# Pedestrian Detection using Sliding Window approach on 3D scenes (Version 1- a 2 Stage Network, Version 2- a 3 Stage Network)

Vamsidhar Kamanuru
UCSD
San Diego
vkamanur@eng.ucsd.edu

Sai Gullapally
UCSD
San Diego
scgullap@eng.ucsd.edu

## Abstract

*The problem of understanding 3D data has become one of paramount importance especially in field of Autonomous Driving, where the objects needs to be localized accurately. While the existing methods perform well for large objects like cars they do not work so as well with smaller classes like pedestrians. In this work we focus on improving the accuracy of localization of the pedestrian class. We implement a new sliding window approach analogous to the 2D case but in this case as we show later, we need to slide only within a 3D frustum thus making the process feasible. Our network then operates on these windows instead of on the entire frustum, this reduction in 3D search space improves the results. We show that this enables us to reduce a 3 stage network like Frustum PointNet-v2 into a much simpler 2 stage network without losing out on the accuracy. We then add an additional regression network to further boost the accuracy and improve the results of Frustum PointNet-v2.*

## 1. Introduction

The problem of understanding 3D scenes has become quite popular in the recent times [5], [6], [3], owing to the diverse applications it possess like autonomous driving, augmented reality, medical diagnostics etc. In this work we are specifically interested in the field of autonomous navigation. We work on the KITTI 3D dataset [**?**] where we have classes like 'car','cyclist' and 'pedestrian'. In the case of autonomous navigation it is desirable to detect and localize the objects in the environment as accurately as possible in the 3D coordinates. This is done by operating networks on the 3D LiDAR data. There are two popular methods which do this one involves converting the 3D points into voxels and giving local shape descriptors to each of the voxel and then using 3D CNN's on this, another approach invovle using pointnets which work directly on the 3D data and are much less computationally expensive than voxels while giving good results. We follow in the line of pointnets for our current work.

Pointnets take point clouds as inputs and can output the class of the 3D structure or can regress on the attributes of the structure present in PCD. Pointnets perform well on small scenes, as scenes in KITTI data are usually very large frustum pointnets have been proposed [5] which are very efficient and elegant. They reduce the 3D search space by making use of the the camera matrix information and mature 2D detectors available, once a 2D bounding box is regressed around the object in 2D image then the camera matrix information is used to extrude a frustum in the 3D search space, this would imply that the object is within the frustum and we can confine the search to that location rather than the entire search space. As explained clearly in the coming sections we go one step further (employing a method for the 3D scene that similar to sliding windows method in 2D case) and reduce the search space once again within the frustum to a much precise region. We do this because existing works [5], [10], perform well on larger objects like the category: 'cars' but not as well as desired for smaller objects like the category:'pedestrians', but clearly pedestrians is a very important category and we believe it would be of significant use to improvise results for that category. In this work we focus only on the problem of Pedestrian localization but as shall be explained in detail in the following sections this can be used for all the categories.

One of the reasons for low accuracy could be the sparsity of 3D points in the LiDAR data for the category:'pedestrians'(there will be fewer points for pedestrians as compared to cars), this might also give rise to a situation where background might have more points than the pedestrian(or maybe a electric lamp pole might have a similar number of points etc). Also it is intuitive to think that the smaller the size of the scene and also the lesser the background clutter (i.e most of the scene is comprised of the object) involved in the scene the better would be the regression results and hence the accuracies. Even though the frustum

pointnets work on reduced search space it can be seen in figure that there can still be lot of background clutter in the frustum in addition to the object and as explained above this could be problematic especially when the object of interest has very few points by itself, So with this in mind we make the following modifications to the basic frustum pointnet network,the following are the contributions of our work to the existing work:

### 1.1. Contributions

- Drawing inspiration from 2D detectors, we propose a novel sliding window approach that is feasible and simple to implement for the 3D case. This will generate one single proposal window and the window is such that most of it is occupied by the object. We then make use a constant heading angle ($pi/6$), the mean sizes of the pedestrian class and the centroid of the points within the window for the bounding box heading angle, sizes ad center respectively to directly get an output(similar to anchor boxes of fixed height) after the second stage this is the version-1 this has slightly better accuracy than frustum PointNet. (OR)

- To further improve thus we add another regression network and send the single proposed window as the input, as the 3D scene is reduced by a lot this should make the task of the regression network easy. We shoe that this further increases the results. This is the version-2 network.

- Region proposal has been employed in 3D but those methods involve using voxels, this method operates directly on 3D point cloud.

- To improve regional proposal accuracy we use Focal loss [4] another popular technique in 2D which we adapted to our 3D case.

### 1.2. Dataset

- We use the KITTI [1] 3D data.

- We pre-process it as suggested by [5] using their methods, i.e we process the data till the stage where frustums are extracted and augmented with various changes like translation etc.

- Train set has 10,635 pedestrian examples generated after augmenting the examples from KITTI train data.

- We split the data set generated after augmenting the KITTI val data(2242 examples) into a small validation set(526 examples) and a testing set(1716 examples) . This is what we shall be using for validation and testing purposes

## 2. Related Work

With the introduction of various deep learning architectures like Faster-RCNN, object detection has become quite mature i.e accurate and in the case of 2D images in terms of both accuracy and processing time. In the case of 3D too there have been made significant leaps in this direction [10]. But it is still not as mature as the 2D case and we believe there are still improvements that need to be and can be made. We will begin with the literature starting from 2D scenario.

**2D:** In the case of 2D we have various architectures like OverFeat [9], YOLO [7], etc which are one stage detectors they are fast but they suffer in terms of accuracy. On the other hand we have Fast R-CNN [2] and Faster R-CNN [8], these are two stage detectors which improve accuracy with the help of a region proposal stage followed by a detector head, although they are accurate they are not fast. RetinaNet [4] which is a single stage network was proposed to address these issues, it follows a sliding window approach i.e classifying a small window, but as there can be a huge number of sliding windows(in the order of 100000) with very few of them having the object of interest in them this causes a huge class imbalance and hence they use a new loss which is the focal loss to address the above issues, thus this is a one stage detector which also has good accuracy and speed. We also make use of the focal loss but in our case we will be sliding across frustums, and also the class imbalance will be around 1:100 for positive versus negative windows. But still this was essential for us in 3D case.

**3D:** In the case of 3D before the current state of the art techniques [6] [3] were proposed the 3D data was converted into an RGB image (by projecting from 3D) and then 2D CNN was used to process it, this results in a loss of the rich 3D structure information available to us especially in cases where there is occlusion. In 3D we will not be losing any information of the object due to occlusion so this method is not ideal. Recently a common method used to group the pointcloud data into voxels and each voxel could be given a local shape descriptor or simply also be converted to binary values based on presense or absence of points in that voxel. Once the voxels and the descriptors were created 3D CNN's were used get the job done, the major disadvantage is that this approach requires huge computational resources especially if the scene is large. Another very popular method so far is using PointNets [6] which operate directly on the 3D point clouds, they take care of the permutation invariance of data points in 3D.They can be used for classification as well as regressing the bounding box values. The pointnet also performs well when the scene is small hence for data like KITTI it cannot be directly applied for getting good results, as operating on the entire point clouds might be difficult in case of large scenes (which is the case in autonomous driv-

ing) an improved network called Frustum PointNet which is a 3 stage network proposes a novel dimensionality reduction technique, they make use of the mature 2D detection techniques to predict the bounding box in 2D and then extrude it out into a frustum in the 3D search space. As the object must lie within the frustum only the points inside this frustum are then taken as inputs and passed to the pointnet thus effetively reducing the scene size by a lot. After the frustum extrusion we have the second stage where the PointNet performs a segmentation of the data in 3D, it is much easier in 3D as there is no occlusion, then the segmentation result is sent to the third stage to a TNet and regression net for prediction of the bounding boxes.

Our work closely resembles and is based on the frustum pointnet, however we make two crucial changes to the existing methodology, drawing inspiration from 2D detectors, we propose a novel sliding window approach that is feasible and simple to implement for the 3D case. This will generate one single proposal window and the window is such that most of it is occupied by the object, this much smaller window proposal, we show that this proposal is good enough to make a bounding box prediction at this stage, in order to boost the accuracy even further instead of predicting the output at this step the single proposed window is sent as input to the regression network in stage 3, thus it should make the task of the regression network easy. Also as we are classifying windows as positive or negative and with a huge imbalance with negative windows dominating(ratio around 1:100), we make use of the focal loss to address the issue of class imbalance and too many easy examples negatives. Thus our work is draws from [4] but also as mentioned above we bring in our contributions.

## 3. Methodology

In this section we give the details of the implementation for our model, similar to the frustum pointnet model we propose a 3 stage model, our model differs from the pointnet model in stage2. The detailed description of each of the stages is given below(As explained above for the present we focus only on pedestrians in the future we plan to expand it to other classes, this would be straightforward with just a few minor adjustments like setting correct hyper parameters for the sliding window depth etc):

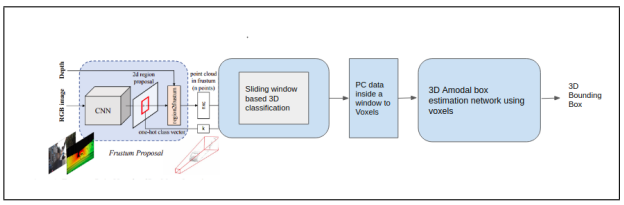


Figure 1. Proposed Model Outline

### 3.1. Stage 1 : Frustum Proposal

In 3D scenes while using Pointnets it is of huge advantage both in terms of computational power and accuracy if the scene is small for the pointnet to act upon, so if we can somehow reduce the 3D search space into a smaller region most of which is occupied by the object it would be useful, this can be achieved using information of the camera projection matrix and using a 2D bounding box predicted in the 2D-image of the given scene by extruding the 2D bounding box as per the camera projection matrix and generating a frustum as shown in figure2. So in the first stage of the network we make use of the RGB data any good state of the art 2D detector and then for each object detected in the RGB image we extrude a frustum for the next stage of the network. For this stage i.e frustum generation we use the code and data provided [5], the outputs of this stage may be considered like preprocessed inputs for stage 2.

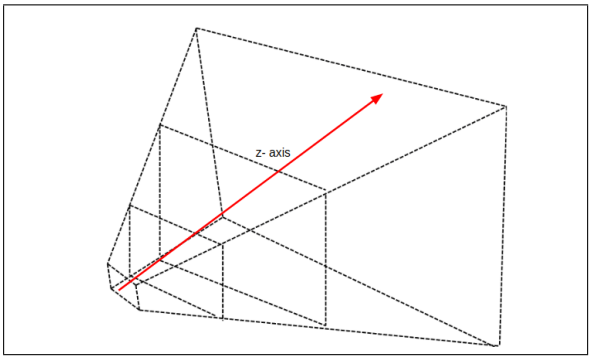


Figure 2. Sliding windows within a frustum, (image source-https://ahbejarano.gitbook.io/lwjglgamedev/cascaded-shadow-maps)

### 3.2. Stage 2 : Region Proposal

In the case of detection and localization in 2D one popular approach is that of using sliding windows over the image and then classifying if that object has the object of interest or not, in case an object is detected in the sliding window the sliding window becomes the bounding box. In order to allow various scales previously an FPN approach was used either on filter sizes or on the image itself as this takes time and computation recently anchor boxes have been proposed.

We consider a similar approach in 3D, the number of possible sliding windows is huge in 2D itself so if we apply the same approach directly the number of sliding windows that we get blow up in 3D scenario and the approach becomes infeasible for real-time approach, instead what we do is make use of the frustums we have from stage 1. The frustum by the way in which it is generated has a unique property, if we assume that the 2D bounding box is tight(which is quite reasonable to do so as 2D detection is now quite

mature) then when we extrude the frustum the assuming the axis as shown in 2 , although the along depth or z axis of the frustum the object can be anywhere the boundary of the frustum will also be reasonable tight along the x,y axis (as the 2D bounding box is tight), hence we do not need to consider sliding along those directions instead we just need to slide along the z-direction i.e along the depth as shown in 2 , also note that for out sliding window size need to assign only the dimension along the z-axis as other dimensions are decided by the boundary of the frustum, the next question is how many such scales (or maybe anchor boxes) do we use?

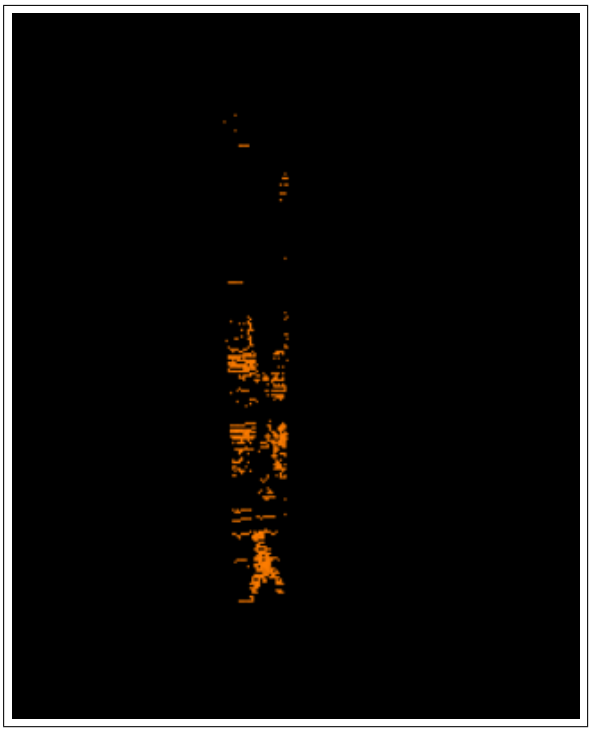

Figure 3. Point cloud within the frustum (entire frustum data) given as input to second stage

#### 3.2.1 Size of the sliding windows, Stride:

We decided to choose the sizes based on the the data statistics, we know the coordinates of the object of interest(pedestrians in our case) from the labeled data in the frustum coordinate reference frame and hence we know the size occupied by the object along z-axis a $z_{max} - z_{min}$. We make a plot of these for all objects of the category 'pedestrians' as can be seen in 5, we see that it is closely resembles a normal distribution centered at 1 meter and standard deviation of 0.23, based on this we initially considered 3 scales at 1 meter,1.4 meter ,0.6 meter. Then observing the results we decided just one scale of size 1 meter was sufficient as addition of the other two scales to this did not bring any

Figure 4. Chosen window(selected proposal from the frustum) which is the output of second stage.

significant improvements in the results for the classification stage also it would increase the complexity. This would also give us an additional advantage in amodal bounding box prediction, as a smaller portion of a few of the larger objects may get cut while making sliding windows and we still force the regression network to predict the correct this would make the network more robust to occlusions and give better amodal bounding box estimates, hence we choose to go with a scale of 1.0 meter along the z-direction. Also experimentally we found that a stride of 0.1 meter was optimal.

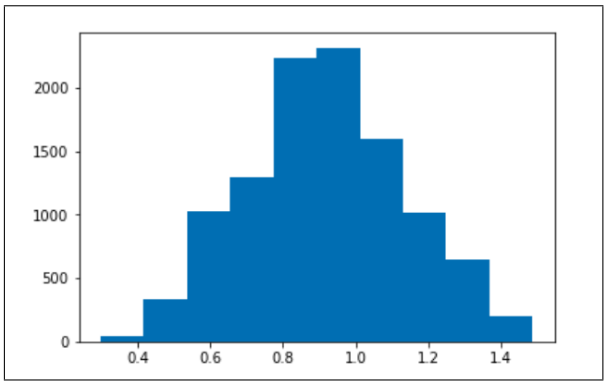


Figure 5. Plot of Histogram of the object sizes along z-direction

#### 3.2.2 Assigning Labels:

So once we are given a frustum and we get all the sliding windows from it we then need to label these windows appropriately, we do this as follows: For each sliding window in the frustum we know the points within the window

which belong to the object and also the total points within the frustum which belong to the object, we then compute a point-wise IOU score as opposed to a area/volume wise IOU score as we found this works much better for our case. Once the point-wise IOU score is determined, if the window has a score greater than 0.7 we assign a positive label to this and if a window has a score less than 0.3 we assign a negative label to it. So we send each image as a mini-batch to the our network which then obtains all the sliding windows and operates on them in parallel, clearly there will be huge imbalances in positive and negative classes (around 1:100 ratio) hence we use focal loss for the classification loss. We can now use the single proposed window to make a prediction for the bounding box by using the centroid of the points within the window, the mean sizes of the class 'pedestrian' and $\pi/6$ as the heading angle as the center, sizes and the heading angle for the bounding box. This is what we propose as the Pedestrian Detector version-1 $\pi/6$ was chosen as it turned out to be marginally better than other angles during our hyper parameter search. We can also proceed to send this proposed window to stage 3 for regression to further improve the accuracy but making the network much more complex and also taking extra time. The three stage network is what we propose as the Pedestrian Detector version-2.

#### 3.2.3 Non-Maximal Suppression

: From the output of this stage we choose the window with highest probability of being a positive label and then pass it on to the next stage i.e we make a single proposal for the next stage. Note that although we focus on improving both precision and recall for each of the inputs but strictly speaking we do not need to have high precision and recall what we exactly need is that the single proposal we predict should be a positive window we call this the single metric accuracy. As we are imposing more stringent training conditions than required we find in the results that the single metric accuracy turn out to be quite good. Next we move on to the regression network.

#### 3.2.4 Data:

For this we use the training set of 10,635 examples, validation set of 526 examples and a test set of 1716 examples(We do not use the test set to determine any hyper parameters we just use it to check results at each stage).

### 3.3. Stage 3 : Amodal 3D Bounding Box regression

The output window is then passed onto the regression network which is a similar network as used by frustum pointnet [5], we adapt their network to our case by making necessary changes. This network predicts the 3D bounding box for the object. To make the network invariant to translations we first represent the points in the window in terms of the coordinate axis of the centroid of the points(which ideally should be the objects centroid). But still even after this it is possible that the center of the bounding box may be quite a bit away from the centroid hence a TNet is used in the to predict the 'stage1 center'. The points in the window are then represented w.r.t the stage1 center as the origin. Then we pass this to a amodal 3D bounding box estimation network. This network regresses the residual center, residual sizes, heading angle class, residual heading angle.The exact details can be found in [5] $\theta$ is equivalent to a heading angle of $180 + \theta$, instead of having 12 different classes each of width $\pi/6$ we have only 6 classes, we did this so as to remove the unnecessary confusion for the network to distinguish an input with heading angle of $\theta$ from another input with a heading angle of $180 + \theta$.

#### 3.3.1 Data:

For this we use in addition the training set of10,6335 examples, we also make set of ground truths (for the training data used in stage1) of size 15,345 samples which we use in training the second network to ensure that any missclassifications in stage 1 do not overly affect the training of stage 2, we again split the augmented dataset from KITTI of 2242 examples into a validation of 1100 examples and a test set of 1142 examples.(again we do not use the test set to determine any hyper parameters we just use it to check results at each stage).

## 4. Experimental Evaluation

We provide and discuss the results for various experiments using our network in this section.

### 4.1. Region Proposal results

We provide the results for this stage in terms of the plots for various metrics like precision etc. Figure 7 shows the loss value for the training, validation and test sets. As we can see the loss keeps decreasing for 5 epochs, we stopped training as soon as validation loss started to increase. It can also be seen that almost all the three datasets have similar trends this shows the data is good. 9 shows how the precision changes with epochs, the precision starts at around $40\%$ for the validation and test cases and reaches up to $63\%$ for the test and $65\%$ for the validation while it is around $79\%$ for the training set. 10 shows how the recall changes with epochs, the recall starts at around $97\%$ for the validation and test cases and goes down to $95\%$ for the test and $96\%$ for the validation while it is around $97\%$ for the training set. This trend is due to the effect of focal loss which is trying to improve precision. Also we need not worry about the actual precision and recall as explained in

3.2.3 we just need to worry about the single metric accuracy which is shown in 8, as we can see the accuracy is quite good and is around $86.5\%$ for test data and $90\%$ for validation and $91.5\%$ for the training data which is really good for our case. We also show visually how this stage works, 3 shows the point cloud data within a frustum i.e this is one of the inputs to the region proposal stage of the network, the output after passing through this stage is shown in 4, as it can be seen the network does a really good job of selecting the appropriate portion of the frustum i.e the correct sliding window.

### 4.2. IOU results after stage 2

As explained above we get the outputs for the Pedestrian Detector version-1 at this stage and they can be seen in table 3 we see that we have a 0.5 IOU accuracy of $63.7\%$ which is quite good, although it is not as good as the outputs of version-2 it is just a 2 stage network so this trade off is expected. Figure 6 shows the histogram of the IOU scores. next We compare it with the results of Frustum PointNet v-2, we however point out that the comparison is not entirely accurate as the Frrustum-PointNet paper reports the IOU scores for the pedestrians under three categories: hard-$53.59\%$, moderate-$61.32\%$ and $easy - 70\%$ . We could not do that due to time unavailability hence we reported the score of Frustum-PointNet as the average of these three category accuracies i.e $61.6\%$. We also note that out results for the overall set are definitely better than moderate and hard example accuracies hence we believe our method is a strong competitor to Frustum-PointNet.

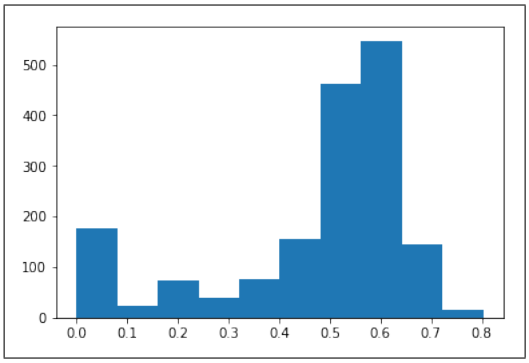


Figure 6. Plot of histogram of IOU scores after stage 2

### 4.3. IOU results after stage 3

We perform a regression very similar to Frustum PointNet. We have multiple losses alike stage1 center-loss for the stage1 center regression TNet, center loss for the final center regression, size loss for sizes, heading angle classification loss for correct classification of heading angle into bins, heading angle residual loss for the residual angle regression and then a corner loss, the final loss is a weighted

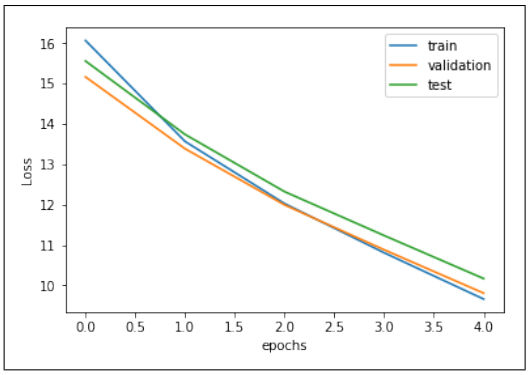


Figure 7. Plot of classification stage network's loss vs epochs

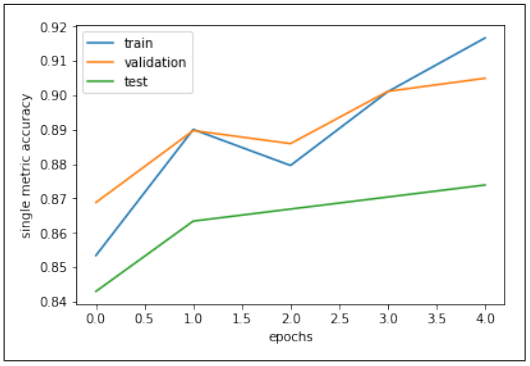


Figure 8. Plot of classification stage network's Single metric Accuracy vs epochs

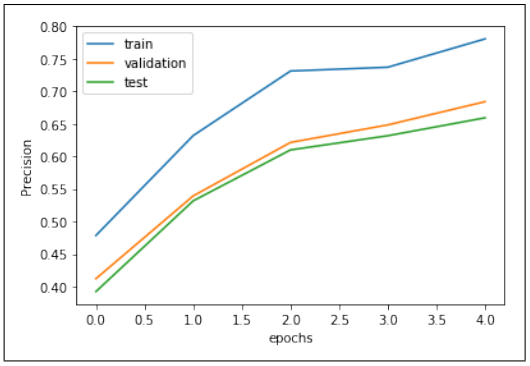


Figure 9. Plot of classification stage network's Precision vs epochs

sum of these losses we change each of these to show how it effects the loss and also to decide which loss is more important. We carefully tune these hyper-parameters on the validation set. We show in table **??** the best results we obtained on the validation set for various combinations, We see that size loss should be given the maximum importance and a weight of 4.0, then the two center losses should be given next preference and a weight of 3.0 is optimal for them, then comes corner loss with a weight of 2.0. We see

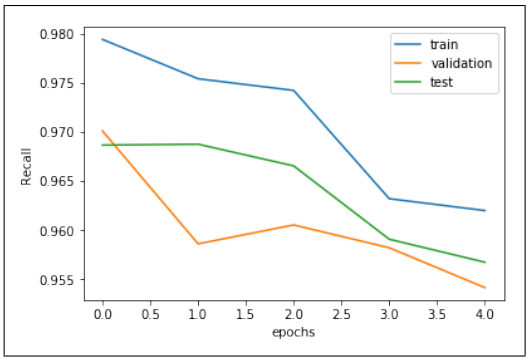


Figure 10. Plot of classification stage network's Recall vs epochs

| Model | 0.5 IOU |
|---|---|
| Stage1 center loss | 0.55 |
| Center loss | 0.557 |
| Size loss | 0.259 |
| heading classification loss | 1.566 |
| heading accuracy | 36.5% |
| heading residual loss | 0.31 |
| Corner loss | 0.689 |

Table 1. Individual loses on test set

| Model | 0.5 IOU |
|---|---|
| With ground truth heading angles | 66.7 |
| With predicted heading angles | 64.6 |

Table 2. Effect of heading angle on IOU accuracy

| No. | stage1 center loss weight | center loss weight | size loss weight | heading angle class loss weight | heading angle reg.loss weight | corner loss weight | 0.4 IOU acc. score | 0.45 IOU acc. score | 0.5 IOU acc. score |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.0 | 1.0 | 4.0 | 4.0 | 4.0 | 2.0 | 78.9 | 70.1 | 62.1 |
| 2 | 2.0 | 2.0 | 4.0 | 4.0 | 4.0 | 2.0 | 78.1 | 71.1 | 61.3 |
| 3 | 4.0 | 4.0 | 4.0 | 4.0 | 4.0 | 2.0 | 79.9 | 72.9 | 63.8 |
| 4 | 3.0 | 3.0 | 4.0 | 2.0 | 2.0 | 2.0 | 78.7 | 71.7 | 62.5 |
| 5 | 3.0 | 3.0 | 4.0 | 1.0 | 1.0 | 2.0 | 80.09 | 74.8 | 64.9 |
| 6 | 3.0 | 3.0 | 4.0 | 1.0 | 1.0 | 0.0 | 81 | 73.5 | 63.9 |
| 7 | 3.0 | 3.0 | 4.0 | 1.0 | 1.0 | 4.0 | 79.7 | 73.5 | 63.6 |
| 8 | 3.0 | 3.0 | 5.0 | 0.5 | 0.5 | 2.0 | 79.8 | 72.5 | 61.3 |
| 9 | 3.0 | 3.0 | 6.0 | 1.0 | 1.0 | 2.0 | 80.27 | 72.6 | 61.81 |

Figure 11. Results on validation set for stage 3 with different hyper-parameter choices

| Model | 0.4 IOU | 0.45 IOU | 0.5 IOU |
|---|---|---|---|
| Our Model (output taken at stage 2) | 77.6 | 73.3 | 63.7 |
| Our Model (output taken at stage 3) | 78.9 | 74.4 | 64.6 |
| Frustum PointNet-$v2^*$ | - | - | 61.6* |

Table 3. Comparison between our two models, Frustum PointNet. *-INDICATES THE FOLLOWING-Our comparison with the Frustum PointNet is not exactly accurate as they have reported the individual scores for 'hard', 'medium' and 'easy' categories, due to time constraints we could not do that hence we just compared with their average loss for all the three losses taken together.

that corner loss does make a difference (64.9% vs 63.9% for No.5, No.6). Also the heading angle losses are given a low importance here because we found that there was not much of a positive effect if we change these values hence we gave importance to other losses, we will discuss about the heading losses again now we focus on the overall loss, Figures 12, 13 show the loss for test and validation data for the regression stage when we set the hyper parameters to the best combination we found as per table **??** i.e combination 5, we can see that over fitting occurs around the 13th epoch(we stop the training there) and then we plot the IOU scores for both the train and validation data with the epoch number, these plots can be seen in Figures 14, 15, the losses are as expected (reducing for training set and showing overfitting on the validation set). the IOU scores also follow the expected trends for 0.4,0.45,0.5 thresholds, we have an IOU accuracy of around 64.9% for 0.5 threshold on the validation set. We then report the IOU's on test set as can be seen in table 3, we see that the final accuracy of 64.6% is achieved on test set. We also report the individual losses in table 1 . Also as mentioned before, changing the heading angle loss weights did not change the results much(this was strange as we usually see some trend), we feel this is the case because the model is unable to learn the heading angles for a pedestrian possibly because the pedestrian is symmetric and hence it is difficult to estimate a pose we found that we get only up to a 40% accuracy in heading bin classification (6 possible bins are there), so to see how the model would perform when it knows the correct heading angle we tried two cases one in which we supplied the true heading angle for 3D bounding box estimation and another where we use the predicted angles, this gave a surprising difference of almost 2.1% in accuracy as can be seen in 2. Thus we find that this is also one the things that could be what that causes the category 'pedestrians' to have less accuracy in general.

## 5. Conclusion

We thus proposed two networks for the purpose of pedestrian detection in 3D scenes, we showed that out networks are strong competitors to state of the art methods like frustum pointnets, we successfully adapted the 2D sliding window technique to the 3D case and got good results with just a 2 stage network whereas the the network which we

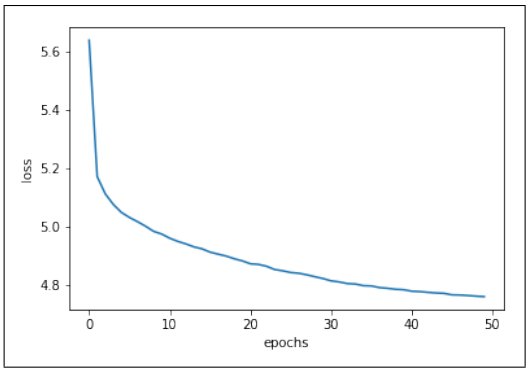


Figure 12. Plot of Regression stage network's loss vs epochs

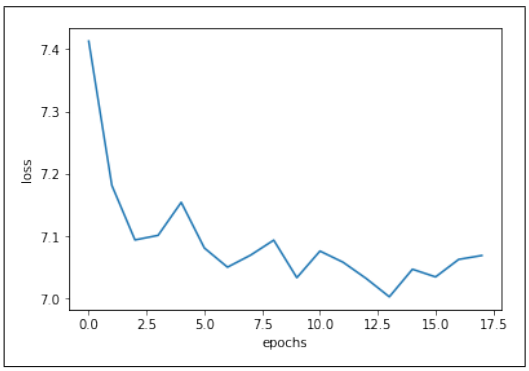


Figure 13. Plot of Regression stage network's loss vs epochs

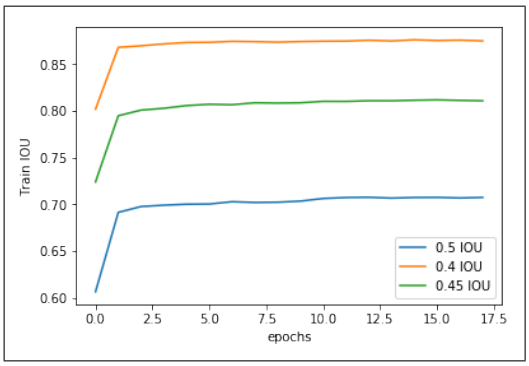


Figure 14. Plot of Final IOU accuracy on training set vs epochs

referred to i.e frustum pointnets v-2 are 3 stage networks, we further introduced a 3rd stage to improve the accuracy even more. Also the network we used much simpler than the state of the art frustum pointnet v-2. Next we analysed which losses are important for the regression and showed how we need to assign the weights, we aso showed the effect of heading angle in the regression outputs.

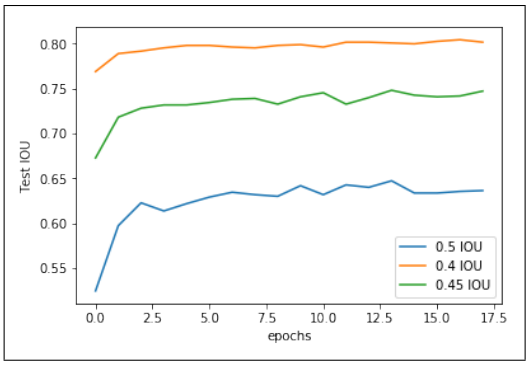


Figure 15. Plot of Final IOU accuracy on validation set vs epochs

## References

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

[2] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 2

[3] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. PCPNET: learning local shape properties from raw point clouds. *CoRR*, abs/1710.04954, 2017. 1, 2

[4] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017. 2, 3

[5] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017. 1, 2, 3, 5

[6] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016. 1, 2

[7] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 2

[8] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. 2

[9] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. 2

[10] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017. 1, 2