3D Bounding Box Prediction for Embedded Systems

Department of Computer Science
Aalborg University
June 10, 2021

Ahmet Pekbas
apekba18@student.aau.dk

Christoffer Najbjerg Knudsen
cnkn16@student.aau.dk

Rasmus Barrett
rbarre16@student.aau.dk

Abstract

As autonomous vehicles become common, the variety of mobile robots increases. These robots are becoming increasingly independent and have to act based on their observations in the environment. An essential task for self-driving robots is the detection and localization of objects in the environment. This paper focuses on developing a model suited for embedded systems to detect objects in 3D. Previous approaches often focus on either image detection or point clouds. We propose a backbone, which utilizes both image and LiDAR data as inputs to infer better information by including a novel fusion. The idea is to leverage the best properties from each input for more accurate detection. For the prediction head, we test two approaches: a center-based and a single-shot detection. Through testing on the KITTI 3D benchmark, we show that our proposed model is inadequate in learning prediction of 3D bounding boxes, both with a front-facing image feed as well as a preprocessed Bird's-Eye View (BEV).

I. Introduction

Deploying autonomous vehicles in unknown or unmapped environments is a complex technological challenge. Among other tasks, the vehicles need to detect and track moving objects such as other vehicles, pedestrians, and cyclists in real-time. Given the advances in deep learning methods for computer vision, much research has been invested in how technology can be applied for object detection. The trend in recent years is to use increasingly powerful computers to produce accurate and precise models for both 2D, and 3D bounding box prediction [2, 31, 3].

However, these improved models depend on powerful computing units, consisting of an increasingly high number of operations for each pass-through. This makes the state-of-the-art models unfeasible for embedded systems such as the Jetson Nano. The embedded systems are intended to be deployed on mobile devices, where both computation time and energy consumption are of interest. Embedded systems have become much more relevant as the next gen-

eration of autonomous vehicles and robots becomes available.

This paper presents an attempt at building a model that provides a learned fusion between sparse point clouds and camera images with dense RGB information. Specifically, our model takes both a point cloud and a single front-facing image as input. For the point cloud, the model uses a standard LiDAR-based encoder, Pillar Feature Net [11], to build a representation, which is flattened into a Bird's-Eye View (BEV). For the input image, we scale and encode the image to a higher dimension representation.

For the backbone, we present a two streams encoder inspired by ESANet [25]. The bottom stream encodes the BEV encoding, and the top encodes the image. The encoding uses a ResNet34 with the residual blocks are replaced with Non-Bottleneck-1D-Blocks (NBt1Ds) [24]. At each dimensional stage of the ResNet34, we present a novel fusion between the two streams. The fusion uses an attention mechanism based on depth-wise separable convolution. The result of the fusion is further encoded in the top stream.

For detection head, we follow CenterPoint [31], which locates objects by the center point of the object and regresses to all other object properties such as 3D size and orientation for each detected center.

For testing the model, we use the KITTI 3D benchmark [4]. Where we show, the model setup is inadequate for convergence to a useful prediction.

II. Related Work

In the past, several methods for detecting objects have been introduced. This chapter focuses specifically on those used for bounding box prediction.

A. 2D and 3D Object Detection

Over the years, object detection has aimed at either predicting either 2D or 3D objects. 2D object detection uses an image and works by predicting axis-aligned bounding boxes. 3D object detection aims to predict three-dimensional rotated bounding boxes, often from point clouds, sometimes augmented with RGB or semantic information.

In 2D detection, the main approaches relate to the RCNN detector family [5, 6, 10, 23], which finds candidate bounding boxes to classify and refine. The best-known approaches are YOLO [22], SSD [18], and RetinaNet [17]. These models rely on a learned number of anchor boxes to predict objects. Many 3D object detectors have drawn inspiration from these 2D object detectors [8, 26, 27, 30]. The use of anchor boxes provides an easy translation, as they can encode a 3D size given the size of the anchor box itself, providing a simple translation from 2D to 3D. However, these methods share the same problems as 2D detection, such as the quantity and scale of the anchor boxes.

3D object detection is more diverse in its approach and differs from 2D detectors in both the input and backbone structure [4, 11, 29, 30]. The input to 3D prediction is often in the form of sparse point clouds. To process a point cloud directly for prediction is computationally expensive and often infeasible. Thus, the point cloud requires encoding into a dense representation. The two mainstream approaches are VoxelNet [34] and PointPillars [11].

VoxelNet divides the sparse point clouds into equally spaced 3D voxels. PointNet [21] is then used inside each voxel to generate a unified feature representation. A prediction head of 3D convolutions and 2D convolutions can then predict objects from this representation.

PointPillars substitutes the voxels with a pillar representation, a single tall elongated voxel for each grid location. By having it this way, 2D convolution replaces expensive 3D convolutions.

B. Object Detection Approaches

The most common approaches for detecting and finding objects can be categorized based on predicting regions, anchors, or key points.

1) Region Classification

One of the first successful deep object detectors is RCNN [6]. RCNN works by looking through object locations from a large set of region candidates [28]. These regions are cropped and classified using a deep network. An improvement of this method is Fast-RCNN [5] which saves computation by cropping image features instead. However, these methods rely on slow low-level region proposal methods, making them unsuited for real-time performance.

2) Anchor Boxes

Another group of predictors works by sampling fixed-shaped bounding boxes, known as anchors, around a low-resolution image grid and classifying each into foreground or background. A well-known example is Faster-RCNN [23] which generates region proposals within the detection model.

An anchor is labeled as foreground if it significantly overlaps with any ground truth object, background if the overlap is below a threshold, and unknown if the overlap is in between. The regions of interest, i.e., the foreground, are classified again to obtain the objects [23]. Changing the proposal classifier to a multi-class is the basis of one-stage detectors. In recent years, there have been several improvements to one-stage detectors include anchor shape priors [22, 13], different feature resolution [18], and loss re-weighting among different samples [17]. However, the extraction and combination of interest regions are still computationally expensive processes.

3) Key Point Prediction

A different approach is the key point prediction. Key points classify important places in the scene. CornerNet [12] detects two corners of the bounding box as key points, while ExtremeNet [33] detects the top-, left-, bottom-, right-most point of all objects. However, both require a combinatorial grouping stage after key point detection, which significantly slows down each algorithm. Current state-of-art approaches like CenterNet [32] and CenterPoint [31], extract a single center point per object and regress the box features. This significantly reduces the complexity of the task.

C. Fusion of Image and Point Cloud

There have been multiple studies on the possibility of multi-sensor fusion to leverage the best properties of each sensor. The benefit of camera-based approaches is the dense information available in the image, while LiDAR provides sparse but precise 3D information.

A direct approach is proposed by HorizonLiDAR3D [1], where a 2D detector is used to predict 2D bounding boxes and semantic segmentation. LiDAR points

falling within a bounding box are augmented with semantic information. Likewise, Cross-Modality3D [35] also augments points with high-level semantic information. However, it handles the sparseness by applying RoI-wise feature fusion to learn denser representations for refinement.

UberATG-MMF [15] takes a different approach by using the point cloud as a sparse depth map. This depth map is learned together with an image to predict a dense depth map, from which a pseudo LiDAR is made to fuse the image and point cloud encodings.

ContFuse [14] proposed to aggregate BEV with image features by projecting LiDAR points onto the image. This approach interpolates BEV pixel location with image features based on K-nearest neighbor search.

D. Approach Inspiration

Based on the current research, we focus on a center-based approach to predicting bounding boxes. This is because center-based representation dramatically simplifies the task and fits better with embedded systems [32, 31].

Objects do not follow any particular orientations in the 3D world, making it hard to predict the bounding boxes, as they usually do not align with any global coordinate system. The reason being 3D objects come in a broad range of sizes, shapes, and aspect ratios than in 2D, e.g., bicycles are flat, buses and limousines are elongated, and pedestrians are tall.

The key advantage of center-based representation is that points have no intrinsic orientation. Furthermore, it simplifies subsequent tasks such as tracking through the relative offset of objects between consecutive frames. The simplification should also make it possible to design much simpler and faster modules [31].

For encoding of the point cloud, we look at the two state-of-the-art approaches, VoxelNet and PointPillars. VoxelNet voxelizes the point cloud, providing more 3D information giving better accuracy at the trade-off of higher memory usage. On the other hand, PointPillars simplifies the representation for speed-up and lower memory, with the trade-off being accuracy. This paper focuses on the PointPillars representation as it matches better with embedded systems in terms of memory and speed.

Most of the previous methods use either images or point clouds, a few methods use augmentation, and only a limited number of methods use a direct fusion of dense image information and the sparse point cloud. The high computational requirement of the methods containing fusion is the reason we investigate a direct fusion between image and point cloud that can be run on embedded systems.

III. Preliminaries

This section describes how the Pillar Feature Net [11] and the CenterPoint prediction head [31] work and the purpose of these. We have used these two models for point cloud encoding and bounding box prediction.

A. Pillar Feature Net

A mainstream approach to process point clouds is to encode them into pillars, with the Pillar Feature Net as presented in PointPillars [11]. The idea is to group points within a grid of pillars and apply a 2D convolutional architecture to get a learned pseudo-image from the pillars.

The first step is to divide the point cloud into an evenly spaced grid in the x-y plane, creating a set of pillars, P. Note that there is no need for a hyperparameter to control the binning of the z dimension. In the point cloud, each point is represented with coordinates x, y, z, and reflectance r. The points are augmented with x_c, y_c and z_c where the c subscript denotes the arithmetic mean distance of all the points in the same pillar. Thus, the augmented points are now $D = 7$ dimensional.

Next, to produce a learned representation of the points of size C, a simplified version of PointNet [21] is applied. For each point, a linear layer[1] is applied, and the result is batch normalized, and a ReLU activation generates a (P, N, C) sized tensor. Afterward, a max operation over the channels, C, creates an output tensor of size (P, C). Once encoded, the pillars are scattered back to the original pillar locations in the grid to create a pseudo-image of size (H, W, C) where H and W indicate the height and width of the grid.

The set of pillars will be mostly empty due to the sparsity of the point cloud, with the non-empty pillars having few points in them. For example, at $0.16m \times 0.16m$ grid the point cloud from KITTI [4] has 6k-9k non-empty pillars which is a $\sim$97% sparsity [11]. By exploiting the sparsity, a limitation on both the number of non-empty pillars per scene, P, and on the number of points per pillar, N, creates a dense tensor of size (P, N, D). If a pillar holds too much data to fit in this tensor, the data is randomly sampled. Conversely, if a pillar has too little data to populate the tensor, zero paddings are applied.

[1]Note that the linear layer is formulated as a 1×1 convolution across the pillar resulting in very efficient computation.

B. CenterPoint Prediction Head

For the predictions in this paper, we follow the prediction head of the first stage in CenterPoint [31], where classes are grouped into tasks. The first stage of CenterPoint predicts dense heat maps, sub-voxel location refinement, object size, and rotation for each task.

1) Heat Map Head

The head's goal is to produce a heat map peak at the center location of any detected object in BEV. The heat map head predicts a K-channel heat map, one channel for each K class in the tasks.

Objects in a BEV are sparser than in an image. In BEV, distances are absolute, while an front-view has perspective distortion. An example of this is a scene of a road with vehicles. In BEV, the area occupied by vehicles is relatively small, but a few large objects may occupy most of front-view. Furthermore, the compression of the depth places object centers closer to each other [31].

2) Regression Head

The goal of the regression head is to predict the bounding boxes. For each object, several properties are predicted. These properties are: a sub-voxel location refinement $o \in \mathbb{R}^2$, the elevation of the center point $h_c \in \mathbb{R}$, the bounding box size $s \in \mathbb{R}^3$, and a yaw-rotation as $(sin(\alpha), cos(\alpha)) \in \mathbb{R}^2$. The sub-voxel location refinement, o, reduces the quantization error from voxelization and striding of the backbone. The elevation, h_c, adds the missing elevation information removed by the BEV projection. The orientation prediction uses the sine and cosine of

the yaw as a continuous regression target. Combined with box size, the regression heads provide the full state information of the 3D bounding box. To better handle boxes of various shapes, sizes are regressed to the logarithmic value. At inference time, all properties are extracted by indexing into the outputs of the regression heads at each object's peak location in the heat map.

IV. Framework

This section covers our framework structure and the reasons we have selected this structure. The layout of our approach can be seen in Figure 1 on page 7. The framework is divided into three stages: encoding the input, a backbone, and the prediction head for predicting the bounding box attributes.

A. Input Streams

Our model takes a front-facing camera image and a point cloud with points from within the camera's field of view as input. The first stage aims to get an equal-sized representation of the image and point cloud to allow for the fusion of the data in the backbone.

The image is first scaled to get a consistent size that can be convoluted to the exact dimensions of the pillar grid. Since the image is wide-angle, it is pixel-wise large in the horizontal direction. Thus, we use an asymmetrical stride in the first convolution to decrease the learned representation's width. The goal is for the convolutions to learn a translation of the input to a BEV representation.

For the point cloud, we use a Pillar Feature Net encoding [11] to convert the point cloud into a learned BEV pseudo-image.

B. Backbone

The image and pillar encoder both use a ResNet34 architecture [9]. Thus, the resulting feature maps at the end of the encoder are 16 times smaller than the input. ResNet34 is chosen as it provides a reasonable trade-off between speed and accuracy in embedded systems [25]. We replace the residual blocks with a spatially factorized version. This version is named NBt1D and is depicted in Figure 1 with purple outline and was initially proposed by ERFNet [24]. NBt1D is better at retaining spatial information, with minimal impact on efficiency.

The encoded pillar features are fused into the image features at the start and after each resolution stage in ResNet34. For the fusion, we propose a novel approach. The approach is outlined in light green on Figure 1. We propose an attention mechanism that utilizes a depth-wise separable convolution. By first performing a per channel 2D convolution, we allow for different weighting of objects at different heights in the feature maps. The following 1 channel convolution combines the weights to a single attention map. The attention map is multiplied by the feature representation. In this way, the model can learn which features to enhance and suppress. After the attention mechanism, the pillar and image representation is concatenated, and a convolution combines the results and brings the channels down to the input channel size.

The rest of the backbone is for decoding the encoding. We follow [25] and utilize three decoding blocks. The number of channels is not decreased for the first block, while the rest of the blocks halve the channel in the first convolution. It is then further

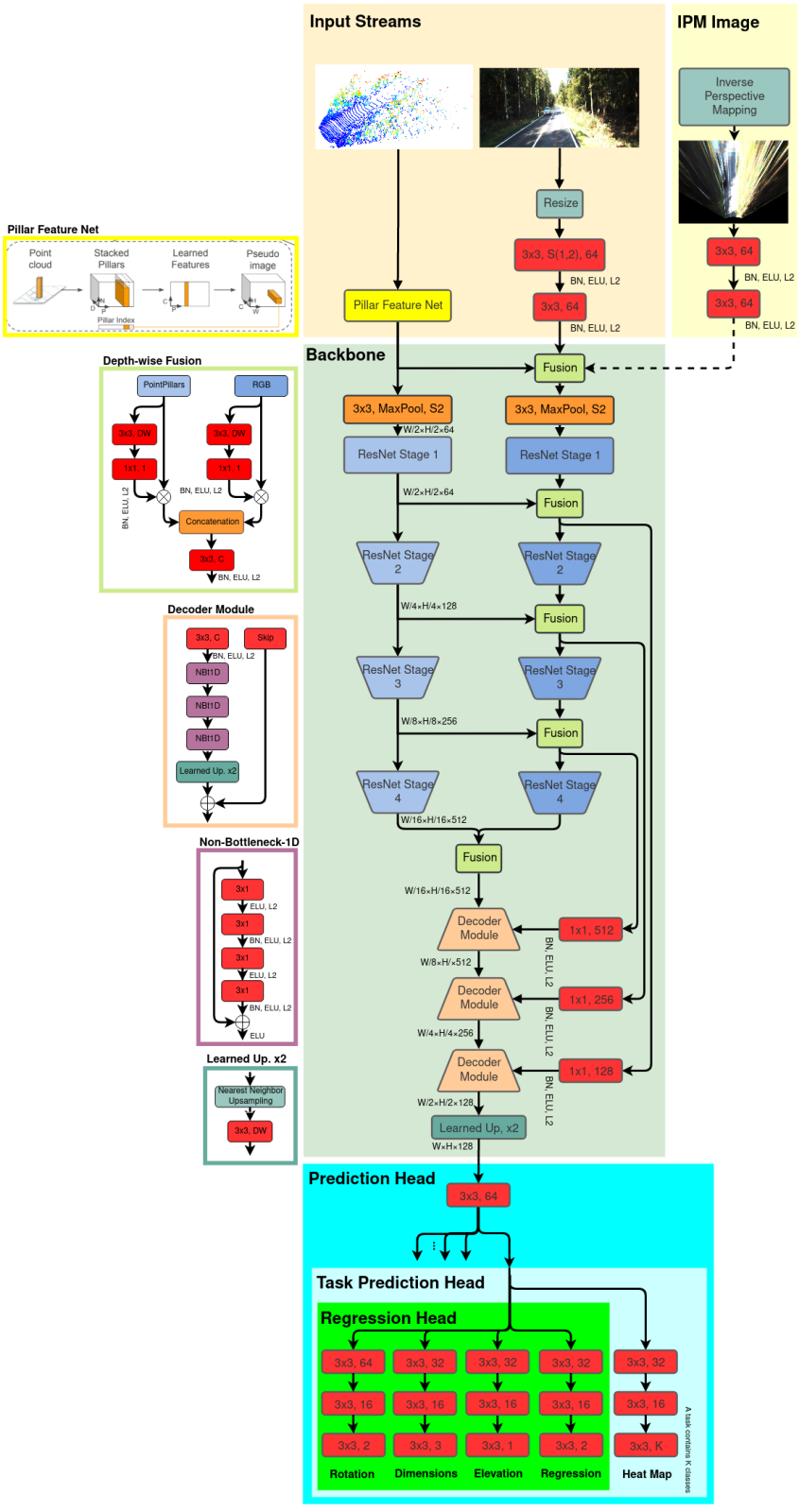

Figure 1. Overview of our proposed model architecture. Red boxes are convolutions, written in the format of MxN for the kernel size, S(x, x) is the stride, and the number at the end is the channels; if no channel is given, it is the same as the input. Convolutions marked with DW are depth-wise convolutions. IPM Image represents an alternate input image approach.

convoluted with three NBt1Ds before being passed to a learned upsampling method. The learned upsampling works by first using nearest neighbor upsampling, and afterward, a depth-wise convolution is applied to combine adjacent features.

C. Prediction Head

We follow the design of CenterPoint [31] for the prediction head. The benefit of the head is that it indicates objects based solely on a single location instead of relying on multiple overlapping bounding boxes.

This removes the requirement of a manual threshold between foreground and background, allowing for only one positive peak per object. Therefore, Non-Maximum Suppression (NMS) is not needed, reducing the computation time and making it better suited for embedded systems. The more straightforward approach of representing objects as points greatly simplifies 3D recognition.

V. Experiments

This section goes through the data set, our model setup, and the training process. We use a cluster server with a single NVIDIA Tesla v100 to training the model. To evaluate the suitability of the model design on embedded systems, we record the inference time on the Jetson Nano Developer Kit running Jetpack v4.5. Jetson Nano is equipped with a 128-core Maxwell GPU, Quad-core ARM A57 CPU, and 4GB 64-bit LPDDR4 25.6 GB/s memory. The experiment is performed in the Jetson Nano's 5W mode. The CPU is limited to two cores in this mode, with the decreased maximum CPU frequency from 1479MHz down to 918MHz. Likewise, the maximum GPU frequency is decreased from 921.6MHz to 640MHz.

Hyperparameter	Value
Min/Max x	0.0m/80.0m
Min/Max y	-52.8m/52.8m
Pillar size x/y	0.2m/0.2m
Max points per pillar, N	50
Max pillars, P	6000
Learned channel, C	64
Image scale size	400px $\times$ 1056px
Tasks	{Car, Van, Truck, Tram}, {Cyclist}, {Pedestrian, Person sitting}, {Misc}

Table I
Model hyperparameters

A. Data Set

We evaluate our approach on the KITTI 3D benchmark [4] containing 7481 training and 7518 test frames. There are eight object classes: car, van, truck, tram, cyclist, pedestrian, person sitting, and misc, evaluated in three categories: easy, moderate, and hard. These categories are assigned based on the object's pixel height, occlusion, and truncation level. The moderate category is used for the ranking of the benchmark. Since the ground truth labels are not available for the test frames, we split the KITTI training set into train, validation, and test sets containing 6347, 423, and 711 frames, respectively. Furthermore, given that we fuse the point cloud and image, we extract only those points from the point cloud within the camera's field of view. We use the provided synced and rectified images, resulting in a change of the resolution from 1392 $\times$ 512 pixels to 1242 $\times$ 375 $\pm$ 5 pixels.

B. Model Setup

For the training of our model, we have multiple hyperparameters, which can be seen in Table I.

Given that we target an embedded system with limited memory, we target a low memory requirement of our point cloud encoding. For the Pillar Feature Net, the pillar size is set to $0.2m \times 0.2m$, the max pillar limit is declared as $P = 6000$, and the max points per pillar is $N = 50$. Based on the distribution of the points in KITTI, the grid is extended to $80m$ forward and $52.8m$ to either side. A pillar size of $0.2m \times 0.2m$ results in a grid size of 400×528. For the subsequently learned encoding of each pillar, we follow PointPillars [11] and use a 64 channel learning. Thus, the output dimensions of Pillar Feature Net is $400 \times 528 \times 64$.

To get the image and point pillar to match up, the image is scaled to 400×1056 pixels. This scaling decreases the width by $\sim$242 pixels and increases the height by $\sim$25 pixels, introducing some distortion. The prediction head is grouped into multiple task-specific prediction heads, following the setup from CenterPoint [31]. We divided the eight classes in KITTI [4] into four tasks based on the similarity of the object types:

1) Car, Van, Truck, and Tram
2) Cyclist
3) Pedestrian and Person sitting
4) Misc

C. Training Setup

The training is performed with a batch size of 4, using the AdamW [19] optimizer with an one-cycle learning rate policy [7]. The one-cycle learning rate policy provides results faster while acting as a regularization method to keep the model from overfitting. For the AdamW, we have experimented with initial learning rates from 10^{-2} to 10^{-6} in factor 10 decrements, with the weight decay kept a factor 10 larger than the learning rate.

Each task's ground truth heat map is generated by projecting the object centers into a 400×528 grid to match the pillar grid. To counteract the high background to foreground ratio, we increased the positive supervision by enlarging the Gaussian peak at the object center, as proposed by [31]. The Gaussian radius is set as $\omega = max(f(wl), \tau)$, where $\tau = 2$ is the smallest Gaussian radius, and f is the radius function defined by CornerNet [12]. Given the radius, the amount of penalty reduction is the unnormalized 2D Gaussian, $e^{-\frac{x^2+y^2}{2\sigma^2}}$, with $\sigma = \frac{1}{3}\omega$. The center of the Gaussian represents the positive location. However, even with the increased positive supervision, the heat map is still very sparse with information on object locations.

D. Loss Functions

Training of our model includes two different loss functions. The first loss function is for the heat map, and the second one is for the regression of the predicted bounding boxes. When summing up the losses to get the total loss, the regression loss is scaled by a factor of 2:

$$L_{total} = L_{HeatMap} + 2 \cdot L_{Regression}. \quad (1)$$

The effect is that initially, the loss of the heat map is larger and will be the main contributor to the loss, but as the heat map loss subsides, the regression loss will overtake, and the correct prediction of bounding boxes will be more important.

1) Heat Map Loss

We have tested three different loss functions for the heat map. The first loss function is Mean Squared Error (MSE). MSE is an

acceptable baseline but has problems dealing with sparse information. MSE weighs positive and negative loss equally, making it sub-optimal in distinguishing the foreground and background in regions of interest.

The second loss function is a variant of focal loss as proposed by CornerNet [12]. The loss function is as follows:

$$L_F = \frac{-1}{N} \sum_{c=1}^{C} \sum_{i=1}^{H} \sum_{j=1}^{W} \begin{cases} (1-p_{cij})^{\alpha} \log(p_{cij}) & \text{if } y_{cij} = 1 \\ (1-y_{cij})^{\beta} (p_{cij})^{\alpha} \log(1-p_{cij}) & \text{otherwise,} \end{cases} \tag{2}$$

where N is the number of objects, and α and β are the hyperparameters that control the contribution of each point. We set α to 2 and β to 4 following CornerNet. With the Gaussian bumps encoded in y_{cij}, the $(1 - y_{cij})$ term reduces the penalty around the ground truth locations.

The last loss function tested is Binary Cross-Entropy (BCE) given as:

$$L_{BCE} = \frac{-1}{N} \sum_{i=0}^{N} y_i \cdot \log(p(y_i)) + (1-y_i) \cdot \log(1-p(y_i)), \tag{3}$$

where y is the label[2] and $p(y)$ is the predicted probability of the point being positive for all N points. The downside of BCE is that it does not use the Gaussian noise in the generated heat map. Given that a single point is too small to learn, we include the Gaussian noise as a positive label.

2) Bounding Box Loss

For training the bounding box prediction, we combine the predicted properties of the regression head to provide the complete 3D bounding box state information. At training, we collect the state information only at the ground truth centers. The prediction at these locations is compared with the truth using an L1 regression loss for each property. The results are summed up, meaning we weigh the different properties equally.

[2]The label is set to 1 for positive locations and 0 for negative locations.

Method	Seconds
End-to-end	0.13
Pillar Feature Net	0.04
Image Encoding	0.05
Backbone + Prediction Head	0.04

Table II
TIME REQUIREMENTS OF DIFFERENT PARTS OF THE MODEL

VI. Model Inference Time

The time requirements of our model are presented in Table II. The total inference requirement of our model is $0.13s$, which translates to $\sim 7.69FPS$. This FPS count is not high, but considering the computational specification of the Jetson Nano, it is decent. When breaking down the time requirements for the different parts of our model, we see that the time required for Pillar Feature Net, Image Encoding, and Backbone + Prediction Head differ at most $0.01s$ from each other. Given the difference in the size of the tasks, it could indicate there can be some performance to gain in optimization of Pillar Feature Net and image encoding. Pillar Feature Net is written in C++ code. Converted to GPU executable code, some speed up in terms of execution time and pipelining is expected on the Jetson Nano. Likewise, for the image encoding, we see it requires the longest time. The time requirement is primarily caused by the TensorFlow implementation of resizing of the image. Thus, with more efficient implementation, speed up is also possible at this part.

VII. Evaluation

This section contains the evaluation of the experiments and a discussion of the various problems that occurred. We present potential solutions to these problems and tests to validate our assumptions of the problems.

A. Model Stability

We experienced problems with the stability of the learning. When back-propagating the loss through the model, the gradients became too large, and a gradient explosion occurred. The gradient explosion seems to occur due to a combination of the learning rate and large loss values. To rectify the problem, we experimented with different learning rates between 10^{-2} and 10^{-6}. We found that a learning rate of $10^{-}5$ allowed for better stability without gradient explosion.

1) Heat Map Loss

To further improve stability, we tested multiple loss functions for the heat map. We discovered that MSE decreased the stability of the model. The stability problem is caused by MSE's lack of distinction between foreground and background. This causes the sparse heat map prediction to end up with a large loss, causing a gradient overflow when back-propagating or causing the model not to learn.

We found that the focal loss is susceptible to small changes in the predicted heat map, resulting in substantial loss values. The significant loss values overpowered the model, making the model overcompensate and ending in an irreversible state. To counteract this, we substituted the sums with the mean. The change yielded more reasonable loss values, which are less likely to generate a gradient explosion.

Using BCE, no problem with too large values occurred. However, it still has the downside of it not distinguishing between the center point and Gaussian noise as described earlier.

2) Upsampling Design

Why the model is susceptible to gradient explosion can be explained by our upsampling approach. In our learned up 2x block, we use a nearest-neighbor interpolation. The nearest neighbor algorithm selects the value of the nearest point and does not consider the values of neighboring points at all, yielding a piecewise-constant interpolation. Thus, compared to more traditional deconvolution where surrounding pixels influence some scaling, we have a single-pixel responsible for the full upsampling. Thus, this pixel is also responsible for all of the gradients in the following calculations. This might not be a problem until we consider that we are working with very sparse data. Therefore, multiple upscaled pixels are more likely to rely on a single-pixel for upscaling.

The reason we use this upscaling method was that [25] shows it to be fast on embedded systems and provide cleaner outputs. However, [25] is research from within the field of semantic segmentation, which focuses on dense image prediction. The expected prediction output in this paper is sparse, for which the suitability of the method is uncertain.

B. Fusion of Image and Point Pillars

While training, we discovered multiple problems with the fusion of the image and pillar representation. In this subsection, we discuss possible causes and solutions for this problem.

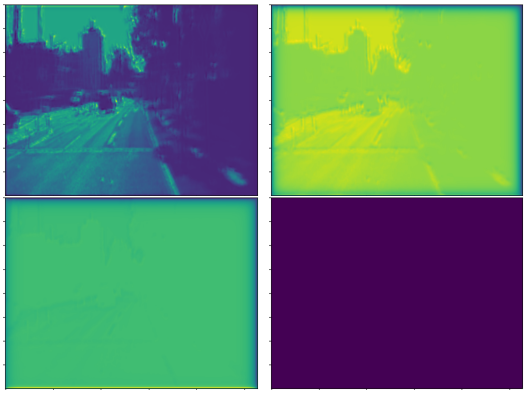

Figure 2. Heat map output from the model. The images are in order top-left, top-right, bottom-left, and bottom-right with 5000 training steps between each image.

When looking at the model's output, clear indications of the image overpowering the output are present. Figure 2 shows the heat map at different steps of training. The images clearly show that the input image is responsible for nearly all influence on the output. The learning of the model requires much training to reduce the influence from the image, thereby resulting in the model predicting blank heat maps. The RGB values of the images are normalized from $[0, 255]$ to $[0, 1]$, which ensures that the model focuses less on the values and makes them correspond better to the value range from PointPillars.

1) Perspective Misalignment

Our model is not able to converge to any meaningful heat map prediction. A probable reason is the inconsistency of the perspective between the image and the point pillars. The pillars are created based on groupings in the x-y coordinates, the encoding of which provides a learned pseudo image representation. Thus, the pseudo image is a BEV of the environment in front of the origin. In contrast, the input image is taking in a front perspective view. Thus, the perspectives of the two inputs differ from each other. The fusion of the image and the pseudo image is impossible unless the model learns a shared spatial representation of the inputs. Without the shared representation, one input will likely act as noise for the other and push the learning in the wrong direction.

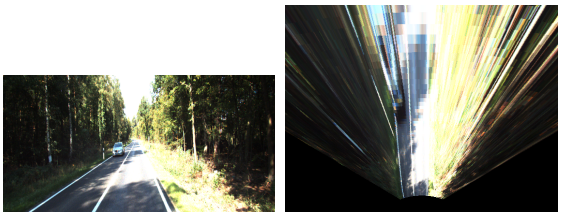

Figure 3. Example of a front-facing image and the image after applying IPM to obtain BEV perspective

To achieve a more unified representation, we warped the input image with Inverse Perspective Mapping (IPM) [20] to be closer to the BEV of the pseudo image. The downside of IPM is the homography warping, leading to unnatural blurring and stretching of objects at further distances. An example of this effect can be seen in Figure 3. Thus, IPM does not work well at distances, but it might still allow for a better fusion between image and point pillars.

Given that IPM is modifying the images, we made some minor changes to our framework to comply with the changes and better fit the new images. We removed the original resizing of the images, as IPM is warping the images and changes the perspective by stretching the pixels. Instead, we resized before and after IPM, as this provides a more accurate perspective. The output of IPM is set to have the same size as our pillar grid. The change in size requires an adjustment

of the first convolution of the image branch of the input-encoder. We changed the asymmetric $(height, width)$ stride of $(1,\ 2)$ to $(1,\ 1)$, as we no longer need to reduce the width to match the grid size.

Unfortunately, the use of IPM did not solve the problem of the model converging to a homogeneous heat map output of the model.

VIII. Analysis

In order to confirm whether the fusion causes the resulting homogeneity of the heat map, we then excluded the image entirely and removed the corresponding connection to the first fusion, using only the point cloud as our input.

We confirmed that the model is still not able to learn even with only point pillar input. In Figure 4, the heat map prediction is seen at different training steps. The prediction of at training step 20000 seems to follow the same pattern as with both inputs. At step $40000 - 60000$, we see a difference in the output, with the frustum of the point pillar showing up in the heat map. The clear frustum indicates the model has impacted the values very little in the passthrough. After step $\sim$60000, the output returns to the homogeneity of the output, and it stays like this with only a tiny variation in intensity over time. The persistence of the homogeneous output indicates that our proposed model cannot learn a general representation that can predict a sparse heat map.

An often-used method for object recognition is a Feature Pyramid Network (FPN) [16]. This FPN outputs feature maps at multiple resolutions and performs the prediction individually from all of them. This way, it attempts to predict objects at different scales. This process is very memory intensive, thus not suited for our goal to design a model for embedded systems. Instead, we use a ResNet34 with NBt1D, which is more efficient but might not be sufficient to retain information of the small surface area of objects captured in the point cloud.

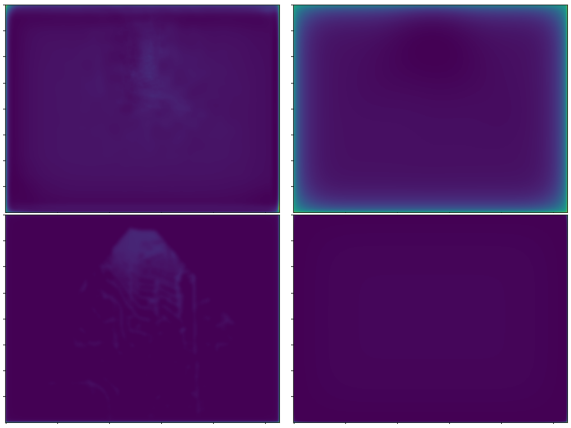

Figure 4. Heat map output from the model at different training steps. Top-left is step 2000, top-right step 20000, bottom-left 48000, and bottom-right step 158000.

A. Alternative Prediction Approach

While investigating why our model predicts homogeneous outputs for the heat map, we found a discussion from Tianwei Yin[3], author of the CenterPoint paper [31]. The discussion describes that PointPillars, for center prediction on the KITTI data set, is so bad that he will not provide any configuration for PointPillars encoding.

To test whether our proposed backbone is suitable for encoding, we replaced the prediction head with a more established bounding box prediction head, the Single Shot Detector (SSD) [18]. For the codebase, we follow the TensorFlow implementation

[3]https://github.com/tianweiy/CenterPoint-KITTI/issues/1

Variable	Original	Changed
focal weight	1.0	3.0
location weight	2.0	2.0
size weight	2.0	2.0
angle weight	2.0	1.0
heading weight	0.2	0.2
class weight	0.2	0.5

Table III
Loss weightings for the alternative setup

by Anjul Tyagi[4]. This follows the PointPillars paper setup [11], but changes the loss weightings to better match with more data sets. The original and changed weights can be seen in Table III.

We trained our model for 100 epochs, with the result being that the model was not able to predict correct bounding boxes.

The occupancy threshold used for the selection of predicted bounding boxes posed a problem in our model. By default, the threshold was set to 0.7, which we lowered to 0.4 since the model was not able to assign higher confidence to any prediction. At inference time, we apply axis-aligned NMS with an overlap threshold of 0.5 given Intersection over Union. With the filtered bounding boxes, we still got an accuracy of zero. The main problem with these boxes was the location prediction. This might indicate that our backbone is poor at retaining spatial information about objects in the point cloud. Given that PointPillars trained for 160 epochs in their paper, and our backbone is larger, we might have trained for too few epochs to achieve a reliable result. However, given time constraints, we have limited the learning time to 100 epochs.

We have designed our upsampling based on efficient proposals from research in semantic segmentation. However, there is an inconsistency between predictions in the two research fields. In semantic segmentation, the goal is to make a dense prediction of the whole image with a class for every pixel. This is in contrast to 3D detection, where we work on sparse data and sparse prediction. Thus, our model might try to make dense predictions in a sparse context. If the model is predicting too densely, it will provide itself with a lot of error and not learn.

In our backbone, we utilize max pooling in the beginning to increase the density of the input. However, max pooling is known to decrease spatial retainment. Thus, if we remove or substitute the max pooling, we might get some improvement.

Another place that might decrease spatial retainment is our introduction of fusion. In the fusion, we have proposed a mechanism for creating an attention map. The mechanism first makes a depth-wise convolution and then a 1×1 convolution with 1 channel output. The reduction in channels provides a considerable compression, especially in the deeper part of the encoder, with the last fusion compressing 512 channels down to 1. The attention map might be inadequate in providing an appropriate scaling of the information with that number of channels, thereby suppressing essential information in some intermediate channels.

[4]https://github.com/tyagi-iiitv/PointPillars

IX. Conclusion

In this paper, we propose a model for 3D bounding box prediction. The model has a two-stream input, the first using standard Pillar Feature Net for encoding a point cloud and the other for processing an image. For the backbone of the model, we use a ResNet34 architecture utilizing the Non-Bottleneck-1D-Block (NBt1D) blocks for each stream. We introduce a novel fusion with an attention mechanism to combine information of the two streams. We experiment with two prediction heads for the prediction of the bounding boxes: a center point-based and a single-shot detection approach. We test our model's inference time on a Jetson Nano. Here we show that the model reaches near real-time performance.

We train the model on the KITTI 3D detection benchmark and have tested and discussed different components in the model. We have confirmed that the backbone is suboptimal at retaining spatial information in the single-shot detection approach and produces homogeneous outputs for the center point-based approach. This gives us reasonable safety to assume that the model is not fit for the task.

X. Future Work

Tianwei Yin commented[5] that the point pillar encoding does not work well with the KITTI data set. Thus, further tests of our approach on a different data set are needed. This way, it can be confirmed whether the problem lies within the backbone or is specific to the KITTI 3D benchmark.

In this paper, we have transformed the images into Bird's-Eye View (BEV) with the use of the classical method Inverse Perspective Mapping (IPM), which causes unwanted stretching and distortion of the image. Using a camera configured or learned transformation method, which is less prone to these side effects, can provide more realistic BEV in terms of object sizes and shapes. The improved BEV should yield a closer one-to-one mapping with point pillars improving the accuracy and feasibility of the fusion.

[5] https://github.com/tianweiy/CenterPoint-KITTI/issues/1

References

[1] Zhuangzhuang Ding et al. "1st Place Solution for Waymo Open Dataset Challenge - 3D Detection and Domain Adaptation". In: *CoRR* abs/2006.15505 (2020). arXiv: 2006.15505. URL: https://arxiv.org/abs/2006.15505 (cit. on p. 3).

[2] Kaiwen Duan et al. *CenterNet: Keypoint Triplets for Object Detection*. 2019. arXiv: 1904.08189 [cs.CV] (cit. on p. 1).

[3] Runzhou Ge et al. *AFDet: Anchor Free One Stage 3D Object Detection*. 2020. arXiv: 2006.12671 [cs.CV] (cit. on p. 1).

[4] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074 (cit. on pp. 2, 5, 8, 9).

[5] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083 (cit. on pp. 2, 3).

[6] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524 (cit. on pp. 2, 3).

[7] Sylvain Gugger. *The 1cycle policy*. Apr. 2018. URL: https://sgugger.github.io/the-1cycle-policy.html (cit. on p. 9).

[8] Chenhang He et al. "Structure Aware Single-Stage 3D Object Detection From Point Cloud". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 11870–11879. DOI: 10.1109/CVPR42600.2020.01189 (cit. on p. 2).

[9] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on p. 6).

[10] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870 (cit. on p. 2).

[11] Alex H. Lang et al. "PointPillars: Fast Encoders for Object Detection from Point Clouds". In: *CoRR* abs/1812.05784 (2018). arXiv: 1812.05784. URL: http://arxiv.org/abs/1812.05784 (cit. on pp. 2, 4–6, 9, 14).

[12] Hei Law and Jia Deng. "CornerNet: Detecting Objects as Paired Keypoints". In: *Int. J. Comput. Vis.* 128.3 (2020), pp. 642–656. DOI: 10.1007/s11263-019-01204-1. URL: https://doi.org/10.1007/s11263-019-01204-1 (cit. on pp. 3, 9, 10).

[13] Hao Li et al. "Enhanced YOLO v3 Tiny Network for Real-Time Ship Detection From Visual Image". In: *IEEE Access* 9 (2021), pp. 16692–16706. DOI: 10.1109/ACCESS.2021.3053956. URL: https://doi.org/10.1109/ACCESS.2021.3053956 (cit. on p. 3).

[14] Ming Liang et al. "Deep Continuous Fusion for Multi-Sensor 3D Object Detection". In: *CoRR* abs/2012.10992 (2020). arXiv: 2012.10992. URL: https://arxiv.org/abs/2012.10992 (cit. on p. 4).

[15] Ming Liang et al. "Multi-Task Multi-Sensor Fusion for 3D Object Detection". In: *CoRR* abs/2012.12397 (2020). arXiv: 2012.12397. URL: https://arxiv.org/abs/2012.12397 (cit. on p. 4).

[16] Tsung-Yi Lin et al. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV] (cit. on p. 13).

[17] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *CoRR* abs/1708.02002 (2017). arXiv: 1708.02002. URL: http://arxiv.org/abs/1708.02002 (cit. on pp. 2, 3).

[18] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325 (cit. on pp. 2, 3, 13).

[19] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101 (cit. on p. 9).

[20] Hanspeter Mallot et al. "Inverse Perspective Mapping Simplifies Optical Flow Computation and Obstacle Detection". In: *Biological cybernetics* 64 (Feb. 1991), pp. 177–85. DOI: 10.1007/BF00201978 (cit. on p. 12).

[21] Charles Ruizhongtai Qi et al. "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 (cit. on pp. 3, 5).

[22] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242 (cit. on pp. 2, 3).

[23] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497 (cit. on pp. 2, 3).

[24] Eduardo Romera et al. "ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation". In: *IEEE Transactions on Intelligent Transportation Systems* 19.1 (2018), pp. 263–272. DOI: 10.1109/TITS.2017.2750080 (cit. on pp. 2, 6).

[25] Daniel Seichter et al. "Efficient RGB-D Semantic Segmentation for Indoor Scene Analysis". In: *CoRR* abs/2011.06961 (2020). arXiv: 2011.06961. URL: https://arxiv.org/abs/2011.06961 (cit. on pp. 2, 6, 11).

[26] 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 (cit. on p. 2).

[27] Martin Simon et al. "Complex-YOLO: Real-time 3D Object Detection on Point Clouds". In: *CoRR* abs/1803.06199 (2018). arXiv: 1803.06199. URL: http://arxiv.org/abs/1803.06199 (cit. on p. 2).

[28] Jasper R. R. Uijlings et al. "Selective Search for Object Recognition". In: *Int. J. Comput. Vis.* 104.2 (2013), pp. 154–171. DOI: 10.1007/s11263-013-0620-5. URL: https://doi.org/10.1007/s11263-013-0620-5 (cit. on p. 3).

[29] Yan Yan, Yuxing Mao, and Bo Li. "SECOND: Sparsely Embedded Convolutional Detection". In: *Sensors* 18.10 (2018). ISSN: 1424-8220. DOI: 10.3390/s18103337. URL: https://www.mdpi.com/1424-8220/18/10/3337 (cit. on p. 2).

[30] Zetong Yang et al. "3DSSD: Point-based 3D Single Stage Object Detector". In: *CoRR* abs/2002.10187 (2020). arXiv: 2002.10187. URL: https://arxiv.org/abs/2002.10187 (cit. on p. 2).

[31] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. "Center-based 3D Object Detection and Tracking". In: *CoRR* abs/2006.11275 (2020). arXiv: 2006.11275. URL: https://arxiv.org/abs/2006.11275 (cit. on pp. 1–5, 8, 9, 13).

[32] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. "Objects as Points". In: *CoRR* abs/1904.07850 (2019). arXiv: 1904.07850. URL: http://arxiv.org/abs/1904.07850 (cit. on pp. 3, 4).

[33] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. "Bottom-Up Object Detection by Grouping Extreme and Center Points". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 850–859. DOI: 10.1109/CVPR.2019.00094. URL: http://openaccess.thecvf.com/content%5C_CVPR%5C_2019/html/Zhou%5C_Bottom-Up%5C_Object%5C_Detection%5C_by%5C_Grouping%5C_Extreme%5C_and%5C_Center%5C_Points%5C_CVPR%5C_2019%5C_paper.html (cit. on p. 3).

[34] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: *CoRR* abs/1711.06396 (2017). arXiv: 1711.06396. URL: http://arxiv.org/abs/1711.06396 (cit. on p. 2).

[35] Ming Zhu et al. "Cross-Modality 3D Object Detection". In: *CoRR* abs/2008.10436 (2020). arXiv: 2008.10436. URL: https://arxiv.org/abs/2008.10436 (cit. on p. 4).