Canonical Transformation and Detransformation: Better Learn the Second Stage Bounding Box Refinement

Yifei Liu, Klaus-Rudolf Kladny

Abstract

In this small virtual paper we introduce a canonical transformation method to better learn the bounding box refinement in a two stage 3D object detection framework purely based on lidar points. The canonical transformation method was introduced and used in [3], and in this paper we are reusing this method and analyzing its results. And we propose an Embedding Canonical Transformation and Detransformation in the training procedure which can allow us to train the model without changing neither input nor targets.

1 Introduction

3D Object detection has been studied in the recent years and there have been many methods used to estimate 3D objects. One related work is estimating the 3D bounding box from 2D images, but these methods can only generate coarse 3D detection results due to the lack of depth information and can be substantially affected by appearance variations. Another way is directly learning from 3D point clouds. Thanks to [1] and [2], we can directly process the sparse 3D point cloud and learn representations from them. And then based on [1] and [2], Shaoshuai Shi.etc proposed PointRCNN [3], which was a two-stage framework to directly operates on 3D point clouds and achieve accurate 3D detection performance. PointRCNN consists of two stages, the first stage aims at generating 3D bounding box proposal in a bottom-up scheme, and the second stage conducts 3D box refinement.

From the DLAD course we are already provided with the coarsely predicted bounding boxes and the learned point features from stage-one as well as a baseline model which can do some basic bounding box refinement in stage-two. The baseline model consists of mainly three parts: The first part is several Set Abstration layers proposed in [2], the second and third parts are regression head and classification head. In order to improve the refinement model in stage-two, we decided to make changes before the Set Abstration layers and implement the canonical method which was proposed in [3].

2 Method

Problem of baseline model: In the default baseline stage-two model there is a problem: all the points from stage-one are in the same coordinate system and then fed into several Set Abstraction layers. But for different bounding boxes, their location in the coordinate system are quite different and this difference in location could be strong noise for the mlps in the Set Abstraction as well as later regression and classification head. In order to eliminate this noise and to make our model invariant to the bounding box location (and orientation), we implement a canonical transformation per bounding box as well as all pooled points in this bounding box.

Canonical Transformation: The canonical transformation is implemented as follows: for each bounding box and the pooled points in it, we transform the points from original coordinate system into a new canonical coordinate system represented by the bounding box. Explicitly, we make the origin of the new coordinate system to be the center of the coarsely predicted bounding box, align the new x axis to be the head direction of the bounding box, remain the y axis unchanged (still pointing downwards), and the z axis is orthogonal to both x and y axis. Since our bounding box is represented in the format (x,y,z,h,w,l,θ), we can have a rather simple two-step implementation to achieve this transformation:

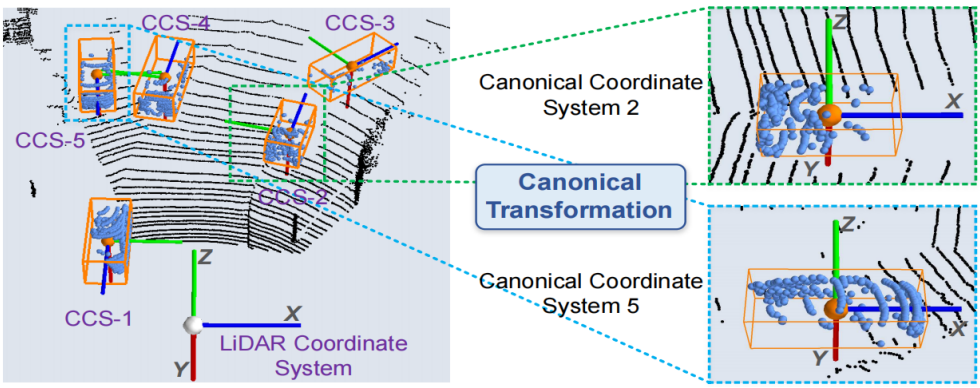


Figure 1: Canonical Transformation illustration. LiDAR Coordinate System is the original coordinate system, and Canonical Coordinate System is the new coordinate system. Note that this figure comes from the PointRCNN paper, and we are using it here just for illustration purpose. We hope we are not violating any regulations as this is a virtual paper for a course project.

Step 1. From every point's coordinate $p = (x^p, y^p, z^p)$ subtract the coordinate of the bounding box's center point $c = (x^c, y^c, z^c)$, that is: $p' = p - c = (x^p - x^c, y^p - y^c, z^p - z^c)$.
Step 2. For every p', rotate it along y axis by θ to get p''.

Now p'' is the coordinate of this point in the new coordinate system. An illustration of this canonical transformation is in Figure 1.

Problem of Canonical Transformation: Now that we have transformed our pooled points into a new local coordinate system, and our model could learn better the local refinement for the bounding box without being misled by the location and orientation of the bounding box. But our targets for the refinements are still in the original coordinate system, so in order to correctly supervise the learning, we need to transform the learned refinements back to original coordinate system before we compare them to the targets and compute the loss.

Reverse Canonical Transformation: In order to compare the predicted refinements with the target refinements, we do a reverse canonical transformation to the predicted refinements to convert them back to the original coordinated system. The predicted refinement is $r = (x^r, y^r, z^r, h^r, w^r, l^r, \theta^r)$ which is in the new (Canonical) coordinate system. Note that we recorded the bounding box's center point coordinate $(x^c, y^c, z^c)$and its θ in the original (LiDAR) coordinate system and we are going to use them:

Step 1: $r' = (x^r, y^r, z^r, h^r, w^r, l^r, \theta^r + \theta)$
Step 2: Rotate r' along y axis by $-\theta$ to get $r'' = (x^{r''}, y^{r''}, z^{r''}, h^{r''}, w^{r''}, l^{r''}, \theta^{r''})$
Step 3: $r''' = (x^{r''} + x^c, y^{r''} + y^c, z^{r''} + z^c, h^{r''}, w^{r''}, l^{r''}, \theta^{r''})$

Now the predicted refinement r''' is in the original coordinate system and we can compute the regression loss by $Loss(r''', r^{target})$ and finish the training procedure, where $Loss$ is the predefined regression loss function. **Note that our method is slightly different from the PointRCNN[3] in the sense that our model embeds the canonical transformation and detransformation into the training procedure and does not require any change for input or targets, while [3] needs to apply canonical transformation to the targets to produce a new set of targets, which can be a problem when test targets are inaccessible, just like in this project.**

Figure 2: Loss-accuracy curves of default model(blue) and CanonicalTransformation model(yellow)

3 Experiments

From Figure 2 we can see the canonical transformation model greatly increases the speed of convergence and also apparently improves the final accuracy.

From Tabel 1 we can see the canonical transformation model improves the accuracy apparently.**And our model ranks 8th in the Codalab leaderboard**.

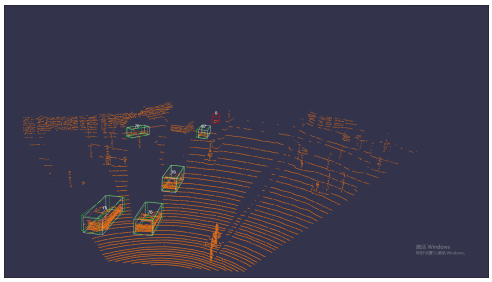
From Figure 3 we can see that the canonical transformation model can better handle far away bounding boxes than the default model.

Name	Canonical Transformation	e_mAP, m_mAP, h_mAP
G52_0624-2143_default_BEV_61b36	×	87.652, 83.458, 77.306
G52_0624-2143_default_BEV_61b36 Codalab	×	87.68, 81.95, 76.18
G52_0701-0640_CT_version2_0a917	✓	94.695 , 89.275, 88.34
G52_0701-0640_CT_version2_0a917 Codalab	✓	91.79, 87.90, 86.94

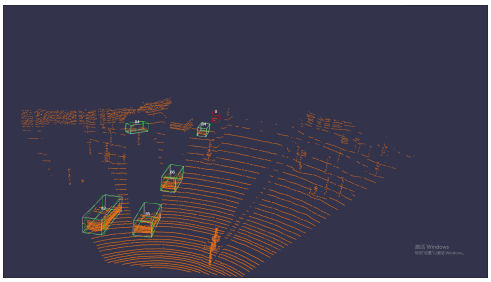
Table 1: Performance of baseline model and canonical transformation model.

4 Discussion

Why it works? From the experiments above we can see that the canonical transformation method brings big improvement to the refinement model, and this is because essentially the refinement represents just a local change relative to the coarsely predicted bounding box. By using canonical transformation we instruct our model not to care about the location or orientation of the bounding boxes but to focus on learning the refinement itself, so the model is kept away from noises and can effectively learn more.



(a) Default model



(b) Canonical Transformation model

Figure 3: Final prediction of bounding boxes, white boxes are ground truth, green boxes are correctly predicted boxes and red boxes are wrongly predicted boxes.

Model Weakness: In the canonical transformation we transform points into a local coordinate system which is helpful for learning the local refinement. But the bounding boxes far from the LiDAR tends to have much sparser points than the near bounding boxes. In order to overcome this issue, we may include the depth information of the bounding box (or of the points) to help our neural network learn this phenomenon. We could also include segmentation masks into the model so that the model knows which points are real foreground points and which points are background points in the enlarged bounding box, and hopefully the model can learn to assign more importance to the foreground points. However, due to consistent crashes of the online training, we didn't manage to explore more possibilities.

Summary of Contribution: We proposed an embedding canonical transformation and detransformation method in the training procedure ,which does not require any input or target changes, and can adapt to the situation where test target is inaccessible.

5 Other Attempts

We also made other attempts trying to improve the default model, such as changing the mlp configurations in the Set Abstraction layer, or adding self attention layer into the model. For the mlp configurations we tried to add one more 1x1 conv at the front of every set abstraction, but in the end the performance was lower than the baseline model. For the self attention part we didn't manage to implement it, **but we come up with a novel idea which we believe can work well**:

Idea of using self-attention: When using segmentation masks, our intention is to let the model know which points are foreground points and hopefully the model can learn by itself to assign more weights(importance) to these foreground points. When using depth information our intention is to let the model know whether the point cloud for this bounding box is dense or sparse. Then we can fuse these information by self-attention in a way that segmentation masks can dynamically assign weights to points by inspecting the depth information: if the depth of the bounding box is low, then we probably want to assign more weights only to the foreground points because there are dense foreground points which would be already enough to predict the bounding box. If the depth of the bounding box is high, then we need to assign a bit larger weights to background points because we don't have many valid foreground points, and we may want to also make use of those misclassified background points which are actually foreground points to help us make the prediction. And this kind of dynamic weighting can be perfectly done by the gate mechanism in self-attention[4], and this is why we think it would work.

References

[1] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *CoRR* abs/1612.00593 (2016). arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

[2] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *CoRR* abs/1706.02413 (2017). arXiv: 1706.02413. URL: http://arxiv.org/abs/1706.02413.

[3] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. "PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud". In: *CoRR* abs/1812.04244 (2018). arXiv: 1812.04244. URL: http://arxiv.org/abs/1812.04244.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is All You Need". In: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf.