Master's Thesis

# 3D Object Detection using Tangent Convolutions

Jan Bechtold

December 2018

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science

**Writing period**

19. 06. 2018 – 19. 12. 2018

**Examiner**

Prof. Dr. Thomas Brox

Dr. Joschka Boedecker

**Adviser**

Maxim Tatarchenko

# Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

| | |
|---|---|
| Place, Date | Signature |

# Abstract

Object detection is one of the classic computer vision tasks. It requires localizing individual objects in visual data and determining their type. State of the art methods for 2D object detection use convolutional neural networks (CNNs), which can be efficiently applied to a dense grid of information, like images. In contrast to images, applying CNNs to 3D data is not straightforward because 3D grid convolution is inefficient for large scenes.

In this work we build an object detection framework for 3D point clouds. It is based on Tangent Convolutions and follows the Faster R-CNN architecture. Tangent Convolutions efficiently implement a CNN for analyzing 3D point clouds, by convolving local approximations of a 3D structure with a 2D kernel.

Our object detector is efficient and scales to large scenes with hundreds of thousands of points, due to the use of tangent convolutions. We evaluate our method on two indoor datasets: ScanNet and Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS). The proposed method is generic and can be applied to both indoor datasets without any changes. Experimental results show that our object detector is able to detect objects of various sizes and shapes, ranging from a small sink to a large couch. A comparison of our ScanNet benchmark score to other groups on the leader board shows that our method is competitive with other approaches.

# Zusammenfassung

Objekterkennung ist eine zentrale Aufgabe in der Bildverarbeitung. Dabei wird bestimmt, wo sich ein oder mehrere Objekte befinden und um welchen Objekt-Typ es sich handelt. Neueste Methoden der Objekterkennung in Bildern nutzen Convolutional Neural Networks (CNNs). Für 3D Daten, wie zum Beispiel Punktwolken, können dieselben Methoden nicht ohne weiteres verwendet werden, da sich die Effizienz verringert.

In dieser Arbeit präsentieren wir eine Methode zur Objekterkennung in 3D Punktwolken, die auf einem CNN basiert. Die Architektur unseres CNN orientiert sich dabei an der Architektur von Faster R-CNN, einer bewährten Methode zur Objekterkennung in 2D. Unsere Methode kann 3D Punktwolken effizient analysieren, weil Tangent Convolutions als Baustein für das CNN verwendet werden. In 3D Punktwolken sind die Oberflächen der gescannten Objekte durch Punkte abgebildet. Tangent Convolutions approximieren die Oberflächen lokal durch eine Ebene, sodass 2D Faltungen darauf angewendet werden können.

Unsere Objekterkennung ist effizient und kann daher für große Punktwolken mit über hunderttausend Punkten angewendet werden. Wir evaluieren unsere Methode auf zwei Datensätzen, die Punktwolken von Innenräumen enthalten: ScanNet und S3DIS. Unsere Methode ist allgemein und funktioniert auf beiden Datensätzen ohne spezielle Anpassungen. Wir zeigen anhand von qualitativen und quantitativen Auswertungen, dass unsere Methode Objekte verschiedener Größen detektiert. Ein Vergleich zwischen unseren Ergebnissen und den Ergebnissen anderer Methoden auf dem ScanNet Benchmark-Test zeigt, dass unsere Methode konkurrenzfähig ist.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

For computers, understanding a scene that is displayed in an image is difficult task. *Segmenting* an image *semantically* requires knowing the object type for each pixel. In contrast, *object detection* coarsely locates objects with boxes, but it can distinguish between instances of an object type. *Instance segmentation* combines the precision of semantic segmentation with the instance notion of object detection. For every pixel the object type and instance are predicted. State-of-the-art algorithms for the three tasks, mentioned in the above, build upon convolutional neural networks (CNNs).

LiDAR sensors provide high-resolution 3D data represented as point clouds. While object detection methods for images are very advanced in detection accuracy, speed, and memory consumption, powerful methods for processing the 3D data are rare. A straightforward extension of CNNs to process point clouds is to use 3D grid convolutions on an occupancy grid map. 3D grid maps and convolutions, however, are inefficient for large-scale 3D scenes. Recent works have tackled processing 3D data with CNNs by using octree structures [1], graph based approaches [2], and parametric convolutions for non-grid structured data [3]. Among them is the tangent convolution method [4]. The core idea of the method is to locally approximate the surface geometry with a plane and perform a regular 2D convolution on this approximation. This is efficient and allows for the processing of city-scale scenes.

In this work we design an object detector for 3D data which uses tangent convolutions as the main building block. The architecture of our object detector is based on the general structure of Faster R-CNN where object detection is split into two subtasks. First, our network identifies regions of interest (ROI) or volumes of interest in the 3D case. Second, our network predicts one object class label for each ROI. Tangent convolutions are used for efficient feature extraction in both tasks. Tangent convolutions perform local processing of the point cloud. This works well for semantic segmentation where a class label is predicted per point. In contrast to per point predictions, our object detector uses the features from tangent convolutions to make a prediction for a region of interest. We evaluate our method on two indoor datasets: ScanNet [5] and Stanford [6]. The ScanNet team provides a benchmark for instance segmentation, where we compare our method with other state of the art methods. We convert detection boxes into instance labels per point by assigning each point within the box the corresponding label. Our method is competitive with other methods on the leader board.

We designed an object detector that follows the design of the successful 2D region based object detection approach and uses efficient processing operations for 3D. It is applicable to large indoor scenes. Furthermore, we are able to detect diverse objects, e.g., a 30cm wide picture on the wall or $2 \times 2$meter bed on the floor, at all positions in the scene.

# 2 Related Work

## 2.1 2D Object Detection

Object detection in computer vision started with detecting whether there is an object in the image or not. For this task strong feature descriptors based on gradients were used. The Histogram of Oriented Gradients (HOG) method from Dalal et al. [7] detects people in images by accumulating the local gradients into more complex features. Dalal et al. use a support vector machine (SVM) on top of the features to predict the class label.

Strong object detection frameworks that use convolutional neural networks (CNN) have been developed for images. There are two main approaches for object detection on images, namely, region based methods and single stage detectors. Region based methods split the object detection task in two sub-tasks. The first one is to predict the area of an object in the image. In the literature this area is also referred to as *region proposal* or *region of interest (ROI)*. The area is represented as a bounding box, which locates the object and encloses all pixels that belong to the object. The second task is to predict one object class label for each ROI. Girshick et al. [8] introduced Region CNN (R-CNN) to explore the capabilities of AlexNet [9] for the object detection task. Selective search [10], which is an external algorithm, extracts regions from the input image. A CNN that is similar to AlexNet extracts features which are then used by a set of SVMs to predict the class label.

Fast R-CNN [11] and Faster R-CNN [12] improve R-CNN in terms of speed and accuracy. Fast R-CNN speeds up the method by computing the object class predictions in a single network instead of multiple SVMs and by sharing convolutions between all proposed regions. Faster R-CNN replaces the external region proposal algorithm with a dedicated CNN, that Ren et al. call region proposal network (RPN). Together the RPN and classifier network form an end-to-end trainable network. Mask R-CNN from He et al. [13] is the current state of the art network for object detection and instance segmentation on images. It extends Faster R-CNN with a binary segmentation branch, such that all pixels in a predicted box are further classified into background and object.

Single stage detection methods aim to make object detection usable for robotic systems by achieving higher frame rates than region based methods. The unified solution comes at a cost of accuracy as Huang et al. [14] describe in their comparison. The single stage detection method YOLO from Redmon et al. [15] applies a fixed coarse resolution grid to the input image. For each cell in this grid they predict one object class label and two object bounding boxes. All predictions of YOLO can be computed in one forward pass. Due to the fixed, coarse resolution grid, the detection accuracy of YOLO is limited to a minimum distance between objects. Liu et al. [16] tackle this problem by applying the grid to multiple feature maps in the network. Since the receptive field increases for deeper layers, the object predictions are more accurate for objects of different scale and size. Furthermore Liu et al. use an initial set of boxes to guide the network in the box prediction.

Feature extraction for images hugely benefits from commonly used architectures, like VGG [17] and ResNet [18], that are trained on big datasets like ImageNet.

## 2.2 3D Deep Learning

More recently, convolutional neural networks are applied to 3D data. There are different approaches to represent and process 3D point clouds. The intuitive extension of pixel grids to 3D are voxel grids. For object classification Dai et al. [5], Maturana et al. [19] and Wu et al. [20] have applied CNNs with 3D convolutions on voxel grids. Methods based on voxel grids however suffer from the cubic complexity of this data representation, which limits their resolution. Hence other methods exploit the sparsity of the data and define a new convolution operation for tree structures such as octrees or kd-trees [1, 21, 22]. Just as the tree based methods, Qi et al. [23] and Tatarchenko et al. [4] exploit the sparsity of the data. Qi et al. do so by per point processing and global aggregation of point features through max pooling operations. Tatarchenko et al. exploit the fact that the 3D point clouds are surface representations of the scenes and achieve dense convolution operations by locally approximating the surface with a plane.

## 2.3 3D Object Detection and Instance Segmentation

More works build up on the basic HOG method to use it in the 3D domain. Gupta et al. [24] use HOG descriptors for image segmentation and object detection in

RGB-D images. Lang et al. [25] use an extended HOG descriptor to perform object classification on data from LiDAR sensors. They solve the classification task with a conditional random field and use a HOG related descriptor that is based on distances between points and planes instead of gradients.

The methods that have been proposed to solve 3D object detection and instance segmentation with deep learning can coarsely be divided into two categories. Into the first category fall methods that use RGB-D data. Methods from the second category work exclusively on point clouds. As stated before, image based object detection methods strongly profit from established pre-trained models, such as VGG. There is no comparable pre-trained model for 3D-only methods.

Song et al. implement the region proposal network of Faster R-CNN for the 3D case, using 3D convolutions on a voxel grid [26]. For the classification step, Song et al. extract features in parallel from voxels and image patches. They obtain the image patches by projecting the 3D box into the RGB image of the scene and cropping it. Both features from 2D and 3D are concatenated to predict a bounding box refinement and the object class label.

The key idea of Qi et al. [27] is to avoid processing large 3D point clouds by performing object detection on images in 2D. Subsequently they use these detections as region proposals in 3D. By projecting the bounding boxes from the image into the point cloud they obtain frustums. For the frustums they perform a binary segmentation in order to distinguish between object and background. Predicting objects in the image alleviates the complexity problem of large 3D scenes.

The voxel grid based processing of Song et al. is not efficient for large scenes and is limited in resolution. Furthermore both methods rely on RGB data and the transformation matrix to map between 2D and 3D. Hence these methods are not applicable to data sets with unknown sensor poses.

The following methods solve object detection without using RGB-D. PIXOR is a single stage approach to object detection, similar to SSD. In order to avoid the sparse 3D point cloud space, Yang et al. [28] transform the points into birds eye view (BEV). They discretize the point cloud in BEV in order to process the data with a single stage detection architecture for 2D data.

PIXOR is presented for the KITTY data set that contains cars pedestrians and bicycles. In a top down view the appearance of these classes can be distinguished well. For data sets with a higher number of classes, such as ScanNet and S3DIS, it is not sufficient to consider only the top down view, since most of the information comes from different view angles. Pictures and doors, and toilets and chairs are easier to tell apart in the x-z or y-z plane.

All methods mentioned before follow a *coarse to fine* approach. Wang et al. [29] chose a *fine to coarse* approach to solve instance segmentation. Their key idea is to form clusters of similar points. Assigning a label to all points of one cluster creates an instance segmentation for the point cloud. Additionally to the clustering, the network outputs a semantic segmentation, which is used to assign a semantic class label to all instances. This per point clustering makes the approach independent of any grid structure. Wang et al. use the PointNet [23] and PointNet++ [30] architecture to compute the similarity between all points. Since the similarity matrix is quadratic w.r.t. the number of points, the efficiency of this method decreases for larger scenes. Furthermore PointNet performs well on structured point clouds, such as ShapeNet and S3DIS, where all rooms follow a similar structure. For unstructured point sets, like ScanNet, where rooms vary a lot more in rotation and appearance, PointNet loses most of its predictive power [31].
Our proposed 3D object detector combines the region based approach of Faster R-CNN with the powerful concept of tangent convolutions. It operates on unstructured point clouds and does not rely on RGB information.

# 3 Background

For our object detector in 3D, we need a suitable architecture and efficient 3D processing. Faster R-CNN is a well established and accurate object detection method. Tangent convolutions have shown efficient processing of big 3D scenes and strong feature extraction skills for semantic segmentation of point clouds. These two works are the foundation of our proposed method. We therefore present a technical description of all relevant parts in this background chapter.

## 3.1 Faster R-CNN

Faster R-CNN is part of the region based approaches for object detection. Region based approaches solve object detection in two steps. The first is to split the original image into smaller images, where each smaller image ideally contains exactly one object. Unfortunately, the smaller images are not fail-safe. They can cover objects only partially or at worst be incorrect and not contain an object at all. The second step, classification, is to assign a label to each smaller image. Both steps are implemented as CNNs.

Figure 1 shows the architecture of Faster R-CNN. It consists of the modules *Region Proposal Network* (RPN) and *Classifier*. Both modules share the convolution layers of a backbone CNN. Ren et al. [12] use the first thirteen layers of VGG as a backbone network. The RPN tells the classifier where to look in the feature maps. For these areas, which are displayed as rectangles, the classifier predicts the object class.

### 3.1.1 Region Proposal Network

The region proposal network replaces the external region extraction algorithm that was used by Fast R-CNN. Because it operates on the feature map of a backbone feature extraction network, it only has three convolution layers. The first layer in the RPN convolves the feature map with a $3 \times 3$ filter. On top of this layer operate two $1 \times 1$ filters in parallel, which compute the predictions of the network, see Figure 2.

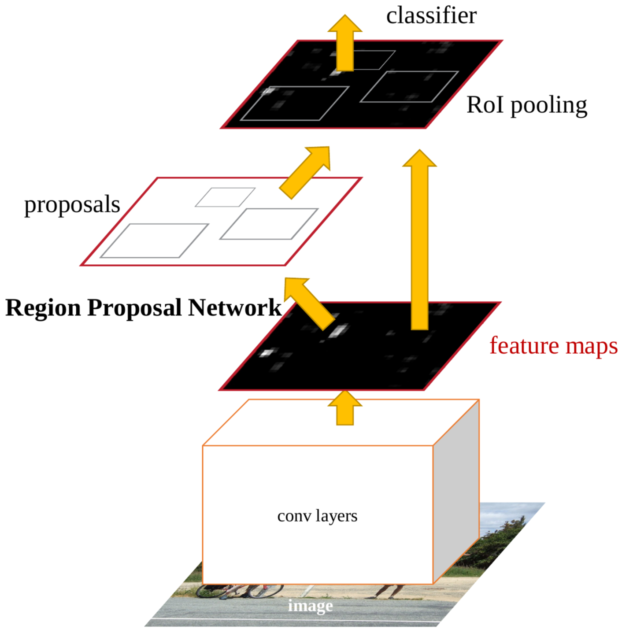


**Figure 1:** Faster R-CNN architecture overview that displays how the two modules RPN and Classifier are connected via shared convolutions. Image source [12].

The RPN solves a dense prediction task in a fully convolutional manner. The RPN predicts rectangular object bounds (4 coordinates) together with an objectness score (2 scores) for each pixel in the feature map. Objectness is a binary measure that specifies, whether a bounding box located at a certain position contains and object or not. The rectangular object bounds are encoded by their lower left and upper right point.

**Anchors** Objects in images appear in different sizes and aspect ratios and so do the boxes that surround all pixels of the object. Therefore, the RPN simultaneously predicts multiple boxes per feature map pixel. At most $k$ boxes are predicted per position, such that the output is $2k$ objectness scores and $4k$ box coordinates. Each predicted box is related to a candidate box that is positioned at every pixel in the feature map. The candidate boxes are called anchors. The four predicted values per $k$ are parameters for center point $(x,y)$, the width and the height of the anchor. Figure 2 shows the correspondence between the RPN prediction and anchors. Nine anchors are positioned at each pixel in the feature map, such that the pixel is at the center of the anchor. The anchor boxes appear in three different scales and three different aspect ratios to cover diverse objects in the image. For a feature map of

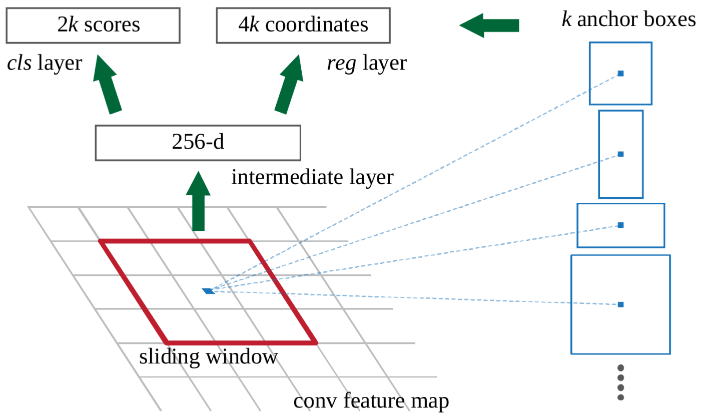


**Figure 2:** Region Proposal Network with anchors boxes. Two parallel $1 \times 1$ convolutions perform a box-regression and objectness classification for each pixel in the feature map. Image source [12].

width $W$ and height $H$ this results in $WHk$ anchors. Anchors are candidate boxes and the RPN has to predict, whether these candidates are reasonable or not.

**Loss** As described in the anchor section, the RPN simultaneously predicts $2k$ scores ($p_i$) and $4k$ refinements ($\mathbf{t}_i$), which corresponds to the number of possible objects per position. The targets that the network should learn for the objectness are computed in the following paragraph. The refinement targets $\mathbf{t}^*$ are defined in Equation (4). In order to train the region proposal network, each anchor is assigned a binary class label $p_i^*$ which relates to the objectness. Whether an anchor is considered positive or negative depends on its intersection over union (IoU) with a ground truth box. All anchors that have an IoU greater than 0.7 are considered positive as well as all anchors with the highest IoU per ground truth box. Negative anchors are those with an IoU lower than 0.3. Anchors with an IoU in the range between 0.3 and 0.7 do not contribute to the RPN training.

The objective function that is minimized in the RPN training is the multi-task loss $L(\{p_i\}, \{\mathbf{t}_i\})$. It combines the binary classification loss $L_{cls}$ used for objectness prediction and the regression loss $L_{reg}$, which is used for box refinements:

$$\begin{aligned} L(\{p_i\}, \{\mathbf{t}_i\}) = & \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) \\ & + \lambda \frac{1}{N_{reg}} \sum_i p_i^* * L_{reg}(\mathbf{t}_i, \mathbf{t}_i^*), \end{aligned} \tag{1}$$

where $p_i$ is the predicted probability that anchor $i$ contains an object and $p_i^*$ is the ground truth objectness label, 0 for negative anchors, 1 for positive anchors. The classification loss is a binary log loss $L_{cls}(p_i, p_i^*) = -log(1 - (|p_i^* - p_i|))$. It is summed for all positive and negative anchors $i$ that contribute to the loss.

Both loss terms in Equation (1) are normalized by factors $\frac{1}{N_{cls}}$ and $\frac{1}{N_{ref}}$ and balanced with a parameter $\lambda$. These are set to $N_{cls} = 256$, $N_{ref} \sim 2400$ and $\lambda = 10$, such that both loss terms contribute roughly equally to the multi-task loss. However, the RPN training is insensitive to a wide range of values for $\lambda$.

The regression loss only considers refinements for positive anchors, where $p_i^* = 1$. It is defined by Girshick [11] as the smooth $L_1$ loss over the difference between refinements:

$$L_{reg}(\mathbf{t}, \mathbf{t}^*) = \sum_{j \in x,y,w,h} \text{smooth}_{L_1}(\mathbf{t}_j - \mathbf{t}_j^*), \tag{2}$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise.} \end{cases} \tag{3}$$

The vector $\mathbf{t}_i$ contains the predicted refinement values that correspond to the four box coordinates. Similarly, the vector $\mathbf{t}_i^*$ denotes the parameterized refinement values between the positive anchor and ground truth box. These are defined as follows:

$$\begin{aligned} t_x &= (x - x_a)/w_a, & t_x^* &= (x^* - x_a)/w_a, \\ t_y &= (y - y_a)/h_a, & t_y^* &= (y^* - y_a)/h_a, \\ t_w &= \log(w/w_a), & t_w^* &= \log(w^*/w_a), \\ t_h &= \log(h/h_a), & t_h^* &= \log(h^*/h_a), \end{aligned} \tag{4}$$

where $t_x$ and $t_y$ denote the difference between the predicted box center $(x, y)$ and the anchor box center $(x_a, y_a)$. $t_w$ and $t_h$ denote the ratio between the box width and anchor width, height respectively. $t_x$ and $t_y$ are normalized by the anchor width, $t_w$ and $t_h$ are log normalized, such that the refinement values are within a range of roughly $[-1, 1]$. These refinements are computed between the ground truth and the anchor boxes in the same way, see right column.

**Training** Negative anchors, which do not sufficiently overlap with an object in the image, typically dominate the positive anchors. In order to focus the RPN training on both, positive and negative anchors, it is trained on a limited set of 256 anchors per image, with an equal split between positive and negative. Random sampling is used to select 128 anchors of each type, if there are more than 128 of either type. If there are less than 128 positive anchors, the free spots are filled with negative anchors.

The backbone architecture is initialized with a set of weights which have been trained on the ImageNet dataset. The three convolutional layers of the RPN are initialized with random weights.

The RPN is trained end-to-end on 80000 images of the PASCAL VOC dataset with a momentum of 0.9, weight decay of 0.0005 and a learning rate of 0.001, which is lowered to 0.0001 after 60000 images. Stochastic gradient descent is used as optimizer.

### 3.1.2 Classifier

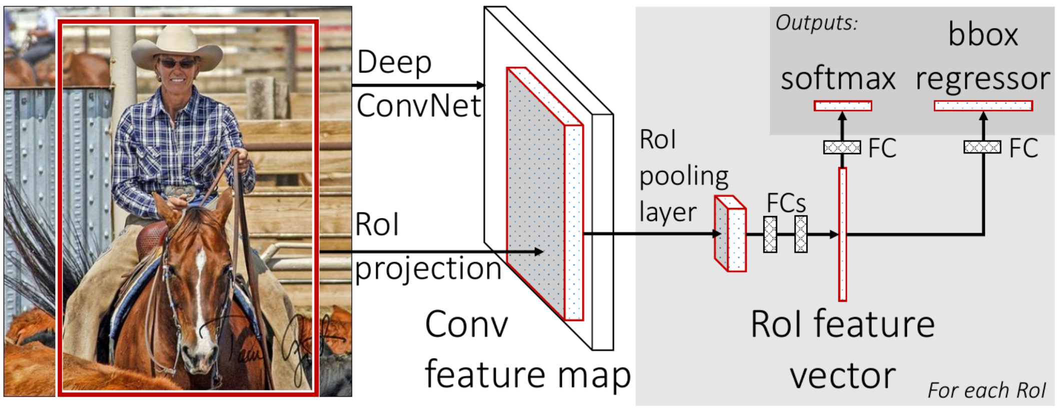


**Figure 3:** Architecture of the Fast R-CNN classifier on gray background that is used in Faster R-CNN. Note that the RPN of Faster R-CNN replaces the ROI projection part on the left and provides the feature map together with region proposals. Image source [11].

**Architecture.** The classifier network predicts a class label for each region of interest from the RPN of Faster R-CNN. Furthermore, it shares the convolutions of a backbone network with the RPN. The last convolutional feature map together with a region proposal (between the image and gray box) is displayed in Figure 3. The classifier architecture is shown on top of gray background. The features within the region of interest are max pooled into a grid of fixed size and further processed by two fully connected (fc) layers, before two dedicated fc layers predict the softmax class probabilities and the box regression. The classifier predicts a box refinement, like the RPN, to tighten the detection box around the object. The features between the RPN and the classifier network are shared, which enables the classifier to reason about the proximity of the object.

**ROI Pooling.** Classification requires aggregation of information over the whole ROI, which is done with fc-layers. Predicted ROIs vary in the number of features that they contain, but fc-layers require a fixed size input, which is why the ROIs are max pooled into a grid of fixed size (e.g., $7 \times 7$).

**Loss.** The loss function $L(\mathbf{p}, \mathbf{t})$ of the classifier is similar to the loss of the RPN in Equation (1), with the one difference that the classification loss is now computed for multiple class predictions.

$$L(\mathbf{p}, \mathbf{t}) = L_{cls}(\mathbf{p}, p^*) + \lambda[p^* \geq 1]L_{reg}(\mathbf{t}, \mathbf{t}^*), \tag{5}$$

where $\mathbf{p}$ is a vector of class predictions and $p^*$ is the corresponding ground truth. Similar to the multi task loss of the RPN as in Equation (1), the refinement loss only considers boxes whose ground truth class $p^*$ is not the background label 0.

### 3.1.3 Joint Training

The classifier and RPN share features of the same backbone network, which requires the backbone to extract suitable features for both tasks. Furthermore the classifier depends on good region proposals. In order to jointly optimize the three networks, they are trained in an alternating fashion using four steps. The training benefits from initializing the backbone network with an ImageNet-pre-trained model, which is used in step 1 and 2. Step

1. Train the RPN while keeping the backbone weights fixed.
2. Train the classifier on the RPN proposals while also updating the backbone weights.
3. Use the backbone from step 2, fix its weights and finetune the RPN.
4. Use the backbone from step 2, fix its weights and finetune the classifier.

## 3.2 Tangent Convolutions for dense prediction in 3D

Depth sensors sample only the surface of objects. Consider a sphere as in Figure 4, the point cloud for the sphere will consist of points from the surface, not from the interior.

3D grid convolutions do not exploit this sparsity of the point clouds and still process the whole volume of the sphere, which makes them memory- and speed inefficient for large scenes. Tatarchenko et al. [4] therefore extend the 2D convolution operator to work on local surface structure. Tangent images are used as a representation of local surface structure. They are computed for every point in the point cloud and contain the projected neighboring points onto the tangent image plane. Such a tangent image is displayed in Figure 4. Computing this for every point on the sphere creates a set of tangent images, which represents the whole sphere with local views. The tangent images are directly related to the point cloud and their structure and can therefore be precomputed. This allows an efficient implementation of a convolutional neural network (CNN) that is based on tangent convolutions.

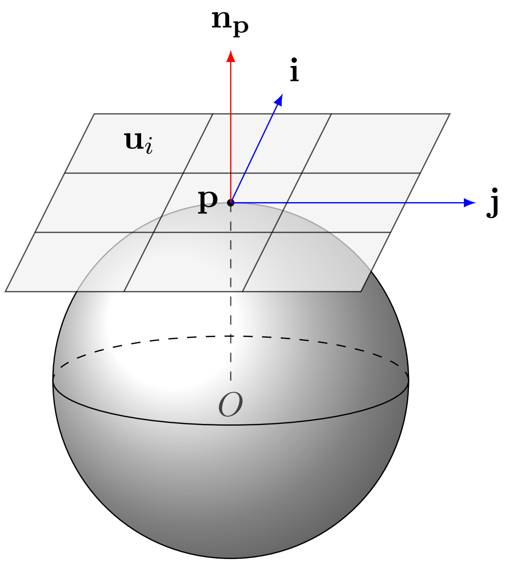


**Figure 4:** Schematic example of one tangent image for a sphere. The vectors $\mathbf{i}$ and $\mathbf{j}$ span the tangent image plane.

**Tangent plane estimation.** Only the local neighborhood of a point contributes to the tangent image at this point. The neighborhood of a point $\mathbf{p}$ from the point cloud $P = \{\mathbf{p}\}$ is defined as a sphere with radius $R$ around the point, thus all neighbors $\mathbf{q}$ satisfy $||\mathbf{p} - \mathbf{q}|| < R$. The orientation of the tangent image is determined by three vectors $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{n}_p$. These are the eigenvectors of the covariance matrix $C = \sum_{\mathbf{q}} \mathbf{r}\mathbf{r}^\top$, where $\mathbf{r} = \mathbf{q} - \mathbf{p}$. The vectors $\mathbf{i}$ and $\mathbf{j}$ correspond to the largest two eigenvalues and span the image plane $\pi_p$, see Figure 4.

**Convolving a tangent image.** The tangent image, as displayed in Figure 4, is a raster image that is convolved with a 2D kernel. This convolution $X(\mathbf{p})$ at point $\mathbf{p}$ is defined as the matrix multiplication of a kernel $c(\mathbf{u})$ with the signals $S(\mathbf{u})$ of one tangent image:

$$X(\mathbf{p}) = \sum_{\mathbf{u}} c(\mathbf{u}) * S(\mathbf{u}), \tag{6}$$

where $\mathbf{u} \in \mathbb{R}^2$ is a pixel in the tangent image. The signal $S(\mathbf{u})$ stands for a value like color, height or abstract features. The following paragraph describes how to map the signal of a point $F(\mathbf{p})$ to the pixel signals.

**Signal interpolation.** The mapping $S(\mathbf{u}) \to F(\mathbf{p})$ happens in two steps. The first step is to project the points $\mathbf{q}$ (blue points) onto $\pi_p$ by multiplying them with the vectors $\mathbf{i}$, $\mathbf{j}$. The points on the image plane (black points) are denoted as $\mathbf{v}$, where $\mathbf{v} = (\mathbf{i}^\top\mathbf{r},\ \mathbf{j}^\top\mathbf{r})$. The projected points $\mathbf{v}$ inherit the signals:

$$S(\mathbf{v}) = F(\mathbf{p}). \tag{7}$$

The projection from above is displayed as continuous- and the projection from below the image plane is displayed as dashed line in Figure 5.

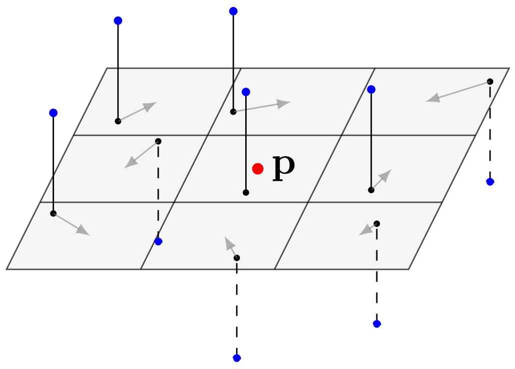


**Figure 5:** Mapping the point signals $F(\mathbf{p})$ to the pixel signals. Projection is shown between blue and black points. The nearest neighbor interpolation between black points and the cell centers is shown as gray arrows. Note that $\mathbf{p}$ is the nearest neighbor of the center cell. Figure reproduced from [4].

The second step is to assign each cell in the tangent image a signal from a point $\mathbf{v}$:

$$S(\mathbf{u}) = \sum_{\mathbf{v}} w(\mathbf{u}, \mathbf{v}) * S(\mathbf{v}), \tag{8}$$

where $w(\mathbf{u}, \mathbf{v})$ is a kernel weight, that determines the influence of $\mathbf{v}$ on $\mathbf{u}$. For the nearest neighbor (NN) interpolation, which Tatarchenko et al. use, $w$ becomes

$$w(\mathbf{u}, \mathbf{v}) = \begin{cases} 1, & \text{if } \mathbf{v} \text{ is } \mathbf{u}\text{'s NN} \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

The nearest neighbor interpolation is displayed in Figure 5 in form of gray arrows, which point from the nearest point v to the cell center.

**Tangent Convolution Layer.** This paragraph describes how the tangent convolution defined in Equation (6) can be used efficiently in a convolutional neural network. The signal sharing between $\mathbf{u}$ and any point $\mathbf{p}$ from the point cloud depends only on the point cloud structure. For simplification, the relation between $\mathbf{u}$ and $\mathbf{p}$ is summarized in the selection function $g(\mathbf{u})$, which can be precomputed. Consistently the signal at $\mathbf{p}$, which is returned by $g(\mathbf{u})$, is denoted as $F(g(\mathbf{u}))$. Figure 6 shows the main components of an efficient implementation of a tangent convolution layer. $\boldsymbol{F_{in}}$ is the input to a tangent layer, where $N$ is the number of points and $C_{in}$ represents the input channels per point, like a red, green and blue value for color. During runtime, the precomputed indices $I$ from $g(\mathbf{u})$ are used to assemble the point signals in the matrix $\mathbf{M}$. The tangent images are represented as a flattened vector $L = l * l$, where $l$ is the filter size.

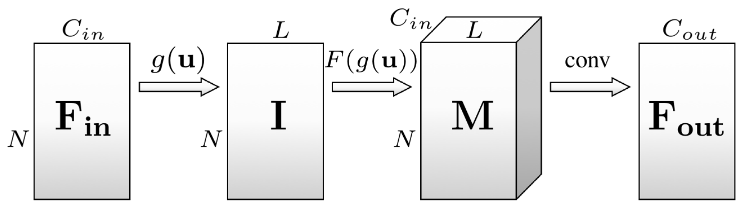


**Figure 6:** Tangent convolution layer. Image source [4].

All the features in one tangent image are convolved with a kernel of the same size as the tangent image. This results in a new feature that corresponds to the center point $\mathbf{p}$ of the tangent image. Hence, succeeding tangent convolution layers reuse the indices $I$ to assemble $\mathbf{M}$ with the previously computed features. By default, the depth of the convolution kernels equals $C_{in}$ and the number of kernels that are used determines $C_{out}$.

**Depth feature.** The local depth feature is defined as distance to the tangent plane $\pi_p$. This distance of a point $\mathbf{q}$ to the tangent plane in $\mathbf{p}$ is computed by $d = \mathbf{n_p}^\top(\mathbf{q} - \mathbf{p})$, where $\mathbf{n_p}$ is the normal vector of $\pi_p$. In contrast to other features like height above ground, depth does not depend on the point itself, but rather on the tangent image. Since one tangent image is defined per point, one point contributes to multiple tangent images as a neighbor. This is displayed in the 2D case in Figure 7. Note how the depth feature (dashed line) of $\mathbf{p_1}$ changes for different tangent planes $\pi_{p2}$, $\pi_{p3}$, $\pi_{p4}$. Therefore the index matrix $\mathbf{I}$ cannot be used to assemble the depth features. Instead,

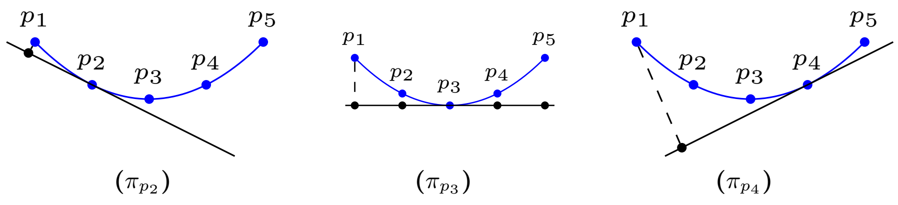


**Figure 7:** The depth feature cannot be assembled with the index matrix $\mathbf{I}$, because the depth feature of a point depends on the tangent image. Instead, the depth images are precomputed and fed to the network as an extra channel. Schematic display of the distance feature and its dependency on the tangent plane. The distance of $\mathbf{p}_1$ changes for tangent planes $\pi$ in points $\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$.

the depth is precomputed for each tangent image pixel and added to the intermediate tensor $\mathbf{M}$ as an extra channel. This needs to be done only once for the first convolution layer, as the subsequent convolution layers process abstract features.

**Pooling.** It is common in convolutional neural networks to spatially aggregate features by inserting pooling layers between convolutional layers. Pooling in the tangent convolution setting is defined on a regular grid, not on tangent images. In each pooling layer the step width of the grid increases by a factor of two. Assuming a pooling filter of size $2 \times 2 \times 2$ and at most one point per grid cell $\mathbf{g}$ (ensured by initial downsampling), then there is a set $\nu_g$ of at most 8 points from the point cloud $P = \{\mathbf{p}\}$, that contributes to a single grid point. Similarly for the point features:

$$\mathbf{p}'_\mathbf{g} = \frac{1}{|\nu_g|} \sum_{\mathbf{p} \in \nu_g} \mathbf{p} \qquad \text{and} \qquad F'(\mathbf{p}'_\mathbf{g}) = \frac{1}{|\nu_g|} \sum_{\mathbf{p} \in \nu_g} F(\mathbf{p}). \tag{10}$$

In order to maintain linear complexity in the number of points during runtime, Tatarchenko et al. precompute an index matrix $\mathbf{I}$ with $N_{in} \times 8$ that contains the indices of all points, which contribute to the same grid cell. Using an index matrix for pooling follows the same process of Figure 6. Using $\mathbf{I}$, an intermediate tensor $\mathbf{M}$

with size $N \times 8 \times C$ of the input features can be assembled. 8 features of one point are reduced to one new feature in the output feature map $\boldsymbol{F_{out}}$ with $N \times C$. Note that pooling does not change the depth $C$ of the feature map.

Since pooling changes the point positions in the point cloud, the index matrices, which are used in the convolutional layers need to be computed once for each pooling layer.

**Unpooling.** Unpooling is used in segmentation networks to increase the spatial resolution in the decoder network part. The pooling indices are shared with the unpooling layers. The indices are used in reverse to copy the features of the low resolution point cloud to points in the higher resolution point cloud.

**Architecture.** The architecture, see Figure 8, follows a U-shaped encoder decoder network for semantic segmentation. There are two pooling and two unpooling layers in the network. Features from the encoder are propagated to the decoder via skip connections. Before each pooling layer, two convolution layers with kernels of size $3 \times 3$ are applied to the tangent images. Each convolution layer, except the last one is followed by a leaky ReLU activation function with a negative slope of 0.2. The last convolution layer applies $1 \times 1$ convolutions to predict a class label for each point. The network is trained using the Adam optimization procedure with an initial learning rate of $10^{-4}$.

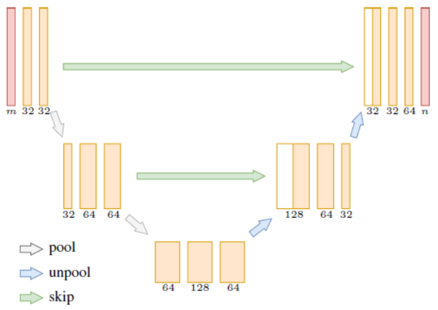


**Figure 8:** The architecture used by Tatarchenko et al. is inspired by the U-Net architecture from Ronneberger et al. [32], but uses tangent convolution layers as the main building block. Image source [4].

# 4 Approach

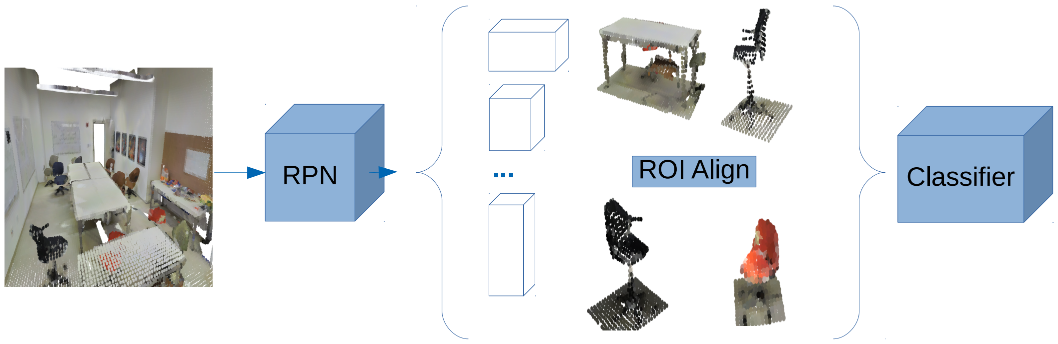


**Figure 9:** Our architecture is composed of three parts: The region proposal network, the ROI Align layer and the classifier network.

We solve the object detection task for 3D data with the architecture shown in Figure 9. Our architecture follows the principle of region based object detection approaches. Region based approaches, as described in Section 3.1, extract regions of interest (ROIs) and then classify them. Our pipeline takes a point cloud as input. The first block is our 3D region proposal network (RPN), which uses tangent convolutions for efficient processing. ROIs are represented as bounding boxes in the point cloud. The RPN predicts their positions and refinements. As a last step, our classifier network assigns a class label to each proposed region. The result of the proposed architecture is a set of boxes with class labels, where each box encloses an object in the input point cloud.

## 4.1 Region Proposal Network

The RPN operates on the whole scene, which is represented by the point cloud, and tries to find volumes that contain single objects. It does so by extracting features from the point cloud in three convolutional blocks and two pooling layers. Within these layers, local features are aggregated into bigger, more complex ones that make

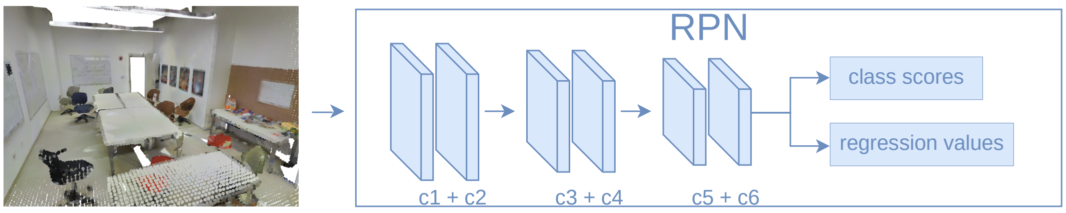


**Figure 10:** Region Proposal Network architecture. The RPN extracts features by applying six convolutions, one for each feature map, and two average pooling steps. The feature maps are represented as vertical blocks, with the corresponding convolutional layer below.

it possible to reason about objects. The features from the last convolutional block are used to predict class scores $p_k$ and regression values $\mathbf{t}_k$. The RPN does not predict box coordinates. Instead, we parameterize the RPN predictions relative to a set of $k$ boxes, which are called anchors. The class score $p_k$ is the probability that the anchor box $k$ contains an object. The regression values $\mathbf{t}_k$ are used to better fit boxes to objects.

Tangent convolutions are used to efficiently process the whole scene. The feature extraction corresponds to the encoder graph that was used by Tatarchenko et al. [4] in the semantic segmentation network. Each convolutional block contains two convolution operations with $3 \times 3$ filters. After each block, the features are pooled into a lower resolution. This increases the receptive field of the CNN. The receptive field size of the RPN is big enough to cover most objects in the scenes. The receptive field describes the part of the point cloud that is visible to a neuron. The receptive field is the same for all neurons in one layer. In the first layer, the receptive field corresponds to the search radius $r$ that is used to construct a tangent image, see Section 3.2. As we stack convolutional layers $s$, the receptive field increases linearly (here $s = 2$ per convolutional block). For pooling layers the receptive field increases exponentially. In the tangent convolution case, $r$ doubles in each pooling layer. For an initial search radius $r = 5cm$, one tangent image covers $2r = 10cm$ of the point cloud. For the whole CNN this sums up to $2 * 10cm + 2 * 20cm + 2 * 40cm = 140cm$ as Tatarchenko et al. describe in [4].

The features from this encoder are used for two predictions: the class score and the refinement values. Both are predicted for every location by performing two $1 \times 1$ convolutions.

### 4.1.1 Anchor Definition

We use anchors as in Faster R-CNN (Section 3.1) and define them in 3D space. Designing anchors for 3D is more difficult than 2D, due to the additional degree of freedom and the different representation of data. Point clouds are a sparse representation of a scene, whereas images consist of a dense grid of pixels.

**Global Positioning.** The information in an image is stored in a grid. Therefore it is straightforward to position anchor boxes at grid cells, to cover multiple locations in the image. For the continuous 3D space, which point clouds span, such a grid is not available by default and as motivated in Section 4.1, we do not impose one for efficiency. Instead of using a grid, we position the anchor boxes at the points from the point cloud. Since the RPN performs a per point prediction in the feature map, we use the point positions that correspond to the last feature map.

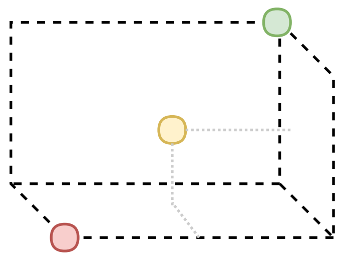


**Figure 11:** The anchor box is represented by its three backward facing sides. The three colored points represent: **Red** lower left corner, **Green** upper right corner, **Yellow** center.

**Relative Positioning.** Figure 11 visualizes three possible ways to position a box relative to a point. We can consider the point being the center of our anchor box (yellow) or we can consider the point being the corner of the anchor box (red and green). Our goal is to position the anchor at a point, such that it surrounds all points that belong to the same object. This is a difficult task for 3D surface scans. First of all, some objects, like tables, naturally do not have points at their geometric center. Moreover, most of the objects from real world scans are partially occluded. So there are no points of the object at the ideal anchor box corners.

**Ground truth box design.** Scenes and objects appear in many rotations, but the ground truth bounding boxes are, by design, aligned with the global coordinate system. If the room is rotated, then all bounding boxes change their sizes. In addition

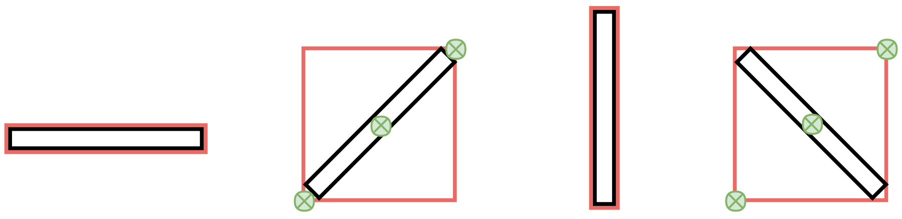


**Figure 12:** Schematic bounding box positioning for a door like object in different rotations.

to the relative positioning problem, this further complicates the alignment of the anchor boxes with the ground truth boxes on the basis of points. This is shown in a schematic view of a door in Figure 12. The ground truth bounding box spans all points of the object. In case the main direction is not the same as one of the coordinate system axes, the bounding box size increases a lot. Defining the main direction of an object is not always possible. It is difficult, e.g., to determine the main direction of a chair. Figure 12 shows a schematic view of a door with its ground truth box from the top down / birds eye view. This schematic demonstrates how the size of the bounding box changes, as the door appears in different rotations. Furthermore, we see the reference points needed for the box in green. In the fourth case, no point of the object is close to one of the reference points, which makes it very difficult to get an anchor at a reasonable position. In addition to the anchor positioning problem,

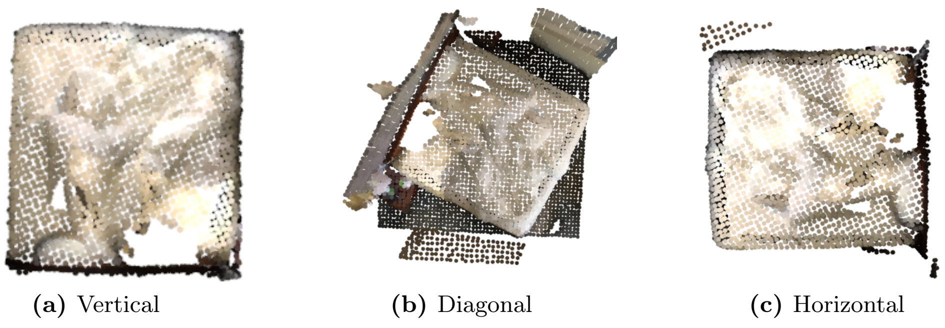

**(a)** Vertical **(b)** Diagonal **(c)** Horizontal

**Figure 13:** The three images show the difference in object appearances for three rotations relative to the coordinate system.

the appearance of the same object changes for different rotations. Figure 13 shows this for a bed. Note how the point cloud for a diagonally rotated bed (b) includes more surrounding structure than a rotation that is aligned with the grid (a,c).

**Appearance.** The bounding boxes of the objects vary greatly in size and aspect ratios. Pictures and doors are thin objects, whereas big objects, like tables and beds are wide and not so high. One anchor is not enough to cover all the objects with a sufficient overlap. Therefore, we design multiple anchors at one location, covering different scales and aspect ratios.

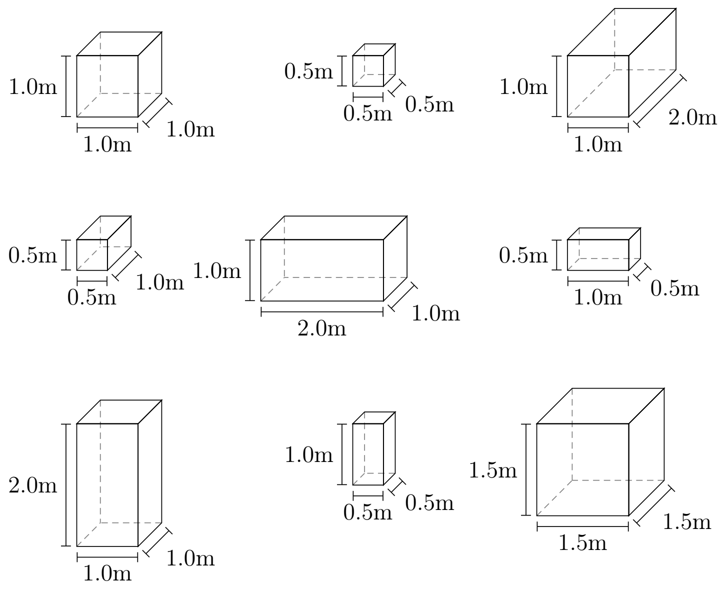


**Figure 14:** Standard boxes that are used as anchors.

Figure 14 shows the standard anchor set, for which we use the standard sizes 0.5, 1.0, 1.5 and 2.0 meter and use them arbitrarily for the x, y and z dimension of the box.

### 4.1.2 Training

Instead of predicting box coordinates or box sizes freely in the room, the RPN relies on anchors as reference boxes. Regressing both, the position and the size of the box in the scene is too difficult for the network. Therefore, we relax the problem into a classification task (object at position or not) and an easier refinement task, which predicts how the box associated with this location should be changed to enclose the object tightly. The RPN predicts the objectness and refinement for each point in the point cloud. We use the term objectness to describe the RPN prediction of object locations. For training the RPN on the classification and regression task, we need to compute targets from the anchors and ground truth boxes.

For the objectness targets $p^*$, we consider a set of anchors $A = \{\mathbf{a}\}$ and compute the Intersection over Union (IoU) between them and all ground truth boxes $G = \{\mathbf{g}\}$. IoU is a measure to describe the overlap of two boxes. If the intersection of the boxes equals their union, then the boxes are equal in shape and position. The IoU $\mathbf{u}$ is computed per box $\mathbf{u} = \frac{\mathbf{a} \cap \mathbf{g}}{\mathbf{a} \cup \mathbf{g}}$. We mark all anchors with an IoU greater than a threshold as positive ($A^+ = \{\mathbf{a}\}, \forall \mathbf{u} > 0.5$). Similarly, we classify all anchors with an IoU smaller than a threshold as negative ($A^- = \{\mathbf{a}\}, \forall \mathbf{u} < 0.3$). The anchors with IoU values between these thresholds overlap with an object too much to be classified as negative, but too little to be classified as positive. These anchors do not contribute to the training. Caused by the imbalance of points and objects, negative anchors occur more often than positive anchors. Thus, we limit ($M$) the number of positive and negative anchors and balance their quantities using random sampling. All remaining positive and negative anchors contribute to the objectness training, where $P^* = \{A^+, A^-\}$. We use a standard cross entropy loss for the objectness prediction.

$$L_{cls}(p, p^*) = -(p^* * \log p + (1 - p^*) * \log(1 - p)). \tag{11}$$

Predicting the refinements is a regression problem. Refinements are the parametrization $\mathbf{t}$ at a location between a box $\mathbf{b}^\top = [x, y, z, w, h, d]$ and the ground truth box $\mathbf{g}^{*\top} = [x^*, y^*, z^*, w^*, h^*, d^*]$, relative to an anchor $\mathbf{a}$. We compute the refinements for all positive anchors.

$$\begin{aligned}
t_x &= (x - x_a)/w_a, & t_x^* &= (x^* - x_a)/w_a, \\
t_y &= (y - y_a)/h_a, & t_y^* &= (y^* - y_a)/h_a, \\
t_z &= (z - z_a)/d_a, & t_z^* &= (z^* - z_a)/d_a, \\
t_w &= \log(w/w_a), & t_w^* &= \log(w^*/w_a), \\
t_h &= \log(h/h_a), & t_h^* &= \log(h^*/h_a), \\
t_d &= \log(d/d_a), & t_d^* &= \log(d^*/d_a).
\end{aligned} \tag{12}$$

In order to get an IoU of one between the predicted box $\mathbf{b}$ and $\mathbf{g}$, the center of the anchor, $x_a, y_a, z_a$, needs to be shifted to the center of the ground truth box, $x^*, y^*, z^*$. Furthermore, the width, height and depth of the anchor need to be scaled to match the ground truth box size. For the refinement ground truth, the center shifts are computed for all three dimensions. These are normalized by the length of the anchor in each direction $w_a, h_a, d_a$. The targets for scaling the predicted box $(t^*_w, t^*_h, t^*_d)$ is given by the ratio between anchor and ground truth box per dimension. The logarithm is applied to the ratios for normalization. The normalization helps in the learning process as neural networks perform best for values that are in a fixed range for all samples, for us these are -1 and 1. The smooth $L_1$ loss is defined by Girshick [11] in Equation (3).

$$L_{reg}(\mathbf{t}, \mathbf{t}^*) = \sum_{j \in x,y,z,w,h,d} \text{smooth}_{L_1}(\mathbf{t}_j - \mathbf{t}^*_j), \tag{13}$$

We combine the classification loss, Equation (11), and the refinement loss, Equation (13) in the multi-task loss:

$$L(\{p_i\}, \{\mathbf{t}_i\}) = \sum_i^M L_{cls}(p_i, p^*_i) + \sum_i^M p^*_i * L_{reg}(\mathbf{t}_i, \mathbf{t}^*_i), \tag{14}$$

where $p$ is the objectness prediction and $\mathbf{t}$ is the refinement prediction. The loss is computed for all target locations $i \in M$ in the point cloud.

Since we use multiple anchors per position, the network has to predict multiple objectness scores and refinements. When $k$ is the number of anchors per location, the RPN predicts $2 * k$ output scores for the objectness and $6 * k$ refinements.

## 4.2 Classifier

### 4.2.1 Architecture

The task of the classifier is to predict a label separately for each region proposal box from the RPN. Therefore, we cut all points within the proposed box and give them to the classifier as input. The classifier extracts features from these points with the aid of several convolutional blocks and pooling layers. The pooling layers aggregate local features into more complex ones and increase the receptive field. This first part corresponds to the Tangent Convolution encoder graph from Figure 8. In order to classify the content of the proposed box, the classifier needs to aggregate information over the entire volume. Therefore we add two fc layers to the network, see Figure 15. Because they are fully connected to all features of the previous layer, they can reason about the whole volume and make a prediction of the possible class labels.

The size of the proposed box and the number and location of points in each box

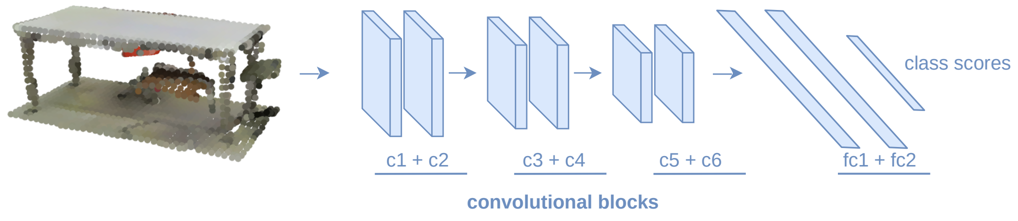


**Figure 15:** The classifier architecture that we propose uses three convolutional blocks with two pooling layers in between them for extracting and aggregating features. Similar to the tangent convolution pooling operation, we use an index matrix to aggregate features in a unified way before the fc layers.

vary across region proposals. This is harmful for a unified, structured assembly of information in the fc layers.

In analogy to images, where the pixel grid enables a standardized way of assembling information, we compute a raster for the ROI point clouds. Before the first fc layer, we assemble all features according to the raster indices.

The ROI alignment procedure in Section 4.3 describes the steps in detail that are necessary to make the RPN and the classifier compatible, such as computing the raster.

### 4.2.2 Training

In our pipeline the classifier network operates on the proposed boxes from the RPN and predicts multiple classes. We train the classifier on ROI predictions from a RPN that has been trained until convergence. We ensure that the ROI predictions have at least 0.1 IoU with a ground truth box. Then we assign the ground truth label to the ROI based on the maximal IoU value. We ensure that the classifier is trained and evaluated on regions that are extracted from scenes of the training and testset of the RPN. During training, the multi class classification loss (cross entropy) is minimized:

$$L_{multi\_cls}(\mathbf{p}, p^*) = -\sum_{c=1}^{C} p_c^* * \log(p_c), \tag{15}$$

where vector $\mathbf{p}$ contains the predicted probabilities for all class labels $\{c\}$ and $p^*$ is a one hot encoded vector of the ground truth class label. We use Adam as optimizer with a fixed learning rate of $10^{-4}$.

## 4.3 ROI Alignment

As explained in Subsection 4.2.1 the classifier needs unified input point clouds across all regions proposals. Like in images, we use a grid of a fixed size and step width to enable consistent information aggregation over all ROIs. We span the grid from the center of the coordinate system to four meter in each dimension ($W = H = D = 4$ meter) and use a step width of 10cm in the initial resolution. The classifier predicts labels independently from the ROIs position in the room, so we shift all points within the ROI, such that the lower left corner of the ROI is at the center of the coordinate system. The difference in object sizes is an important information in the data. Therefore we do not scale the objects to fit the dimensions of the grid. All points that exceed the limits of the grid in any dimension are removed. Their information is lost to the classifier network. An analysis of the ScanNet objects shows, that this affects less than 0.65% of all objects. The design of this grid satisfies three requirements. First, it is big enough to cover large objects. Second, the step width (cell size) is small enough to represent small objects well enough for classification. Third, the resulting number of cells is not too large. $\frac{4m}{0.1m} = 40$ cells in each dimension. After two pooling layers the step width increased to 40cm, which results in $10 \times 10 \times 10$ voxels that are processed with the fc layers.

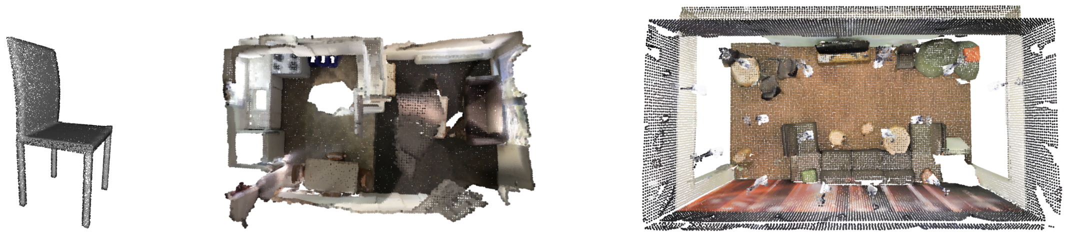
The ROI Alignment step uses the precomputations of tangent convolutions to average pool the points within the ROI into the grid cells. In contrast to the tangent convolutions approach we now store the voxel indices together with the point indices and extend the voxel indices by those who do not contain any points. Note that this is crucial for the standardized assembly of features. We unroll the voxel grid that corresponds to the point cloud after the second pooling layer into a $1 \times N_{out}$ vector, where $N_{out} = 10^3$. Each position (voxel) in this vector contains the point index or an empty cell label. This is our precomputed index vector $\mathbf{I}$, compare Figure 6. During runtime, we use $\mathbf{I}$ to assemble the intermediate matrix $\mathbf{M}$, which is of size $N_{out} \times C$ from the feature map $F_{in}$. $\mathbf{M}$ is then convolved with the weights of the first fully connected layer.

# 5 Experiments

We evaluate the performance of the presented approach on the task of 3D semantic instance segmentation. We provide results of our method on two real world data sets, together with qualitative and quantitative results for single components and compare our results to the leader board of the ScanNet 3D instance segmentation benchmark.

## 5.1 Datasets

For training and evaluating our models we used three datasets. ScanNet and S3DIS are used for the evaluation of our object detection method. We use ShapeNet to test our classifier network on artificial data. Figure 16 shows rooms of both real world data sets.



**(a)** ShapeNet **(b)** ScanNet **(c)** S3DIS

**Figure 16:** (a) Example object from ShapeNet. (b) Example room of ScanNet. (c) Example room of S3DIS.

**ShapeNet [33]** is a large data set of artificial, 3D CAD models. The objects are represented as meshes. To transform them to point clouds, we use poisson disc sampling (PDS). For each model we sample as many points as it has faces, but minimum 25,000. PCD ensures that the points are evenly spread across the object. Just like laser scans, only the surface of a model is sampled. From the whole ShapeNetCore model database we sample a subset of six model classes. Some model classes occur more often then others. We balance the class frequencies by removing models from the more frequent ones. The balanced class frequencies are within 10% to 20%. We split our data into a training and test set, such that no model is present in both of them. we manually ensure that the class frequencies in the training and test set are similar.

**ScanNet [5]** contains more than 1500 scans of real world scenes. All scans are annotated with instance level semantic segmentations for 20 classes. The scenes of ScanNet are not aligned, i.e., rooms are not aligned with the coordinate system, which makes it the most complex data set for indoor scene understanding. On top of that the ScanNet team started a benchmark challenge for instance segmentation of 18 classes. Methods that submit to their leader board are evaluated against a non-public test set. In addition to the 3D point clouds, ScanNet also provides 2.5 million images for the scenes. So methods that publish on the leader board are not limited to 3D only. We use the train, validation and test split that is provided by the authors of ScanNet.

**Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) [6]** provides surface scans of 272 indoor scenes together with 3D semantic instance labels for 13 classes. S3DIS consists of 6 areas, as suggested by the authors, we use Area 5 for testing and the other areas for training.

## 5.2 Evaluation metrics

We perform a point-based evaluation of the predictions. Each point can end up in one of the following categories:

1. True Positive (TP),
2. True Negative (TN),
3. False Positive (FP),
4. False Negative (FN).

For the example case of predicting, whether there is an object at a position or not, the types illustrate that: the object presence is correctly predicted (TP) or the absence of an object is correctly predicted (TN) or we falsely predicted an object to be present (FP) or we falsely predicted the absence of an object (FN). Following the conventional protocol, we use multiple standard evaluation metrics.

**Accuracy.** The accuracy is the proportion of correct predictions.

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

**Overall Accuracy.** We report the overall accuracy when examining the classifier performance.

$$oA = \frac{TP}{TP + TN + FP + FN}$$

**Precision.** The precision points out how accurate the method is at predicting positives.

$$P = \frac{TP}{TP + FP}$$

**Recall.** The recall points out how many of the ground truth positives the method under test can identify.

$$R = \frac{TP}{TP + FN}$$

**IoU** Intersection over Union is used to measure the overlap between two sets $A$ and $B$. The IoU is typically computed for boxes, in the object detection task, and for points, in segmentation tasks.

$$IoU = \frac{\text{Intersection}}{\text{Union}} = \frac{A \cap B}{A \cup B}$$

We report IoU values for boxes, when we test the overlap between anchors and ground truth boxes. The evaluation on the ScanNet benchmark is for instance segmentation, thus reports mean Average Precision for point IoUs.

**Average Precision (AP).** We use the Average Precision from the ScanNet Benchmark evaluation. Average precision values are reported for IoU values of 25%, 50% and in the range [0.5 : 0.95 : 0.05] AP. This means that a detection only counts as TP if it sufficiently overlaps with the ground truth box.

$$AP = \frac{1}{N} \sum_{r}^{N} P(r),$$

where $N$ is the total number of recall values $r$ and $P(r)$ is the precision value at this recall.

**Mean Average Precision (mAP).** We report the mAP as the average over the per class AP.

$$mAP = \frac{1}{C}\sum_{c=1}^{C} AP(c).$$

The ScanNet instance segmentation benchmark evaluates the performance of methods based on points. Our method predicts bounding boxes around objects. In order to compare our method on the instance segmentation benchmark, we propagate the box label to all pixels within the box. Some objects in the point cloud overlap along the z-axis, e.g., chairs and tables. The detection box of a table cannot exclude the points that belong to the chair from the label assignment and therefore introduces an error. Figure 17 shows that the AP for 25% IoU of most classes is not or only slightly

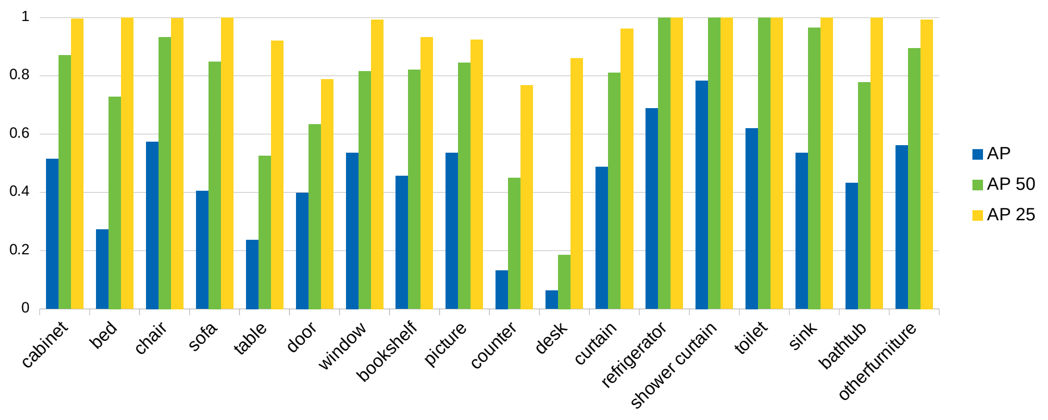


**Figure 17:** When evaluating the object detector on the instance segmentation benchmark, we wrongly classify points of overlapping objects. This plot shows the evaluation scores for instance masks produced by ground truth boxes.

affected. The more restrictive the IoU value is, the more impact the wrongly labeled points have on the AP values. Furthermore, we see that the error correlates with the classes. Shower curtains are the least affected, whereas desks are affected the most.

Table 1 shows that over all classes the average precision declines to 45.8%. These are the scores that our current object detector can maximally achieve on the ScanNet Benchmark.

| mAP | mAP 50% | mAP 25% |
|---|---|---|
| 0.458 | 0.784 | 0.952 |

**Table 1:** Mean AP values for instance masks produced by ground truth boxes evaluated on the ScanNet instance segmentation benchmark. AP at IoU 25% is slightly affected by the inaccurate point labelling. The more restrictive the IoU values are, the bigger the impact of the wrongly included points.

## 5.3 Region Proposal Network

As described in the approach chapter, the RPN uses anchors as an initial uninformed guess about objects in the scene. Predicting region proposals corresponds to predicting whether there is an object of a certain size, which is determined by the anchor, at a certain location. Since the standard anchors cannot perfectly match all objects in the scene, there is also a refinement module. This makes anchors an important parameter of the network. In the following subsections we experiment with the appearance, number and positioning of anchors.

### 5.3.1 Anchor Appearance

In this experiment we test whether it is important to make the anchor design dependent on the object dimensions. Therefore, we tailor the appearance of the anchors to the mean shapes of the ground truth boxes per object class. Afterwards, we compare the performance of the RPN for the standard anchor set defined in Section 4.1.1 and the tailored anchors.

Figure 18 shows the mean and the standard deviation for the length of the bounding box in the x, y and z direction per class. This plot integrates data for all bounding boxes in ScanNet, this includes rotations of objects. The bars depict the mean values in meter. The single gray lines, which are centered at the mean values, depict the standard deviation.

Big standard deviations stand for diverse object appearances. The standard deviations are in general smaller for the z-values, than for the x- and y-values, because rotations of the room and objects themselves increases the variance in the x- and y-direction, but not in the z-direction. In contrast to this, refrigerators have a square base area and vary mostly in the height. Furthermore we see from the diagram that chairs and toilets are about the same size and can therefore be covered with one anchor box.

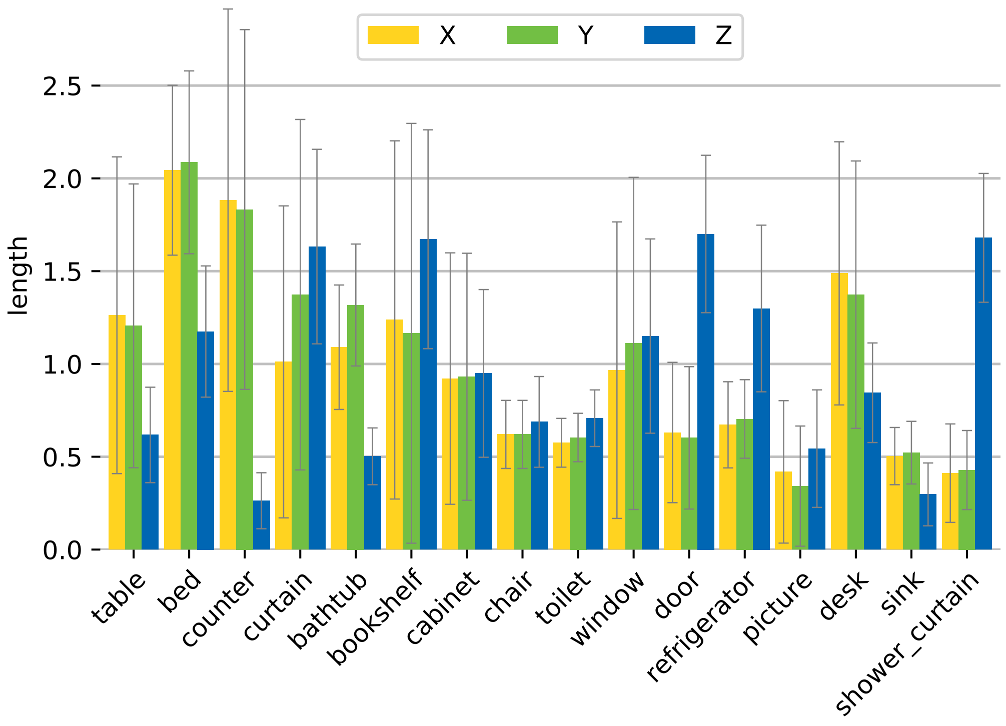
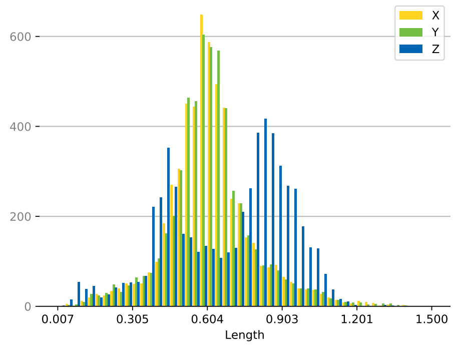


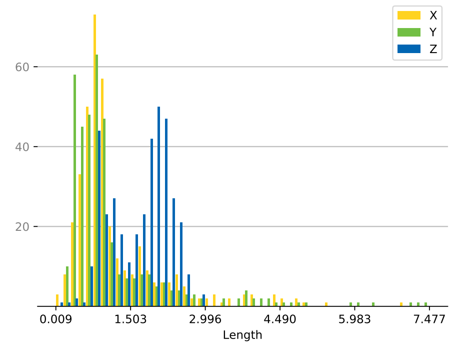
**Figure 18:** The size of the bounding boxes for each dimension. Averaged over the whole ScanNet data set. Mean shape of a bounding Box per class. The error bars in light gray depict the standard deviation for each bounding box dimension.

There are three cases that we need to consider when we design tailored anchors. Imagine two pictures whose bounding boxes have the same size. The pictures are put onto perpendicular walls. In this case the width (x-value) and the depth (y-value) are swapped. Thus we introduce an averaging error for our mean shape bar chart. This means that although the x- and y-value in the bar chart are similar for picture objects, we should still design tight boxes for both directions. In the case described above, the bounding boxes have the same size. There are however many objects of the same object class which vary greatly in size. Since the bar chart only displays mean values per dimension, manifold appearances of objects are averaged. Examples for this case are long and short tables, and, chairs with and without legs. These multi-modalities are visible in the per object class distribution of box sizes. Figure 19 shows this for chairs and bookshelves.

Figure 20 shows that bounding boxes are always drawn relative to the coordinate system in a scene from ScanNet. The left room is aligned with the coordinate system



**(a)** chair



**(b)** bookshelf

**Figure 19:** Histogram for the bounding box size of the classes chair and bookshelf from ScanNet. Histogram of chair is clipped at 1.5 meter.

and so most of the objects in the room are enclosed by tight boxes. The right room is not aligned with the coordinate system, which drastically changes the enclosing box size. We describe this in detail in Chapter 4.1.1. The third case considers the variance that is introduced by the bounding box design. This is not related to the bar chart. The fact that all objects occur rotated with respect to the global coordinate system implies, that we need to consider cube like boxes for otherwise narrow objects.

From Figure 18 and the per class histograms, of which two are exemplarily displayed in Figure 19, we define tailored anchor boxes. Figure 21 shows the tailored anchors. The two boxes in the upper left account for different appearances of objects, like chairs with and without legs. The two boxes below cover tall and small objects. The four boxes in the upper right reflect the general tendency towards cube like boxes.

We compare the overall performance of the RPN for standard anchors and tailored anchors in Table 2. We do this by reporting four measures. The *Baseline IoU* measure is independent of the RPN. It shows the IoU for all positive anchors with the ground truth boxes. The *Objectness Prediction* of the RPN selects anchor boxes and their locations. Ideally this would reproduce the positive anchor selection. Therefore, we can compare the IoU value that the objectness prediction achieves to the *Baseline IoU*, which is an upper bound. For the *Refinement IoU* we apply the refinements, which are predicted by the RPN, to the positive anchors, which are also used for the *Baseline IoU*. If the RPN predicted zero refinements, the *Refinement IoU* shown in this evaluation would be equal to the *Baseline IoU*. For this reason we use the *Baseline IoU* as lower bound for comparing against the refinement. *RPN IoU* combines the

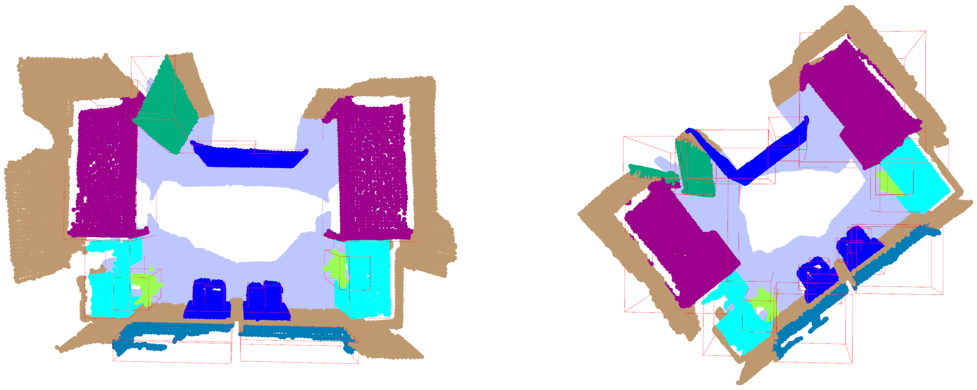

**(a)** aligned **(b)** rotated

**Figure 20:** Room from ScanNet with bounding boxes and instance labels. (a) The room is aligned with the axes of the coordinate system. (b) The same room, but rotated. The bounding box shape and the object shape are not similar any longer in (b). Note how the orientation of the room in the coordinate system changes the box layout.

objectness and refinement predictions. We apply the refinement values only to the anchor boxes that are predicted by the RPN and compute the IoU between the refined anchors and the ground truth boxes.

| | Baseline IoU | Objectness IoU | Refinement IoU | RPN IoU |
|---|---|---|---|---|
| tailored anchors | **44.0** | **22.3** | **45.1** | 23.9 |
| general anchors | 41.0 | 21.8 | 43.8 | **24.6** |

**Table 2:** RPN performance comparison for general and tailored anchors based on box - IoU values in percent and averaged over all classes. 5 anchors per position are used. Note that it is not a weighted average, all classes contribute equally.

Taking the baseline as upper bound, the objectness prediction reaches $\frac{22.3}{44.0} * 100 = 50.7\%$ and $\frac{21.8}{41.0} * 100 = 53.2\%$. When we compare the refinement to the baseline as lower bound, we get a relative improvement of $45.1 - 44.0 = 1.1\%$ for tailored anchors and $43.8 - 41.0 = 2.8\%$ for the general anchors respectively.

So the RPN performs better on the standard anchor set if we look at the relative improvements. The RPN on the standard anchor set also has a better overall performance, as we can see in the last column in Table 2. We can see this effect, that the tailored anchors achieve higher *Baseline, Objectness and Refinement IoUs*, in more detail in the per class evaluation in Table 3.

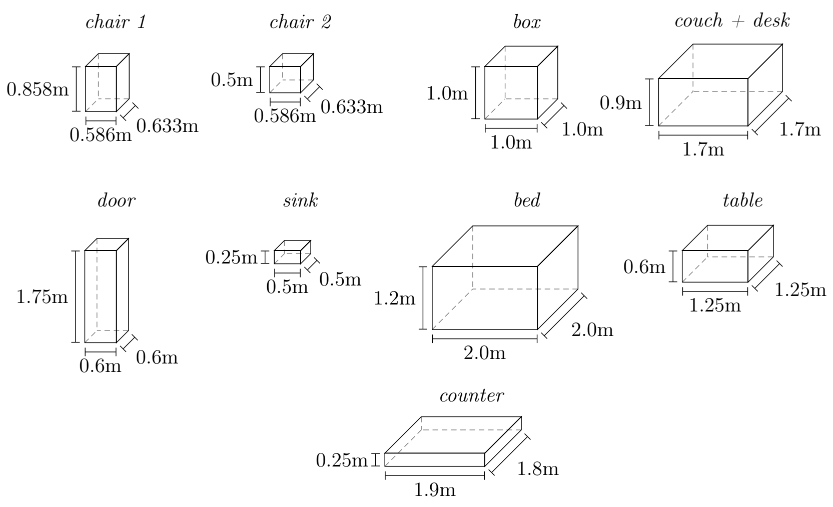


**Figure 21:** Tailored Anchors for the ScanNet dataset, based on the mean shape of the bounding boxes. The mean shape is integrated over all instances of an object class, including rotations.

| | bed | bookshelf | cabinet | chair | couch | counter | curtain | desk | door | sink | table | toilet | window |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S: RPN Prediction | 25.8 | **17.7** | **22.6** | **36.4** | **34.4** | **10.4** | **20.3** | **30.7** | **18.9** | 20.7 | **25.7** | 39.0 | **17.2** |
| T: RPN Prediction | **35.4** | 16.1 | 21.1 | 35.0 | 33.6 | 8.4 | 15.8 | 27.0 | 16.6 | **21.1** | 23.3 | **40.1** | 16.9 |
| S: Refinement | 47.3 | 43.6 | 40.0 | 51.2 | **54.9** | **23.3** | **42.2** | **47.8** | 38.6 | 38.7 | 45.5 | 52.8 | **42.9** |
| T: Refinement | **55.4** | **44.2** | **40.7** | **53.8** | 53.5 | 23.1 | 39.3 | 39.9 | **51.5** | **41.2** | **48.9** | **56.6** | 38.4 |
| S: Baseline | 39.3 | 42.4 | **41.2** | 46.2 | 55.9 | 18.4 | **42.0** | 48.5 | 35.6 | 35.4 | 41.8 | 41.8 | **45.1** |
| T: Baseline | **51.2** | **44.0** | 39.6 | **53.0** | **56.1** | **20.4** | 38.6 | **49.9** | **38.6** | **41.2** | **45.8** | **55.9** | 38.2 |
| S: Objectness | 20.7 | **17.4** | **22.0** | 30.3 | 31.8 | **10.0** | **20.4** | **29.3** | **15.8** | 17.6 | **23.1** | 27.3 | **17.5** |
| T: Objectness | **28.6** | 14.9 | 20.1 | **32.6** | **33.1** | 9.0 | 14.5 | 25.8 | 15.7 | **20.3** | 21.7 | **37.5** | 15.9 |

**Table 3:** Standard Anchors (S) - Tailored Anchors (T) comparison per class.

Furthermore, this experiment shows that the definition of anchor shapes is not a crucial step for the performance of our RPN. Tailoring the anchors can increase their overlap with the ground truth boxes, however it does not increase the performance of the RPN compared to a standard set of anchors.

Figure 22 shows predicted boxes from an RPN that was trained with five specialized ScanNet anchors. The reference point represents the center of the anchor box.

### 5.3.2 Anchor Reference Points

Besides the anchor design, the relative positioning of the anchor box is another key factor for covering points that belong to an object. As described in Chapter 4, we position the anchor boxes relative to points of the point cloud. In this experiment we evaluate the IoU between the ground truth box and the anchor box for three different

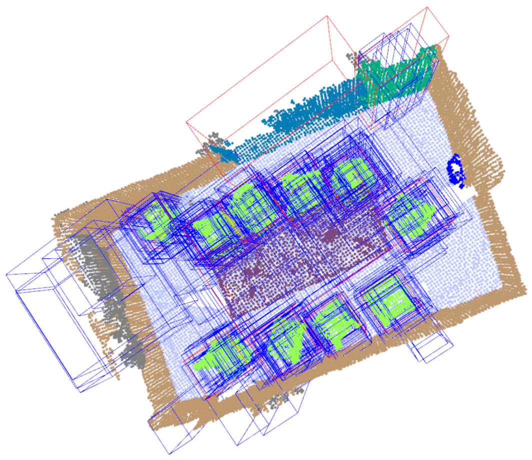


**Figure 22:** Blue: predicted boxes from the RPN, Red: ground truth boxes. It is part of the method to have multiple predictions per object, these can be removed by non maximum suppression (NMS). False positive boxes (see bottom wall) and false negatives (no blue box has an overlap with the big red box in the top center) are bad, since the object detector cannot recover from these wrong predictions.

relative positionings. The box positioning relative to a point is shown in Figure 11.

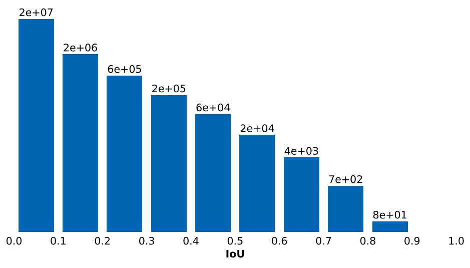
We compute three histograms, which are shown in Figure 23, one for each relative positioning. We evaluate the quality of the positioning based on the IoU of the anchors with the ground truth boxes. Therefore, we accumulate IoU values to bins of width 0.1.

Lower left (a) and upper right (c) reference points produce lower IoU values than the center (b) reference point. Anchors from (a) and (c) do not create IoU values $\geq 0.9$. In addition, the sum of anchors with an IoU value higher than 0.5, is the largest one of the three evaluations.

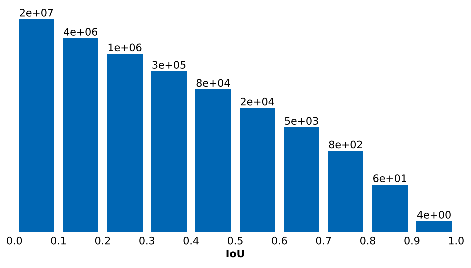
It is the goal of the region proposal network to create many proposals and avoid false negatives. This means that, if the RPN does not predict an object location, our method will not be able to identify the object that the RPN missed. Because we get more anchor boxes with a good overlap for the center reference point, it is used in further experiments.

In this experiment we did not check the reference points per class. It could be that for some objects it is best to use the reference point as lower left in order to get a higher initial IoU. Using a specific reference point per object class could increase the number of anchors with a high IoU value further.

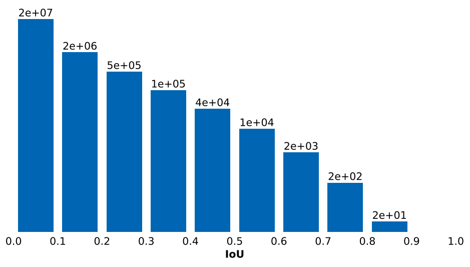
We use anchors for the supervision of the RPN by classifying them into positive



**(a)** lower left



**(b)** center



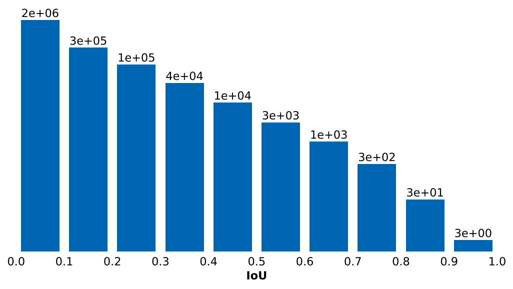
**(c)** upper right

**Figure 23:** Histograms for three different anchor reference points, where the point corresponds to the (a) lower left corner, (b) the center, (c) the upper right corner of the anchor box. Evaluated for nine anchors per position. The number of occurrences is in logarithmic scale.

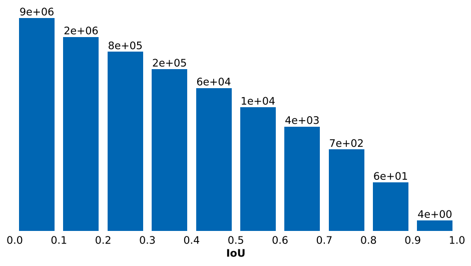
and negative anchors. In object detection for images Ren et al. [12] set the threshold for negative anchors to 0.3 and the threshold for positive anchors to 0.7. Images are projections onto a 2D plane. Without the depth channel, objects are a lot closer to each other in the image. Anchors therefore tend to overlap with several ground truth boxes. Since we operate on sparse 3D point clouds, we can lower the thresholds to 0.1 for negative anchors and 0.5 for positive anchors. Especially the lower threshold for positive anchors is important as the RPN would otherwise create too many false negatives.

### 5.3.3 Number of Anchors

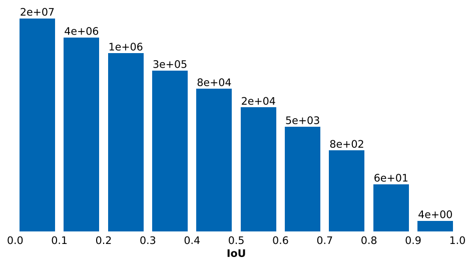
As described in Subsection 4.1.1 *Anchor Definition*, our region proposal network can work with one and more anchors per position. If the refinement prediction was strong enough, only one anchor could be used for predicting objects in the point cloud. Since we have seen in the previous experiments that the refinement is not strong enough to do so, we inspect the RPN performance for three different anchor settings. We test 1, 5 and 9 anchors per location. With 9 anchors we expect to get better IoU values, as we can cover wide, tall, small and thin objects better from the start. For the following evaluations we use ScanNet tailored anchors as defined in Figure 21. Chairs are the most frequent objects in this data set. We use the anchor box (top left) that is tailored to chairs in all three settings. For the 5 anchor setting, we use the top row together with the left anchor of the second row. The 9 anchor setting uses all displayed boxes. We compare the overall IoU values, the number of positive- and the number of negative anchors for the three settings in Figure 24.



**(a)** 1 anchor



**(b)** 5 anchors



**(c)** 9 anchors

**Figure 24:** We evaluate three settings of anchor(s) per position: (a) 1 anchor , (b) 5 anchors and (c) 9 anchors, based on the IoU value with the ground truth boxes. The number of occurrences is in logarithmic scale. The number of good IoU values ($\geq 0.5$) increases as we use more anchors per position, but not with the same rate as the total number of anchors.

One anchor achieves an IoU between 0.9 and 1.0 3 times. Increasing the number of anchors per location by a factor of 5 and 9 does not improve this very good overlap by the same factor. Instead it only slightly increases by one occurrences over the full dataset.

Most of the additional anchors from the settings (b) and (c) in Figure 24 have low IoU values. The negative anchors ($IoU \leq 0.1$) increase by almost the same factor, 4.5 for 5 anchors and 10 for 9 anchors, as the overall anchors. Nonetheless the quantity of positive anchors ($IoU \geq 0.5$) also increases, which is beneficial for the RPN training.

In order to derive the relevance of multiple anchors for the RPN, we compare the validation measures of the RPN for the three settings in Table 4. As we can also see

| | Baseline IoU | Objectness IoU | Refinement IoU | RPN IoU |
|---|---|---|---|---|
| 1 anchor | 27.8 | 19.1 | 25.0 | 20.1 |
| 5 anchors | 44.0 | **22.3** | 45.1 | **23.9** |
| 9 anchors | **48.0** | 20.2 | **48.1** | 22.5 |

**Table 4:** Comparison of the RPN performance for 1, 5 and 9 initial anchors per location. 5 anchors lead to the best RPN performance of 23.9%.

from the histograms, the more anchors we use per location, the more positive anchors we get and the higher is the baseline IoU. One disadvantage thereof is, that we need a lot more anchors overall for it. Thus the network has to predict more refinements. Overall the RPN performs best for five anchors per location out of the three settings that we test. We use this setting for all evaluations of our object detector.

### 5.3.4 ROI Evaluation

In order to test the quality of the ROIs that the RPN predicts, we evaluate them on the ScanNet instance segmentation benchmark. We use the trained RPN from Subsection 5.5.1 with all inputs (DHNC) and assign the predictions the class label of the ground truth box that it overlaps with most. Note, that we do not use the ground truth so select a subset of the proposals. If the proposal and the ground truth do not overlap, one of the ground truth class labels is assigned at random. During evaluation, these proposals count as false positives, because they do not overlap with the ground truth, their class label is not relevant. The ROI boxes are converted into segmentation masks as described in Section 5.2.

Figure 25 shows three AP values per class. The ROIs capture chairs the best and counters the worst. The AP scores of the classes are similar, the RPN is able to detect objects of all classes and does not overfit to a particular class or subset of classes.

| mAP | mAP 50% | mAP 25% |
|---|---|---|
| 0.005 | 0.023 | 0.262 |

**Table 5:** Mean AP values for region proposals with ground truth class label evaluated on the ScanNet instance segmentation task.

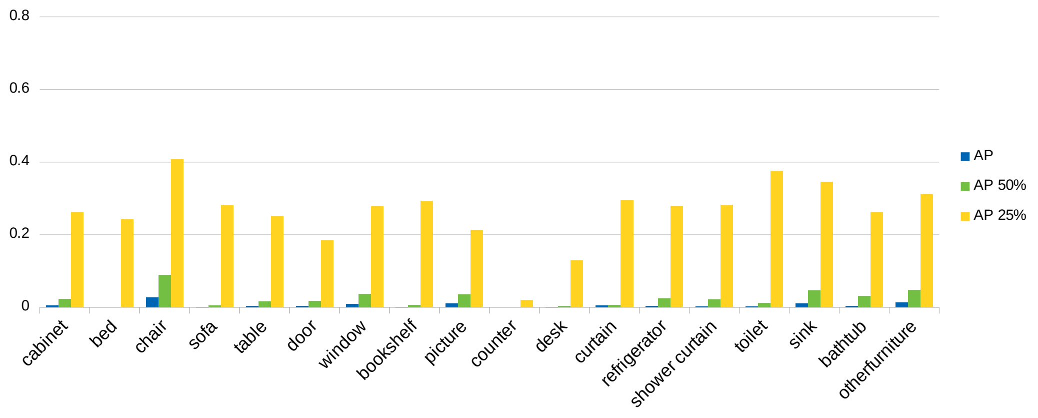


**Figure 25:** RPN proposal evaluation on the ScanNet instance segmentation benchmark. The AP scores across all classes are similar. The RPN predicts ROIs that capture all classes. The objectness prediction is not dominated by a subset of classes.

Table 5 shows the mean AP values for the evaluation. The mAP 25% score is significantly better than the more restrictive mAP evaluation scores. We believe that the position and size of the ROI is not accurate enough to achieve good precision scores for IoUs bigger than 25%.

In total, the validation set contains 4,000 ground truth masks for 304 scenes. In this experiment, the RPN predicts 50,000 masks for the same set. On average, these are 150 masks too many per room. Each predicted mask that does not match a ground truth is considered a FP. Furthermore only one predicted mask that overlaps with a ground truth box counts as TP, all other predictions that target the same ground truth box count as FP. False positives have a negative impact in the AP computation. To conclude, the RPN predicts too many ROIs per scene.

## 5.4 Classifier

(a) ShapeNet (b) S3DIS

**Figure 26:** A table from (a) ShapeNet and (b) S3DIS. In contrast to the table from ShapeNet, the table from S3DIS is incomplete. Points from the front left leg and the back legs are missing.

### 5.4.1 Performance Evaluation

We want to analyze the power of our classifier that uses tangent convolutions as local processing units. The classifier architecture used for these experiments is the same as in Figure 15. In order to test the performance, we train and evaluate the classifier on three different data sets: (1) ShapeNet as described in Section 5.1, (2) objects from S3DIS and (3) RPN region proposals on S3DIS. We refer to (2) as S3DIS-GT and to (3) as S3DIS-RPN. The difference between ShapeNet and S3DIS-GT is displayed

**TC Classifier on ShapeNet: 81.9% oA**

| | bench | cabinet | chair | display | sofa | table |
|---|---|---|---|---|---|---|
| bench | 110 | 0 | 3 | 2 | 15 | 13 |
| cabinet | 14 | 141 | 0 | 8 | 6 | 11 |
| chair | 4 | 0 | 204 | 0 | 8 | 8 |
| display | 9 | 10 | 2 | 99 | 2 | 5 |
| sofa | 26 | 2 | 10 | 0 | 176 | 3 |
| table | 22 | 4 | 1 | 3 | 9 | 173 |
| Recall | 0.595 | 0.898 | 0.927 | 0.884 | 0.815 | 0.812 |
| Precision | 0.769 | 0.783 | 0.911 | 0.78 | 0.811 | 0.816 |
| Accuracy | 0.902 | 0.95 | 0.967 | 0.963 | 0.927 | 0.928 |

**Table 6:** Confusion matrix together with the per class recall, precision and accuracy for the classifier on the ShapeNet validation set.

in Figure 26. Whereas artificial objects from ShapeNet are complete, objects from

S3DIS can have missing parts, due to a blocked view of the sensor. For S3DIS-GT, we use the instance labels to create a point cloud for each individual object and classify it. S3DIS-RPN is constructed as follows. In the object detection pipeline, the classifier operates on proposal point clouds that are predicted by the RPN. These regions can be smaller, bigger or shifted compared to the ground truth objects. We extract regions with a RPN setting that uses five standard anchor boxes per location. We use at maximum 66 regions with a minimum IoU of 0.1 with the ground truth box. These settings are also used for the classifier training and ensure that the classifier is trained on good proposals. The question is if we can see performance drops for real world data and for region proposals as the quality of the objects drops. Table 6

**TC Classifier on S3DIS: 89.0% oA**

| | window | door | table | chair | sofa | bookcase | board |
|---|---|---|---|---|---|---|---|
| window | 35 | 0 | 0 | 0 | 0 | 4 | 5 |
| door | 5 | 127 | 1 | 1 | 0 | 8 | 0 |
| table | 0 | 0 | 147 | 2 | 1 | 19 | 0 |
| chair | 0 | 0 | 3 | 238 | 4 | 1 | 0 |
| sofa | 0 | 0 | 0 | 1 | 6 | 0 | 0 |
| bookcase | 8 | 0 | 3 | 16 | 0 | 183 | 3 |
| board | 4 | 0 | 0 | 0 | 0 | 2 | 34 |
| Recall | 0.673 | 1 | 0.955 | 0.922 | 0.545 | 0.843 | 0.81 |
| Precision | 0.795 | 0.894 | 0.87 | 0.967 | 0.857 | 0.859 | 0.85 |
| Accuracy | 0.97 | 0.983 | 0.966 | 0.967 | 0.993 | 0.926 | 0.984 |

**Table 7:** Confusion matrix together with the per class recall, precision and accuracy for the classifier on S3DIS-GT. The oA is better than for the complete, artificial objects.

shows the confusion matrix, together with the precision, accuracy and recall values for the classes from ShapeNet. The overall accuracy of the classifier is 81.9%.

We compare Table 6 to Table 7 and see that the performance of the classifier is higher for the real world data. This is in contrast to the expectation that real world data would be more difficult to classify since only parts of the model are visible. There are three possible explanations for this effect. The first one is that missing object parts also contain useful information for classification. The second one is that the models from ShapeNet vary greatly in shape and appearance, whereas models from S3DIS mostly are of one type, like office chairs, living room chairs, lecture hall chairs versus only office chairs. The third one is that the datasets differ in size and

**Classifier comparison. S3DIS-GT: 89.0% oA - S3DIS-RPN: 78.9% oA**

| | | window | door | table | chair | sofa | bookcase | board |
|---|---|---|---|---|---|---|---|---|
| S3DIS-GT | Recall | 0.673 | 1 | 0.955 | 0.922 | 0.545 | 0.843 | 0.81 |
| | Precision | 0.795 | 0.894 | 0.87 | 0.967 | 0.857 | 0.859 | 0.85 |
| | Accuracy | 0.97 | 0.983 | 0.966 | 0.967 | 0.993 | 0.926 | 0.984 |
| S3DIS-RPN | Recall | 0.385 | 0.470 | 0.718 | 0.921 | 0.078 | 0.791 | 0.130 |
| | Precision | 0.761 | 0.519 | 0.693 | 0.838 | 0.25 | 0.818 | 0.750 |
| | Accuracy | 0.978 | 0.963 | 0.895 | 0.884 | 0.970 | 0.896 | 0.993 |

**Table 8:** Classifier performance comparison for the two datasets S3DIS-GT and S3DIS-RPN.

number of objects per class. This experiment shows, that real world data does not harm the predictive power of our classifier network. In Table 6 we can see that the

**Classifier mean comparison on S3DIS-GT and S3DIS-RPN**

| | mAcc | mPrec | mRec | oA |
|---|---|---|---|---|
| S3DIS-GT | 0.970 | 0.870 | 0.821 | 0.894 |
| S3DIS-RPN | 0.940 | 0.661 | 0.499 | 0.789 |

**Table 9:** Classifier comparison between S3DIS-GT and S3DIS-RPN for the mean evaluation measures. The classifier achieves higher scores on the S3DIS-GT dataset. We believe that the performance drop is caused by less accurate ROIs.

tangent convolution classifier confuses around 10% of benches with chairs. Figure 26 (a) shows a table from ShapeNet, but from the appearance it could also be a bench without back rest. Similar confusions happen for the S3DIS data. Table 7 shows that boards and windows are very likely to be confused. Both are planar and nearly indistinguishable from walls. These confusions show that the classifier can actually reason about the appearance of the object and learns something about their structure.

Table 7 shows the confusion matrix, together with the recall values for the classes from S3DIS. We compare this in Table 8 with the classifier performance on the RPN data. The performance of the classifier on the RPN objects drops significantly for the classes board, window, sofa and door. This can be caused by the class imbalance in the data or due to the region proposal boxes that can include more points from the surrounding scene, like in Figure 27.

From Table 9 we conclude that the classifier is strong enough to learn the structure of objects and that it is able to classify them. However, on crops from the 3D point cloud the classifier performance drops. We believe that this is due to inaccurate boxes, which contain additional points of the surrounding scene or only parts of the object.

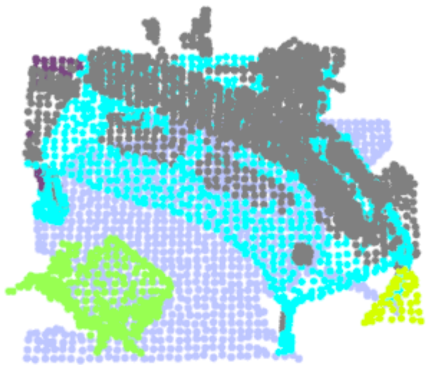


**Figure 27:** Proposed region for a table (cyan). The main direction of the table is not aligned with the coordinate system. Therefore the table cannot be extracted without the chair (green).

### 5.4.2 Tangent- vs 3D Convolutions

In the experiment in Subsection 5.4.1 we evaluated the tangent convolutions classifier (TC) performance on different data sets and show that it is able to distinguish between objects. In order to reason about the strength of tangent convolutions for the classification task, we compare them against 3D grid convolutions. Therefore, we implement a baseline classifier network (BL) that uses 3D grid convolutions. The architecture of the baseline network has the same number of convolution and pooling operations. Furthermore, we ensure pooling from the same volume size and that both networks operate on the same resolution of the point cloud. Hence no network can outperform the other due to a bigger receptive field. However, the input for the networks is different. Tangent convolutions operate on depth to the tangent image, whereas 3D grid convolutions operate on an 24 x 24 x 24 occupancy grid, with 7cm large voxels. Given the grid layout, we convert the point cloud into a voxel grid by assigning an *occupied* label to all voxels that contain a point, all remaining voxels are marked as *free*.

In Table 11 and Table 10 we compare the power of 3D grid convolutions to tangent convolutions. The results from the grid convolutions are 2% better than for the

**Classifier mean comparison for BL and TC**

| | mAcc | mPrec | mRec | oA |
|---|---|---|---|---|
| BL | **0.974** | 0.836 | **0.838** | **0.909** |
| TC | 0.970 | **0.870** | 0.821 | 0.894 |

**Table 10:** Classifier comparison between BL and TC on the mean evaluation measures. The performance difference between the two classifiers is small. The baseline method performs better.

tangent convolutions for the overall accuracy. Tangent convolutions achieve better recall values for doors and boards.

| method | window | door | table | chair | sofa | bookcase | board |
|---|---|---|---|---|---|---|---|
| baseline Recall | **0.692** | 0.984 | **0.968** | **0.938** | **0.636** | **0.885** | 0.762 |
| TC Recall | 0.673 | **1** | 0.955 | 0.922 | 0.545 | 0.843 | **0.81** |
| baseline Precision | 0.735 | **0.947** | **0.949** | **0.968** | 0.583 | **0.869** | 0.800 |
| TC Precision | **0.795** | 0.894 | 0.87 | 0.967 | **0.857** | 0.859 | **0.85** |
| baseline Accuracy | 0.966 | **0.99** | **0.985** | **0.972** | 0.99 | **0.937** | 0.979 |
| TC Accuracy | **0.97** | 0.983 | 0.966 | 0.967 | **0.993** | 0.926 | **0.984** |

**Table 11:** Comparison BL - TC on S3DIS. The performance of the two networks is similar. The baseline network achieves higher values across the three evaluation measures.

TCs operate directly on points and can pick up dense areas. This is more detailed than the grid information that 3D convolutions operate on. One possible explanation for the small improvement of 3D convolutions is that they overfit less than TC during training and therefore generalizes better for the validation set. This also explains why tangent convolutions perform better on the classes that show little structure, like board and door.

All in all we can summarize in this experiment that the tangent convolutions classifier performs well on the S3DIS-GT dataset, but not better than the baseline method with 3D convolutions. The performance of both classifier networks is very similar. Therefore both can be used for the object detection task.

## 5.5 ScanNet Benchmark Evaluation

ScanNet makes 1513 fully annotated 3D scans publicly available. Another 100 scans are provided without ground truth data for instance segmentation. Predictions for these 100 scans can be submitted to their benchmark website[1]. Public submissions are ranked in a leader board by the mean average precision (mAP) for three overlap values on points, see Section 5.2. Additionally the average precision per class is provided. The classes *floor* and *wall* are excluded for instance tasks. We test the performance of our 3D object detector on the 3D Semantic instance benchmark. Furthermore the evaluation script is available on github, so that we can also evaluate on the validation set[2].

### 5.5.1 Input Signal Configurations

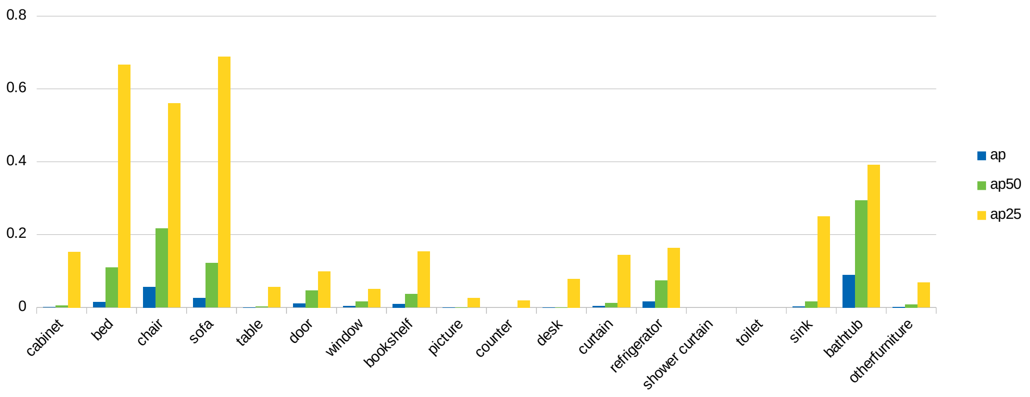


**Figure 28:** ScanNet validation for depth as input channel. The plot shows the detection scores for AP, AP at 50% IoU and AP at 25% IoU. The object detector performs better for the classes bed, chair, and sofa than for the classes picture, counter, shower curtain, and toilet. The latter are not detected at all.

Before submitting to the leader board of the benchmark, we compare the performance of our 3D object detector for two input and parameter settings on the validation set. For the first setting, we use depth as the only input for the network. Depth in the tangent convolutions case means depth to the tangent image plane as described in more detail in Section 3.2. For the second setting, we use depth, height,

[1]`http://kaldir.vc.in.tum.de/scannet_benchmark/semantic_instance_3d`
[2]`https://github.com/ScanNet/ScanNet/tree/master/BenchmarkScripts/`

normals and color information as input. Additionally, we lower the tolerated overlap of boxes in the non maximum suppression (NMS) post processing from 20% to 5%. The NMS lowers the number of object predictions per location, which reduces the penalty in the average precision evaluation metric for false positives.

In Figure 28 we see the average precision (AP) values itemized per class and per overlap value. The higher the overlap value, the more restrictive it is. It represents the IoU on points threshold, for which the detections are considered to be a true positive. We achieve the highest scores for the classes bed, chair, sofa and bathtub. In general the AP drops drastically for more restrictive IoU values. Shower curtain and toilet are not predicted. We believe that this is due to a class imbalance in the classifier training data. The box size of toilets is very similar to chairs and shower curtains are similar to curtains. Following the imbalance, the classifier did not learn to distinguish between chairs and toilets and predicts always chairs in this case. Our ablation study in Subsection 5.3.4 shows that the RPN captures all objects and achieves similar average precision scores. The object detector in this experiment misses shower curtains. This indicates that the object detector predictions are a result of imbalanced classifier predictions. Figure 29 shows the evaluation of the

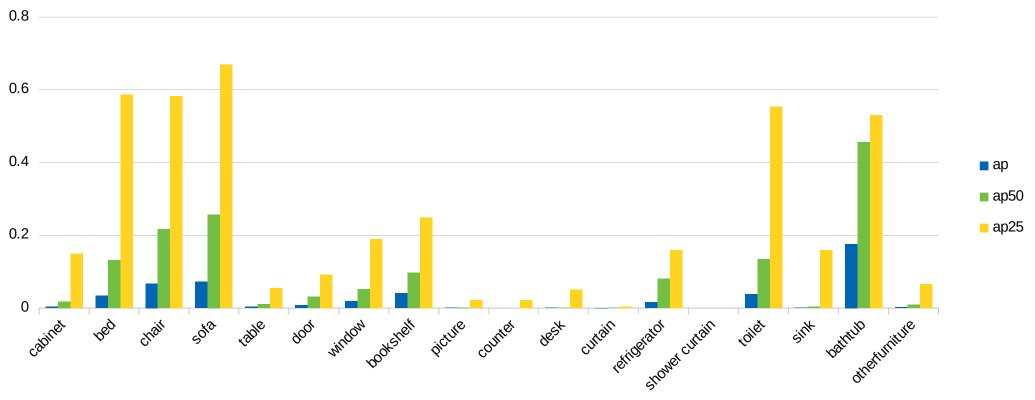


**Figure 29:** ScanNet validation for depth, height, normals and color as input channels. The plot shows the detection scores for AP, AP at 50% IoU and AP at 25% IoU. Overall this detector achieves higher detection scores than the one that uses only depth. The detection score for toilets increased drastically.

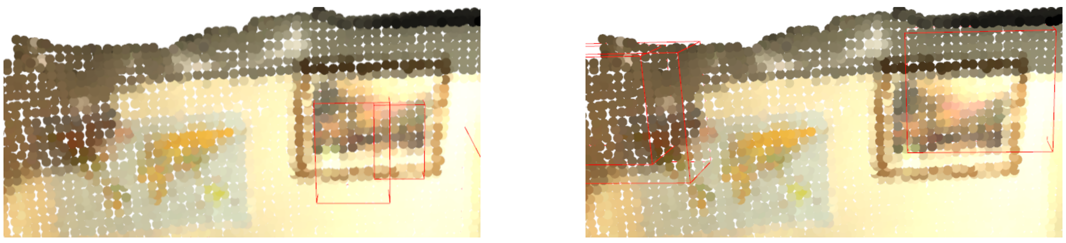
second setting on the validation set. The prediction is still strong for the classes bed, chair, sofa and bathtub. In contrast to Figure 28, curtain predictions decrease and toilet predictions increase drastically. Our explanation for this effect is that with color input the mostly white toilets are now better distinguishable from black chairs. From the evaluation values in Table 12 we can see that on average the performance

| | D | DHNC |
|---|---|---|
| mean AP 25% | 0.198 | 0.230 |
| mean AP 50% | 0.053 | 0.083 |
| mean AP | 0.013 | 0.027 |

**Table 12:** Object Detector Performance Comparison for the two different input signal settings. We compare the performance for only depth (D) against depth, height, normal, color (DHNC) based on the Mean Average Precision score. More input signals improve the performance of the object detector by a factor of two for mAP.

of the object detector improved for more input channels and a stronger NMS setting.

Conclusion: as expected the network with more inputs performs better. In some cases, the color input per point helps identifying objects, like toilets and pictures. In Figure 30 the detection in image (b) improves noticeably compared to the two small detection boxes in image (a).



**(a)** D

**(b)** DHNC

**Figure 30:** (a) Detection using depth only. (b) Detection using all inputs. The picture is localized better in (b).

### 5.5.2 Submission to Leader Board

We submit our method with the settings from subsection 5.5.1 to the ScanNet Benchmark and compare it to three other methods in Figure 31. Mask-RCNN Projection is the baseline method that we mainly compare against. This method runs Mask RCNN [13] on images and projects the instance labels onto the 3D point cloud. The other methods, Similarity Matrix based segmentation (SGPN) and Multi-Task Metric Learning (MTML), are anonymous submissions. Whereas SGPN and

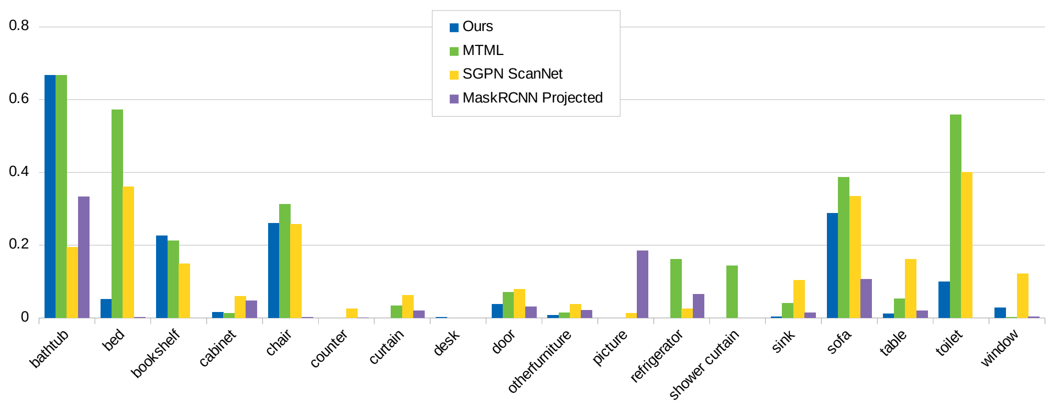


**Figure 31:** Submission to the leader board of ScanNet. This plot shows the results for AP with 50% point overlap. Our method uses only 3D information.

MaskRCNN make use of RGB images, MTML and our method use only the 3D point cloud. Table 13 shows that we are better than the baseline method and can almost double the mAP values for high overlaps. SGPN performs less than 1% better than our method does for the mean AP. The performance difference increases to around 4% for mean AP 50%. MTML is the strongest method in this evaluation and performs around two times better than our method for mean AP and mean AP 50%. In Figure 31 SGPN ScanNet has the most balanced predictions. It does not achieve the highest overall values, but performs good for all classes. Our method performs best for the classes bookshelf, bathtub and desk, and is competitive for cabinet, chair, door, other furniture, sofa and window. So for these classes the performance difference between MTML and our method is not that big. Unfortunately our method misses the classes counter, curtain, picture, refrigerator and shower curtain completely, which restricts our overall performance. Note that our method is an object detector and

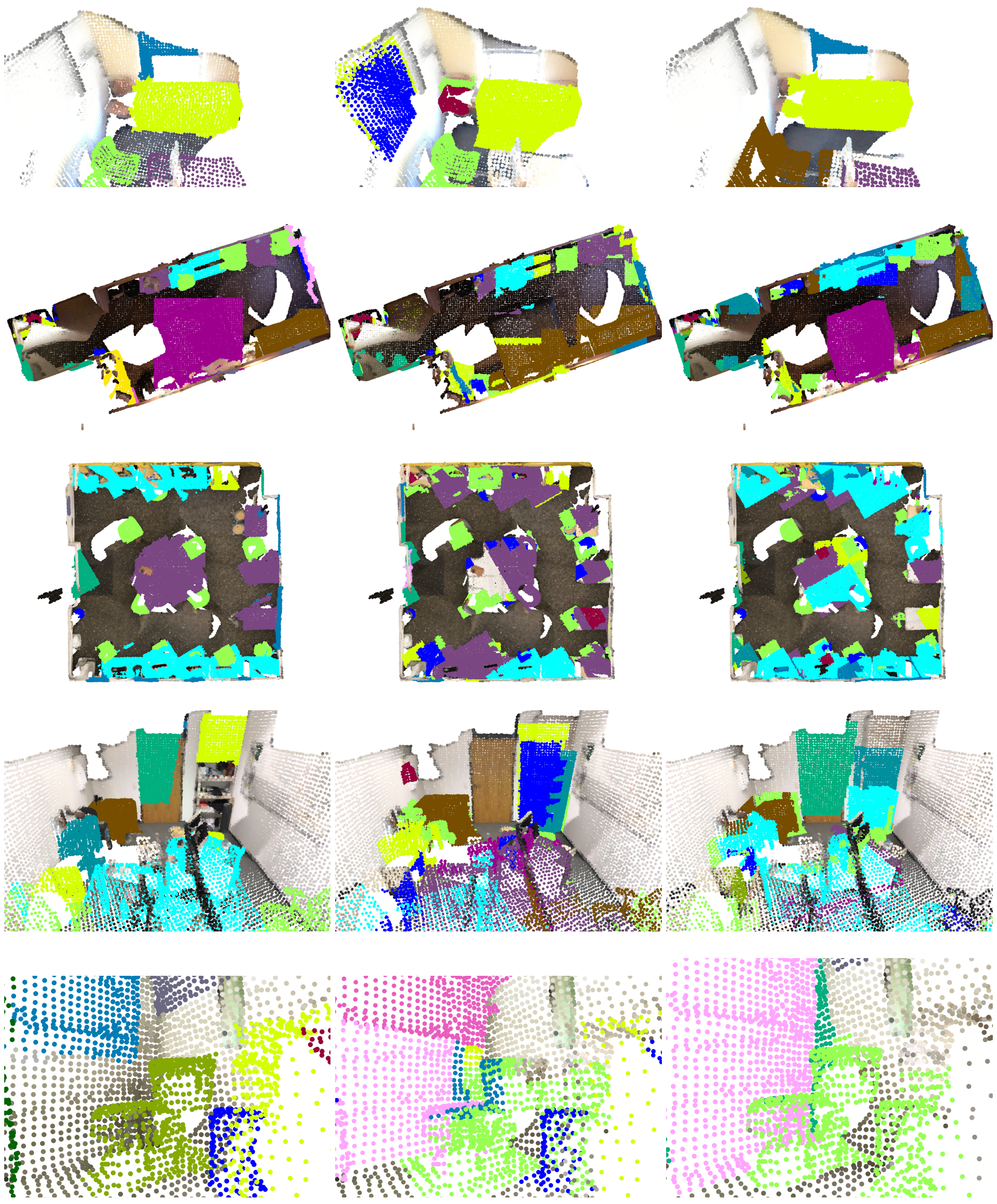
| | Ours | SGPN ScanNet | MTML | MaskRCNN Projected |
|---|---|---|---|---|
| mean AP 25% | 0.203 | 0.317 | 0.243 | 0.227 |
| mean AP 50% | 0.094 | 0.133 | 0.180 | 0.053 |
| mean AP | 0.043 | 0.047 | 0.086 | 0.021 |

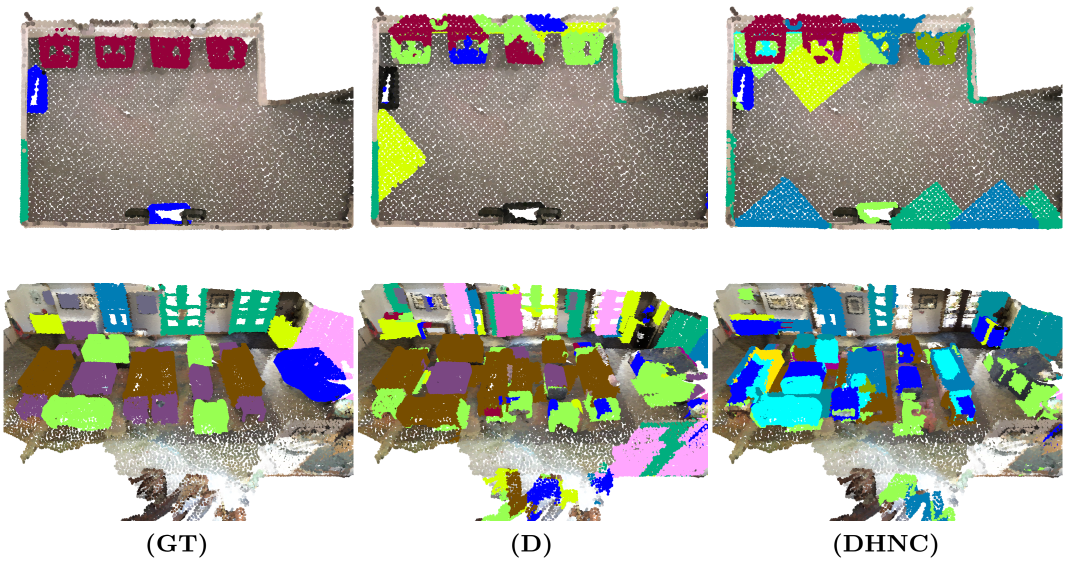
**Table 13:** We compare our method to the methods on the ScanNet leader board based on the mean AP scores. For mAP 50% and mAP we achieve higher scores than the Mask R-CNN Projected method.

assigns an instance label to all points within the predicted box. In cases where a box is the wrong tool to contain points of an instance, see Figure 27, we introduce a rather large error. With a binary mask prediction per box and a more balanced prediction for all classes, our method will become even more competitive.

### 5.5.3 Qualitative Results for ScanNet

In Figure 32 we show one qualitative example per row. The left image shows the ground truth, the image in the middle shows the result of our method for depth to the tangent plane (D) as input and the image on the right shows the result for multiple input channels. These are depth, height, normals and color (DHNC). The first four rows show successful detections for either D or DHNC. It is also visible in these pictures that our method confuses some objects, like two chairs next to each other as couch (first row DHNC), and table and desk (third row, DHNC) In many cases DHNC performs better than D, as can be seen in row four, where we get a good detection for a door. Also note that the ground truth labelling is not always complete. The door in the fourth row is only partially labelled. The fifth row shows a case where our method confuses a toilet with a chair. Row six and seven are examples where our method does not perform well and where more input seems to be harmful. This is the case for the bathroom in row six, where we get many false positive predictions at the bottom wall. Row seven shows a difficult room with many objects. These range from 2 meter wide sofas to small, 40 centimeter high pictures. Furthermore we see in this example that sofas (brown) are correctly detected for D, but not for DHNC.





● Cabinet ● Bed ● Chair ● Sofa ● Table ● Door ● Window ● Bookshelf ● Picture ● Counter
● Desk ● Curtain ● Fridge ● Shower curtain ● Toilet ● Sink ● Bathtub ● Other furniture

**Figure 32:** Qualitative results for ScanNet: The images on the left show the ground truth (GT). The center images show the predictions of the object detector that uses depth (D) as input, and the images on the right show the results for (DHNC) depth, height, normal, color inputs.

## 5.6 Evaluation for S3DIS

In this experiment we train and evaluate our object detector on the S3DIS data set, without any parameter adjustments. We evaluate on the same measures that are used in the ScanNet benchmark.

| | window | door | table | chair | sofa | bookcase | board | **mean** |
|---|---|---|---|---|---|---|---|---|
| AP 25% | 0.187 | 0.039 | 0.241 | 0.678 | 0.008 | 0.211 | 0.001 | **0.195** |
| AP 50% | 0.102 | 0.011 | 0.025 | 0.325 | 0.000 | 0.046 | 0.000 | **0.073** |
| AP | 0.025 | 0.002 | 0.006 | 0.113 | 0.000 | 0.015 | 0.000 | **0.023** |

**Table 14:** Object detector results on S3DIS. Our method can detect chairs the best and boards the least.

Table 14 shows the results of the object detector per class, together with the mean AP over all classes. Our method can detect chairs the best and boards the worst.

Figure 33 visualizes the results per class. In comparison to Figure 29 the performance is similar for all classes except sofa. We achieve similar mean average precision values on S3DIS and on ScanNet. This proves that our object detector does not overfit to one data set. It can be used without parameter and anchor adjustments across different data sets.

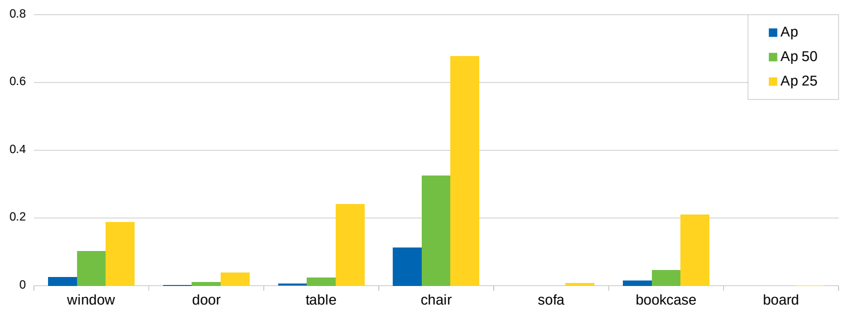
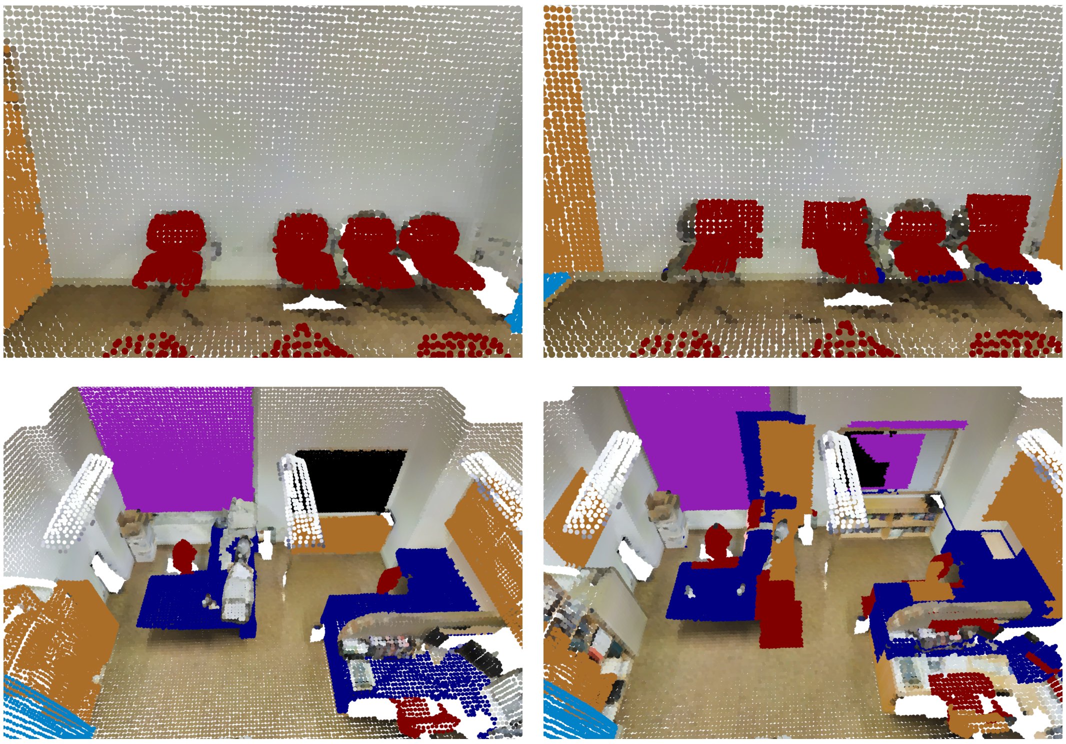


**Figure 33:** Stanford Validation DHNC.

In Figure 34 we show one qualitative example per row. The image on the left shows the ground truth. The image on the right shows the predictions. In total we present six qualitative examples, one in each row. As Figure 33 highlights, we get the best detections for chairs. The right image in row one shows that our method correctly identifies the three chairs in the front and four chairs in the back. In many cases the detection box contains parts of neighboring surfaces, points that belong



to the wall but are labeled as chair illustrate this. Example two is an office scene that contains many objects. The only object that our method misses is the bookshelf below the board in the back right corner of the room. All other object types are captured correctly, e.g., the door in the bottom left, the window and the chairs and desks. The office rooms in S3DIS comprise large bookshelves. The standard anchor boxes that we define in Figure 14 are not big- and the refinements are not strong enough to cover such large objects as shown in row three. On this account our method identifies several bookshelves instead of one at the back wall in row three. As we found out in the ablation study in Subsection 5.3.4, the region proposal network outputs many regions of interest. Row five and six show qualitative results of false positive detections in the scene. For the bathroom in row five our method wrongly detects the basin as table and bookshelves on the wall. In row six we falsely detect a window on the left wall.

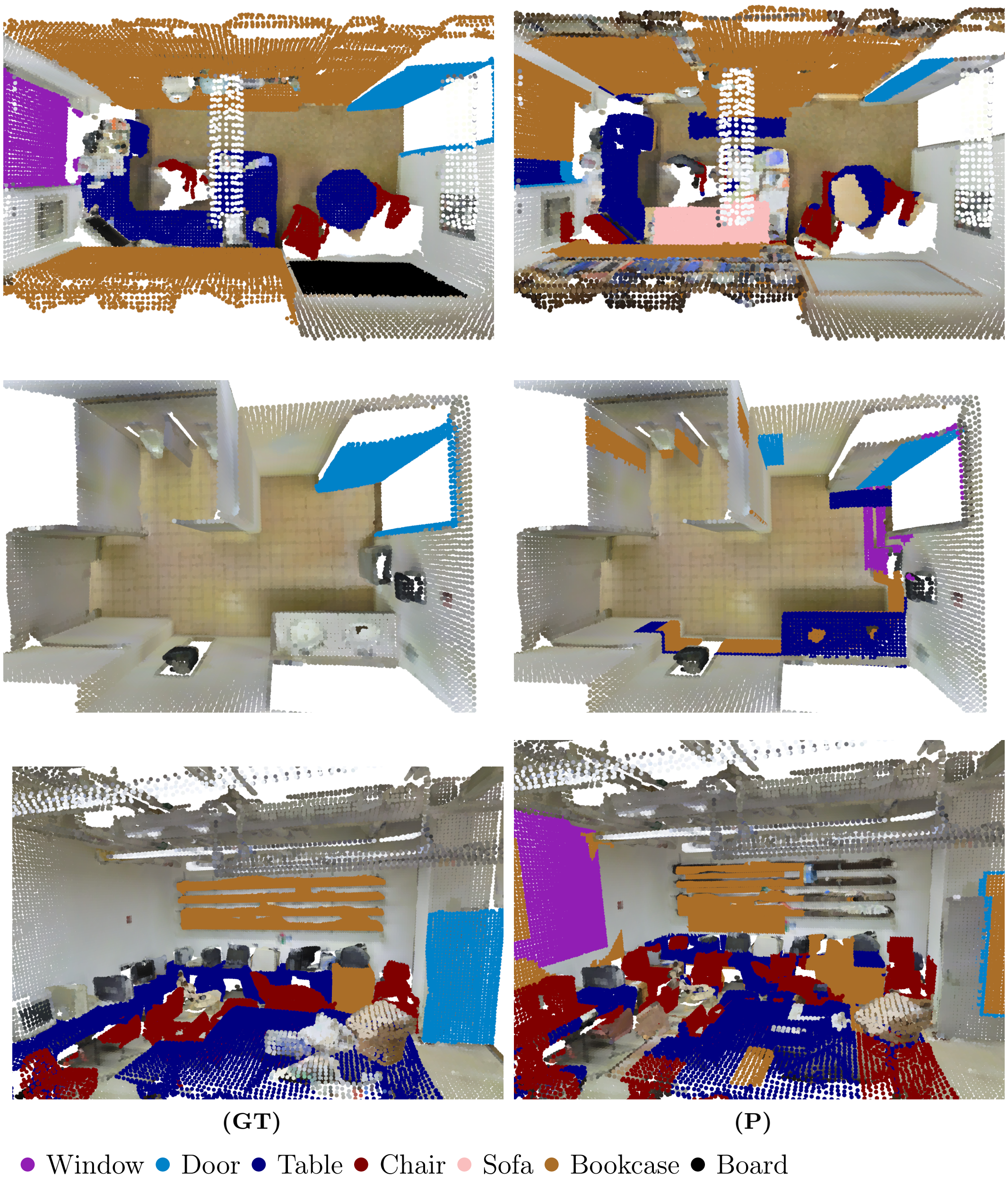


**Figure 34:** Qualitative results for S3DIS. The images on the left show the ground truth. The images on the right show the prediction.

# 6 Conclusion and Future Work

## 6.1 Conclusion

We have presented a new convolutional neural network for object detection in 3D point clouds. Our architecture is based on Faster R-CNN and uses crucial functionality to deal with 3D point clouds, such as 3D anchor design and the tangent convolution operation as the main building block. The latter, instead of 3D grid convolutions, ensures efficient and accurate processing of the point cloud. In contrast to other methods, which rely on additional 2D images of the 3D scene, our method uses only 3D information.

Our experiments on the ScanNet benchmark and Stanford dataset show that our method is able to detect objects at various scales, sizes and positions large scenes. On the ScanNet instance segmentation leader board we achieve a higher mean Average Precision than the baseline method *Mask R-CNN 3D Projections* for IoU values higher than 25%. The comparison on the leader board of the ScanNet benchmark also shows that our method is competitive with other approaches operating in the same mode. Our ablation studies show that the region proposal network is not sensitive to the anchor design and that it detects all object types. They further show, that the classifier network, which is trained on the RPN proposals, suffers from the uneven number of objects per class.

## 6.2 Future Work

In the following we suggest extensions to our method that we believe will improve the overall performance.

As we have shown in Figure 13, by the current design of the bounding boxes, objects are not enclosed by tight boxes, depending on the rotation of the object relative to the coordinate system. We believe that this limits the RPNs refinement prediction strength, since the bounding box shape and the object are not strongly related in this case. In order to make the objects box shape independent of the relative orientation to the grid (tight bounding box at all rotations), we suggest to extract this information from the point cloud and make the RPN predict the orientation of the box in addition to the refinement values.

In order to evaluate our object detector on an instance segmentation benchmark, we assign all points within the box an instance class label and do not distinguish between fore- and background. Especially boxes of big objects like tables and sofas overlap with other objects and thus the IoU over points decreases. We expect that adding a binary segmentation branch to our network architecture, following the Mask R-CNN rolemodel, improves our detection results on the ScanNet leader board.

Our object detection method uses the encoder part of the U-net proposed by Tatarchenko et al. [4] for feature extraction in the RPN and the classifier network. This is a rather small network with two pooling- and six convolution layers. In comparison, feature extraction networks for images, like VGG [17] with 19- and ResNet [18] with 34-convolutional layers are much deeper. He et al. [13] found that deeper backbone networks improve the overall performance. We expect the same effect for object detection in 3D.

Furthermore, Ren et al. [12] initialize their feature extraction networks with weights that have been trained on ImageNet. We initialize our networks with random weights. Using generic, pre-trained weights for 3D data is very likely to improve training and generalization of our method.

Throughout this work we tested only a few parameter settings for our approach. A structured search of better hyperparameters is one more option to improve the performance of our method.

# Bibliography

[1] G. Riegler, A. O. Ulusoy, and A. Geiger, “Octnet: Learning deep 3d representations at high resolutions,” *CoRR*, 2016.

[2] M. Simonovsky and N. Komodakis, “Dynamic edge-conditioned filters in convolutional neural networks on graphs,” in *CVPR*, 2017.

[3] S. Wang, S. Suo, W. Ma, A. Pokrovsky, and R. Urtasun, “Deep parametric continuous convolutional neural networks,” in *CVPR*, 2018.

[4] M. Tatarchenko*, J. Park*, V. Koltun, and Q.-Y. Zhou., “Tangent convolutions for dense prediction in 3D,” in *CVPR*, 2018.

[5] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner, “Scannet: Richly-annotated 3d reconstructions of indoor scenes,” in *CVPR*, 2017.

[6] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. K. Brilakis, M. Fischer, and S. Savarese, “3d semantic parsing of large-scale indoor spaces,” in *CVPR*, 2016.

[7] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *CVPR*, 2005.

[8] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *CVPR*, 2014.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *NIPS*, 2012.

[10] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, “Selective search for object recognition,” *International Journal of Computer Vision*, 2013.

[11] R. B. Girshick, “Fast R-CNN,” in *ICCV*, pp. 1440–1448, 2015.

[12] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” in *NIPS*, 2015.

[13] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask r-cnn," in *ICCV*, 2017.

[14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *CVPR*, 2017.

[15] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," in *ECCV*, 2016.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2014.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[19] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *IROS*, 2015.

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

[21] P. Wang, Y. Liu, Y. Guo, C. Sun, and X. Tong, "O-CNN: octree-based convolutional neural networks for 3d shape analysis," *CoRR*, 2017.

[22] R. Klokov and V. S. Lempitsky, "Escape from cells: Deep kd-networks for the recognition of 3d point cloud models," *CoRR*, 2017.

[23] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017.

[24] S. Gupta, P. A. Arbeláez, R. B. Girshick, and J. Malik, "Indoor scene understanding with RGB-D images: Bottom-up segmentation, object detection and semantic segmentation," *International Journal of Computer Vision (IJCV)*, 2015.

[25] D. Lang, S. Friedmann, and D. Paulus, "Semantic 3d octree maps based on conditional random fields," in *International Conference on Machine Vision Applications (IAPR)*, 2013.

[26] S. Song and J. Xiao, "Deep Sliding Shapes for amodal 3D object detection in RGB-D images," in *CVPR*, 2016.

[27] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from RGB-D data," in *CVPR*, 2018.

[28] B. Yang, W. Luo, and R. Urtasun, "PIXOR: real-time 3d object detection from point clouds," in *CVPR*, 2018.

[29] W. Wang, R. Yu, Q. Huang, and U. Neumann, "SGPN: similarity group proposal network for 3d point cloud instance segmentation," in *CVPR*, 2018.

[30] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *NIPS*, 2017.

[31] V. Tananaev, "Semantic segmentation in point clouds with deep networks," Master's thesis, Albert-Ludwigs-Universität Freiburg, 2017.

[32] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI*, 2015.

[33] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet: An information-rich 3d model repository," *CoRR*, 2015.