IBB-Net: Fast Iterative Bounding Box Regression for Detection on Point Clouds

Brendan Miller

June 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Simon Lucey, Advisor
Katerina Fragkiadaki
Allie Chang

Submitted in partial fulfillment of the requirements for the degree of Master of Robotics.

Keywords: Bounding Box Regression, 3D Detection, PointNet, Iterative Networks

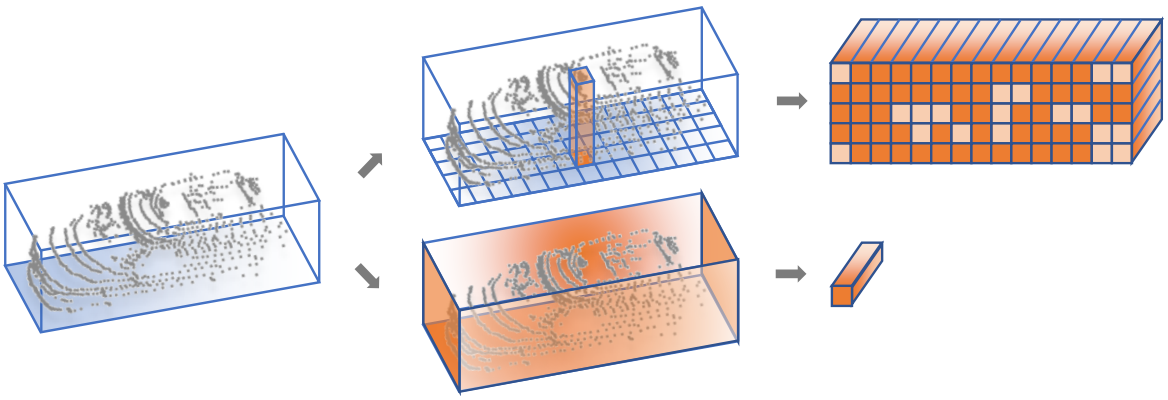


Figure 1: PointNet vs PointPillar representations of point clouds. Given LIDAR data of a car shown on the left, either a PointNet representation or a voxelized representation of a point cloud can be generated. Orange is used to indicate the embedding in either voxelized or compact PoitnNet form. The top lane shows the application of PointPillars style voxelization, which applies PointNet to voxels individually to create a high dimensional feature map. The bottom lane shows the direct application of PointNet to create a compact global feature of the entire cloud. The PointNet embedding has two advantages. It is computationally cheaper to generate, and because the representation is compact, networks that utilize the representation as input are also faster. This insight motivates new ways of using PointNet that we call IBB-Net (Iterative Bounding Box Network). As we will show, it is faster to apply several iterations of PointNet on bounding box regression than to regress in a single shot using PointPillars.

Abstract

Currently, most point cloud based detection pipelines are focused on producing high accuracy results while requiring significant computational resources and a high-end GPU. Our research explores how to reduce the computational overhead by improving a key element of detection: bounding box regression. We demonstrate a fast and iterative method of bounding box regression which has significant run-time performance advantages over existing leading methods. The iterative structure of our method also gives the system control over the trade-offs between accuracy and speed at run-time. We furthermore integrate our bounding box regression method into an existing detection pipeline and motivate additional research into how the first stage of the pipeline can be modified to take better advantage of the performance characteristics of our bounding box regression method.

Acknowledgments

I would like to express gratitude towards my advisor Simon Lucey. Under Simon's guidance I've learned both how to do scientific research, and how to clearly convey my ideas to the research community. I would also like to thank Allie Chang who has given me advice on detection and tracking and helped introduce me to much of the current research. Finally, I would also like to thanks all of my fellow students in the CI2CV lab who are doing great research and served as an inspiration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

PointNet [Lang et al.(2018)Lang, Vora, Caesar, Zhou, Yang, and Beijbom] at its origins was an innovative framework for compactly representing 3d point clouds for tasks as varied as classification, alignment, and segmentation. One of the reasons the vision community got excited about the framework was its computational efficiency in terms of processing the points, the compactness of the representation, and the subsequent efficiency of performing tasks with such a compact representation.

In subsequent years several extensions to PointNet have been proposed. A common thread to all of these innovations has been what we refer to as the *voxelization* of PointNet. We define PointNet voxelization to mean the local application of PointNet to regions of 3D space, resulting in a multi-channel voxel representation for use with a convolutional neural network. Examples include VoxelNet [Zhou and Tuzel(2018)], Second [Yan et al.(2018)Yan, Mao, and Li], and PointPillars [Yan et al.(2018)Yan, Mao, and Li]. Although achieving substantial performance gains, these innovations have moved away from PointNet's key benefit: a compact and efficient global representation.

In this paper, we question whether a voxelization strategy is always required. Specifically, we look at the PointPillar representation, which is now commonly used for detection and bounding box prediction in the 3D detection space, and propose an alternative approach using PointNet and iterative transformations of the point cloud we call IBB-Net (Iterative Bounding Box Network). Our approach embraces the efficiency of PointNet to do bounding box regression with far less computational and memory overhead.

This strategy is inspired by IT-Net [Yuan et al.(2018)Yuan, Held, Mertz, and Hebert] and also derives inspiration from classic alignment techniques such as Lucas-Kanade [Bruce and Kanade(1981)] and ICP (Iterative Closest Point) [Besl and Neil D. McKay(1992)]. Lucas-Kanade demonstrated that alignment posed as a non-linear optimization problem could be solved iteratively using simple pixel features and a local linear approximation. We apply this insight to point clouds and bounding box regression, and show that cheap to compute global PointNet features can be used to iteratively refine bounding boxes.

This iterative strategy also has the advantage that it allows for an explicit trade off between accuracy and computational overhead. Reducing the number of iterations makes the system faster at the expensive of accuracy. Many robotics applications have limited computational power on board, so giving the implementer explicit control of this trade off allows our approach to be deployed to systems that other methods would not be able to run on.

We demonstrate the strength of our method with IBB-Net, a PointNet based network which iteratively regresses bounding boxes, compared against a PointPillar based network doing single shot regression under the same conditions. We find that performing PointNet regression iteratively is 9 times faster than single shot PointPillars regression. Furthermore, we find that while the PointPillars based regression is more accurate than single shot PointNet based regression, an iterative PointNet regression is even better.

We also integrate our network into a two stage detection pipeline and demonstrate that IBB-Net is more robust to error than the leading two stage network that we compare against. This opens up the possibility of creating an overall detection pipeline that is far more computationally efficient, by reducing the time spent in the Region Proposal Network, and instead relying on IBB-Net's robustness to regress bounding boxes from relatively inaccurate region proposals.

Contributions

- We make an argument against voxelization as a means of processing point clouds, as being inefficient and

memory intensive. Instead, we advocate for using compact global representations of point clouds, specifically PointNet, for common tasks such as bounding box regression.

- We further propose IBB-Net, an iterative method for bounding box regression, which uses a global PointNet representation and is 9 times faster than the fastest voxelized representation for the same task.
- Our approach gives the implementation control over the trade off between accuracy and runtime in order to allow it to run on real world systems with limited computational resources.
- We demonstrate the possibility of creating much faster detection pipelines that pair a computationally cheap and less accurate Region Proposal Network with our robust and fast iterative bounding box regression network IBB-Net.

Chapter 2

Related Work

2D Detection

Modern 2D object detection pipelines utilize deep convolutional feature maps. Early pipelines were multi stage, and utilized a separate region proposal network, then resampling the image and performed detection within image patches [Girshick et al.(2014)Girshick, Donahue, Darrell, and Malik]. The computational expensive of resampling and recomputing the feature map led to the development of architectures which avoid resampling raw pixels [Ren et al.(2017)Ren, He, Girshick, and Sun], and later single shot architectures [Redmon et al.(2016)Redmon, Divvala, Girshick, and [Liu et al.(2015)Liu, Anguelov, Erhan, Szegedy, Reed, Fu, and Berg].

The introduction of single shot architectures mitigated the performance problem, but introduced an extreme imbalance of positive and negative examples. Focal loss [Lin et al.(2017)Lin, Goyal, Girshick, He, and Dollar] was developed to help address this problem, and is typically used in conjunction with data augmentation [Lang et al.(2018)Lang, Vora, Caesar,

3D Detection

In order to do 3D object detection on LIDAR data, a variety of methods have been developed which transform sparse point cloud data into a format suitable for detection on a convolutional pipeline. Earlier methods use hand designed encodings for birds eye view [Ku et al.(2017)Ku, Mozifian, Lee, Harakeh, and Waslander] [Chen et al.(2017)Chen, Ma, Wan, Li Many recent works use learned features based on PointNet divided into a a voxel representation [Zhou and Tuzel(2018)] [Yan et al.(2018)Yan, Mao, and Li] [Lang et al.(2018)Lang, Vora, Caesar, Zhou, Yang, and Beijbom].

VoxelNet [Zhou and Tuzel(2018)] divides the point cloud into voxels, generates PointNet features for each voxel, then uses a 3D convolutional middle layer to remove the third dimension. Afterwards, a 2D detection pipeline is applied to the resulting 2D feature map. The use of 3D and 2D convolution on sparse point clouds limits the performance of VoxelNet. Second [Yan et al.(2018)Yan, Mao, and Li] speeds up the VoxelNet framework by using a custom sparse convolution algorithm.

PointPillars [Lang et al.(2018)Lang, Vora, Caesar, Zhou, Yang, and Beijbom] takes a different tactic, and instead of using a complicated sparse convolution algorithm, uses pillars, essentially voxels with infinite extent in the z dimension, to avoid having to perform 3D convolution. PointPillars provides major performance gains, but still must create a dense 2D feature map from the sparse point cloud. Even on high end GPU's, only one or two point clouds can be held in memory at a time using this very large feature map. Due to the high memory requirements, PointPillars and other convolutional approaches are not usable in scenarios without large amounts of GPU memory, such as most robotic scenarios.

PointRCNN [Shi et al.(2018)Shi, Wang, and Li] is an example of a two stage approach to detection. It uses a Region Proposal Network (RPN) based on PointNet++ [Qi et al.()Qi, Yi, Su, and Guibas] estimate locations of objects within a point cloud. It then uses a refinement network, RCNN, to get an accurate bounding box. Our work contains experiments which modify the PointRCNN pipeline and replace RCNN with our own network, IBB-Net.

PointNet

PointNet [Qi et al.(2016)Qi, Su, Mo, and Guibas] was developed to address the question of how deep networks can process unordered sets of data such as point clouds. It creates a pointwise deep embedding, and does max pooling to combine the pointwise embedding into an embedding for the entire cloud.

Compared to voxelized representations of point clouds, a PointNet embedding is far more efficient to compute and results in a compact fixed length representation which consumes far less memory. The drawbacks of PointNet

are that the individual elements of a PointNet embedding vector do not have an interpretable spatial location in the way that a voxel does, and as a deep embedding it can't be passed directly into a multi-layer convolutional architecture.

PointNet Based Alignment

While PointNet lacks direct spatial interpretability, several works on point cloud alignment using PointNet [Yuan et al.(2018)Yuan, Held, Mertz, and Hebert] [Aoki et al.(2019)Aoki, Goforth, Srivatsan, and Lucey] show that spatial information can be extracted from PointNet. The original PointNet paper has an embedded T-Net which attempts to internally align the point cloud to a canonical orientation before further processing.

IT-Net [Yuan et al.(2018)Yuan, Held, Mertz, and Hebert] improves on this design by predicting rigid Euclidean transformations and performing the transformations iteratively. While IT-Net aligns a point cloud to a canonical orientation, PointNetLK [Aoki et al.(2019)Aoki, Goforth, Srivatsan, and Lucey] aligns a point cloud to a template point cloud using a differentiable variant of the Lucas-Kanade algorithm [Bruce and Kanade(1981)].

Chapter 3

Method

3.1 Background

IT-Net [Yuan et al.(2018)Yuan, Held, Mertz, and Hebert] demonstrated that a point cloud could be aligned to a canonical orientation iteratively using simple PointNet features. At each step, a transformation is predicted, and the original point cloud is transformed.

This pattern of using relatively weak, but cheap features to iteratively regress a transformation comes from the Lucas Kanade algorithm [Bruce and Kanade(1981)]. Lucas Kanade performs Gauss-Newton gradient descent [Baker and Matthews(2004)], and the IT-Net paper showed that the network also appears to be performing gradient descent, with the magnitude of transformations decreasing as they approach the optimum. Our PointNet based iterative bounding box regression, IBB-Net, is derived from IT-Net's design, but we extend it beyond alignment to regress all of the bounding box parameters used in point cloud detection.

3.2 Common Framework

Both the IBB-Net regression and the PointPillars based regression are built on top of a common framework which provides identical inputs for training, and evaluates both networks with the same criteria. The training examples are derived from KITTI [Geiger et al.(2012)Geiger, Lenz, and Urtasun] object detection. Following the method of PointPillars [Lang et al.(2018)Lang, Vora, Caesar, Zhou, Yang, and Beijbom], additional noise is added. Point clouds in a window surrounding each car are extracted, centered, and then translation error is added. This forms the input to both networks. The networks output a transformation of the bounding box to canonical orientation and scale as discussed in Section 3.3. This is done so that transformations from multiple iterations can be composed. Bounding box parameters are extracted from the final transformation.

Both networks use mean euclidean distance of points to canonical position as a loss. Given $\mathbf{T}$ and $\mathbf{S}$ as ground truth euclidean transformation and scale transformation to canonical position, and given $\tilde{\mathbf{T}}$ and $\tilde{\mathbf{S}}$ as the estimated transformations, the loss $\mathcal{L}$ can be computed like so:

$$\mathcal{L}((\mathbf{T},\mathbf{S}),(\tilde{\mathbf{T}},\tilde{\mathbf{S}})) = \frac{1}{|\mathbb{X}|}\sum_{\mathbf{x}\in\mathbb{X}} \|(\mathbf{S}\mathbf{T}\mathbf{x}) - (\tilde{\mathbf{S}}\tilde{\mathbf{T}}\mathbf{x})\|_2 \tag{3.1}$$

This is similar to PLoss [Xiang et al.(2018)Xiang, Schmidt, Narayanan, and Fox], but does not take the square of the distance. It's also identical to EMD, earth mover's distance [Fan et al.(2017)Fan, Su, and Guibas] when the assignment between point sets is fixed.

3.3 Transformation to Canonical Bounding Box

Our network regresses the parameters of a bounding box iteratively using a series of Euclidean transformations $\mathbf{T}^{(i)}$, and scaling transformations $\mathbf{S}^{(j)}$. The combined result is of the form $\mathbf{TS} = \mathbf{T}^{(t)}\mathbf{T}^{(t-1)}\cdots\mathbf{T}^{(1)}\mathbf{S}^{(j)}\mathbf{S}^{(j-1)}\ldots\mathbf{S}^{(1)}$.

Care must be taken to regress Euclidean and scaling transformations separately, as interleaving them results in a combined affine transformation $\mathbf{A} = \mathbf{S}^{(t)}\mathbf{T}^{(t)}\mathbf{S}^{(t-1)}\mathbf{T}^{(t-1)}\cdots\mathbf{S}^{(1)}\mathbf{T}^{(1)}$ that cannot be decomposed into $\mathbf{A} = \mathbf{ST}$. The geometric interpretation of this is that scaling a non-axis aligned bounding box results in shear. In practice, we found it sufficient to use only a single scaling transformation.

We use 7 parameters to describe a bounding box. The center of the bounding box x, y, z. The width, length, and height w, l, h. Finally, a single angle in the plane θ. Our networks regress these parameters and use them to transform the point cloud into canonical orientation and scale. For the parameters x, y, z, θ we form a restricted euclidean transformation T.

$$\mathbf{T} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & x \\ \sin(\theta) & \cos(\theta) & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

IBB-Net regresses $\mathbf{T}$ in several steps similar to IT-Net [Yuan et al.(2018)Yuan, Held, Mertz, and Hebert].

The input to the network is a set of n homogeneous coordinates $x_j = [x_j, y_j, z_j, 1]^T$ for $j = 1, \cdots n$. Let us denote this initial set of points as:

$$\mathbb{X}^{(0)} = \{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \cdots, \mathbf{x}_n^{(0)}\} \tag{3.3}$$

Then we treat our network as a function $\Phi(\cdot)$ that takes a point set $\mathbb{X}^{(i)}$ and returns a transformation $\mathbf{T}^{(i+1)}$ that moves point set closer to the canonical orientation:

$$\mathbf{T}^{(i+1)} = \Phi(\mathbb{X}^{(i)}) \tag{3.4}$$

Given the transformation $\mathbf{T}^{(i+1)}$ and the point cloud $\mathbb{X}^{(i)}$, we can calculate the transformed point cloud which is the input to the next stage of the iteration:

$$\mathbb{X}^{(i+1)} = \{\mathbf{T}^{(i+1)}\mathbf{x}_j^{(i)} : j = 1, \cdots, n\} \tag{3.5}$$

If we perform t applications of equations 3.4 and 3.5, we can calculate the final Euclidean transformation:

$$\mathbf{T} = \mathbf{T}^{(t)}\mathbf{T}^{(t-1)}\cdots\mathbf{T}^{(1)} \tag{3.6}$$

After the euclidean transformation matrix $\mathbf{T}$ is calculated, a second network regresses the parameters for a scaling transformation $\mathbf{S}$.

$$\mathbf{S} = \begin{bmatrix} \frac{a_w}{w} & 0 & 0 & 0 \\ 0 & \frac{a_\ell}{\ell} & 0 & 0 \\ 0 & 0 & \frac{a_h}{h} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Here the scaling incorporates both the bounding box parameters w, l, h, and fixed anchor sizes a_w, a_l, a_h. The anchor width, length, and height describe the canonical length, width, and height of a car. The scaling transformation transforms a car point cloud in canonical position into canonical scale.

3.4 IBB-Net: Iterative PointNet Bounding Box Regression

IBB-Net processes points in a similar manner to IT-Net [Yuan et al.(2018)Yuan, Held, Mertz, and Hebert]. As shown in Figure 3.1, our network produces a sequence of euclidean transformations $\mathbf{T}^{(1)}, \mathbf{T}^{(2)}, \cdots \mathbf{T}^{(n)}$ to move the car's bounding box into canonical orientation. Then a scale transformation $\mathbf{S}$ is produced to move the car into canonical scale. Finally, the original bounding box parameters are extracted from the composed euclidean transformations and the scale transformation.

The network is implemented as a pointnet embedding, followed by fully connected layers to extract the bounding box parameters. A simple 3 layer fully connected network is used to produce the PointNet embedding. ReLU and batch normalization are applied to each layer. The last layer has no ReLU, but instead performs max pooling.

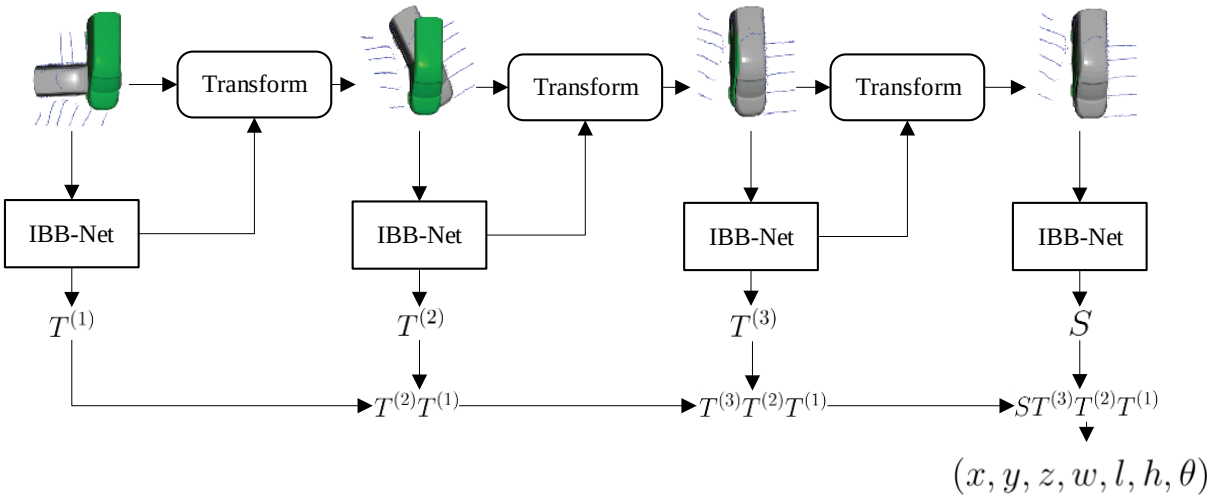


Figure 3.1: IBB-Net diagram. We take advantage of the fact that the PointNet representation is cheap to compute in order to regress bounding boxes through several iterations faster than a voxelized representation can calculate in a single shot. The point cloud in birds eye view format at the top left is the input. The bounding box in is indicated in red. IBB-Net outputs a Euclidean transformation $\mathbf{T}^{(i)}$ which transforms the bounding box closer to canonical orientation and scale. The transformation transforms the point cloud, which is used as input to the next step. After the bounding box is in canonical orientation, the network predicts a scaling transformation $\mathbf{S}$ which transforms the bounding box into canonical scale. Finally, the transformations are composed, and the bounding box parameters are extracted.

The PointNet features are then passed through a 3 layer fully connected network to produce the transformation parameters. Note that the network outputs the angle θ as a unit complex number. This parallels IT-Net which outputs 3D rotations as a unit quaternion. The network is initialized so that the identity transformation is predicted by default. A subnetwork with the same structure, but different outputs produces the scaling transformation $\mathbf{S}$. We calculate $\mathbf{S}$ in one shot, though an iterative regression would be an obvious extension.

3.5 PointPillars Bounding Box Regression

We built a PointPillar based bounding box regression network as a baseline to compare IBB-Net against. Our PointPillars regression network accepts the point cloud in stacked pillar format as described in the PointPillar paper [Lang et al.(2018)Lang, Vora, Caesar, Zhou, Yang, and Beijbom]. Generating this point cloud format is an expensive operation, so to get a fair comparison, we utilize the same optimized function from the PointPillars implementation.

Once the point cloud is in voxelized representation, we pass it to our network. Our network again utilizes the PointPillars implementation to produce what the PointPillars paper calls a pseudo-image, which is essentially a 2D feature map. From here, we apply a 3 layer convolutional network. The output of the convolutional network is then passed to a 4 layer fully connected network, which outputs our transformation parameters in similar manner to the PointNet layer.

The major difference between the PointPillar based network and IBB-Net is the use of input representation, the use of 2D convolution internally, and that the PointPillar network regresses in one shot. Because our PointPillars based network does not output multiple anchors, we do not need to perform non-maximum-suppression. Details are in Section 4.3. The PointPillar network also has a subnetwork for producing the scaling transformation in the same manner as in section 3.4.

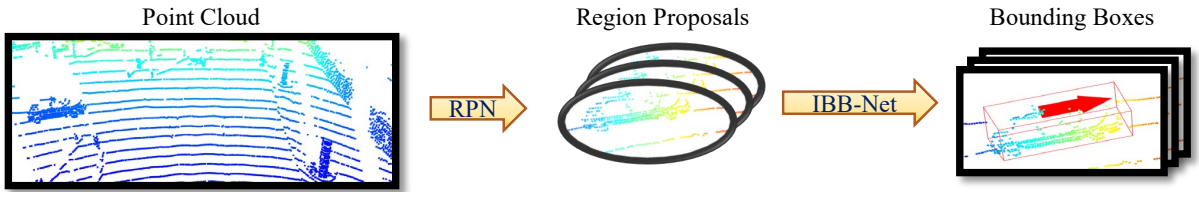


Figure 3.2: We use IBB-Net for detection by integrating it into a two stage pipeline. In the first stage the Region Proposal Network extracts approximate regions of a point cloud where detectable objects are likely located. The second stage uses IBB-Net to regress the bounding box within the region proposal.

3.6 IBB-Net for Detection

IBB-Net is designed to regress a single bounding box from a point cloud using cheap global features. IBB-Net cannot be used directly with large point clouds which contain many detectable objects. Instead, it must be integrated into a two stage detection pipeline containing a region proposal network as shown in Figure 3.2. The point cloud is input into the Region Proposal Network, which outputs several rough estimates of the location of detectable objects within the point cloud. IBB-Net is then fed the points from the region proposals, and regresses an accurate bounding box for each region.

We used PointRCNN's [Shi et al.(2018)Shi, Wang, and Li] Region Proposal Network as in conjunction with IBB-Net. We also performed experiments comparing the performance of IBB-Net and RCNN: PointRCNN's built in bounding box regression network. Our experiments using IBB-Net for detection are discussed in detail in section 4.11.

Chapter 4

Experiments

In the environment described in section 3.2, we compared both the accuracy and the runtime performance of IBB-Net and the PointPillars based networks IBB-Net is 9 times faster than the PointPillars based network as shown in Table 4.6. There is no loss of accuracy as seen in Figure 4.4.

4.1 Data Generation

Our networks are trained and tested on point cloud data from the KITTI object detection dataset [Geiger et al.(2012)Geiger, Lenz, and U
During training time, additional noise is added to the dataset in the same manner as used to train PointPillars [Geiger et al.(2012)Geiger, Lenz, and Urtasun]. We assume that our networks will be used in conjunction with a RPN (Region Proposal Network). Because of this, it's reasonable to assume that our network will be provided with approximate, but noisy, prior knowledge about where cars reside in the point cloud.

Thus, we choose 5 by 5 regions in the x y plane around each car. This is a large enough region to contain any car, and also picks up a reasonable amount of background terrain. We then center the car at the origin, and offset it with random translations in x, y, and z according to a unit gaussian distribution in each dimension. In other words, we sample from $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma = 1$. Because some examples in the KITTI dataset are of distant objects with few points, we discard any extracted point cloud regions with fewer than 125 points. If there are more than 1000 points in a point cloud, we only keep 1000 selected randomly. This allows us to process the point cloud in batch using a dense tensor.

4.2 Network Training

Both networks were trained with the Adam optimizer with an intial learning rate of 0.0005 decayed by 0.7 every 2000 step. We use a batch size of 100 for 2600 batches on the KITTI dataset. Each batch element contains a single point cloud containing a single 5 x 5 point cloud region as described in 4.1.

4.3 Network Parameters

IBB-Net first extracts PointNet features with 3 linear layers. The first layer has 3 input channels for x,y,z and 64 outputs channels. The subsequent layers have 128 and 1024 output channels respectively. BatchNorm and ReLU are applied after each layer, except the last. Then max pooling is applied to get the global PointNet feature.

The global PointNet feature is then fed into another 3 linear layers with output channels 512, 256, and 5. Of the final 5 channels, the first 3 are taken as the translation x, y, z. The final 2 are interpreted as a unit complex number to represent a rotation in the x, y plane. This is analogous to how unit quaternions are used to represent 3 DoF rotations.

The PointPillar based network takes voxels in stacked pillar format as input as described in [Lang et al.(2018)Lang, Vora, Caesar, Zh
It then creates a pseudo-image with the same network structure as described in PointPillars. This pseudo-image is

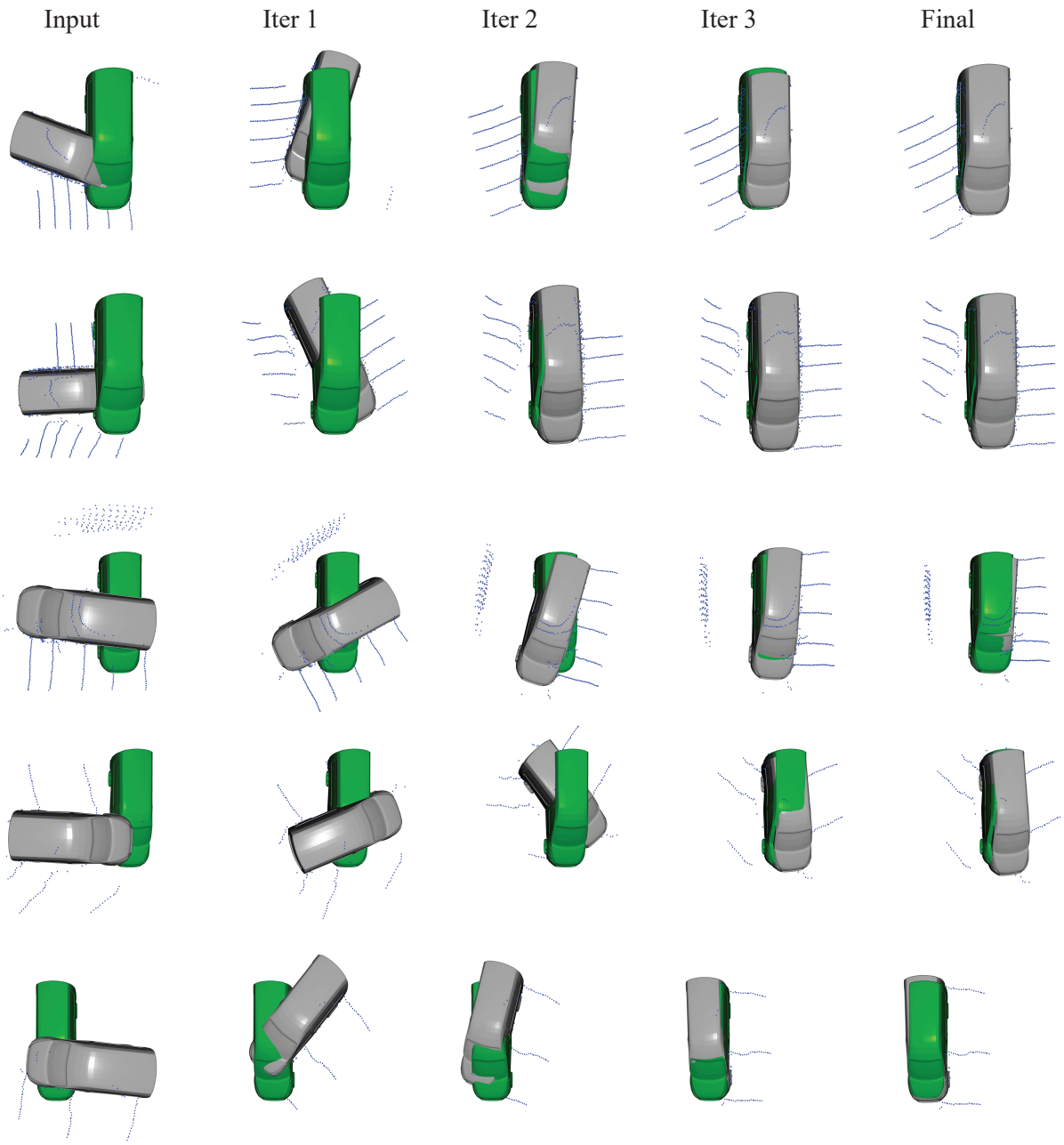


Figure 4.1: Birds-eye view visualizations of IBB-Net bounding box regression with 3 iterations. The gray car model shows the location of the car within the point cloud as it is iteratively transformed into canonical orientation and scale. The green car model in the final column indicates the desired canonical orientation and scale. The scaling transformation is applied only in the final column.

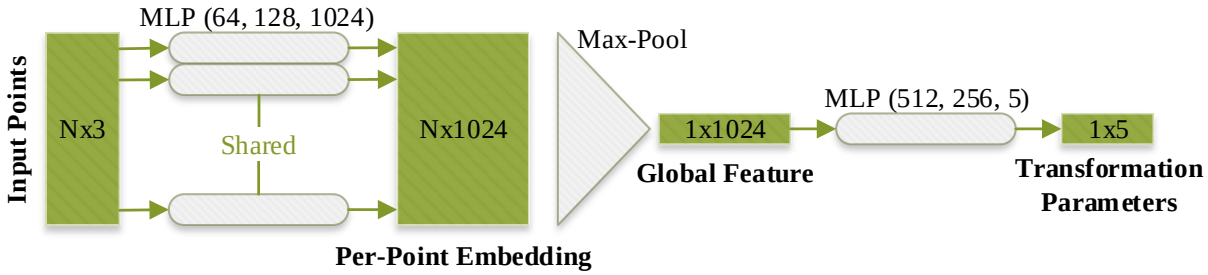


Figure 4.2: IBB-Net Network Architecture. For N input points, IBB-Net uses an MLP and max-pooling to generate a PointNet global feature. Then an MLP is used to generate 5 parameters for a Euclidean transformation as described in Section 3.3.

effectively a feature map, and is fed into a series of 3 2D convolutional layers with 64 inputs channels, and output channels 64, 128, 256 respectively. A kernel size of 3 is used, and stride of 2. BatchNorm and ReLU are applied to each layer. The output of the convolutional layers is then flattened and passed to 4 linear layers with input channel 12544, and output channels 1024, 512, 256, and 5 respectively. BatchNorm and ReLU are applied to each layer except the last. The 5 output channels are interpreted in the same way as the 5 output channels of IBB-Net.

For both IBB-Net and the PointPIllars based network, there are corresponding scale networks. These scale networks have identical structure in terms of network layers to the networks discussed above. The only difference is that instead of having an output of 5 parameters, they have an output of 3 parameters which are interpreted as the diagonal of the scale matrix $\mathbf{S}$ as discussed in Section 3.3.

4.4 Network Architecture

Figure 4.2 visualizes the internal structure of IBB-Net as described in Section 4.3. Each MLP layer applies BatchNorm and ReLU to its output, except for the last layer. The first 3 parameters of the output transformation parameters are taken as translation in x, y, z. The last two parameters are taken as a complex number used to represent a rotation in the X, Y plane in a similar way that a quaternion is used to represent a 3 DoF rotation.

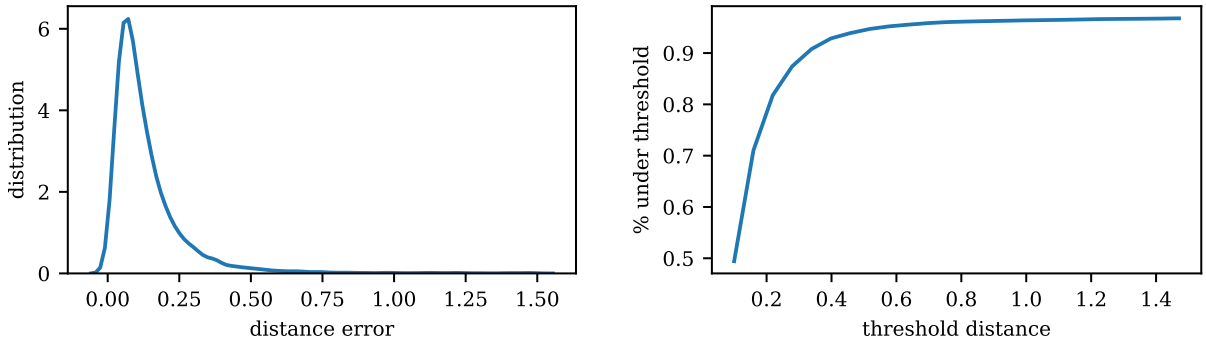
4.5 Canonical Parameters

IBB-Net and the PointPillars based network both produce transformations of the point cloud to a canonical orientation and scale. In our implementation, we consider the canonical orientation to be with the car's bounding box centered at the origin and facing down the negative y axis. The canonical scale is described by 3 anchor parameters as used in 3.3. The canonical width, length, and height a_w, a_l, a_h are $1.6, 3.9, 1.5$. These are identical to the size anchors used within PointPillars.

4.6 Area Under the Curve (AUC)

In order to perform experiments where we compare the accuracy of out network in terms of rotation and translation using a similar metric, we introduce an Area Under the Curve (AUC) metric. When the network is run over the test data set, every result has a certain rotation and translation error as shown in Figure 4.3 (a). For each kind of error, a series of thresholds is established and the fraction of examples with error less than that threshold is recorded as shown in figure 4.3 (b). The chart relating an increasing threshold to increasing accuracy forms a monotonically increasing curve. We sum the area under the curve, and normalize it to a value between 0 and 1, resulting in the AUC metric.

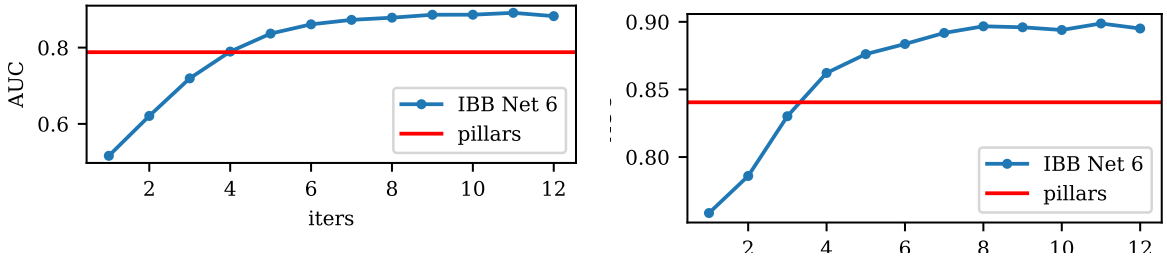
The AUC metric has the advantage that it produces a single metric which considers all error thresholds simultaneously. The alternative is to consider multiple thresholds separately as in Figure 4.7.



(a) Distribution of translation error

(b) Threshold curve

Figure 4.3: In this paper, we use a metric dubbed Area Under the Curve (AUC) that is useful for measuring both translational and rotational error in a uniform way. (a) shows the the error over a test set in terms of translation. Given that data set, (b) shows the fraction of results with less than a given threshold of error. The area under the curve in (b) is summed and normalized to between 0 and 1 to create the AUC metric.



(a) Rotation accuracy vs iterations.

(b) Translation Accuracy vs iterations.

Figure 4.4: Error over iterations. (a) Rotation accuracy in terms of Area Under the Curve metric. (b) Translation accuracy in terms of Area Under the Curve metric. The blue line refers to IBB-Net, and the red line indicate the PointPillars based network. For a single shot, the PointPillars based network performs best, but after several iterations IBB-Net pulls ahead.

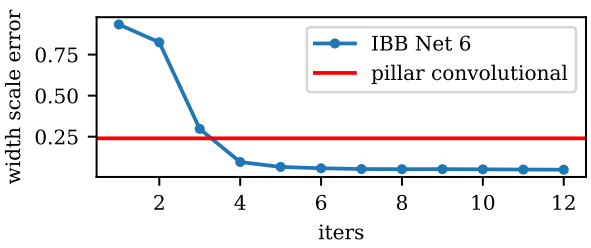
4.7 Accuracy Comparison

We compared the performance of bounding box regression by measuring the rotation and translation accuracy of the networks separately. As seen in figure 4.4, (a) for IBB-Net, as the number of iterations increased, the accuracy in terms of the AUC metric also increases.

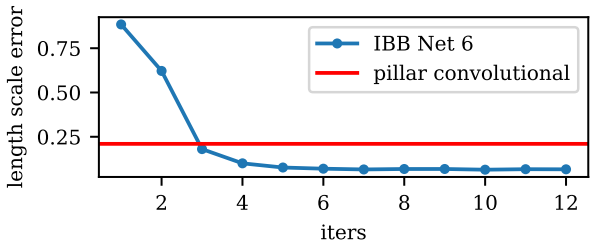
Interestingly, the network was only trained on 6 iterations, but rotation continued to improve as more iterations were performed at evaluation time. This is in accordance with what was seen in the IT-Net paper, and gives evidence to the idea that the network is using a gradient descent method to align the point cloud at each step. For the first few iterations, IBB-Net is outperformed by the single shot network, but by iteration 6, it has shot ahead. As will be covered in the performance section, 6 iterations of IBB-Net is still far computationally cheaper than the a single shot PointPillar regression.

4.8 Scale Error

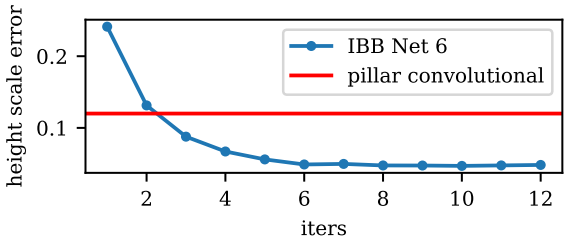
In Figure 4.5 we show the median absolute error in scale in the width, length, and height dimensions. The scale networks for both IBB-Net and the PointPillars based network regress in a single shot. However, as seen in Figure



(a) Width scale error vs iterations.



(b) Length scale error vs iterations.



(c) Height scale error vs iterations.

Figure 4.5: Median absolute scale error over iterations. Note that scale is regressed in only a single iteration. The blue line indicates IBB-Net error, and the red line indicates the scale error from the PointPillars based network. The number of iterations here refers to the number of iterations of Euclidean transformations that precede the scale transformation. As can be seen, if we start with a better Euclidean alignment, the single shot scale alignment performs much better.

4.5 IBB-Net still does a better job regressing object scale if the Euclidean transformation described in Section 3.3 is regressed using more iterations.

4.9 Runtime Performance

We compared the running time of IBB-Net running with 6 iterations, and our single shot PointPillars network. We excluded the time to load the point cloud from disk and segment out the car example. We timed from when the point cloud segment containing the car was provided, until when when the network had output the final transformation to canonical orientation and scale.

For the PointPillars based network this running time included the time to convert the point cloud into a stacked pillar format using PointPillars optimized code. For IBB-Net the running time included 6 iterations for every example. We calculated both the PointPillars and IBB-Net examples in batch, and divided the running time by the total number of examples processed. Figure 4.6 summarizes the performance. We observe that a single shot of PointPillars bounding box regression is many times slower than IBB-Net with 6 iterations. We attribute this to the expense of creating and processing the much larger PointPillars representation, vs the compact PointNet representation.

4.10 Impact of Iterations on Accuracy

In Section 4.9 we showed that multiple IBB-Net iterations are computationally cheaper than PointPillar based single shot bounding box regression. In this section we will show that increasing the number of iterations used at train and evaluation time is an easy way to increase the accuracy of regressed bounding boxes. This allows implementations to easily dial up the accuracy of a system to the level needed within the hardware constraints present in robotics and autonomous driving scenarios.

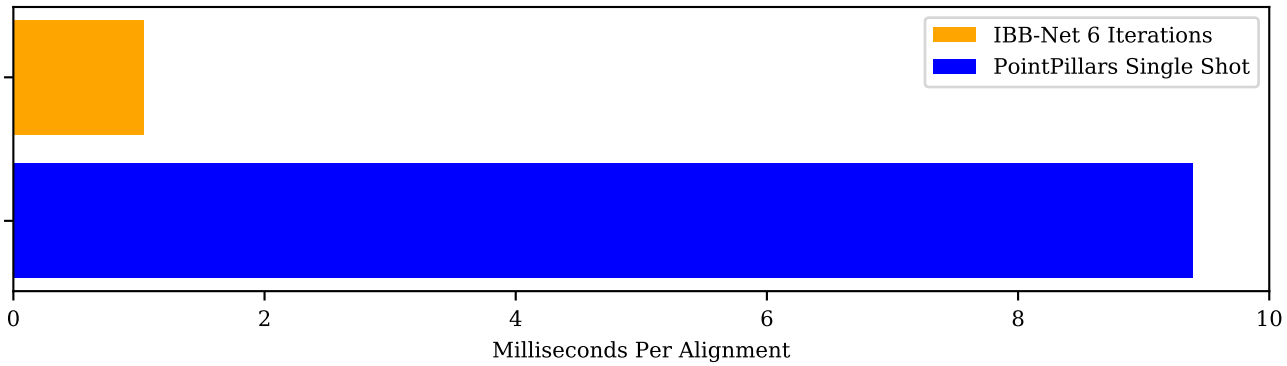


Figure 4.6: Milliseconds per alignment. A single shot of PointPillars is 9 times slower than IBB-Net run over 6 iterations. This demonstrates the large computational benefit of using the compact PointNet representation over a voxelized representation. Running time is calculated in batch, and the reported numbers are mean running time per example.

Model	$\leq 5°$	$\leq 10°$	$\leq 20°$
IBB-Net 1	24.8	40.8	57.0
IBB-Net 2	28.1	45.8	62.7
IBB-Net 3	38.4	53.7	66.7
IBB-Net 4	44.2	58.4	68.3
IBB-Net 5	49.3	62.1	70.1
IBB-Net 6	53.4	64.7	71.4

Table 4.1: Rotation accuracy % for different iterations and thresholds. For any threshold, the accuracy increases as the number of iterations increases, validating the efficacy of applying iterations to bounding box regression across multiple measures.

Similarly to IT-Net [Yuan et al.(2018)Yuan, Held, Mertz, and Hebert] we found that networks trained on multiple iterations continue to improve in accuracy as the number of iterations is increased at evaluation time. This is even true if the network is evaluated on more iterations than it is trained on. This indicates that the network is performing something akin to gradient descent as it aligns a point cloud to a canonical orientation and scale.

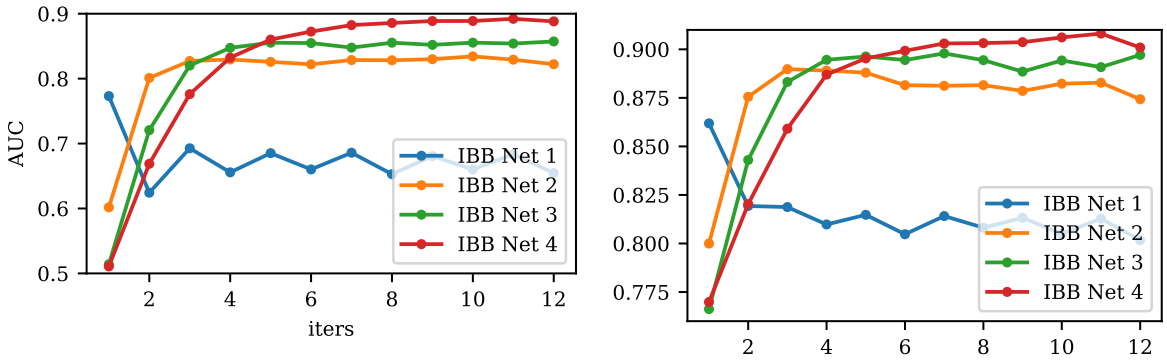
In Figure 4.7a we show that the rotation accuracy of the regressed bounding box improves with the number of applied iterations. The network trained with 6 iterations has the highest performance. Table 4.1 shows that rotation accuracy is increasing no matter which iterations threshold we choose. In this table, we are are performing evaluation with the same number of iterations that we train on.

In Figure 4.7b we show the median translation error decreasing as the number of evaluated iterations increases, with the IBB-Net trained on 6 iterations having the lowest median translation error after 12 evaluation time iterations. Median error is used instead of mean in order to ignore noisy outliers in the KITTI dataset for which the network cannot regress a bounding box.

4.11 Detection Experiments

All experiments up to this point have considered the accuracy and run-time of IBB-Net independently from a detection pipeline. We also integrated IBB-Net into PointRCNN [Shi et al.(2018)Shi, Wang, and Li], a leading two stage detection pipeline. IBB-Net replaces the second stage of the detection pipeline. We use region proposals from PointRCNN's Region Proposal Network, and process them with IBB-Net to regresss a bounding box.

We compare the Average Precision (AP) of our network acting as the second stage with the normal PointRCNN refinement stage (RCNN). Furthermore, because the region proposals from the RPN are very accurate, we introduce



(a) Rotation accuracy vs iterations

(b) Translation error vs iterations.

Figure 4.7: Comparison of training vs test iterations. (a) Rotation accuracy in terms of Area Under the Curve metric.. (b) Rotation accuracy in terms of Area Under the Curve metric. These charts show that training on more iterations improves performance as long as the network is evaluated using at least as many iterations. Evaluating a network on more iterations than it was trained on continues to boost performance.

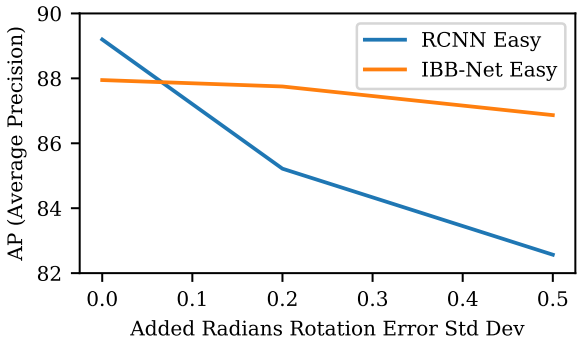


Figure 4.8: Average Precision of IBB-Net and PointRCNN's refinement stage (RCNN) operating as the second stage of a two stage detection network. As the X axis increases, more rotational error is introduced into the region proposals. Note that RCNN is slightly more accurate with very small amount of error, but that IBB-Net is much more robust to error. RCNN assumes a very accurate initialization from the Region Proposal Network.

increasing amounts of noise into the region proposals in order to compare the robustness of the IBB-Net and RCNN as shown in Figure 4.8.

We find that with very low error in region proposals, RCNN is slightly more accurate, but as the error increases, IBB-Net starts to outperform. This shows the relative robustness to error of our iterative approach, and suggests that IBB-Net can be used with less accurate Region Proposal Networks.

In Figure 4.9 we compare the runtime performance of IBB-Net and RCNN with the Region Proposal Network. We find that the RPN takes the vast majority of the run-time. This shows that if a computationally cheaper, but less accurate RPN was used with IBB-Net, we should be able to dramatically reduce the overall run-time of the detection pipeline. As we had shown in our previous experiment, the robustness of IBB-Net makes it well suited for use with less accurate Region Proposal Networks.

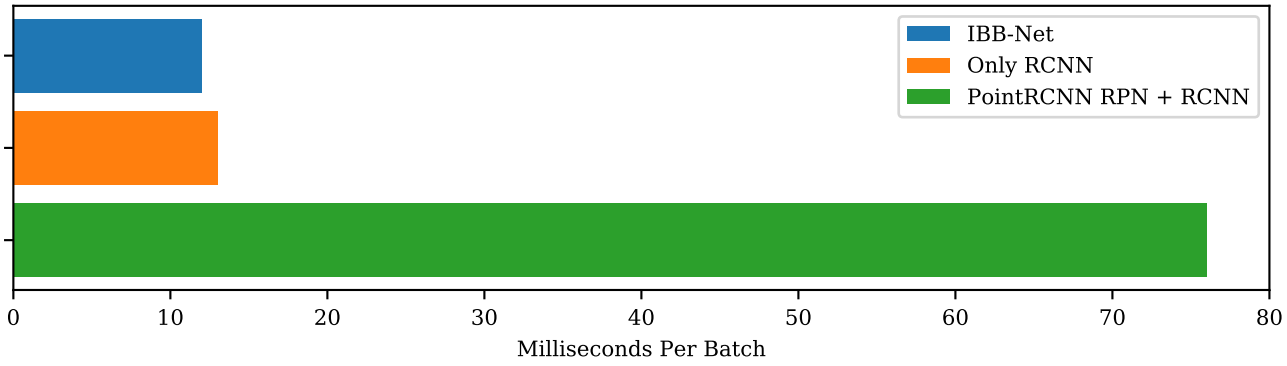


Figure 4.9: Milliseconds per batch. We compare the time spent on the two refinement networks with the time spent on the Region Proposal Network. This demonstrates that the vast majority of run-time is spent on the Region Prosposal Network. Because IBB-Net is much more robust to inaccurate initialization from the RPN, this suggests that major performance gains can be achieved if an RPN which is computationally cheaper, but less accurate can be developed.

Chapter 5

Discussion

We demonstrate the speed and accuracy of our network on the bounding box regression task in isolation, and in conjunction with a Region Proposal Network as part of a two stage detection Pipeline. Our experiments show that our approach is both fast, and more robust than competing approaches. However, because the Region Proposal Network we compared against is very accurate and computationally intensive, while IBB-Net is robust to noise it would be highly desireable to replace the RPN with a less accurate, but faster Region Proposal Network. This is currently an open area of research which we would like to pursue in the future.

Chapter 6

Conclusion

We introduce IBB-Net, a fast iterative method of bounding box regression that can be integrated into a two stage detection pipeline. Our approach has the advantage that it allows that it allows the implementation to control the trade off between accuracy and computation simply by changing the number of iterations that are performed.

Our results compare the performance of PointNet and dense convolutional features in the form of PointPillars representation for bounding box regression. We find that IBB-Net, our PointNet based network with multiple iterations, is at least as accurate, and far faster than PointPillars. Our work suggests that a non-voxelized approach can outperform voxelized approaches.

We also integrate IBB-Net into a two stage detection pipeline, and show the robustness our our network in the face of error. This suggests a new avenue of research to create a computationally cheaper, but less accuracy Region Proposal Network which takes advantage of this robustness of IBB-Net to dramatically decreases the overall run-time of the detection pipeline.

Bibliography

[Aoki et al.(2019)Aoki, Goforth, Srivatsan, and Lucey] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. PointNetLK: Robust & Efficient Point Cloud Registration using PointNet. In *CVPR2019*, 2019. URL http://arxiv.org/abs/1903.05711. 2

[Baker and Matthews(2004)] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56(3):221–255, feb 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000011205.11775.fd. URL https://doi.org/10.1023/B:VISI.0000011205.11775.fd. 3.1

[Besl and Neil D. McKay(1992)] Paul J Besl and Neil D. McKay. A Method for Registering of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–255, 1992. 1

[Bruce and Kanade(1981)] D. Lucas Bruce and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. *IJCAI*, 130:121–130, 1981. 1, 2, 3.1

[Chen et al.(2017)Chen, Ma, Wan, Li, and Xia] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D object detection network for autonomous driving. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:6526–6534, 2017. doi: 10.1109/CVPR.2017.691. 2

[Fan et al.(2017)Fan, Su, and Guibas] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3D object reconstruction from a single image. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:2463–2471, 2017. doi: 10.1109/CVPR.2017.264. 3.2

[Geiger et al.(2012)Geiger, Lenz, and Urtasun] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. ISSN 10636919. doi: 10.1109/CVPR.2012.6248074. 3.2, 4.1

[Girshick et al.(2014)Girshick, Donahue, Darrell, and Malik] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587. IEEE, jun 2014. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.81. URL http://ieeexplore.ieee.org/document/6909475/. 2

[Ku et al.(2017)Ku, Mozifian, Lee, Harakeh, and Waslander] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. 2017. ISSN 09574484 13616528. doi: arXiv:1712.02294v4. URL http://arxiv.org/abs/1712.02294. 2

[Lang et al.(2018)Lang, Vora, Caesar, Zhou, Yang, and Beijbom] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast Encoders for Object Detection from Point Clouds. 2018. URL http://arxiv.org/abs/1812.05784. 1, 2, 3.2, 3.5, 4.3

[Lin et al.(2017)Lin, Goyal, Girshick, He, and Dollar] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:2999–3007, 2017. ISSN 15505499. doi: 10.1109/ICCV.2017.324. 2

[Liu et al.(2015)Liu, Anguelov, Erhan, Szegedy, Reed, Fu, and Berg] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-yang Fu, and Alexander C Berg. SSD: Single Shot MultiBox

Detector. 1:21–37, dec 2015. doi: 10.1007/978-3-319-46448-0_2. URL http://arxiv.org/abs/1512.02325http://dx.doi.org/10.1007/978-3-319-46448-0{_}2. 2

[Qi et al.()Qi, Yi, Su, and Guibas] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NIPS*. 2

[Qi et al.(2016)Qi, Su, Mo, and Guibas] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. dec 2016. URL http://arxiv.org/abs/1612.00593. 2

[Redmon et al.(2016)Redmon, Divvala, Girshick, and Farhadi] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 27, pages 779–788. IEEE, jun 2016. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.91. URL http://ieeexplore.ieee.org/document/7780460/. 2

[Ren et al.(2017)Ren, He, Girshick, and Sun] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. ISSN 01628828. doi: 10.1109/TPAMI.2016.2577031. 2

[Shi et al.(2018)Shi, Wang, and Li] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. 2018. URL http://arxiv.org/abs/1812.04244. 2, 3.6, 4.11

[Xiang et al.(2018)Xiang, Schmidt, Narayanan, and Fox] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. 2018. doi: 10.15607/rss.2018.xiv.019. 3.2

[Yan et al.(2018)Yan, Mao, and Li] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors (Switzerland)*, 18(10):1–17, 2018. ISSN 14248220. doi: 10.3390/s18103337. 1, 2

[Yuan et al.(2018)Yuan, Held, Mertz, and Hebert] Wentao Yuan, David Held, Christoph Mertz, and Martial Hebert. Iterative Transformer Network for 3D Point Cloud. nov 2018. URL http://arxiv.org/abs/1811.11209. 1, 2, 3.1, 3.3, 3.4, 4.10

[Zhou and Tuzel(2018)] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4490–4499. IEEE Computer Society, dec 2018. ISBN 9781538664209. doi: 10.1109/CVPR.2018.00472. 1, 2