FMTT : Fused Multi-head Transformer with Tensor-compression for 3D Point Clouds Detection on Edge Devices

Zikun Wei, Tingting Wang, Chenchen Ding, Bohan Wang, Ziyi Guan, Hantao Huang, and Hao Yu

School of Microelectronics, Southern University of Science and Technology, Shenzhen China

***Abstract*—The real-time detection of 3D objects represents a grand challenge on edge devices. Existing 3D point clouds models are over-parameterized with heavy computation load. This paper proposes a highly compact model for 3D point clouds detection using tensor-compression. Compared to conventional methods, we propose a fused multi-head transformer tensor-compression (FMTT) to achieve both compact size yet with high accuracy. The FMTT leverages different ranks to extract both high and low-level features and then fuses them together to improve the accuracy. Experiments on the KITTI dataset show that the proposed FMTT can achieve 6.04× smaller than the uncompressed model from 55.09MB to 9.12MB such that the compressed model can be implemented on edge devices. It also achieves 2.62% improved accuracy in easy mode and 0.28% improved accuracy in hard mode.**

***Index Terms*—Deep Learning, 3D Object Detection, Tensor Compression**

I. Introduction

3D objects real-time detection is a critical and challenging task in autonomous driving and robotics. Existing works [1]–[3] adopt 3D convolution networks to extract voxel features from point clouds. However, 3D convolution networks struggle with capturing rich context information with limited receptive fields. To overcome this problem, VoTr [4], a Transformer-based 3D backbone is proposed to build long-range relationships between voxels features and serve as a substitute for conventional 3D convolution backbones. However, the large model size of Transformers poses a challenge for deployment on edge devices.

Model compression methods aim to reduce the complexity of neural networks. One popular method is quantization-aware training (QAT) [5], which reduces the bitwidth of network weights and activations during training. However, QAT methods are time-consuming and require additional GPU memory to train the quantization parameters. Network pruning [6] is another method that removes redundant network structures but involves a complex retraining procedure. Tensor-compression, based on tensor decomposition, reduces the storage and computational overhead of high-order tensors. It represents a tensor as a series of low-rank tensors' product, reducing parameter counts and storage requirements. Tensor-compression uses core tensors and factor matrices to represent the tensor, capturing correlations between different modes. While tensor-compression significantly reduces parameters and storage, it is still challenging to capture important information from feature maps, leading to lower accuracy compared to uncompressed models.

Corresponding author: yuh3@sustech.edu.cn, huanght@sustech.edu.cn

In this work, we propose a novel fused multi-head transformer-based tensor-compression model for 3D point clouds detection. We achieve better accuracy with fewer parameters. The main contributions of our work are as follows: 1) An end-to-end 3D point clouds voxel transformer based model is fully compressed by the tensor-compression. In comparison with uncompressed model, it achieves $6.04\times$ times compression rate and 2.62% accuracy improvements. 2) A novel fused multi-head tensor compression for both attention and convolution is proposed to compress the model. 3) A tensor-train rank selection strategy is proposed with consideration of model size, computation load and accuracy during training. Based on this selection strategy, rank 4 and rank 8 are selected and fused to achieve 73.27% accuracy, which outperforms the uncompressed model and rank 16 based tensor-compression model with smaller model size.

II. Related Work

A. 3D Point Clouds Detection Methods

3D object detection from point clouds data can be summarized into 2 streams: point-based [7], [8] and voxel-based [1], [2], [4]. Point-based detectors directly work on raw point data for detection. For example, PointNet [7] processes raw point data for detection. However, this method suffers from the large number of input point data, resulting in a large memory usage. On the other hand, voxel-based detectors will process point clouds into voxel (voxelization) and then apply convolution to detect objects. VoxelNet [1] uses sparse voxel grids and introduces voxel feature embedding (VFE) layers to extract voxel features. SECOND [2] employs sparse convolution to efficiently convert voxels to Bird's Eye View (BEV) maps. In 3D point clouds detection, VoTr [4] introduces transformer-based modules like the sparse voxel module and submanifold voxel module. However, VoTr's large size poses challenges for deployment on edge devices. Therefore, compressing the model to make it smaller and faster is crucial. Our proposed fused multi-head tensor-compression method reduces redundant parameters significantly while maintaining slightly higher accuracy compared to the uncompressed model.

B. Network Compression

Existing model compression strategy mainly focuses on quantization and pruning. Quantization [5] is to decrease

979-8-3503-4859-0/DATE24/© 2024 EDAA

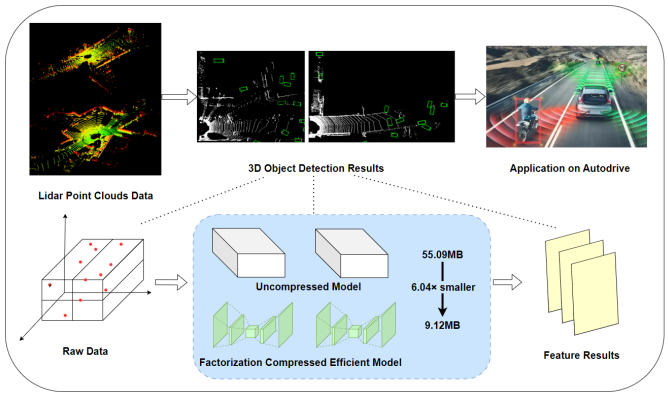

Fig. 1: Visualization on 3D Car Detection for Automatic Drive and the Intuitive Comparison between Uncompressed Model and Factorization Model

the bit width from float 32 bits to int 8 bits, or even 4 bits whereas pruning method [6] is to set unimportant parameters to zero. These methods work well but reach their limits to further compress the model. On the other hand, an orthogonal method, low-rank factorization, can further compress the model with or without quantization and pruning. Low-rank factorization decomposes tensors using low-rank tensors, which can effectively compress a large matrix to a list of small matrices. For example, tensor-train decomposition breaks down dimensions into a sequence of low-rank tensors, which has been successfully applied to deep learning tasks, such as loop closure detection in robotic visual SLAM [9], video LSTM networks [10], and fall-detection model [11] [12], showing its efficient performance. In this work, we propose fused multi-head tensor-compression strategy to keep the size small while keep the accuracy the same or even better than the uncompressed model.

III. 3D Point Clouds Detection

The goal of 3D point clouds detection is to extract valuable information from point clouds and identify objects of interest, such as vehicles, pedestrians, or buildings. Fig. 1 illustrates the potential application and the data flow path of 3D point clouds detection. The classic structure of 3D point clouds detection involves voxelization module, backbone module and detection head module. Voxelization module [2] used to take raw point clouds as input and partition the space into voxels, which decreases the sparsity of the raw data. The main ideas of voxelization are grouping the data points according to the voxel and encoding the voxel features. The 3D voxel grid size is $D' = \frac{D}{v_D}, H' = \frac{H}{v_H}, W' = \frac{W}{v_W}$, where D, H, W are the 3D space range and v_D, v_H, v_W are voxel size. After the grouping process, the voxel feature comes to $\mathbf{V} \in \mathbb{R}^{N \times M \times C}$, where N represents numbers of voxels, M represents max points per voxel and C represents the position x, y, z and reflect density of points. Then a normalizer is applied to calculate mean value which is formulated as:

$$\begin{aligned} Norm &= Clamp(Sum(V_{points}), min), \\ V_{VFE} &= Sum(V_{num})/Norm \end{aligned} \tag{1}$$

where the V_{points} means the point numbers of each voxels and clamp minimum to one ensures divisor is nonzero. $Sum(V_{num})$ calculates the sum of all points positions in voxels so the new voxel feature comes from the average of each voxels, which heavily decreases redundant data and keeps information.

The primary challenge in 3D detection lies in the sparsity of point clouds data, making it difficult to extract effective features. Conventional 3D convolutional backbones used in voxel-based detectors have limited receptive fields, hindering their ability to capture essential contextual information required for object recognition and localization. VoTr [4], a transformer-based architecture, addresses this issue by leveraging self-attention to establish long-range relationships between voxels. To optimize the impact of Transformers on voxels, VoTr incorporates the sparse voxel module and the submanifold voxel module. VoTr consistently outperforms convolutional baselines on the KITTI dataset and the Waymo Open dataset, demonstrating its immense potential for practical applications. However, the large parameter size and heavy computational load of VoTr pose significant challenges for deploying it on edge devices, limiting its real-life application. The parameters and computations of VoTr primarily concentrate in the linear layer and convolution layer. To address this, we propose fused multi-head transformer tensor-compression (FMTT), which not only reduces the parameter size but also improves the model's accuracy, as Fig. 1 shows. This compression technique makes the transformer-based model more suitable for edge devices, improving its practicality.

TABLE I: Notations in our fused multi-head tensorized model

Notations	Descriptions
O, W, x, b	Outputs, weights, inputs, bias
$\mathscr{Y}, \mathscr{W}, \mathscr{X}, \mathscr{B}$	Tensorized outputs, weights, inputs and bias
$W \in \mathbb{R}^{M \times N}$	Tensor W of size $M \times N$
$C_{m_1}, C_{m_2} ..., C_{m_d}$	Reshaped size of M
$C_{n_1}, C_{n_2} ..., C_{n_d}$	Reshaped size of N
$\mathscr{G}_k \in \mathbb{R}^{r_{k-1} \times p_k \times r_k}$	Tensor cores of tensor-train data format
$\mathscr{G}_k^*(i_k, j_k)$	Tensor cores with double index
$r_0, r_1, ..., r_d$	Ranks of tensor cores
$\alpha_1, \alpha_2, ..., \alpha_p$	Coefficients of FMTT before normalized
$\beta_1, \beta_2, ..., \beta_p$	Normalized coefficients of FMTT
$\mathscr{G}_k^{p*}(i_k, j_k)$	Tensor cores of each head with double index
$\#FLOPs$	Floating point operations
C_{in}, C_{out}	Channel-in and channel-out of the feature
H, W	Height and width of the feature
C_{rank}	The criterion of rank selection
C_{loss}, C_{comp}	The criterion from loss and complexity
$\#PARAMs$	The parameters number of the operation

IV. Tensorized 3D Point Clouds Detection Network

In this part, we present the whole tensorized 3D point clouds network. The prototype of the model is VoTr [4] and we focus

on how to apply the fused multi-head tensor-compression method on the model.

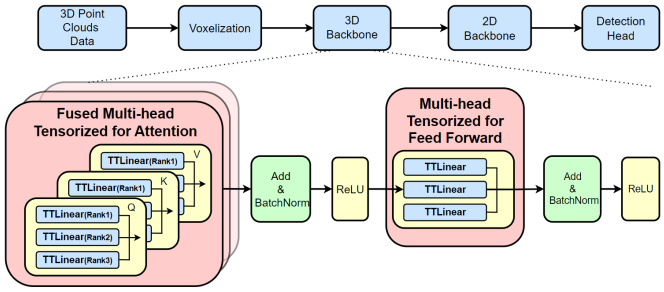

Fig. 2: The Overall Model Architecture with Fused Multi-head Tensorized Blocks for Attention and Feed Forward

As shown in Fig. 2, the overall model is divided to voxelization module, 3D backbone module, 2D backbone module and detection head. On the first step, the model obtains the raw point clouds data from Lidar and the point clouds data will be processed to voxel formation. The voxelization can highly change the redundant sparse data to more efficient voxel data. Then, the voxel will pass through the 3D backbone, which extracts 3D features and generate Bird's Eye View(BEV) features. The BEV features are verified by many works that can embody 3D information in 2D features and save the computation resources. Besides, the BEV features are also easier for the object detection to extract features. After obtaining the BEV features, the most significant 2D backbone will extract the features before detection heads.

Overall, the 3D backbone is based on transformer, which is first used in 3D object detection in VoTr [4]. The 3D backbone is mainly formulated by two transformer-based module, submanifold voxel module and sparse voxel module. Those two modules both have the attention blocks and Mlp blocks and need lots of operations. And our proposed fused multi-head tensor-compression attention block displaces the attention block. Fig. 3 shows the comparison between the conventional attention block and our compressed attention block. The details of the fused multi-head tensor-compression attention will be discussed in section V later. Furthermore, the fused multi-head tensor-compression linear replaces the conventional linear as the novel feed forward layer. In this way, tensorized multi-head attention(TMA) block formulates as:

$$\begin{aligned} TMA(\mathrm{h}) &= TTLinear(Concat(h_1, h_2, ...h_i)) \\ h_i &= softmax(\frac{TQ_i \cdot TK_i^T}{\sqrt{d}})TV_i \\ TQ_i, TK_i, TV_i &= Fused(H_1, H_2, ..., H_j) \\ H_j &= TTLinear(x, r_j) \end{aligned} \tag{2}$$

where $TTLinear$ represents the tensor-compression linear function, h_i is the head of attention, TQ_i, TK_i, TV_i represents the tensorized query, key, value, H_j is the head of our method and r_j is the rank of tensor-compression. The 2D backbone uses the pyramid structure, which has downsample blocks and upsample blocks. Both downsample and upsample blocks

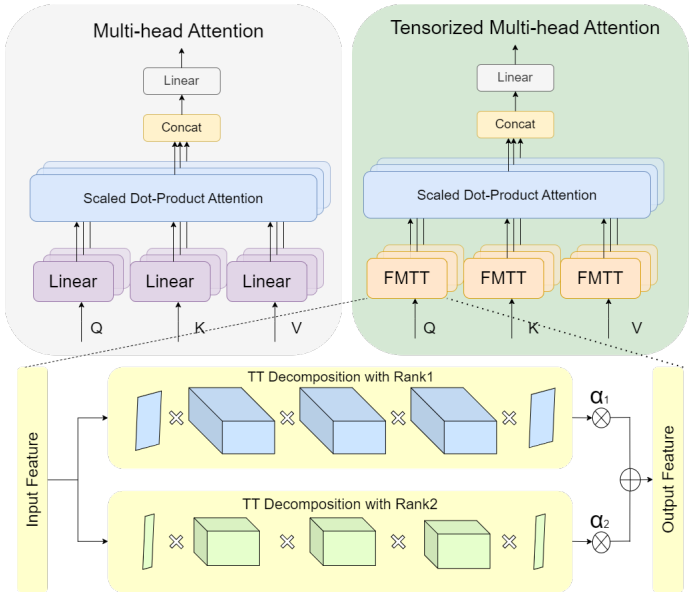

Fig. 3: Comparison between Our Compressed Attention and Conventional Attention

are convolution-based module. The fused multi-head tensor-compression convolution is applied here. Our compression only compresses the weights without changing the inputs and outputs so the pyramid structure still can downsample and upsample the features layer by layer. We note that our proposed method can also apply in other networks flexibly to compress the linear or convolution layers.

V. Fused Multi-head Tensor-compression for Attention

A. Tensor-Train Compression

In this section, we discuss the basic tensor compression method. To compress the tensor with maximum feature reserved, we choose tensor-train compression [13], which can maintain more information and provide more diversity for the features than traditional tensor decomposition method.

The normal form of linear function can be expressed as $O = Wx + b$ where O, W, x, b respectively denotes the output, weight, input and the bias of the linear function respectively. To use tensor-train decomposition to simplify the linear function with much fewer parameters, the weight matrix $W \in \mathbb{R}^{M \times N}$ can be reshaped to a tensor $\mathscr{W} \in \mathbb{R}^{C_{m_1} \times C_{n_1} \times ... \times C_{m_d} \times C_{n_d}}$, where $M = \prod_{i=1}^{d} C_{m_i}$ and $N = \prod_{i=1}^{d} C_{n_i}$. For a more general discussion, give a d-dimensional tensor $\mathscr{Y} \in \mathbb{R}^{p_1 \times p_2 \times ... \times p_d}$ as an example, where $\mathscr{Y}(k_1, k_2, ..., k_d)$ is a specific element of $\mathscr{Y}$. Then it can be approximated by a number of tensor cores $\mathscr{G}_k \in \mathbb{R}^{r_{k-1} \times p_k \times r_k} (k \in [1, d])$, which can be represented as follows:

$$\mathscr{Y}(k_1, k_2, ..., k_d) = \mathscr{G}_1(k_1)\mathscr{G}_2(k_2)...\mathscr{G}_d(k_d) \tag{3}$$

where r_k is the rank of $\mathscr{G}_k$. To consider each integer p_k, if it can be decomposed as $p_k = m_k \times n_k$ for example, the $\mathscr{Y}$ can be represented as $\mathscr{Y} \in \mathbb{R}^{m_1 \times n_1 \times m_2 \times n_2 \times ... \times m_d \times n_d}$ with $\mathscr{G}_k^* \in \mathbb{R}^{r_{k-1} \times r_k \times m_k \times n_k}$.

Besides, the tensor-train convolution layer is compressed in the similar way. The difference between linear and convolution is that convolution can extract the adjacent information from the input feature. Previous work [14] uses a prepositive small convolution layer and common tensor-train linear to replace the convolution layer. It can extract adjacent features similar to convolution operation but keep the model highly compressed. We follow the same strategy as [14].

B. Fused Multi-head Tensor-Train Attention

In this part, we introduce our proposed tensor compression method and compare it to the conventional attention block. In the attention block, the compression rate using classic tensor-train compression is difficult to set. As a result, it easily loses the key features leading to a lower accuracy. To tackle this issue, we propose the fused multi-head tensor-compression method and apply it to the transformer. Following the discussion in tensor-train compression, the compressed weight can be expressed as:

$$\begin{aligned} &\mathscr{W}((i_1, j_1)(i_2, j_2), ..., (i_d, j_d)) = \\ &\mathscr{G}_1^*(i_1, j_1)\mathscr{G}_2^*(i_2, j_2)...\mathscr{G}_d^*(i_d, j_d) \end{aligned} \tag{4}$$

where $\mathscr{G}^*(i, j)$ represents the tensor core with double indexes. Similarly, x and b can be reshaped into d-dimentional tensors $\mathscr{X} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$ and $\mathscr{B} \in \mathbb{R}^{m_1 \times m_2 \times ... \times m_d}$. As a result, the output y can become as d-dimentional tensor $\mathscr{Y} \in \mathbb{R}^{m_1 \times m_2 \times ... \times m_d}$. Therefore, linear function in transformer attention blocks can be tensorized and decomposed as:

$$\begin{aligned} \mathscr{Y}(i_1, i_2, ..., i_d) = \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} ... \sum_{j_d=1}^{n_d} [\mathscr{G}_1^*(i_1, j_1)\mathscr{G}_2^*(i_2, j_2) \\ ...\mathscr{G}_d^*(i_d, j_d)\mathscr{X}(j_1, j_2, ..., j_d)] + \mathscr{B}(i_1, i_2, ..., i_d) \end{aligned} \tag{5}$$

In transformer, multi-head attention blocks use multi-head technique to extract different features from different heads. In our work, we also want to extract different features with different ranks. Furthermore, we propose the normalized weight coefficients for each heads to allocate different weights' coefficients during training. In this way, the more important features are allocated to higher coefficients. Moreover, it uses only one training process for both the weights and the coefficients, which avoids further fine tuning or retraining. Specifically, the coefficients are formulated as:

$$\beta_p = Softmax(\alpha) = \frac{exp^{\alpha_p}}{\sum_1^d exp^{\alpha_p}} \tag{6}$$

where the β_p represents the final normalized coefficient of each head. The overall operation of multi-head tensor-compression methods can be expressed as:

$$\begin{aligned} \mathscr{Y}(i_1, i_2, ..., i_d) = \sum_{p=1}^{q} \beta_p \{ \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} ... \sum_{j_d=1}^{n_d} [\mathscr{G}_1^{p*}(i_1, j_1)\mathscr{G}_2^{p*} \\ (i_2, j_2)...\mathscr{G}_d^{p*}(i_d, j_d)\mathscr{X}(j_1, j_2, ..., j_d)] + \mathscr{B}^p(i_1, i_2, ..., i_d) \} \end{aligned} \tag{7}$$

C. Rank Selection

Rank in tensor-train compression is a significant hyper parameter, which affects accuracy, number of parameters and FLOPs. To balance the accuracy and computation load, we firstly discuss the FLOPs with comparisons of different ranks and different operations. Without losing generality, a calculation example is given as follows. If the input feature is $\mathscr{F} \in \mathbb{R}^{C_{in} \times H \times W}$, which can be reshaped to $\mathscr{F} \in \mathbb{R}^{C_{in_1} \times C_{in_2} \times C_{in_3} \times C_{in_4} \times H \times W}$ and output feature is $\mathscr{F} \in \mathbb{R}^{C_{out} \times H \times W}$, which can be shaped to $\mathscr{F} \in \mathbb{R}^{C_{out_1} \times C_{out_2} \times C_{out_3} \times C_{out_4} \times H \times W}$. In this case, $C_{in} = \prod_{i=1}^{4} C_{in_i}$ and $C_{out} = \prod_{i=1}^{4} C_{out_i}$. So the computation load of our method with ranks $[r_1, r_2, r_3, r_4]$ will be expressed as:

$$\#FLOPs_{TT} = H \times W \times \sum_{i=1}^{4} (\prod_{k=i}^{i+4} C_k \prod_{k=i}^{i+1} r_k), \tag{8}$$

where C_1 to C_4 represent C_{in_1} to C_{in_4}, C_5 to C_8 represent C_{out_1} to C_{out_4} and r_5 is one in this case.

From the calculation, we know the effects of different ranks. To select the ranks more precisely, here a complexity-accuracy co-aware rank selection criterion function is applied as:

$$C_{rank} = C_{loss} + \gamma \cdot C_{comp} \tag{9}$$

$$C_{comp} = \theta \cdot \#PARAMs + \#FLOPs \tag{10}$$

where the C_{rank} is the final criterion we use to judge and select ranks. The C_{loss} is the cross entropy loss, $\#PARAMs$ represents the number of parameters, $\#FLOPs$ represents the GFLOPs of the operation, γ and θ are the manual-set parameters to control the priority of different parts. Compare with different C_{rank}, we select ranks with best trade-off for fused multi-head tensor-compression model. Detailed experimental results will be discussed in section VI.

Overall, we summarize our algorithm in Algorithm 1. It begins at a short pre-train to select the efficient ranks for multi-head. Then each head will use tensor-train operation and multiply the trained coefficient normalized by softmax, and finally fuse them as outputs.

VI. Experimental Results

A. Experiment Set-up

The dataset used in this work is KITTI dataset [15]. The KITTI dataset is a large-scale computer vision dataset designed to support research in the field of autonomous driving, and it provides a rich set of sensor data that can be used to develop and evaluate algorithms for tasks such as object detection, tracking, and segmentation. In 3D Lidar data, it has 7481 items for training and 7518 items for testing. To compare

Algorithm 1: Fused multi-head tensor-compression methods

input : the input feature x, the rank selection space
output: the output feature y, the selected ranks $r_1...r_d$

Five epochs pre-train for different ranks in rank selection space to get the average loss, number of parameters and floating point operations.
for $i \leftarrow 1$ *to* N **do**
  $C^{r_i}_{comp} = \theta \cdot \#PARAMs^{r_i} + \#FLOPs^{r_i}$
  $C^{r_i}_{rank} = C^{r_i}_{loss} + \gamma \cdot C^{r_i}_{comp}$
$r_1...r_d$ = sorted(C_{rank})[:d], sort the C_{rank} and select the minimum d items' ranks.
for $i \leftarrow 1$ *to* d **do**
  $\beta_i = Softmax(\alpha) = \frac{exp^{\alpha_i}}{\sum_1^d exp^{\alpha^i}}$, normalize the coefficients of each heads.
  $y+ = TTLinear_{r_i}(x) \cdot \beta_i$, apply the fused multi-head tenor-train compression and get the output feature

with other works, we use the mean Average Precision (mAP) with Intersection over Union (IoU) 0.7 for car detection as the comparison testing item. Our results show the compressed model accuracy, computation load measured in GFLOPs and model size. As such, these results indicate that our method could be applied to most edge devices with reduced model size and computation load.

B. Performance Comparison

To assess the performance of our fused multi-head tensor compression method, we firstly compare our compressed model with the other 3D point clouds models. Secondly, we compare with the other compression methods which are applied on the same model.

For the details about training, we train our model in 100 epochs with learning rate 0.003. Because of the small size models required on edge devices, we mainly compare with Single-Stage Detector (SSD) models. Table II shows the whole comparison results. Comapred to the VoTr-SSD model, our TT-VoTr-SSD model achieves 6.04 $\times$ smaller size and 2.62% improvement on easy mode and 0.28% improvement on hard mode.

In table III, we compare our fused multi-head tensor-compression method with other compressed method on the same model. Baseline is the original model from VoTr [4] using nothing to compress. The tensorized model applies one-head rank 8 tensor-train on the model and it cannot achieve a satisfactory performance. The quantization aware training (QAT) method we used is learned step size quantization (LSQ) [23]. LSQ has been proved that it can achieve better performance than most quantization methods. In our training, we use the original model as the pre-trained model and train another 50 epochs with $1e-4$ learning rate. Structured sparsity learning (SSL) [6] is more regular and friendly to hardware design than unstructured sparsity. In our work, we prune each channel weights with 50% sparsity and calculate the size and GFLOPs of one single operation. Through the comparison, our fused multi-head tensorized model achieves the best performance, much higher compression rate than quantization and sparsity. Meanwhile, the GFLOPs is still less than the original model and the quantized model, but only a little higher than sparsity.

TABLE II: The average accuracy in easy, moderate and hard mode and model size comparison on the KITTI dataset car category by 40 recall positions.

Method	Mode	Acc(%) Easy	Mod.	Hard	Size
Part-A2 Net [16]	TSD	87.81	78.49	73.51	226MB
PV-RCNN [17]	TSD	90.25	81.43	76.82	50.1MB
PointRCNN [18]	TSD	86.96	75.64	70.70	14.9MB
VoxelNet [1]	SSD	77.47	65.11	57.73	78.26MB
Patches [19]	SSD	88.67	77.20	71.82	-
STD [20]	SSD	87.95	79.71	75.09	-
PointPillars [8]	SSD	82.58	74.31	68.99	18.4MB
HVNet [3]	SSD	87.21	77.58	71.79	77.29MB
3DSSD [21]	SSD	88.36	79.57	74.55	30.0MB
SA-SSD [22]	SSD	88.75	79.79	74.16	40.7MB
VoTr-SSD [4]	SSD	86.73	78.25	72.99	55.09MB
TT-VoTr-SSD(Ours)	SSD	**89.35**	77.58	**73.27**	**9.12MB**

TABLE III: Comparison between our fused multi-head tensorized method and other compressed methods

Method	GFLOPs	Acc(%) Easy	Mod.	Hard	Size
Baseline	10.4	86.73	78.25	72.99	55.09MB
Tensorized	5.84	84.77	71.82	66.77	8.96MB
QAT [5]	10.5	85.62	72.50	69.37	14.91MB
Sparsity [6]	5.26	85.46	70.81	67.44	28.31MB
FMTT	7.32	**89.35**	77.58	**73.27**	9.12MB

C. Ablation Studies

In the ablation studies, we analyse the performance of different ranks in several aspects. Through the utilization of tensor-train algorithm, the only one hyperparameter rank controls the extended dimensions and determines accuracy, the compression rate and the FLOPs. From the selection method proposed in section IV, our fused multi-head tensor-compression model finally chooses the rank 4 and rank 8 at each head to extract different features. In our experiment, we perform the experiments with different ranks includes rank 2, rank 4, rank 8, rank 16. To prove our rational selection

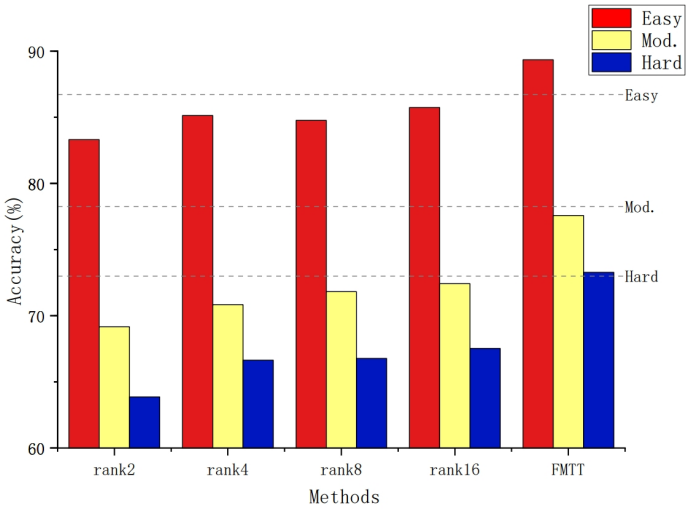

Fig. 4: Accuracy Comparison of Our Compressed Models with Different Ranks

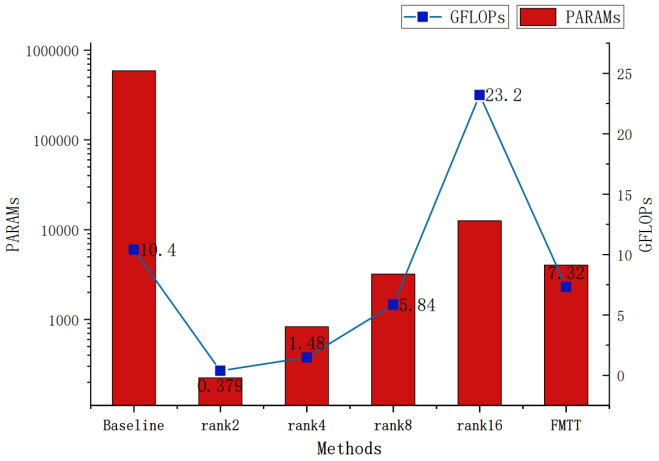

Fig. 5: Complexity Comparison of Our Compressed Operations with Different Ranks

method, the model training results, the memory usage and the FLOPs of each operator are shown in Fig. 4 and Fig. 5. In Fig. 4, baseline is the VoTr model illustrated by dotted lines. In Fig. 5, baseline is the convolution operator. From figures, we know the accuracy of all ranks are lower than the baseline but rank 2 remarkably has worse performance. FLOPs is also an important index for forwarding rate so rank 16 operation is too complex to forward in model. Therefore, the selection results of rank 4 and rank 8 as each head of our model are reasonable.

VII. Conclusion

In this paper, we present a fused multi-head transformer with tensor-compression for 3D point clouds detection. We further propose a rank selection strategy for the fused multi-head tensor-compression method to have a higher compression rate without accuracy drop.

VIII. Acknowledgement

This work was supported in part by the National Key RD Program of the Ministry of science and technology under Grant 2021YFE0204000, and in part by the Shenzhen Science and Technology Program under Grant KQTD20200820113051096.

References

[1] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *CVPR*, 2018, pp. 4490–4499.
[2] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.
[3] M. Ye, S. Xu, and T. Cao, "Hvnet: Hybrid voxel network for lidar based 3d object detection," in *CVPR*, 2020, pp. 1631–1640.
[4] J. Mao, Y. Xue, M. Niu, H. Bai, J. Feng, X. Liang, H. Xu, and C. Xu, "Voxel transformer for 3d object detection," in *ICCV*, 2021, pp. 3164–3173.
[5] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.
[6] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *NeurIPS*, vol. 29, 2016.
[7] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017, pp. 652–660.
[8] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *CVPR*, 2019, pp. 12 697–12 705.
[9] C. Ding, H. Ren, Z. Guo, M. Bi, C. Man, T. Wang, S. Li, S. Luo, R. Zhang, and H. Yu, "Tt-lcd: Tensorized-transformer based loop closure detection for robotic visual slam on edge," in *ICARM*. IEEE, 2023, pp. 166–172.
[10] C. Man, C. Chang, C. Ding, A. Shen, H. Ren, Z. Guan, Y. Cheng, S. Luo, R. Zhang, N. Wong, and H. Yu, "Ranksearch: An automatic rank search towards optimal tensor compression for video lstm networks on edge," in *DATE*, 2023, pp. 1–2.
[11] Z. Guan, S. Li, Y. Cheng, C. Man, W. Mao, N. Wong, and H. Yu, "A video-based fall detection network by spatio-temporal joint-point model on edge devices," in *DATE*, 2021, pp. 422–427.
[12] Y. Cheng, G. Huang, P. Zhen, B. Liu, H.-B. Chen, N. Wong, and H. Yu, "An anomaly comprehension neural network for surveillance videos on terminal devices," in *DATE*, 2020, pp. 1396–1401.
[13] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011.
[14] Y. Pan, M. Wang, and Z. Xu, "Tednet: A pytorch toolkit for tensor decomposition networks," *Neurocomputing*, vol. 469, pp. 234–238, 2022.
[15] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *IJRR*, vol. 32, no. 11, pp. 1231–1237, 2013.
[16] 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, 2020.
[17] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "Pv-rcnn: Point-voxel feature set abstraction for 3d object detection," in *CVPR*, 2020, pp. 10 529–10 538.
[18] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *CVPR*, 2019, pp. 770–779.
[19] J. Lehner, A. Mitterecker, T. Adler, M. Hofmarcher, B. Nessler, and S. Hochreiter, "Patch refinement–localized 3d object detection," *arXiv preprint arXiv:1910.04093*, 2019.
[20] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "Std: Sparse-to-dense 3d object detector for point cloud," in *ICCV*, 2019, pp. 1951–1960.
[21] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3dssd: Point-based 3d single stage object detector," in *CVPR*, 2020, pp. 11 040–11 048.
[22] C. He, H. Zeng, J. Huang, X.-S. Hua, and L. Zhang, "Structure aware single-stage 3d object detection from point cloud," in *CVPR*, 2020, pp. 11 873–11 882.
[23] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," *arXiv preprint arXiv:1902.08153*, 2019.