our model.

Comparison to FPNN pipeline The FPNN pipeline consists of multiple steps as seen in Figure 30. SSNN adapts operations from object recognition to object detection, meaning instead of using input point clouds of single objects, the SSNN pipeline uses input point clouds of entire scenes with many possible objects. In FPNN, the authors approach point cloud processing in a different way compared to existing work [2] [18]: they do not require voxelization as an initial pre-processing step and instead use probes to represent pointclouds. Because voxelizing large 3D scenes would be impractical both computationally and for memory efficiency, we opted to use the ideas from FPNN for pointclouds.

The FPNN pipeline starts with their trainable probing layer followed by a Gaussian layer which applies a Gaussian transform to values to minimize the effect of outliers. This is then fed into a dot product layer, similar to the SSNN pipeline. While FPNN and SSNN both share similar operations of probing and dot product layers, the pipelines are very different.

Starting with the probing layer, FPNN runs its probe operation just once, which is referred as the sensor layer in Figure 30. Meanwhile, we slide probes across the entire scene, running the probe operation thousands of times per scene. This means the output of the SSNN probing layer has extra dimensions for each spatial dimension: x, y, and z. It is important to note that FPNN probing layer has trainable locations while the SSNN pipeline does not. The dot product layer acts the same for both pipelines, but applied multiple times across each extra spatial dimension for the SSNN pipeline. After the dot product layer, both pipelines feed into a neural network. Since FPNN does not convolve its probe layer, it is able to feed directly into a regular fully connected neural network. Meanwhile,

since SSNN has spatial dimensions from convolution, it feeds into a 3D convolutional network as described above.

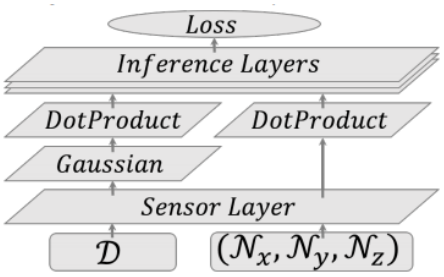


Fig. 30: Diagram of the FPNN pipeline to contrast with the SSNN pipeline in Figure 1. The input to this network is D and $\boldsymbol{N}$ which are the pointcloud and the normal vectors, respectively.

5 Conclusions and Future Work

We introduce the Sparse Sampling Neural Network, a novel deep neural network for object detection in point clouds. Our network converts point clouds from a continuous input space into discrete voxels without information loss common in other approaches. This allows us to quickly and efficiently detect and classify objects within large point clouds by inferring bounding boxes. We have open-sourced our code at https://github.com/ryangoy/SSNN.

Future work includes extending our work in 3D object detection to more types of scenes. Currently, our ideal input dataset will have similar dimensions with evenly distributed points. Creating a more robust network that can learn from fewer points per pointcloud will be necessary for processing outdoor scenes and will improve performance on 2.5D datasets as well. Applying one-shot learning techniques to SSNN or devising a strategy to identify several hundred objects reliably may be another useful investigation. A detection pipeline with an ability to recognize a larger number of categories would be useful in applications such as robotics. Finally, 3D detection one-shot learning pipeline may have powerful implications in real-time detection.

References

1. Hosang, J., Benenson, R., Dollár, P., Schiele, B.: What makes for effective detection proposals? IEEE transactions on pattern analysis and machine intelligence **38**(4) (2016) 814–830
2. Wang, P.S., Liu, Y., Guo, Y.X., Sun, C.Y., Tong, X.: O-cnn: Octree-based convolutional neural networks for 3d shape analysis. ACM Transactions on Graphics (TOG) **36**(4) (2017) 72
3. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2015) 1912–1920
4. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. Proc. Computer Vision and Pattern Recognition (CVPR), IEEE **1**(2) (2017) 4
5. Lahoud, J., Ghanem, B.: 2d-driven 3d object detection in rgb-d images. In: 2017 IEEE International Conference on Computer Vision (ICCV), IEEE (2017) 4632–4640
6. Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., Zhang, Y.: Matterport3d: Learning from rgb-d data in indoor environments. International Conference on 3D Vision (3DV) (2017)
7. Armeni, I., Sax, S., Zamir, A.R., Savarese, S.: Joint 2d-3d-semantic data for indoor scene understanding. arXiv preprint arXiv:1702.01105 (2017)
8. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. (2015) 91–99
9. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision, Springer (2016) 21–37
10. Kaiming He, Xiangyu Zhang, S.R.J.S.: Deep residual learning for image recognition. arXiv:1512.03385 (2015)
11. Song, S., Xiao, J.: Deep sliding shapes for amodal 3d object detection in rgb-d images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 808–816
12. Ren, Z., Sudderth, E.B.: Three-dimensional object detection and layout prediction using clouds of oriented gradients. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 1525–1533
13. Song, S., Xiao, J.: Sliding shapes for 3d object detection in depth images. In: European conference on computer vision, Springer (2014) 634–651
14. Engelcke, M., Rao, D., Wang, D.Z., Tong, C.H., Posner, I.: Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In: Robotics and Automation (ICRA), 2017 IEEE International Conference on, IEEE (2017) 1355–1361
15. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from rgb-d data. arXiv preprint arXiv:1711.08488 (2017)

16. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3d object detection network for autonomous driving. In: IEEE CVPR. Volume 1. (2017) 3
17. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural Information Processing Systems. (2017) 5105–5114
18. Yin Zhou, O.T.: Voxelnet: End-to-end learning for point cloud based 3d object detection. arXiv:1711.06396 (2017)
19. Yangyan Li, Sren Pirk, H.S.C.R.Q.L.J.G.: Fpnn: Field probing neural networks for 3d data. arXiv:1605.06240v3 (2016)
20. Geoffrey E. Hinton, Nitish Srivastava, A.K.I.S.R.R.S.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580 (2012)
21. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. (2010) 249–256
22. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)