Point Cloud Instance Segmentation

Semester Project Report

Author: Kiarash Farivar

Advisor: Mathieu Salzmann

Assistant: Edoardo Remelli

EPFL

5th June 2020

1 Introduction

The goal of this project was finding different instances of pipes using 3D point cloud data. The 3D point cloud was generated by a Lidar scanner. An example of one scene can be seen in figure 1 .

The main challenge while working with point cloud data compared to 2D images is their format. In a 2D image all pixels are ordered according to their location in the image but a point cloud data is an unordered set of points in 3D space this is due to the fact that scans of scenes using sensors like lidar only capture the surface of the objects and thus the results are very sparse and it is unreasonable to store them in an ordered format. One idea might be to try to order these sets in a 3D tensor and then apply usual methods such as 3D convolutional networks. But due to the sparsity this adds a lot extra unnecessary computation time and is not plausible.

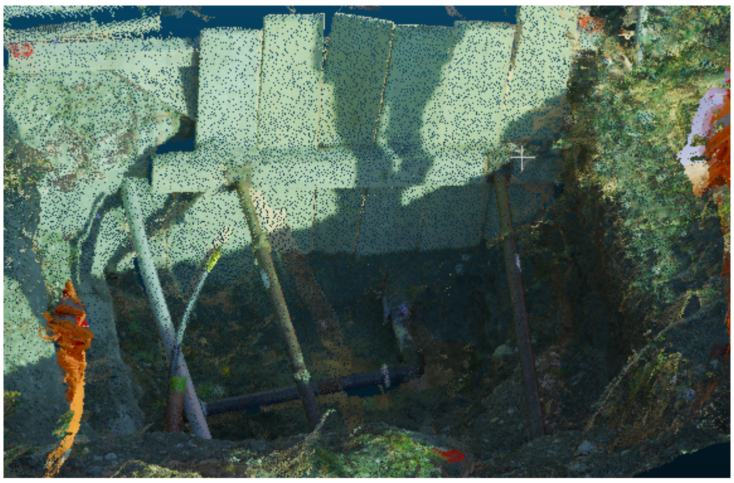


Figure 1
The point cloud of jussy_3 folder. color view

A better alternative to 3D convolution that was introduced in 2017 was Pointnet++ Qi et al. 2017. This approach samples points from the point cloud, considers points inside a sphere around the sampled points and calculates local features of these points according to multilayer perceptrons. To overcome the fact that the input points to the network have no meaningful order the maximum function is involved in the calculations. This process is continued in a hierarchical fashion so that the sampled points are sampled again and etc. At the end after getting high level features of the scenes for a few sampled points the features are mapped back to the original points using more perceptron layers. So at the end each point has features that could contain information about not only themselves but a more general scope.

This method has been applied to the KITTI dataset Geiger et al. 2013 as part of the code by Shi, Wang and Li 2019 and shows good performance on this dataset. I have tried to adapt the same method as Shi, Wang and Li 2019 only using the point cloud data to segment the scenes. Only the first network (region proposal network (RPN)) has been used. The three main steps of the project were **cleaning the data, sampling the point cloud and modifying the network**.

2 The Data

The data was provided in 8 folders one for each scene. Each folder contained one folder for the point cloud files called clouds and one or more folders that contained images of the same scene using different devices.

The clouds folder contains files in the ".las" format and the file "*_rtc360_*" is the point cloud that was collected using the lidar sensor and the only point cloud that has been labeled. The other files are the point clouds that have been created using the photos taken from the scenes and photogrammetry. This project concentrated on using only the lidar data. All the point clouds are in the same coordinate system.

In the other folders for each scene we have the photos folder and calibration and orientation files for the photos taken (*_camera_calib.xml and *_opk.txt). These were used during sampling of the lidar point cloud.

It is important to note that a point cloud data file is unlike an image a set of points(tuples) with no order. Each point in our lidar file has a coordinate x,y,z a classification and rgb values. More information about las files and the library used to work with them can be found here: *Laspy* 2020. The points that didn't belong to any specific pipe where labeled as 0 and others were labeled according to their classes(refer to the read me file in the data folder). In figure 2 we can see an example of labeled data.

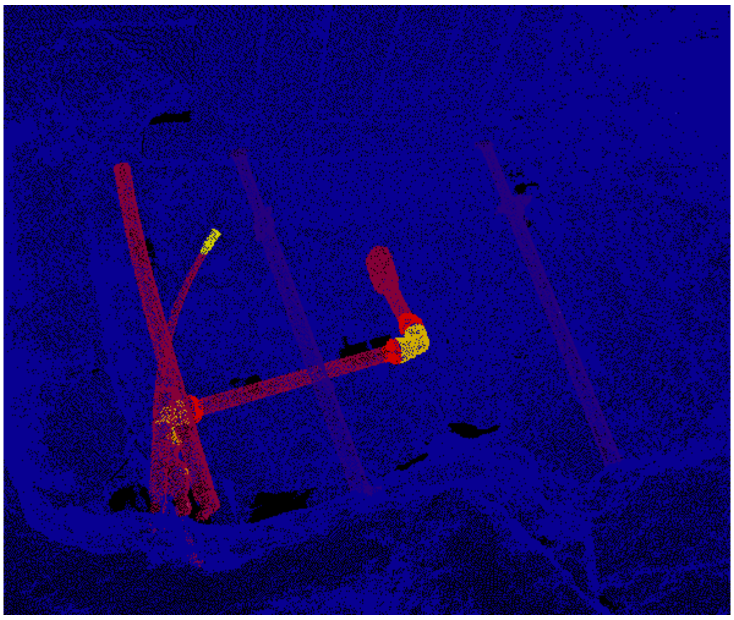


FIGURE 2
jussy_3 classification view

3 Data Cleaning and Labeling

The data provided was not ready to be used for instance segmentation, only the class of each point was determined. As usual for instance segmentation tasks we need each instance to be in its own separate bounding box. This was achieved by normalizing the scene, separating points according to their class, performing clustering using the DBSCAN algorithm (implemented in scikit-learn) and finally applying PCA to find the bounding boxes.

The DBSCAN algorithm is a density based clustering algorithm and has two parameters epsi and min_samples that control how the algorithm clusters the points. By experimenting and visualizing the results proper values were chosen for the parameters and each class was separated into its instances.

To find the bounding boxes(from now on called bboxes) I used PCA to find the direction of most variance for each dimension and then took the min and max of the point's in each dimension to get a tight bbox. I also checked that all the points are inside the bbox and with a good error (due to floating point precision) the sides of the box are orthogonal to each other. Finally each box was saved as its 8 corners. the final result for part of a scene can be seen in figure 3.

There were some challenges during this phase:

1. some pipes were too close to each other and couldn't be properly separated into two instances.
2. some pipes overlapped each other and this makes their bboxes overlap which breaks down the fundamental assumption that each instance can be separated by a bbox.

4 Sampling the Lidar Point Cloud

Since there was not enough training data we decided to sample viewpoints of the lidar point clouds we had. The point cloud was provided as a 360 degree view of the scene by sampling a viewpoint we chose a

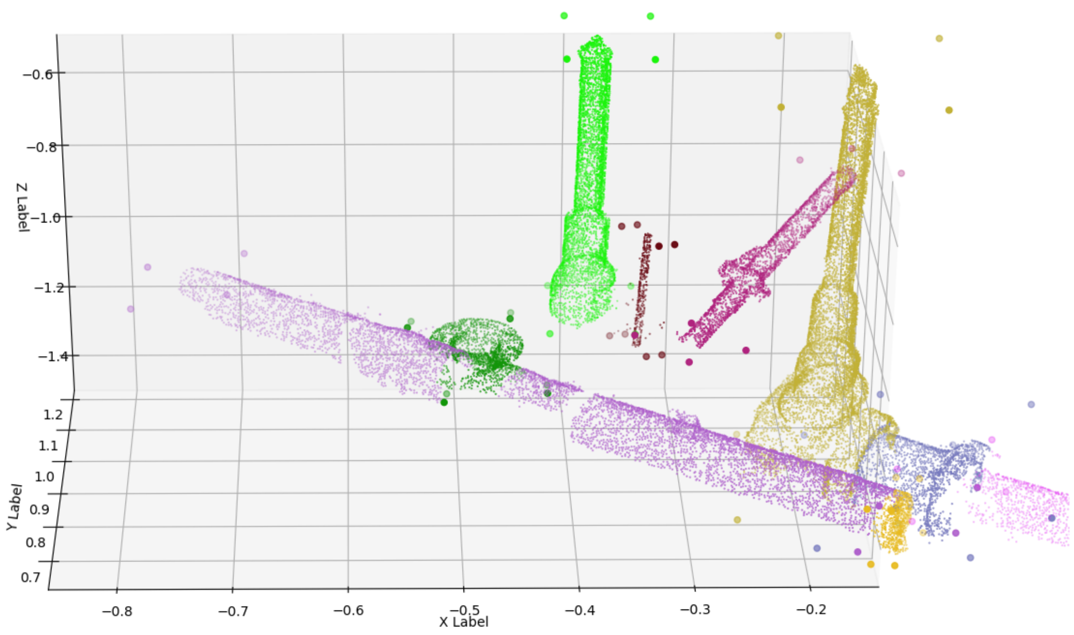


FIGURE 3
Part of the Champel point cloud. Each object and its corresponding bbox corners have the same color. We can see an example of two bboxes overlapping with the purple and the green pipes in the center.

location and orientation for an imaginary lidar scanner (or simply a camera) and only selected points that would be visible from this viewpoint. This also had the effect of cutting some of the pipes that overlapped with other pipes that were closer, and created silhouettes of the pipes closer to the camera. To create the effect of points overlapping each other the scene was discretized into pixels and in each pixel the closest point was chosen. (see figure 4)

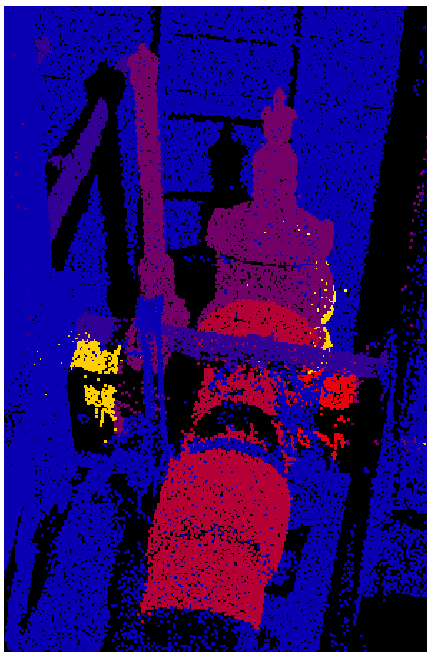


FIGURE 4
Result of sampling a viewpoint of Plan_ouates. The silhouettes of the two purple pipes can be clearly seen. Due to discretization the density of he point cloud has also decreased.

To choose a reasonable location for the imaginary camera and its orientation, I used the images of the scene and their corresponding position and orientation. This generated about 1000 scenes. Furthermore I used averages of positions and orientation angles of pairs of images (provided in the opk file of each image folder) to produce even more scenes. This raised the number of scenes to more than 6000.

5 Modifying the Original Code

The first step was creating new classes to read our data. When reading the data a random rotation and scaling is applied to the scenes. For the rotation a random value is chosen uniformly in range $[-\pi/2, \pi/2)$ for each of the 3 angles we have (rotation along x,y,z axes) and all the points and the corners of bboxes are rotated accordingly. In scaling I multiply all the points with a uniform random value in $[0.5, 1.5)$. With probability 0.5 I also flip the x axis of the scenes by multiplying x values by -1.

Since bboxes were represented differently in Shi, Wang and Li 2019 it was necessary to transform them from the 8 corners representation to a representation using the centre of the box the height,width,length and the angles of rotation along the three axes. The center is simply the average of the coordinates of the corners. The dimensions can be calculated using the distances of the adjacent corners. Finally Euler angles were used to represent the orientation of the bbox. The box was centered at the origin parallel to the axis then comparing it to the original box a rotation matrix was found using singular value decomposition and then the rotation matrix was used to find the Euler angels. see : *Finding Rotation Matrix* 2020 and *Euler Angles* 2020

In this project I only used the first network in Shi, Wang and Li 2019 to predict points that are part of pipes (pipe points vs non-pipe points) and find bboxes for the pipes.

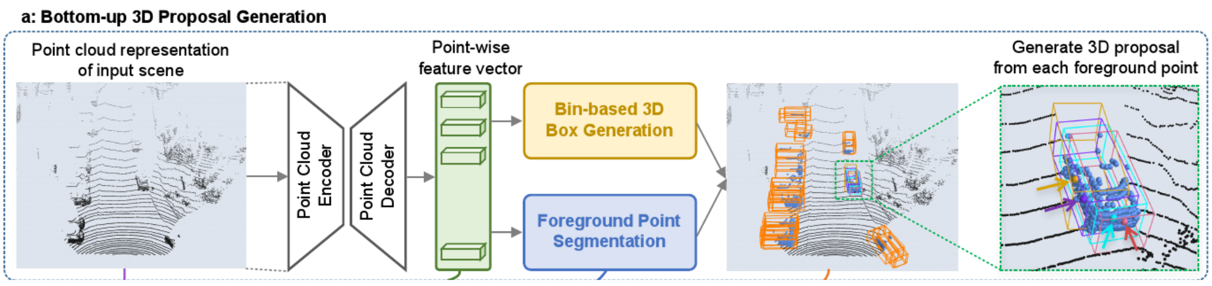

a: Bottom-up 3D Proposal Generation
Point cloud representation of input scene
Point Cloud Encoder
Point Cloud Decoder
Point-wise feature vector
Bin-based 3D Box Generation
Foreground Point Segmentation
Generate 3D proposal from each foreground point

FIGURE 5
The region proposal network (RPN) from Shi, Wang and Li 2019

I used a simple smooth L1 norm to regress the properties of the bboxes instead of the scheme in Shi, Wang and Li 2019. I also used a similar method as non-maximum suppression during evaluation based on the distance between the center of bboxes to select promising bboxes. But it appeared that the network even had problem in classifying the points. So I disabled the box generation head and concentrated on the classification head.

Focal loss was used as the loss for the classification head (same as Shi, Wang and Li 2019) due to the fact that background points are much more common.

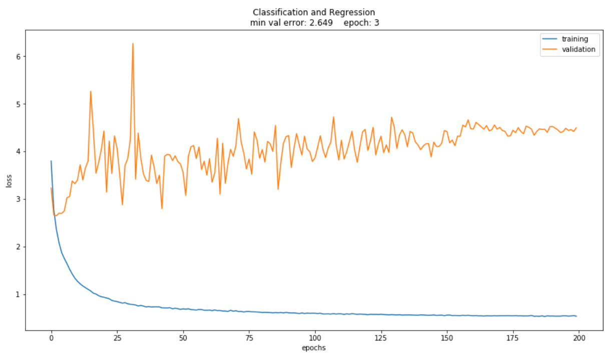
> The number of foreground points is generally much smaller than that of the background points for a large-scale outdoor scene. Thus we use the focal loss to handle the class imbalance problem. Shi, Wang and Li 2019

Initially I only used the lidar viewpoints that corresponded to the actual photos taken (around 1000 scenes) without any data augmentation. This was not enough data and there was over-fitting when looking at the loss of train vs validation (see the results section). This made me sample more viewpoints so I also included the average viewpoints. Now the total number of scenes was more than 6000 and I also added data augmentation. This somewhat reduced over-fitting. I also tried reducing the number of neurons in the deep layers of pointnet++ network (encoder-decoder in figure 5) by dividing the number of neurons by 2, and this also reduced the over-fitting further.

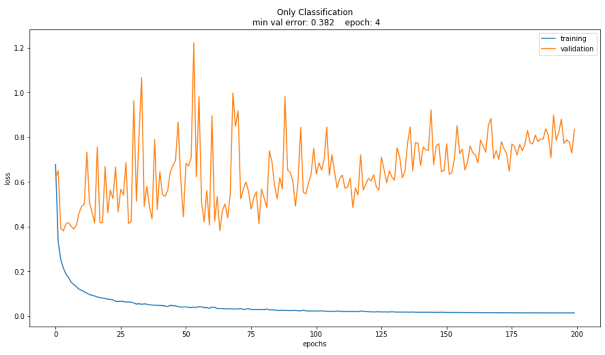
6 Results

The scenes used for training the network were: jussy1/2/3, athenaz and tram. The scenes used for validation were: plan_ouates and champel.

Below, the left plot 6a is the train vs validation loss and corresponds to training both classification and box generation heads. It can be seen that the network overfits the data after a few epochs. In the right plot 6b (and all following) the box generation head was disabled.



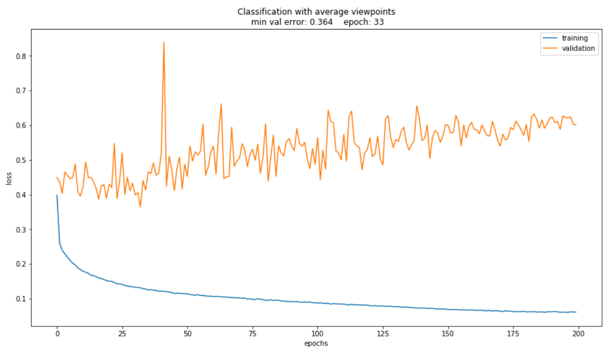
(A) Training with both heads of the network.
min validation error:2.649



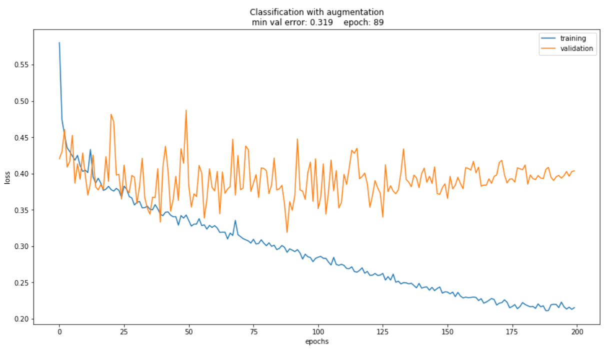
(B) Only performing classification.
min validation error:0.382

FIGURE 6

Below, the left plot 7a shows the result of adding average viewpoints without augmentation. Average viewpoints were used in all the following cases. In the right plot 7b data augmentation was added.



(A) Training with average viewpoints.
min validation error:0.364



(B) Performing data augmentation as well.
min validation error:0.319

FIGURE 7

Below, the left plot 8a pointnet++ was simplified by dividing the number of neurons in the deep layers by 2.

The scales of the coordinates in my dataset and the dataset used by Shi, Wang and Li 2019 were drastically different since they are different datasets and I had normalized each of my scenes. This inspired us to scale the radius of spheres used in pointnet++. To do this I considered all the points in all the train set scenes in our dataset and in the KITTI dataset, calculated the center, calculated the distance of each point from the center and calculated the median. The median for the KITTI dataset was around 9 while my scenes had a median around 1, so I divided all the radii used in pointnet++ by 9 but it increased the minimum of validation loss as seen in the right plot 8b. (pointnet++ was still simplified by dividing by 2)

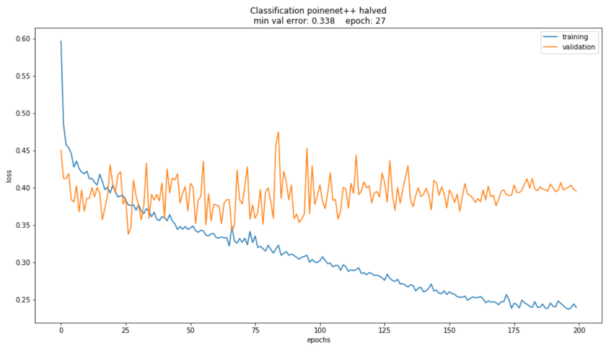
There were other attempts such as dividing the number of neurons in deep layers of pointnet++ by 4 9a and multiplying the number of input points to the network by 3 9b that did not reduce the minimum validation loss below 0.319 7b and can be seen below.

In evaluation the measure used to determine the performance of the classification head was:

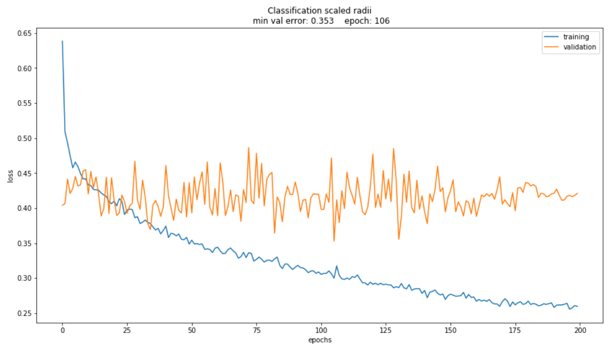
$$\frac{TruePositive}{TruePositive + FalsePositive + FalseNegative}$$

which excludes the True Negative since it is too common as mentioned before for focal loss. The performance of the validation set is the average of all the scenes.

During the evaluation it was also important to determine a proper threshold for the sigmoid function of the classification head, so that every value above the threshold would be classified as a positive case (the

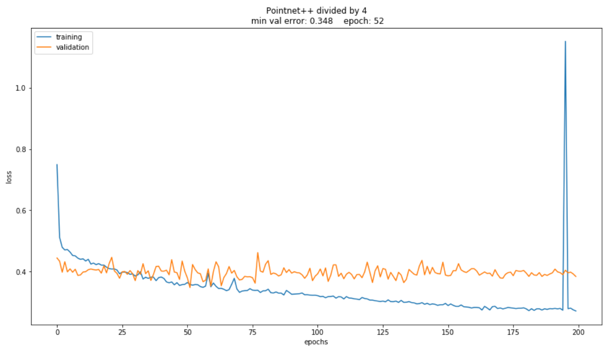


(A) Pointnet deep layers divided by half.
min validation error:0.338

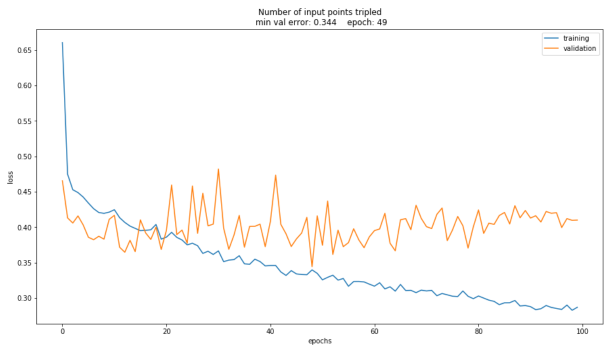


(B) Radii scaled.
min validation error:0.353

FIGURE 8



(A) Pointnet deep layers divided by 4.
min validation error:0.348

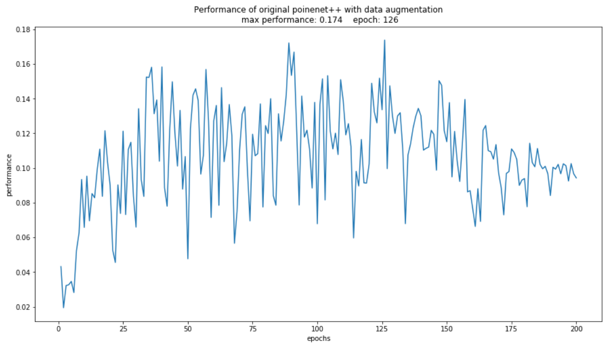


(B) Number of points tripled,only trained for 100 epochs.
min validation error:0.344

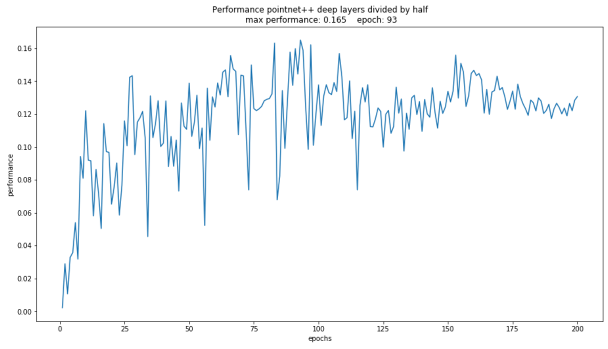
FIGURE 9

point predicted as belonging to a pipe). 10 values were tried from 0.1 to 0.3 and the best value for the epoch with minimum validation error seemed to be close to 0.2 so this value was used.

The performance in each epoch for halving the pointnet++ and without halving it can be seen below. In both cases I used data augmentation and average viewpoints.



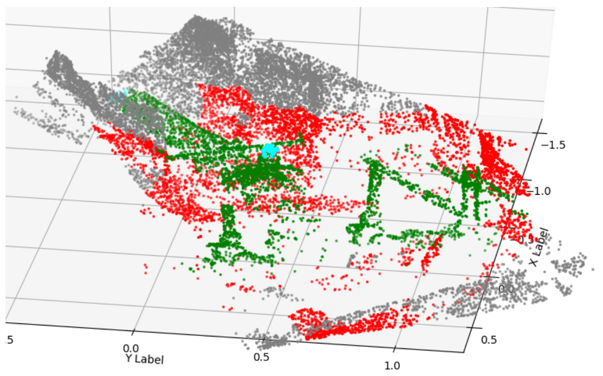
(A) pointnet++ deep layers intact
max performance: 0.174



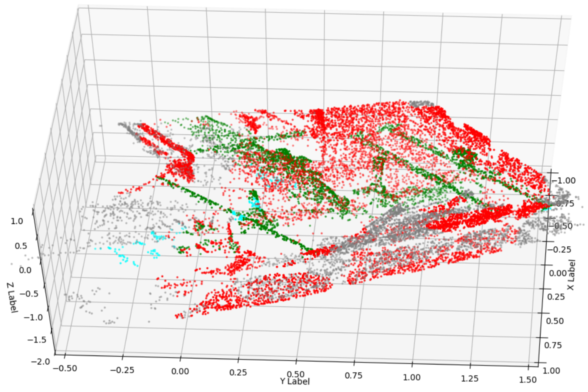
(B) pointnet++ deep layer halved
max performance: 0.165

FIGURE 10

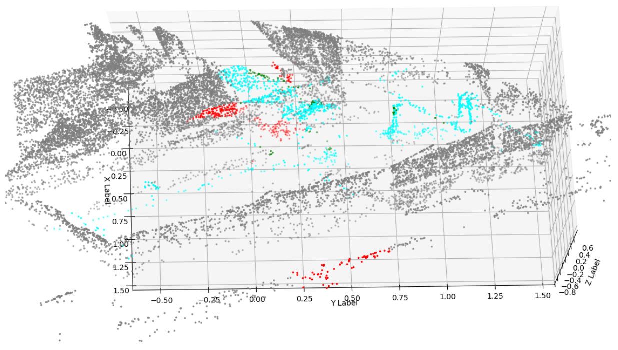
The best result seems to belong to the case where we didn't change the pointnet++(7b) although the difference in performance compared to halving the pointnet doesn't seem significant. So this model was used to visualize the results of classification below. The gray points are the true negative points, cyan are the false negative, green are the true positive and red are the false positive. In the first figure 11a we see the result for a scene with good performance. In the second figure 11b we see a problematic case where the classifier classifies almost every point as a pipe point so the false positive rate is very high and in the third scene 11c we have a high false negative rate .



(A) a relatively good scene.
scene performance: 0.415



(B) a scene with lots of false positives(red).
scene performance: 0.289



(C) a scene with lots of false negative(cyan).
scene performance: 0.066

FIGURE 11
All the scenes are from the Plan_ouates point cloud from relatively the same angle. But sampling has changed the point cloud in different scenes.

7 CONCLUSION

In this project I tried to apply the same method as Shi, Wang and Li 2019 to the pipe dataset available. First the data was cleaned and labeled. Then to augment the data we sampled viewpoints and finally I adjusted the different hyper-parameters of our network to improve the results but no major improvements were possible due to a lack of data.

Clearly it would have been better to test our models on a separate test set but since we didn't have enough scenes this was not possible. Even though we are considering the best result of the validation set, our performance (0.17) compared to the KITTI dataset(0.5) is much lower and until it is improved with more data there isn't much hope of getting quality results from the box generation head.

References

Qi, Charles Ruizhongtai et al. (2017). ‘PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space’. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 5099–5108. URL: `http://papers.nips.cc/paper/7095-pointnet-deep-hierarchical-feature-learning-on-point-sets-in-a-metric-space.pdf`.

Geiger, A et al. (Sept. 2013). ‘Vision meets robotics: The KITTI dataset’. In: *The International Journal of Robotics Research* 32.11, pp. 1231–1237. ISSN: 0278-3649, 1741-3176. DOI: `10.1177/0278364913491297`. URL: `http://journals.sagepub.com/doi/10.1177/0278364913491297` (visited on 27th Mar. 2020).

Shi, Shaoshuai, Xiaogang Wang and Hongsheng Li (16th May 2019). ‘PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud’. In: *arXiv:1812.04244 [cs]*. arXiv: `1812.04244`. URL: `http://arxiv.org/abs/1812.04244` (visited on 27th Feb. 2020).

Laspy (2020). URL: `https://laspy.readthedocs.io/en/latest/` (visited on 2nd June 2020).

Finding Rotation Matrix (2020). URL: `http://nghiaho.com/?page_id=671` (visited on 3rd June 2020).

Euler Angles (2020). URL: `https://www.learnopencv.com/rotation-matrix-to-euler-angles/` (visited on 3rd June 2020).