NPM3D final project : Deep Hough Voting for 3D Object Detection in Point Clouds

Loïck Chambon
MVA - ENS Paris Saclay
loick.chambon@eleves.enpc.fr

Abstract

Automatic modeling and segmentation is a major application in 3D computer vision. There are two classical main families of methods, those using surface propagation and those using surface extraction by voting. The Hough method belongs to the second category. Its principle is to perform a vote in the parameter space rather than in the point space. With the arrival of neural networks, the voting principle has been extended and used to predict bounding box for scene points in the context of 3D object detection. In this project we will study all the pipeline of a new neural network [12] whose originality is to perform 3D bounding box estimation using only 3D point clouds and no external 2D data.

1. Introduction

3D modeling and segmentation has many practical applications: in architecture with the construction of 3D cities or in industry with the maintenance of power plants and factories. Typically, the input to segmentation models is a point cloud collected by Lidar, extracted from RGB-D images, or sampled on reconstructed meshes. Then the data is processed by algorithms and networks.
Classical methods for 3D modeling use either propagation or voting to extract surfaces from the point cloud. We have seen in course the RANSAC algorithm and the Hough transform. RANSAC is a method of voting on random samples of quorum points that model the surface by a minimal number of points, while the Hough transform exploits the duality between the points on a curve and the parameters of that curve [6]. It has been generalized to non parametric curves in [1] but it is not suited for 3D data and complex shapes.
Recently, deep learning methods have emerged and tackled a related problem: 3D object detection, which has many applications in augmented reality, robotics and autonomous driving. The objective is to surround objects on scenes using bounding boxes despite an often partial representation of the objects. Choosing the rights bounding boxes is a difficult task since there is a large search space, varying sizes and orientations. As a first approach, the methods have tried to extend the 2D to 3D detection frameworks using 3D convolutions, a bird's eye viewpoint [2] or a reduction in search space performed by a 2D detectors to guide the search [9]. Some are well suited for real time inference and autonomous driving. However, most of them suffer from data scarcity, computational cost, are limited to certain types of scenes and use external data such as multiple RGB-D images. Even if 2D images can bring useful information, in 3D computer vision, they are not the most natural data to use. To capture the geometry of the scene, we often prefer to work with voxels or point clouds. Point clouds have the advantage of representing accurate 3D geometry and being robust to illumination, but they imply high computational costs, they are often scarce and noisy. The scarcity of the data is a major problem due to the way point clouds are collected because the cheapest way to collect point cloud is to use depth sensors that only capture surfaces of objects, so 3D object centers are likely to be in empty space, far away from any point.
Then, in 2019, R. Qi et al. [12] successfully used a raw point cloud without any external information and 2D segmentation algorithm to segment objects in complex scenes and accurately predict the bounding boxes of objects. To deal with the scarcity of the data, they have adapted the Hough voting using features calculated via a PointNet++ backbone network [13] that enrich points with more features.

2. Method

Deep Hough voting [12] relies on two main concepts: the backbone neural network PointNet++ [13] used to enrich points with external features and the Hough voting used to select important points. This allows the network to generate new points called the *seeds* that lie close to objects centers which can be far from collected points. Then, the seeds can be grouped and aggregated to generate box proposals.

2.1. PointNet++

The network is an improvement of PointNet [11] whose we have seen the architecture in course and implemented during the last TP. The idea is to approximate a general function $f : 2^{\mathbb{R}^N} \to \mathbb{R}$ defined on a point set S by applying a symmetric function $g : \mathbb{R}^K \times ... \times \mathbb{R}^K \to \mathbb{R}$ on transformed elements $x \in S$ in the set by applying $h : \mathbb{R}^N \to \mathbb{R}^K$.

$$f(\{x_1, ..., x_n\}) \sim g(h(x_1), ..., h(x_n))$$

They propose to represent h by an MLP and g by a composition of a single variable function and a max pooling function.

This setting is justified by an Universal Approximation Theorem (see. Section 4.3 in [11]). In g, symmetry is important because the points in the point clouds are not ordered and must be treated independently of their order.
PointNet has drawbacks. It fails to recognize fine-grained patterns, to capture local-context and needs and takes always the same sub-sampled low number of points as input (1024 points). PointNet++ [13] leverages some drawbacks by applying PointNet recursively on a nested partitioning of the input point cloud.

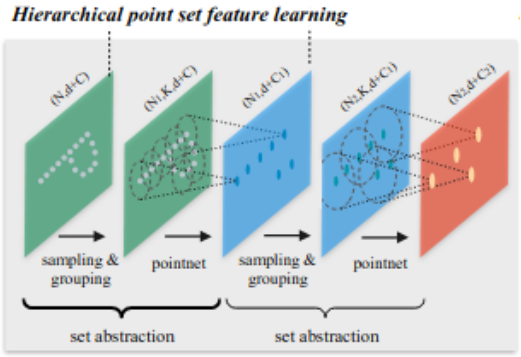


Figure 1: Hierarchical point set feature learning of PointNet++

The architecture (Fig. 1) relies on the set abstraction layers which processes the N input points to create M points enriched with features. It is composed of three submodules: the sampling layer, the grouping layer and the PointNet layer. Since they are used in deep Hough voting, we will briefly describe them.

- *Sampling layer:* selects a set of points from input points, which defines the centroids of local regions. For selection, it uses Farthest point sampling (FPS) because it catches points from less dense regions and covers a better region than random sampling [1]. On Fig. 1 centroids are the dots in grey on the green left image.
- *Grouping layer:* constructs local region sets by finding "neighboring" points around the centroids. It uses Ball Query to get a maximum of K points around each centroid inside a sphere of a certain predefined radius. Ball query is preferred to KNN because with KNN, the sparsity of the data can lead to match a point with a neighbour locally far from it. It is represented by circles on the second green image of Fig. 1.
- *(mini) PointNet layer:* encodes local region patterns into feature vectors. It contains a shared MLP network with a series of convolution, normalization, Relu layers followed by a max pooling layer. Before feeding PointNet, the coordinates of the points in the local regions are expressed in the local referential. This allows the network to use MLP with shared parameters independently of the global coordinates.

[1] https://minibatchai.com/ai/2021/08/07/FPS.html

The use of multiple set abstraction layers allows PointNet++ to work with the same input data at different scales and to catch different levels of context. But, in set abstraction layer, the point set is sub-sampled and we want to obtain features for all the points. One solution is to always sample all points as centroids in all set abstraction levels, but it is computationally expensive so they have decided to add a *feature propagation layer* that propagates features from sub-sampled points to the original points. In practice they have adopted a hierarchical propagation strategy with distance based interpolation and across level skip connections.

2.2. Hough voting

To understand how Hough voting works, let us recall the 2D pipeline of Hough voting (Fig. 2), the second important principle of the network. It is composed of an online step and an offline step. In the offline step, we learn a codebook that stores the correspondence between image patches and their offset from the centers of the corresponding objects. In the online step, given a 2D image, we perform a selection of points of interest using, for example, the Harris corner detector, SIFT, ORB or SURF. Then we match the patch around the points of interest to a training patch in the codebook. After that, each match tells us the offset of the point to reach the center of the object, so we translate our point to produce a vote. Finally, we apply clustering to find the dense regions and find the patches that voted for the dense regions by back-projection before drawing bounding boxes using the coordinates of the selected patches.

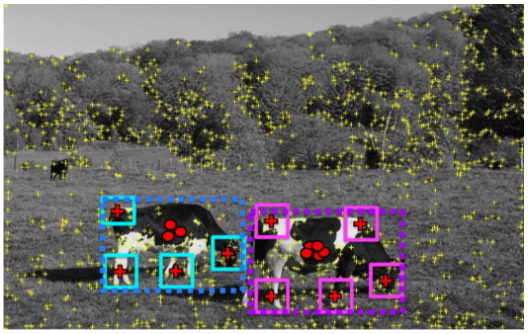


Figure 2: Illustration of Hough voting. Yellow cross represent interest points. Red circles represent the remaining votes after clustering on objects center. Squares are the patches that voted for the dense regions. Dots represent the object bounding box proposal. (*Credits from U.Toronte, CSC420, 2015, slides 57/91.*)

Hough voting relies only on interest points and can aggregate points from long range into a virtual center. That is why it is well suited for sparse data such as point clouds. But, this formulation is non differentiable. Thus, it can not be optimized using a neural network.

2.3. Deep Hough voting

The 3D pipeline (Fig. 3) is really close to the one presented before. To make it differentiable, they did some ad-

justments: interest points relies on a neural network instead of handcrafted features, the codebook is replaced by a Voting module, the cluster selection is realized using a sampling strategy and a module with trainable parameters, and bounding boxes are generated directly from the votes features without back-tracing the votes.

2.3.1 Model

VoteNet takes as input a collection of scenes represented by a raw point cloud of size $(B, N, 3)$ composed of xyz coordinates $x_i \in \mathbb{R}^3$. Where, B is the batch size, N is the number of points in a batch and 3 corresponds to the number of coordinates. Each scene is augmented on the fly using data augmentation such as random point selection, flipping and rotation. They have also add optional extras features such as: a height feature corresponding to the distance to the floor $z - z_{floor}$, estimated as the 1% percentile of all points' heights, and RGB color features. For clarity of the explanation, we do not use them afterwards. Then, after pre-processing, the data is processed by three networks (see steps Fig. 3).

- *Backbone network:* takes as input the batch of points $(B, N, 3)$ and outputs a batch of M sub-sampled points (e.g 1024) enriched with C (e.g 256) features f_i leading to a batch of size $(B, M, 3+C)$. The enriched points are called the seeds. The architecture relies on PoinNet++ to create features. It is composed of four set abstraction layers and two feature propagation layers. Each set abstraction layers has a receptive field parametrized by a ball-region radius. The deeper we are, the higher is the radius.

- *Voting module:* processes existing points to generate votes. It is a MLP with ReLU and batch-normalization taking as input the seeds: $s_i = [x_i; f_i]$, $(B, M, 3 + C)$ and that outputs a space and a feature offset $\Delta s_i = [\Delta x_i; \Delta f_i]$, $(B, M \cdot V, 3 + C)$ where V is the number of votes per seed, such that the vote can be written as: $v_i = s_i + \Delta s_i$. In the appendix of the paper, they have showed that $V = 1$ leads to the best results. After being added to the seeds, the offsets allow to obtain the coordinates and the features of the votes. Then, it takes as inputs, the $M \cdot V$ vote proposals of the previous module and samples K votes using farthest point sampling strategy on coordinates (without using features). They are the K cluster centroids. After that, it assigns the neighbouring votes of each centroid to the corresponding cluster, resulting in K clusters.

- *Proposal module:* aggregates the feature votes and produces a bounding box proposal. The architecture is a set of abstraction layer followed by another MLP after the max-pooling in each local region. In details, after the voting module, the networks returns clusters that contains a center point $w_j = [x_j, f_j]$ and neighbours $(w_i)_i$. They begin to represent neighbours in the local referential: $x_i^{'} = (x_i - x_j)/r$ where r is a radius. Then, they use an extremum selection calculate via a (mini) PointNet module:

$$\text{MLP}_2\Big(\max_{i=1,\dots,n} \{\text{MLP}_1([x_i^{'}, f_i])\}\Big)$$

  At the end of the module, the network outputs a multidimensional vector containing 3D oriented bounding box parameters (center, heading and scale), objectness scores and semantic scores. The output has $(5 + 2NH + 4NS + NC)$ channels, where NH is the number of heading bins, NS the number of box anchors and NC the number of semantic classification (see box loss for more details about NH and NS). 5 corresponds to 2 objectness scores and 3 center regression values, 2 NH corresponds to one classification score and one regression offset (only the azimuth is considered) and 4 NS corresponds to one classification score and three regression offsets.

During inference, a non maximum suppression module is used with an IoU threshold of 0.25 or 0.50 to filter box proposals.

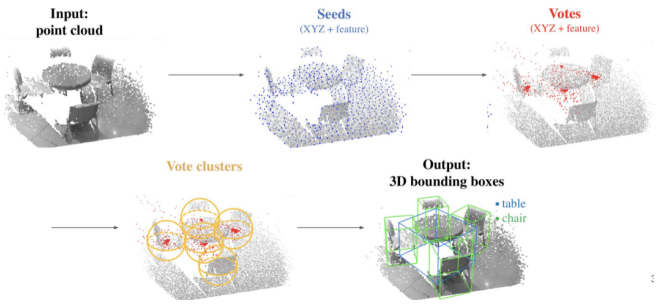


Figure 3: Evolution of the data through the Deep Hough Voting pipeline: raw data, after backbone network, after voting network, after vote clustering strategy and after the proposal module.

2.3.2 Loss

VoteNet is trained end-to-end with a multi-task complex loss composed of four terms.

$$\mathcal{L}_{votenet} = \mathcal{L}_{vote} + \lambda_1.\mathcal{L}_{obj-cls} + \lambda_2.\mathcal{L}_{box} + \lambda_3.\mathcal{L}_{sem-cls}$$

In the paper the authors took $\lambda_1 = 0.5, \lambda_2 = 1, \lambda_3 = 0.1$. If we want to be more precise on the bounding box or the classification loss, we can change the weights.

- $\mathcal{L}_{vote}$ is the vote loss that compared predicted votes to ground-truth votes. Intuitively, if a seed point $s_i = [x_i; f_i]$ belongs to an object, it has to vote for the object center $x_i + \Delta x_i^*$, where the delta term is the offset from the seed to the center of the object. Hence, this loss

incites the network to predict votes that are close to the object centers.

$$\mathcal{L}_{vote} = \frac{1}{M_{pos}} \sum_i ||\Delta x_i - \Delta x_i^*|| \mathbb{1}[s_i \in M_o]$$

Where M_o is the count of total number of seeds on object surface and $\mathbb{1}[s_i \in M_o]$ indicates if the seed belongs to the object or not.

- $\mathcal{L}_{obj-cls}$ is the objectness loss defined as a supervised weighted binary cross-entropy loss. The aim is to tell whether the predicted centers correspond to real object centers or not and to filter outliers. In practice, it incites the network to give high positive objectness scores when the predicted center is near a ground truth center and to assign false objectness scores when the predicted center is far from any real object center. It has three different behaviours ($0 < d_0 < d_1$) depending on the distance from the votes to the predicted center.
  - If the predicted center has a distance lower than d_0, then the label is positive.
  - If the distance is between d_0 and d_1, then the prediction is ignored.
  - Otherwise, the predicted center is far from any real object center and the corresponding label is negative.

  In their implementation, they have chosen $d_0 = 0.3$, $d_1 = 0.6$, $w_0 = 0.8$, $w_1 = 0.2$.
- $\mathcal{L}_{box}$ is the 3D bounding box estimation loss. According to [8] 3D bounding box are parametrized by their centers, their size and their orientation (due to the nature of our data, the azimuth is sufficient to represent the angle). Thus, the loss is composed of center regression, heading estimation and size estimation sub-losses as in [9]. Note that it is only applied on the positive proposals ($< d_0$) of the objectness loss.

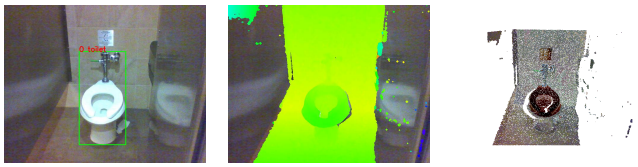
$$\mathcal{L}_{box} = \mathcal{L}_{center-reg} + 0.1 \cdot \mathcal{L}_{angle-cls} + \mathcal{L}_{angle-reg} + 0.1 \cdot \mathcal{L}_{size-cls} + \mathcal{L}_{size-reg}$$

  - $\mathcal{L}_{center-reg}$ is the Chamfer distance between centers proposals and ground truth centers.
  - $\mathcal{L}_{angle}$ is a hybrid classification (cross-entropy) and regression (Huber losses) losses. To do this, they discretize the orientation of the 3D space into heading bins and want the predicted angle to belong to the same bins as the actual angle. The residual rotation is simply the correction that needs to be applied to the orientation of the center ray of that bin in order to obtain the output angle.
  - $\mathcal{L}_{size}$ is similar to the angle loss but aims to find the right size of the bounding box. The number of scaling bins is set to be the number of object classes.
- $\mathcal{L}_{sem-cls}$ is the semantic classification loss which is a cross-entropy loss over all the classes using only the positive proposals. It aims at classifying the objects between all the different classes NC.

3. Experiments

3.1. Datasets

For our experimentations we have used SUN RGB-D [14] and ScanNet [4]. SUN RGB-D is an amodal dataset containing 10k real RGB-D images of room scenes with dense 2D and 3D annotations for both objects and rooms. The dataset is captured by 4 different RGB-D sensors. For our experiments, we have only used the two different versions of Kinect, namely v1 and v2. Each RGB image has a corresponding depth and segmentation map (Fig. **??**). The original dataset contains 47 scene categories and about 800 object categories while the version of the dataset used in Deep Hough voting contains only the 10 most common objects categories ($NC = 10$). The classes are unbalanced and chairs and tables are the most represented categories. The training and test sets contain both 5k images with 20k extracted points per image.
About the hyper-parameters, the authors have chosen: $NH = 12$ and $NS = NC = 10$.



(a) RGB image. (b) Depth image. (c) Point cloud.

Figure 4: Input of SUN RGB-D

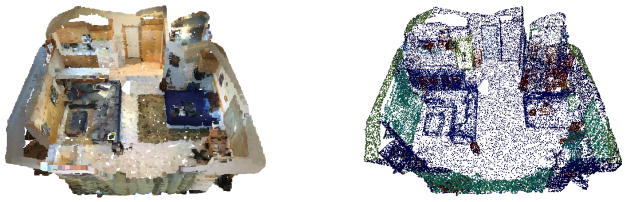
ScanNet is a richly annotated dataset of 3D reconstructed meshes of indoor scenes. It contains 1.5k training examples collected from hundreds of different rooms, and annotated with semantic and instance segmentation for 18 object categories ($NC = 18$). The dataset is not amodal and has no pose information. Point clouds are extracted on the reconstructed meshes and each contains 40k points (Fig. 6). ScanNet is divided into three splits: a training split, a validation split and a testing split, but we only have the ground truth for training and validation.
About the hyper-parameters for ScanNet the authors have chosen $NH = 1$ [2], $NS = NC = 18$. Since ScanNet does not provide oriented bounding box annotations, the network aims to predict axis aligned bounding boxes. That explains why there is only one heading bins. The number of maximum object per scenes is set to 64, that is far more than the most charged scene.

[2] It is written 12 in the paper and set to 1 in the code.

	cab	bed	chair	sofa	tabl	door	wind	bkshf	pic	cntr	desk	curt	fridg	showr	toil	sink	bath	ofurn	mAP
VoteNet	36.27	87.92	88.71	89.62	58.77	47.32	38.10	44.62	7.83	56.13	71.69	47.23	45.37	57.13	94.94	54.70	92.11	37.20	58.65
	8.07	76.06	67.23	68.82	42.36	15.34	6.43	28.00	1.25	9.52	37.52	11.55	27.80	9.96	86.53	16.76	78.87	11.69	33.54
Our	32.51	87.62	87.16	88.94	58.12	44.88	36.55	48.78	43.83	57.94	63.69	41.23	44.15	66.52	95.15	50.59	89.06	37.44	57.49
	5.58	71.73	60.90	63.39	35.49	13.36	8.26	27.30	0.15	21.29	30.83	11.11	29.29	19.12	86.04	29.44	75.42	8.75	33.20

Table 1: 3D object detection scores per category on the ScanNetV2 dataset, evaluated with mAP@0.25 IoU (1st line) and mAP@0.50 (2nd line). VoteNet represents the paper results and "our" represents our results.

For both datasets, each class has a different mean size that is used to calculate the size residual during the bounding box loss.



(a) RGB scan of a room. (b) Semantic segmentation.

Figure 5: Input of Scannet

3.2. Metric

The metric used is the mean average Precision (mAP) with a 3D Intersection over Union (IoU) threshold. They have chosen 0.25% and 0.5% as thresholds.

3.3. Training details

The original training uses an Adam optimizer with a learning rate of 0.001 that is divided by 10 after 80 epochs, 120 epochs and 160 epochs. According to the authors, the learning rate scheduling has boosted the performance on ScanNet compared with the first version of the paper. Originally they had exponential decay of the learning rate for paper version-1. Later they updated it to a step-wise learning rate decay. They have also used a schedule for the batch momentum. It goes from 0.5 to 0.999 with the following formula: momentum $= 1 - max(0.001, 0.5^{P+1})$ where P increases by 1 every 20 epochs and stars from 0.
Unlike the authors, we have used a P100 provided by Colab Pro and we have trained our models for 200 epochs on Scannetv2 (3 hours) and 140 epochs for SUN RGB-D v1 and v2 (14 hours each).

3.4. Implementation

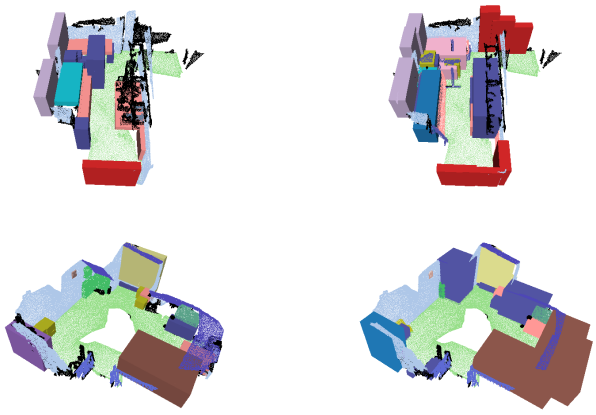
The original code relies [3] on PyTorch 1.1.0 and Tensorflow 1.14.0. For compatibility reasons with the C++ code of PointNet++, we have not changed the versions (see error [4]).
We have re-written several part of the codes to make it more readable, leading to six main folders. Our final implementation has the following folders. *Losses* contains one file per main loss. *Models* contains the backbone, the voting and the proposal module. *PointNet2* contains the implementation of PointNet++. *ScanNet* and *Sunrgbd* contains the code for the pre-processing and the creation of the training folder associated with the ScanNet or the Sun RGB-D dataset. Finally, *utils* contains distance functions, metrics and other useful functions.

[3] https://github.com/facebookresearch/votenet/

[4] UserWarning: Attempted to use ninja as the BuildExtension backend but we could not find ninja.

3.5. Results

We have obtained similar results for Scannet in around 3 hours. Results are better for the "counter" class. Otherwise, they are slightly worse (Tab. 1). The difference is due to the randomness in optimisation. Visually, we note three things. The first one is that it has a better localization of objects than a classification. The second one is that it predicts bounding boxes for un-annotated points (cf. black dots on 6). The network therefore predicts in undetermined areas, which can distort the results since these predictions penalize the optimization. The third one is that it tends to predict more bounding box than there are objects and often leads to larger bounding box than the one annotated (cf. left red door on the 2nd image 8). It also does not hesitate to increase the size of the bounding box in the ground. Moreover, the network confuses relatively close classes leading to relatively non-penalizing error.



(a) Ground-truth. (b) Predictions.

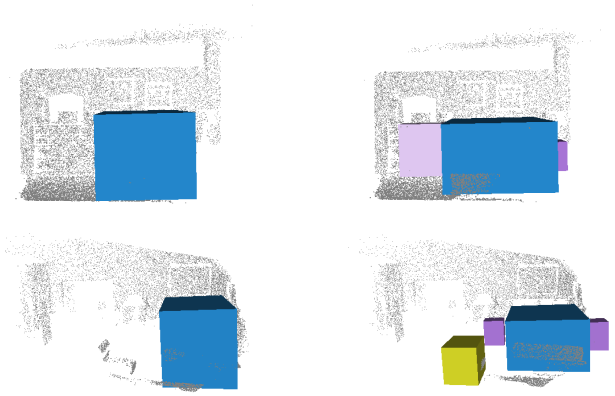
Figure 6: Output of VoteNet on Scannet with an objectness threshold of 0.5 after non-maximum suppression.

About SunRGBD, we have decided to train and evaluate our model on both the v1 and the v2 of the dataset in order to have updated bounding boxes (Tab. 2). The training lasts for 14 hours until the 140th epoch using the default parame-

Model	Metric	bed	table	sofa	chair	toilet	desk	dresser	night-stand	bookshelf	bathtub	Average
Votenet (v1)	mAP@0.25	83.0	47.3	64.0	75.3	90.1	22.0	29.8	62.2	28.2	74.4	57.7
Our (v2)	mAP@0.25	84.52	49.10	65.90	76.76	90.07	24.50	27.69	61.11	31.43	77.43	58.86
	mAP@0.50	46.65	15.17	38.82	53.39	59.88	39.44	14.00	36.49	6.80	33.71	30.88
Our (v1)	mAP@0.25	83.37	47.97	62.75	74.21	88.50	21.51	20.96	60.82	31.19	79.07	57.03
	mAP@0.50	49.79	14.14	39.02	50.00	57.49	39.76	9.77	25.91	4.87	34.04	28.90

Table 2: 3D object detection scores per category on the SUN RGB-D v2 dataset compared to the results of the article obtained on SUN RGB-D v1.

ters. As the data is more sparse, it appears that the network tends to have implicitly learned the presence of objects relative to another. With blue representing beds, we notice that the network predicts a nightstand twice without it actually being present (Fig. 2, 9).



(a) Ground-truth. (b) Predictions.

Figure 7: Output of VoteNet on SUN RGB-D v2 and v1 with an objectness threshold of 0.5 after non-maximum suppression.

4. Improvements

4.1. Pre-training

Votenet leads to impressive results, but it presents some defaults that deserve reflections. To begin with, we can criticise the data that is used for evaluation. As we have noticed, black points represent un-annotated points. They have no ground truth, so when the network predicts a bounding box around them, it automatically increases the loss. Even if this point is not directly linked to the network it is always important to keep in mind.

With the point cloud, one way to improve the network is to use additional features as input. We have xyz coordinates and the network has an option to use rgb colors. Also, in the SUN RGB-D dataset, there is a pose estimate, so we can add a ground distance as an additional feature. Moreover, RGB-D images allows to use depth as a feature and some point clouds provide laser intensity. In some situations, other hand-created features can be useful. We have seen in class other features: verticality, flatness, linearity and sphericity which are based on the eigenvalues of the covariance matrix.

Another point of improvement, this time less specific to a dataset, is data-augmentation. During training, the data are augmented using two different techniques: flipping and rotation. To improve the data-augmentation we can include point sampling, cropping, adding noise, isotrocpic scaling, random dropout or chromatic jittering. Sampling, noise dropout and cropping are of particular interest as we provide a bounding box for partial objects, while jitter is useful to account for object color diversity.

4.2. Architecture

Regarding the architecture, improvements have been made since the release of VoteNet. Even if the trend is not to modify the backbone network, we can change the backbone PointNet++ network that creates features using for instance a convolutional approach such as a KPConv layer [15] that leads to promising results on semantic segmentation and classification.

We can also try to change the core architecture. One way is to add additional inputs. The authors of Deep Hough Voting, Qi et al. [10] proposed an ImVoteNet detector that uses 2D RGB information. Compared to prior work on multi-modal detection, they explicitly extract both geometric and semantic features from the 2D images. They leverage the camera parameters to lift these features to 3D. To improve the synergy of 2D-3D feature fusion, they also propose a multi-tower training scheme and obtained 63.4 mAP@0.25 on the SUN RGB-D dataset.

But, to improve VoteNet it is more coherent to look at networks that use only 3D geometrical point cloud information. To better capture the fine local structural features surrounding the potential objects from the raw point clouds, BRNET [3] generatively back-traces the representative points from the vote centers and rethinks seed points around these generated points. Also, Liu et al. [7] introduces the usage of Transformers in GroupFree3D that modified the part between object candidates and bounding box estimation. Instead of grouping local points to each object candidate, they compute the feature of an object from all the points in the point cloud with the help of an attention mechanism, where the contribution of each point is automatically learned in the network training. Results are among the best obtained on the SUN RGB-D segmentation benchmark (Tab. 3). We can see that the mAP@0.50 is largely improved with the recent architecture. Considering the results, VoteNet seems good to predict the objectness but has some difficulties to precisely localise objects.

Model	VoteNet	BRNET	GroupFree3D	ImVoteNet
Year	2019	2021	2021	2020
Only Geo.Info	Yes	Yes	Yes	No
mAP@ 0.25/0.50	59.1 / 35.8	61.1 / 43.7	63.0 / 45.2	63.4 / -

Table 3: Networks that beat VoteNet on the SUN RGB-D segmentation benchmark.

Concerning the predictions, one drawback is the unstability of the virtual center point for a partially occluded object. To overcome this issue, Feng et al. [5] added an auxiliary branch of direction vectors to improve the prediction accuracy of virtual center points and 3D candidate boxes. In addition, a 3D object-object relationship graph between proposals is built to emphasize useful features for accurate object detection.

5. Conclusion

Deep Hough Voting is an interesting paper that uses only geometric information to predict the bounding box of several objects in a point cloud. It leads to promising results and rethinks the traditional Hough voting method in a modern way. The pipeline is simply understandable but it is composed of complex modules that takes me time to understand. The network is trained relatively fast even using a colab GPU and the results are good and reproducible. Its ability to predict objects with scattered point clouds impressed me as its predictions were close to reality. Nonetheless, the network does not always have an accurate estimate of the center of the bounding box and predicts objects outside the map.
These drawbacks invite us to test more recent approaches in the same vein as the paper. With more time, I could have tried to implement an improvement, although in this case it might be more interesting to work on a complementary article with a more recent approach.

References

[1] Danna H Ballard. *Generalizing the hough transform to detect arbitrary shapes*. 1981.

[2] Xiaozhi Chen and Huimin Ma. *Multi-View 3D Object Detection Network for Autonomous Driving*. 2019. arXiv: 1611.07759.

[3] Bowen Cheng et al. "Back-tracing Representative Points for Voting-based 3D Object Detection in Point Clouds". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.

[4] Angela Dai et al. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes". In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2017.

[5] Mingtao Feng et al. "Relation Graph Network for 3D Object Detection in Point Clouds". In: 2019.

[6] Paul VC Hough. *Machine analysis of bubble chamber pictures*. 1959.

[7] Ze Liu et al. "Group-Free 3D Object Detection via Transformers". In: *arXiv preprint arXiv:2104.00678* (2021).

[8] A Mousavian, D. Anguelov, and J. Flynn. *3D Bounding Box Estimation Using Deep Learning and Geometry*. 2017.

[9] Charles R Qi et al. *Frustum pointnets for 3d object detection from rgbd data.* 2018.

[10] Charles R Qi et al. "Imvotenet: Boosting 3d object detection in point clouds with image votes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[11] Charles R Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2016.

[12] Charles R. Qi, Or Litany, and Kaiming He. *Deep Hough Voting for 3D Object Detection in Point Clouds*. 2019. arXiv: 1904.09664.

[13] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017.

[14] S. Song, S. Lichtenberg, and J. Xiao. *SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite*. 2015.

[15] Hugues Thomas et al. "KPConv: Flexible and Deformable Convolution for Point Clouds". In: (2019). URL: http://arxiv.org/abs/1904.08889.

6. Gallery

You can find on the next pages more outputs obtained during testing of our networks.

Figure 8: Point cloud, ground-truth and prediction on some scans of the 3rd testing batch of ScannetV2.

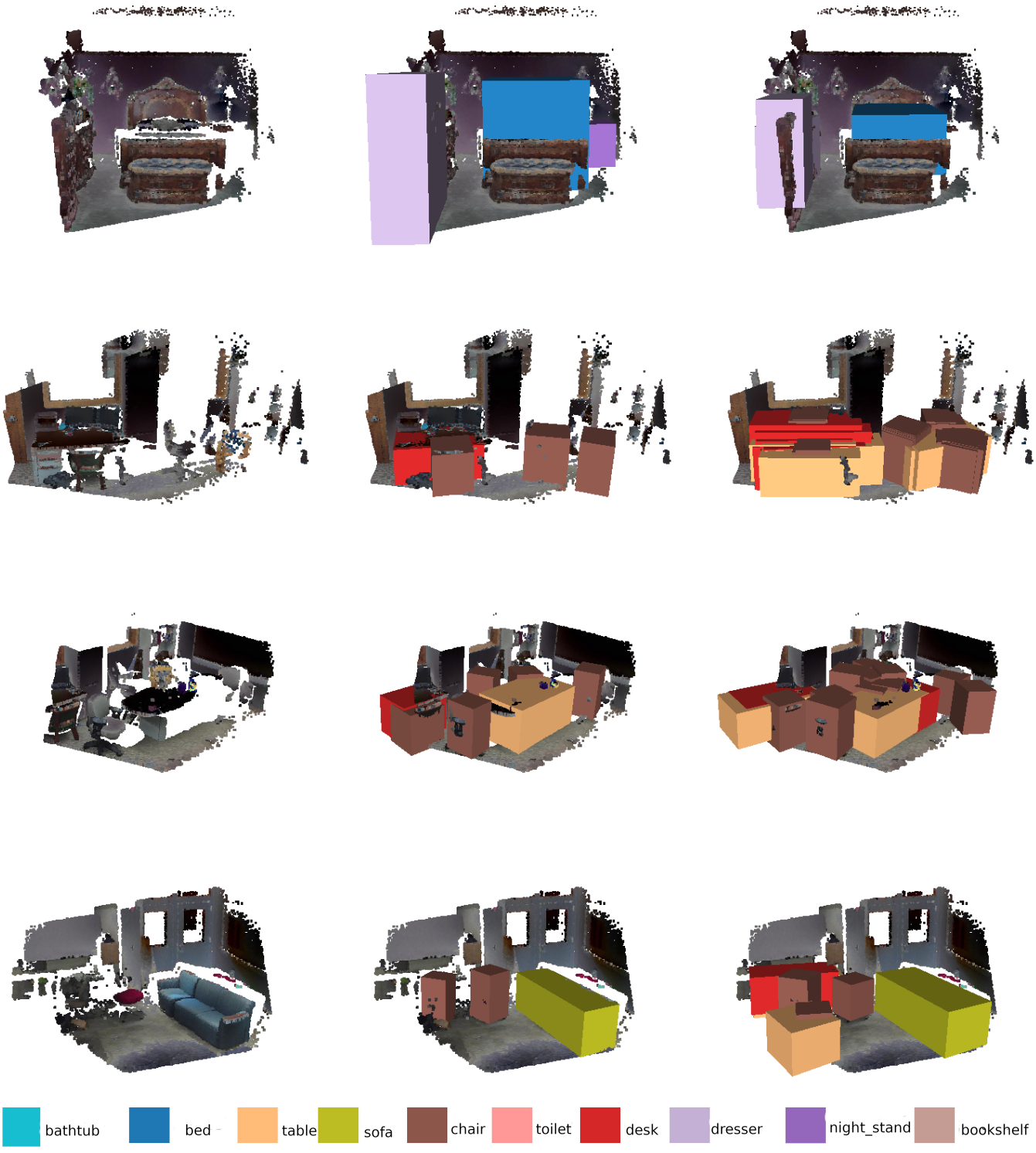


Figure 9: Point cloud, ground-truth and prediction on some scans of the 10th testing batch of Sun RGB-D v2.