Real-time 3D Object Detection
Project Report

Varun Dhuldhoya
New York University
vyd208@nyu.edu

Sai Likhith K
New York University
slk522@nyu.edu

1. Problem Description

3D object detection has become inevitable for day-to-day applications be it autonomous driving or constructing images or even for augmented reality since it directly links to environmental understanding and therefore builds the base for prediction and motion planning. But, in the case of real-time 3D object detection, we need to deal with highly sparse data considering its usage amongst real-life objects. In those cases, it is extremely inefficient to process data in general methods of detection and also to reduce the noise during this process is tiresome.
Thus, we need to introduce efficient techniques in order to perform 3D object detection with greater accuracy and speed. So, we shall introduce a technique named "YOLO - You Only Look Once" considering the speed of execution in real life. We shall use it as a fast 2D standard object detector for RGB images, by a specific complex regression strategy to estimate multi-class 3D boxes in Cartesian space.

2. Literature Survey

2.1. Complex-YOLO: An Euler-Region-Proposal for Real-time 3D Object Detection on Point [1]

Clouds Lidar based 3D object detection is inevitable for autonomous driving, because it directly links to environmental understanding and therefore builds the base for prediction and motion planning. The capacity of inferencing highly sparse 3D data in real time is an ill-posed problem for lots of other application areas besides automated vehicles, e.g. augmented reality, personal robotics or industrial automation. We introduce Complex-YOLO, a state of the art real-time 3D object detection network on point clouds only. In this work, is described a network that expands YOLOv2, a fast 2D standard object detector for RGB images, by a specific complex regression strategy to estimate multi-class 3D boxes in Cartesian space. Thus,it proposes a specific Euler-Region-Proposal Network (E-RPN) to estimate the pose of the object by adding an imaginary and a real fraction to the regression network

2.2. Expandable YOLO: 3D Object Detection from RGB-D Image [2]

This paper aims at constructing a light-weight object detector that inputs a depth and a color image from a stereo camera. Specifically, by extending the network architecture of YOLOv3 to 3D in the middle, it is possible to output in the depth direction. In addition, Intersection over Union (IoU) in 3D space is introduced to confirm the accuracy of region extraction results

2.3. YOLO 3D: End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud [4]

In this paper, the one-shot regression meta-architecture in the 2D perspective image space is extended to generate oriented 3D object bounding boxes from LiDAR point cloud. The 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

2.4. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection [3]

In this work, the need of manual feature engineering for 3D point clouds is replaced by VoxelNet, a generic 3D detection network that unifies feature extraction and bounding box prediction into a single stage, end-to-end trainable deep network. Specifically, VoxelNet divides a point cloud into equally spaced 3D voxels and transforms a group of points within each voxel

into a unified feature representation through the newly introduced voxel feature encoding (VFE) layer. In this way, the point cloud is encoded as a descriptive volumetric representation, which is then connected to a RPN to generate detections.

3. Description of Data sets

We will use the 3D KITTI detection dataset, this dataset contains:

- 7481 training images
- and 7518 test images.
- 500,000+ records which includes the datasets, calibrations and evaluation results images.

The dataset will include the following:

3.1. Velodyne point clouds (29 GB)

Information about the surrounding for a single frame gathered by Velodyne HDL64 laser scanner. This is the primary data we use as input data to the YOLO 3D model.100k points per frame, stored as a binary float matrix. Here, data contains 4*num values, where the first 3 values correspond to x,y and z, and the last value is the reflectance information. All scans are stored row-aligned, meaning that the first 4 values correspond to the first measurement.

3.2. Training labels of object data set (5 MB)

Input label to the YOLO3D model. Stored as a text file.

3.3. Camera calibration matrices of object data set (16 MB)

For visualization of predictions. Camera, Camera-to-GPS/IMU, Camera-to-Velodyne, stored as text file. Used for calibrating and rectifying the data captured by the camera and sensor.

3.4. Left color images of object data set (12 GB)

For visualization of predictions. The cam- eras were one color camera stereo pairs. We use left Images corresponding to the velodyne point clouds for each frame.

4. Description of Models which are implemented

4.1. Preprocessing using LIDAR 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. 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. 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: $min(1.0, log(N+1)/log(64))$, Where N is the number of points in each grid cell.

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. 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.

4.2. Complex-YOLO

Complex-YOLO is a 3d version of the YOLOv2, one of the fastest state of the art object detectors. The Complex-YOLO network takes a birds-eye-view RGB-map as input. It uses a simplified YOLOv2 CNN architecture, extended by a complex angle regression and E-RPN, to detect accurate multi-class oriented 3D objects while still operating in real-time.

Complex YOLO 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.

4.3. Architecture

We design a new CNN architecture base on YOLOv2 to detect 3D objects in real-time, called 3DNet. It has 18 convolutional layers and 5 maxpooling layers. After each convolutional layer, there is a LReLU and BN layer, except in the last layer.

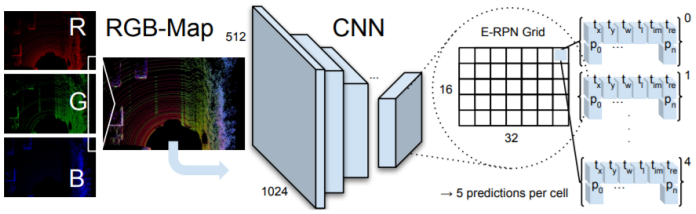


Figure 1. Complex-YOLO Pipeline

We present a slim pipeline for fast and accurate 3D box estimations on point clouds. The RGB-map is fed into the CNN.

The RGB-map is fed into the CNN (see Fig. 4). The E-RPN grid runs simultaneously on the last feature map and predicts five boxes per grid cell. Each box prediction is composed by the regression parameters t (see Fig.3) and object scores p with a general probability p0 and n class scores p1...pn.

5. Loss Function

Our network optimization loss function L is based on the the concepts from YOLO and YOLOv2 , who defined L_{Yolo} as the sum of squared errors using the introduced multi-part loss. We extend this approach by an Euler regression part L_{Euler} to get use of the complex numbers, which have a closed mathematical space for angle comparisons. This neglect singularities, which are common for single angle estimations:

$$L = L_{Yolo} + L_{Euler}$$

Where:

$$\mathcal{L}_{\text{Euler}} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left| (e^{\text{i}b_\phi} - e^{\text{i}\hat{b}_\phi})^2 \right|$$

$$= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(t_{im} - \hat{t}_{im})^2 + (t_{re} - \hat{t}_{re})^2 \right]$$

Figure 2. Loss Function Equation

Where λ_{coord} is a scaling factor to ensure stable convergence in early phases and 1_{ij}^{obj} denotes that the jth bounding box predictor in cell i has highest intersection over union (IoU) compared to ground truth for that prediction.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

The values $e^{ib\phi}, e^{\hat{ib}\phi}, t_{im}, \hat{t_{im}}, t_{re}, \hat{t_{re}}$ are taken from the Euler region proposal in the below diagram:

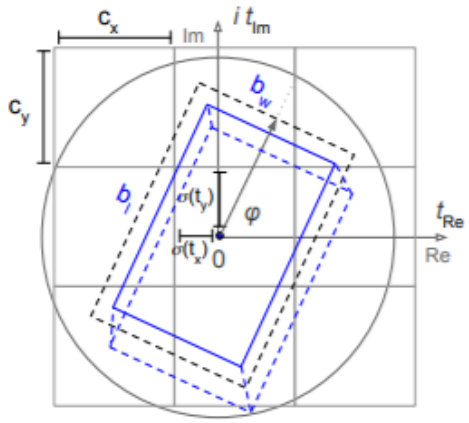


Figure 3. Loss Function Diagram

6. Training and Hyper parameters

We used the following while training the model.
* We trained the network for 100 epochs, with a batch size of 32.
* Our learning rate is 0.001.
* We wrote a function Region Loss to calculate the training loss of the model for 7 classes and we used 5 anchors.
* To classify the objects, we used a threshold of 0.7 for cars and 0.5 for rest of the classes.

6.1. Adam's Optimizer

Since, we have large amounts of data, and Adam's hyper-parameters have intuitive interpretation and typically require little tuning, we have used Adam Optimizer. Adam is one of the most popular optimiser which have a bias correction which helps it to have a better accuracy. It's default hyper parameters perform for a wider range of models. Though it has convergence issue, we have used it in order to get fast predictions.

6.2. Convolution Neural Networks

We have used Darknet CNN's structure for image classification and recognition because of its high accuracy. We have modified it a little to suit complex YOLO.

6.3. Rectified Linear Unit

The main function of it is to introduce non-linear properties into the network. What it does is, it calculates the 'weighted sum' and adds direction and decides whether to 'fire' a particular neuron or not. For output layer, activation function is ReLU(Rectified Linear Unit).

6.4. Train vs validation split

Train test split in 80 percent for training and 20 percent for validation which includes 6000 and 1480 approximately.

7. Modelling and the results

Our model is developed to predict exact 3D boxes with localization and an exact heading of the objects in real-time, even if the object is based on a few points. Our implementation is based on modified version of the Darknet neural network framework. The evaluation of the Complex-YOLO on the challenging KITTI object detection benchmark is implemented. We tried running for the model where we have LIDAR image as input for the object detection. We have modelled via several stages.

```
[16] ComplexYOLO(
       (conv_1): Conv2d(3, 24, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (pool_1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (conv_2): Conv2d(24, 48, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_2): BatchNorm2d(48, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (pool_2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (conv_3): Conv2d(48, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_4): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1))
       (bn_4): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_5): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (pool_3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (conv_6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_6): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_7): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (pool_4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (conv_9): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_10): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
       (bn_10): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_11): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (pool_5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
       (conv_12): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_12): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_13): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1))
       (bn_13): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_14): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_14): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_15): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_15): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_16): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_16): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_17): Conv2d(2048, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
       (bn_17): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
       (conv_18): Conv2d(1024, 75, kernel_size=(1, 1), stride=(1, 1))
       (relu): ReLU(inplace=True)
     )
```

Figure 4. Complex YOLO Modelling using CNN

7.1. Stages

7.1.1 Bounding box for cars

This Modelling is executed on only one class i.e; cars.

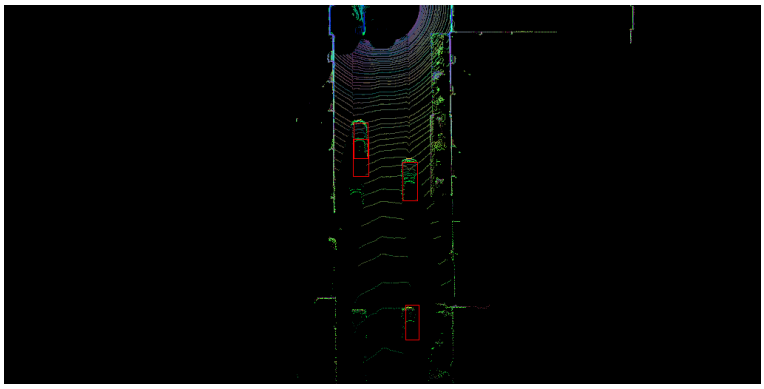


Figure 5. Modelling with only cars

7.1.2 Bounding box for cars with labels

This Modelling is executed on only one class i.e; cars.

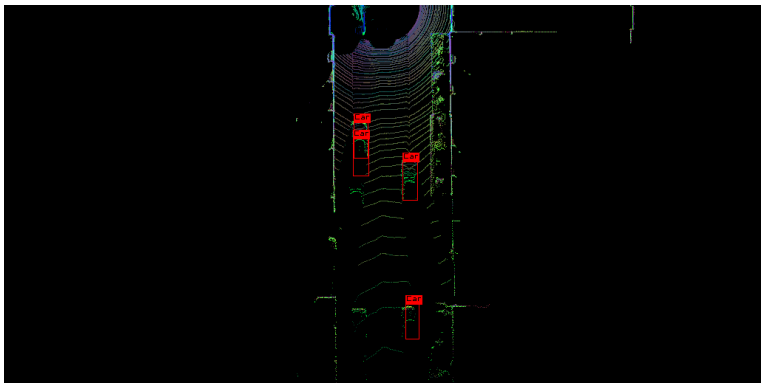


Figure 6. Modelling with only cars with labels

7.1.3 Bounding box for other categories

This Modelling is executed on all other classes like cars, cycles, pedestrians, trams.

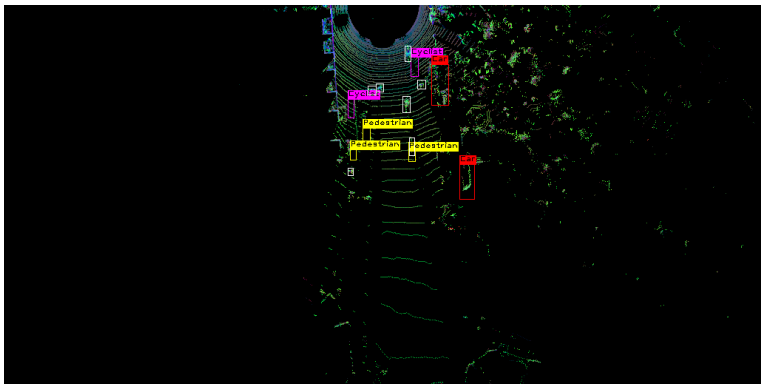


Figure 7. Modelling with other classifiers

7.1.4 improving accuracy

Initially, we faced overfitting, later we had trained using less epochs compared to previous iteration and this time we are able to recognize things similar to the actual box. This Modelling is executed with better accuracy of the bounding boxes with 62 epochs and training batch size of 32.

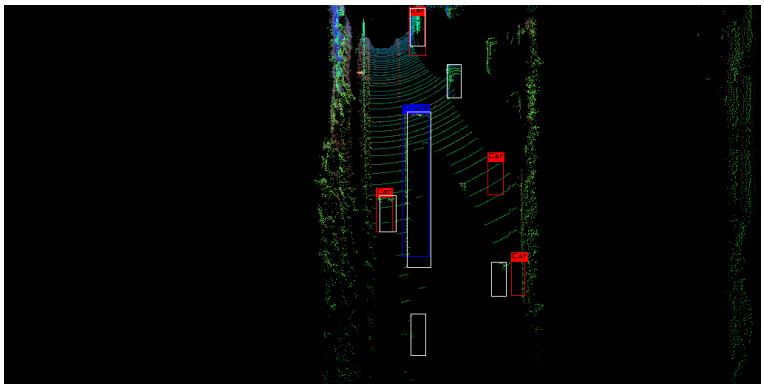
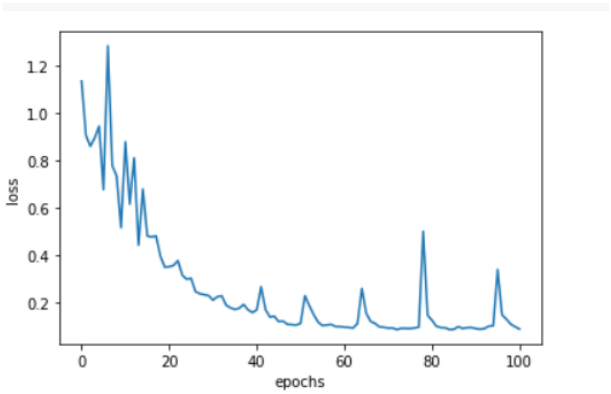


Figure 8. Modelling with better accuracy

7.2. results

This is how our results would look:

7.2.1 Loss Vs Epoch




```
Time for epoch 87 is 1581017325.6592736 with loss 0.09247976541519165
89
Time for epoch 88 is 1581017743.2069185 with loss 0.09548349678516388
90
Time for epoch 89 is 1581018158.5886335 with loss 0.09624829143285751
91
Time for epoch 90 is 1581018577.7125947 with loss 0.09175735712051392
92
Time for epoch 91 is 1581018990.5092044 with loss 0.08933086693286896
93
Time for epoch 92 is 1581019405.8048682 with loss 0.09184416383504868
94
Time for epoch 93 is 1581019819.4362295 with loss 0.10201045870780945
95
Time for epoch 94 is 1581020233.407166 with loss 0.1035606637597084
96
Time for epoch 95 is 1581020646.7162929 with loss 0.34129106998443604
97
Time for epoch 96 is 1581021059.1039379 with loss 0.14970700442790985
98
Time for epoch 97 is 1581021475.5607717 with loss 0.1315600872039795
99
Time for epoch 98 is 1581021893.1522398 with loss 0.11030305922031403
100
Time for epoch 99 is 1581022305.5333292 with loss 0.09923838824033737
101
Time for epoch 100 is 1581022723.8974829 with loss 0.08986161649227142
```

Figure 9. Loss Vs Epochs

As we can see that our graph is showing overfitting after 100 epochs, we decided to go over with 62 epoch approximately where we see the curve going minimum for the first time. Beside is the image which has the epoch 62 has minimum loss compared to other epochs.

7.3. Results vs Epochs

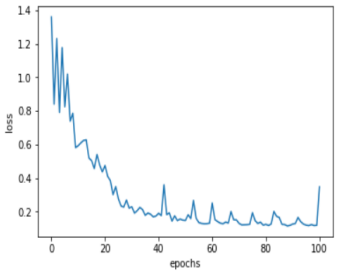
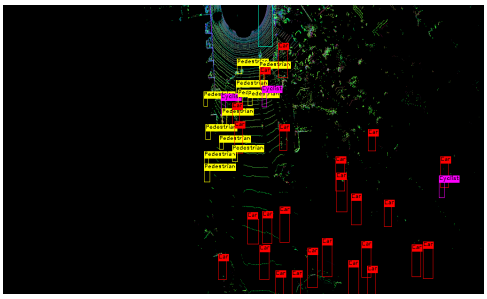
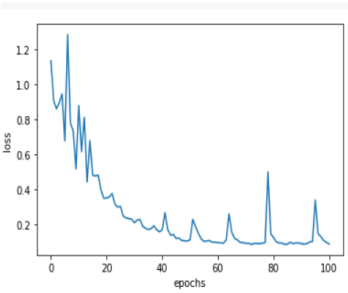
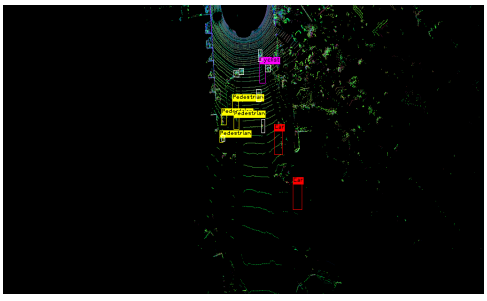
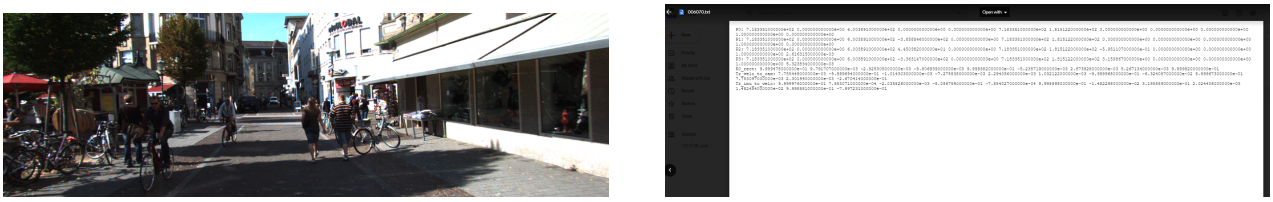
Hyper Parameters	Graphs	Predictions
• Alpha=0.001 • Batch size = 12 • Epochs= 100		
• Alpha=0.001 • Batch size = 32 • Epochs= 62		

Table 1. Overview of the parameters, loss vs epoch graph and the predictions.

This shows a good fit at 62 epoch configuration where there are more accurate predictions and less false positives. The original image is shown below.

7.3.1 Object Detection



(a) pedestrians and car (b) point cloud image

Figure 10. Label Prediction

(a) pedestrians and car (b) point cloud image

Figure 11. Without labelling

(a) label 1 (b) label 2

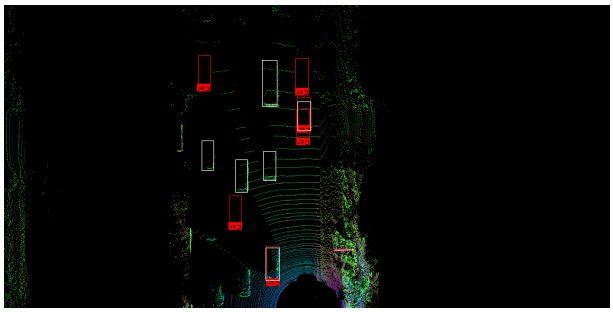
Figure 12. With labelling

We shall now, discuss on how good are our predictions and also some examples of bad predictions by our model. This can help us to understand how exactly are we defining the performance of the model.

7.3.2 Bad Prediction

This Model prediction is unable to match with actual labelling. This is missing a couple of detections which are supposed to be recognized.

(a) Tram and cars



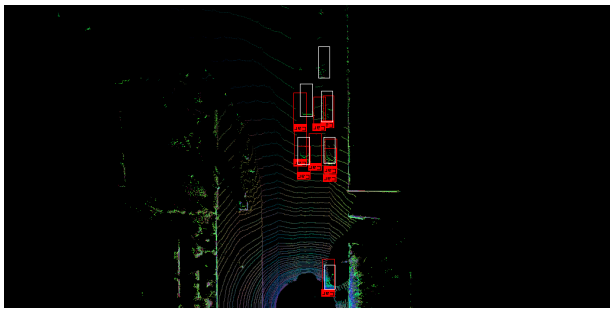
(b) point cloud image

Figure 13. Detecting Tram and cars

7.3.3 Fair Prediction

This Modelling is making sure that objects within visible range are detected properly, but hidden objects are not predicted properly.

(a) Tram and cars



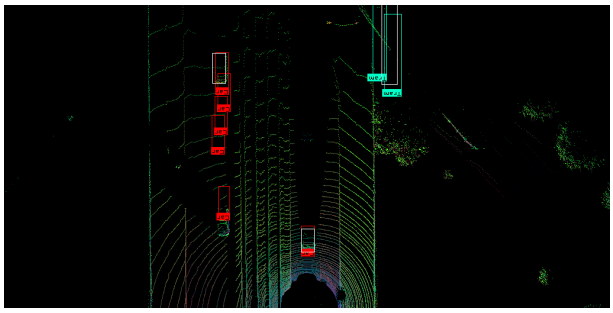
(b) point cloud image

Figure 14. Detecting Tram and cars

7.3.4 Good prediction

This Model prediction is detecting all the objects that are labelled and also some objects that are present but are not labelled in actual data. This model extrapolated in detecting beyond actual data much accurately. This is an example of real-world application of the model where we can't have always efficient labelled data as inputs. That's where we can see here remaining 2 cars, that weren't labelled using white boxes are still found accurately using these model predictions.

(a) Tram and cars



(b) point cloud image

Figure 15. Detecting Tram and cars

8. Challenges(if any)

8.1. Region Loss

The first challenge is the reduction of region loss since it is real-time

8.2. Accuracy of bounding box

Accuracy is a challenging issue here considering the noise while object detection due to multiple objects appearing at the same time in real time.

8.3. Over-fitting

With less than 4000 training point clouds, training our network from scratch will inevitably suffer from over fitting. To reduce this issue, we tried data augmentation. The augmented training data are generated on-the-fly without the need to be stored on disk.We tried Data Augmentation but found that this was not feasible as each image that has to be augmented has an associated label file, velodyne cloud image and calibration data that needs augmentation as well hence augmentation is not a good strategy for the 3D KITTI object detection dataset.

8.4. Dataset Size

The datasets are large (29GB+13GB+1GB). It took a lot of time to train. So, we implemented parallel batch training which we mentioned while proposing project plan. Even this was found to be inefficient as each epoch took an average time of 20 mins even while using 25 GB GPU storage.

9. Expected Results vs Actual Results, Conclusion and Future Scope

9.1. Expected Results vs Actual Results

We expected to implement Complex YOLO for real-time 3D Object detection on point clouds and we implemented it successfully.
As mentioned in the intermediate report, we have extended our model to detect other objects such as trams, trucks, cyclists and pedestrians. We have also included the left-color images for reference. We expected our model to predict exact bounding boxes for each object in the image but unfortunately since the dataset did not have enough data for objects other than cars it caused our model to over fit and hence our model was a little inaccurate in detecting objects other than cars.

9.2. Conclusion

In this paper we successfully implemented the Complex-YOLO: Real-time 3D Object Detection on Point Clouds. This is a real-time LiDAR based system for 3D OBB detection and classification, based on extending YOLO-v2 where 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.

9.3. Future Scope

We can say that since dataset has more car labels than the other categories, it lead to overfitting. In order to predict efficiently with general dataset for future scope, we can try to somehow augmentation techniques though it takes a lot of changes in process and varying the learning rate at several stages of training. We can also add height information to the regression, enabling a real independent 3D object detection in space for a better class distinction, accuracy and detection.

10. Repository Links

* These are the link to our repository: `https://github.com/Dhuldhoyavarun/Dlproject`

* Our Dataset links: `http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d`

References

[1] Complex-YOLO: An Euler-Region-Proposal for Real-time 3D Object Detection on Point Clouds. https://arxiv.org/pdf/1803.06199.pdf.

[2] Expandable YOLO: 3D Object Detection from RGB-D Images*. https://arxiv.org/ftp/arxiv/papers/2006/2006.14837.pdf.

[3] VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. https://arxiv.org/pdf/1711.06396.pdf.

[4] YOLO3D: End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud. https://arxiv.org/pdf/1808.02350v1.pdf.