YOLO3D: End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud

Waleed Ali*[0000−0002−2687−1272], Sherif Abdelkarim*[0000−0002−7456−6133],
Mahmoud Zidan[0000−0003−0122−7779], Mohamed Zahran[0000−0002−4082−814X],
and Ahmad El Sallab[0000−0001−6616−1446]

Valeo AI Research, Cairo, Egypt
{waleed.ali, sherif.abdelkarim,
mahmoud.ismail-zidan.ext, mohamed.zahran,
ahmad.el-sallab}@valeo.com

Abstract. Object detection and classification in 3D is a key task in Automated Driving (AD). LiDAR sensors are employed to provide the 3D point cloud reconstruction of the surrounding environment, while the task of 3D object bounding box detection in real time remains a strong algorithmic challenge. In this paper, we build on the success of the one-shot regression meta-architecture in the 2D perspective image space and extend it to generate oriented 3D object bounding boxes from LiDAR point cloud. Our main contribution is in extending the loss function of YOLO v2 to include the yaw angle, the 3D box center in Cartesian coordinates and the height of the box as a direct regression problem. This formulation enables real-time performance, which is essential for automated driving. Our results are showing promising figures on KITTI benchmark, achieving real-time performance (40 fps) on Titan X GPU.

Keywords: 3D Object Detection, LiDAR, Real-time

1 Introduction

Automated Driving (AD) success is highly dependent on efficient environment perception. Sensors technology is an enabler to environment perception. LiDAR-based environment perception systems are essential components for homogeneous (same sensor type) or heterogeneous (different sensors types) fusion systems. The key feature of LiDAR is its physical ability to perceive depth at high accuracy.

Among the most important tasks of the environment perception is Object Bounding Box (OBB) detection and classification, which may be done in the 2D (bird-view) or the 3D space. Unlike camera-based systems, LiDAR point clouds are lacking some features that exist in camera RGB perspective scenes, like colors. This makes the classification task from LiDAR only more complicated.

---
* These authors contributed equally

On the other hand, depth is given as a natural measurement by LiDAR, which enables 3D OBB detections. The density of the LiDAR point cloud plays a vital role in the efficient classification of the object type, especially small objects like pedestrians and animals.

Real-time performance is essential in AD systems. While Deep Learning (DL) has a well-known success story in camera-based computer vision, such approaches suffer high latency in their inference path, due to the expensive convolution operations. In the context of object detection, rich literature exists that tackles the problem of real-time performance. Single shot detectors, like YOLO [1] and SSD [2] are some of the best in this regard.

In this paper, we extend YOLO V2 [3] to perform 3D OBB detection and classification from 3D LiDAR point cloud (PCL). In the input phase, we feed the bird-view of the 3D PCL to the input convolution channels. The network architecture follows the meta-architecture of YOLO with architecture adaptation and tuning to match the nature of the sparse LiDAR input. The predictions include 8 regression outputs + classes (versus 5 regressors + classes in case of YOLO V2): the OBB center in 3D (x, y, z), the 3D dimensions (length, width and height), the orientation in the bird-view space, the confidence, and the object class label. Following the one-shot regression theme, we do not depend on any region proposal pipelines, instead, the whole system is trained end to end.

The main contributions of this work can be summarized as follows:

1- Extending YOLO V2 [3] to include orientation of the OBB as a direct regression task.

2- Extending YOLO V2 [3] to include the height and 3D OBB center coordinates (x,y,z) as a direct regression task.

3- Real-time performance evaluation and experimentation with Titan X GPU, on the challenging KITTI benchmark, with recommendations of the best grid-map resolution, and operating IoU threshold that balances speed and accuracy.

The results evaluated on KITTI benchmark shows a clear advantage of the proposed approach, in terms of real-time efficiency (40 fps), and a promising accuracy. The rest of the paper is organized as follows: first, we discuss the related works, followed by a description of the proposed approach, and the combined one-shot loss for the 3D OBB, then we present, and discuss the experimental results on the KITTI benchmark dataset. Finally, we provide concluding remarks in section 5.

2 Related Work

In this section, we summarize 3D object detection in autonomous driving for LiDAR point clouds. We then summarize related works in orientation prediction, which we use to predict the real angle of the vehicle. Finally, we discuss the implications of 3D object detection on the real-time performance.

Fig. 1. Sample of the output shown in 3D and projected on the top view map

2.1   3D Object Detection

There are three ways to do 3D object detection in terms of sensor type. Firstly, it is the LIDAR-only paradigm, which benefits from accurate depth information. Overall, these paradigms differ in data preprocessing. Some approaches project point cloud in 2D view (Bird-View, Front-View) such as [4] and [5]; and some [6] convert the point cloud to a front view depth map. Others like [7] and [8], convert the point cloud to voxels producing a discrete square grid.

The Second one is the camera-only paradigm; which works by adding prior knowledge about the objects' sizes, and trying to predict 3D bounding box using monocolor camera.[9] and [10] can produce highly accurate 3D bounding boxes using only camera images.[11] uses stereo vision to produce high-quality 3D object detection.

The LIDAR-camera fusion comes at the last. This paradigm tries to utilize the advantages of both paradigms mentioned above. The LIDAR produces accurate depth information, and the camera produces rich visual features; if we combine the output of the two sensors, we can have more accurate object detection and recognition. MV3D [5] fuses bird view, front view and the RGB camera to produce 3D vehicle detection. F-pointnet [12] combines raw point cloud with RGB camera images. Object detection on RGB image produces a 2D bounding box which maps to a frustum in the point cloud. Then, 3D object detection is performed directly on frustum to produce accurate bounding boxes. However, fusing lidar data with camera suffers from adding more time complexity to the problem.

In this work, we are following the first paradigm of using only lidar point cloud projected as special bird view grid to keep the 3D information, more details will be discussed in section III-A.

2.2 Orientation Prediction

One approach in finding the orientation is introduced by MV3D [5], where the orientation vector is assumed to be in the direction of the longer side of the box. This approach fails in regards to pedestrians because they don't obey this rule.

Another approach is to convert the orientation vector to its component, as shown in [13] and [14]. AVOD [13] converts the orientation vector to sine and cosine. Complex YOLO [14] converts the orientation vector to real and imaginary values. The problem with this is that the regression does not guarantee, or preserve any type of correlation between the two components of the angle.

2.3 Real Time Performance

Object detection is fundamental to automated driving, yet it suffers from computational complexity. There is a need to make the models as efficient as possible in terms of size and inference time maintaining a good accuracy.

Some work has been done to tackle the efficiency of models, such as Squeeze-Net [15], Mobile-Net [16], and Shuffle-Net [17], and for object detection, there is Tiny YOLO and Tiny SSD [18]. All these architectures are optimized for camera images, and they cannot easily be adapted to work on images produced from LiDAR point clouds. The reason is that, unlike camera images, LiDAR images consist of very sparse information. Vote3Deep [19] performs 3D sparse convolution to take advantage of this sparsity.

Extending YOLOv2 [3], we include the orientation of the OBB as a direct regression task, unlike the work in [14], which suggests two separate losses for the real and imaginary parts of the angle, without explicit nor implicit correlation between them, which may result in wrong or invalid angles in many cases.

In addition, in [14], 3D OBB height and z-center are not a natural or exact output from the network, but rather a heuristic based on statistics and average sizes of the data. In this work, we extend YoLo v2 [3] to include height and 3D OBB center as direct regression tasks. A sample of our output can be seen in Figure (1), taken from KITTI benchmark test data.

3 Approach

3.1 Point Cloud Representation

We project the point cloud to create a bird's eye view grid map. We create two grid maps from the projection of the point cloud as shown in Figure (2). The first feature map contains the maximum height, where each grid cell (pixel) value represents the height of the highest point associated with that cell. We
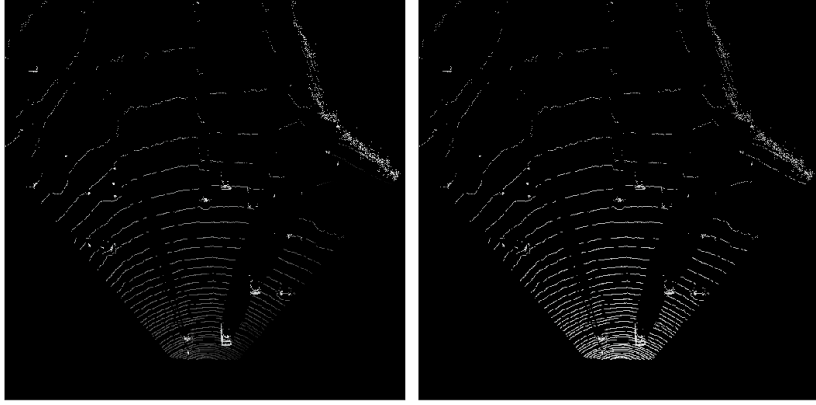
Fig. 2. Sample of the input grid maps. Left: the height map. Right: the density map.

use only the maximum height instead of a more complex statistic over the z dimension because we are only interested in the highest point of the object. The second grid map represent the density of points. Which means, the more points are associated with a grid cell, the higher its value would be. The density is calculated using the following equation taken from MV3D paper [5]:

$$min(1.0, \frac{\log(N+1)}{\log(64)})$$ (1)

Where N is the number of points in each grid cell.

3.2   Yaw Angle Regression

The orientation of the bounding boxes has a range from -π to π. We normalized that range to be from -1 to 1, and adapted our model to directly predict the orientation of the bounding box via a single regressed number. In the loss function, we compute the mean squared error between the ground truth and our predicted angle:

$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} (\phi_i - \hat{\phi}_i)^2$$ (2)

In our experimentation, we used using tanh as an activation for the angle output yaw (to bound the output between -1 and 1), but it did not offer an improvement over the linear activation.

3.3   3D Bounding Box Regression

We added two regression terms to the original YOLO v2 [3] in order to produce 3D bounding boxes, the z coordinate of the center, and the height of the box.
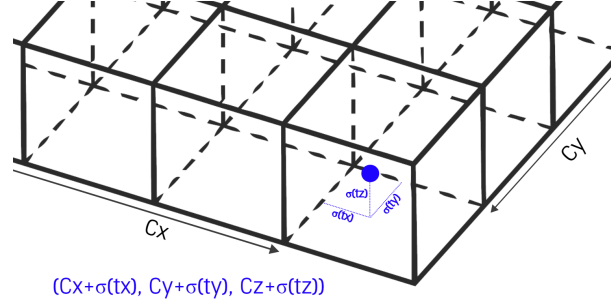
Fig. 3. Sample of the grid output when extended to the third dimension where c_z equals 0 since the grids are only one level high in the z dimension.

The regression over the z coordinate in Eq. (5) is done in a way similar to the regression of the x Eq. (3) and y Eq. (4) coordinates via a sigmoid activation function.

While the x and y are regressed by predicting a value between 0 and 1 at each grid cell, locating where the point lies within that cell, the value of z is only mapped to lie within one vertical grid cell as illustrated in Figure (3). The reason for choosing to map z values to only one grid while x and y are mapped to several grid cells is that the variability of values in the z dimension are much smaller than that of the x and y (most objects have very similar box elevations).

The height of the box h Eq. (8) is also predicted similarly to the width w in Eq. (6) and length l in Eq. (7)

$$b_x = \sigma(t_x) + c_x \tag{3}$$

$$b_y = \sigma(t_y) + c_y \tag{4}$$

$$b_z = \sigma(t_z) + c_z \tag{5}$$

$$b_w = p_w e^{t_w} \tag{6}$$

$$b_l = p_l e^{t_l} \tag{7}$$

$$b_h = p_h e^{t_h} \tag{8}$$

3.4 Anchors Calculation

In YOLO-v2 [3], anchors are calculated using k-means clustering over the width and length of the ground truth boxes. The point behind using anchors, is to find priors for the boxes, onto which the model can predict modifications. The anchors must be able to cover the whole range of boxes that can appear in the data. In [3], the model is trained on camera images, where there is a high variability of box sizes, even for the same object class. Therefore, calculating anchors using clustering is beneficial.

On the other hand, in the case of bird's eye view grid maps, there is no such high variability in box dimensions within the same object class (most cars have similar sizes). For this reason, we chose not to use clustering to calculate the anchors, and instead, calculate the mean 3D box dimensions for each object class, and use these average box dimensions as our anchors.

3.5   Combined Loss for 3D OBB

The loss for 3D oriented boxes is an extension to the original YOLO loss for 2D boxes. The loss for the yaw term is calculated as described in subsection 3.2 and Eq. (2). The loss for the height is an extension to the loss over the width and length in (9). Similarly, the loss for the z coordinate is an extension to the loss over the x and y coordinates, as shown in (9).

The total loss shown in Eq. (9) is calculated as the scaled summation of the following terms: the mean squared error over the 3D coordinates and dimensions (x, y, z, w, l, h), the mean squared error over the angle, the confidence score, and the cross entropy loss over the object classes.

$$
\begin{aligned}
L = {} & \lambda_{coor} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (z_i - \hat{z}_i)^2] \\
& + \lambda_{coor} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{l_i} - \sqrt{\hat{l}_i})^2 \\
& + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
& + \lambda_{yaw} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} (\phi_i - \hat{\phi}_i)^2 \\
& + \lambda_{conf} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
& + \lambda_{conf} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
& + \lambda_{classes} \sum_{i=0}^{s^2} \sum_{j=0}^{B} L_{ij}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{9}
$$

Where: λ_{coor} : the weight assigned to the loss over the coordinates, λ_{conf} : the weight assigned to the loss over predicting the confidence, λ_{yaw} : the weight assigned to the loss over the orientation angle, $\lambda_{classes}$: the weight assigned to the loss over the class probabilities, L_{ij}^{obj} : a variable that takes the values of 0 and 1 based on whether there is a ground truth box in the ith and jth location. 1 if there's a box, and 0 otherwise, L_{ij}^{noobj} : the opposite of the previous variable.

takes the value of 0 if there's no object, and 1 otherwise, x_i, y_i, z_i : the gound truth coordinates, $\hat{x}_i, \hat{y}_i, \hat{z}_i$: the ground truth and predicted orientation angle, $\phi_i, \hat{\phi}_i$: the ground truth and predicted orientation angle, $C_i, \hat{C}_i$: the ground truth and predicted confidence, w_i, l_i, h_i : the ground truth width, height, and length of the box, $\hat{w}_i, \hat{l}_i, \hat{h}_i$: the predicted width, height, and length of the box and $p_i(c), \hat{p}_i(c)$:The ground truth and predicted class probabilities. B is the number of boxes, and s is the length of one of the sides of the square output grid, thus s^2 is the number of grids in the output.

4 Experiments and Results

4.1 Network Architecture and Hyper Parameters

Our model is based on YOLO-v2 [3] architecture with some changes, as shown in Table 1.

1. We modified one max-pooling layer to change the down-sampling from 32 to 16 so we can have a larger grid at the end; this has a contribution in detecting small objects like pedestrians and cyclists.
2. We removed the skip connection from the model as we found it resulting in less accurate results.
3. We added terms in the loss function for yaw, z center coordinate, and height regressions to facilitate the 3D oriented bounding box detection.
4. Our input consists of 2 channels, one representing the maximum height, and the other one representing the density of points in the point cloud, computed as shown in Eq. (1).

4.2 Dataset and Preprocessing

We used KITTI benchmark dataset. The point cloud was projected in 2D space as a bird view grid map with a resolution of 0.1m per pixel, same resolution is used by MV3D [5].

The range represented from the LiDAR space by the grid map is 30.4 meters to right and 30.4 meters to the left, and 60.8 meters forward. Using this range with the above mentioned resolution of 0.1 results in an input shape of 608x608 per channel.

The height in the LiDAR space is clipped between +2m and -2m, and scaled to be from 0 to 255 to be represented as pixel values in the maximum height channel.

Since in KITTI benchmark only the objects that lies on the image plane are labeled, we filter any points from the point cloud that lie outside the image plane. The rationale behind this, is to avoid giving the model contradictory information. Since objects lying on the image plane would need to be detected, while the ones lying outside that plane should be ignored, as they are not labeled. Therefore, we only include the points that lie within the image plane.

Table 1. Network Architecture

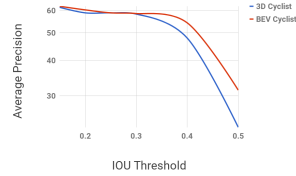| layer	filters	size	feature maps
conv2d	32	(3 , 3)	608x608x2
maxpooling	-	(size 2 , stride 2)	
conv2d	64	(3 , 3)	
maxpooling	-	(size 2 , stride 2)	
conv2d	128	(3 , 3)	
conv2d	64	(3 , 3)	
conv2d	128	(3 , 3)	
maxpooling	-	(size 2 , stride 1)	
conv2d	256	(3 , 3)	
conv2d	128	(3 , 3)	
conv2d	256	(3 , 3)	
maxpooling	-	(size 2 , stride 2)	
conv2d	512	(3 , 3)	
conv2d	256	(1 ,1)	
conv2d	512	(3 , 3)	
conv2d	256	(1 ,1)	
conv2d	512	(3 , 3)	
maxpooling	-	(size 2 , stride 2)	
conv2d	1024	(3 , 3)	
conv2d	512	(1 , 1)	
conv2d	1024	(3 , 3)	
conv2d	512	(1 , 1)	
conv2d	1024	(3 , 3)	
conv2d	1024	(3 , 3)	
conv2d	1024	(3 , 3)	
conv2d	1024	(3 , 3)	
conv2d	1024	(1 ,1)	38x38x33
reshape	-	-	38x38x3x11

4.3 Training

The network is trained in an end-to-end fashion. We used stochastic gradient descent with a momentum of 0.9, and a weight decay of 0.0005. We trained the network for 150 epochs, with a batch size of 4.

Our learning rate schedule is as follows: for the first few epochs, we slowly raise the learning rate from 0.00001 to 0.0001. If we start at a high learning rate, our model often diverges due to unstable gradients. We continue training with 0.0001 for 90 epochs, then 0.0005 for 30 epochs, and finally, 0.00005 for the last 20 epochs.

4.4 KITTI Results and Error Analysis

As discussed in [20], and from the results reported in [1] and [3], YOLO performs very well with the detection metric of mean average precision at IOU threshold
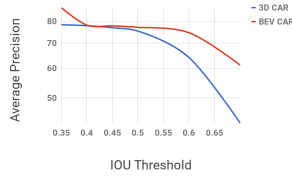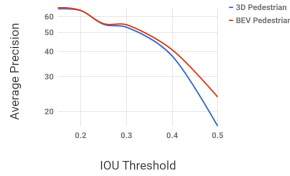
Fig. 4. Car   **Fig. 5.** pedestrian   **Fig. 6.** cyclist

Fig. 7. Performance against IOU threshold

of 0.5. This gives us an advantage over the previous work in 3D detection from point cloud in terms of speed with an accepted mAP, as shown in Figure (7).

However, performance drops significantly as the IOU threshold increases indicating that we struggle to get the boxes perfectly aligned with the object, which is an inherited problem in all YOLO versions [1], [3], [20]. Fig 7 shows that the model succeeds in detecting the objects but struggles with accurately localizing them.

Compared with the state of the art approaches on 3D object detection, such as MV3D [5], which fails in detecting pedestrians and cyclists despite its relatively large, and complex multi view, multi sensor network, as well as, AVOD [13], which dedicates a separate network for detecting cars, and one for pedestrians and cyclists, our proposed architecture can detect all objects from only a two channel bird view input, and with just one single network, achieving a real time performance of 40 fps , and a 75.3% mAP on 0.5 IOU threshold for moderate cars. The precision and recall scores on our validation set (about 40% of the KITTI training set) are shown in Table 2.

Table 2. Validation Results

label	precision	recall
pedestrian	44.0%	39.2%
cyclist	65.13%	51.1%
car	94.07%	83.4%

4.5 Effect of Grid Map Resolution

Grid map resolution is a critical hyper-parameter that affect memory usage, time and performance. For instance, if we want to deploy the model on an embedded target, we have to focus on fast inference time with small input size, and reasonable performance.

The area of the grid map grows proportionally to the length or width of the grid map squared. This means increasing the resolution of the grid map, increases
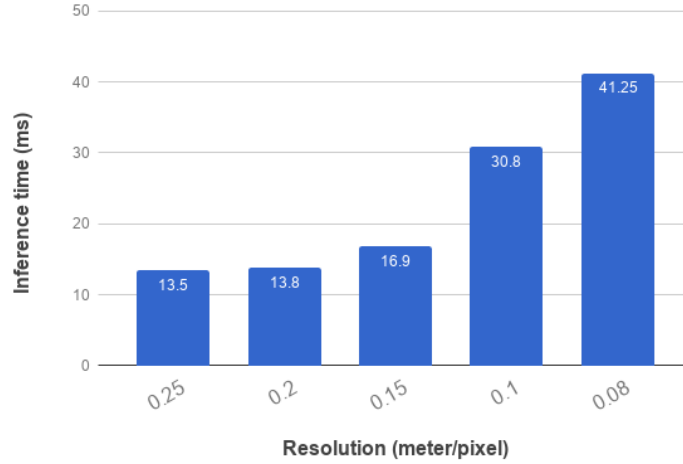
Fig. 8. Inference time at different resolutions

the area of the grid map (and thus the inference time) quadratically. This can be seen in Figure (8), where there is a rapid increase in the inference time after the 0.15 meter/pixel mark, where only increasing the resolution by 0.05 meters/pixel (from 0.15 meters/pixel to 0.1 meter/pixel) causes the inference time to double from 16.9ms to 30.8ms.

5   Conclusions

In this paper we present real-time LiDAR based system for 3D OBB detection and classification, based on extending YOLO-v2 [3]. The presented approach is trained end to end, without any pipelines of region proposals which ensure real time performance in the inference pass. The box orientation is ensured by direction regression on the yaw angle in bird-view. The 3D OBB center coordinates and dimensions are also formulated as a direct regression task, with no heuristics. The system is evaluated on the official KITTI benchmark at different IoU thresholds, with recommendation of the best operating point to get real time performance and best accuracy. In addition, the real time performance is evaluated at different grid-map resolutions. The results suggest that single shot detectors can be extended to predict 3D boxes while maintaining real-time performance; however this comes with a cost on the localization accuracy of the boxes.

References

1. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer

vision and pattern recognition. (2016) 779–788

2. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: European conference on computer vision, Springer (2016) 21–37

3. Redmon, J., Farhadi, A.: Yolo9000: Better, faster, stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE (2017) 6517–6525

4. Li, B., Zhang, T., Xia, T.: Vehicle detection from 3d lidar using fully convolutional network. arXiv preprint arXiv:1608.07916 (2016)

5. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3d object detection network for autonomous driving. In: IEEE CVPR. Volume 1. (2017) 3

6. Asvadi, A., Garrote, L., Premebida, C., Peixoto, P., Nunes, U.J.: Depthcn: vehicle detection using 3d-lidar and convnet. In: IEEE ITSC. (2017)

7. Li, B.: 3d fully convolutional network for vehicle detection in point cloud. In: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, IEEE (2017) 1513–1518

8. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. arXiv preprint arXiv:1711.06396 (2017)

9. Mousavian, A., Anguelov, D., Flynn, J., Košecká, J.: 3d bounding box estimation using deep learning and geometry. In: Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, IEEE (2017) 5632–5640

10. Chabot, F., Chaouch, M., Rabarisoa, J., Teulière, C., Chateau, T.: Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In: Proc. IEEE Conf. Comput. Vis. Pattern Recognit.(CVPR). (2017) 2040–2049

11. Chen, X., Kundu, K., Zhu, Y., Ma, H., Fidler, S., Urtasun, R.: 3d object proposals using stereo imagery for accurate object class detection. IEEE transactions on pattern analysis and machine intelligence **40**(5) (2018) 1259–1272

12. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from rgb-d data. arXiv preprint arXiv:1711.08488 (2017)

13. Ku, J., Mozifian, M., Lee, J., Harakeh, A., Waslander, S.: Joint 3d proposal generation and object detection from view aggregation. arXiv preprint arXiv:1712.02294 (2017)

14. Simon, M., Milz, S., Amende, K., Gross, H.M.: Complex-yolo: Real-time 3d object detection on point clouds. arXiv preprint arXiv:1803.06199 (2018)

15. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)

16. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)

17. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. arXiv preprint arXiv:1707.01083 (2017)

18. Wong, A., Shafiee, M.J., Li, F., Chwyl, B.: Tiny ssd: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. arXiv preprint arXiv:1802.06488 (2018)

19. Engelcke, M., Rao, D., Wang, D.Z., Tong, C.H., Posner, I.: Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In: Robotics and Automation (ICRA), 2017 IEEE International Conference on, IEEE (2017) 1355–1361

20. Farhadi, J.R.A.: Yolov3: An incremental improvement