# PIXOR: Real-time 3D Object Detection from Point Clouds

Bin Yang, Wenjie Luo, Raquel Urtasun
Uber Advanced Technologies Group
University of Toronto
{byang10, wenjie, urtasun}@uber.com

## Abstract

*We address the problem of real-time 3D object detection from point clouds in the context of autonomous driving. Speed is critical as detection is a necessary component for safety. Existing approaches are, however, expensive in computation due to high dimensionality of point clouds. We utilize the 3D data more efficiently by representing the scene from the Bird's Eye View (BEV), and propose PIXOR, a proposal-free, single-stage detector that outputs oriented 3D object estimates decoded from pixel-wise neural network predictions. The input representation, network architecture, and model optimization are specially designed to balance high accuracy and real-time efficiency. We validate PIXOR on two datasets: the KITTI BEV object detection benchmark, and a large-scale 3D vehicle detection benchmark. In both datasets we show that the proposed detector surpasses other state-of-the-art methods notably in terms of Average Precision (AP), while still runs at 10 FPS.*

## 1. Introduction

Over the last few years we have seen a plethora of methods that exploit Convolutional Neural Networks to produce accurate 2D object detections, typically from a single image [12, 11, 28, 4, 27, 23]. However, in robotics applications such as autonomous driving we are interested in detecting objects in 3D space. This is fundamental for motion planning in order to plan a safe route.

Recent approaches to 3D object detection exploit different data sources. Camera based approaches utilize either monocular [1] or stereo images [2]. However, accurate 3D estimation from 2D images is difficult, particularly in long ranges. With the popularity of inexpensive RGB-D sensors such as Microsoft Kinect, Intel RealSense and Apple PrimeSense, several approaches that utilize depth information and fuse them with RGB images have been developed [32, 33]. They have been shown to achieve significant performance gains over monocular methods. In the context of autonomous driving, high-end sensor like LIDAR (Light Detection And Ranging) is more common because higher accuracy is needed for safety. The major difficulty in dealing with LIDAR data is that the sensor produces unstructured data in the form of a point cloud containing typically around $10^5$ 3D points per 360-degree sweep. This poses a large computational challenge for modern detectors.

Different forms of point cloud representation have been explored in the context of 3D object detection. The main idea is to form a structured representation where standard convolution operation can be applied. Existing representations are mainly divided into two types: 3D voxel grids and 2D projections. A 3D voxel grid transforms the point cloud into a regularly spaced 3D grid, where each voxel cell can contain a scalar value (e.g., occupancy) or vector data (e.g., hand-crafted statistics computed from the points within that voxel cell). 3D convolution is typically applied to extract high-order representation from the voxel grid [6]. However, since point clouds are sparse by nature, the voxel grid is very sparse and therefore a large proportion of computation is redundant and unnecessary. As a result, typical systems [6, 37, 20] only run at 1-2 FPS.

An alternative is to project the point cloud onto a plane, which is then discretized into a 2D image based representation where 2D convolutions are applied. During discretization, hand-crafted features (or statistics) are computed as pixel values of the 2D image [3]. Commonly used projections are range view (i.e., 360-degree panoramic view) and bird's eye view (i.e., top-down view). These 2D projection based representations are more compact, but they bring information loss during projection and discretization. For example, range-view projection will have distorted object size and shape. To alleviate the information loss, MV3D [3] proposes to fuse the 2D projections with the camera image to bring additional information. However, the fused model has nearly linear computation cost with respect to the number of input modals, making real-time application infeasible.

In this paper, we propose an accurate *real-time* 3D object detector, which we call **PIXOR** (**OR**iented 3D object detection from **PIX**el-wise neural network predictions), that operates on 3D point clouds. PIXOR is a single-stage,
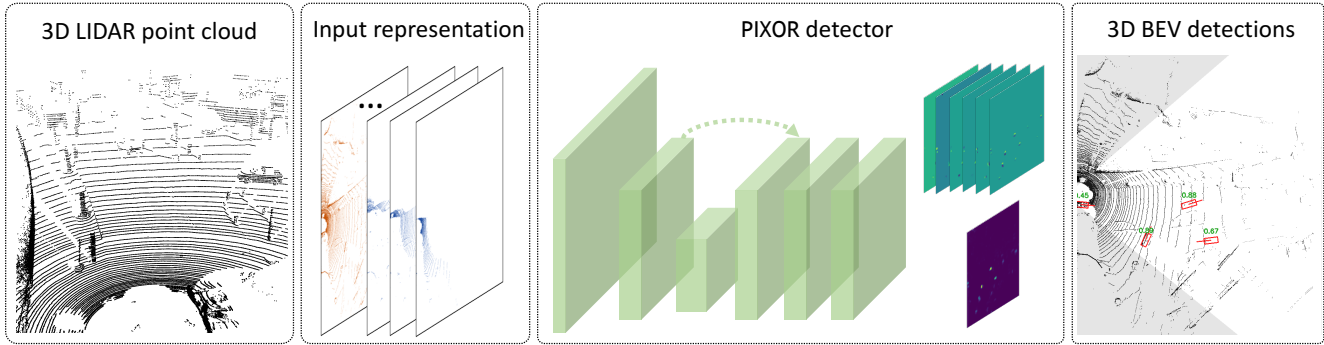
Figure 1. Overview of the proposed 3D object detector from Bird's Eye View (BEV) of LIDAR point cloud.

proposal-free dense object detector that exploits the 2D Bird's Eye View (BEV) representation in an efficient way. We choose the BEV representation as it is computationally friendly compared with 3D voxel grids, and also preserves the metric space which allows our model to explore priors about the size and shape of the object categories. Our detector outputs accurate oriented bounding boxes in real-world dimension in bird's eye view. Note that these are 3D estimates as we assume that the objects are on the ground. This is a reasonable assumption in the autonomous driving scenario as vehicles do not fly.

We demonstrate the effectiveness of our approach in two datasets, the public KITTI benchmark [10] and a large-scale 3D vehicle detection dataset (ATG4D). Specifically, PIXOR achieves the highest Average Precision (AP) on KITTI bird's eye view object detection benchmark among all previously published methods, while also runs the fastest among them (over 10 FPS). We also provide in-depth ablation studies on KITTI to investigate how much performance gain each module contributes, and prove the scalability and generalization ability of PIXOR by applying it to the large-scale ATG4D dataset.

## 2. Related Work

We first review recent advances in applying Convolutional Neural Networks to object detection, and then revisit works in two related sub-fields, single-stage object detection and 3D object detection.

### 2.1. CNN-based Object Detection

Convolutional Neural Networks (CNN) have shown outstanding performance in image classification [18]. When applied to object detection, it is natural to utilize them by running inference over cropped regions representing the object candidates. Overfeat [30] slides a CNN on different positions and scales and predicts a bounding box per class at each time. Since the introduction of class-agnostic object proposals [36, 26], proposal based approaches popu-

late, with Region-CNN (RCNN) [12] and its faster versions [11, 4] being the most seminal work. RCNN first extracts the whole-image feature map with an ImageNet [5] pre-trained CNN and then predicts a confidence score as well as box position per proposal via a RoI-pooling operation on the whole-image feature map [13]. Faster-RCNN [28] further proposes to learn to generate region proposals with a CNN and share the feature representation with detection, which leads to further gain in both performance and speed. Proposal based object detectors achieve outstanding performances in many public benchmarks [7, 29]. However, the typical two-stage pipeline makes it unsuitable for real-time applications.

### 2.2. Single-stage Object Detection

Different from the two-stage detection pipeline that first predicts proposals and then refines them, single-stage detectors directly predict the finals detections. YOLO [27] and SSD [23] are the most representative works with real-time speed. YOLO [27] divides the image into sparse grids and makes multi-class and multi-scale predictions per grid cell. SSD [23] additionally uses pre-defined object templates (or *anchors*) to handle large variance in object size and shape. For single-class object detection, DenseBox [17] and EAST [38] show that single-stage detector also works well without using manually designed anchors. They both adopt the fully-convolutional network architecture [24] to make dense predictions, where each pixel location corresponds to one object candidate. Recently RetinaNet [22] shows that single-stage detector can outperform two-stage detector if class imbalance problem during training is resolved properly. Our proposed detector follows the idea of single-stage dense object detector, while further extends these ideas to real-time 3D object detection by re-designing the input representation, network architecture, and output parameterization. We also remove the hyper parameter of pre-defined object anchors by re-defining the objective function of object localization, which leads to a simpler detection framework.

## 2.3. 3D Object Detection from Point Clouds

Vote3D [37] uses sliding window on sparse volumes in a 3D voxel grid to detect objects. Hand-crafted geometry features are extracted on each volume and fed into an SVM classifier [34]. Vote3Deep [6] also uses the voxel representation of point clouds, but extracts features for each volume with 3D Convolutional Neural Networks [35]. The main issue with voxel representations is efficiency, as the 3D voxel grid usually has high dimensionality. In contrast, VeloFCN [20] projects the 3D point cloud to front-view and gets a 2D depth map. Vehicles are then detected by applying a 2D CNN on the depth map. Recently MV3D [3] also uses the projection representation. It combines CNN features extracted from multiple views (front view, bird's eye view as well as camera view) to do 3D object detection. However, hand-crafted features are computed as the encoding of the rasterized images. Our proposed detector, however, uses the bird's eye view representation alone for real-time 3D object detection in the context of autonomous driving, where we assume that all objects lie on the same ground.

## 3. PIXOR Detector

In this paper we propose an efficient 3D object detector that is able to produce very accurate bounding boxes given LIDAR point clouds. Our bounding box estimates not only contain the location in 3D space, but also the heading angle, since predicting this accurately is very important for autonomous driving. We exploit a 2D representation of LIDAR point clouds, as it is more compact and thus amenable to real-time inference compared with 3D voxel grid representation. An overview of the proposed 3D object detector is shown in Figure 1. In the following we introduce our input representation, network architecture and discuss how we encode the oriented bounding boxes. We also present details about the learning and inference of the detector.

### 3.1. Input Representation

Standard convolutional neural networks perform discrete convolutions and thus assume that the input lies on a grid. 3D point clouds are however unstructured, and thus standard convolutions cannot be directly applied. One option is to use voxelization to form a 3D voxel grid, where each voxel cell contains certain statistics of the points that lie within that voxel. To extract feature representation from this 3D voxel grid, 3D convolution is often used. However, this can be very expensive in computation as we have to slide the 3D convolution kernel along three dimensions. This is also unnecessary because the LIDAR point cloud is so sparse that most voxel cells are empty.

Instead, we can represent the scene from the bird's eye view (BEV) alone. By reducing the free degree from 3 to 2, we don't lose information in point cloud as we can
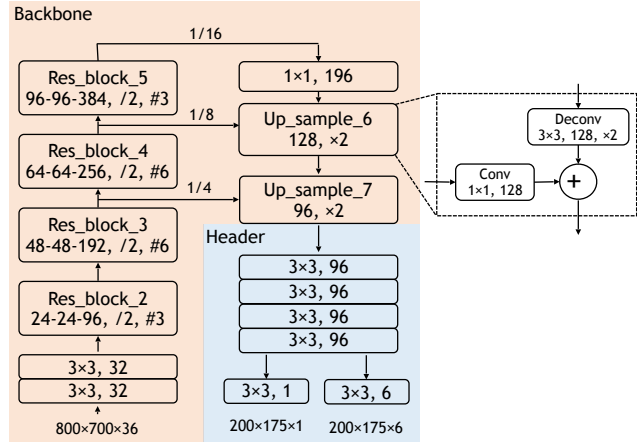


Figure 2. The network architecture of PIXOR.

still keep the height information as channels along the 3rd dimension (like the RGB channels of 2D images). However, effectively we get a more compact representation since we can apply 2D convolution to the BEV representation. This dimension reduction is reasonable in the context of autonomous driving as the objects of interest are on the same ground. In addition to computation efficiency, BEV representation also have other advantages. It eases the problem of object detection as objects do not overlap with each other (compared with front-view representation). It also keeps the metric space, and thus the network can exploit priors about the physical dimensions of objects.

Here we elaborate the projection and discretization process of our BEV representation. We first define the 3D physical dimension $L \times W \times H$ of the scene that we are interested to detect objects. The 3D points within this 3D rectangular space are then discretized with a resolution of $d_L \times d_W \times d_H$ per cell. The value for each cell is encoded as occupancy (i.e., $1$ if there exist points within this cell, and $0$ otherwise). After discretization, we get a 3D occupancy tensor of shape $\frac{L}{d_L} \times \frac{W}{d_W} \times \frac{H}{d_H}$. We also encode the reflectance (real value normalized to be within $[0, 1]$) of the LIDAR point in a similar way. The only difference is that for reflectance we set $d_H = H$. Our final representation is a combination of the 3D occupancy tensor and the 2D reflectance image, whose shape is $\frac{L}{d_L} \times \frac{W}{d_W} \times (\frac{H}{d_H} + 1)$.

### 3.2. Network Architecture

PIXOR uses a fully-convolutional neural network designed for dense oriented 3D object detection. We do not adopt the commonly used proposal generation branch [11, 28, 4, 3]. Instead, the network outputs pixel-wise predictions at a single stage, with each prediction corresponds to a 3D object estimate. As a result the recall rate of PIXOR is $100\%$ by definition. Thanks to the fully-convolutional architecture, such dense predictions can be computed very

efficiently. In terms of the encoding of the 3D object in the network prediction, we use the direct encoding without resorting to pre-defined object anchors [11, 28, 4], which works pretty well in practice. All these designs make PIXOR extremely simple and generalize well thanks to zero hyper-parameter in network architecture. To be specific, there is no need to design object anchors, nor to tune the number of proposals passed from the first stage to the second stage along with the corresponding Non-Maximum-Suppression threshold.

We show the architecture of PIXOR in Figure 2. The whole architecture can be divided into two sub-networks: a *backbone network* and a *header network*. The *backbone network* is used to extract general representation of the input in the form of convolutional feature maps. It has high representation capacity to learn robust feature representation. The *header network* is used to make task-specific predictions, and in our case it has a single-branch structure with multi-task outputs: a score map representing the object class probability, and the geometry maps encoding the size and shape of the oriented 3D objects.

### 3.2.1 Backbone Network

Convolutional Neural Networks are typically composed of convolutional layers and pooling layers. Convolutional layers are used to extract an over-complete representation of the input feature, while pooling layers are used to down-sample the feature map size to save computation and help create more robust representation. The backbone networks in many image based object detectors usually have a down-sampling factor of 16 [28, 11, 4], and is typically designed to have fewer layers in high-resolution and more layers in low-resolution. It works well for images as objects are typically large in pixel size. However, this will cause a problem in our case as objects can be very small. A typical vehicle has size of $18 \times 40$ pixels when using a discretization resolution of $0.1m$. After $16\times$ down-sampling, it covers around 3 pixels only.

One direct solution is to use fewer pooling layers. However, this will decrease the size of the receptive field of each pixel in the final feature map, which limits the representation capacity. Another solution is to use dilated convolutions. However, this would lead to checkerboard artifacts [25] in high-level feature maps. Our solution is simple, we use $16\times$ downsampling factor, but make two modifications. First, we add more layers with small channel number in lower levels to extract more fine-detail information. Second, we adopt a top-down branch similar to FPN [21] that combines high-resolution feature maps with low-resolution ones so as to up-sample the final feature representation.

We show the backbone network architecture in Figure 2. To be specific, we have in total five blocks of layers in the backbone network. The first block consists of two convo-
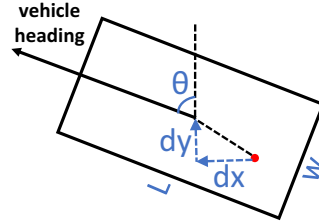


Figure 3. The geometry output parameterization for one positive sample (the red pixel). The learning target is $\{\cos(\theta), \sin(\theta), \log(dx), \log(dy), \log(w), \log(l)\}$, which is normalized before-hand over the training set to have zero mean and unit variance.

lutional layers with channel number 32 and stride 1. The second to fifth blocks are composed of residual layers [15] (with number of layers equals to 3, 6, 6, 4 respectively). The first convolution of each residual block has a stride of 2 in order to down-sample the feature map. In total we have a down-sampling factor of 16. To up-sample the feature map, we add a top-down path that up-samples the feature map by 2 each time. This is then combined with the bottom-up feature maps at the corresponding resolution via pixel-wise summation. Two up-sampling layers are used, which leads to a final feature map with $4\times$ down-sampling factor with respect to the input.

### 3.2.2 Header Network

The header network is a multi-task network that handles both object recognition and localization. It is designed to be small and efficient. The classification branch outputs 1-channel feature map followed with sigmoid activation function. The regression branch outputs 6-channel feature maps without non-linearity. There exists a trade-off in how many layers to share weights among the two branches. On the one hand, we'd like the weights to be utilized more efficiently. On the other hand, since they are different sub-tasks, we want them to be more separate and more specialized. We make an investigative experiment of this trade-off in next chapter, and find that sharing weights of the two tasks leads to slightly better performance.

We parameterize each object as an oriented bounding box $b$ as $\{\theta, xc, yc, w, l\}$, with each element corresponds to the heading angle (within range $[-\pi, \pi]$), the object's center position, and the object's size. Compared with cuboid based 3D object detection, we omit position and size along the $Z$ axis because in applications like autonomous driving the objects of interest are constrained to the same ground plane and therefore we only care about how to localize it on that plane (this setting is also known as 3D localization in some literatures [3]). Given such parameterization, the representation of the regression branch is $\{cos(\theta), sin(\theta), dx, dy, w, l\}$ for each pixel at position $(px, py)$ (shown as the red point in Figure 3). Note that

| Method | Time | $AP_{0.7}$, val (%) | | | | $AP_{KITTI}$, val (%) | | | $AP_{KITTI}$, test (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0-30 | 30-50 | 50-70 | 0-70 | Easy | Moderate | Hard | Easy | Moderate | Hard |
| VeloFCN [20] | 1000 ms | - | - | - | - | - | - | - | 0.15 | 0.33 | 0.47 |
| 3D FCN [19] | >5000 ms | - | - | - | - | - | - | - | 69.94 | 62.54 | 55.94 |
| MV3D [3] | 240 ms | 80.53 | 53.68 | 1.36 | 66.32 | 86.18 | 77.32 | 76.33 | 85.82 | 77.00 | 68.94 |
| MV3D+im [3] | 360 ms | 76.16 | 58.41 | 4.87 | 65.31 | 86.55 | 78.10 | **76.67** | **86.02** | 76.90 | 68.49 |
| PIXOR | **93 ms** | **87.68** | **60.05** | **21.62** | **75.74** | **86.79** | **80.75** | 76.60 | 81.70 | **77.05** | **72.95** |

Table 1. **Evaluation of 3D object detectors that use LIDAR as input on KITTI bird's eye view benchmark.** MV3D+im uses image as additional input. We use $AP_{0.7}$ (AUC of PR Curve with 0.7 IoU thresholds on all cars) and $AP_{KITTI}$ (official KITTI metric that computes the AUC with 11 sampling points only, evaluated on three sub-sets) as evaluation metrics on *val* [3] and *test* set. We also show fine-grained evaluation with regard to different ranges (distance in meters to the ego-car), which makes more sense in 3D detection.

the heading angle is factored into two correlated values to enforce the angle range constraint. We decode the $\theta$ as $atan2(sin(\theta), cos(\theta))$ during inference. $(dx, dy)$ corresponds to the position offset from the the pixel position to the object center. $(w, l)$ corresponds to the object size. It is worth notice that the values for the object position and size are in real-world metric space. The learning target is $\{\cos(\theta), \sin(\theta), \log(dx), \log(dy), \log(w), \log(l)\}$, which is normalized before-hand over the training set to have zero mean and unit variance. In next chapter we further find out that decoding the oriented box at training time and computing regression loss directly on the coordinates of four box corners can bring additional performance gain.

### 3.3. Learning and Inference

We adopt the commonly used multi-task loss [11] to train the full network. Specifically, we use cross-entropy loss on the classification output $p$ and a smooth $\ell_1$ loss on the regression output $q$. We sum the classification loss over all locations on the output map, while the regression loss is computed over positive locations only.

$$L_{\text{total}} = \text{cross\_entropy}(p, y_{cls}) + \text{smooth}_{L_1}(q - y_{reg}) \quad (1)$$

$$\text{cross\_entropy}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise,} \end{cases} \quad (2)$$

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

Note that we have severe class imbalance since a large proportion of the scene belongs to background. To stabilize the training process, we adopt the focal loss with the same hyper-parameter as [22] to re-weight all the samples. In the next chapter, we also propose a biased sampling strategy for positive samples that leads to better convergence. During inference, we feed the computed BEV representation from LIDAR point cloud to the network and get one channel of confidence score and six channels of geometry information. We then decode the geometry information

into oriented bounding boxes only on positions whose confidence scores are above certain threshold. Non-Maximum-Suppression is used to get the final detections, where the overlap is computed as the Intersection-Over-Union of two oriented boxes.

## 4. Experiments

We conduct three types of experiments here. First, we compare PIXOR with other state-of-the-art 3D object detectors on the public KITTI bird's eye view object detection benchmark [10]. We show that PIXOR achieves best performance both in accuracy and speed compared with all previously published methods. Second, we conduct an ablation study of PIXOR in three aspects: optimization, network architecture, and speed. Third, we verify the generalization ability of PIXOR by applying it to a new large-scale vehicle detection dataset for autonomous driving.

### 4.1. BEV Object Detection on KITTI

#### 4.1.1 Implementation Details

We set the region of interest for the point cloud to $[0, 70] \times [-40, 40]$ meters and do bird's eye view projection with a discretization resolution of 0.1 meter. We set the height range to $[-2.5, 1]$ meters in LIDAR coordinates and divide all points into 35 slices with bin size of 0.1 meter. One reflectance channel is also computed. As a result, our input representation has the dimension of $800 \times 700 \times 36$. We use data augmentation of rotation between $[-5, 5]$ degrees along the Z axis and random flip along $X$ axis during training. Unlike other detectors [3] that initialize the network weights from a pre-trained model, we train our network from scratch without resorting to any pre-trained model.

#### 4.1.2 Evaluation Metric

We use Average Precision (AP) computed at 0.7 Intersection-Over-Union (IoU) as our evaluation metric in all experiments unless mentioned otherwise. We compute the AP as Area Under Precision-Recall Curve (AUC) [8]. We evaluate on 'Car' category and ignore 'Van', 'Truck', 'Tram' and 'DontCare' categories in KITTI during evaluation, meaning that we don't count True Positive (TP) or
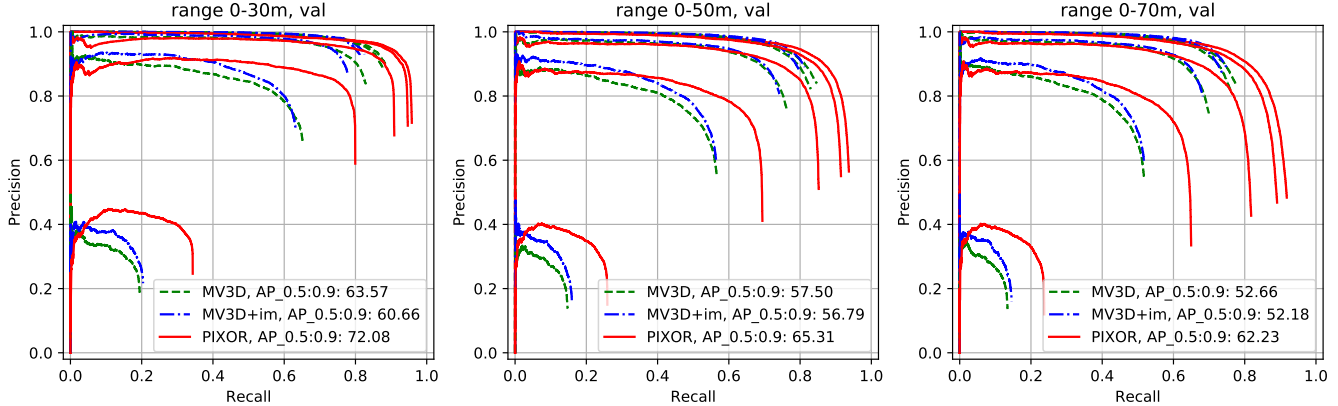
Figure 4. **Evaluation of PIXOR and MV3D [3] on KITTI** *val* **set.** For each approach, we plot 5 Precision-Recall curves corresponding to 5 different IoU thresholds between 0.5 and 0.9, and report the averaged AP (%). We compare in three different ranges.

False Negative (FN) on them. Note that the metric we use is different from what KITTI reports in the following two aspects: (1) KITTI computes AP by sampling at 11 linearly sampled recall rates (from $0\%$ to $100\%$), which is a rough approximation to real AUC. (2) KITTI divides labels into three subsets with image-based definition (e.g, object height in pixels, visibility in image), and reports AP on each subset, which doesn't suit pure LIDAR based object detection. In contrast, we evaluate on *all* labels within the region of interest, and do fine-grained evaluation with respect to ranges (object distance to the ego-car).

### 4.1.3 Evaluation Result

We compare with 3D object detectors that use LIDAR on KITTI benchmark: VeloFCN [20], 3D FCN [19] and MV3D [3]. We show the evaluation results in Table 1. From the table we see that PIXOR largely outperforms other approaches in AP at 0.7 IoU within 70 meters range, leading the second best by over $9\%$. We also show evaluation results with respect to ranges, and show that PIXOR outperforms more in the long range. On KITTI's test set, PIXOR outperforms MV3D in *moderate* and *hard* settings. Note that PIXOR has a lower AP in *easy* setting on the test set, which is caused by the instability of the 11-point AP metric.

Since MV3D is the best approach among all state-of-the-art methods, we'd like to make a more detailed comparison using the AUC based AP metric. We show fine-grained Precision-Recall (PR) curves of both PIXOR and MV3D in Figure 4. From the figure, we get the following observations: (1) PIXOR outperforms MV3D in all IoU thresholds, especially at very high IoU like 0.8 and 0.9, showing that even without using proposal, PIXOR can still get super-accurate object localization, compared with the two-stage proposal based detector MV3D. (2) PIXOR has similar precision with MV3D at low recall rates. However, when it comes to higher recall rates, PIXOR shows huge advantage. At the same precision rate of the end point of MV3D's

| Classification | Regression | $AP_{0.7}$ | $AP\_avg$ |
|---|---|---|---|
| cross-entropy | smooth_L1 | 73.46% | 55.25% |
| focal | smooth_L1 | 74.93% | 55.89% |
| focal | decoding | 71.05% | 53.05% |
| focal | smooth_L1 + decode (f.t.) | **77.16%** | **58.31%** |

Table 2. Ablation study of different loss functions. **smooth_L1 + decode (f.t.)** means that the network is trained with smooth L1 loss first, and then fine-tuned by replacing the smooth L1 loss with decoding loss.

| Training Samples | Data Aug. | $AP_{0.7}$ | $AP\_avg$ |
|---|---|---|---|
| all pixels | none | 71.10% | 53.99% |
| ignore boundary pixels | none | 74.54% | 55.79% |
| ignore boundary pixels | rotate + flip | **74.93%** | **55.89%** |

Table 3. Ablation study of different data sampling strategies.

curve, PIXOR generally has over $5\%$ higher recall rate in all ranges. This shows that dense detector like PIXOR does have an advantage of higher recall rate, compared with two-stage detectors. (3) In the more difficult long range part, PIXOR still shows superiority over MV3D, which justifies our input representation design that reserves the 3D information well and our network architecture design that captures both fine details and regional context.

### 4.2. Ablation Study

We show an extensive ablation study of the proposed detector in terms of optimization, network architecture, speed and failure mode.

#### 4.2.1 Experimental Setting

Since we also compare with other state-of-the-art methods on the *val* set, it would be inappropriate to do the ablation study on the same set. Therefore we resort to KITTI Raw dataset [9] and randomly pick 3000 frames that are not overlapped with both *train* and *val* sets in KITTI object detection dataset, which we call *val-dev* set. We report ablation study results on this set. We use AP at 0.7 IoU as well as

| Backbone Network | $AP\_avg$ |
|---|---|
| pvanet | 51.28% |
| resnet-50 | 53.03% |
| vgg16-half | 54.46% |
| resnet-lidar | **55.07%** |

Table 4. Ablation study of different backbone networks.
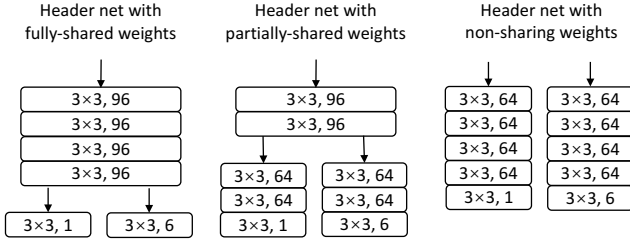


Figure 5. Three versions of header network architectures.

| Header Network | $AP_{0.7}$ | $AP\_avg$ |
|---|---|---|
| non-sharing | 74.93% | 55.89% |
| partially-shared | 74.66% | 55.75% |
| fully-shared | **75.13%** | **56.04%** |

Table 5. Ablation study of different header network architectures.

| | digitization | network | NMS | total |
|---|---|---|---|---|
| time (ms) | 17 | 66 | 10 | 93 |

Table 6. The detailed timing analysis of PIXOR.

AP averaged from 0.5 to 0.95 IoUs (with a stride of 0.05) as the evaluation metrics.

### 4.2.2 Optimization

We investigate into four topics here: the classification loss, the regression loss, the sampling strategy, and data augmentation.

**Classification loss** RetinaNet [22] proposes the focal loss to re-weight samples for dense detector training. For simplicity, we use their hyper-parameter setting. We show results in Table 2, and find that focal loss improves the AP_0.7 by more than 1%.

**Regression loss** For the box regression part, our default choice is a smooth L1 loss [11] on every dimension of the regression targets. We also propose the *decoding loss*, where the output targets are first decoded into oriented boxes and then smooth L1 loss is computed on the (x, y) coordinates of four box corners directly with regard to ground-truth. Since the decoding the oriented box from regression targets is just a combination of some normal mathematic operations, this decoding process is differentiable and gradients can be back-propagated through this decoding process. We believe that this decoding loss is more end-to-end and implicitly balances different dimensions of the regression targets. In the results shown in Table 2, we show that directly training with the decoding loss doesn't work very well. However, training with conventional loss first and then fine-tune with the proposed decoding loss helps improve the performance a lot.

**Data sampling and augmentation** When training dense detectors, one issue is how to define positive and negative samples. In proposal based approaches, this is defined by the IoU between proposal and ground-truth. Since PIXOR is a proposal-free method, we go for a more straightforward sampling strategy: all pixels inside the ground-truth are positive samples while outside pixels are negative samples. This simple definition already gives decent performance. However, one issue with this definition is that the variance of regression targets could be large for pixels near the object boundary. Therefore we propose to sub-sample the pixels, i.e, to ignore pixels near object boundary during training. Specifically, we zoom the ground-truth object box twice with 0.3 and 1.2 zooming factors respectively, and ignore all pixels that lie between these two zoomed boxes. From the results shown in Table 3, we find that this sub-sampling strategy is beneficial to stabilize training. We also find that our data augmentation for KITTI helps a bit since PIXOR is trained from scratch instead of from a pre-trained model.

### 4.2.3 Network Architecture

**Backbone network** We first compare different backbone networks: vgg16 with half channel number [31], pvanet [16], resnet-50 [14], and resnet-lidar as presented in Figure 2. All of these backbone networks run below 100 milliseconds. All backbone networks except for *vgg16-half* uses residual unit as building blocks. We find that *vgg16-half* converges faster in *train* set and gets lower training loss than all other residual variants, but the performance drops quite a lot when evaluated on *val* set. This doesn't happen to the other three residual-based networks. We conjecture that this is because *vgg16-half* is more prone to overfit without implicit regularization imposed by residual connections.

**Header network** We also compare different structures for the header network. We investigate into how much we should share the parameters for the multi-task outputs. Three versions of header network are proposed with different extent of weight sharing in Figure 5 and compared in Table 5. All these three versions have very close number of parameters. We find that fully-shared structure works best as it utilizes the parameters most efficiently.

### 4.2.4 Speed

We show detailed timing analysis of PIXOR in Table 6 for one single frame. The computation of input representation and final NMS are both processed on CPU in Python. The network time is measured on a NVIDIA Titan Xp GPU and averaged over 100 non-sequential frames in KITTI.
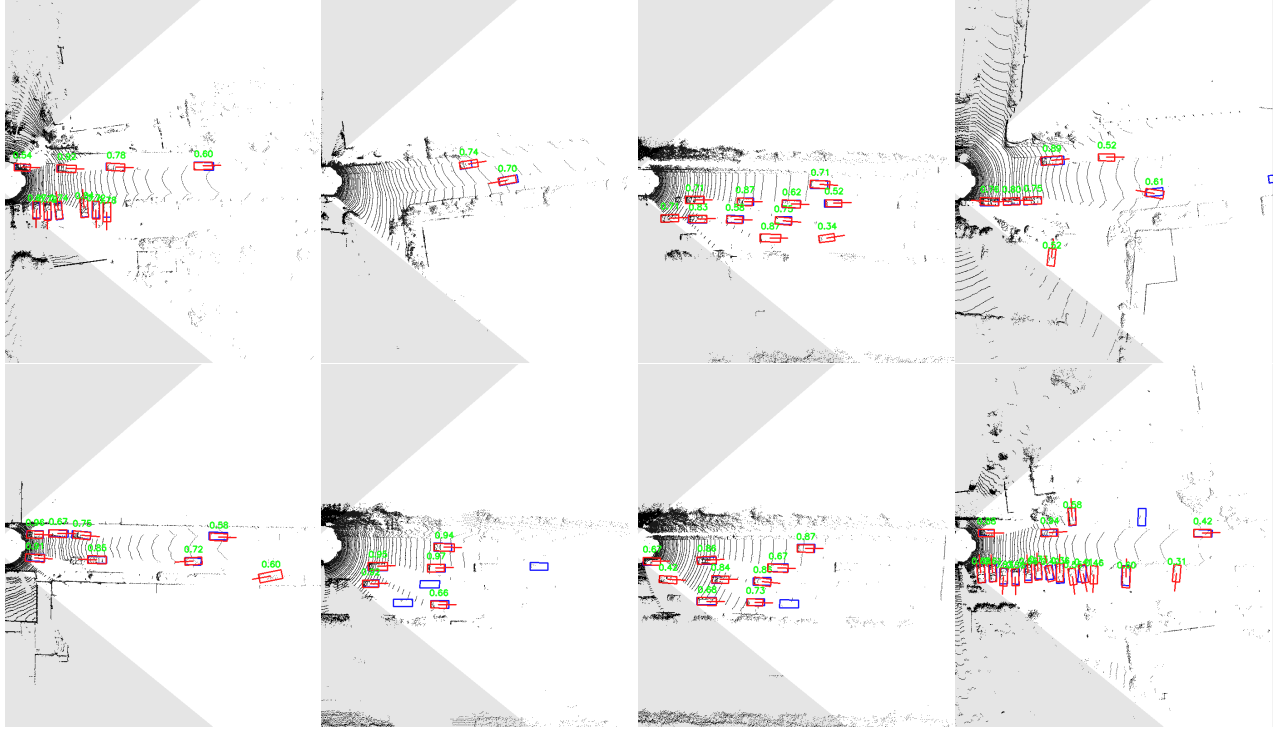
Figure 6. **Example detection results of PIXOR on KITTI Object** *val* **set.** The detection is in red color, while the ground-truth is in blue color. Gray area is out of the scope of camera view and therefore has no labels.

### 4.2.5 Failure Mode

We show some detection results of PIXOR in Figure 6, and discover some failure modes. In general PIXOR will fail when there's no observed LIDAR points. In longer range we have very few evidence of the object, and therefore object localization becomes inaccurate, leading to false positives at higher IoU thresholds.

## 4.3. BEV Object Detection on Large-scale Dataset

### 4.3.1 ATG4D Dataset

We also collect a large-scale 3D vehicle detection dataset called ATG4D which has different sensor configuration from KITTI and is collected in North-American cities. There are in total 6500 sequences collected, with are divided into 5000/500/1000 as train/val/test splits. The training sequences are sampled at 10 Hz into frames, while validation and testing sequences are sampled at 0.5Hz. As a result, there are over 1.2 million frames in training set, 5969 and 11969 frames in the val and test sets. All vehicles are annotated with bird's eye view bounding boxes labels.

### 4.3.2 Evaluation Result

We apply PIXOR to ATG4D with the "vgg-half" backbone network because ATG4D has much more training data than KITTI and therefore preventing over-fitting is not our main concern. We train and test the detector at ranges up to 100

| Method | $AP\_0.7$ |
|---|---|
| Baseline [27] | 69.4% |
| PIXOR | 73.3% |

Table 7. Evaluation of PIXOR on ATG4D.

meters, and increase the input discretization resolution to 0.2 meter. As a result, the detector still runs at $> 10$ Hz. In comparison, we build a YOLO-like [27] baseline detector with a customized backbone network on ATG4D, and add object anchors and multi-scale feature fusion to further improve the performance. The evaluation results are listed in Table 7, where we show that PIXOR outperforms the baseline by 3.9% in AP_0.7, proving that PIXOR is simple and easy to generalize, with no hyper-parameter to tune.

## 5. Conclusion

In this paper we propose a real-time 3D object detector called PIXOR that operates on LIDAR point clouds. PIXOR is a single-stage, proposal-free, dense object detector that achieve extreme simplicity in the context of 3D object localization for autonomous driving. PIXOR takes bird's eye view representation as input for efficiency in computation. A novel decoding loss is proposed that suits the 3D object localization task better. We evaluate PIXOR on the challenging KITTI benchmark as well as a large-scale vehicle detection dataset ATG4D, and show that it outperforms the other methods by a large margin in terms of Average Precision (AP), while still runs at 10 FPS.

# References

[1] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016. 1

[2] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals using stereo imagery for accurate object class detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 1

[3] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1, 3, 4, 5, 6

[4] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, pages 379–387, 2016. 1, 2, 3, 4

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 2

[6] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1355–1361. IEEE, 2017. 1, 3

[7] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 2

[8] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. 5

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 6

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012. 2, 5

[11] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015. 1, 2, 3, 4, 5, 7

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 1, 2

[13] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 2

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 7

[15] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 4

[16] S. Hong, B. Roh, K.-H. Kim, Y. Cheon, and M. Park. PVANet: Lightweight deep neural networks for real-time object detection. *arXiv preprint arXiv:1611.08588*, 2016. 7

[17] L. Huang, Y. Yang, Y. Deng, and Y. Yu. Densebox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*, 2015. 2

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[19] B. Li. 3d fully convolutional network for vehicle detection in point cloud. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 1513–1518. IEEE, 2017. 5, 6

[20] B. Li, T. Zhang, and T. Xia. Vehicle detection from 3d lidar using fully convolutional network. In *Robotics: Science and Systems*, 2016. 1, 3, 5, 6

[21] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 4

[22] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017. 2, 5, 7

[23] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016. 1, 2

[24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 4

[25] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016. 4

[26] J. Pont-Tuset, P. Arbelaez, J. T. Barron, F. Marques, and J. Malik. Multiscale combinatorial grouping for image segmentation and object proposal generation. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 39(1):128–140, 2017. 2

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016. 1, 2, 8

[28] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015. 1, 2, 3, 4

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 2

[30] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations (ICLR 2014)*, 2014. 2

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 7

[32] S. Song and J. Xiao. Sliding shapes for 3d object detection in depth images. In *European conference on computer vision*, pages 634–651. Springer, 2014. 1

[33] S. Song and J. Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016. 1

[34] J. A. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999. 3

[35] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015. 3

[36] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. 2

[37] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, 2015. 1, 3

[38] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. East: An efficient and accurate scene text detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2