TinyPillarNet: Tiny Pillar-Based Network for 3D Point Cloud Object Detection at Edge

Yishi Li, Yuhao Zhang, and Rui Lai, *Member, IEEE*

***Abstract*— Limited by huge computational cost, high inference latency and large memory consumption, existing 3D point cloud object detection methods are hard to be deployed on Internet of Things (IoT) edge devices. To handle this challenge, we present an extremely tiny framework termed TinyPillarNet. This framework leverages innovative pillar encoder to represent point cloud as immensely tiny pseudo-maps for extremely shrinking the input 3D sensing data. Moreover, a compact dual-stream feature extraction network is put forward to respectively extract intrinsic feature and distributional saliency map, which jointly boosts the detection precision with the lowest hardware cost. Extended experiments on KITTI benchmark demonstrated that our TinyPillarNet yields applicable precision with a record tiny weight size of 1.69 MB at a high inference speed of 1.67 times faster than the current record. Furthermore, the specially designed prototype verification system achieves a superior energy efficiency, which outperforms the similar deep learning based point cloud processing solutions on FPGA with a big margin.**

***Index Terms*— 3D object detection, point cloud, tiny machine learning (TinyML), FPGA.**

I. Introduction

OBJECT detection in real-world 3D scene is a fundamental component of perception systems widely used in autonomous driving, robotics, remote sensing and medical treatment [1], [2]. LiDAR directly provides reliable depth information and rich geometric, shape and scale information in outputting point cloud, which is critical to accurately locate and recognize objects [3], [4]. Different from 2D images, 3D point cloud data is characterized by irregular and sparse distribution, which presents great challenges to transmit and process such massive unstructured data in feature representation with strict hardware resource limitation.

With the expanding application of Internet of Things (IoT), there is an increasing demand for implementing computationally complex deep learning models on resource constrained edge devices. Generally, it is required to process data near sensors with only hundreds of milliwatts of power consumption and several megabytes of memory. To address this challenge, tiny machine learning (TinyML) attracts growing attention. Increasing research focuses on lightweight networks [5], [6], [7], [8] and hardware-efficient deployment solution on edge computing platforms [9], [10], [11], [12], [13], [14], [15], [16], [17].

To implement precise 3D object detection, representative works were successively presented [18], [19], [20], [21], most of which came at the expense of huge memory consumption as shown in Fig.1.

Early point-based methods [22], [23] (marked yellow in Fig.1) introduce end-to-end deep neural networks [24], [25] to directly learn point-wise features from a large-scale unordered point cloud. Such methods achieve high precision with tiny input size. However, the point-wise computation mode inevitably results in unaffordable computational cost [2], [26]. Given this, bird's-eye-view (BEV) methods [27], [28], [29], [30] (marked green in Fig.1) convert point cloud to small and low-dimensional BEV representation by projecting points to X-Y plane and discarding the height information. However, the projection results in the considerable loss of 3D structure information, which makes the detection precision sharply drop. To balance the inference efficiency and precision, voxel-based methods [31], [32] organize the point cloud in a set of voxels. Then, a PointNet extractor and a CNN backbone are applied to predict object locations. However, representation with dense voxels sharply increases the input size to over 750MB. In view of this, PointPillars [33] introduces a pillar encoder to further reduce the input size. Even though, the excessive storage consumption of current models remains the fatal bottleneck for the application on edge devices.

Aiming at deploying on IoT devices, the total memory consumption of input and weight has to be reduced to within several megabytes. To this end, we propose an extremely tiny framework termed TinyPillarNet for 3D object detection from point cloud. As the structure shown in Fig.2, an innovative pillar pseudo-map encoder (PPME) is proposed to solve the great difficulty of oversize input data. PPME encodes point cloud to 2D intrinsic and distributional pseudo-maps, which is convenient for 2D CNN processing and leverages 3D information efficiently. Then, a dual-stream feature extraction network is proposed to further boost the precision, which consists of a tiny backbone network (TBN) and a saliency enhancement

Manuscript received 23 February 2023; revised 14 June 2023; accepted 16 July 2023. Date of publication 21 July 2023; date of current version 7 March 2024. This work was supported in part by the National Science and Technology Innovation 2030-Major Projects under Grant 2021ZD0114400, in part by the National Key Research and Development Program of China under Grant 2018YFE0202800, and in part by the Proof of Concept Foundation of Xidian University Hangzhou Institute of Technology under Grant GNYZ2023JC0402. This article was recommended by Associate Editor L. Agostini. *(Corresponding author: Rui Lai.)*

The authors are with the School of Microelectronics, Xidian University, Xi'an 710071, China, and also with the Chongqing Innovation Research Institute of Integrated Circuits, Xidian University, Chongqing 400031, China (e-mail: yshlee1994@outlook.com; stuyuh@163.com; rlai@mail.xidian.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSVT.2023.3297620.

Digital Object Identifier 10.1109/TCSVT.2023.3297620

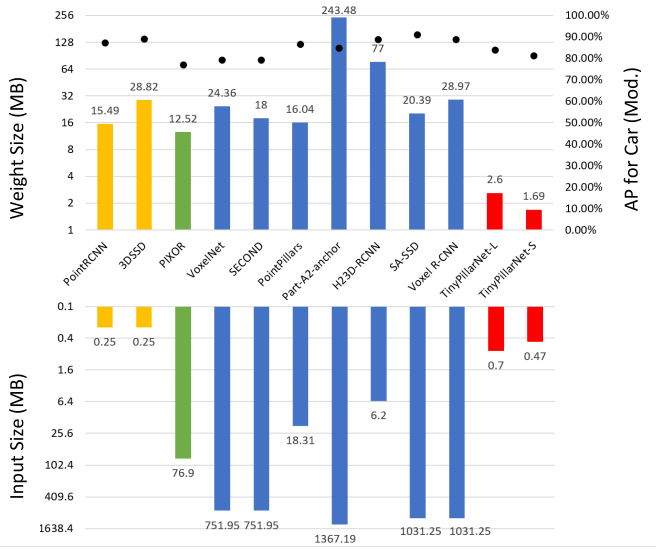

Fig. 1. The comparison of our proposed TinyPillarNet with existing methods in terms of memory consumption (weight size and input size) and AP for *Car* in KITTI test split.

network (SEN). As for TBN, it constitutes with our proposed memory-efficient linear residual blocks and extracts intrinsic features containing all 3D shape information of objects. SEN generates a distributional saliency map, which is multiplied to intrinsic features to fuse the location information for further promoting the detection precision.

The extensive performance assessment on KITTI [34] benchmark indicated that our proposed TinyPillarNet (marked red in Fig.1) maintains applicable precision with record lowest weight size of 1.69MB and input size of 0.47MB. Moreover, the specially designed prototype system demonstrated that our proposed TinyPillarNet can be implemented on memory constrained FPGA with high peak performance and power efficiency on the task of real-time 3D object detection.

The major contributions of this paper are summarized as follows:

- We propose a pillar pseudo-map encoder to represent raw points as 2D intrinsic and distributional pseudo-maps, which sharply prunes over 88.7% of input size of existing voxel-based models.
- We propose a memory-efficient linear residual block and further construct the tiny backbone network with this block to extract intrinsic features of objects, which remarkably boosts the detection precision under the extreme limitation of weight size.
- We present a distribution-aware saliency enhancement network that extracts the distributional saliency map containing the location information of objects, which is then polymerized with the intrinsic features to promote the detection precision.
- We specially design an FPGA-based prototype system on Xilinx ZC706 platform for implementing the proposed TinyPillarNet in real time. Extensive experiments indicated that the system achieves outstanding record of 304.31GOPS peak performance and 83.83GOPS/W power efficiency in practical point cloud object detection.

II. Related Works

A. 3D Point Cloud Object Detection

Existing 3D object detection methods with LiDAR point cloud mainly fall into three categories: point-based, bird's-eye-view (BEV) based and voxel-based methods.

Point-based methods take raw points as input, and generate predictions based on 3D point-wise features. Compared to early works that only use 2D features, F-PointNet [35] and PointFusion [36] firstly fuse 3D point-wise features with 2D features to generate bounding boxes more precisely. However, the performance of such methods relies heavily on 2D prediction and cannot take full advantage of 3D information. Given this, PointRCNN [22] and STD [37] employ PointNet++ [25] as backbone for generating point-wise features. PointRCNN solves the ambiguous proposals of point groups by utilizing point-wise features. STD converts features from sparse to dense representation, and directly employs CNNs in refinement stage for saving computation. While achieving higher accuracy, previous methods ignore the practicality of inference speed. Considering feature propagation (FP) layers and refinement stage consume most computation, 3DSSD [23] respectively proposes a feature distance-based sampling strategy to replace FP layers, and designs a more effective box prediction network. As a result, the inference time is reduced to 38ms on powerful GPU. In summary, point-based methods achieve high precision at the expense of unaffordable inference latency and hardware cost.

Unlike point-based methods, BEV-based methods transform raw points into a structured 2D projective representation, which means they can utilize the mature CNN designs in given hardware environments. BirdNet [27] and PIXOR [29] are classical dense object detectors that exploit efficient data representation and frameworks with specific coding methods. BirdNet encodes height, intensity and density information in one image, while PIXOR encodes coordinates in the first two dimensions and keeps height information as channels along the third dimension, forming a 3D data cube. Although these methods continuously strive to improve the precision, there is still a big precision gap between such methods and precious point-based works. For further promoting the precision, HDNET [30] specially designs a complex detector to fuse the BEV representation from LiDAR with the high-definition maps from an independent map predictor. However, the increase in the complexity makes it lose the most important advantage of lightweight structure.

Voxel, as another category of structured representation, has been widely used in all 3D tasks. Voxel-based methods generally discretize 3D space into regular voxels and apply 3D/2D convolutions to extract features. To address the huge computation in representing the massive voxel feature, Vote3D [38] proposed a voting scheme exercised only by the non-empty voxels. To further exploit 3D shape and invariances, VoxelNet [31] presented voxel feature coding (VFE) instead of manual design to extract point-wise features, and then aggregates them with following 3D convolutions. However, the computational cost for 3D convolutions hinders their expansion in real-time applications. For achieving higher precision,

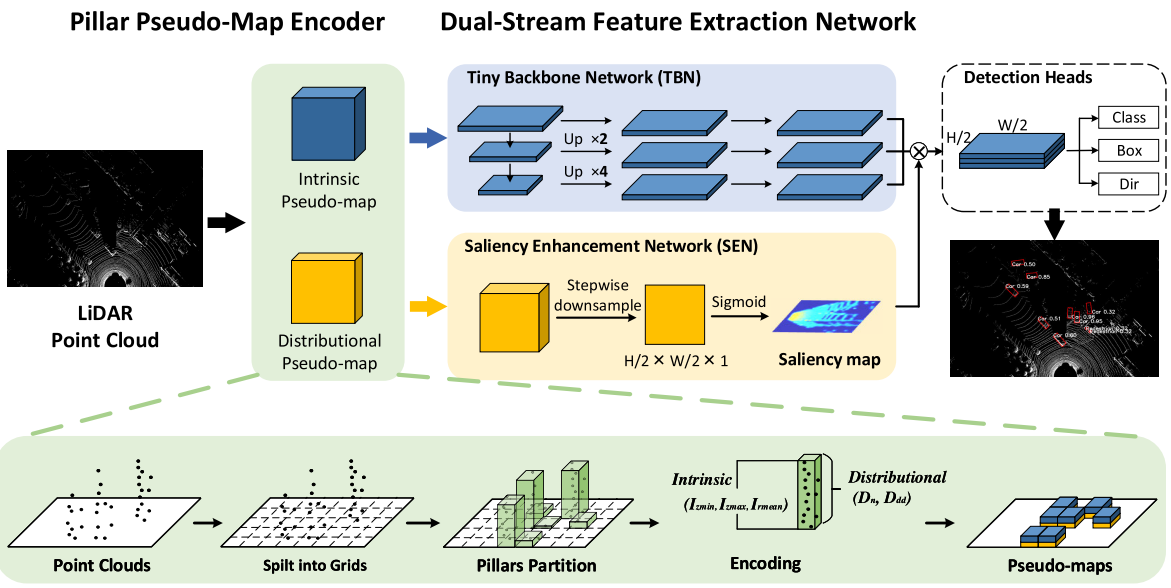

Fig. 2. The structure of the proposed TinyPillarNet framework. The Pillar Pseudo-Map Encoder encodes the point cloud to an intrinsic pseudo-map and a distributional pseudo-map. Then, the Dual-Stream Feature Extraction Network predicts intrinsic feature and distributional saliency map. The intrinsic feature is multiplied by the saliency map, and is processed by detection heads, getting bounding boxes.

Part-A^2 [39] introduced an RoI-aware pooling operation to preserve information of all points within the proposals to eliminate the ambiguity. For the same purpose, $H^2$3D-RCNN [40] extracts 3D voxel features from both perspective view and bird-eye view and then detects objects with a novel box refinement module. However, Part-A^2 and $H^2$3D-RCNN achieves comparative high precision to point-based methods at the cost of higher memory consumption and computational load with more complex computation workflow. Given this, low supervised methods [41], [42], [43], [44] aim at reducing the requirement for large quantities of 3D supervision.

To further boost the speed, SECOND [32] proposes a novel sparse convolution to directly extract information from unsampled 3D data, which reduces the inference latency to 50ms without much precision loss. PointPillars [33] introduces an innovative point cloud encoder to generate 2D pseudo-map, facilitating the feature processing with 2D convolutions. As a result, it reduces the inference latency to 23ms with memory consumption only of 16.04MB for weight parameters and 18.31MB for input data. Even so, the memory cost of existing methods far exceeds the limitation of TinyML applications. In view of this, we will extremely shrink the storage requirements of the LiDAR object detection network.

B. Tiny Machine Learning

TinyML is an up-and-coming concept which deals with executing optimized ML models on ultra-low-power MCUs with minimal power consumption [45]. Since TinyML requires few hardware resources and is usually low-cost, it is often used as a powerful co-processing unit to enhance the AI computing capabilities of traditional devices. To be deployed on resources limited IoT edge devices, ML algorithms need to be optimized as an extremely lightweight and hardware-efficient framework. Towards this end, several compression techniques are employed, such as architecture design, pruning, and quantization.

For the architectural design, several guidelines are successfully developed after years of validation, such as macro design [5], [46], [47], depthwise separable convolution [5], [6], inverted bottleneck [6], [48], [49], etc. Based on the exploration of redundancy between convolutional channels, network pruning methods [50], [51], [52], [53], [54] are proposed to remove the connectivity among hidden layers that are insensitive to performance. Although the pruning operation decreases computation load and memory cost, it generally brings uncertainty to the network structure that may cause inconvenience to hardware deployment. To further shrink the network and implement ML on IoT devices, quantization [55], [56], [57], [58], [59] generally acts as a necessary step that reduces the bit-width of weight parameters. For instance, the weight of the ML models, which are typically of 32-bit or 64-bit, can be mapped down to 8-bit and executed on general purpose devices. In certain cases, quantization shrinks the network scale of 4 times by trading accuracy around 1-3%.

III. Method

A. Overview

In order to implement 3D object detection on the edge with low hardware cost, we present an extremely tiny point cloud object detection framework with global view in Fig.2. As can be seen, the proposed architecture consists of three main components: (1) pillar pseudo-map encoder (in the green region), (2) dual-stream feature extraction network (in the blue and the yellow region), and (3) detection heads (in the white region).

As for the workflow, 3D space is first clipped into grids. The points are then classified into the subspace of each grid, forming a set of pillars. Thereafter, we convert the pillars to 2D intrinsic and distributional pseudo-maps with our proposed

PPME. Following that, the features of 2D pseudo-maps with different characteristics are further extracted by TBN and SEN, which respectively generate the intrinsic features and the distributional saliency map. At the end, SSD-style detection heads respond to locate and classify the objects using the refined features from dual-stream feature extraction network.

B. Pillar Pseudo-Map Encoder

1) Pillar Partition: PointPillars [33] proves that ignoring the Z-axis of voxels and creating pillars are not harmful to the detection precision. Accordingly, we apply pillar partition on point cloud for data dimensionality reduction firstly.

Let $[x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}]$ be taken as the detected 3D range of point cloud, and $pt_n = (x_n, y_n, z_n, r_n)$ denotes a point with coordinates and reflectance. The pillars partition is the process of clipping the 3D space into grids only along the X and Y axes. Then, the points are classified into the subspace of each grid. The subspace is called a pillar, which has (X,Y) coordinates and a height corresponding to detection range. All the pillars form a set, and a pillar can be defined as

$$P_{i,j} = \{pt_n \mid i = (0, 1, \ldots, \lfloor x_n/g_x \rfloor), \quad j = (0, 1, \ldots, \lfloor y_n/g_y \rfloor)\} \tag{1}$$

where $\lfloor \cdot \rfloor$ is the floor function. (i, j) is the coordinate of the pillar. The grid size is represented as g_x and g_y.

Therefore, the shape of pillar set is $\mathbb{R}^{H \times W \times N_{max} \times 4}$, where W and H are the numbers of pillars in each row and column, N_{max} is a hyper-parameter that controls the maximum number of points in pillars, and 4 represents that each point has 4 attributes (x, y, z and r).

2) Encoding: The pillar set, as a representation of the point cloud proposed by PointPillars, saves much input size by abandoning the splitting along Z-axis. However, the processed input data, being non-empty pillars, still takes about 18.31MB for each frame of KITTI dataset. It's far beyond the on-chip memory space of edge devices.

To find a more streamlined representation of the point cloud, we visualize the raw point cloud and pillars of Car, Pedestrian and Cyclist in Fig.3. As can be seen, the red colored pillar set reserves the prime 3D architectural feature, and the targets can still be located and recognized through pillars instead of raw points. Given this, replacing the raw point cloud with pillar set for characterizing objects is far more memory-efficient.

Accordingly, we propose an innovative PPME to respectively encode pillars to intrinsic pseudo-map and distributional pseudo-map. Since the grid size is fixed, the pillar's height is significant to represent the 3D space occupation of objects. Therefore, I_{zmin} and I_{zmax} are applied to denote the minimum/maximum height of each pillar alone Z axis. Meanwhile, we remove the coordinate of the pillar in X-Y plane, which is implicitly contained by element order of the intrinsic pseudo-map. Considering the reflectance of the same material is similar, we introduce the mean reflectance I_r of each pillar to provide shape-related information for recognizing the object more precisely. Given this, the intrinsic pseudo-map can be defined as

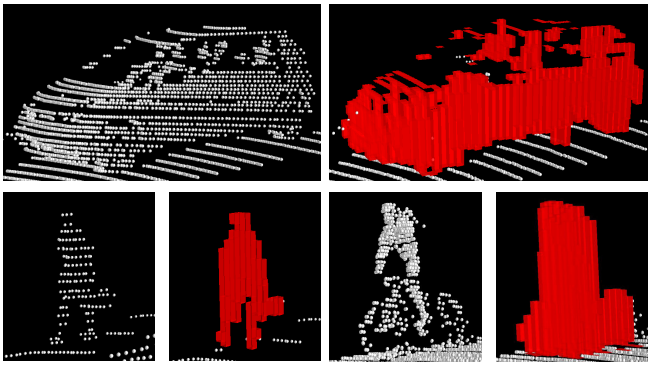

Fig. 3. The point cloud and the red colored visualization of pillar set of Car, Pedestrian and Cyclist.

$$I_{i,j} = \{(I_{zmin})_{i,j}, (I_{zmax})_{i,j}, (I_r)_{i,j}\} \tag{2}$$

where

$$\begin{aligned} (I_{zmin})_{i,j} &= min\{z_n | z_n \in P_{i,j}\} \\ (I_{zmax})_{i,j} &= max\{z_n | z_n \in P_{i,j}\} \\ (I_r)_{i,j} &= mean\{r_n | r_n \in P_{i,j}\} \end{aligned} \tag{3}$$

Since the 3D space is split into pillars, precisely locating the object only with intrinsic information is inadequate. Given this, the distributional pseudo-map $D_{i,j}$ is further proposed to improve the location accuracy, which encodes the spatial distribution information of pillars in 3D space to the density D_n and disorder degree D_{dd}. The element in D is mathematically represented as

$$\begin{aligned} D_{i,j} &= \{(D_n)_{i,j}, (D_{dd})_{i,j}\} \\ (D_n)_{i,j} &= N_{i,j} \\ (D_{dd})_{i,j} &= mean(\sqrt{(x_n - x_{mean})^2 + (y_n - y_{mean})^2}) \end{aligned} \tag{4}$$

where x_{mean} and y_{mean} are the distribution center of points in $P_{i,j}$. $N_{i,j}$ denotes the number of points in $P_{i,j}$. Since D_n and D_{dd} respectively describe the quantitative distribution and geometric distribution, $D_{i,j}$ is obviously introduced to improve the location precision.

It is noted that the pseudo-maps encoded by PPME typically consumes only 720KB of input buffer space and is further low to 480KB for small version in the KITTI scenario, which is much smaller than the buffer requirement of PointPillars. At the same time, the PPME realizes the efficient on-chip memory access in point cloud processing, which also helps to reduce transmission delay. Unlike traditional encoded LiDAR data, the 2D pseudo-maps maintain most of the vital 3D structural information, and can be directly processed by 2D-CNN for saving computation and memory consumption.

3) Analysis: Superficially, PPME uses the similar encoding strategy with RT3D [28]. As for the similarity, both of our PPME and RT3D encode the height information of each pillar into the 2D pseudo-map. As for the difference, PPME additionally encodes the reflectance and distribution information to further promote the representation precision. Specifically, the

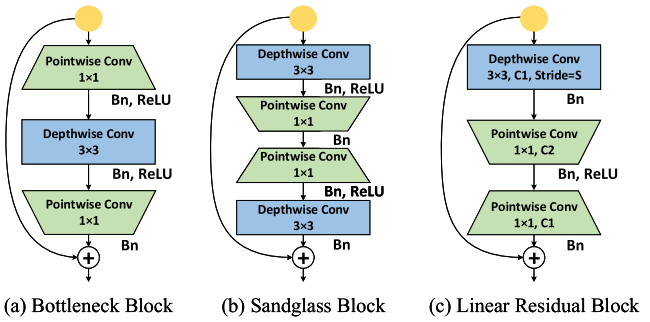

Fig. 4. The comparison of building blocks for different lightweight networks. (a) Bottleneck block in MobileNetV2 [6]. (b) Sandglass block in MobileNext [60]. (c) Proposed Linear Residual block in our TinyPillarNet.

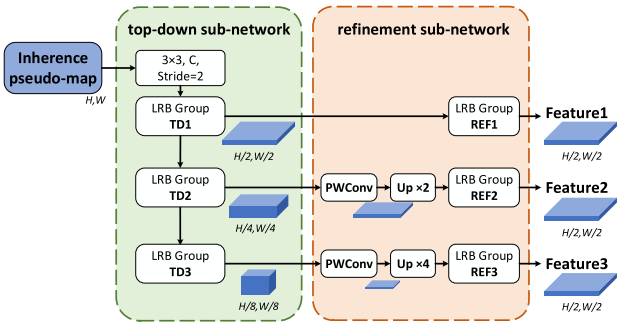

Fig. 5. The structure of our tiny backbone network.

differences lie in the following two aspects: (1) We introduce the reflectance descriptor I_r to reflect the different reflectance of objects and backgrounds, which remarkably improves the detection precision. (2) We introduce the spatial distribution descriptors, D_n and D_{dd}, to provide additional location information for further boosting the detection precision. The effectiveness of our improvements can be verified by the results in Tab.VI and Tab.VII.

C. Tiny Backbone Network

1) Linear Residual Block: In view of building block designs for TinyML, we propose a memory-efficient Linear Residual Block (LRB) to extremely reduce the network size and keep the detection precision. Prior works on lightweight network reveal the following results: (1) The shortcut connection on features with more channels can preserve more information from bottom layers and leads to more gradients propagated across multiple layers in training [60]; (2) Sparse coding methods represent arbitrary signal with linear combination of few dictionary elements. As an analogy, the orthogonal dictionary convolution strategy [61] proposed that convolutions can be linearly combined when there is no non-linear layer between them. As a result, the optimization space of the linear convolution group is hence expanded, and it certainly will bring the performance promotion.

Fig.4 shows the classic bottleneck block, sandglass block and our Linear Residual block. Bottleneck block uses a pointwise convolution (PWConv) to expand channels and extract spatial feature by depthwise convolution (DWConv). It's widely used for lightweight tasks. Then, sandglass block exchanges the order of convolutions and add an extra DWConv. It proves that putting the shortcuts between features with more channels delivers more information from the previous input.

We modify the structure of sandglass block to fit the point cloud processing in two aspects: (1) We remove the last DWConv. As the work [62] points out, the stacking of 3×3 convolutions will cause the sparsity of features to disappear rapidly and further make the shape of objects be blurred, which reduces the 3D detection precision. In view of this, we remove the last DWConv of sandglass block to alleviate this problem. (2) We tune the activation layers. Following the orthogonal dictionary convolution strategy, we remove the ReLU between the first DWConv and its following PWConv to construct a linear convolution group. Moreover, as the MobileNetV2 [6] and MobileNext [60] demonstrated, adding a ReLU layer after the last convolution will reduce the precision. Therefore, we only reserve a ReLU layer after the linear convolution group to introduce necessary non-linearity.

Let $\mathbf{F} \in \mathbb{R}^{C \times H \times W}$ and $\mathbf{G} \in \mathbb{R}^{C \times H \times W}$ respectively represent the input and output feature maps. The LRB can be mathematically formulated as

$$\mathbf{G} = \phi_{p,2}(\sigma(\phi_{p,1}(\phi_d(\mathbf{F})))) + \mathbf{F} \tag{5}$$

where ϕ_d denotes the DWConv. $\phi_{p,1}$, $\phi_{p,2}$ indicates 1st and 2nd PWConvs for channel reduction and expansion, respectively. $C1$, $C2$ and $C3$ correspondingly represent the number of filters for convolutions ϕ_d, $\phi_{p,1}$, $\phi_{p,2}$.

2) Architecture of Tiny Backbone Network: Based on the LRB, we develop a modularized multistage backbone TBN that involves two sub-networks. The top-down sub-network (in the green region of Fig.5) produces features at increasingly small spatial resolution. Accordingly, the feature refinement sub-network (in the orange region of Fig.5) performs up-sampling and refinement for multi-scale features.

Top-down sub-network begins with 3×3 convolution with C filters and stride 2. Following that, LRB groups are stacked. Refinement sub-network applies up-sampling layers to normalize the feature shape. Considering the hardware resource and software environment of existing embedded devices, we utilize the hardware-efficient "nearest" up-sampling to perform the deconvolution operation. Thereafter, LRB groups are arranged to suppress the shape distortion in up-sampling and refine the features for high precision detection.

D. Saliency Enhancement Network

The distribution-aware SEN takes the distributional pseudo-map as input and predicts the single-channel output. Generally, the higher density and entropy of point distribution exists in the target area. Under the supervision of SEN branch for the target location in the training phase will result in strong response around objects in feature map, which highlights the targets and makes the output of SEN work as a saliency map.

The specific network structure of SEN shown in Fig.6 starts with a 3×3 convolutional layer with stride 2, and is followed by numbers of depthwise separable convolutions. The proposed SEN applies a feature compression strategy

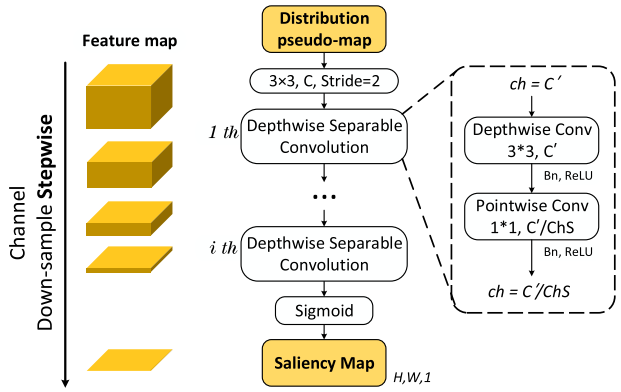

Fig. 6. The structure of our saliency enhancement network.

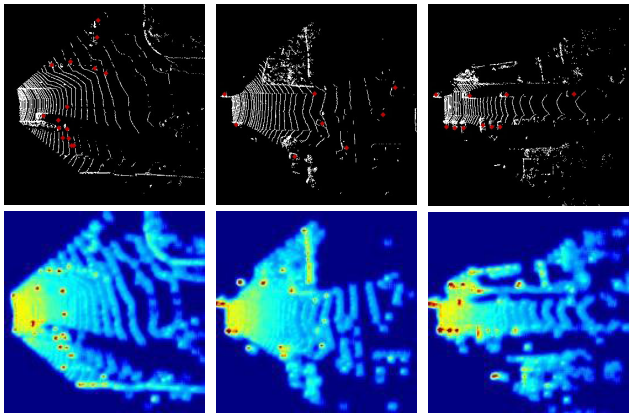

Fig. 7. The frames in KITTI validation split and the corresponding distributional saliency maps produced by SEN.

to gradually decrease the channel of each convolution pair at a rate of $1/ChS$. Finally, the distributional feature is downsampled to 1 channel. $Sigmoid$ is applied at the end for converting the feature to saliency map. As a result, the SEN obtains a larger receptive field and extracts richer distribution information, boosting the detection accuracy.

Qualitatively, BEV frames of point cloud and their corresponding predicted saliency maps are shown in Fig.7. For BEV frames in the top row, the centers of ground truth bounding boxes are marked as red points. For saliency maps in the bottle row, red color marks the region of interest (ROI). It can be easily observed that centers of truth objects approximately correspond to the ROIs in saliency maps. Since the semantics of distributional pseudo-map is clear, the saliency map focuses naturally on the location information without other specific tricks in SEN.

E. Detection Heads and Loss

Referring to 2D object detection works, we introduce three detection heads from Single Shot Detector (SSD) [63] to predict class, box, and direction of objects. Two anchors are defined for two directions 90 degrees apart for each class in every grid. To simplify the network model, we only use one convolution for each head.

We match the prior boxes to the ground truth using 2D Inter-section over Union (IoU). Non-Maximum Suppression (NMS) is a post-processing algorithm that searches for boxes with local maxima of confidence values for object detection. It selects the box with the highest score from multiple detection boxes and removes the other boxes that repeatedly predict the same target. We use the IoU threshold to control the strictness of removing redundant boxes. In the inference phase, we apply axis-aligned NMS with an overlap threshold of 0.5 IoU.

We employ the same loss functions in SECOND [32] and PointPillars [33]. The 3D ground truth box is parameterized as $(x^{gt}, y^{gt}, z^{gt}, w^{gt}, l^{gt}, h^{gt}, \theta^{gt})$, and a matching positive anchor is defined as $(x^a, y^a, z^a, w^a, l^a, h^a, \theta^a)$. (x, y, z) represent the center location, (w, l, h) stand for the dimension (width, length, height) of the box, and θ is the yaw rotation around Z-axis. Then, the ground truth residual vector $\mathbf{u^{gt}} = (\Delta x, \Delta y, \Delta z, \Delta w, \Delta l, \Delta h, \Delta\theta)$ is represented as

$$\begin{aligned} \Delta x &= \frac{x^{gt} - x^a}{d^a}, \Delta y = \frac{y^{gt} - y^a}{d^a}, \Delta z = \frac{z^{gt} - z^a}{h^a} \\ \Delta w &= \log \frac{w^{gt}}{w^a}, \Delta l = \log \frac{l^{gt}}{l^a}, \Delta h = \log \frac{h^{gt}}{h^a} \\ \Delta\theta &= \sin(\theta^{gt} - \theta^a) \end{aligned} \tag{6}$$

where $d^a = \sqrt{(w^a)^2 + (l^a)^2}$ is the diagonal of base for the anchor box.

The regression loss for the box is formulated as

$$L_{box} = SmoothL1(\mathbf{u^{gt}} - \mathbf{u^{pred}}) \tag{7}$$

where the superscript $pred$ represents the prediction output.

As for the classification head and direction head, we jointly utilize the focal loss [64] and the cross-entropy loss with softmax, gaining L_{cls} and L_{dir}. The focal loss is designed to address the one-stage object detection scenario in which there is an extreme imbalance between foreground and background classes during training. The formula is

$$L_{cls} = -\alpha(1-p)^{\gamma} \log p \tag{8}$$

where p is the class probability of an anchor. We use $\alpha = 0.25$ and $\gamma = 2$ for the parameters of focal loss following SECOND and PointPillars.

Therefore, total loss is defined as

$$L = \frac{1}{N_{pos}}(\beta_{cls} L_{cls} + \beta_{box} L_{box} + \beta_{dir} L_{dir}) \tag{9}$$

where N_{pos} is the number of positive anchors. $\beta_{cls} = 1$, $\beta_{box} = 2$ and $\beta_{dir} = 0.2$ for the weights of loss is used in our work.

IV. Implementation Details

A. Overview

We use different configurations of PPME and network to trade off the model complexity and precision. TinyPillarNet-L is the standard version of TinyPillarNet, with a wider network and higher precision, suitable for deployment on mobile computing platforms. TinyPillarNet-S is the small version designed for edge platforms, which has smaller memory cost and faster inference speed. Compared with the standard

TABLE I
ARCHITECTURE DETAILS OF THE PROPOSED TBN

Operation	TinyPillarNet-L			TinyPillarNet-S		
	Ch $(C1, C2, C3)$	R	S	Ch $(C1, C2, C3)$	R	S
Conv	64	-	2	16	-	2
TD1	64, 32, 64	6	1	16, 8, 16	6	1
TD2	128, 64, 128	6	2	64, 32, 64	6	2
TD3	256, 128, 256	6	2	256, 128, 256	6	2
REF1	64, 32, 64	3	1	16, 8, 16	3	1
REF2	64, 32, 64	3	1	16, 8, 16	3	1
REF3	64, 32, 64	3	1	16, 8, 16	3	1

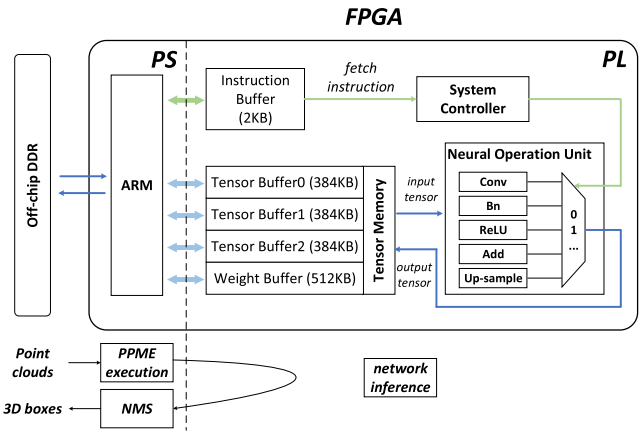

Fig. 8. The hardware architecture of the prototype system.

version, TinyPillarNet-S mainly decreases the detection range and the width of network, fitting our prototype system based on FPGA.

B. PPME

For the KITTI dataset, the detection range is within [0, −30.72, −3, 61.44, 30.72, 1]m for TinyPillarNet-L and [0, −20.48, −3, 61.44, 20.48, 1]m for TinyPillarNet-S. The pillar size is [0.16, 0.16]m. The detection range of TinyPillarNet-S is decreased to reduce the input size.

For saving memory space, 8-bit quantization is applied on pseudo-maps. The maximum number of points per pillar N_{max} is set as 127. I and D are converted to integers ranging [−127, 127] linearly.

C. TBN and SEN

The detailed configuration of TBN is illustrated in Tab.I, where Ch represents the numbers of filters in LRB which are respectively denoted by $C1, C2, C3$. Moreover, R indicates the repeat times, and S stands for the stride.

As for the SEN shown in Fig.6, we set the first Conv as 32 filters with stride 2, followed by 5 depthwise separable convolutions with $ChS = 2$. For TinyPillarNet-S, the first Conv is shrunk to 16 filters, followed by 4 depthwise separable convolutions with $ChS = 2$.

D. Hardware Architecture of Prototype System

For efficient implementation of lightweight networks, hardware designs of neural network accelerators are proposed. Aiming at solving the complex computation on 3D tasks, works [13], [14], [15], [17] are introduced to fulfill the requirements of real-world applications regarding resource usage, frame rates, and precision on 3D stereo reconstruction, 3D point cloud segmentation and 3D classification, respectively. Though the operations of points cloud computation are supported on edge devices, the peak performance and power efficiency are still low, which causes these 3D processing hardware architectures to be not used widely.

In this subsection, a special FPGA-based prototype system for TinyPillarNet is presented. The overall hardware architecture is shown in Fig.8. The TinyPillarNet framework includes three major stages, namely PPME execution, network inference, and NMS. They are arranged to be executed in programming system (PS) and programmable logic (PL) of FPGA. PS refers to the embedded processing system of SoC FPGA, being independent of PL, which makes software development be easier and yields abundant peripherals and interfaces.

PPME encodes the irregular and unordered point cloud data, while NMS performs sort operation, both of which mainly handle serial processing with less computational load. Given this, we assign PPME execution and NMS to the high-performance ARM core on PS.

The network inference is executed by CNN accelerator on PL, which includes a system controller, neural operation unit, instruction buffer, and tensor memory. The neural operation unit (NOU) supports common neural network operations, such as DWConv, PWConv, BatchNorm, ReLU, element addition, up-sample, etc. There are 288 of 8-bit MACs in DWConv module, 1024 of 8-bit MACs in PWConv module, and 32 floating-point MACs in BatchNorm module. For flexibly implementing the network, we build an application specific instruction-set to control NOU. As shown in Fig.8, the system controller decodes the instructions and controls the operations and tensor transmission. The tensor memory includes three 384KB sized buffers for features and a 512KB sized buffer for weight.

Before inference, ARM respectively sends instructions and weights to instruction buffer and weight buffer. During inference, the system controller fetches instructions and deploy the NOU to implement the intensive computational workloads. Running the proposed TBN and SEN, the specially designed accelerator can work in a single-chip manner, which reduces the system complexity as well as the memory accessing power and latency to the greatest extent. Then, the output features are further processed by NMS in ARM.

The execution of the accelerator at the PL is triggered by a rising edge and can work asynchronously with the PS. For running at high efficiency, our system supports task-level pipeline parallelism and heterogeneous execution between the PL and PS.

V. EXPERIMENTS

A. Training Strategy

1) Dataset: KITTI [34] is a common 3D object detection dataset, consisting of 7481 training data and 7518 test data. For experimental studies, we split the official training set into

TABLE II
THE PERFORMANCE IN BEV DETECTION ON KITTI TEST SPLIT

Method	Input Size (MB)	Weight Size (MB)	mAP Mod.	Car BEV AP (%) Easy	Car BEV AP (%) Mod.	Car BEV AP (%) Hard	Pedestrian BEV AP (%) Easy	Pedestrian BEV AP (%) Mod.	Pedestrian BEV AP (%) Hard	Cyclist BEV AP (%) Easy	Cyclist BEV AP (%) Mod.	Cyclist BEV AP (%) Hard
Point-based:												
PointRCNN	0.25 (0.36×)	15.49 (5.96×)	66.92	92.13	87.39	82.72	54.77	46.13	42.84	82.56	67.24	60.28
STD	0.25 (0.36×)	-	68.15	89.66	87.76	**86.89**	**60.99**	**51.39**	45.89	81.04	65.32	57.85
3DSSD	0.25 (0.36×)	28.82 (11.08×)	68.86	92.66	89.02	85.86	60.54	49.94	45.73	**85.04**	67.62	61.14
BEV-based:												
BirdNet	-	-	33.11	75.52	50.81	50.00	26.07	21.35	19.96	38.93	27.18	25.51
PIXOR	76.90 (110×)	12.52 (4.81×)	-	81.70	77.05	72.95	-	-	-	-	-	-
Voxel-based:												
VoxelNet	751.95 (1074×)	24.36 (9.37×)	58.25	89.35	79.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
SECOND	751.95 (1074×)	18.00 (6.92×)	60.56	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78
PointPillars	18.31 (26×)	16.04 (6.17×)	65.97	90.07	86.56	82.81	57.60	48.64	45.78	79.90	62.73	55.58
Part-A^2-anchor	1367.19 (1953×)	243.48 (93.65×)	68.01	89.52	84.76	81.47	59.72	51.12	**48.04**	81.91	**68.12**	**61.92**
$H^2$3D-RCNN	6.2 (8.85×)	77.00 (29.61×)	**69.06**	92.85	88.87	86.07	58.14	50.43	46.72	82.76	67.90	60.49
SA-SSD	1031.25 (1473×)	20.39 (7.84×)	-	**95.03**	**91.03**	85.96	-	-	-	-	-	-
Voxel R-CNN	1031.25 (1473×)	28.97 (11.14×)	-	94.85	88.83	86.13	-	-	-	-	-	-
TinyPillarNet-L	0.70 (1.0×)	2.60 (1.0×)	61.91	89.37	83.89	78.53	47.59	39.46	37.19	75.07	62.39	56.10
TinyPillarNet-S	**0.47 (0.67×)**	**1.69 (0.65×)**	56.81	89.14	81.21	75.96	40.31	33.31	31.13	66.99	55.91	50.73

TABLE III
THE PERFORMANCE IN 3D DETECTION ON KITTI TEST SPLIT

Method	mAP Mod.	Car 3D AP (%) Easy	Car 3D AP (%) Mod.	Car 3D AP (%) Hard	Pedestrian 3D AP (%) Easy	Pedestrian 3D AP (%) Mod.	Pedestrian 3D AP (%) Hard	Cyclist 3D AP (%) Easy	Cyclist 3D AP (%) Mod.	Cyclist 3D AP (%) Hard
Point-based:										
PointRCNN	57.94	86.96	75.64	70.70	47.98	39.37	36.01	74.96	58.82	52.53
STD	61.26	87.95	79.71	75.09	53.29	42.47	38.35	78.69	61.59	55.30
3DSSD	61.79	88.36	79.57	74.55	**54.64**	44.27	40.23	**82.48**	**64.10**	56.90
BEV-based:										
BirdNet	12.55	14.75	13.44	12.04	14.31	11.80	10.55	18.35	12.43	11.88
PIXOR	-	-	-	-	-	-	-	-	-	-
Voxel-based:										
VoxelNet	49.05	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37
SECOND	56.69	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
PointPillars	58.29	82.58	74.31	68.99	51.45	41.92	38.89	77.10	58.65	51.92
Part-A^2-anchor	61.70	85.94	77.86	72.00	54.49	44.50	**42.36**	78.58	62.73	**57.74**
$H^2$3D-RCNN	**63.18**	90.43	81.55	**77.22**	52.75	**45.26**	41.56	78.67	62.74	55.78
SA-SSD	-	88.75	79.79	74.16	-	-	-	-	-	-
Voxel R-CNN	-	**90.90**	**81.62**	77.06	-	-	-	-	-	-
TinyPillarNet-L	53.76	80.85	70.40	65.12	42.44	34.64	32.37	70.22	56.24	50.24
TinyPillarNet-S	48.43	79.39	66.30	59.68	35.26	28.10	26.06	63.15	50.89	46.11

3712 training samples and 3769 validation samples following [65]. We train our model on LiDAR point cloud only. According to KITTI benchmark, we focus on training three categories as *Car*, *Cyclist* and *Pedestrian* simultaneously in one network.

2) Training Settings: According to SECOND [32], we collect objects in ground truth boxes in training set and mix the boxes and associated points in current frame randomly. For each class, we select [15, 10, 10] samples for Car, Pedestrian and Cyclist in each frame. Then, random flipping along axis X, random global rotation and global scaling are applied to enhance the spatial variability and improve the generalization ability of our network. The flipping probability is 0.5, the rotation degree range is $[-\pi/6, \pi/6]$ and the scaling ratio is between [0.95, 1.05].

Following the settings in PointPillars [33], anchors are set as [3.9, 1.6, 1.56] for Car, [0.8, 0.6, 1.73] for Pedestrian and [1.76, 0.6, 1.73] for Cyclist with two vertical rotations. In the training, anchors are selected whose ground truth box IoU are larger than [0.6, 0.5, 0.5] for Car, Pedestrian and Cyclist. Anchors have IoU lower than [0.45, 0.35, 0.35] are marked as negative samples. In validation and test splits, according to experimental results, we set the prediction score threshold in NMS as [0.4, 0.25, 0.3] for Car, Pedestrian and Cyclist.

We train our proposed model for 200 epochs with the batch size of 16. Moreover, Onecycle [66] learning rate scheduler and Adam [67] optimizer are employed. The maximum lr in Onecycle is 0.03, pct start (the percentage of lr increasing part) is 0.4, and the weight decay is 0.01.

In addition, we directly apply existing weight quantization DoReFa [55] scheme in training. The weights and activation are pre-trained on float32, and then quantized to the range of $[-127, 127]$. We fine-tune the network by 80 epochs, with the maximum lr of Onecycle being 3e-4. The training code is based on OpenPCDet [68], which is a clear, simple, self-contained open source project for LiDAR-based 3D object detection. All the training tasks are implemented on two NVIDIA RTX 2080Ti GPUs.

B. Experimental Results

1) Quantitative Analysis: We report the precision (APs) and the memory cost on KITTI test split provided by official service in Tab.II and Tab.III, and compare inference time in Tab.IV to comprehensively assess the performance of our

TABLE IV
THE COMPARISON FOR INPUT SIZE, WEIGHT SIZE AND INFERENCE TIME OF EXISTING METHODS

Method	Time (ms)	Device (FP32 TFLOPs)
TinyPillarNet-S	**11.57**	GTX 2080Ti (13.45)
TinyPillarNet-S	13.74	GTX 1080Ti (11.34)
TinyPillarNet-L	18.30	GTX 2080Ti (13.45)
PointPillars	23	GTX 1080Ti (11.34)
$H^2$3D-RCNN	27	GTX 2080Ti (13.45)
TinyPillarNet-S	27.37	GTX 1050Ti (2.138)
TinyPillarNet-L	29.93	GTX 1080Ti (11.34)
3DSSD	38	TITAN V (14.90)
SA-SSD	40	GTX 2080Ti (13.45)
Voxel R-CNN	40	GTX 2080Ti (13.45)
SECOND	50	GTX 1080Ti (11.34)
TinyPillarNet-L	67.66	GTX 1050Ti (2.138)
Part-A^2-anchor	70	Tesla V100 (14.13)
STD	80	TITAN V (14.90)
PIXOR	93	TITAN XP (12.15)
PointRCNN	100	TITAN XP (12.15)
VoxelNet	222	TITAN X (6.691)

proposed method. Our TinyPillarNet-L gains applicable precision of 61.91% mAP in BEV detection and 53.76% mAP in 3D detection. Remarkably, we only use 0.7MB of input size, 2.6MB of weight size and 18.3ms of inference latency. The small version, TinyPillarNet-S, is far lighter and faster, which is only 0.47MB of input size, 1.69MB of weight size and 11.57ms of inference latency.

Compared with point-based methods, since the operations of our method are simple, high-efficient and light, TinyPillarNet runs at far higher speed and takes extremely tiny memory. As for the reason, PPME reduces the dimensions of the point cloud, making pseudo-maps achieve equivalent tiny input size of point-based methods. Meanwhile, the on-chip memory access mode of lightweight 2D feature extraction network greatly reduces the latency. Compared to BEV-based methods, TinyPillarNet achieves significantly outperforming results in precision. Since the PPME encodes the important 3D structure feature of points, our pseudo-maps preserves more height information of objects. Compared with voxel-based methods, TinyPillarNet yields competitive precision to PointPillars, and is superior to VoxelNet and SECOND. More than that, PPME sharply saves the input size, which makes our solution easily be deployed on IoT devices. Compared to $H^2$3D-RCNN with previous lowest record input size, our TinyPillarNet-L saves 88.7% of input buffer, and TinyPillarNet-S further saves 92.41%.

To assess the inference speed, we implement competitive methods on GPUs with different computation capacity. The testing results are arranged in order of latency and shown in Tab.IV. As can be seen, our TinyPillarNet achieves faster inference speed on equivalent or even lower performance GPUs. Specifically, TinyPillarNet-S is 1.67 times faster than the current record of PointPillars on GTX 1080Ti. Moreover, the TinyPillarNet-S yields 27.37ms latency even on GTX 1050Ti (2.138TFLOPs), which indicates our method can run in real-time on edge devices.

TABLE V
THE PERFORMANCE IN BEV AND 3D DETECTION ON KITTI VALIDATION SPLIT FOR CAR

Method	Car BEV AP (%)			Car 3D AP (%)		
	Easy	Mod.	Hard	Easy	Mod.	Hard
PointRCNN	95.52	91.25	88.99	92.38	85.29	82.86
STD	90.50	88.50	88.10	89.70	79.80	79.30
3DSSD	-	-	-	89.71	79.45	78.67
PIXOR	86.79	80.75	76.60	-	-	-
VoxelNet	89.60	84.81	78.57	81.97	65.46	62.85
SECOND	89.96	87.07	79.66	87.43	76.48	69.10
PointPillars	-	87.70	-	-	77.40	-
Part-A^2-anchor	90.42	88.61	87.31	89.47	79.47	78.54
SA-SSD	-	-	-	90.15	79.91	78.78
Voxel R-CNN	**95.52**	**91.25**	**88.99**	**92.38**	**85.29**	**82.86**
TinyPillarNet-L	92.30	86.55	83.73	86.47	75.98	71.35
TinyPillarNet-S	91.96	83.84	81.00	84.41	72.75	68.14

Note that in Tab.II and Tab.III, APs of *Car* and *Cyclist* show less precision loss than *Pedestrian*. The main reason lies in that the low density sampling of small objects from long distance leads to inaccurate representation of height and distributional information in pseudo-maps. To address this issue, a potential solution is to complete the points and introduce super-resolution process for under sampled small targets like pedestrians in 3D space.

In order to quantitatively assess the generalization ability, we further test our proposed method on KITTI validation split for car and expose the results of competitive methods in Tab.V. As can be seen, our method still achieves the above median precision in existing methods, which shows the stability and scenario adaptability of our method.

2) Qualitative Analysis: Qualitative results with tight oriented 3D bounding boxes are shown in Fig.9. As can be seen clearly, our method can produce high-quality 3D boxes for the classes of *Car* and *Cyclist*, which have point cloud occlusion or strongly dense objects. In contrast, detecting *Pedestrian* is a more challenging task, and it only behaves as the confusion with narrow bar-type objects in complex scenes, such as poles, flowerpots or chairs. The reason lies in that a small object in long distance can rarely be covered by dense point cloud, which leads to the inadequate representation of the object feature. Generally, these qualitative results demonstrate that our proposed extremely lightweight framework is practicable and effective in real-world point cloud object detection.

C. Ablation Study and Analysis

In this subsection, the effectiveness of each main unit will be comprehensively investigated via ablation experiments. Our proposed TinyPillarNet is tested on KITTI validation split with same setting in all the following experiments. The results are quantitatively assessed by mAP of three categories on three difficulty levels.

1) Optimal Configuration of Geometric Descriptors: To figure out the optimal configuration of descriptor considering both precision and memory consumption, we conduct an ablation study on PiontPillars' encoder and illustrate the results in Tab.VI. The size of each pillar indicates the memory consumption of different configurations, which is calculated by $C_{pillar} \times N_{max}$ (C_{pillar} denotes the number of descriptors, N_{max} indicates the maximum number of points in pillars).

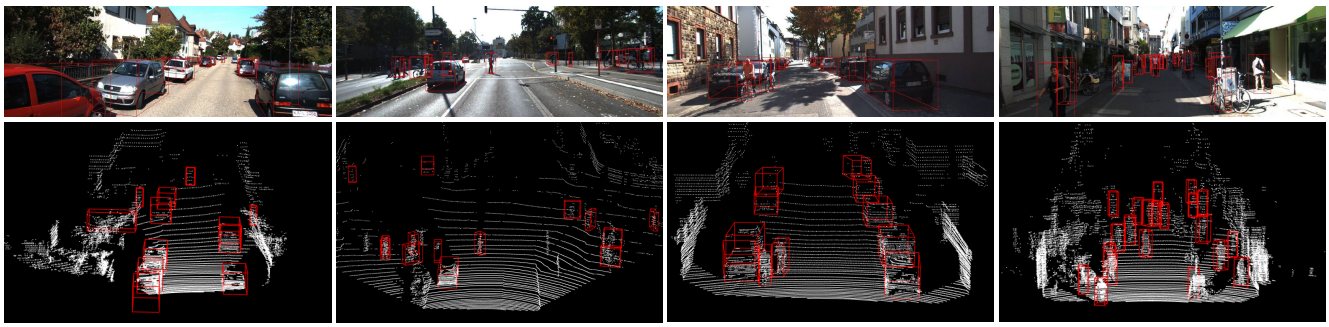

Fig. 9. The frames with 3D bounding boxes predicted by our TinyPillarNet on KITTI validation split.

TABLE VI
THE COMPARISON OF DIFFERENT COMBINATIONS OF DESCRIPTORS IN POINTPILLARS' ENCODER

Line	Descriptors									Size of each pillar	3D mAP (%)		
	x	$x_c\&x_p$	y	$y_c\&y_p$	z	$z_c\&z_p$	$z_{min}\&z_{max}$	r	N_{max}		Easy	Mod.	Hard
1	✓	✓	✓	✓	✓	✓		✓	100	10×100	74.28	62.76	58.66
2	✓	✓	✓	✓	✓	✓			100	9×100	72.45	60.53	55.99
3			✓	✓	✓	✓		✓	100	7×100	73.76	61.57	57.04
4	✓	✓			✓	✓		✓	100	7×100	73.71	61.20	56.72
5					✓	✓		✓	100	4×100	73.31	61.02	56.35
6	✓	✓	✓	✓	✓	✓		✓	1	10×1	63.35	52.83	49.26
7	✓	✓	✓	✓			✓	✓	1	9×1	70.55	59.55	55.38
8							✓	✓	1	3×1	71.51	61.10	56.50
9							✓		1	2×1	70.33	59.14	53.82

The baseline model in Line 1 (also known as PointPillars) contains 10 descriptors, including points (x, y, z, r), the distances to arithmetic mean (x_c, y_c, z_c), and offsets from pillar center (x_p, y_p, z_p). Firstly, compared with Line 1 and Line 8, Line 2 and Line 9 remove r and cause the mAP corresponding decrease of $[-1.83\%, -2.23\%, -2.67\%]$ and $[-1.18\%, -1.96\%, -2.67\%]$, which proves the indispensability of reflectivity r. Then, we test the effects of coordinates. Line 3 and Line 4 respectively remove (x, x_c, x_p) and (y, y_c, y_p) and cause a relatively less precision loss of $[-0.52\%, -1.19\%, -1.62\%]$ and $[-0.57\%, -1.56\%, -1.94\%]$. Furthermore, by removing all (x, y) coordinates, Line 5 mostly maintains the performance comparable to Line 3 and Line 4, which indicates the coordinates (x, y) are not indispensable for 3D detection. As for the reason, the location information carried by (x, y) has been kept implicitly in regular data arrangement.

Following that, we directly reduce N_{max} from 100 to 1 in Line 6, which causes the mAP decrease of $[-10.93\%, -9.93\%, -9.4\%]$. This remarkable precision loss can be interpreted as representing a pillar with only one point cannot work well for the abandon of height information of objects. In view of this, we replace z with the descriptors z_{min} and z_{max} to provide height information in Line 7, as a result, the precision greatly boosts $[+7.2\%, +6.72\%, +6.12\%]$, which means a more refined description of height should be introduced.

Finally, we turn to remove coordinates x, y and replace z with z_{min} and z_{max} in Line 8. This configuration shrinks the pillar size in Line 1 over 300 times only with an acceptable mAP decrease of $[-2.77\%, -1.66\%, -2.16\%]$.

2) Effects of Distributional Descriptors: On the basis of optimal configuration of geometric descriptors, we further discuss the effects of the proposed distributional descriptors D_n and D_{dd}.

TABLE VII
THE COMPARISON OF DIFFERENT DESCRIPTORS IN PROPOSED PPME

Descriptors				3D mAP (%)		
I_{zmin}/I_{zmax}	I_r	D_n	D_{dd}	Easy	Mod.	Hard
✓	✓			72.10	60.11	55.94
✓	✓	✓		72.80	61.51	57.13
✓	✓		✓	72.65	61.34	56.94
✓	✓	✓	✓	**74.15**	**62.77**	**57.95**

TABLE VIII
THE COMPARISON OF BACKBONES WITH DIFFERENT BLOCKS

Block Design	Weight Size	3D mAP (%)		
		Easy	Mod.	Hard
Plain Backbone	4.04MB	72.65	61.66	57.04
Bottleneck Block	665.02KB	70.67	58.33	54.40
SandGlass Block	753.52KB	73.23	61.88	57.03
SandGlass Block (removing the last DWConv)	665.02KB	74.02	62.16	57.54
Linear Residual Block	665.02KB	**74.15**	**62.77**	**57.95**

As shown in Tab.VII, Line 1 is the baseline with optimal configuration with I_{zmin}, I_{zmax} and I_r, by introducing proposed descriptors of D_n and D_{dd} respectively in Line 2 and Line 3, the mAP accordingly boosts $[+0.7\%, +1.4\%, +1.19\%]$ and $[+0.55\%, +1.23\%, +1.0\%]$. Given this, we further employ both of D_n and D_{dd} in Line 4, which brings a more remarkable improvement of $[+2.05\%, +2.66\%, +2.01\%]$. This phenomenon indicates the distributional information is beneficial for promoting the precision of 3D object detection.

3) Effects of Building Block LRB: To verify the effectiveness of the proposed building block, we compare the detection precision of different backbone structures in Tab.VIII. The baseline model shown in Line 1 is the plain backbone used in PointPillars, which consists of 16 convolution layers. However,

TABLE IX
THE COMPARISON OF DIFFERENT DESIGNS FOR THE DISTRIBUTION-AWARE SEN

Method	Processing Pipeline	Network Structure	3D mAP (%)		
			Easy	Mod.	Hard
Baseline	-	-	72.10	60.11	55.94
(a)	TBN	-	58.49	46.26	43.11
(b)	TBN+SEN(b)	Fig.10(b)	73.88	61.69	57.67
(c)	TBN+SEN(c)	Fig.10(c)	73.74	61.41	57.46
(d)	TBN+SEN(d)	Fig.10(d)	74.14	62.36	57.82
(e) Ours	TBN+SEN(e)	Fig.10(e)	**74.15**	**62.77**	**57.95**

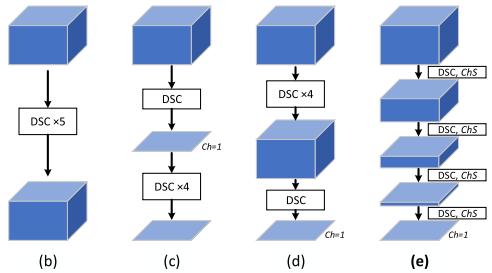

Fig. 10. The multiple designs for the Distribution-aware SEN. "Ch" represents the number of channels. DSC represents Depthwise separable convolution.

without using the grouped convolutions, its weight size is up to 4.04MB.

As for the commonly used bottleneck block, it greatly shrinks the weight size to 665.02KB but results in a precision decline of [−1.98%, −3.33%, −2.64%]. The sandglass block shown in Line 3 yields a similar precision to plain backbone with only 753.52KB weight size. The results shown in Line 4 indicate a remarkable promotion compared with the original sandglass structure.

It is worth noting that our proposed LRB introduces high parameter efficiency linear convolutions and remarkably boosts the precision up to the highest [74.15%, 62.77%, 57.95%] with the same minimal weight size as bottleneck block, which proves the proposed building block can balance the memory consumption and processing precision to a great extent.

4) Effects of SEN Branch: To illustrate the effects of the specially designed SEN, we compare the precision of different structures illustrated in Fig.10. As the results reported in Tab.IX, the baseline shown in Line 1 extracts features from intrinsic pseudo-map with TBN, which is just as the processing mode of existing voxel-based methods.

Method (a) processes the mixture of intrinsic and distributional pseudo-maps with TBN, which results in a serious precision decline compared with the proposed method (e). As for the reason, the intrinsic pseudo-maps mainly represent geometric information, while the distributional pseudo-maps focus on delivering location information. Because of the great difference in characteristics, these two types of pseudo-maps can't be processed with TBN in a mixed manner.

Methods (b)-(e) process distributional information with an independent network. Specially speaking, method (b) extracts the saliency feature using convolutions without downsampling channels, gaining mAP promotion of [+1.78%, +1.58%, +1.73%]. Method (c) and (d) change the number of channels to 1 at the beginning and the end, respectively. Method (c) has a similar precision as (b), while (d) gains a great improvement of [+2.04%, +2.25%, +1.88%]. Such a result reveals that adequate feature extraction in anterior layers and output single-channel mask can yield positive and interesting recommendations for the distributional feature.

Finally, by introducing the feature compression strategy, method (e) achieves the highest precision of [74.15%, 62.77%, 57.95%] as well as saves much more computational costs. In view of this, we apply structure (e) to produce more accurate distribution-aware saliency maps for further enhancing the feature representation.

5) Inference Speed on GPU: To deeply analyze the processing latency, each step in the proposed model is tested on GPUs. In order to compare with the algorithms deployed on different devices, we verify the inference time of our TinyPillarNet on multiple GPUs. The results are illustrated in Tab.X.

As follows, we analyze the timing of each step running on 2080ti. According to the processing pipeline of our proposed framework, the points are firstly loaded and organized as pillars, respectively taking 5.32ms and 4.72ms for TinyPillarNet-L and TinyPillarNet-S. The distinct latency in this step mainly comes from the different preparing time of various point range. Then, the pillars are uploaded on the GPU, taking 2.72ms and 2.7ms. Next, the encoding step only takes 1.81ms and 1.78ms to realize dimension reduction of feature. Finally, the features are extracted by network and further processed by NMS, taking 8.45ms and 2.37ms, in which NMS is performed on GPU and latency is about 0.8ms. As a result, the dual-stream feature extraction network consumes most of the inference latency, and the lightweight version of TinyPillarNet-S shows greater advantage in speed.

D. Implementation on FPGA-Based Prototype System

To verify the effectiveness of our proposed lightweight optimization strategies in the design of point cloud object detection model, the extremely simple TinyPillarNet-S model is specially deployed and implemented on FPGA.

In order to satisfy the strict storage constraint of edge devices, DoReFa scheme [55] is firstly applied to perform the 8-bit quantization for weight, and the resulting precision on KITTI test split is shown in Tab.XI. Though the quantization causes 1.55% and 3.66% of precision loss, respectively, in BEV and 3D detection, such a detection accuracy is still applicable in practical scenarios.

Then, we validate the prototype system specially designed for our TinyPillarNet on Xilinx ZC706 platform. The pipeline of FPGA implementation and the inference time are shown in Fig.11. Since the system supports cooperative workflow between the PL and PS, we design an efficient pipeline structure according to the latency of different operation stage, which promotes the parallelism and boosts the detection speed up to 34Hz (corresponding to 29.3ms of inference time per frame) for 3D detection.

The hardware resource utilization of the prototype system is reported in Tab.XIII. By comparing with hardware solutions for image object [69] and typical point cloud processing tasks (such as objects detection [69], point cloud classification [14] and point cloud segmentation [13], [15]) in Tab.XII, our

TABLE X
THE LATENCY OF EACH STEP OF TINYPILLARNET

Operation	TinyPillarNet-L			TinyPillarNet-S		
	1050Ti	1080Ti	2080Ti	1050Ti	1080Ti	2080Ti
Organize Pillars	5.32	5.32	5.32	4.72	4.72	4.72
Upload on GPU	6.23	3.27	2.72	5.69	2.73	2.70
Pillar Pseudo-map Encoding	5.84	2.02	1.81	5.31	1.96	1.78
Network Inference and NMS	50.27	19.32	8.45	11.65	4.33	2.37
Total	67.66	29.93	18.30	27.37	13.74	11.57

TABLE XI
THE PERFORMANCE OF 8-BIT QUANTIZATION STRATEGY IN BEV AND 3D DETECTION ON KITTI TEST SPLIT

Method	mAP	Car BEV AP (%)			Pedestrian BEV AP (%)			Cyclist BEV AP (%)		
	Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
TinyPillarNet-S (float)	56.81	89.14	81.21	75.96	40.31	33.31	31.13	66.99	55.91	50.73
TinyPillarNet-S (8-bit)	55.26	88.60	80.64	73.49	38.73	31.17	29.11	65.68	53.96	48.81
Method	**mAP**	**Car 3D AP (%)**			**Pedestrian 3D AP (%)**			**Cyclist 3D AP (%)**		
	Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
TinyPillarNet-S (float)	48.43	79.39	66.30	59.68	35.26	28.10	26.06	63.15	50.89	46.11
TinyPillarNet-S (8-bit)	44.77	74.23	60.55	55.63	32.90	26.01	24.10	59.85	47.77	43.09

TABLE XII
THE COMPARISON OF OTHER IMPLEMENTATIONS FOR POINT CLOUD TASKS

	[69]	[14]	[15]	[13]	Ours
Platform	Virtex-7 VC707	Zynq XC7z045	Zynq ZCU102	Cyclone V	Zynq ZC706
Application	Object Detection	Point Classification	Point Segmentation	Point Segmentation	3D Object Detection
Model	Yolov3	O-PointNet [14]	SS U-Net [62]	3D Capsule Network	TinyPillarNet-S
Frequency	200 MHz	100 MHz	270 MHz	200 MHz	150 MHz
Precision	INT8	16 Fixed	INT8/INT16	INT4/INT8	INT8/FP32
DSP	2640 (94.28%)	-	256 (10.16%)	278 (81%)	883 (98.11%)
Power	9.478* W	2.149 W	3.45 W	2.15 W	3.63 W
Peak Performance	767.3 GOPS	2.416 GOPS	17.73 GOPS	108.8 GOPS	304.31 GOPS
Energy Efficiency	80.95* GOPS/W	1.12 GOPS/W	5.14 GOPS/W	50.60 GOPS/W	**83.83** GOPS/W

"*": We evaluate the inference power of [69] with Xilinx Power Estimator (XPE) tool according to the reported resource utilization.

TABLE XIII
FPGA FREQUENCY AND RESOURCE UTILIZATION

Frequency (MHz)	LUT	FF	BRAM	DSP
150	128721 (58.88%)	111118 (25.42%)	382.5 (70.18%)	883 (98.11%)

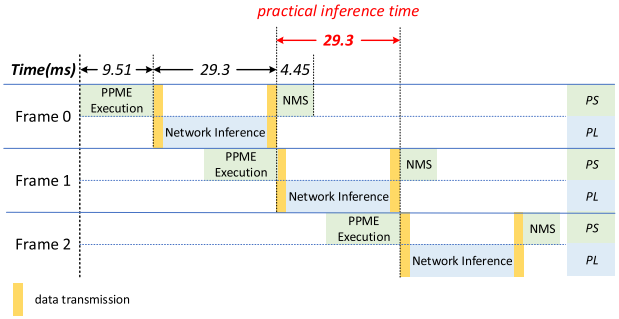

Fig. 11. The pipeline and latency of processing.

system only consumes 3.63W power for achieving a superior peak performance of 304.31GOPS, which results in an outstanding energy efficiency of 83.83GOPS/W. As for the reason, the main workloads in our TinyPillarNet model are constructed by specially designed building blocks consisting of DWConv and PWConv operations, while other designs mainly implement operators of sparse Conv [15], 1D Conv [13], [14] and standard Conv.

As for sparse Conv operations, the processes of dividing tiles and generating masks indicated in [15] make the hardware circuit more complex. Meanwhile, the uncertain number of tiles decreases the computing resource utilization, and thus results in lower energy efficiency than the computing-dense DWConv and PWConv operations in our solution.

Compared with 1D Conv and standard Conv, DWConv needs only calculate each channel of feature map once, thus the memory access for weights and feature maps is correspondingly less [5], which results in far lower energy consumption [16]. As for PWConv, it is equivalent to standard Conv with 1×1 kernel size, and thus the energy efficiency is the same. In terms of the above principle, the proposed TinyPillarNet model is more energy efficient in computing process than existing methods in this field.

In addition, our method fully considers the considerable power consumption of memory access in inference phase. By introducing high efficient point cloud encoder and lightweight network design strategies, our proposed TinyPillarNet can implement the complicated end-to-end network inference without accessing off-chip memory, which further promotes the energy efficiency.

VI. CONCLUSION

In this paper, we propose an extremely tiny 3D point cloud object detection framework, called TinyPillarNet, for application in the TinyML scenario. The main contribution of our work is to break the energy efficiency bottleneck from the perception of the point cloud encoding strategy and the ultra-light building blocks of the feature extraction network. In particular, a novel Pillar Pseudo-Map Encoder is proposed to considerably shrink 3D point cloud data to 2D

pseudo-maps for saving input buffer. Moreover, an innovative Linear Residual Block is proposed in the backbone of the feature extraction network to reduce the weight size of the model. The above-mentioned efforts realize strong efficient inference in an on-chip memory access manner. Quantitative evaluation shows that our tiny model achieves comparable results on the KITTI benchmark as state-of-the-art competitors at higher inference speed. Meanwhile, the prototype verification experiments on FPGA additionally demonstrate the outstanding peak performance and energy efficiency of the proposed framework. Our work significantly advances the TinyML community and promises to significantly expand the scope of LiDAR applications at the edge.

In the future, we will explore more network lightweight solutions, such as distillation [70], [71], memory-efficient module design [44]. At the same time, in order to cope with the increasing difficulty of 3D data labeling, we will also focus on low-supervised learning methods [41], [42], [43], [44].

REFERENCES

[1] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A survey on 3D object detection methods for autonomous driving applications," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 10, pp. 3782–3795, Oct. 2019.

[2] G. Zamanakos, L. Tsochatzidis, A. Amanatiadis, and I. Pratikakis, "A comprehensive survey of LiDAR-based 3D object detection methods with deep learning for autonomous driving," *Comput. Graph.*, vol. 99, pp. 153–181, Oct. 2021.

[3] Y. Guo, F. Sohel, M. Bennamoun, M. Lu, and J. Wan, "Rotational projection statistics for 3D local surface description and object recognition," *Int. J. Comput. Vis.*, vol. 105, no. 1, pp. 63–86, Oct. 2013.

[4] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, and J. Wan, "3D object recognition in cluttered scenes with local surface features: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2270–2287, Nov. 2014.

[5] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake, UT, USA, Jun. 2018, pp. 4510–4520.

[7] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake, UT, USA, Jun. 2018, pp. 6848–6856.

[8] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2018, pp. 122–138.

[9] K. Xu, H. Zhang, Y. Li, Y. Zhang, R. Lai, and Y. Liu, "An ultra-low power TinyML system for real-time visual processing at edge," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 7, pp. 2640–2644, Jul. 2023, doi: 10.1109/TCSII.2023.3239044.

[10] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 33. Red Hook, NY, USA: Curran Associates, 2020, pp. 11711–11722.

[11] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," 2021, *arXiv:2110.15352*.

[12] J. Guan, R. Lai, H. Li, Y. Yang, and L. Gu, "DnRCNN: Deep recurrent convolutional neural network for HSI destriping," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 7, pp. 3255–3268, Jul. 2022, doi: 10.1109/TNNLS.2022.3142425.

[13] G. Park, D. Im, D. Han, and H.-J. Yoo, "A 1.15 TOPS/W energy-efficient capsule network accelerator for real-time 3D point cloud segmentation in mobile environment," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 9, pp. 1594–1598, Sep. 2020.

[14] X. Zheng, M. Zhu, Y. Xu, and Y. Li, "An FPGA based parallel implementation for point cloud neural network," in *Proc. IEEE 13th Int. Conf. ASIC (ASICON)*, Chongqing, China, Oct. 2019, pp. 1–4.

[15] Z. Wang, W. Mao, P. Yang, Z. Wang, and J. Lin, "An efficient FPGA accelerator for point cloud," in *Proc. IEEE 35th Int. Syst. Chip Conf. (SOCC)*, Belfast, U.K., Sep. 2022, pp. 1–6.

[16] C. Xiao, D. Xu, S. Qiu, C. Shi, and K. Ning, "FGPA: Fine-grained pipelined acceleration for depthwise separable CNN in resource constraint scenarios," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Social Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, New York, NY, USA, Sep. 2021, pp. 246–254.

[17] G.-T. Michailidis, R. Pajarola, and I. Andreadis, "High performance stereo system for dense 3-D reconstruction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 6, pp. 929–941, Jun. 2014.

[18] J. Yin, J. Shen, X. Gao, D. Crandall, and R. Yang, "Graph neural network and spatiotemporal transformer attention for 3D video object detection from point clouds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 8, pp. 9822–9835, Aug. 2021, doi: 10.1109/TPAMI.2021.3125981.

[19] F. Yang, X. Li, and J. Shen, "Nested architecture search for point cloud semantic segmentation," *IEEE Trans. Image Process.*, vol. 32, pp. 2889–2900, 2022, doi: 10.1109/TIP.2022.3147983.

[20] J. Yin, J. Shen, C. Guan, D. Zhou, and R. Yang, "LiDAR-based online 3D video object detection with graph-based message passing and spatiotemporal transformer attention," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 11492–11501.

[21] Z. Yuan, X. Song, L. Bai, Z. Wang, and W. Ouyang, "Temporal-channel transformer for 3D LiDAR-based video object detection for autonomous driving," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 4, pp. 2068–2078, Apr. 2022.

[22] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 770–779.

[23] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3DSSD: Point-based 3D single stage object detector," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 11037–11045.

[24] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 77–85.

[25] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30. Long Beach, CA, USA: Curran Associates, 2017, pp. 5099–5108.

[26] S. Shi et al., "PV-RCNN: Point-voxel feature set abstraction for 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 10526–10535.

[27] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. García, and A. De La Escalera, "BirdNet: A 3D object detection framework from LiDAR information," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, Maui, HI, USA, Nov. 2018, pp. 3517–3523.

[28] Y. Zeng et al., "RT3D: Real-time 3-D vehicle detection in LiDAR point cloud for autonomous driving," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3434–3440, Oct. 2018.

[29] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D object detection from point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake, UT, USA, Jun. 2018, pp. 7652–7660.

[30] B. Yang, M. Liang, and R. Urtasun, "HDNET: Exploiting HD maps for 3D object detection," in *Proc. Mach. Learn. Res. (PMLR), Conf. Robot Learn. (CoRL)*, Zürich, Switzerland, vol. 87, Oct. 2018, pp. 146–155.

[31] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake, UT, USA, Jun. 2018, pp. 4490–4499.

[32] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, Oct. 2018.

[33] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 12689–12697.

[34] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Providence, RI, USA, Jun. 2012, pp. 3354–3361.

[35] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D object detection from RGB-D data," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake, UT, USA, Jun. 2018, pp. 918–927.

[36] D. Xu, D. Anguelov, and A. Jain, "PointFusion: Deep sensor fusion for 3D bounding box estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, Jun. 2018, pp. 244–253.

[37] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "STD: Sparse-to-dense 3D object detector for point cloud," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Seoul, Korea (South), Oct. 2019, pp. 1951–1960.

[38] D. Zeng Wang and I. Posner, "Voting for voting in online point cloud object detection," in *Proc. 11th Robot., Sci. Syst.*, Rome, Italy, Jul. 2015.

[39] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li, "From points to parts: 3D object detection from point cloud with part-aware and part-aggregation network," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 8, pp. 2647–2664, Aug. 2021.

[40] J. Deng, W. Zhou, Y. Zhang, and H. Li, "From multi-view to hollow-3D: Hallucinated hollow-3D R-CNN for 3D object detection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 12, pp. 4722–4734, Dec. 2021.

[41] Q. Meng, W. Wang, T. Zhou, J. Shen, Y. Jia, and L. Van Gool, "Towards a weakly supervised framework for 3D point cloud object detection and annotation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 8, pp. 4454–4468, Aug. 2022.

[42] J. Yin et al., "$Proposalcontrast$: Unsupervised pre-training for LiDAR-based 3D object detection," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2022, pp. 17–33.

[43] J. Yin et al., "Semi-supervised 3D object detection with proficient teachers," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2022, pp. 727–743.

[44] Z. Zhao, S. Zhao, and J. Shen, "Real-time and light-weighted unsupervised video object segmentation network," *Pattern Recognit.*, vol. 120, Dec. 2021, Art. no. 108120.

[45] D. L. Dutta and S. Bharali, "TinyML meets IoT: A comprehensive survey," *Internet Things*, vol. 16, Dec. 2021, Art. no. 100461.

[46] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[48] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 2815–2823.

[49] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, Long Beach, CA, USA, May 2019, pp. 6105–6114.

[50] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[51] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2018, pp. 815–832.

[52] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 30. Long Beach, CA, USA: Curran Associates, 2017, pp. 2181–2191.

[53] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1398–1406.

[54] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 2755–2763.

[55] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.

[56] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, 2017, pp. 1–10.

[57] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2016, pp. 525–542.

[58] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 8604–8612.

[59] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," 2018, *arXiv:1805.06085*.

[60] D. Zhou, Q. Hou, Y. Chen, J. Feng, and S. Yan, "Rethinking bottleneck structure for efficient mobile network design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2020, pp. 680–697.

[61] Y. Li, K. Xu, R. Lai, and L. Gu, "Towards an effective orthogonal dictionary convolution strategy," in *Proc. AAAI Conf. Artif. Intell.*, Jun. 2022, vol. 36, no. 2, pp. 1473–1481.

[62] B. Graham, M. Engelcke, and L. V. D. Maaten, "3D semantic segmentation with submanifold sparse convolutional networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake, UT, USA, Jun. 2018, pp. 9224–9232.

[63] W. Liu et al., "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Cham, Switzerland: Springer, 2016, pp. 21–37.

[64] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 2999–3007.

[65] X. Chen et al., "3D object proposals for accurate object class detection," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28. Montreal, QC, Canada: Curran Associates, 2015, pp. 424–432.

[66] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," 2017, *arXiv:1708.07120*.

[67] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[68] O. D. Team. (2020). *OpenPCDet: An Open-Source Toolbox for 3D Object Detection From Point Clouds*. [Online]. Available: https://github.com/open-mmlab/OpenPCDet

[69] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in FPGA design for CNN-based object detectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 6, pp. 2450–2464, Jun. 2021.

[70] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[71] J. Shen, Y. Liu, X. Dong, X. Lu, F. S. Khan, and S. Hoi, "Distilled Siamese networks for visual tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 8896–8909, Dec. 2022.

Yishi Li received the B.S. degree in electronics science and technology from Xidian University, Xi'an, China, in 2017, where he is currently pursuing the Ph.D. degree in electronics science and technology.

His research interests are deep learning for computer vision and neural network accelerator; including image classification, object detection, and 3D object detection.

Yuhao Zhang received the B.S. degree in microelectronics science and engineering and the M.S. degree in integrated circuit engineering from Xidian University, Xi'an, China, in 2020 and 2023, respectively.

His research interests include 3D object detection and hardware acceleration for deep learning.

Rui Lai (Member, IEEE) received the B.S. degree in information engineering from Xidian University, Xi'an, China, in 2002, and the M.S. and Ph.D. degrees in electronic science and technology in 2005 and 2007, respectively.

He is currently a Professor with the Microelectronics Department, Xidian University. He is the author of more than 60 journal and conference papers. His research interests include smart sensors, image processing, computer vision, and deep learning.